ElectricFlow 8.5
User Guide

Electric Cloud, Inc.
125 South Market Street, Suite 400
San Jose, CA 95113
www.electric-cloud.com

Electric Cloud

ElectricFlow version 8.5

Published 4/25/2019

Trademarks

Contents

About This Guide

The *ElectricFlow User Guide* discusses how to use the ElectricFlow® DevOps Release Automation platform for provisioning, build, and release management of multi-tiered applications. ElectricFlow is a unified platform for automating bottlenecks in the software delivery lifecycle and making software delivery faster, easier, and more reliable. This guide shows DevOps teams how to get started with automation by plugging in their existing tools and processes and future-proofing their work by using ElectricFlow's flexible, scalable orchestration platform.

Audience

This guide is for DevOps professionals such as product managers, software developers, and IT staff who want to automate the processes between their respective teams to build, test, and deploy or release software faster and more reliably.

Organization

This guide covers topics in the following chapters and appendices:

Chapter	Description
Introduction to ElectricFlow on page 1	Provides an overview of the product, including details about its purpose and key benefits.
Deployment Automation on page 51	Explains how to use the ElectricFlow Deploy module to make deployments manageable, reproducible, and error-proof by modeling an application or microservice and the environments to which it will be deployed and automating the workflow needed for the deployment.
Pipelines on page 413	Describes how to create a representation and then orchestrate the flow of your software delivery process on its journey to production to enable Application Release Automation (ARA) and Continuous Delivery (CD). It shows how to take software through the hops and milestones (stages) of the software release life cycle by modeling and executing one or more applications, microservices, procedures, or workflows in a pipeline.
Release Management on page 619	Shows how to use the ElectricFlow Release module to capture, execute, visualize, and control the life cycle of multiple application enterprise releases. With the Release capability, outputs from multiple teams can be coordinated to produce a final release to be pushed to production. You can create a Release model consisting of multiple applications deployed on different systems, such as traditional, cloud, mainframes, and remote servers. You can then run the Release pipeline reliably and repeatedly until the software release is complete.

Chapter	Description
DevOps Insight on page 691	Describes how to use the DevOps Insight dashboards, which give you insights into deployment and release activities over time. Dashboards visualize this information to help you understand the overall status of you processes, identify hotspots that need action, understand trends, and find opportunities for further improvement. DevOps Insight provides several bundled dashboards as well as the ability to create custom dashboards.
Self-Service Catalogs on page 860	Explains how to set up and use the ElectricFlow Self-Service Catalogs to help you accelerate application deployment. This feature lets you automate routine deployment tasks and publish them as templates so that end users can use them by providing minimal information and without learning to use ElectricFlow.
Security on page 875	Describes how to set up security by assigning roles and privileges to specific users and groups on system objects including applications, environments, projects, jobs, and schedules as well as actions performed on deployment models. ElectricFlow uses access control, project-level security, and credentials and impersonation to enforce roles and privileges when executing deployment steps.
Change Tracking on page 1374	Shows how to track every change between every state of non-runtime ElectricFlow objects and revert to any previous state of these objects. ElectricFlow tracks the changes to tracked objects including applications, procedures, workflows, workspaces, resources, and project-owned components such as library components. It records a change history of the historical states of the system and the changes between them.
Notifications on page 943	Describes how to configure and manage email notifications, also called email notifiers, for the following ElectricFlow objects: application processes, application process steps, jobs and job steps, workflow states, procedure results, procedure step results, and state definition results.
Automation Platform on page 959	Shows how to create, configure, and manage objects in the Automation Platform that make the automation of build-test processes, application deployments, and pipelines possible in ElectricFlow.
Plugins That Are Bundled with ElectricFlow on page 1	Provides a list of plugins that are included with this version of ElectricFlow.
Using Special Characters in ElectricFlow Object Names on page 1	Lists which special characters to avoid when naming objects in ElectricFlow. These characters have special purposes or meanings within ElectricFlow or in the scripting language being used.

Related Documentation and Online Help

Product Documentation

ElectricFlow product documentation is available at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html as follows:

- *ElectricFlow Installation Guide*
- *ElectricFlow User Guide* (this document)
- *ElectricFlow API Guide*
- *ElectricFlow Release Notes*
- *ElectricFlow SDK Plugin Developer Guide* (updated on its own release cycle)
- *ElectricFlow SDK Plugin Developer Release Notes* (updated on its own release cycle)

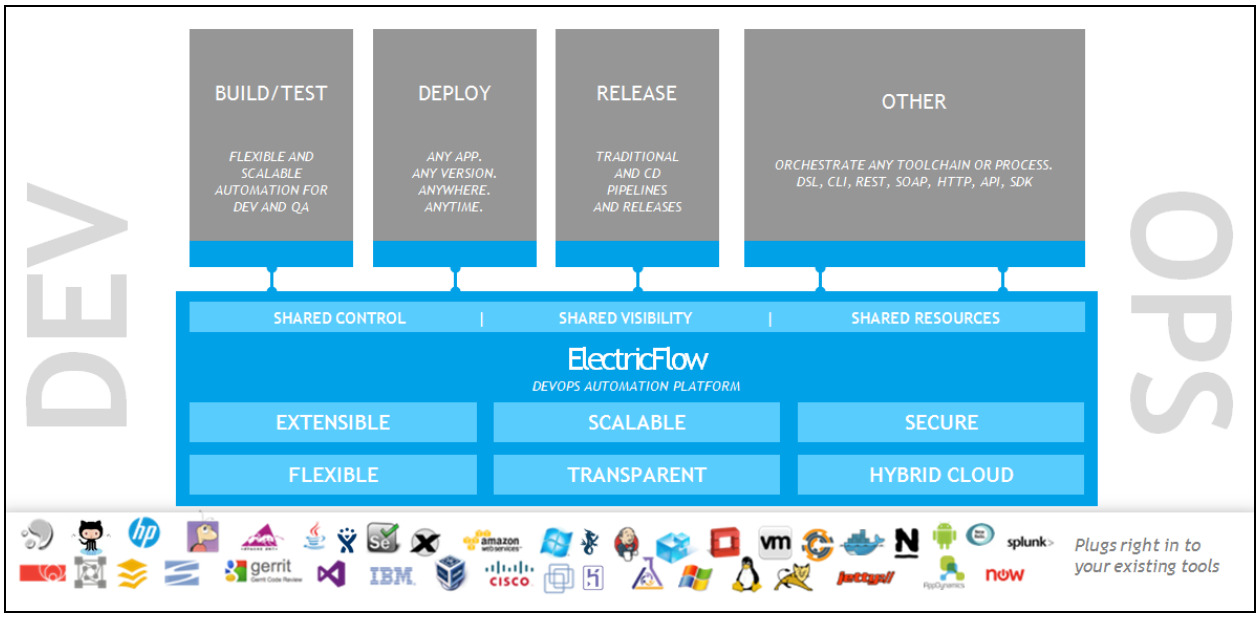
Documentation on the website is updated periodically.

Automation Platform Online Help

The ElectricFlow Automation Platform has a complete, robust, context-sensitive online help system. To use it, click the **Help** button in any page of the Automation Platform web UI.

Chapter 1: Introduction to ElectricFlow

ElectricFlow® is an enterprise-grade DevOps Release Automation platform that simplifies provisioning, build, and release of multi-tiered applications and microservices. Its model-driven approach to managing environments, applications, and microservices allows teams to coordinate multiple pipelines and releases across hybrid infrastructure in an efficient, predictable, and auditable way.



Web-Based System

At its core, ElectricFlow automation platform is a web-based system for automating and managing the build, test, deployment, and release process. It provides a scalable solution and solves some of the biggest challenges of managing these "back end" software development tasks, including:

- Time wasted on script-intensive, manual, home-grown systems that
  - Are error prone
  - Do not scale well
  - Have little or no management visibility or reporting
- Multiple, disconnected build and test systems across locations, resulting in:
  - Redundant work
  - Inability to share or reuse code files across teams
  - Hard to manage build and test data
- Slow overall build and release cycles that directly impact:
  - Release predictability
  - Time-to-market

Automation Platform

The automation platform has a three-tier architecture, an AJAX-powered web interface, and a first-of-its-kind build and release analytic capabilities for reporting and compliance. With this solution, your developers, release engineers, build managers, QA teams, and managers gain:

- A shared platform for disseminating best practices and reusing common procedures
- The ability to support geographically distributed teams
- Continuous integration and greater agility
- Faster throughput and more efficient hardware utilization
- Visibility and reporting for better project predictability
- Better software quality by integrating and validating against all target platforms and configurations

For examples of ElectricFlow architecture configurations, see ElectricFlow Architecture on page 2.

What Makes ElectricFlow Unique?

ElectricFlow provides enterprise-class speed and scalability for software build and release management. It is easy to install and use on a simple build, yet it scales to support the largest and most complex build and test processes. ElectricFlow distributes jobs in parallel across multiple resources for faster overall cycle time.

ElectricFlow supports multiple teams working in multiple locations and programming in multiple languages in an environment that can be centrally managed. Shared assets and reuse make individual teams more efficient by eliminating duplicate work and gives organizations the power to deploy cross-company standards.

ElectricFlow's unique analytics provide visibility into one of the best indicators of project success: compiled, tested, working code. ElectricFlow's analytics database stores all build and test information for real-time and trend reporting to give your organization the power to collect pinpoint statistics and to gain visibility into important productivity metrics such as trends in error rates.

Additionally, out-of-the-box reports provide information about cross-project status and build trends by project and resource utilization. ElectricFlow's integration with virtual lab automation (VLA) solutions also lets you snapshot or reproduce a specific build for auditing or troubleshooting.

ElectricFlow provides unified process automation across the entire build-test-deploy life cycle and across heterogeneous tools via integrations with leading ALM tools. Integrations with SCM tools enable continuous integration and triggering of builds whenever code is checked into the specified repository or branch. When used with VMware Lab Manager, ElectricFlow can dynamically provision either physical or virtual resources without manual intervention. This feature delivers efficient, dynamic resource provisioning and reduces development and QA dependence on IT operations.

ElectricFlow Architecture

ElectricFlow was designed to support small, mid-range, or enterprise scale software production. Based on a three-tier architecture, ElectricFlow scales to handle complex environments. The ElectricFlow multi-threaded Java server provides efficient synchronization even under high job volume.

- The ElectricFlow server manages resources, issues commands, and generates reports.
- An underlying database stores commands, metadata, and log files.

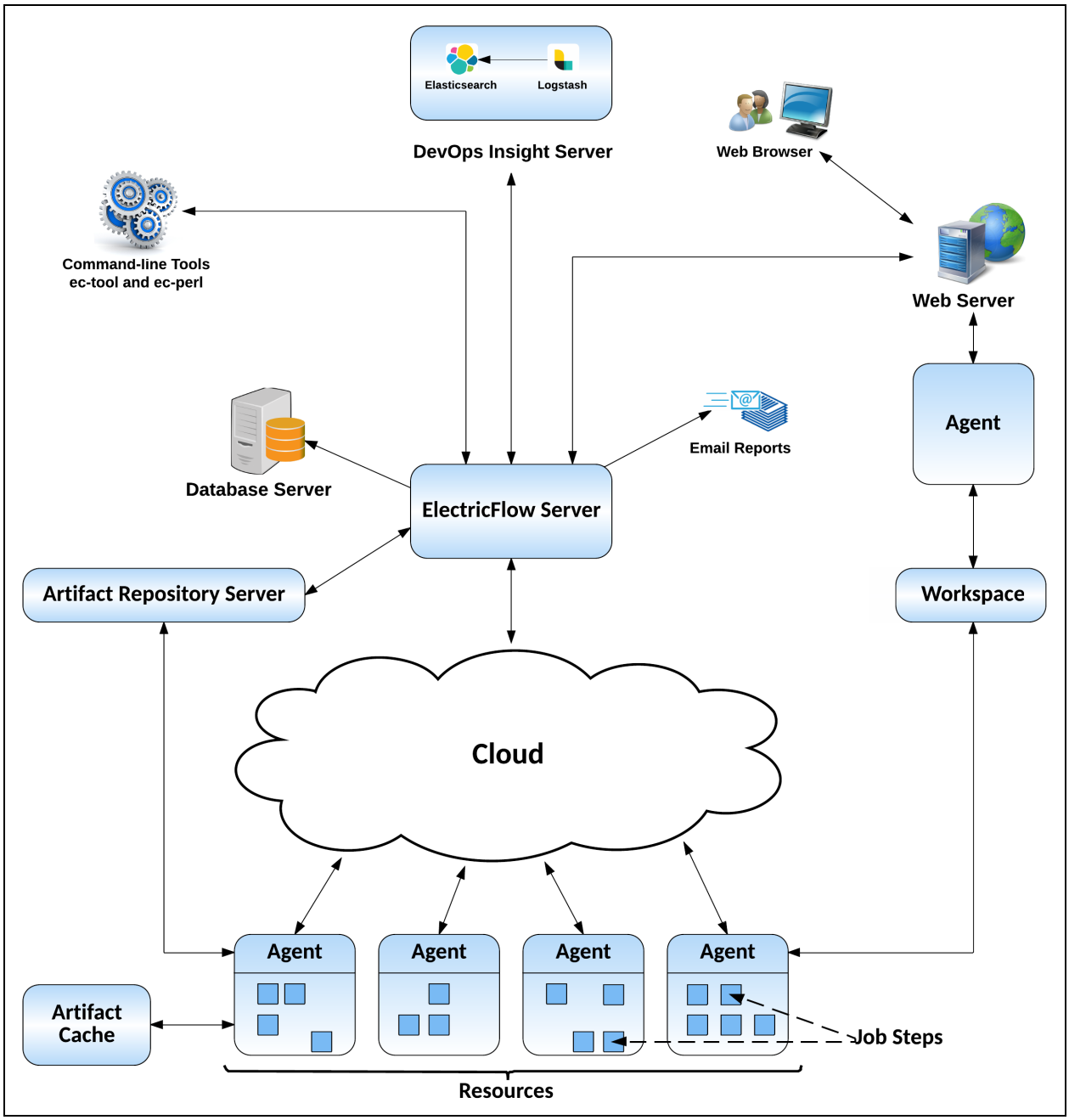
- Agents execute commands, monitor status, and collect results in parallel across a cluster of servers for rapid throughput.

Simple Architectural Overview

This local configuration applies to all the use cases. The ElectricFlow server, web server, artifact cache, Artifact Repository server, workspace, command-line tools, resources, agents, and job steps are all in the automation platform.

In this local configuration:

- The ElectricFlow server manages resources, issues commands, and generates reports.
- Resources, agents, and databases are managed in the automation platform.
- An underlying database stores commands, metadata, and log files.
- Procedures, which include job steps, are defined in the automation platform.
- Job steps are executed on resources in the defined environments.
- Applications (which include processes), components, microservices (which also include processes), containers, and environments are defined for deployment automation.
- Pipelines, stages, and tasks are defined for pipeline management.



For a production environment, Electric Cloud recommends that you install the database on a separate machine from the ElectricFlow server to prevent performance issues. It is acceptable for the ElectricFlow server, web server, and repository server to reside on the same machine in a local configuration, but not required. If you are only evaluating ElectricFlow, the ElectricFlow software, the database, the ElectricFlow server, the web server, and the repository server can reside on the same machine.

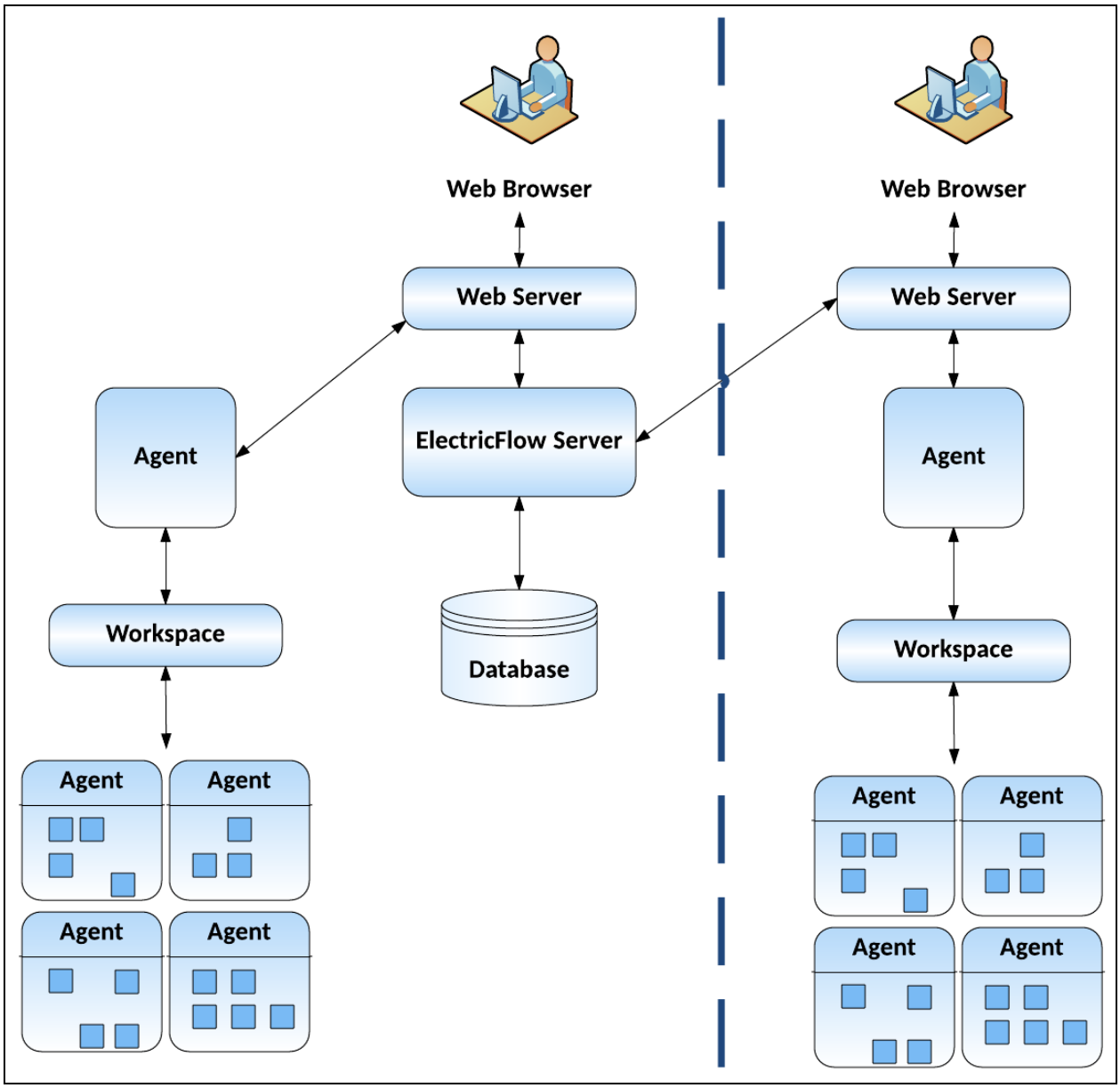
For a production environment, Electric Cloud recommends that you install the DevOps Insight server on a system other than systems running other ElectricFlow components (such as the ElectricFlow server, web server, repository server, or agent). If you must install it on the same system (such as for testing or other nonproduction or trial-basis situations only), see the "Running the DevOps Insight

Server on a System with Other ElectricFlow Components" section in the "Installing ElectricFlow" chapter of the *ElectricFlow User Guide* at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html for instructions.

Expanded Remote Configuration

ElectricFlow is not limited by only the components shown in the previous configuration. This configuration applies to all the use cases.

The following shows a remote web server configuration and is an example for how you may set up a remote web server installation.



This type of remote web server configuration helps prevent network latency. If you have multiple sites, ElectricFlow can be configured to help you work more efficiently.

Other Configurations

Go to the ElectricFlow Installation Guide (http://docs.electric-cloud.com/eflow_doc/FlowIndex.html) for other architecture configurations:

- Proxy (universal) resources
- Remote database
- Multiple remote web servers
- Multiple remote repository servers
- Clustered configuration for horizontal scalability and high availability

Roadmap to ElectricFlow

Use Case	Definition	Go to
Build-test automation	Executing automated build and test processes, resulting in reduced costs, increase quality, reliability and traceability, and accelerated time to market. Build-test automation is a key enabler of continuous integration (CI).	Automation Platform on page 959
Deployment automation	Executing automated application or microservice deployment processes, resulting in reduced costs, increase quality, reliability and traceability, and accelerated time to market. Deployment automation is a key enabler of Application Release Automation (ARA) and continuous delivery (CD).	Deployment Automation on page 51 Deployment Examples on page 252
Pipelines	Executing automated end-to-end software development and delivery processes, resulting in reduced costs, increase quality, reliability and traceability, and accelerated time to market. Pipeline management is a key enabler of Application Release Automation (ARA) and Continuous Delivery (CD).	Pipelines on page 413

Use Case	Definition	Go to
Release planning and management	The Release feature captures, executes, visualizes, and controls the life cycle of multi-application or multi-microservice enterprise releases, where outputs from multiple teams are coordinated to produce a final release to be pushed to production. Releases allow you to execute and manage automated software release processes (including end-to-end software development and delivery processes), enabling Application Release Automation (ARA) and Continuous Delivery (CD). You can use the Release feature to deliver software releases through traditional methods and/or ARA and CD methodology. ElectricFlow manages all the releases, regardless of the release methodologies used by the release team.	Release Management on page 619
Automation platform	Where you create, configure, and manage the objects that make the build-test, deployment, and pipeline management possible.	Automation Platform on page 959

Plugins and Electric Cloud Field-Contributed Solutions

See https://github.com/electric-cloud for plugins and Electric Cloud field-contributed solutions. The assets in these repositories are free to use. You can modify them as needed.

You can share enhancements and fixes via GitHub. You can also offer suggestions in the form of GitHub project "issues" so that Electric Cloud or others in the GitHub community can address them.

These assets are not officially supported by Electric Cloud and have not undergone formal testing. Electric Cloud is not liable for any repercussions of using this software.

EC-Admin

EC-Admin is a collection of administrative procedures to help you manage your ElectricFlow server. These procedures were developed to respond to requests from customers during Electric Cloud Professional Services engagements.

https://github.com/electric-cloud/EC-Admin

This repository contains the following modules:

- System health
- Look and feel

- Jobs and workspaces management
- Plugins
- Artifact management
- Object export backup and restore
- Schedules
- Semaphore management
- License logger
- postp debugger helper
- Communication
- Miscellaneous

DSL-Samples

This repository contains sample projects created with the ElectricFlow Domain Specific Language (DSL). This DSL is based on the Groovy programming language and is run directly on the ElectricFlow server JVM. It has full access to the ElectricFlow API and can be used for authoring ElectricFlow content (such as procedures, workflow, and pipelines) and for automation.

https://github.com/electric-cloud/DSL-Samples

For more information about ElectricFlow DSL, see the *ElectricFlow API Guide* at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html.

EC-DSLIDE

This is an ElectricFlow DSL web-based tool for editing and running DSL scripts.

https://github.com/electric-cloud/EC-DSLIDE

The DSL IDE is an ElectricFlow plugin.

ec-specs-tool

This is an easy-to-use testing tool for writing specifications and acceptance tests.

https://github.com/electric-cloud/ec-specs-tool

Guided Tutorials

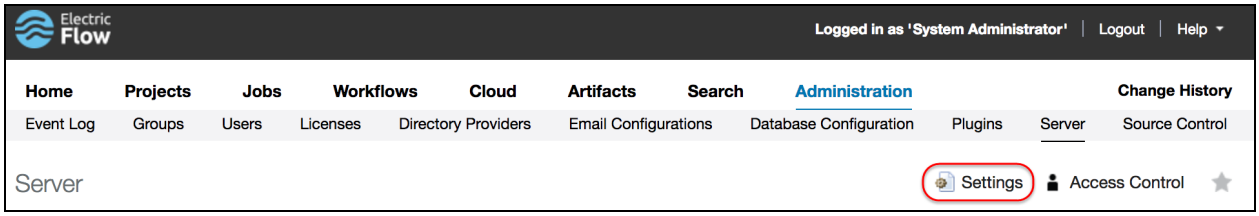
Whether you are starting fresh in ElectricFlow or learning a new concept, the guided tutorials make it easy to get started quickly and learn about the object model and product capabilities. These step-by-step tutorials coach you along the way with popups, which point to the next action to take and provide detailed instructions about related content.

An ElectricFlow server setting determines whether to enable the tutorials. For first-time installations, the tutorials are enabled by default. However, after an upgrade from version 7.3 or 8.0, they might not be enabled depending on your prior setting. If you need to enable them after upgrading from those versions, see the following instructions.

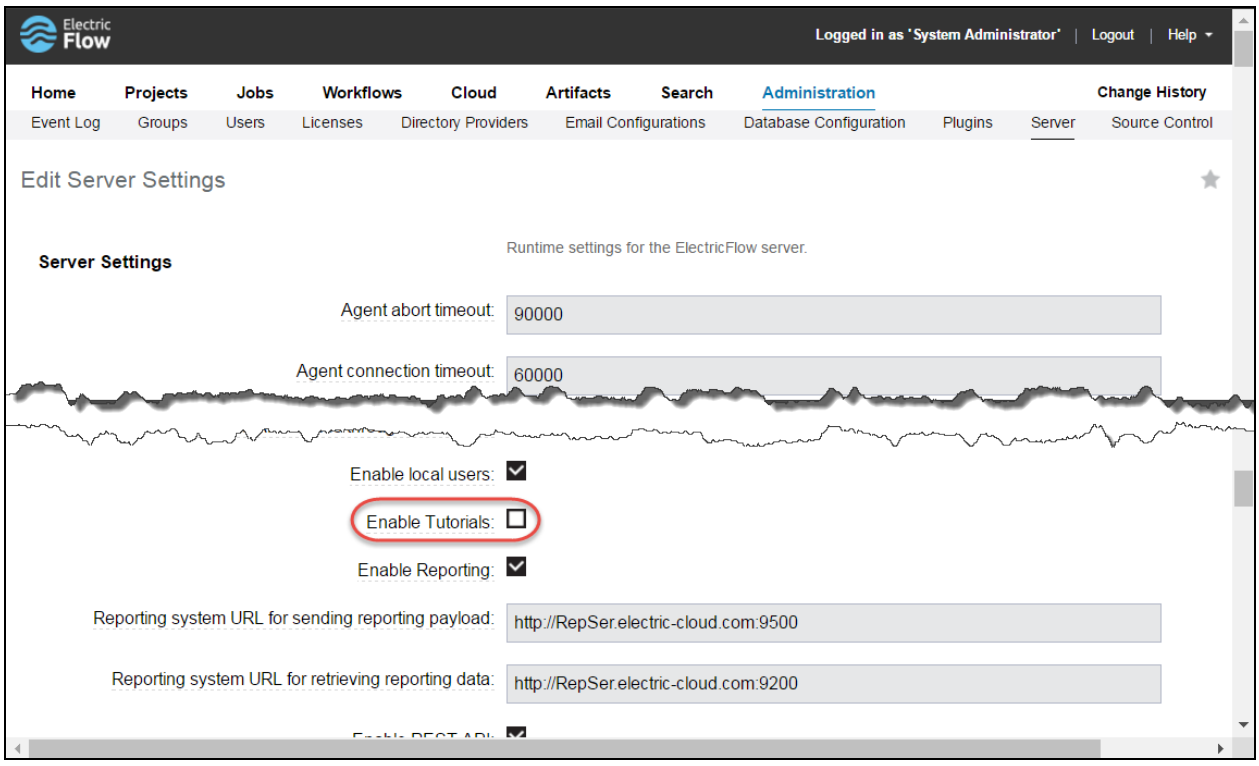
Enabling or Disabling the Tutorials

To enable or disable the tutorials:

1. Open the home page of the Automation Platform web UI by browsing to https://<*ElectricFlow_server*>/commander/ and logging in.
2. Click the **Administration** tab, then click the **Server** subtab, and then click **Settings**:



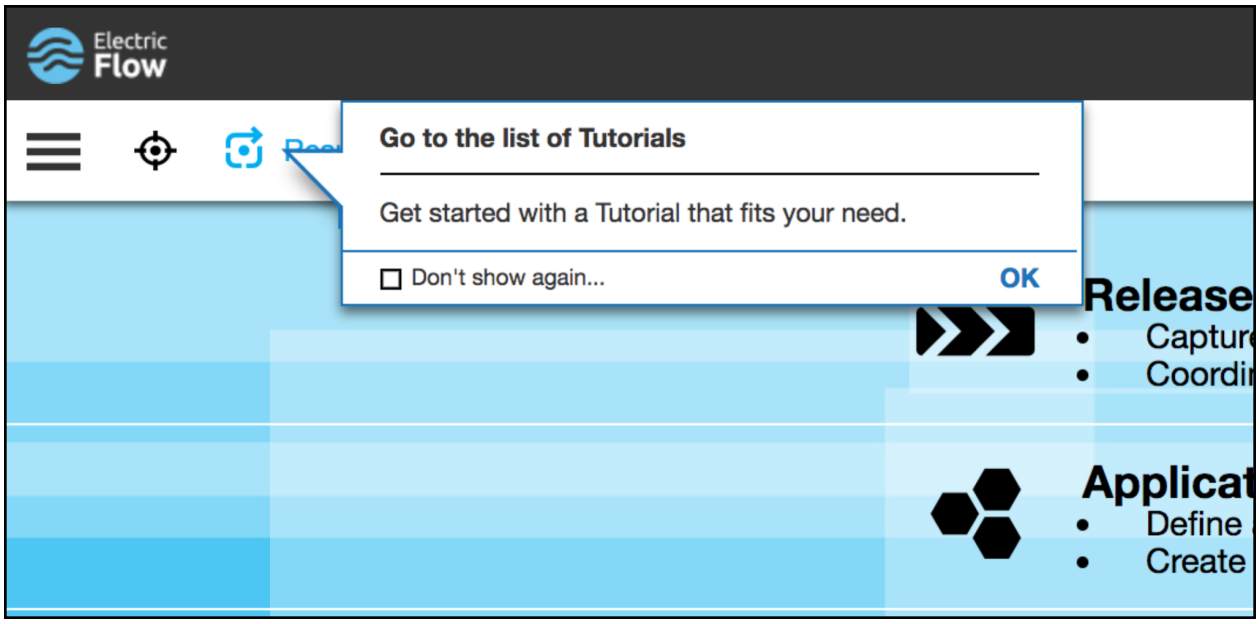
3. Enable or disable the tutorials by using the **Enable Tutorials** checkbox:



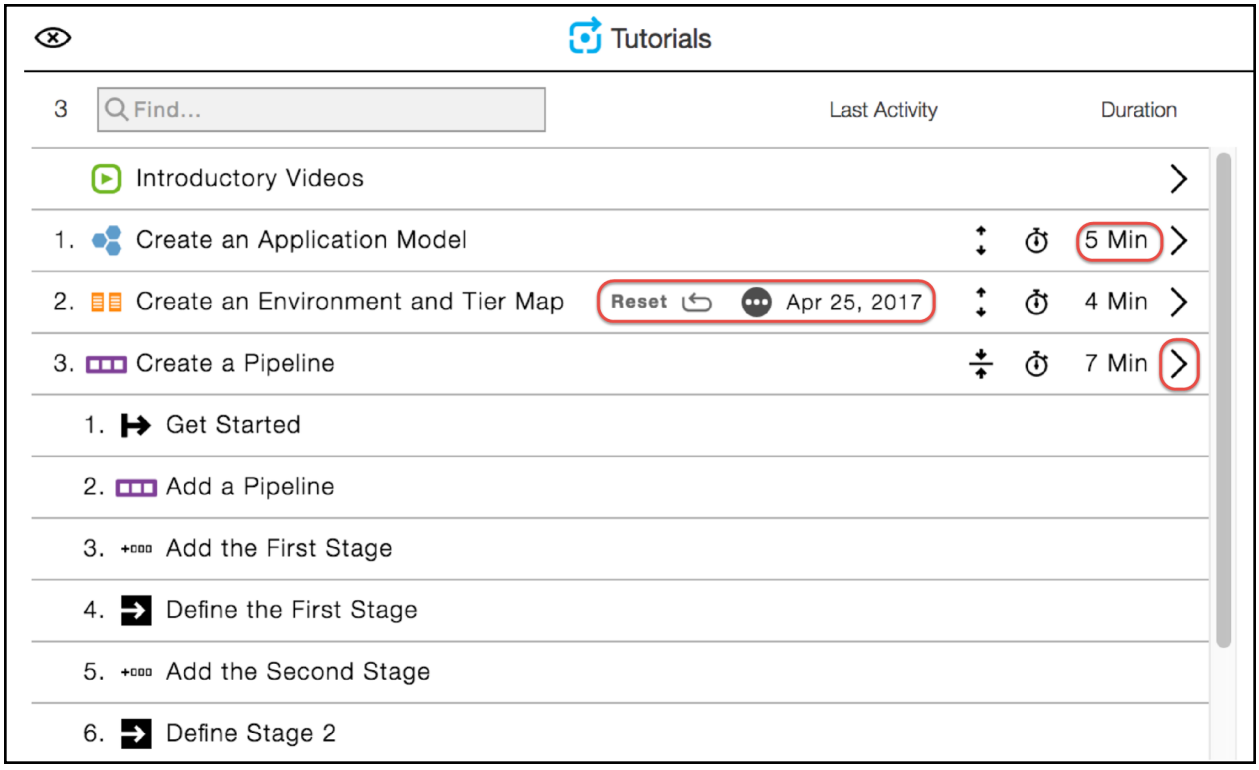
4. Click **OK**.

Viewing the List of Tutorials

When the tutorials are enabled, go to the ElectricFlow main page (https://<ElectricFlow_web_server>/flow) and then click the  (tutorials) button in the top left to see the full menu of available tutorials:



The **Tutorials** menu lets you expand the view to see the detailed steps in each tutorial. Each tutorial shows an estimated amount of time needed for completion. The menu also lists each tutorial that is in progress and the time stamp of the last activity. You can reset a tutorial by clicking the **Reset** button:

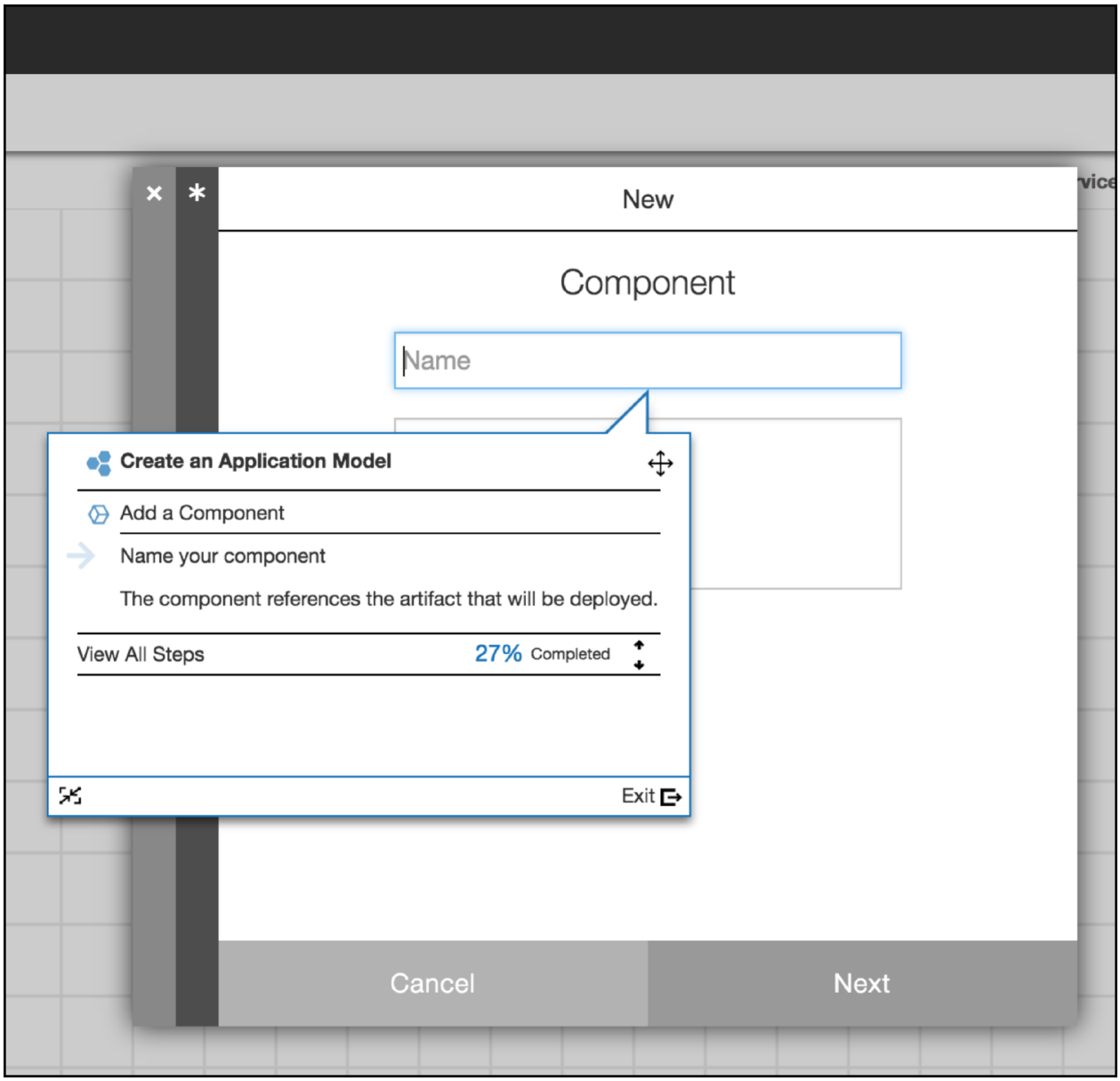


Starting or Continuing a Tutorial

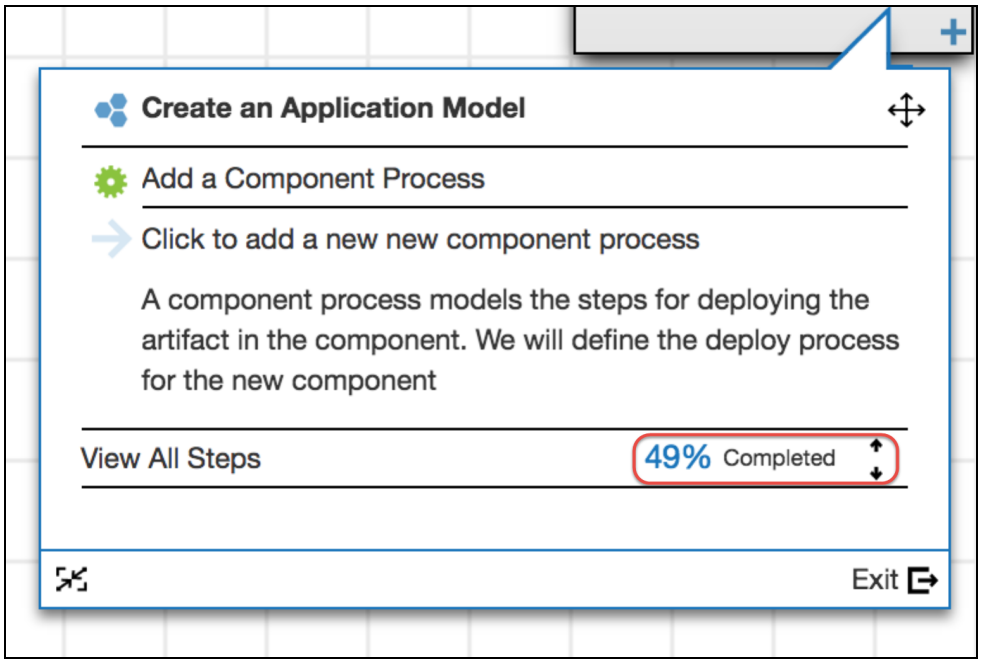
To start a specific tutorial or continue one in progress, click the (next) arrow adjacent to the corresponding item. The tutorial starts with a basic instruction for all tutorials and then starts the guided walk-through of the use case. The subsequent popups point to where you must take the next action. A different action might be required based on the UI item:

- Text entry: Enter text into the text box
- Link or button: Click the link or button
- Dropdown: Select an item from the list
- No arrow: Perform the instructions described and then click **Continue** in the popup

For example:



The coach mark popup gives a detailed description of the next action and the additional learning content. It also displays the percentage of the tutorial that you have completed to track your progress, which you can click to expand to show the full list of steps and the status of each step:



Deploy UI Elements

The DeployUI has numerous layout, interaction, and navigation features. These include headers, buttons, menus, and icons as well as popover, hover, and object-selected states. Topics included in this section:

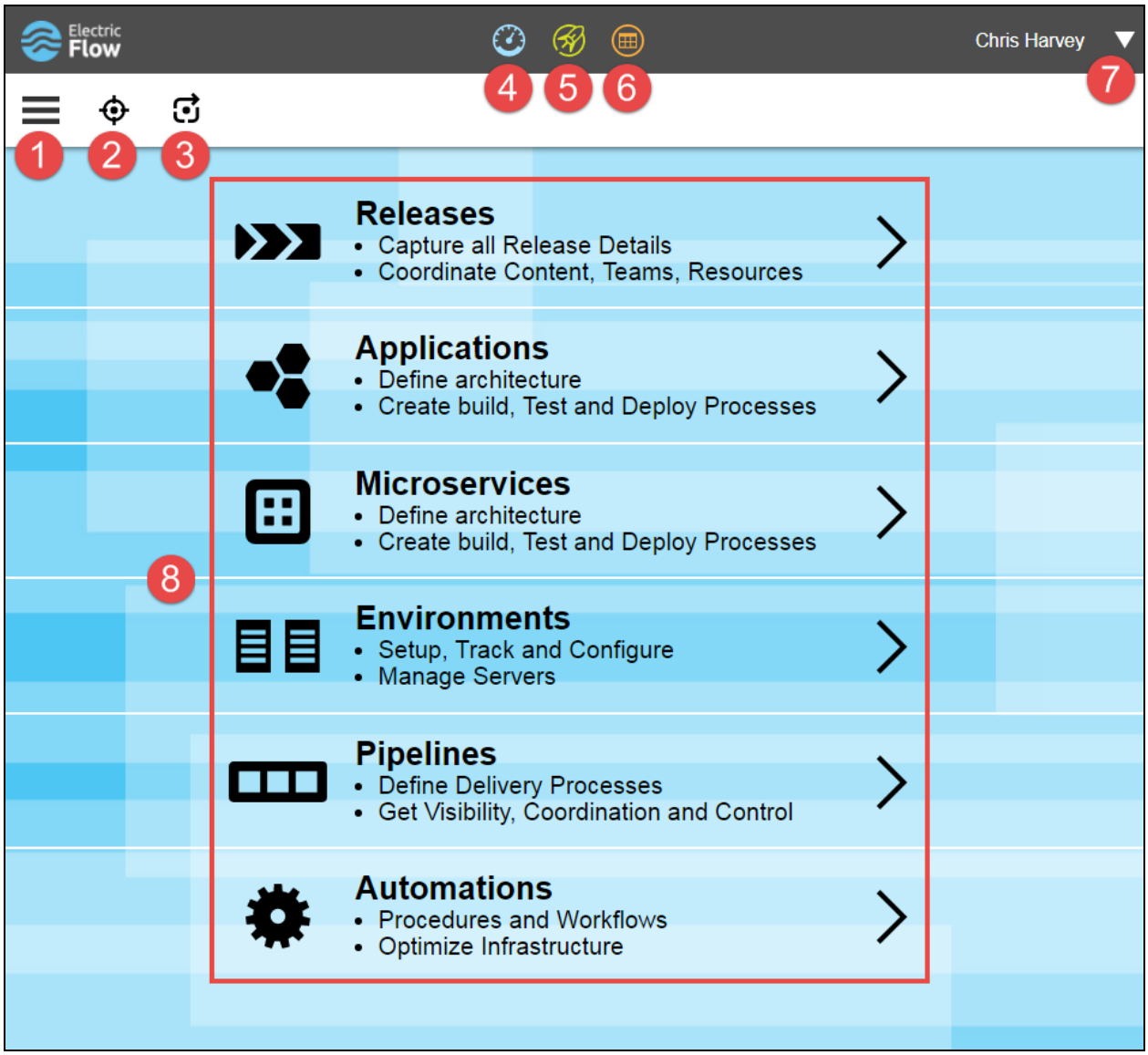
Home Page

How to get to here: Browse to https://<*ElectricFlow_server*>/flow/), enter your user name and password, and click **Login**.

> **Tip:**
> You can configure ElectricFlow to open either the Deploy UI or the Automation Platform UI without appending /flow or /commander respectively to the end of the https://<*ElectricFlow_server*> URL. For example, you can configure ElectricFlow so that it opens the Deploy UI whether you browse to https://ecdevopsserver1 or https://ecdevopsserver1/flow.

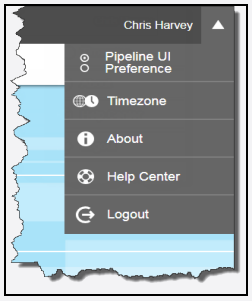
> You use the ecconfigure `--webDefaultUI` option to configure this behavior. For details, see Web Server Configuration Options on page 1057.

From the Home page, you can model applications or microservices for your continuous delivery solution:



1	**Main menu** button—Opens a list of destinations, which includes the list of launch pads on this page
2	**Change History** button—Opens the **Change History—Search** dialog box

3	**Tutorials** button (if present)—Opens the guided tutorials. If this button is not present, see Enabling or Disabling the Tutorials on page 8 for instructions on how to enable the tutorials
4	**DevOps Insight Dashboards** button—Opens the **DevOps Insight Dashboards** page
5	**DevOps Foresight** button (if present)—Opens the **DevOps Foresight** home page. DevOps Foresight requires an additional license; this button is not present if DevOps Foresight is not installed and licensed.
6	**Self-Service Catalogs** button—Opens the **All Items** list page for the Self-Service Catalogs
7	**Release Calendar** button—Opens the release calendar page for all projects or a specific project

8	Indicates the user who is currently logged in; opens the **Timezone**, **About**, or **Help Center** dialog boxes; or logs you out of ElectricFlow if you choose the **Log Out** option.
	 
	
	• **Pipeline UI Preference**—Opens a dialog box to let you toggle between the Pipeline Stage View and the Release Kanban View. For details, see Pipeline Views on page 437
	• **Timezone**—Opens a dialog box containing a pulldown menu for selecting the timezone in which you will use the software
	• **About**—Displays a popup containing the build version, UI version, build label, and protocol version of the installed release
	• **Help Center**—Opens the ElectricFlow page in the Electric Cloud Help Center. In this page, you can:
	  • Get started fast with easy video tutorials.
	  • Find helpful explanations in the Knowledge Base.
	  • Ask questions and get answers from the Community.
	  • See the documentation for completing key tasks.

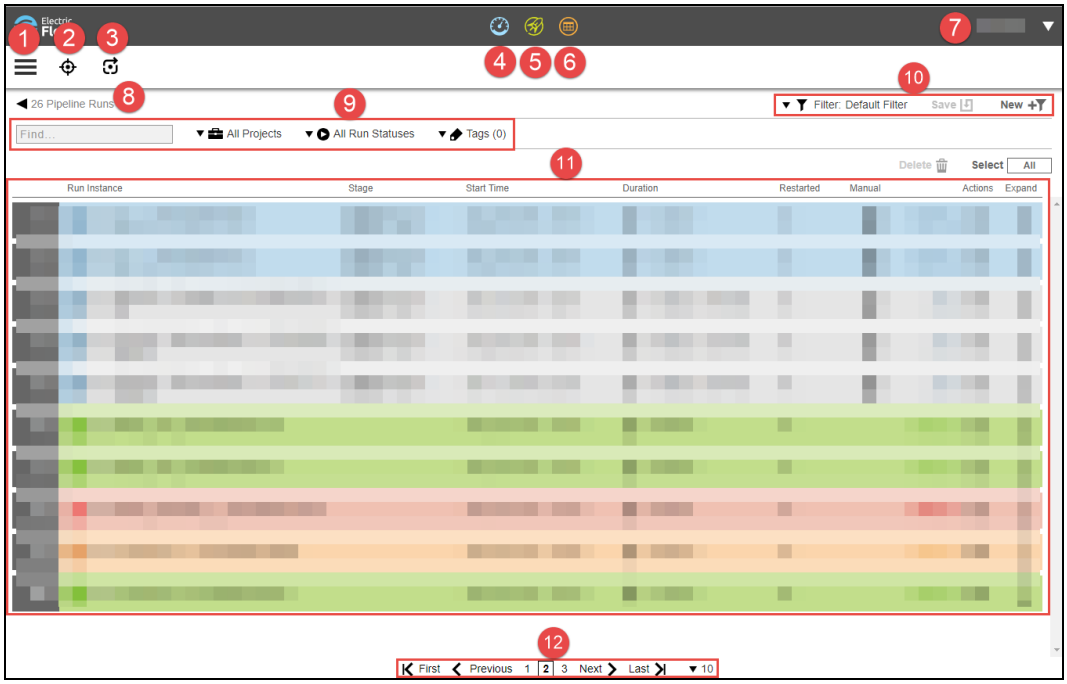
9	Launch pads—Click one of these links to open an objects list, a dashboard, or the Automation Platform to start using ElectricFlow: • **Releases**—Opens the Releases Dashboard. From here, you start to model releases. • **Applications**—Opens the Applications List. From here, you start to model applications by defining the application architecture and modeling the component and application processes. • **Microservices**—Opens the Microservices List. From here, you start to model microservices by defining the service architecture and modeling the container and service processes. • **Environments**—Opens the Environments List. From here, you start to model the environments where applications are deployed by assigning and managing resources. • **Pipelines**—Opens the Pipelines List. From here, you start to model pipelines. • **Projects**—Opens the Projects List. From here, you can create, edit, copy, or delete a project. **Dashboards**—Opens the list of available dashboards or the list of available reports. These are any dashboards and reports included in ElectricFlow as well as dashboards and reports that you have created. From the Dashboards List, you view, create, or edit DevOps Insight dashboards. From the Reports List, you view, create, or edit DevOps Insight reports. • **Automations**—Opens the ElectricFlow Automation Platform pages for procedures, workflows, artifacts, credentials, and projects. • **Admin**—Opens the ElectricFlow Automation Platform pages for users, groups, plugins, licenses, and ElectricFlow server configuration.

Object List Layout

An object list page contains numerous layout, interaction, and navigation features are contained in object lists for applications, environments, microservices, and pipelines. These include headers, buttons, menus, and icons as well as popover, hover, and object-selected states.

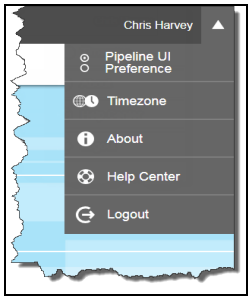
The **Projects** list page, below, shows the counts for child objects that belong to each project. This includes the number of applications, pipelines, environments, and releases.

All other object list pages follow the format as shown below.



1	**Main menu** button—Opens a list of destinations, which includes the list of launch pads on this page
2	**Change History** button—Opens the **Change History—Search** dialog box

3	**Tutorials** button (if present)—Opens the guided tutorials. If this button is not present, see Enabling or Disabling the Tutorials on page 8 for instructions on how to enable the tutorials
4	**DevOps Insight Dashboards** button—Opens the **DevOps Insight Dashboards** page
5	**Self-Service Catalogs** button—Opens the **All Items** list page for the Self-Service Catalogs
6	**Release Calendar** button—Opens the release calendar page for all projects or a specific project

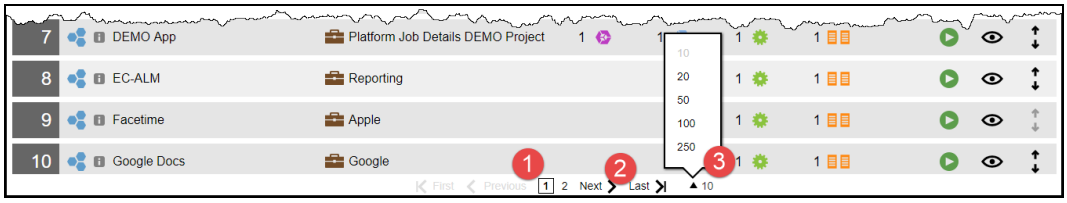
7	Indicates the user who is currently logged in; opens the **Timezone**, **About**, or **Help Center** dialog boxes; or logs you out of ElectricFlow if you choose the **Log Out** option.
	 
	• **Pipeline UI Preference**—Opens a dialog box to let you toggle between the Pipeline Stage View and the Release Kanban View. For details, see Pipeline Views on page 437
	• **Timezone**—Opens a dialog box containing a pulldown menu for selecting the timezone in which you will use the software
	• **About**—Displays a popup containing the build version, UI version, build label, and protocol version of the installed release
	• **Help Center**—Opens the ElectricFlow page in the Electric Cloud Help Center. In this page, you can:
	  • Get started fast with easy video tutorials.
	  • Find helpful explanations in the Knowledge Base.
	  • Ask questions and get answers from the Community.
	  • See the documentation for completing key tasks.

8	Breadcrumbs—Provides context for the current page and links to previous pages.
9	Default filter criteria—Configure the default filter here. **All Run Statuses** is available on run time object lists. See Searching and Filtering on page 19 for filter details; see Object Tags on page 45 for configuring tags.
10	Filter selector—Select the filter to use on this page. Create custom filters here; see Searching and Filtering on page 19 for details.

11	List contents—List content appears here. Columns and content vary based on list type.
12	Pagination controls—See Pagination on page 19 for details.

Pagination

Control the line item pagination in an object list with the pagination controls at the bottom of a list.

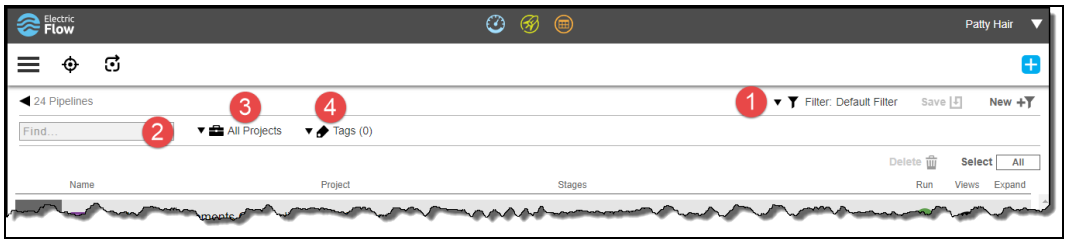


1	Page number links—Click to go directly to the specified page.
2	Quick links—Click to go directly to first, last, next, or previous pages.
3	Number of line items per page—Set the number of line items to display per page.

Searching and Filtering

Object lists support an extensive searching and filtering capability. Each list has both a default filter and user-defined custom filters.

<access>



1	**Filter** dropdown—The filters defined for the current object list. As shipped, each page has a default filter. One or more user-defined custom filters can be manually added to the list. Click the down arrow and select the filter of choice. Once selected, the object list reflects the filter criteria.
2	**Find** box—The current search criteria for the default filter.
3	**All projects** dropdown—Restrict search criteria for the default filter to specified projects. By default, all projects are searched.

4	**Tags** dropdown—Restrict search criteria for the default filter to specified tags. See Object Tags on page 45for further information.

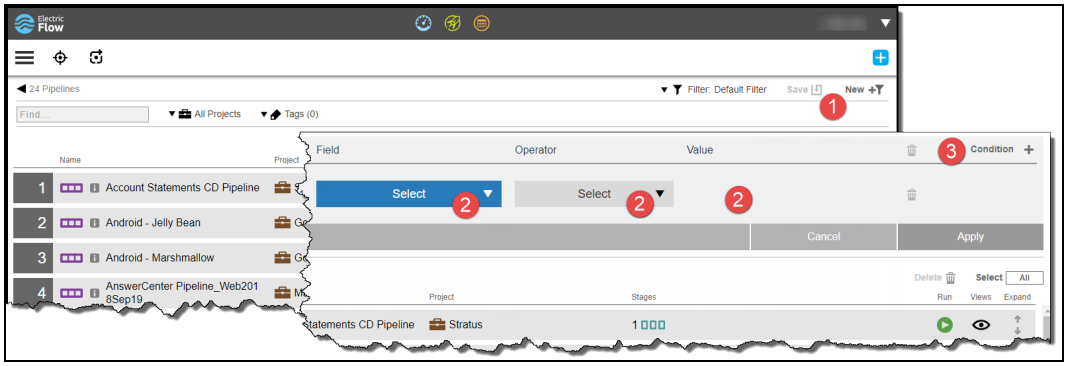
The Default Filter

This filter is used to search object names. Enter all or part of an object name in the **Find** box on the object list. Further restrict your search by specifying projects and tags. Contents of the list dynamically changes based on the criteria. The criteria is active until another one is entered and is cleared when you navigate away from the object list.

Custom Filters

Custom filters give you the ability to define finer grained search criteria than with what the default filter provides. Saved custom filters persist and are available on subsequent visits to the list. The scope of custom filters is the object list. There are no global custom filters.

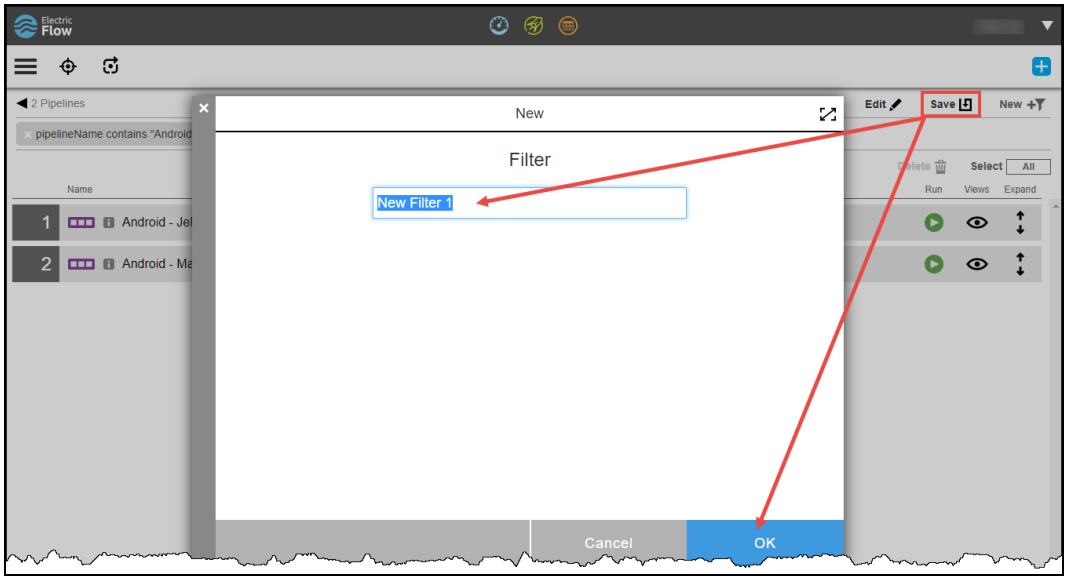
Creating a Custom Filter



1. Click the **New** button.
2. Select **Field**, **Operator**, and **Value**. Operator and value choices are context sensitive based on the field name.
3. Add more conditions to the filter.

The new filter appears in the Filter dropdown, but is cleared when you navigate away from the object list unless you save it.

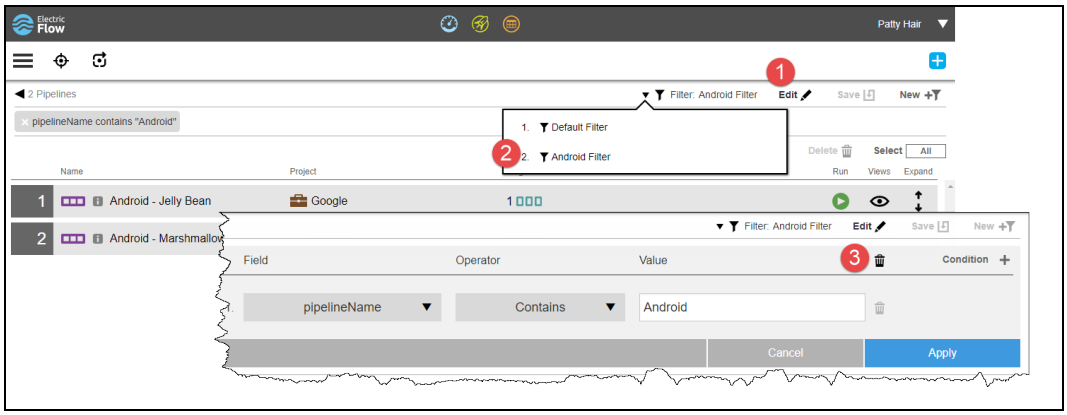
Saving a Custom Filter



Saving a custom filter keeps it in the filter list, available each time you visit the object list page. To save:

1. Click **Save**.
2. Enter the filter name.
3. Click **OK**.

Deleting a Custom Filter



1. Click the **Edit** button.
2. Select the filter to be deleted.
3. Click the trash can button and click **OK** on the confirming **Delete Filter** dialog.

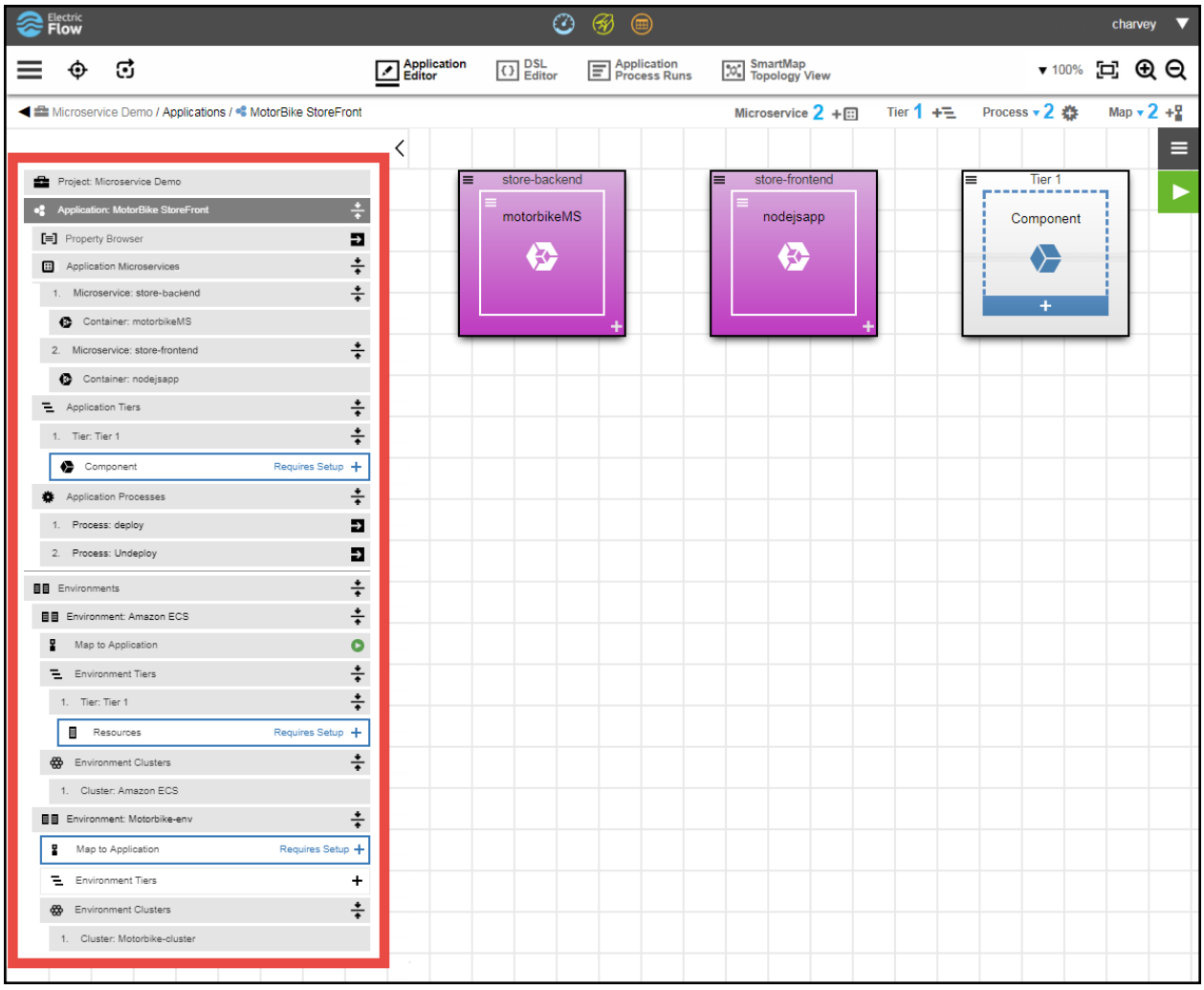
Hierarchy Menu

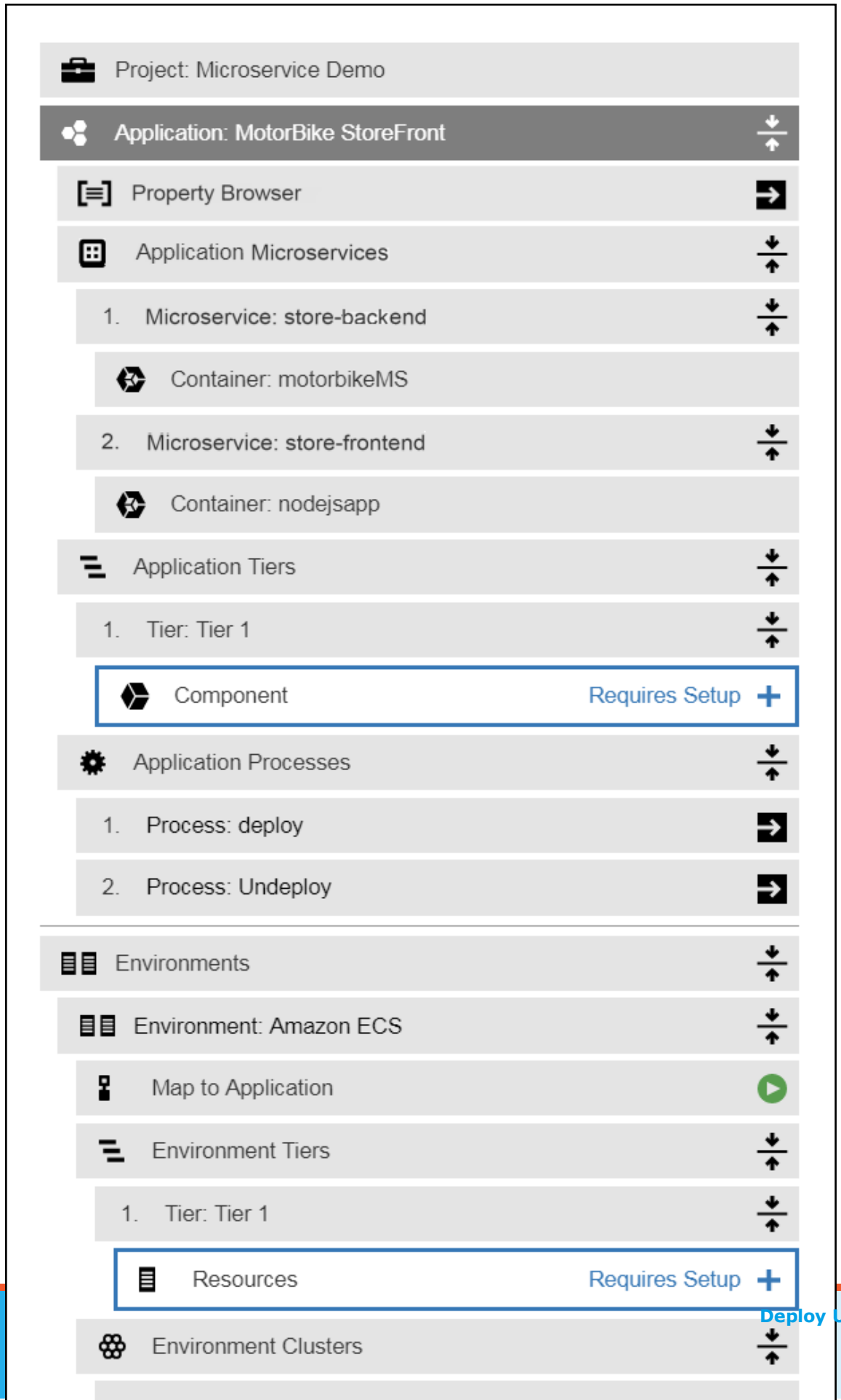
Whether you are starting fresh in ElectricFlow or learning a new concept, you can use the Hierarchy Menu to learn about the ElectricFlow object model structure and product capabilities. This is a wizard-like menu that aids in application, microservice, and environment modeling in ElectricFlow by helping you to visualize the relationships between objects in a project and coaching you along the way by indicating the next actions that are needed. This menu provides an easy-to-understand view of the model structure and where you are within it, highlights missing pieces, and streamlines navigation across objects.

The Hierarchy Menu is available in the Application Editor, the Microservice Editor, the Environment Editor, pipelines, and releases.

Viewing the Hierarchy Menu

To view the Hierarchy Menu for a project, open the Application Editor, the Microservice Editor, or the Environment Editor in that project. The Hierarchy Menu is expanded and visible by default. The following examples show the Hierarchy Menu:




Project: Microservice Demo
Application: MotorBike StoreFront
Property Browser
Application Microservices
1. Microservice: store-backend
Container: motorbikeMS
2. Microservice: store-frontend
Container: nodejsapp
Application Tiers
1. Tier: Tier 1
Component
Requires Setup
Application Processes
1. Process: deploy
2. Process: Undeploy
Environments
Environment: Amazon ECS
Map to Application
Environment Tiers
1. Tier: Tier 1
Resources
Requires Setup
Environment Clusters

The content of the menu is dynamically updated whenever you create or edit an application, microservice, or environment. Child objects that are not yet set up do not have Actions enabled.

Certain Actions (such as clicking the button) invoke popup dialog boxes. Each of these dialog boxes has modal behavior, which means that you must exit it before continuing.

Expand and Collapse Buttons

To collapse the menu, click the (hide) button. To expand the menu, click the (show) button.

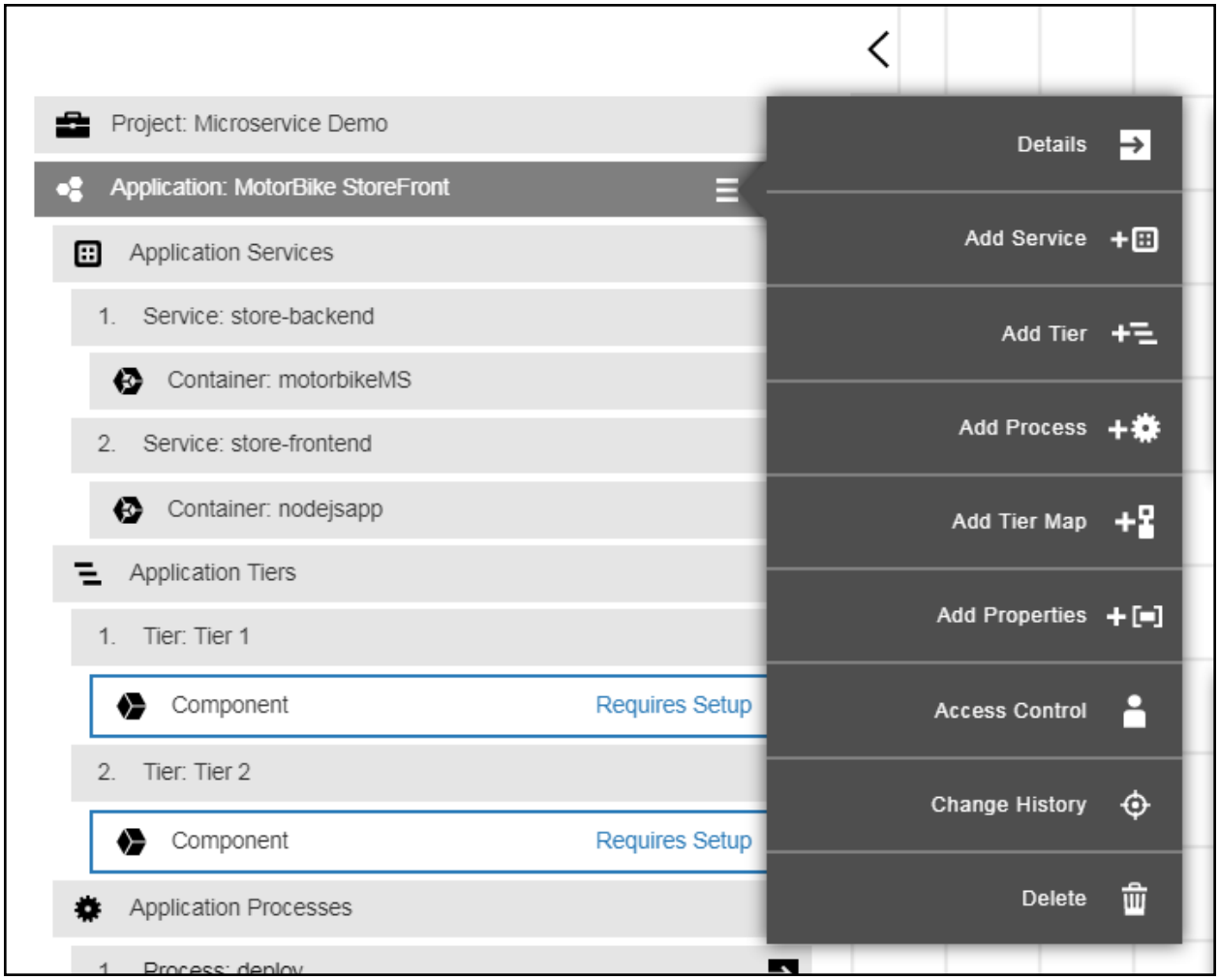
Every object with one or more child objects has a (collapse) button or an (expand) button to collapse or expand the visibility of its child objects. The following example shows the button:

The following example shows the button:

Popover Menu

Some objects (rows) contain a popover menu, which is visible only when you hover over the object. Following is an example of the menu button:

The following example shows the expanded menu after you click :



The menu provides all of the actions for that object that are available in the Visual Editor canvas for that object.

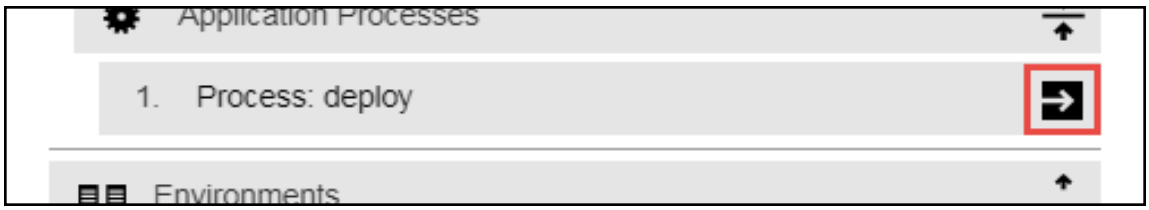
Add Button

Some objects (rows) have an (add) button, which is visible only when you hover over the object:

The button opens the dialog box to add an object.

Visual Editor Button

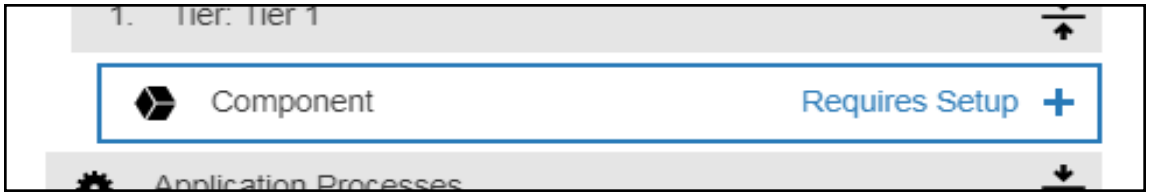
If an object is editable in a Visual Editor, the button appears next to it. The following example shows the button:



The button opens the Visual Editor for that object.

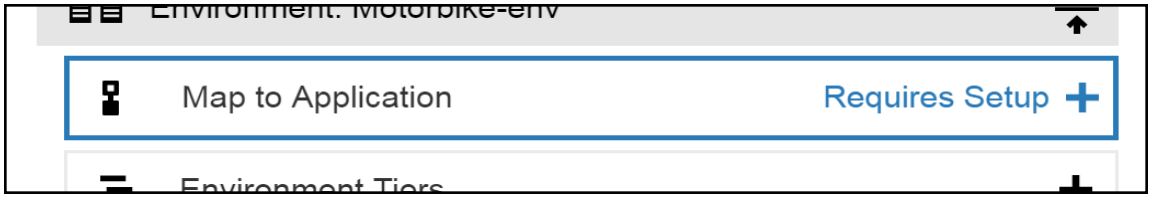
Requires Setup Button

If you have not completed the definition of an Application Component, Tier Map, or Environment Resources, the button appears for the object that needs to be defined. The following example shows the button for an Application Component that needs setup:



The button opens the dialog box for adding an Application Component.

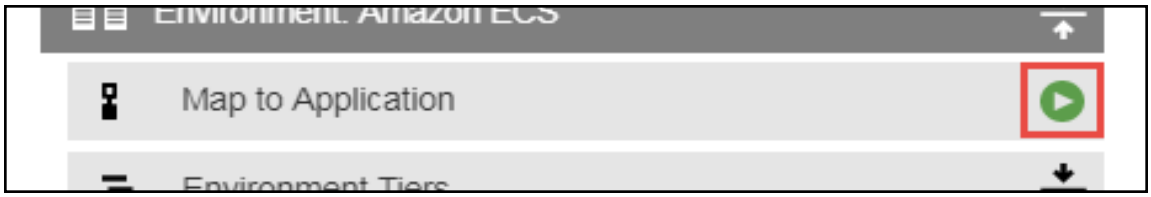
The following example shows the button for a Tier Map that needs setup:



The button opens the dialog box for adding a Tier Map.

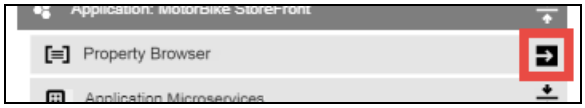
Application Run Button

The (Application Run) button opens the application runtime settings dialog box, which you use to set up and start an application run. The following example shows the button:



Property Browser Button

The (Property Browser) button lets you navigate properties more easily by allowing you view all properties on a hierarchy of objects in a project. The following example shows the button:



For details about this functionality, see Property Browser on page 27.

Property Browser

Properties are a very powerful ElectricFlow feature, but it can be time consuming to navigate to and view all properties associated with an object or a set of objects. Managing deep hierarchies of properties and updating or moving properties can be challenging for a large project.

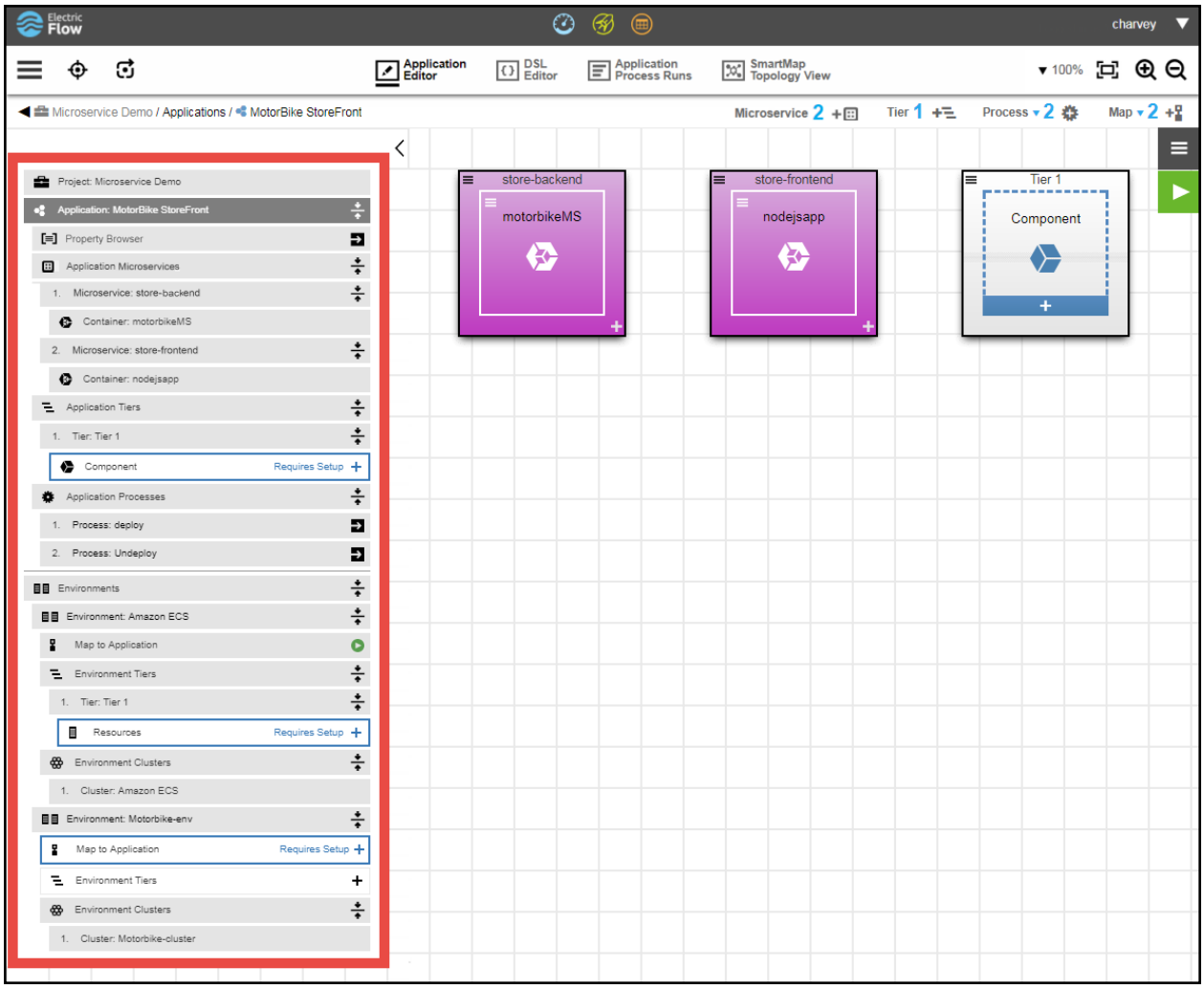
The Property Browser makes navigation of properties easier by letting you view all properties in a hierarchy of objects. Functions to copy and move properties or folders make it simple to create complex structures across an object. This feature saves time for experienced users who otherwise need to browse, search for, and update properties. The Property Browser helps new users learn and understand the value of properties in ElectricFlow by making them easily accessible and viewable.

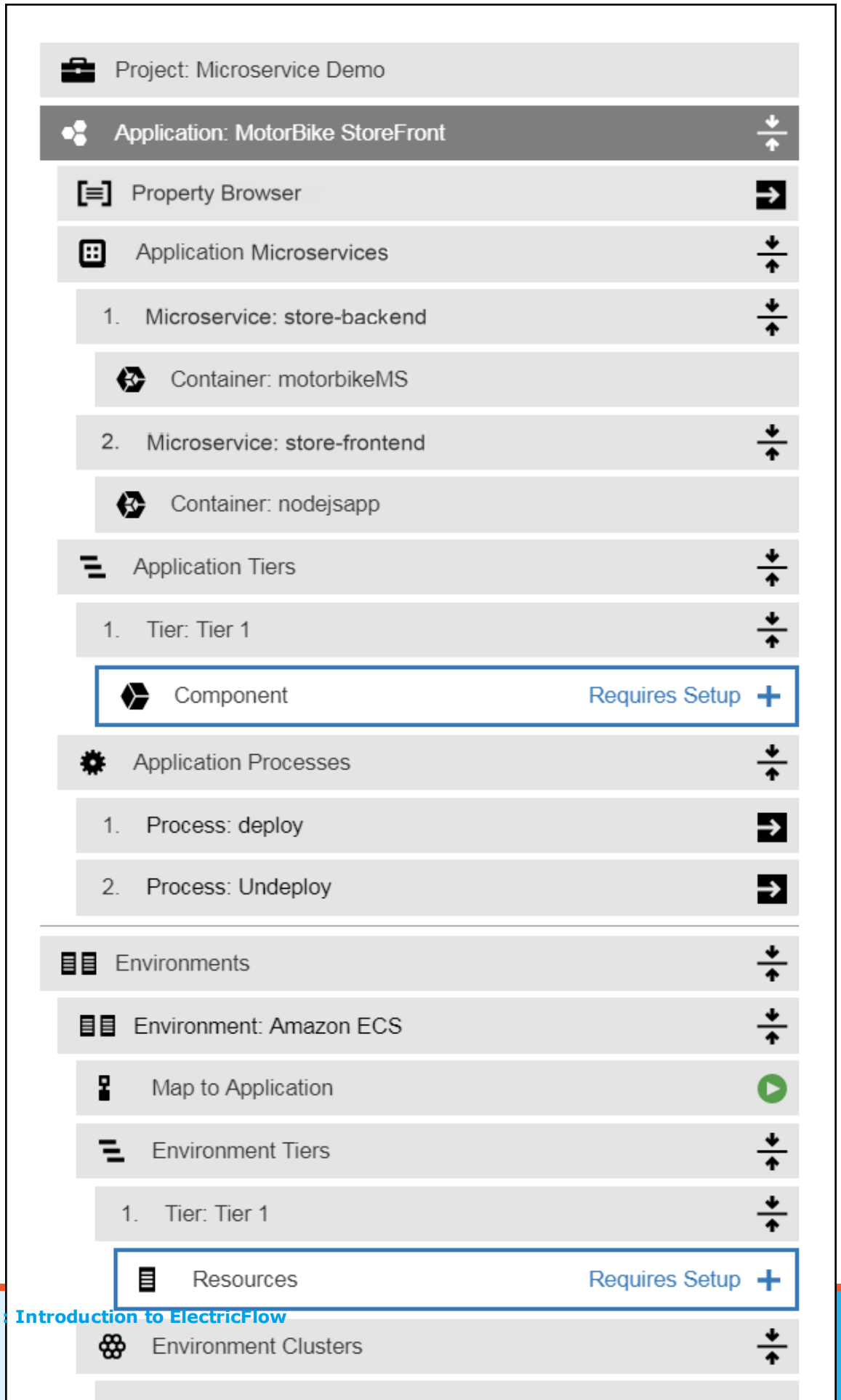
Opening the Property Browser

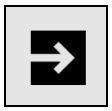
The Property Browser is part of the Hierarchy Menu and is therefore available in the Application Editor, the Microservice Editor, the Environment Editor, the Pipeline Editor, and releases. For more information about the Hierarchy Menu, see Hierarchy Menu on page 22. The Property Browser is also available from within the **Properties** dialog box for a specific object.

Opening the Property Browser from the Hierarchy Menu

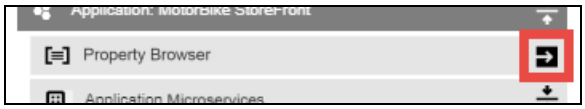
To open the Property Browser from the Hierarchy Menu, you must first open the Application Editor, the Microservice Editor, the Environment Editor, or the Pipeline Editor in that project. The Hierarchy Menu is expanded and visible by default:


Electric Flow
charvey
Application Editor
DSL Editor
Application Process Runs
SmartMap Topology View
100%
Microservice Demo / Applications / MotorBike StoreFront
Microservice 2
Tier 1
Process 2
Map 2
Project: Microservice Demo
Application: MotorBike StoreFront
Property Browser
Application Microservices
1. Microservice: store-backend
Container: motorbikeMS
2. Microservice: store-frontend
Container: nodejsapp
Application Tiers
1. Tier: Tier 1
Component
Requires Setup
Application Processes
1. Process: deploy
2. Process: Undeploy
Environments
Environment: Amazon ECS
Map to Application
Environment Tiers
1. Tier: Tier 1
Resources
Requires Setup
Environment Clusters
1. Cluster: Amazon ECS
Environment: Motorbike-env
Map to Application
Requires Setup
Environment Tiers
Environment Clusters
1. Cluster: Motorbike-cluster
store-backend
motorbikeMS
store-frontend
nodejsapp
Tier 1
Component

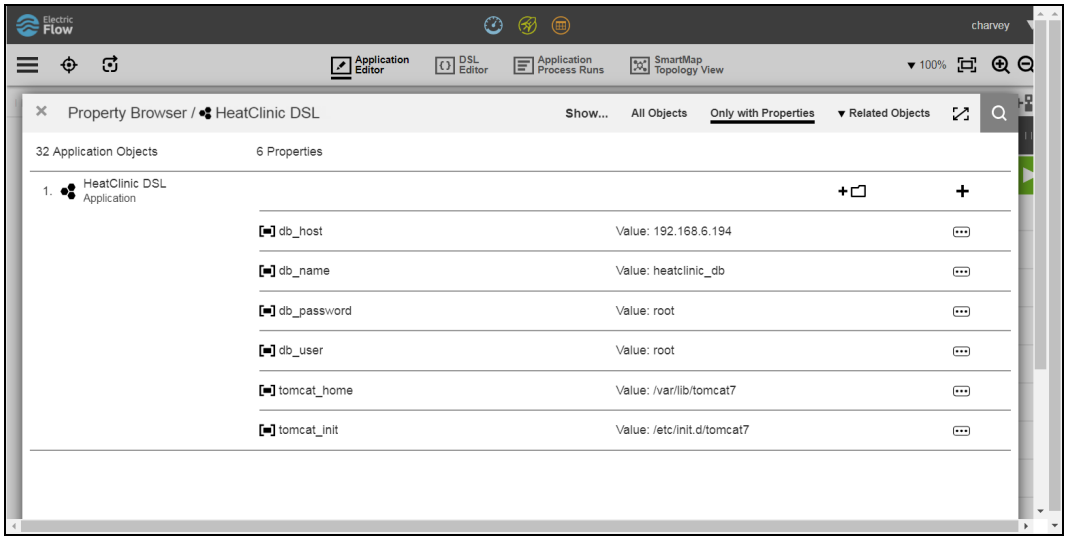

Project: Microservice Demo
Application: MotorBike StoreFront
Property Browser
Application Microservices
1. Microservice: store-backend
Container: motorbikeMS
2. Microservice: store-frontend
Container: nodejsapp
Application Tiers
1. Tier: Tier 1
Component
Requires Setup
Application Processes
1. Process: deploy
2. Process: Undeploy
Environments
Environment: Amazon ECS
Map to Application
Environment Tiers
1. Tier: Tier 1
Resources
Requires Setup
Environment Clusters



Then click the (Property Browser) button in the Hierarchy Menu:

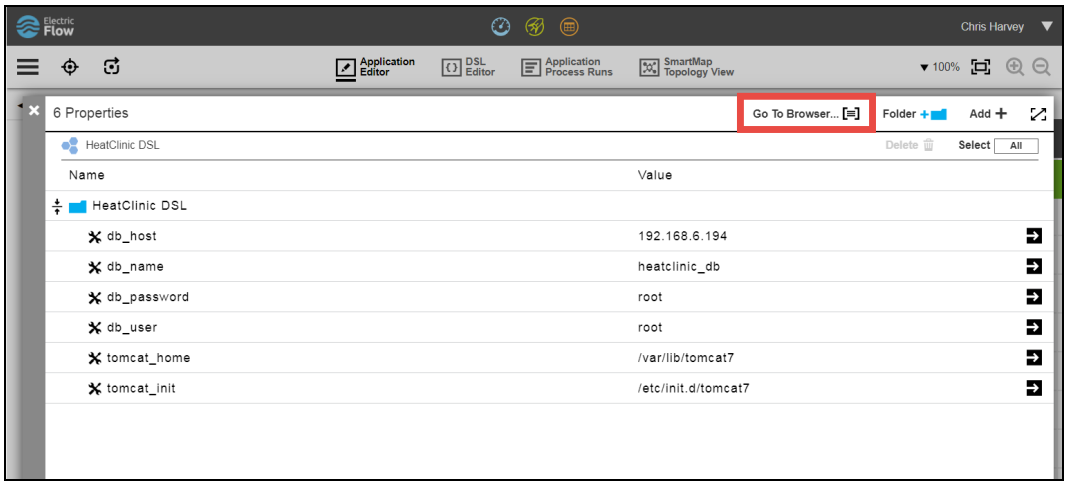


The Property Browser opens. For example:



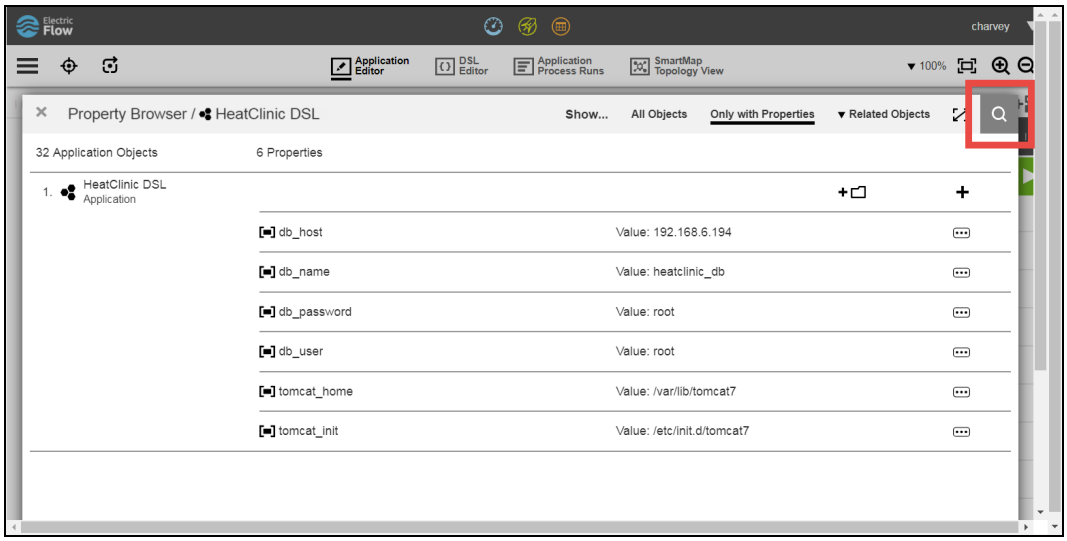
Opening the Property Browser from a Properties Dialog Box

To open the Property Browser from within the **Properties** dialog box for a specific object, click the **Go To Browser** button. For example:

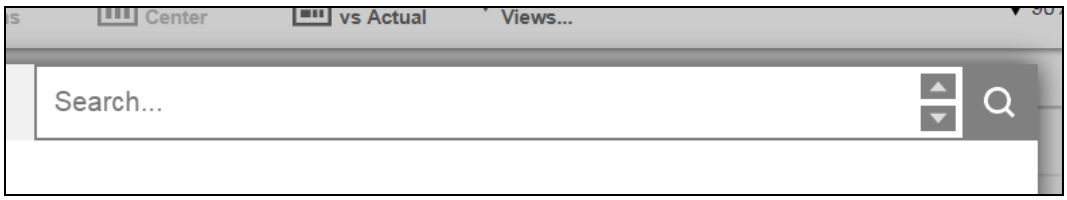


Searching for Properties

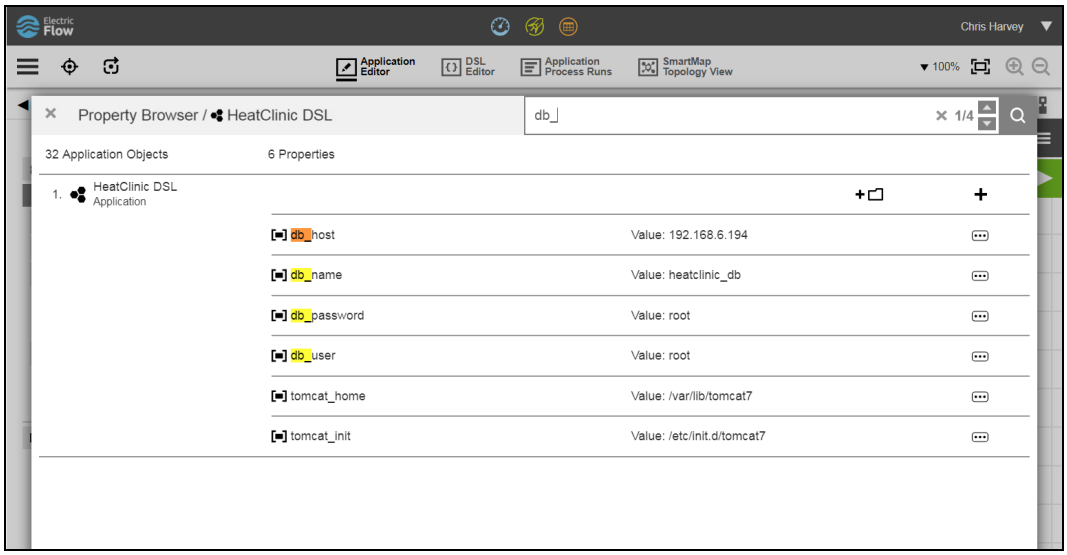
The (Toggle Search Form) button lets you search for properties in the project:



Clicking the button opens a field for entering search terms:

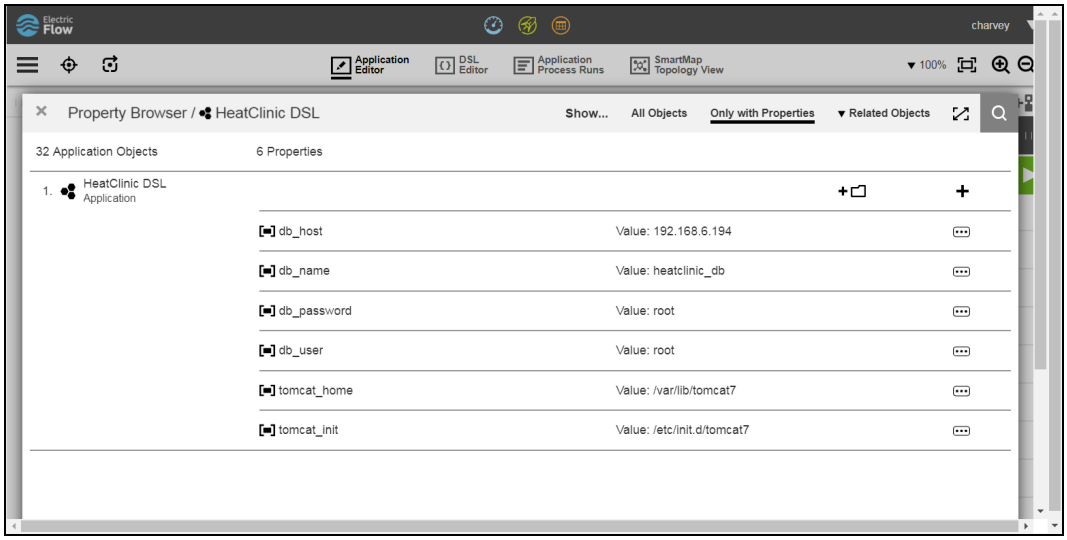


To start the search, simply type the search terms into the field. As you type, the search hits are highlighted in color in the property list. For example:



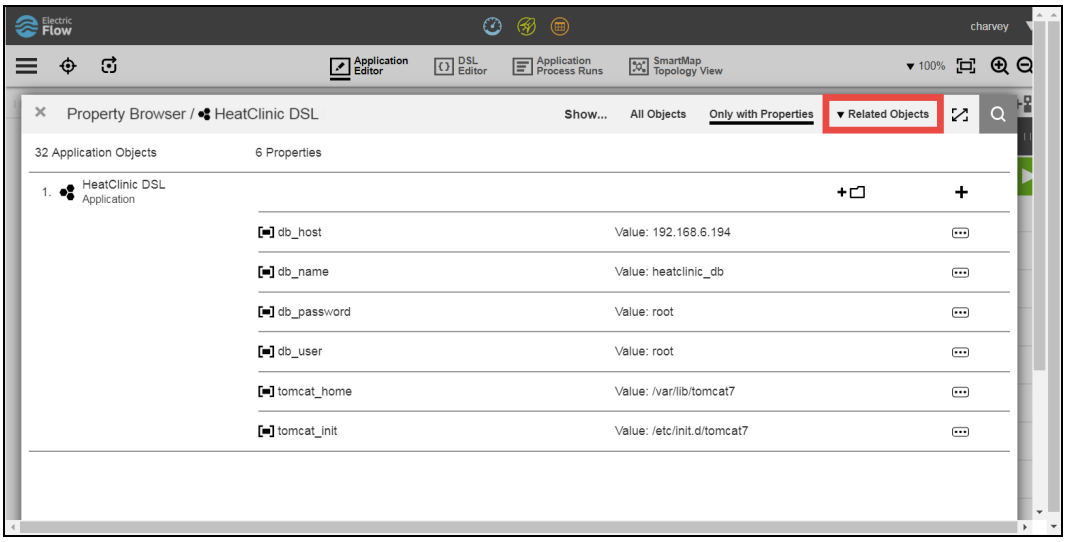
Filtering Out Objects with No Attached Properties

By default, all objects appear in the list. If at least one object in the project has one or more properties, then the **Show...** button, **All Objects** button, and the **Only with Properties** button let you toggle between a view of all objects in the project or a view of just the objects with properties:

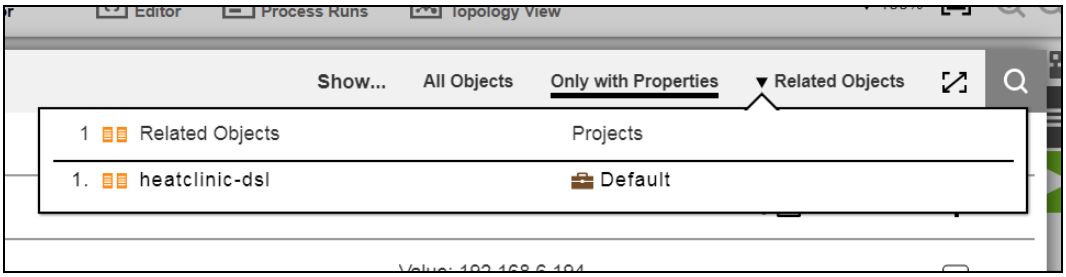


Viewing or Navigating to Related Objects

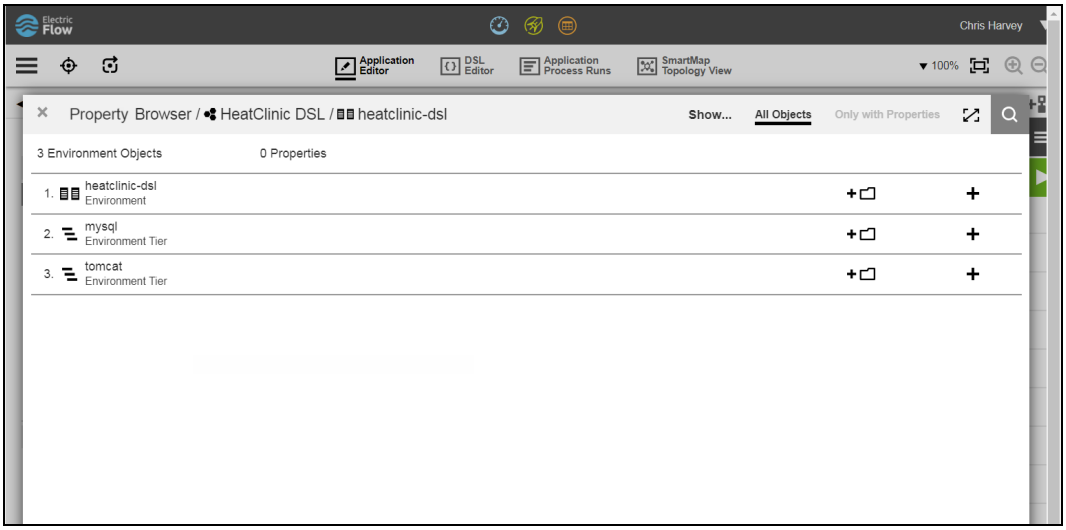
To view all objects (such as environments) that are related to the properties in this project, click the **Related Objects** button. For example for viewing the properties in an application, if the application is mapped to various environments, you can view those objects as well as their child objects:



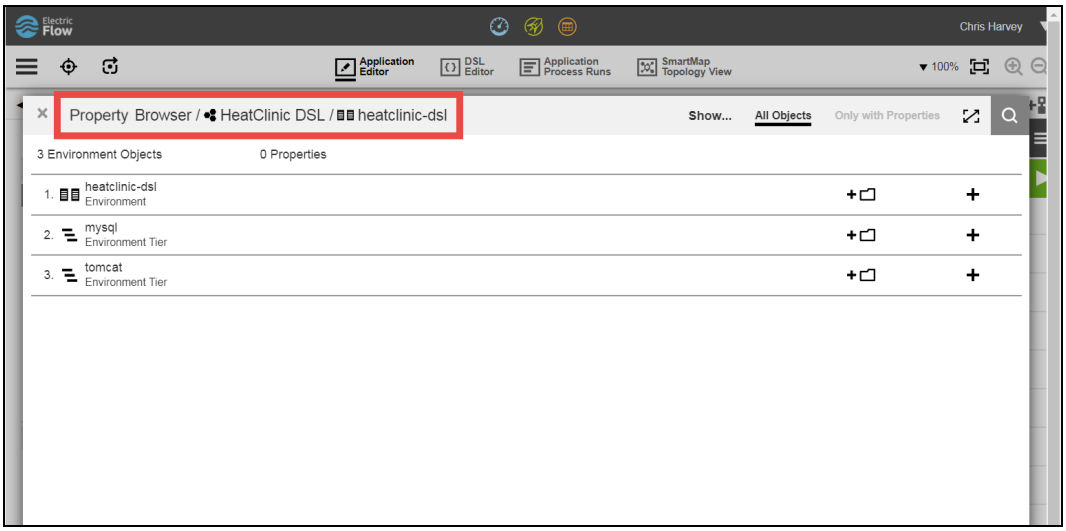
Following is an example of a list of related objects that appears:



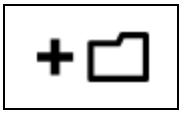
To navigate to a related object in the list, simply click the object. For example, if you click the **heatclinic-dsl** environment in the screenshot above, the properties for that environment appear:



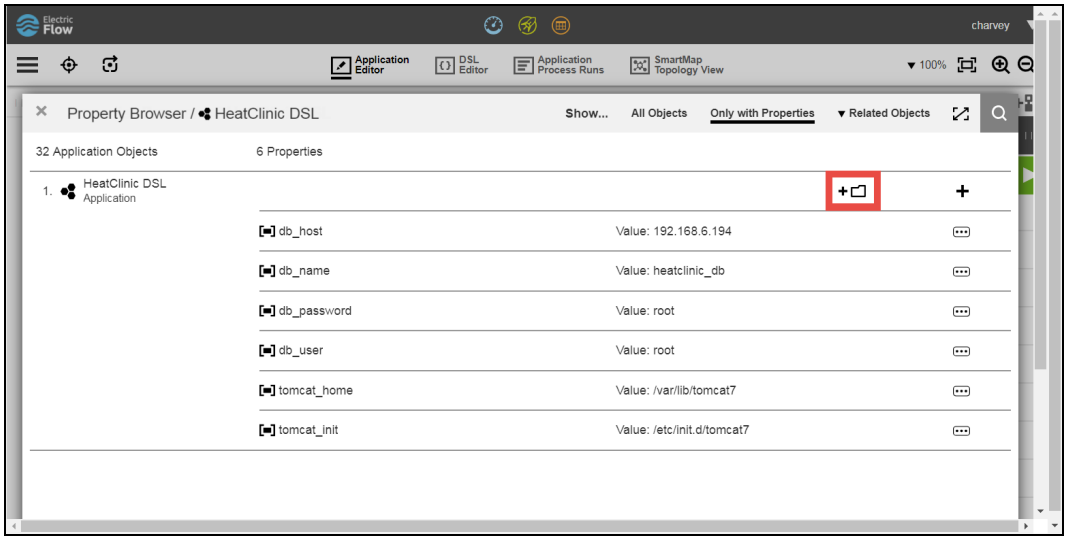
Notice that the breadcrumbs at the top of the dialog box are updated to indicate the navigation path that you used to browse to your current location:



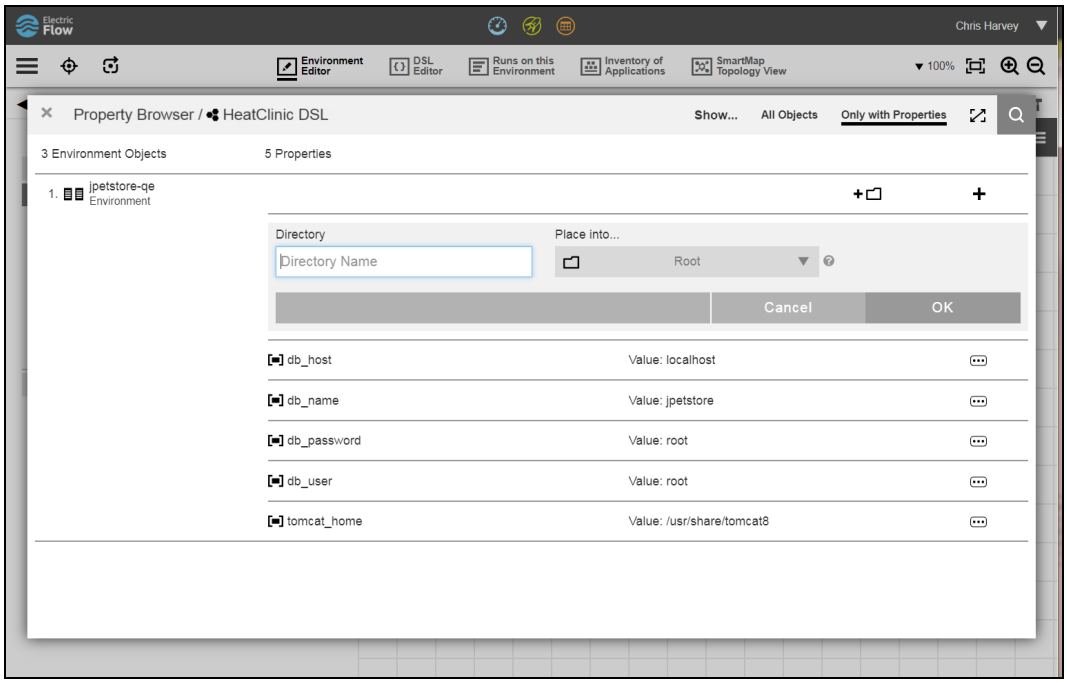
Adding a Properties Directory



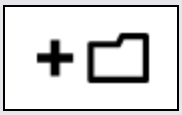
The (Add Directory) button lets you create a hierarchy of directories (property sheets) for properties in the project. The following example shows the button:



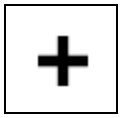
Clicking this button opens a dialog box for entering the details for the new directory:

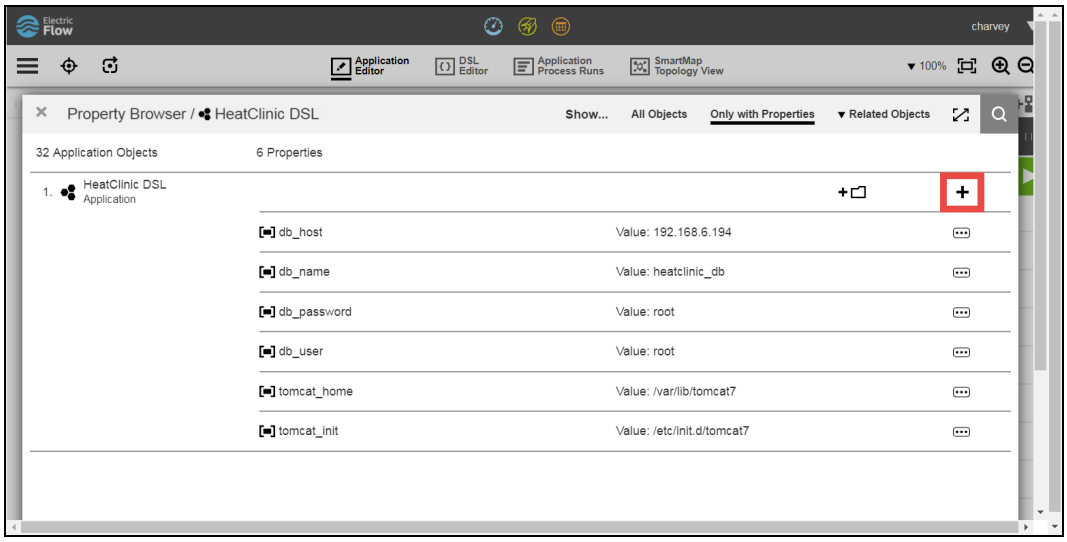


Complete the items in the dialog box as follows:

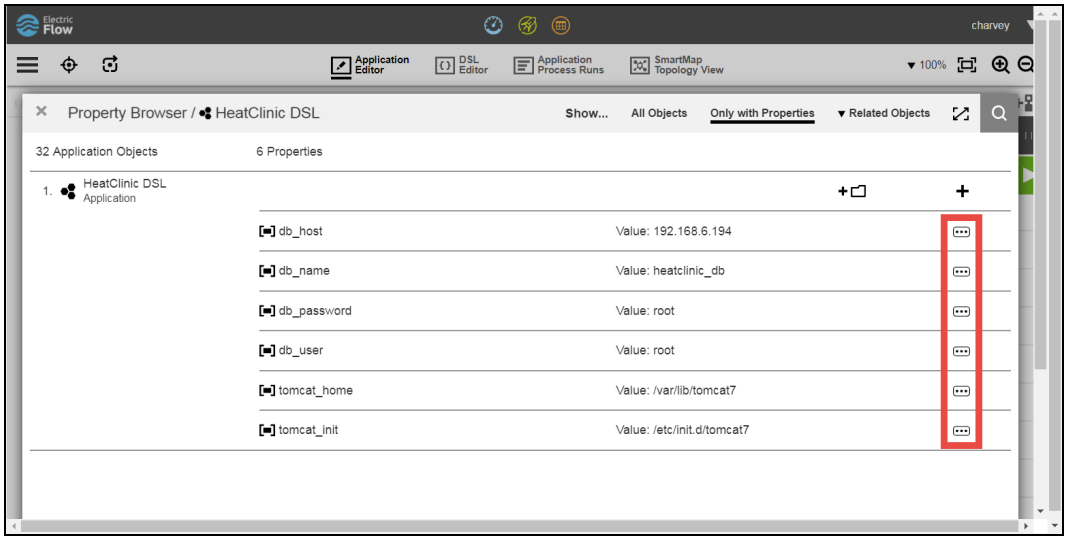
Field or Menu	Description
Directory Name	Name of the property. For naming guidelines, see Using Special Characters in ElectricFlow Object Names.
Place into...	Directory (property sheet) to contain the property. While you build out the properties that belong to an object, this menu lets you edit the structure or hierarchy of the object's property data. Properties are in the object's `Root` directory (the top of its property hierarchy) by default. If this menu is grayed out, then you have not yet created any directories to populate this menu with other choices. If you want to create a directory, click the  (Add Directory) button in this dialog box.

Creating or Editing a Property

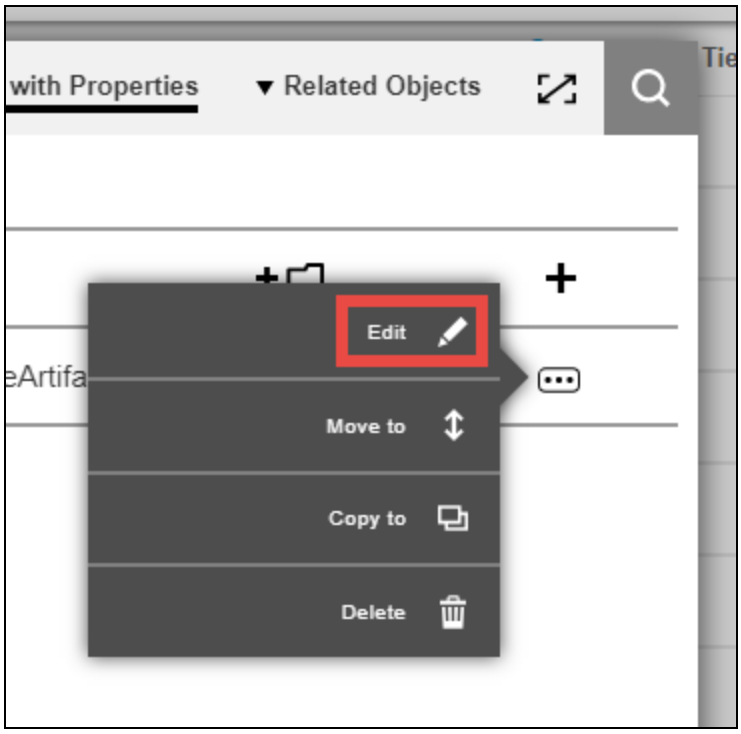
To add a property, click the  (Add Property) button:



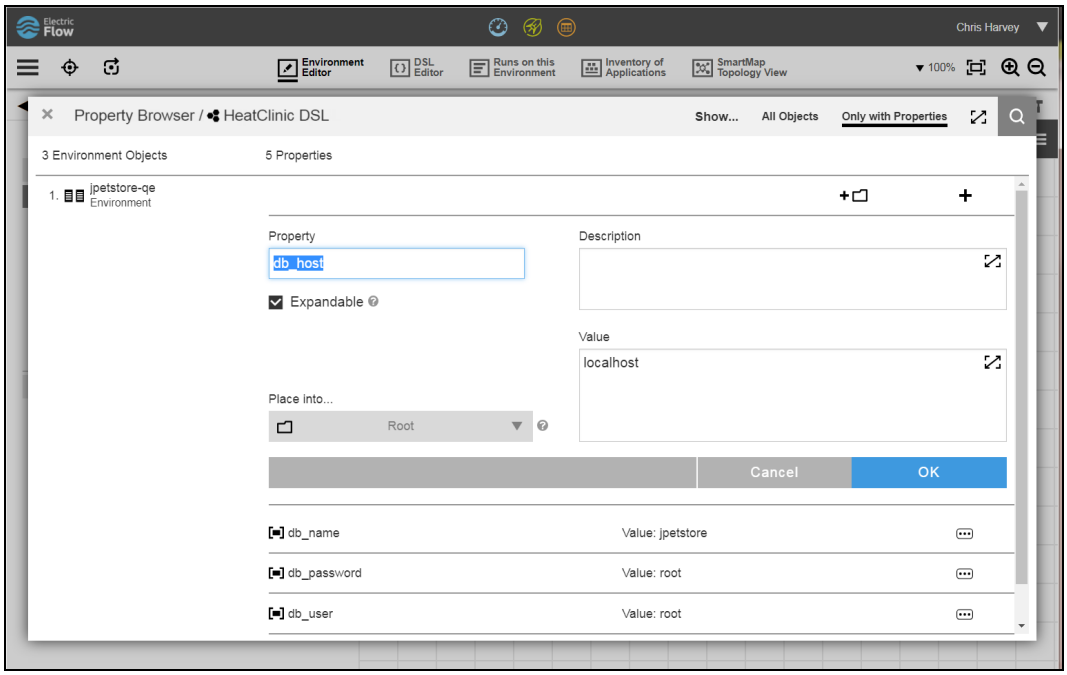
Or to edit a property, click its corresponding button:



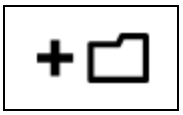
And then click **Edit** from the popup menu that appears:



Either of these options opens a dialog box for entering the property details. For example:



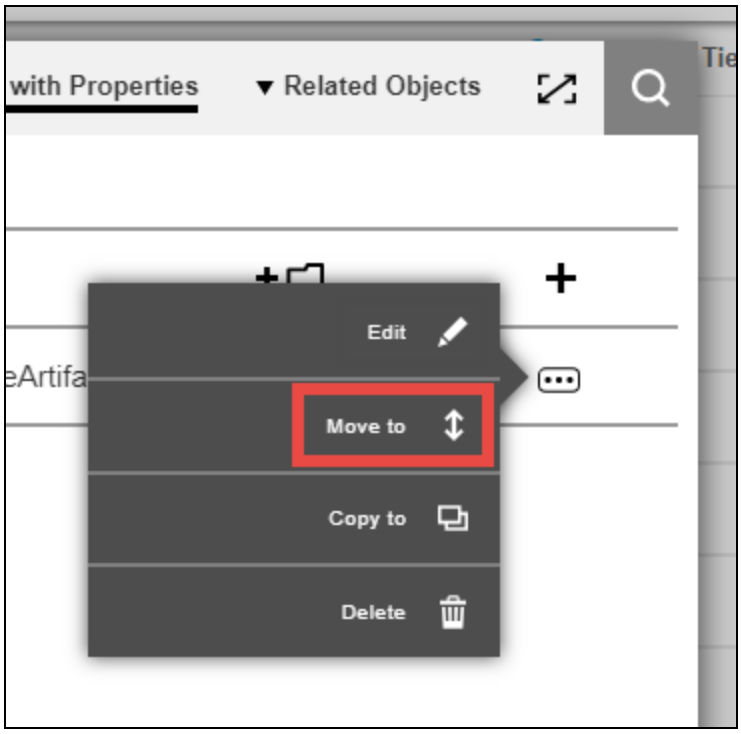
For either creating or editing a property, complete the items in the dialog box as follows:

Field or Menu	Description
Property Name	Name of the property. For naming guidelines, see Using Special Characters in ElectricFlow Object Names on page 1
Description	Description of the property. The contents of this field are ignored by ElectricFlow.
Value	Value for the property.
Expandable	Checkbox to allow the property to be referenced via expansion in other properties. For example, let's say that you create a property named `foo` with a value of `hello $[bar]` and then create an expandable property named `bar` with a value of `world`. If you reference `foo` (by using `$[foo]` or `ectool getProperty foo`), the value `hello world` is returned.
Place into...	Directory (property sheet) to contain the property. While you build out the properties that belong to an object, this menu lets you edit the structure or hierarchy of the object's property data. Properties are in the object's `Root` directory (the top of its property hierarchy) by default. If this menu is grayed out, then you have not yet created any directories to populate this menu with other choices. If you want to create a directory, click the  (Add Directory) button in this dialog box.

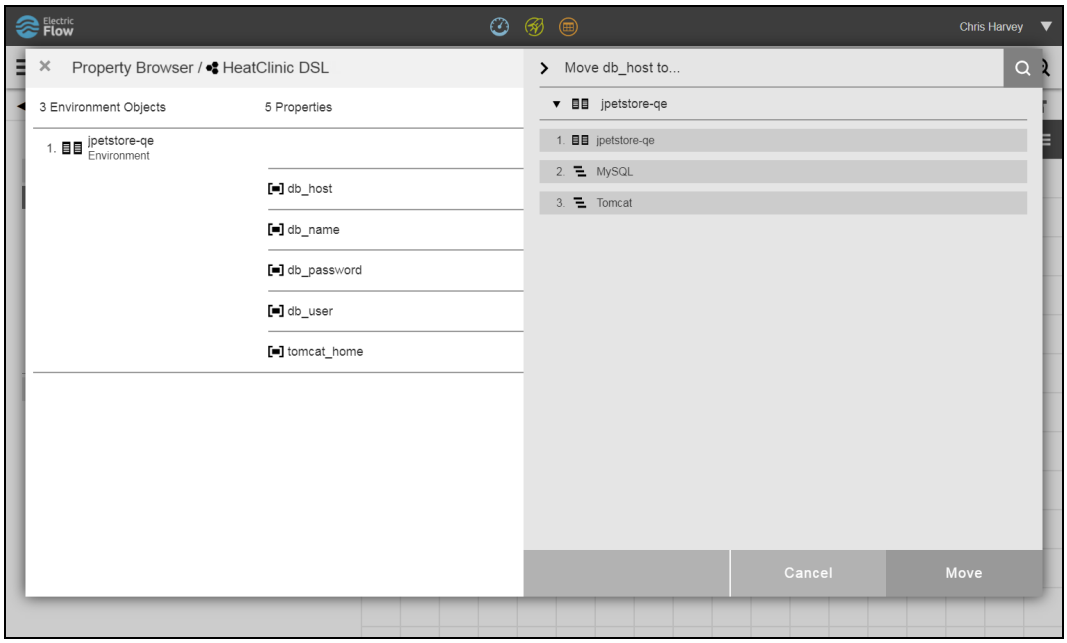
And then click **OK** to save your changes.

Moving a Property

To move a property to another directory, click its corresponding  button, and then click **Move to** from the popup menu:



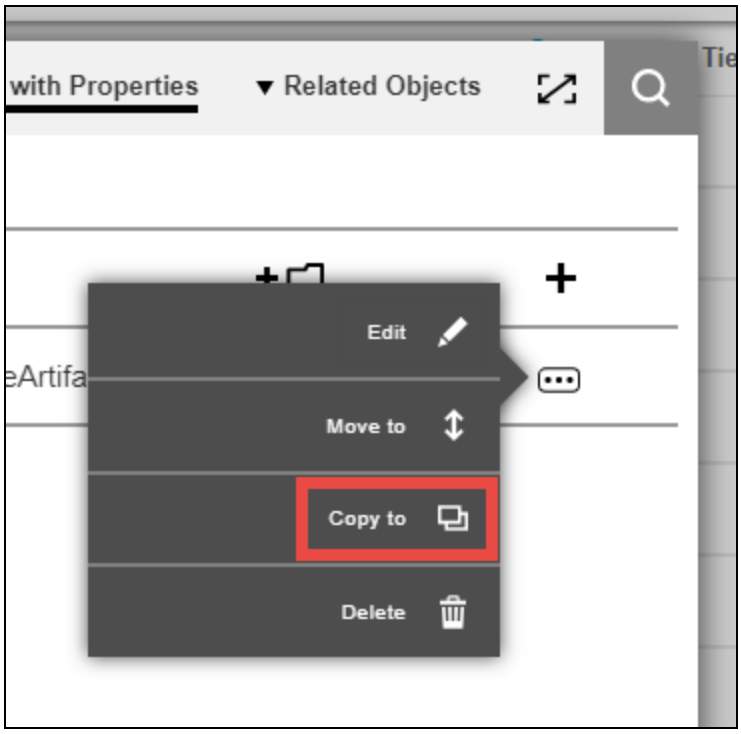
The **Move *<property_name>* to...** dialog box appears. For example:



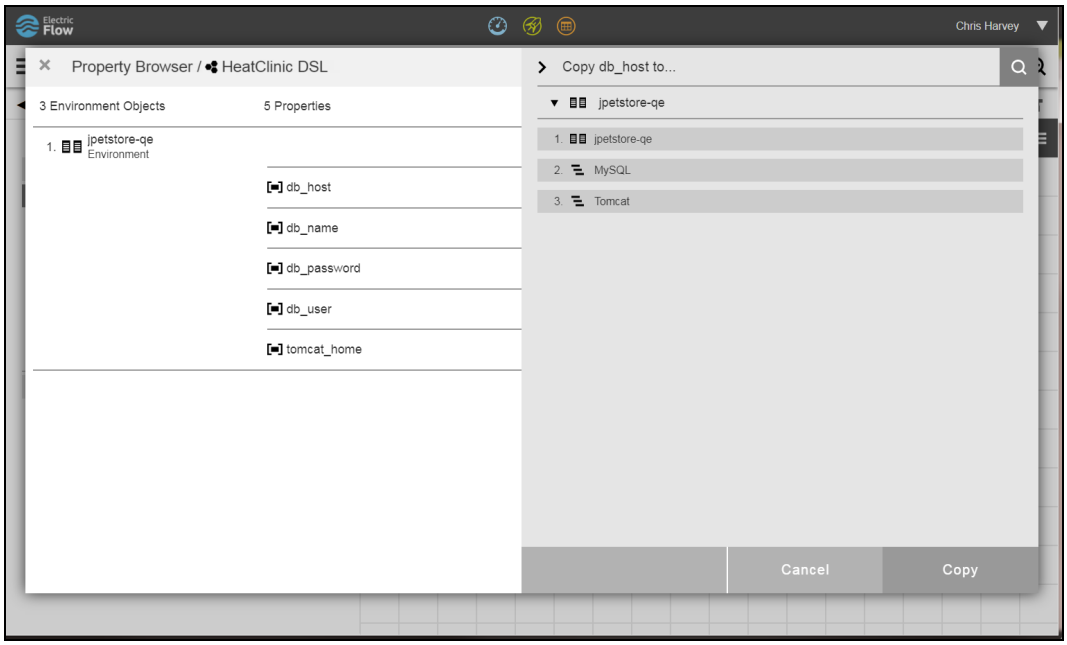
Then in the dialog box, click the new location (another object or directory), and then click **Move**.

Copying a Property

To create a property by copying an existing property, click its corresponding  button, and then click **Copy to** from the popup menu:



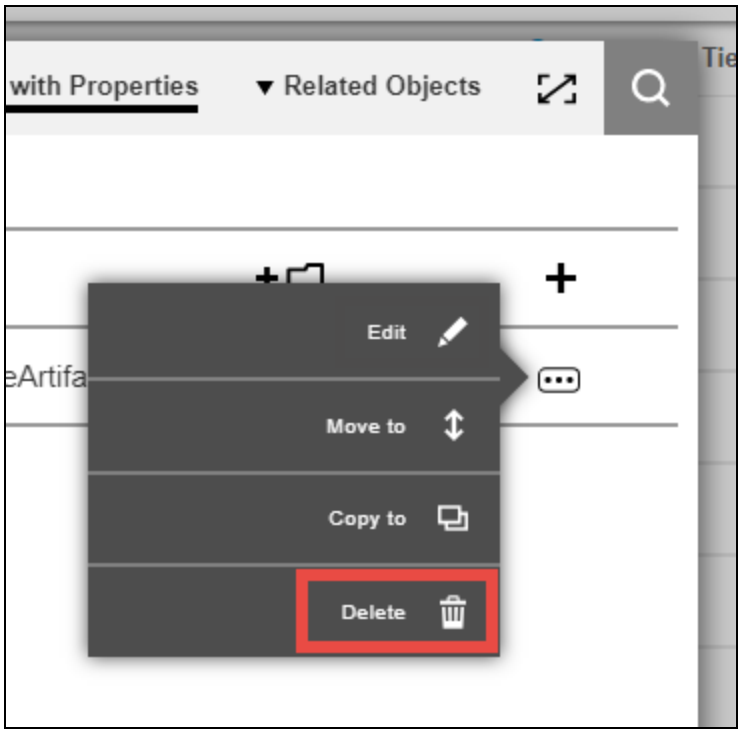
The **Copy *<property_name>* to...** dialog box appears. For example:



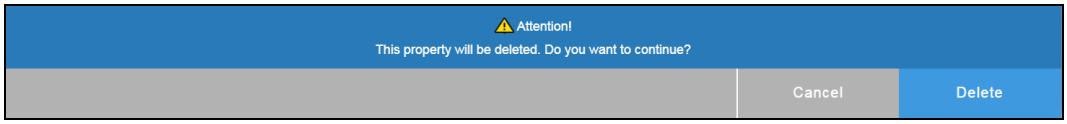
Then in the dialog box, click the new location (another object or directory), and then click **Copy**.

Deleting a Property

To delete a property, click its corresponding  button, then click **Delete** from the popup menu:



A confirmation dialog box appears:



Then click **Delete** to confirm the deletion.

Property Browser Usage Examples

Following are a few examples of how you can use the Property Browser.

Determining the Location of a Property

Let's say that you have an application that consists of various application tiers, components, and application and component processes. The application is mapped to a set of environments that consist of various environment tiers. You can use the Property Browser to view all properties (and current values) associated with these objects comprising the application and its mapped environments. This helps you determine where a property is located so that you can reference it while you continue authoring.

Troubleshooting Deployments

For the application and setup described above, if the deployment has issues, you can use the Property Browser to troubleshoot why a run of that application behaved in an unexpected way. This could be traced to a number of configuration properties defined across the objects, and you can use the Property Browser to quickly check a number of properties to speed up investigation and debugging.

Adding a Layer of Property Sheets

If you want to change the property sheet and the nested structure of properties and add a new layer of property sheets, rather than deleting and recreating properties individually, you can do this easily with the Property Browser.

Searching-and-Replacing Properties

You can search-and-replace numerous properties that are stored in different locations.

Projects in ElectricFlow

A project is a top-level container within ElectricFlow. Most information about software production processes, such as procedures, schedules, jobs, and workflows are contained within a project.

Project Purposes

- Projects let you create separate work areas for different purposes or groups of people so they do not interfere with each other.

  For example, different projects can reuse the same names internally without conflict, and each project has its own access control that determines who can use and modify the project.

  In a small organization, you might choose to keep all work in a single project, but in a large organization, you might want to use projects to organize information and simplify management.

- Projects simplify sharing.

  You can create library projects containing shared procedures and invoke these procedures from other projects. After creating a library project, you can copy it easily to other ElectricFlow servers to create uniform processes across your organization.

Default Projects

ElectricFlow includes the following projects:

- EC-Utilities—Contains procedures that you can use to perform basic ElectricFlow tasks.

  Also, you can use these procedures as templates by copying them to another project and then modifying them for your purpose. By default, only the "admin" user has execute privileges on the EC-Utilities project. The admin user can enable privileges for additional users or groups.

- EC-Examples—Contains templates for procedures that perform basic ElectricFlow tasks.
- Electric Cloud—Contains procedures to manage the ElectricSentry Sentry Monitor for schedules and also contains ElectricFlow global reports.

Procedures in each of these projects are maintained by Electric Cloud.

Creating or Editing Projects

You can create or edit a project through either the Deploy web UI or the Automation Platform web UI. For details, see

- Creating or Editing a Project in the Deploy Web UI on page 56
- Project—Create or Edit a Project in the Automation Platform Web UI on page 1187

Object Tags

Tagging provides a way to group related objects with a user-defined term. Start by configuring tags on objects of interest then use tag names as filter criteria. Runtime objects inherit tags from their definition object counterparts; the table in Object Inheritance on page 48 shows this relationship.

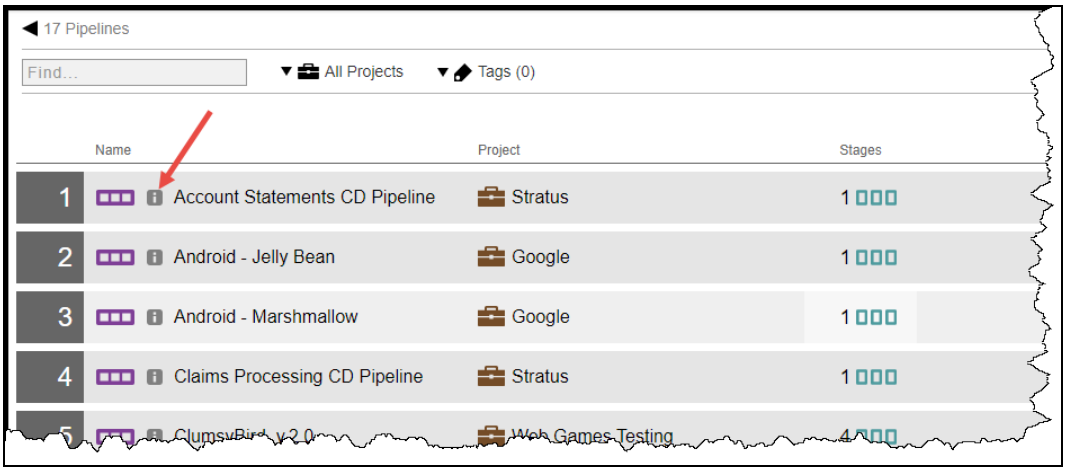
The following objects can have tags:

Application	Pipeline	Stage
Component (master | application)	Procedure	Step (Process)
Environment	Process	Step (Procedure)
EnvironmentTemplate	Project	Task
FlowRuntime	Release	Workflow
FlowRuntimeState	Resource	WorkflowDefinition
JobStep	ResourceTemplate	Artifact
Job (backed by root jobStep)	Microservice	ArtifactVersion

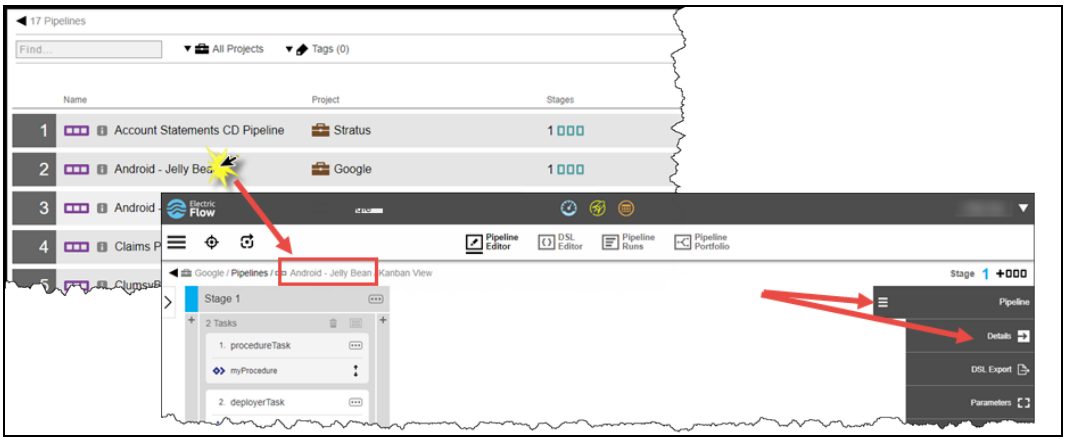
Configuring Tags on a Object Definition

Use the **Tags** dialog to configure tags on an object definition. This dialog is available from these areas:

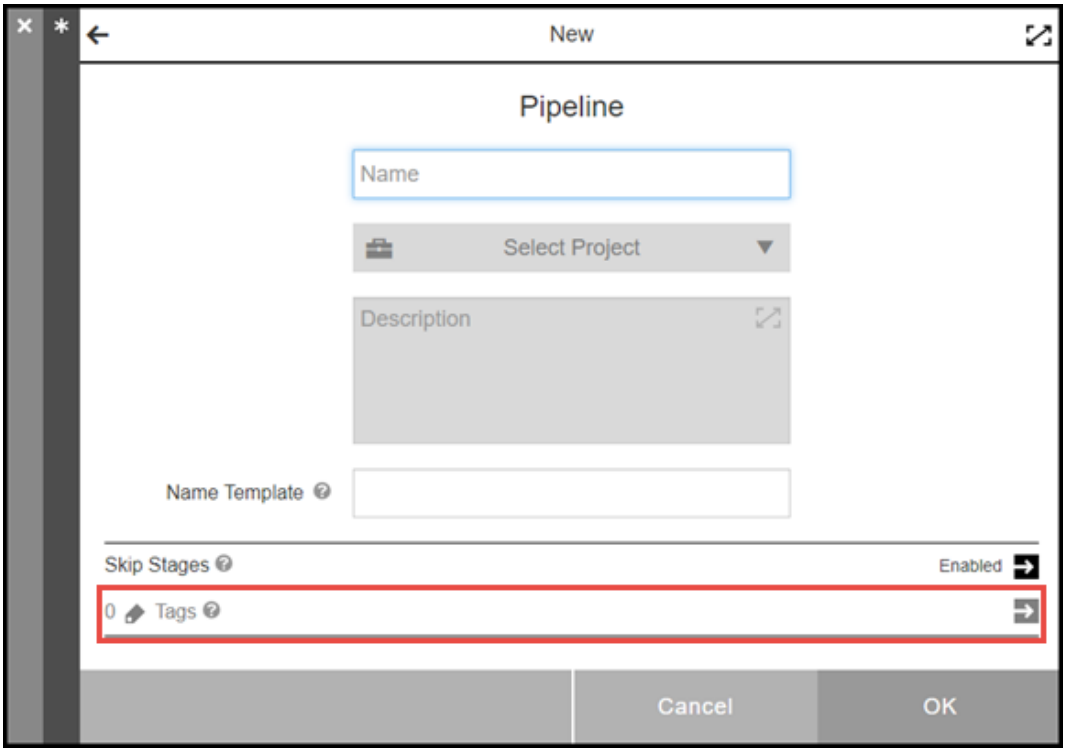
- On an object list page—Click the information icon for the desired object.



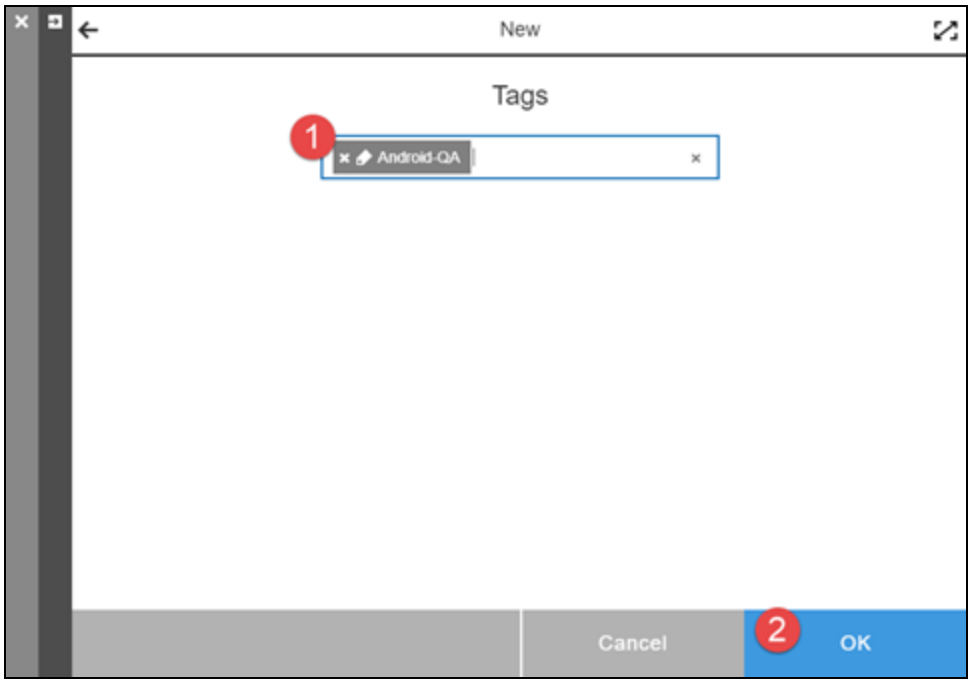
- On a object definition page—Click the Actions menu  and select **Details**, then select **Tags**.



- On a create new object page—Click the right arrow on the **Tags** line.



From here, the Tags dialog opens to configure the set of tags for the object.



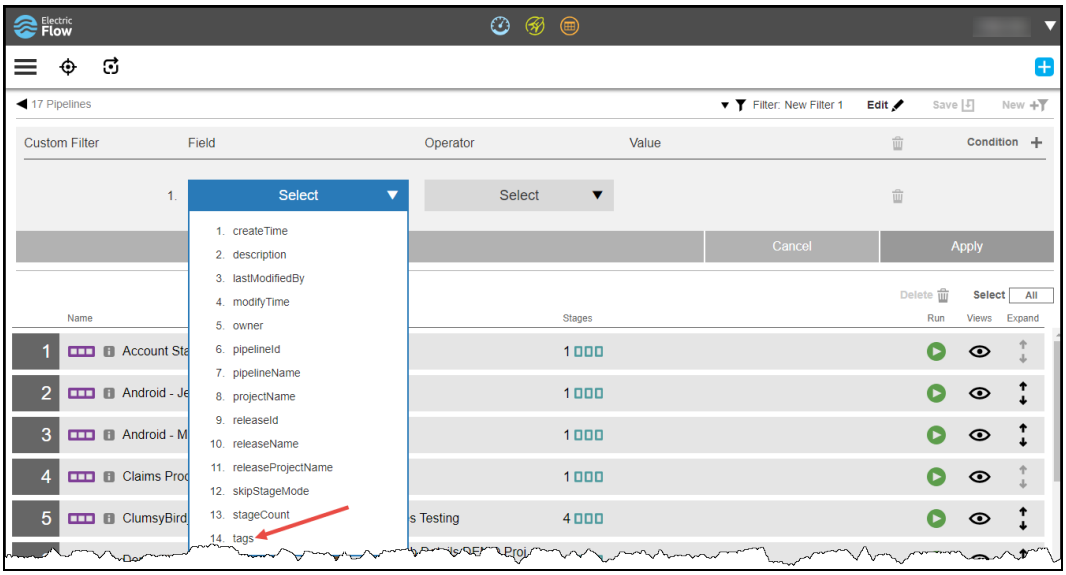
1. For each tag to be configured, click into the **Tags** box and perform one of the following actions:
    - Select an existing tag: Enter a space to see all available tags or enter a string to see a filtered list. Select the desired tag.
    - Create a new tag: Enter the new tag name followed by **Enter**. Tag names are not case-sensitive.
        - Use only alpha-numeric, - (dash), or _ (underscore) in tag names.
        - Length is limited to 255 characters.
    - Remove an existing tag: Click on its **X** to remove a tag.
2. Click **OK** to save the configuration.

Using Tags

To view tagged objects, specify filter criteria on object definition lists or their associated runtime objects in one of the following ways:

- Use the **Tags** selector

- Set up a custom filter specifying one or more criteria with the **tags** field and then invoke the filter. See Searching and Filtering on page 19 for more information about custom filters.



For details about using tags in DevOps Insight dashboards, see the section for your dashboard in DevOps Insight on page 691.

Object Inheritance

You set up tags on the definition objects and then the related runtime objects inherit them according to the table below.

Operation	Definition Object (non-pipeline context)	Definition Object (pipeline context)	Target Runtime Object
Start Release	N/A	Release CallingFlowRuntimeState (task which triggered the pipeline run) CallingPipelineFlowRuntime (Pipeline flowRuntime associated with the release task)	PipelineFlowRuntime
Run Pipeline	N/A	Pipeline CallingFlowRuntimeState (task which triggered the pipeline run) CallingPipelineFlowRuntime (Pipeline flowRuntime associated with the pipeline task)	FlowRuntime

Operation	Definition Object (non-pipeline context)	Definition Object (pipeline context)	Target Runtime Object
Run Process	Application Process Environment Microservice	FlowRuntimeState (task which triggered the application run) FlowRuntime (Pipeline flowRuntime) Application Process Environment Microservice	Job
Run Procedure	Procedure	FlowRuntimeState (task which triggered the application run) FlowRuntime (Pipeline flowRuntime) Procedure Application Process Environment Microservice	Job
Run Workflow	Workflow Definition	FlowRuntimeState (task which triggered the application run) FlowRuntime (Pipeline flowRuntime) Workflow Definition	Workflow
Provision Environment	Environment Template	N/A	Environment
Provision Resource Pool	Resource Template	N/A	Resource

Chapter 2: Deployment Automation

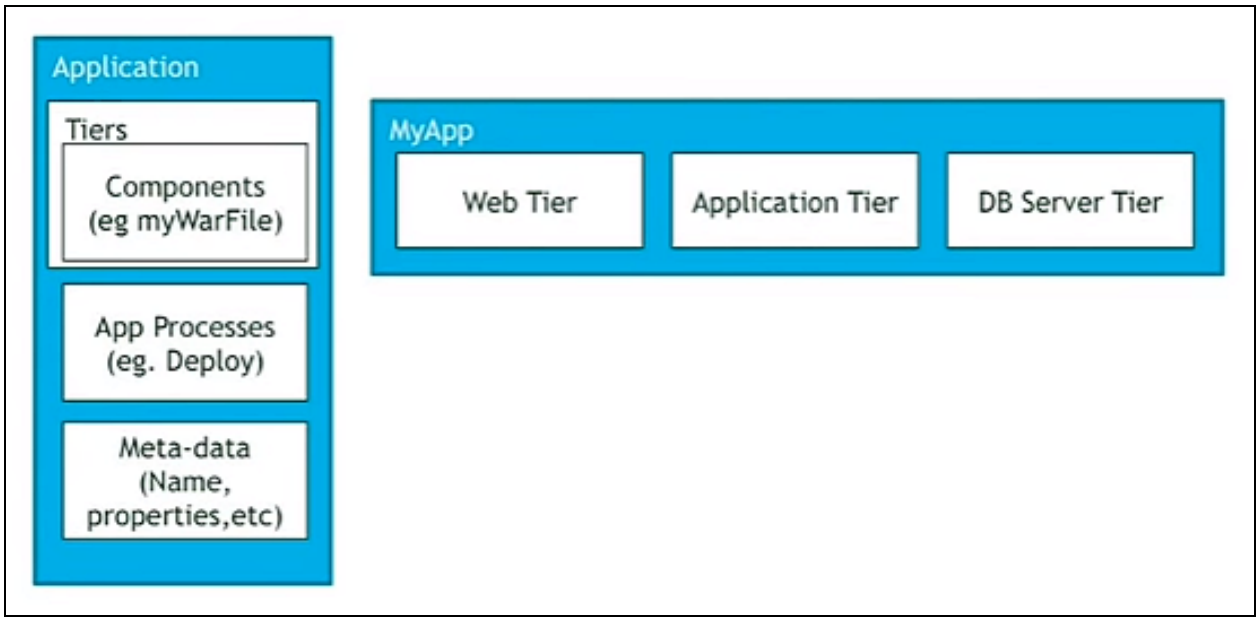
ElectricFlow uses a "model driven" approach to abstract the application or microservice architecture from the environment, infrastructure details so that the same application or microservice can be deployed to many environments using parameterized processes. ElectricFlow supports the creation, versioning, deployment, required middleware, and infrastructure details for a complete release package. You can model and version applications, microservices, and environments to orchestrate the software release flow and also to standardize and automate the deployment of applications or microservices between environments.

ElectricFlow Deploy makes deployments manageable, reproducible, and error-proof by modeling the application or microservice, the environments to which it will be deployed, and automating the workflow needed for the deployment. The model can be broken into these parts of your application or microservice deployment:

1. What—You are deploying the application or microservice, which contains references to the files being deployed.
2. Where—The application or microservice will be deployed to an ElectricFlow environment.
3. How—You run processes that orchestrate deployment tasks to deploy the application or microservice to the environment.

Applications and Processes

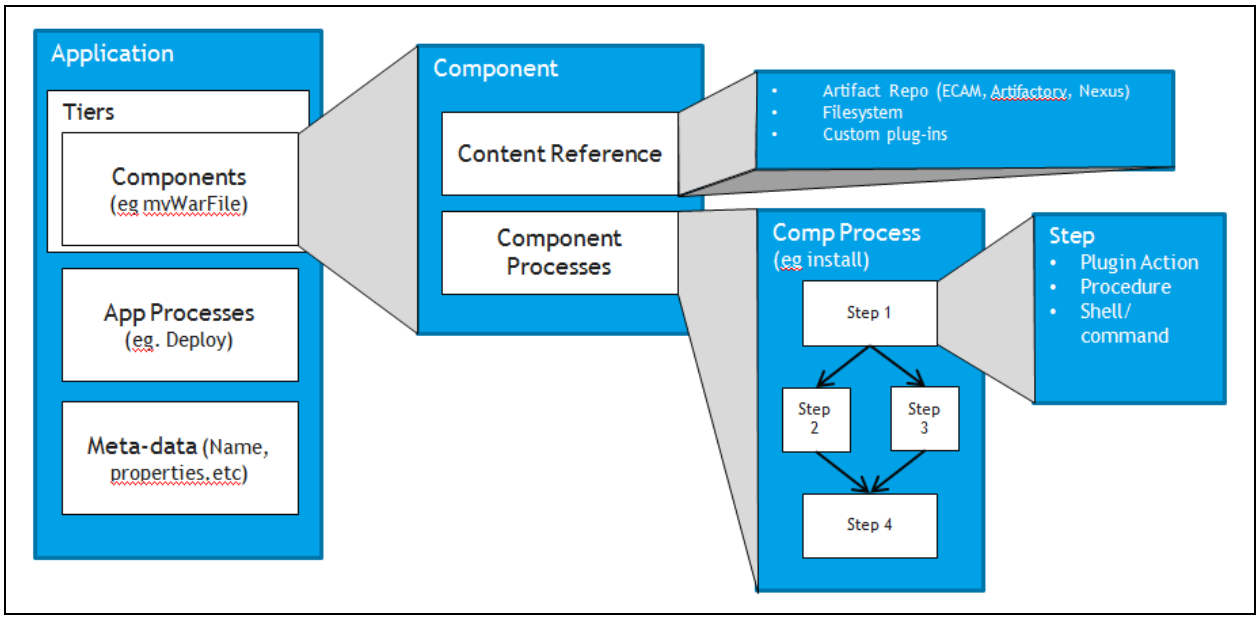
Applications are modeled by providing information about their components and defining the processes that orchestrate the deployment of those components. An application object in ElectricFlow consists of one or more application tiers. For example, MyApp is a three-tiered application with the Web, Application and database server (DB server) tiers.



An application tier is a logical grouping of components. Each tier contains components. A component is based on an artifact and contains a reference to that artifact. This can be through the file system or an

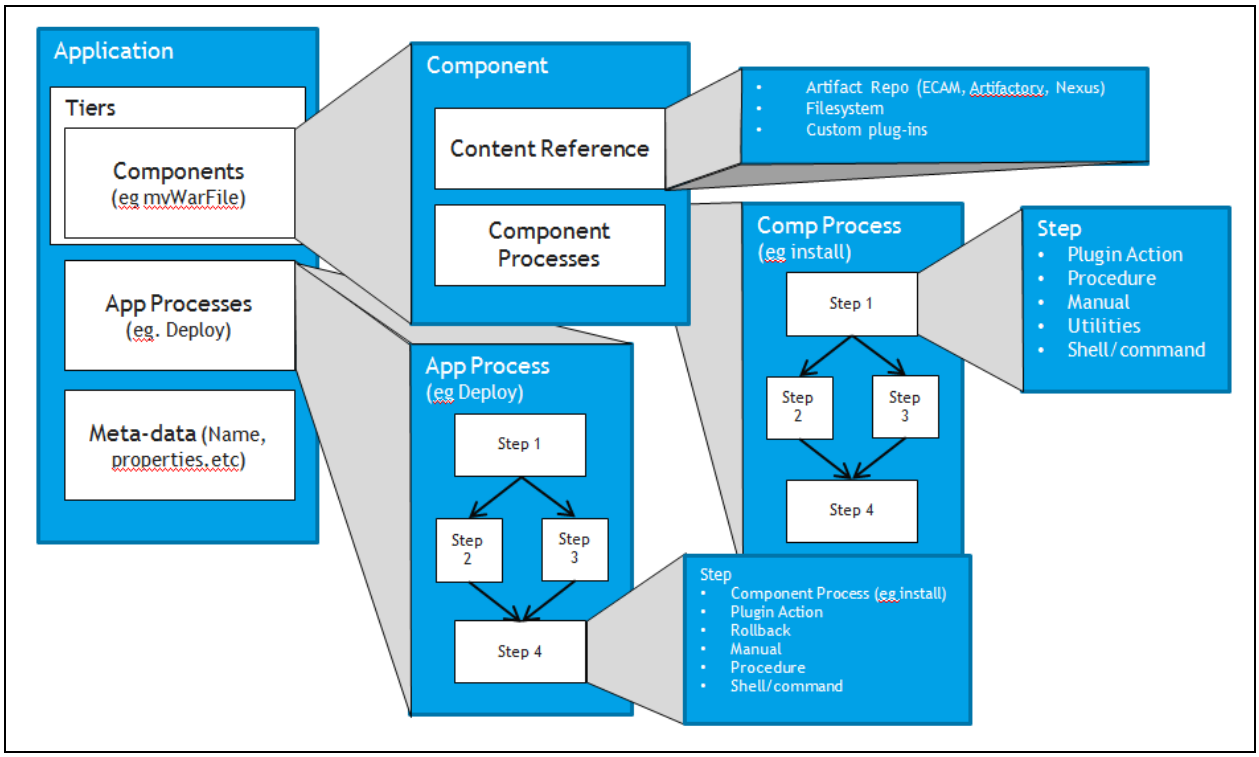
artifact repository. The application tier also contains component processes. A component process is a set of actions to be taken on that specific component when the application is deployed. The set of steps in a component process can be defined as

- Direct commands
- Calls to ElectricFlow plugins, procedures, or utility functions
- Component operations based on the component definition
- Manual steps



Application processes are at the application (parent) level. These processes are invoked to orchestrate operations against the application. The steps in an application process can be defined as:

- Calls to component processes
- Direct commands
- Calls to an ElectricFlow plugin, procedure, or utility function
- Manual steps
- A rollback step to roll back to a previous state or to a specific snapshot



Properties and parameters allow the same application to be deployed to many environments in a repeatable, reliable, and scalable way. They are used extensively throughout ElectricFlow and can be dynamically passed to all operations, including application deployments.

- Properties provide a flexible and powerful mechanism to manage data during all operations. They can be attached to any ElectricFlow object, such as applications, processes, tiers, components, and credentials. See Properties for more information.
- Parameters provide a way to get input from users at runtime to deploy the application to a variety of environments. Parameters can be attached to processes, process steps, components in the application. See Deployment Examples on page 252 and Parameter—create new or edit existing parameter on page 1182 for more information.

See Deployment Examples on page 252 for examples of how to author and deploy an application in the UI.

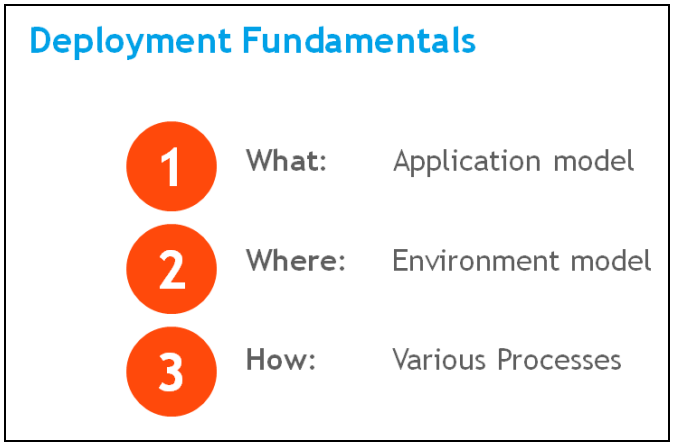
In an application, you can also:

- Define process steps based on commands or scripts using the ElectricFlow DSL, ElectricFlow REST API, ElectricFlow Perl API, Groovy, or JRuby. See the ElectricFlow API Guide on the Electric Cloud Documentation page for the commands and code samples.
- Create a library of standardized reusable components to promote best practices across the enterprise and speed up application authoring with master components. See Master Components on page 67 for more information.
- Save snapshots of the application throughout the software release process. A snapshot is an immutable version of an application with specific artifact versions that can be used to optimize and troubleshoot the application. See Snapshots on page 119 for more information.

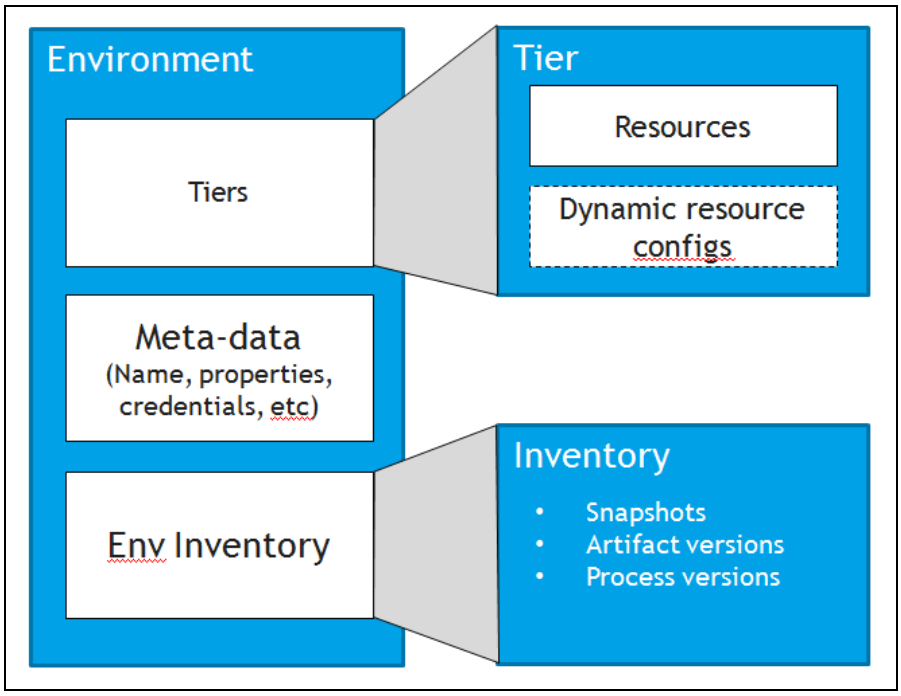
- Track changes for non-runtime objects such as projects, applications, environments, processes, components, artifacts, resources, and properties in the Change History. See Configuring Change Tracking on page 1375 for more information.

Environments

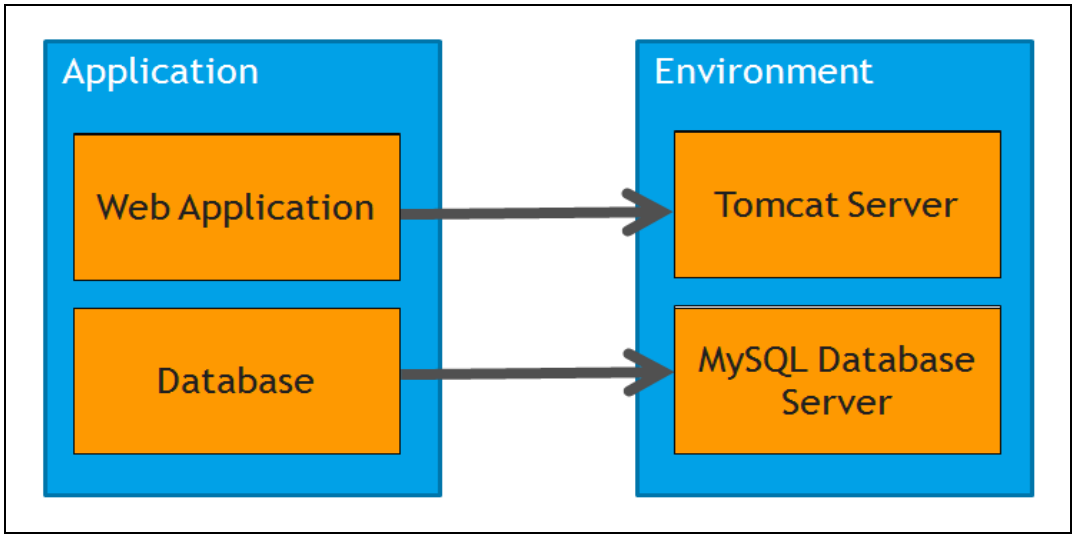
This section describes where applications are deployed.



Environments are modeled using resources that are statically defined on the system, dynamically spun up at deployment time, or a combination of both. An environment object in ElectricFlow contains one or more tiers. These tiers should map with the tiers in the application and are logical groupings of resources. Each tier defines the resources in that tier, which could be static resources in the ElectricFlow system or dynamic configurations that are spun up on deployment. An environment also contains an Environment Inventory that details the current versions of each component or snapshot deployed in that environment.



Once you have defined your application and environment, they must be connected with a tier map. This is a mapping of the application tiers to the corresponding environment tiers where the application will run. Tier maps provide a level of indirection, allowing you to freely add or remove resources from an environment without having to make any changes to the automation.



See Creating Environments on page 275 for more information about modeling static environments.
See Deploying Applications in Dynamic Environments on page 343 for more information about modeling dynamic environments.

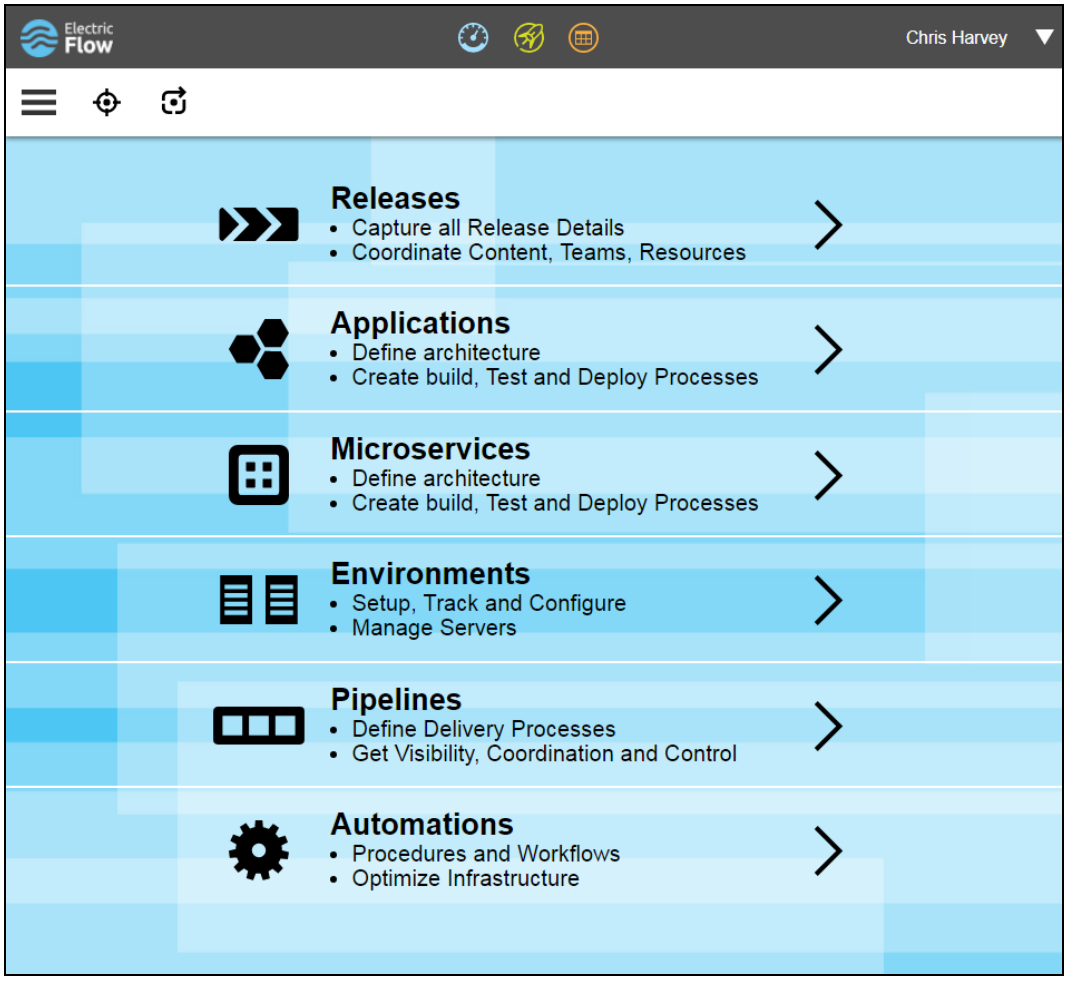
Creating or Editing a Project in the Deploy Web UI

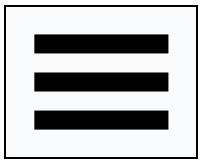
The Deploy web UI lets you create a project from the very beginning or create a project by copying another project. This copying capability lets you maintain a "library" of projects from which to create new ones.

Creating a Project

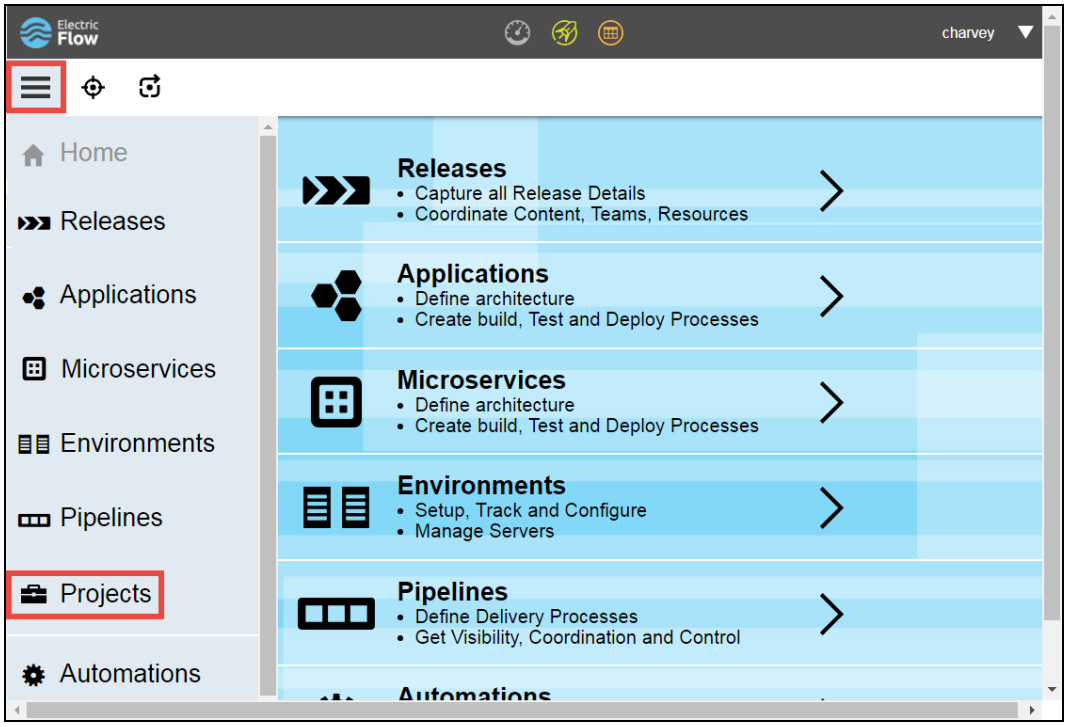
1. Open the home page of the Deploy web UI by browsing to https://<*ElectricFlow_server*>/flow/ and logging in.

   The home page appears:





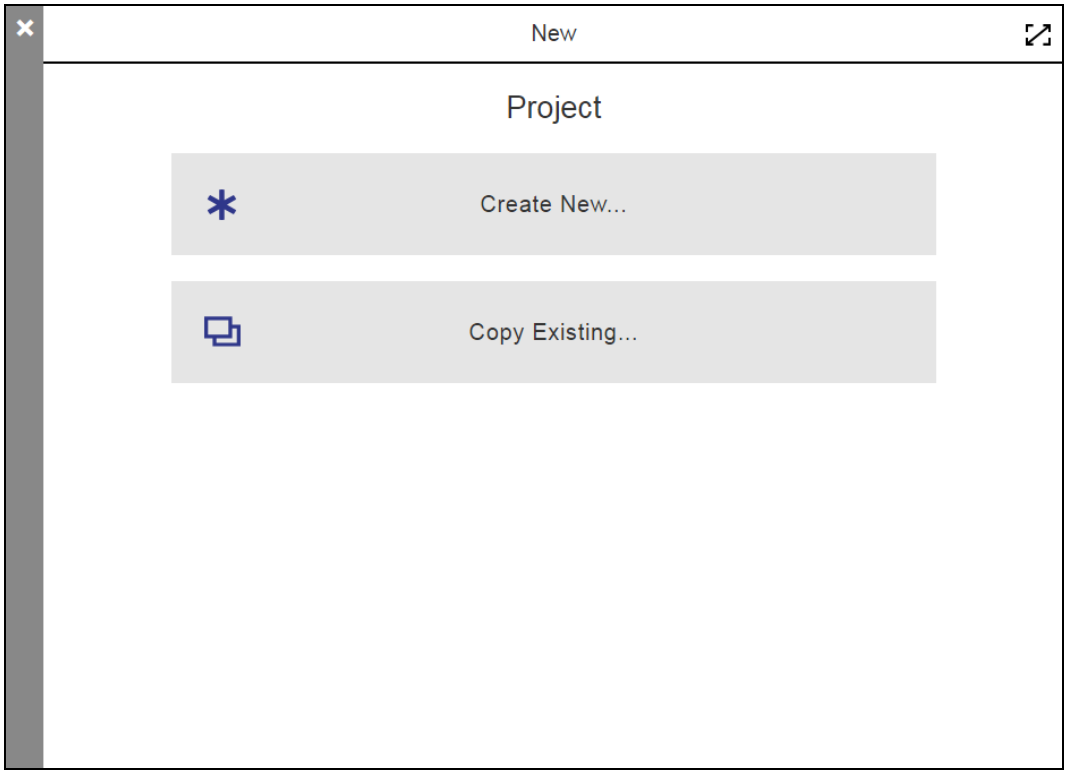
2. Click the menu button in the upper left corner and then click **Projects**:



The project list page appears:

3. Click the  (New Project) button.

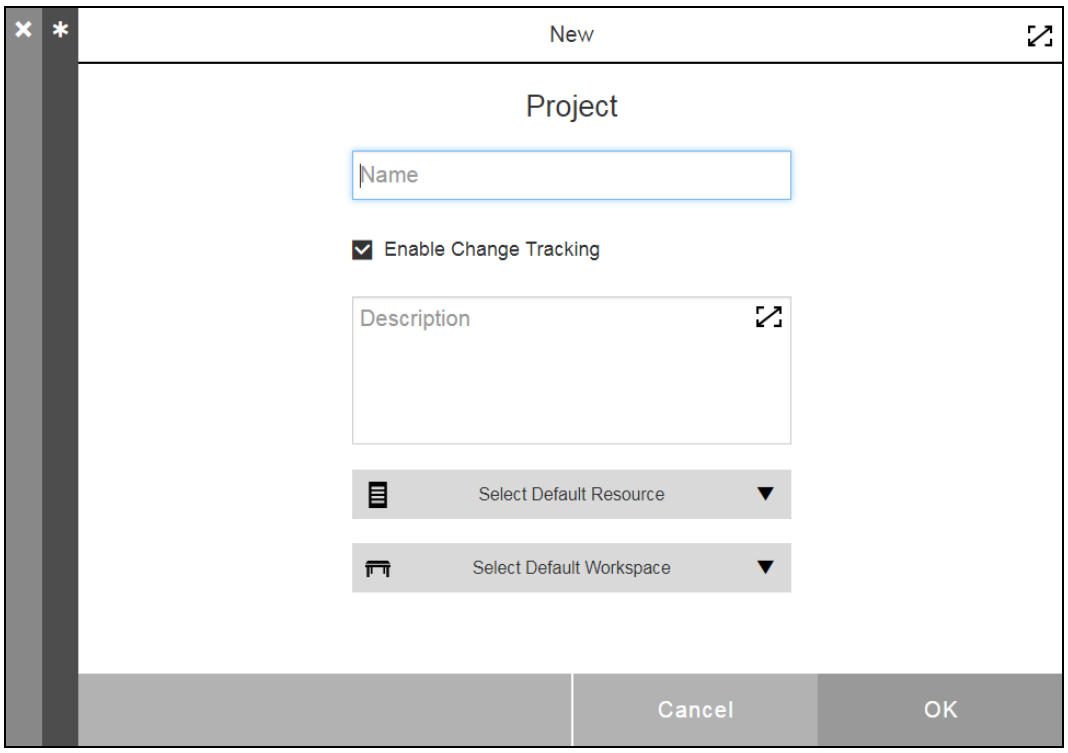
   The **New Project** dialog box opens:



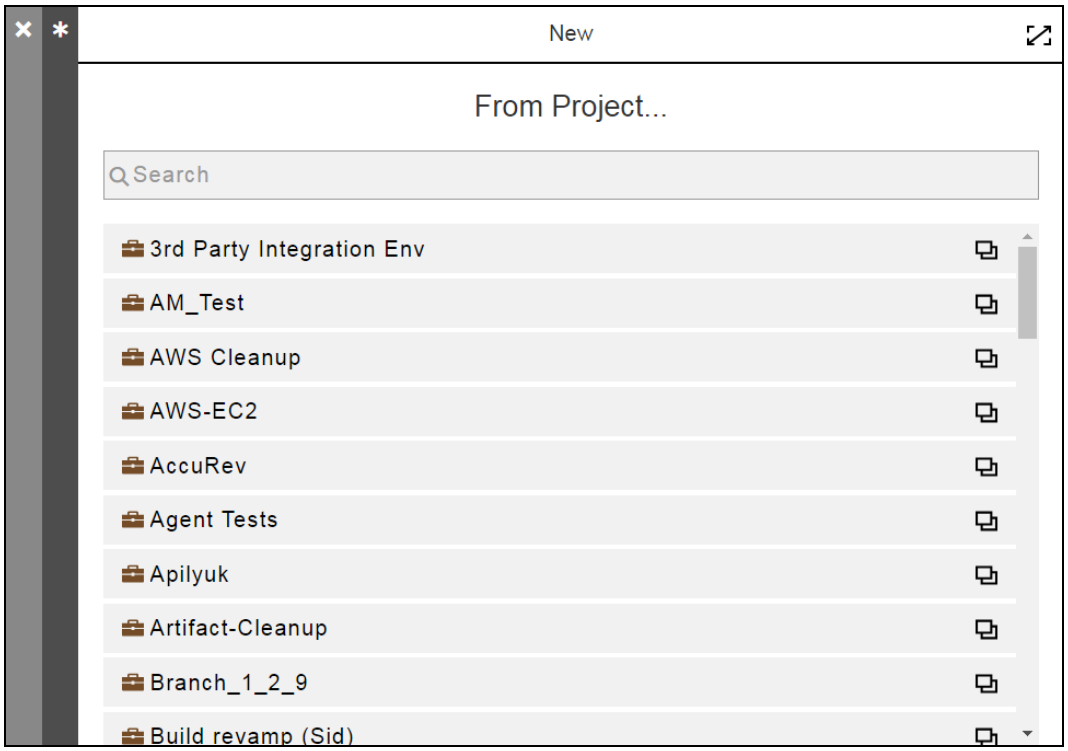
You can create a new project from "scratch," or you can copy (clone) another project to save time.

4. Click **Create New** or **Copy Existing**.

   If you clicked **Create New**, the following dialog box appears:

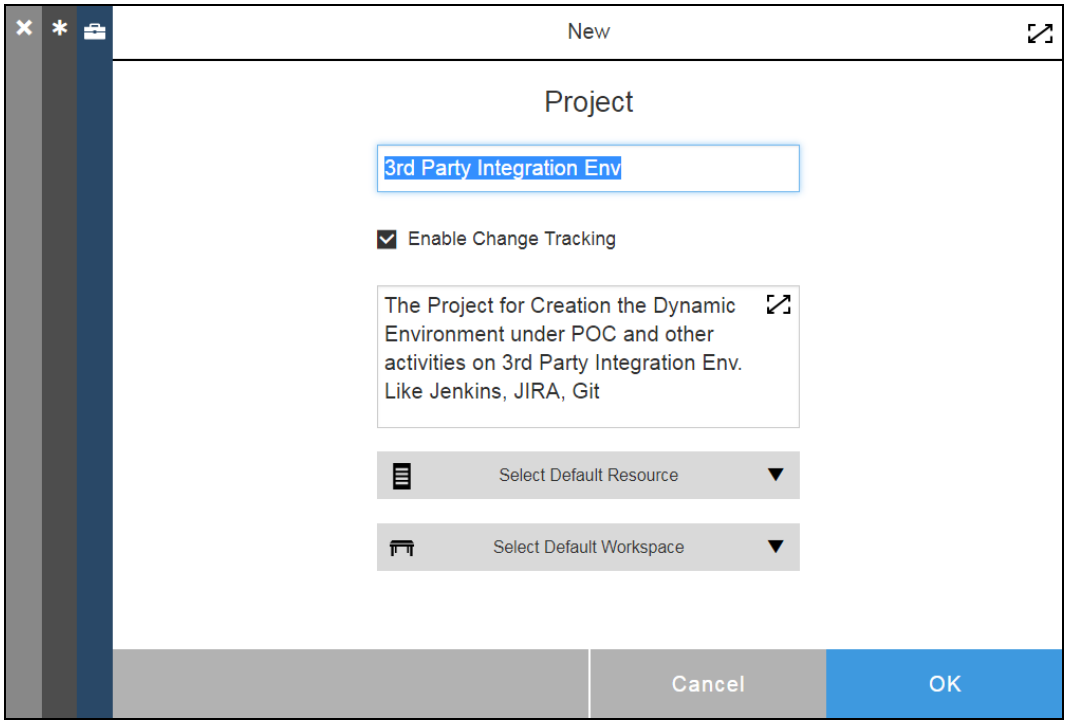


   If you clicked **Copy Existing**, the following dialog box appears to let you select a project to copy:


New
From Project...
Search
3rd Party Integration Env
AM_Test
AWS Cleanup
AWS-EC2
AccuRev
Agent Tests
Apilyuk
Artifact-Cleanup
Branch_1_2_9
Build revamp (Sid)

5. If you are copying an existing project, click to select that project.

   The following dialog box appears. This dialog box is prefilled with the settings from the copied project. For example:



6. Enter information into the fields as follows:

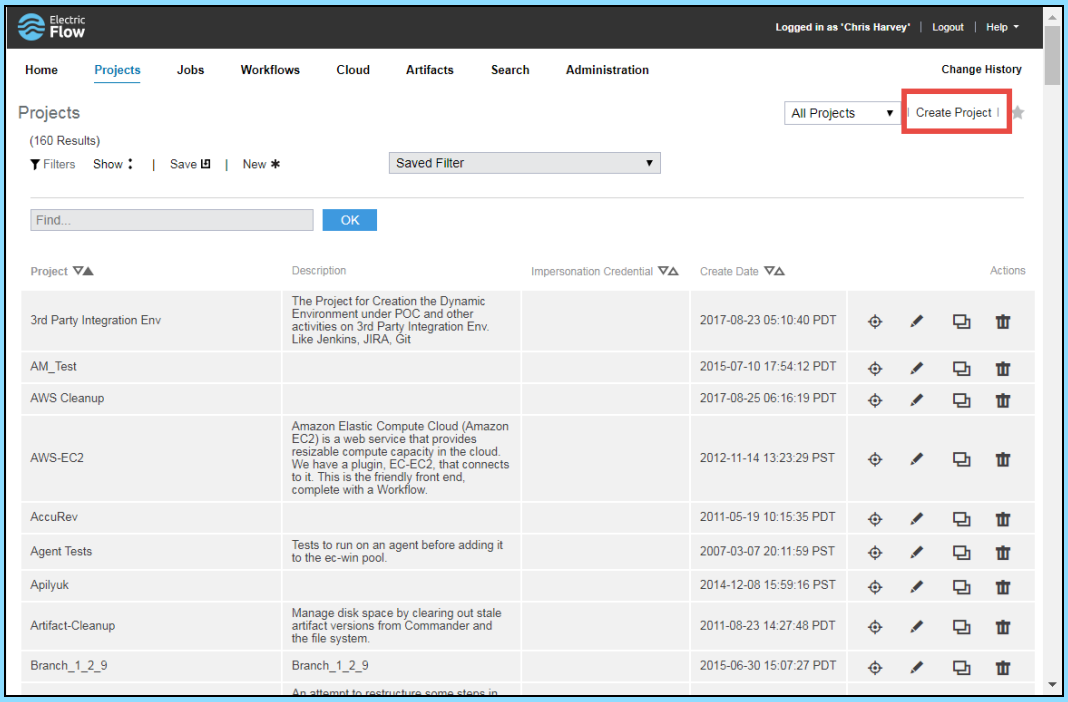
Field Name	Description
Name	Enter a unique project name. Use a meaningful name such as a work group or product. **Note:** To avoid conflicts, do not use "Electric Cloud" or any other ElectricFlow-supplied project name. Also, the "EC-" prefix is reserved for ElectricFlow-supplied project names.
Enable Change Tracking	Disable or enable change tracking for this project. This feature records every change between every state of non-runtime objects and lets you revert to an object's prior state. Tracked objects include applications or microservices, procedures, workflows, workspaces, resources, and project-owned components such as library components. For more information, see Change Tracking on page 1374
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Credentials	(Optional) Credentials and impersonations available to this project. For new projects, you are prompted to create project credentials on a subsequent dialog after project details are entered. See Credentials and User Impersonation on page 876 for further information.
Default Resource	(Optional) Default resource for all jobs that run under the project. This is a convenient way to use a single resource for an entire project. You can specify resources in several other places such as in procedures and individual job steps. A workspace specified elsewhere takes precedence. If you have not set up resource names or locations yet, see Resources on page 1208 for instructions.

Field Name	Description
Default Workspace	(Optional) Default workspace for all jobs that run under the project. This is a convenient way to use a single workspace for an entire project. There are several other places where you can specify workspaces, such in as procedures and individual job steps. A workspace specified elsewhere takes precedence. If you have not set up workspace names or locations yet, see Workspaces and Disk Space Management on page 1587 for more information.

7. Click **OK** and you are prompted to create or edit project credentials. If you choose **Yes**, the Credentials dialog appears.

   Your new project name will appear on the project list page.

Tip:
You can also create or copy a project via the **Create Project** link on the **Projects** tab in the Automation Platform web UI:

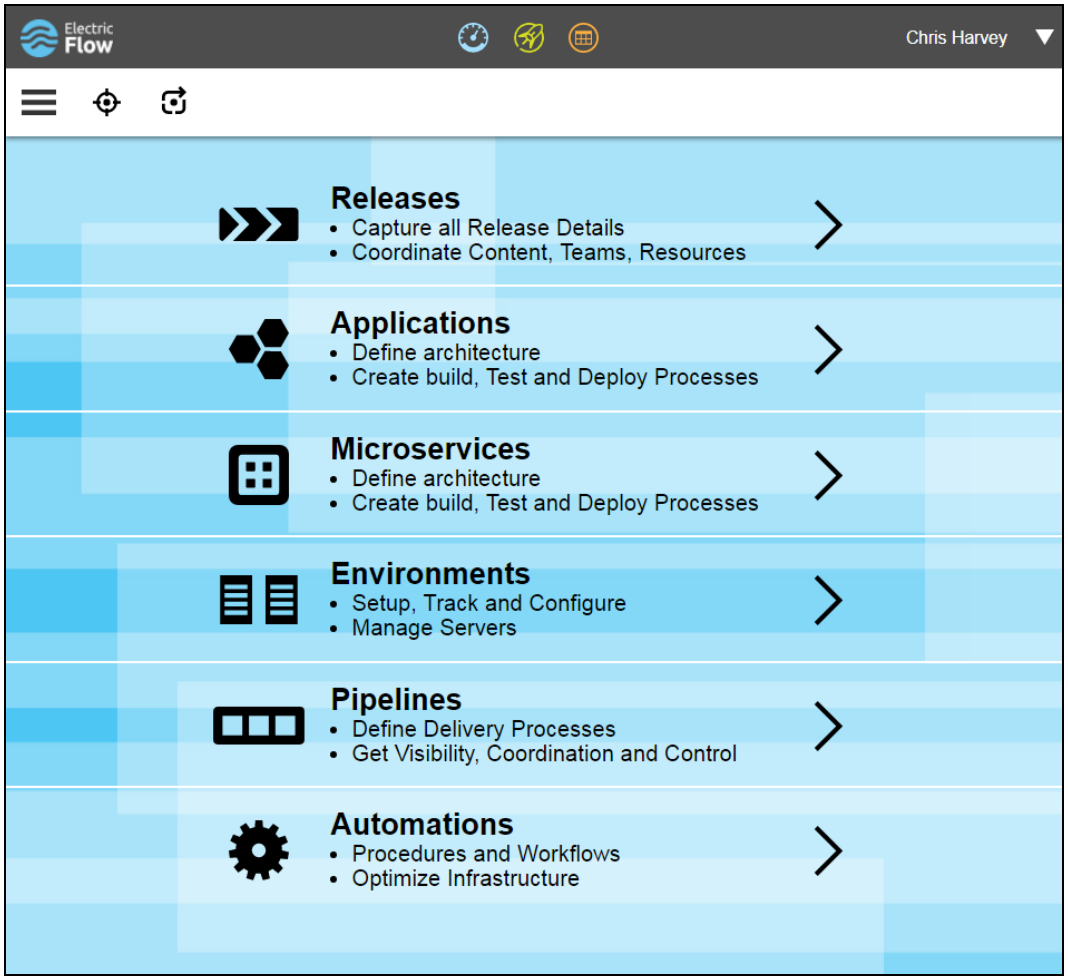


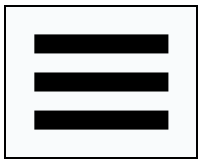
For details, see Project—Create or Edit a Project in the Automation Platform Web UI on page 1187.

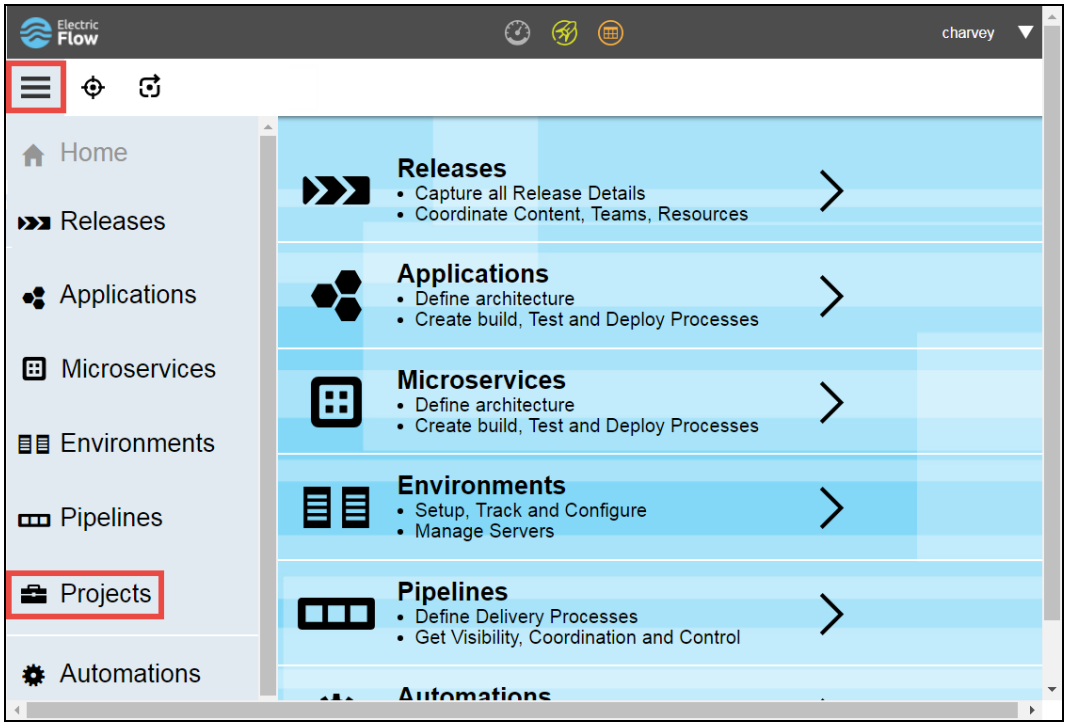
Editing a Project

1. Open the home page of the Deploy web UI by browsing to https://<*ElectricFlow_server*>/flow/ and logging in.

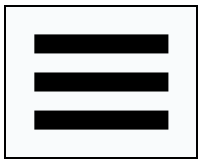
   The home page appears:



2. Click the  menu button in the upper left corner and then click **Projects**:



The project list page appears:



3. Click menu that corresponds to the project that you want to edit and then click **Details**. For example:

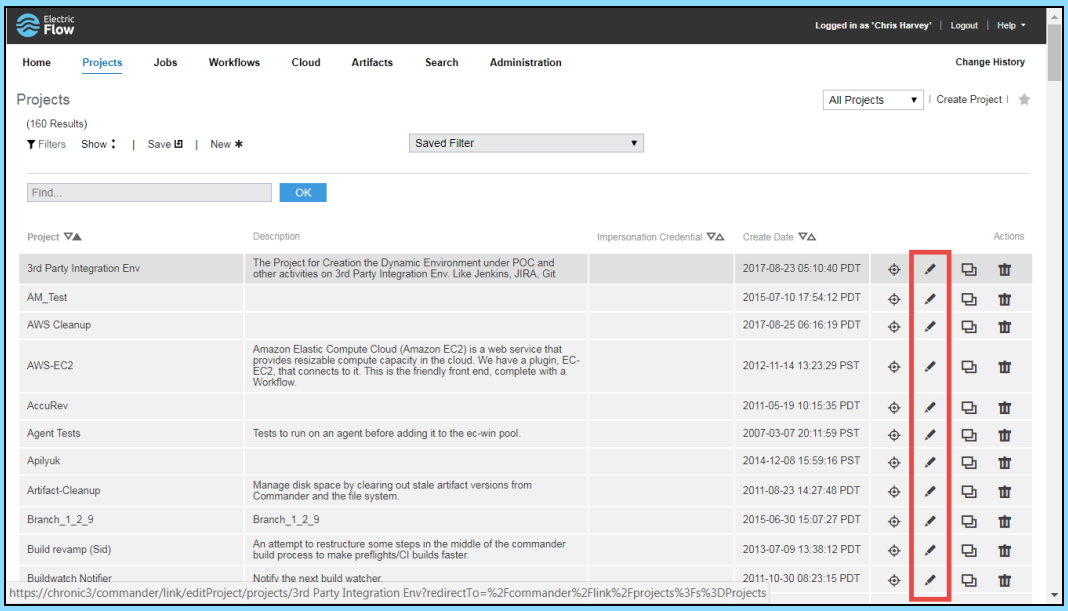
   The **Edit Project** dialog box opens. For example:

4. Update the information in the fields as needed. See the table in Creating a Project on page 56 above for field descriptions.
5. Click **OK** to save your changes.

   Your changes will appear on the project list page.

Tip:

You can also edit a project by clicking the project's (Edit) button on the **Projects** tab in the Automation Platform web UI:



For details, see Project—Create or Edit a Project in the Automation Platform Web UI on page 1187.

Master Components

Master components promote component reuse. They provide an easy way to create new components by reusing existing component definitions and the underlying process definitions. Master components are components that can be created and managed independent of any application. A master component consists of a component definition and one or more component processes. Master components are then leveraged to create application components either by referencing or copying them.

You can use master components to create a library of standardized reusable components which can help to promote best practices across the enterprise (such as a standard WebSphere component, standard IIS component, and so on). When deployments have large numbers of similar components, this can be used to speed up application authoring by leveraging prebuilt component and component-process logic.

A component has two parts:

- *Component details* describe the artifact on which the component is based. It can include where the artifacts are stored (the source or repository server), artifact name, artifact version, the directory where the artifact can be retrieved, and so on. You can parameterize part of the artifact definition to account for variations in the artifact details every time you deploy the application. This allows you to reuse the master component in multiple deployment scenarios. For example, you can create a master component that has a standard process to deploy and undeploy, and the only thing that changes is which specific artifact to use based on parameters.
- *Component processes* are used to define the automation steps to perform on the artifact defined in the component details. These processes can be used to deploy, undeploy, or do any other action. Each master component can have one or more component processes, and each process can have specific automation steps. These steps can describe how to perform specific tasks such as publish or retrieve an artifact, stop a server process, copy files, restart the server, and so on. A master component must have at least one component process. You can parameterize the master component processes so that certain behaviors can be achieved when master components are used in an application definition. For example, you can have *port number* as a component process parameter for a master component, and have different values for the port number in different application models where this master component is used.

After creating the master component, you can reference or copy it to create a new component in an application.

- You can reference a master component when authoring an application. The new application component will have the same properties as the master component. Any changes made to the master component will affect all the application components that reference it.

  Use the *reference* method when you want to maintain lineage with the master component so that the application component continues to inherit definitions from the master component. Before making a change to a master component, check how many applications reference it and determine the impact of changing it.

- You can also copy a master component when authoring an application. The new application component will have the same properties as the master or existing application component when it is created.

  Use the *copy* method when you want to use a master component just as a baseline for your application component. After an application component is created from a master component, any subsequent changes to the master component will not affect the application component. To get the latest changes in the application component, you delete and recreate the application component, and copy the latest master component to create a new application component.

  You can also take an existing component that was originally copied from a master component, modify it, and then save it as a new master component that can be reused and shared by other users.

Master Component Examples

This example shows how to create a master component, create parameters for the artifact definition, define the component processes, create parameters for the component processes, and use the master component in an application, promoting the component reuse throughout your organization. It does not include how to author applications and environments or how to deploy the application. For information about these tasks, see the example in Examples: Modeling and Deploying Applications on page 255.

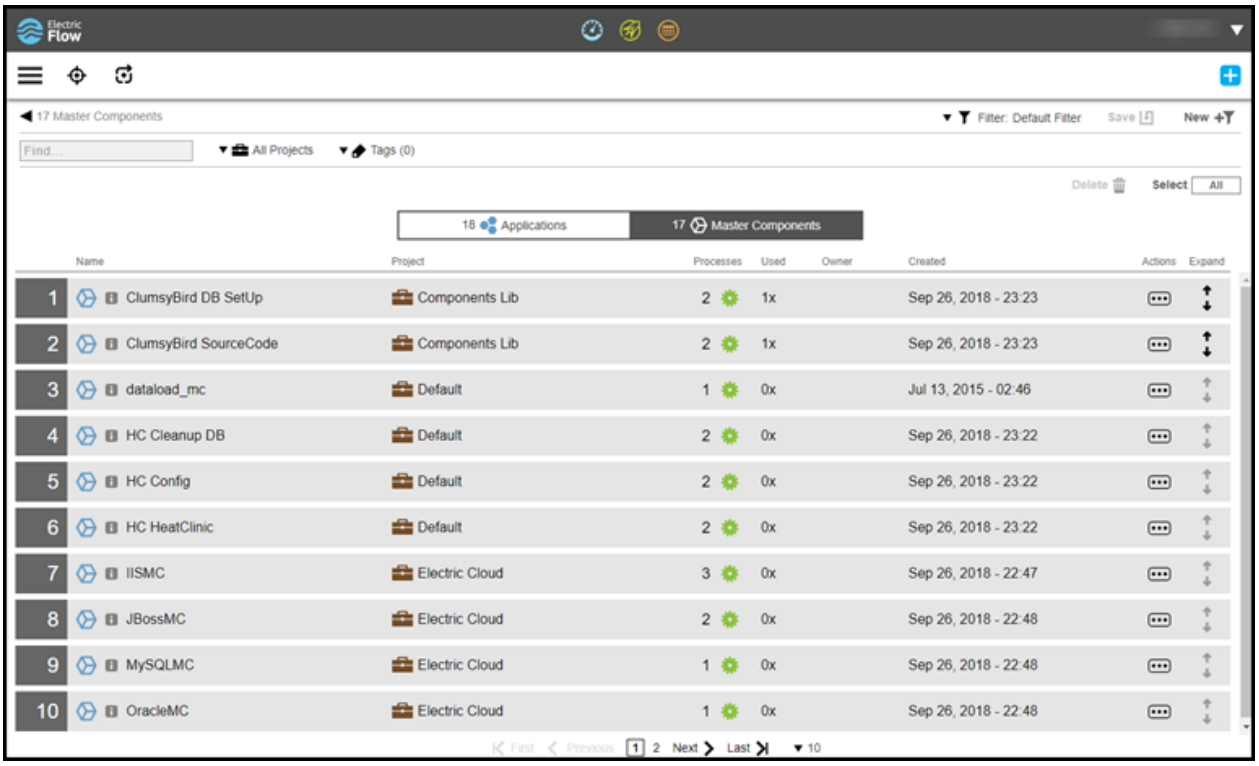
In the following steps, you will learn how to:

1. Create a master component and parameters for the artifact definition.
2. Author component processes and parameters.
3. Use the master component in an application.

Creating a Master Component and Parameters for the Artifact Definition

This section shows how to create a new master component and create parameters for the artifact definition.

Example:



By default, the master components for all projects are displayed.

To see only the objects for a specific project, click the down arrow in the **All projects** field and then select one or more projects in one of these ways:

- Click on the name of one or more projects.
- Enter the search criteria in the Search field. The projects that match the criteria appear in the list.

If there are no matches, a message appears stating that there are no resource templates in the selected projects.

Starting from the Master Components List:

1. Click the **Add +** button.

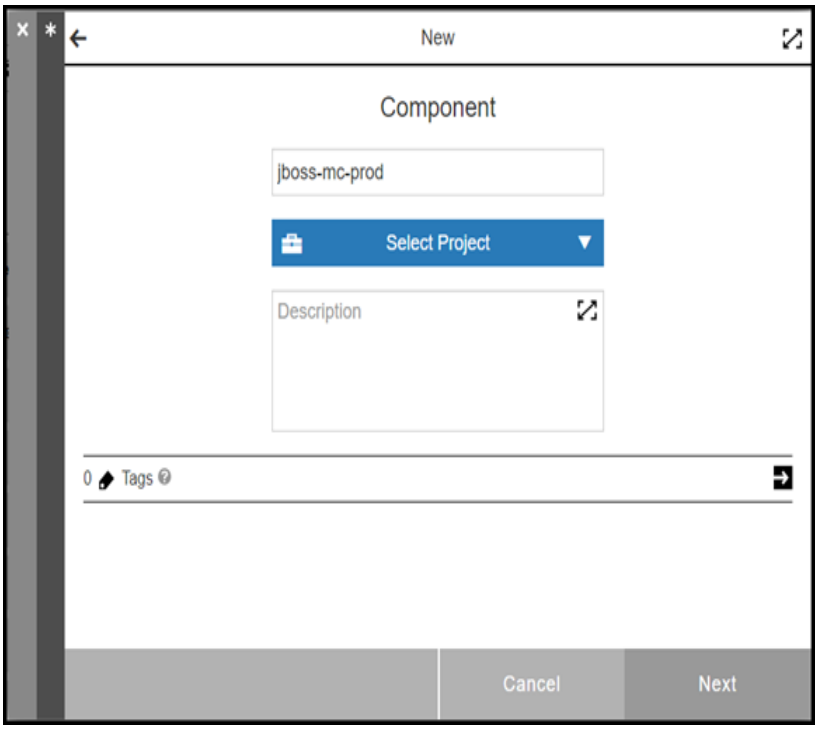
   The **New** dialog box opens.

2. Click **Master Component** > **Create New** to create a new master component.

3. Enter the name of the new master component, configure **Tags**, and select a project to which the master component belongs.

   Tip: You can include hyperlinks as part of an object description for any ElectricFlow object.

   ## Example:



4. Click **Next**.

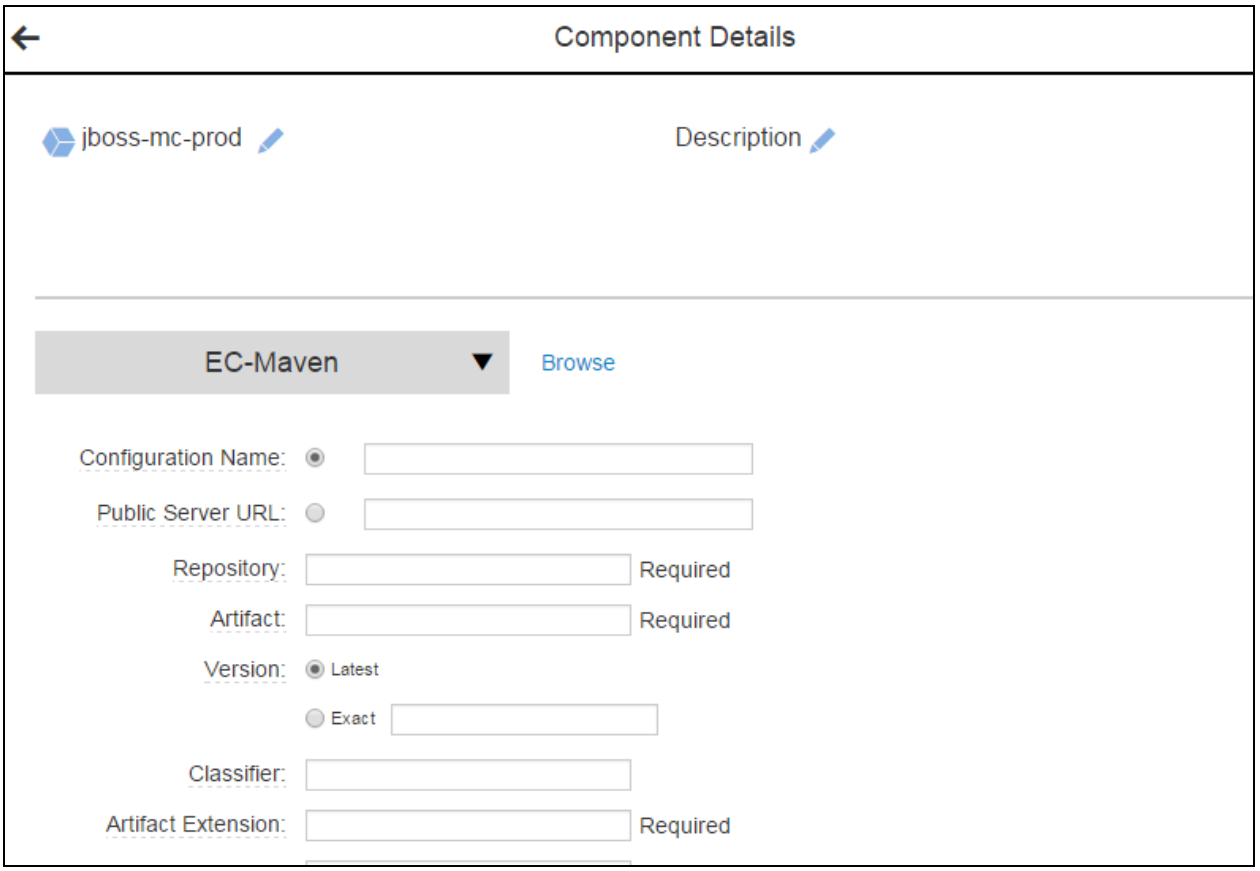
   The **Component Details** page dialog box opens.

5. Click the down arrow in the **Content Location** field.

6. Select an artifact repository where artifacts are stored, which is represented as a plugin.

Example:

Select **EC-Maven**.



7. Enter the information about the artifact.

   The information that you enter depends on the artifact repository (plugin) that you selected.

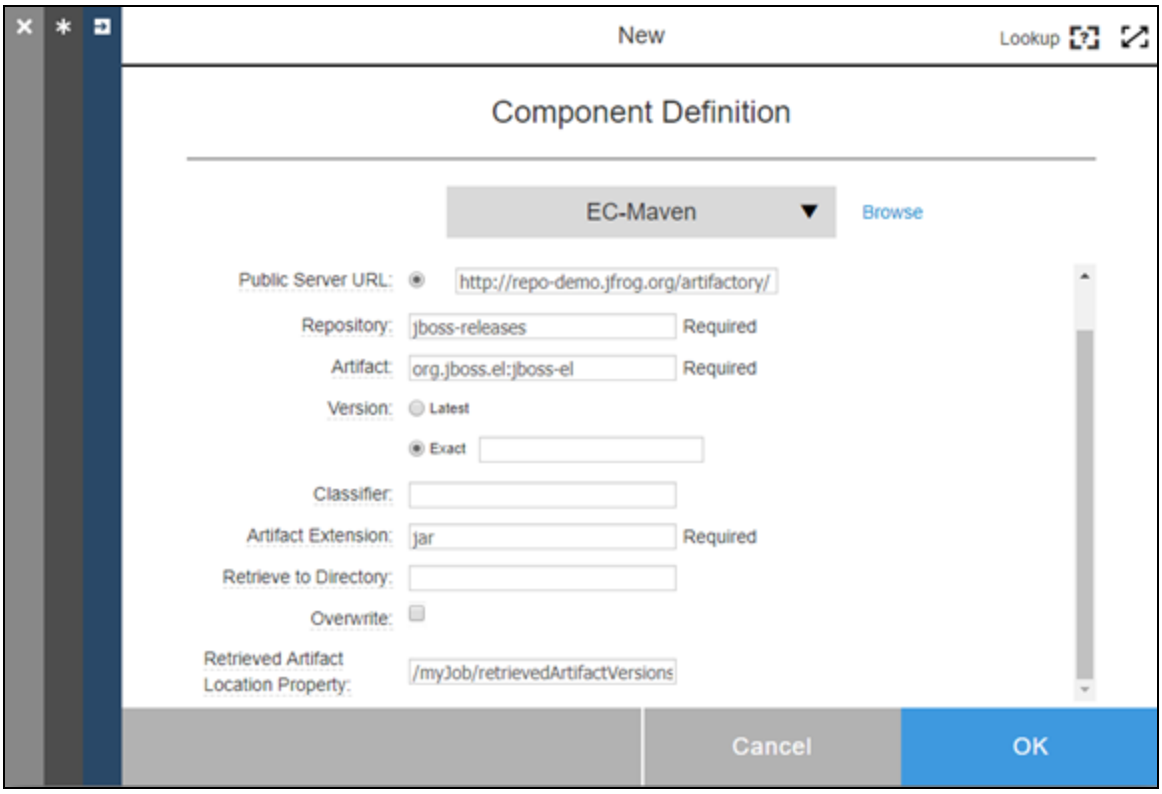
   ## Example:

   For **EC-Maven**, enter the following information.

   - Server URL: http://repo-demo.jfrog.org/artifactory/
   - Repository: jboss-releases
   - Artifact: org.jboss.el:jboss-el
   - Version: Exact
   - Artifact extension: jar

   We will be parameterizing the artifact name and replacing **jboss-el** with a variable. For now, enter the information in the **Artifact** field above because it is required and the component cannot be created without this required information.

   If you had selected another plugin, you may not have to do this.



8. Click **OK**.

   The component that you created now appears in the list.

9. Go to the row for the master component that you just created, and click its **Actions** button.
10. Select **Parameters**.

11. In the Parameters dialog box, click **There are no Parameters yet.**, or click **Input +** in upper right.

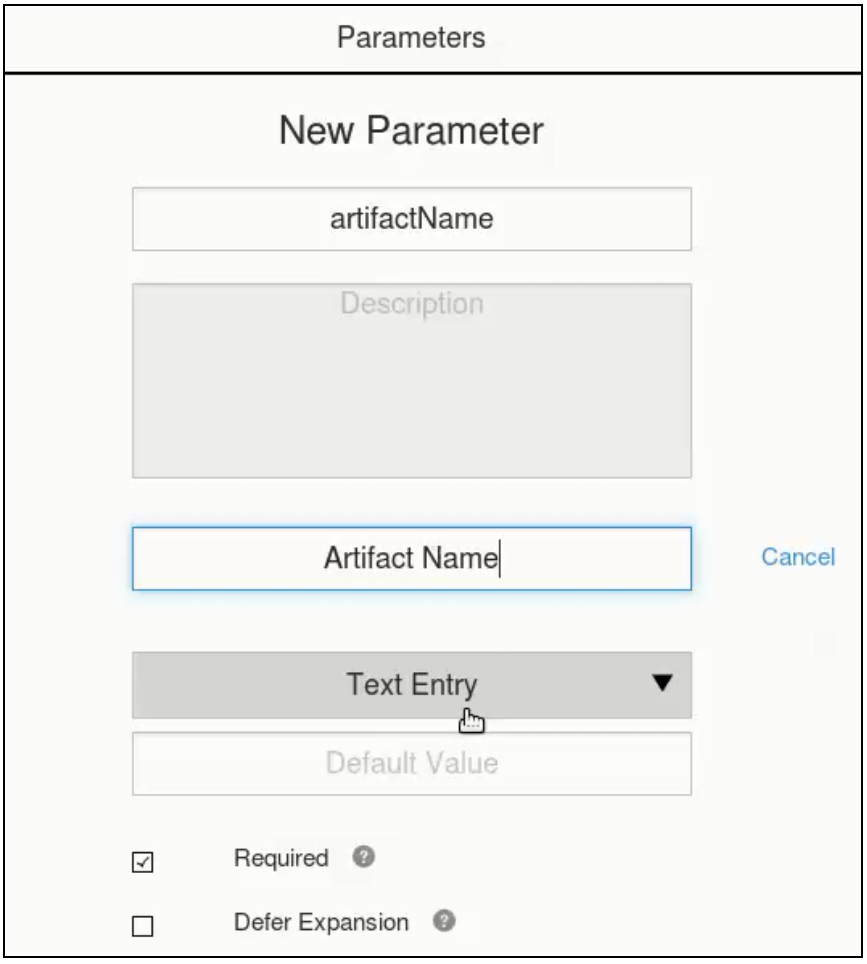
Example:

12. In the New Parameter dialog box, enter the parameter settings.

Example:

- In the **Name** field, enter **artifactName**.
- In the **Label** field, enter **Artifact Name**.

Keep the rest of the fields as they are.



This parameter has default value and is required. The user must enter a value for this parameter when this master component is used in an application.

13. Click **OK**.

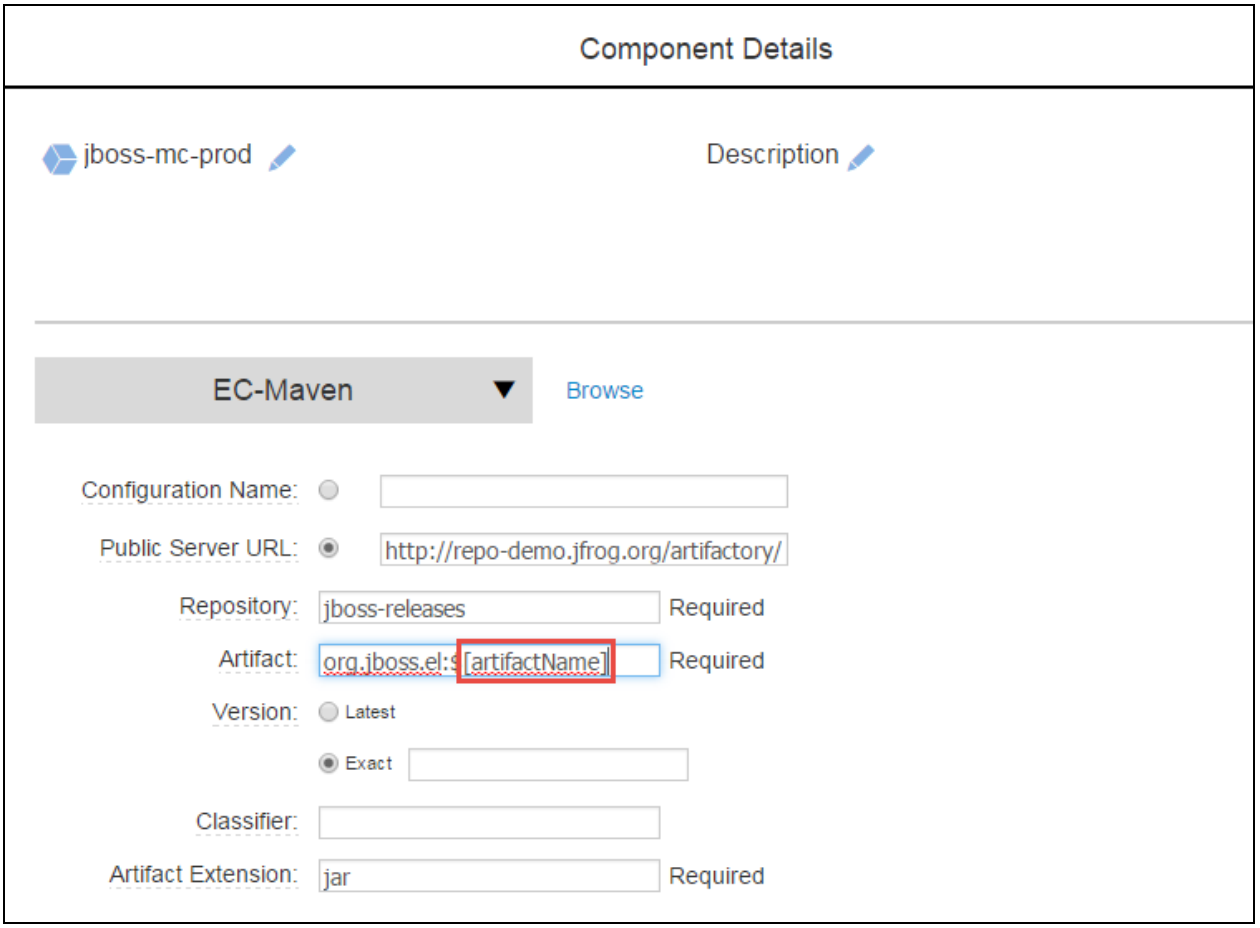
Example:

The Parameters dialog box now shows that there is one required parameter for the master component called Artifact Name and that the user has to enter the text (the artifact name) for it.

14. Go to the row for the master component that you created, and click its **Actions** button.
15. Select **Component Definition** to modify the artifact definition with the artifact-name parameter.

16. In the Component Details dialog box, go to the **Artifact** field and replace `jboss-el` with `$[artifactName]`.

Example:



17. Click **OK**.

You have now created a master component with one parameter for the artifact definition.

> **Note:**
> The other ways to create a master component are:
>
> - Create it from an existing master component by selecting an existing component from the "Create from existing component" list in the dialog box.
> - Create from an existing application component by selecting an application and an application tier associated with that application and then an existing component in that application and application tier.

The next step is to add processes to the master components and create parameters for the processes.

Authoring Component Processes and Creating Parameters

This section shows how to author component processes for the master component created in the previous section and create parameters for those processes.

Starting in the Master Component list:

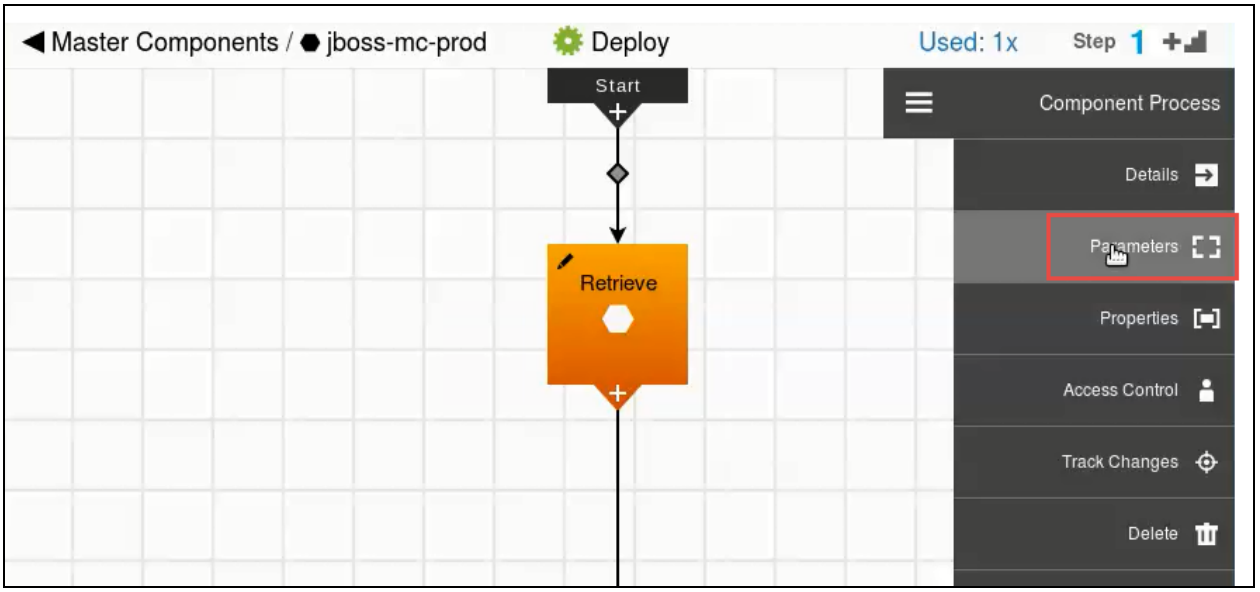
1. Go to the row for the master component that you created, and click its **Actions** button.
2. Select **Add process**.

   The **Component Process** dialog box opens.
3. Enter the component process details and click **OK**.
4. Define the component process in the Component Process Visual Editor.

   For detailed instructions and examples, see Authoring Component Processes on page 256.
5. Select the **Deploy** process.

   The Component Process Visual Editor for it opens.

Example:



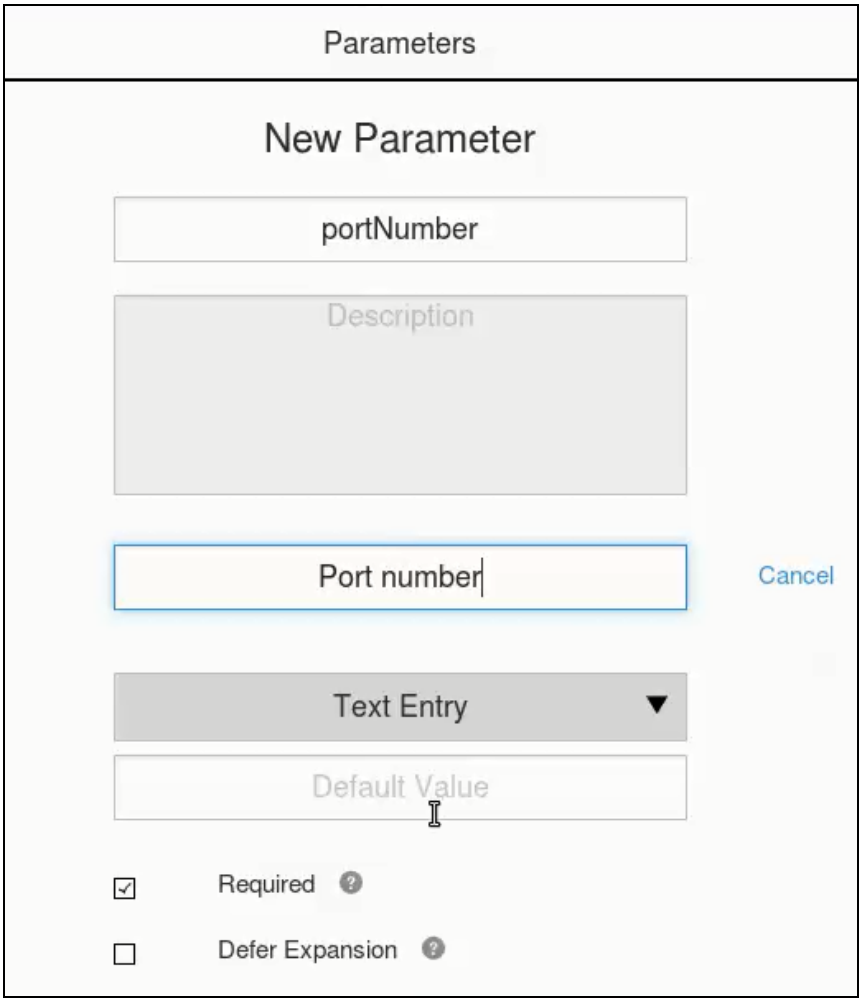
6. Click the button on the right side and select **Parameters**.
7. In the Parameters dialog box for the Deploy process, click **There are no Parameters yet.**
8. Click **Input** .

9. In the New Parameter dialog box, enter the parameter settings, and click **OK**.

Example:

- In the **Name** field, enter **portNumber**.
- In the **Label** field, enter **Port Number**.

Keep the rest of the fields as they are.

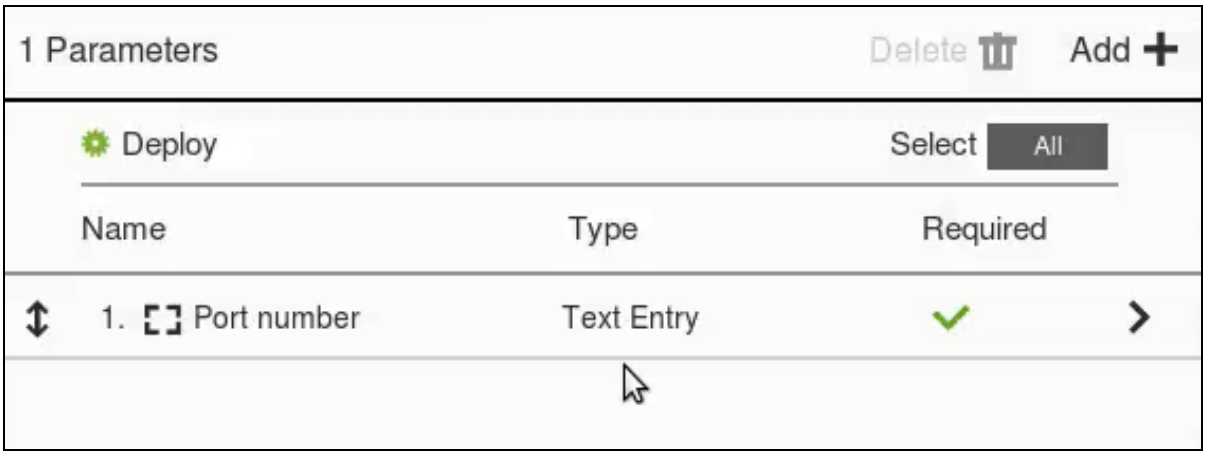


This parameter has no default value and is required. The user must enter a port number when this master component is used in an application.

10. Click **OK**.

Example:

The port number parameter is created for the Deploy process. The Parameters dialog box shows that the port number parameter is required for the Deploy process.



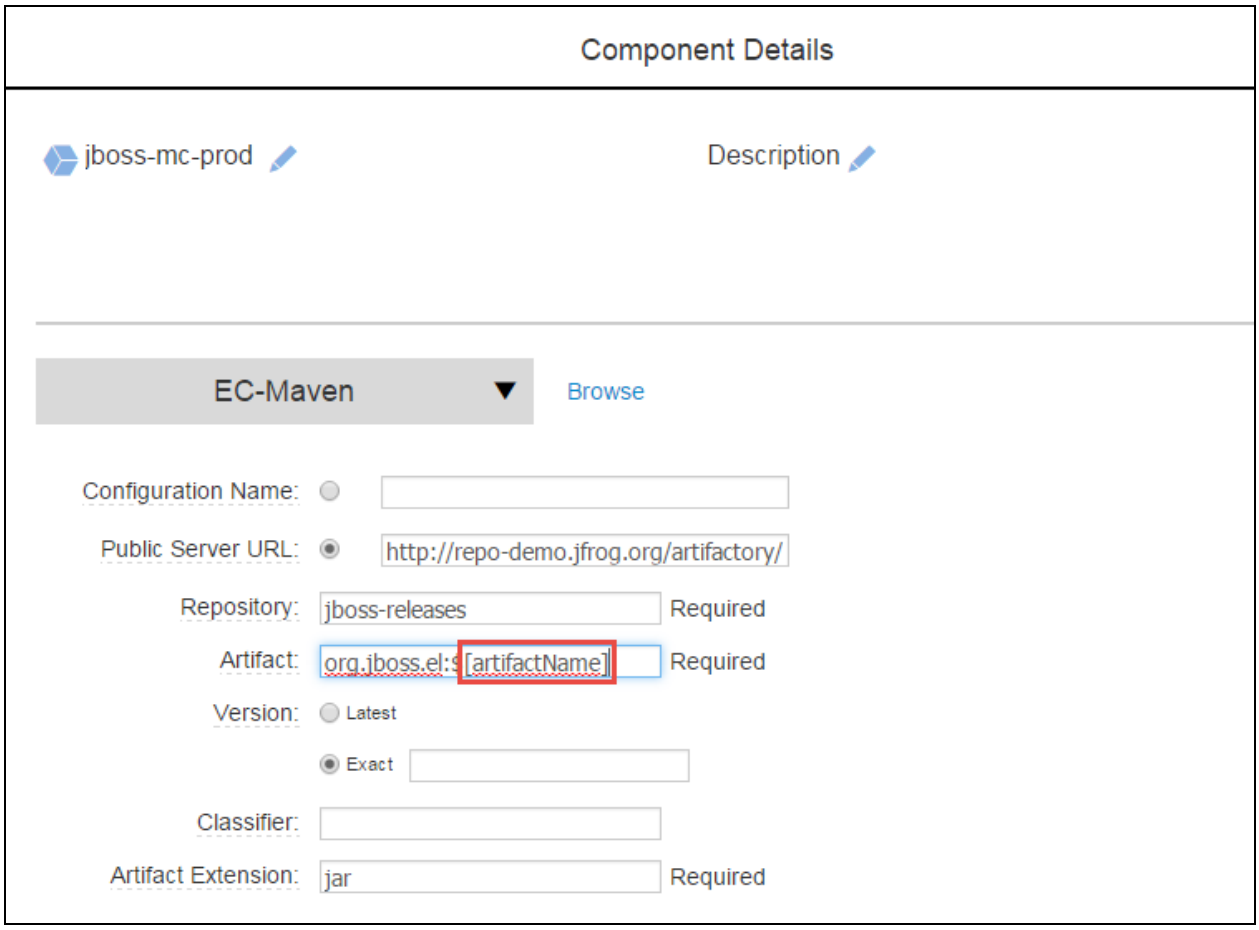
11. Close the Parameters dialog box.
12. Reference the master component in an application, or copy it to an application.
    - To reference the master component in an application, see Referencing a Master Component in an Application on page 81.
    - To copy the master component to an application, see Copying a Master Component to an Application on page 85.

    After you add the component to an application tier, a dialog box appears prompting you to enter values for the parameters for the master component process, starting with this step.

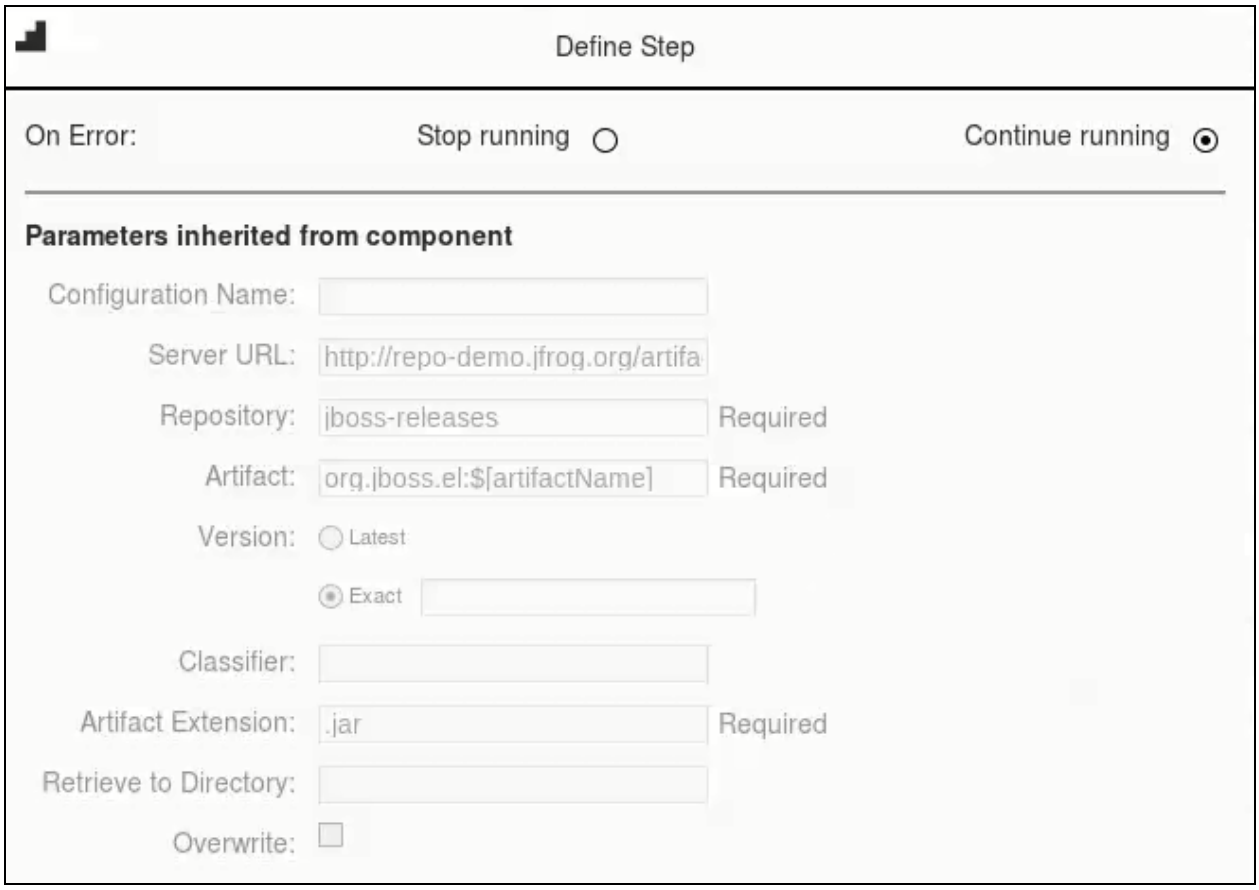
You have now authored component processes for the master component and have created a parameter for it.

Viewing the Master Component Details

You can ensure that the artifact definition and component process step defined by the master component are correct by viewing the master component details in the following places. They first appear in the artifact definition:



They also appear in the component process step details. When you view the component process details for the Deploy process in the Component Process Editor, you can see that the Deploy process consists of one step called Retrieve, which is defined by the jboss-mc-prod master component.



Using a Master Component in an Application

After master components have been defined, they can be used in application models. There are two ways to leverage master components in an application:

- Reference the master component in an application.
- Copy the master component to an application.

To review the implications of these methods, see Master Components.

Referencing a Master Component in an Application

When you reference a master component in an application, it is added to the application as an application component with the same definition, processes, parameters, and properties as the referenced master component. Any changes made to the master component will affect all the application components that reference it.

Use the reference method when the component is used in multiple applications and you want to ensure that these application components continue to inherit all changes made to the master component. This section shows how to author a new application that references the master component created in the previous sections.

Starting in the Applications List:

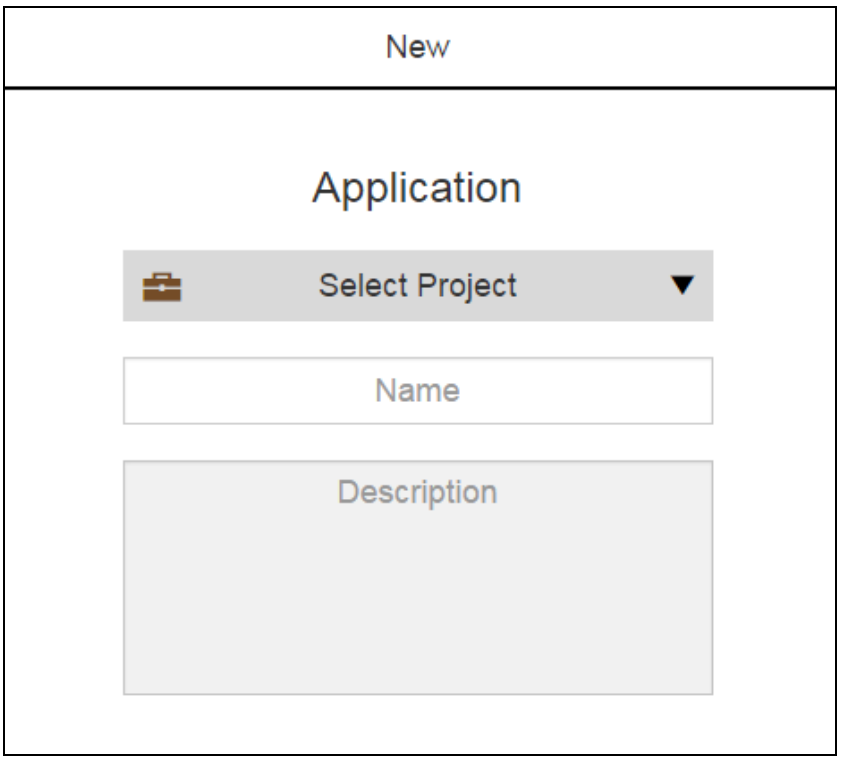
1. Click the **Add +** button.

   The New dialog box opens.

2. Click **Application** > **Create New** to create an application.

3. Enter the application name, select a project where the applications in the pipeline will be deployed, configure Object Tags on page 45, and enter a description of it.

   Example:



4. Click **OK**.

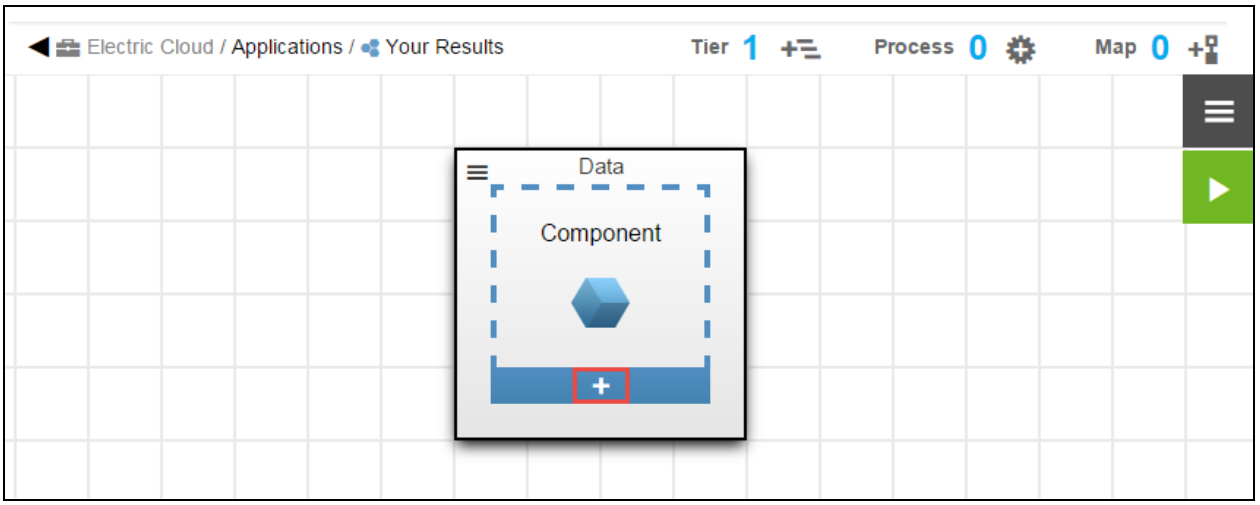
   The Application Editor for your application opens. This is where you will author the application. Notice that there is an undefined application tier (Tier 1) with an undefined component in it and an undefined microservice with an undefined container in it.

5. Click the  button in the application tier and select **Details** to open the Applications Tier Details dialog box.

6. Change the tier name to **Data** in the Name field, and click **OK** to rename the application tier.

   The Application Editor now shows one environment tier called Data.

   ## Example:



7. Click the **+** button in the component to add a component to the application tier.
8. Select **Create from existing master component**.

   A list showing all available the master components opens. You can also create a new application component using one of these options:

   - **Create a new application component**—Clicking this creates a new component as described in Creating a New Application on page 256.
   - **Create from existing application component**—Clicking this opens the **Create from existing component** dialog box.

     Click in the **Select Application** field and select an application. Then click in the **Select Tier** field and select an application tier. A list of components in the selected application and application tier appears. Select a component from this list.

9. In the **Create a reference to a master** list, select a master component.

   In this example, select the master component called **jboss-mc-prod**.

10. Click **Next**.

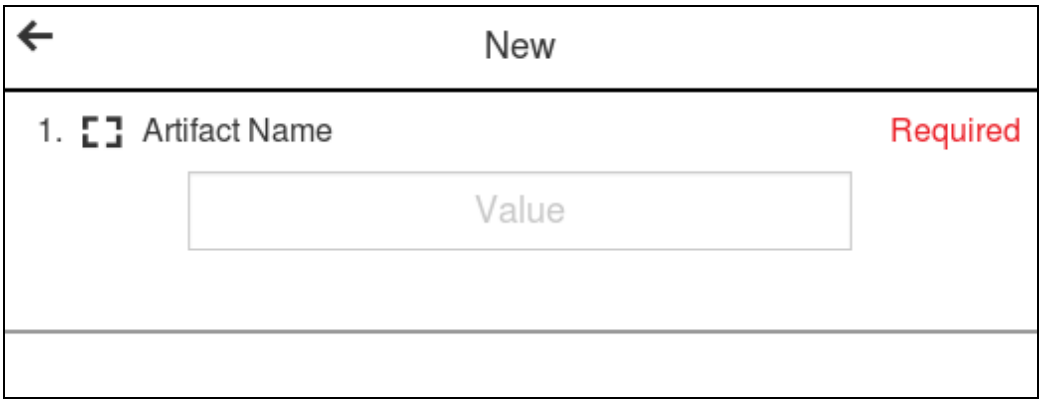
    The **Create from existing component** dialog box opens.

11. In the **New** dialog box, change the name to **jboss-mc-results**, and click **Next**.

12. Click **Next**.

    If the master component has parameter for its artifact definition, you are prompted to enter values for those parameters in the next step.

    ### Example:

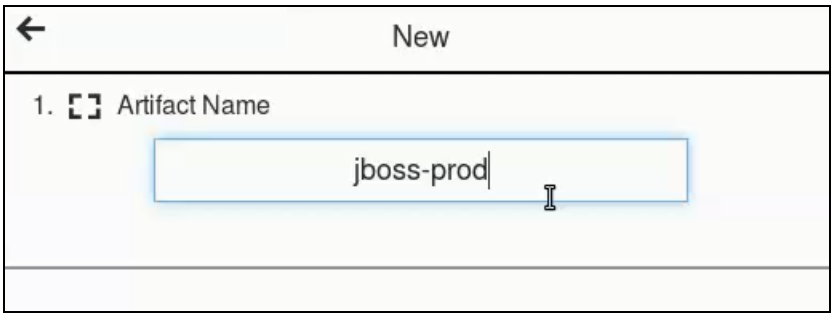
    

    If the component does not have parameters for its artifact definition, click **OK**. Then go to the Component Process Visual Editor and add a parameter to the component, starting at Step 9 in Creating a Master Component and Parameters for the Artifact Definition on page 68Authoring Component Processes and Creating Parameters on page 77

13. Enter the values for the parameters, and click **OK**.

    ### Example:

    The jboss-mc-results component has one parameter called Artifact Name. Enter **jboss-prod** in the Artifact Name field.

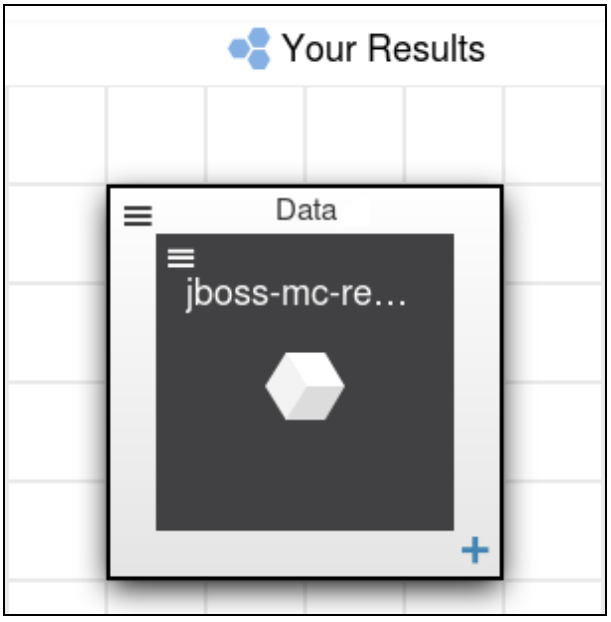
    

14. Click **OK**.

> **Note:** If you want to edit the master component, you need to do this in from the Master Component list. (There is no button in the lower right corner where you can access the component processes.) The UI also lets you know that this is a master component because the icon is different than that for a normal component.

Example:

The Application Editor now shows that the application has an application tier called Data with a referenced master component called jboss-mc-results.



15. Define the application process to include a step that references the master component.

    The second step prints the port number that you entered in the previous step.

Now you can deploy your application as described in Deploying and Troubleshooting Applications on page 287.

Copying a Master Component to an Application

When you copy a master component to an application, it becomes an application component with the same definition, processes, parameters, and properties as the master component on which it is based. After an application component is created from a master component this way, any subsequent changes to the master component will not affect the application component. Use this method when you want to use a master component just as a baseline for your application component.

For details about how to create master components and parameters for them, see Creating a Master Component and Parameters for the Artifact Definition on page 68 and Authoring Component Processes and Creating Parameters on page 77.

This example describes how to copy a master component in an application.

Starting in the Applications list:

1. Click the **Add +** button.

   The **New** dialog box opens.

2. Click **Application** > **Create New**.

3. Enter the name of the new master component, select a project to which the master component belongs.

   In this example, we will create a new application called "Your Group Summary" that will include the master component named jboss-mcjc.

4. Click **OK**.

   The Application Editor for your application opens. This is where you will author the application.

   Notice that there is an undefined application tier (Tier 1) with an undefined component in it and an undefined microservice with an undefined container.

5. Click the  button in the application tier and select **Details** to open the Applications Tier Details dialog box.

6. Change the tier name to **Data** in the Name field, and click **OK** to rename the application tier.

7. Click the **+** button in the application tier to add a component to it.

8. Select **Create from existing master component**.

   A list showing all available master components opens.

9. In the "Create a reference to a master" list, select a master component.

   Select the master component called **jboss-mcjc**, select **Use a copy**, and select a project to which the master component belongs.

10. Click **Next**.

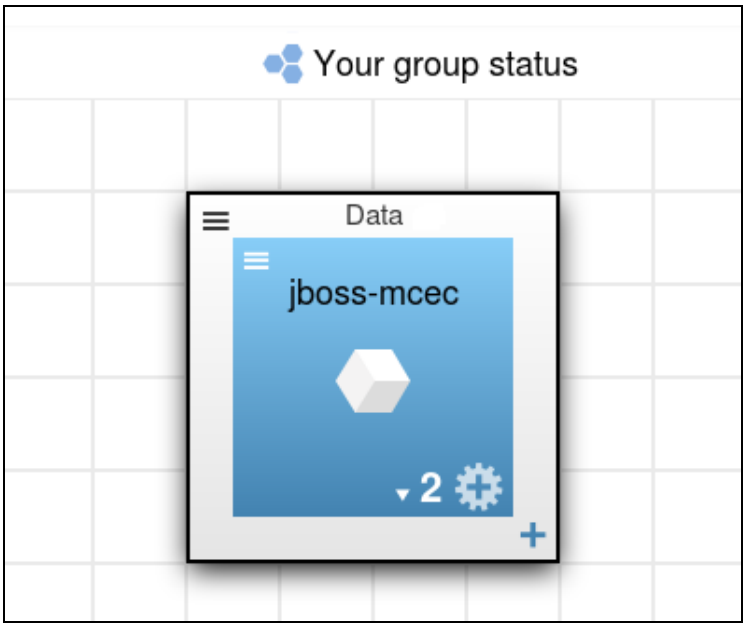
    The **Create from existing component** dialog box opens.

Example:

11. Change the name of the component to something that gives it more context.

Example:

After changing the component name to **jboss-mcec** and clicking **OK**, the Data tier now has a new application component called jboss-mcec, where "ec" refers to a software version:



12. If the master component has parameters for the artifact details, modify the artifact details of the new application component and remove the parameter reference.
    1. Go to the Component Details dialog box.
    2. Replace the parameter (`$[artifactName]`) in the artifact details section with a specific value.

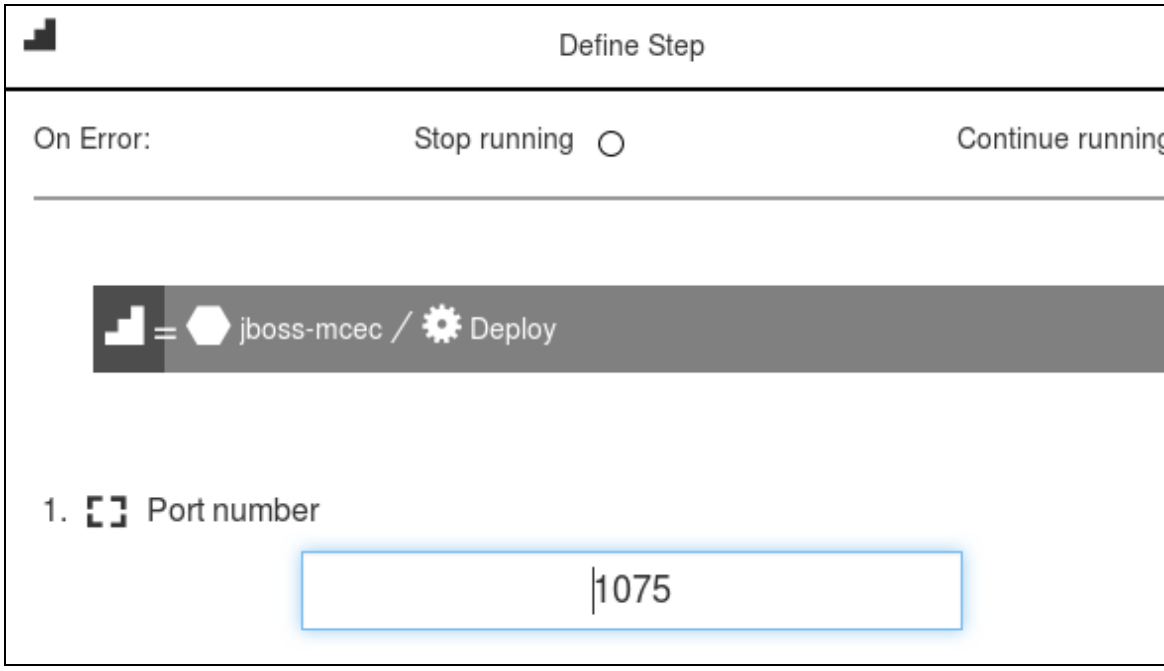
       In the example, replace `$[artifactName]` with **jboss-5-maintenance**, where software versions starting with lower case letters are maintenance releases
    3. Click **OK**.

13. Define the application process to include a step that references the new application component.
    1. The first step is defined by the jboss-mcec component and its Deploy process.

       After you select the Deploy process, you are prompted enter to a value for the port number. Then click **OK**.

       ## Example:

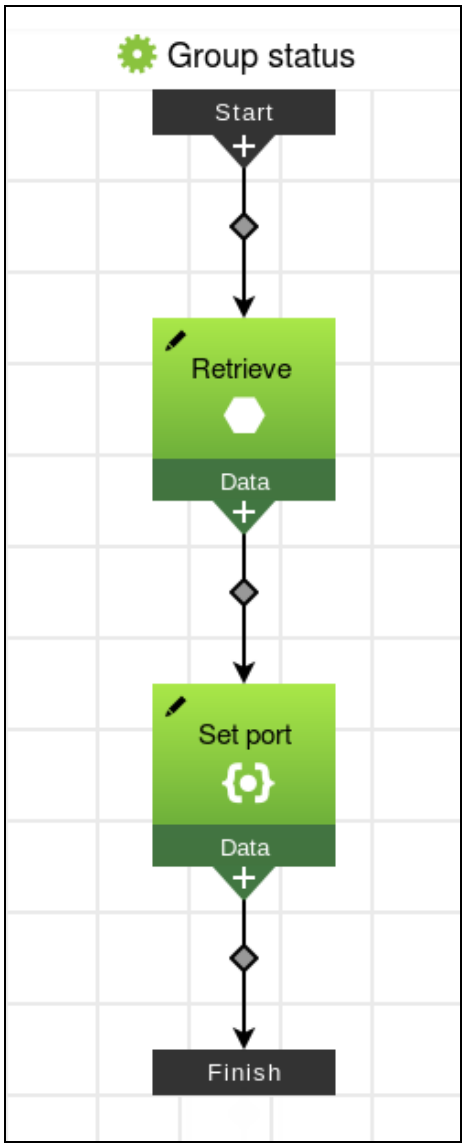
       This is the parameter that is created for the component process.



2. The second step prints the port number that you entered in the previous step.

Example:

This is the application process:



14. Make other modifications to the application component and its component processes within application.

Now you can deploy your application as described in the Guidelines for Modeling and Deploying Applications in ElectricFlow on page 252.

Creating Master Components Based on Existing Application Components

Starting from the Master Components List:

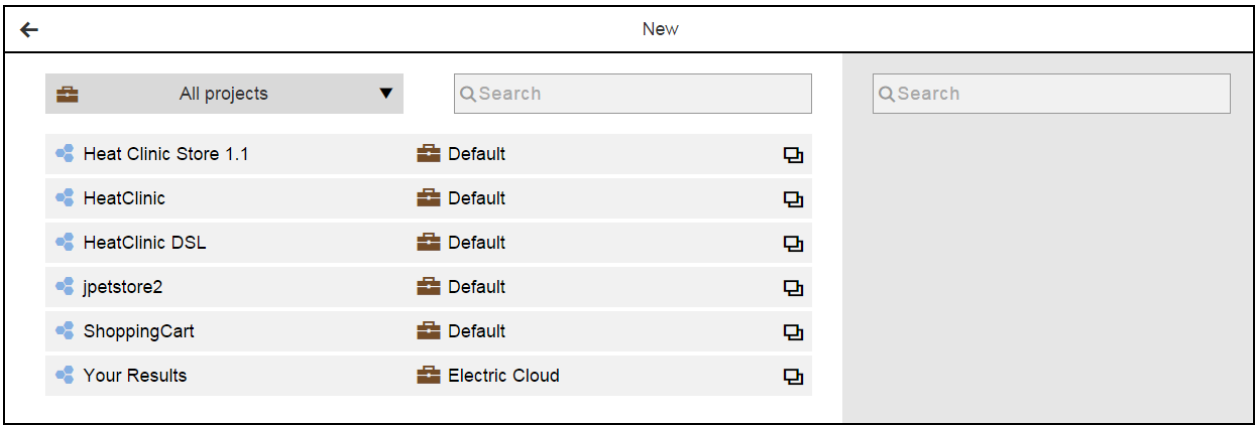
1. Click **Add +**.

   The **New** dialog box opens.

2. Click **Master component** > **Copy Application Component**.

   A list of existing applications opens.It shows the list of available applications. The default is to show the applications for all projects. If you want to see only the applications for a specific project, click the down arrow in the **All projects** field to select one or more projects one of these ways:Click on the name of one or more projects.Enter the search criteria in the Search field. The projects that match the criteria appear in the list.You can also search for one or more applications by entering the search criteria in the **Search** field next to the **All projects** field. If there are no matches, a message appears stating that there are no applications in the selected projects.

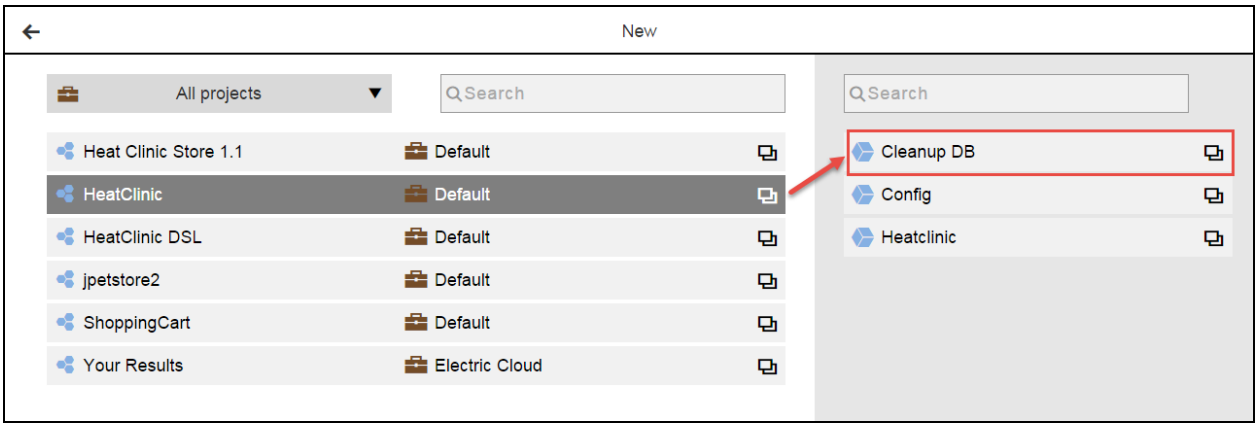
Example:



3. On the left side of the dialog box, select an existing application.

   A list of existing components in that application appears on the right side.

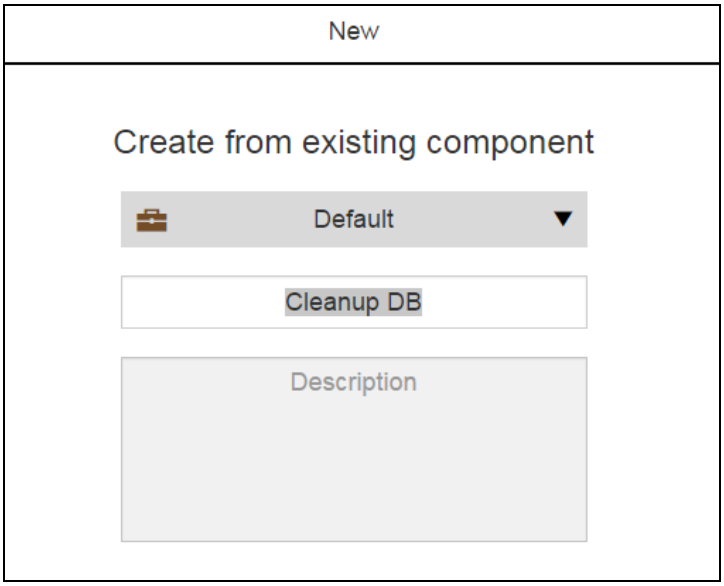
   Example:



4. Select an existing application component.

   The **Create from existing component** dialog box opens with the name of the selected component highlighted.

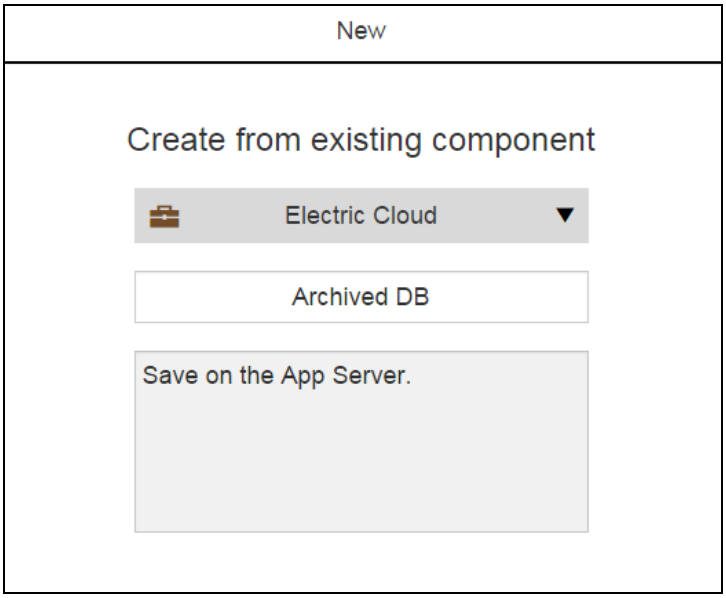
   Example:



5. Change the name of the component, and enter an optional description.

   You can also change the project to which the master component belongs.

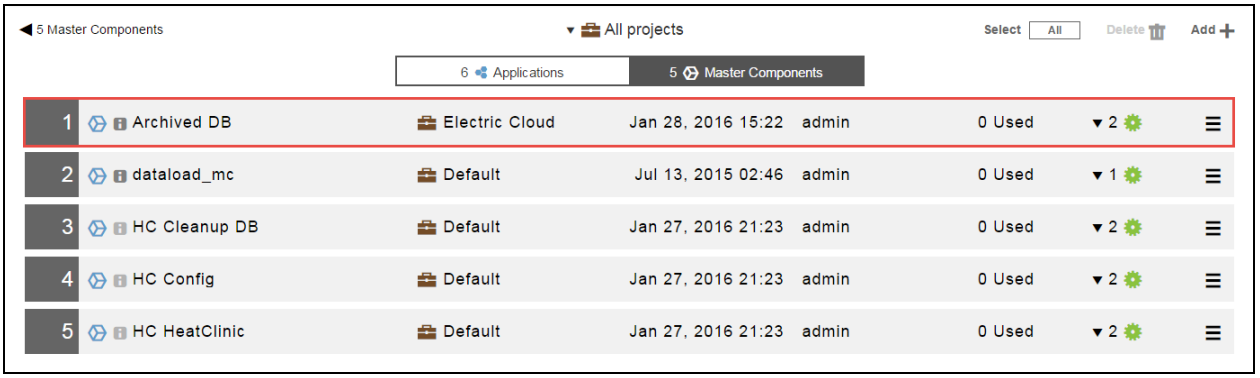
6. **Example:**



7. Click **OK**.

The Master Component List now includes master component you just created. It shows the list of available master components. The default is to show the master components for all projects. If you want to see only the master components for a specific project , click the down arrow in the **All projects** field to select one or more projects one of these ways:Click on the name of one or more projects.Enter the search criteria in the Search field. The projects that match the criteria appear in the list.You can also search for one or more master components by entering the search criteria in the **Search** field next to the **All projects** field. If there are no matches, a message appears stating that there are no master components in the selected projects.

Example:



Creating Master Components Based on Existing Master Components

Starting from the Master Components List:

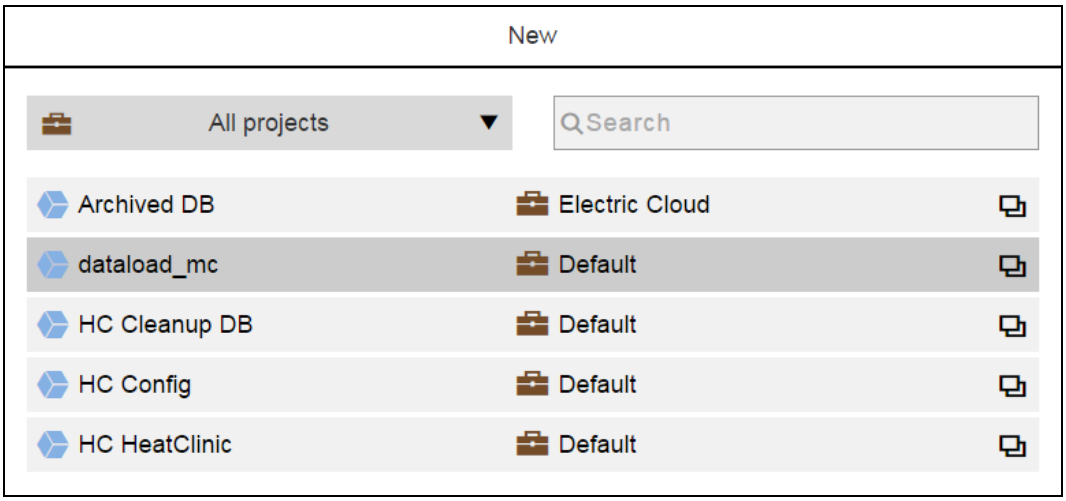
1. Click **Add +**.

   The **New** dialog box opens.

2. Click **Master component** > **Copy Master Component**.

   A list of existing master components opens.It shows the list of available master components. The default is to show the master components for all projects. If you want to see only the master components for a specific project , click the down arrow in the **All projects** field to select one or more projects one of these ways:Click on the name of one or more projects.Enter the search criteria in the Search field. The projects that match the criteria appear in the list.You can also search for one or more master components by entering the search criteria in the **Search** field next to the **All projects** field. If there are no matches, a message appears stating that there are no master components in the selected projects.

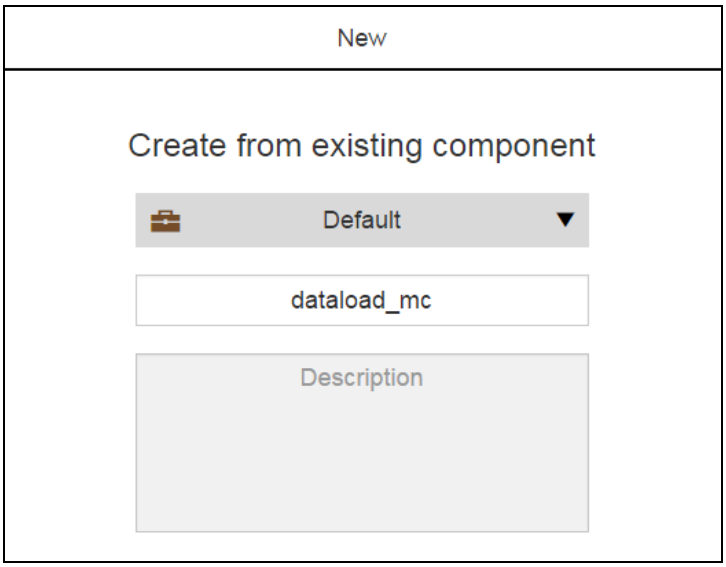
Example:



3. Select an existing application component.

   The **Create from existing component** dialog box opens with the name of the selected component highlighted.

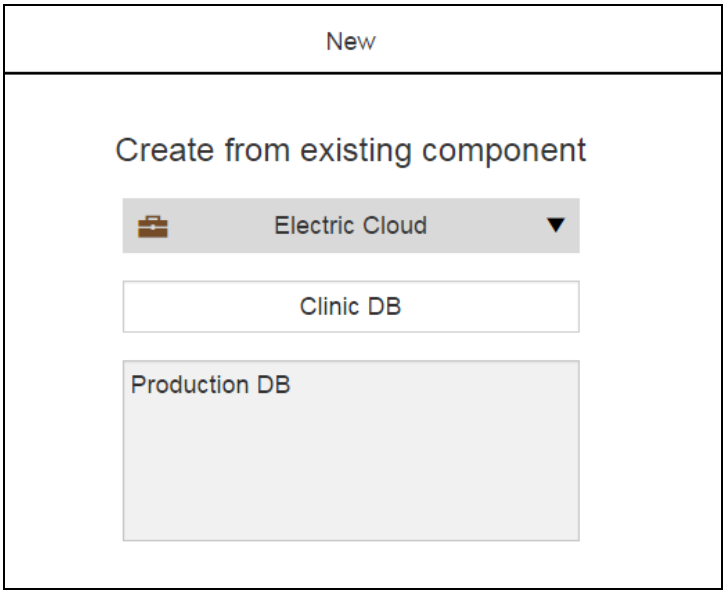
   Example:



4. Change the name of the component, and enter an optional description.

   You can also change the project to which the master component belongs.

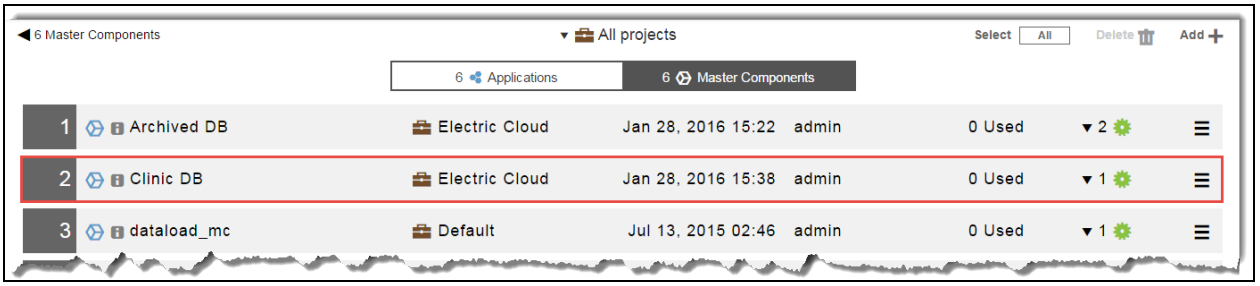
5. **Example:**



6. Click **OK**.

   The Master Component List now includes master component you just created. It shows the list of available master components. The default is to show the master components for all projects. If you want to see only the master components for a specific project , click the down arrow in the **All projects** field to select one or more projects one of these ways:Click on the name of one or more projects.Enter the search criteria in the Search field. The projects that match the criteria appear in the list.You can also search for one or more master components by entering the search criteria in the **Search** field next to the **All projects** field. If there are no matches, a message appears stating that there are no master components in the selected projects.

Example:



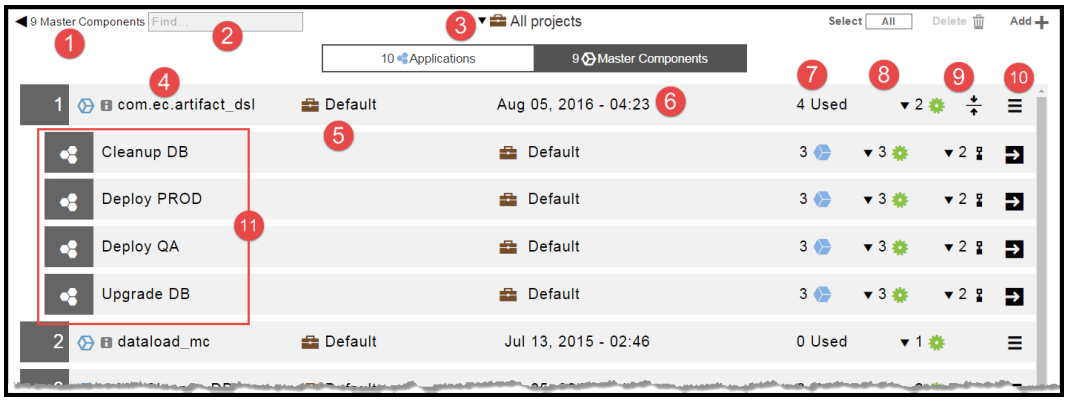
Master Components List UI

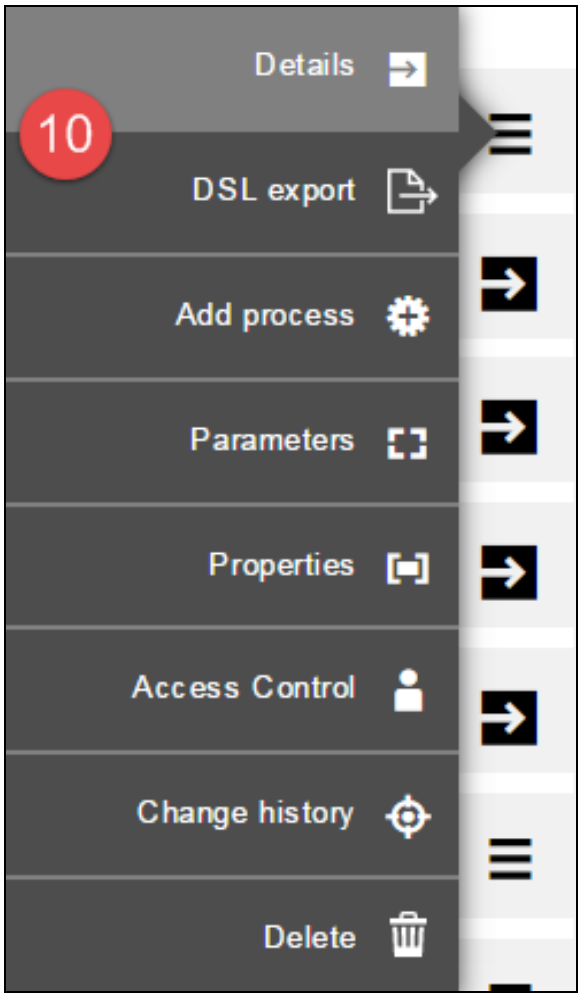
How to get here:

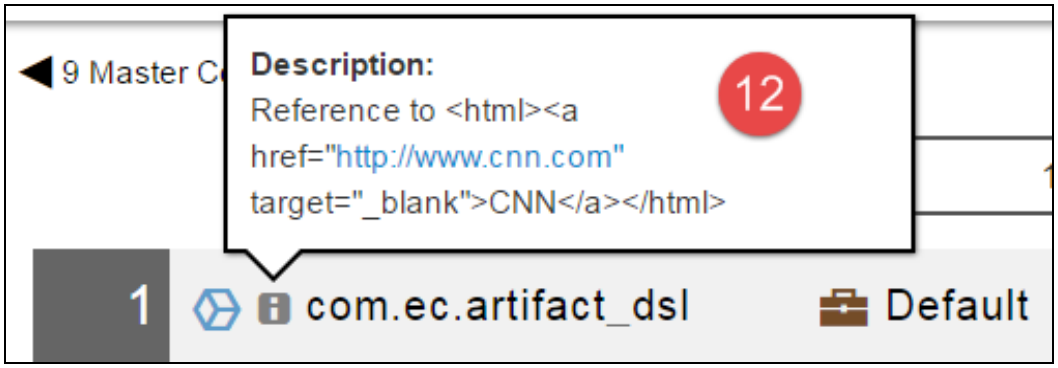
One of these ways:

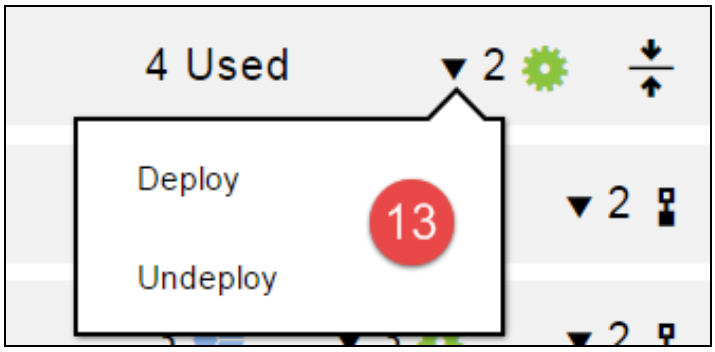
- From the Home page (https://<*ElectricFlow_server*>/flow/):
  - Click the **Applications** launch pad to open the Applications List.
  - Click the **Master components** tab.
- Click the **Main menu** button, and then click **Applications** > **Master Components**.

This is a Master Components List:




Details
10
DSL export
Add process
Parameters
Properties
Access Control
Change history
Delete


9 Master C
Description:
Reference to <html><a
href="http://www.cnn.com"
target="_blank">CNN</html>
12
1
com.ec.artifact_dsl
Default



1	Breadcrumb identifying the object you are viewing and the total number of master components.
2	Breadcrumb
3	It shows the list of available master components. The default is to show the master components for all projects. If you want to see only the master components for a specific project , click the down arrow in the **All projects** field to select one or more projects one of these ways: • Click on the name of one or more projects. • Enter the search criteria in the Search field. The projects that match the criteria appear in the list. You can also search for one or more master components by entering the search criteria in the **Search** field next to the **All projects** field. If there are no matches, a message appears stating that there are no master components in the selected projects.
4	Name of the master component.
5	Name of the project to which the master component belongs.
6	Date and time when the master component was created.
7	Number of applications dependent on this master component reference.
8	Number of component processes for the master component.
9	Hide or show the applications that reference the master component.

10	Click the **Menu** button to open the context menu. You can get more details about the master component: • **Details**—The name and description of the object. • **DSL Export**–Click this to quickly and easily generate and export a DSL script from the ElectricFlow UI. • **Add Process**-Add a component process. The **Component Process** dialog box opens. A master component can have one or more component processes. • **Parameters**—The parameters defined for the master component. • **Properties**—The properties in the object. • **Access Control**—The access control configuration in the automation platform for the object. • **Track Changes**—The change history of the object. • **Delete**–Delete this object.
11	Applications that reference the master component. This allows you to perform quick impact analysis to understand which applications would be impacted if a referenced master component is changed.
12	Clicking the **i** button shows information about the element. This shows the description of the master component.
13	Clicking on the down arrow next to number of component processes (number 9 above) shows the list of processes for this master component.

Artifact Staging

You can use artifact staging to retrieve artifacts that will be deployed in an application run before the deployment starts. This ensures that all the artifacts are downloaded and available on the target, either the artifact cache (the default) or a directory specified in the component definition. Staging artifacts minimizes the time it takes to retrieve artifacts and reduces downtime during the deployment.

Artifact staging is enabled by default in the UI. However, when you use the API commands in the command line interface or in a script, make sure to enable artifact staging when modeling the application.

See Deploying and Troubleshooting Applications on page 287 for how to use artifact staging when deploying an application.

At the Application Level

When artifact staging is enabled in a deployment model, ElectricFlow dynamically creates and executes a job step to stage the artifacts. If the application process being deployed includes process branching, the staging job step retrieves all artifacts in the application process in parallel to reduce the staging time. In addition, the deployment processes, including the staging job step, are skipped on disabled resources.

You can deploy applications with the artifact staging enabled and following options:

- With smart deploy enabled, the staging job step retrieves artifacts only for those objects that have not been deployed to specific resources.
- With full deploy enabled, the staging job step retrieves artifacts only for all artifacts in the application.
- With partial deploy enabled, the staging job step retrieves artifacts only for artifacts in the application.
- With a selected snapshot, the staging job step retrieves artifacts only for artifacts in the selected snapshot.

> **Note:** For a component process, the process step to retrieve an artifact is optional because the component process is referenced in the application process. If you author a component process, the process type should be "Deploy" to ensure that the artifacts defining the component can be staged.
> If an artifact staging for a component fails, it will then be retried when the process execution continues and when the actual component process execution happens.

You can view the progress of the staging job step in the Application View Run page.

- When the staging job step is successful, the artifact is retrieved and staged before the deployment run starts.
- When the staging job step fails, the application process automatically retrieves the artifact during the component retrieval step in the application run.

The environment inventory is updated as the job step runs. In addition to viewing the status of the resources in the environment, you can also view the state of other objects including the application processes, components and component processes, and artifacts being deployed.

At the Pipeline and Release Levels

Artifact staging is supported in pipelines and Releases.

- When you define the tasks in a pipeline stage, you can select application processes with artifact staging. The staging job step is executed as part of the application in the pipeline run as described in the previous section. See Pipeline Tasks on page 419 for more information about stage tasks in a pipeline.
- The Release definition includes the pipeline and applications that you want to run in the Release. See Release Definition on page 646 for more information about the release definition.

Rollback

When an application process has a predefined rollback step and the application deployment fails, you can restore the deployment model to a previous state. Application rollback occurs only for components that were not skipped (because of branching) in the application process. You can define multiple rollback steps.

How ElectricFlow Determines the Rollback Deployment

When a rollback step is encountered because of a deployment failure, one of the following options is invoked based on the rollback step configuration:

- Roll back to a previous state—ElectricFlow looks for a successful deployment with the same process name. Then, it starts the rollback using the same process and parameters from that deployment with component versions determined by the inventory state before the failed deployment. If a successful deployment is not found, the rollback fails. This option requires a successful deployment even when you are deploying with a snapshot. The last successful deployment could be part of a snapshot deployment or a regular application deployment.
- Roll back to a snapshot—ElectricFlow uses a user-specified snapshot for the rollback. The last successful deployment process and parameters that are stored as part of the environment snapshot are used for the rollback. Component versions are defined as part of the snapshot. If this snapshot is an application snapshot, ElectricFlow looks for a successful deployment of the process with the same name, which is the same as the first option.

For both options, a combination of component definitions and deployment runtime parameters determine the version of the components that will be deployed.

Ways to Perform Rollback

ElectricFlow provides these ways to do rollback:

- Automatic rollback—ElectricFlow automatically reverses what was deployed up to the failure point in the process flow. See Rollback on page 100 for more information.
- Snapshots—You manually create and save snapshots of successful deployments and run a snapshot the next time the application is deployed. See Snapshots on page 119 for more information.
- A component process defined as an undeploy process–When defining a component process, select **Undeploy** as the process type.

Triggering Rollback Even if Undeploy Fails

You can trigger rollback even if undeploy fails. To do so, use the **Continue running** option in the **Application Process Step** dialog box when configuring a rollback step.

Enabling Rollback Only on Components that Successfully Undeployed

You can limit rollback to those components that undeployed successfully. To do so, use the **Rollback only successfully undeployed components** option in the **Define Step** dialog box when defining a rollback step.

Enabling Rollback Only on Failed Components

ElectricFlow automatically tracks components that failed during application deployment and will roll back only those components. This option is useful when the components do not have version dependencies. You can use **Component Level Rollback** with automatic rollback and with snapshots.

Using Automatic Rollback

ElectricFlow supports a rollback step in application processes. This allows an automatic rollback to the environment before a bad deployment or to any previous environment snapshot. This reduces the time to author the rollback logic.

During an application deployment, the predefined rollback operation is invoked when the rollback step runs. ElectricFlow reverses what it executed in the process flow up to the application process step where the failure occurred.

This operation is supported only in application processes. It puts the deployment in the same state as the last successful deployment, which is usually the same as the previous deployment state.

If the **Roll back to previous state** option is enabled and no last good deployment exists (for the restore point), the rollback step is skipped. The undeploy step (if configured) will still run.

When you define an application process step as a rollback step, the application can roll back to the previous state up to where the deployment failed (the default) or to a specific snapshot. Also, you can specify how the application will be rolled back:

- By default, the **Smart Deploy** option is enabled. If you want to execute a full or partial run the next time the application is deployed, disable this option.
- When the **Undeploy Application before Rollback** option and the application process that you want to undeploy (in the **Select Process** field) are selected, the environment inventory record is removed before the rollback operation is executed. This allows dirty files in the bad deployment to be removed automatically. When you use the **Undeploy Application before Rollback** option with the **Component Level Rollback** option, failed components are undeployed and then rolled back.

When a specific snapshot is selected for rollback, ElectricFlow tries to determine the last successfully-deployed process and then tries to execute it from the snapshot. While a native rollback step is available, you have complete control over creating a custom rollback handling operation and using that as part of the error handling logic.

> **Note:** If the last successful deployment was a partial deployment and the rollback job is reverted, ElectricFlow reverts only the components included in the partial deployment.

Requirements for Using Rollback

You must:

- Enable change history.
- Redeploy an application successfully before using rollback after a full import.
- Keep the original application process name. If you rename it after a successful deployment, and if rollback to the previous state is configured, ElectricFlow cannot determine the last successful deployment. In this case, rollback will fail (but the undeploy process will still occur).

Guidelines for a Successful Rollback

You should:

- Use environment snapshots for rollback (not application snapshots).
- Use smart deployments instead of partial deployments:
    - If the last successful deploysment was a partial deployment, the rollback job reverts only components in the partial deployment.
    - If you use partial deployments, you should take a manual snapshot (using EF-Utilities) and use it with the rollback step.
    - Use the `$[]` property reference to reference the snapshot so that it can be dynamically set.
- Use environment snapshots instead of application snapshots for reliable software deployments.

Managing Application or Microservice Dependencies

Today's business applications are generally built by integrating third-party applications or microservices with in-house applications or microservices. ElectricFlow lets you model and manage dependencies across applications or microservices such as these in your product. With the dependency information captured, ElectricFlow enforces these dependency rules to ensure quality deployments.

Key Benefits

Capturing and using application or microservice dependencies has the following benefits.

Easier Deployments

With dependencies, ElectricFlow can help construct your deployment plan. For example, if a dependency is captured as A > B > C > D, then ElectricFlow can enforce these dependency rules at runtime.

More Reliable and Safer Deployments

Captured dependencies make safe deployments by preventing you from making mistakes that violate the dependency information that was captured.

Application or Microservice Dependency Rules

Dependency rules can be captured at two levels:

- At the application or microservice level—Application- or microservice-level dependency rules let you capture simple dependencies between applications or microservices. For example, a marketing application might require just a version of a deployed user authentication application to exist in order for it to function.
- At the application or microservice version level—An application or microservice version in ElectricFlow maps to a snapshot name. A snapshot stores the state of the application or microservice at a point in time. You can use the application or microservice snapshot names to create complex version dependency rules.

You can also use regular expressions with the dependency rule configuration. For example, version 'ms1' (a snapshot) of a marketing application depends on versions 'uas2' and 'uas3' (snapshots) of a user authentication application.

Application or Microservice Dependency Rule Examples

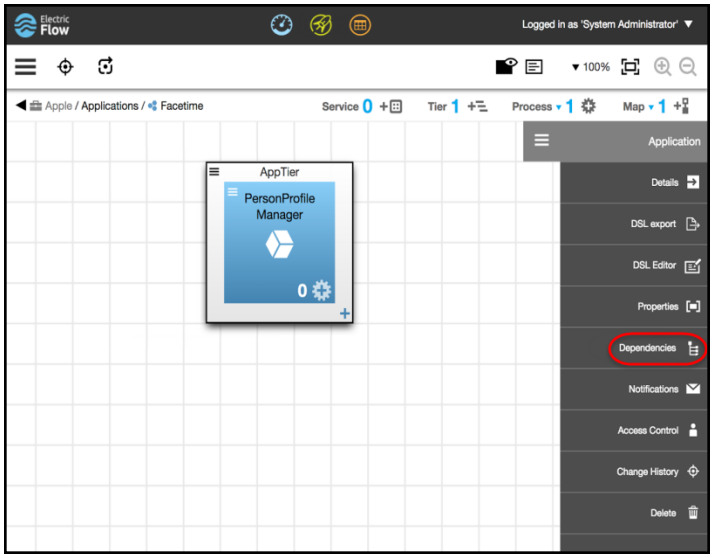
Rule	Parent Application or Microservice	Version	Child Application or Microservice	Version	Notes
1	A	Any	B	Any	Application or microservice A depends on B. This is a very generic rule. With only this rule, you can deploy any version of application or microservice A as long as any version of application or microservice B is deployed (rule 1).
2	A	1.2	B	1.1.5	Application or microservice A version 1.2 requires application or microservice B version 1.1.5. If you try to deploy application or microservice A version 1.2, the system ensures that application or microservice B version 1.1.5 (rule 2) is deployed.
3	A	1.2	B	1.2.0	Application or microservice A version 1.2 depends on application or microservice B version 1.1.5 or 1.2.0. If you try to deploy version 1.2 of application or microservice A, the system ensures that application or microservice B version 1.1.5 (rule 2) or 1.2.0 (rule 3) is deployed.

Rule	Parent Application or Microservice	Version	Child Applic-ation or Microservice	Version	Notes
4	A	2.*	B	1.8.*	Any 2.* version of application or microservice A requires any 1.8.* version of application or microservice B. If you try to deploy version 2.2 of application or microservice A, the system ensures at least one 1.8.* version of application or microservice B (rule 4) is deployed.
5	A	2.*	C	3.*	Any 2.* version of application or microservice A depends on any 3.* version of application or microservice C. If you try to deploy application or microservice A version 2.2, the system validates to ensure at least one 1.8.* version of application or microservice B is deployed (rule 4) and at least one 3.* version of application or microservice C is deployed (rule 5). If you try to deploy version 4.0 of application or microservice A, then the system checks to ensure at least one version of application or microservice B is deployed (rule 1).

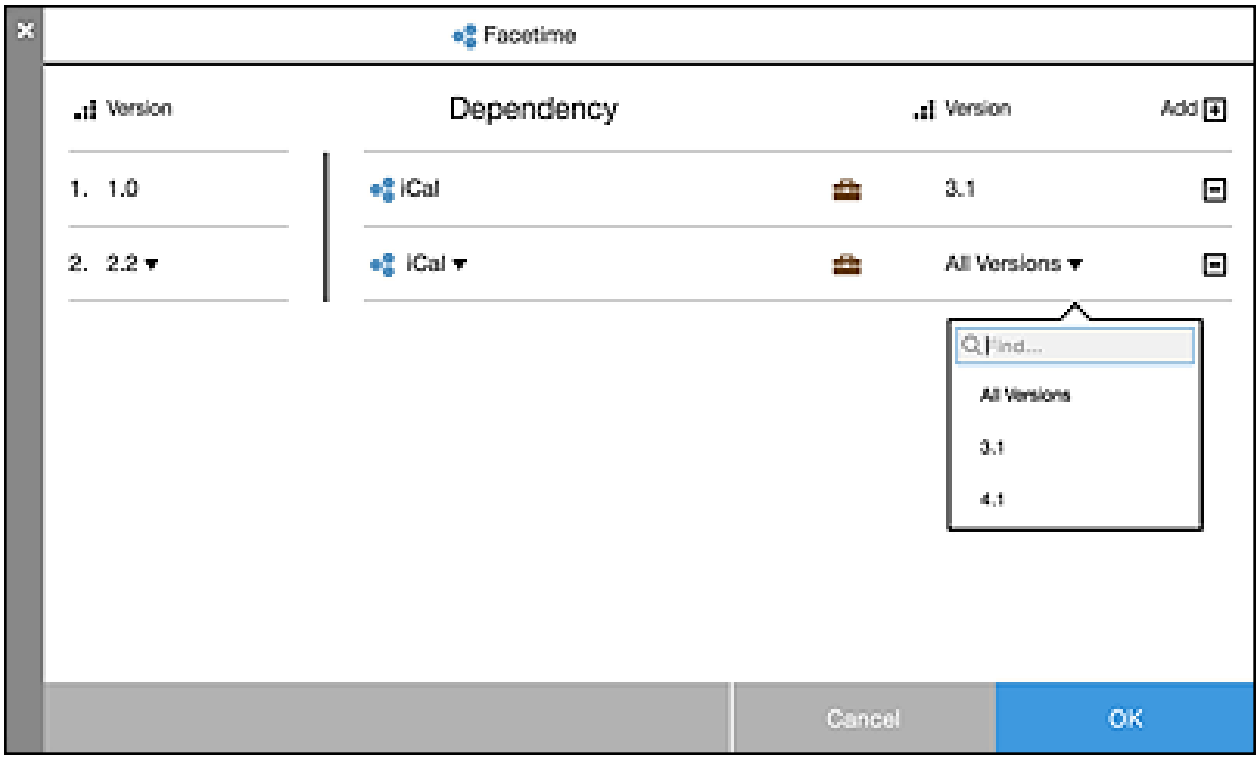
Rule	Parent Application or Microservice	Version	Child Applic-ation or Microservice	Version	Notes
6	A	2.7 Note: application or microservice A version 2.7 is created on 05/21/2017	B	1.9.2	You can use an `effective_date` attribute such as `EFFECTIVE_DATE 05/20/2017` on the dependency rule. This is an advanced feature that is not available from the UI. Specifying an effective date applies these rules only to versions that are created on or after the date. If you try to deploy application or microservice A version 2.7 on 5/25/17, then the system checks to ensure that version 1.9.2 of application or microservice B is deployed. This is true for all versions of application or microservice A created after 05/20/2017 (rule 6) and at least one 3.* version of application or microservice C is deployed (rule 5). If you try to deploy application or microservice A version 2.2 (or any version created before 05/20/2017), the system validates to ensure at least one 1.8.* version of application or microservice B is deployed (rule 4) and that at least one 3.* version of application or microservice C is deployed (rule 5).

Example of How to Create Application or Microservice Dependencies

Navigate to the application or microservice Architecture page and select **Dependencies** from the menu. The following example shows the menu option for an application:



A dialog appears in which you can create the dependency rules. For example:

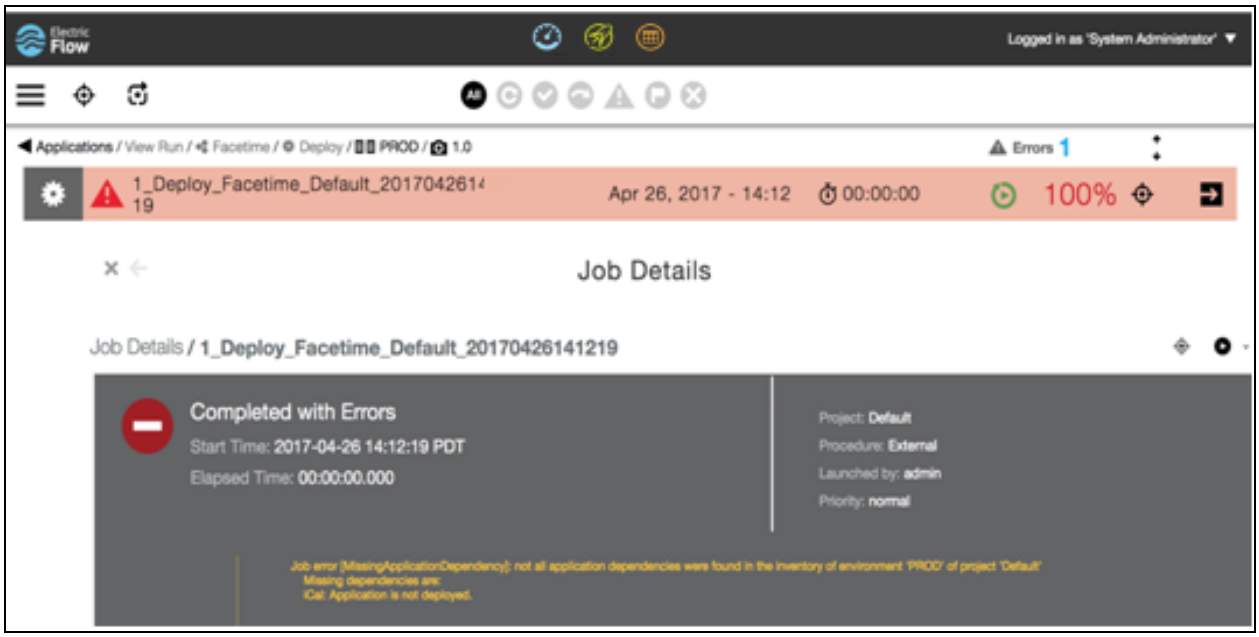


In this example, you create a dependency rule for a FaceTime application that has a dependency on the iCal application that manages the calendar information. The rule specifies that FaceTime version 1.0 requires iCal version 3.1 to be deployed.

You can also specify dependency rules based on regular expressions while setting up the dependencies. For example, you can enter `4.*` for the iCal version as part of the second rule shown above.

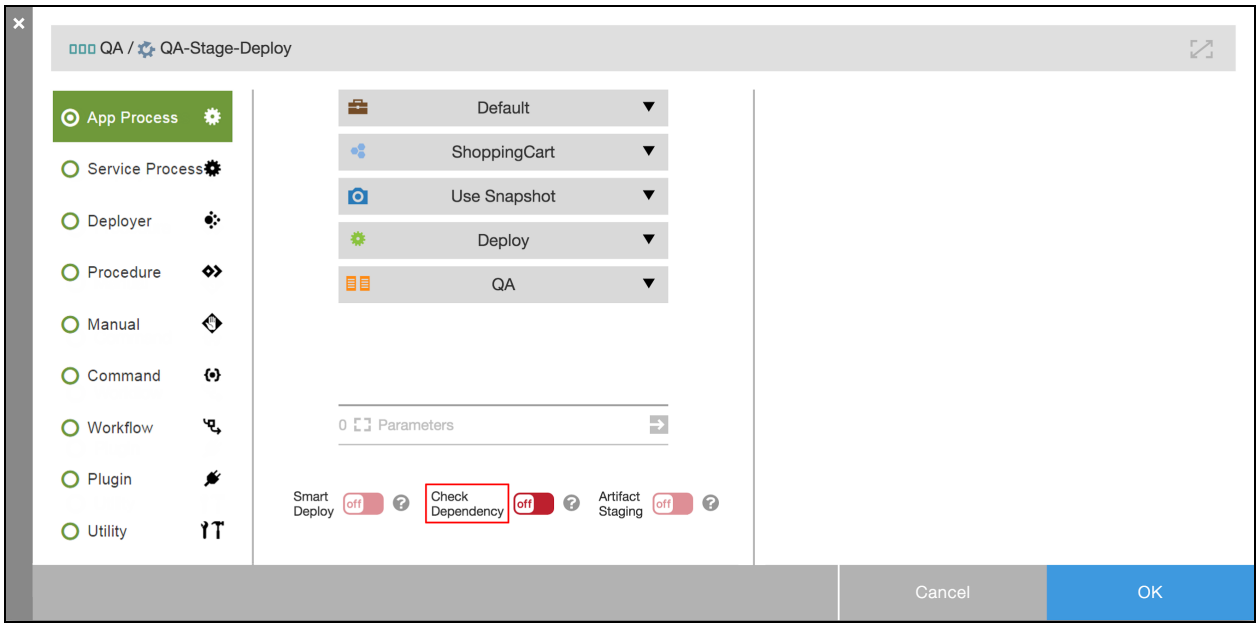
Once all the dependencies are captured, then at application or microservice deployment time, you can enforce these dependencies or ignore them. By default, dependency checks are enforced. This means that if the dependency validation fails, then the deployment errors out. You can disable this behavior by toggling the **Dependencies** option shown below. When enforcement is disabled, then violations are treated as just warnings, and deployment will proceed.

If the user tries to deploy the Facetime application version 1.0, it will error out because of a missing iCal version dependency as in the example below:



Application or Microservice Dependency Configuration in a Pipeline

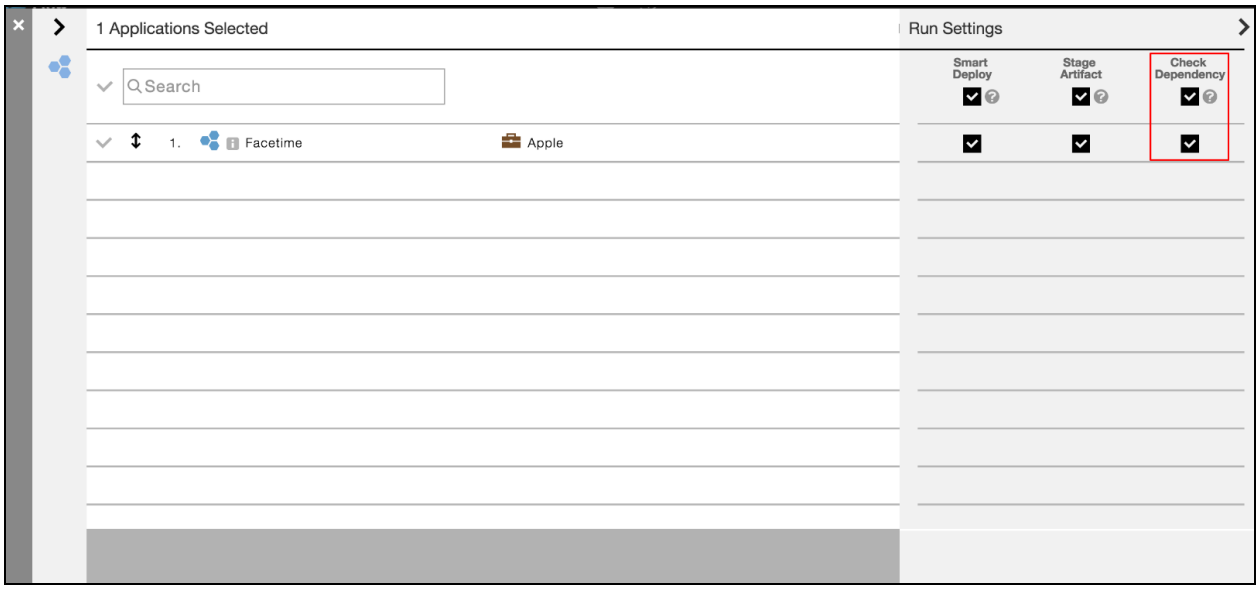
You can enable a dependency check while defining the App Process or Service Process pipeline stage task definition. For example:



Application or Microservice Dependency Configuration in a Release

Similarly, while setting up a release, you can configure a dependency check for each application or microservice in the release. Once the dependency checks are configured, the deployer ensures that dependency requirements are met before deploying each of these applications or microservices in the

release. The following example shows how to configure dependency checks for a release that contains one application:



Manual Tasks and Steps

You can define manual steps and tasks in application or component processes and in release delivery pipelines. They describe the actions that must be taken by a human to complete the process step and/or pipeline task. In application and component processes, these actions are referred to as *manual steps*. In pipelines, they are referred to as *manual tasks*. Any time a manual task or step is reached in the process flow, an email notification is sent to all users in the assignee list.

Manual steps and tasks consist of user-specified instructions and assignees.

> **Note:** Email notifications for manual process steps are sent to assignees whether or not notifications are enabled at the application or process level. If a user does not want to receive an email notification, their name should be removed from the assignee list.

- To set up a manual task or step:
  - Enter instructional text to help the approvers. This text can include URLs.
  - You can also define optional parameters to collect information during the approval step.
  - Specify who can approve. You can define the step or task to accept either user or group IDs *or* variables using the `${[ ]}` notation to represent assignees as property references, but *not both*.

- To attach a parameter to a manual task or step:
  - Use the `$[ ]` notation as a property reference to access other user-specified parameters at runtime, such as evidence and status.
  - For a pipeline manual task, use `$[/javascript myFlowRuntime.flowRuntimeStates[0].evidence]` to collect the user's response in subsequent tasks. This command using (`0`) assumes that the manual task is the first task in the stage.
  - For an application process, use `$[/myJobStep/comment]` or `$[/myJob/jobSteps/stepName/comment]` to access the instructional text, which is usually in the notification emails sent to the users who perform the actions or who approve the completion of the manual step.
  - Put hyperlinks in the instructions and comments fields in the GUI. You can create a file on a server or other storage device and add the URL to that file in the GUI.
- To specify a default email template:
  - For a manual task in a pipeline, use `/ec_deploy/ec_pipelineNotifierTemplates/ec_default_pipeline_manual_task_notification_template`.
  - For a manual process step in an application, use `/ec_deploy/ec_notifierTemplates/ec_default_manual_process_step_notification_template`.
- See Example: Authoring a Pipeline with Manual and Utility Tasks on page 557 for how to define a manual task in a pipeline stage.

Process Branching

You can use process branching to specify the path through an application or component process based on transition conditions other than out-of- the-box options. Decisions about the next step in the process are made while the process runs. This is similar to the transition conditions for workflows in the automation platform. If the application or component process applies to multiple use cases, you can design one process with two or more branches instead of designing multiple processes for each use case. You can also define steps that run in parallel.

For example, to install or upgrade software, you can define one process for multiple use cases and use the same steps except for the following:

- The source files can be in .zip or .tar format. The steps to extract the files depend on the format.
- The operating system can be Linux or Windows. The steps to download the files, install them on the server, and enter commands depend on the operating system.

ElectricFlow supports the following branching conditions. The default is **Always**.

- Completion status of the previous process step
- A property set in another part of the system, not the in the previous step
- Custom validation rules

Using Process Branching

You define process branching when authoring an application or component process.

Process Branching Example

This example shows an application process with process branching that deploys a .war file.

Clicking a connector in a link opens the branching conditions menu. Depending on the location of the connector, some or all of these options may be available (enabled):

- **Always**–Always go to the next step, referred to as the target.
- **Successful**–Go to the next step if the previous step, referred to as the source, is successful.
- **Failure**–Go to the next step if the previous step fails.
- **Add Condition**–Add a custom condition that must be met before going to the next step.
- **Add Connector**–Add a connector from the source of the link to a new target by selecting one of the highlighted eligible steps. You can only select an eligible step.
- **Change Source**–Change the source by selecting one of the highlighted eligible steps, which has a red outline. You can only select an eligible step.
- **Change Target**–Change the target by selecting one of the highlighted eligible steps, which has a red outline. You can only select an eligible step.
- **Delete**–Delete the selected connector and link.

When you select the connector between the Start step and first step after it, only some of conditions appear and only some are available. The condition between the Start and the next step is **Always**, the default branching condition.

When you select the connector between subsequent steps, all of the conditions are available.

Usage Guidelines

Follow these guidelines when the process includes process branching:

- When you add a step, you must define it before adding another step.
- You can only configure branching conditions on a connector between two process steps.
- You cannot configure branching conditions between these objects:
  - The start of the process and the steps immediately after it.
  - The end of the process and the steps immediately before it.
- When defining a step in an application or component process, you configure what ElectricFlow does when an error occurs. *This setting overrides any job-step-level branching condition.*

  You can select **Stop running** or **Continue running** in the **On Error** field in the Define Step dialog box. If an error occurs in a job step and **Stop running** is selected, ElectricFlow aborts the process even if the branching condition is set to **Failure**.

Process Branching States and Conditions

State of the Branching Condition Connectors in the UI

In the ElectricFlow UI, the status of the link is based on the shape and color of the connector.

Shape	Color	Link Status
Diamond	Light gray	Always
Diamond	Dark gray	Disabled

Shape	Color	Link Status
Square	Green	Successful
Circle	Red	Failure

Examples of Branching Conditions

These are examples of branching conditions that you can apply in your processes.

- Based on the status of the previous step

  Follow the branch based on the result of the previous step: Successful, Failure, or both (Always).

  Example:

  - **Successful**–If the file is downloaded successfully, the next step is to extract the files.
  - **Failure**– If the file was not downloaded properly, the next step is to abort the process.
  - **Always**—The next step is to always extract the files.

- Based on a value of an operation during the step

  Follow the branch that matches the result of an operation such as calculating a value or processing data during the step.

  Example: The result of an operation is a file type.

  - If the result is an XML zip file, the next step is open an XML text editor.
  - If the result is a .htm file, open a web browser.
  - If the result is a .mov file, open an application to play the movie.

- Based on a property in another part of the system

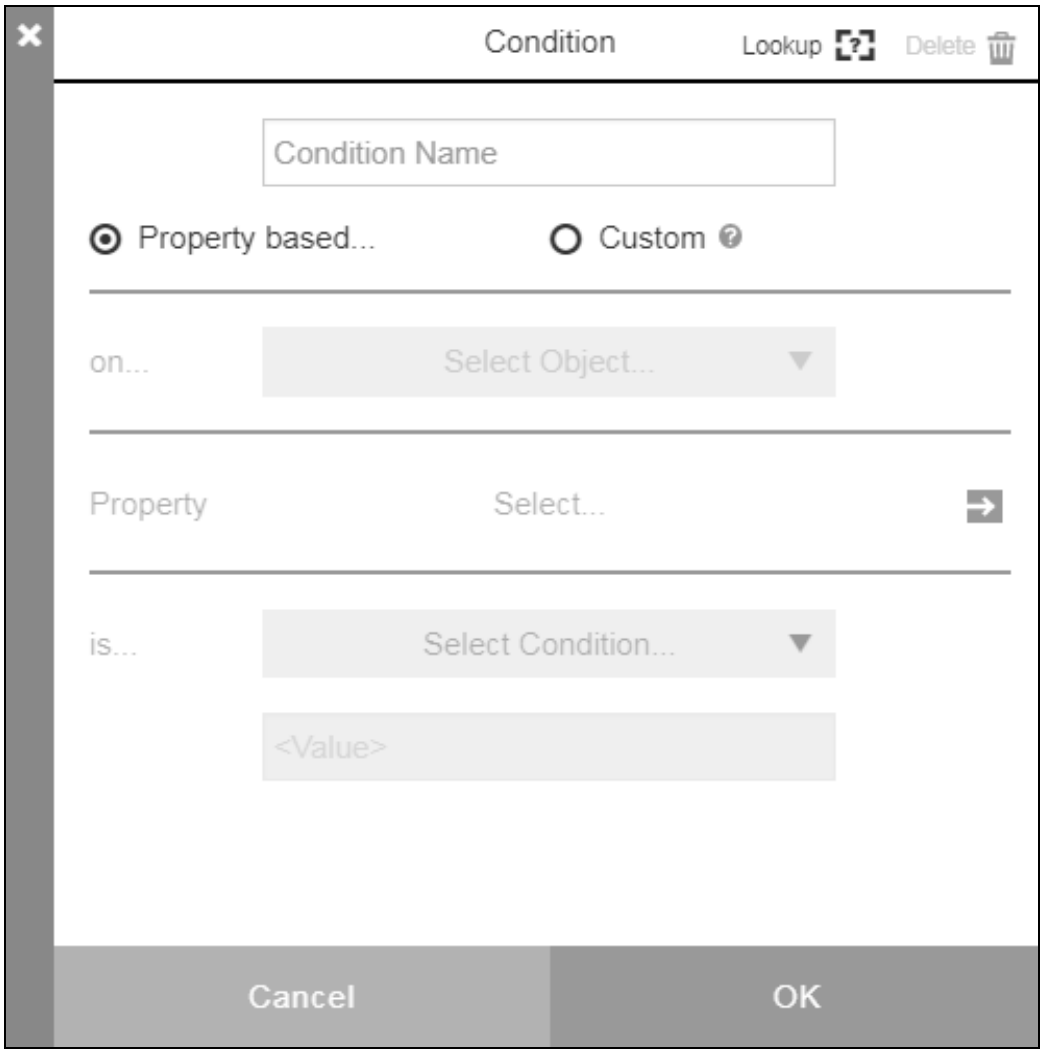
  Follow the branch based on a property set in another part of the system, not in the previous process step.

  Example:

  - If the property `os_type = linux` is set on a resource, always follow the branch for Linux steps.
  - If the property `release_type` is set to minor in the application, always follow the branch for minor releases when running the process.

Custom Conditions in Process Branching

In a component or application process with branching, when you click the connector on a link between two steps, the Condition dialog box opens.

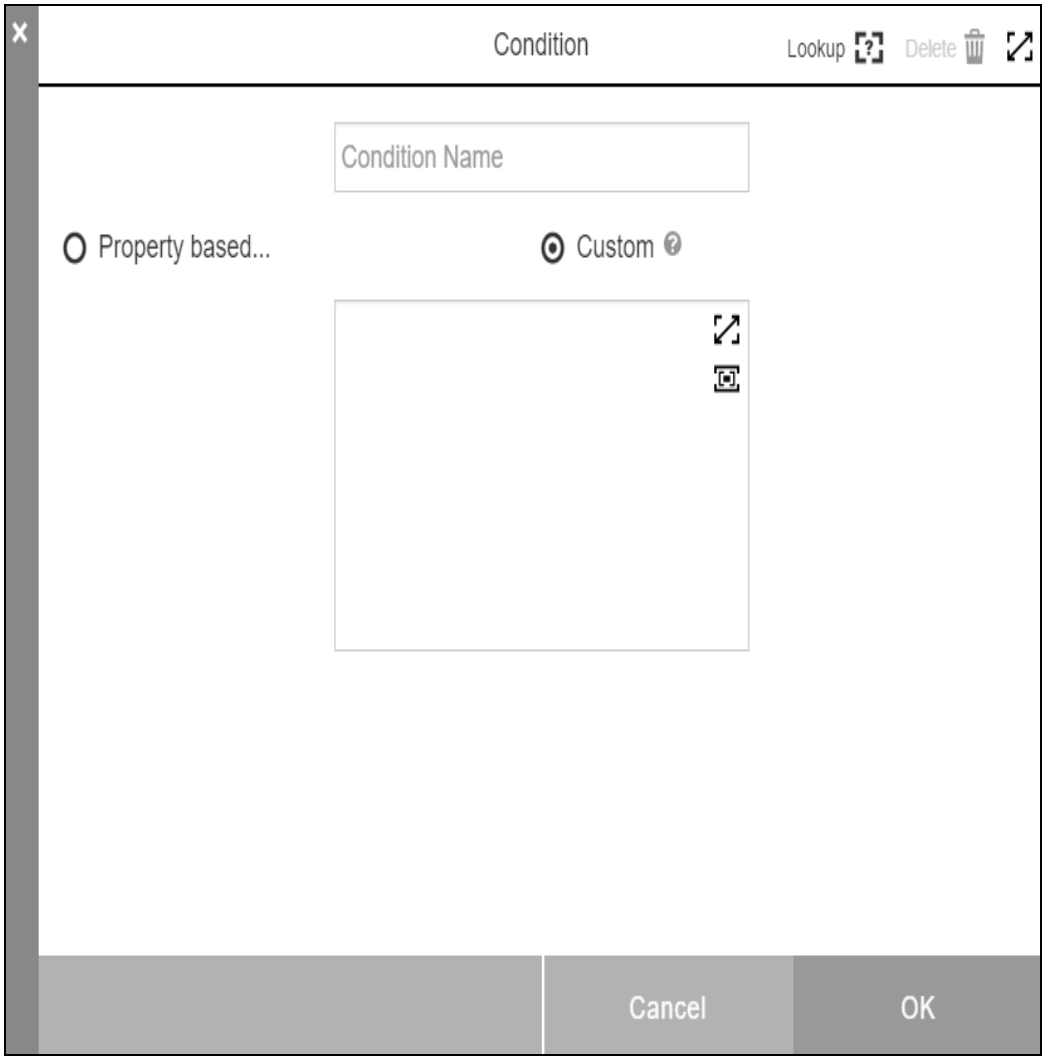


When you click the connector between the "Deploy War" and "Start Server" steps, these conditions appear:

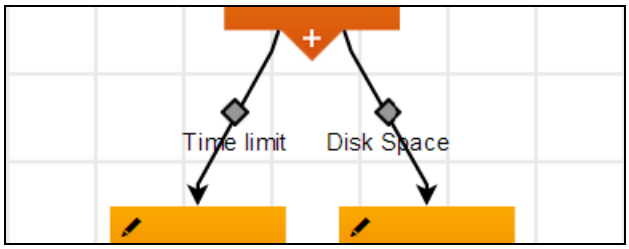
- **Always**–Always go to the next step, referred to as the target.
- **Successful**–Go to the next step if the previous step, referred to as the source, is successful.
- **Failure**–Go to the next step if the previous step fails.
- **Add Condition**–Add a custom condition that must be met before going to the next step.
- **Add Connector**–Add a connector from the source of the link to a new target by selecting one of the highlighted eligible steps. You can only select an eligible step.
- **Change Source**–Change the source by selecting one of the highlighted eligible steps, which has a red outline. You can only select an eligible step.
- **Change Target**–Change the target by selecting one of the highlighted eligible steps, which has a red outline. You can only select an eligible step.
- **Delete**–Delete the selected connector and link.

When you configure a **Property based** condition, the fields in the Condition dialog box remain the same.

When you configure a **Custom** condition, the fields change. The property picker becomes available to you from the button.



After you configure your conditions, they appear near the affected connectors in the process.



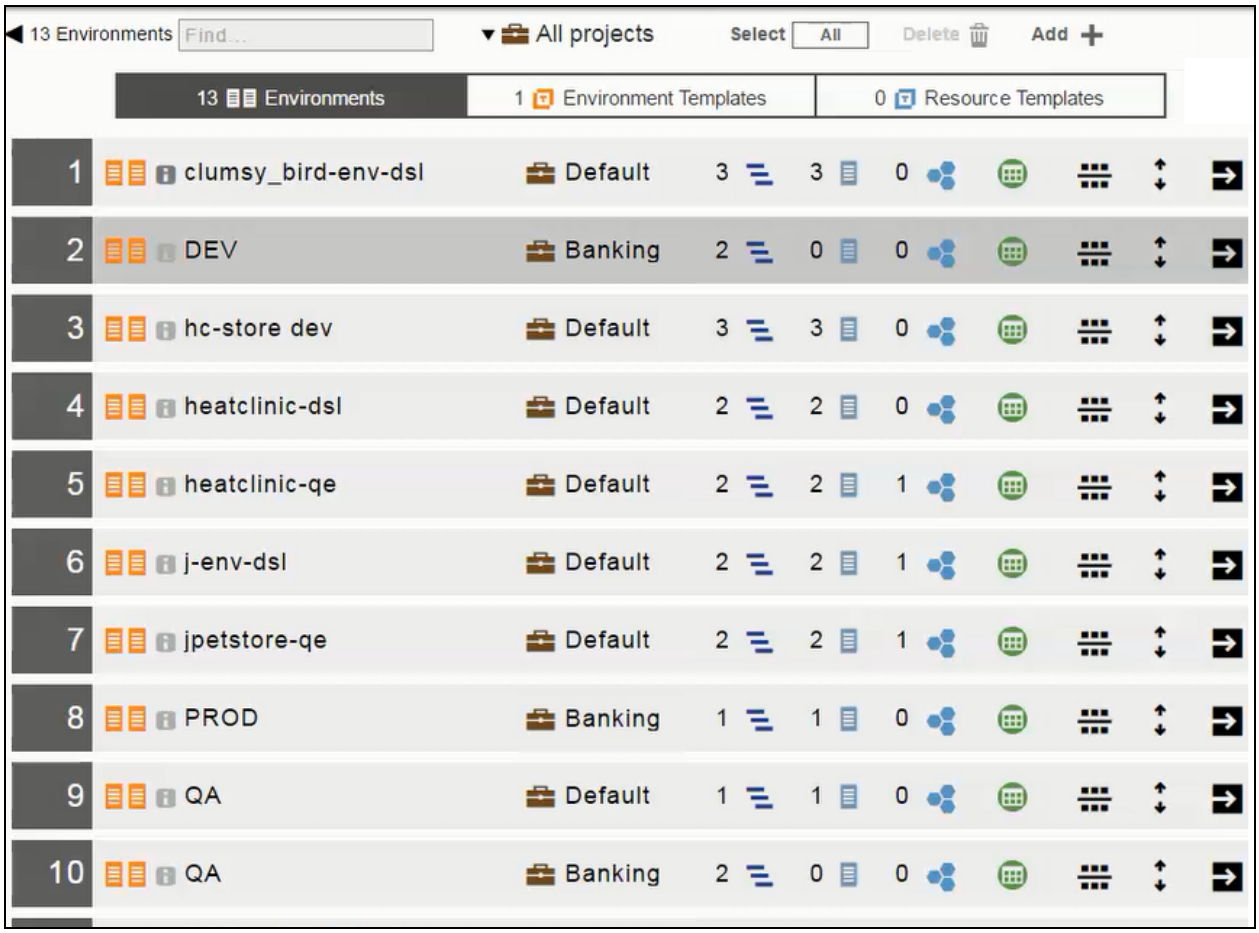
Property Reference Use Case

Property referencing is one of the most powerful features in the ElectricFlow product suite. Mastering property referencing will let you extend ElectricFlow to meet your organization's deployment needs.

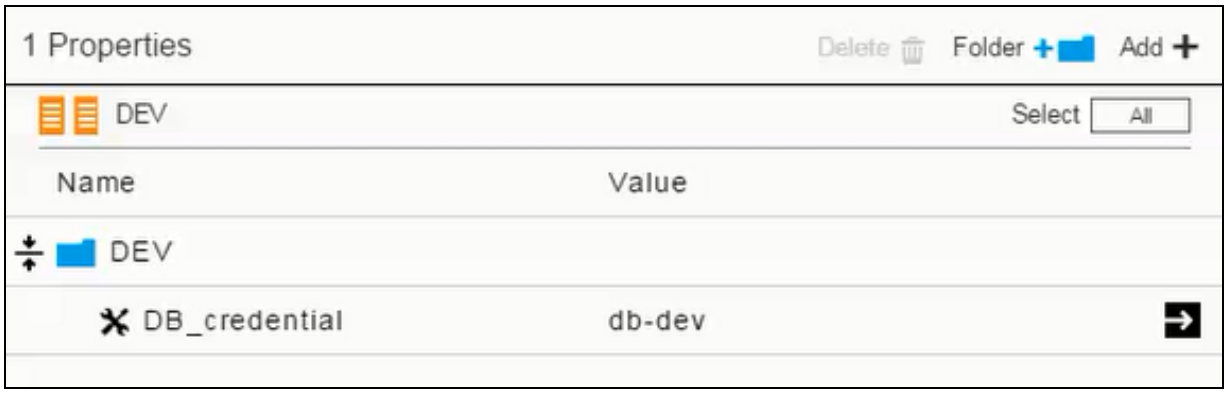
This use case shows how to leverage properties to create a flexible deployment model. It assumes that you are developing an ElectricFlow application for a web application deployment to production. The deployment process requires that the application is properly tested in the DEV and QA environments before deploying it to the production. Each of these environments uses different database connections for certifying various aspects of the web application. The QA environment uses a database that is created specifically to test the scalability aspects of the application. The challenge is to determine how to manage the database configuration across these environments.

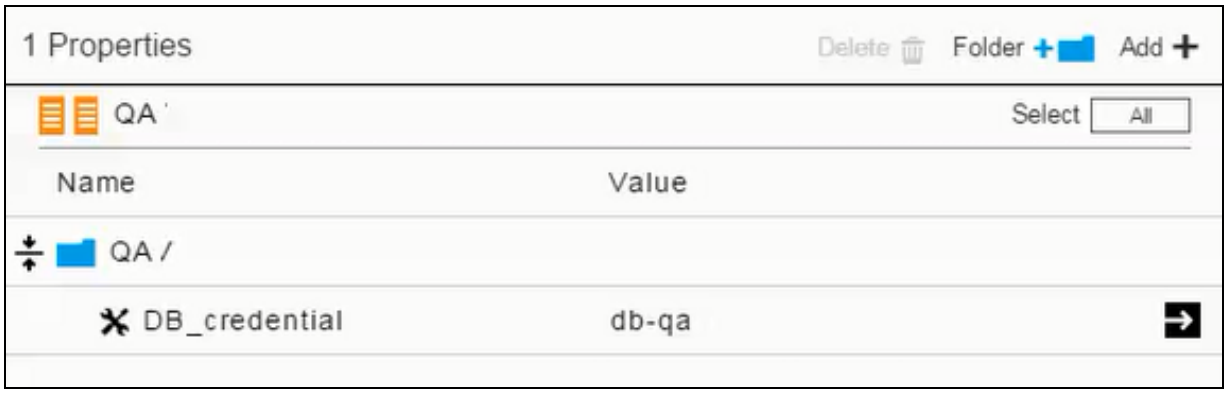
Start by modeling three environments under a project called Banking:

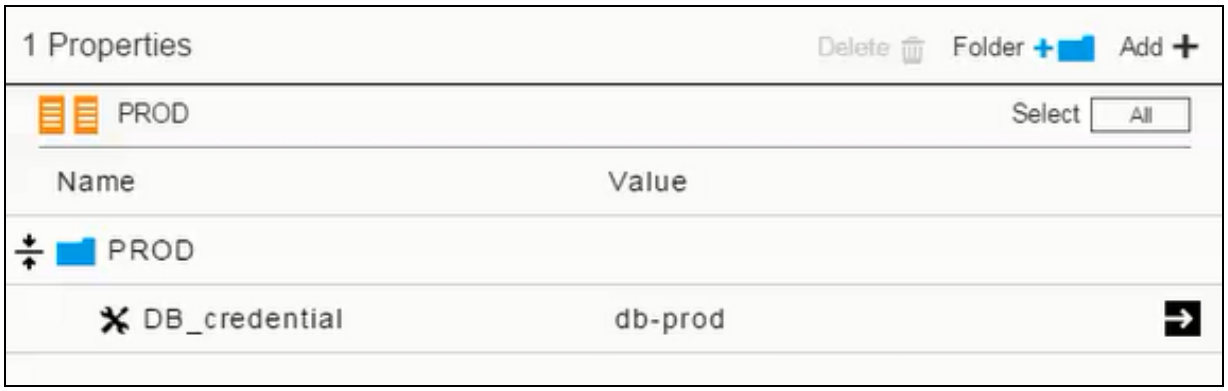
- DEV
- QA
- PROD



Then create the appropriate database (DB) credentials in the Banking project for each database used with these stages. Each credential is identified by a name:







For each environment, create a property called `DB_credential` and initialize it with its corresponding DB credentials name.

To leverage these properties in an application or component process step, use `$[/myEnvironment/DB_CREDENTIAL]` to identify and access the DB credential information as described in these following knowledge base articles:

- https://helpcenter.electric-cloud.com/hc/en-us/articles/202827083-KBEC-00032-Hiding-an-application-s-password-with-a-credential
- https://helpcenter.electric-cloud.com/hc/en-us/articles/206611313-KBEC-00325-Can-t-access-attached-credential-using-ectool-getFullCredential

Using this credential information, you can connect ElectricFlow to the appropriate database and populate the necessary test data. In this use case, `myEnvironment` refers to the currently running environment. There are similar shortcuts available for other deploy objects, such as `myApplication` or `myEnvironmentTier`. You can extend this concept to store environment- or environment-tier specific attributes such as port numbers or URLs.

Snapshots

In a software release process, the goal is to have a consistent deployment process that can be run often throughout the release life cycle. When the deployments are run in production, there are no unexpected results. Snapshots are used to implement these deployments.

A snapshot captures both the component versions and the process that were used for deployments. These snapshots make the deployments repeatable and stable.

ElectricFlow supports these types of snapshots:

- Environment snapshot—An environment snapshot is created from an environment where the application or microservice is deployed and tested. A snapshot will capture the exact artifact or container versions, process versions, configuration parameters, and so on used for the deployment to ensure repeatability. This is considered the reliable version of a snapshot for reliability and repeatability.
- Application or microservice snapshot—A snapshot is created directly from the application or microservice model and does not depend on any environment deployment. When a snapshot is captured from the application or microservice model, it locks the architecture, components, and process versions. If you specify the "Latest" artifact version as part of the component definition in the snapshot, the latest artifact versions will be used every time it is used for deployment. The main purpose of an application- or microservice-based snapshot is to allow the creation of user-defined versions of the models. The application or microservice snapshot name can be used to indicate a specific user-defined version.

For example, in a software delivery process, the developer checks in code, the components are built automatically by a continuous integration system, and the objects in the resulting build are published to a binary artifact management system. When the full complement of component artifacts is available, a release candidate pipeline is initiated. The first stage is to deploy to a development environment. From this deployment, a snapshot is created and used as the model for subsequent deployments to other environments.

There will be some differences between the development, QA and production environments, such as database connection strings, credentials, and debug-vs-production artifacts. ElectricFlow can manage these differences in the application or microservice model using parameters and properties in templates, only modifying the deployment process flow in a controlled manner.

The process described in the example avoids some of the pitfalls encountered in traditional release scenarios. For example, configuration teams often perform their own builds. This means that the binaries supplied by the CM team may be different than what has already been tested. Release teams may also manually edit configuration files to adjust for differences in the deployment environments. Both of these practices are prone to error and can negatively impact the quality of the final software release.

In ElectricFlow, you can deploy snapshots for more reliable and repeatable software releases. Here are some of the use cases snapshots can serve:

- You can model and save a version of your software release with specific artifact versions and rerun it later, even if the latest release version of it has changed.
- If you save snapshots during development and test phases, you can ensure that the components that were deployed and tested are the same as those in the release candidate. You can redeploy the snapshot any time.
- You can create and save more than one snapshot for different deployment scenarios.
- You can use an application or microservice snapshot to capture a version of the model. This can be used to define a future or desired state of the application or microservice model as well.
- You can view the snapshots in the Snapshot List. From this list, you can manage all your snapshots, compare two snapshots, or get more information about them.

- Comparing snapshots helps you to deploy releases with reliable and repeatable results during ongoing software release cycles. You can build and test releases using snapshots, and do not have to design new applications or microservices for each release. Comparing snapshots also allows you to compare a future or desired state of the application model to any saved version of the model. See Comparing Snapshots on page 124 and Comparing Results on page 125 for more information.
- You can use snapshots to refine and optimize applications or microservices that fit your deployment scenario and ensure that this version is properly developed, tested, and released.

Example: Creating, Deploying, and Comparing Snapshots

This example shows how to create application, microservice, and environment snapshots, deploy snapshots, and compare them during the software release life cycle.

Creating Application Snapshots

1. In the Applications List, choose an application and click the  button.
2. Select **Snapshot List**.

   The Snapshot List appears.
3. To add a new snapshot, click **There are no Snapshots. Add one +**.

   The New Snapshot dialog box appears.
4. Enter the name for the snapshot that must be unique within the application and enter an optional description of it.

   This name can be used to indicate the user defined version of the model. For example, in the figure below, see *HeatClinic v1.2*.

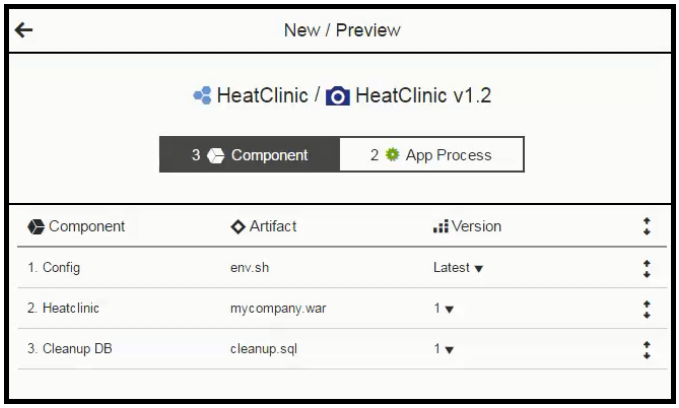
   Application snapshots can be used for various purposes like creating a baseline version of the model or even for modeling a future state or desired state of the model.
5. Click **Next**.

6. The New/Preview dialog box opens.

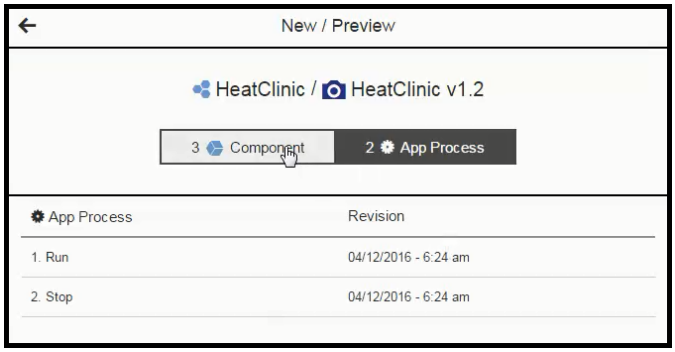
   You can toggle between the **Component** and **App Process** views to see the components and application processes for the selected application.

   Example:

   In the **Component** view, change the version of the Config component to **Latest**.

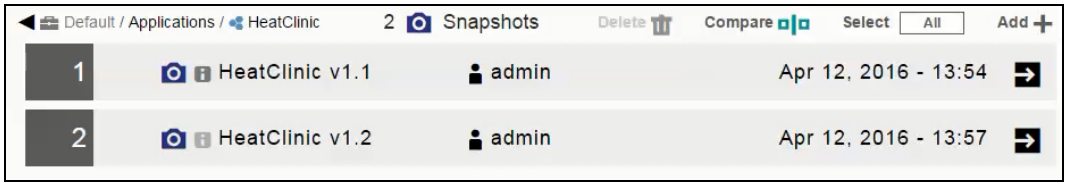


   This is the **App Process** view.



7. Click **OK**.

   A message appears stating that the snapshot has been created, and the snapshot list is updated:



Deploying Application Snapshots

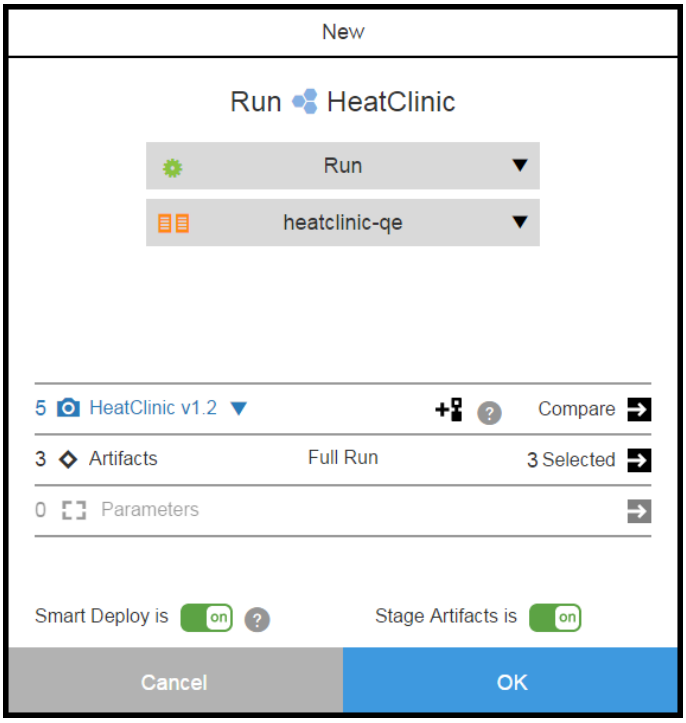
- In the Applications List, choose an application and click the  button (the **Run process** button).
- Select **New Run**.

  The dialog box to select the settings for the application run opens.
- Select these settings for the application run:

  In the **Select Process** field, select **Run**.

  In the **Select Environment** field, select **heatclinic-qe**.

  In the **Select a Snapshot** field, select **HeatClinic v1.2**.

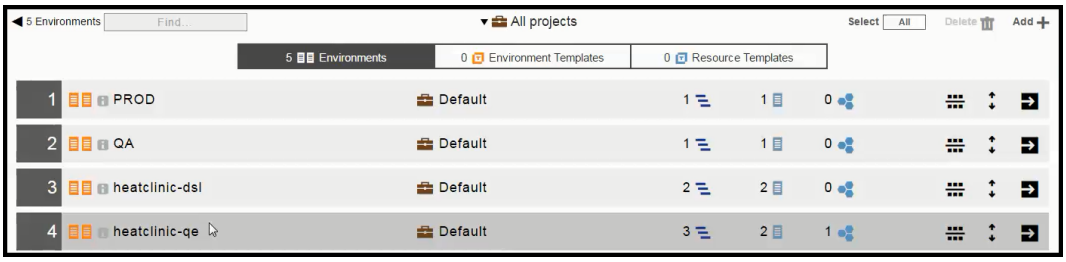


> **Note:** Clicking  next to **Compare** shows the comparison of the selected snapshot to last deployment of the **Run** process in the **heatclinic-qe** environment.

- To deploy the snapshot, click **OK**.

Creating Environment Snapshots

- In the Environments List open an environment inventory.



- Choose an application, and click the  button.
- Click **New Snapshot**.

  The New Snapshot dialog box opens.
- Enter the name and a description of the snapshot.

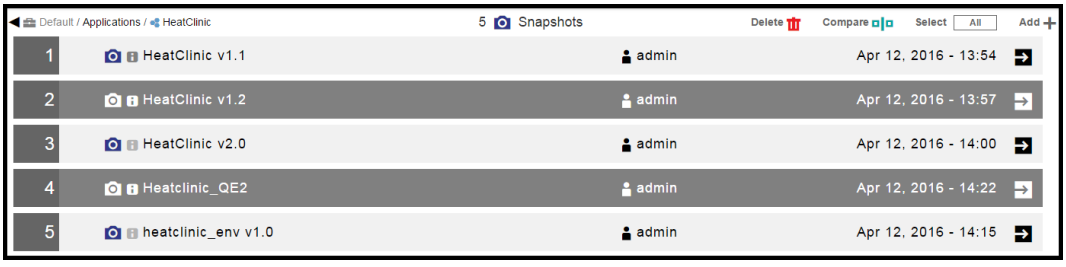
  The name of this environment snapshot is *Heatclinic_QE2*.
- Click **Next**.

  The New/Preview dialog box opens. You can view the snapshot settings in the Component and App Process tabs but cannot change them because the application has been deployed to the heatclinic-qe environment.
- Click **OK**.

Comparing Snapshots

This example shows how to compare an application snapshot to an environment snapshot.

- In the Snapshot List, select two snapshots that you want to compare.



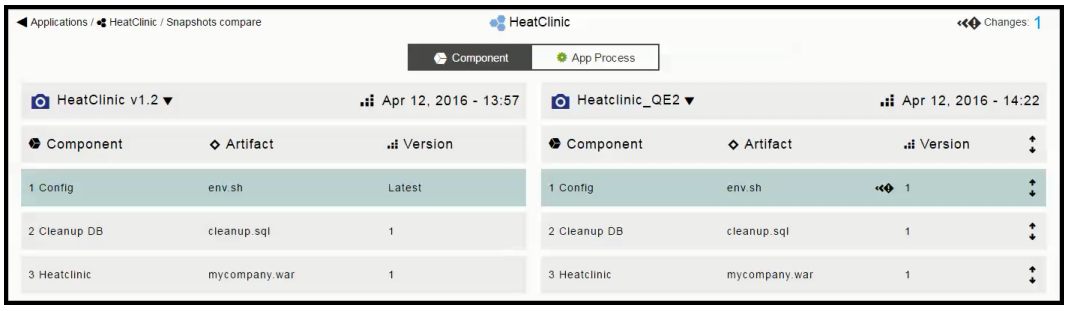
- Click the  button (**Compare** button).

  The Snapshot Comparison page opens.

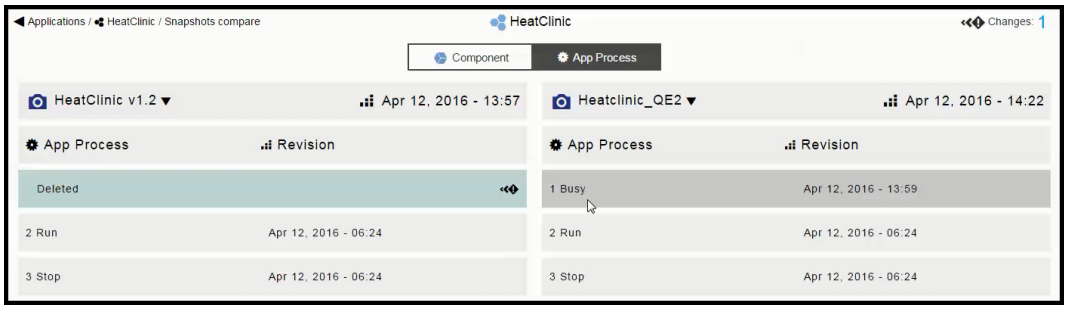
Comparing Results

In the Snapshot Comparison page, the Component and App Process views show the differences between the snapshots, Heatclinic v1.2 (the application snapshot) and Heatclinic_QE2 (the environment snapshot). Heatclinic_QE2 is the more reliable, repeatable, and up-to-date version of the application.

The **Component** view shows that the Config component is different. Heatclinic v1.2 uses the "Latest" version of the artifact that may not be the same as the artifact version in Heatclinic_QE2, the more reliable and repeatable version of the snapshots.



The **App Process** shows one difference. Heatclinic v1.2 has one less application process than Heatclinic_QE2.



> **Note:** Clicking the pull-down button next to the snapshot name allows you to select a different snapshot.

Application Deployment Options

ElectricFlow supports the following out-of-the-box options to deploy applications:

- When authoring processes or procedures:
    - When defining a step for an application or component process, you must select the actions if a step fails:

      On Error: Select **Stop running** to stop the process or **Continue running** to go to the next step in the process. The default is **Continue running**.

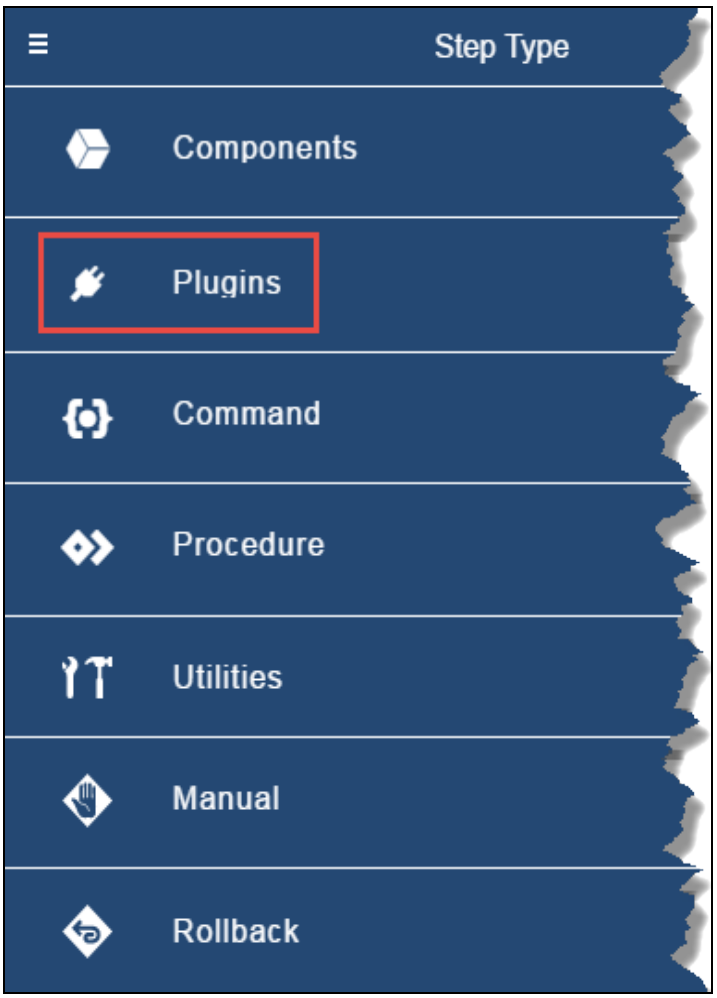
      Run if: When there are conditions to be met before going to the next step, select one of the following options. The default is **all**.

        - **all**: All conditions must be met before the step proceeds.
        - **any**: At least one condition must be met for the step to proceed.

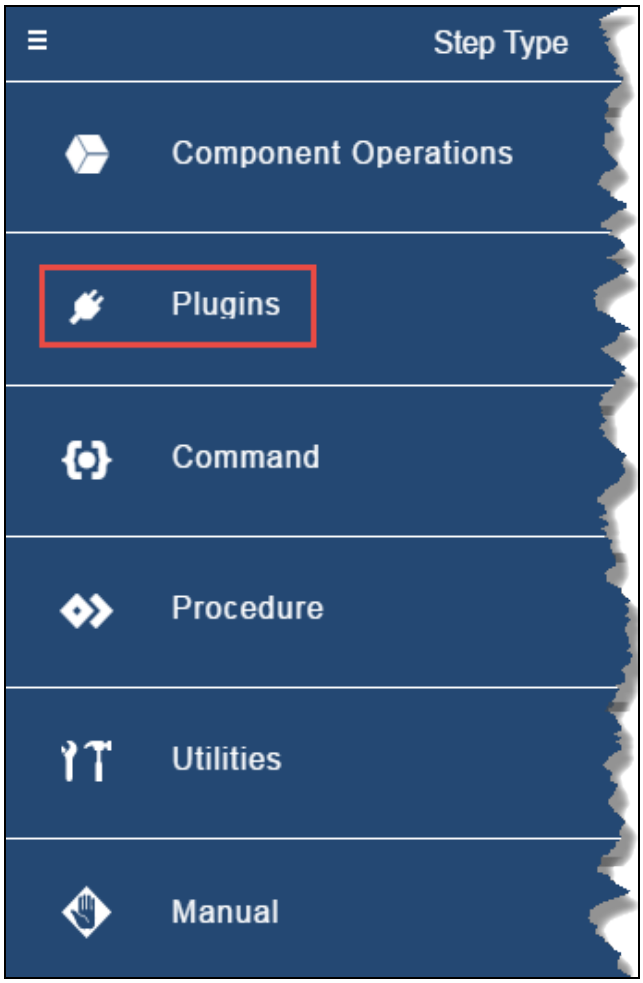
      You can also attach parameters that affect how the step is executed at runtime. Depending on the type of step that you selected, different sets of parameters or fields are displayed.

      See Examples: Modeling and Deploying Applications on page 255 for examples of how to do define component and application process steps.

- When defining a step for an application process, you can select one of these options:
  - **Components**–Use this to run a component process when the step is executed.
  - **Plugins**–Use this to run a third-party plugin.
  - **Command**–Use this to run one or more commands or a script.
  - **Procedure**–Use this to run a set of best practices, subroutines, modules, or functions that you can create and reuse at the platform level.
  - **Utilities**–Use this to run a higher-order operation than a third-party plugin for application modeling.
  - **Manual**–Use this when a manual step is to be performed by a designated assignee.
  - **Rollback**–Use this to restore the deployment model to a previous state before a deployment failed.



- When defining a step for a component process, you can select one of these options:
  - **Component Operations**–Use this to run a component operation.
  - **Plugins**–Use this to run a third-party plugin.
  - **Command**–Use this to run one or more commands or a script.
  - **Procedure**–Use this to run a set of best practices, subroutines, modules, or functions that you can create and reuse at the platform level.
  - **Utilities**–Use this to run a higher-order operation than a third-party plugin for application modeling.
  - **Manual**–Use this when a manual step is to be performed by a designated assignee.



- You can use *process branching* to specify the path through an application or component process based on transition conditions other than out-of- the-box options. Decisions about the next step in the flow are made while the process runs. This is similar to the transition conditions for workflows in the automation platform (see Workflow Overview on page 1570 and Workflow Details on page 1292).

  See Process Branching on page 112 for more information about process branching states and conditions and how to set them in a process.

- When defining a step for a procedure, you must select the **Error handling** action if the step fails.

  You can set the parameters for subprocedure steps.

  You can also set run conditions and preconditions for command and subprocedure steps.

  See Step—create new or edit existing step on page 1171on how to set these options.

- When modeling environments:

  Environment tiers in the environment model represent groups of targets defined by their purpose, such as web servers, databases, and artifact repositories. For example, an environment may have environment tiers for the different types of web servers called "Apache," "Tomcat," and "MS SQL", which contain all the files for your company's website in one or more locations. The resources in the Apache environment tier are the Apache servers in each location.

  Environment Reservation and Calendaring allows you to manage and reduce deployment conflicts and risk when deploying applications, pipelines, or releases.You can schedule blackout periods for planned maintenance work and reserve environments to ensure that resources are available when you need them. The Calendar view makes it easy to visualize planned deployments and conflicts.

- When staging artifacts before the runtime:

  Artifact staging retrieves artifacts that will be deployed before the deployment run starts. This ensures that all the artifacts are downloaded and available during runtime. It also reduces the application runtime. Artifact staging is enabled by default. See Artifact Staging on page 99 for more information.

- At runtime, select one of the following ways to deploy the application. See Examples: Modeling and Deploying Applications on page 255 for deployment examples.
  - Smart deploy

    This deployment method reduces risk and time by only deploying changed objects. ElectricFlow automatically detects the differences between what is about to be deployed and what has already deployed to an environment. It deploys only those objects not yet deployed to a specific resource or to resources where the object version is different. You can use this method throughout the software release life cycle.
  - Full run

    The system deploys the application with all the application processes, components, and artifacts.
  - Partial run

    The system deploys the application with only the selected application processes, components, and artifacts.
  - Selecting artifacts with specific versions to deploy

    The system runs the application with only the selected versions of the artifacts.
  - Snapshot

    The system deploys a snapshot of the application. Also See Example: Creating, Deploying, and Comparing Snapshots on page 121 for an example using snapshots.
  - A combination of these ways:

	Smart deploy	Full run	Partial run	Artifacts with specific versions	Snapshot
Smart deploy		No	Yes	Yes	Yes
Full run	No		No	Yes	Yes
Partial run	Yes	No		Yes	Yes
Artifacts with specific versions	Yes	Yes	Yes		Yes
Snapshot	Yes	Yes	Yes	Yes	

- Artifact Staging

  ElectricFlow retrieves the artifacts that will be deployed in an application run before the deployment starts. This ensures that all the artifacts are downloaded and available on the target. It also minimizes the time it takes to retrieve artifacts and reduces downtime during the deployment. See Artifact Staging on page 99 for more information.

- When an action fails in the process

  When an application deployment fails, you can use *rollback* to restore the deployment to a previous state before the failure occurred. This operation is supported only in application processes in the deployment model. See Rollback on page 100 for more information.

Inventory Tracking

Electric Flow uses Inventory Tracking to track what is built, tested, and deployed by the application, including artifacts, artifact versions, resources on which the applications are run, and environments to which the resources are assigned. If there is an issue when an application is deployed you can find the details about what was deployed.

- Use the Environment Inventory to track and view details of the objects that were deployed and artifacts in the application. It shows the status of the application deployment at a point in time. See Examples: Modeling and Deploying Applications on page 255 and Environment Inventory on page 131 for more information.
- Use the Application Inventory (on the Applications Run View page) to track and view the deployment results. It shows more details about the application at a point in time. Go to the Applications Run View page in the GUI to see these details. Go to Example: Modeling and Deploying Applications for an example.

Tracking at the Component Process Level

Inventory Tracking occurs at the component process level.

When defining a component process, you select one of these process types:

- **Deploy**–Enables Inventory Tracking. The ElectricFlow server tracks artifacts deployed to environments. This is the default.
- **Undeploy**–After the first successful job step in a component process with this setting, the automation platform removes the environment inventory record.
- **Other**–Disables Inventory Tracking.

Environment Inventory

The Environment Inventory is the state of the environment at a point in time.

- When an application is running, you can see its progress as it runs.
- After an application runs, you can see the details for the objects in application.

Viewing an Environment Inventory

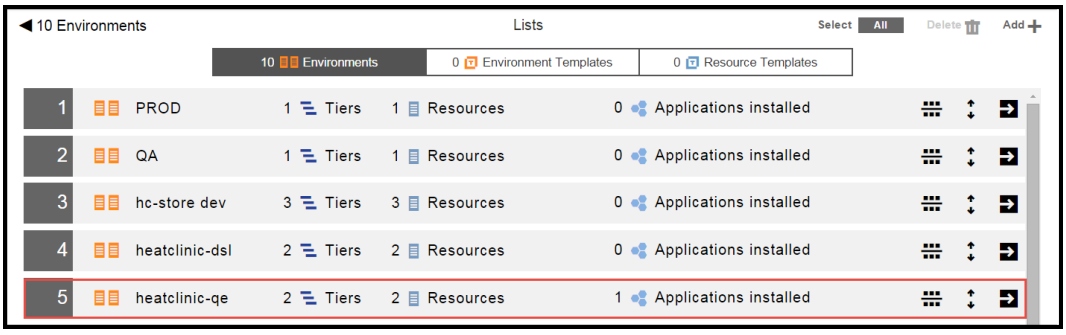
This example shows the Environment Inventory for the *heatclinic-qe* environment to which the HeatClinic application is deployed. The application has two application processes called Run and Stop.

In the Environments List, choose an environment and click the **Inventory** button to open the Environment Inventory.

Clicking in the row for an object in the Environment Inventory displays more details about that object, such as:

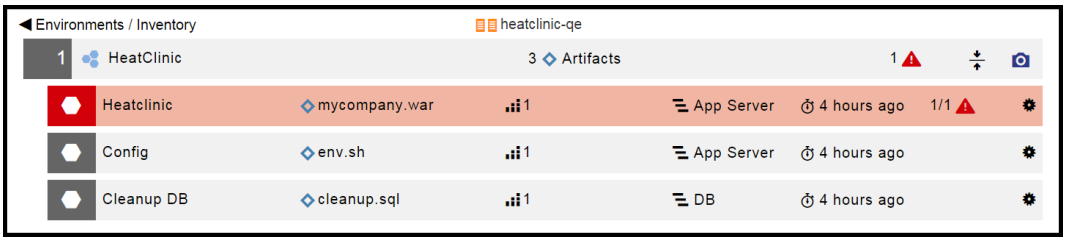
- Environment name
- Application mapped to this environment
- Number of deployed artifacts in the application
- When the artifacts were deployed
- Status of the deployment: success or failure

- Go to the heatclinic-qe row, and click to open the Environment Inventory.



This is the Environment Inventory.

- Click  to view the list of components in the Run process of the HeatClinic application.

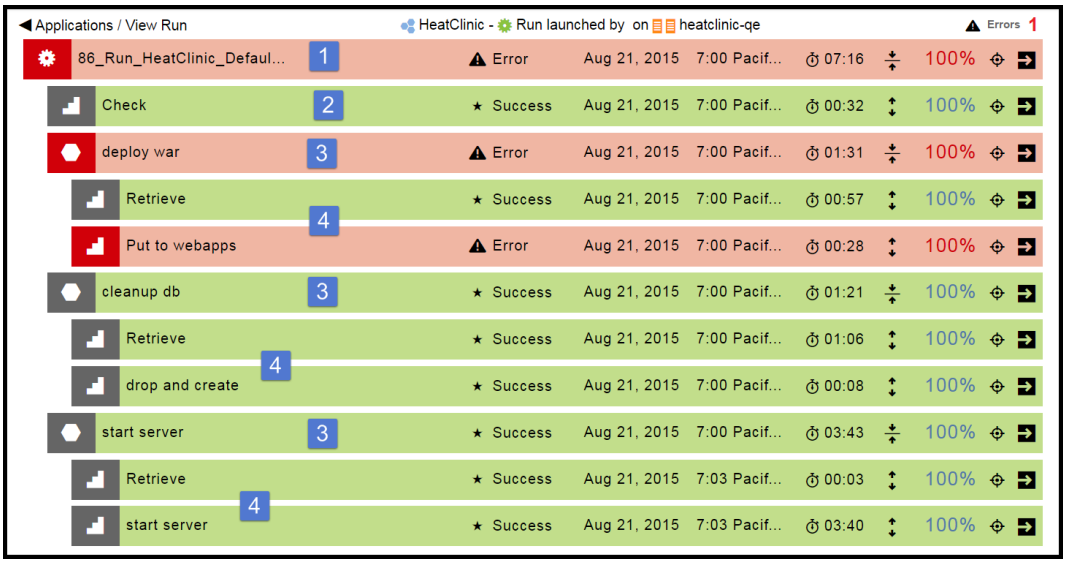


- Click the **Process** button in any of the component rows to view the Run application process runs in the heatclinic-qe environment.

  The results of running the Run application process, application process steps, and component process steps now appear in the inventory. An application process step can be based on a component (defined by component processes), a plugin, a command, or a procedure.

  The Run application process consists of four process steps: Check, deploy war, cleanup db, and start server.

  - The Check step is defined by a command.
  - The remaining steps are defined by components in the HeatClinic application.

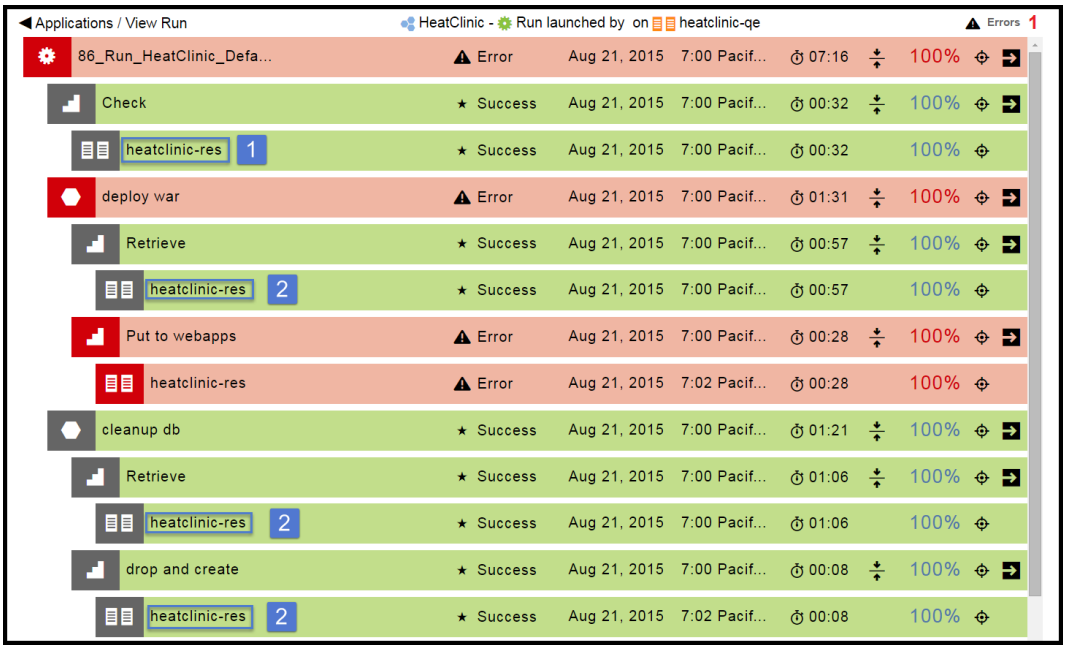


	Object
1	Application process
2	Application process step defined by a command
3	Application process step defined by a component
4	Component process step

- To open the Job Details page for an application process, an application process step, or component process step, do one of the following:

  - Click the object name to open the Job Details page for that object.
  - Click the **View details** button at the end of the row.

- For a process step, click  to view the results of deploying process steps.

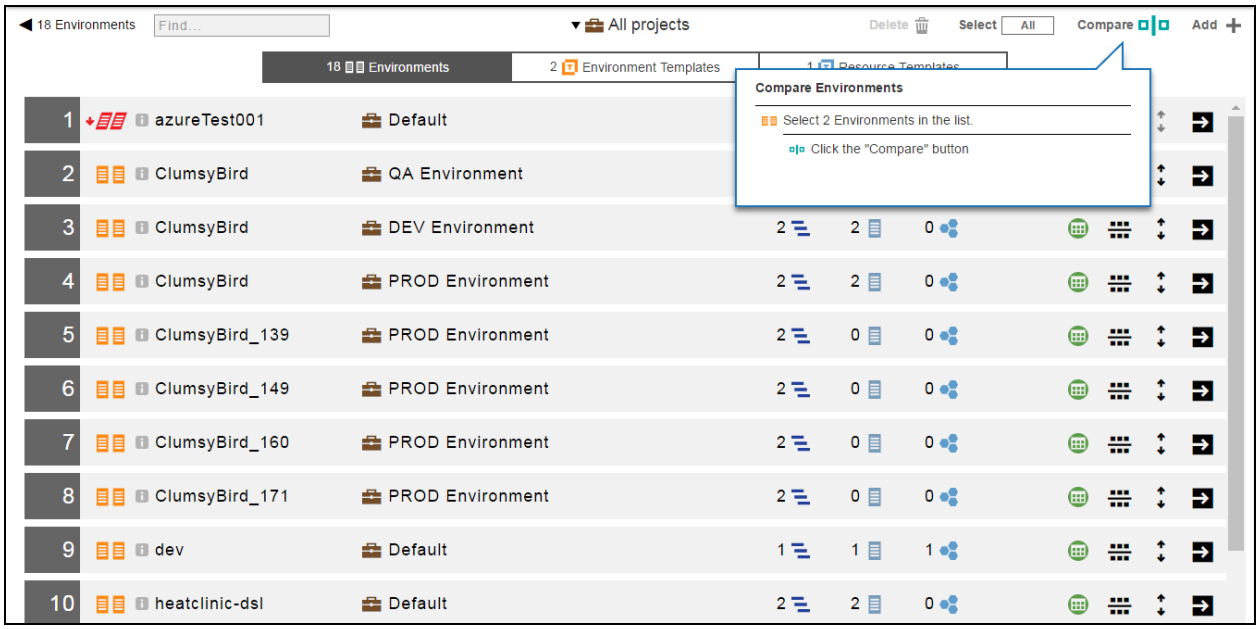


	Object
1	Click here to view the results of running the application process step in the Job Details page.
2	Click here to view the results of running the component process step in the Job Details page.

Comparing Environment Inventories

This feature lets you directly compare inventories of two environments (such as a pre-production environment to a production environment). When you conduct troubleshooting or auditing, this feature lets you easily determine the drift (how one environment's inventory differs from another). This feature eliminates the need to create custom reports or go to one environment's inventory, note all the artifact versions, then go to another environment's inventory and visually compare.

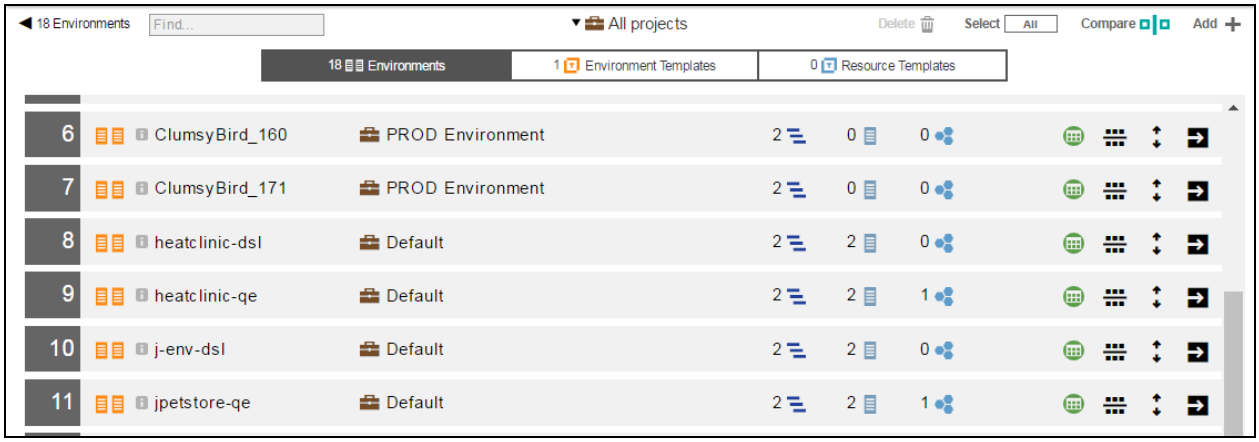
For example, if your pre-production environment is working but your production environment is not, you can determine the drift and quickly make the production environment match the pre-production environment.



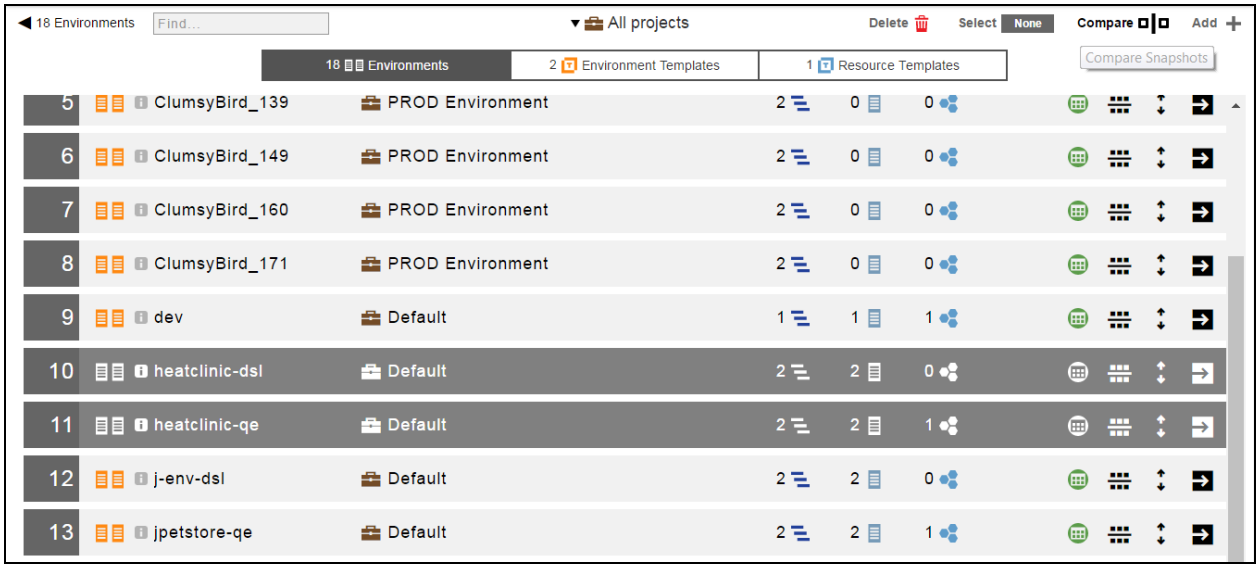
To compare two environment inventories, complete the following steps.

1. Open the Environments List.

   The applications column of the **Environments** screen shows the number of applications that is installed or deployed onto each environment. For example, the heatclinic-dsl environment and the heatclinic-qe environment have zero and 1 applications respectively:

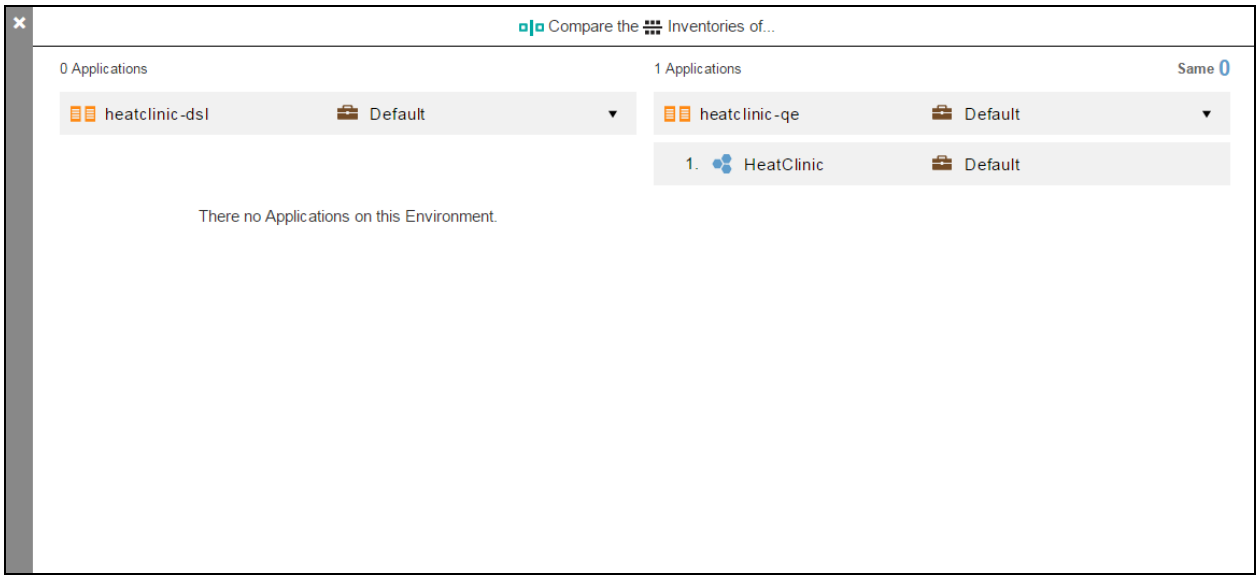


2. Click to highlight two environments as follows:



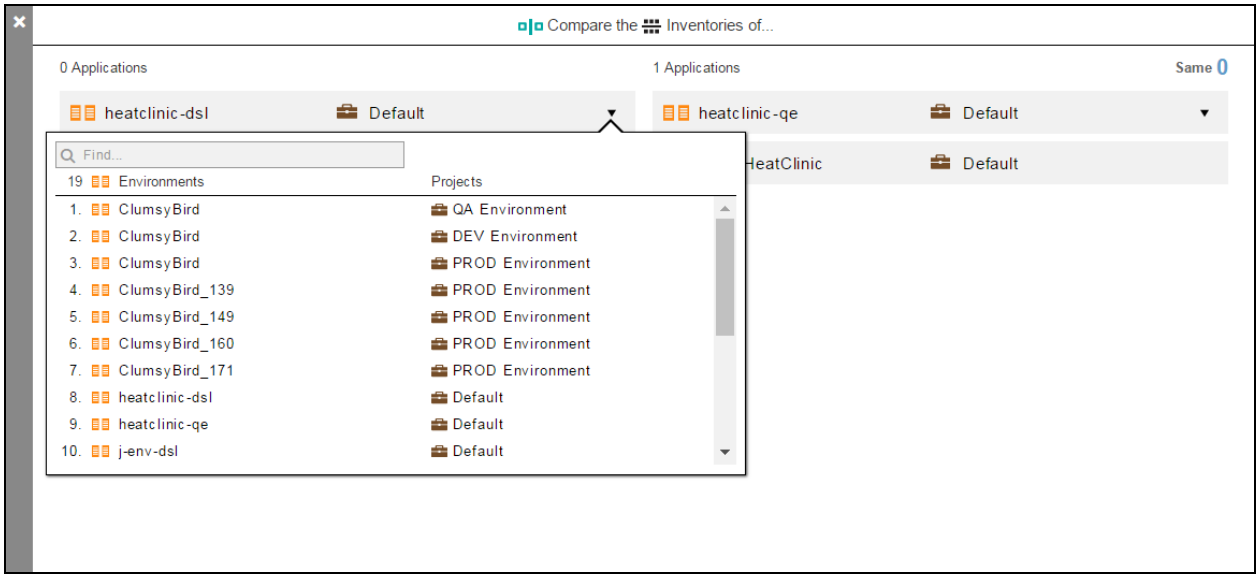
3. Click the Compare (Compare) button.

The **Compare the Inventories of...** screen appears. The screen displays the inventories of both environments side by side. The following example shows that one of the inventories contains no applications:

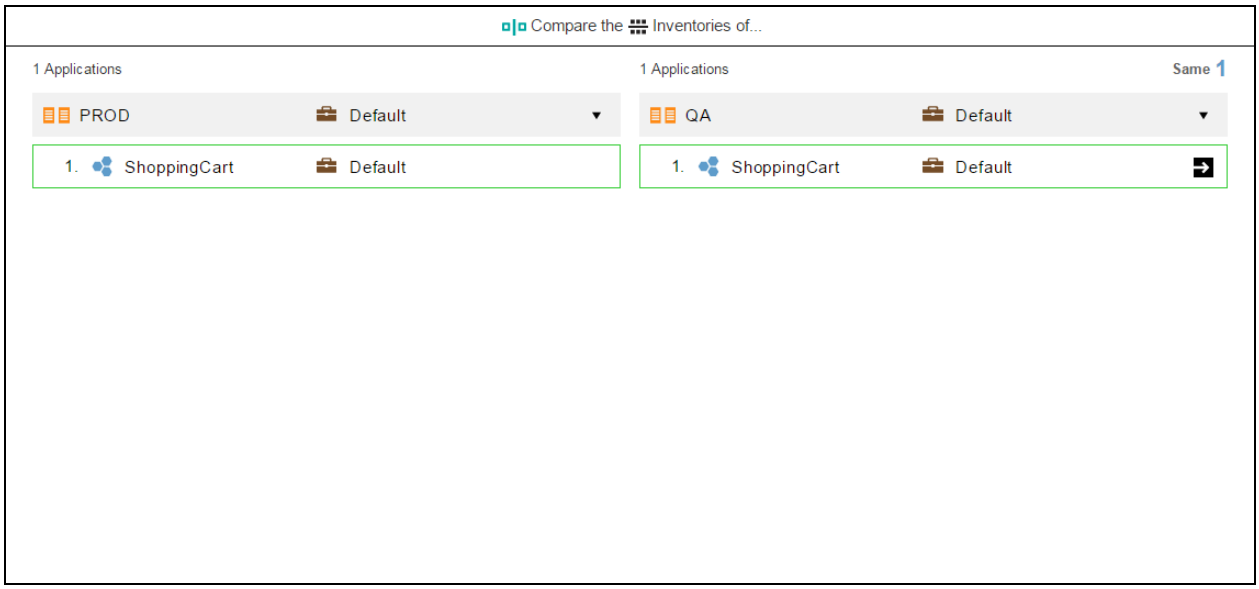


The screen displays the number of applications with the same name across both inventories as well as the number of applications in each inventory.

4. Click the environment selection menu ( )in either column to select an environment for comparison. For example:



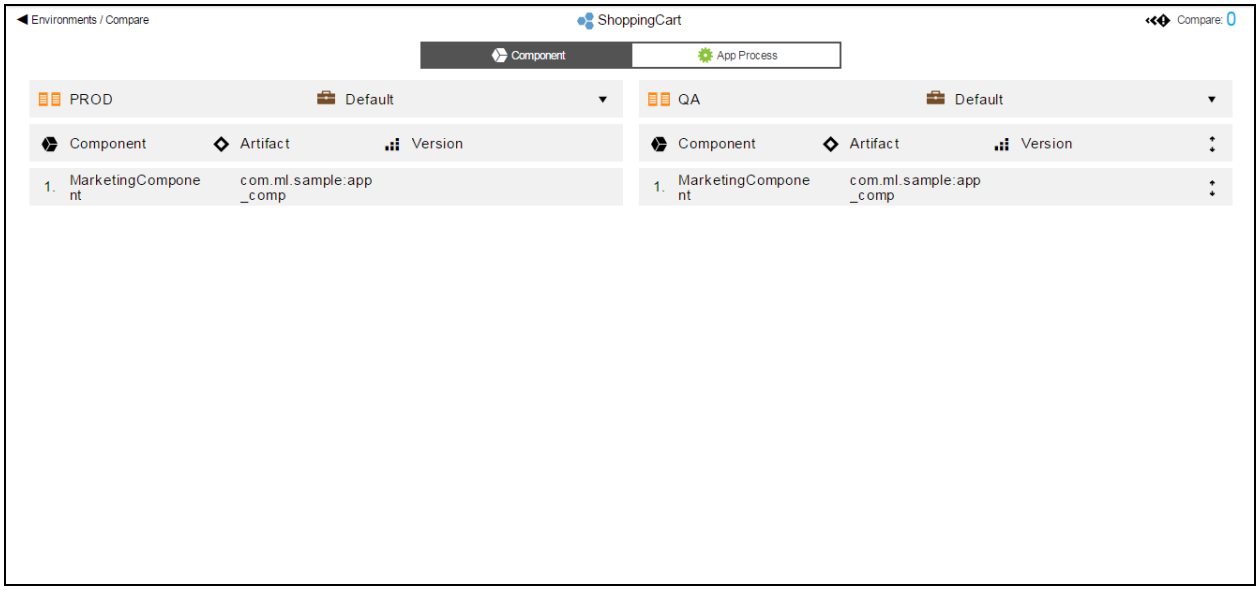
The following screen shows two different environments that contain the same application:



Note that if two instances of an application of the same name appear in both environments, the Details button ( ) appears, which lets you drill down into the applications to compare them.

5. Click the  button.

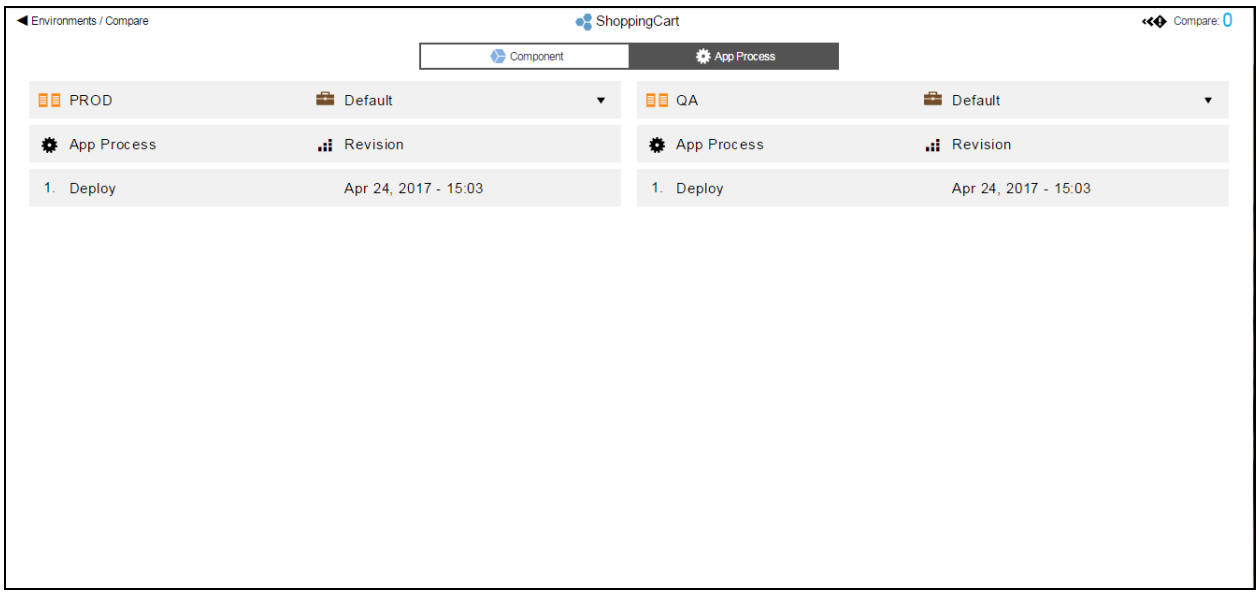
The **Component** comparison screen appears:



You can click the expand () button to expand the row and see the details.

6. Click the **App Process** button.

   The **App Process** comparison screen appears:



   Note that this screen shows the last date of deployment for the applications.

Microservice Deployment Using Containers

ElectricFlow brings the application release automation practices of modeling, automation, and coordination to containers and microservices. This model-based solution avoids cloud provider or container lock-in by separating application models from environment models that are platform specific.

Container and Microservice Concepts

ElectricFlow lets enterprises safely deploy microservices and containers at scale in large, enterprise environments. DevOps teams can streamline and simplify processes to get containerized applications up and running, easily promote applications across different container environments, and gain visibility into application pipelines—both traditional and microservice/container-based—through a single pane of governance. Microservices architecture and container utilization provide the speed, flexibility, predictability to let teams build functionality quickly and deliver it instantly across multiple clouds and environments.

Containers and microservices in complex, large-scale, hybrid environments present operations, governance, and orchestration challenges. ElectricFlow lets teams coordinate software releases when deploying containerized workloads or microservices throughout the software delivery lifecycle. ElectricFlow lets teams coordinate and manage three types of releases:

- Large, monolithic application releases
- Microservices application releases that are backed by containers
- Hybrid monolithic and container-backed microservice application releases

ElectricFlow supports legacy or traditional monolithic applications, hybrid data centers and clouds, and large-scale infrastructure.

ElectricFlow lets developer, QA, and operations teams deploy microservices to their preferred container infrastructure or cluster manager solution such as Google Container Engine (GCE), Amazon EC2 Container Service (ECS), or Kubernetes without knowledge of the underlying integrations. This lets them coordinate and manage a mix of monolithic application and microservice releases across legacy or continuous delivery pipelines in both on-premises and cloud environments. Also, the model-based approach is agnostic of container environments or cluster orchestration solutions and enables easy portability among providers.

Implementing Containers and Microservices for Use with Your Runtime Environment

ElectricFlow containers and microservices are " container runtime-agnostic." This means that you use ElectricFlow to set them up independent of any runtime environment details and then deploy them to the runtime environment of your choice. ElectricFlow supports the GCE and ECS runtime environments.

ElectricFlow supports three types of applications: monolithic applications, microservices applications backed by containers, and hybrid applications, which are a combination of the first two. You can create all three types independent of where they will be deployed.

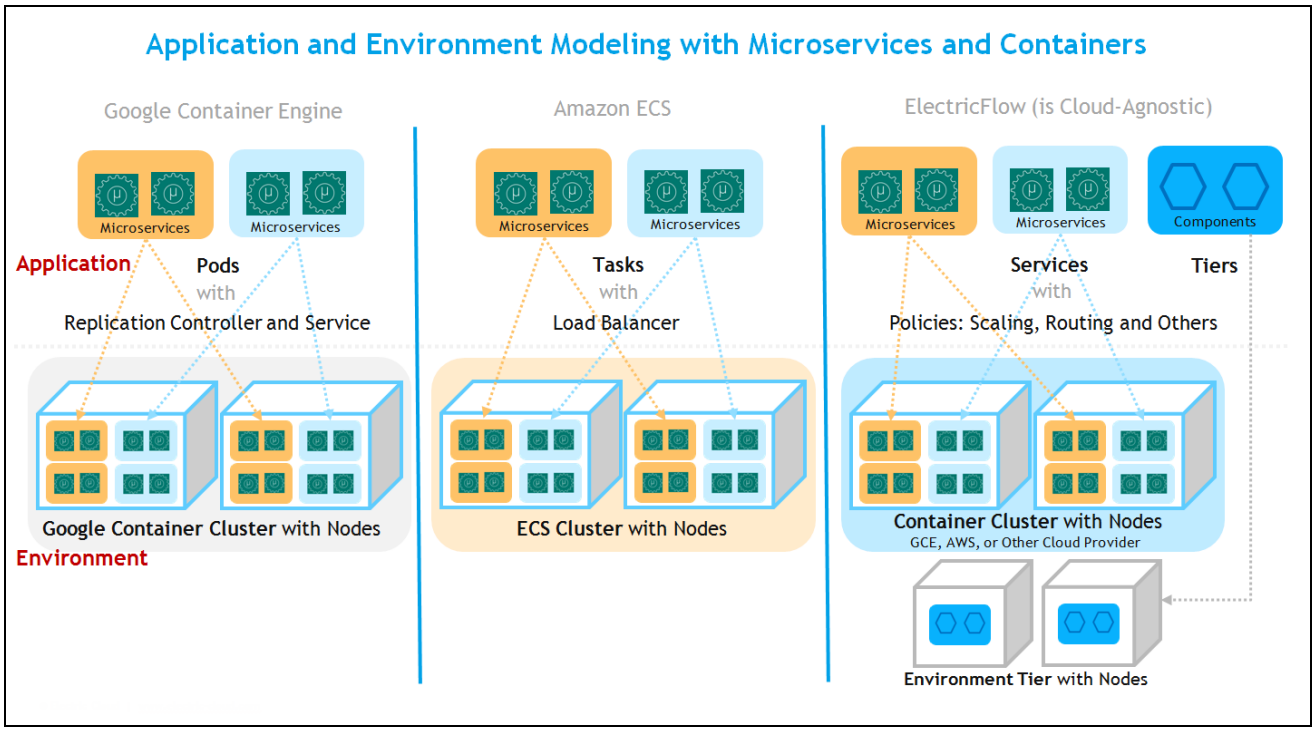
- Monolithic applications—These are typically a set of applications that point to traditional build artifacts such as .war files, .jar files, database scripts, and database components, none of which are not backed by containers.
- Microservices applications backed by containers—A microservices-based containerized application is one that is composed of smaller, focused services, each of which is in a container. Microservices let you scale your development teams and enable higher deployment frequency. Although you can take advantage of the benefits of a microservices architecture without containers, using containers lets you remove configuration challenges across environments by creating lightweight, standalone, deployable packages that contain everything that your software requires.
- Hybrid applications—A hybrid application is a combination of monolithic, traditional components such as . war files, .jar files, and databases and other components that are backed by containers. Using hybrid applications lets you re-architect monolithic applications into microservice applications backed by containers in phases rather than all at once.

Comparison of Modeling Among GCE, ECS, and ElectricFlow

The following figure compares how containers are modeled and deployed in ECS, GCE, and ElectricFlow. In ECS, tasks are used to represent a microservice that is made up of one or more containers. The same microservice in GCE is modeled as pods. These tasks and pods can then be deployed to appropriate cluster nodes that were created in their respective platforms.

With ElectricFlow, each microservice is modeled as an independent microservice or an application microservice. Each microservice is made up of one or more containers.

In addition, if you are using application microservices, you can use application tiers that represent monolithic application components to support a hybrid model. This makes the application model agnostic to the deployment platform. Then you create environments using the clusters and optionally the tiers that correspond to each platform that you want to support. And through the application tier mapping, you associate the application model with the environment model for deployment. You can create multiple tier mappings that correspond to each of the platforms to which you want to deploy.



The following table shows a comparison of the key terms used by the container management service supported by ElectricFlow and the equivalent terms used in the ElectricFlow user interface.

Amazon EC2 Container Service (ECS)	Google Container Engine (GCE)	ElectricFlow
Application	Application	Application
Cluster	Cluster	Cluster
Task definition	Pod	Microservice/container
Load balancer	Service/replication controller	Container ports and service port

Implementing Containers and Microservices in ElectricFlow

Using Independent Microservices

Following is the high-level process for setting up containers and microservices using ElectricFlow and then integrating them with the runtime environment of your choice:

1. **Create independent microservices.**

   You complete this step independent of the environment(s) that you will use. Each microservice can have one or more containers.

In this step, you group the containers into a cluster. Also, in this step, you enter the metadata for each container to specify the location of the container image, the CPU requirement, the scaling criteria, the memory requirement, and so on.

At the time of deployment, ElectricFlow ensures that the microservice calls the correct APIs.

2. **Create one or more environment models.**

   You complete this step independent of the independent microservice that you will deploy. In each environment, you create a cluster of nodes, each of which has a set of containers that hold microservices. You enter the metadata to point each environment to a specific provider (a runtime engine such as GCE or ECS). You can create an environment in a hybrid manner in which some pieces are clustered and others are traditional physical resources or static VMs.

3. **Map the microservice model to one or more environment models.**

   This lets you deploy the same microservice to any container runtime platform of your choice.

Using Application Microservices

Following is the high-level process for setting up containers and microservices using ElectricFlow and then integrating them with the runtime environment of your choice:

1. **Create an application model that is composed of microservices.**

   You complete this step independent of the environment(s) that you will use. Each microservice can have one or more containers.

   In this step, you group the containers into a cluster. Also, in this step, you enter the metadata for each container to specify the location of the container image, the CPU requirement, the scaling criteria, the memory requirement, and so on.

   At the time of deployment, ElectricFlow ensures that the application calls the correct APIs.

   You can create an application in a hybrid manner in which some pieces point to traditional build artifacts such as .war files, .jar files, database scripts, and database components and others are containerized microservices.

2. **Create one or more environment models.**

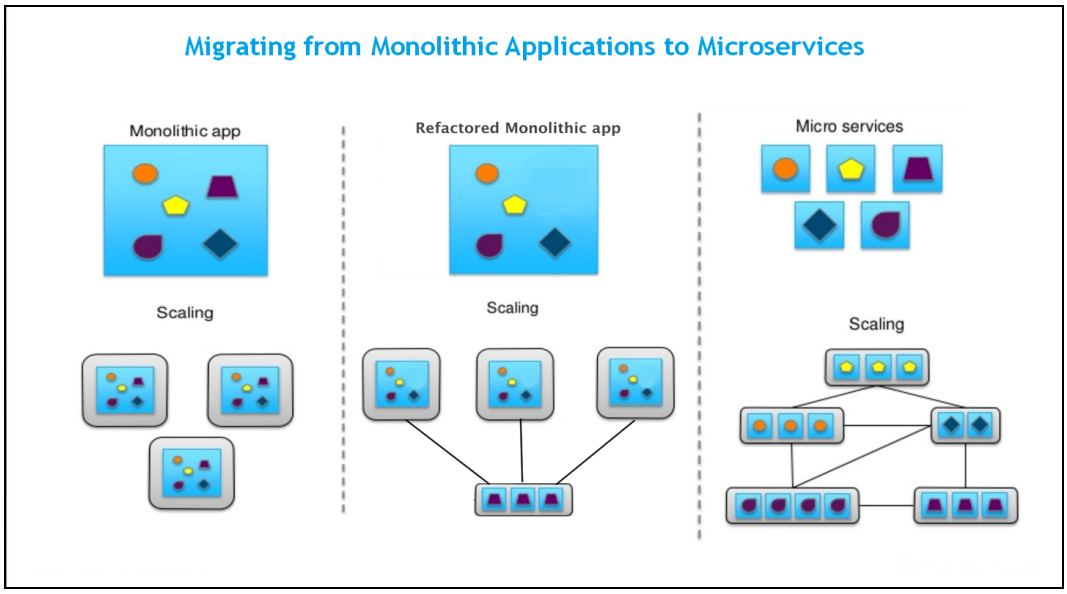
   You complete this step independent of the application that you will deploy. In each environment, you create a cluster of nodes, each of which has a set of containers that hold microservices. You enter the metadata to point each environment to a specific provider (a runtime engine such as GCE or ECS). You can create an environment in a hybrid manner in which some pieces are clustered and others are traditional physical resources or static VMs.

3. **Map the application model to one or more environment models.**

   This lets you deploy the same application to any container runtime platform of your choice.

Migrating from Monolithic Applications to Microservices

The following figure shows how ElectricFlow supports a hybrid model, which means that you do not need to convert an entire monolithic application to containers all at once. That is, you can continue to deploy part of the application, and you can convert other parts to containers and deploy them alongside the remaining monolithic functionality using the same process. This lets you adopt containers over time without converting the entire application at once (without losing the overall visibility of the entire application definition/model). This also lets you test parts of the application to lower risk.



Volume Support

To be able to save (persist) data and share data between containers, Docker uses volumes. Volumes are directories (or files) that are outside of the default Union File System and exist as normal directories and files on the host file system. ElectricFlow lets you specify the volume pointing to a storage device or a partition that can be mounted by a container.

Private and Public Registries and the Use of Credentials

Container infrastructure	Docker public registry	Docker or Docker certified private registry	Google Container Registry (GCR)	Amazon EC2 Container Registry (ECR)
Amazon ECS	Supported natively	Requires preconfiguration of images (VMs)	Requires preconfiguration of images (VMs)	Requires preconfiguration of images (VMs)
Google Container Engine	Supported natively	Supported via secrets. (Uses registry URL/credential)	Supported via secrets. (Uses registry URL/credential)	Supported via secrets. (Uses registry URL/credential)

Object Naming Guidelines

Following are the restrictions in ElectricFlow for naming microservices, containers, clusters, microservice ports, and environment variables. (These restrictions also apply to values that you use with the `newName` argument.)

Restrictions

ElectricFlow has restrictions for naming the following objects: microservices, containers, container and microservice ports, and environment variables. Object names must contain:

- Between 1 and 253 characters
- Only alphanumerics, commas, underscores, and hyphens. Spaces are not allowed.

Kubernetes has length restrictions for fields. ElectricFlow does not limit or truncate the lengths in the ElectricFlow plugins for GCE and ECS, and the API reports an error if the length restrictions are violated.

Because ECS and GCE have their own naming restrictions for microservices and containers, these restrictions let you use your model across those platforms.

The ElectricFlow plugins for ECS and GCE do not use the ElectricFlow cluster name. The platform cluster name is passed in as a procedure-specific parameter and is separate from the Deploy cluster name, so these naming restrictions do not apply.

Conversion of Characters in Object Names by ECS and GCE

- The ElectricFlow plugin for GCE converts spaces and underscores to dashes and converts uppercase characters to lowercase.
- The ElectricFlow plugin for ECS converts spaces to dashes.
- The ElectricFlow plugins for GCE and ECS remove leading and trailing dashes.

Guidelines for Using Clusters

Creating clusters in GCE requires more time than creating clusters in ECS. The Provision Cluster procedure in the GCE and ECS container plugins creates the cluster on the container platform if it does not already exist. Google Container clusters take time to spin up, so the Provision Cluster procedure waits (five minutes by default) for the cluster creation to finish before continuing.

Limitations

- Amazon ECS does not support providing the private registry credentials at the microservice deployment time. The registry credentials must be available on the EC2 container instances when the ECS agent starts. You must ensure that the registry credentials are available on the instance by following the guidelines in the *Private Registry Authentication* page at http://docs.aws.amazon.com/AmazonECS/latest/developerguide/private-auth.html. Deploy users can provide the required information (using these guidelines) through the User Data parameter value provided with the Provision Cluster procedure.
- Rollback of containers is not supported.

Independent or Project-Level Deployable Microservices

You can create microservices that can be independently managed and deployed. You create these microservices at the ElectricFlow project level. This provides you the autonomy for the deployment of the microservices for which you and your team are responsible by allowing you to create microservices under an ElectricFlow project without creating applications. This means that separate teams can develop microservices without being involved in the applications.

Similarities Between Independent Microservices and Applications

Independent microservices provide capabilities that are very similar to those of applications. For example, you can:

- Create a microservice under a project
- Map a project-level microservice to an environment cluster

- Create and modify a process under the microservice
- Run the microservice process and view its inventory
- Create a snapshot for the microservice—from the definition as well as from an environment where the microservice is deployed
- Define dependencies between microservices or their versions
- Define dependencies between a microservice and an application
- Create schedules for microservice deployment
- Create an environment reservation for a microservice
- Create a pipeline task for microservice deployment
- Add microservices to a release and configure microservices in a release

The following table describes the similarities between independent microservices and applications:

Item or Task	Description
User interface	The independent microservices UI is similar to the applications UI. This means that if you are familiar with using the applications UI to create and manage applications, you can easily learn how to use the microservices UI to create and manage microservices.
Mapping to environments	You can map a microservice to an environment just as you can with an application.
Calling processes from pipelines	Pipeline stage tasks can call microservice processes in addition to application processes.
Process UIs	The UIs for microservice processes and application processes are very similar.
Snapshots	You can take a snapshot of a microservice just as you can with an application (although the dialog boxes differ slightly). A microservice snapshot is similar to an application snapshot. You can take a snapshot from a service definition or from an environment inventory.
Scheduling	Microservice scheduling is similar to application scheduling.
Environment inventory after deployment	As with an application, once you create a microservice and deploy it, you can see it in the environment inventory.
Releases	You add microservices to a release in the same way that you add applications to a release (the lists of applications and microservices appear alongside each other for convenient selection). You can select the process, snapshot runtime settings, error handling settings, and order of deployment. For every service, you can do the following in different stages: select the environment, configure process parameters, and select deployer tasks.

Item or Task	Description
Environment reservation	You can create environment reservations and blackouts for a microservice just as you can with an application.
Runtime settings	These settings are same as for applications, except that Smart Deploy and Stage Artifact are not available.
"On Error" settings	The available settings for microservice error handling are similar to those in applications.
Release dashboard	The DevOps Insight Release dashboard displays microservices as well as applications.
Path to Production view	The Path to Production view displays the list of microservices and applications.
Dependencies	You can mix and match microservice and application dependencies. Configuration for these dependencies uses the same UI.
Hierarchy Menu	The Hierarchy Menu works for microservices as well as for applications (and environments).

Automated Microservice Discovery and Onboarding

From existing container-based microservices, you can automatically create microservice models based on common microservice-definition file formats supported by container platforms such as Docker and Kubernetes. This modeling automation is done via ElectricFlow's discovery and onboarding functionality. With automated discovery and onboarding, you can quickly translate your existing assets to start leveraging ElectricFlow's native support for managing and deploying microservices.

You perform discovery and onboarding via items that are bundled in the ElectricFlow Self-Service Catalog. These items rely on underlying plugins that are bundled with ElectricFlow. When using these catalog items, you enter the details about your Kubernetes cluster or your Kubernetes YAML or Docker Compose files.

- Microservice Discovery Using Kubernetes Discovery on page 148
- Microservice Onboarding by Importing a Kubernetes YAML File on page 155
- Microservice Onboarding by Importing a Docker Compose File on page 160

Microservice Discovery Using Kubernetes Discovery

You can create microservice models automatically in ElectricFlow for the microservices and the pods discovered within a namespace on a Kubernetes cluster. You connect to Kubernetes for discovery by specifying an existing ElectricFlow environment and cluster or by specifying details to connect directly to a Kubernetes endpoint.

Choosing the Discovery Method (Existing ElectricFlow Cluster or Kubernetes Endpoint)

You have two choices:

- Existing ElectricFlow cluster—You specify the ElectricFlow cluster representing the Kubernetes backend cluster to be discovered and its environment and parent project.
- Kubernetes endpoint—You enter the endpoint and microservice account API token to connect directly to the endpoint to discover the clusters and pods. You can enter names to create a new environment and cluster in ElectricFlow based on the discovered microservices and pods.

Choosing Where the Discovered Microservices Will Be Created in ElectricFlow

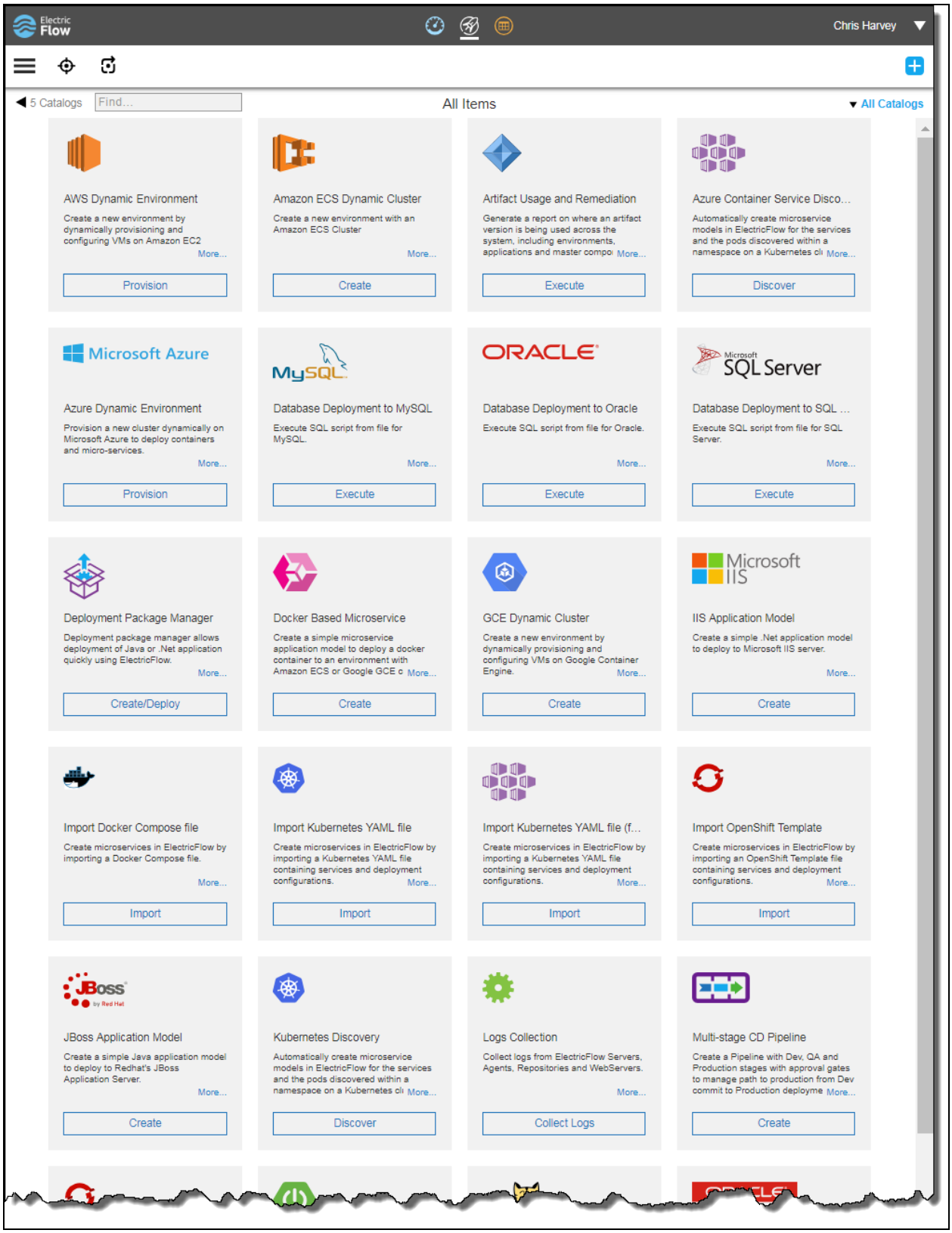
You have two choices:

- Individually at the project top level—You simply enter the project name (as described in the procedure below).
- Within a new application—You enter the project name and application name and also enable ElectricFlow to create microservices within the application.

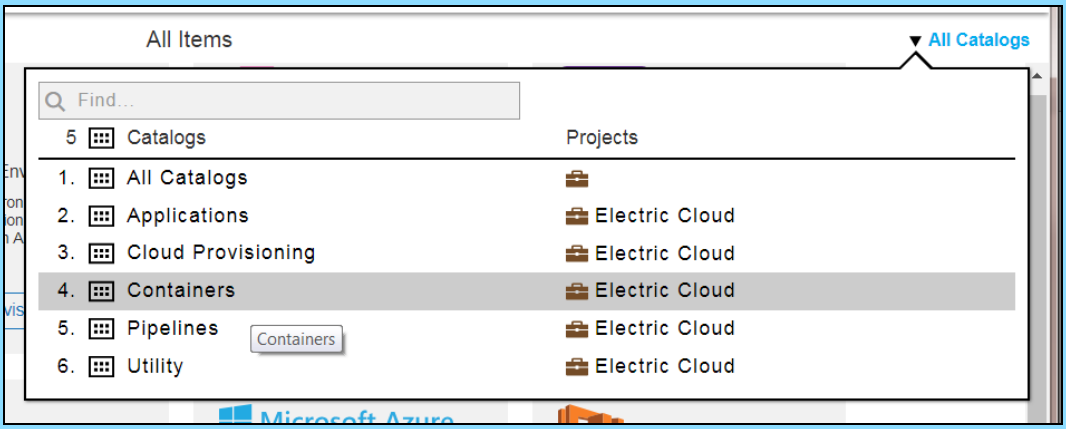
Running a Kubernetes Discovery

1. Click the  (Self Service Catalogs) button in the banner bar.

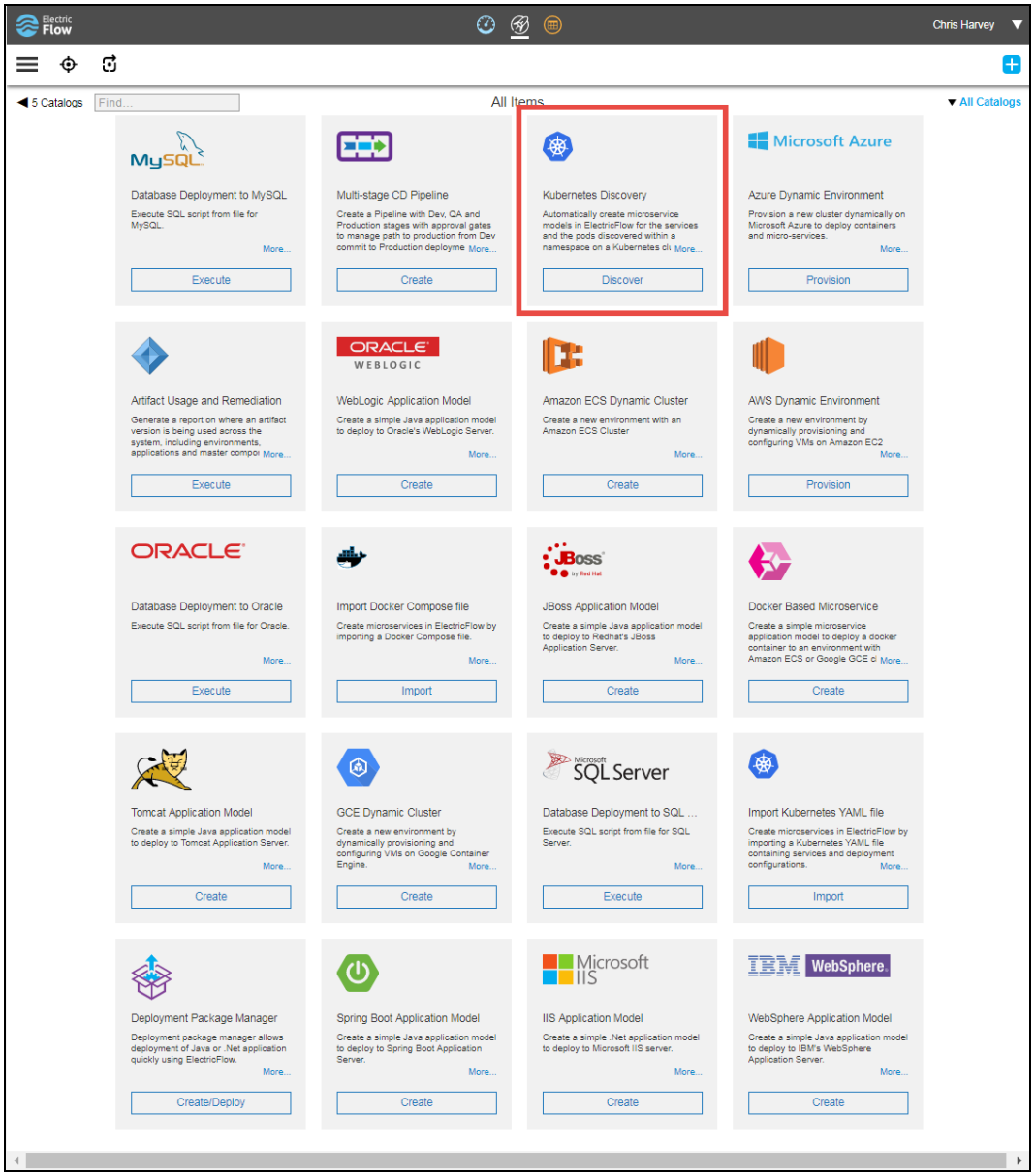
   The Self-Service Catalog opens:


Electric Flow
Chris Harvey
5 Catalogs
Find...
All Items
All Catalogs
AWS Dynamic Environment
Create a new environment by dynamically provisioning and configuring VMs on Amazon EC2
More...
Provision
Amazon ECS Dynamic Cluster
Create a new environment with an Amazon ECS Cluster
More...
Create
Artifact Usage and Remediation
Generate a report on where an artifact version is being used across the system, including environments, applications and master compo
More...
Execute
Azure Container Service Disco...
Automatically create microservice models in ElectricFlow for the services and the pods discovered within a namespace on a Kubernetes cl
More...
Discover
Microsoft Azure
Azure Dynamic Environment
Provision a new cluster dynamically on Microsoft Azure to deploy containers and micro-services.
More...
Provision
MySQL
Database Deployment to MySQL
Execute SQL script from file for MySQL.
More...
Execute
ORACLE
Database Deployment to Oracle
Execute SQL script from file for Oracle.
More...
Execute
Microsoft SQL Server
Database Deployment to SQL ...
Execute SQL script from file for SQL Server.
More...
Execute
Deployment Package Manager
Deployment package manager allows deployment of Java or .Net application quickly using ElectricFlow.
More...
Create/Deploy
Docker Based Microservice
Create a simple microservice application model to deploy a docker container to an environment with Amazon ECS or Google GCE c
More...
Create
GCE Dynamic Cluster
Create a new environment by dynamically provisioning and configuring VMs on Google Container Engine.
More...
Create
Microsoft IIS
IIS Application Model
Create a simple .Net application model to deploy to Microsoft IIS server.
More...
Create
Import Docker Compose file
Create microservices in ElectricFlow by importing a Docker Compose file.
More...
Import
Import Kubernetes YAML file
Create microservices in ElectricFlow by importing a Kubernetes YAML file containing services and deployment configurations.
More...
Import
Import Kubernetes YAML file (f...
Create microservices in ElectricFlow by importing a Kubernetes YAML file containing services and deployment configurations.
More...
Import
Import OpenShift Template
Create microservices in ElectricFlow by importing an OpenShift Template file containing services and deployment configurations.
More...
Import
JBoss by Red Hat
JBoss Application Model
Create a simple Java application model to deploy to Redhat's JBoss Application Server.
More...
Create
Kubernetes Discovery
Automatically create microservice models in ElectricFlow for the services and the pods discovered within a namespace on a Kubernetes cl
More...
Discover
Logs Collection
Collect logs from ElectricFlow Servers, Agents, Repositories and WebServers.
More...
Collect Logs
Multi-stage CD Pipeline
Create a Pipeline with Dev, QA and Production stages with approval gates to manage path to production from Dev commit to Production deployme
More...
Create

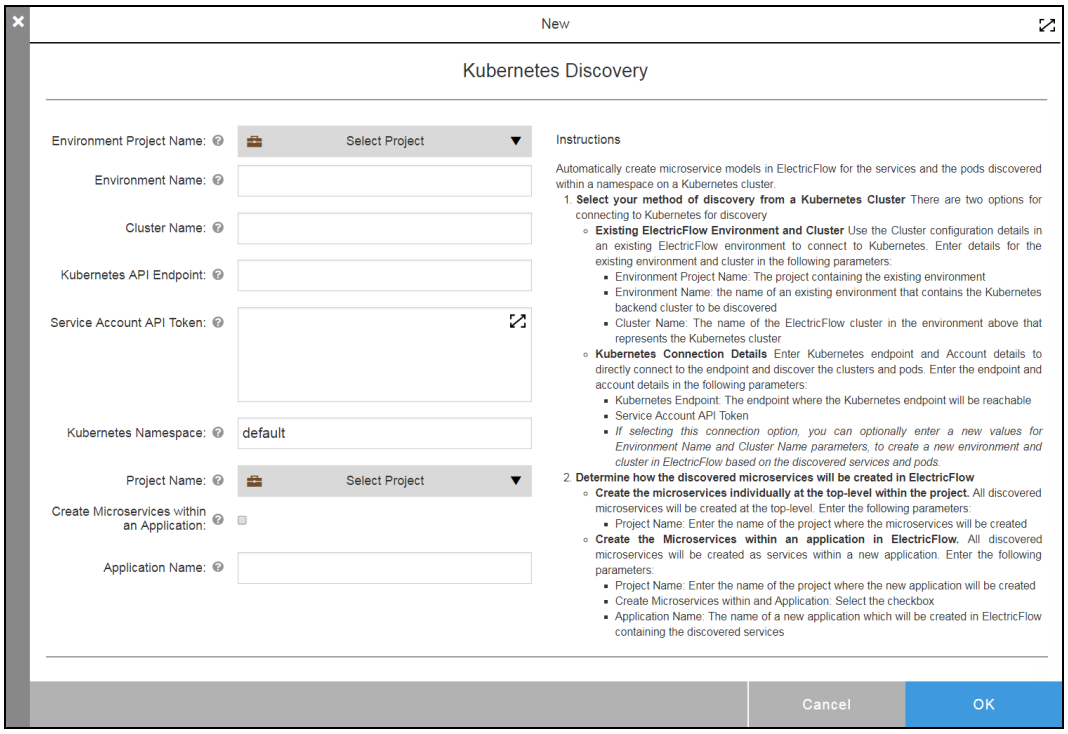
Tip:
This page shows all available catalog items by default. You can narrow down the list by choosing **Containers** from the catalogs menu:



2. Click the **Discover** button on the **Kubernetes Discovery** catalog item:



The **Kubernetes Discovery** dialog box appears:


New
Kubernetes Discovery
Environment Project Name:
Select Project
Environment Name:
Cluster Name:
Kubernetes API Endpoint:
Service Account API Token:
Kubernetes Namespace:
default
Project Name:
Select Project
Create Microservices within an Application:
Application Name:
Instructions
Automatically create microservice models in ElectricFlow for the services and the pods discovered within a namespace on a Kubernetes cluster.
1. Select your method of discovery from a Kubernetes Cluster There are two options for connecting to Kubernetes for discovery
Existing ElectricFlow Environment and Cluster Use the Cluster configuration details in an existing ElectricFlow environment to connect to Kubernetes. Enter details for the existing environment and cluster in the following parameters:
Environment Project Name: The project containing the existing environment
Environment Name: the name of an existing environment that contains the Kubernetes backend cluster to be discovered
Cluster Name: The name of the ElectricFlow cluster in the environment above that represents the Kubernetes cluster
Kubernetes Connection Details Enter Kubernetes endpoint and Account details to directly connect to the endpoint and discover the clusters and pods. Enter the endpoint and account details in the following parameters:
Kubernetes Endpoint: The endpoint where the Kubernetes endpoint will be reachable
Service Account API Token
If selecting this connection option, you can optionally enter a new values for Environment Name and Cluster Name parameters, to create a new environment and cluster in ElectricFlow based on the discovered services and pods.
2. Determine how the discovered microservices will be created in ElectricFlow
Create the microservices individually at the top-level within the project. All discovered microservices will be created at the top-level. Enter the following parameters:
Project Name: Enter the name of the project where the microservices will be created
Create the Microservices within an application in ElectricFlow. All discovered microservices will be created as services within a new application. Enter the following parameters:
Project Name: Enter the name of the project where the new application will be created
Create Microservices within and Application: Select the checkbox
Application Name: The name of a new application which will be created in ElectricFlow containing the discovered services
Cancel
OK

3. Enter the connection details of your existing cluster in Kubernetes as follows:

Field or Option	Description
Environment Project Name	Project containing the existing environment or where the new environment will be created.
Environment Name	Name of an existing environment that contains the Kubernetes backend cluster to be discovered. If the environment does not yet exist, provide the Kubernetes connection details below, and a new environment will be created.
Cluster Name	Name of the ElectricFlow cluster in the environment above that represents the Kubernetes cluster whose deployed microservices are to be discovered. If the environment does not exist, provide the name of the cluster to be created in the new environment.
Kubernetes API Endpoint	Endpoint where the Kubernetes API is reachable. This must be an IP address or a resolvable DNS name. This field is required only if you are not providing an existing environment above for discovery.
Service Account API Token	Bearer token for a service account that has permissions to create resources in the Kubernetes cluster. This field is required only if you are not providing an existing environment above for discovery.
Kubernetes Namespace	Name of the Kubernetes namespace within which the deployed microservices should be discovered. The default is the "default" namespace.
Project Name	Name of the project in which discovered microservices will be created.
Create Microservices within an Application	(Optional) Select this checkbox to create all discovered microservices in the Kubernetes namespace within one application in ElectricFlow. If selected, then you must enter the application name. If unselected, microservices are created at the top level in the project.
Application Name	Name of the new application to contain the microservices. This is required only if **Create Microservices within an Application** is selected.

4. Click **OK** to save your changes.

   ElectricFlow automatically discovers the microservices and creates the associated microservice models and environments in ElectricFlow that are ready to deploy.

Microservice Onboarding by Importing a Kubernetes YAML File

Kubernetes objects are persistent entities in the Kubernetes system. Kubernetes uses these entities to represent the state of your cluster. A microservice in ElectricFlow is represented through two such Kubernetes objects—a Kubernetes service and a deployment configuration.

You can create microservices in ElectricFlow by importing a Kubernetes YAML file containing microservice and deployment configurations. You enter your Kubernetes YAML file detailing your microservice definitions and dependencies to automatically create deployable microservice models in ElectricFlow.

Choosing Where the Microservices Will Be Created in ElectricFlow

You have two choices:

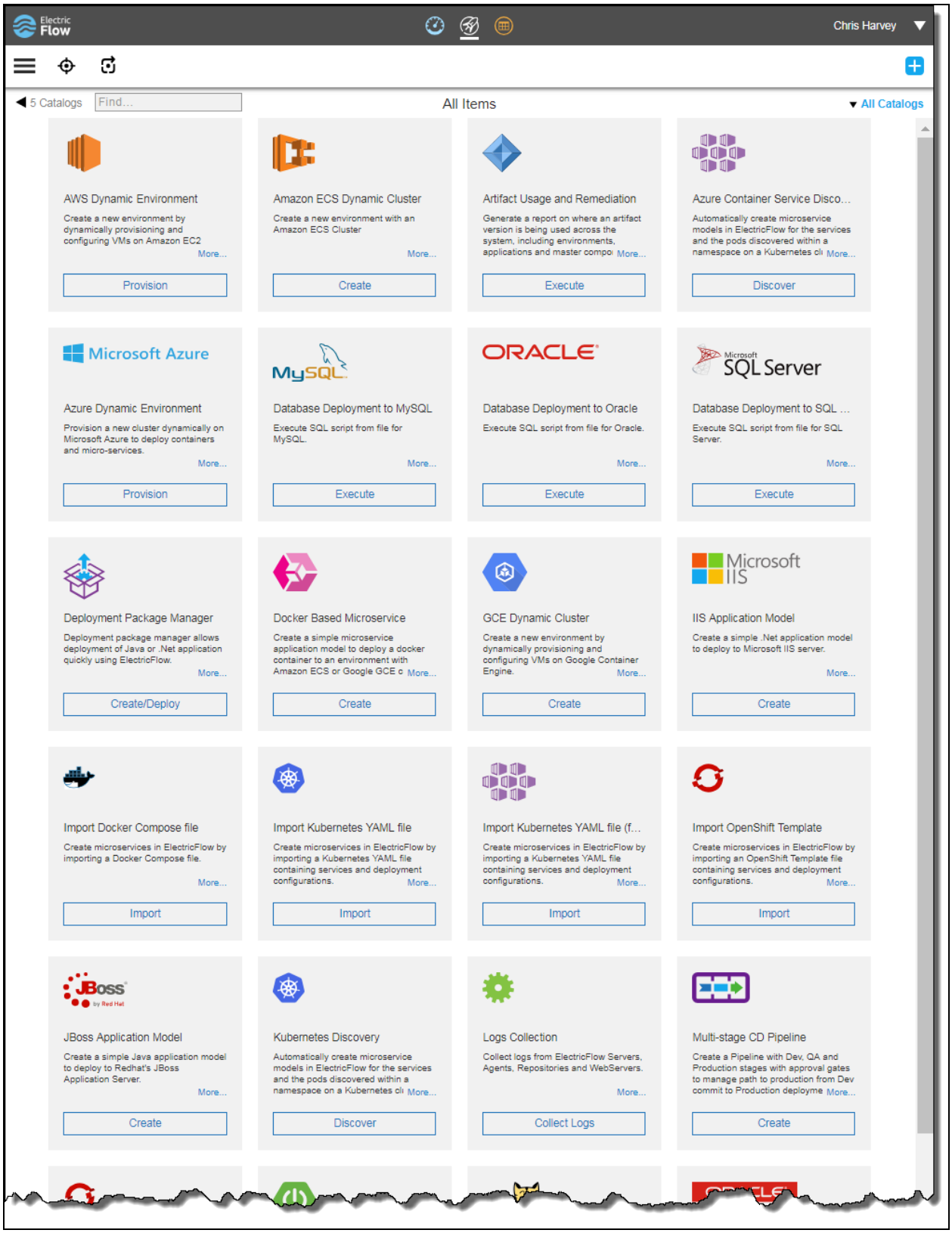
- Individually at the project top level—You simply enter the project name (as described in the procedure below).
- Within a new application—You enter the project name and application name and also enable ElectricFlow to create microservices within the application.

(Optional) Mapping the Microservices to an Existing Environment Cluster

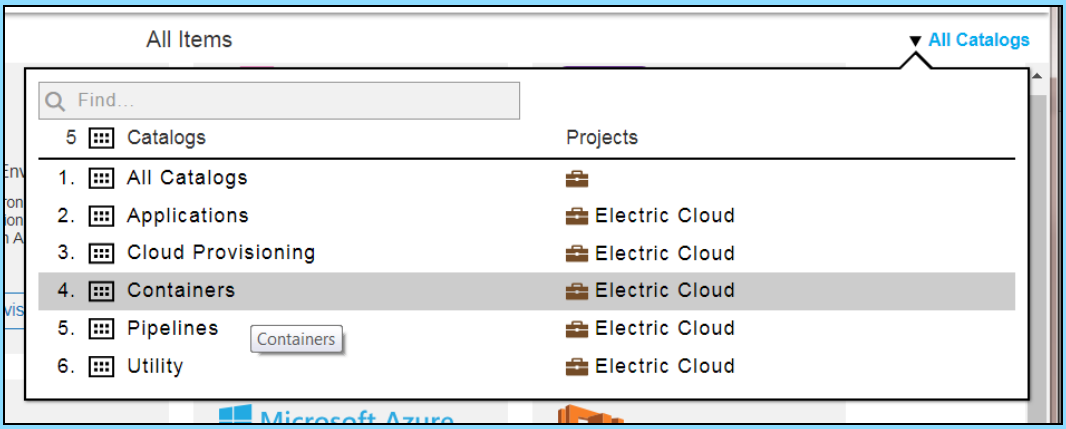
You can select an environment that contains a cluster with Kubernetes configuration details where the new microservices will be deployed. To do so, you enter an environment name and its project name and the name of an EC-Kubernetes backed cluster in the environment.

1. Click the  (Self Service Catalogs) button in the banner bar.

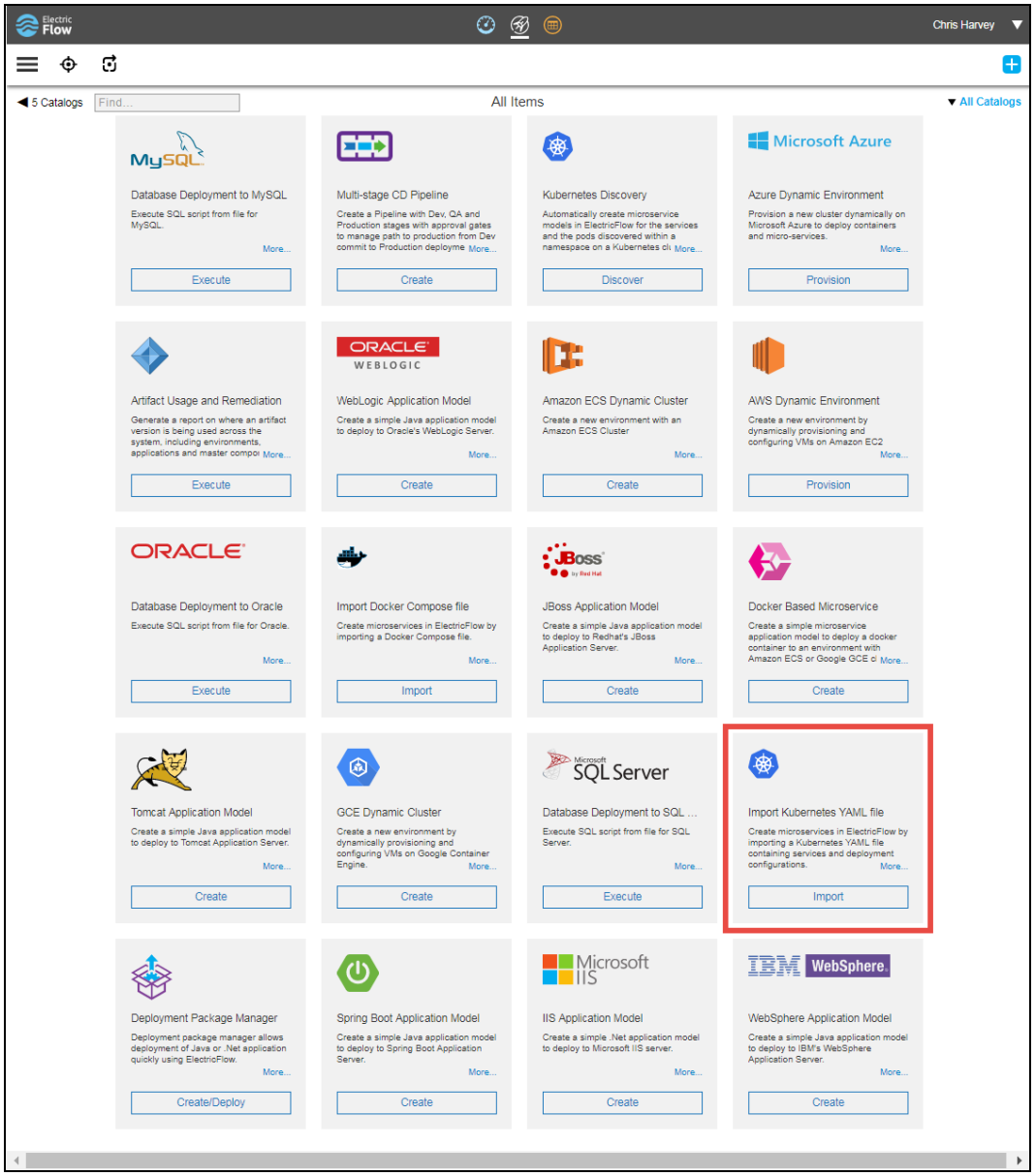
   The Self-Service Catalog opens:


Electric Flow
Chris Harvey
5 Catalogs
Find...
All Items
All Catalogs
AWS Dynamic Environment
Create a new environment by dynamically provisioning and configuring VMs on Amazon EC2
More...
Provision
Amazon ECS Dynamic Cluster
Create a new environment with an Amazon ECS Cluster
More...
Create
Artifact Usage and Remediation
Generate a report on where an artifact version is being used across the system, including environments, applications and master compo
More...
Execute
Azure Container Service Disco...
Automatically create microservice models in ElectricFlow for the services and the pods discovered within a namespace on a Kubernetes cl
More...
Discover
Microsoft Azure
Azure Dynamic Environment
Provision a new cluster dynamically on Microsoft Azure to deploy containers and micro-services.
More...
Provision
MySQL
Database Deployment to MySQL
Execute SQL script from file for MySQL.
More...
Execute
ORACLE
Database Deployment to Oracle
Execute SQL script from file for Oracle.
More...
Execute
Microsoft SQL Server
Database Deployment to SQL ...
Execute SQL script from file for SQL Server.
More...
Execute
Deployment Package Manager
Deployment package manager allows deployment of Java or .Net application quickly using ElectricFlow.
More...
Create/Deploy
Docker Based Microservice
Create a simple microservice application model to deploy a docker container to an environment with Amazon ECS or Google GCE c
More...
Create
GCE Dynamic Cluster
Create a new environment by dynamically provisioning and configuring VMs on Google Container Engine.
More...
Create
Microsoft IIS
IIS Application Model
Create a simple .Net application model to deploy to Microsoft IIS server.
More...
Create
Import Docker Compose file
Create microservices in ElectricFlow by importing a Docker Compose file.
More...
Import
Import Kubernetes YAML file
Create microservices in ElectricFlow by importing a Kubernetes YAML file containing services and deployment configurations.
More...
Import
Import Kubernetes YAML file (f...
Create microservices in ElectricFlow by importing a Kubernetes YAML file containing services and deployment configurations.
More...
Import
Import OpenShift Template
Create microservices in ElectricFlow by importing an OpenShift Template file containing services and deployment configurations.
More...
Import
JBoss by Red Hat
JBoss Application Model
Create a simple Java application model to deploy to Redhat's JBoss Application Server.
More...
Create
Kubernetes Discovery
Automatically create microservice models in ElectricFlow for the services and the pods discovered within a namespace on a Kubernetes cl
More...
Discover
Logs Collection
Collect logs from ElectricFlow Servers, Agents, Repositories and WebServers.
More...
Collect Logs
Multi-stage CD Pipeline
Create a Pipeline with Dev, QA and Production stages with approval gates to manage path to production from Dev commit to Production deployme
More...
Create

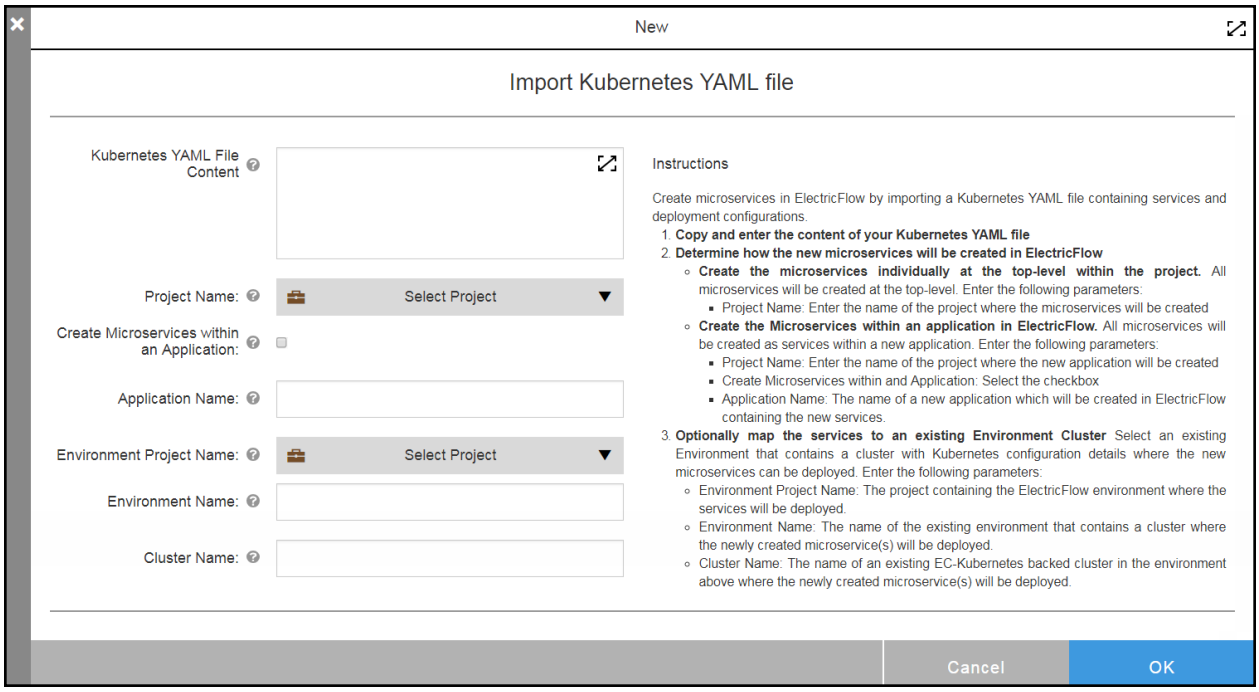
Tip:
This page shows all available catalog items by default. You can narrow down the list by choosing **Containers** from the catalogs menu:



2. Click the **Import** button on the **Import Kubernetes YAML File** catalog item:



The **Import Kubernetes YAML File** dialog box appears:


New
Import Kubernetes YAML file
Kubernetes YAML File Content
Project Name:
Select Project
Create Microservices within an Application:
Application Name:
Environment Project Name:
Select Project
Environment Name:
Cluster Name:
Instructions
Create microservices in ElectricFlow by importing a Kubernetes YAML file containing services and deployment configurations.
1. Copy and enter the content of your Kubernetes YAML file
2. Determine how the new microservices will be created in ElectricFlow
Create the microservices individually at the top-level within the project. All microservices will be created at the top-level. Enter the following parameters:
Project Name: Enter the name of the project where the microservices will be created
Create the Microservices within an application in ElectricFlow. All microservices will be created as services within a new application. Enter the following parameters:
Project Name: Enter the name of the project where the new application will be created
Create Microservices within and Application: Select the checkbox
Application Name: The name of a new application which will be created in ElectricFlow containing the new services.
3. Optionally map the services to an existing Environment Cluster Select an existing Environment that contains a cluster with Kubernetes configuration details where the new microservices can be deployed. Enter the following parameters:
Environment Project Name: The project containing the ElectricFlow environment where the services will be deployed.
Environment Name: The name of the existing environment that contains a cluster where the newly created microservice(s) will be deployed.
Cluster Name: The name of an existing EC-Kubernetes backed cluster in the environment above where the newly created microservice(s) will be deployed.
Cancel
OK

3. Enter the details about your Kubernetes YAML file detailing your microservice definitions and dependencies as follows:

Field or Option	Description
Kubernetes YAML File Content	Content of a Kubernetes YAML file containing related microservice and deployment definitions.
Project Name	Name of the project in which the application or microservices will be created.
Create Microservices within an Application	(Optional) Select this checkbox to create all microservices defined in the Kubernetes YAML file within one application in ElectricFlow. If selected, then you must enter the application name. If unselected, microservices are created at the top level in the project.
Application Name	Name of the new application to contain the microservices. This is required only if **Create Microservices within an Application** is selected.
Environment Project Name	Project containing the environment where the microservices will be deployed.
Environment Name	(Optional) Name of an existing environment containing a cluster where the new microservices will be deployed.
Cluster Name	Name of the existing ElectricFlow cluster representing a Kubernetes cluster in the environment above where the new microservices will be deployed. If the environment does not exist, provide the name of the cluster to be created in the new environment.

4. Click **OK** to save your changes.

   ElectricFlow creates the associated microservice models and environments in ElectricFlow that are ready to deploy.

Microservice Onboarding by Importing a Docker Compose File

Docker Compose is a tool for defining and running multicontainer Docker applications. With Compose, you use a YAML file to configure all of the application's services.

You can create microservices in ElectricFlow by importing a Docker Compose file (version 3.0 or newer). The Compose file can contain configuration details for one or more microservices.

Choosing Where the Microservices Will Be Created in ElectricFlow

You have two choices:

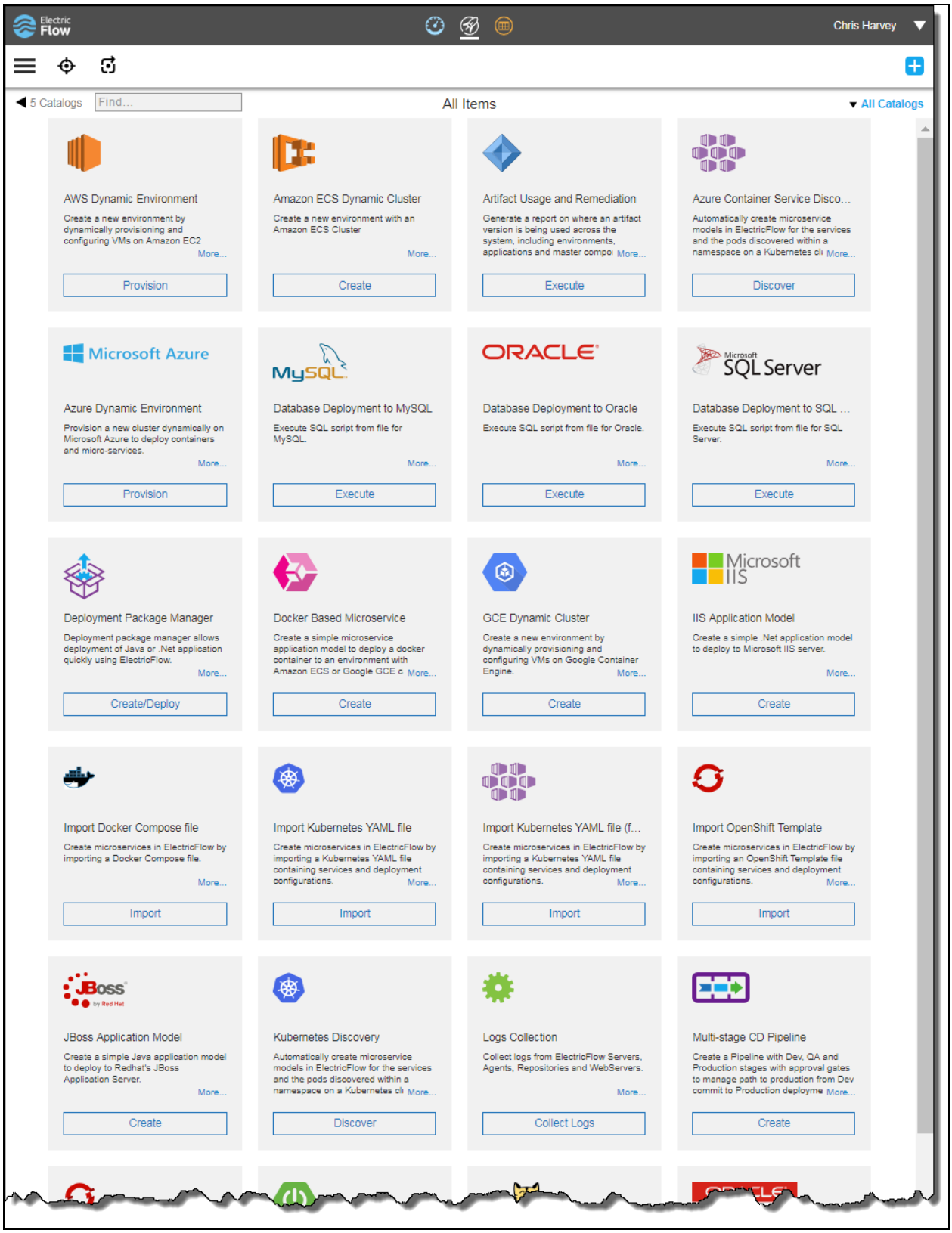
- Individually at the project top level—You simply enter the project name (as described in the procedure below).
- Within a new application—You enter the project name and application name and also enable ElectricFlow to create microservices within the application.

(Optional) Mapping the Microservices to an Existing Environment Cluster

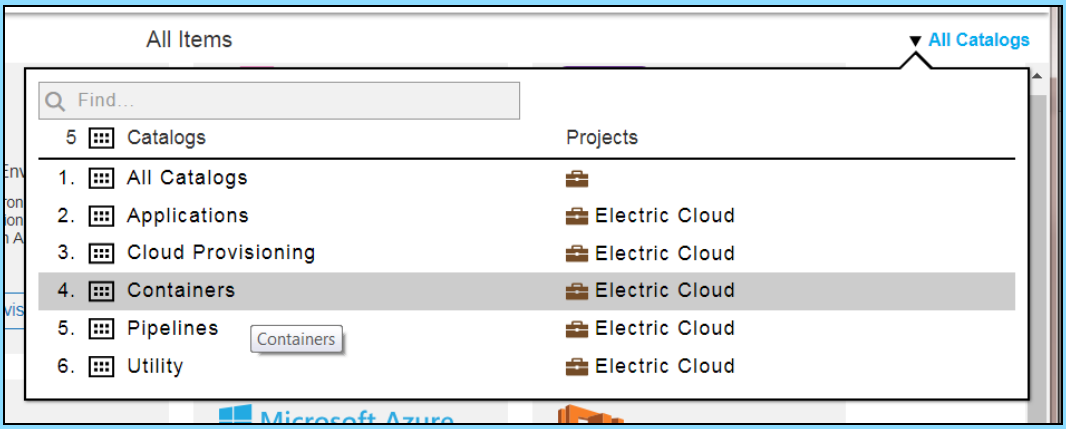
You can select an environment that contains a cluster with EC-Docker configuration details where the new microservices can be deployed. To do so, you enter an environment name and its project name and the name of an EC-Docker backed cluster in the environment.

1. Click the  (Self Service Catalogs) button in the banner bar.

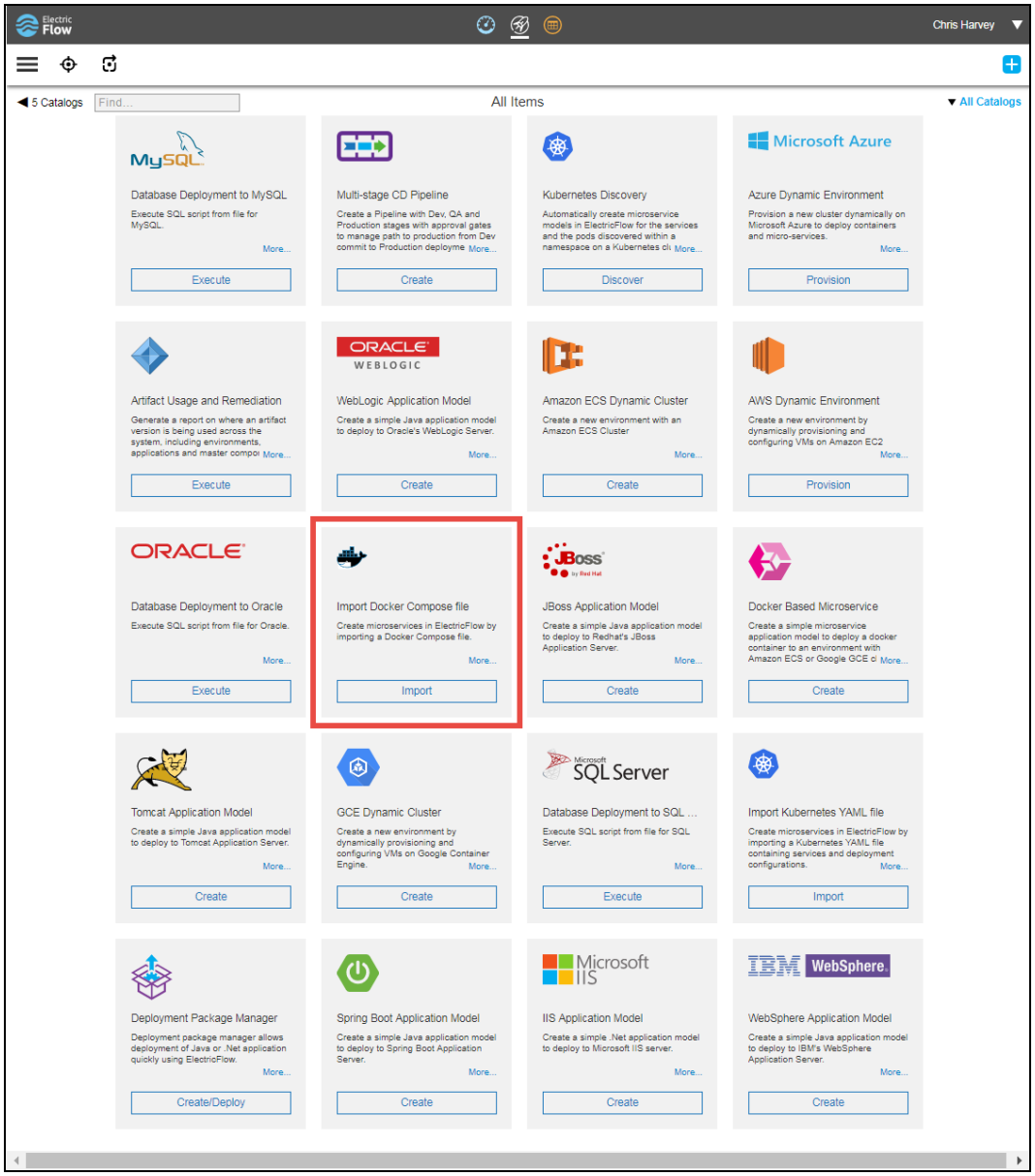
   The Self-Service Catalog opens:


Electric Flow
Chris Harvey
5 Catalogs
Find...
All Items
All Catalogs
AWS Dynamic Environment
Create a new environment by dynamically provisioning and configuring VMs on Amazon EC2
More...
Provision
Amazon ECS Dynamic Cluster
Create a new environment with an Amazon ECS Cluster
More...
Create
Artifact Usage and Remediation
Generate a report on where an artifact version is being used across the system, including environments, applications and master compo
More...
Execute
Azure Container Service Disco...
Automatically create microservice models in ElectricFlow for the services and the pods discovered within a namespace on a Kubernetes cl
More...
Discover
Microsoft Azure
Azure Dynamic Environment
Provision a new cluster dynamically on Microsoft Azure to deploy containers and micro-services.
More...
Provision
MySQL
Database Deployment to MySQL
Execute SQL script from file for MySQL.
More...
Execute
ORACLE
Database Deployment to Oracle
Execute SQL script from file for Oracle.
More...
Execute
Microsoft SQL Server
Database Deployment to SQL ...
Execute SQL script from file for SQL Server.
More...
Execute
Deployment Package Manager
Deployment package manager allows deployment of Java or .Net application quickly using ElectricFlow.
More...
Create/Deploy
Docker Based Microservice
Create a simple microservice application model to deploy a docker container to an environment with Amazon ECS or Google GCE c
More...
Create
GCE Dynamic Cluster
Create a new environment by dynamically provisioning and configuring VMs on Google Container Engine.
More...
Create
Microsoft IIS
IIS Application Model
Create a simple .Net application model to deploy to Microsoft IIS server.
More...
Create
Import Docker Compose file
Create microservices in ElectricFlow by importing a Docker Compose file.
More...
Import
Import Kubernetes YAML file
Create microservices in ElectricFlow by importing a Kubernetes YAML file containing services and deployment configurations.
More...
Import
Import Kubernetes YAML file (f...
Create microservices in ElectricFlow by importing a Kubernetes YAML file containing services and deployment configurations.
More...
Import
Import OpenShift Template
Create microservices in ElectricFlow by importing an OpenShift Template file containing services and deployment configurations.
More...
Import
JBoss by Red Hat
JBoss Application Model
Create a simple Java application model to deploy to Redhat's JBoss Application Server.
More...
Create
Kubernetes Discovery
Automatically create microservice models in ElectricFlow for the services and the pods discovered within a namespace on a Kubernetes cl
More...
Discover
Logs Collection
Collect logs from ElectricFlow Servers, Agents, Repositories and WebServers.
More...
Collect Logs
Multi-stage CD Pipeline
Create a Pipeline with Dev, QA and Production stages with approval gates to manage path to production from Dev commit to Production deployme
More...
Create

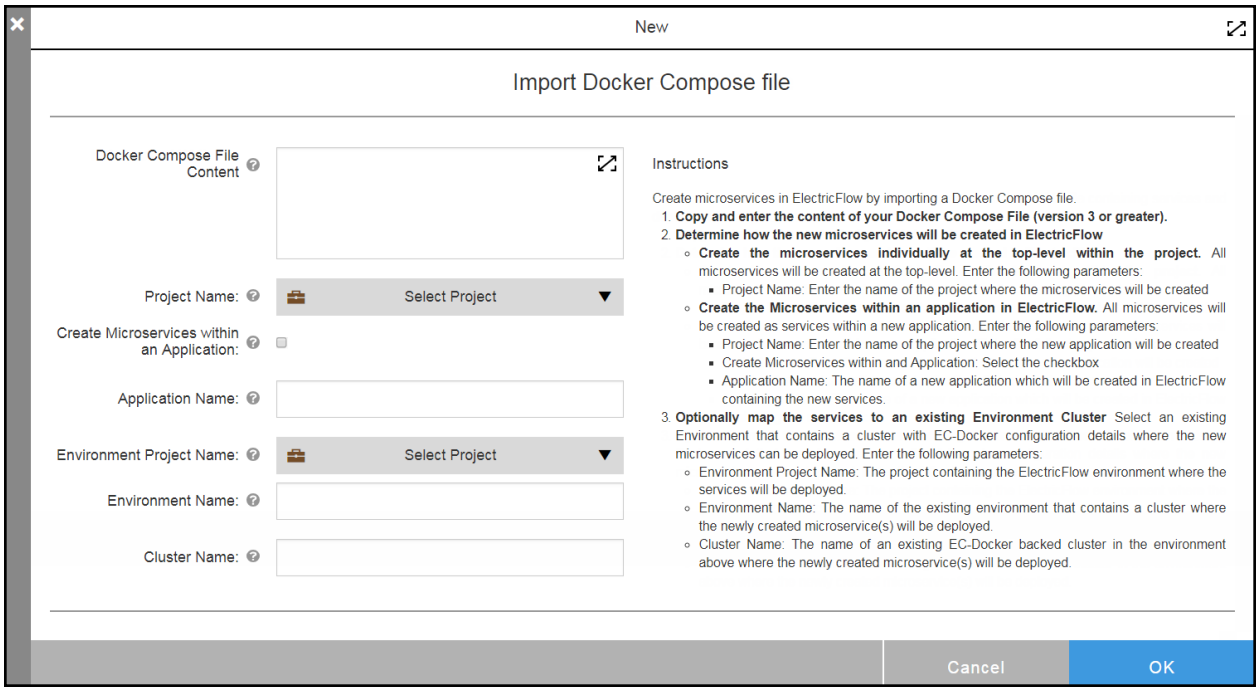
Tip:
This page shows all available catalog items by default. You can narrow down the list by choosing **Containers** from the catalogs menu:



2. Click the **Import** button on the **Import Docker Compose File** catalog item:



The **Import Docker Compose File** dialog box appears:


New
Import Docker Compose file
Docker Compose File Content
Project Name:
Select Project
Create Microservices within an Application:
Application Name:
Environment Project Name:
Select Project
Environment Name:
Cluster Name:
Instructions
Create microservices in ElectricFlow by importing a Docker Compose file.
1. Copy and enter the content of your Docker Compose File (version 3 or greater).
2. Determine how the new microservices will be created in ElectricFlow
Create the microservices individually at the top-level within the project. All microservices will be created at the top-level. Enter the following parameters:
Project Name: Enter the name of the project where the microservices will be created
Create the Microservices within an application in ElectricFlow. All microservices will be created as services within a new application. Enter the following parameters:
Project Name: Enter the name of the project where the new application will be created
Create Microservices within and Application: Select the checkbox
Application Name: The name of a new application which will be created in ElectricFlow containing the new services.
3. Optionally map the services to an existing Environment Cluster Select an existing Environment that contains a cluster with EC-Docker configuration details where the new microservices can be deployed. Enter the following parameters:
Environment Project Name: The project containing the ElectricFlow environment where the services will be deployed.
Environment Name: The name of the existing environment that contains a cluster where the newly created microservice(s) will be deployed.
Cluster Name: The name of an existing EC-Docker backed cluster in the environment above where the newly created microservice(s) will be deployed.
Cancel
OK

3. Enter the details about your Docker Compose file detailing your microservices definitions and dependencies as follows:

Field or Option	Description
Docker Compose File Content	Content of a Docker Compose file containing related microservice and deployments definitions.
Project Name	Name of the project in which the application or microservices will be created.
Create Microservices within an Application	(Optional) Select this checkbox to create all microservices defined in the Docker Compose file within one application in ElectricFlow. If selected, then you must enter the application name. If unselected, microservices are created at the top level in the project.
Application Name	Name of the new application to contain the microservices. This is required only if **Create Microservices within an Application** is selected.
Environment Project Name	Project containing the environment where the microservices will be deployed.
Environment Name	(Optional) Name of an existing environment containing a cluster where the new microservices will be deployed.
Cluster Name	Name of the existing EC-Docker backed cluster in the environment above where the new microservices will be deployed. If the environment does not exist, provide the name of the cluster to be created in the new environment.

4. Click **OK** to save your changes.

   ElectricFlow creates the associated microservice models and environments in ElectricFlow that are ready to deploy.

Using Lift-and-Shift to Migrate Applications to Containers

The ElectricFlow Lift and Shift functionality for microservices lets you quickly start realizing the benefits of running applications inside containers by "lifting and shifting" your existing monolithic artifacts without the effort required to decompose and rewrite them. This capability significantly accelerates container adoption and quickly provides benefits such as code portability, simplified deployments, and easy scaling to cater to the increased demand that containers bring.

You use Lift and Shift to quickly move monolithic or other applications to containers. Lift and Shift begins where your existing Continuous Integration processes end. It takes traditional build output and uses built-in templates to build and publish Docker images automatically that are ready to deploy to your container runtime. Lift and Shift lets you start from existing artifacts for container deployments rather than starting after containers are in the registry. This functionality allows a pragmatic

organization to shift their existing artifacts as-is, and then over time, re-architect monolithic applications into smaller services.

Usage Scenario Example

An ideal usage scenario is one where your organization uses a well-established Continuous Integration process and is creating artifact files supported by ElectricFlow Lift and Shift such as .jar or .war files. This feature lets you use ElectricFlow to take the build output, create a container, store it to a Docker registry, and then even deploy it at the right time in the pipeline.

Using the EC-Docker Plugin for Lift-and-Shift

You use the EC-Docker plugin (bundled with ElectricFlow) to perform lift-and-shift. This plugin contains the `Artifact2Image` procedure, which builds and publishes a container image using an artifact published to an artifact repository as follows:

1. Generates a Dockerfile using the artifact repository details.
2. Uses a built-in Dockerfile template for the supported artifact type to build the Docker container image using the generated Dockerfile.
3. Publishes the image to the specified Docker registry.

The plugin uses a registry of Dockerfile file templates for the artifact types supported by this feature. The supported types are:

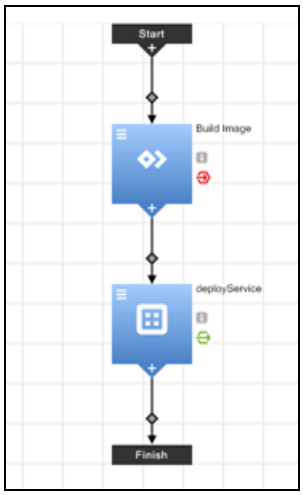
- .war
- .jar
- .NET
- .csproj

For details about using the `Artifact2Image` procedure, click **Administration** > **Plugins** in the Automation Platform and then click the **Help** link next to the EC-Docker plugin name. This plugin procedure can be used in various ways to model lift-and-shift as in the following examples.

Modeling Lift and Shift within a Microservice Model

You use a microservice in ElectricFlow as always. You simply point the microservice definition to a container image name that you want to automatically generate and publish in a registry using lift-and-shift.

To model lift-and-shift within a microservice model, use the `Artifact2Image` procedure in the microservice process as described above. You configure where the traditional artifact resides and use the same image name described in the previous paragraph. For example, see the step named **Build Image** below:

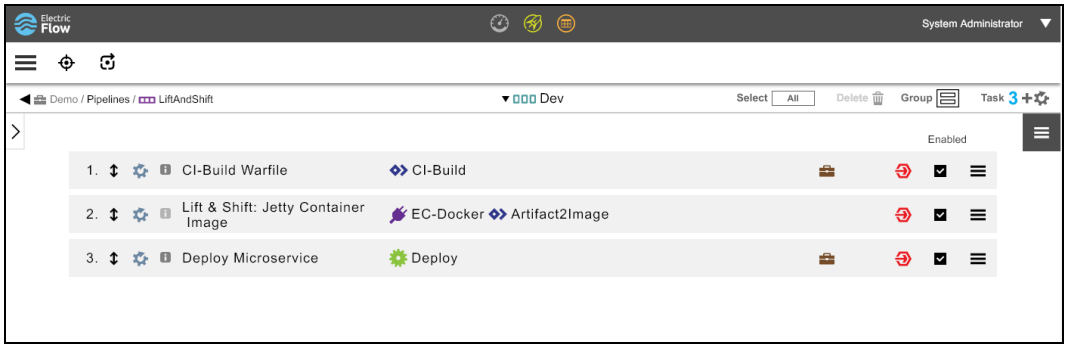


You can also include a step to "deploy" the container as in the **deployService** step above.

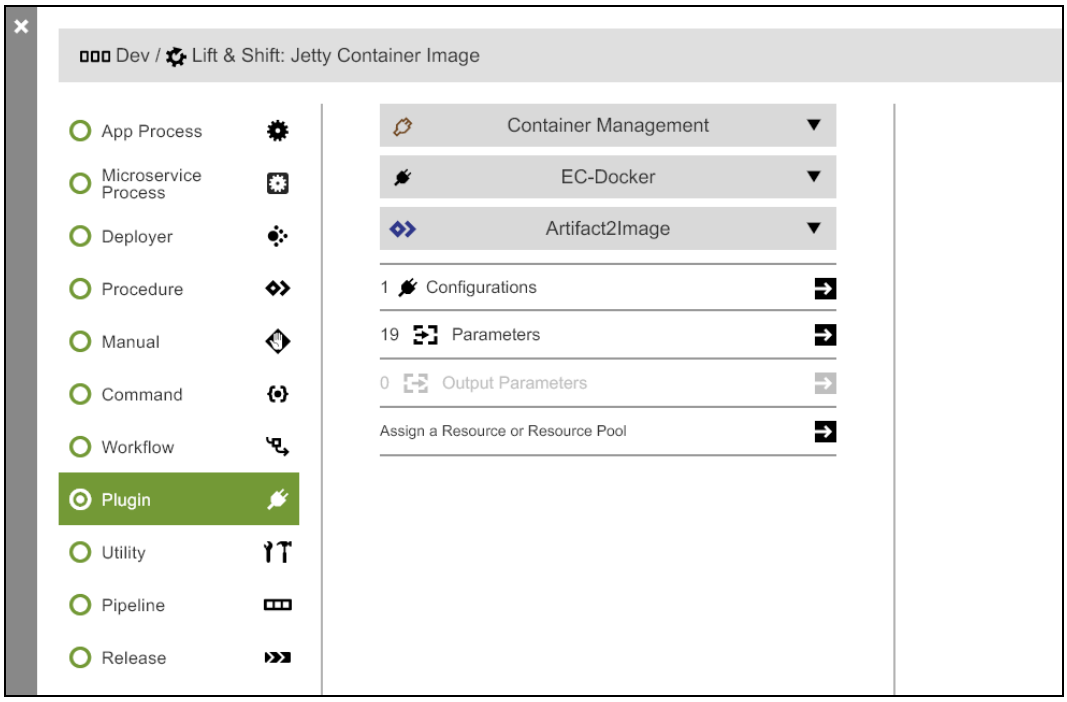
When this process is run, ElectricFlow automatically gets the monolithic artifact from the artifact repository, creates an appropriate Docker image based on the type of artifact and existing templates, and publishes it a Docker-certified registry.

Modeling Lift and Shift by Using Pipelines

You can also use the `Artifact2Image` procedure as a pipeline task such as the **Lift & Shift: Jetty Container Image** task below:



You configure this task to use the `Artifact2Image` procedure as in the following example:



Using SmartMap to Visualize Running Microservices with Dependencies and Connections to Applications

The SmartMap Topology View provides a 3-D view into container platform clusters. The SmartMap Topology View lets you visualize running microservices with their dependencies and connections to applications.

Container runtime environments can include numerous services deployed across clusters with detailed interdependencies between them. It can be challenging to understand the complex mesh of clusters, namespaces, services, nodes, and so on and how they are related, especially as the numbers grow. It is also not easy for DevOps and development teams to know where their service is deployed and how well is it doing or to access logs for troubleshooting without talking with the Kubernetes administrators.

While microservices simplify the deployment problem and allows parts to be managed independently, it creates another problem of management downstream. Applications made up of a few monolithic artifacts are decomposed into lots of small microservices, which can be deployed independently. Also, even though deployment time dependencies are removed, there are runtime dependencies among microservices. In other words, a version of a microservice needs specific versions of other microservices to be present.

The SmartMap Topology View helps you visualize and manage your container cluster environments. The interactive 3-D visualization displays the interdependent services, clusters, and pods deployed in Kubernetes as well as the connections to the microservice and environment models in ElectricFlow. With capabilities to interactively rotate and zoom in on focus areas, view live status of deployments, and drill down into details, SmartMap makes it easy to quickly understand and "touch" your complex microservice topology. It also helps with troubleshooting and impact analysis.

SmartMap Topology View Example

The SmartMap Topology View retrieves application and service dependencies along with cluster and environment information. For a given application or service, SmartMap Topology View retrieves

- All services and applications it is dependent on (including dependencies among dependent services)
- All environments and clusters it is deployed to

For a given environment or cluster, SmartMap Topology View retrieves

- All services or applications deployed to it
- Dependencies between services

SmartMap Topology View then renders the topology view from this information. For example:

Clicking an object opens a details panel for that object as shown in the screenshot above.

Benefits of SmartMap Topology View

Because of this increased granularity and resolution, more pieces need to be managed together to provide the service to end users. This problem is more difficult if users have multiple container runtime platforms. As you move from the experimental phase of microservices to starting to manage production load, you can use the SmartMap Topology View to

- Visualize which microservices are deployed and how are they connected to each other
- Understand which application your running microservices are supporting and which applications will be impacted if these microservices go down
- See whether all the required version dependencies are fulfilled

SmartMap Topology View Usage Scenario

The SmartMap Topology View is useful for an application support person or a DevOps team member responsible for production who needs to understand which microservices are running in their cluster and how are they connected to each other. They might have tried to deploy microservices and have modeled them with a version dependency graph. They want to start with a top-level microservice and ensure that all the required dependent versions are deployed automatically.

Without the SmartMap Topology View, they would need to use visualization solutions from other vendors and open-source solutions, but those solutions lack the connection to applications. Or they

would need to use visualizations from application performance management (APM) vendors, which lack connection to the release process.

Configuring a Database for Containers and Microservices

Following are the recommended sources of information that describe how to connect to a database for containers and microservices.

Connecting to Kubernetes

See *Connecting from Google Container Engine* at https://cloud.google.com/sql/docs/container-engine-connect or *Connecting MySQL Client Using IP Addresses* at https://cloud.google.com/sql/docs/mysql-client.

Connecting to Amazon

See *Connecting to an Amazon RDS DB Instance* at http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_CommonTasks.Connect.html.

Configuring the CIDR Range of Containers Allowed to Connect

Another option is configure the CIDR range of containers that are allowed to connect to a database. See *Creating a DB Cluster and Connecting to a Database on an Amazon Aurora DB Instance* at http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_GettingStarted.CreatingConnecting.Aurora.html.

Deployment Packages

Introduction

A deployment package is a way to enable users to quickly deploy Java or .NET applications to any environment using ElectricFlow. A deployment package is a .zip file that is made up of a manifest file and one or more components that make up an application. This manifest file will have the required inputs to create application and environment mappings automatically in ElectricFlow. In addition to creating models, it is even possible to trigger application deployments.

A deployment package can be generated as part of any Continuous Integration (CI) automation (even with third-party CI tools such Jenkins, TFS, or TeamCity). Also, anyone with a zip utility and a text editor can manually create a deployment package.

Once the deployment package is created, there are multiple options to consume it to generate the application and environment for deployment:

- Use the Deployment Package Manager from the Service Catalog UI
- From Jenkins using the ElectricFlow plugin
- Use the Deployment Package Manager API
    - From the CLI
    - From any other third-party CI tool

Creating a Deployment Package

Deployment packages can be created automatically as part of a CI process, or they can be created manually by hand.

Manifest File (manifest.json)

The manifest file is a JSON metadata file that describes the application and its contents. You can find a sample manifest.json file at https://github.com/electric-cloud/DeploymentPackageManager/SampleManifests.

Here is a sample manifest.json file for deploying a Java WAR file to JBoss:

```
{
   "application":{
      "name":"Application Name",
      "version":"1.0",
      "components":{
         "component":[
            {
               "artifact":{
                  "artifactType":"JBOSS",
                  "artifactName":"org.ec:jboss",
                  "artifactVersion":"1.0",
                 "artifactFileName": "jboss-as-helloworld.war"
               }
            }
         ]
      }
   }
}
```

Following are the attributes in the sample file above:

Attribute	Descriptions
Name	Name of the application you want to create.
Version	Version of the application. You should increment the version each time a new build is created.
artifactType	Type of the component. For example, JBOSS, Tomcat, WEBSPHERE, or WEBLOGIC.

Attribute	Descriptions
`artifactName`	Component artifact name. This follows the maven GAV (*groupId*, *artifactKey*, and *version*) naming convention.
`artifactVersion`	Version of the component. You should increment the component version each time a new artifact is created.
`artifactFileName`	Name of the component artifact file.

Automatically Deploying the Application from the Deployment Package

A manifest file can optionally include environment and application tier to environment tier mapping information. See the sample manifest.json file that is described above. If this additional information is provided, then the Deployment Package Manager will create the environment, complete the environment mapping, and deploy the application.

Components

Components map to physical files that are generated as part of a Jenkins CI build or a Maven build. For example, jboss-helloworld.war. ElectricFlow supports Java application deployment to JBoss, Tomcat, WebSphere, and WebLogic as well as .NET application deployment to a Microsoft IIS server. If components are provided as part of the zip file, they will be automatically published as artifacts in the ElectricFlow Artifact Repository. Component physical files are optional when you create the deployment package. Users can always specify references to the actual files in the manifest json file.

Creating the Zip File

Once you have created the manifest.json file and the components, you simply zip these files into a .zip file of your choice.This .zip file is called the deployment package.

hello.zip

|-- manifest.json

|-- jboss-helloworld.war

Note: For a continuous delivery (CD) use case, the continuous integration build will generate this package and publish it to the artifact repository.

Deployment Package Manager

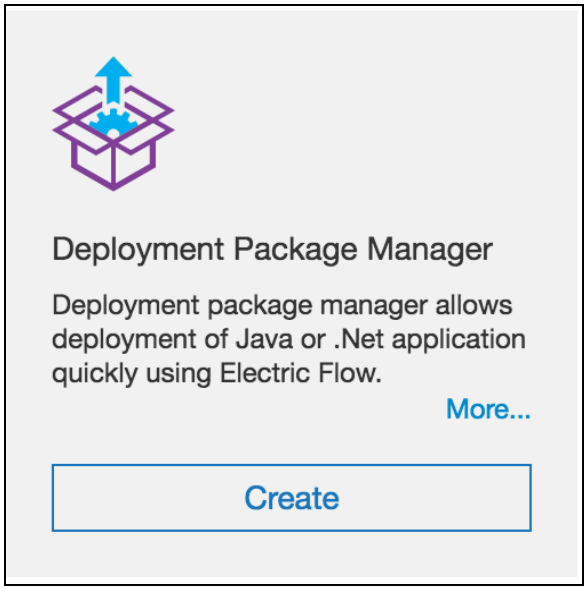
This is a special type of catalog item that allows deployment of Java or .NET applications quickly using ElectricFlow. This item is included in ElectricFlow. This is useful for consuming deployment package zip files manually.

Before you can create this catalog item, you must create the deployment package outside of ElectricFlow. See Creating a Deployment Package on page 172 to create a deployment package.

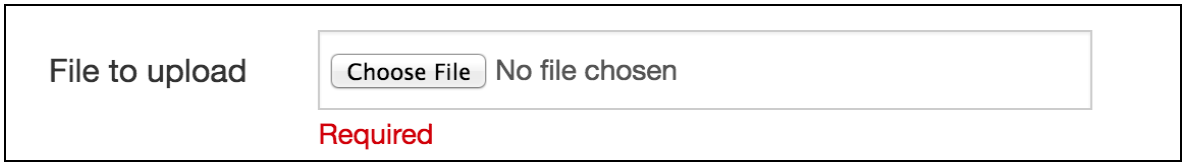
Once you have created the deployment package, you can then leverage the Deployment Package Manager catalog item to create an application model that corresponds to the uploaded application package with no or very little knowledge of ElectricFlow.

Using the Deployment Package Manager from the Service Catalog UI

By clicking the launch ( ) button, you can view all the catalogs that are available to you as an end user. Select the Deployment Package Manager:



You are prompted you to upload the deployment package. Once you click OK, the system starts processing the package:



As part of the package processing, the system publishes the component file included in the package to the artifact repository using the `artifactName` and `version` specified in the manifest file. Once all the components are published, the system generates an application that orchestrates the deployment of each of these components using an application process.

Once the application is created, you can deploy the application to any environment by completing the application mapping.

Deployment Package Manager API

You use this API to invoke the Deployment Package Manager to create an application model and deploy it to any existing environment. The model is based on the deployment package from the CLI and from external applications such as Jenkins.

From the CLI

You can call the `createApplicationFromDeploymentPackage` API from the CLI in order to create an application from a deployment package. Following is the usage:

```
createApplicationFromDeploymentPackage <artifactGroup> <artifactKey> <artifactVersion>
<artifactFileName>

[--dslString <dslString>]

[--retrieveToFolder <retrieveToFolder>]

[--dslFile <dslFile>]
```

The API assumes you have already uploaded your package to the ElectricFlow artifact repository following the GAV (*Group-artifactKey-version*) naming convention as described above.

```
ectool createApplicationFromDeploymentPackage org.ec hello 1.0 hellopackage.zip --
retrieveToFolder /tmp
```

The DSL used by the package manager is defaulted from the Service Catalog item that is referenced by the following custom server property value in the Automation Platform: **Administration > Server > ec_selfservice > defaultPackageManagerCatalogItem > Deployment Package Manager**.

Integrating with Third-Party Continuous Integration Tools Such as Jenkins

While ElectricFlow itself can be used as an enterprise level continuous integration platform, it can be easily integrated with all the leading third-party CI platforms. The example below describes how to invoke actions in ElectricFlow from Jenkins CI.

ElectricFlow has bidirectional integration with Jenkins:

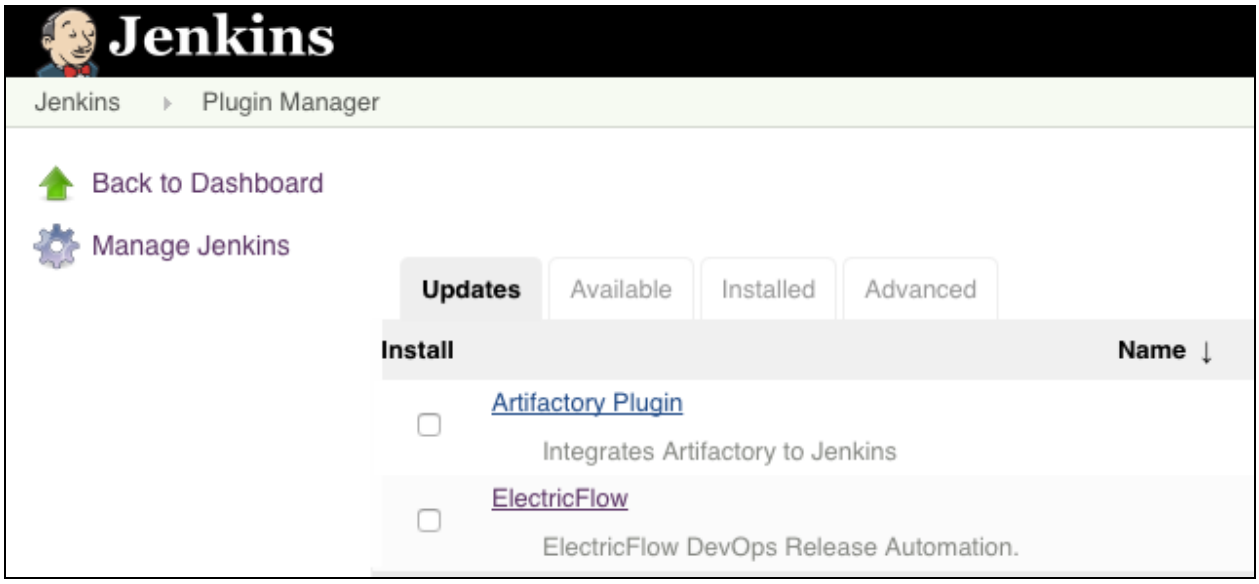
1. Jenkins plugin on the ElectricFlow side: This plugin lets you trigger Jenkins jobs from ElectricFlow. For details, see http://electric-cloud.com/plugins/directory/p/jenkins.
2. ElectricFlow plugin for Jenkins: This plugin lets you launch various actions in ElectricFlow from Jenkins using Jenkins post-build actions such as:
    1. Create Application from Deployment Package to ElectricFlow: Allows for seamless deployment of a package from Jenkins CI to ElectricFlow. This post-build action can be used to push deployment packages to ElectricFlow after a build is completed. This action creates the application in ElectricFlow and can also deploy to its environments.
    2. Publish Artifact to ElectricFlow: This post-build action lets you publish artifact outputs of Jenkins builds to the ElectricFlow Artifact Repository.
    3. Run Pipeline in ElectricFlow: This post-build action lets you trigger a pipeline in ElectricFlow from a Jenkins job or a pipeline.

    For more information, see https://wiki.jenkins-ci.org/display/JENKINS/ElectricFlow+Plugin.

The following example focuses on **Post-Build' action - Create Application from Deployment Package** to ElectricFlow.

Installing and Configuring the Plugin

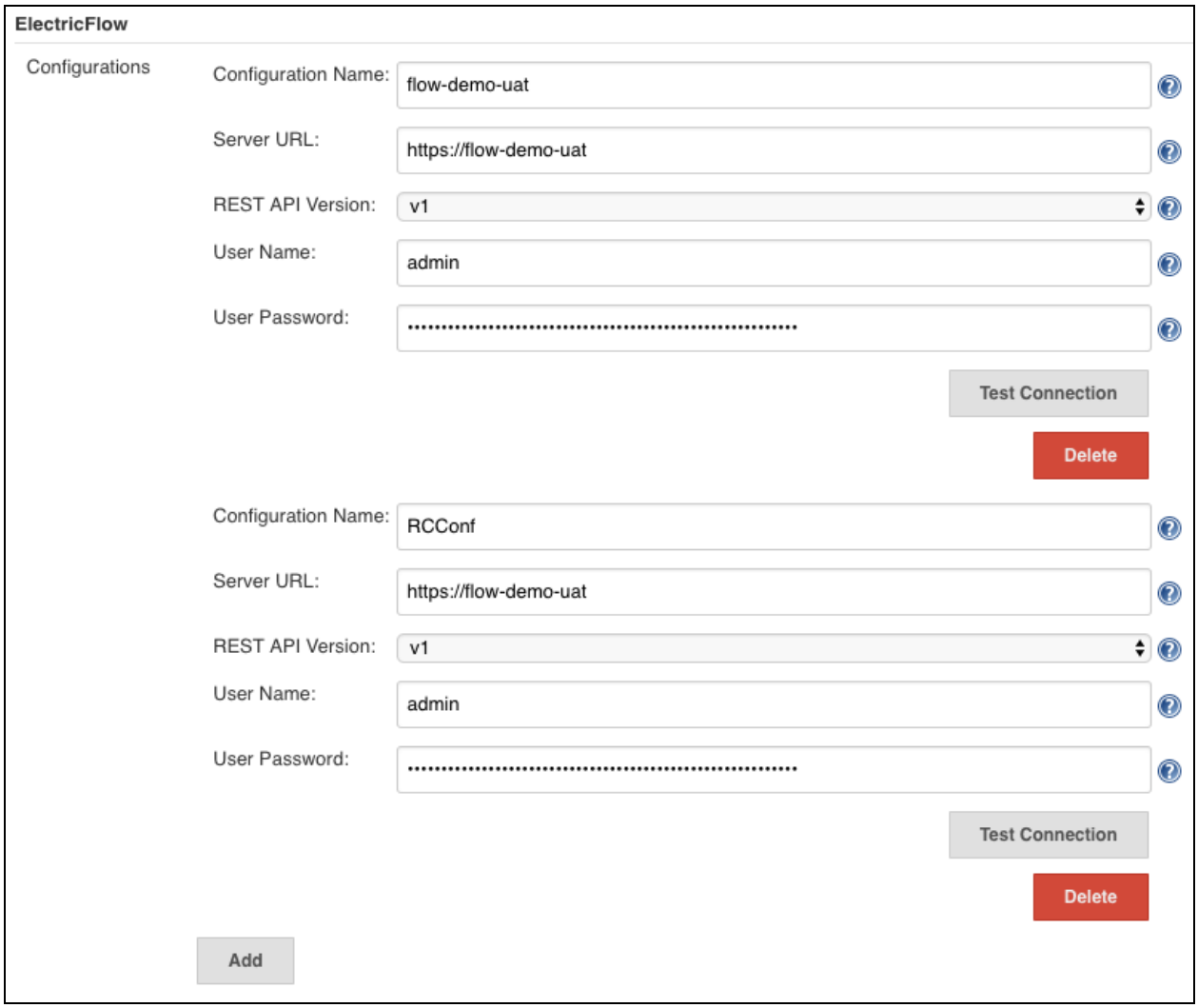
This plugin is available in the Jenkins Plugin Manager and can be downloaded and installed similar to any other Jenkins plugin:



To use and integrate with ElectricFlow, you must create configurations in Jenkins. To do so, navigate to **Manage Jenkins > Configure System** and go to the **ElectricFlow** section.

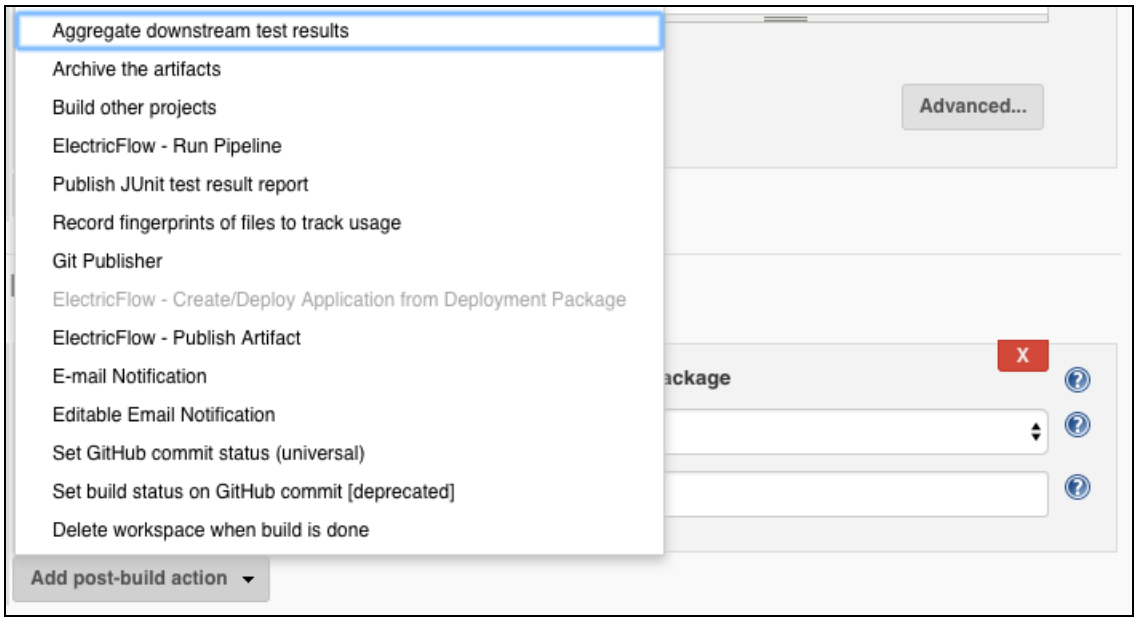
One or more configurations can be created to connect to and call APIs into ElectricFlow. For each configuration, specify the following attributes:

- **Configuration Name**: Name of the ElectricFlow configuration
- **Server URL**: URL for the ElectricFlow Server. For example, `https://<electric-flow-server>`
- **REST API Version**: Version for the ElectricFlow REST API. For example, `v1`
- **User Name**: User name in ElectricFlow
- **User Password**: Password for the above user

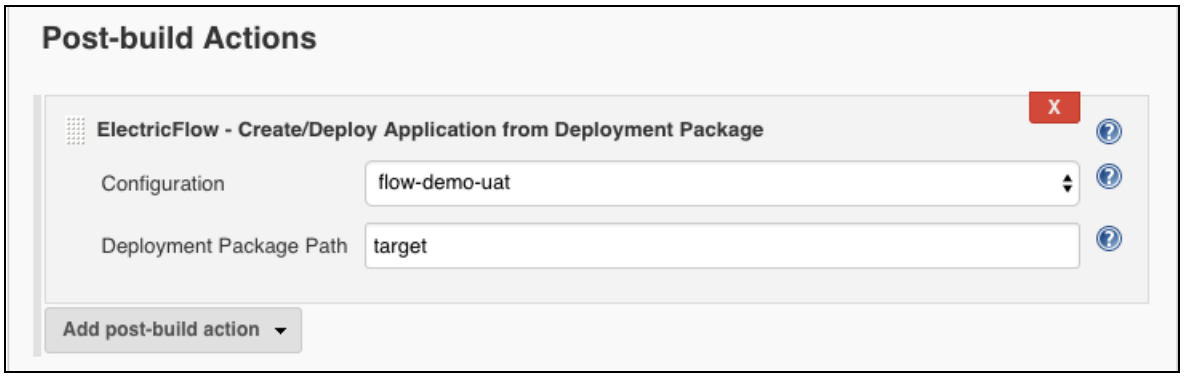


Using the Plugin for the Jenkins Project

The ElectricFlow plugin installs a post-build action named "ElectricFlow - Create/Deploy Application from Deployment Package," which can be added to your Jenkins project. This action will be available as a drop-down:



This build action has the following parameters:



- Configuration: Name of the ElectricFlow configuration. This points to the URL and credentials for the ElectricFlow system. For details, see the plugin Jenkins wiki page at https://wiki.jenkins-ci.org/display/JENKINS/ElectricFlow+Plugin.
- Deployment Package Path: Location or path for the deployment package to be published to ElectricFlow. For example, the target folder where the build output files are located. This folder should contain:
    - Manifest file (manifest.json): This a JSON file that describes details of the application, its components, and artifacts. If the manifest file contains details for the environment and environment mappings, then ElectricFlow will deploy to that environment automatically. See https://github.com/electric-cloud/DeploymentPackageManager/tree/master/SampleManifests for examples of manifest files.
    - Build artifacts files: The actual build outputs such as binaries, packages, .war/ear/jar or .NET files.

As a developer or integrator, follow these steps:

1. Create a manifest file (manifest.json) in your source control system. See https://github.com/electric-cloud/hello-world-war for an example. Also see https://github.com/electric-cloud/DeploymentPackageManager/tree/master/SampleManifests for examples of manifest files.
2. If the manifest.json file has environment information, then ensure that the environment, environment tiers, and resources are defined in ElectricFlow so that the automated deployment flow works.
3. If the Jenkins build is already set up for your SCM project, ensure that the manifest.json file is copied to the Deployment Package Path (such as the target/folder). Also ensure that the WAR file (or any build artifact) is copied to the same folder. For example, the target/ folder should have the following content:

   target/

   |-- manifest.json

   |-- helloworld.war

   Note that manifest.json would have a reference to helloworld.war. Both application and component versions in the manifest.json file should be incremented based on the build number or otherwise.
4. Choose **Add Post-build Action > ElectricFlow - Create/Deploy Application from the Deployment Package** (see the screenshot above).
5. Fill in the ElectricFlow **Configuration** field and the **Deployment Package Path** field (such as the target).

After the above steps are complete, then the next Jenkins build will execute the action to create and/or deploy the application into ElectricFlow.

For More Information

See the "Deployment Package Manager" video in the Electric Cloud YouTube channel.

Deployment Strategies

ElectricFlow provides several ways to deploy applications, including deployment policies (such as rolling deployments, blue/green, canary, and dark launch) and deployment options (such as smart deploy, staging artifacts, and error handling). By default, applications are deployed to all resources in the environment at once, and process steps are run sequentially. You can customize how an application is deployed using the options described in Application Deployment Options on page 125. For example:

- When authoring processes, you can define the error handling method for a process step or use rollback.
- Before the running an application, you can stage artifacts to ensure that all the artifacts are available at runtime, reducing the deployment time.
- At runtime, you can deploy everything (full run), only changed objects (smart deploy), specific objects (partial run), or snapshots.

In some situations, instead of deploying an application to all machines in the environment at once, it is better to deploy the application in batches.

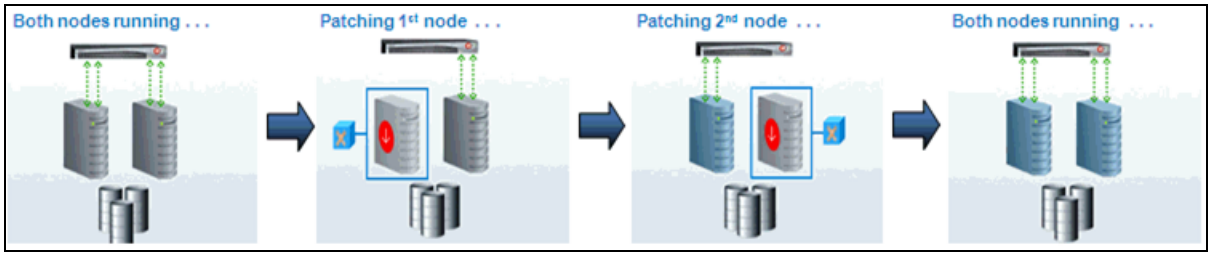
As you get closer to production environments, reducing the downtime and risk while releasing newer versions becomes business critical. This is especially critical in production environments with live end-user traffic all the time. In some cases, the applications are the backbone of a website and are critical to the business, such as a banking or investment website. Any downtime on the production website can adversely affect the business. ElectricFlow allows *modeling* the various deployment strategies described in the following sections. The end goals of all these strategies are very similar – reducing the downtime and risk. ElectricFlow also allows modeling strategies such that they can be practiced in lower environments like QA or Pre-Prod to ensure success during production rollouts.

Rolling Deployments

One way to reduce the downtime and the risk associated with application deployments is to use *rolling deployments*. The goal is to minimize downtime to zero or as low as possible without impacting the availability of business critical applications. Rolling deployments are release patterns where the application is gradually deployed to the machines one at a time or in batches. Rolling deployments can be run throughout the release, but they are especially useful near the end of the release process, close to the production environment.

The Rolling Deployment strategy is applicable when the environment caters to end-user traffic and the environments being upgraded is the same.

This is an example of a rolling deployment in a load-balanced environment. When a new version needs to be added to both nodes, it is deployed to the first node while the second node is actively handling the end-user traffic. After the new version is successfully installed on the first node, it starts to handle the end-user traffic while the new version gets applied to the second node. After the new version is successfully installed on the second node, both nodes can actively handle traffic.



Rolling deployment is an excellent strategy for reducing downtime when you have environments with large numbers of static resources. It is also a cost-effective strategy as no additional resources are required, unlike other strategies. A Rolling deployment generally requires thinking about backward compatibility across the application components.

ElectricFlow natively supports the modeling of rolling deployments for the desired environments. For an environment, you can choose if rolling deployment is enabled. For environments in which you want to run a rolling deployment, you can choose between a phased-based or batch-based deployment.

- Rolling deployment using phases: This rolling deployment strategy is useful where there is a deterministic mapping between resources and the phase to which they belong. You can either assign resources to phases manually or dynamically using expressions.

- Rolling deployment using batches: This rolling deployment strategy is useful for environments with large numbers of resources where deterministic mapping between resources and batches is not necessary.

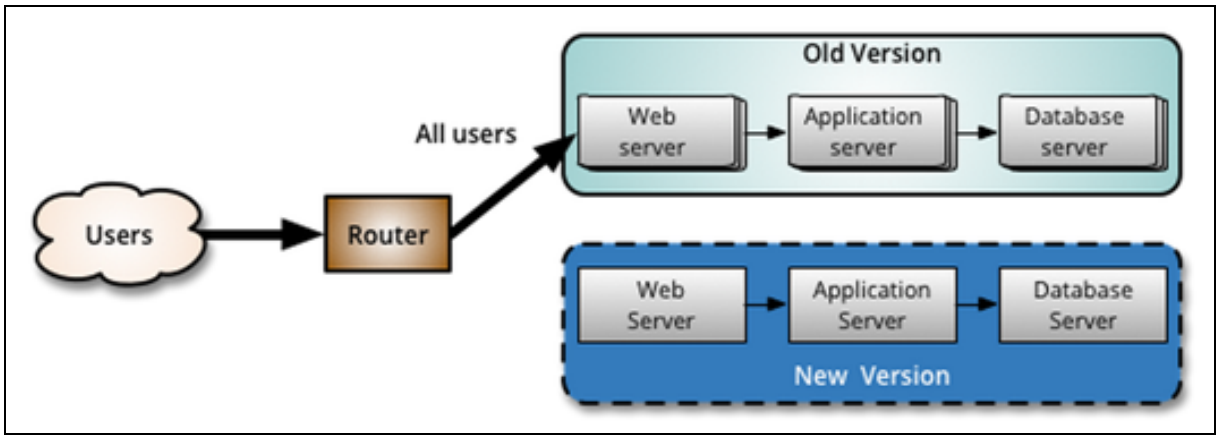
  You can decide batch sizes by specifying a *number* or *percentage* per tier. Examples are deployment to two resources at a time for the Web tier or to 25% of the resources at a time in the application server tier.

  You can also specify a property reference (`$[]`). This is useful when you want to apply a property reference to control the batch size based on applications, external configuration files, and so on.

  See Rolling Deployment Use Case on page 184 for an example of modeling rolling deployment.

Blue/Green Deployments

This technique reduces downtime and risk by running two identical production environments referred to as *blue* and *green*. The key difference between rolling deployment and blue/green deployment is that there are two physically separate environments. At any time, only one of these environments is live, serving all of the production traffic. While new versions of the applications are deployed to the second (blue) environment, the first environment (green) is serving production traffic. When the new versions are satisfactorily deployed to the second (blue) environment, all of the end-user traffic (100%) is diverted to it to make it live. After the switch, the green environment becomes inactive and next release can be applied to it and the process can be repeated.



The blue/green deployment strategy is especially useful with dynamic cloud environments. If environments are automatically spun up for every new deployment,the blue/green deployment strategy becomes compelling as the older environment can be decommissioned easily. The blue/green strategy is also useful when it is mandatory to maintain a separate mirrored environment for disaster recovery, as required by some financial companies.

While blue/green deployment makes it much simpler to recover from unforeseen deployment errors, as there are two separate environments, it comes with a higher cost. Because the blue/green strategy requires more than one environment and resource duplication, the overall costs might be higher than rolling deployments.

ElectricFlow allows modeling blue/green deployments with multiple options, but the most logical option is to create separate blue and green environments in the product. These environments are identical for all practical purposes. They can be created before the deployment if they are using static resources, or they can be created dynamically at the deployment runtime using *environment templates*. Modeling

blue and *green* as separate environments also allows keeping track of the component inventory independently so that users can always know what is deployed to the blue vs green environments at any time.

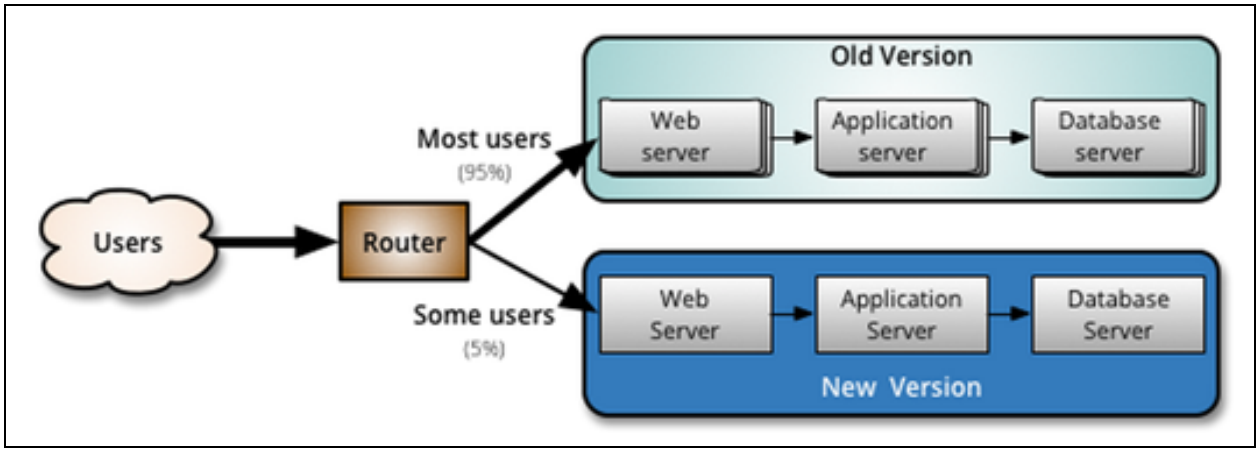
See Blue/Green Deployment Use Case on page 194 for an example of modeling a blue/green deployment.

Canary Deployments

This technique is an advanced deployment strategy to reduce the risk of new version rollouts by initially releasing them only to a subset of users. Even a canary deployment strategy typically has two separate environments.

During deployment, new versions are applied to a second inactive environment (as with a blue/green deployment). After the second environment with the newer versions is tested satisfactorily, part of the end-user traffic is diverted to it using a load balancer configuration. The first environment runs the old production versions of the applications and bears the majority of the traffic, while the second environment runs the new versions and handles a small part of the traffic.

This is an effective way to test new versions with live traffic and reduce the risk by containing the exposure. If everything looks fine, all traffic can be diverted to the environment with new versions. Or if the new versions have issues, the older versions can be kept instead.



You can model canary deployments natively by creating separate environment objects and then automating loading balancer interactions. These separate environments can be created before the deployment, if they are using static resources, or they can be created dynamically at deployment runtime using environment templates. Modeling them as separate environments also lets you keep track of the component inventory independently so that users can always know what is deployed to which environment at any time.

See Canary Deployment Use Case on page 198 for an example of modeling a canary deployment.

Dark Launch Deployments

You normally use the Dark Launch deployment strategy for deploying new features to a production environment but enabling them only partially or not at all. This is a good practice for testing new code in production without exposing it to end users. This also lets business owners and product managers enable or disable features to test their impact before a wide rollout.

See Dark Launch Deployment Use Case on page 203 for an example of modeling a blue/green deployment.

Hot Deployments

A hot deployment strategy typically means changing a running application without causing downtime or without restarting the middleware or infrastructure components that can impact end users. This type of deployment usually depends on the capabilities of the underlying middleware technology. Many application server technologies (such as Tomcat, Red Hat JBoss, IBM WebSphere, and Oracle WebLogic) allow deployments and other adjustments to the application while the server is running. This overcomes a major issue of deployment practices that require a full server restart.

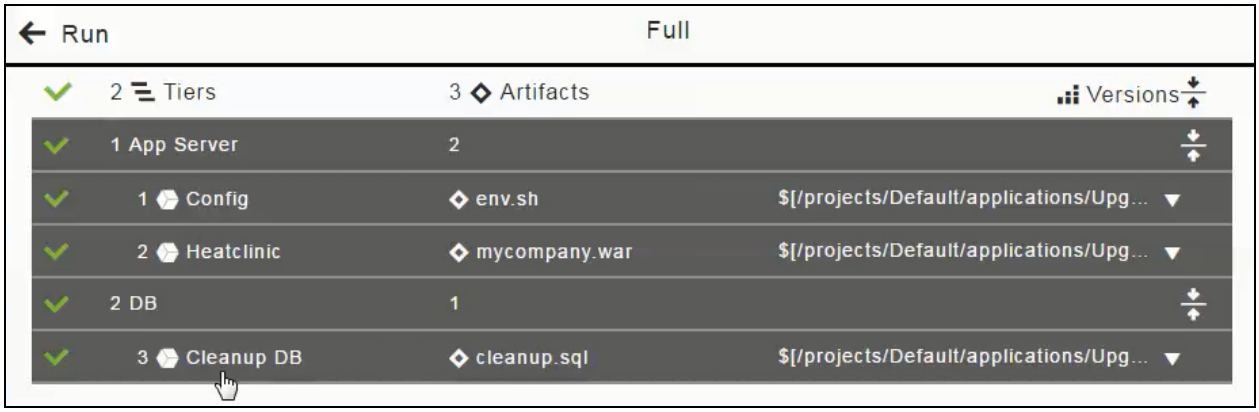
See Hot Deployment Use Case on page 211for an example of modeling a hot deployment.

Partial Deployments

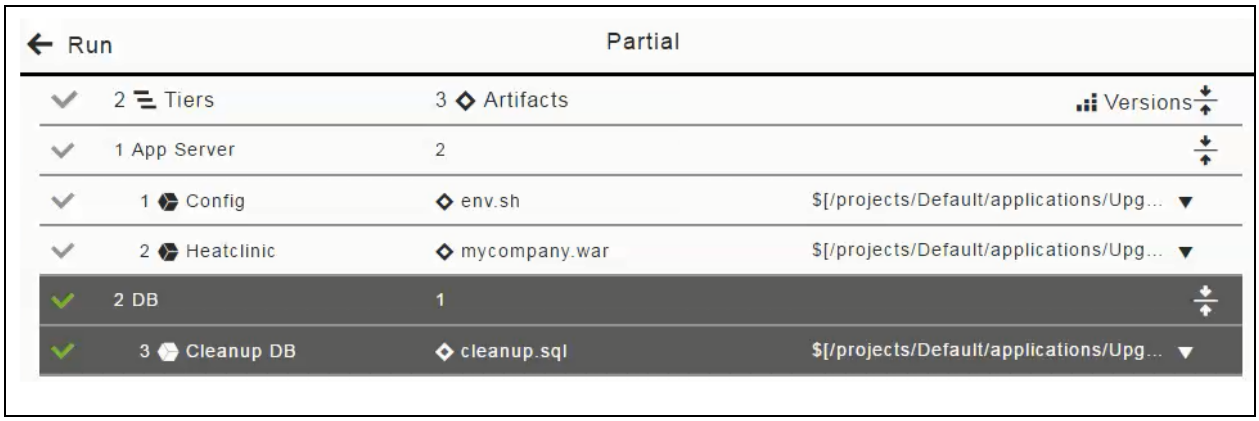
You can deploy only certain objects in scenarios in which other objects are not ready for deployment or you just want to test certain objects independently such as when you want to verify incremental changes. You can also do a partial deployment with only specific artifact versions.

The following screenshot shows how to do a partial deploy with smart deploy and artifact staging enabled:

When you run an application deployment, clicking the (details) opens the list of artifacts and containers that can be deployed in the application process. Artifacts are grouped by application tier. You can enable or disable an artifact within a tier or the entire tier itself as part of a deployment.



If you want to only deploy the DB artifact (cleanup.sql), click in the "1 App Server" column to deselect (disable the selection of) the Config and Heatclinic components (based on the `env.sh` and `mycompany.war` artifacts), and click **OK**.



The **Artifacts** field now looks like this for a partial run:

Other Deployment Strategies

Deployment strategies can be named differently, but they all serve the core purpose of reducing downtime and risk. While modeling such a deployment strategy in ElectricFlow, the best practice is to ask a question–does this strategy require a single environment or multiple environments logically? Based on that answer, you can choose to model using a rolling deployment style, blue/green style, or even a combination. ElectricFlow allows the flexibility to model most of these scenarios.

Rolling Deployment Use Case

Rolling Deployment is one of the core strengths of the ElectricFlow. You can develop your application model independent of the deployment strategy and the environment model. When you model an environment, you can enable rolling deployment. You can also enable rolling deployment on an environment by selecting the rolling deployment option when you deploy an application.

In the list of application run details, ElectricFlow filters out any component process step that has no resource to execute in a particular phase of the rolling deployment. (The platform job details page continues to show all steps in the process.)

This example shows how an e-commerce company uses the rolling deployment strategy to deploy its shopping cart application to production. The shopping cart application consists of three tiers: the Web, App, and DB tiers. The company wants to deploy this application to the PROD environment in four phases:

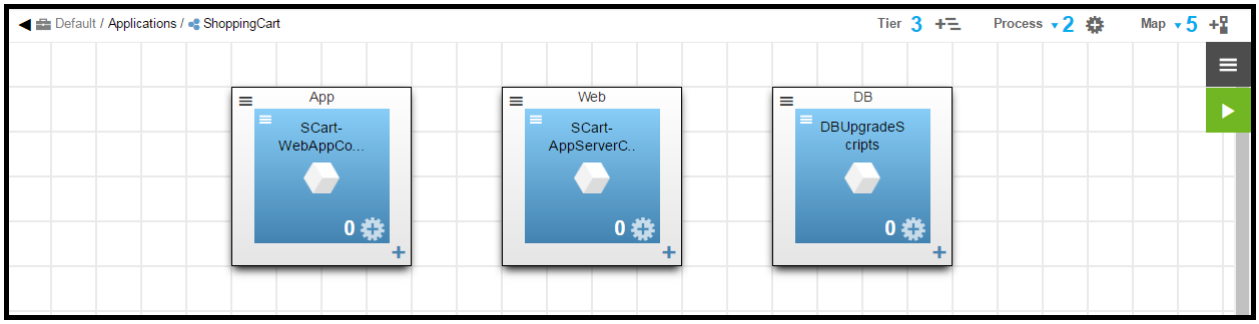
- First phase: Deploy only to the DB tier and verify the DB upgrade.
- Second phase: Deploy to a few resources in the App and Web tiers and verify that the application was deployed successfully.
- Third phase: Deploy to a few more resources in the App tier based on some rules.
- Fourth phase: Deploy to the remaining resources.

Here are the overall steps for modeling a rolling deployment for this use case:

1. Create the "ShoppingCart" application with three tiers that map to the corresponding components
2. Create an environment named PROD with tiers matching applications tiers
3. Configure the PROD environment to support rolling deployment
4. Define the rolling deployment phases
5. Run the application on the PROD environment using rolling deployment
6. View the application deployments details

Creating the "ShoppingCart" Application

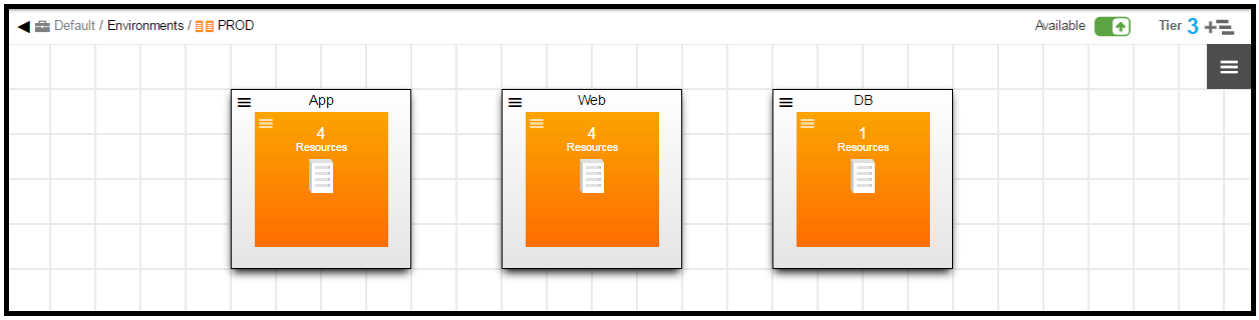
Create the "ShoppingCart" application with three tiers that map to the following components.



Note: You can develop your application model based only on business requirements without special considerations of the deployment strategy.

Creating the PROD Environment

Create an environment named PROD with tiers corresponding to the applications tiers.

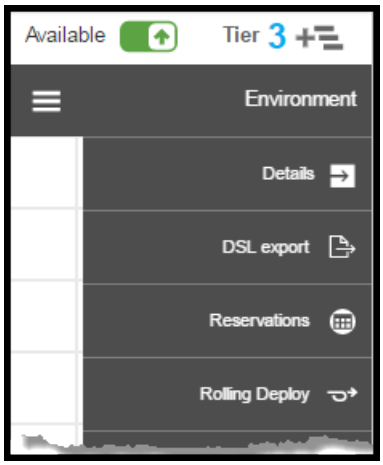


Configuring the PROD Environment

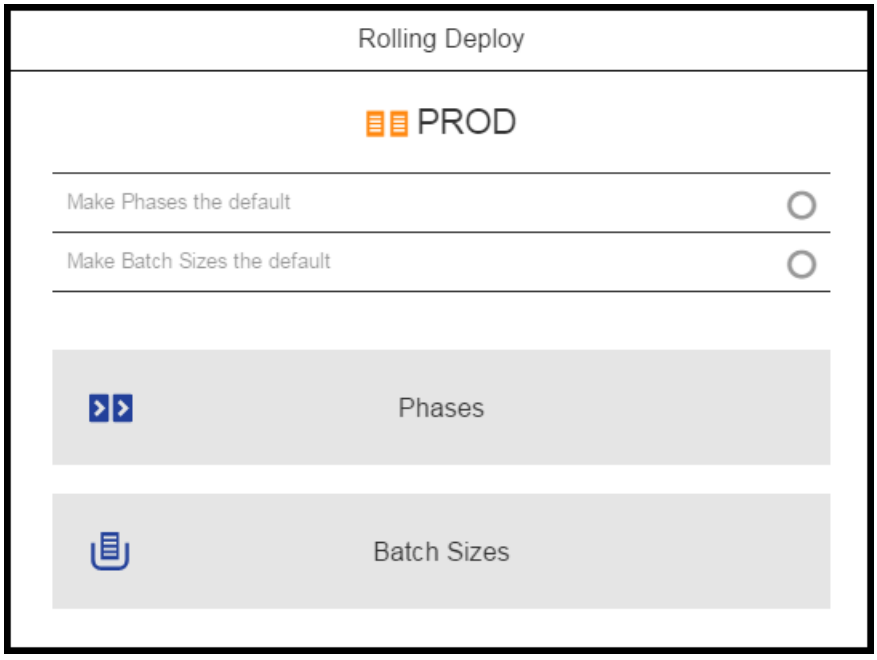
Configure the PROD environment to support rolling deployment.

To enable rolling deployment on the environment:

Click  in the upper right corner of the Environment Editor, and select **Rolling Deploy**.



The Rolling Deploy dialog box opens.



There are two rolling deployment strategies that are supported out of the box:

- Phasing: **Phases** are the preferred option when you want to control the resources and specify the order in which they are used.
- Batching: **Batch Sizes** are the preferred option when you do not need to control the resources. The resources are selected at random based on the specified batch size.

You can configure both these strategies on the environment, but only one can be active at a time

Defining the Rolling Deployment Phases

In this example, the phasing strategy will be used during the rolling deployment.

What you need to know about phases:

- Phases are ordered, and rolling deployment relies on this phase order.
- There are three types of phases:
    - Tagged: Tagged phases are useful when you want to explicitly control the resources used during a rolling deployment
    - Expression: Expression is useful when you want to pick resources for a phase dynamically at run time using conditions. This is done using JavaScript expressions. This is useful when you do not want to tag individual resources and instead want to apply an expression to pick the resources.
    - Broadcast: This type of phase can be used where you want the deployment to run on all the unused resources in environment.

      While not mandatory, the broadcast phase will most likely be the last phase in the phase order. The broadcast phase combined with the expression will provide flexibility during the rolling deployment.

      There can be only one broadcast phase for an environment.

In this example:

- PHASE1 is a tagged phase.
- PHASE2 is a tagged phase.
- PHASE3-EXP is an expression phase.
- PHASE3-BC is the broadcast phase.

To add phases:

1. In the Rolling Deploy dialog box, click **Phases** to use phasing as the rolling deployment strategy.
2. In the Phases dialog box, click **Add+** in the upper right corner to add a phase.
3. Enter the name of the first phase (PHASE1) and click **Save**.
4. Enter the name of the second phase (PHASE2) and click **Save**.
5. Enter the name of the third phase (PHASE3-EXP) and click **Save**.
6. Enter the name of the fourth phase (PHASE-BC) and click **Save**.

To assign resources to the phases

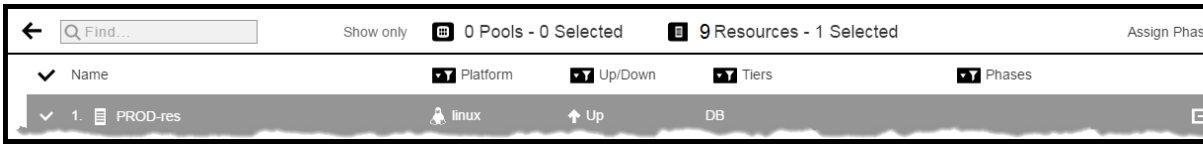
- In PHASE1, assign the only resource in the DB tier to PHASE1.
    1. Select **PHASE1**.

    2. Click **Assign Phases to this Environment Resources**.

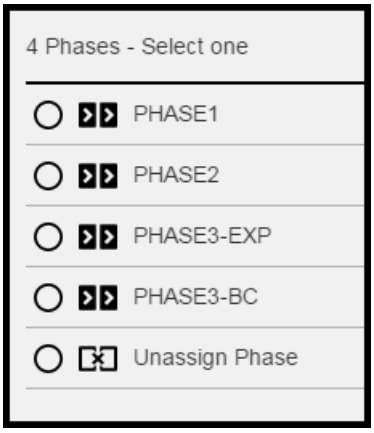
       A dialog box showing all the resources across all the tiers opens.

    3. Select the resources that you want for this phase, and then click **Assign Phase** in the upper right corner.

       For PHASE1, select the DB resource:



4. Select **PHASE1**.

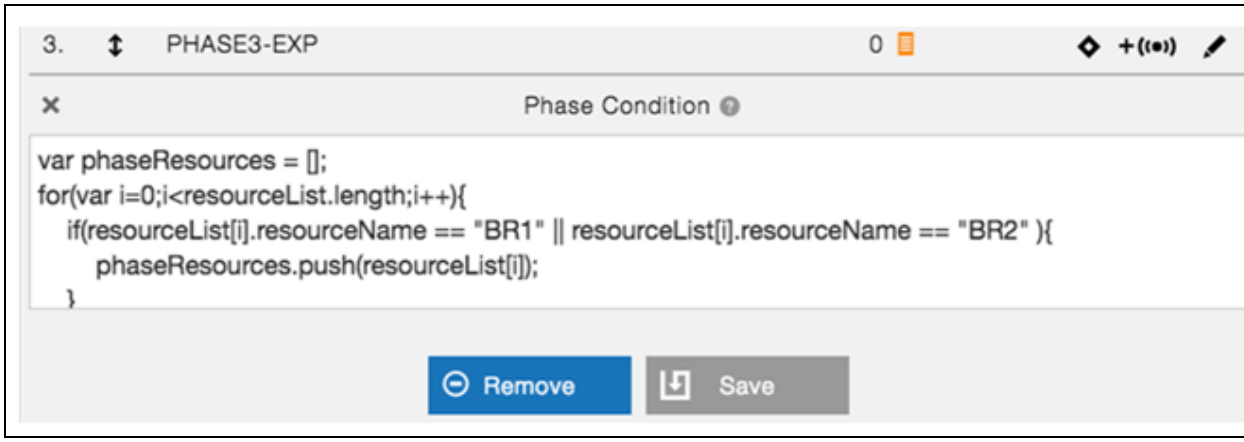


5. Click **OK**.

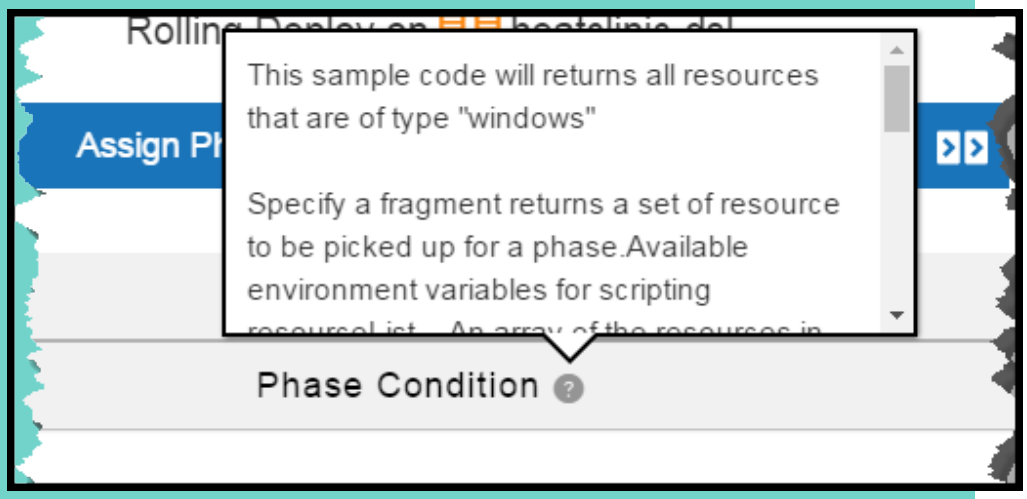
- In PHASE2, repeat the same steps described for PHASE1 and assign a few resources from the App and Web tiers to PHASE2.

- For PHASE3-EXP, enter an expression to deploy a few more resources in the App tier based on the some rules.

  - Select **PHASE3-EXP** and click to define the phase with an expression phrase.
  - Enter the expression phrase.

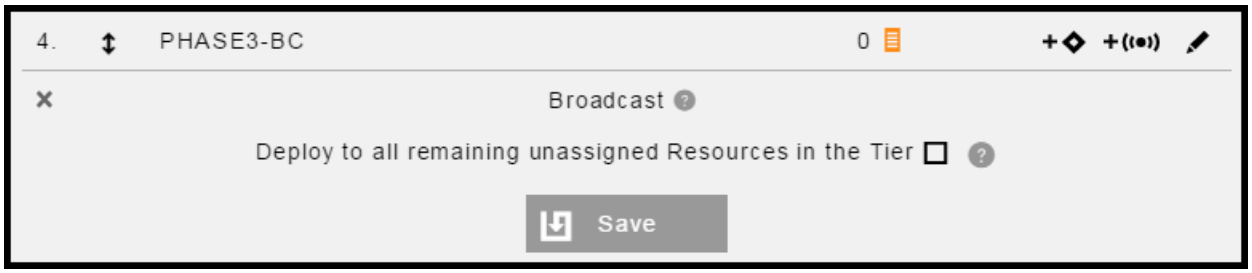


Note: When you click **?**, a pop-up window opens and shows sample code for phase conditions.



  - Click **Save**.

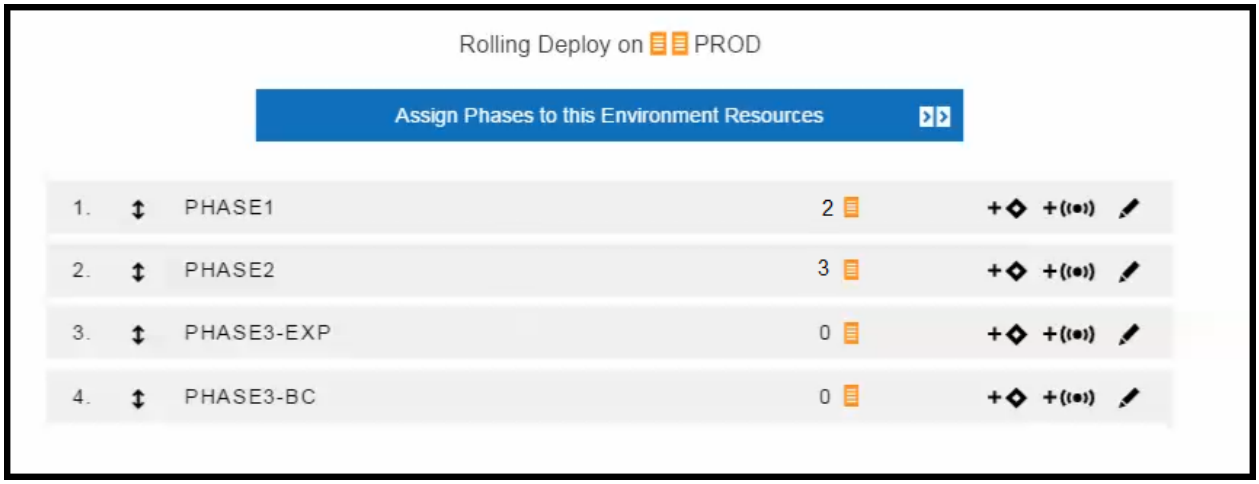
- For PHASE3-BC, click  to make this a broadcast phase to deploy to the remaining resources.



- Click **Save** to save the definition.

Tip: You can name the phases in your rolling deployment any name you want, such as ORANGE, YELLOW, EAST, WEST, and so on. What is important when defining the phases is the order and type of the phases.

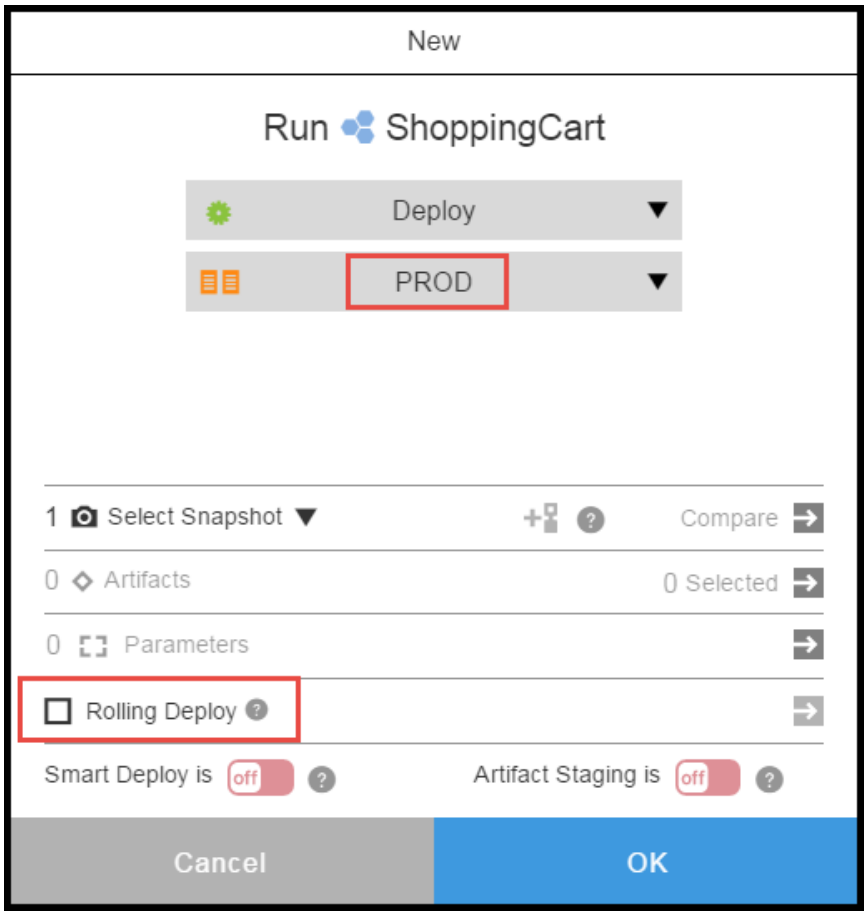
At this point, rolling deployment is enabled on the environment with phasing as the default strategy, and phases are defined as follows:



Note: This is an example setup explaining the available options. It is not necessary to use all the different types of phases. For your deployment, choose the phase types that apply to your use case.

Running the Application on the PROD Environment Using Rolling Deployment

When you are ready to run the application, go to the dialog box to set the runtime settings for the deployment. When you select the PROD environment, the option to enable Rolling Deployment is available because rolling deployment was enabled on the PROD environment. If you select another environment where Rolling Deployment is not enabled, the rolling deploy option will not be available. This way, the same application process can automatically behave in different ways based on the chosen environment, giving you the benefit of the *model-driven* approach.



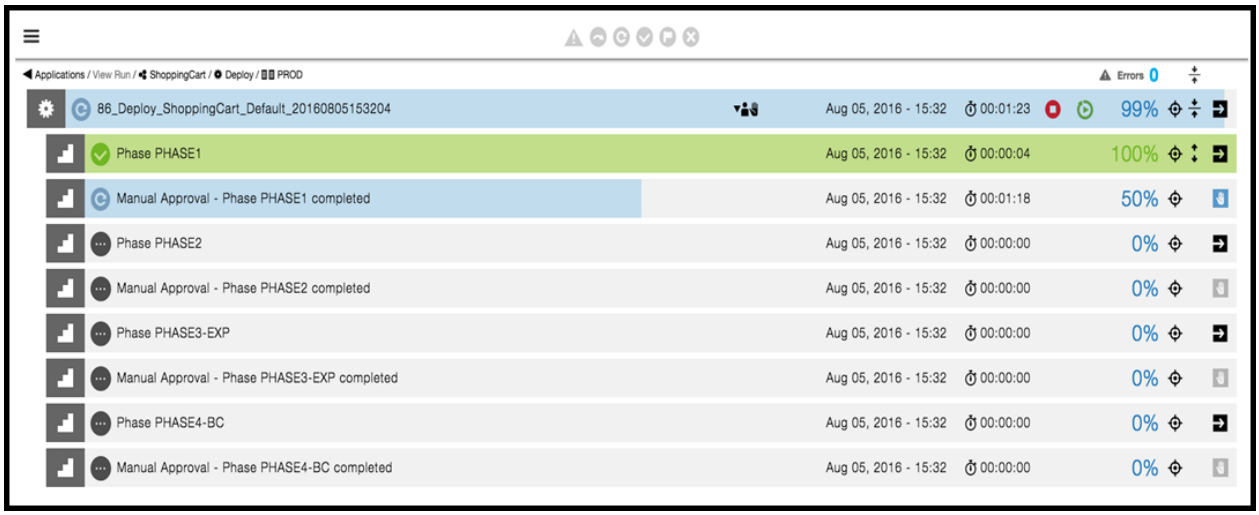
Rolling deployment is now enabled.

Clicking  shows the rolling deployment details.

You can enable or disable phases and to insert a manual step after each phase. When you click **Insert Manual Step after each Phase**, the options to run the manual step and the field to assign users who can perform the step are available.

Viewing the Application Deployment Details

Once the rolling deployment starts on the PROD environment, you can view the job details to see the breakdown and progress of the deployment. You will notice the job step breakdown corresponding to PHASE1, PHASE2, PHASE3-EXP, and PHASE3-BC. You will also see the automatically inserted manual step after each phase.



Any phase that is skipped because there is no resource associated with it is not shown.

You can also use property references to access the current rolling deployment type or the iteration that is in progress:

- To return the rolling deployment type (`phase` or `batch`), use `$[/myJob/rollingDeployType]`.
- To return the rolling deployment iteration that is in progress (such as `PHASE1`, `PHASE2`, and so on for batch rolling deployment strategy or phase name for phase strategy), use `$[/myJob/currentRollingDeployIteration]`.

Summary

- The deployment ran in four phases.
- In the first phase, deploy only to the DB tier and verify the DB upgrade.
- In the second phase, deploy to a few resources in the App and Web tiers and verify that the application was deployed successfully.
- In the third phase, deploy to a few more resources in the App tier based on some rules.
- In the fourth phase, deploy the remaining resources.

Blue/Green Deployment Use Case

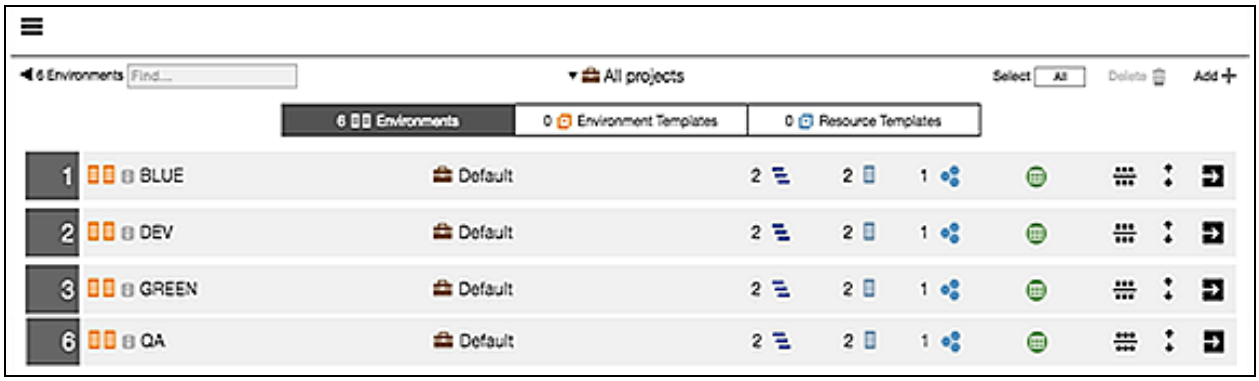
This example shows how an e-commerce company can use the blue/green deployment strategy to deploy its shopping cart application to production. The current deployment process requires that the application is tested in development and QA environments before it is deployed to the production environment where the blue/green strategy is used.

Here are the overall steps for modeling a blue/green deployment for this use case:

1. Create four environments
2. Map the environments to the ShoppingCart application
3. Create a pipeline with three stages
4. Create three tasks for the PROD stage
5. Run the pipeline

Creating Four Environments (DEV, QA, BLUE, and GREEN)

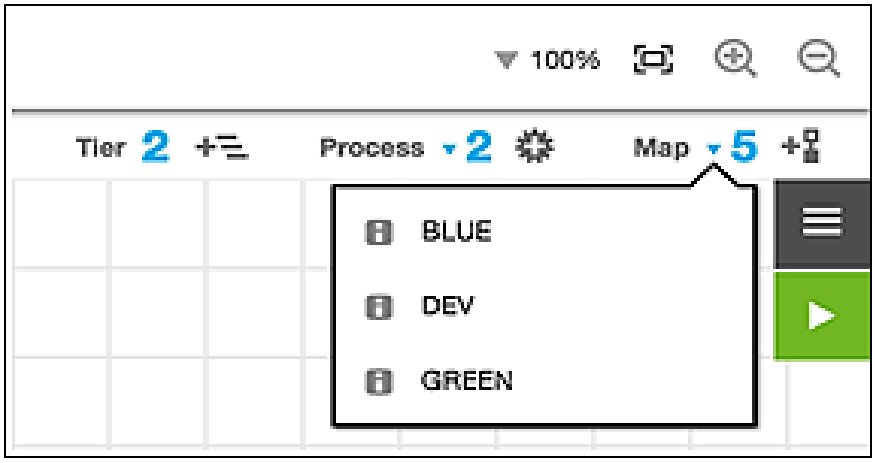
Create four environments named DEV, QA, BLUE, and GREEN.



Note: DEV, QA, BLUE, and GREEN are names for the environments in this use case. You can use any names you want for them as long as it is clear how they map to BLUE and GREEN.

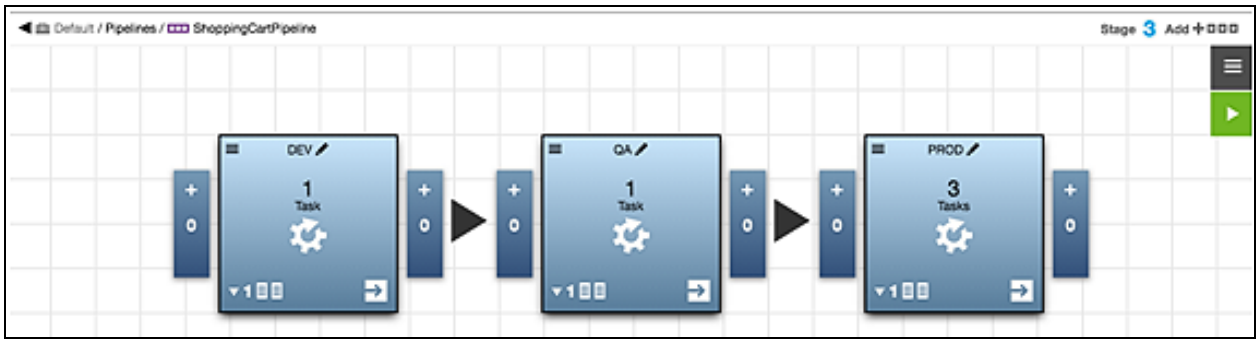
Mapping the Environments to the ShoppingCart Application

Select the ShoppingCart application, and then map the application tiers to the corresponding environment tiers in DEV, QA, BLUE, and GREEN.



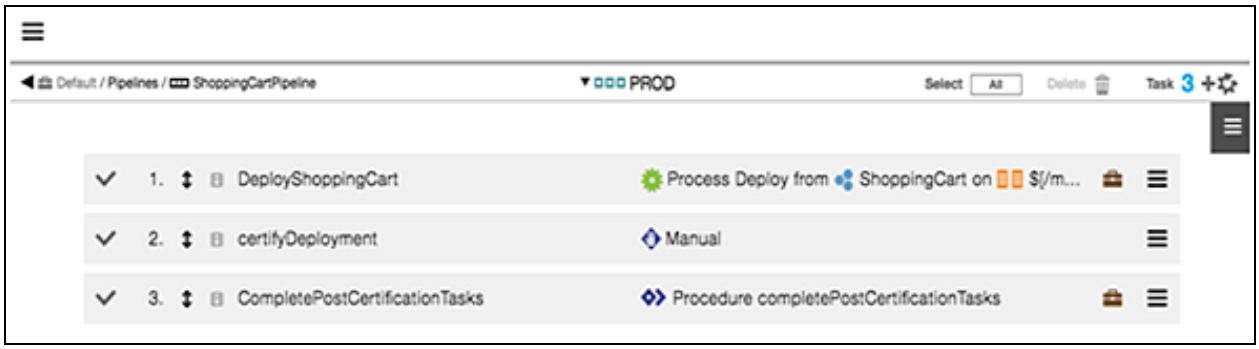
Creating a Pipeline with Three Stages

Create a pipeline with three stages named DEV, QA and PROD.



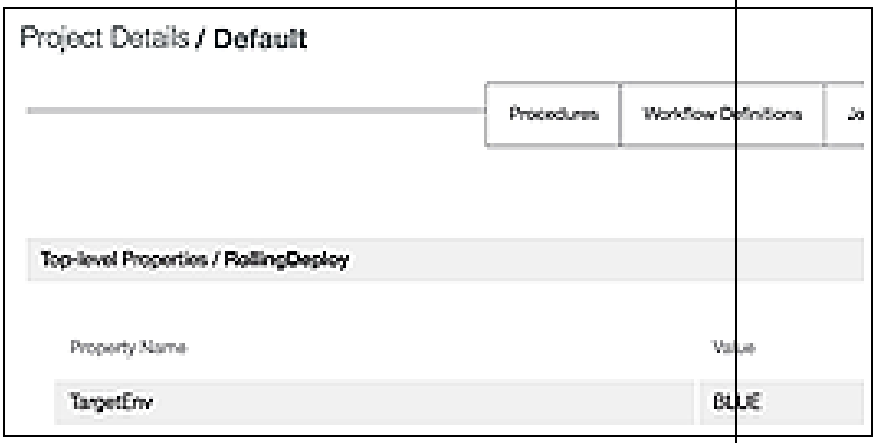
Creating Three Tasks for the PROD Stage

In the PROD stage, create three tasks:



These are the task details:

Task Num-ber	Task Name	Description
1	DeployShoppingCart	This is another process task similar to what is in the DEV and QA stages that the environment name is parameterized. The system will automatically switch deployments between the BLUE and GREEN environments.
2	CertifyDeployment	This is a manual step to review and certify the deployment.

Task Num-ber	Task Name	Description
3	completePostCertificationTasks	This is an automation task to run any post certification tasks such as updating the load balancer and switching the environment in preparation for the next pipeline run (for example, update the *TargetEnv* parameter) 

Running the Pipeline

In a blue/green deployment, before you run the pipeline, the GREEN environment is serving all of the production traffic, and the BLUE environment is ready to be upgraded to the new software version.

Before starting the deployment, set the `RollingDeploy/TargetEnv` property on the project to the first environment to which you want to deploy. `RollingDeploy/TargetEnv` will be set to `BLUE` because the GREEN environment is currently catering to end-user traffic.

When the pipeline starts:

1. The pipeline will start at the DEV stage and continue to the PROD stage.
2. Once the deployment reaches the PROD stage, the application process task will deploy to the BLUE environment based on the property value `$[/myProject/RollingDeploy/TargetEnv]`, which was set to `BLUE` at the beginning of the run.
3. Once the deployment task has completed, there is a manual step to functionally verify the deployment.

   If there was a need for some automated testing, those tasks can easily be added to the pipeline stage.
4. Once the deployment completes successfully, switch the user traffic to the BLUE servers, and update the `RollingDeploy/TargetEnv` property to `GREEN`.

   Setting the property value to `GREEN` will ensure that the next deployment will happen to the GREEN environment while the BLUE environment serves all of the production traffic.

Canary Deployment Use Case

The purpose of a canary deployment is to deploy an application to a small set of servers for validation by a subset of users to reduce the risk of a new version rollout. After user validation, the application is rolled out to a larger set of users.

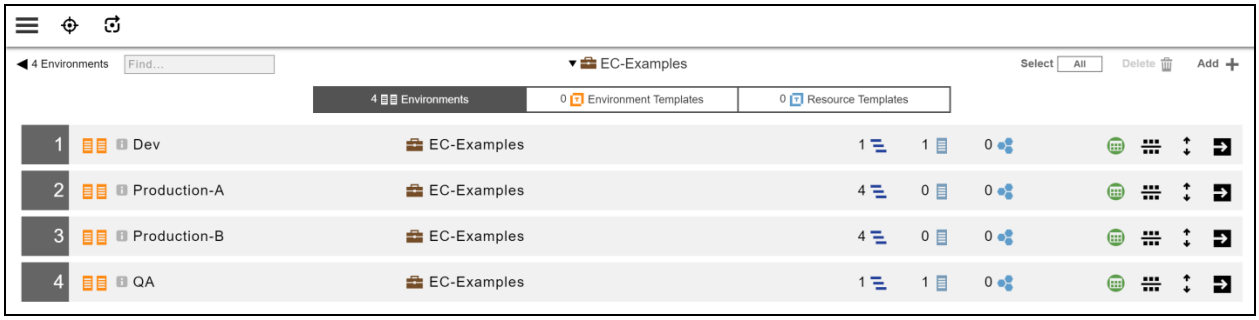
This example shows how an e-commerce company can use canary deployment to deploy its Shopping Cart application to production. In this example, the release process requires that the application is tested in the development and QA environments before the application is deployed to the production environment where the canary strategy is used.

Here are the overall steps for modeling a canary deployment for this use case:

- Create environments for the application
- Map environments to the application
- Create a pipeline with three stages
- Create multiple tasks for the Production stage
- Run the pipeline

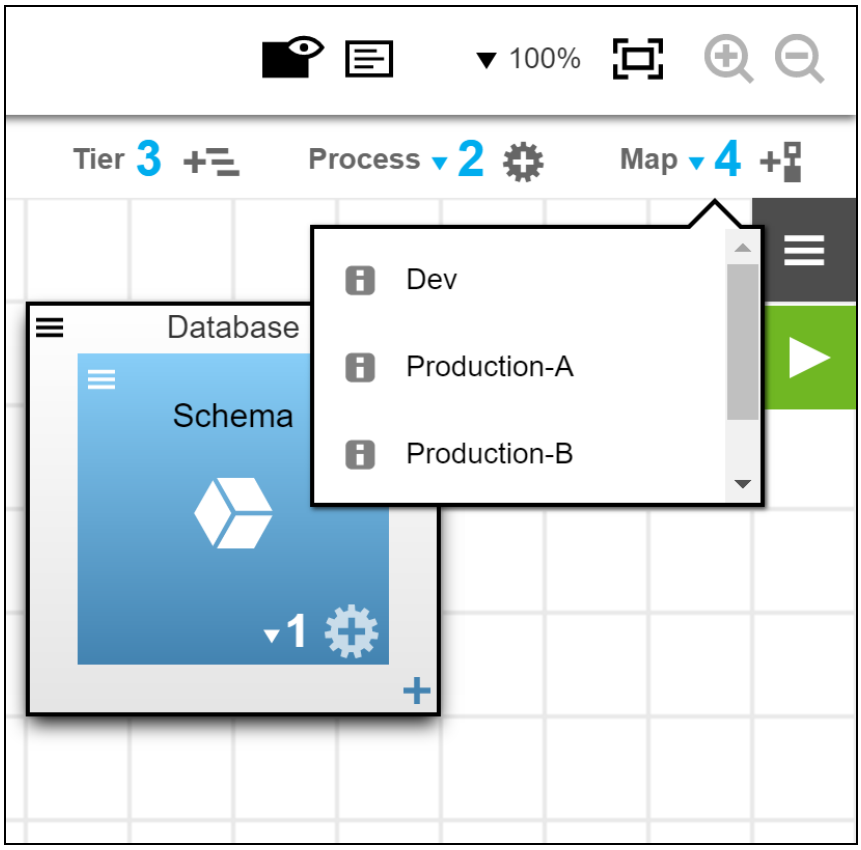
Create Four Environments (Dev, QA, Production—A, and Production—B)

Create four environments named Dev, QA, Production—A, and Production—B.



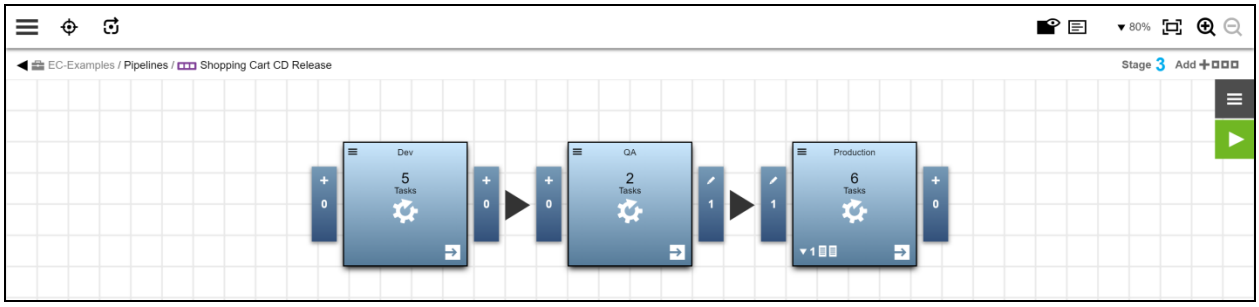
Map the Environments to the Application

Select the application and map the application tiers to the corresponding environment tiers in Dev, QA, Production—A, and Production—B.



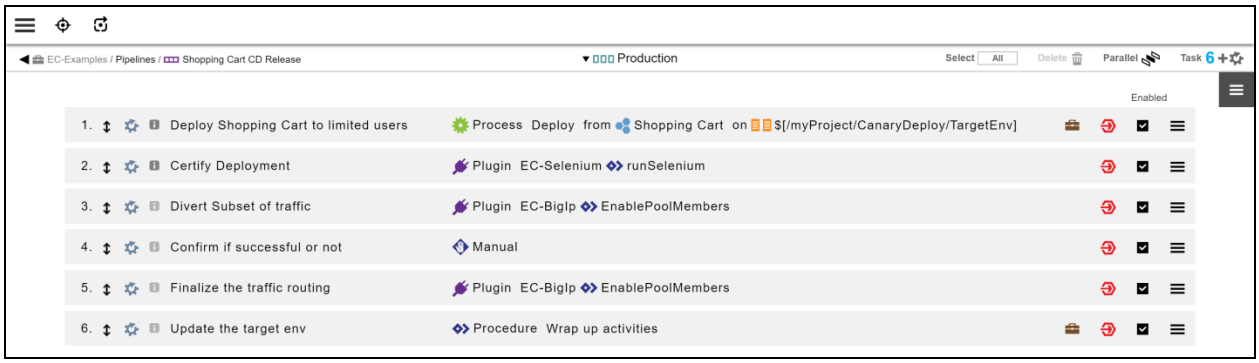
Create a Pipeline with Three Stages

Create a pipeline with stages named Dev, QA, and Production.

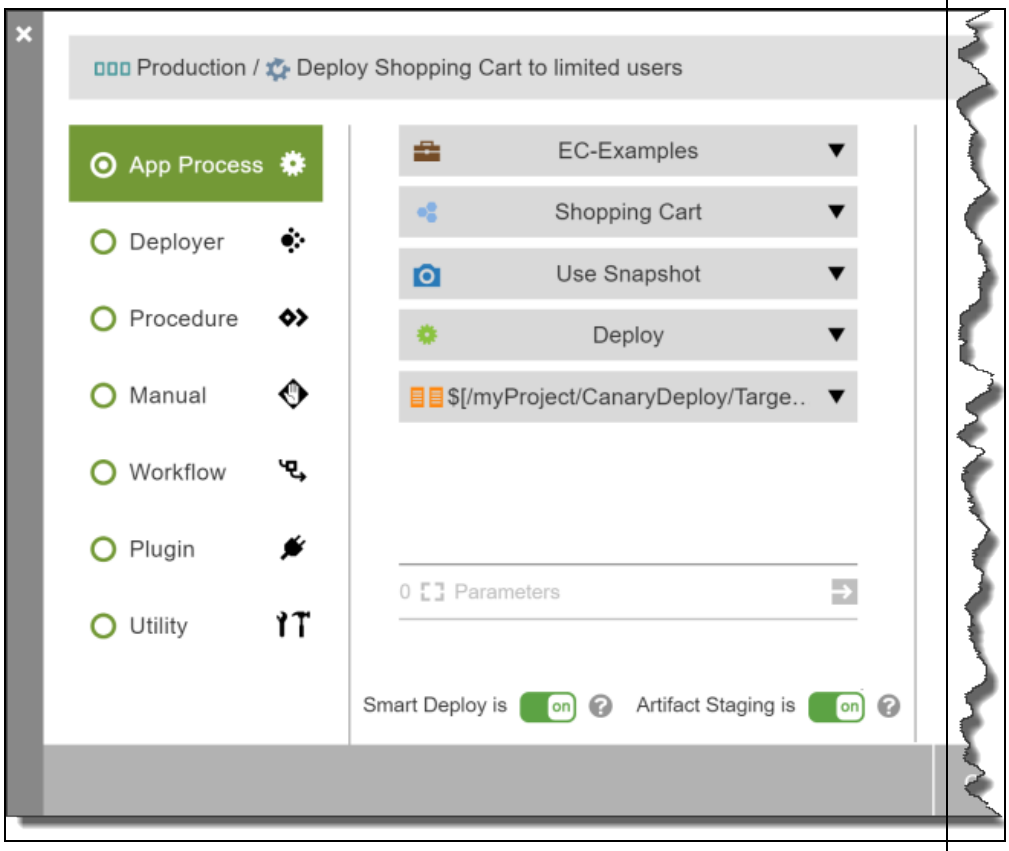


Create Multiple Tasks for the Production Stage

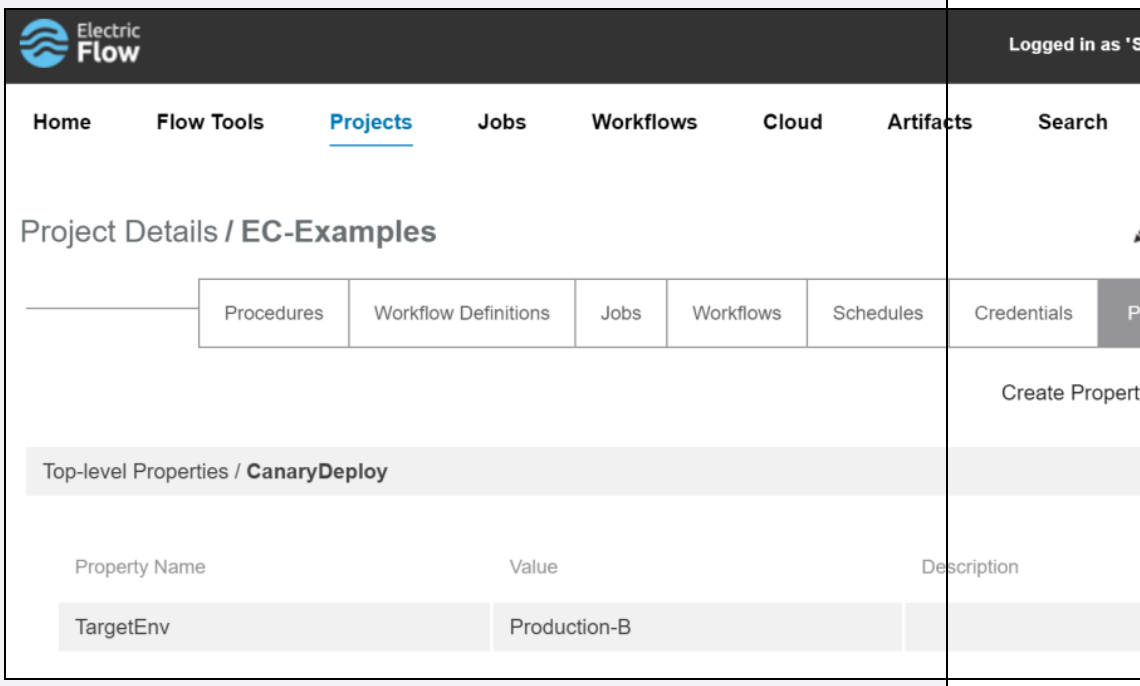
In the Production stage, create tasks as follows:



These are the task details:

Task Num-ber	Task Name	Description
1	Deploy Shopping Cart to limited users	This task deploys the Shopping Cart application to environments in the production stage. This uses a deployment process where the target environment is parameterized using a `$[CanaryDeploy/TargetEnv]` property that is set at the project level. The system automatically switches the target environments for each new release. For example, if a new version will be deployed to Production—A to test with a limited user base, the target environment would be Production—A. 
2	Certify Deployment	This task can run any test automation tool using plugins included with ElectricFlow or can even be a manual step to review and certify the deployment. If needed, you can add steps to certify the deployment.
3	Divert Subset of traffic	This is an automation task to interact with the load balancer to divert a subset of end user traffic to this just-deployed version. This task can use the Electric Cloud F5 plugin or can even point to an existing script or a custom automation.

Task Num-ber	Task Name	Description
4	Confirm if success ful or not	This can be a manual or automated task to confirm if the rollout to the subset of end users is successful.
5	Finalize the traffic routing	This is an automated task to interact with the load balancer to divert the remaining traffic to • the environment with new version or if the canary rollout was successful *or* • the environment with the current version if the canary rollout was unsuccessful This task can use the F5 plugin or can even point to an existing script or custom automation.
6	Update the target env	This is a wrapup automated task to ensure that the `$[CanaryDeploy/TargetEnv]` property is set appropriately depending on the success or failure of the canary deployment.



Running the Pipeline

In a canary deployment, before you run the pipeline, either the Production—A or the Production—B environment is serving all production traffic. Let's assume that Production—B is serving the production

traffic currently.

Before starting the rollout, ensure that the `$[CanaryDeploy/TargetEnv]` property on the project is set to value `Production—A`. When the pipeline starts:

- The pipeline runs the Dev stage and continues to the Production stage.
- Once the deployment reaches the Production stage, the application process task deploys to the Production—A environment based on the `[/myProject/CanaryDeploy/TargetEnv]` property value, which was set to Production—A at the start of the run.
- Once the deployment task completes, the deployment is certified using a test automation tool or manual testing. This task (number 2 above) ensures that the new version is functionally verified to work in the Production—A environment.
- Once the deployment is certified, a subset of traffic is automatically diverted to the Production—A environment as task 3 updates the load balancer configurations. The application is now in a canary state in which a fraction of users are tested on the newer version, whereas the larger portion still uses the safe current version.
- Once sufficient time is allowed and testing is done, step 4 asks for user confirmation of success or failure of the new version.
- If the new version in the Production—A environment is working satisfactorily, all traffic is diverted to it automatically. But if the user input in task 4 indicates failure, all traffic is routed back to the Production—B environment. (If you want to gradually move traffic to the newer version instead of moving the entire traffic at this time, just add additional tasks to the pipeline.)
- The system automatically updates the `$[CanaryDeploy/TargetEnv]` property to Production—B if the rollout is successful, so that the next time, canary deployment begins on it. If the rollout failed as indicated by user input in task 4, the `$[CanaryDeploy/TargetEnv]` property is kept as Production—A.

Dark Launch Deployment Use Case

Here are the overall steps to use ElectricFlow to model a Dark Launch deployment:

1. Implement feature flags
2. Create an application model for the Shopping Cart application
3. Create an application processes to deploy the entire Shopping Cart application
4. Create another application process to enable or disable feature flags
5. Create input parameters for the process
6. Deploy with feature flags initially disabled
7. Enable the features after deployment

Creating Feature Flags

Before ElectricFlow can orchestrate dark launch deployments, you must create a flag for each feature that you want to enable or disable. Feature flags are a software development best practice for gating functionality. With feature flags, you can manage the entire lifecycle of a feature.

You should add the flags to a configuration file and write your application source code to check for values in that file. Following is sample code for a new version of the `shoppingCart.war` file. It uses a configuration file named `config.properties` as shown below to control the feature flag values.

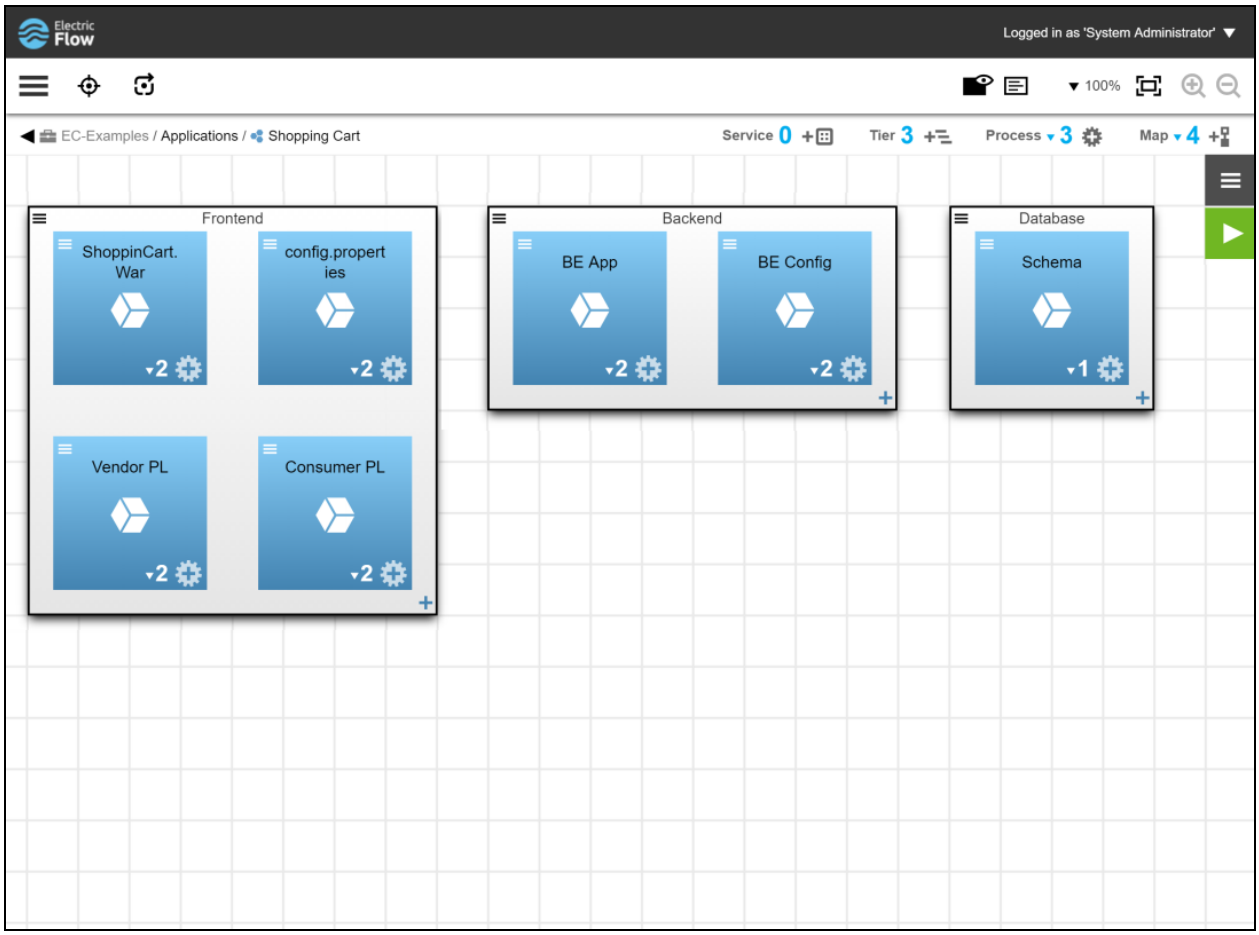
In this example, the `config.properties` file has two feature flags: `ShoppingCart_Feature1` and `ShoppingCart_Feature2`. These flags are disabled by default:

```
#
# Configuration settings
#

# This file contains config properties used by ShoppingCart.war
# These can be changed to control behavior of the application

# Feature Flag settings
ShoppingCart_Feature1 = 'Disabled'
ShoppingCart_Feature2 = 'Disabled'

#
# SSL settings
#
ShoppingCart_HTTPS_PORT=8443
ShoppingCart_KEYSTORE=conf/keystore
ShoppingCart_CRL_FILE=conf/security/crl.pem
ShoppingCart_KEY=conf/security/ca_pk.pem
ShoppingCart_CERT=conf/security/ca.pem

# Message broker settings
ShoppingCart_STOMP_PORT=61613
ShoppingCart_MQ_DATADIR=broker-data
```

Typically the `shoppingCart.war` file and its configuration file are deployed together. You publish the appropriate versions of the `shoppingCart.war` file and the `config.properties` file to an artifact repository (such as EC-Artifact, which is included with ElectricFlow).

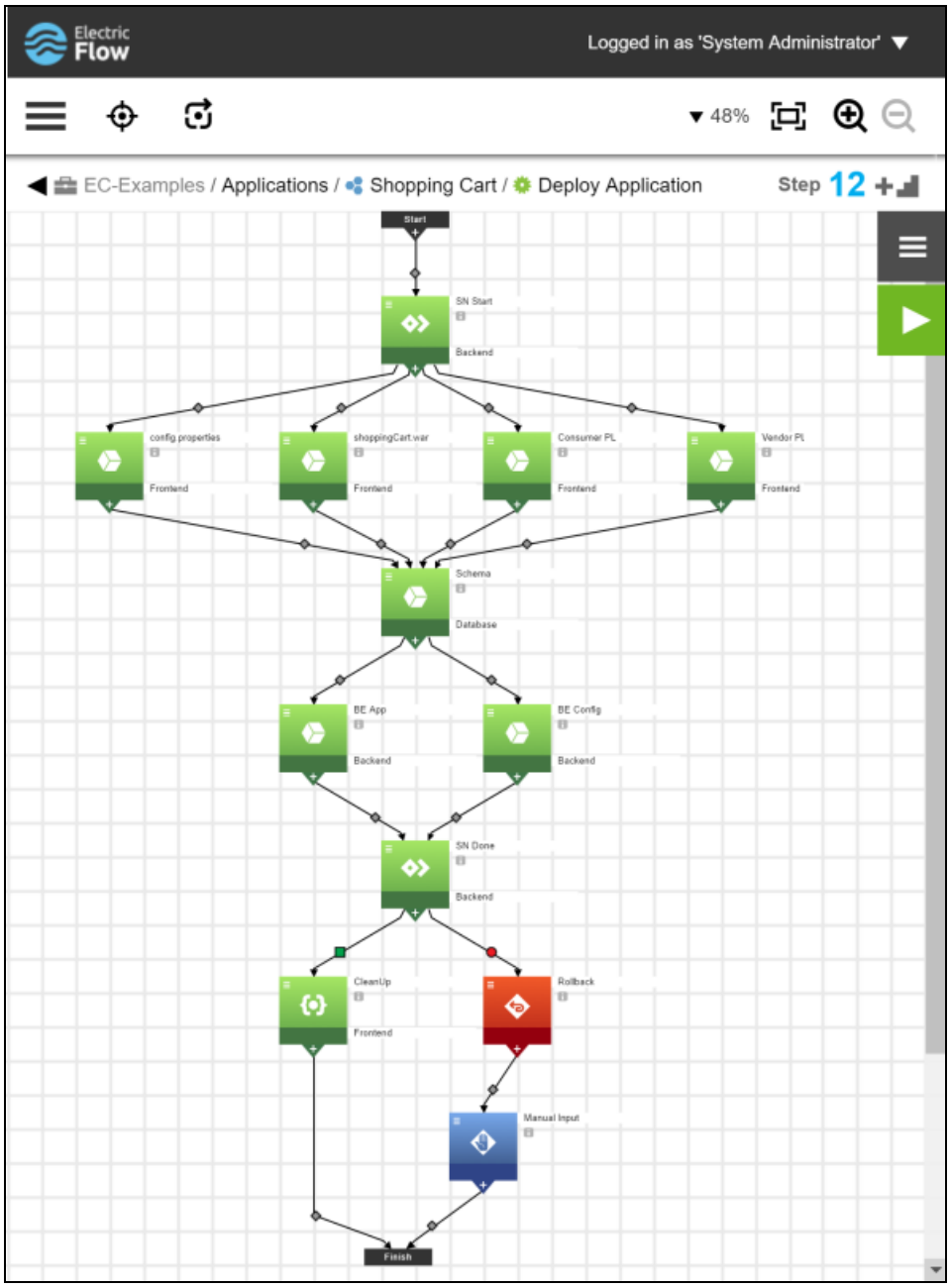
Creating an Application Model for the Shopping Cart Application

You model the Shopping Cart application so that it has components that point to the `shoppingCart.war` and `config.properties` artifacts along with other components.



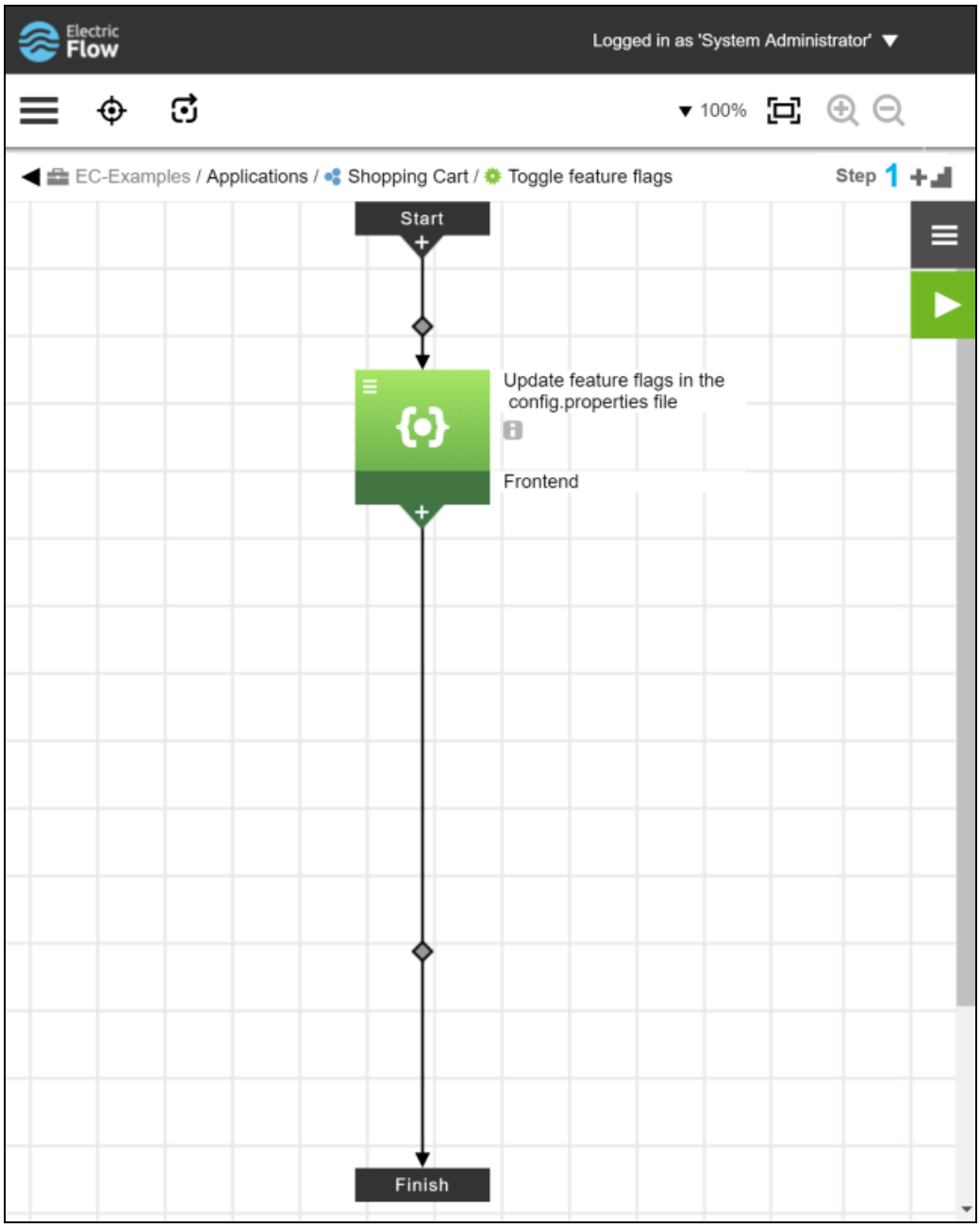
Creating an Application Process to Deploy the Entire Shopping Cart Application

This is a normal process needed to deploy any application.



Creating Another Application Process to Enable or Disable Feature Flags

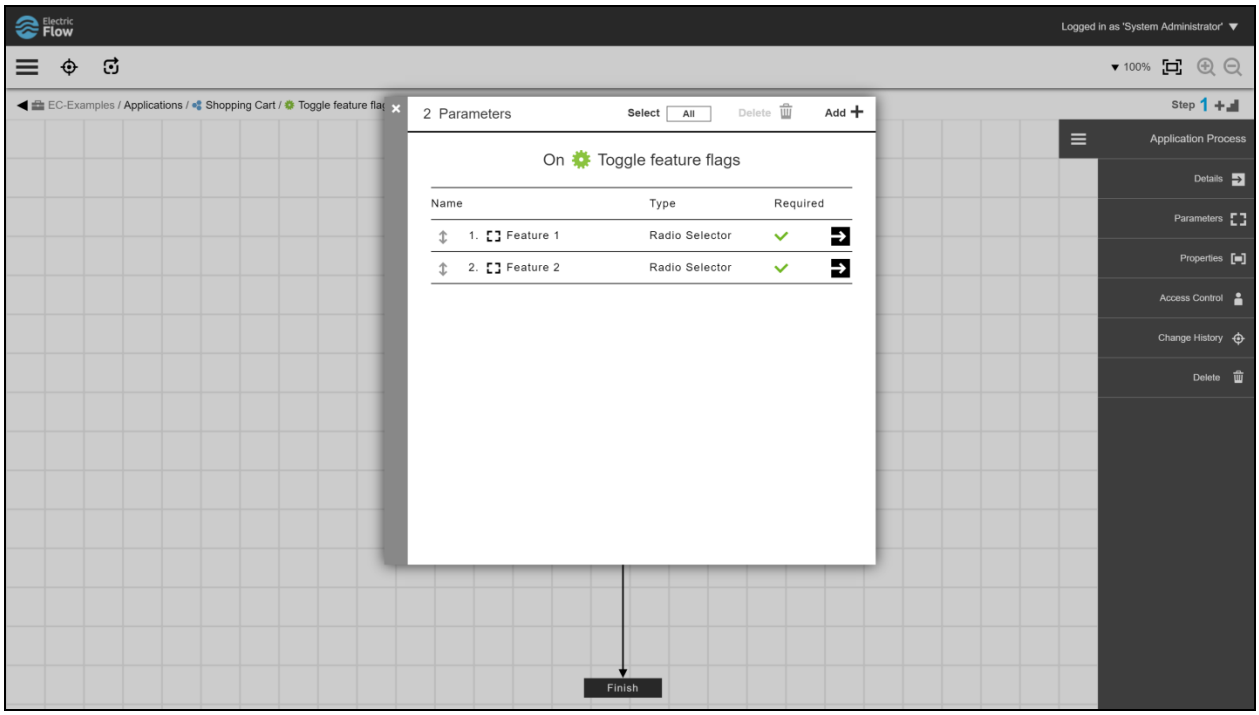
For example, create a process named "Toggle feature flags" for the Shopping Cart application:

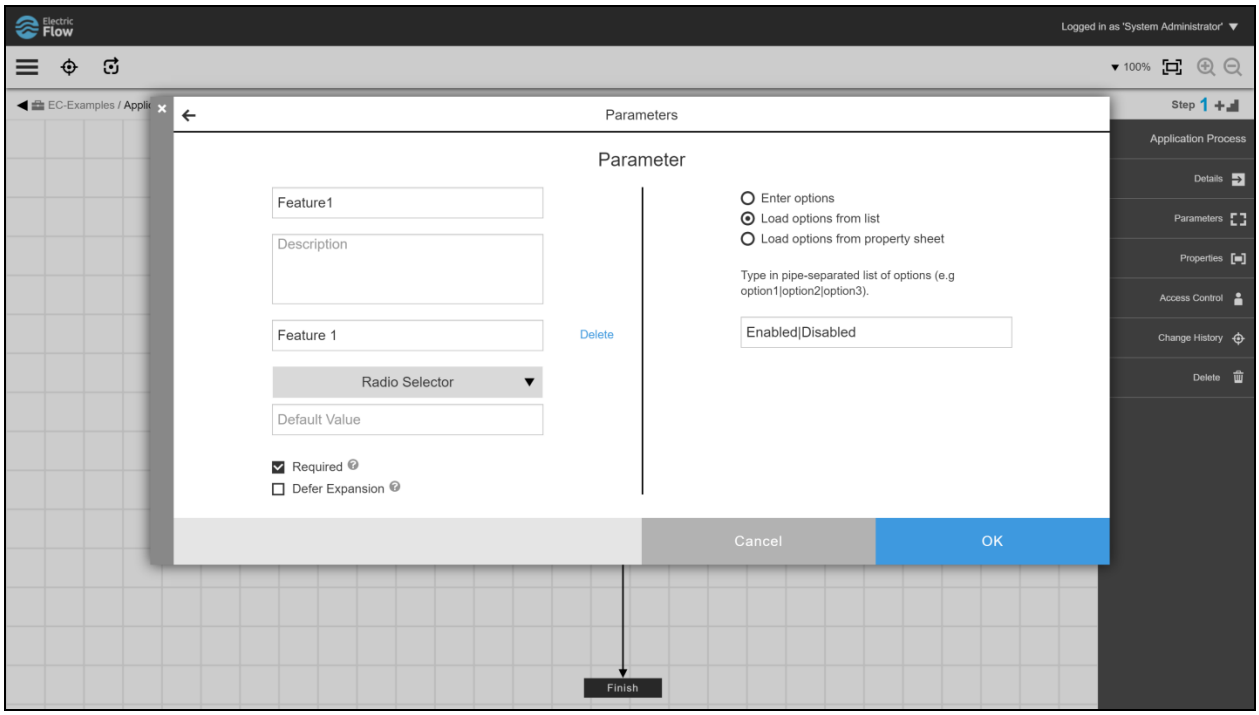


This process has a simple command step that takes end user input using the parameters defined below and then updates the `config.properties` file on each target machine. Parameters are a way to take dynamic input from end users or from other automations in ElectricFlow.

Creating Input Parameters for the Process

You create two parameters for this process. For example, create parameters named `Feature1` and `Feature2`, each with a possible value of `Enabled` or `Disabled`.

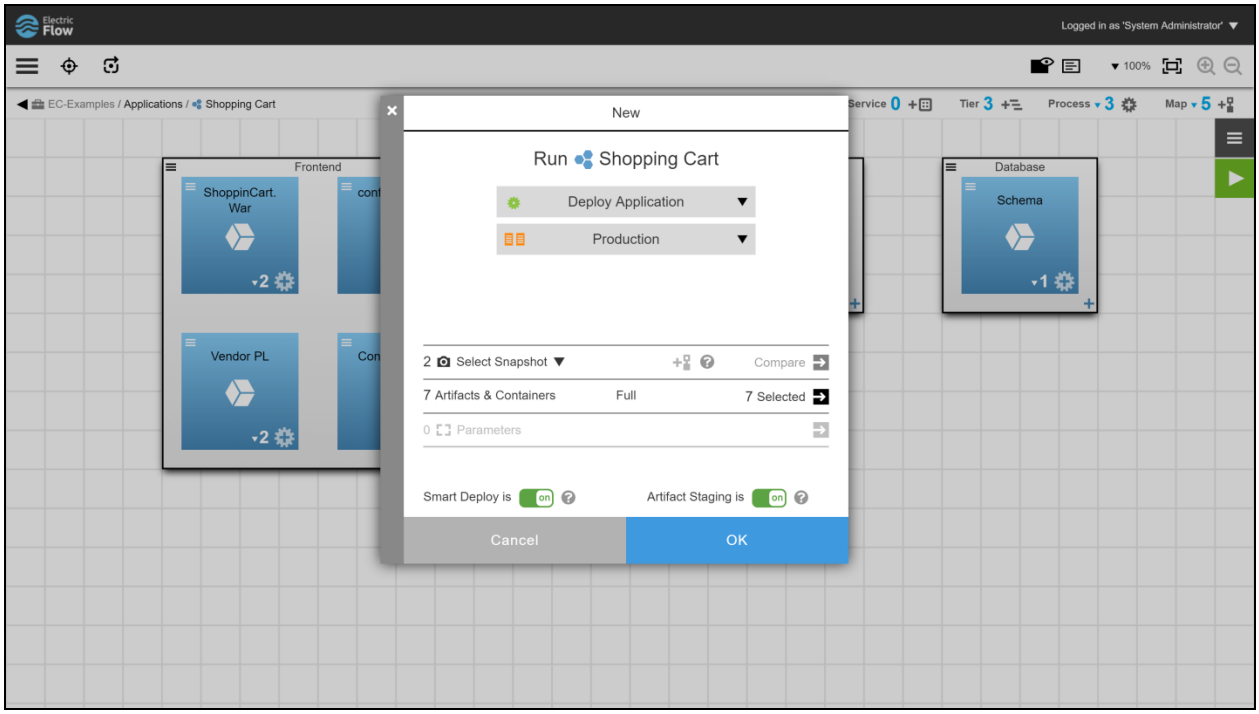




These parameters gather end user input to enable or disable feature flags. The command step in the "Toggle feature flags" process uses values from these parameters dynamically. When the process is run, the user input to enable or disable the parameters is used to update the configuration file.

Deploying with Feature Flags Initially Disabled

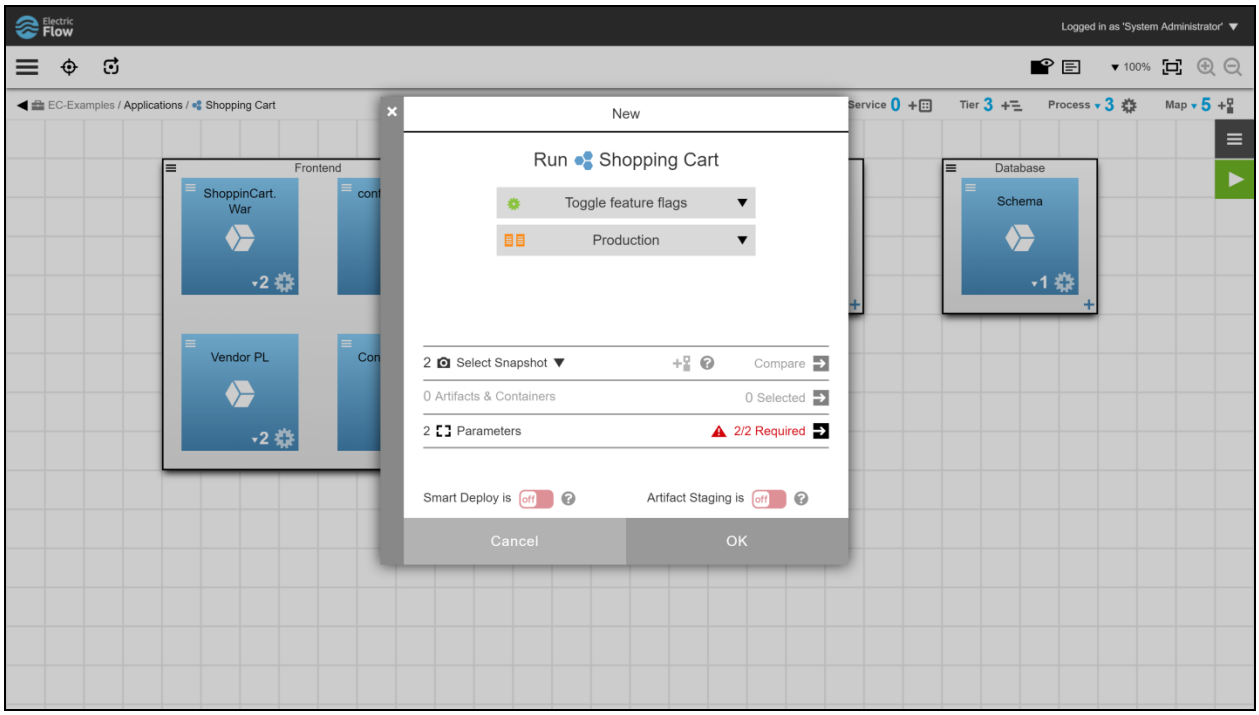
Once you model the application, its processes, and its production environment, you deploy the latest version to the production environment. By default, the features are turned off, because the default values for the `ShoppingCart_Feature1` and `ShoppingCart_Feature2` settings in the `config.properties` file are set to Disabled.

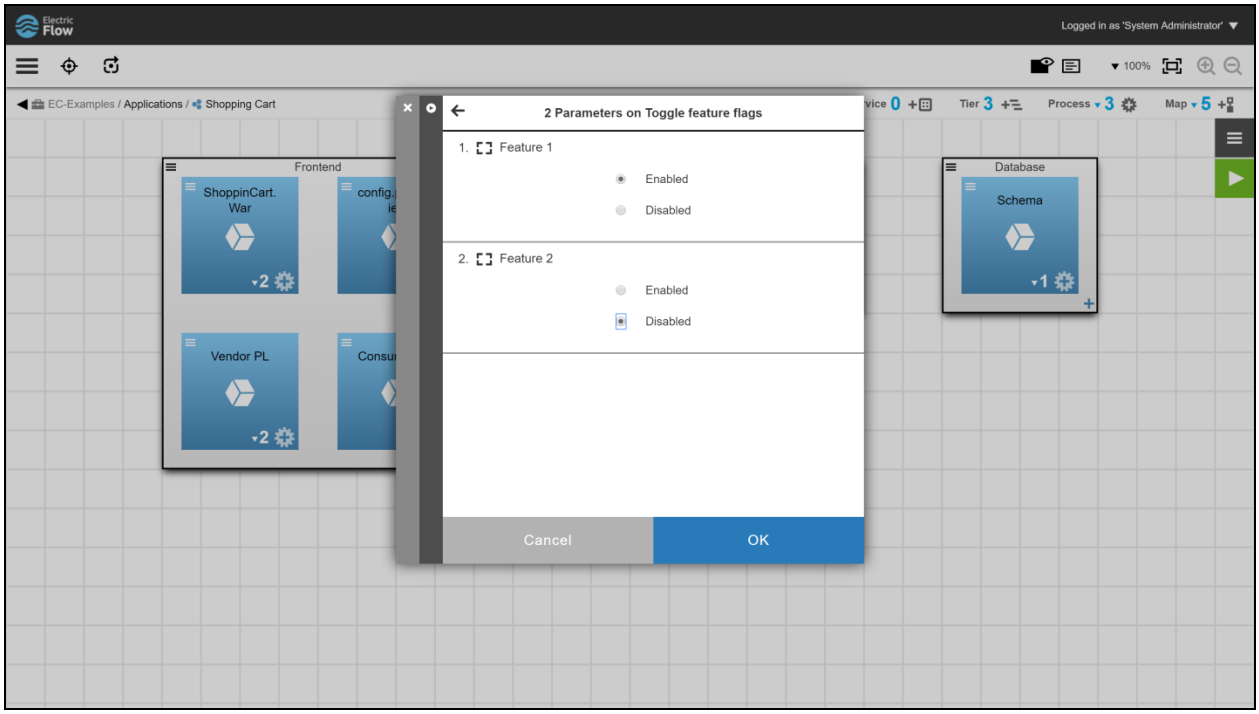


Once this deployment is done, the new features are launched darkly. This means that they are now in production, but end users cannot access them because they are turned off.

Enabling the Features After Deployment

To enable one or both features, users run the "Toggle feature flags" process. When that process is run, users are asked to select the values for the feature flags. The process takes the end user input for both parameters and updates the `config.properties` file to set the feature flags:





As shown in the images above, if the process is run, the values in the `config.properties` file are set as follows:

- `ShoppingCart_Feature1 = Enabled`
- `ShoppingCart_Feature2 = Disabled`

With the change in place, only Feature 1 is now activated, and end users can use it. Any authorized user can turn that feature off or turn the second feature on just by rerunning the "Toggle feature flags" process.

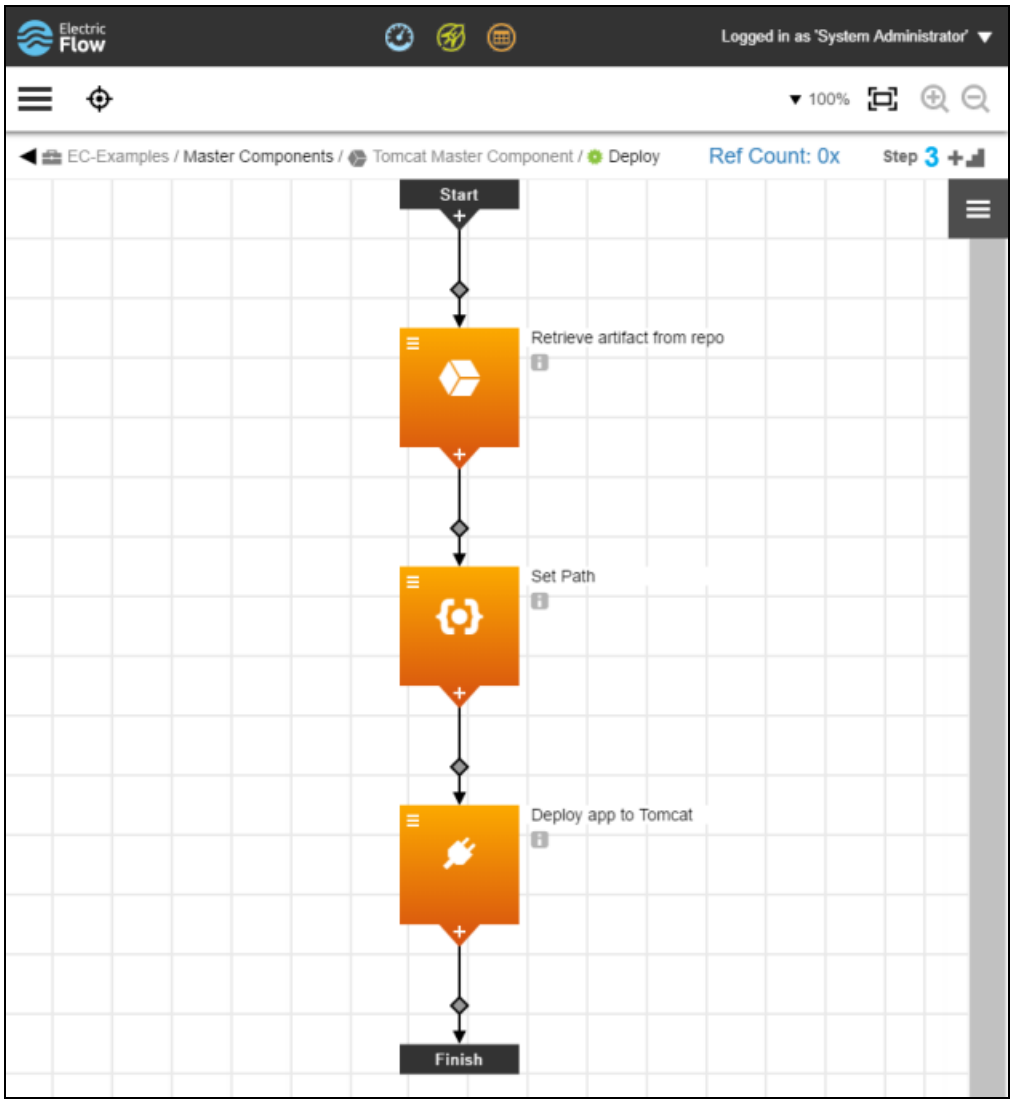
Hot Deployment Use Case

ElectricFlow lets you build your component or application processes to cater to hot deployment practices easily. Every application server plugin in ElectricFlow has functions to stop or start servers, deploy applications, and many more.

ElectricFlow lets you decide which functions to call and the order in which to call them when you build deployment processes. With hot deployment, you do not need to call the functions that 'stop' the server but instead directly call the deploy application-related function. This ensures the application updates occur in a "hot" manner. Also, you can create a process that decides hot or not-hot deployment based on a parameter, so that a single process can serve both scenarios. Following are examples of how to set up hot deployment in ElectricFlow.

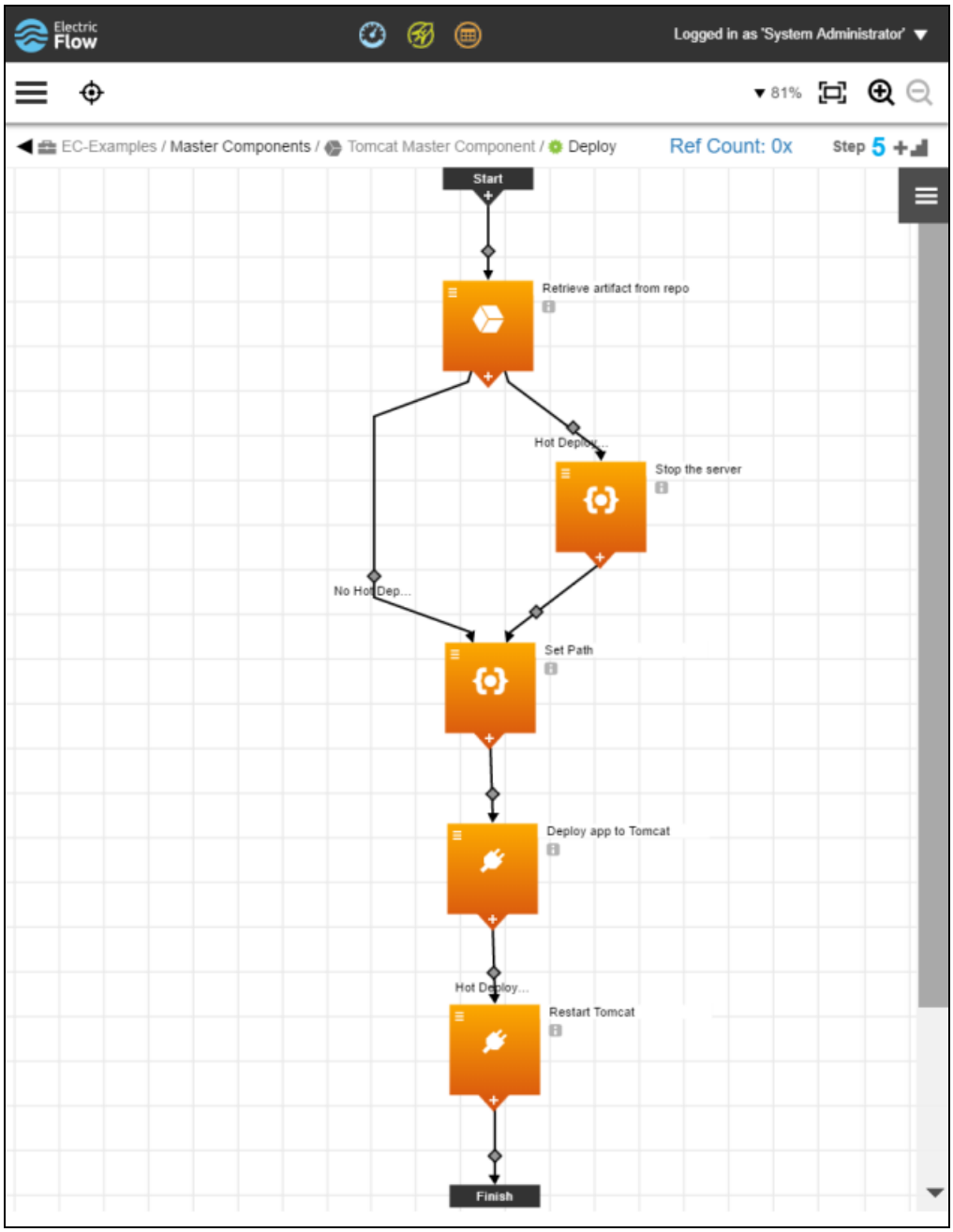
Example: Hot Deployment with a Component Process Without the Steps to 'Stop' the Server

For example, you are deploying components that are always deployed in a hot fashion to a Tomcat server. You can create a component process similar to the one in the diagram below, which contains no steps to stop or start the server. Any new version or change is deployed directly to the right path in Tomcat, and any update occurs in a hot fashion. This component is created as a master component so that it can be used in any application.



Example: Optional Hot/Not-Hot Deployment With a Parameterized Component Process

Sometimes the same master component can be used to model both hot as well as regular (not hot) deployments that require a restart. You can create a single component process that conditionally skips the stop and restart steps based on a parameter.



In this example, the steps to stop and start the Tomcat service are executed only when you do a hot deployment. If you do a regular (not hot) deployment, the Tomcat service is stopped and restarted at the right time. This user input is gathered from a process parameter.

Thus, ElectricFlow provides complete flexibility to execute hot deployments easily depending on the application server technology of choice.

Automated Environment Discovery

Automated Environment Discovery, also referred to as *auto-discovery*, makes it easier to create models using your existing resources and environments for deployment automation. You need to set up the

resources and environments before creating the deployment model and authoring processes in ElectricFlow. When deployments have large numbers of resources and environments, automated environment discovery speeds up application and microservice authoring by:

- Providing resource discovery using middleware plugins, such as EC-WebSphere, on existing resources and environments.
- Allowing the use of discovered resource configurations to accelerate process authoring and deployment automation.

You can initiate the discovery operation on resources from the following pages in the UI:

- At the platform level, go to the **Cloud** > **Resources** page.
- From the Environment Editor for a specific environment.
- From an environment tier in the Environment Editor.

Resources Page

Starting from the **Cloud** > **Resources** page:

1. Select one or more resources.

Example:

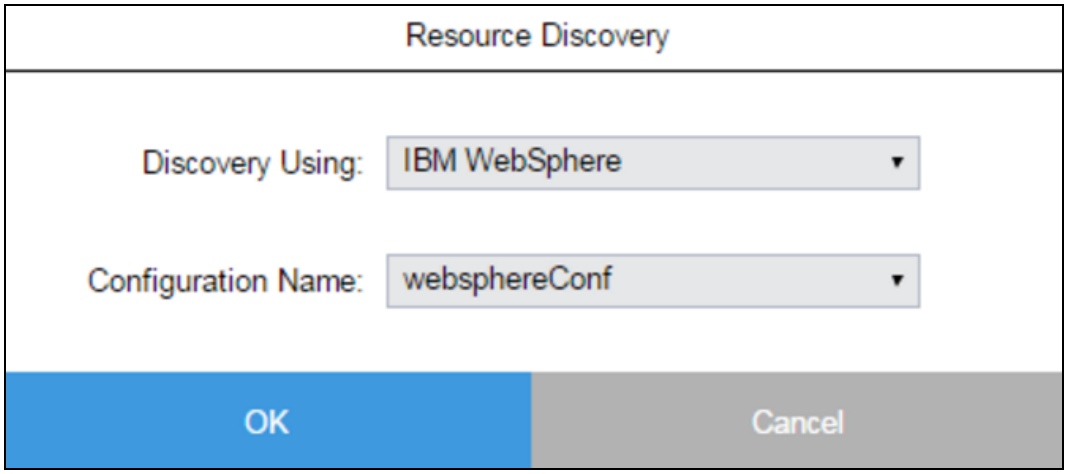
2. Click the **Resource Discovery** button ( ) in the upper right corner.

   The Resource Discovery dialog box opens.

3. In the **Discover Using** field, select the plugin to use to run the Discover procedure for the selected resources.

4. (Optional) If the plugin requires a configuration for the Discover procedure, select the configuration from the drop-down list in the **Configuration Name** field.

   ### Example:



5. Click **OK**.

If the discovery request was successfully submitted, a message appears in the dialog box:

If the discovery could not be run because the user does not have the appropriate permissions or because there were errors, an error message appears describing the error.

Environment Editor

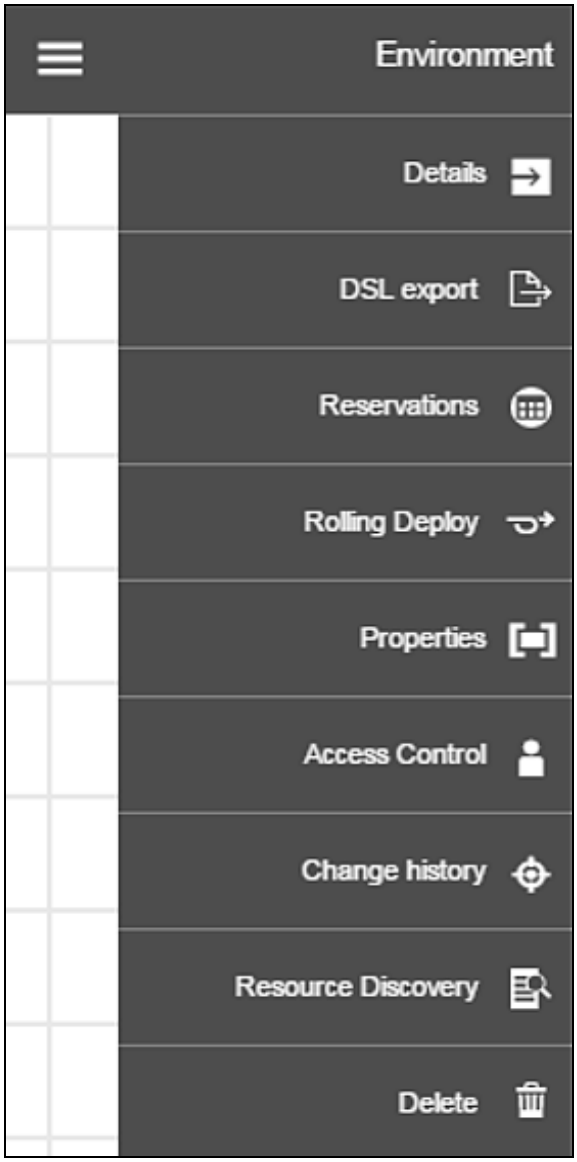
Starting from the Environments List:

1. Click on an environment name to select it.

   The Environment Editor opens.

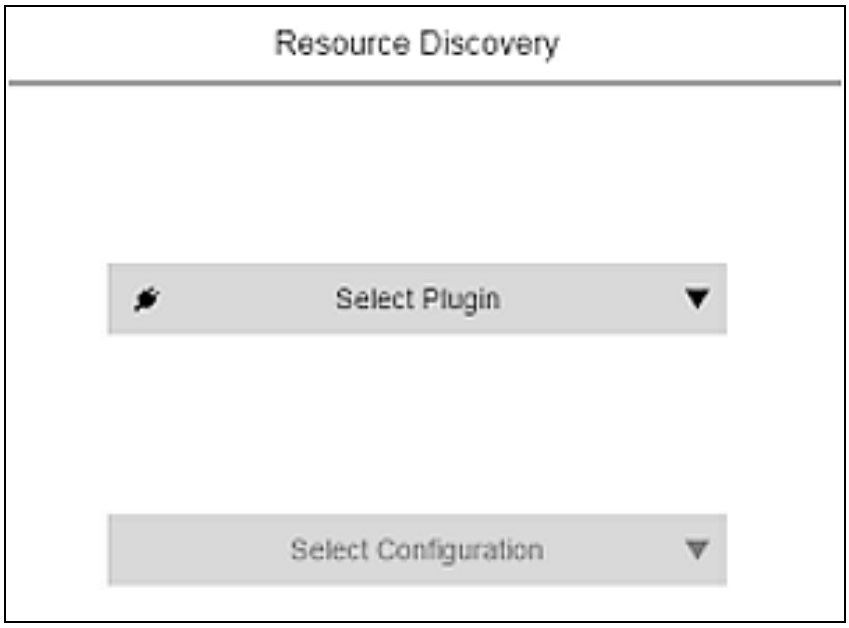
2. Click the button and select **Resources Discovery**.

Example:



The Resource Discovery dialog box opens.

Example:



3. In the **Discover Using** field, select the plugin to use to run the Discovery procedure for the selected resources.
4. (Optional) If the plugin requires a configuration for the Discovery procedure, select the configuration from the drop-down list in the **Configuration Name** field.

Example:

5. Click **OK**.

If the discovery request was successfully submitted, a message appears in the dialog box:

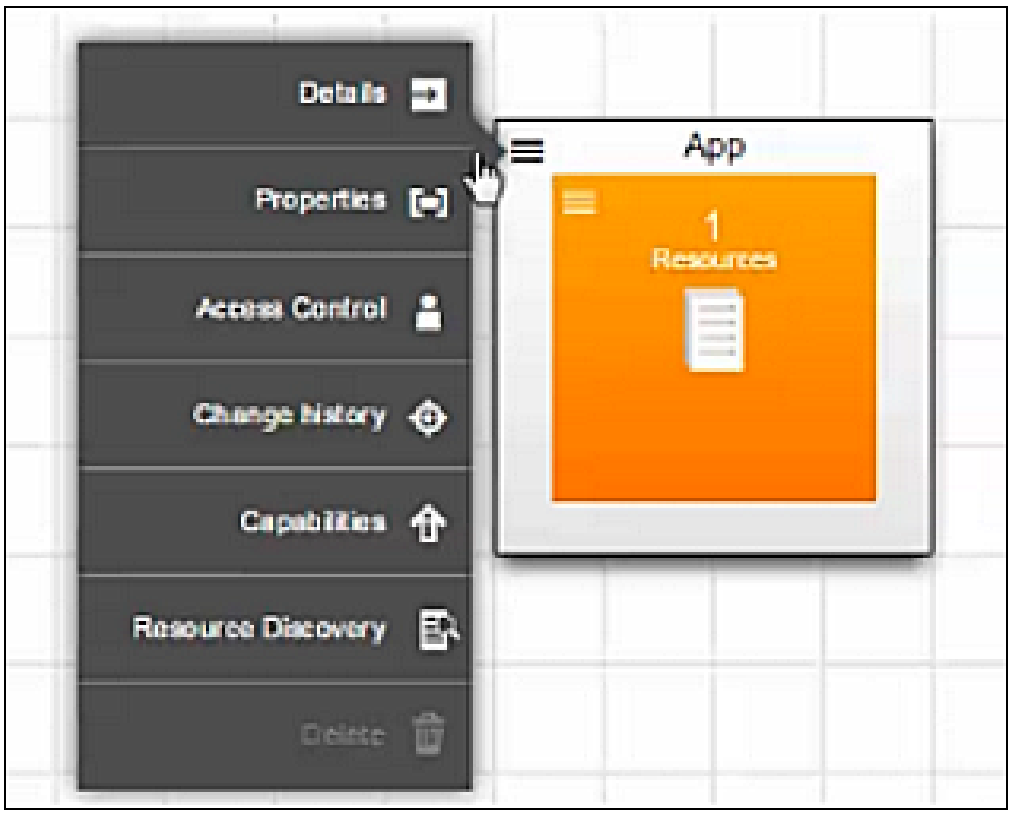
If the discovery could not be run because the user does not have the appropriate permissions or because there were errors, an error message appears describing the error.

Environment Tier

Starting from an Environment Tier:

- Select an environment tier in the Environment Editor.
- Click the button for the tier, and select **Resource Discovery**.

Example:

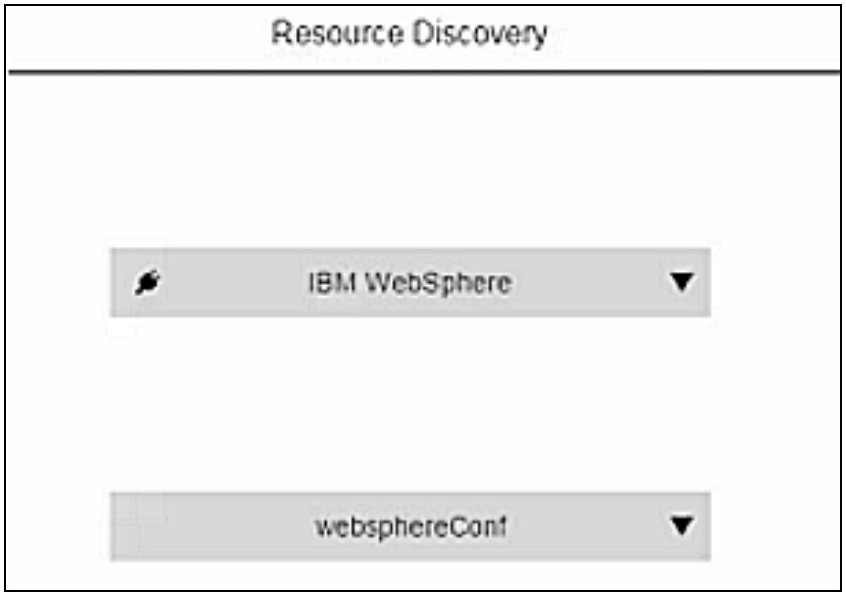


The Resource Discovery dialog box opens.

- In the **Discover Using** field, select the plugin to use to run the Discovery procedure for the selected resources

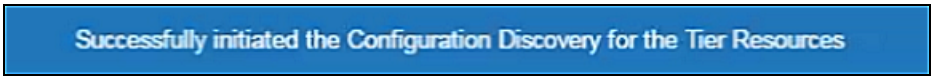
- (Optional) If the plugin requires a configuration for the Discovery procedure, select the configuration from the drop-down list in the **Configuration Name** field.

  Example:



- Click **OK**.

If the discovery request was successfully submitted, a message appears in the dialog box:



If the discovery could not run because the user does not have the appropriate permissions or because there were errors, an error message appears describing the error.

Environment Reservations

Environment Reservation equips the operations team with capabilities that allow it to impart control on environments or environment tiers, making application deployments more reliable and predictable.

Every environment has an *enable* or *disable* flag that you can leverage to control the deployment, acting as an ON/OFF switch for the environment. Once enabled, an environment is open for deployments 24 hours a day. An additional option called **Reservation Required** can be used to restrict deployments on enabled environments. Deployment to environments that require reservation are permitted only if the application, pipeline, or release has a valid reservation. This is a good way to control deployments to controlled environments, such as production environments.

There are two main types of environment reservations:

- Blackouts

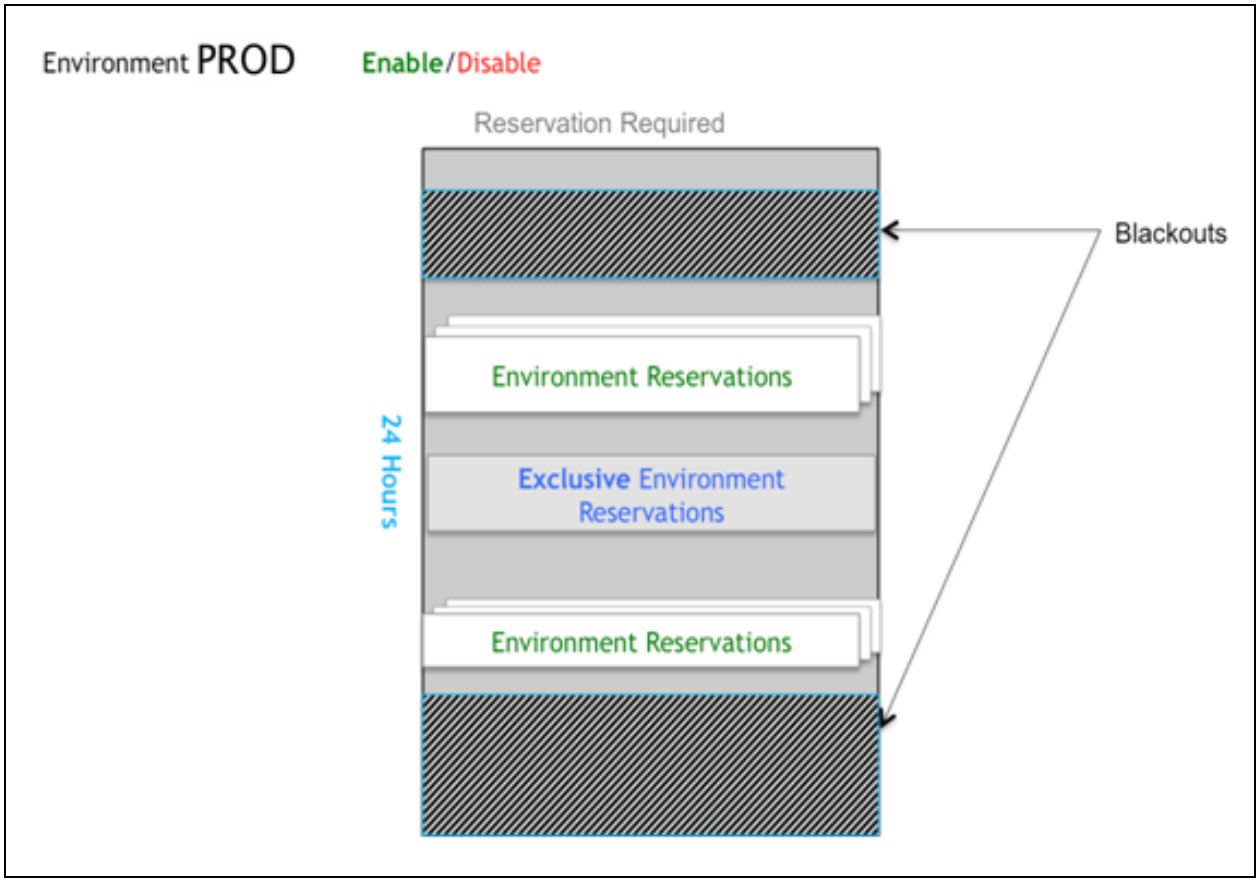
  These are times during which all deployments are prohibited to the environment. They are useful when an environment needs to be brought down for system upgrades or maintenance tasks or to reflect restricted time periods based on business needs such as *trading windows* for financial institutions.

- Environment reservations for applications, pipelines, and releases

  Environment reservations can be exclusive or nonexclusive.

  Exclusive reservations are useful when you want to control deployment conflicts. Only one application, pipeline, or release can have an exclusive reservation on an environment at any time. This helps to avoid conflicts due to double booking. Exclusive reservations can be set by not allowing overlap.

  Nonexclusive reservations are useful when it is acceptable for reservations to overlap at the same time.



Important: Environment reservation rules are only validated and enforced at the beginning of application deployments. If you try to run an application when the reservation exists in the

uture, the job will wait for the reservation window to open before it is executed. This will relieve ou of having to wait for the reservation period to open before the application is deployed.

This is a summary of the business rules for environment reservations that are validated when you try to run an application.

- Check whether the environment is `ENABLED`.

  If `TRUE`, continue.

- Check whether the deployment falls within a `BLACKOUT` period.

  If `TRUE`, the job will wait until the blackout period is over (given that the maximum job wait time is not exceeded).

- An environment reservation is not required.

  Look for `EXCLUSIVE` reservations.

  - Check whether a `RESERVATION` exists for a different application.
    - If `TRUE`, the job will not run.
  - Check whether a future `RESERVATION` exists for the application.
    - If `TRUE`, the job will wait till the reservation time.
  - Check whether an `ACTIVE RESERVATION` exists for the application or if no other application has an `EXCLUSIVE` reservation.
    - If `TRUE`, start the deployment.

- An environment reservation is required.
  - Check for an application reservation.
    - If `TRUE`:
      - If the reservation is currently active, the deployment starts.
      - If the reservation is in the future, ElectricFlow *waits* for the reservation period to open.
    - If `FALSE`:
      - The job will not run.

In a pipeline, execution starts immediately. However, when an application deployment task begins, ElectricFlow looks for the pipeline reservation on the environment. A release follows the same pattern as the pipeline. The release will start immediately. However, when the deployer task begins deploying the application, the system will validate the reservation to make sure the appropriate environment reservation exists for that release. In both of these scenarios, the rules previously mentioned for application deployments apply to each of the applications that are deployed by the pipeline task and the deployer task.

Environment Reservation Use Case

ABC Bank is one of the leading banks that pride themselves for their cutting edge transaction processing system, which is leveraged by major businesses. Keeping systems up and running at all

times is paramount to their business. This describes how environment reservations can be leveraged to streamline deployments by reducing deployments conflicts and improving system uptimes.

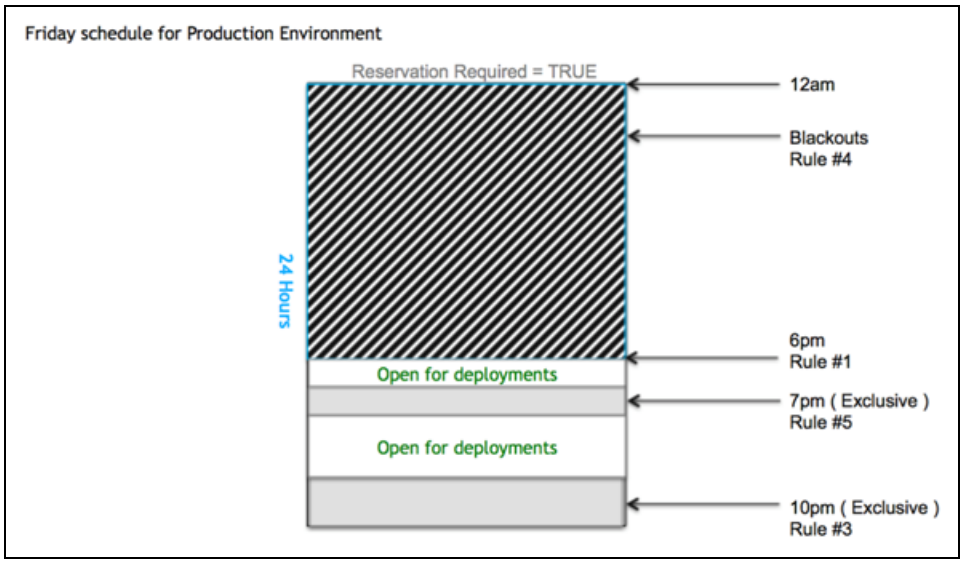
These are the main business rules for the production environment:

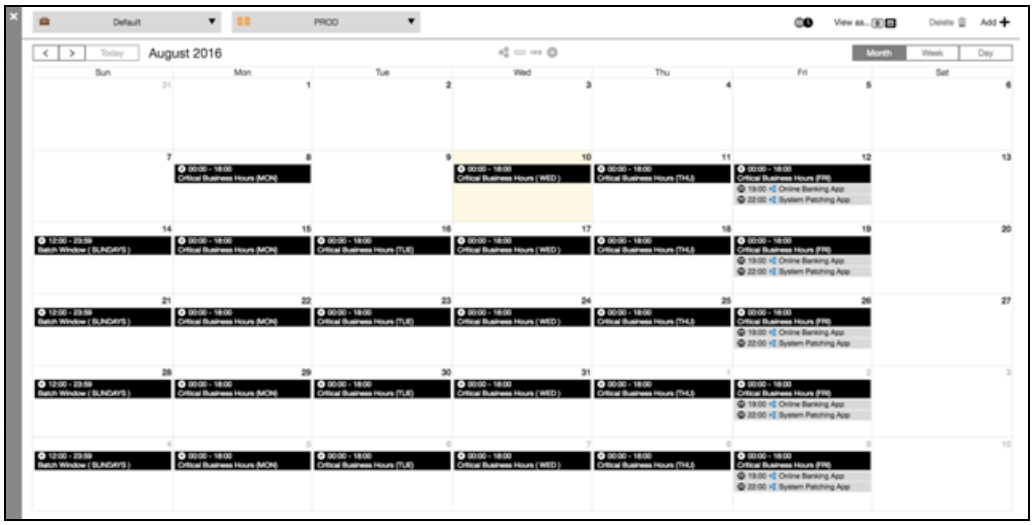
1. No deployments can happen to production environments without proper approval. Approval is required primarily to make sure the deployment window is properly planned to reduce deployment conflicts and downtime.
2. No deployments can happen on Sunday between noon and midnight. This is when all the transaction settlements (batch processing) happen, and the bank does not want any interruption or degradation of system performance during these critical times.
3. Maintenance windows are on Friday nights from 10:00 P.M. to midnight. The bank has an application process for patching production machines.
4. No deployments can happen between midnight and 6:00 P.M. on Monday through Friday because these times are considered critical business hours.
5. At 7:00 P.M. on Friday, the "Online Banking" application needs an exclusive reservation to upgrade to newer version. No other deployments are allowed during this time because avoiding conflicts for this critical rollout is important.

To implement these rules, starting with the most restrictive clause:

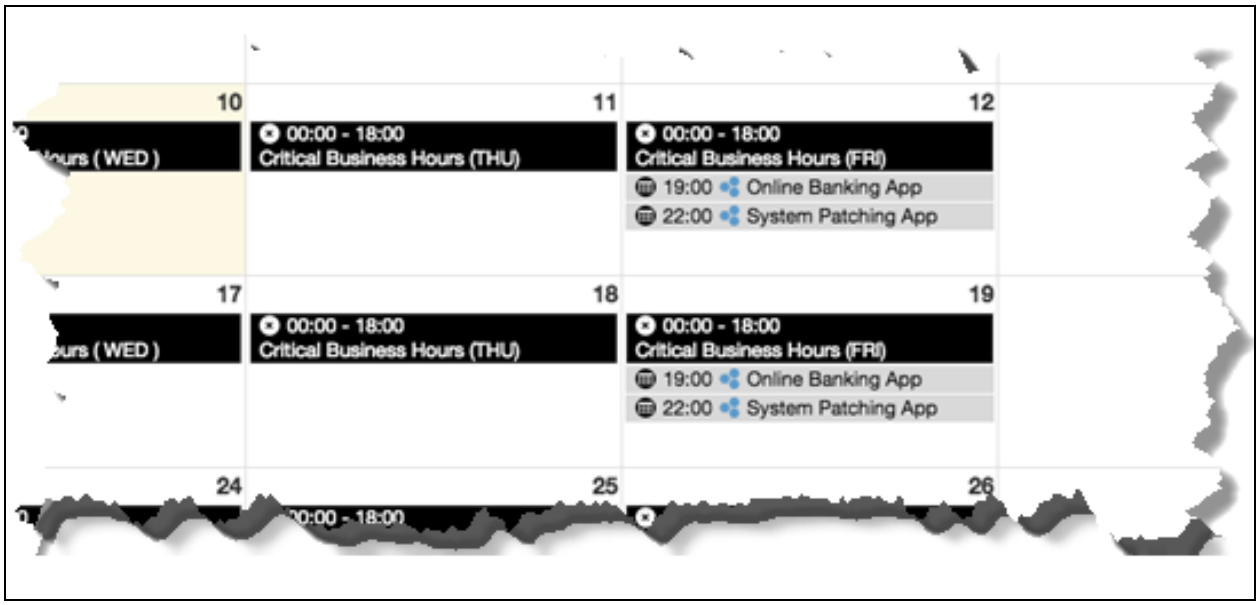
1. First implement the fourth rule by creating a recurring blackout from Monday though Friday between midnight and 6:00 P.M. This will ensure that no one will be able to deploy during this time period.
2. Implement the second rule next by creating a recurring blackout on Sundays from noon to midnight. This will prevent anyone from deploying during the transaction settlement window.
3. For the third rule, create a recurring exclusive reservation starting on Friday from 10:00 P.M. to midnight to deploy the application process used for system patching.
4. For the fifth rule, create a recurring exclusive reservation for the "Online Banking" application starting on Friday at 7:00 P.M.
5. For the first rule, start by setting the **Reservation Required** attribute on the production environment. This will prevent any unscheduled deployments to production. Then enhance the approval process to create an environment reservation during the available open deployment time periods for the application you want to deploy. It also makes sense to create an application run schedule to match the reservation.

The following are visual representations of the schedule for the production environment on Friday.

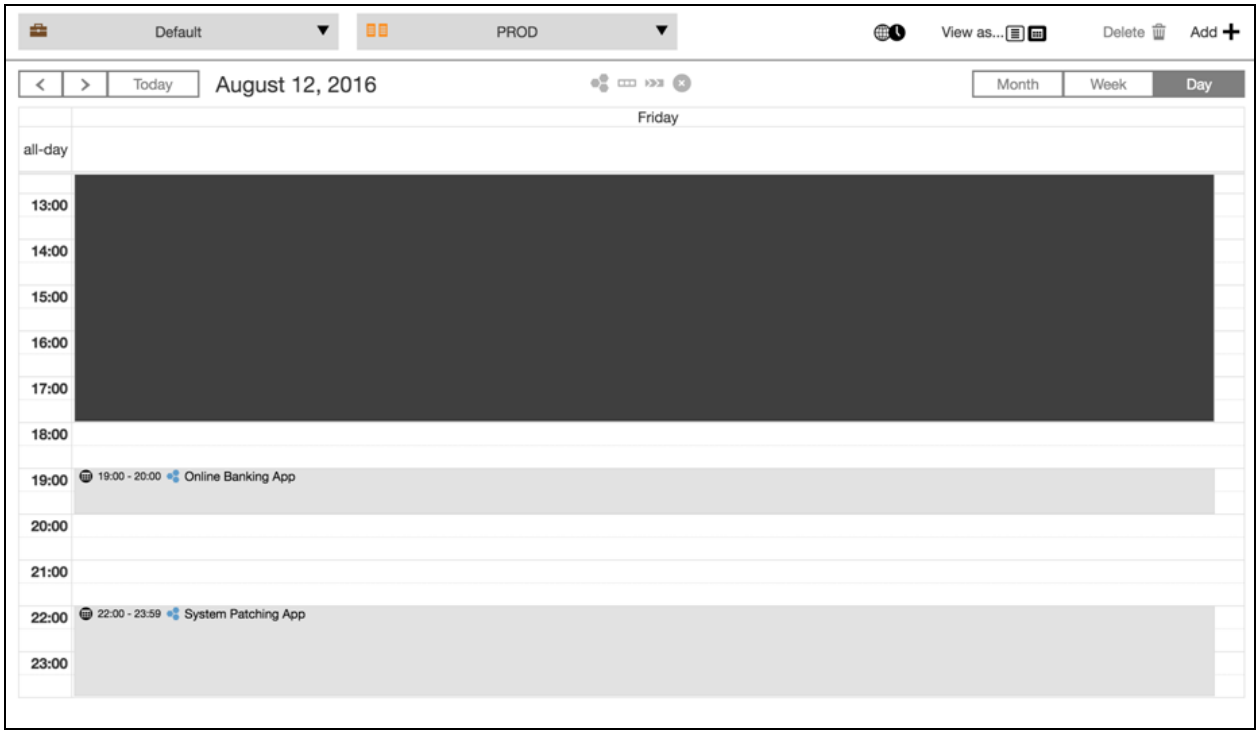


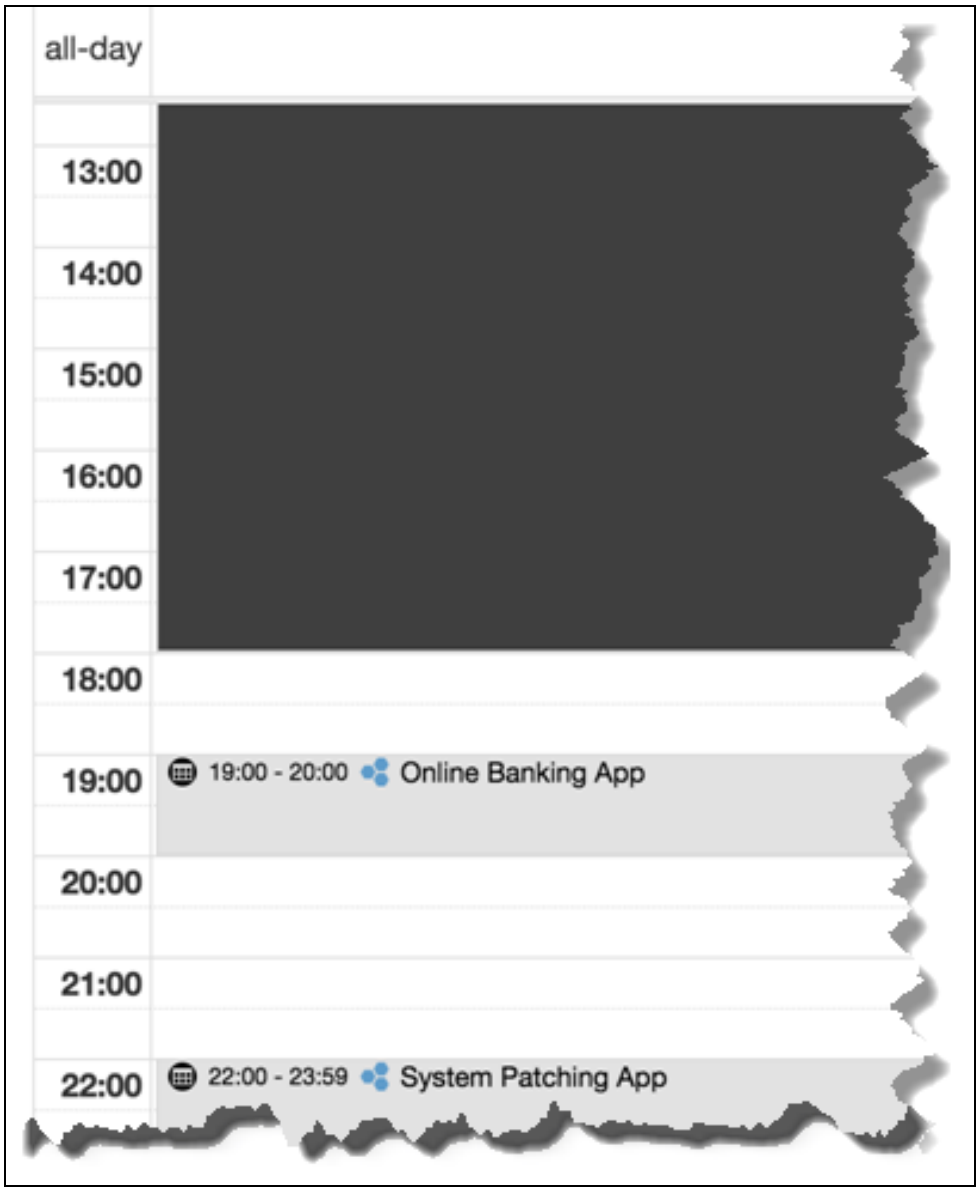


This shows the details for August 11 and 12 and also for August 18 and 19.



This is the daily view of the environment reservations for the production environment.

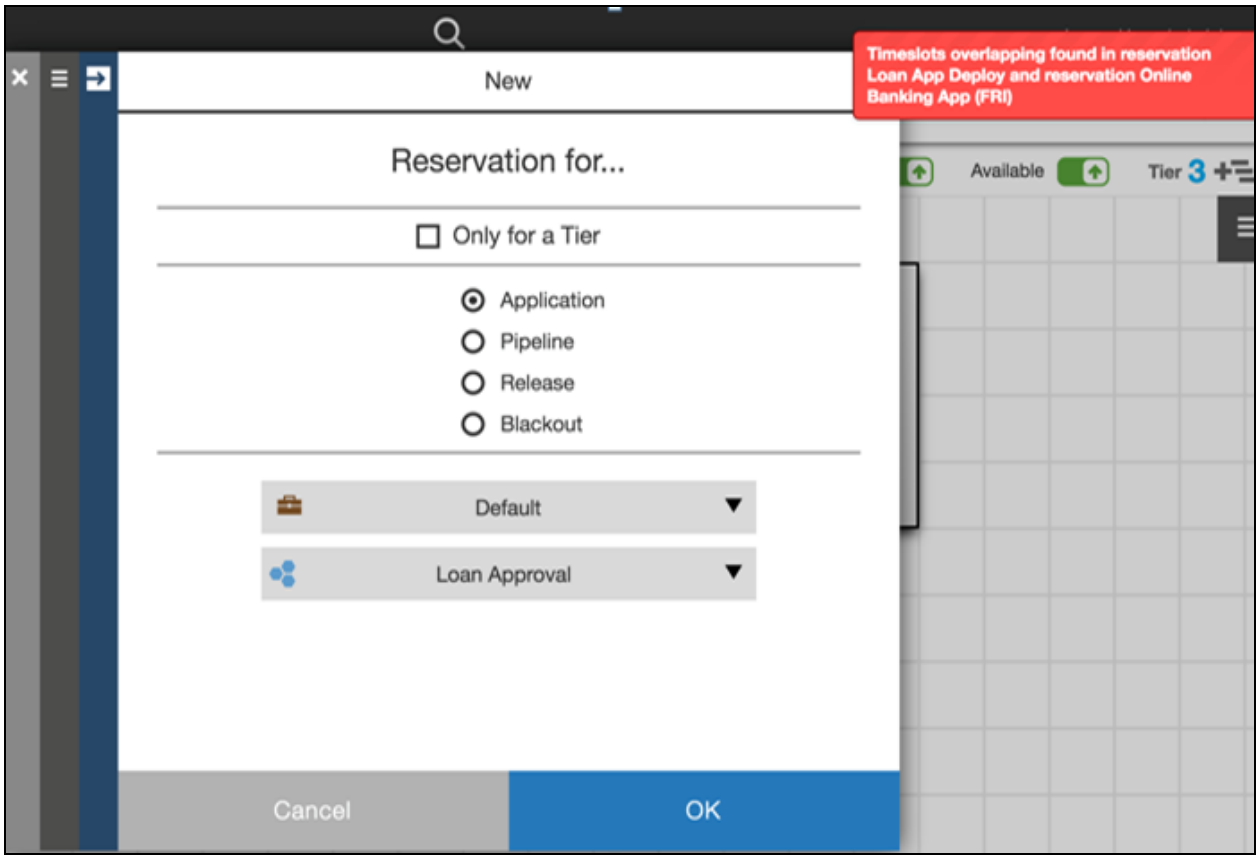




Conflict Resolution

If you try to deploy during the blackout or exclusive reservation windows in the previous section, the job will wait until the blackout period is over (given that the maximum job wait time is not exceeded).

When you try to create a reservation that conflicts with existing exclusive reservation, the job will also fail with an error.

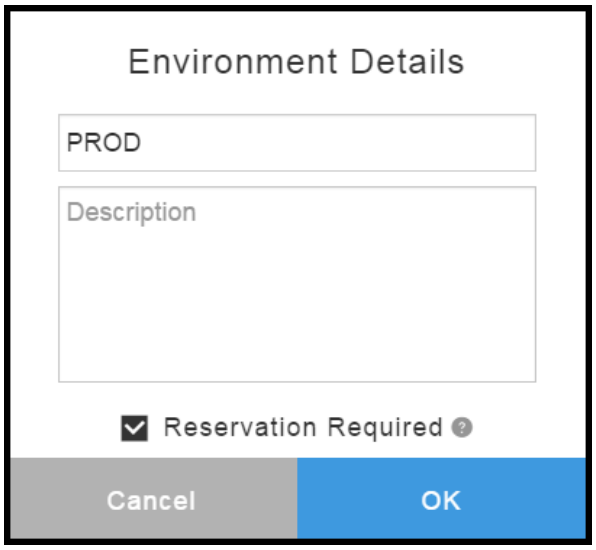


This is the error message:

Environment Reservation UI

Restricting Deployments to Enabled Environments

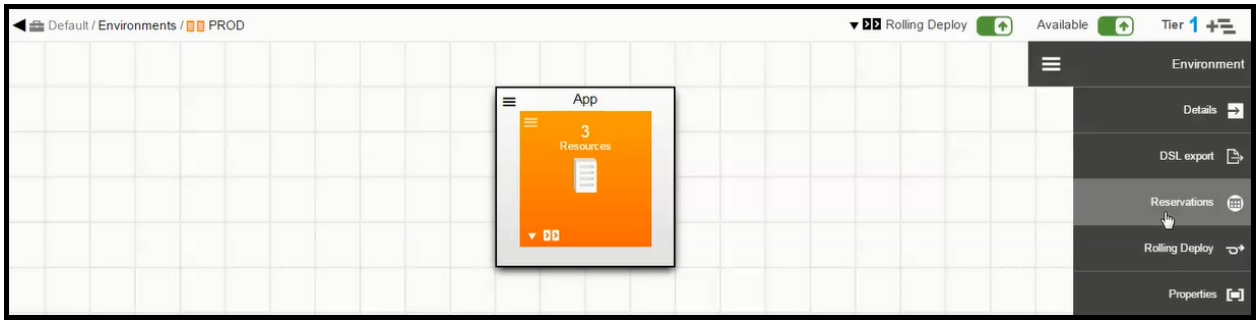
You can restrict deployments to environments by selecting the **Reservation Required** option for an environment. In the Environment Details dialog, select **Reservation Required** and click **OK**. Deployment to this environment permitted only if an application, pipeline, or release has a valid reservation.



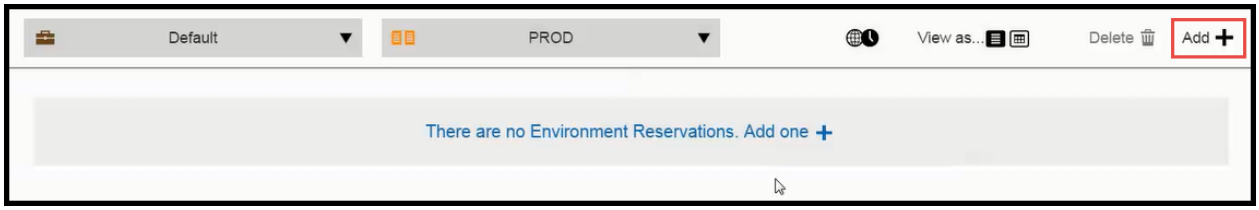
Adding Environment Reservations

This example shows how to add environment reservations to the PROD environment. Starting from the Environments list:

1. Select the environment to which you want to add reservations.
2. In the Environment Editor, click  in the upper right corner, and select **Reservations**.

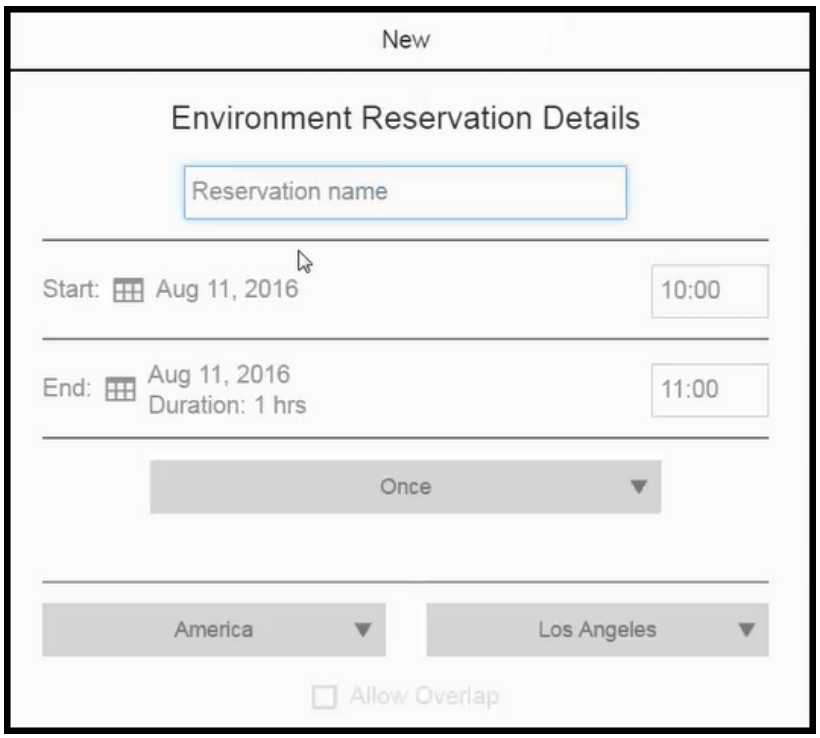


The dialog box showing the list of environment reservations opens.

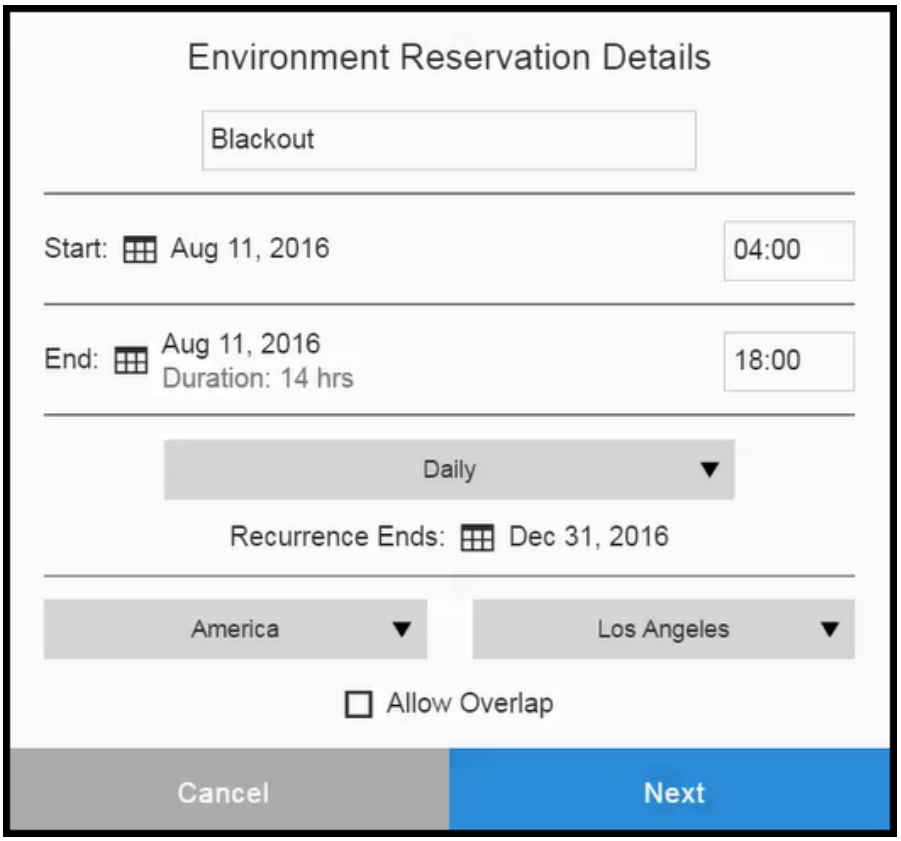


3. Click **Add+** to add an environment reservation.

   The **Environment Reservation Details** dialog box opens.

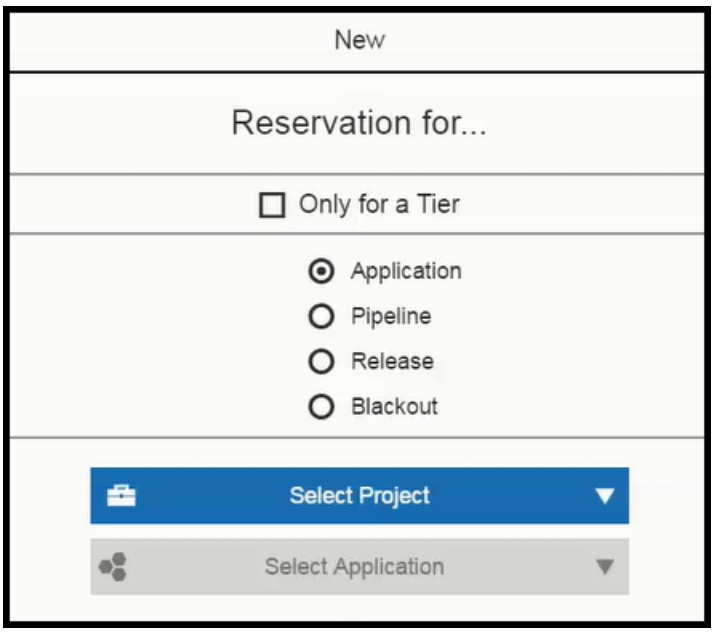


4. To create a blackout:
    1. In the **Reservation name** field, enter the name of the reservation.
    2. In the **Start** field, select the start date and enter the start time.
    3. In the **End** field, select the end date and enter the end time.
    4. In the **Once** field, select the frequency: **Once**, **Daily**, **Weekly**, or **Monthly**.
    5. (Optional) If you select **Daily**, **Weekly**, or **Monthly**, select the date when the reservation ends in the **Recurrence Ends** field.
    6. (Optional) Change the time zone.
    7. (Optional) Select **Allow Overlap** to make the reservation nonexclusive.

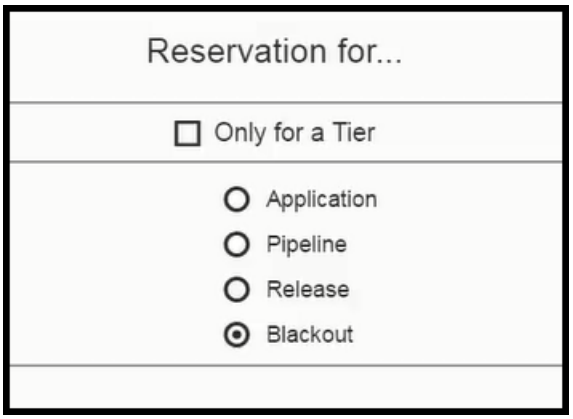


8. Click **Next**.

   The **Reservation for** dialog box opens.

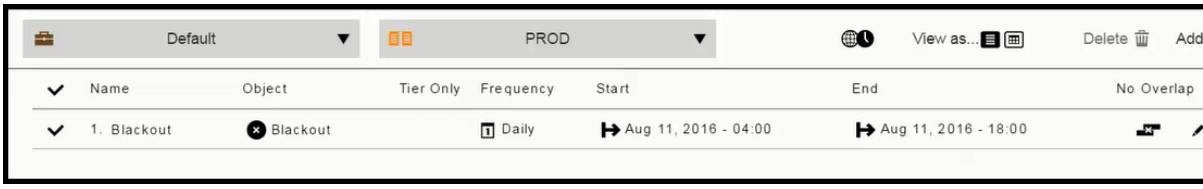


9. (Optional) Select **Only for a Tier** if the reservation only applies to an environment tier.
10. Select **Blackout**.



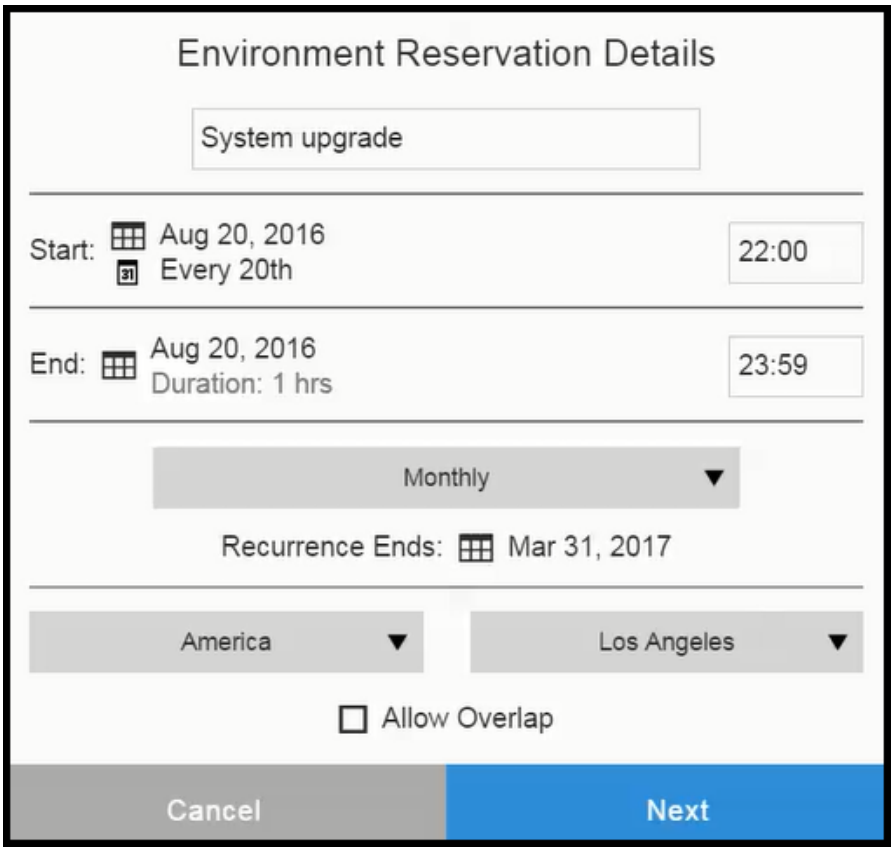
11. Click **OK**.

    The Environment Reservations list returns and has been updated with the reservation you just created.



12. (Optional) Click **Add +** to add another reservation.

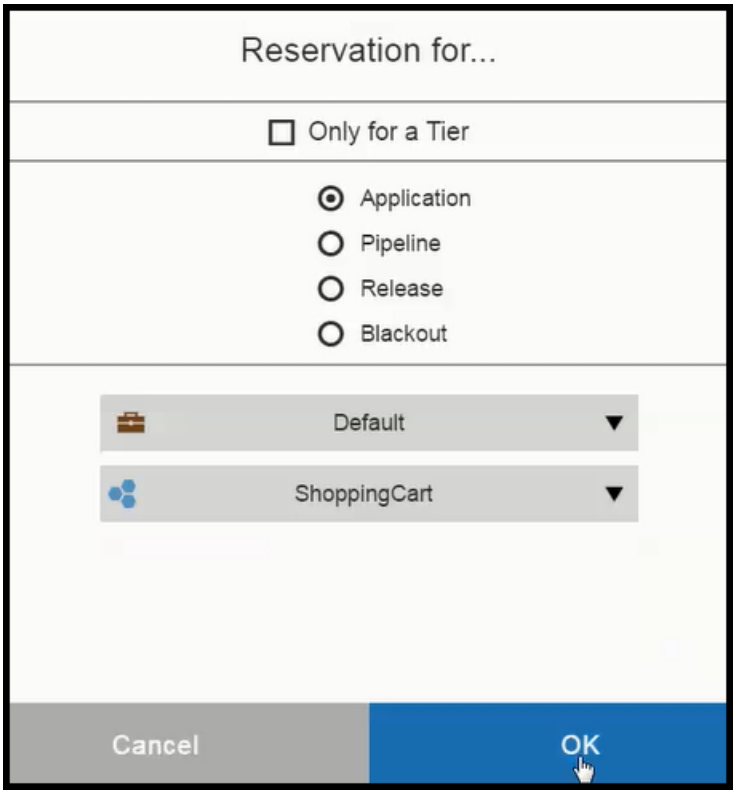
5. To create an exclusive invitation:
    1. In the **Reservation name** field, enter the name of the reservation.
    2. In the **Start** field, select the start date and enter the start time.
    3. In the **End** field, select the end date and enter the end time.
    4. In the **Once** field, select the frequency: **Once, Daily**, **Weekly**, **Monthly**.
    5. (Optional) If you select **Daily**, **Weekly**, or **Monthly**, select the date when the reservation ends in the **Recurrence Ends** field.
    6. (Optional) Change the time zone.
    7. (Optional) Select **Allow Overlap** to make the reservation nonexclusive.



    8. Click **Next**.

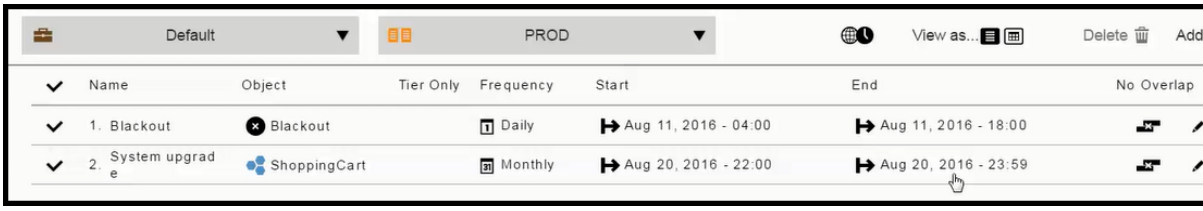
       The **Reservation for** dialog box opens.
    9. (Optional) Select **Only for a Tier** if the reservation only applies to an environment tier.

10. Select the object that will have the reservation (**Application**, **Pipeline**, or **Release**) and the object's project and object name.



11. Click **OK**.

    The Environment Reservations list returns and has been updated with the reservation you just created.

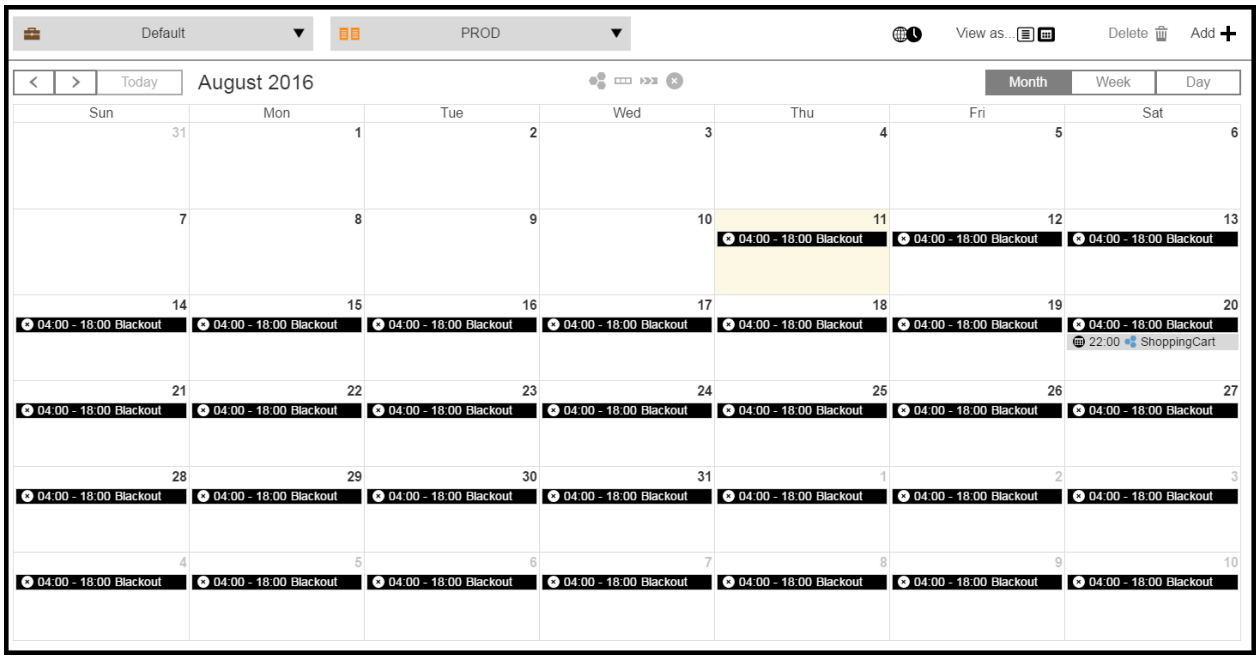


Viewing Environment Reservation Calendars

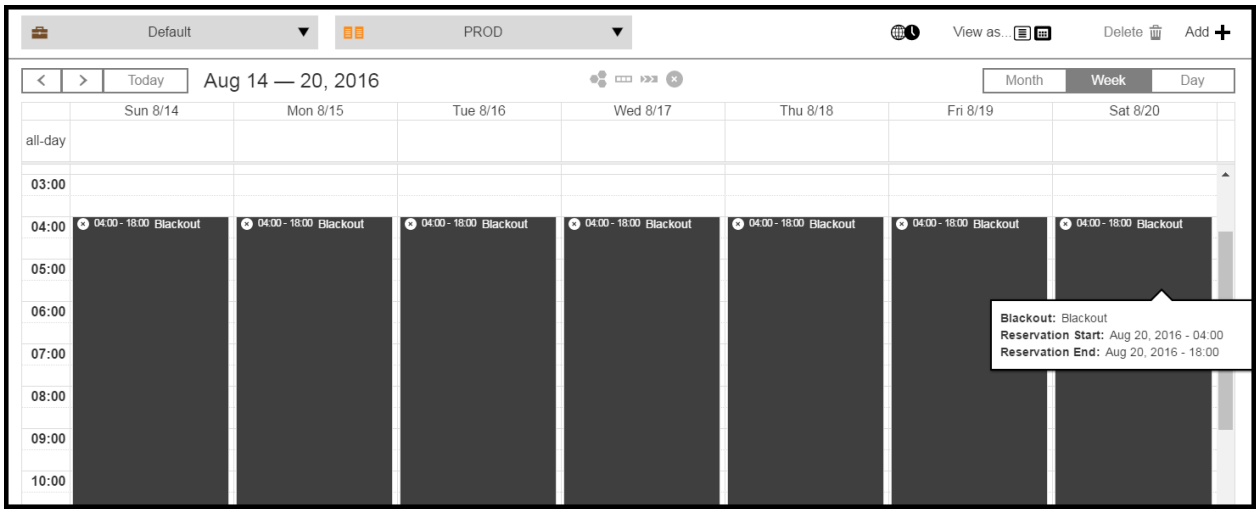
There are two ways to access the environment reservation calendars:

- From the Environment Reservations list, click  for the calendar view.
- From the Environments List, select an environment and click  to view the Environment Reservations list. Then click  at the upper right of the dialog box.

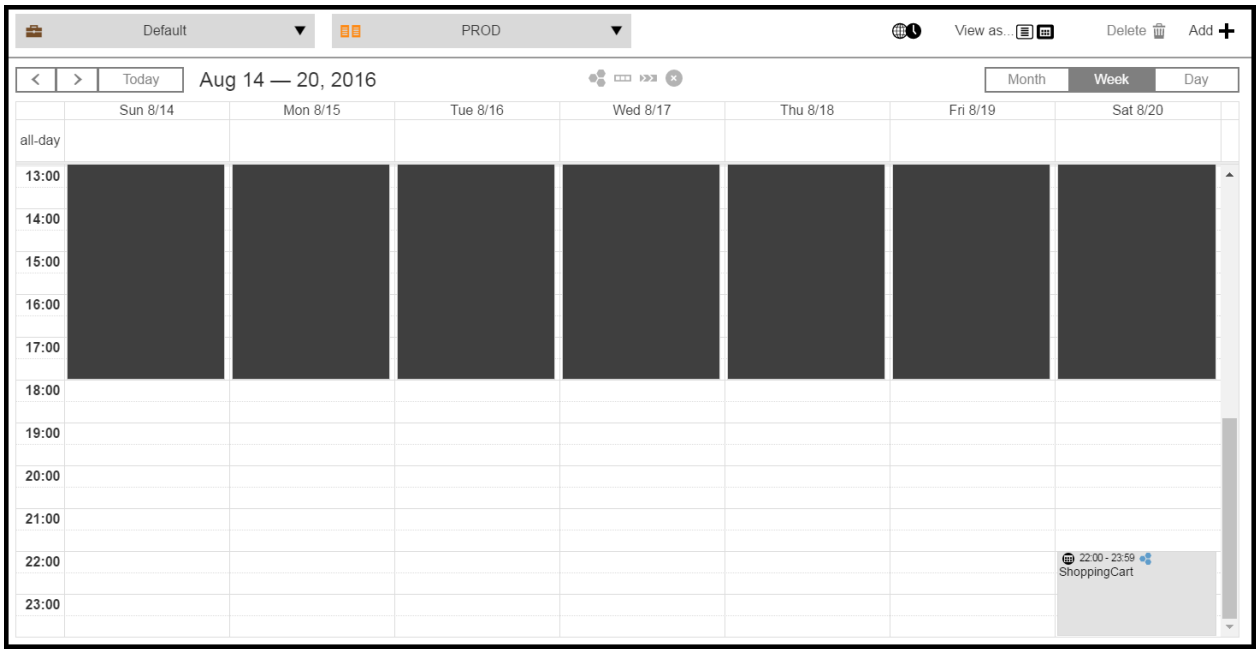
This is the **Month** view:



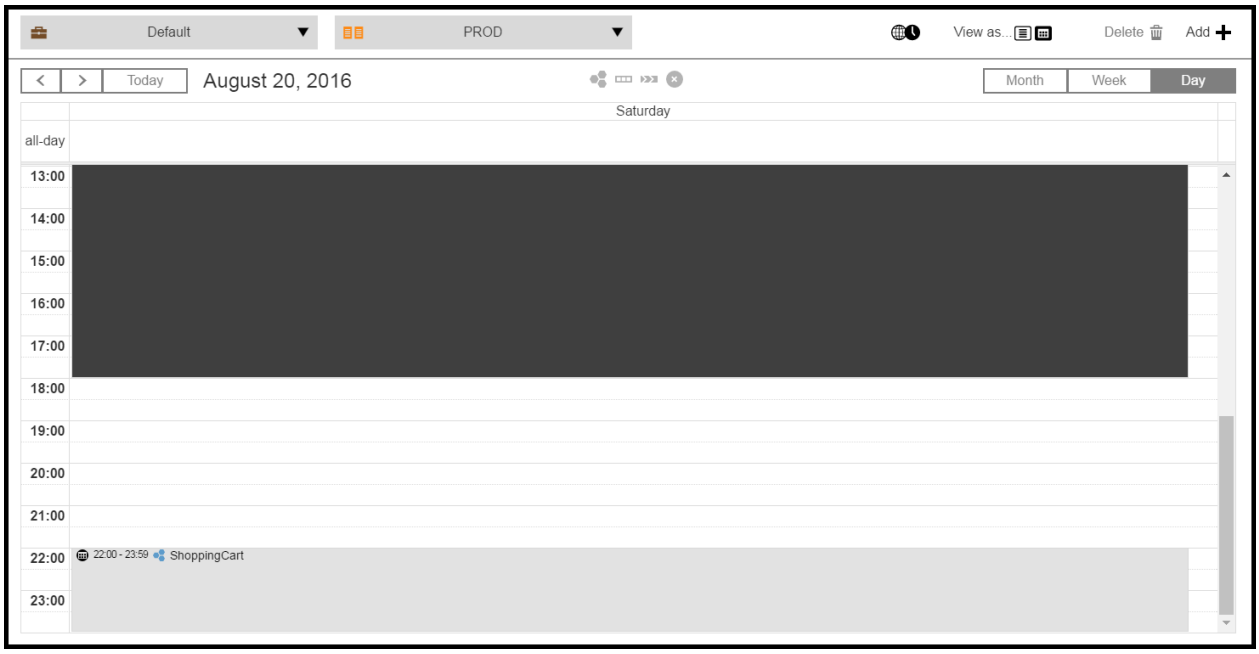
Click **Week** to see the **Week** view:



Scroll down to see more:



Click **Day** to see the **Day** view:



For examples of conflict resolution, see Conflict Resolution on page 225

Full-Stack Dependency View

The full-stack dependency view helps you to visualize and manage dependencies in application models. In distributed environments, different teams can author applications and manage the infrastructure, which means that managing dependencies across the stack is crucial. This view shows the mapping between application tiers and environment tiers so that you can verify that the infrastructure capabilities can support the minimum application requirements.

Viewing Dependencies and the Stack Comparison

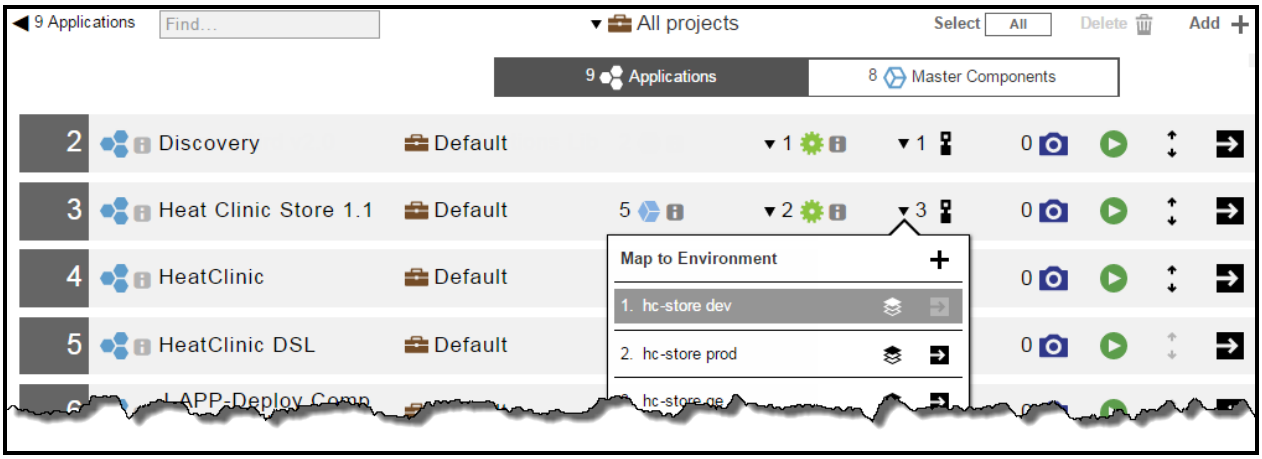
Stack Definition

Stack Templates

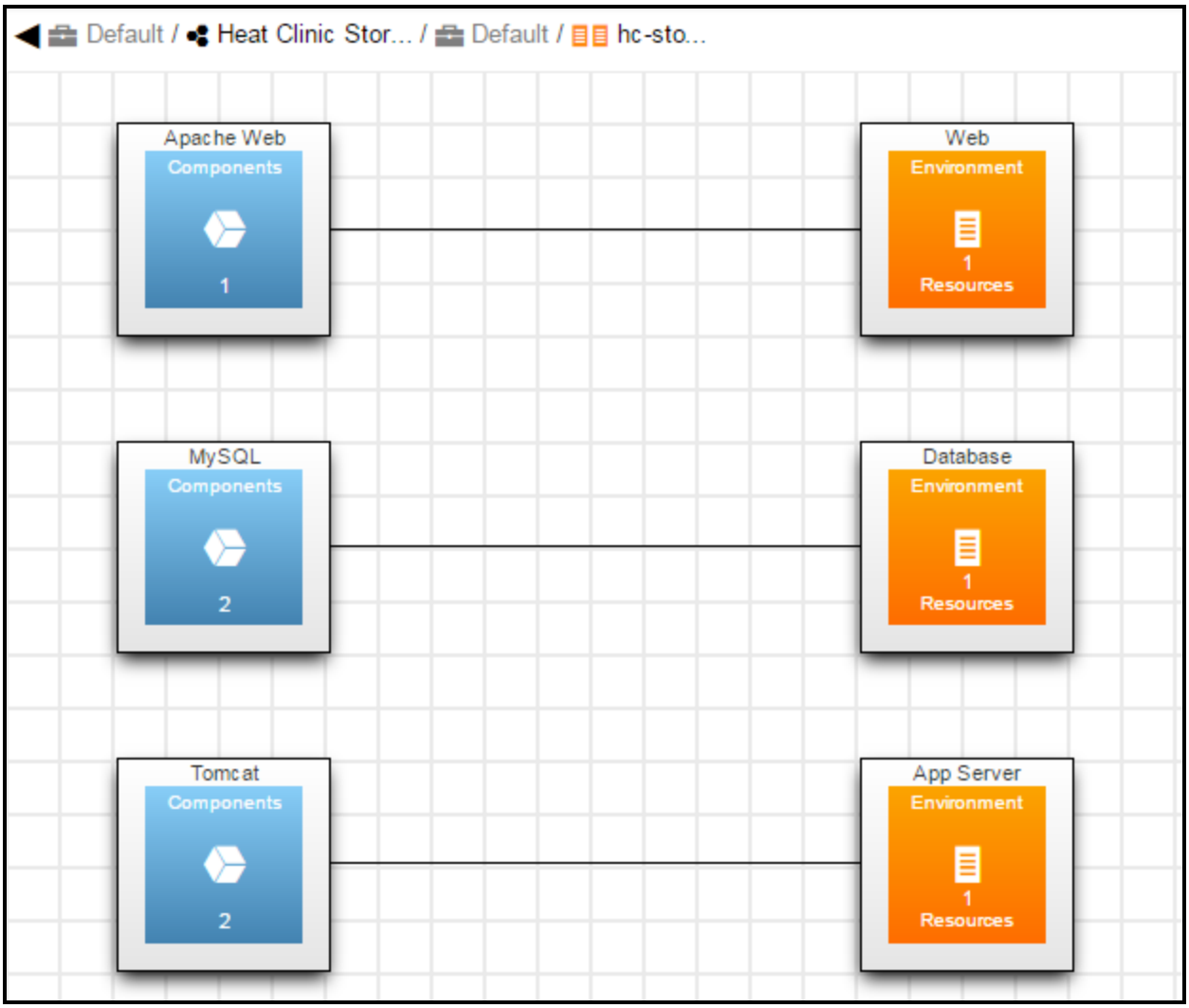
Viewing Dependencies and the Stack Comparison

The full-stack dependency view shows the mapping of application tiers to environment tiers in an application model. This lets you compare application tier requirements to environment tier capabilities for a tier mapping. There are two ways to access this view:

- From the Applications list, click the  (tier maps) button for an application and then select a tier map:



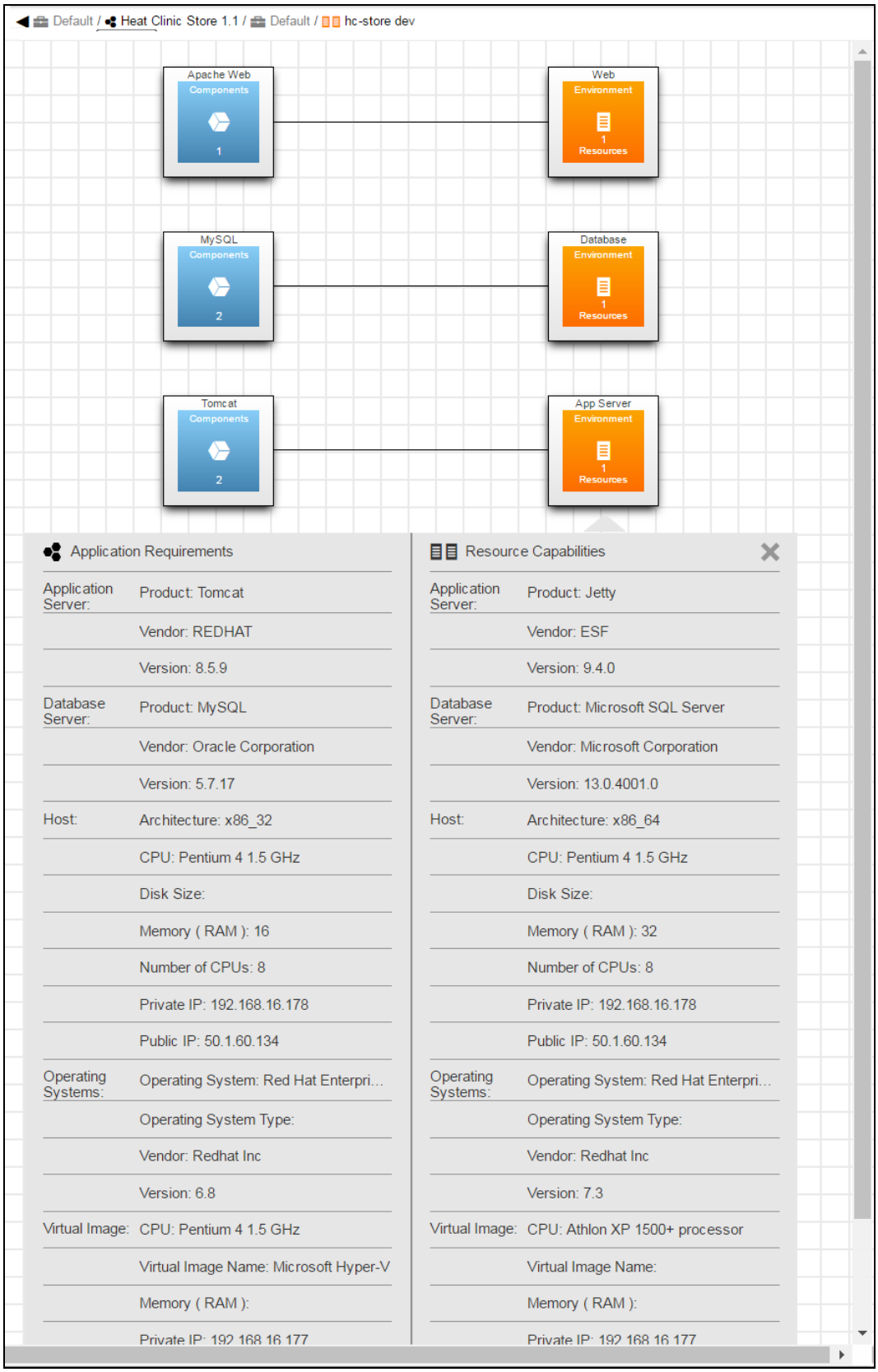
The tier map appears:



- Click either an application tier or an environment tier for each mapping pair.

  The stack comparison view appears. See Stack Definition on page 240 for more information.

You can click on either tier in a tier mapping to see the currently set values for application requirements and environment capabilities. This lets you do a side-by-side comparison to see if any application requirements are out of compliance. For example:


Default / Heat Clinic Store 1.1 / Default / hc-store dev
Apache Web
Components
1
Web
Environment
1
Resources
MySQL
Components
2
Database
Environment
1
Resources
Tomcat
Components
2
App Server
Environment
1
Resources
Application Requirements
Application Server: Product: Tomcat
Vendor: REDHAT
Version: 8.5.9
Database Server: Product: MySQL
Vendor: Oracle Corporation
Version: 5.7.17
Host: Architecture: x86_32
CPU: Pentium 4 1.5 GHz
Disk Size:
Memory (RAM): 16
Number of CPUs: 8
Private IP: 192.168.16.178
Public IP: 50.1.60.134
Operating Systems: Operating System: Red Hat Enterpri...
Operating System Type:
Vendor: Redhat Inc
Version: 6.8
Virtual Image: CPU: Pentium 4 1.5 GHz
Virtual Image Name: Microsoft Hyper-V
Memory (RAM):
Private IP: 192.168.16.177
Resource Capabilities
Application Server: Product: Jetty
Vendor: ESF
Version: 9.4.0
Database Server: Product: Microsoft SQL Server
Vendor: Microsoft Corporation
Version: 13.0.4001.0
Host: Architecture: x86_64
CPU: Pentium 4 1.5 GHz
Disk Size:
Memory (RAM): 32
Number of CPUs: 8
Private IP: 192.168.16.178
Public IP: 50.1.60.134
Operating Systems: Operating System: Red Hat Enterpri...
Operating System Type:
Vendor: Redhat Inc
Version: 7.3
Virtual Image: CPU: Athlon XP 1500+ processor
Virtual Image Name:
Memory (RAM):
Private IP: 192.168.16.177

Stack Definition

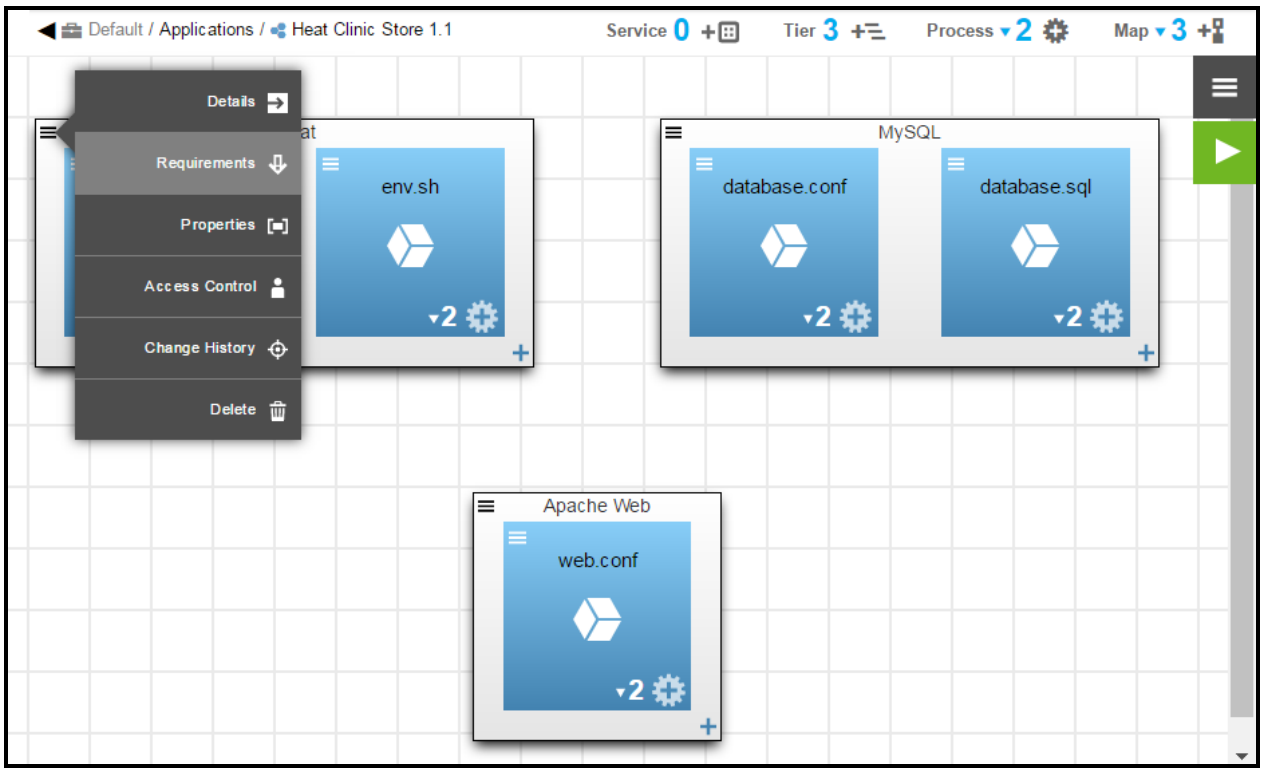
When you are modeling an application, you have a target environment in mind. You might be modeling an application that is compute-intensive or data-intensive. For optimal performance, the application should run in an environment with 4 cores and 32 GB of RAM. If the application is deployed to an environment with one core and 8 GB of RAM, the application deployment might take a long time to run or might fail due to timeout errors.

You can define application requirements and environment capabilities such as:

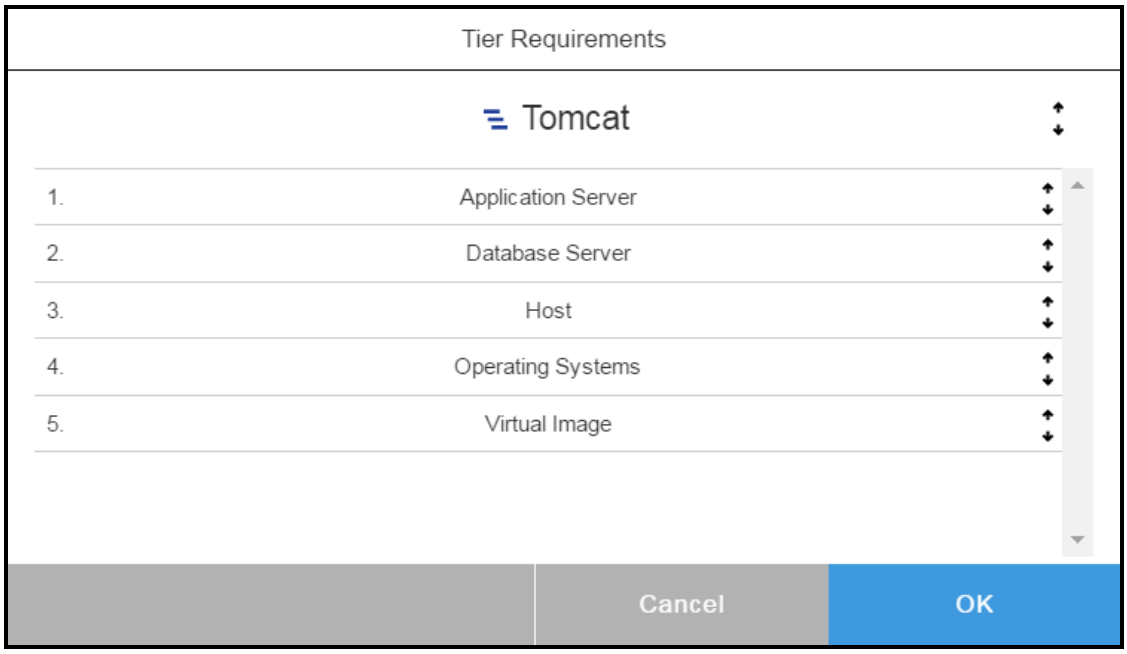
- Supported platforms
- Hardware requirements
- Database requirements
- Disk usage
- Memory usage

Modeling Application Requirements

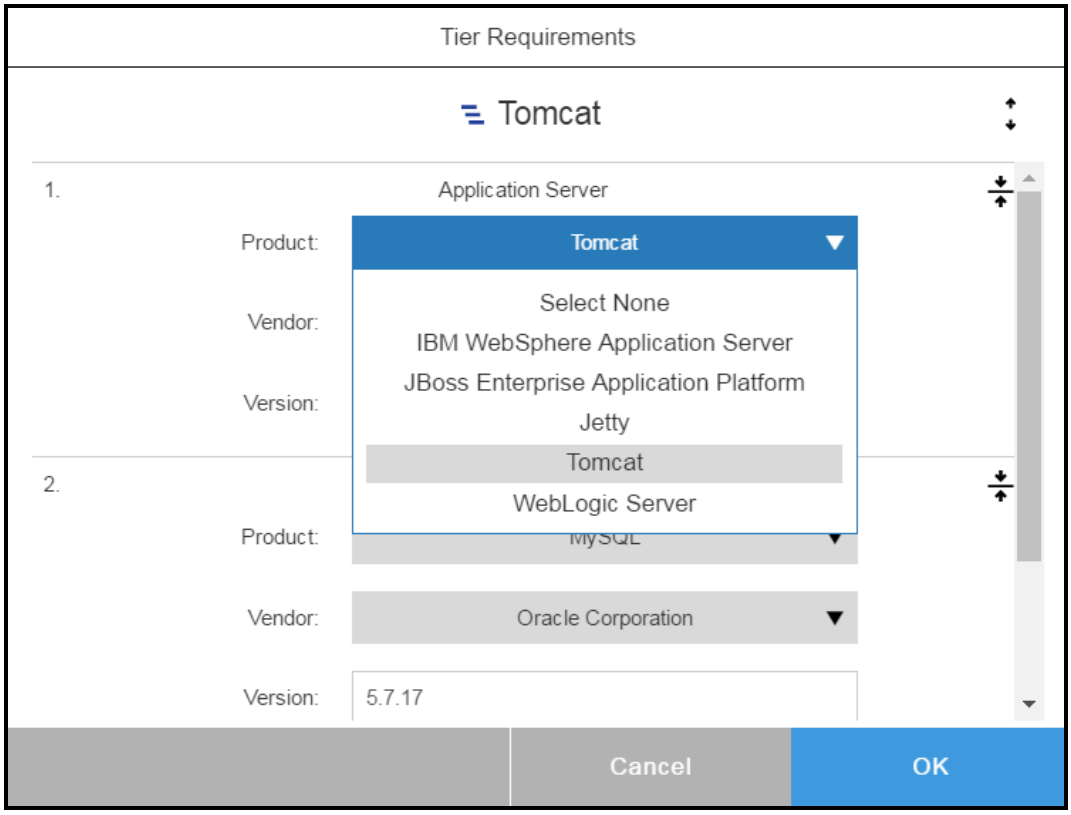
1. Click ☰ (the tier menu) in an application tier, and then click **Requirements:**



The **Tier Requirements** dialog box opens:



2. In the **Tier Requirements** dialog box, enter the requirements for the tier. For example:



3. Click **OK**.

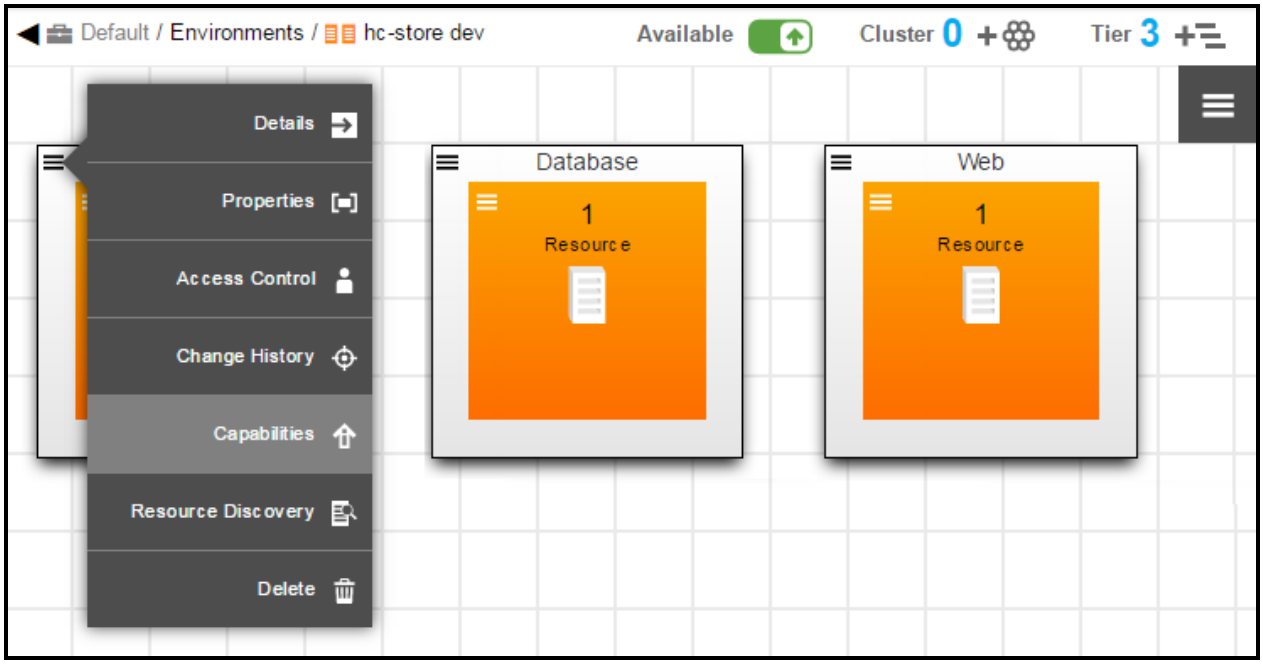
4. Repeat these steps to define the requirements for the other application tiers.

Modeling Environment Capabilities

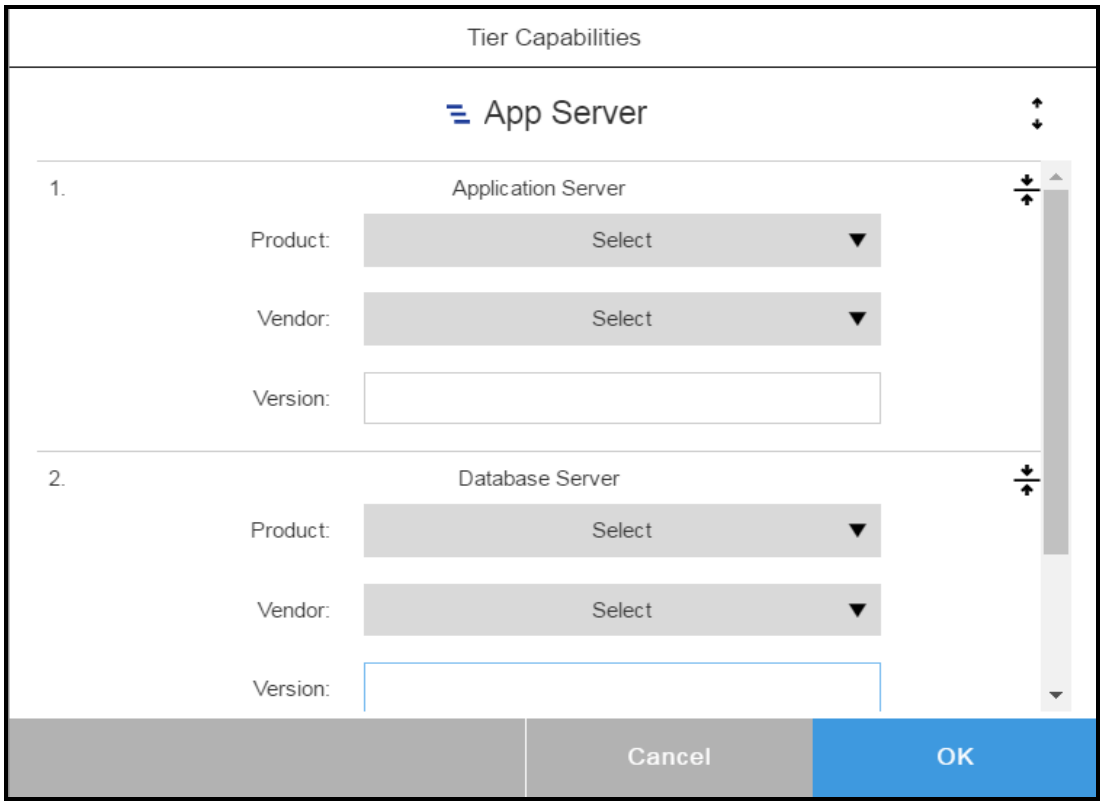
You define capabilities per environment tier.

1. Open the Environment Editor for the environment that you want to model.

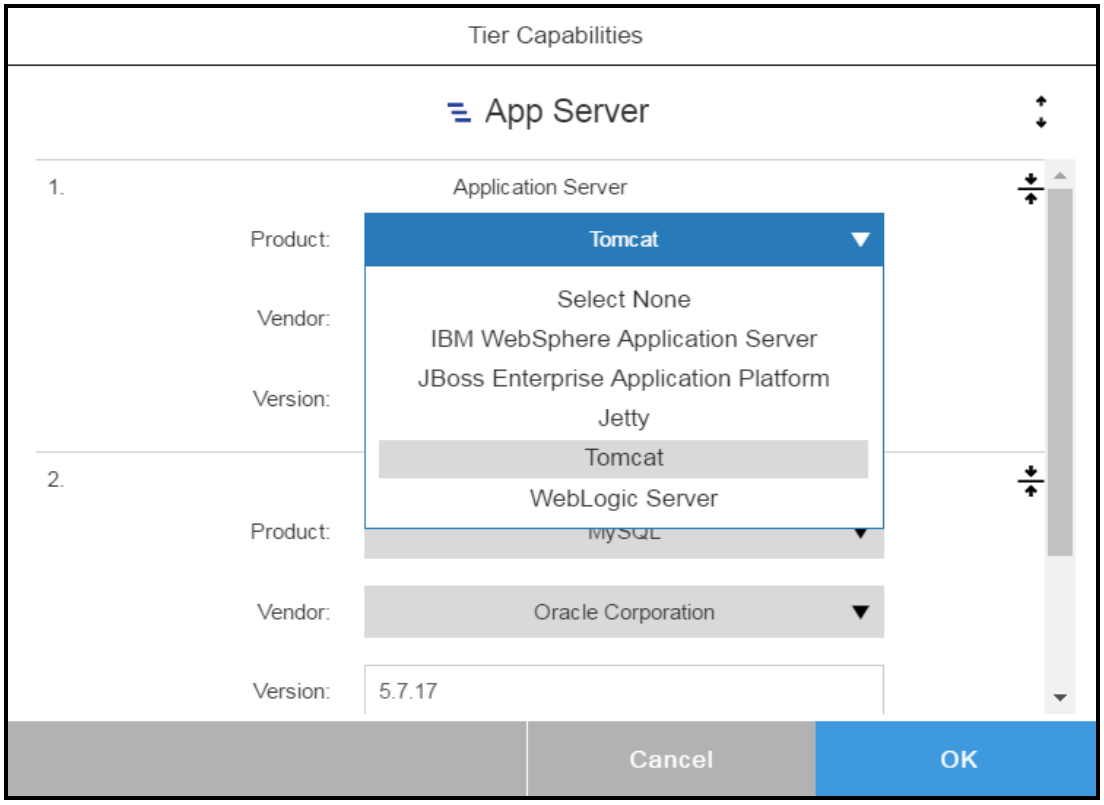
2. Click (the tier menu) in an environment tier, and then select **Capabilities**:



The **Tier Capabilities** dialog box opens:


Tier Capabilities
App Server
1.
Application Server
Product:
Select
Vendor:
Select
Version:
2.
Database Server
Product:
Select
Vendor:
Select
Version:
Cancel
OK

3. In the **Tier Capabilities** dialog box, enter the capabilities for the tier. For example:



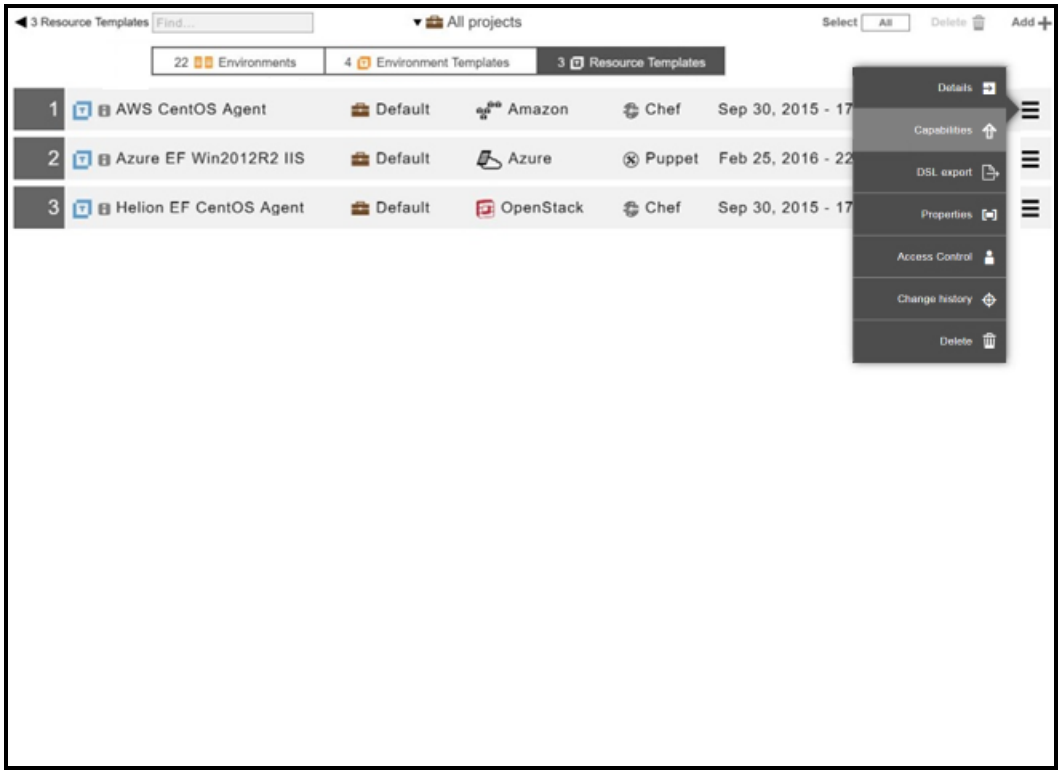
4. Click **OK**.
5. Repeat these steps to define the capabilities for the other environment tiers.

Modeling Infrastructure and Middleware Capabilities

ElectricFlow also allows the modeling of infrastructure and middleware capabilities for resource templates as well as environment tiers.

1. Go to the Resource Templates list and then select any resource template. If no resource template exists, create one.

2. Click ☰ for a resource template, and then select **Capabilities**:



The **Resource Template Capabilities** dialog box opens. This dialog box lets you capture the infrastructure and middleware details:


Resource Template Capabilities
AWS CentOS Agent
1.
Application Server
Product:
JBoss Enterprise Application Platform
Vendor:
Redhat Inc
Version:
7.1
2.
Database Server
3.
Host
Architecture:
CPU:
Athlon XP 1500+ processor
Cancel
OK

3. You can model many attributes for the resource template. Some examples of these attributes are:
    - Application Server:
        - Product
        - Vendor
        - Version
    - Database Server:
        - Product
        - Vendor
        - Version
    - Host:
        - Architecture
        - CPU
        - Disk size
        - Memory (RAM)
        - Number of CPUs
        - Public IP
        - Private IP
    - Operating System:
        - Operating System
        - Operating System Type
        - Vendor
        - Version
    - Virtual Image:
        - CPU
        - Virtual Image Name
        - Memory (RAM)
        - Public IP
        - Private Subnet Mask
        - Private IP

    This list is extensible and more modeling attributes can be added. These attributes let you model infrastructure and Middleware as well as hardware capabilities. These can then be used to compare with the requirements defined on the application tier level.

Stack Templates

A stack template defines the categories of resources (such as Application Server) that are available when you specify the requirements for an application tier. Within each category, the stack template specifies the available properties (such as Jetty).

The Default Stack Template

A property named ec-deploy under **Administrator > Server > Properties** contains a property sheet named ec_stack_templates, which contains several default categories: Application Server, Database Server, Host, Operating Systems, and Virtual Image, each of which contains several properties that you want to capture. For example, the Host category has properties such as memory and num_cpus for capturing a system's memory and number of CPUs.

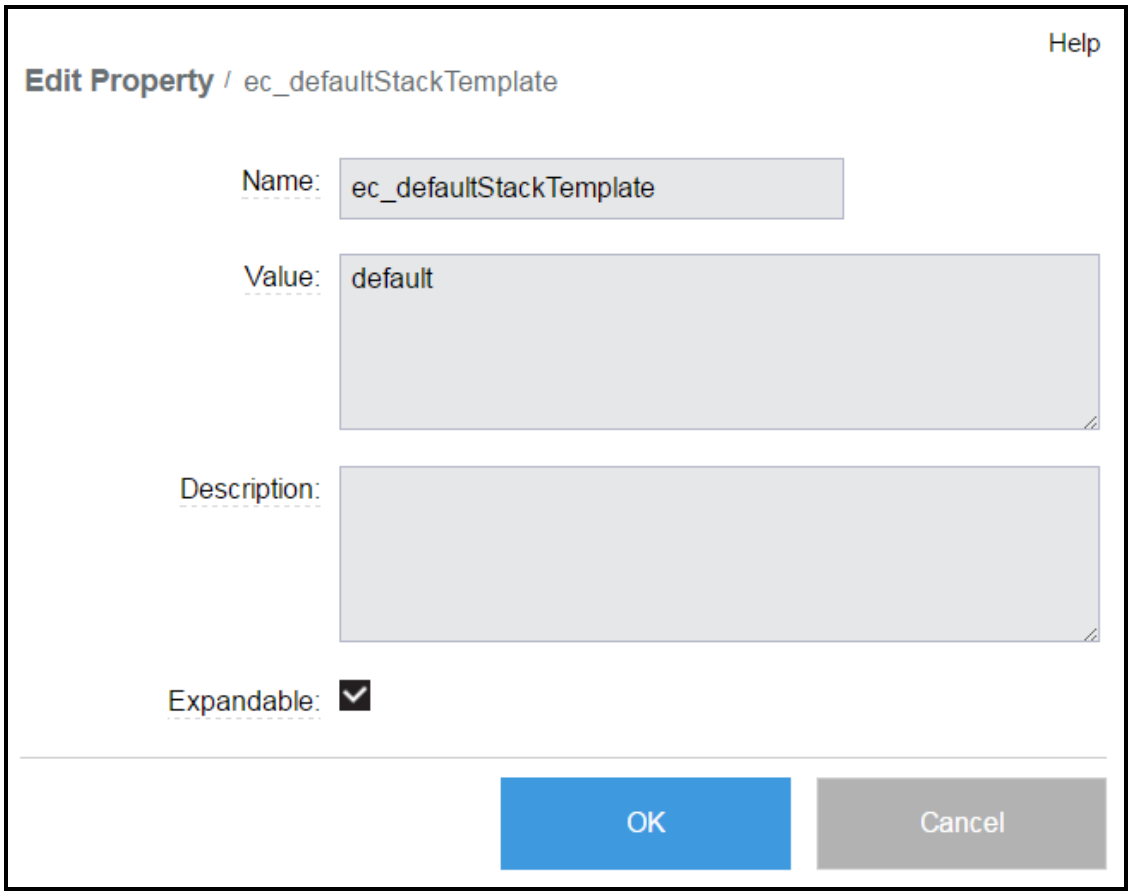
Modifying the Default Stack Template

You can add more properties. You can also create new categories. For example, you could create a category named Operating System and then beneath it, you could create a type that says whether it is a Linux or a Windows system.

A stack template does not contain values. It just determines the values that you want to capture from an application standpoint and from an environment standpoint. The values are stored as attributes.

You can specify two types of values: string or text values and a list of values. For example, to capture the operating system type, you could define a list of the following values: Linux, Windows, AIX, and Lattice.

Important: To prevent an ElectricFlow upgrade from overwriting customizations of the existing ec_defaultStackTemplate, you must copy that template into a new one. Then, you must set the ec_deploy/ec_defaultStackTemplate server property with your new template name.



Configuration Drift

Throughout the software release process, ElectricFlow tracks changes in the release to ensure that the applications are repeatedly and reliably deployed and that the release is successfully completed. ElectricFlow uses snapshots and change tracking to track the changes and identify differences, and then can notify users about them.

Snapshots are a version of the application at a point in time, encapsulating the specific state of the application, its environment states, its components, and the artifacts and artifact versions that define the components. You can manually take snapshots and compare two to determine the differences between them. You can also automate a procedure that takes snapshots on a schedule, create custom reports, and send them to users, making it easier to identify differences between deployments and resolve issues in the release. See Snapshots on page 119 for more information about snapshots.

Change tracking is another way to track differences between deployments. Change tracking can track the changes between every state of non-runtime objects and allows you to revert to any previous state of the objects, such as applications, application processes, components, artifacts, procedures, workflows, resources, and workspaces. In the Change History, you can find what changed for a specific object, such as an artifact, and when the change occurred. See Change Tracking on page 1374 for more information about this feature and how to use it.

Deployment Examples

This section has examples of application deployments and includes:

- How to identify ElectricFlow objects—These are easily identified by icons and colors in the Visual Editors (such as the Application Editor, Environment Editor, and Application Process Visual Editor) and lists (such as the Applications List and the Environment List).
- How to view details about the objects

  In the Application Editor and Environment Editor, you can view details and other information of the application tiers, components, environment tiers, and resources.

  In the Application Process Visual Editor and Component Process Visual Editor, you can view details about the process steps.
- How a process can take different paths in the deployment based on parameter inputs.
- What to expect with different deployment options.
- How to model dynamic environments that provision (commission) environments and spin them up at runtime.
- How to use master components.
- How to use snapshots.

Guidelines for Modeling and Deploying Applications in ElectricFlow

When Modeling Applications

Keep the following in mind:

- By default, change tracking is enabled for all ElectricFlow Deploy projects.
- When authoring a component process, you specify the process type. The default is **Deploy**.

  In the **Component Process** dialog box, select one of these values in the **Type** field:

  - **Deploy**–Enables Inventory Tracking. The ElectricFlow server tracks the artifacts deployed to environments.
  - **Undeploy**—The next time that the process is run, the ElectricFlow server removes information about the artifacts deployed to environments.
  - **Other**–Disables Inventory Tracking.
- You can also create a library of standardized reusable components to promote best practices across the enterprise and speed up application authoring with *master components*. This is an easy way to reuse existing definitions and their underlying process definitions. Go Master Components on page 67 for more information.

- After creating an application, you can version the entire application model by taking a snapshot of it. A snapshot is an immutable version of an application with specific artifact versions.

  You can save more than one snapshot of an application during the software release life cycle. You can compare snapshots to optimize and troubleshoot the application. You can also deploy a snapshot even if the latest version of the application has been modified.

  See Snapshots on page 119 for more information.

- You can include hyperlinks as part of an object description for any ElectricFlow object.
- Known issues:
  - If an application tier has two components that cannot be deployed to the same host, the components need to be in different application tiers.
  - If a formal parameter in a manual process step is renamed, the formal parameter properties are not updated.
- See Inventory Tracking on page 131 for more information.
- Go to Environment Inventory for more information.
- See Examples: Modeling and Deploying Applications on page 255 for an example of the Application Inventory in the Application View Run page.

When Modeling Environments

You can optimize how resources are used in application deployments. You can model dynamic environments that are automatically spun up when an application is deployed. These environments can have cloud and static resources. Cloud resources are provisioned using resource templates, which are specified in environment templates.

Using dynamic environments allows you to:

- Provide ways to optimize how cloud resources are used.
- Reuse provisioned resource pools.
- Track how provisioned cloud resources are used.
- Provide the status of the provisioning process.
- Verify the credentials of cloud resources before provisioning them.
- Configure the middleware of cloud resources on an on-demand basis.

When Deploying Applications

How an application is deployed depends on what you want to deploy. You can select the following deployment options.

- **New run**–Specify the runtime parameters and settings in the dialog box.
- **Last run**–Use the parameters from a previous run. You can modify one or more of these parameters as described in the dialog box where the runtime parameters and settings are set.

  > **Note:** The option does not appear the first time an application is deployed.

- **Schedule**–Set the application to run on a schedule as described in Running Applications with Schedules.

You can also deploy the application with these options and methods:

- Full deploy–ElectricFlow deploys all the objects (including application processes, components, and artifacts) in the application. When you deploy an application for the first time, you must do a full deploy. Later, you can use this method to verify that the software is ready to be released.
- Smart deploy–This deployment method reduces risk and time by only deploying changed objects. ElectricFlow automatically detects the differences between what is about to be deployed and what has already deployed to an environment. It deploys only those objects not yet deployed to a specific resource or to resources where the object version is different. You can use this method throughout the software release life cycle.
- Partial deploy—The system deploys only objects that you select. You can use this method to verify incremental changes.
- Partial deploy with specific artifact versions—The system deploys only the artifacts with selected versions. You can use this method to verify incremental changes.
- Schedule–You can create schedules to deploy applications on a one-time, daily, weekly, or monthly basis. You can schedule nightly builds so that the developers and quality always have a new version to work on every day.
- Snapshot–You select a snapshot to deploy. A snapshot is an immutable version of the application with specific artifact versions. You can save and compare snapshots throughout a release life cycle.

  > **Note:** If you select a snapshot and modify it, it is no longer a snapshot and becomes a different version of the application, which you can save as a new snapshot.

- Based on custom parameters defined in application processes–Set these parameters when you deploy the application or when you define an application process step. They determine how the application should be deployed.
- Artifact staging–ElectricFlow retrieves the artifacts that will be deployed in an application run before the deployment starts. This ensures that all the artifacts are downloaded and available on the target. It also minimizes the time it takes to retrieve artifacts and reduces downtime during the deployment. See Artifact Staging on page 99 for more information.
- Rollback–When an application deployment fails, you can use the rollback functionality to restore the deployment model to a previous state before the failure occurred. This operation is supported only in application processes. See Rollback on page 100 for more information.

You can deploy applications by combining methods, such as:

- Smart deploy and partial deploy
- Smart deploy, artifact staging, and a partial deploy with specific artifact versions
- Full deploy and artifact staging with a snapshot
- Smart deploy and artifact staging
- Artifact staging and partial deploy

For example, this sequence is an example of how you may deploy your software over time:

- When you create an application in ElectricFlow and deploy it for the first time, you must do a full deploy. By default, smart deploy is disabled the first time that you run an application.
- Throughout these deployments, you can schedule the deployments to occur on a daily or weekly basis to provide builds that can be tested and installed on a regular basis.
- You can also save snapshots of the software at regular intervals or milestones. When you need to troubleshoot the software or want to do performance testing, you can use one of these snapshots for comparison.
- If the application does not deploy successfully, you can redeploy parts of it to troubleshoot the application or component processes that failed.

  You can do a partial deploy and redeploy the application with only the objects that failed.

  You can also do a partial deploy only with specific versions of artifacts to determine if one or more specific versions of artifacts are causing problems.
- Later, after successfully deploying the application, you can redeploy parts of the application when new versions of artifacts or new resources are available.
- When a new version of an artifact is released, you can deploy only the artifact by selecting the new version and doing a partial deploy.
- When you add artifacts and resources to the application, you deploy the new artifacts to resources and specific versions of selected artifacts to the new resources, a combination of smart deploy and partial deploy with specific artifact versions.

Getting the Real-Time Status of Application Runs and Troubleshooting

Use the Environment Inventory and Application Inventory to view the progress of the application as it runs and the results of previous runs. They show detailed results that can be used to troubleshoot the application. See Inventory Tracking on page 131for more information.

- In the Application Inventory (on the Applications Run View page), you can get information about the application, its application processes, components, and job steps and about the status of these objects.
- In the Environment Inventory, you can get more details about the environment, the applications mapped to it, number of deployed artifacts in the applications, where the artifacts are deployed, and the status of these objects.

Use Change Tracking to troubleshoot a failed job, look for more information about non-run object, or look for differences between objects. See Change Tracking on page 1374 for more information.

Compare snapshots to refine and troubleshoot an application. See Snapshots on page 119 for more information.

Examples: Modeling and Deploying Applications

You can model applications using one of the following:

- Tiers and Components
- Services and Containers

You can use Change Tracking to track all the changes for non-runtime objects such as the application model, components, artifacts, resources, the environment model, and projects. See Configuring Change Tracking on page 1375 for more information.

When defining a component or container, you specify the artifact repository (plugin) and the artifact version that you want to use. ElectricFlow supports artifacts from a variety of Artifact Repositories, such as the EC-Artifact repository, Maven based repositories such as Nexus or Artifactory (using EC-Maven), or artifacts on any file system (using EC-FileSysRepo). These artifacts could point to any build outputs, configuration scripts, database, scripts, and so on. The information that you enter in the component or container definition depends on the artifact repository that you select. For more information about artifacts, see Artifact Management on page 1323 and Artifacts on page 1092.

Applications consist of services and the processes that orchestrate the deployment of these services. Each application object has one or more application services, where a service is a logical group of containers. A container is based on an artifact, and the container definition contains a reference to this artifact.

When using containers and services, you can:

- Create services and containers under applications
- Create clusters in environments (ElectricFlow supports GCE and ECS)
- Map services to clusters and set provider-specific values

These examples show how to model and deploy an application in ElectricFlow.

- Creating a New Application on page 256
- Authoring Application Processes on page 268
- Creating Environments on page 275
- Defining Tier Maps and Cluster Maps on page 282
- Deploying and Troubleshooting Applications on page 287
- Viewing the Real-Time Progress of Deployments on page 296
- Troubleshooting Deployments on page 298

Creating a New Application

Applications consist of components and the processes that orchestrate the deployment of these components. Each application object has one or more application tiers, where a tier is a logical group of components, and one or more services, where a service is a logical grouping of containers.

A component is based on an artifact, and the component definition contains a reference to this artifact.

This example shows how to create a “hybrid” application consisting of an application tier with a component and two services with a container in each service.

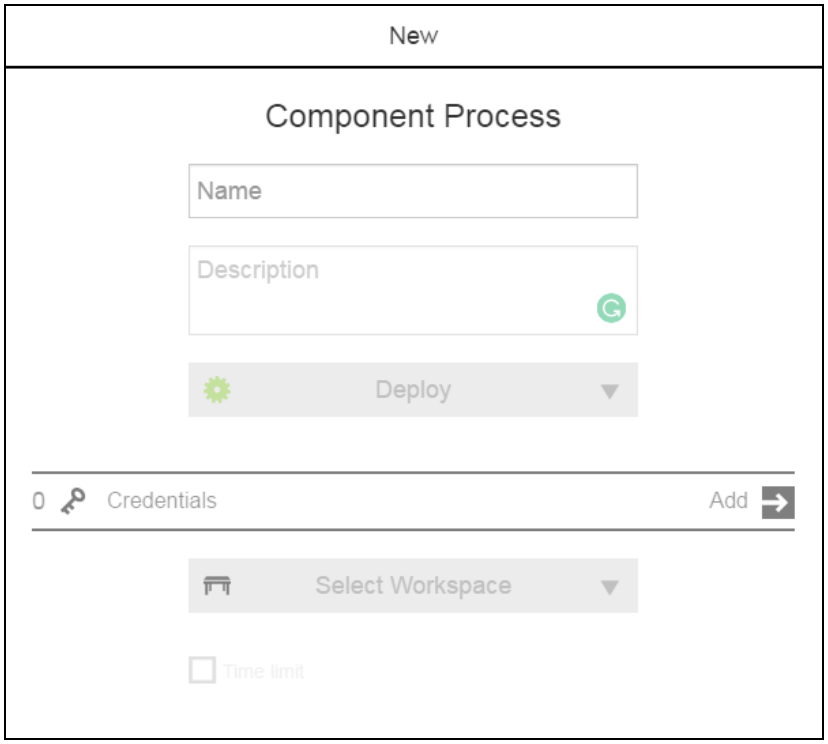
Authoring Component Processes

In a component process, you define a set of actions that will be executed on a specific component during the deployment. A component process step can be defined as a call to an ElectricFlow plugin or procedure, a direct command or script in any scripting language, a manual task, or a utility function (a higher-order operation than a plugin).

In the Cleanup DB component of the "DB" tier:

1. Click  to start authoring a component process.

   The **Component Process** dialog box appears:



2. In the **Component Process** dialog box, enter the following details:
   - Enter "updateDB" in the **Name** field.
   - Enter an optional description in the **Description** field.
   - Keep the selection as **Deploy** in the **Type** field. This value determines track what is built, tested, and deployed during the application deployment. The default is **Deploy**.

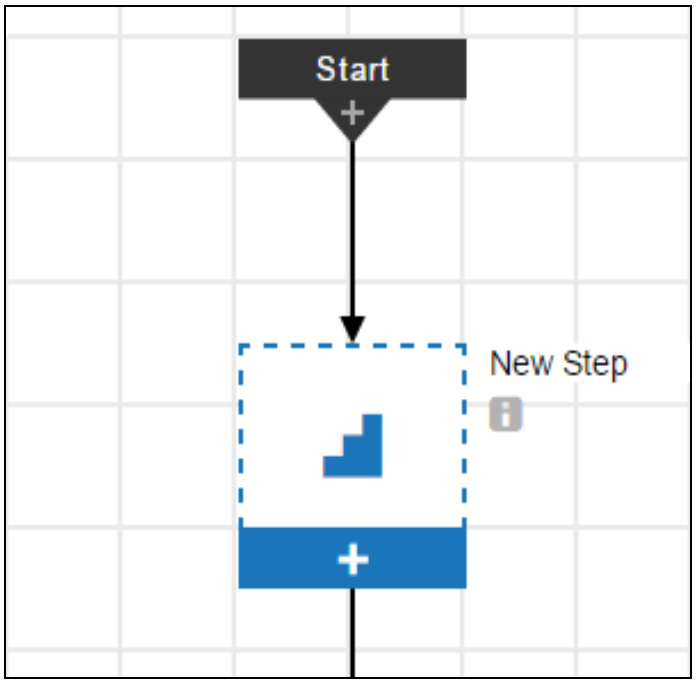
3. Click **OK**.

   The Component Process Visual Editor opens.

   The process model shows a process with **Start** and **Finish** steps with a **New Step** in between them.

   > **Note:** Clicking **i** under the step name displays the details about the process step.

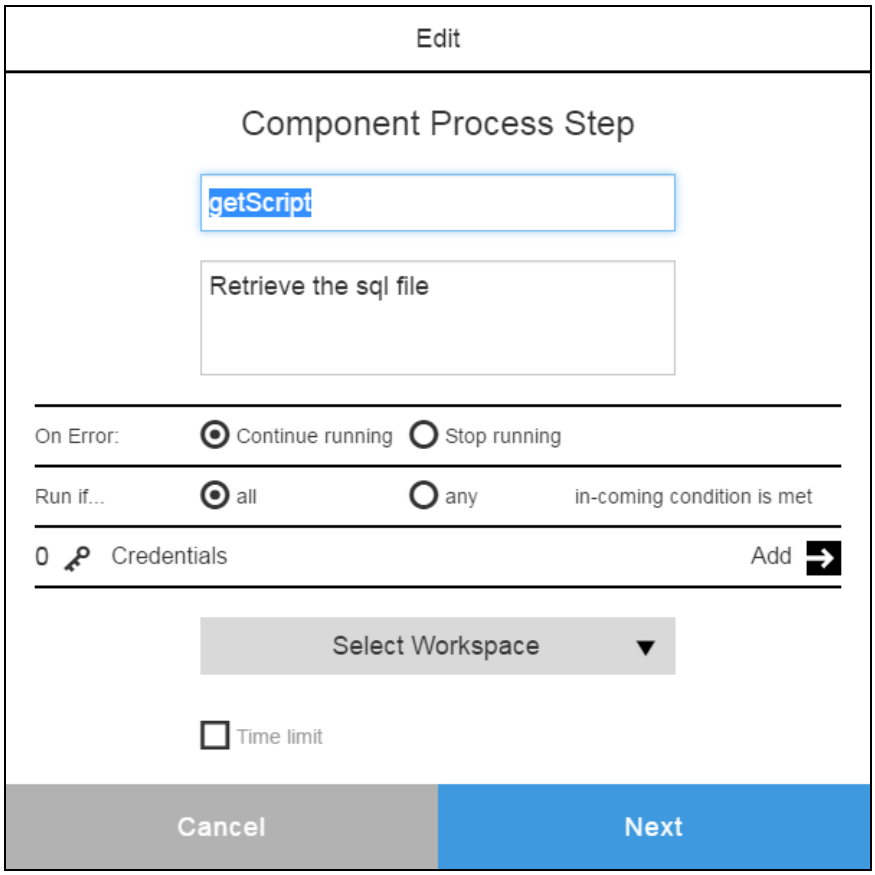
   Example:



4. Click **+** in the first step.

5. Define the first step in the Component Process Step dialog box:
    1. Enter "getScript" in the **Name** field.
    2. Enter "Retrieve the new sql file" in the **Description** field.
    3. Use the **Continue Running** setting in the **On Error** field. Go here for more information.
    4. Use the **all** setting in the **Run if** field. Go here for more information.
    5. Click **Next**.

    Example:



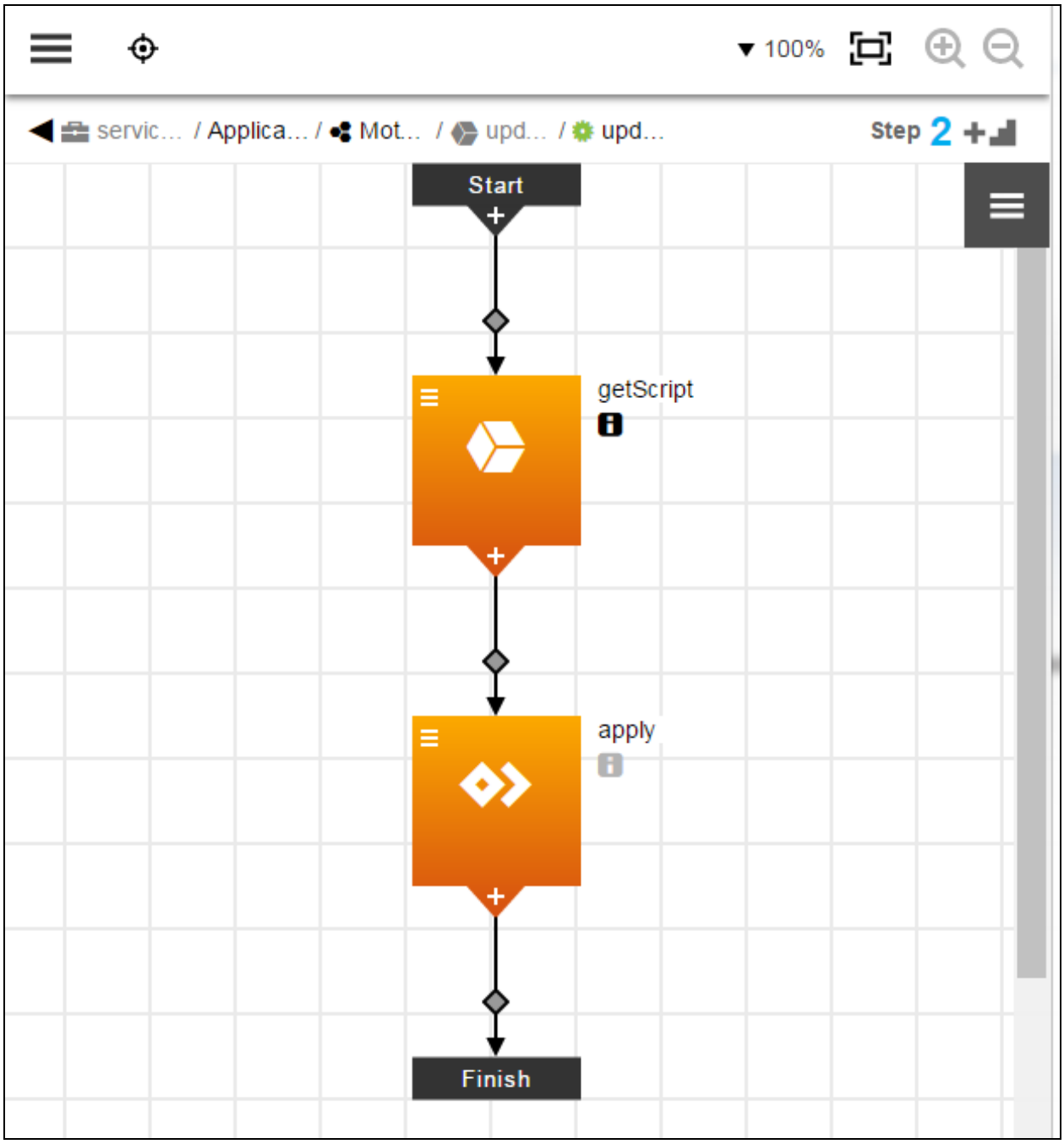
6. Define the first step in the Component Process Step dialog box based on a component operation, select **Component Operations** as the **Step Type**. The parameters for this process step are automatically inherited from the "update.sql" component.

   Example:

7. Click **OK**.
8. Click **+** on the bottom of the first step to add a new step below it.
9. Click **+** in the second step.
10. Define the second step in the Component Process Step dialog box:
    1. Enter "apply" in the **Name** field.
    2. Use the **Stop Running** setting in the **On Error** field. Go here for more information.
    3. Use the **all** setting in the **Run if** field. Go here for more information.
    4. Click **Next**.

11. Define the second step in the Component Process Step dialog box based on a component operation by selecting **Procedure** as the **Step Type**. The parameters for this process step are automatically inherited from the "update.sql" component.
12. Click **OK**.

This is the component process:



- You can get details about each step by clicking **i** under the step name.
- The first step is a component operation.

  The second step is a procedure.
- These steps are deployed in series.

  The second step can only start after the first step is completed.

- Clicking  displays menu where you can select options:
  - **Details**–View step details
  - **Properties**–View step properties.
  - **Access Control**–View access control settings.
  - **Add Connector**–Add another connector to the step.
  - **Track Changes**–View the Change History.

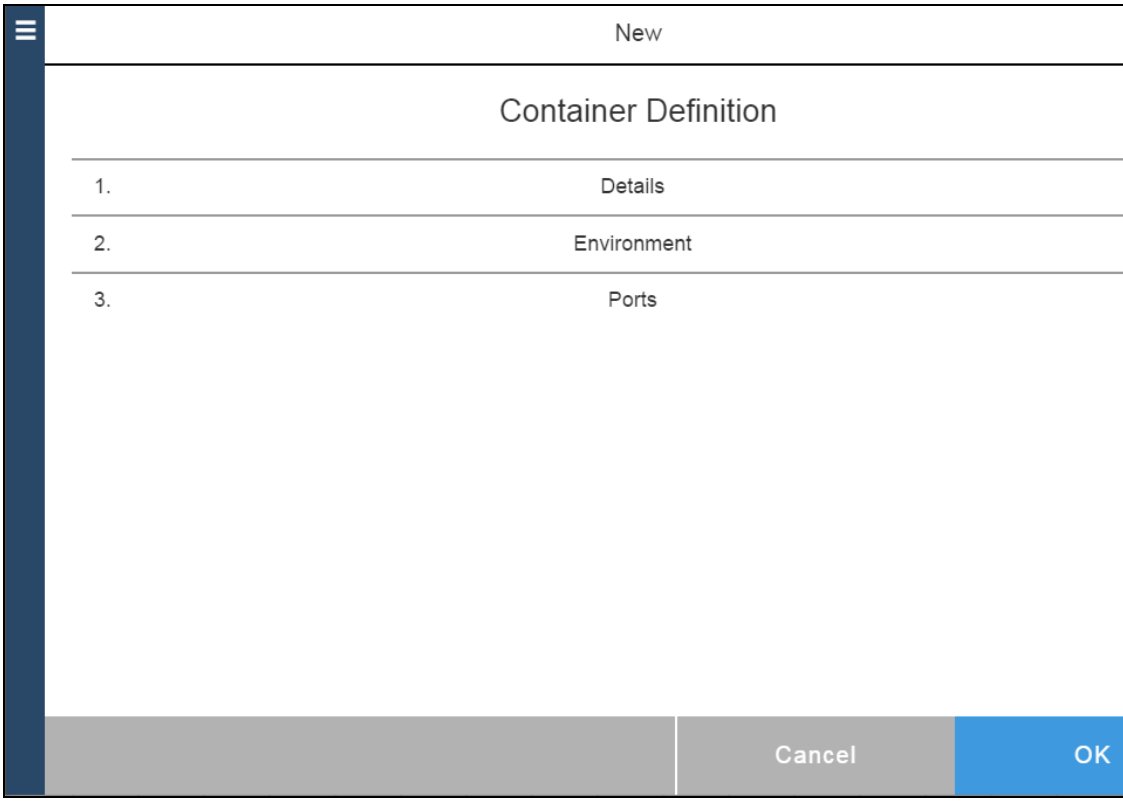
1. Define an application service and the containers in it.

   A new service named Service1 appears.

   In Service 1, click  and select **Details** to edit the details in the **Container Definition** dialog box.

1. Rename the application service to "store-front-end" and click **OK**.

2. To define the container:
   - Click **+** under the **Container**.
   - Click **Create new container**.
   - Enter `Config` as the container name, and click **Next**. The **Container Definition** dialog box appears:



   - Click the "down" arrows to fully expand the dialog box.

- Enter the container attributes as needed. The following table describes the available attributes.

Label	Description	Default
Container Definition		
Registry Type	Public: Specifies that your container images are stored in a public Docker registry. Private: Specifies that your container images are stored in registries other than a public Docker registry.	Public
Registry URL	(Optional) Private container registry URL. This is the registry where your image is stored. For example, `http://gcr.io`.	–
Registry Credential	(Optional) Credentials (username and password) required to access the private registry. You must create a credential under a project and attach to it (by providing the absolute path when using ectool or by browsing and selecting it in the UI).	–
Image	Relative or absolute path to the container image. For example, `username/image`.	–
Version	(Optional) Version of the image. For example, `v1.0`. This will be used as a tag when you publish an image to Docker Hub. If a tag is not provided, Docker tags it as `latest`.	`latest`

Volume Mounts	(Optional) Volume mounts required by the container. Use JSON format. For example:
	`[{`
	`"name": "scratch-space", // required attribute. To be used to reference volume definition at service level.`
	`"mountPath": "/tmp/cache" // required attribute. Mount location for volume in container.`
	`}]`
		–

Label	Description	Default
Minimum CPU Requested	(Optional) Amount of CPU (in number of cores) requested by the container. Specifying the amount of CPU needed allows the underlying container service to make better placement decisions when deploying the services on the nodes in the cluster based on the available capacity. If no value is specified, the underlying container service determines the default.	–
Maximum CPU Allowed	(Optional) Maximum amount of CPU (in number of cores) that the container may use. If the limit is exceeded, the behavior depends on the container service.	If this is left empty, the platform-specific default is used.
Minimum Memory Requested	(Optional) Amount of memory (in MB) requested by the container. Specifying the amount of memory needed by the container allows the underlying container service to make better decisions when deploying the services on the nodes in the cluster based on the nodes' available capacity. If no value is specified, the underlying container service determines the default.	None. However, on ECS, the plugin will use a default soft memory limit (memory request) of 128 MB.

Label	Description	Default
Maximum Memory Allowed	(Optional) Maximum amount of memory (in MB) that the container may use. If the limit is exceeded, the behavior depends on the container service.	If this is left empty, the platform-specific default is used.
Entry Point	(Optional) Comma-separated list of one or more values to override the `ENTRYPOINT` instruction in the Docker image at runtime. `ENTRYPOINT` specifies a command that will always be executed when the container starts. For example, `/bin/ping`. For details about how `CMD` and `ENTRYPOINT` interact, see https://docs.docker.com/engine/reference/builder/#/understand-how-cmd-and-entrypoint-interact.	–
Command	(Optional) Comma-separated command arguments to use with the container's entry point. If specified, they override the `CMD` instruction defined in the container image. `CMD` specifies arguments that will be passed to the `ENTRYPOINT` (for example, `localhost`).	–
Environment		
Variable	(Optional) Environment variable name used in the container. For example, `TEMP_FOLDER` or `DB_PORT`.	–
Value	(Optional) Default value for the environment variable. You can override these values during the service-to-cluster mapping phase.	–
Ports		
Name	(Optional) Logical name for the container port. For example, `http` or `https`.	–

Label	Description	Default
Container Port	(Optional) Port that the container is listening on.	–

- Click **OK**.

2. Click the button to add a service named store-backend as described above and enter the container attributes as needed.

Authoring Application Processes

At the application (parent) level, you author application processes. When deploying an application, the application process that you select is executed to orchestrate operations against the application. An application process step can be defined as a call to an ElectricFlow plugin, procedure, or component process as well as a direct command, script, a manual task, or a utility function (a higher-order operation than a plugin).

Property Picker

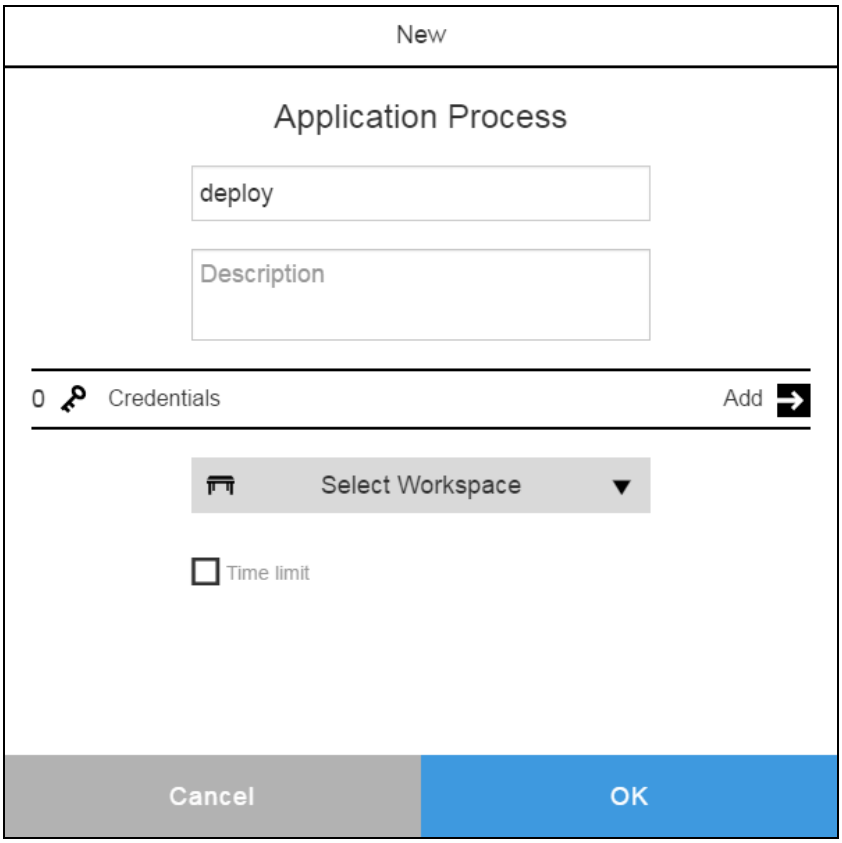
The Property Picker is available from application processes to make it easy to create custom deployments. It includes properties set on the project.

- The Property Picker contains a pull-down menu to let you select project or related object properties.
- The Property Picker lets you select built-in or custom properties or parameters in process step conditions that you need to reference without remembering and writing out property names and paths.

In the Application Editor for the Upgrade application:

1. Click  to start authoring an application component process.
2. In the Application Process Details dialog box, enter the following details:
   - Enter "deploy" in the **Name** field.
   - Enter an optional description in the **Description** field.
   - Keep the selection as **Deploy** in the **Type** field. This value determines track what is built, tested, and deployed during the application deployment. The default is **Deploy**.
     - **Deploy**–Enables Inventory Tracking. The ElectricFlow server tracks artifacts deployed to environments.
     - **Undeploy**–After the first successful job step in a component process with this setting, the automation platform removes the environment inventory record.
     - **Other**–Disables Inventory Tracking.

   Example:



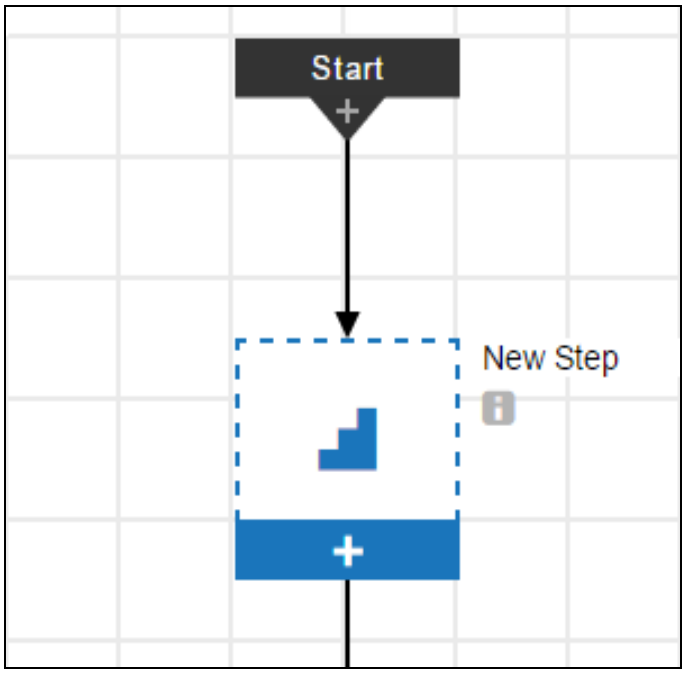
3. Click **OK**.

   The Application Process Visual Editor opens.

   The process model shows a process with **Start** and **Finish** steps with a **New Step** in between them.

   > **Note:** Clicking **i** under the step name displays the details about the process step.

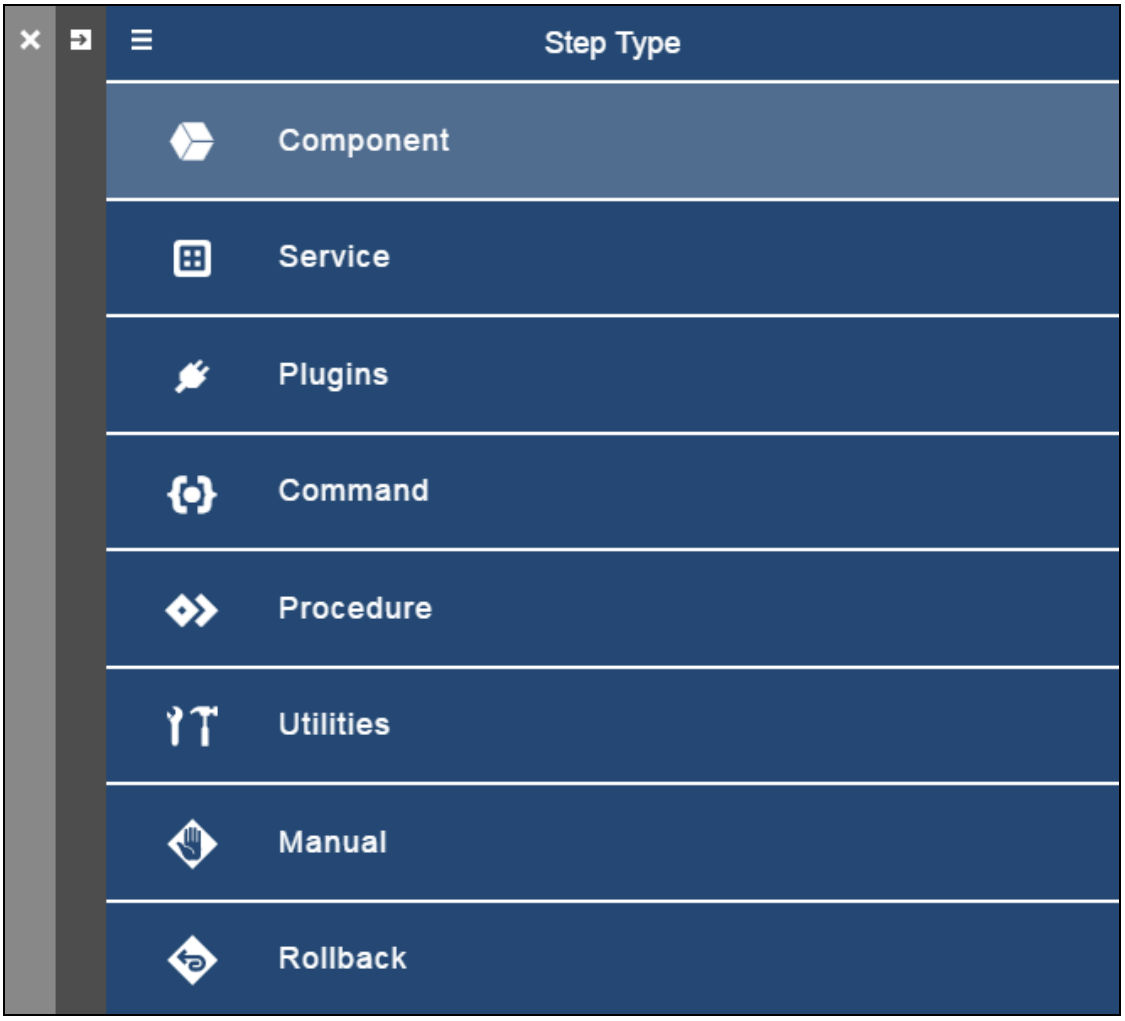
   Example:



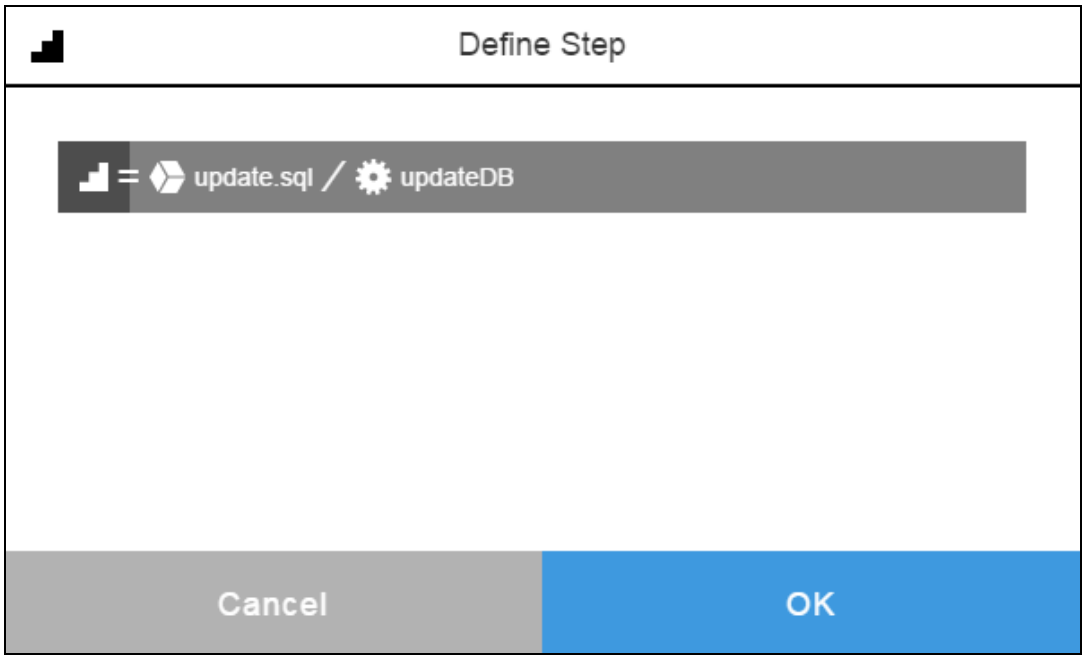
4. Click **+** in the first step.

5. Define the first step in the Application Process Step dialog box :
    1. Enter "applyDBChange" in the **Name** field.
    2. Use the **Stop running** setting in the **On Error** field. Go to Application Deployment Options on page 125for more information.
    3. Use the **all** setting in the **Run if** field. Go When defining a step for an application or component process, you must select the actions if a step fails: on page 126 for more information.
    4. Click **Next**.

       The **Step Type** dialog box appears:



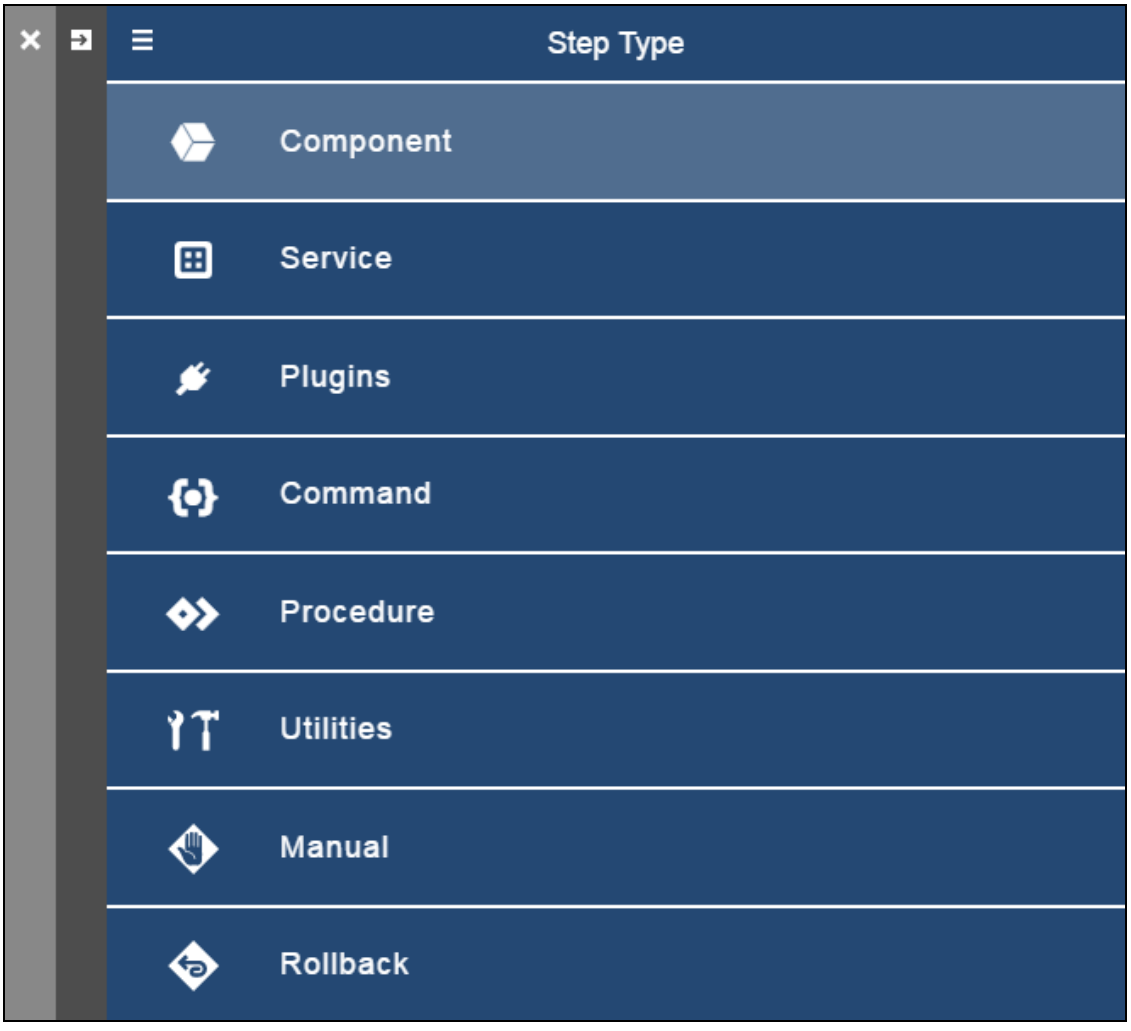
6. Define the second step in the Application Process Step dialog box based on a service by clicking **Component** > **update.sql**> **updateDB**. The step is defined by the component Deploy process for the update.sql component:



7. Click **OK**.
8. Click **+** on the bottom of the first step to add a new step below it.
9. Click **+** in the second step.

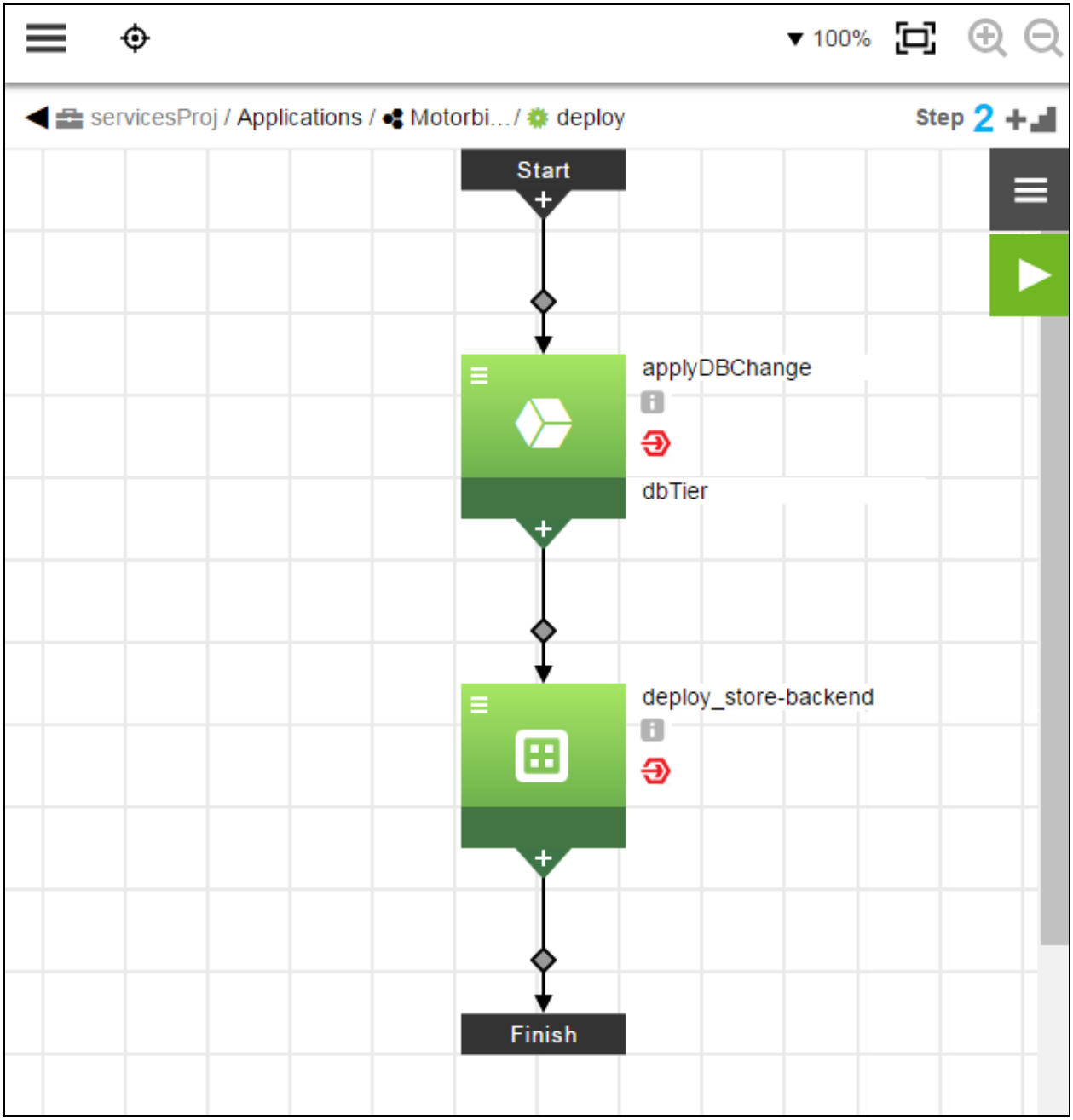
10. Define the second step in the Application Process Step dialog box as follows:
    1. Enter "deploy_store-backend" in the **Name** field.
    2. Use the **Stop running** setting in the **On Error** field. Go When defining a step for an application or component process, you must select the actions if a step fails: on page 126 for more information.
    3. Use the **all** setting in the **Run if** field. Go When defining a step for an application or component process, you must select the actions if a step fails: on page 126 for more information.
    4. Click **Next**.

       The **Step Type** dialog box appears:



    1. Define the second step in the Application Process Step dialog box based on a service by selecting **Service** > **store-backend**. The step is defined by the component Deploy process for the motorbikeMS service.
    2. Click **OK**.

This is the application process called "deploy.":



- You can get details about each step by clicking under the step name.

- Clicking  displays a menu where you can select the following options:
  - **Details**–View step details
  - **Properties**–View step properties.
  - **Access Control**–View access control settings.
  - **Add Connector**–Add another connector to the step.
  - **Track Changes**–View the Change History.

Creating Environments

Environments are logical grouping of machines to which various applications can be mapped and deployed. Using environments and templates, you can model the infrastructure and the middleware available for various applications. Environments consist of logical groups of resources with a similar function or role called environment tiers. An environment tier can be a group of machines with a similar function or role, such all the web servers or application servers or database servers for an environment. Resources are actual target end point machines (such as physical servers), virtual machines, or mobile devices.

Environments can be static, dynamic, or hybrid. A *static environment* has resources that are already provisioned and managed at the platform level. Each resource has its own logical name to identify it from the other resources in the system. It also can be assigned to one or more resource pools or to a zone (a collection of agents). Several resources can correspond to the same physical host or agent machine. Resources can also be configured as *standard* or *proxy*. Standard resources are machines running the ElectricFlow agent on a supported agent platform while proxy resources (agents and targets) are on remote platforms or hosts that exist in your environment and requires SSH keys for authentication. The ElectricFlow agent does not need to run on the remote platform or host. See Resources on page 1208 for more information about to create, configure, and manage resources.

This example shows to how model a static environment to which the application will be deployed. For information about dynamic environments and how to model them, see Deploying Applications in Dynamic Environments on page 343.

Creating Environment Tiers

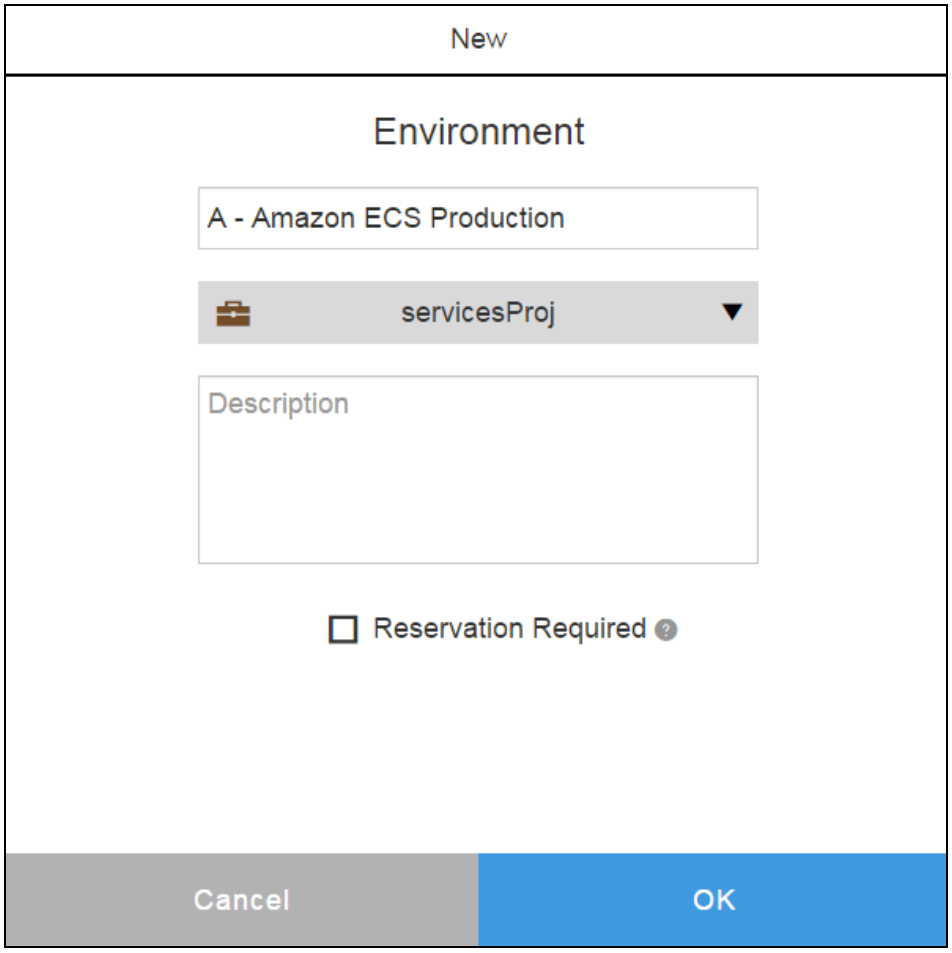
In the Environments List:

1. Click **Add +** (in the upper right corner) to add an environment.
2. The **Environment** dialog box appears.
3. Enter 'A—Amazon ECS Production' into the **Name** field.

4. Select a project to which the new environment will belong in the field and enter a description of the environment.

   Tip: You can include hyperlinks as part of an object description for any ElectricFlow object.

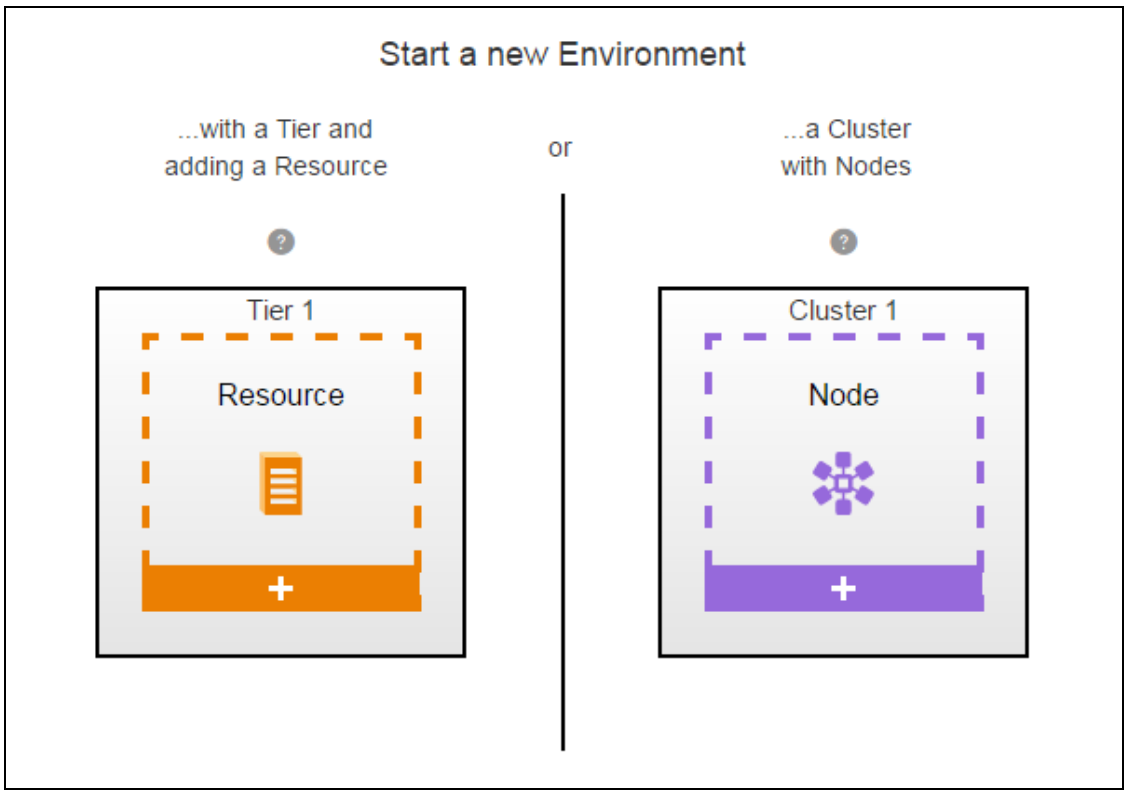
   Example:



   This dialog box lets you quickly define an environment tier and the resources in it or a service and the nodes in it.

5. Click **OK**.

   The Environment Editor opens.



6. Click **Tier 1**.

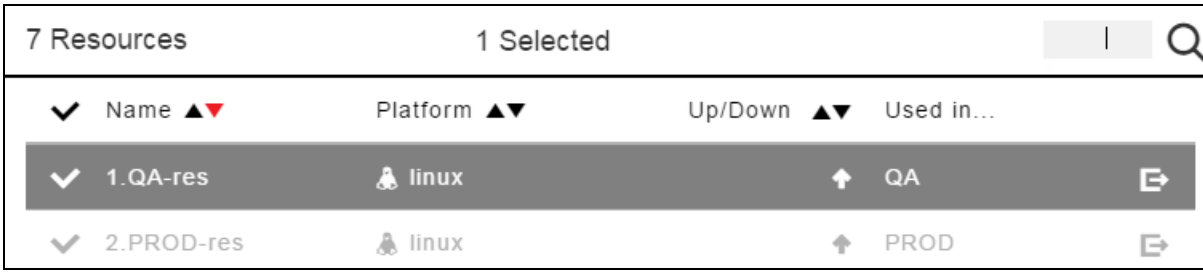
7. Define an environment tier and assign resources it.
    1. In the New dialog box, click **Add resources**.
    2. Click , then click **Details**, then rename the environment tier to "mysql," and then click **OK**.
    3. To define the resource:

        Click **+** under the **Resource**.

        Click **Add resources**.

        Select the resource that you want to assign to the environment tier, and click . A resource is available if it is enabled.

        Example:



The Environment Editor now has an environment tier called "mysql" with one resource.

Example:

Creating Environment Clusters

1. Click **+** at the bottom of Cluster 1. The **Cluster Definition** dialog box appears:
2. In the **Cluster Definition** dialog box, select a platform. For example, Amazon EC2 Container

Service.

3. Enter the cluster attributes as needed. The following table describes the available attributes.

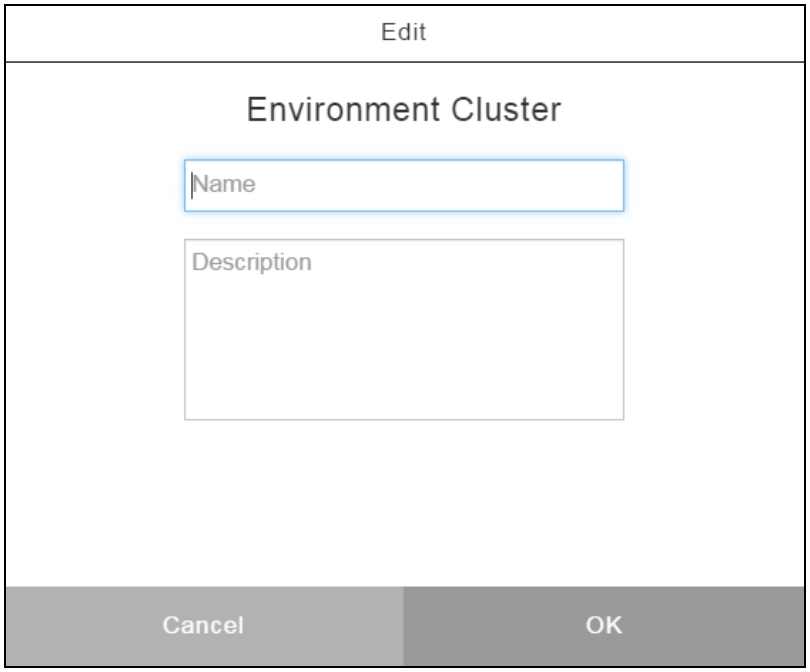
Description	Default
Configuration	Name of an existing configuration which holds the connection information for Amazon ECS.
Container Cluster Name	Name of the cluster to be provisioned for in Amazon ECS.
Description	(Optional) Description of the cluster that needs to be provisioned.
Desired Capacity	Number of EC2 instances that should be running in the group.
Maximum Size	(Optional) Maximum size of the group.
Minimum Size	(Optional) Minimum size of the group.
VPC Subnet Ids	(Optional) Comma-separated list of subnet identifiers for your virtual private cloud (VPC) in which to launch the EC2 instances.
Availability Zones	Availability zones in which to launch your EC2 instances.
Image	ID of the Amazon Machine Image (AMI) to use to launch your EC2 instances.
Instance Type	Instance type of the EC2 instance.
Security Groups	One or more security groups with which to associate the instances.
Key Name	(Optional) Name of the key pair.
Associate Public IP	(Optional) Checkbox that specifies whether to associate a public IP address.
Container Instance IAM Role	ECS container instance IAM role for the launched container instances to use.

Description	Default
Results Property Sheet	Name of the property sheet to hold results.

4. Click **OK**.

5. In the cluster that you just created, click  > **Details**.

   The **Environment Cluster** dialog box appears:



6. In the **Environment Cluster** dialog box, enter a cluster name into the **Name** field. For example, `ecs-cluster1`.
7. Click **OK**.

Defining Tier Maps and Cluster Maps

Before deploying application that use an application tier or an application cluster, you must define a tier map to associate the application with an environment.

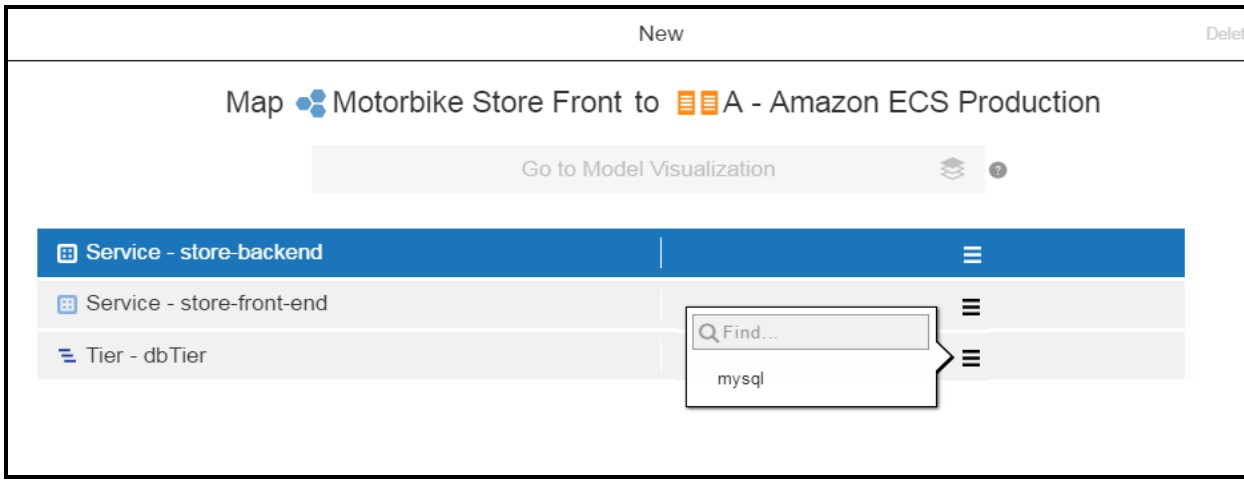
Defining an Application Tier Map

In the Application Editor for the application that you want to deploy:

1. Click **Environment**.
2. Example: Select Amazon ECS Production.

3. Map the application tier to the environment tier:

- In the application tier, click in the corresponding environment tiers column and select an environment tier. Select one from the list or enter the search criteria in the **Search** field.



- Click **OK** when the application tier is mapped to an environment tier.

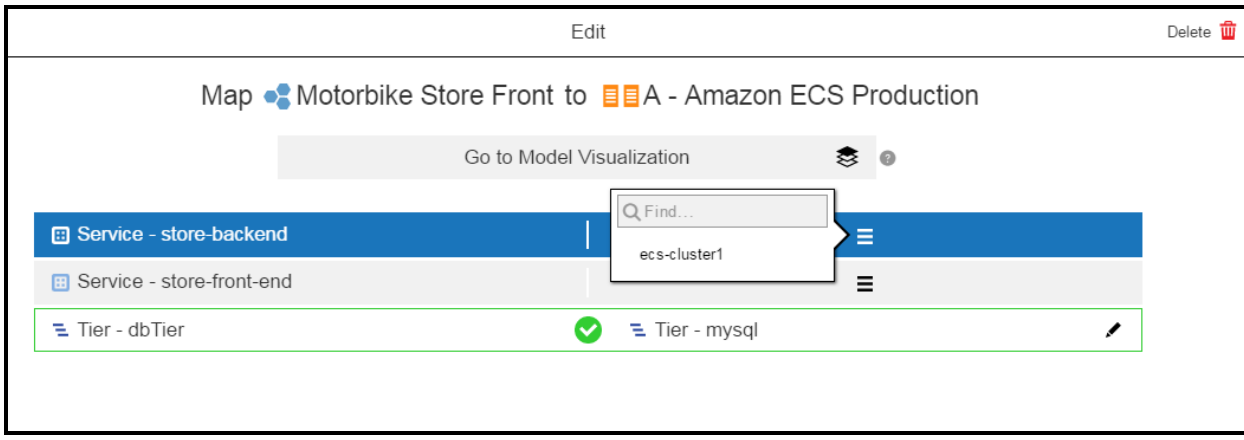
Defining an Application Cluster Map

You can override the service and container attributes and add platform-specific service and container attributes.

1. Map the store-backend to an environment tier:

- In the service, click in the corresponding environment tiers column and select an environment tier. Select one from the list or enter the search criteria in the field.

  Example:

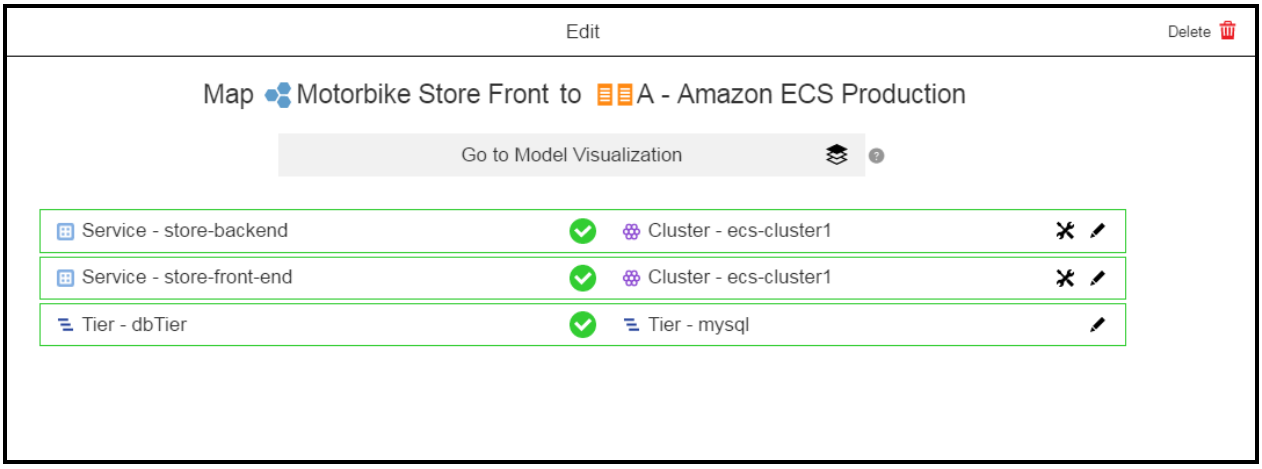


- Click **OK** when the application tier is mapped to an environment tier.

2. Map the store-front-end service to an environment tier as described above.

   The application now has one tier map and two cluster maps.

   Example:



3. Click  to configure the cluster map for the store-backend service.

4. Enter the cluster map attributes as needed. The following table describes the available attributes.

	Label	Description	Default
Service Configuration Details **You can use these fields to override the service and container attributes and add platform-specific service and container attributes.**			

	Volume	(Optional) Volume pointing to a storage device or a partition that can be mounted by a container. Use JSON format. For example:
		`[{`
		`""name"": ""web-content"",`
		`""hostPath"": ""/var/html""`
		`Location on host/container instance where volume is expected to be already attached.`
		`}]`
			–

	Number of Service instances	(Optional) Number of service instances to use.	1
	Rolling Deployment—Min Service Instances	(Optional) Minimum number of services that can be brought down during a rolling deployment.	If this is left empty, the platform-specific default is used.
	Rolling Deployment—Max Service Instances	(Optional) Maximum number of services that can be created during a rolling deployment.	If this is left empty, the platform-specific default is used.
Port Mapping			

	Label	Description	Default
	Name	Reference to the container port name.	–
	Container Port	Reference to the container port.	–
	Listener Port	External or listener port mapping for the container port. A listener port setup is required for pods to communicate with each other.	–
Amazon EC2 Container Service Specifications			
	LoadBalancer Name	Name of the load balancer on Amazon EC2. Required if the service contains port mappings.	–
	IAM role for Service and Loadbalancer	Name or full Amazon Resource Name (ARN) of the AWS Identity and Access Management (IAM) role that allows ECS to make calls to your load balancer on your behalf.	–

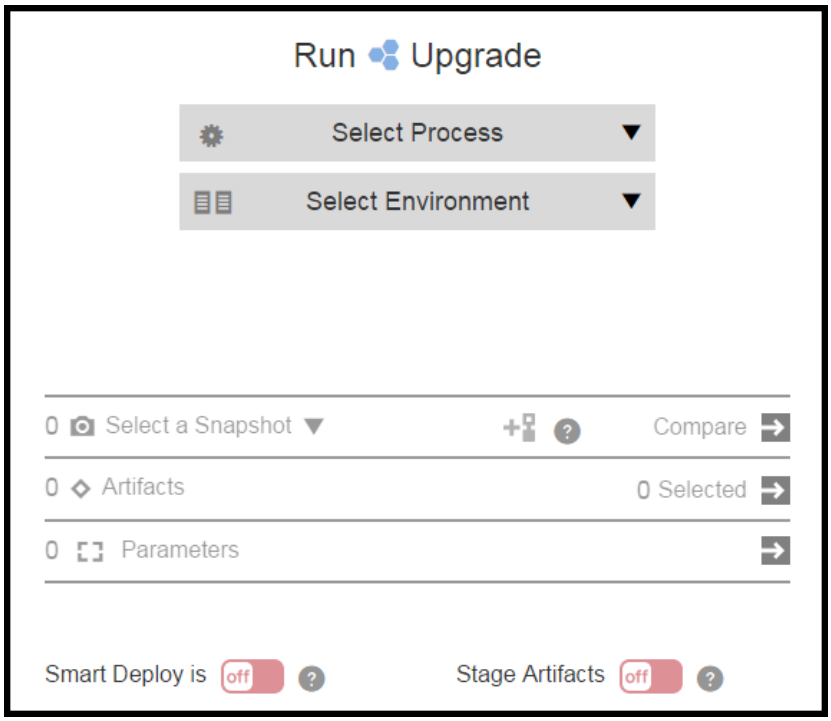
5. Click  to configure the cluster map for the store-front-end service as described above.

Deploying and Troubleshooting Applications

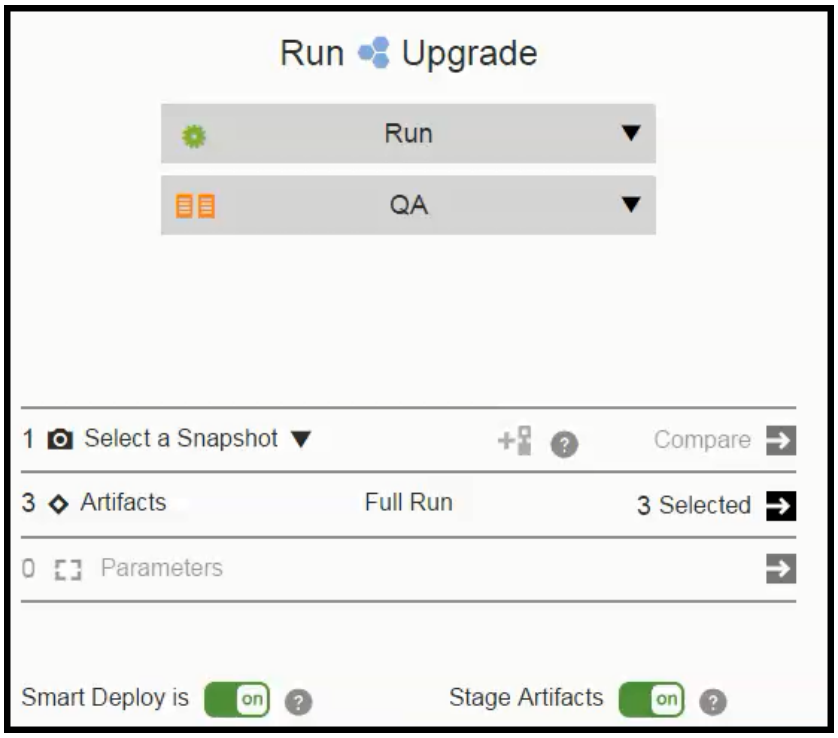
This example shows how to deploy the application multiple times with different runtime conditions.

First Run

In the Applications List, clicking  and selecting **New Run** opens the dialog box where you can set the runtime settings.



- In the Select Process field, select **Run**.
- In the Select Environment field, select **QA**.
- You can select a snapshot in the **Select a Snapshot** field if you have one or more.
- For the first run, all artifacts will be deployed.
- **Smart Deploy** and **Stage Artifacts** are enabled.

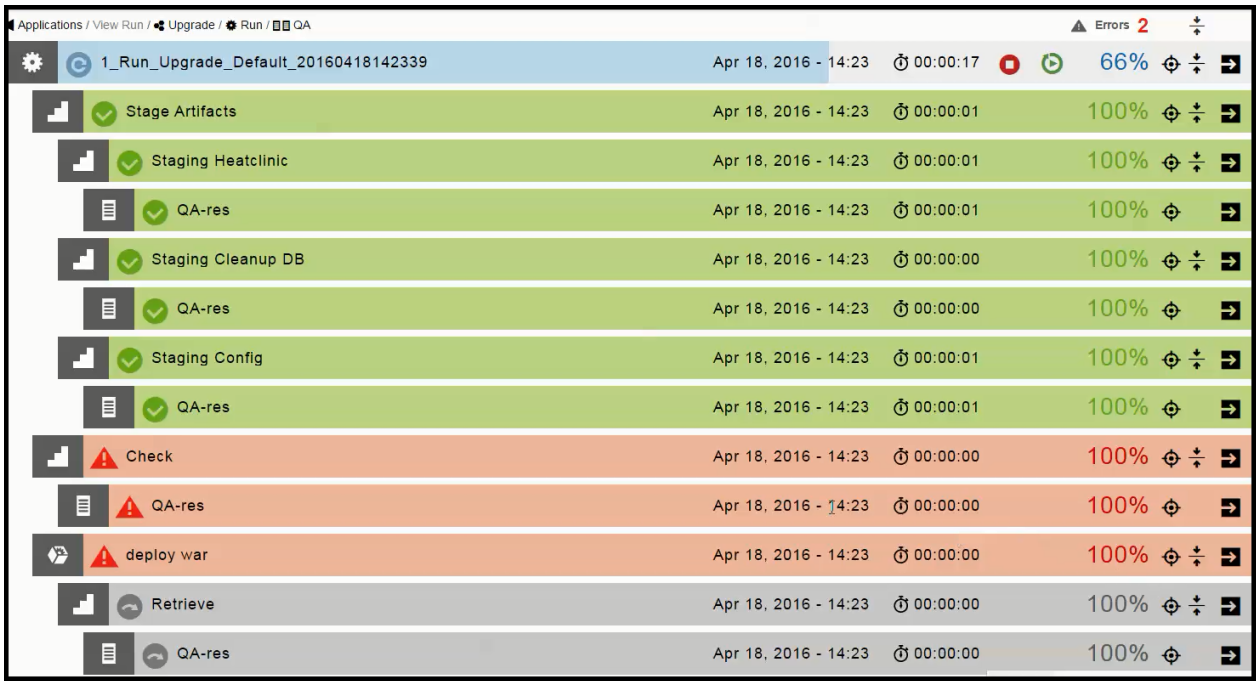


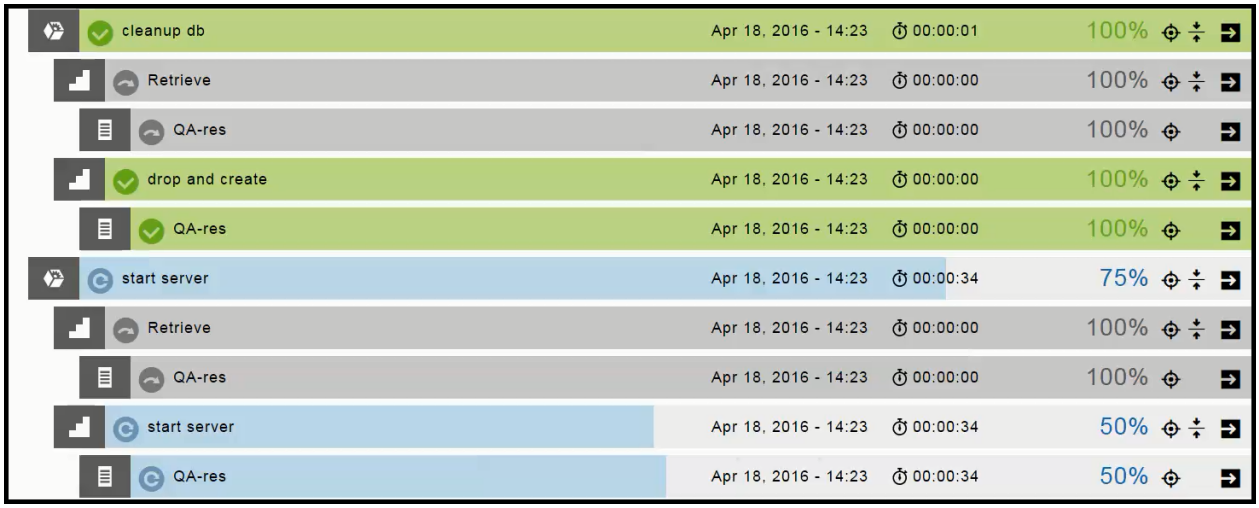
Click **OK** to start the application run.

The Applications View Run page shows the progress of the application run:

- The blue steps are running (in progress).
- The green steps were completed successfully.
- The red steps failed.

  Clicking  opens the Job Details page where you can begin the troubleshoot the step.
- The gray steps were skipped.





Subsequent Runs

After the first application run, you can deploy the **Run** process to the **QA** environment with other deployment options.

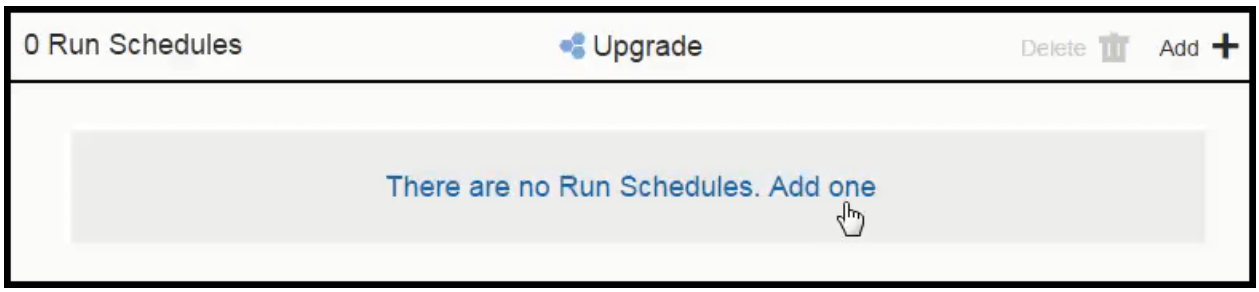
Based on the Last Run

Clicking and selecting **Last Run** opens the dialog box where you can set the runtime settings. It displays the runtime settings from the previous. You can deploy the application using the same settings

as the previous run or you can modify one or more them for the current run.

Based on Schedules

Clicking and selecting **Schedule** opens the list of schedules for the application. If there are no schedules, the list looks like this:



Clicking **There are no Run Schedules. Add one** (or **Add +**) opens the Run Schedule dialog box.

- Select **Create new** to create a new schedule, and then enter the schedule details.
- Select **from Previous Runs** to create a schedule based on a previous application run.

  A list of the last five application runs appears:

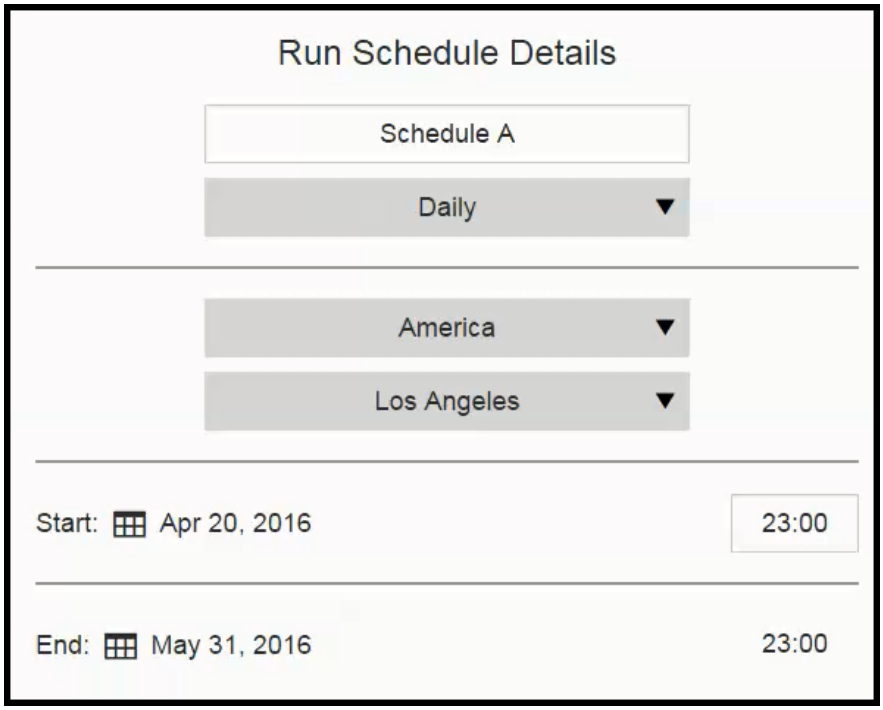
  Select an application run, and then enter the schedule details.

In the **Run Schedule Details** dialog box, enter the schedule details:

- In the **Schedule name** field, enter the schedule name.
- In the **Frequency field** field, click the down arrow and select **Once**, **Daily**, **Weekly**, or **Monthly**.
- In the **Country** field, click the down arrow and select a value in the drop-down list.
- In the **City** field, click the down arrow and select a value in the drop-down list.

- In the **Start** field, click  if you want to change the start date, and click in the **hh:mm** box to set the start time using the 24-hour clock.

- In the **End** field, click  and select the end date from the pop-up calendar. The end time is the same as the start time.

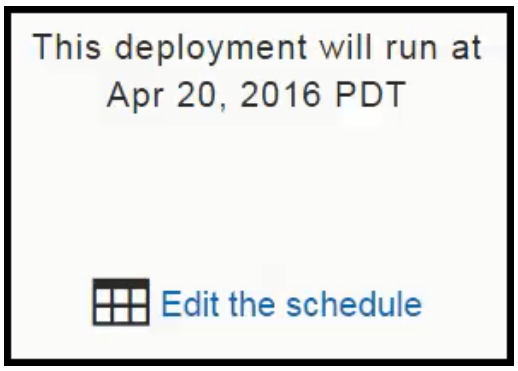


- Click **Next**.

  The dialog box where you can set the runtime settings appears. You can also modify the runtime settings at this time.

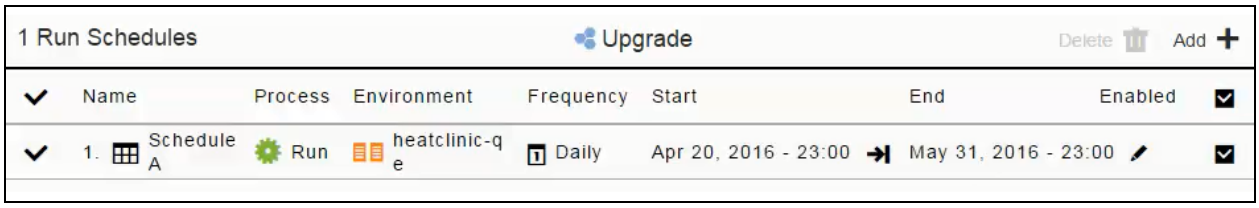
- Click **OK**.

  This message appears:



  Click OK to close it.

When you view the list of schedules for the application again, the schedule that you created appears in the list.



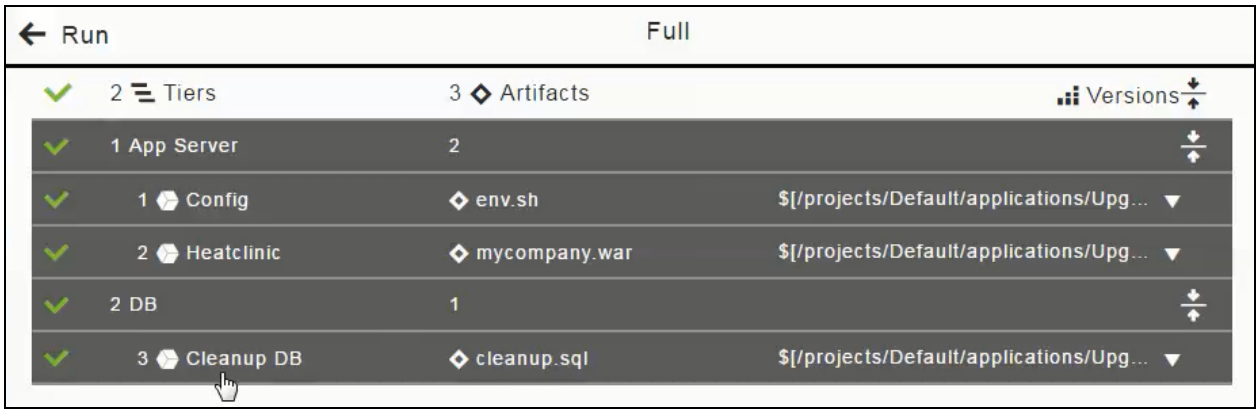
Runtime Settings

In the dialog box where you can set the runtime settings, the **Artifacts** field looks like this for a full run:

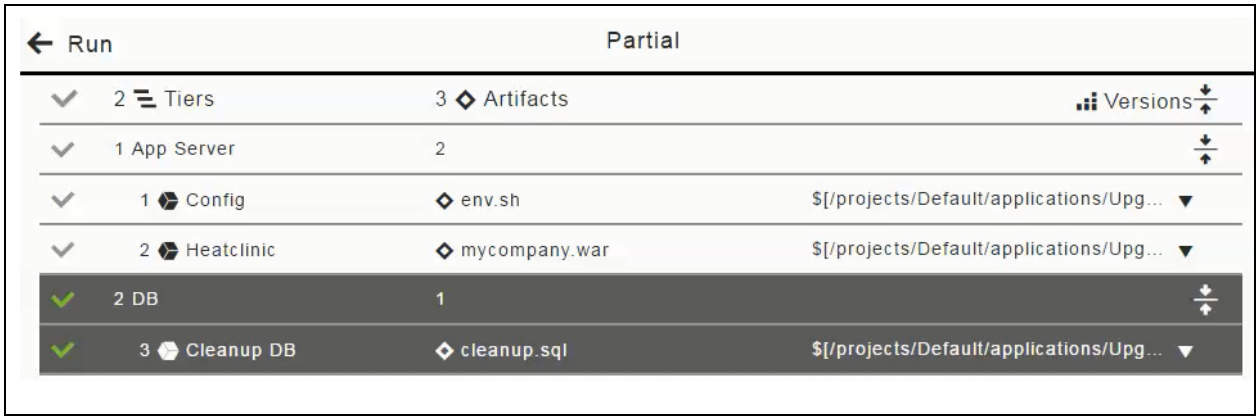
- Partial deploy with smart deploy and artifact staging are enabled:

  Clicking opens the list of artifacts and containers that can be deployed in the application process. Artifacts are grouped by application tier. You can enable or disable an artifact within a tier or the entire tier itself as part of a deployment.

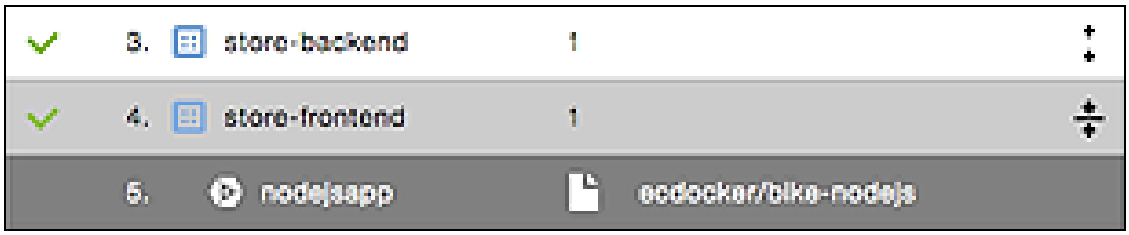


If you want to only deploy the DB artifact (cleanup.sql), click in the "1 App Server" column to deselect (disable the selection of) the Config and Heatclinic components (based on the `env.sh` and `mycompany.war` artifacts), and click **OK**.



The **Artifacts** field now looks like this for a partial run:

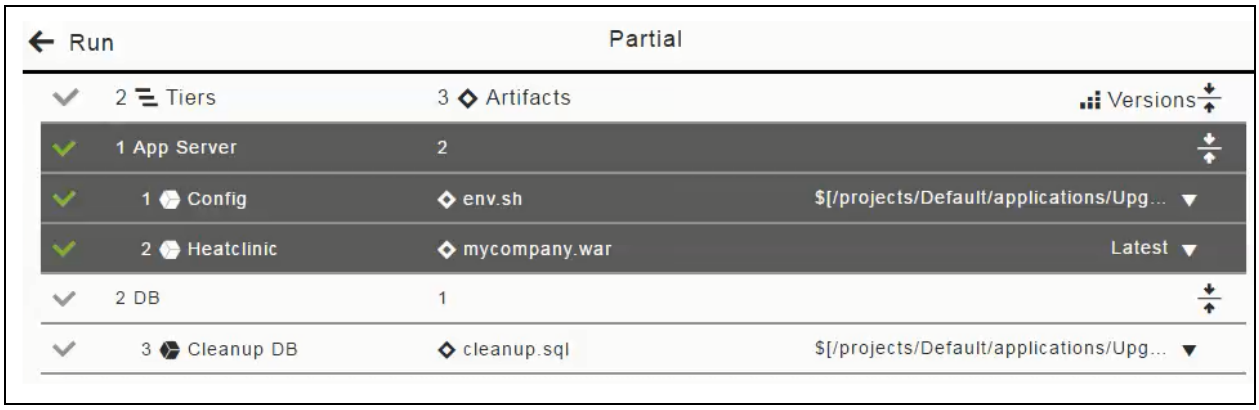
With microservices and containers, you can enable or disable a service:



(You cannot disable an individual container within a service.)

- Partial deploy with specific artifact versions and with smart deploy and artifact staging are enabled:

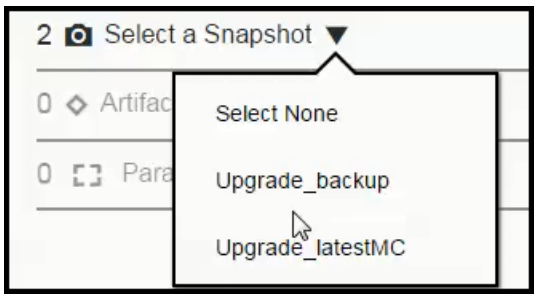
  In the list of artifacts to be deployed in the application process, click in the "2 DB" row to deselect the "Cleanup DB" component (based on the `cleanup.sql` artifact), select the **Latest** version of the `mycompany.war` artifact, and click **OK**.



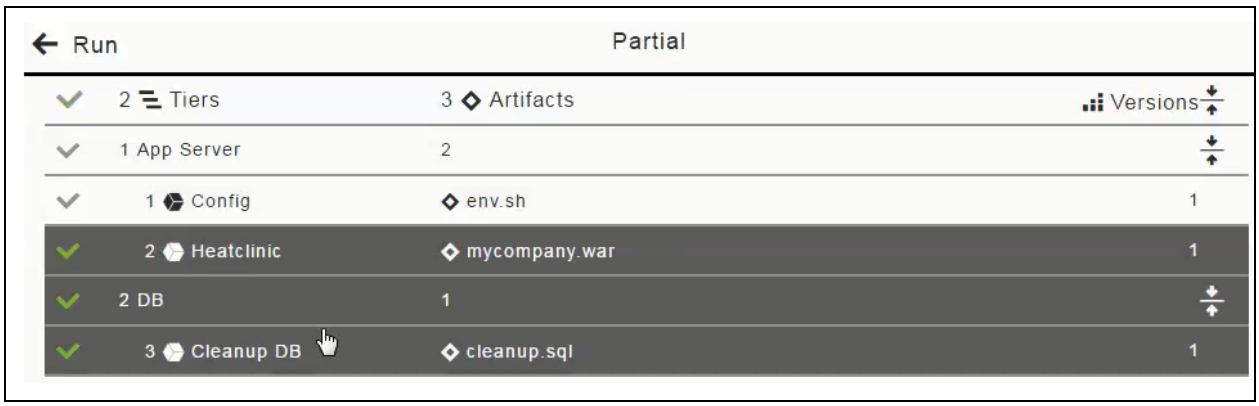
The **Artifacts** field now looks like this for the partial run:

- Snapshot with partial deploy and with smart deploy and artifact staging are enabled:

  In the **Select a Snapshot** field, click the down arrow and select the **Upgrade_latestMC** snapshot.



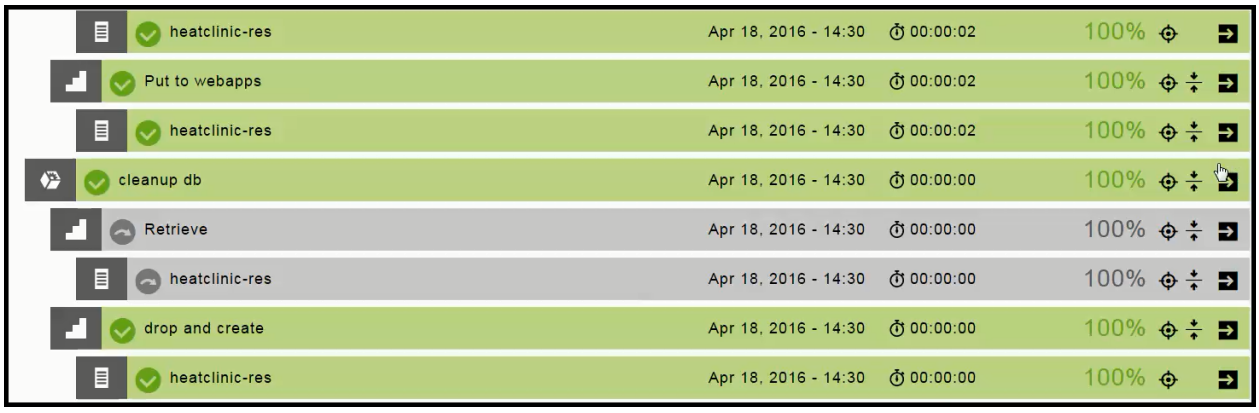
  In the list of artifacts to be deployed in the application process, click in the "1 Config" row to deselect (disable the selection of) the Config component (based on the `env.sh` artifact) and click **OK**.

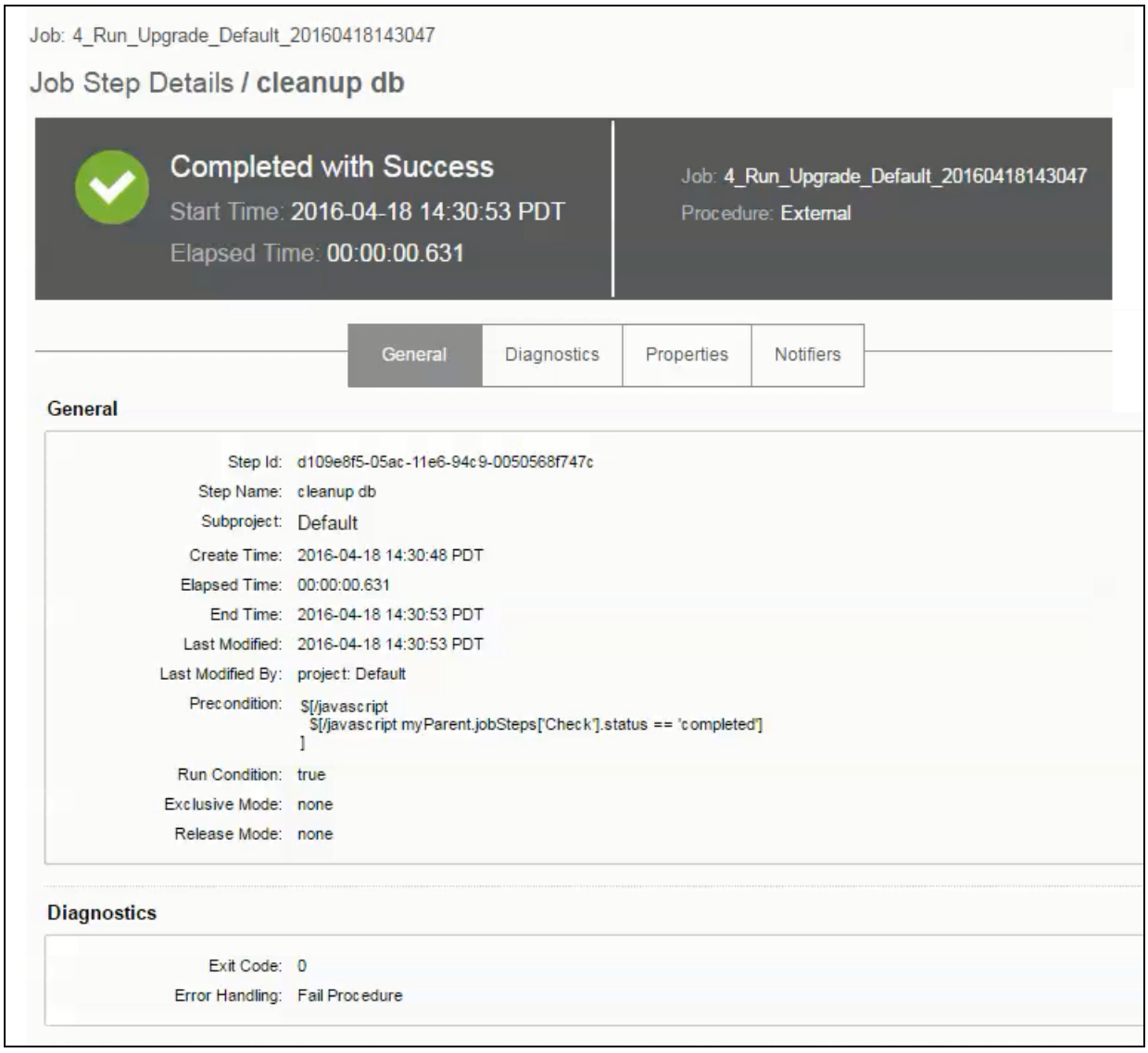


  The **Select a Snapshot** and **Artifacts** fields now look like this for the partial run:

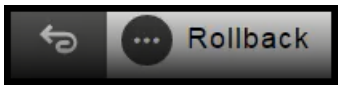
Viewing the Real-Time Progress of Deployments

This example shows the job details for a specific step. Clicking  in the "cleanup db" step opens the Job Details page:

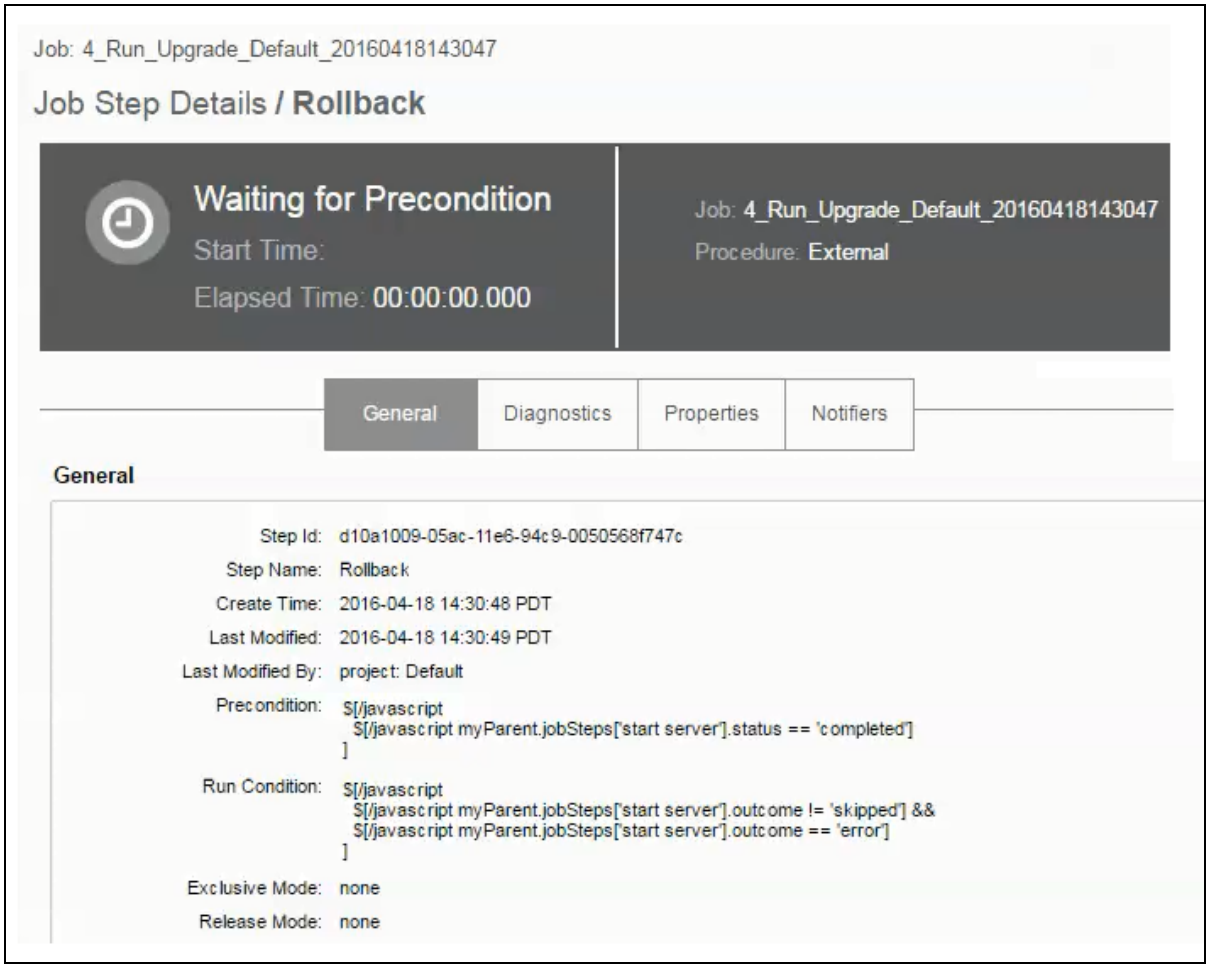




This example shows that the rollback step has not run yet:



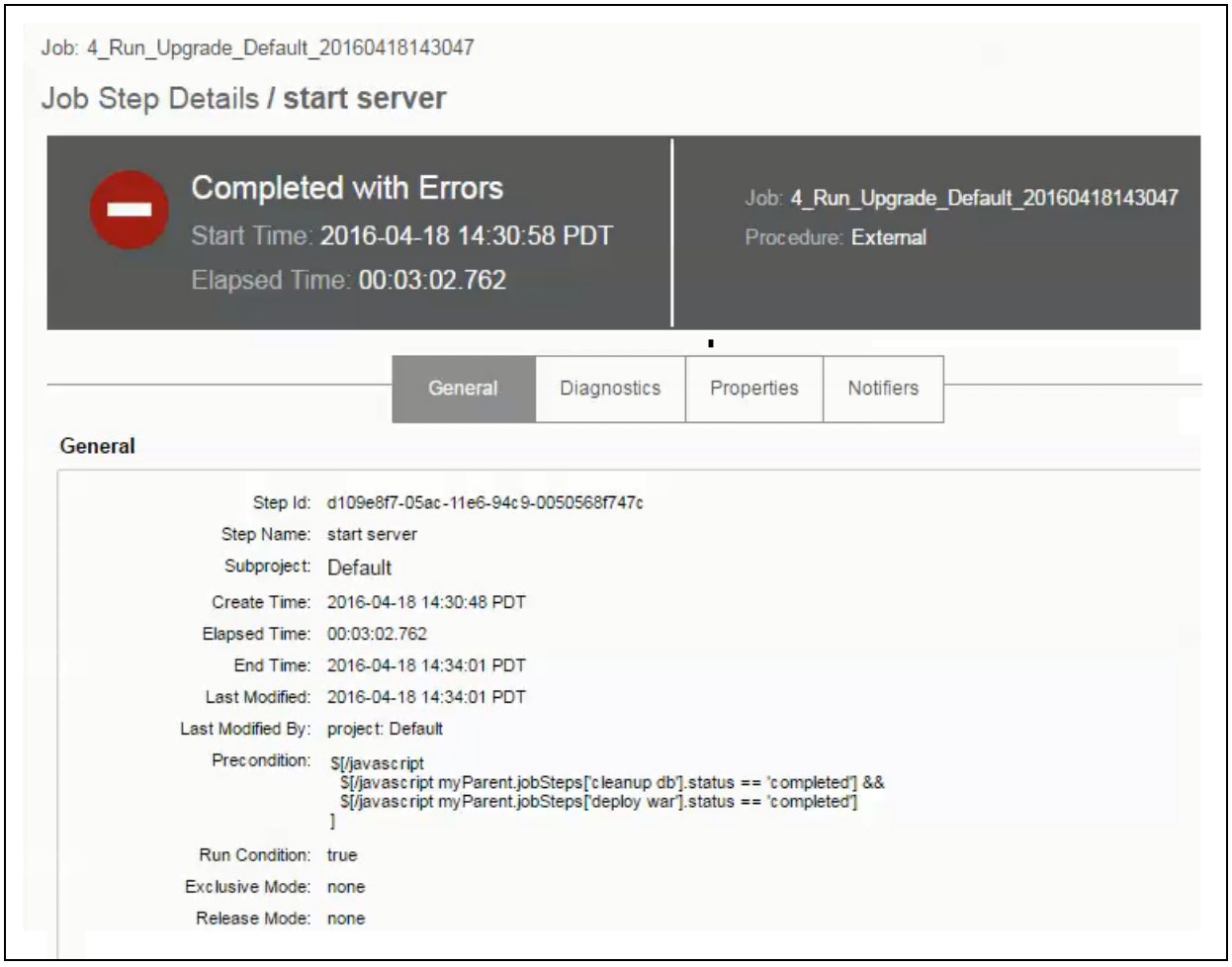
Clicking in the rollback step opens the Job Details page, showing that the status is *Waiting for Precondition*. This step will not run until the precondition is met.



Troubleshooting Deployments

This example shows that the "start server" step was *Completed with Errors*.

Clicking  in the "start server" step opens the Job Details page.



Clicking in the Diagnostics tab can provide more information about the errors.

Example: Manual Steps with Runtime Parameters

This topic describes manual steps with parameter inputs and how the parameter inputs influence the process flow to take different paths. You can define manual steps in an application process or component process.

In the following example, an application process called Deploy has the following steps. When deployed, this process updates the website for an online store.

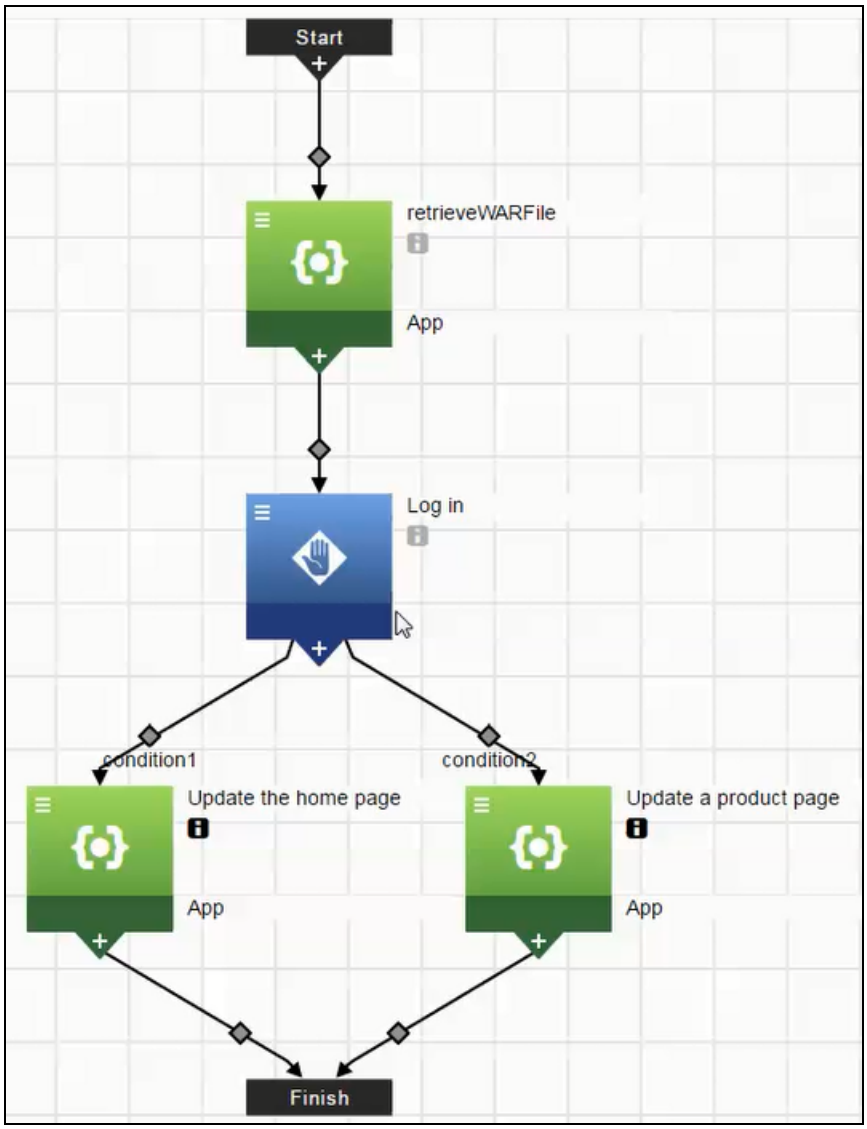
- The first step is a command step to retrieve a .war file that will be deployed to the website.
- The second step is a manual step to log into a terminal where you can enter CLI commands to update the website.

  If you have *admin* privileges, you can update the home (parent) page. If you do not have privileges to update the home page, you can update only the product (child) pages.

  Depending on your privileges, the parameter input to the manual step is either *pass* (you have the privileges to update the home page) or *fail* (you can only update the product pages).
- The parameter inputs to the manual step are the preconditions to the third and fourth steps.
  - *condition1* is fulfilled if the input to the manual step is *pass*.
  - *condition 2* is fulfilled if the input to the manual step is *fail*.

  These preconditions determine the step that is executed after the manual step.
- The next step is either the third or fourth step.

  If the parameter input for the manual step is *pass*, the process continues to the "Update the home page" step. If the input is *fail*, the process continues to the "Update a product page" step.



Process Flow Details

After starting the deployment, the .war file is automatically retrieved in the first step.

In the second step, email notifications are sent to one or more assignees. One of the required assignees performs the manual task, which is to log into a terminal and enter the appropriate CLI commands to set up the website update. After clicking on the link in the email, the assignee goes to the ElectricFlow UI and provides input to the manual process step using the `pass_fail_value` parameter.

- If the assignee has admin privileges, the value of the `pass_fail_value` parameter is *pass*.

  The process continues to the "Update the home page" step.

- If the assignee does not have admin privileges, the value is *fail*. The process continues to the "Update a product page" step.

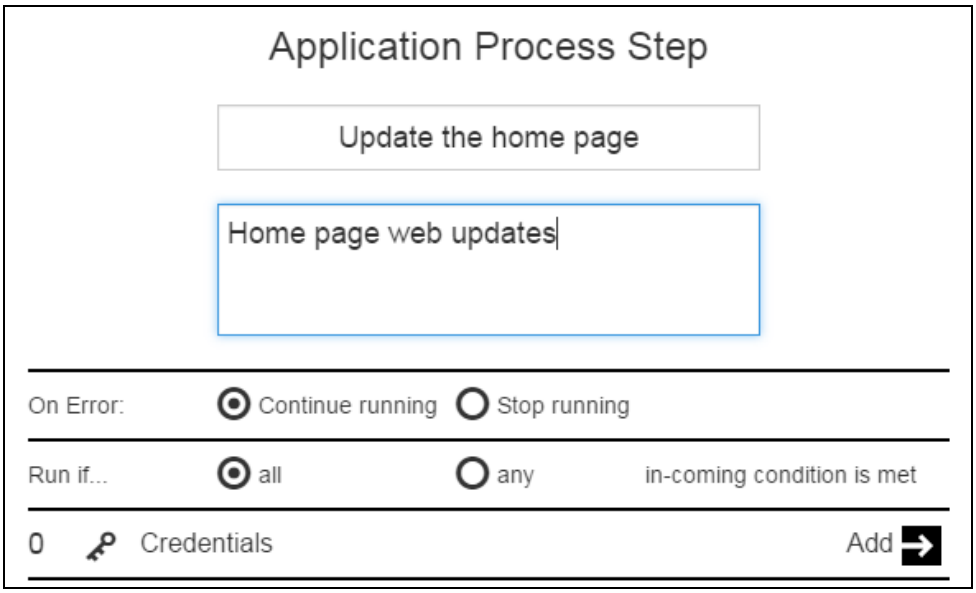
In the third and fourth steps:

- The **On Error** setting is **Continue running**.

  If the process encounters an error while the web page is being updated, it completes the step with errors and then continues to next step. The default is **Stop running**, where the process will abort if the process encounters an error.

- The **Run if** setting is **all** (the default). All the preconditions to the step have to be met before the process goes to the next step.

  If **any** is selected, only one of the conditions has to be met before the process continues running.

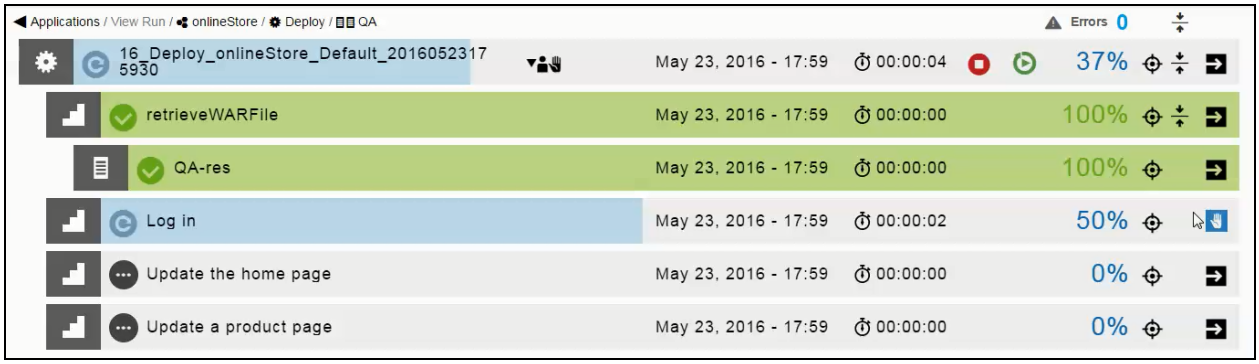


Deployment Scenarios

These are possible deployment scenarios.

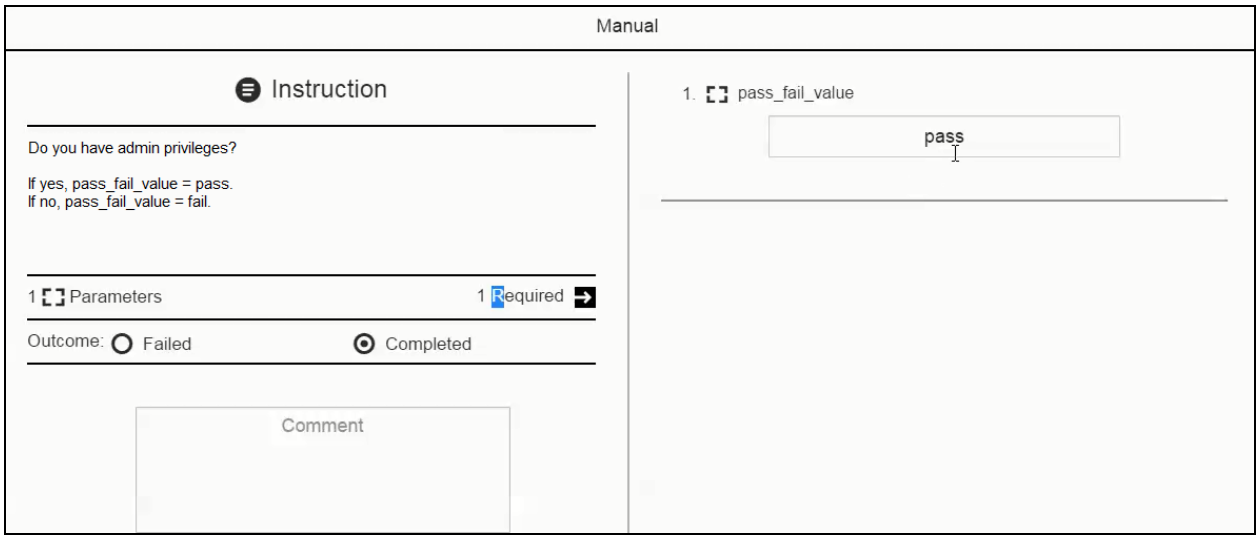
Assignee has admin privileges and the deployment is completed successfully

The home page is updated in this scenario. When the Deploy application process is deployed to the QA environment, you can view the progress of the deployment in the "Applications / View Run" page.



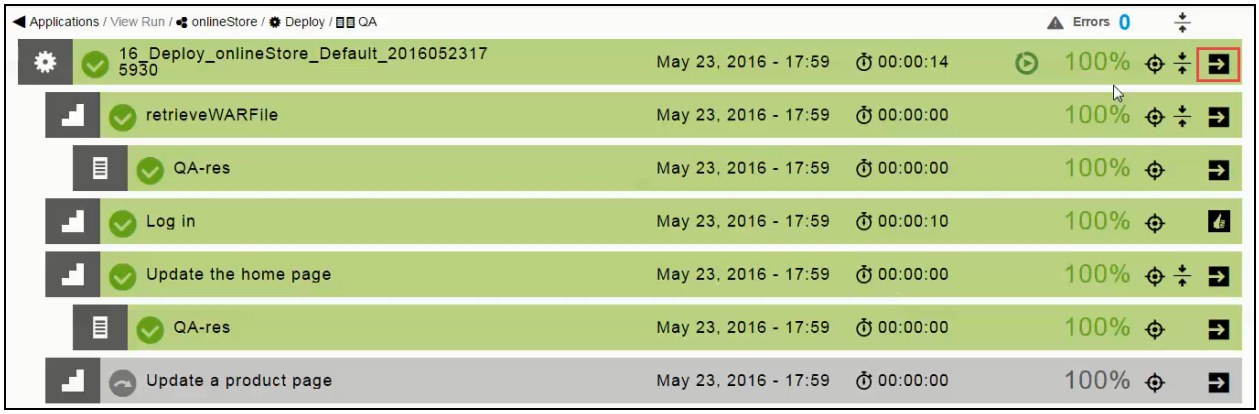
Clicking opens the Manual dialog box. Clicking in the Parameters row displays the required parameters on the right side of the dialog box.

If you enter *pass* in the **pass_fail_value** field, select **Completed** as the **Outcome**, and click **OK,** the process continues to the "Update the home page" step.

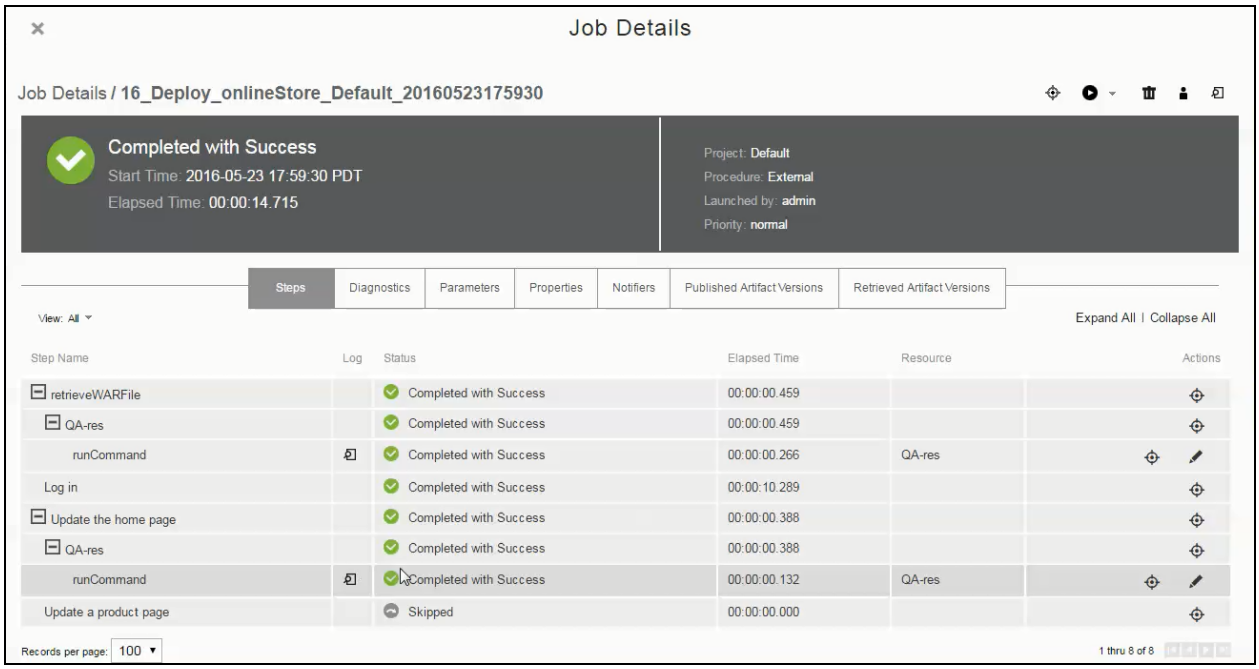


The parameter value (*pass*) is used in the precondition to determine the next step in the process flow. It is also passed to the next step.

The step to update the home page is successfully completed. Notice that the "Update a product page" step is skipped.



In the top row, clicking takes you to the Job Details page.



Clicking in the runCommand row takes you to the log file for that step, which displays the parameter value from the manual step that was passed to the "Update the home page" step.

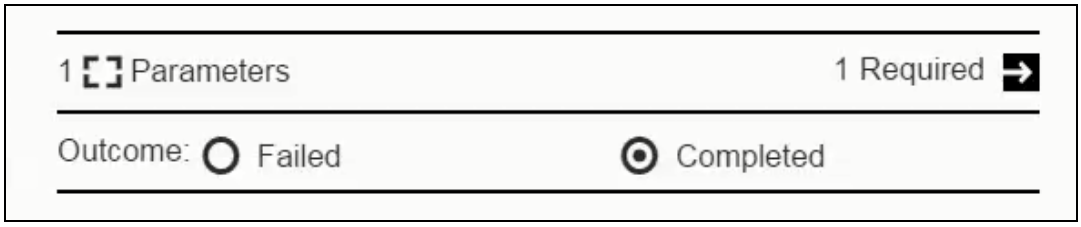
Assignee does not have admin privileges and the deployment is completed successfully

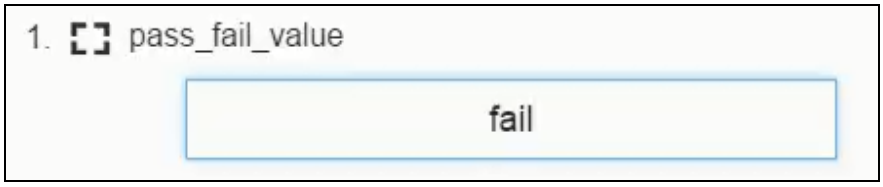
A product page is updated in this scenario. When the Deploy application process is deployed to the QA environment again, you can view the progress of the deployment in the "Applications / View Run" page.

When the process reaches the second step, clicking  opens the Manual dialog box.

Clicking  in the Parameters row displays the required parameters on the right side of the dialog box.

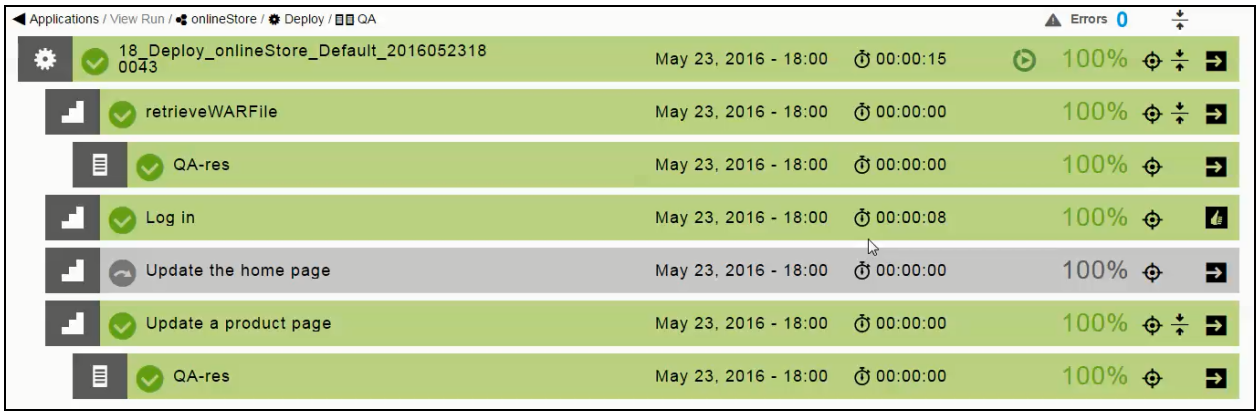
If you enter *fail* in the **pass_fail_value** field, select **Completed** as the **Outcome**, and click **OK**, the process continues to the "Update a product page" step.



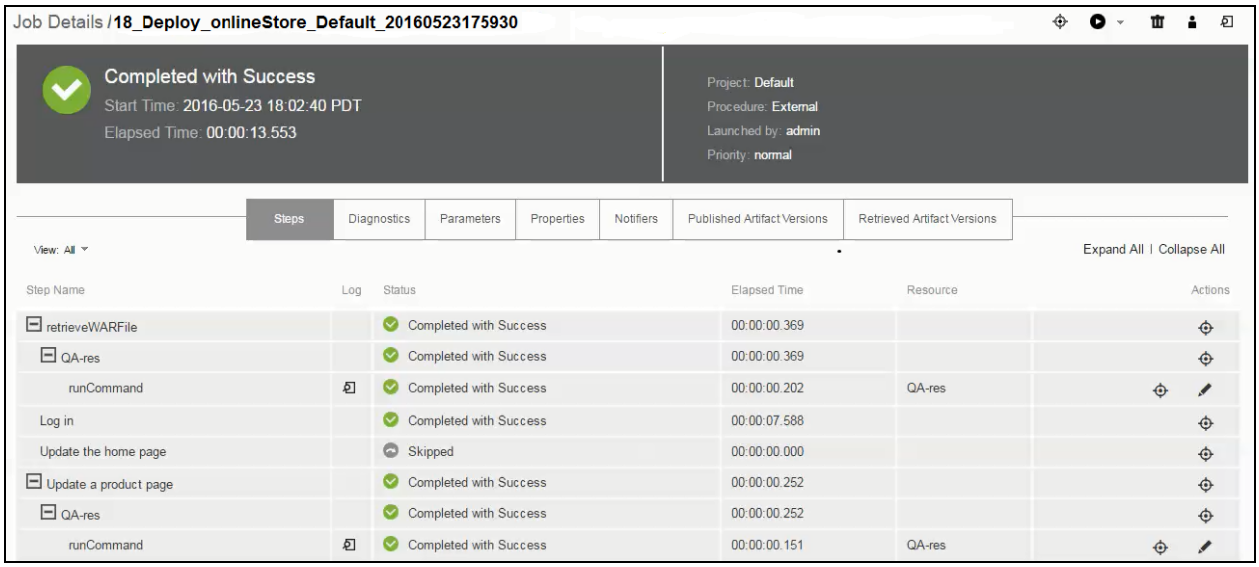


The parameter value (*fail*) is used in the precondition to determine the next step in the process flow. It is also passed to the next step.

In this deployment, both the Deploy process and the step to update product page are completed successfully.



In the top row, clicking  takes you to the Job Details page.



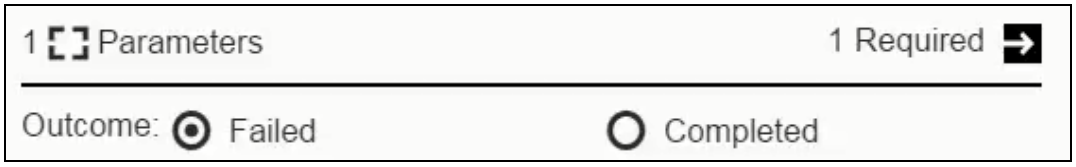

Clicking in the runCommand row takes you to the log file for that step, which displays the parameter value from the manual step that was passed to the "Update the home page" step.

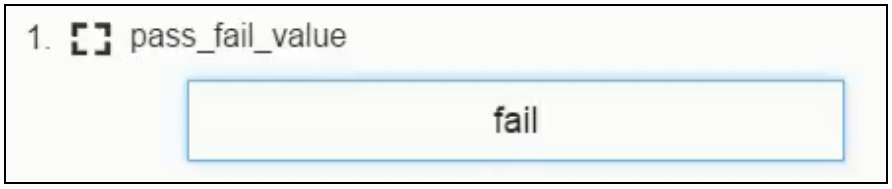
Assignee does not have admin privileges and the deployment is completed with errors

A product page is updated in this scenario. When the Deploy application process is deployed to the QA environment again, you can view the progress of the deployment in the "Applications / View Run" page.

When the process reaches the second step, clicking  opens the Manual dialog box.

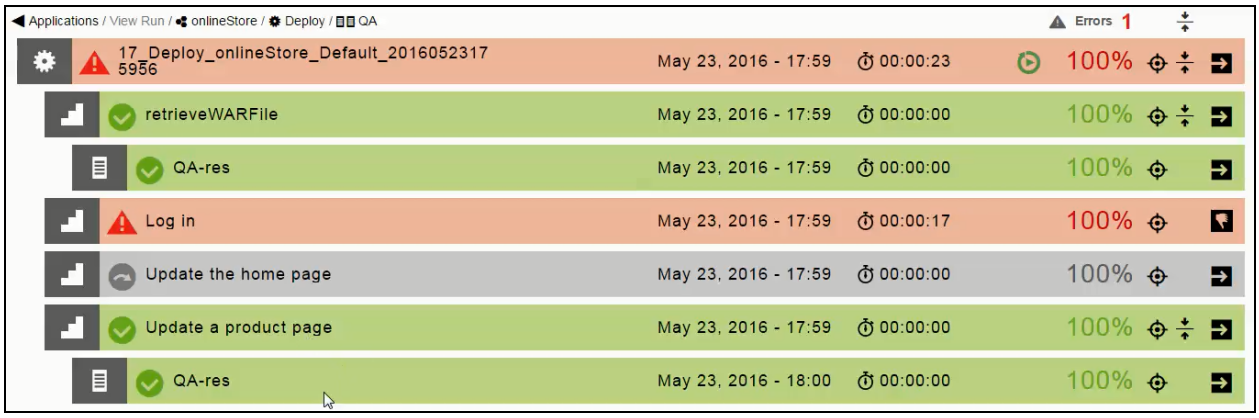
Clicking in the Parameters row displays the required parameters on the right side of the dialog box. If you enter *fail* in the **pass_fail_value** field, select **Failed** as the **Outcome**, and click **OK**, the process continues to the "Update a product page" step.





The parameter value (*fail*) is used in the precondition to determine the next step in the process flow. It is also passed to the next step.

In this deployment, the step to update the product page is completed successfully even though the Deploy application was completed with errors. The error is in the second step where the outcome is **Failed**.



In the top row, clicking  takes you to the Job Details page.

Clicking  in the runCommand row takes you to the log file for that step, which displays the parameter value from the manual step that was passed to the "Update the home page" step.

Setting Up Custom Parameters for Application and Component Processes

Parameters are used throughout the release planning and management process. They provide an easy way to model releases, pipelines, and applications. They can be inputs to procedures, workflows, applications, pipelines, and releases. You can use parameters in place of predefined credentials, property paths, user or group names for notifications, runtime conditions. Parameters are created and managed independent of any object at the platform level.

These examples show how to add custom parameters and setting the parameter labels in the GUI.

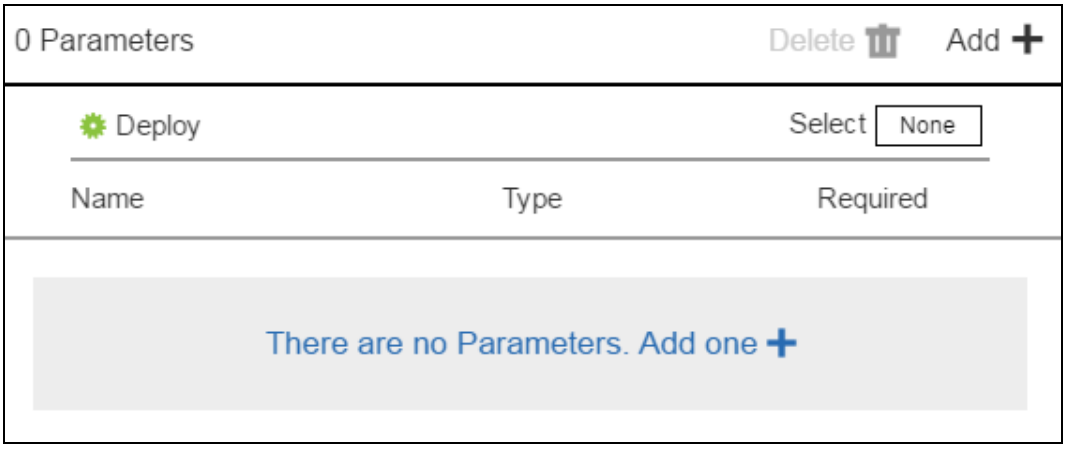
Adding Custom Parameters

In the Application Process or Component Process Visual Editor:

1. Click  on the upper right corner of the Visual Editor.
   - Click **Parameters**.

     The **Parameters** dialog box opens.

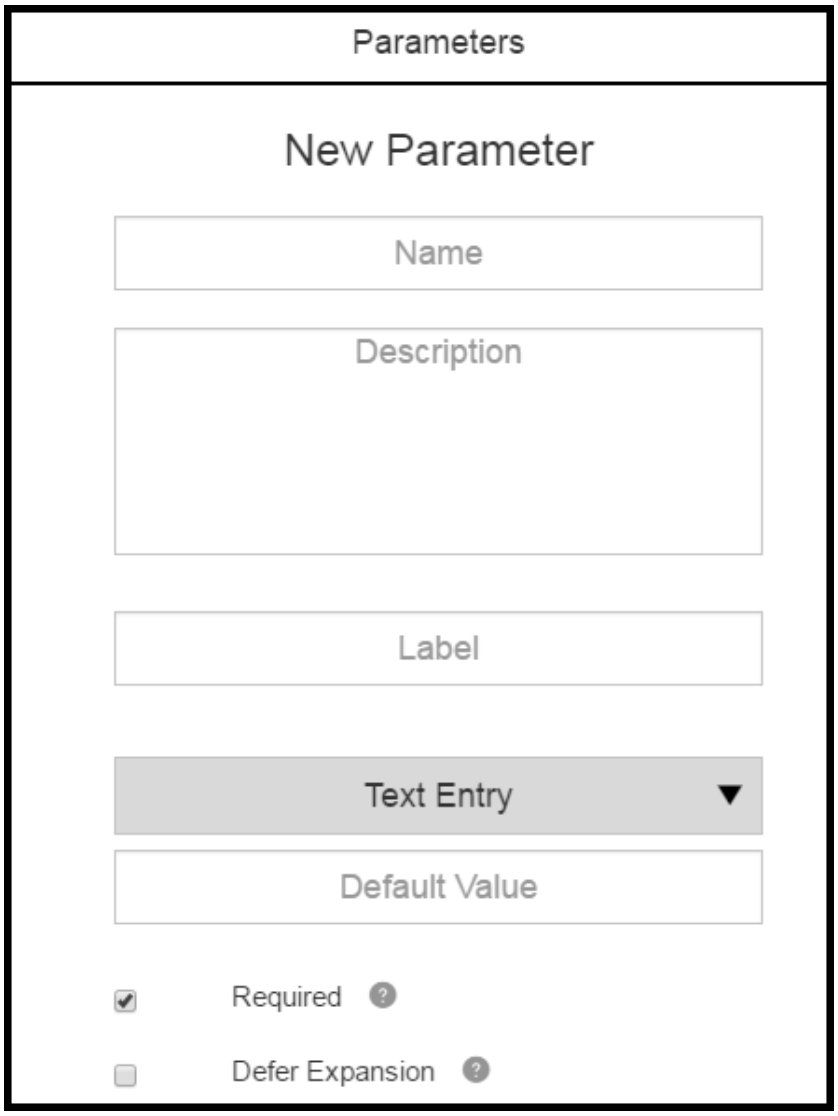
     Example:



2. Click **Add +** or if no parameters have not yet defined for the object, click **There are no Parameters yet. Add one +** to add a parameter.

   The **New Parameter** dialog box opens.

   Example:



3. Enter the following information:
   - **Parameter Name**—Name of the parameter.
   - **Description**—This is optional.

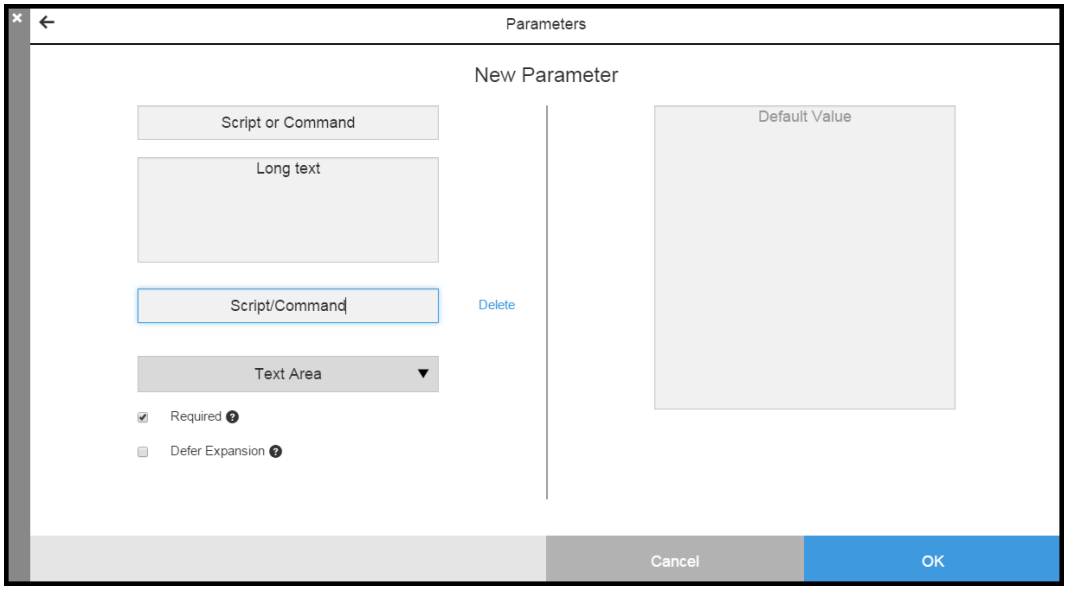
     > **Tip:** You can include hyperlinks as part of an object description for any ElectricFlow object.

   - **Label**—Label that appears on the GUI. This is optional.
   - **Text Entry**—How the user selects the value for this parameter. Valid options are **Text Entry**, **Test Area**, **Dropdown Menu**, **Radio Selector**, **Checkbox**, or **Credential**.
   - **Default Value**—Default value for the parameter. This is optional.
   - **Required**—If the parameter is required, select the **Required** check box.
   - **Defer Expansion**—Determines whether to expand the parameter value. A parameter value expansion occurs in one of two places:
     - When the procedure request is made (the default) (the check box is unchecked)
     - When the step executes (the check box is checked)

     The check box is unchecked by default (meaning that expansion is not deferred).
     - Depending on the parameter type that you select, other fields appear in the dialog box. Go to the following appropriate step to complete setting up the parameter.
4. If you select **Text Entry** as the parameter type, the **Default Value** field appears. You can enter a default value but is not required, and then click **OK**.

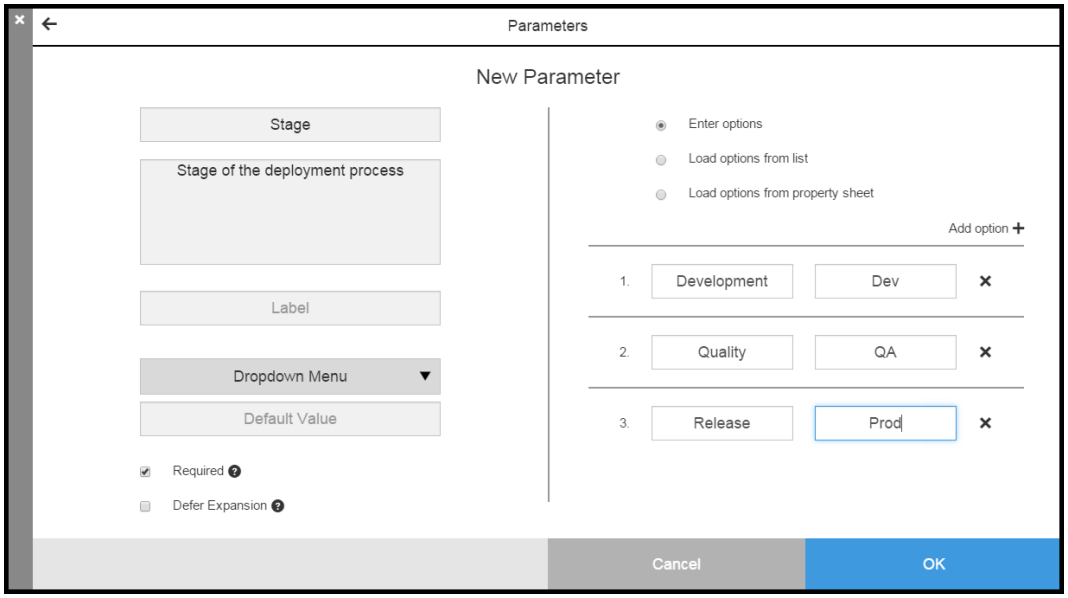
5. If you select **Text Area** as the parameter type, the **Default Value** field appears. You can enter a default value but is not required, and then click **OK**.

   Example:



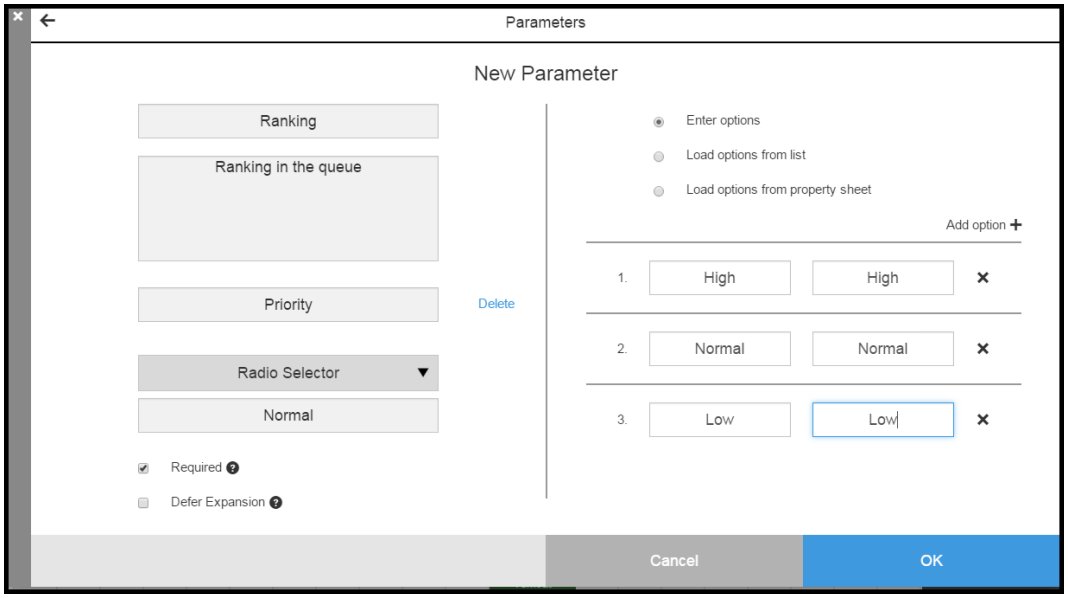
6. If you select **Dropdown Menu** as the parameter type, the **Default Value** field and ways to add the menu options appear. You can enter a default value but is not required.
   - On the right side of the dialog box, enter the menu options.
   - Click **OK**.

     Example:



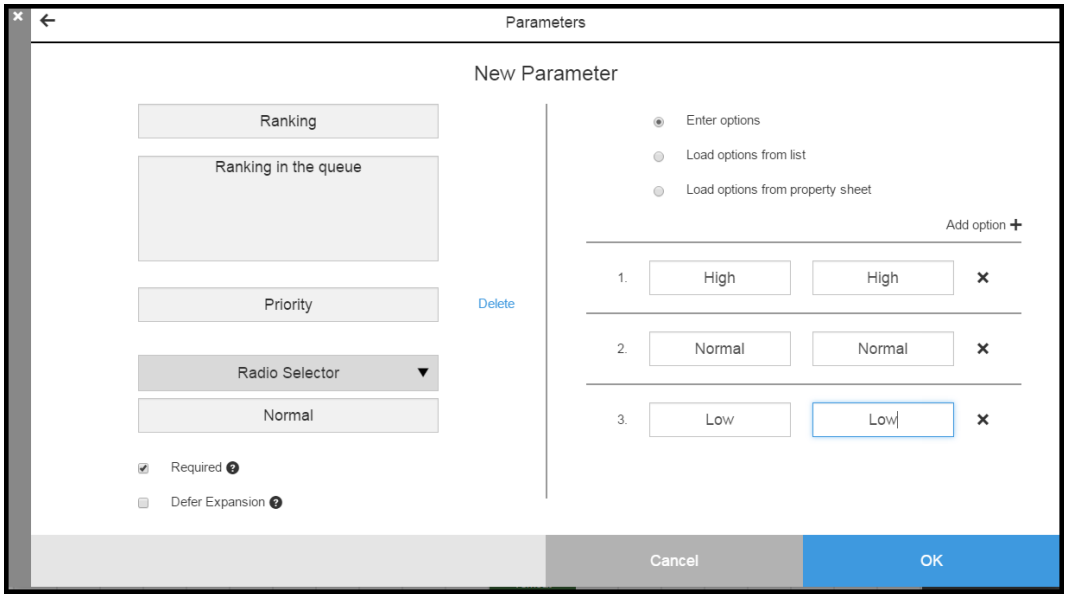
7. If you select **Radio Selector** as the parameter type, the **Default Value** field and ways to add the menu options appear. You can enter a default value but is not required.
   - On the right side of the dialog box, enter the menu options.
   - Click **OK**.

     Example:



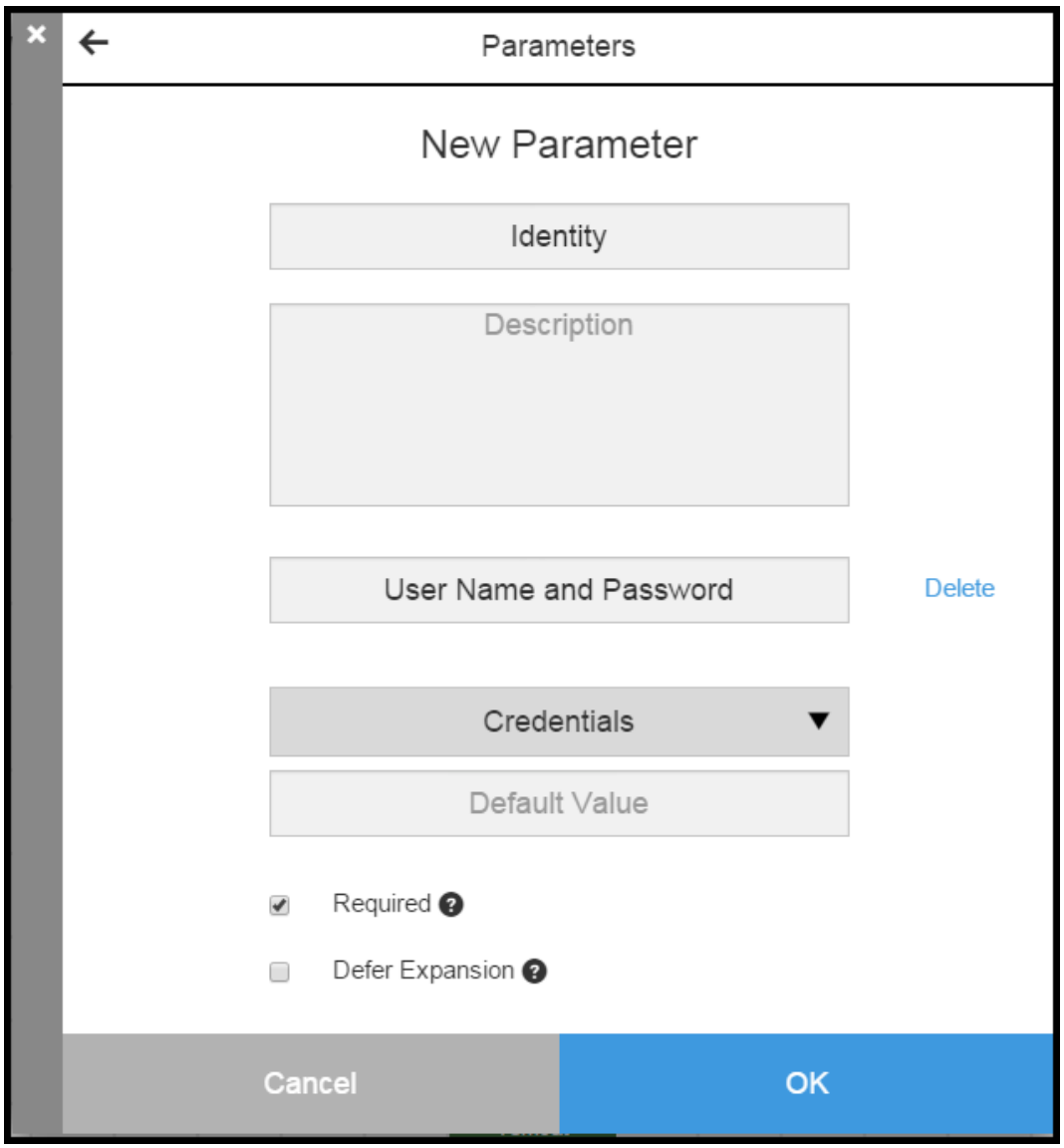
8. If you select **Checkbox** as the parameter type, the **Default Value** field and values for the check box appear. You can enter a default value but is not required.
   - On the right side of the dialog box, enter the values.
   - Click **OK**.

   Example:



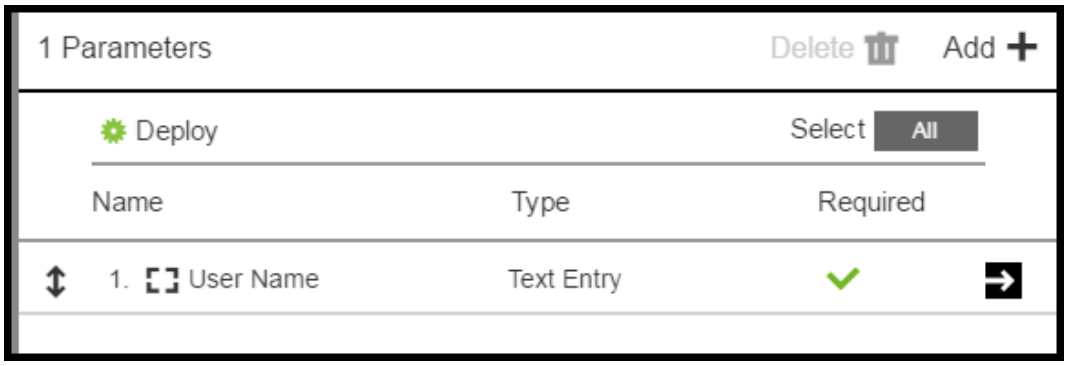
9. If you select **Credentials** as the parameter type, the **Default Value** field appears. You can enter a default value but is not required, and then click **OK**.

   Example:



After completing these steps, the **Parameters** dialog box opens and shows the new parameter in the list.

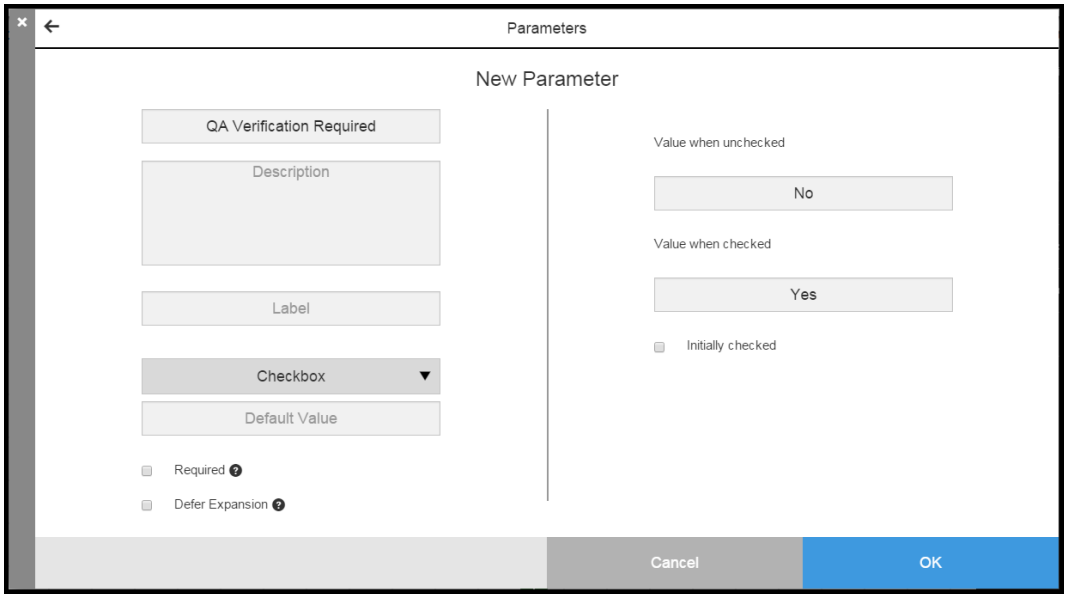
Example:



Setting and Modifying the Parameter Label

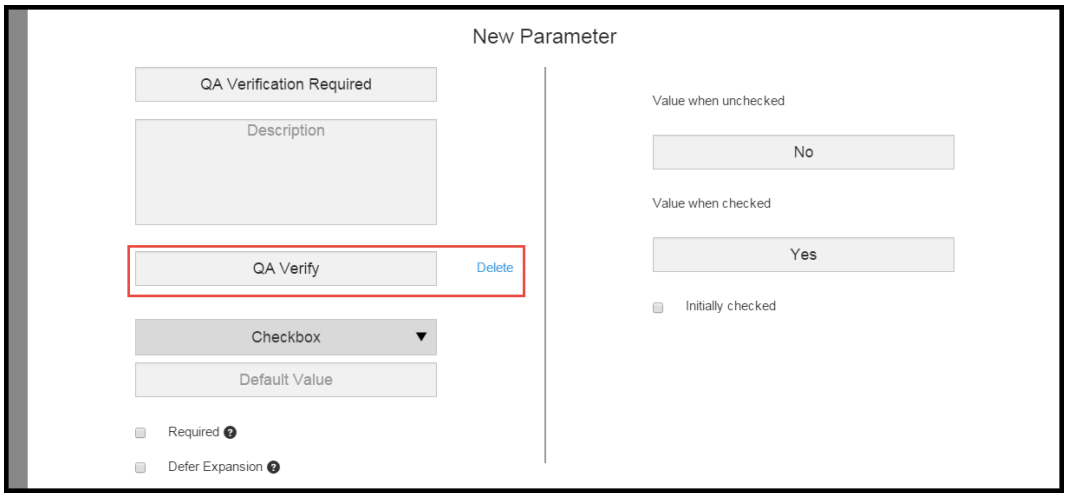
In the **New Parameter** dialog box:

Example:



1. Add a label in the **Label**. This is optional.

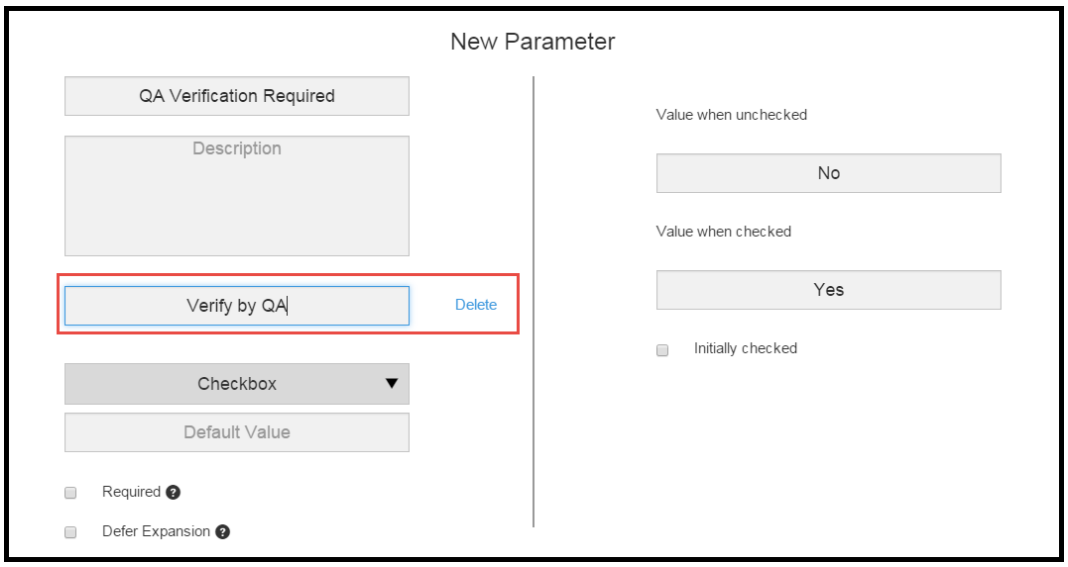
   Example:



If you enter a label, it appears in the UI form when you deploy the application. If you do not enter a label, the parameter name appears in the UI form when you deploy the application.

2. Click **OK**.
3. To modify the label:
   - Open the **New Parameter** dialog box.
   - Clear the **Label** field.
   - Enter a new label.

     Example:

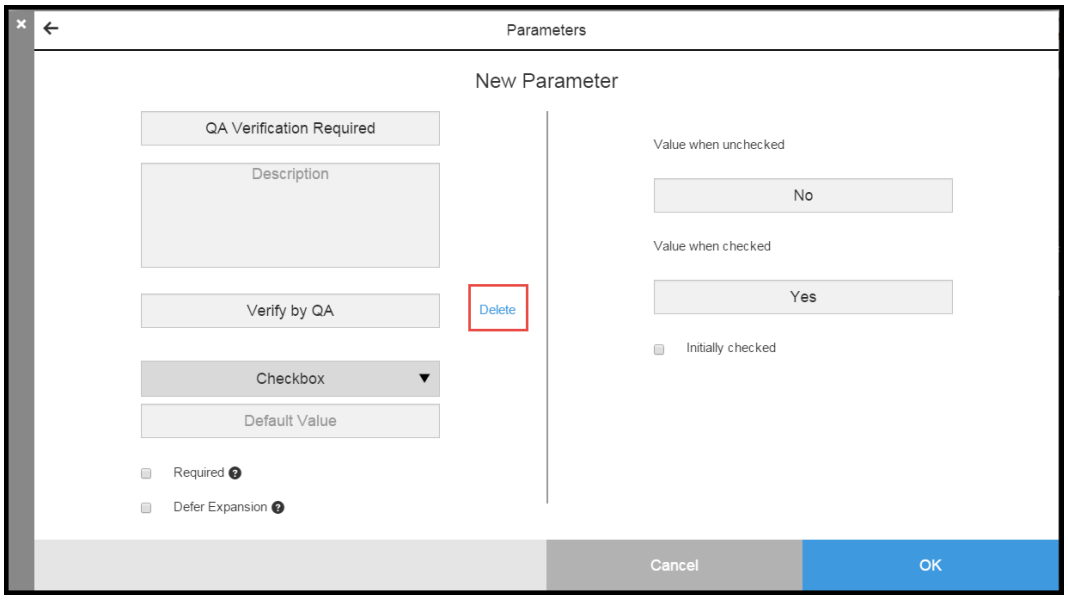


   - Click **OK** to save the change.

4. To delete the label:

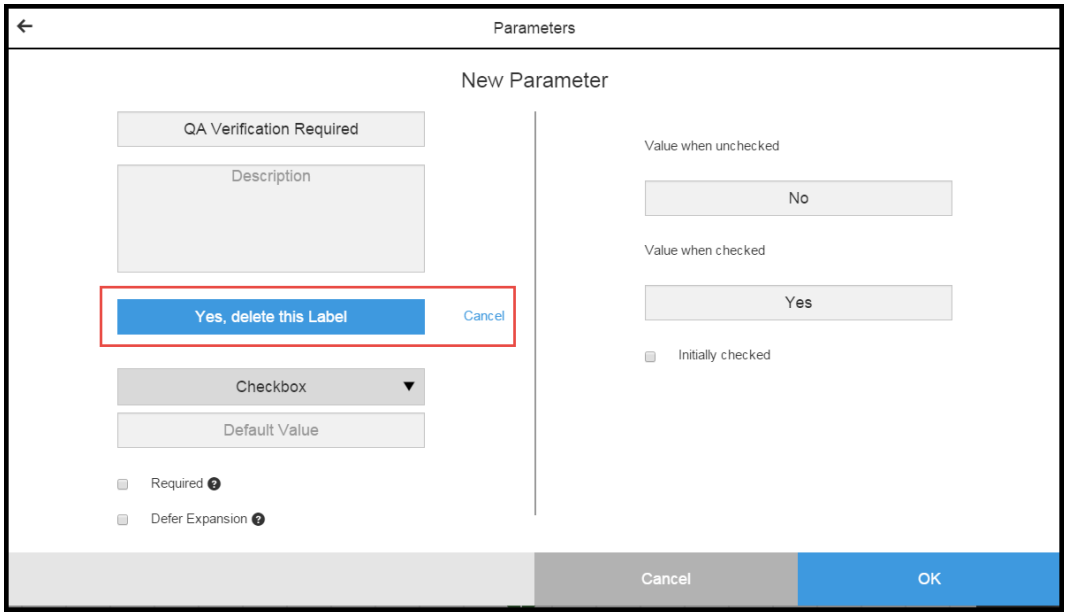
   - Click **Delete** next to the **Label** field.

     Example:



     Notice that the text in the **Label** field changes.

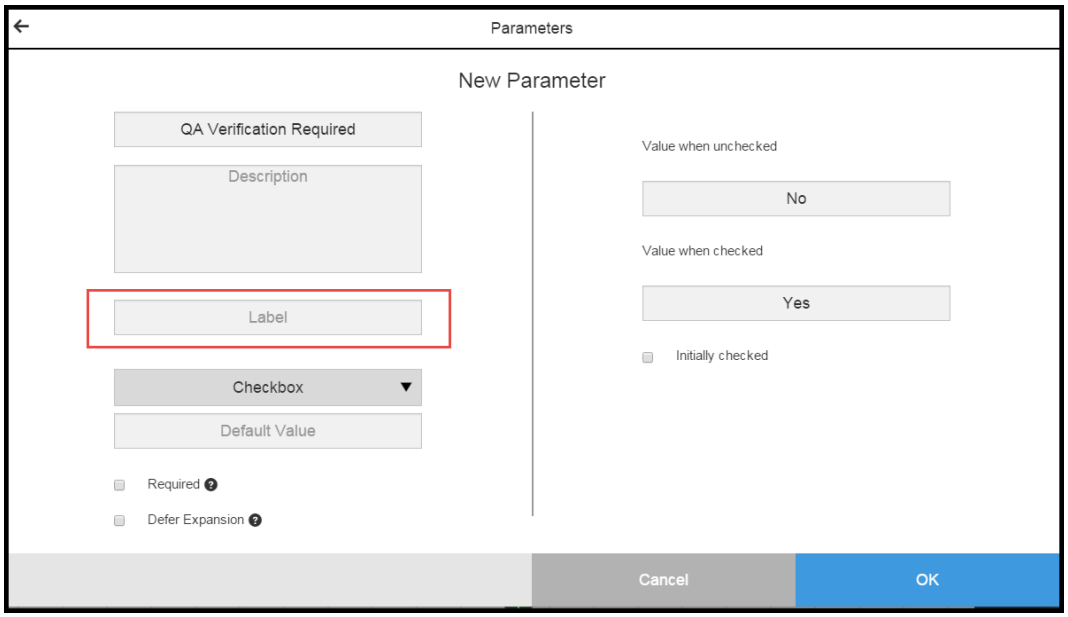
     Example:



- Click in the **Label** field.

  The label disappears.

  Example:



- Click **OK**.

  The **Parameters** dialog box opens. The parameter name now appears in the name column.

Looking Up Parameters in Application and Component Processes

To apply parameters to an application or component process step, starting in the Application Process or a Component Process Visual Editor:

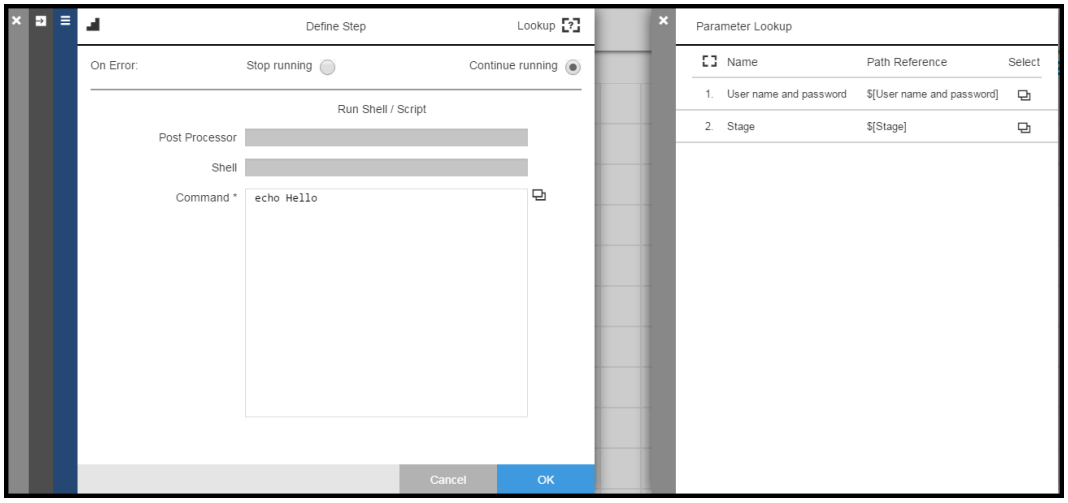
1. Add a new step to the process.
2. Define the process step in the **Define Step** dialog box.

3. When you define the process step with a plugin (**Plugins**), command or script (**Command**), or project (**Procedure**), click the **Lookup** button to open the **Parameter Lookup** dialog box.

   Example:

   The **Parameter Lookup** dialog box opens.



4. Choose a parameter and click the **Copy** button to copy the path reference.

   A message appears in the row : <*Parameter Name*> has been copied.

   The **Define Step** dialog box now has a Parameter field.

5. Click in the **Parameter** field and paste the path reference that you copied in it.
6. Repeat the previous two steps to apply another parameter to the process step.
7. Click **OK**.

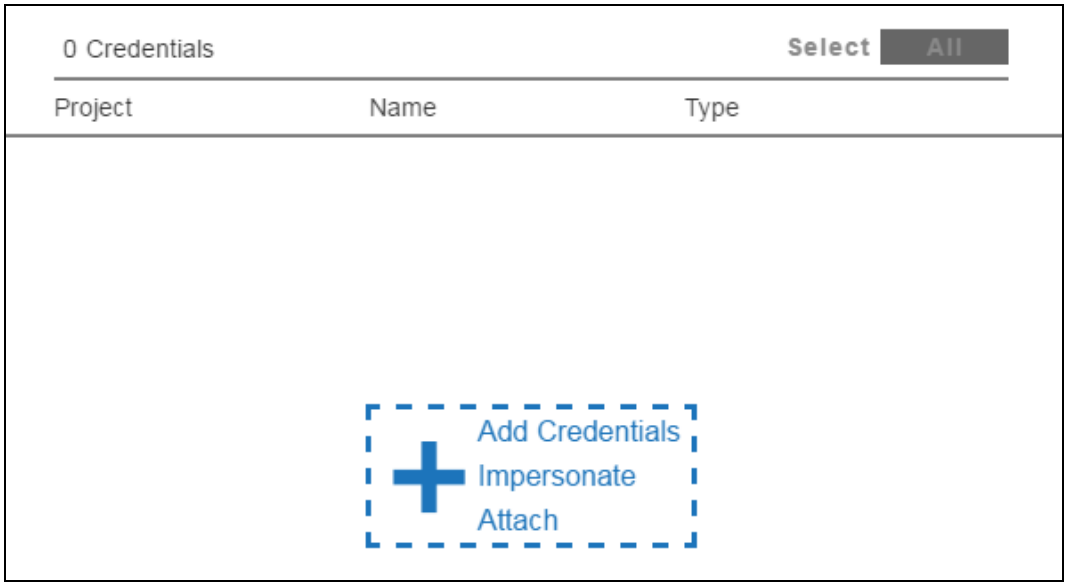
Attaching Credentials to Application and Component Processes

This topic describes how to add credentials and parameters to a component processes and apply custom parameters to them. These parameters are required at runtime when the application is deployed.

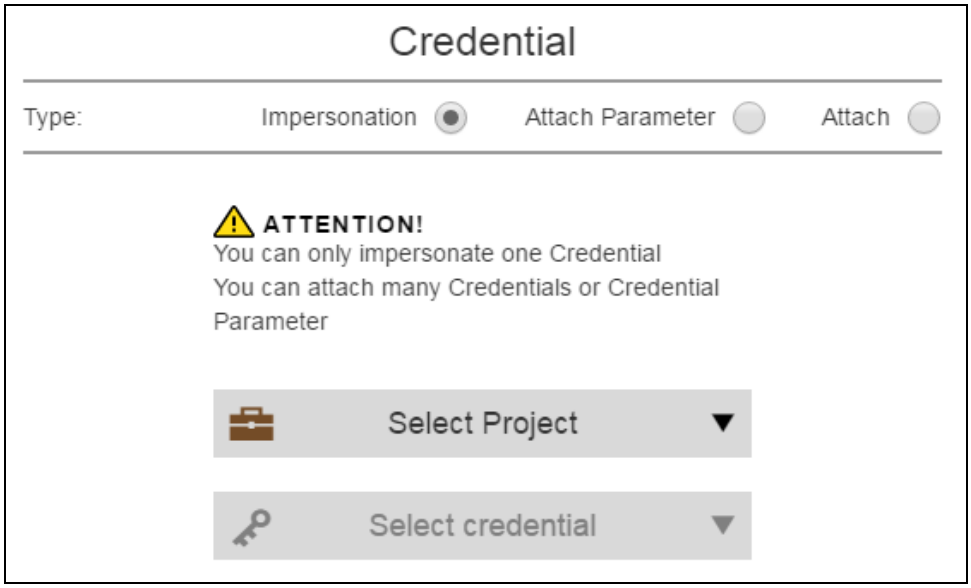
When defining an application or component process step:

1. Click  in the Credentials row in the dialog box where you can set the runtime settings.

The Credentials dialog box opens and shows the list of credentials for this process step. If there are no credentials, the dialog box looks like this:



2. Click **+ Add Credentials Impersonate Attach** to add a credential.

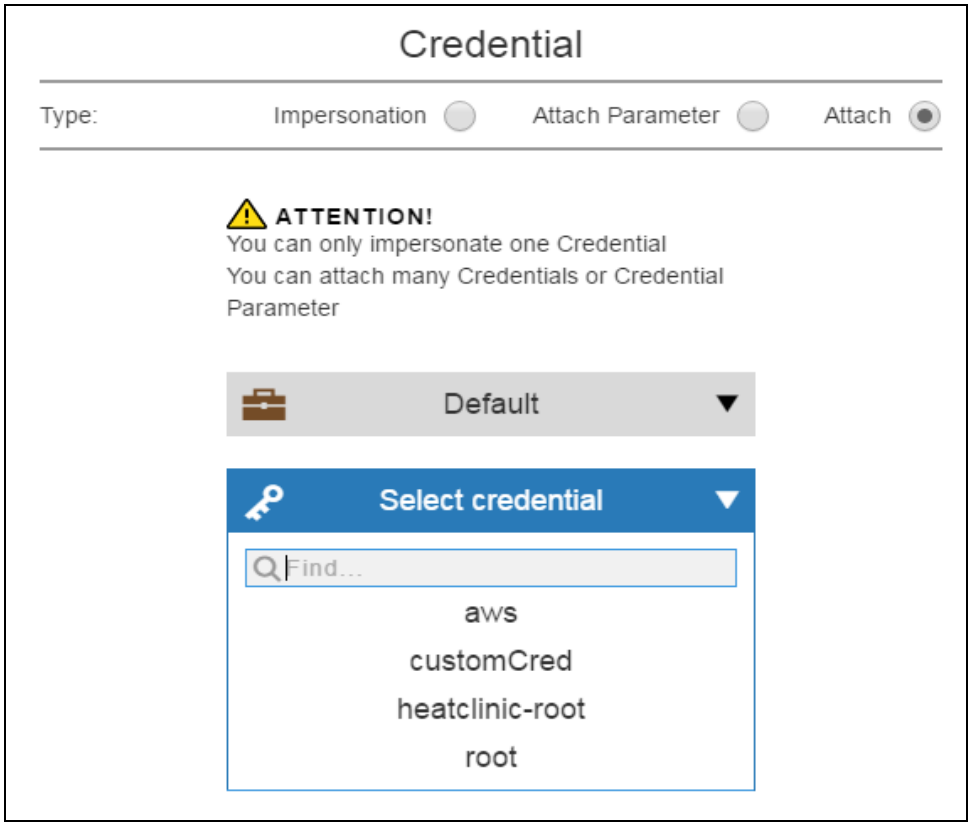


3. Select **Attach** to attach a credential to the process step. This credential is created and managed at the platform level.

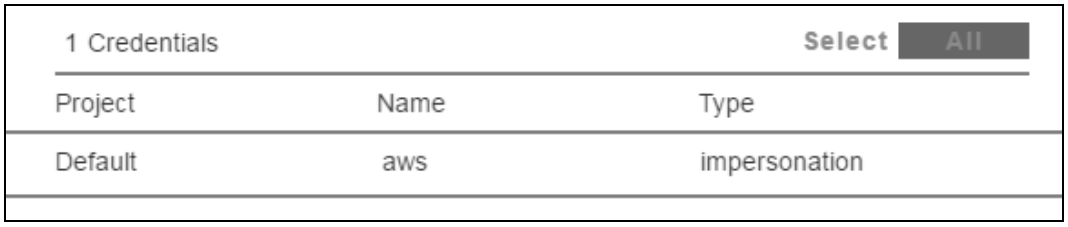
   The other ways to add credentials to a process step are:

   - If you want to impersonate one credential, select **Impersonation**, select the project to which the credential belongs, and the credential that you want to use for impersonation.
   - If you want to attach a credential parameter to the process step, select **Attach Parameter**. Click in the **Select credential parameter** field to set the parameter.

4. Select the project to which the credential belongs in the **Select Project** field, the credential in the **Select Credential** field, and click **Next**.

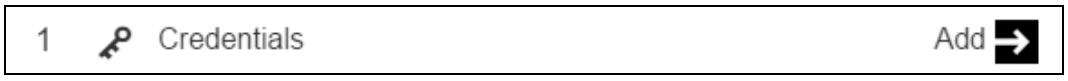


The list of credentials now shows the credential you added:



5. Click **Next**.

The dialog box where you can set the runtime settings now shows that one credential is attached to the process step.



6. Repeat the previous steps to attach another credential or use impersonation.

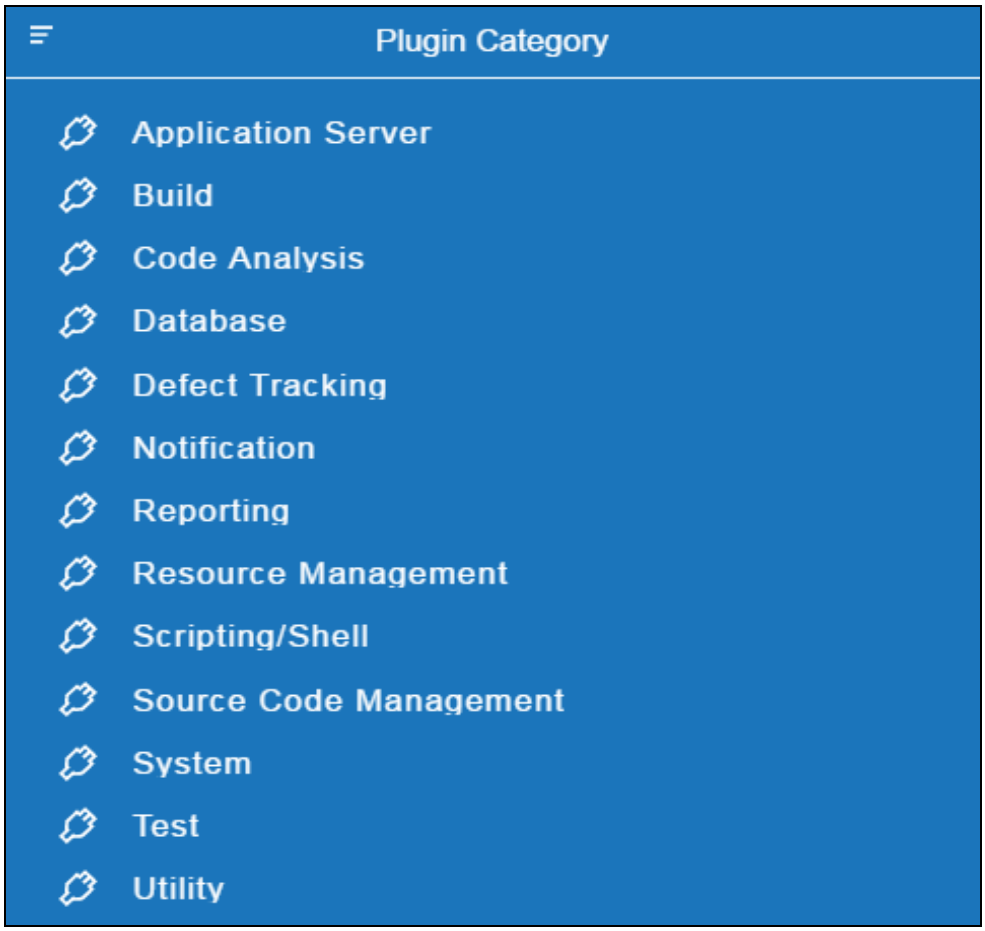
> **Important:** When you impersonate using a credential, make sure that the impersonated user has the absolute path to the `bin` directories in the `$PATH` environment. If you define a process step with a command, you must enter the absolute path in the **Post Processor** and **Shell** fields in the **Define Step** dialog box.

Plugin Process Steps

Plugin steps allow you to orchestrate third-party tools at the appropriate time in your component or application process. The following third-party tools can be used with ElectricFlow:

- Application Servers—Examples are IIS, JBoss, and Tomcat.
- Builds—Examples are eMake, Maven, or Visual Studio.
- Code Analysis—Examples are Clover CMD, Coverity, or Klocwork..
- Databases—Examples are DBI, Oracle, or SQLServer.
- Defect Tracking—Examples are Bugzilla, JIRA, or Rally.
- Notification—An example is Twitter.
- Reporting—An example is Reports.
- Resource Management—Examples are Chef, EC2, or OpenStack.
- Scripting or Shell operations—Examples are Groovy, Python, or Ruby.
- Source Code Analysis—Examples are ECSCM-ClearCase, ECSCM-Git, or ECSCM-Perforce.
- System —Examples are Artifact or FileSysRepo.
- Test —Examples are HPQualityCenter, Jasmine, or Selenium.
- Utility—Examples are FileOps or SendEmail.

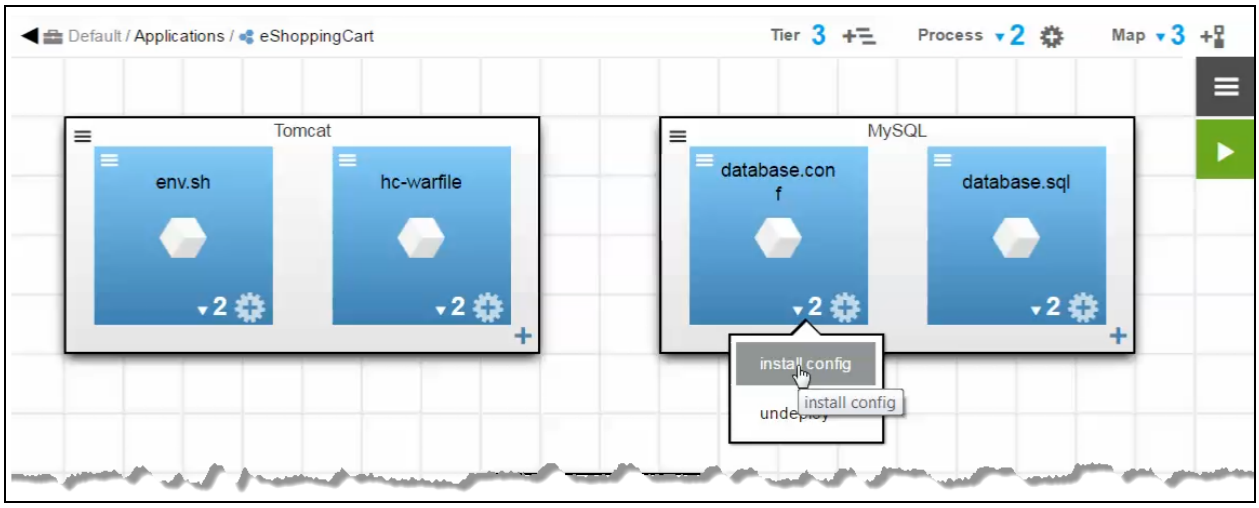
You can access these options from the Plugin Category dialog box:



Plugin Steps in a Component Process

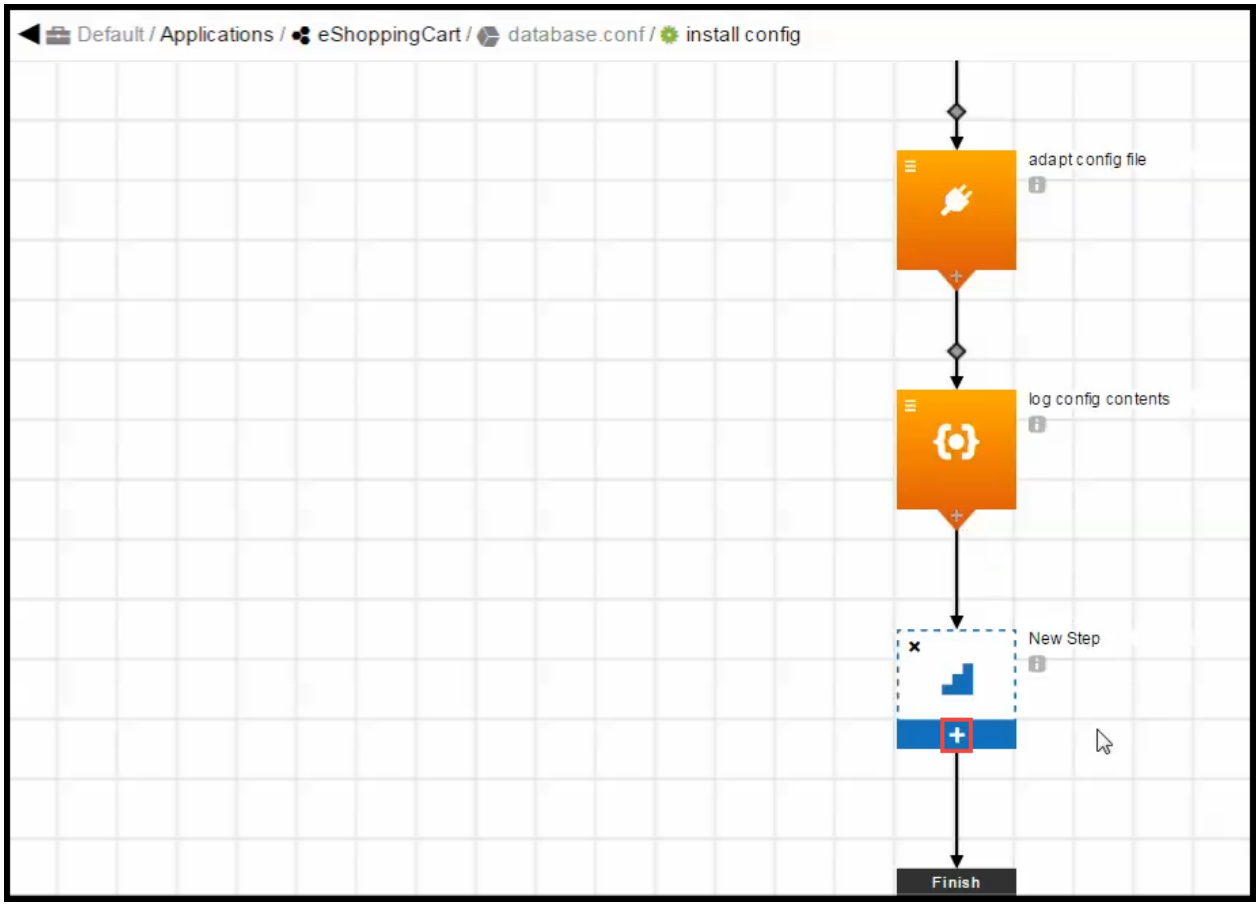
This example shows how to define a test plugin step in a component process.

1. In the Application Editor for the eShoppingCart application, select the "install config" process for the database.conf component in the MySQL application tier.



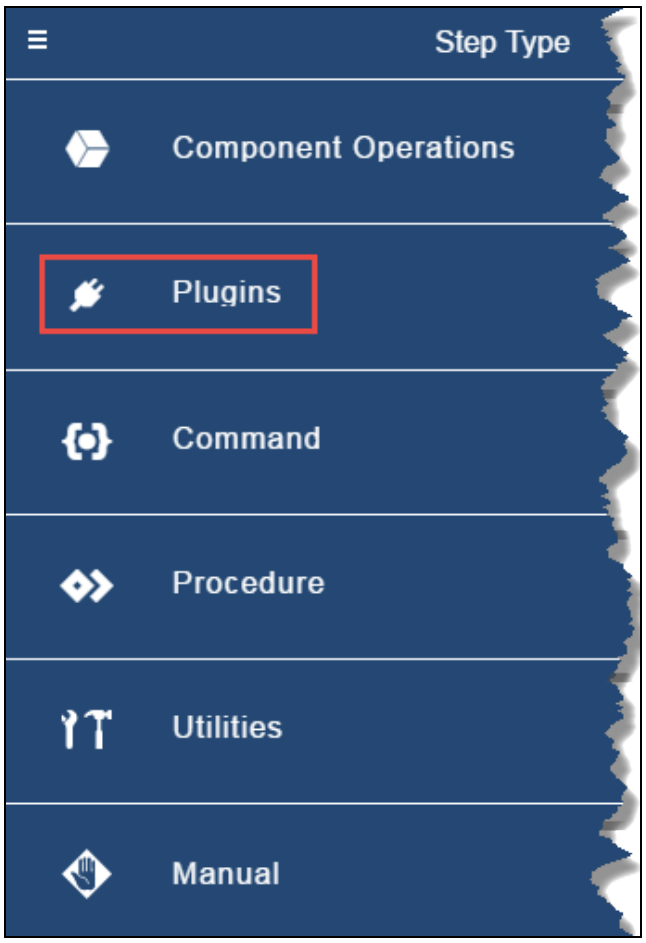
2. In the Component Process Visual Editor, add a step after the "log config contents" step.

   The "New Step" appears after this step.

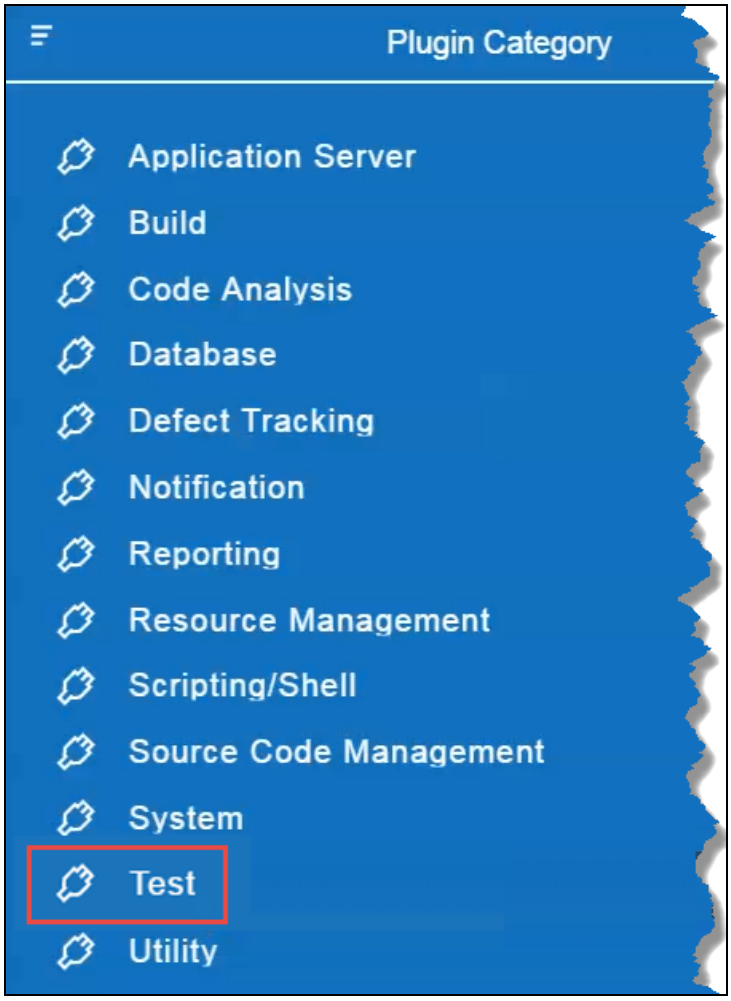


3. Click **+** in the "New Step" to define it.

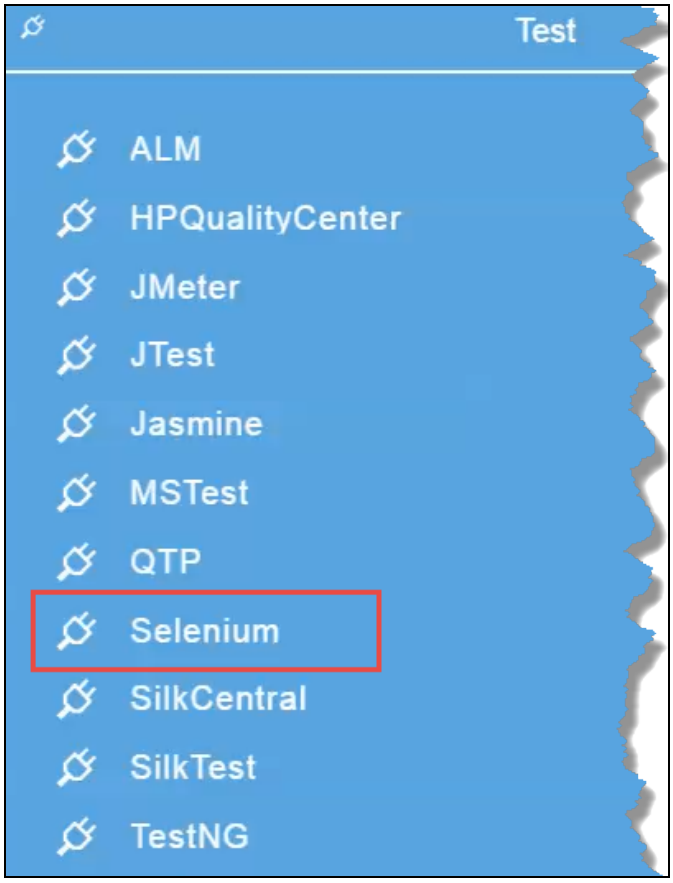
4. Name the step "Verify configuration", use the default values, and click **Next**.
5. In the Step Type dialog box, select **Plugins**.



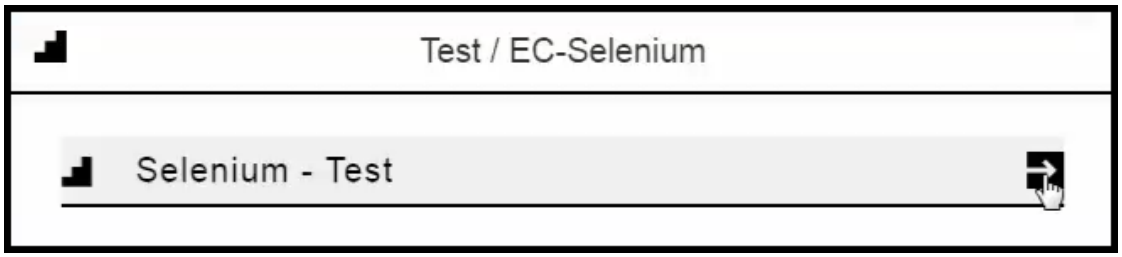
6. In the Plugins Category dialog box, select **Test**.



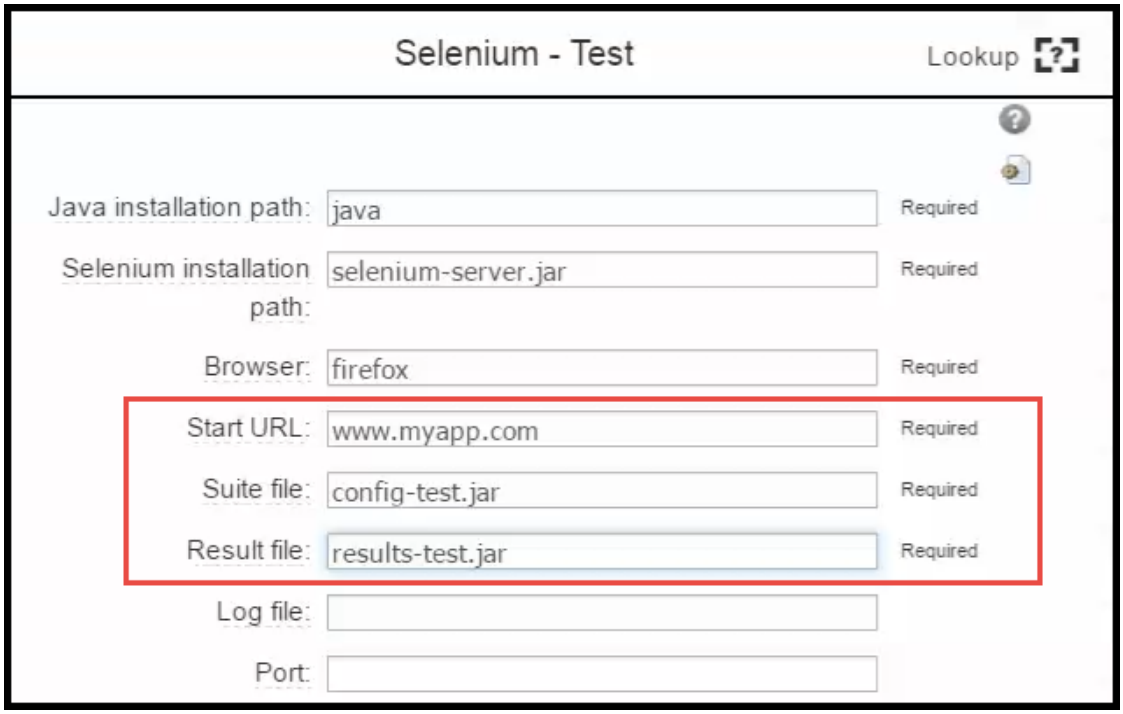
7. In the Test dialog box, select **Selenium**.



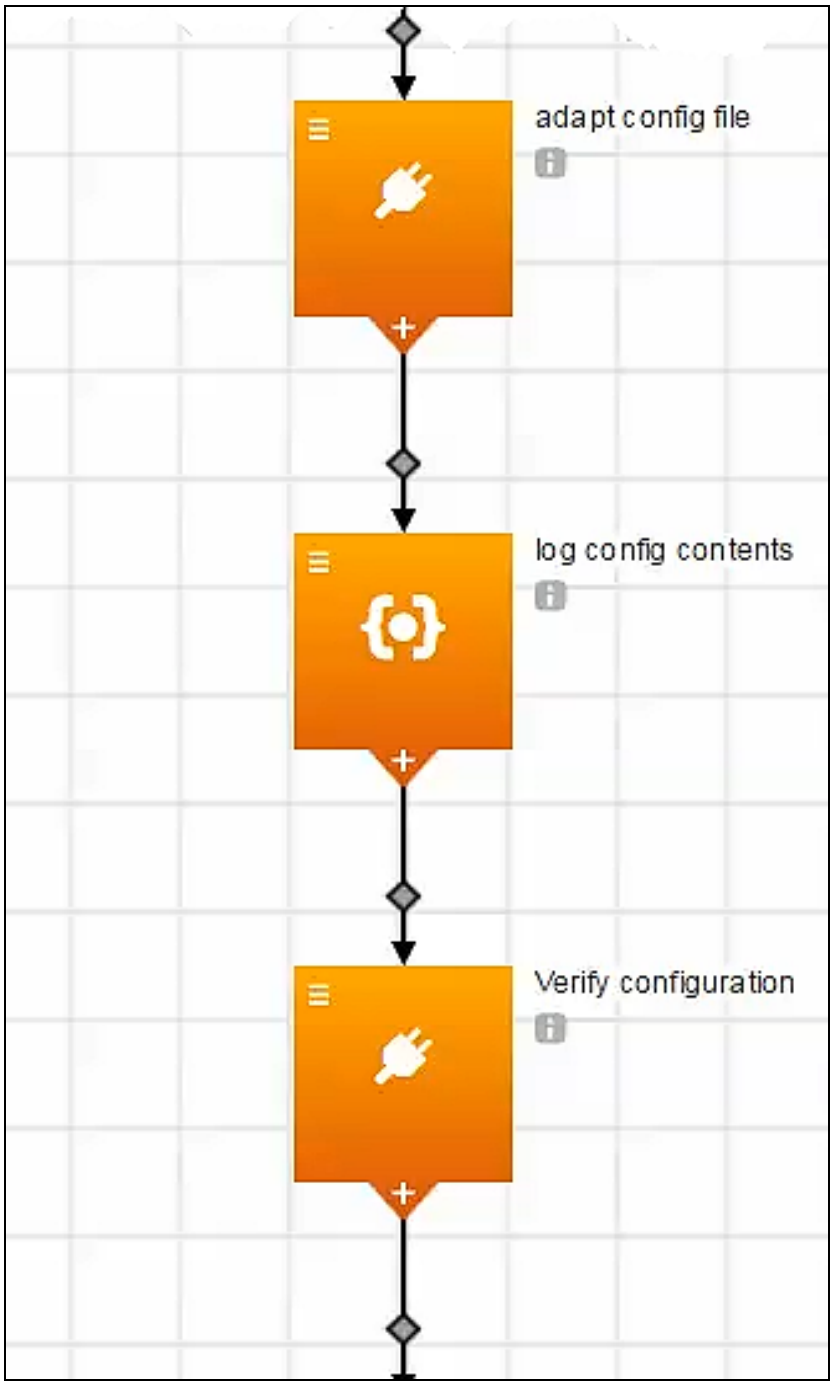
8. Select the **Selenium—Test** procedure.



9. Enter the required parameters for the procedure, and click **OK**.



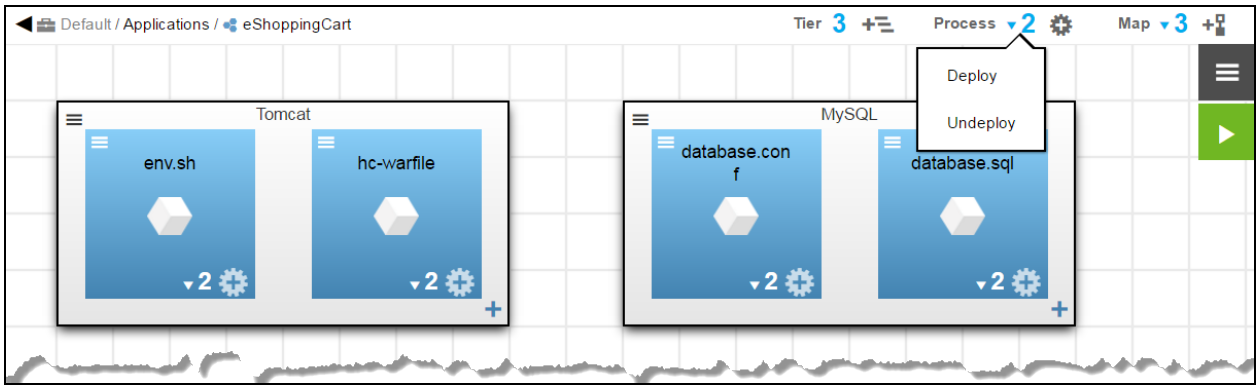
The Component Process Visual Editor now shows this step:



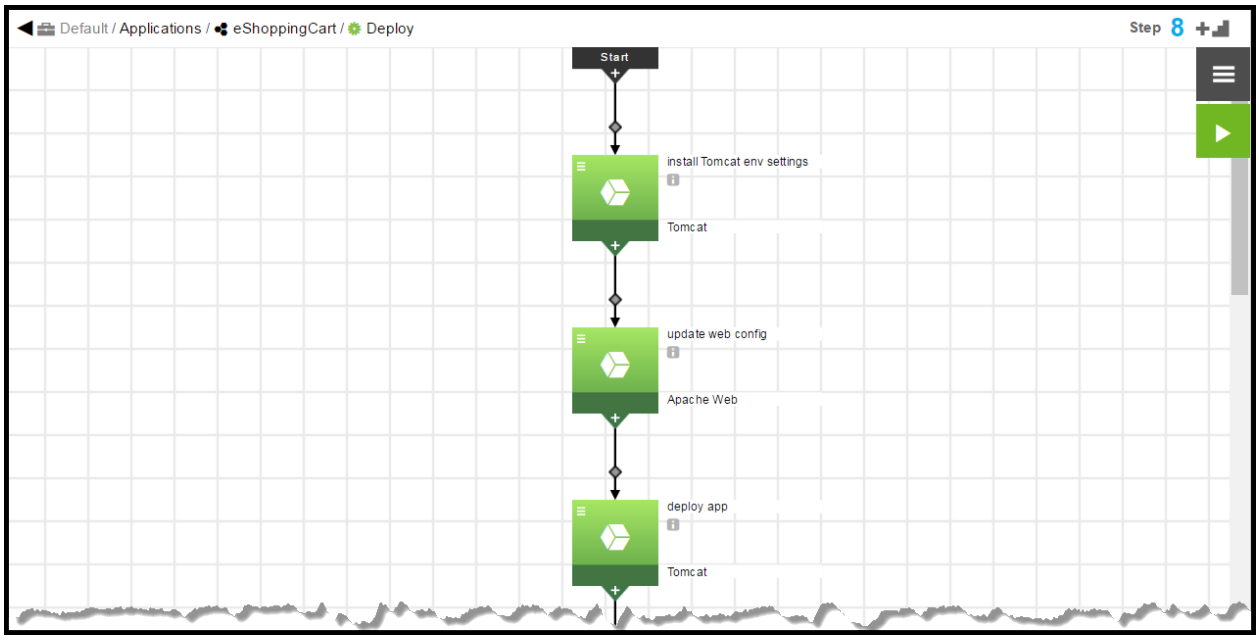
Plugin Steps in an Application Process

This example shows how to add a test plugin step in an application process.

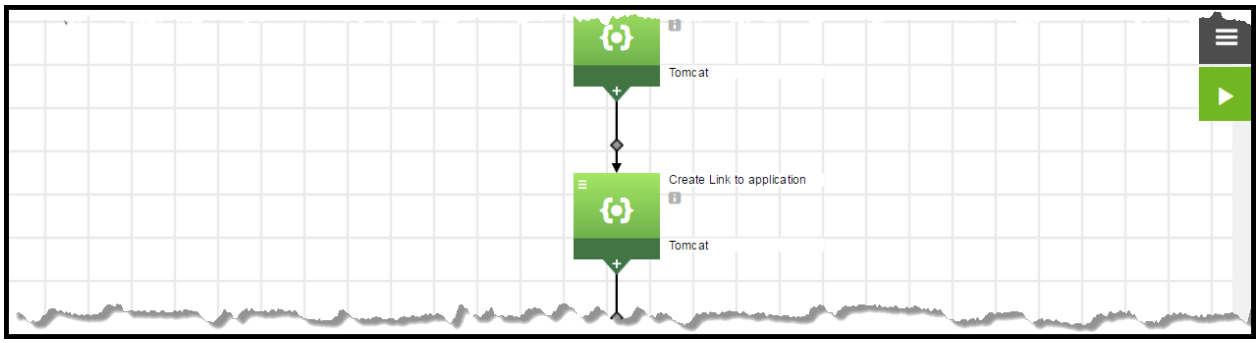
1. In the Application Editor for the eShoppingCart application, select the "Deploy" application process.



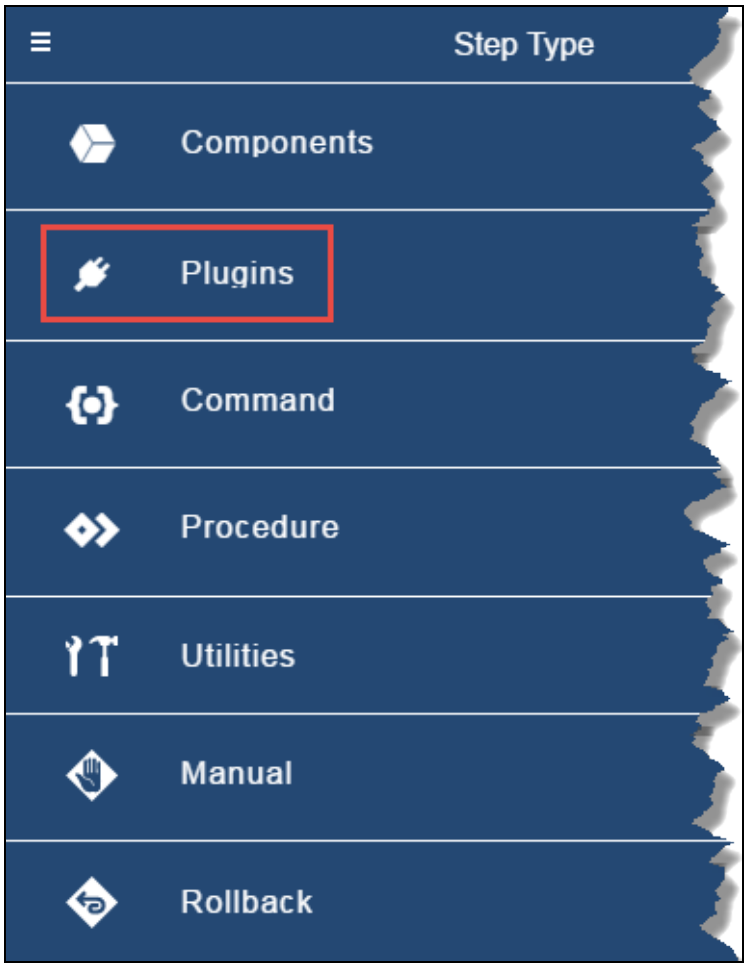
The Application Process Visual Editor opens.



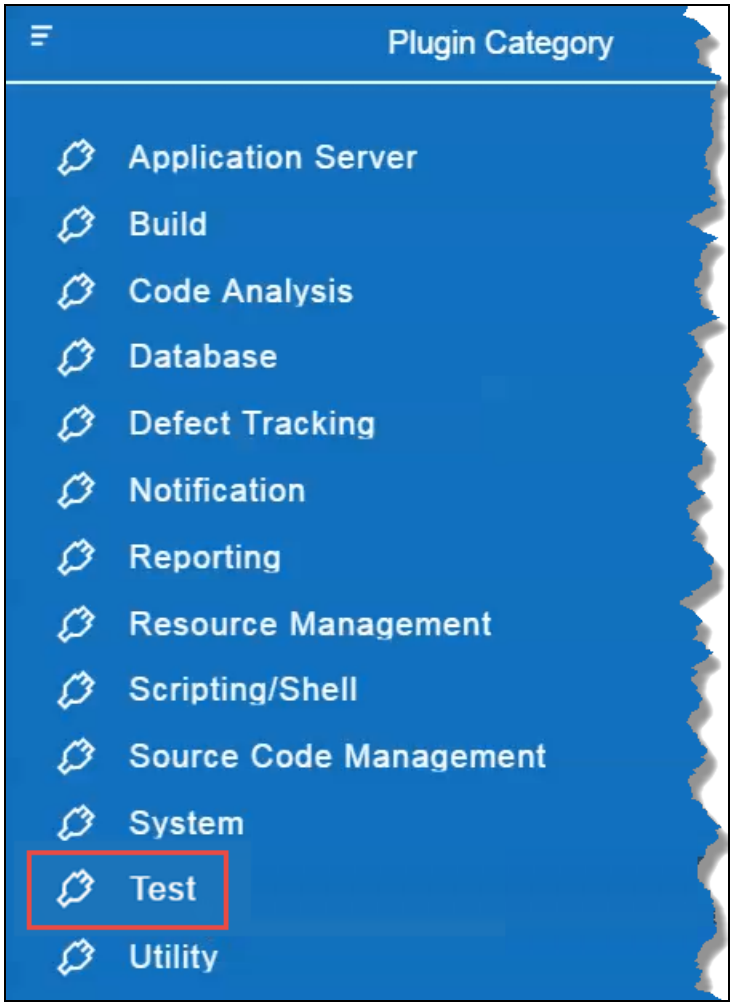
2. Add a step after the "Create Link to application" step.



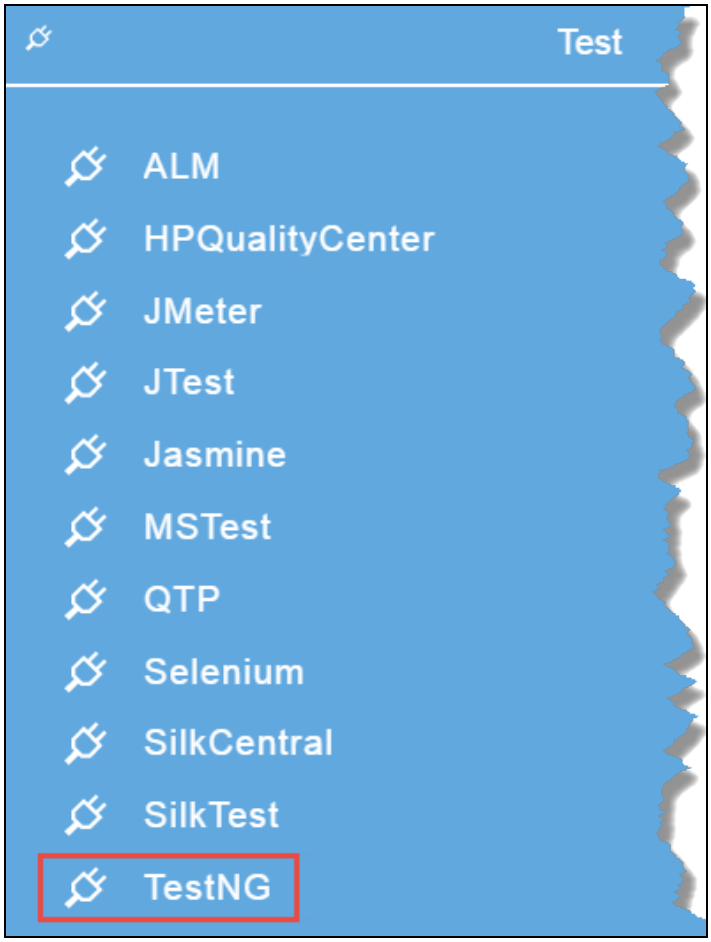
3. Click **+** in the "New Step" to define it.
4. Name the step "Verify setup", use the default values, and click **Next**.
5. In the Step Type dialog box, select **Plugins**.



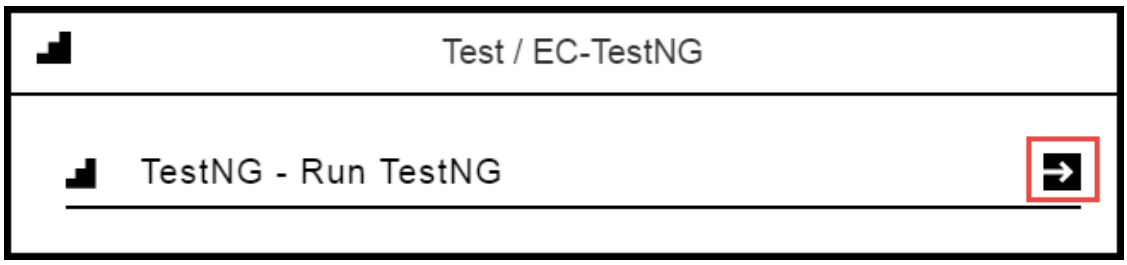
6. In the Plugins Category dialog box, select **Test**.



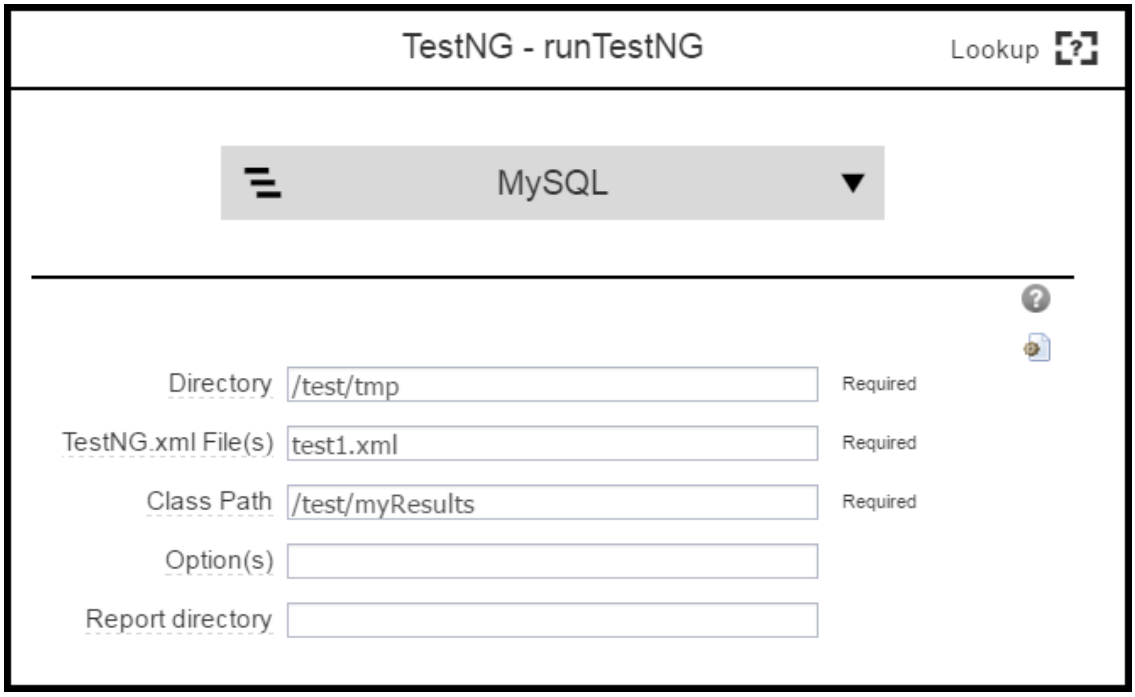
7. In the Test dialog box, select **TestNG**.



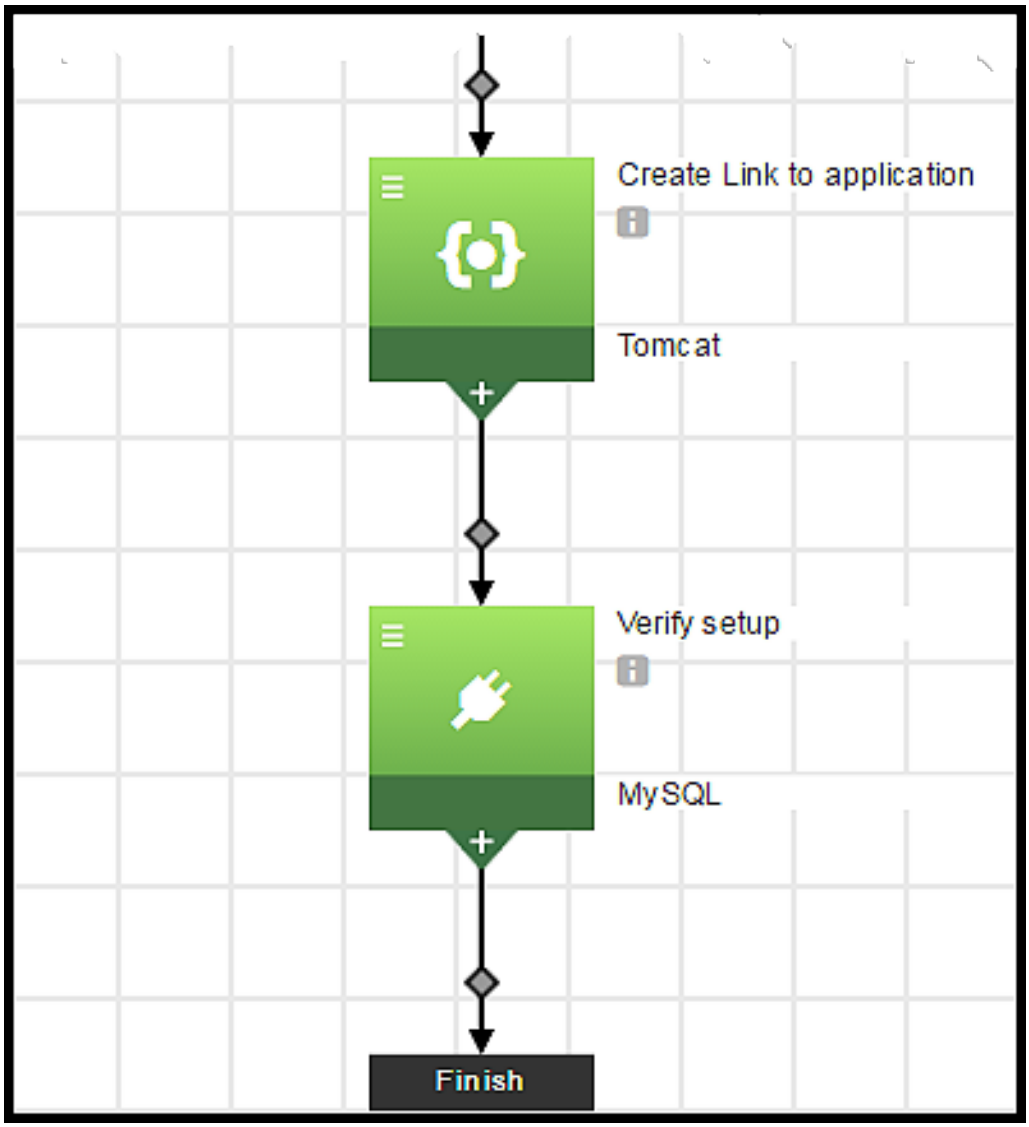
8. Select the **TestNG—Run TestNG** procedure.



9. Enter the required parameters for the procedure, and click **OK**.



The Application Process Visual Editor now shows this step:



Adding Process Steps

How to get to the Application Process Visual Editor:

- In an existing application process:

  From the Application Editor, click the down arrow next to the number-of-application-processes button and select an application.

  The Application Process Visual Editor for that application process appears.

- In a new application process:

  From the Application Editor, click the **Add process** button, set the details in the **Application Process Details** dialog box, and click **OK**.

  The Application Process Visual Editor for the application appears.

How to get to the Component Process Visual Editor:

- In an existing component process:

  From the Application Editor, click the down arrow next to the number-of-component-processes button in a component, and select a component process in the drop-down list.

  The Component Process Visual Editor for that component process appears.

- In a new component process:

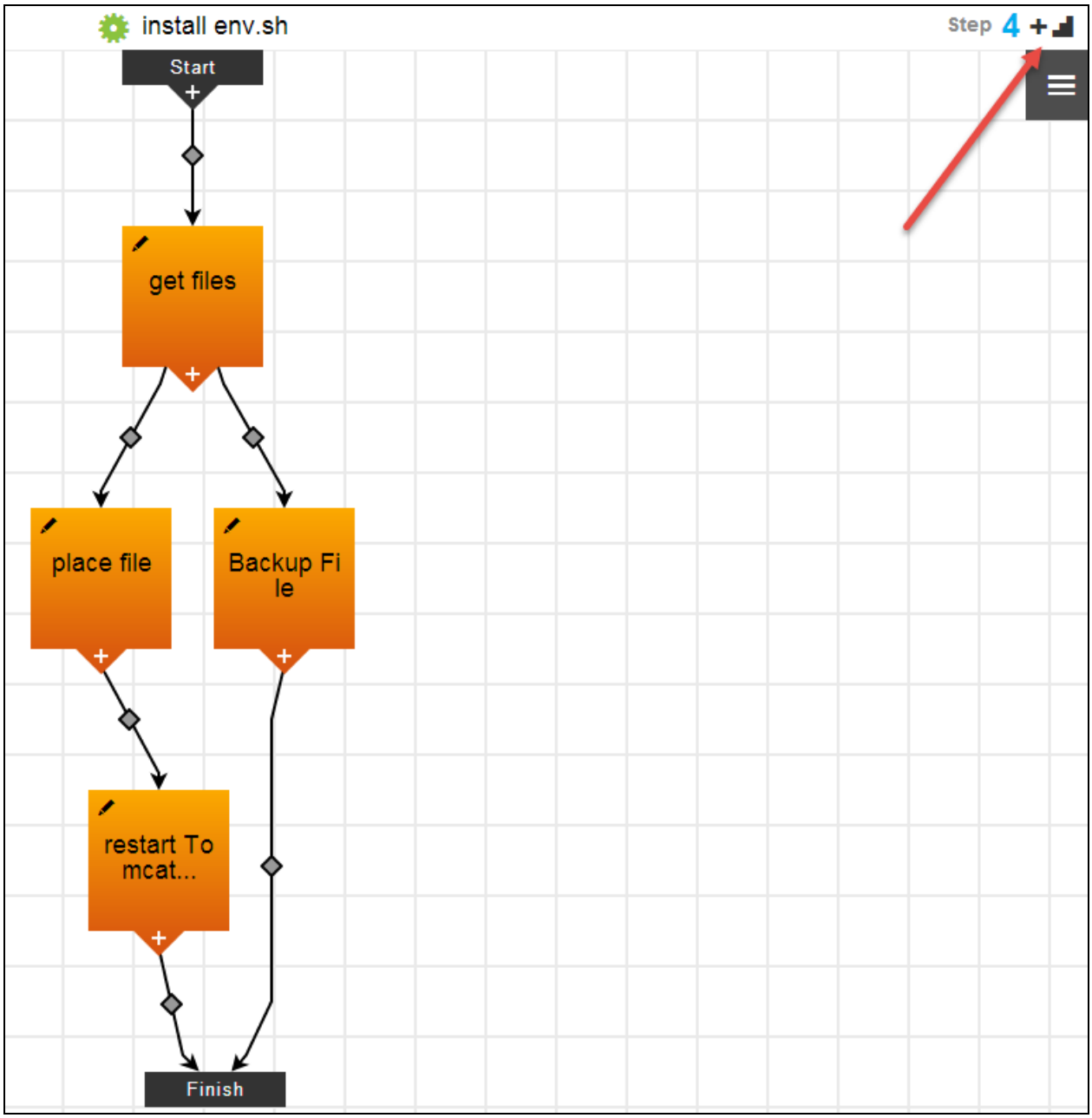
  From the Application Editor, click the **Add process** button in a component, set the details in the **Component Process** dialog box, and click **OK**.

  The Component Process Visual Editor for the component process appears.

To drag and drop a new step in a component or application process:

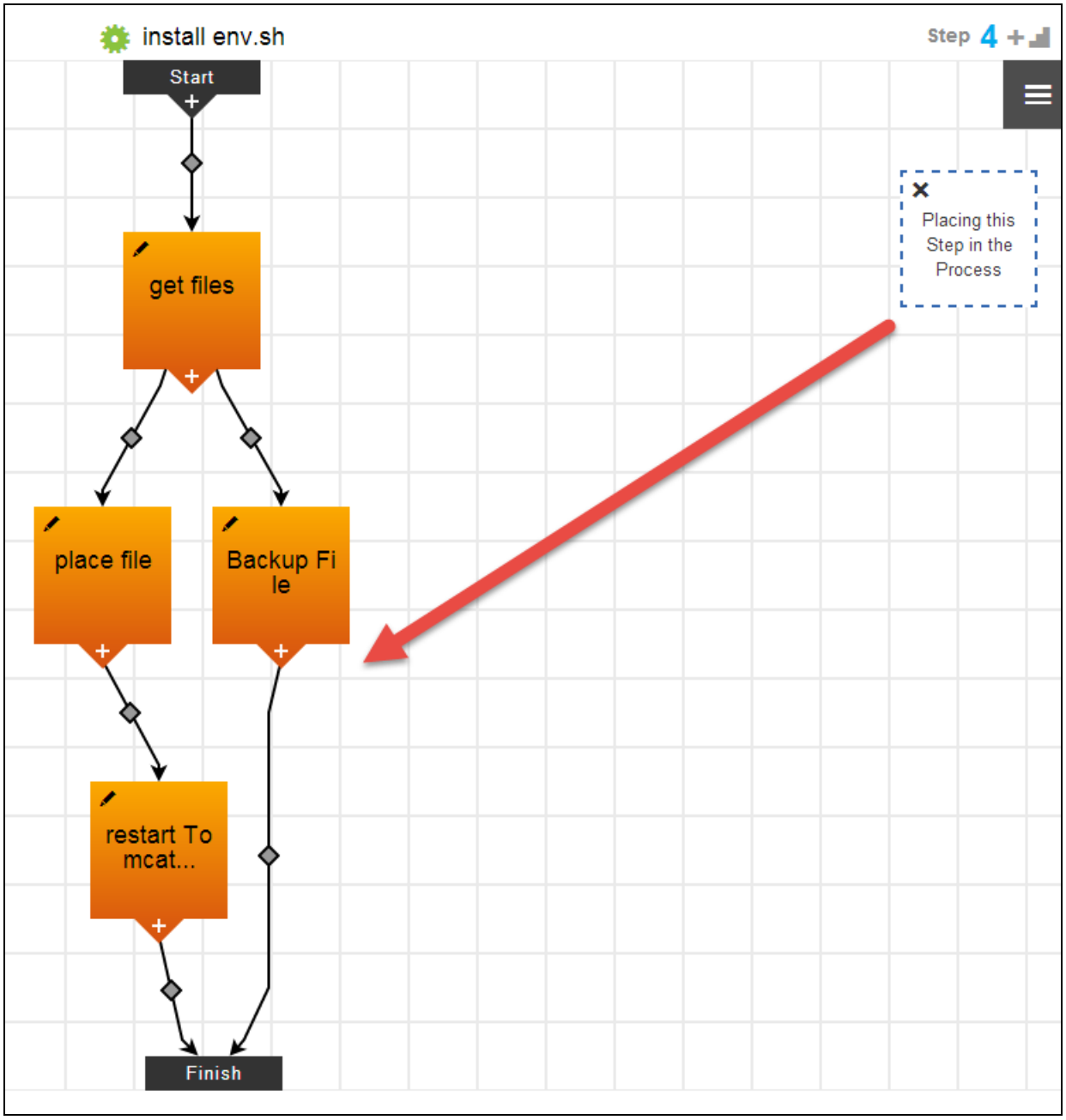
1. Click the **Add step** button in the upper right corner of the Component Process Visual Editor or Application Process Visual Editor.

   A new undefined step appears.



2. Select the new step.

3. Drag the step to where you want to add it in the process.



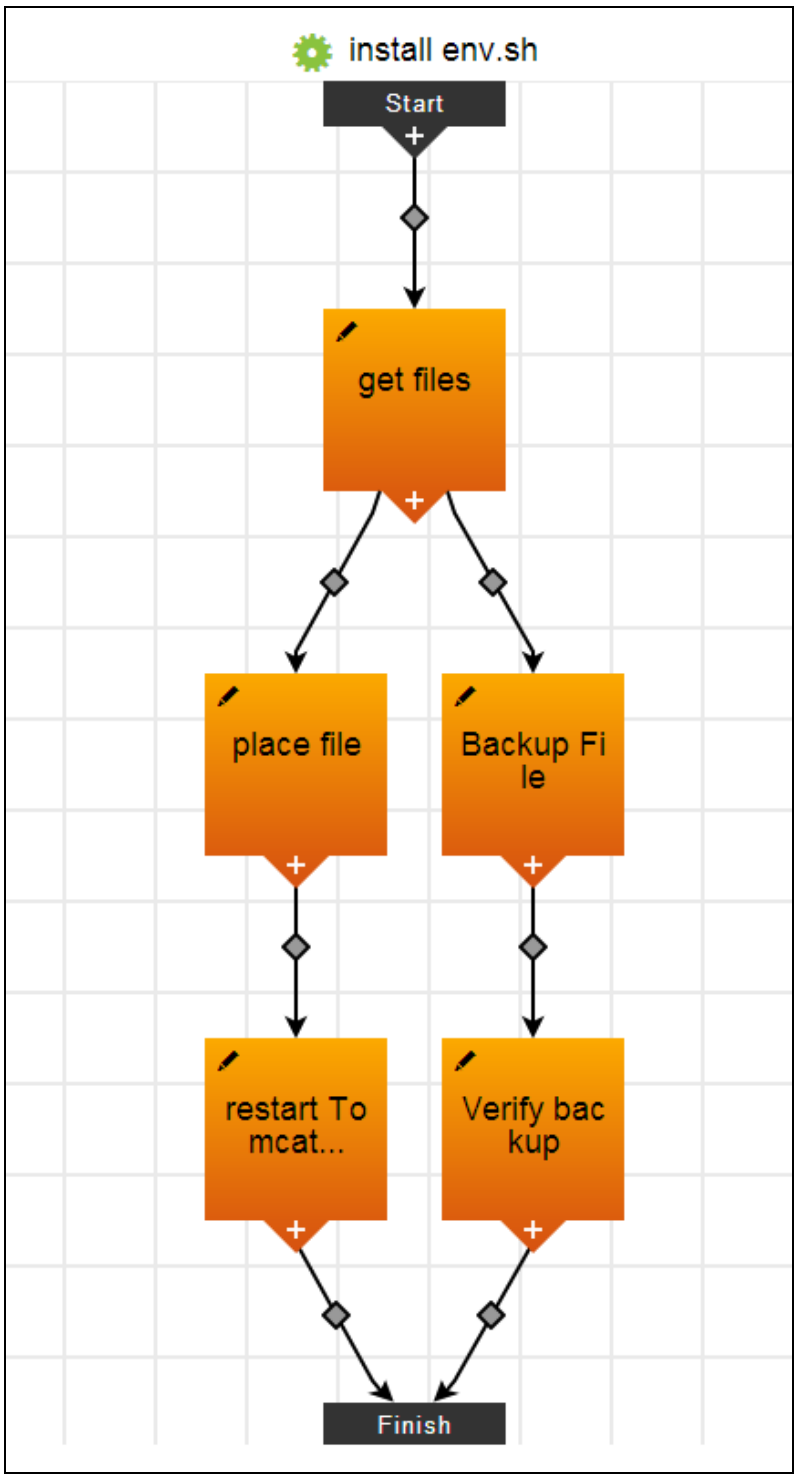
When you are near where you want to add the step in the process, notice that the icon changes shape and the text in it changes to "Dropping this Step in the Process."

4. Drop the step in the process.

The **Component Process Step** dialog box appears.

5. Enter information about the step.

   The new step is in the process.



Deploying Applications in Dynamic Environments

This section describes dynamic environments and how to model them. Environments can be static, dynamic, or hybrid. Any *dynamic environment* is automatically spun up when the application is deployed. Environments can have a combination of static and cloud resources that will be provisioned and configured dynamically. Go here for more information about static environments and how to model them.

Using dynamic environments allows you to:

- Provide ways to optimize how cloud resources are used.
- Reuse provisioned resource pools on an on-demand basis.
- Track how provisioned cloud resources are used.
- Provide the status of the provisioning process.
- Verify the credentials of cloud resources before provisioning them.

You provision cloud resources by creating a resource template and adding it to an environment template. When the application deployment that includes a dynamic environment begins, the environment is spun up at runtime with the required cloud resources, and the application is deployed there.

Resource templates allow you to

- Define the cloud provider details to provision of machines at the appropriate time and the configuration management tools to install the appropriate middleware.
- Configure the resources for optimal performance.

About Resources

Each resource in an environment has its own logical name to identify it from the other resources in the system. It also can be assigned to one or more resource pools or to a zone (a collection of agents). Several resources can correspond to the same physical host or agent machine. Resources can also be configured as standard or proxy. Standard resources are machines running the ElectricFlow agent on a supported agent platform while proxy resources (agents and targets) are on remote platforms or hosts that exist in your environment and requires SSH keys for authentication. The ElectricFlow agent does not need to run on the remote platform or host. See Resources on page 1208 for more information about to create, configure, and manage resources.

Examples

Infrastructure in ElectricFlow is modeled by configuring the cloud provider for dynamic environments. This configuration is versioned when you name and save the resource template. For an example of an Amazon EC2 infrastructure configuration, go to . For an example of an OpenStack infrastructure configuration, go here. Example: Resource Template using Amazon EC2 on page 355 and Configuring OpenStack as a Cloud Provider. For examples of infrastructure configurations, see Example: Resource Template using Amazon EC2 on page 355 and Configuring OpenStack as a Cloud Provider.

Middleware is modeled by configuring the configuration management tool for dynamic environments. This configuration is versioned when you name and saved the resource template. For an example of a middleware configuration, see Example: Resource Template using Chef on page 359.

Modeling Dynamic Environments

This is the high-level process to provision cloud resources that can be dynamically spun up during the application deployment.

- Create and define one or more resource templates.

  The resource template has the information required to spin up the resources on an on-demand basis an d to provision dynamic resource pools. It has the following information:

  - Cloud provider and cloud instance details
  - Configuration management settings
  - Both cloud provider and configuration management settings
  - Environment capabilities (see Full-Stack Dependency View on page 236 for more information)

- Create and define one or more environment templates.

  When modeling an environment template, you define the environment tiers and then add resources to the tiers. You can either add static resources or a resource template to an environment tier.

  When adding static resources, you can select one or more resources to add to the tier.

  When adding resource templates, you can select only one to add to the tier and then enter the number of resources to provision.

  Environment templates have this information:

  - Name and description
  - Environment tier details and properties
  - Resources assigned to the environment tiers
  - Cloud resource details from the resource templates

Usage Guidelines and Best Practices

Review the following information before creating and using dynamic environments.

- This matrix shows the resource types that are allowed in environments and environment templates.

Resource Type	Origin	Environment Tier	Environment Template Tier	Use Rules
Static resources	–	Yes	Yes	You may only add them to one environment. You cannot reuse them in more than one environment. You can add one or more static resources to an environment tier or an environment tier template.
Static resource pool	–	Yes	No	You may only add it to one environment. You cannot reuse it in more than one environment. You can add only one resource pool to an environment tier.
Resource pool provisioned in the resource template	Platform	Yes	No	You may only add it to one environment. You cannot reuse it in more than one environment. You can add only one resource pool to an environment tier.

Resource Type	Origin	Environment Tier	Environment Template Tier	Use Rules
Resource pool provisioned in an environment template	ElectricFlow	No	No	You cannot reuse the resource pools provisioned in an environment template. You can only use it in the dynamic environment created by the environment template.
Resource template	–	No	Yes	You may only add one resource template to an environment template tier.

In environments and environment templates, you can create a tier with static resources, a static resource pool, or a static resource template. When you are editing the tier, ElectricFlow maintains the initial resource-type association.

For example, if you initially add static resources to an environment tier, you can add only static resources to it. You are not allowed to add a resource this environment tier.

- ElectricFlow validates the information you provide for configuring and provisioning the cloud provider.

  For instance, when you configure Amazon EC2 as the cloud provider, ElectricFlow validates the Service URL using the Access ID and Access Key. The configuration information that you provide is also used to retrieve and display the available options from the cloud provider service for the provisioning parameters such as the Image or the Security group to use.

  However, if your ElectricFlow server is unable to access external services such as the Amazon EC2 service, you may disable this validation and the dynamic options retrieval capability for the cloud provider plugin by setting the plugin property `ec_disable_dynamic_operations` to `1`.

Creating AMIs

Before creating a resource template, you must create an Amazon Machine Image (AMI) with a pre-installed agent for the cloud provider. In ElectricFlow 5.4, Amazon EC2 and OpenStack are the supported cloud providers.

AMIs with ElectricFlow Agents

1. Install agents on ElectricFlow server node machines.

   See the following sections in the *ElectricFlow Installation Guide* at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html for detailed instructions:

   - "Agent Platforms" section in the "System Requirements and Supported Platforms" chapter
   - "Server and Agent Compatibility" section in the "System Requirements and Supported Platforms" chapter
   - "Express Agent Command-Line" section in the "Installing ElectricFlow" chapter
   - "Non-Server Platform Agent Installation Method" section in the "Installing ElectricFlow" chapter
   - "Configuring Proxy Agents" section in the "Configuration" chapter

2. Create the AMI on the Amazon EC2 or OpenStack platform.

   - See http://aws.amazon.com/amazon-linux-ami/ for Amazon EC2.
   - See http://docs.openstack.org/image-guide/content/index.html for OpenStack.

3. Create a resource template in ElectricFlow.

   When you set the cloud provider, you can select **Amazon** or **OpenStack** and enter information in the fields in the dialog boxes. The fields that you see depend on the cloud provider that you select.

   - For detailed Amazon EC2 instructions, see Configuring Amazon EC2 as the Cloud Provider.
   - For detailed OpenStack instructions, see Configuring OpenStack as the Cloud Provider.

AMIs with ElectricFlow Agents and Chef Configuration Management

1. Install agents on ElectricFlow server node machines.

   See the following sections in the *ElectricFlow Installation Guide* at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html for detailed instructions:

   - "Agent Platforms" section in the "System Requirements and Supported Platforms" chapter
   - "Server and Agent Compatibility" section in the "System Requirements and Supported Platforms" chapter
   - "Express Agent Command-Line" section in the "Installing ElectricFlow" chapter
   - "Non-Server Platform Agent Installation Method" section in the "Installing ElectricFlow" chapter
   - "Configuring Proxy Agents" section in the "Configuration" chapter

2. Configure a Chef server with run-lists that will be applied to your cloud resources.

   See http://docs.chef.io/ for details.

3. Create the AMI on the Amazon EC2 or OpenStack platform.

   - See http://aws.amazon.com/amazon-linux-ami/ for Amazon EC2.
   - See http://docs.openstack.org/image-guide/content/index.html for OpenStack.

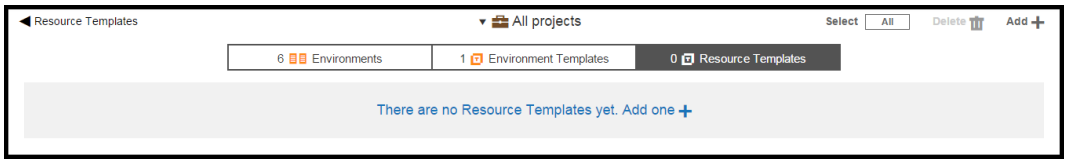
4. Create a resource template in ElectricFlow.
    - When you set the cloud provider, you can select **Amazon** or **OpenStack** and enter information in the fields in the dialog boxes. The fields that you see depend on the cloud provider that you select.
        - For detailed Amazon EC2 instructions, see Configuring Amazon EC2 as the Cloud Provider.
        - For detailed OpenStack instructions, see Configuring OpenStack as the Cloud Provider.
    - When you set the configuration management tool, select **Chef**.

      For detailed instructions, see Configuring Chef as the Configuration Management Tool.

Creating Resource Templates

1. Open the **Resource Templates** list.

   If there are no defined resource templates, the **Resource Templates** list is empty. For example:

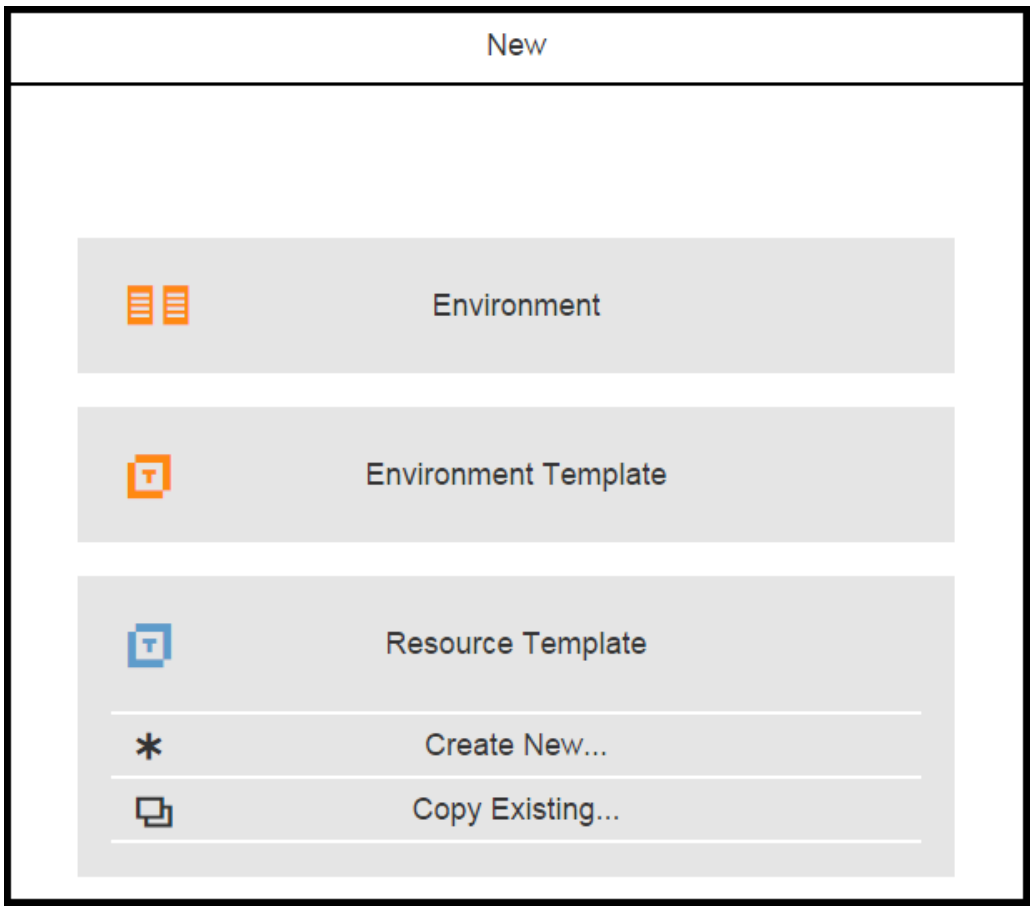


2. Add a resource template.

   To do so, do one of the following:

   - If the **Resource Templates** list is empty, click either **There are no Resource Templates yet. Add one +** or click the **Add +** button.
   - If the **Resource Templates** list is not empty, click the **Add +** button.

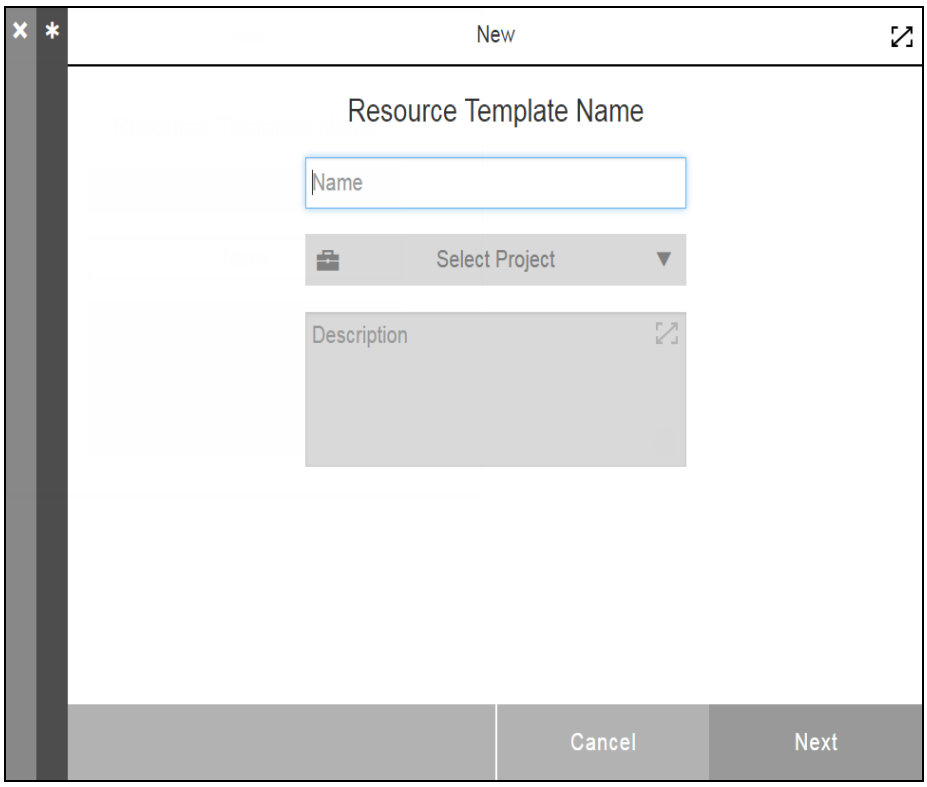
   The **New** dialog box opens:



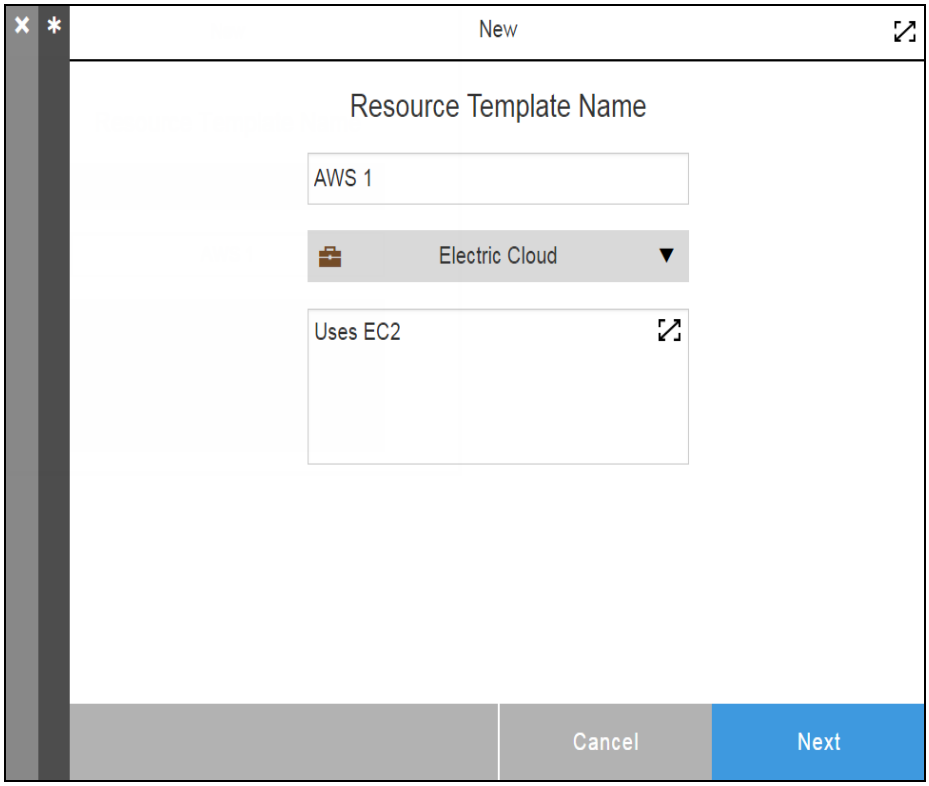
3. In the **New** dialog box, click **Resource Template**.

4. Click **Create New** to create a new resource template.

   The **New Resource Template Name** dialog box opens:

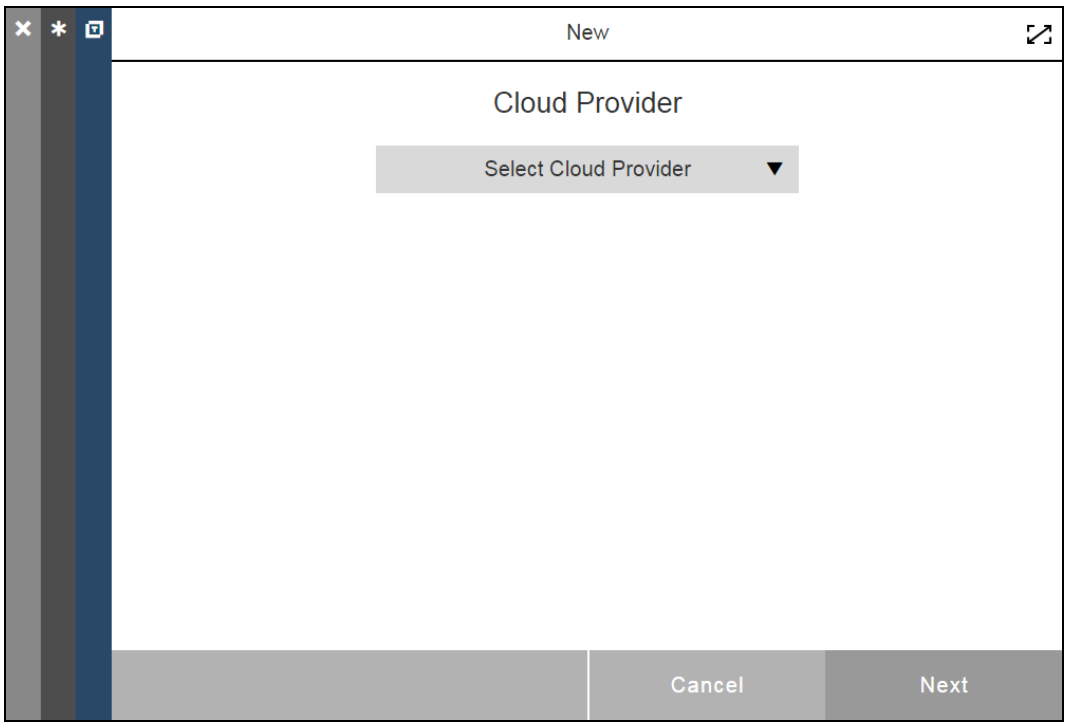


5. Enter a name, select a project to which it will belong, and (optional) enter a description. For example:



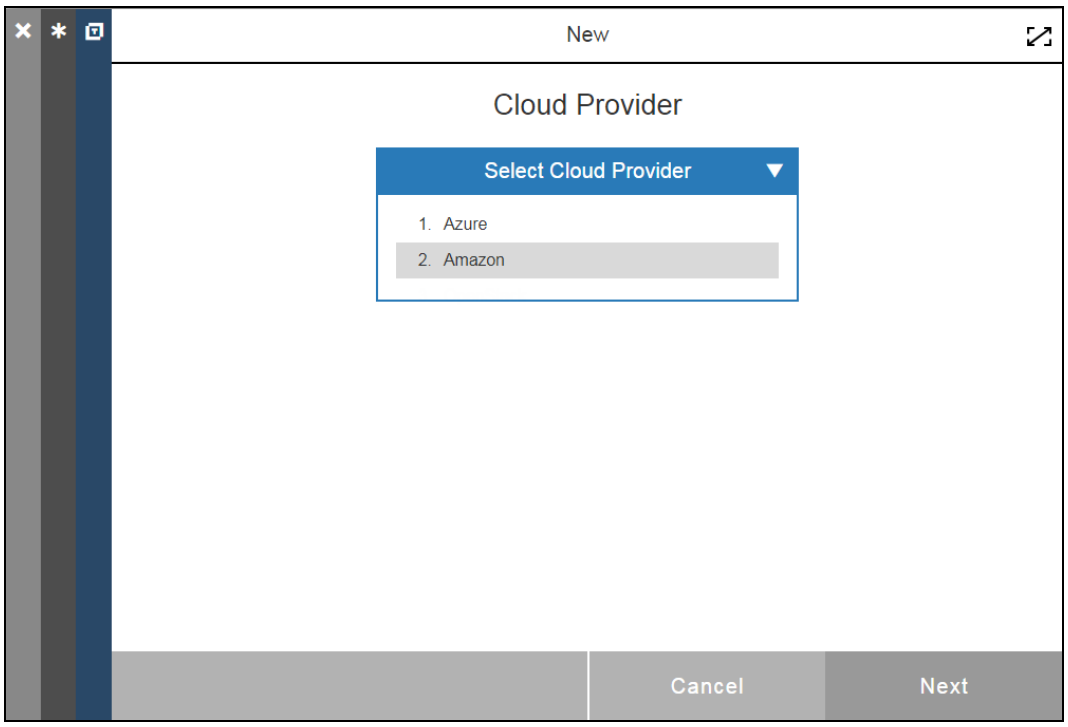
6. Click **Next**.

   The **New Cloud Provider** dialog box appears:



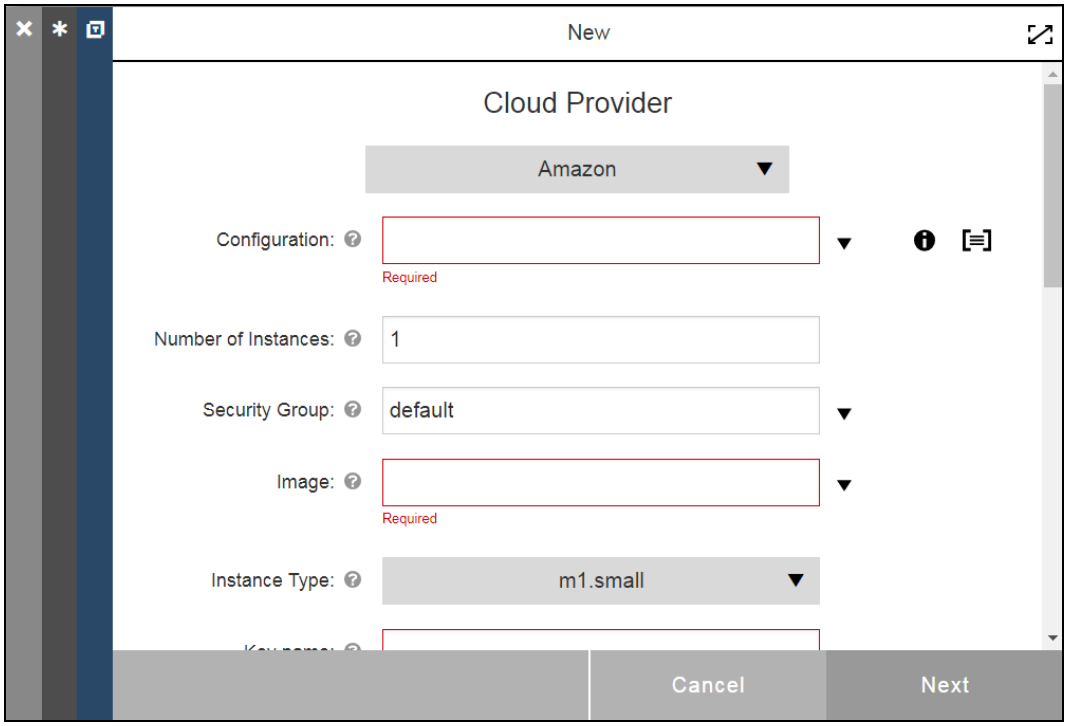
7. Choose a cloud provider from the **Select Cloud Provider** pulldown menu.

   For example:



8. Click **Next**.

   The **New Cloud Provider** dialog box becomes populated with additional settings for the cloud provider that you selected. For example:

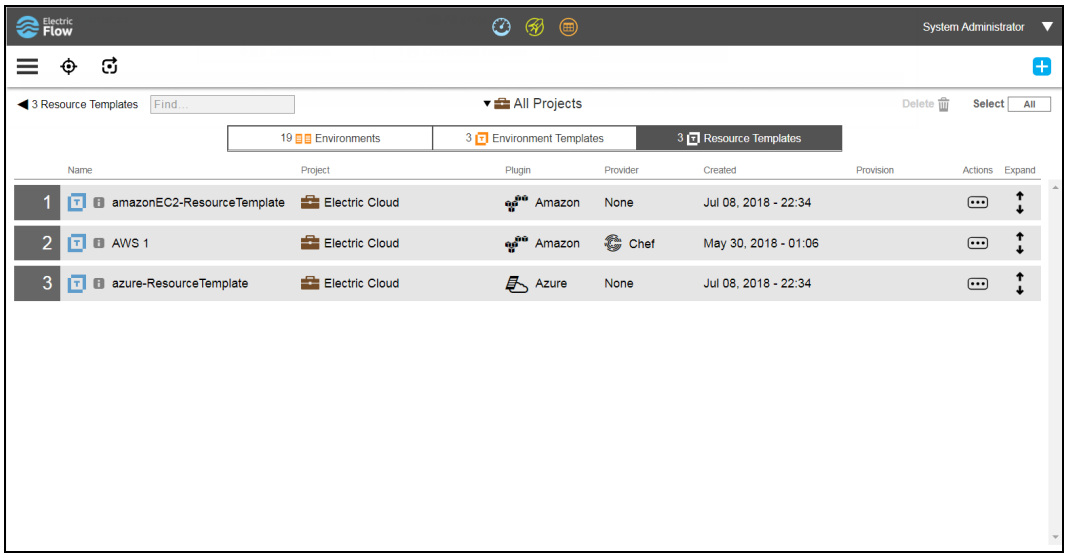


9. Enter your settings into the **New Cloud Provider** dialog box.

   - To use Amazon EC2 as the cloud provider, see Example: Resource Template using Amazon EC2 on page 355 for detailed instructions.

     Important: Before configuring the cloud provider, you must have an Amazon Machine Image (AMI) with a pre-installed agent available.

After you complete these steps, the **Resource Templates** list now shows the template that you just created. The breadcrumb in the upper left corner shows the object hierarchy.



By default, the resource templates for all projects are displayed.

To see only the objects for a specific project, click the down arrow in the **All projects** field and then select one or more projects in one of these ways:

- Click on the name of one or more projects.
- Enter the search criteria in the Search field. The projects that match the criteria appear in the list.

If there are no matches, a message appears stating that there are no resource templates in the selected projects.

Example: Resource Template using Amazon EC2

This section describes how to configure Amazon EC2 as the cloud provider for a resource template.

> **Important:** Before configuring the cloud provider, you must have an Amazon Machine Image (AMI) with a pre-installed agent available.

To enter the cloud provider settings:

1. Choose **Select Cloud Provider** > **Amazon**.

2. Enter or select the provisioning settings:

   - **Configuration**—The name of the configuration that holds the connection information. This must reference a valid existing configuration. This menu is prepopulated with the available configurations.
   - **Number of Instances**—The number of instances to start. The default is 1.
   - **Security Group**—The default is *default*.
   - **Image**—Select the EC2 image to deploy. The drop-down menu displays the images that you own in your current region. (Required)

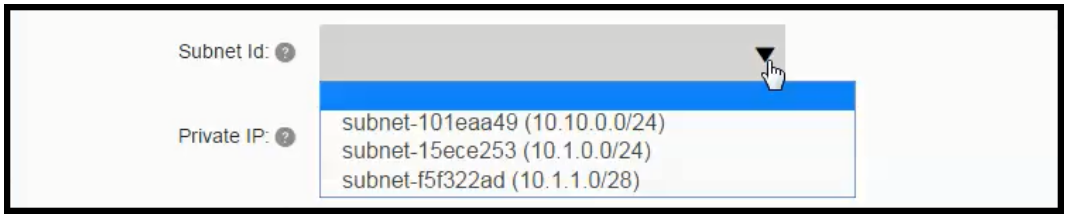
   - **Instance Type**—You get the list of instance types at runtime. The default is **Small (m1.small)**.
   - **Key name**—Select the name of the key pair to use. (Required)

   - **Results Location**—Location where the output properties will be saved. If not specified, it defaults to `/myParent/parent`.
   - **User Data**—User data passed to the newly launched instance in Amazon EC2 for performing common automated configuration tasks and running scripts after the instance starts.
   - **Availability Zone**—The zone where the instance will be created. (Required)

     Selecting a zone causes ElectricFlow to find a subnet in the selected zone.

- **Subnet Id**—The ID of the subnet in which the instances are launched. This parameter is used with VPCs.

  After selecting the zone where the instance will be created, you can select a subnet in that zone.

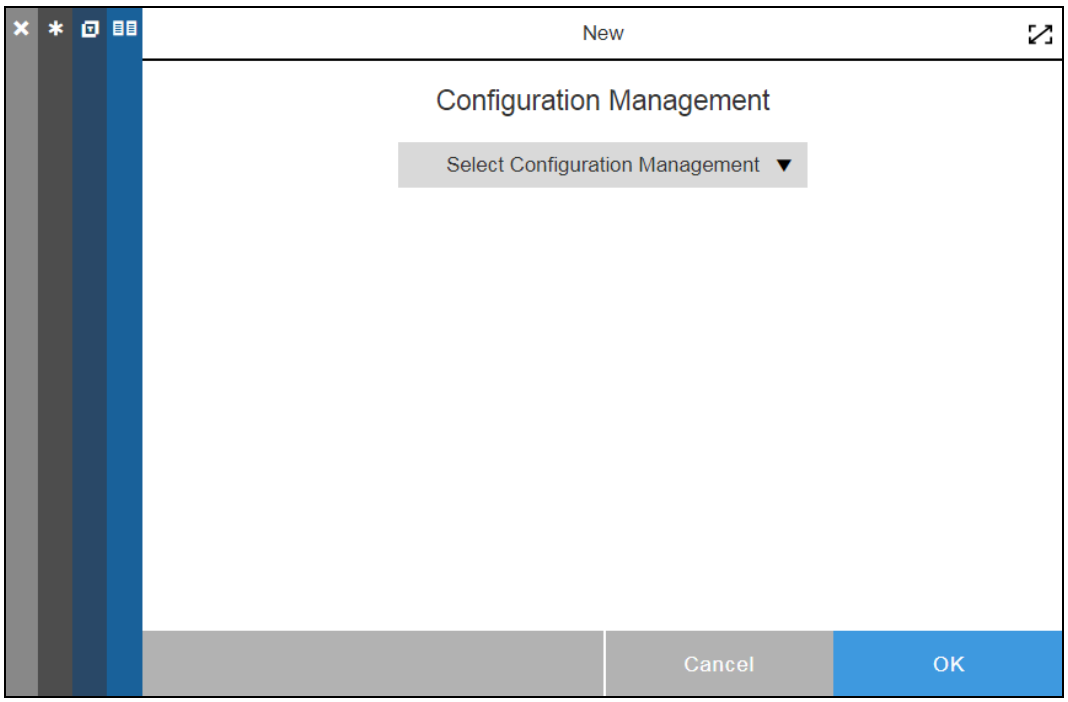


- **Private IP**—The private IP address to assign. If you specify a value, it must be from the IP address range of the subnet. If no value is specified, an available IP address from the IP address range of the subnet is selected. This parameter is used with VPCs.
- **Use Private IP for subnet?**—If this check box is selected, the private IP address assigned to the instance will be used for the newly created resource. This is enabled by default.
- **Instance Initiated Shutdown Behavior**—Specify the instance behavior when an OS-level shutdown is performed. Instance can be either terminated or shut down.
- **Tenancy**—Each instance that you launch into a VPC has a tenancy attribute to indicate what it runs on. Choose one: **default** (shared hardware), **dedicated** (single-tenant hardware), or **host** (dedicated host, which is an isolated server with configurations that you can control).
- **Resource Pool**—The name of the resource pool. If you enter a name, a new resource pool is created, and new resources are added to the pool for the instances created in Amazon EC2.

- **Resource Port**—If you specify a resource pool name in 'Resource Pool' field, this is the port that will be used when creating the resource. If no value is specified, port 7800 will be used by default when creating the resource.
- **Commander Workspace**—If you specify a resource pool name in **Resource Pool** field, this is the workspace that will be used when creating the resource.
- **Resource Zone Name**—The zone to which the newly created resources will belong. The default zone is *default*.

3. Click **Next** to enter the configuration management settings.

   The **New Configuration Management** dialog box appears:



4. If you do not want to enter the configuration management settings, click **OK**.

   The **Resources Templates** list now shows the resource template that you created.

5. Go to this step in Creating Resource Templates on page 348 to continue creating the resource template.

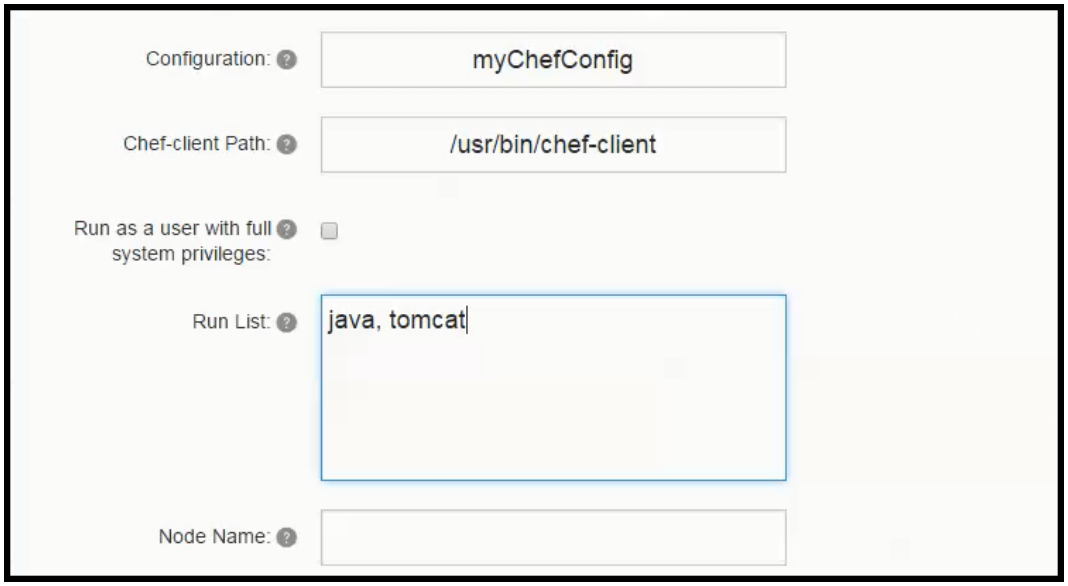
Example: Resource Template using Chef

This section shows how to configure Chef as the configuration management tool for a resource template after you have entered the cloud provider settings.

Select configuration management

To enter the configuration management settings:

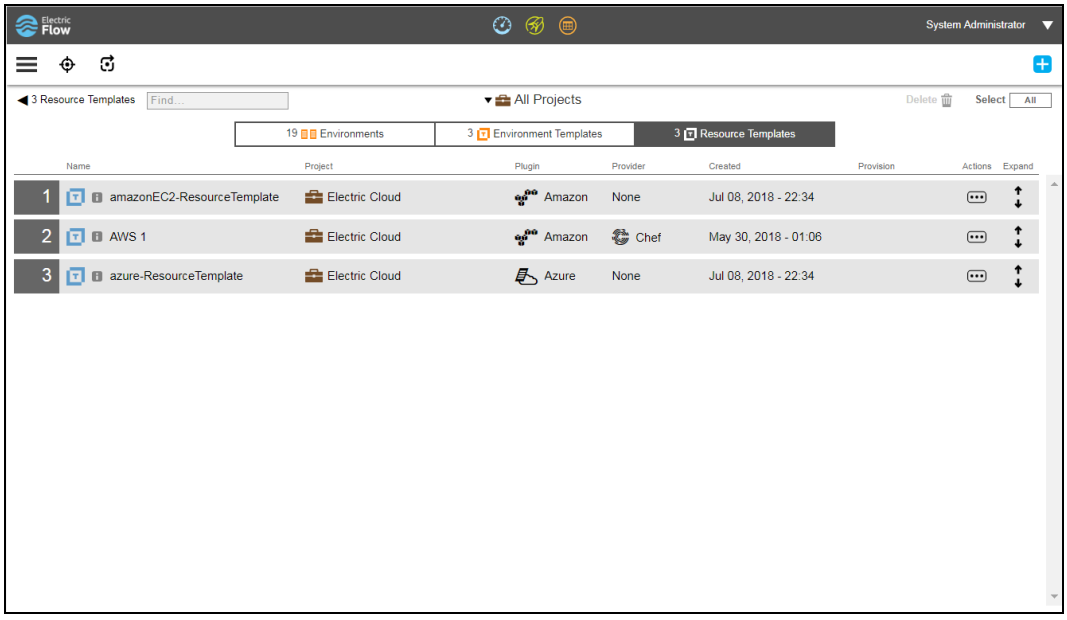
1. In the **New Configuration Management** dialog, choose **Chef**.
2. Enter these settings:
   - **Configuration**–Name of the Chef configuration. This is required.
   - **Chef-client Path**—The path to the chef-client executable. The default value is `/usr/bin/chef-client`. (Required)
   - **Run as user with full system privileges**–If this check box is selected, chef-client is run as a user with elevated privileges on Unix systems (using sudo).
   - **Run List**—The ordered list of Chef recipes to run. This is required.



   - **Node Name**–Unique name for the node that will be maintained by the chef-client. If not specified, Chef will use the Fully Qualified Domain Name of the node as the node name. (Required)
   - **Additional Arguments**–Any additional arguments that need to be passed to the chef-client.

3. Click **OK**.

   The **Resource Templates** list now shows the resource template that you created. For example:



Viewing and Editing Resource Templates

You can view and edit this information about resource templates:

- Resource Template Details on page 361
- Viewing and Editing Resource Templates on page 361
- Viewing and Editing Resource Templates on page 361
- Viewing and Editing Resource Templates on page 361

Resource Template Details

Starting in the **Resource Templates** list, follow these steps to view and edit the resource template details:

1. Choose a resource template and click the **Menu** button.

   A list of menu options appears. For example:

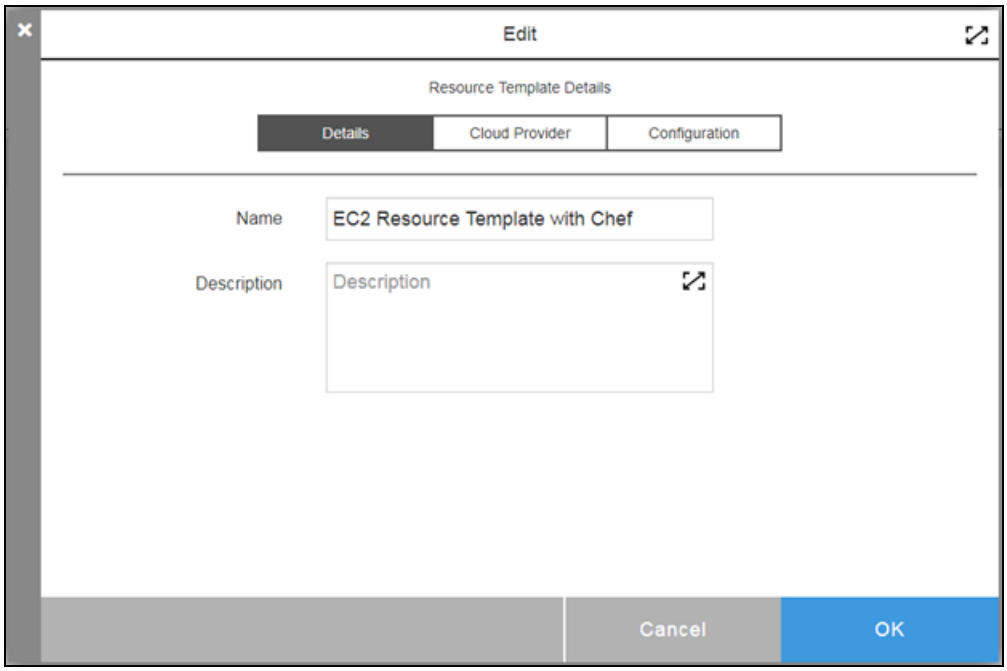
To see only the objects for a specific project, click the down arrow in the **All projects** field and then select one or more projects in one of these ways:

- Click on the name of one or more projects.
- Enter the search criteria in the Search field. The projects that match the criteria appear in the list.

If there are no matches, a message appears stating that there are no resource templates in the selected projects.

2. Select **Details**.

   The **Edit Resource Template Details** dialog box opens. For example:



3. In the **Details** tab, edit the name and description of the resource template.
4. In the **Cloud Provider** tab, edit the cloud provider account information.
5. In the **Converge** tab, view and edit the details about how the virtual instances are converged in the defined configuration.

6. Click **OK** to save your changes.

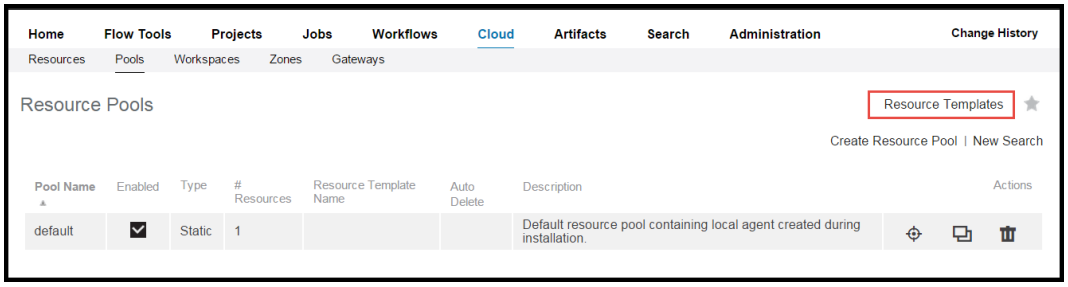
Accessing the Resource Templates in the Automation Platform

A resource template has the required information to provision and spin up cloud resources on an on-demand basis to model a dynamic environment. You set the cloud provider and configuration management details in the resource template. Then you define the environment tiers and add resource templates to these tiers in an environment template to complete the dynamic environment configuration.

You can generate new resource pools in ElectricFlow by accessing the resource templates from the automation platform UI.

- Go to **Cloud** > **Pools**.

  Example:



- Click **Resource Templates**.

  The **Resource Templates** list opens.

  Example:

- (Optional) To view details about a resource template, choose a template and click the **Menu** button

  ( ) for it.

  The Context menu opens.

- Click one of these options:
  - **Details**—The **Edit Resource Template Details** dialog box opens.

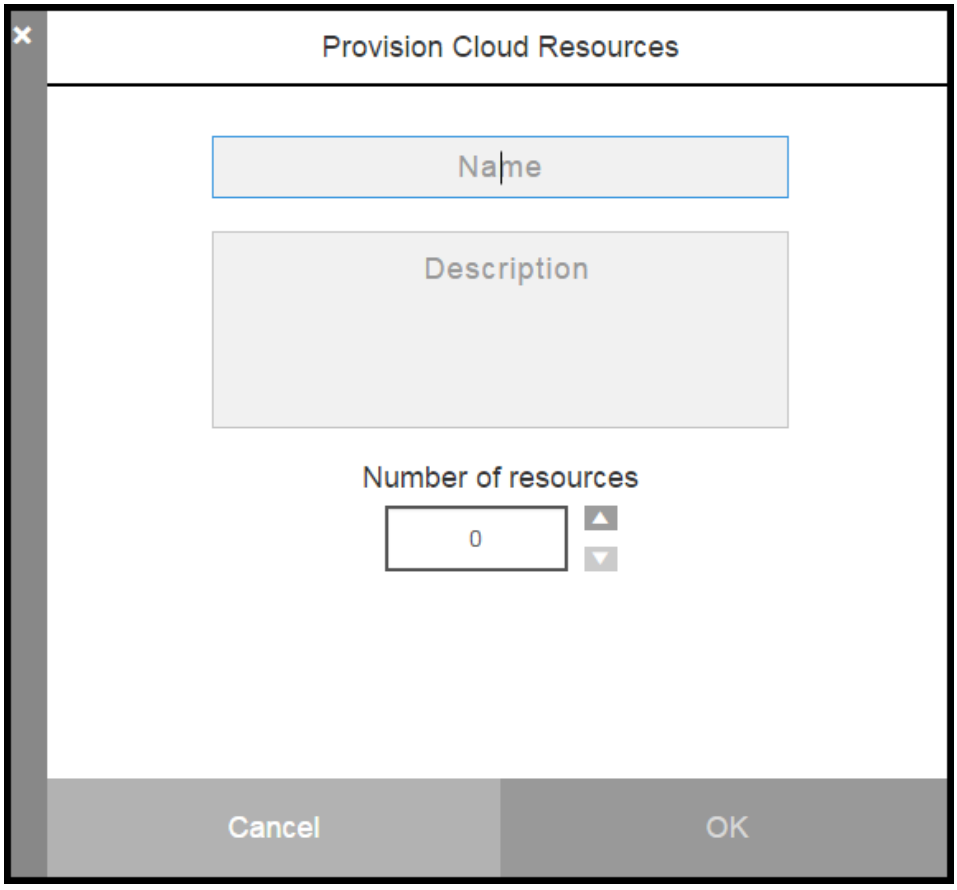
    The tabs that appear in the dialog box depend on the cloud provider specified in the resource template.
  - **Properties**—The **Properties** dialog box opens.

    You can view and edit the properties that apply to the resource template.
  - **Access Control**—The Access Control page opens.
  - **Track Changes**—The Change History for the resource template opens.

- To provision the resource template, choose a resource template, click the **Menu** button ( ), and select **Details** > **Provision**.

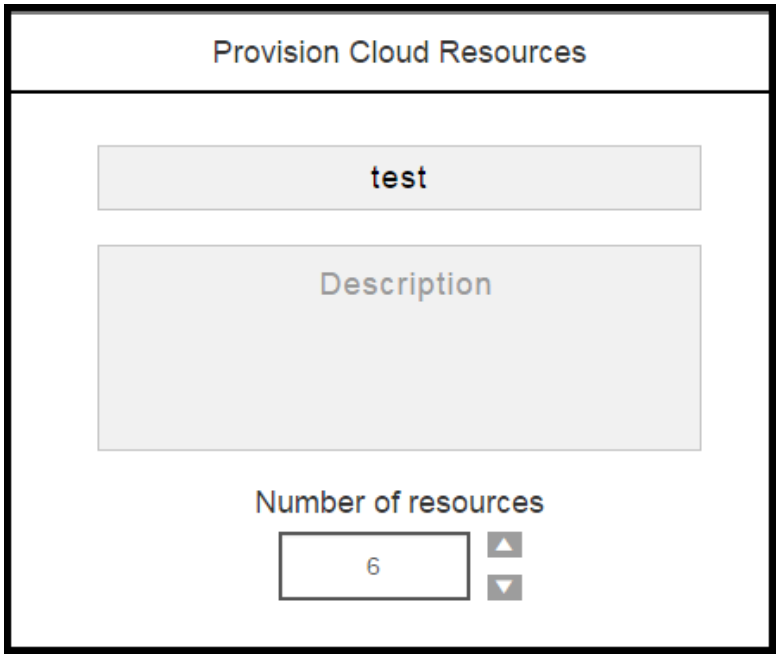
  The **Provision Cloud Resources** dialog box opens.

  ## Example:



- Enter the resource pool name, an optional description, and the number of resources in the pool, and then click **OK**.

  Example:



  You go to the Job Details page in the automation platform.

Creating Environment Templates

You can create a new environment template from scratch or create one based on an existing template.

- Creating a New Environment Template on page 365
- Creating an Environment Template Based on an Existing Template on page 377

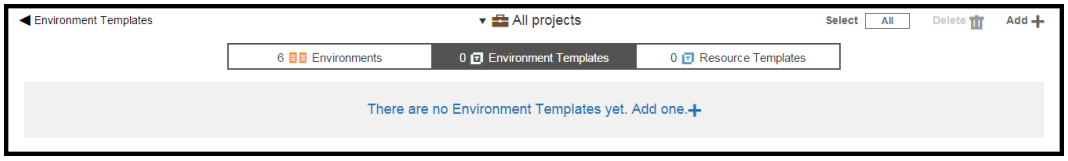
Creating a New Environment Template

Starting from the Environment Templates List, follow these steps to create an environment template from scratch:

- Go to the Environment Templates List.

  If there are no defined environment templates, the Environment Templates List is empty.

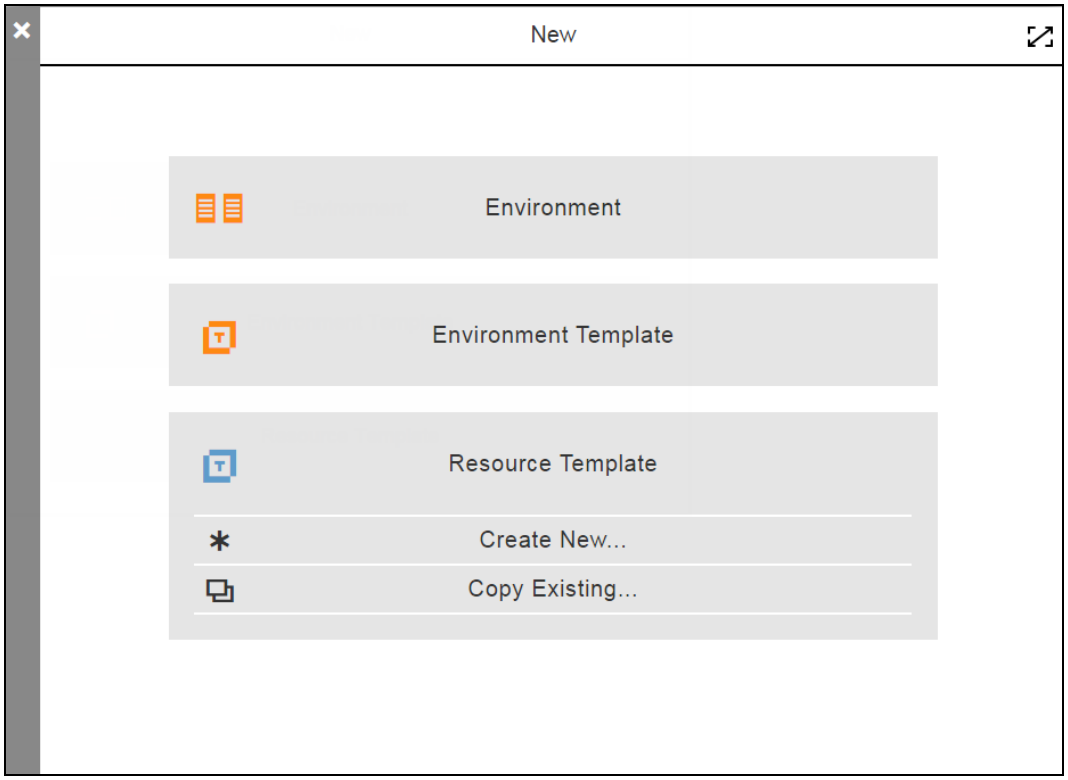
Example:


Environment Templates
All projects
Select
All
Delete
Add
6 Environments
0 Environment Templates
0 Resource Templates
There are no Environment Templates yet. Add one.

- Add an environment template.
  - If the Environment Templates List is empty, click **There are no Environment Templates yet. Add one +** or click the **Add +** button.

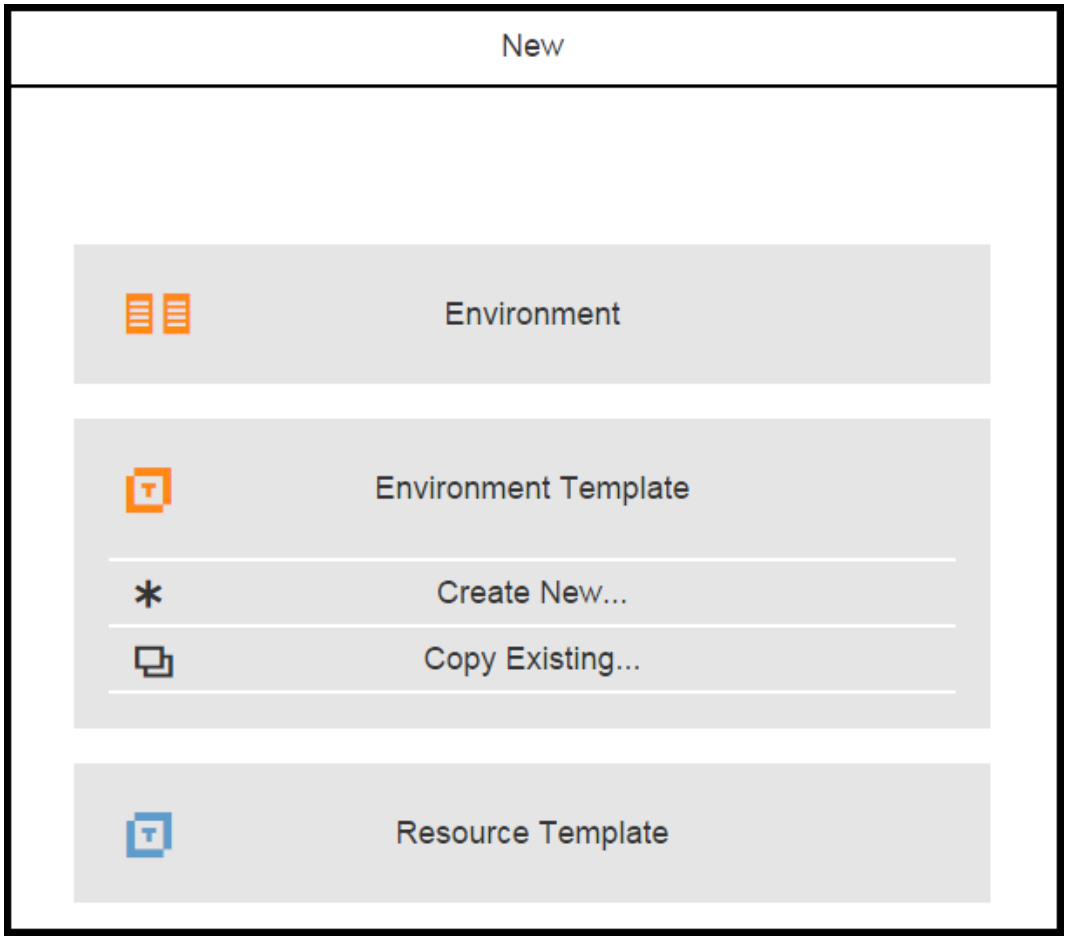
    In the **New** dialog box, click **Environment Template**.

    Example:

    The **New** dialog box opens.

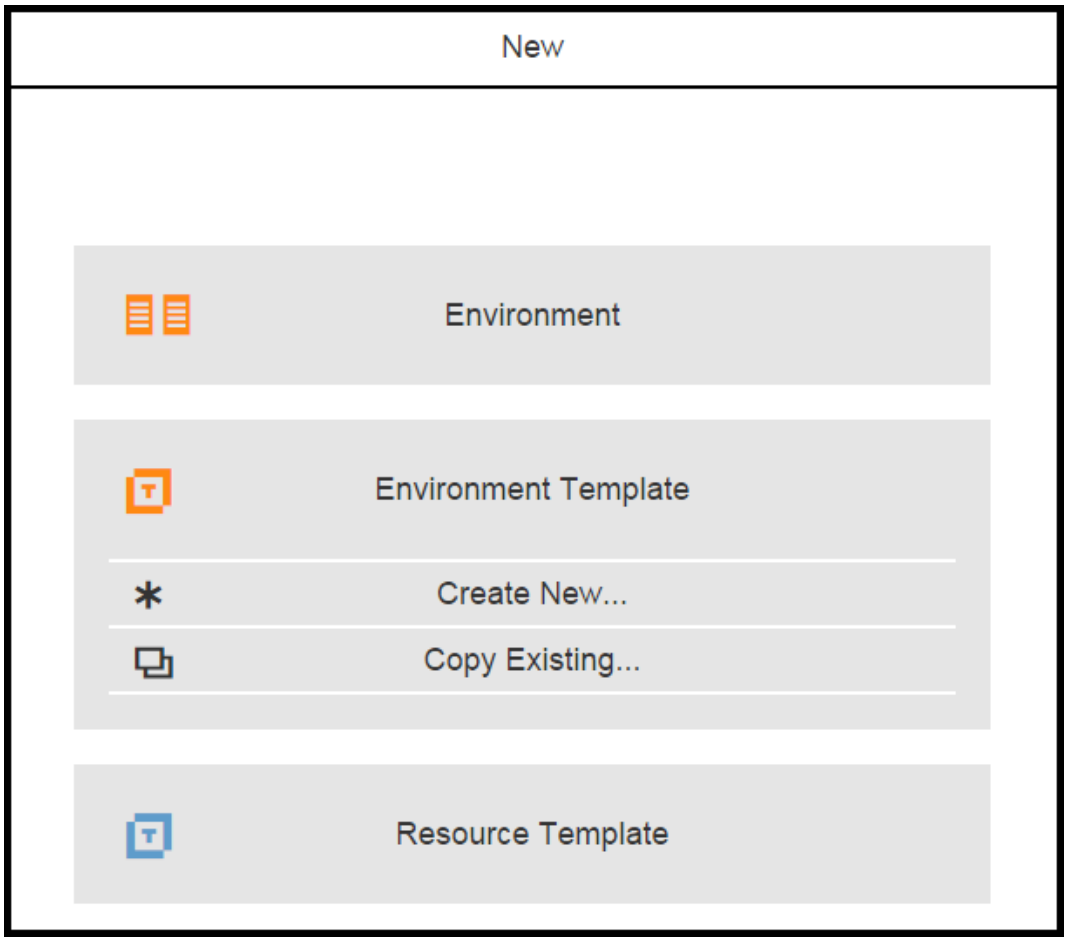
Example:


New
Environment
Environment Template
Create New...
Copy Existing...
Resource Template

- If the Environment Templates List is not empty, click the **Add +** button.

  The **New** dialog box opens.

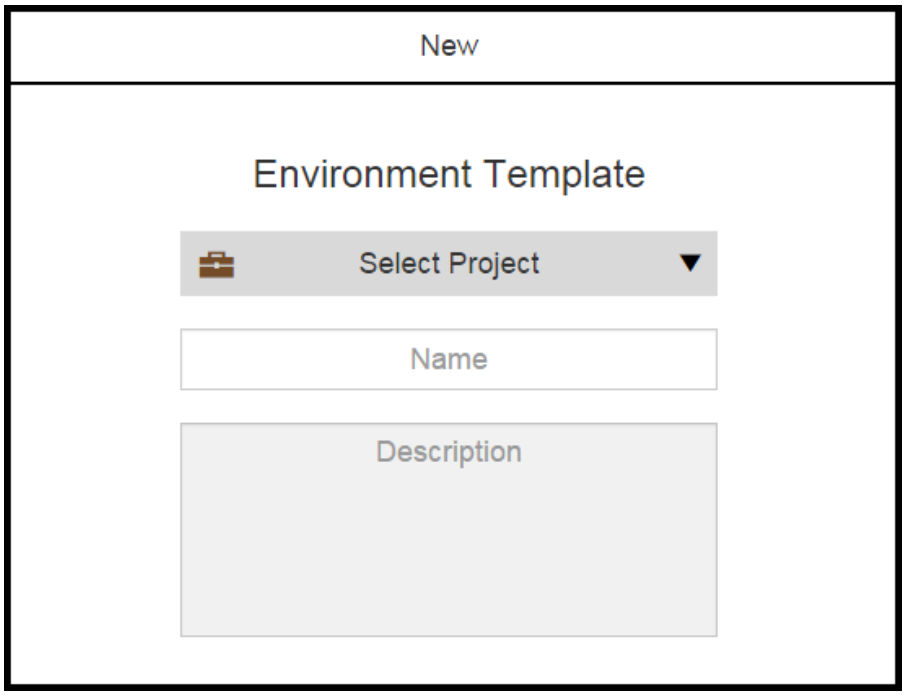
  Example:



- Click **Create new** to create a new environment template.

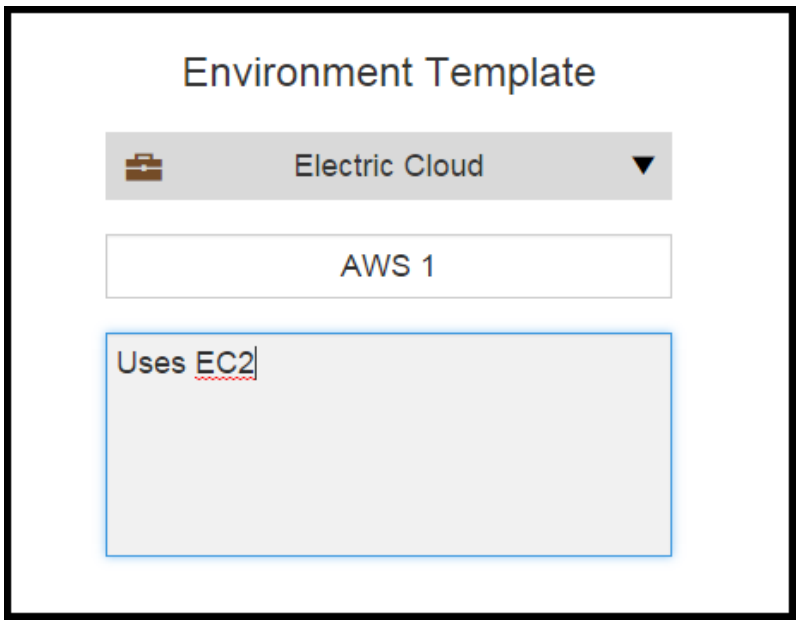
  The **New** dialog box opens.

  ### Example:



- Enter a name for the environment template, select a project to which it belongs, and enter an optional description.

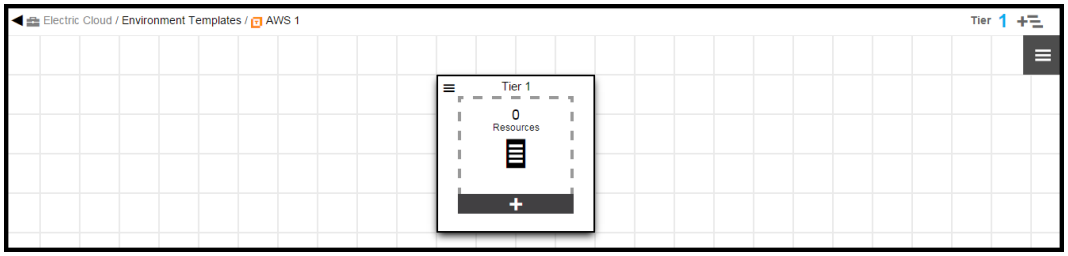
  Example:



- Click **OK**.

  The Environment Templates Visual Editor opens. The environment template has one tier with no assigned resources. The breadcrumb in the upper left corner shows the object hierarchy.

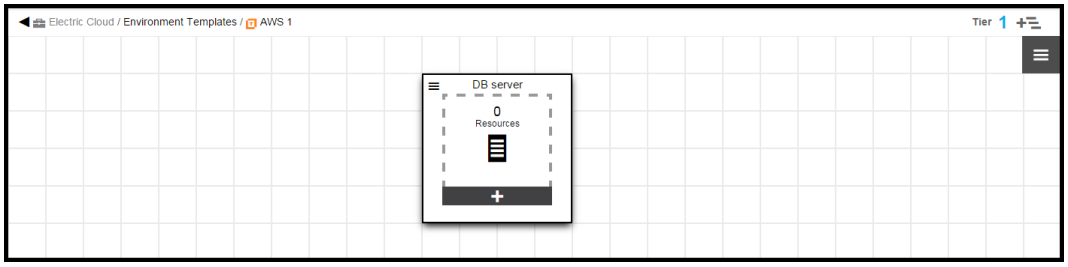
  Example:



- In Tier 1, click the  button and select **Details**.

- In the **Environment Tier Details** dialog box, enter a new name for the environment tier and an optional description, and then click **OK** to save the settings.

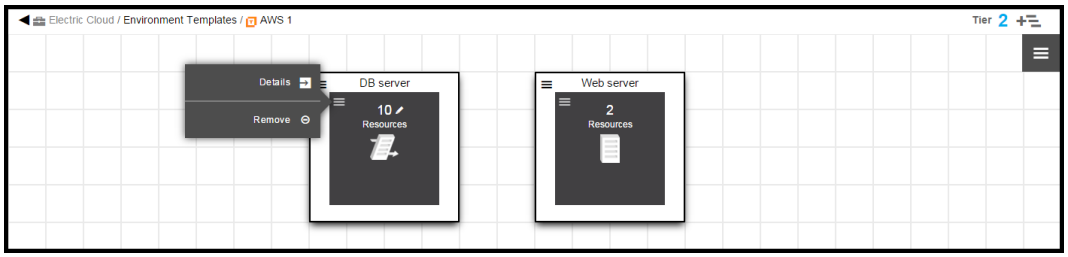
  ## Example:

- In the tier, click the **+** button to add a resource to it.

  The **New** dialog box to add resources opens.

  ## Example:

- Click **Add resource template** to select a resource template.

  The Resource Templates List opens.

- Select a resource template, and click **OK**.

  The dialog box to select a resource template opens.

  It shows the list of available environment templates. The default is to show the environment templates for all projects.

  If you want to see only the environment templates for a specific project, click the down arrow in the **All projects** field to select one or more projects one of these ways:

  - Click on the name of one or more projects.
  - Enter the search criteria in the Search field. The projects that match the criteria appear in the list.

  You can also search for one or more environment templates by entering the search criteria in the **Search** field next to the **All projects** field. If there are no matches, a message appears stating that there are no environment templates in the selected projects.

  ## Example:

  The **New** dialog box to select the number of resources to provision opens.

  ## Example:


New
Number of Resources to Provision
1
Cancel
OK

- Enter the number of resources to provision, and click **OK**.

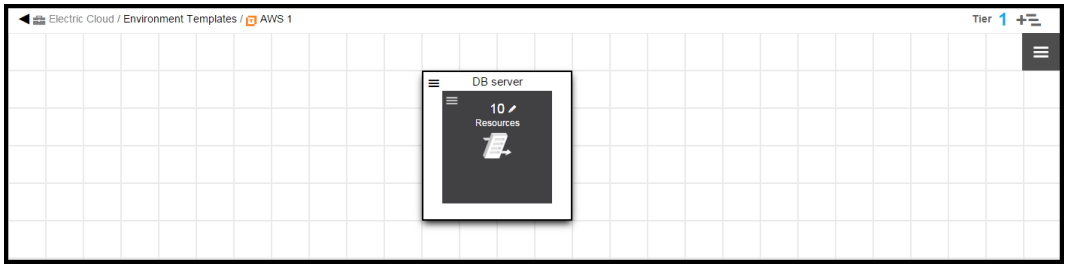
  You must provision at least one resource (**1**).

  > **Note:** You cannot override the number of resources when deploying an application with an environment template.

  Example:

The Environment Templates Visual Editor now shows that the tier (Tier 1) has 10 dynamic cloud resources set to be provisioned when the application is deployed.

Example:



- To add static resources to an environment template:
  - Click the **Add tier** button to create a new tier.

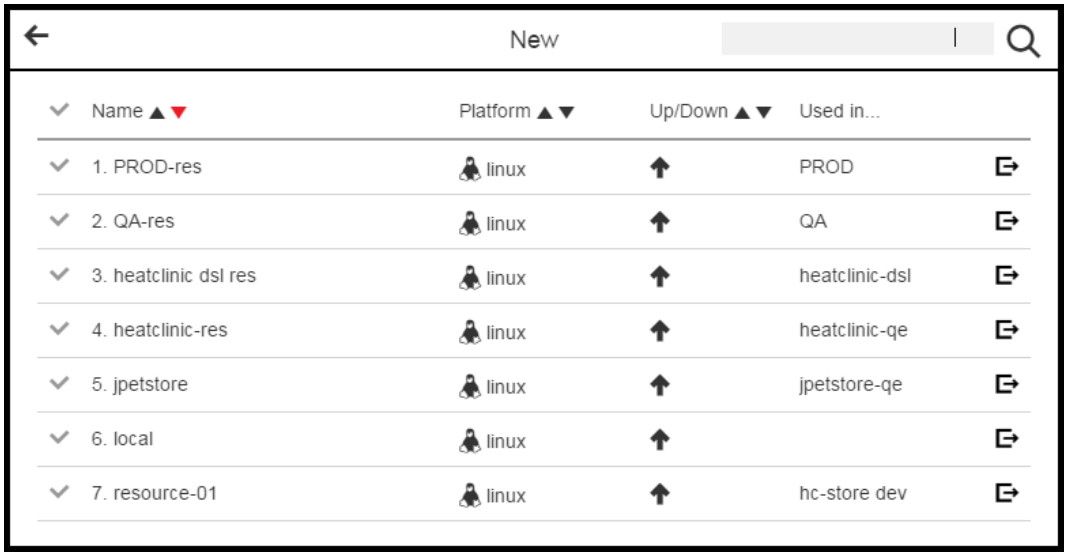
    A new environment tier appears in the Environment Templates Visual Editor.
  - Rename the new environment tier.
  - Click the **+** button in the new tier to add resources to it.

    The **New** dialog box opens.
  - Click **Add resources** to add static resources to the environment tier.

    A list of static resources opens.

    Note: You cannot override the setting for the number of static resources when deploying the application with an environment template

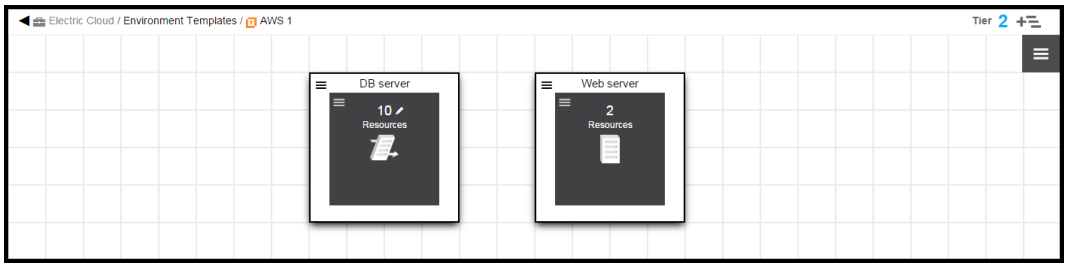
Example:



- Select one or more resources in the list, and click **OK**.

  The Environment Templates Visual Editor now shows that the "Web server" tier has two static resources.

  Example:



Creating an Environment Template Based on an Existing Template

Starting from the Environment Templates List, follow these steps to create an environment template that is based on an existing one:

- Go to the Environment Templates List.
- Add a resource template.

  For detailed instructions, see Add an environment template. on page 367

- Click **Copy existing** to create a template based on an existing one.

  The list of existing templates opens.

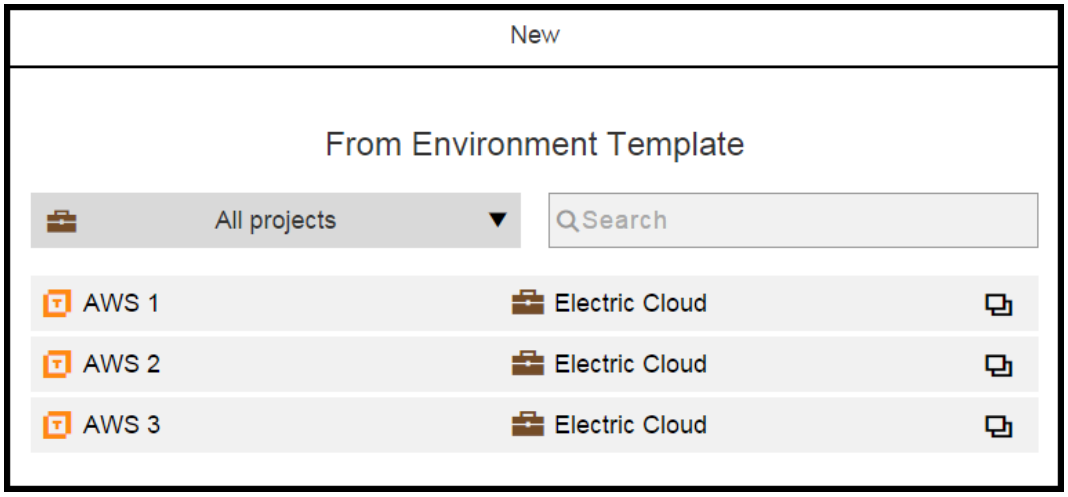
  It shows the list of available environment templates. The default is to show the environment templates for all projects.

  If you want to see only the environment templates for a specific project, click the down arrow in the **All projects** field to select one or more projects one of these ways:

  - Click on the name of one or more projects.
  - Enter the search criteria in the Search field. The projects that match the criteria appear in the list.

  You can also search for one or more environment templates by entering the search criteria in the **Search** field next to the **All projects** field. If there are no matches, a message appears stating that there are no environment templates in the selected projects.

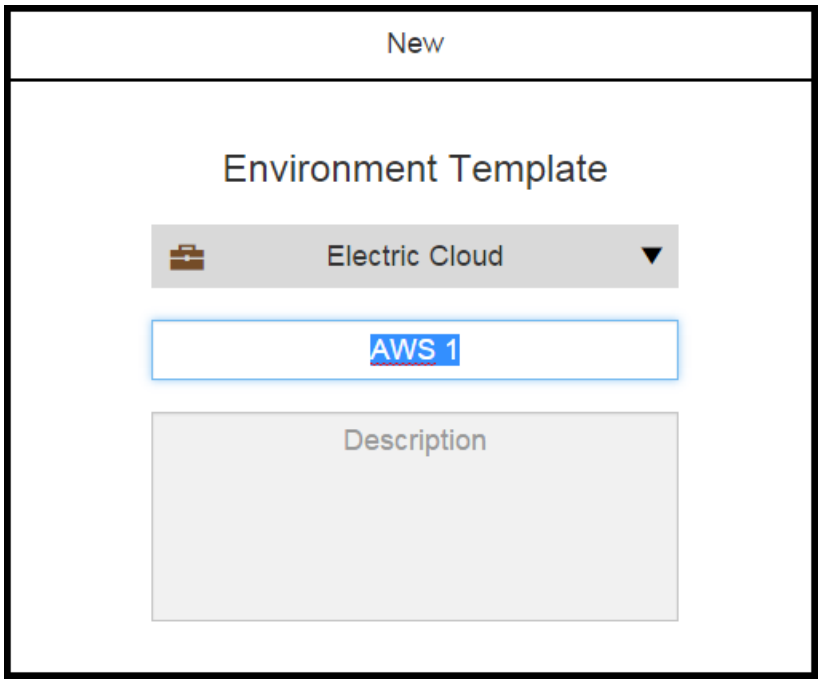
  ### Example:



- Select an environment template.

  The **New** dialog box opens.

  ### Example:

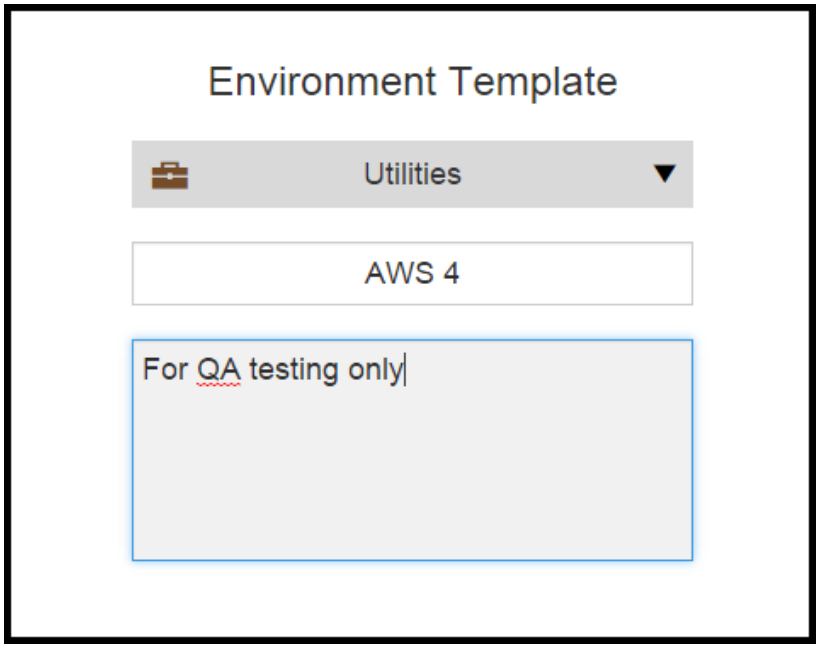


- Rename the environment template, and enter an optional description of it.

  You can also change the project to which the environment template belongs.

  Click **OK** to save these settings.

  Example:



The Environment Templates Visual Editor now shows that the same environment tiers as the original version. The breadcrumb in the upper left corner shows the object hierarchy.

You can now modify the environment template and its objects (resources and resource templates) for your deployments

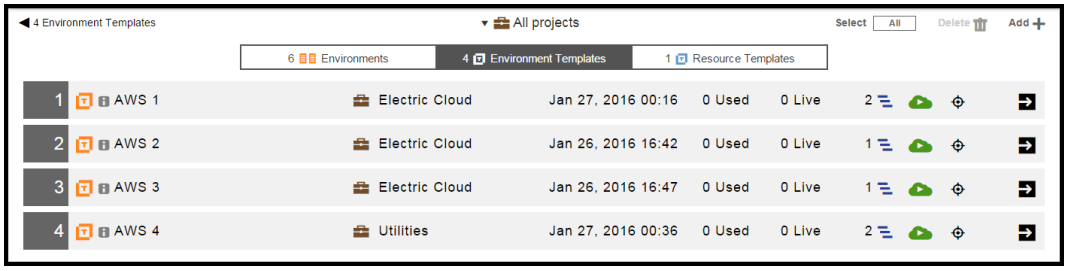
Example:

Viewing and Editing Environment Templates

Starting in the Environment Templates List, follow these steps to view and edit the environment template details:

- Choose an environment template and click the **View Details** button ( ).

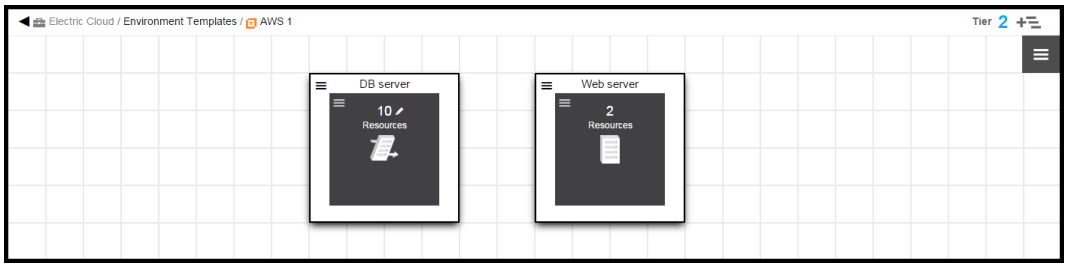
Example:



The Environment Templates for the selected template opens.

The breadcrumb in the upper left corner shows the object hierarchy.

Example:



- Click to **Menu** button ( ) to open the context menu for the environment template.
    - Click **Details** to view or edit the name and description of the environment template.
    - Click **Properties** to view or edit the environment template properties.
    - Click **Access Control** to go to the Access Control page in the automation platform.
    - Click **Track Changes** to open the Change History of the environment template.
    - Click **Delete** to delete this template.

- To view and edit an environment tier with dynamic resources, click the **Menu** button ( ) for the resources.

  The context menu opens with these options:

  - Click **Details** to view and edit the resource details for the environment tier.
  - Click **Remove** to remove the resources from the environment tier.

- To change the resource template used by the environment tier, click **Details** and select a different resource template.

  The list of available resource templates opens.

  It shows the list of available resource templates. The default is to show the resource templates for all projects.

  If you want to see only the resource templates for a specific project, click the down arrow in the **All projects** field to select one or more projects one of these ways:

  - Click on the name of one or more projects.
  - Enter the search criteria in the Search field. The projects that match the criteria appear in the list.

  You can also search for one or more resource templates by entering the search criteria in the **Search** field next to the **All projects** field. If there are no matches, a message appears stating that there are no resource templates in the selected projects.

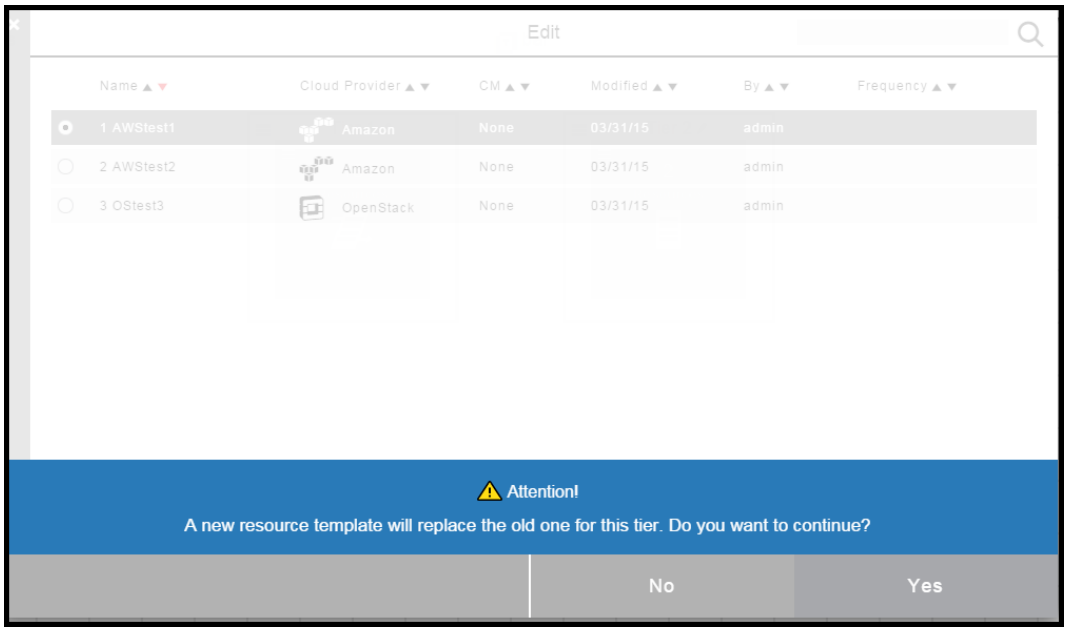
  The template being used is selected.

Example:

- Select a different environment template.

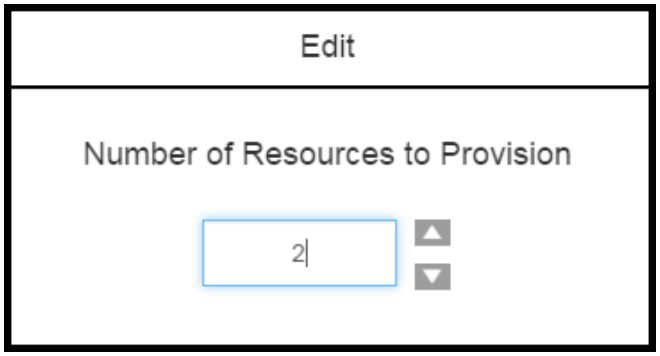
  A message appears. Select **Yes** if you want to continue.

  Example:



  The Edit dialog box number box opens and shows the number of provisioned dynamic resources.

  Example:



- To change the number of provisioned dynamic resources, enter a new number or use the up/down arrows.

- Click **OK**.

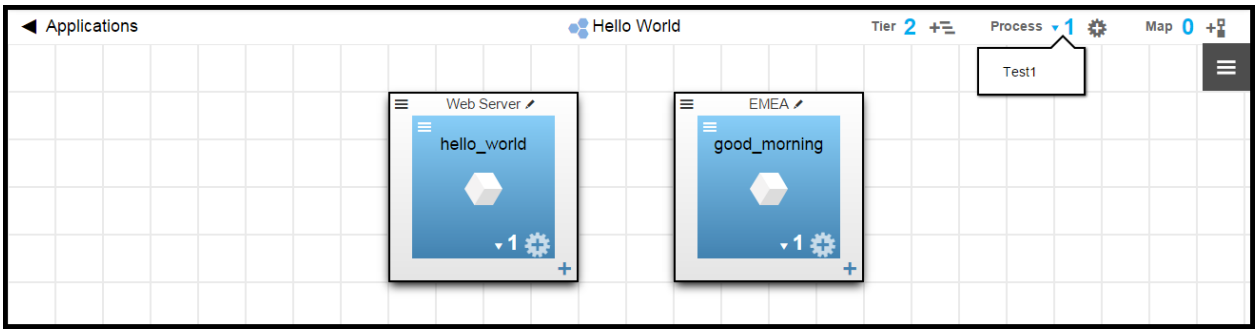
  The Environment Templates Visual Editor now shows the changes that you made.

Example:

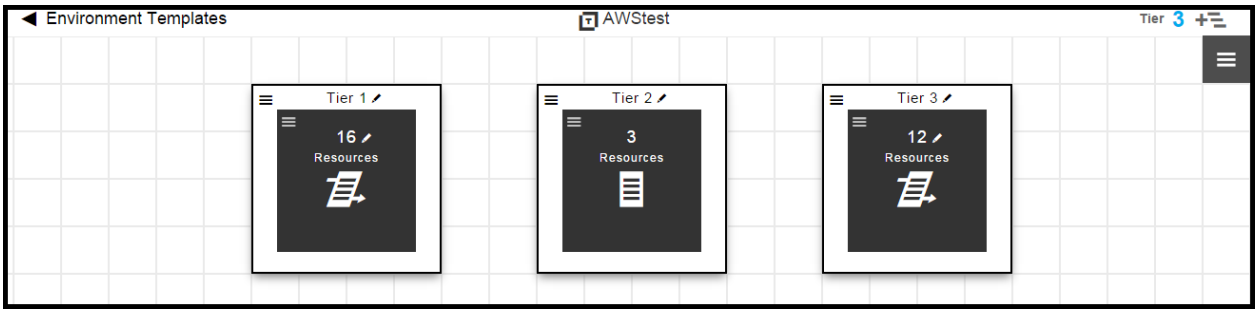
Deploying Applications With Provisioned Cloud Resources

About the example in this topic:

The example in this topic consists of an application called "Hello World" application with one application process called *Test1*, which has two steps.



Test1 will be deployed in a dynamic environment called *CloudEnv*. You model the CloudEnv environment using the *AWStest* environment template.



The AWStest environment template has three tiers:

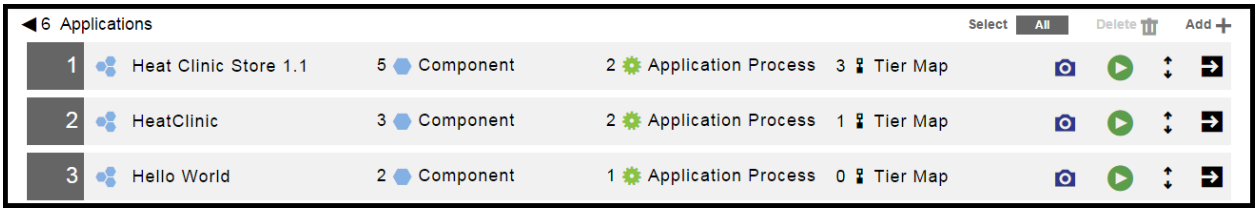
- Tier 1 and Tier 3 have cloud resources defined in resource templates. These resource can be provisioned when you deploy the application.
- Tier 2 has static resources, which cannot be provisioned.

1. Open the ElectricFlow Home page at https://<*ElectricFlow_server*>/flow/.
2. Enter your username and password and click **Login**.
3. Go to the Applications List using one of the following methods:
   - Starting from the Main menu, click the **Menu** button, and then select **Applications**.
   - Starting from the Home page, click **Applications**.

   The Applications List opens.

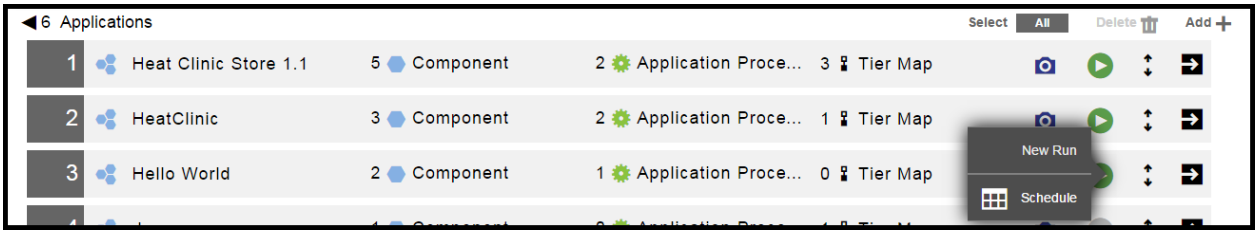
   ## Example:

   This example uses the "Hello World" application.



4. Choose an application, and click the **Run process** button for that application.

   ## Example:

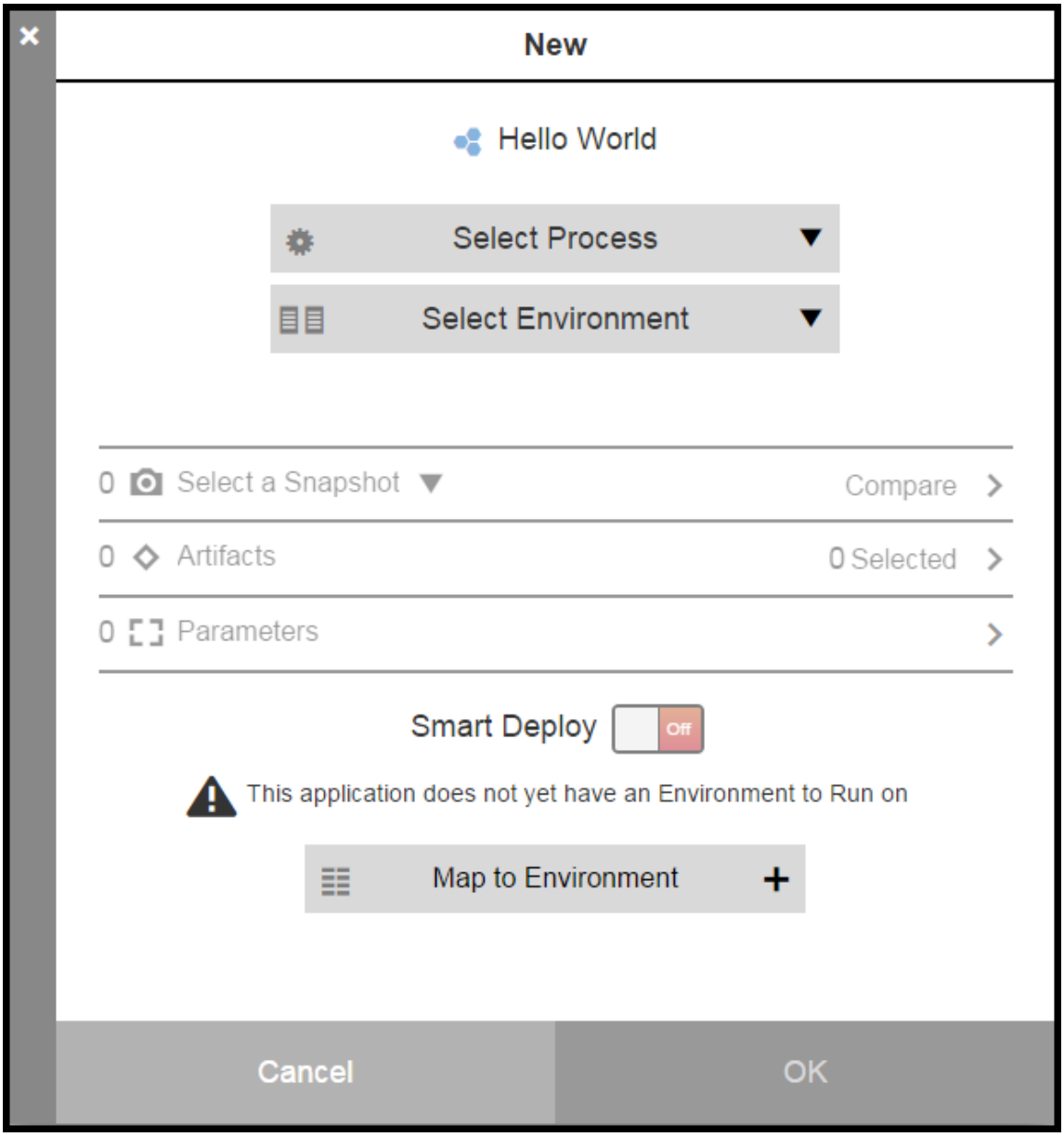


5. Click **New Run**.

   The **New** dialog box to deploy the application opens.

   See the messages in this dialog box for hints about what you need to do to deploy the application.

   Example:

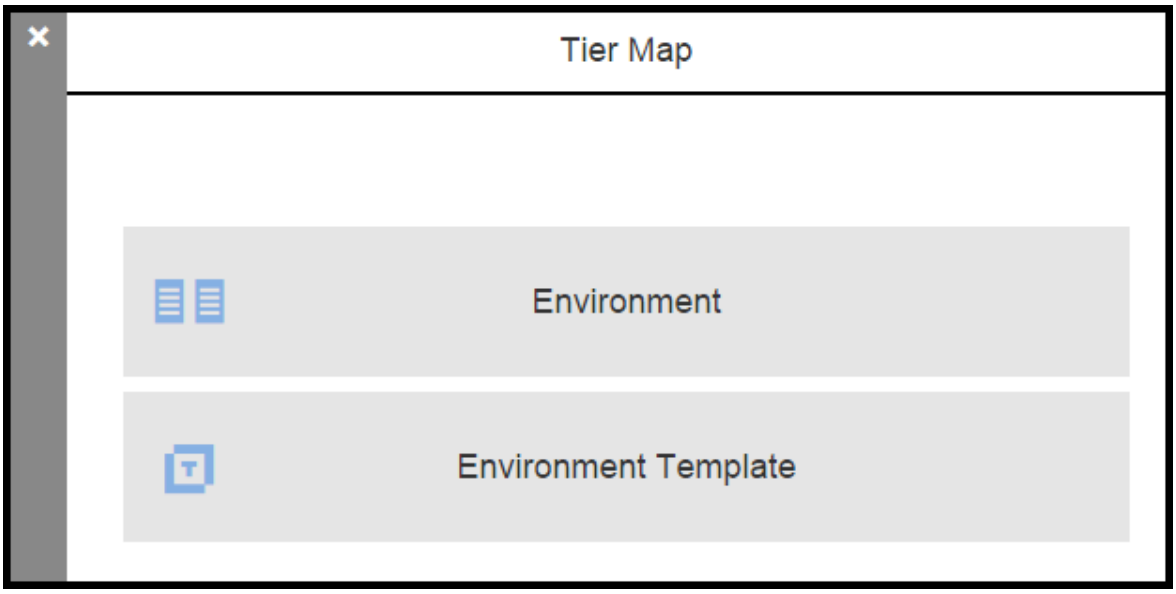


6. Select the application process to deploy.

7. Click **Map to Environment +** to create a tier map for the application.

   The **Tier Map** dialog box opens.

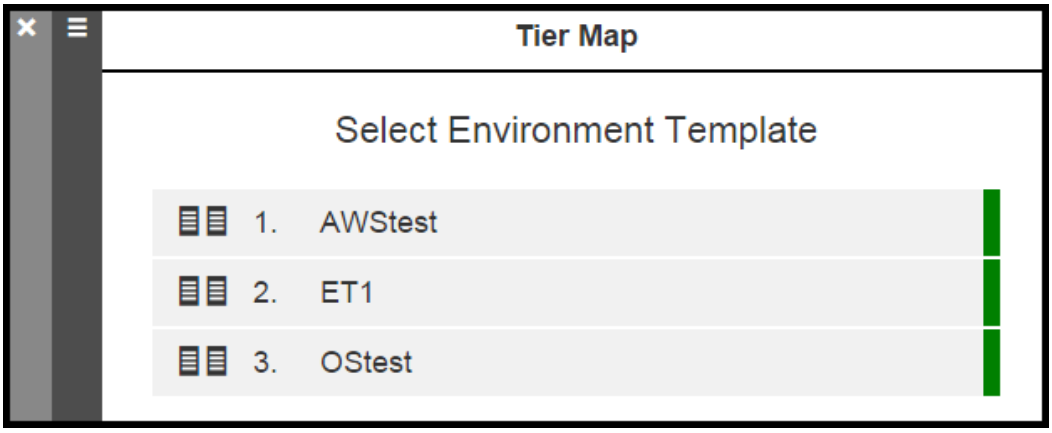
   Example:



8. Select **Environment Template**.

   The **Tier Map** dialog box to select an environment tier opens.

   Example:

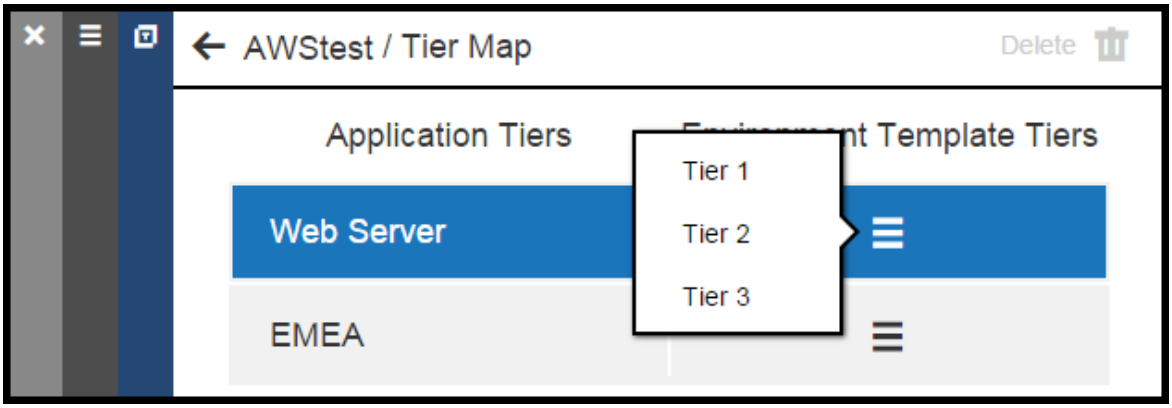


9. Select an environment template.

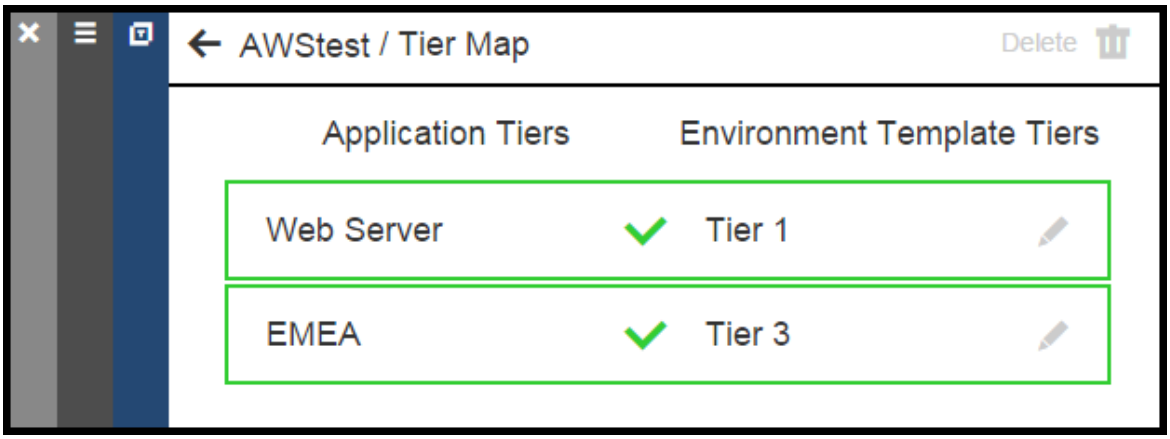
   The <Environment template>/Tier Map dialog box opens.

10. For each application tier, click the **Menu** button to select an environment tier to which the application tier is mapped.

Example:



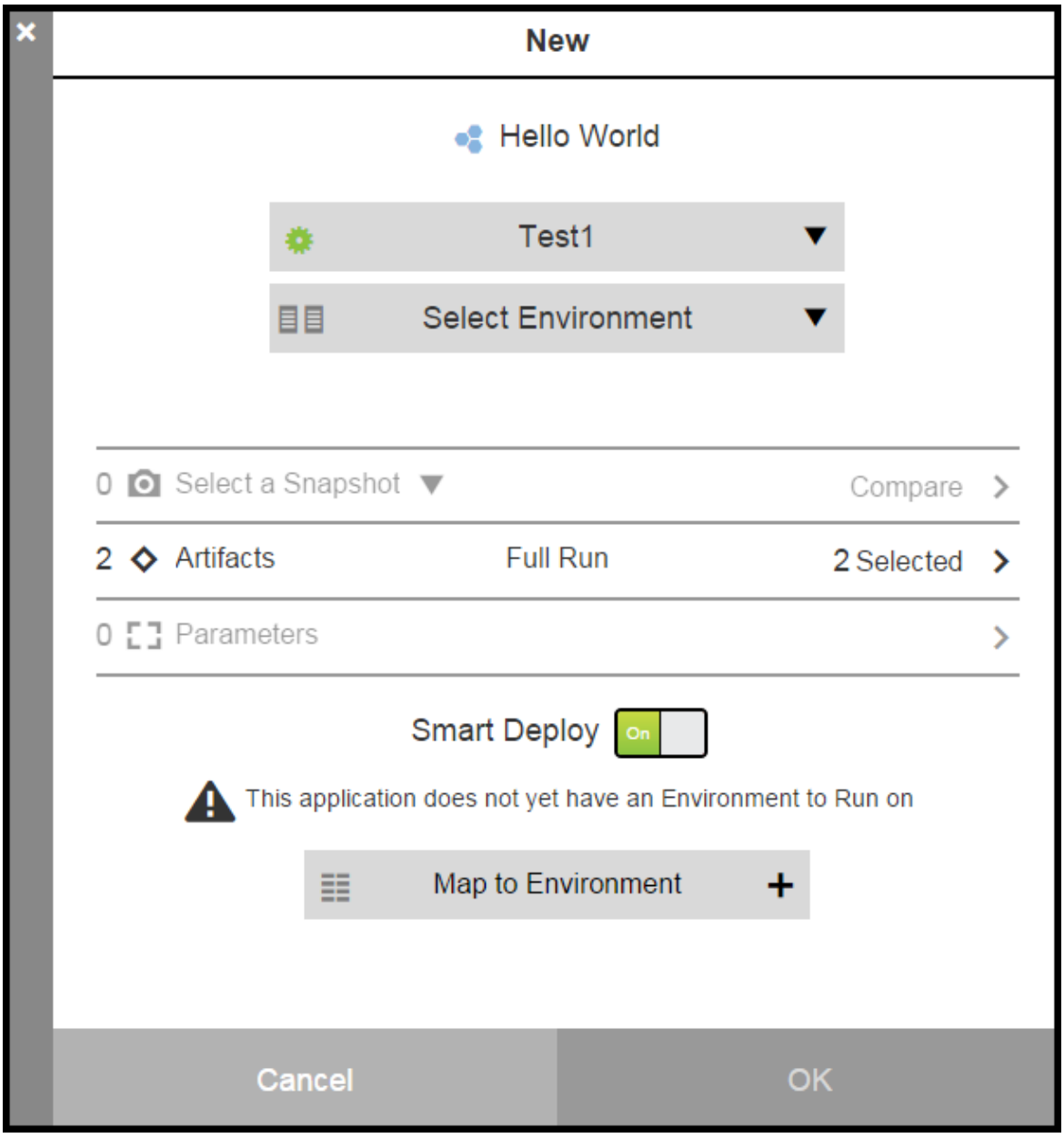
Example:



11. Click **OK**.

    The **New** dialog box to deploy the application re-opens.

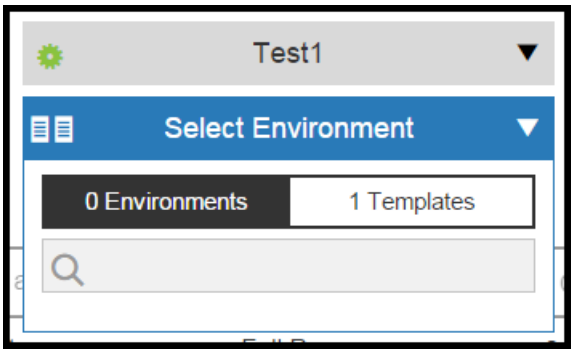
Example:



12. Click **Select Environment** to select an environment template.

    A list of available environments and environment template based on the tier map opens.

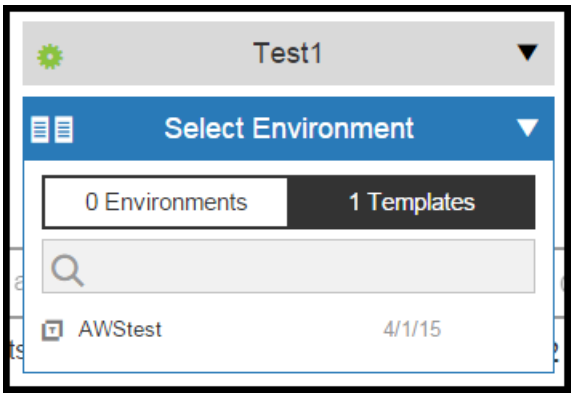
    Example:



13. Select **1 Templates**.

    A list of available environment templates appears.

    Example:

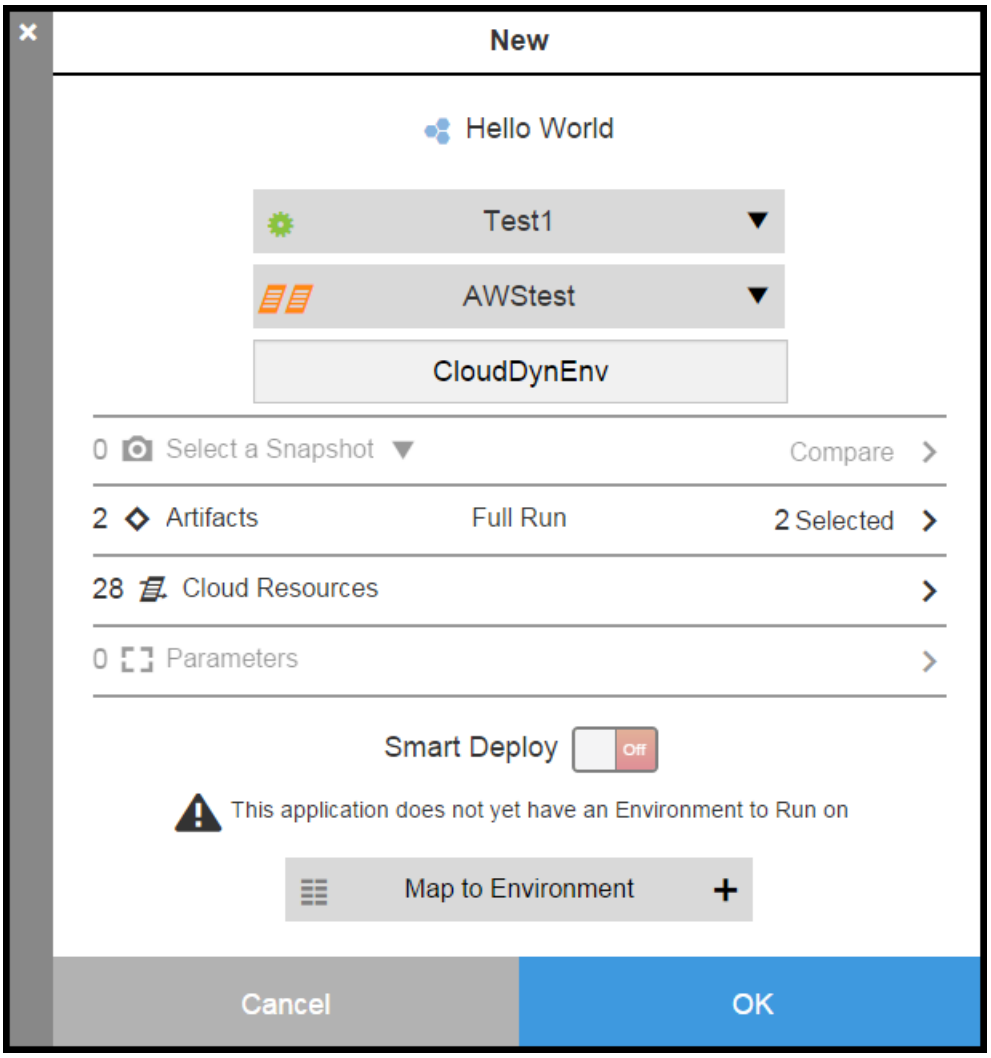


14. Click the environment template that you want to use.

15. Enter a name for the dynamic environment that will be created from the selected environment template.

    The **New** dialog box to deploy the application now shows the environment template name below the application process name. It also shows the number of cloud resources provisioned in the environment templates.

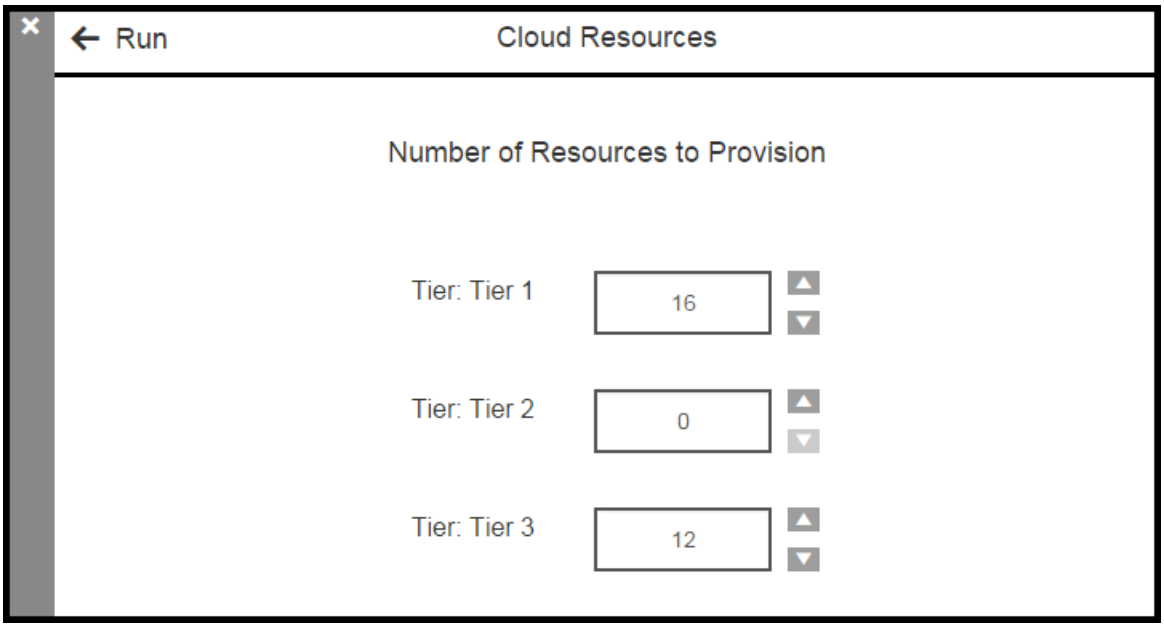
Example:



16. Click in the **Cloud Resources** row.

    The **Cloud Resources** dialog box opens.

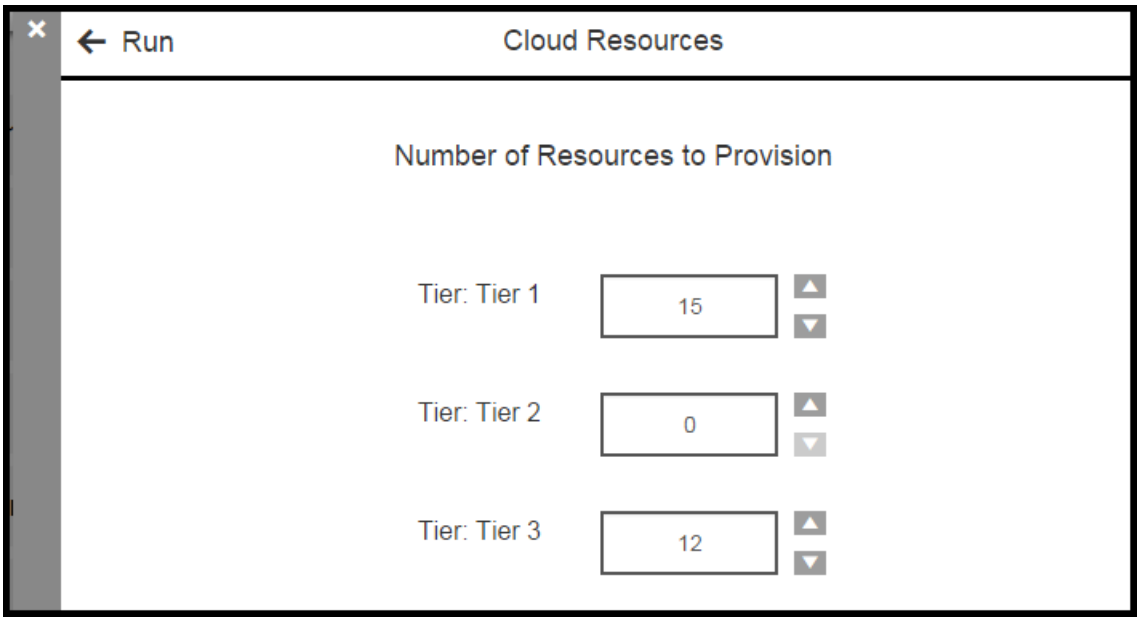
Example:



17. Change the number of cloud resources to provision

Example:

In this example, Tier 1 and Tier 3 have one or more resources to provision because they have cloud resources. You cannot provision resources in Tier 2 because it has only static resources.



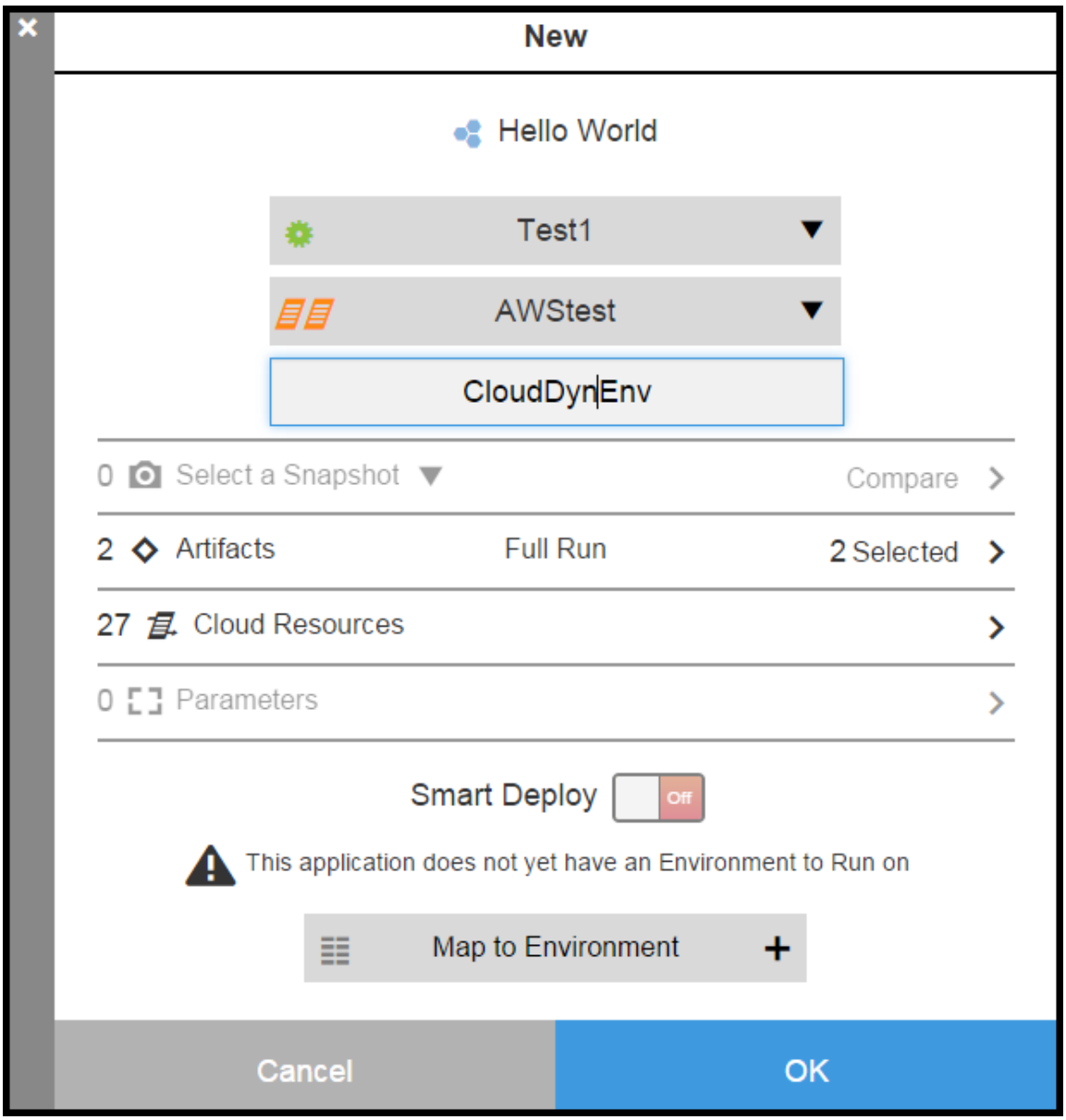
18. Click **OK**.

    The **New** dialog box now shows the new number of resources to provision.

    The message "This application does not yet have an Environment to Run on" still appears.

    When you click **OK**, ElectricFlow first attempts to create the dynamic environment. If this is successful, it deploys the application.

Example:



The Applications List opens.

19. To view the job details, click the **View Details** button to open to the Job Details page.

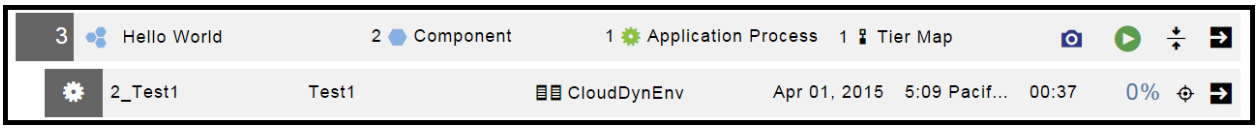
    ElectricFlow first runs the job to create the dynamic environment. If this job is successful, it deploys the application.

    Example:

    You can also see the status of jobs as they progress in the Applications List.

    Example:

    This example shows the Application List when the job starts.



    If you provision resources that are not available, such as 20 cloud resources in Tier 1 and 3 static resources in Tier 2, the job to create the dynamic environment fails and the application is not deployed.

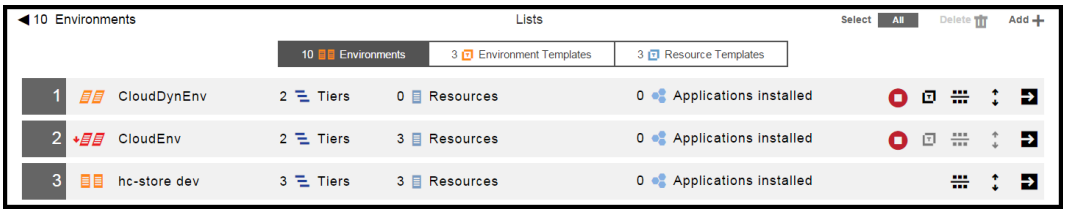
Retiring Dynamic Environments

After you have deployed your application in a dynamic environment, you can retire it and destroy (decommission) the resources, which makes them unavailable for any future deployments.

Starting in the Environments List:

1. Choose a dynamic environment.

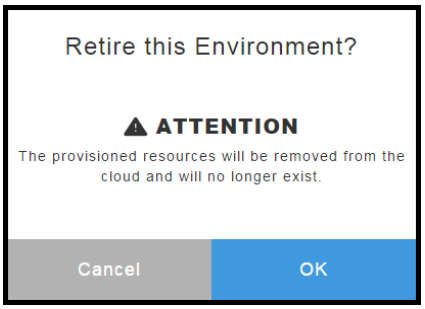
    Example:



2. Click the **Tear down** button for that environment.

   A message appears.

   Example:



3. Click **OK** to verify that you want to retire (decommission) the environment.

   The Job Details page opens.

   Example:

4. View the results in the Job Details page.

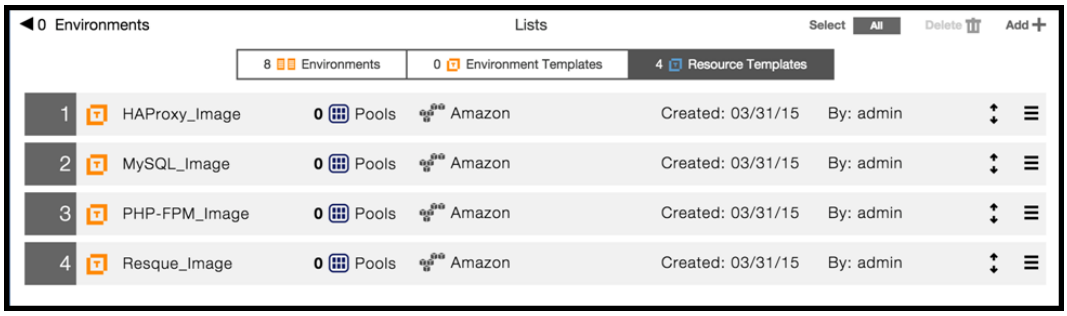
Dynamic Environment Example with Amazon and Chef

This example shows what you see in the ElectricFlow UI when your dynamic environment is configured with Amazon as the cloud provider and Chef for the configuration management.

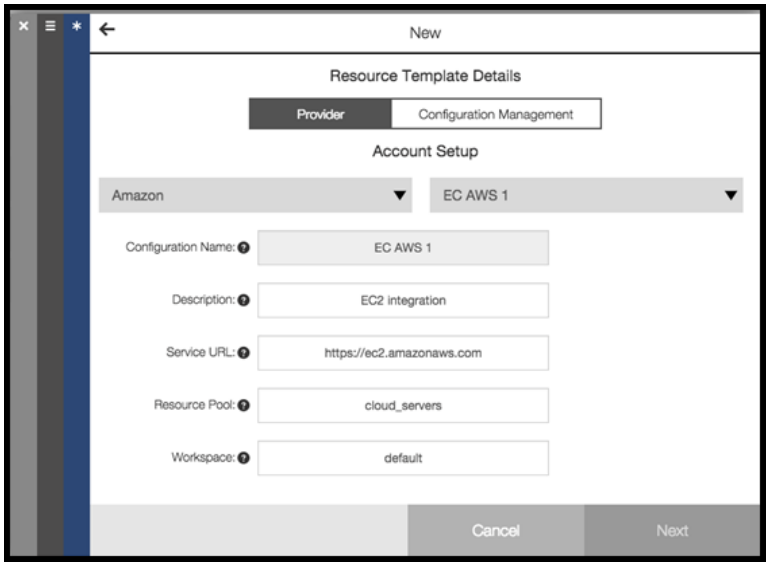
Resource Templates

In resource templates, you define cloud resources that will be provisioned for dynamic environments. Examples of cloud resources are a HAProxy server, MySQL server, PHP-FPM server, and Resque server.

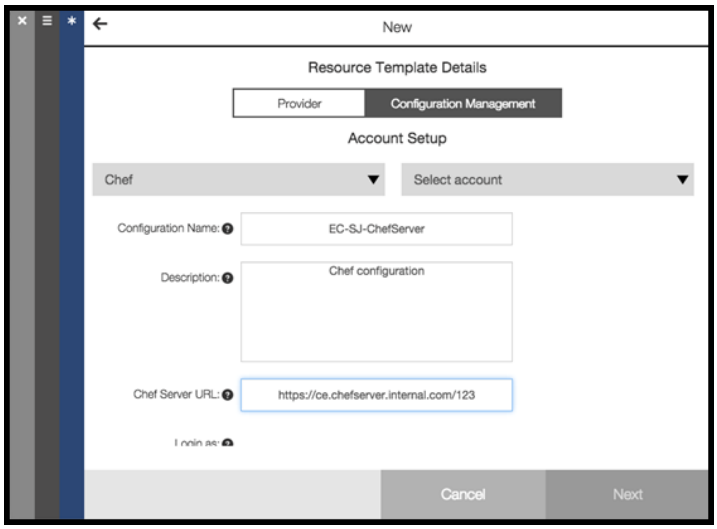
This is the Resource Template List.



These are the cloud provider settings for a resource template. It is defined by cloud provider account credentials and by an Amazon Machine Image (AMI).

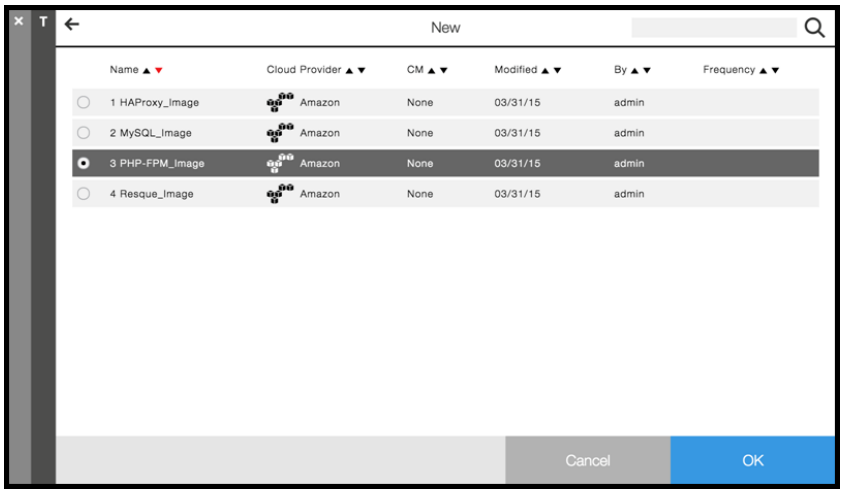


These are the configuration management settings for a resource template. It calls a Chef recipe to configure the cloud resources. You can configure the resources with other recipes or predefined configuration, such as a MySQL stack configuration or a HAProxy load balancer configuration.



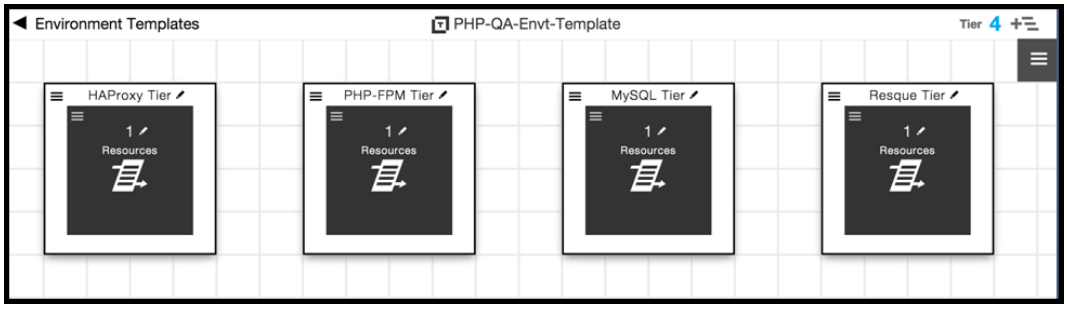
Environment Templates

When you create a new environment template, you can add one resource template to an environment tier. These are the resource templates from which you can select.



After you select a resource template and click **OK**, you enter the number of resources to provision in an environment tier.

This is an environment template with four tiers. Each tier is configured with a resource template, and resource template has a provisioned resource.

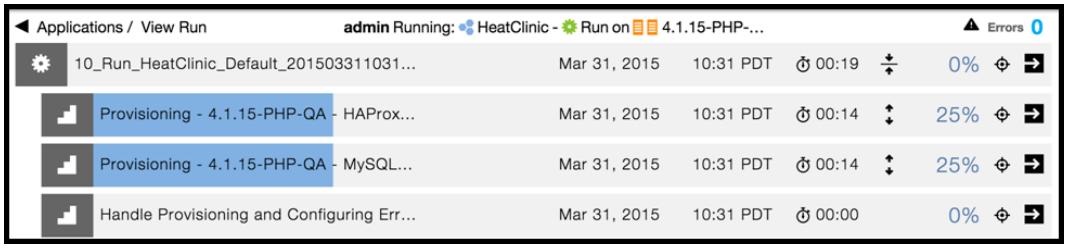


Deploying Applications to Dynamic Environments

After you select an application to deploy, you select the application process to run and the environment in which to deploy the application.

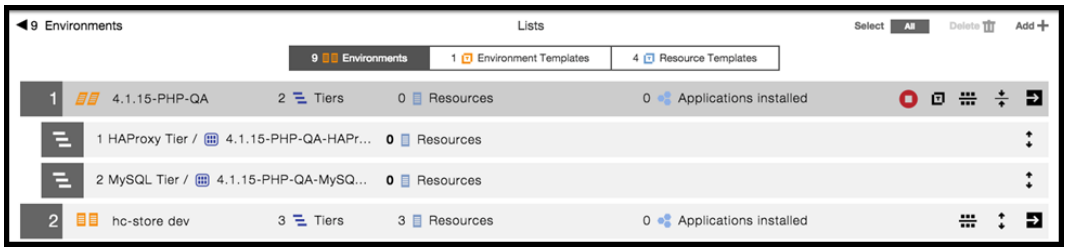
To deploy the application in a dynamic environment, you can select an environment template. A dynamic environment is created with the provisioned cloud resources. After the resources are provisioned and the dynamic environment is created, the application is deployed.

You can view the status of the provisioning process.



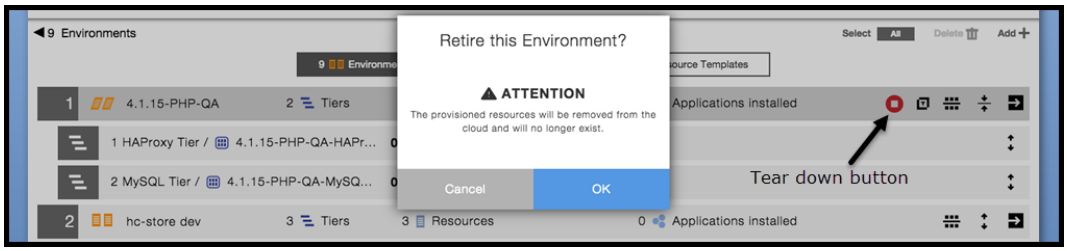
Retiring Dynamic Environments

After the provisioning process completes, the new dynamic environment appears in the Environments List.



You can retire (decommission) dynamic environments on an on-demand basis to prevent excessive use of cloud resources and reduce costs.

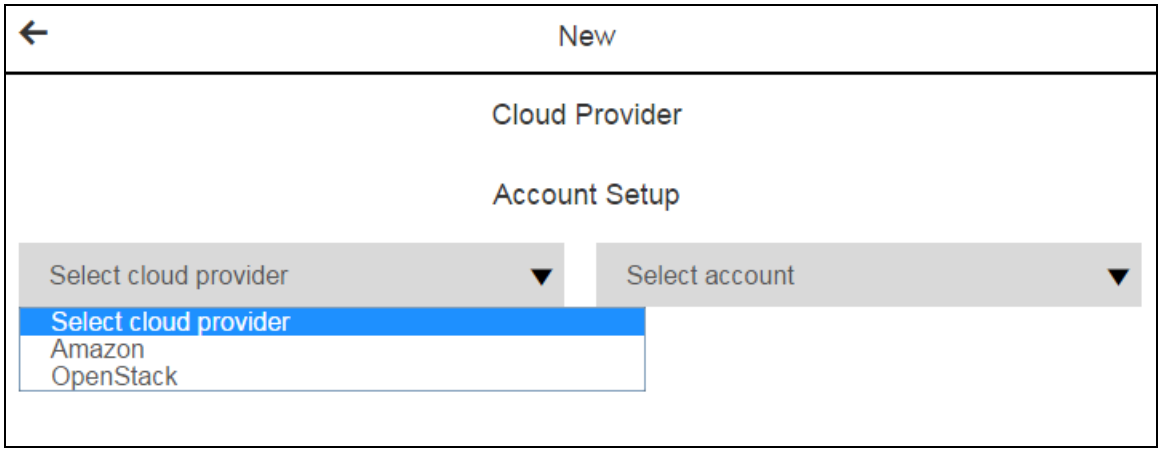
You click the **Tear down** button to retire the selected dynamic environment.



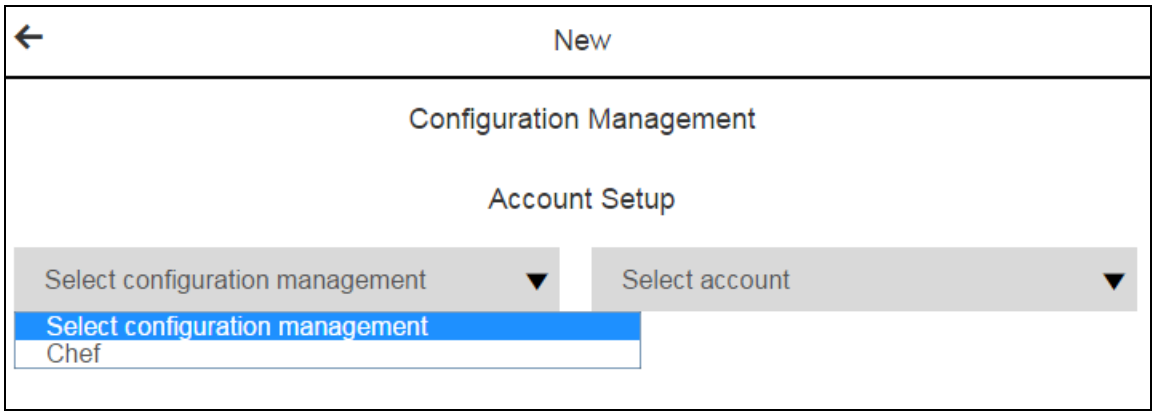
Developer Task: Creating Custom Plugins

ElectricFlow uses third-party plugins to provision cloud resources in resource templates for Dynamic Environments. Amazon EC2 (EC-EC2) and OpenStack (EC-OpenStack) are supported as out-of-the-box cloud provider plugins, and Chef (EC-Chef) is supported as an out-of-the-box configuration management plugin.

To view the supported cloud provider plugins, open the **New Cloud Provider** dialog box and click **Select cloud provider** to view the drop-down menu of cloud providers.



To view the supported configuration management plugins, click **Select configuration management** to view the drop-down menu of configuration management tools.



If you have an existing procedure that you would prefer to use instead of an out-of-the-box third-party plugin to provision cloud resources, you can create a custom plugin based on your deployment scenario.

The process to create custom cloud provider and configuration management plugins uses metadata to loosely couple them to the application you want to deploy. The custom plugins define specific known properties that are automatically recognized by Dynamic Environments.

To create your own plugin for other cloud platforms, see https://electric-cloud.github.io/index.html to get started. This website describes how to create open source plugins and has this information:

- For an introduction to the process of building and developing open source plugins, go to the Quick Start.
- To build and deploy your plugin, see Build and Deploy.
- For more information, see More information.
- See Creating Custom Cloud Provider Plugins on page 402 for the details to create custom cloud provider plugins.
- See Creating Custom Configuration Management Plugins on page 407 for the details to create custom configuration management plugins.

Creating Custom Cloud Provider Plugins

This section applies to developers who want to create custom cloud provider plugins for Dynamic Environments in ElectricFlow.

How to Create a Custom Cloud Provider Plugin

This procedure assumes that you already have a plugin and want to use it with ElectricFlow. It describes how to define properties in your plugin that ElectricFlow can read.

1. To convert your procedure to a plugin, define the name of the plugin and the ElectricFlow project to which the plugin belongs.

   Later in this procedure, you will use these properties to create pre- and post-hooks to the plugin in ElectricFlow.

2. Define the following properties in your plugin under a top-level plugin property called `ec_cloudprovisioning_plugin`, which the system will read to Dynamic Environments.

Property Name	Description
`ec_cloudprovisioning_plugin/`	Top-level plugin property directory
Properties defined by the custom plugin under `ec_cloudprovisioning_plugin`	
`displayName`	Name of the plugin that appears in the **New Cloud Provider** dialog box. Example: **Amazon** for the EC-EC2 plugin
`hasConfiguration`	<Boolean flag —`0|1|true|false`> • Set this property to `0` if the plugin does not have any configuration procedures (`CreateConfiguration` and `DeleteConfiguration`). • Set this property to `1` if the plugin has configuration procedures.
`configurationLocation`	Name of the property sheet used by the plugin to store the saved configurations. This value is relative to the plugin's top-level properties. For example, if this value is set as *ec2_cfgs*, the plugins configurations are read from */plugins/<PLUGIN_KEY>/project/ec2_cfgs*.
`operations/`	Property sheet for the specific operations required by the Dynamic Environment system.
Properties under operations/ – These operations are mapped to the plugin operations through the following child properties.	
`createConfiguration/`	Properties for the procedure that creates the plugin configuration. It is usually called `createConfiguration`.
`deleteConfiguration/`	Properties for the procedure that deletes the plugin configuration. It is usually called `deleteConfiguration`.
`provision/`	Properties for the procedure that provisions virtual instances.

Property Name	Description
retireResource/	Properties for the procedure that tears down (decommissions) a previously provisioned virtual instance backing the specified resource.
retireResourcePool/	Properties for the procedure that tears down (decommissions) all previously provisioned virtual instances for the specified resource pool.
Properties for the previously listed operations defined in the `operations` property sheet	
procedureName	Name of the procedure name in the plugin to which the operation is mapped.
ui_formRefs/	Property sheet that references the `ui_forms` properties defined under the plugin's top-level properties.
parameterRefs/	Property sheet for the input parameters to the procedure that can be used by the Dynamic Environment system. The parameter list is operation-specific.

See Example: Property Structure for a Cloud Provider Plugin on page 405 for an example of the properties for the EC-EC2 (Amazon) plugin.

3. Save the plugin file in the appropriate location on your ElectricFlow server.
4. To import the plugin file to the automation platform:
    1. In the automation platform, go to **Administration** > **Plugins** to open the **Plugin Manager** page.
    2. Click the **Install from File/URL** tab.
    3. In the **File Install** field, click **Choose file** to select the plugin file.
    4. Click **Upload** to install it.

       The plugin file appears in the **Currently Installed** tab.
    5. Find your plugin and click **Promote** in the Actions column to make it available for use by ElectricFlow.

5. In ElectricFlow, to create a resource template:
    1. See Creating Resource Templates on page 348 for the details.
    2. In the **Select cloud provider** field, select your plugin as the cloud provider.
    3. Enter the plugin settings in the form.
    4. (Optional) Set the configuration management tool.

       See Creating Resource Templates on page 348 for the details.
6. In the ectool API, enter the following commands to set pre- and post-hooks linking the plugin to an application that you later deploy in a dynamic environment.
    1. To create a pre-hook, enter

       ```
       ectool createHook postConfigurationHook --hookType POST_CONFIGURATION --
       procedureName <hookProcedureName> --procedureProjectName
       <hookProjectName> --resourceTemplateName <resourceTemplateName> --
       projectName <resourceTemplateProjectName>
       ```

       where `hookProcedureName` is the plugin procedure name, `hookProjectName` is the name of the project to which the plugin procedure belongs, `resourceTemplateName` is the name of the resource template you created, and `resourceTemplateProjectName` is the name of the project to which the resource template belongs.
    2. To create a post-hook, enter

       ```
       ectool createHook preConfigurationHook --hookType PRE_CONFIGURATION --
       procedureName <hookProcedureName> --procedureProjectName
       <hookProjectName> --resourceTemplateName <resourceTemplateName> --
       projectName <resourceTemplateProjectName>
       ```

       where `hookProcedureName` is the plugin procedure name, `hookProjectName` is the name of the project to which the plugin procedure belongs, `resourceTemplateName` is the name of the resource template you created, and `resourceTemplateProjectName` is the name of the project to which the resource template belongs.
7. Create an environment template with the resource template that you created.

Example: Property Structure for a Cloud Provider Plugin

This is the EC-EC2 (Amazon) plugin property structure.

Property Name	Property Values
`ec_cloudprovisioning_plugin/`	
`displayName`	Amazon
`hasConfiguration`	1
`configurationLocation`	ec2_cfgs

Property Name	Property Values
operations/	createConfiguration/ deleteConfiguration/ provision/ retireResource/ retireResourcePool/
ec_cloudprovisioning_plugin/createConfiguration/	
procedureName	CreateConfiguration
ui_formRefs/	
ui_formRefs/parameterForm	ui_forms/EC2CreateConfigForm
parameterRefs/	
parameterRefs/ configuration	config
ec_cloudprovisioning_plugin/deleteConfiguration/	
procedureName	DeleteConfiguration
ui_formRefs/	–
parameterRefs/	configuration called config
ec_cloudprovisioning_plugin/provision/	
procedureName	API_RunInstances
ui_formRefs/	parameterForm called ec_parameterForm
parameterRefs/	configuration called config resourcePool called res_poolName count called count
ec_cloudprovisioning_plugin/retireResource/	
procedureName	API_TearDownResource
ui_formRefs/	–
parameterRefs/	resourcePool called res_poolName

Property Name	Property Values
ec_cloudprovisioning_plugin/retireResourcePool/	
procedureName	API_TearDownResource
ui_formRefs/	–
parameterRefs/	resourcePool called res_poolName

Creating Custom Configuration Management Plugins

This section applies to developers who want to create custom configuration management plugins for Dynamic Environments in ElectricFlow.

How to Create a Custom Configuration Management Plugin

This procedure assumes that you already have a plugin and want to use it with ElectricFlow. It describes how to define properties in your plugin that ElectricFlow can read.

1. To convert your procedure to a plugin, define the name of the plugin and the ElectricFlow project to which the plugin belongs.

   Later in this procedure, you will use these properties to create pre- and post-hooks to the plugin in ElectricFlow.

2. Define the following properties in your plugin under a top-level plugin property called `ec_configurationmanagement_plugin`, which the Dynamic Environment system can access.

Name	Description
`ec_configurationmanagement_plugin/`	Top-level plugin property directory
Properties defined by the custom plugin under `ec_configurationmanagement_plugin`	
`displayName`	Name of the plugin that appears in the Dynamic Environment UI Example: **Chef** for the EC-Chef plugin
`hasConfiguration`	<Boolean flag —`0|1|true|false`> • When this property is set to `0` or `false`, the plugin does not have any configuration procedures (`CreateConfiguration` and `DeleteConfiguration`). • When this property is set to `1` or `true`, the plugin has configuration procedures.
`configurationLocation`	Name of the property sheet used by the plugin to store the saved configurations. This value is relative to the plugin's top-level properties. If this value is set as *chef_cfgs*, the configurations are in */plugins/<PLUGIN_KEY>/project/chef_cfgs*.
`operations/`	Property sheet for the specific operations required by the Dynamic Environment system.
Properties under `operations/` – These operations are mapped to the plugin operations through the following child properties.	
`createConfiguration/`	Properties for the procedure that creates the plugin configuration. It is usually called `createConfiguration`.
`deleteConfiguration/`	Properties for the procedure that deletes the plugin configuration. It is usually called `deleteConfiguration`.

Name	Description
converge/	Properties for the procedure that converges the virtual instances to the defined configuration (including policies and roles).
teardown/	Properties for the procedure that deletes the configuration details on the specified dynamic resource or resource pool. For Chef, properties for the procedure that deletes the Chef API client and Chef node on the specified dynamic resource or resource pool.
Properties for the previously listed operations defined in the `operations` property sheet	Properties for the previously listed operations defined in the `operations` property sheet
procedureName	
ui_formRefs/	Property sheet that references the `ui_forms` properties defined under the plugin's top-level properties.
parameterRefs/	Property sheet for the input parameters to the procedure that can be used by the Dynamic Environment system. The parameter list is operation-specific.

See Example: Property Structure for a Configuration Management Plugin on page 410 for an example of the properties for the EC-Chef plugin.

3. Save the plugin file in the appropriate location on your ElectricFlow server.
4. To import the plugin file to the automation platform:
    1. In the automation platform, go to **Administration** > **Plugins** to open the **Plugin Manager** page.
    2. Click the **Install from File/URL** tab.
    3. In the **File Install** field, click **Choose file** to select the plugin file.
    4. Click **Upload** to install it.

       The plugin file appears in the **Currently Installed** tab.
    5. Find your plugin and click **Promote** in the Actions column to make it available for use by ElectricFlow.

5. In ElectricFlow, to create a resource template:
    1. See Creating Resource Templates on page 348 for the details.
    2. In the **Select cloud provider** field, select a cloud provider.
    3. Enter the account settings for the selected cloud provider.
    4. Enter the **provision parameters** on the next page.
    5. In the **Select configuration management** field, select your plugin as the configuration management tool.

       See Creating Resource Templates on page 348 for the details.
6. In the ectool API, enter the following commands to set pre- and post-hooks linking the plugin to an application that you later deploy in a dynamic environment.
    1. To create a pre-hook, enter

       ```
       ectool createHook postConfigurationHook --hookType POST_CONFIGURATION --
       procedureName <hookProcedureName> --procedureProjectName
       <hookProjectName> --resourceTemplateName <resourceTemplateName> --
       projectName <resourceTemplateProjectName>
       ```

       where `hookProcedureName` is the plugin procedure name, `hookProjectName` is the name of the project to which the plugin procedure belongs, `resourceTemplateName` is the name of the resource template you created, and `resourceTemplateProjectName` is the name of the project to which the resource template belongs.
    2. To create a post-hook, enter

       ```
       ectool createHook preConfigurationHook --hookType PRE_CONFIGURATION --
       procedureName <hookProcedureName> --procedureProjectName
       <hookProjectName> --resourceTemplateName <resourceTemplateName> --
       projectName <resourceTemplateProjectName>
       ```

       where `hookProcedureName` is the plugin procedure name, `hookProjectName` is the name of the project to which the plugin procedure belongs, `resourceTemplateName` is the name of the resource template you created, and `resourceTemplateProjectName` is the name of the project to which the resource template belongs.
7. Create an environment template with the resource template you created.

Example: Property Structure for a Configuration Management Plugin

This is the EC-Chef plugin property structure.

Name	Values
ec_configurationmanagement_plugin/	
displayName	Chef
hasConfiguration	1
configurationLocation	chef_cfgs

Name	Values
operations/	createConfiguration/ deleteConfiguration/ converge/ teardown/
ec_configurationmanagement_plugin/createConfiguration/	
procedureName	CreateConfiguration
ui_formRefs/	parameterForm called forms/CreateConfigForm
parameterRefs/	configuration called config
ec_configurationmanagement_plugin/deleteConfiguration/	
procedureName	DeleteConfiguration
ui_formRefs/	–
parameterRefs/	configuration called config
ec_configurationmanagement_plugin/converge/	
procedureName	_RegisterAndConvergeNode
ui_formRefs/	parameterForm called ec_parameterForm
parameterRefs/	configuration called config
ec_cloudprovisioning_plugin/teardown/	
procedureName	_DeleteNode
ui_formRefs/	–
parameterRefs/	resourceName called resource_name

Chapter 3: Pipelines

Pipelines are a way to orchestrate the flow of the software release process. You can choose what part of the process in the pipeline to automate, based on your needs. Some examples are:

- From code check-in to production: After developers check in code, it is run through a pipeline that includes build, test, pre-production, and production stages, running various automation tasks in each of the stages like deploying the applications or microservices, running test automation suites, and so on, ending when the software is delivered to the production server.
- From system test to production: After the software passes the system tests, it is run through a pipeline to integrate the code from multiple teams, run user acceptance tests, and deploy the software to for beta testing and then all the way to the production.

Starting in ElectricFlow 6.2, multiple project support has been extended to ElectricFlow objects in application deployments, pipelines, and releases. These objects as well as the objects belonging to them can be in any project within ElectricFlow. (In releases earlier than ElectricFlow 6.2, these objects were limited to the Default project.)

All objects in ElectricFlow have these capabilities for better management and maintenance of end-to-end software releases:

- Projects are top-level security containers that provide access control at the project level. All objects in a project inherit the access control settings from the project to which they belong.
- Projects are also ways to provide logical groupings of objects. If your deployment is large and has many production environments, you can organize production environments by project, with each project having its own access control settings. This makes it easier to manage and maintain multiple environments during the software delivery process.

In this section, you will learn:

- About pipeline models–See Pipeline Concepts on page 413.
- How to author and run a pipeline–See Example: Authoring and Running Pipelines on page 454.
- About pipeline stage summaries–See Pipeline Stage Summary on page 603.

Important: When you export a project while a pipeline is in progress, only the full export includes flow runtimes that have been completed. If you want to include in-progress pipeline runs in the path-to-production view and the visual indicators showing their percentage completed in the Release Dashboard, set the `excludeJobs` argument to `0` or `false` in the `export` command. When the XML file is imported, the in-progress pipeline runs in the imported project are displayed in the path-to-production view and the Release Dashboard. The jobs might be incomplete once the XML is imported.

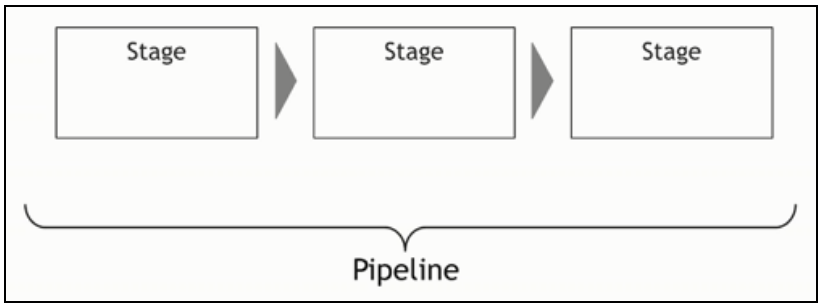
Pipeline Concepts

A pipeline lets you model the complete software delivery process. You can create multiple milestones called *stages* in a pipeline.

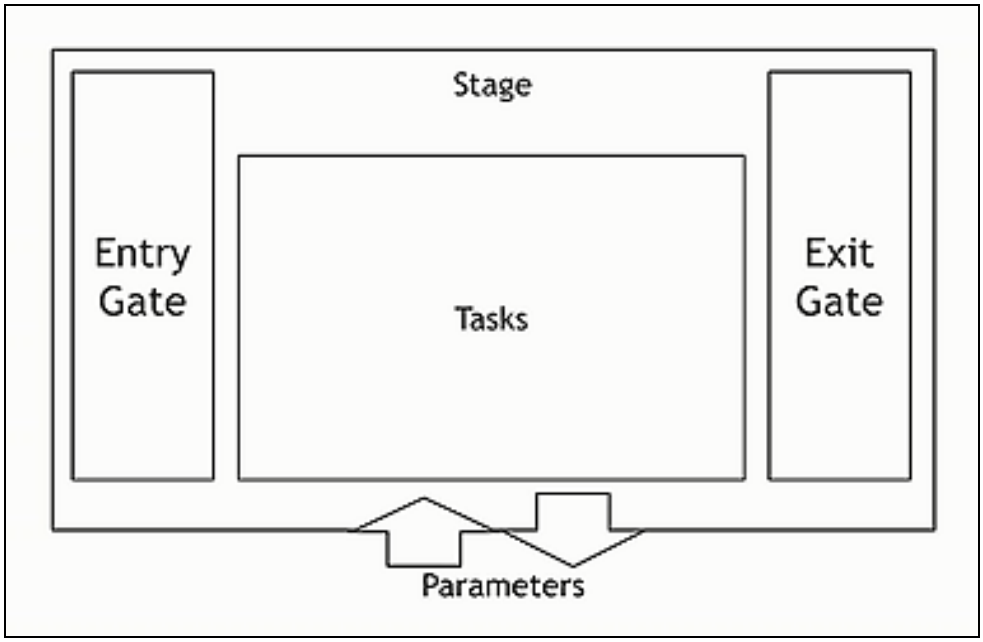
- Pipeline Stages and Gates on page 414
- Pipeline Tasks on page 419

Pipeline Stages and Gates

Examples of stages are development (Dev), QA, user acceptance test (UAT), preproduction, and production. These stages typically map closely to the environments where applications or microservices will be deployed.



Each stage consists of an entry gate, an exit gate, and the automation tasks to complete the stage.



- An entry gate controls the approvals required before a pipeline can enter a stage. The entry gate ensures all the checks and balances are met before tasks in that stage can begin.

- The exit gate allows you to control when a pipeline can exit a stage and what approvals are needed.
- The tasks allow you to execute the automation logic. This is where you specify what to do.

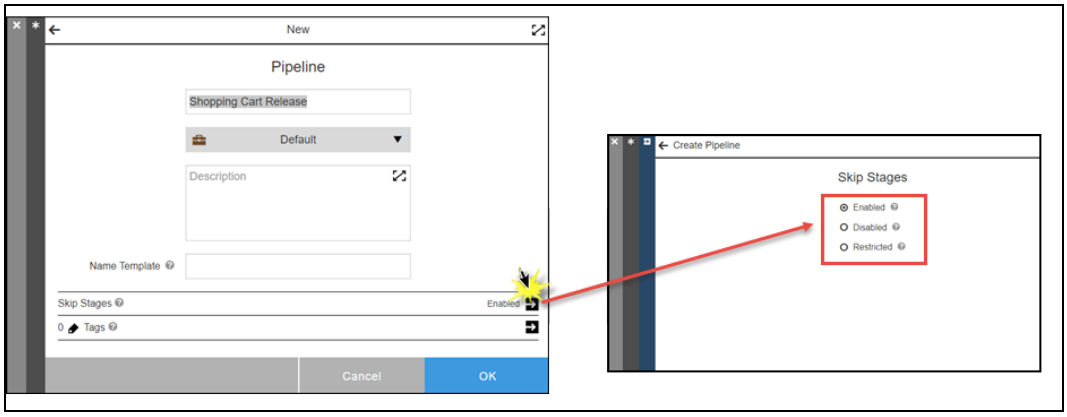
  For example, if your stage is a test stage, it contains the tasks that must be executed to ensure that the test automation for your software delivery process is successfully completed.
- Parameters can be created for various objects and automation tasks in the stage. This makes the pipeline reusable, allowing orchestration of the inputs and outputs for the stage and of the properties passed between stages.
- **Run if**conditions, **Wait until**conditions and wait dependencies can be set on pipeline objects. See Pipeline Objects and Conditions on page 576 for more information.
- Wait conditions can be set for triggered pipelines, triggered releases, and specific pipelines and pipeline objects. See Pipeline Hierarchies on page 429 for more information.

Stage Skip Control

By default, there is no restriction on which users are allowed to skip stages in a pipeline; anyone can select stages they want to skip. This level of flexibility may allow users to inadvertently circumvent security requirements. The admin or any user with modify privileges on pipeline objects can specify users and user groups authorized to skip stages or to disable stage skipping all together.

Configuring Stage Skipping

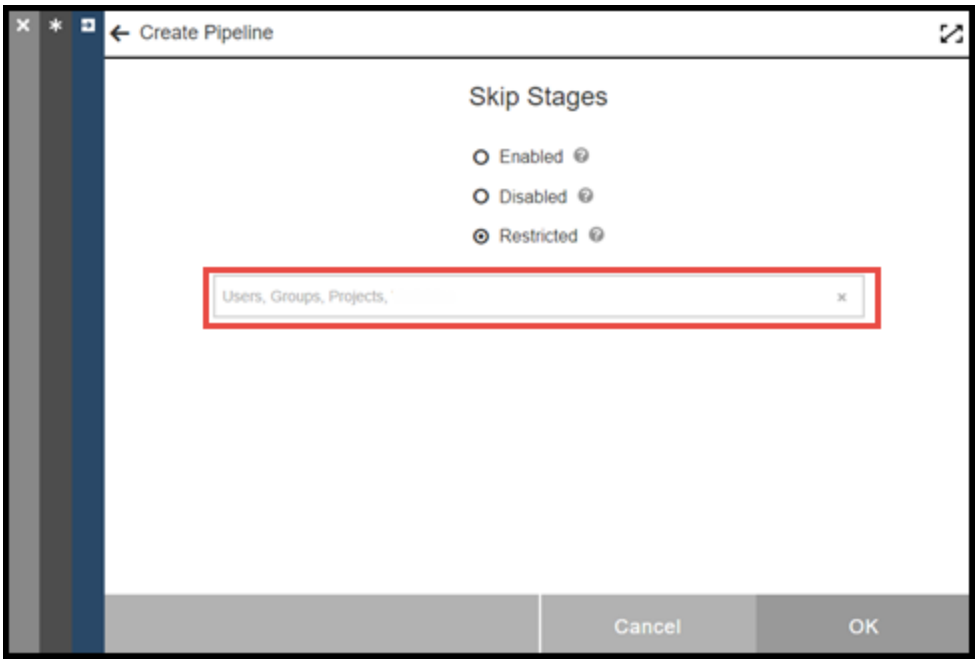
At pipeline create time, set level of stage skip control in the New Pipeline dialog. At runtime, stage selection is disabled if the current user is not permitted to skip stages or if skipping stages is disabled.



Three levels of stage skip control include:

- Enabled—All users and groups can select stages to skip.
- Disabled—No user or group is permitted to select stages to run.

- Restricted—Only specific users and groups are permitted to pick stages to run. When this level is selected, you enter the list of permitted users and groups in the space provided as show below.



Running a Pipeline

When configuring a pipeline run the user selects which stages to disable. If stage skipping is configured, stage selection is disabled for users restricted from stage skipping, which is clearly denoted in the **Configure Run** dialog.

The image below shows a pipeline where the user is not allowed to skip stages.

Restarting a Pipeline

If a user, who is not permitted to skip stages, restarts a pipeline run, they are not able to skip any stages which are after the stage being restarted. For the examples below consider a pipeline with five stages, skip stage mode is RESTRICTED for all users except User1.

Example 1:

- User1 starts the pipeline and selects stages 1, 2, and 3 to run.
- The pipeline runs and the stops at stage 3 due to an error.
- User2 attempts to restart the pipeline. Since User1 configured the pipeline to skip stage 4 and 5, User2 cannot restart the pipeline because he does not have stage skip privileges.

Example 2:

- User1 starts pipeline and selects stages 2,3,4, and 5 to run.
- The pipeline runs and the stops at stage 2 due to an error.
- User2 is able to restart the pipeline because no stages are being skipped after stage 2.

Authentication Context Control

All stages in a pipeline run within the authentication context of the user who starts the pipeline and the owning project authentication context. The user who initiates the pipeline usually has access to the lower environments at the start of the pipeline, but as the pipeline progresses to more controlled environments, access is more limited, and the originator likely will not have the required access. In this case, the project principal is given the appropriate access so the pipeline can continue, within the context of the originator. This may not be appropriate for the project and can open security vulnerabilities.

As an illustration of this vulnerability, consider a pipeline with two stages, DEV and PROD.

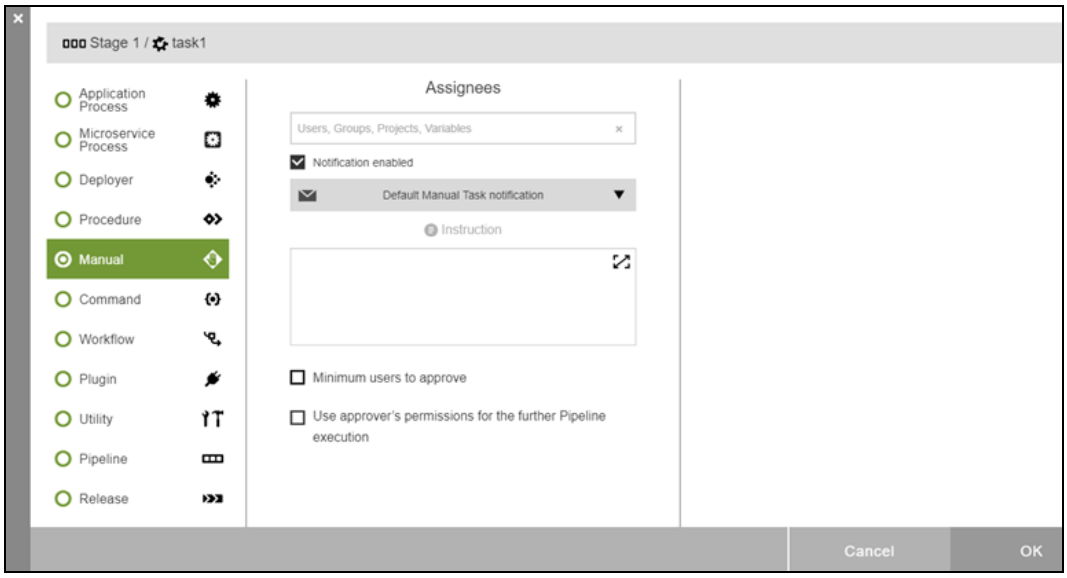
- UserD has access to run deployments only in DEV env, and userP has access only in PROD.
- UserD starts the pipeline to run in DEV; deployment runs with UserD's access.
- UserP approves entry gate to PROD. However, the stage runtime is still in UserD's authentication context, which doesn't have access, and therefore relies on the Project Principal's access and context in order to execute in the PROD environment.

The admin or any user with modify privileges on a pipeline or release is able to specify the runtime authentication context for the stage, task, or gate rule instead of keeping the authentication of the original pipeline launcher.

- When defining a gate rule or manual task, the user indicates if the authentication context should be switched to approver's authentication context.
- After the gate rule or manual task is approved, the authentication context of approving user is added to the session. The authentication context is switched in the pipeline and the currently active stage or gate and remains in force until it is switched again by downstream objects.
- Authentication context does not switch if the task or gate rule is rejected or fails, even if error handling for the task is set to **Continue on Error**.

Configuring Authentication Context Control

Add authentication control to a task or gate on the task or gate definition dialog.



Specify users or groups that can approve, configure notification, and the check boxes for **Minimum users to approve** and **Use approver's permissions for the further Pipeline execution**.

Make sure the following has execute privilege on:

- The pipeline.
- All stages until the stage or gate containing a task with **Use approver's permissions for the further Pipeline execution** checked. if pre-gate has such a task then execute privilege on stage is not required.
- All tasks before, and including the current task, with **Use approver's permissions for the further Pipeline execution** checked.

And read priviledge on:

- All stages after the one containing a task with **Use approver's permissions for the further Pipeline execution** checked.
- All tasks after the one with **Use approver's permissions for the further Pipeline execution** checked.

Restarting a Pipeline

When a pipeline run is restarted, the authentication context is reset and the restarting pipeline user authentication is added. For the examples below, the initial pipeline session primary authentication user is User1.

Example 1:

- User1 runs a pipeline with stage 1 and stage 2.
- The pipeline fails on stage 2 and User2 restarts stage 2: the pipeline session primary authentication user is now User2. The same for session 2.

Example 2:

- User1 runs pipeline with stage 1 and stage 2. Stage 2 has a manual task requiring approval in stage 1.

- User2 approves that task: the pipeline session primary authentication user is now User2.
- The pipeline fails on stage 2 and User3 restarts stage 2 . A new session is created with pipeline session primary authentication user as user3.

Pipeline Tasks

Each task can specify automation logic defined by an application or microservice process, procedure, or workflow. You can also call integrations to third-party tools using plugins, or you can specify manual tasks performed by one or more users.

Types of Tasks

- **Application or microservice process**: Use this to deploy applications or microservices at the right time in a pipeline. For an example of how to deploy applications or microservices, see Deploying and Troubleshooting Applications on page 287.

  When artifact staging is enabled for an application, ElectricFlow tries to retrieve artifacts that will be deployed before the application run starts. This ensures that the artifacts are downloaded and available on the target (either the artifact cache (the default) or a directory specified in the component definition). This minimizes the time to retrieve artifacts and reduces downtime during the application deployment. For details about artifact staging, see Artifact Staging on page 99

- **Procedure**: Use this to run a set of best practices, subroutines, modules, or functions that you can create and reuse at the platform level. Examples are scripts, command sets, or other automation logic objects. For details, see Pipeline Task Definitions in the UI on page 448.
- **Manual**: Use this when a task should be performed manually. The appropriate notifications are sent to the user or group who performs that task and to the user or group who responds when the task is completed. For more information about completing a manual task and restarting its pipeline, see Example: Visibility in a Release Pipeline from the Release Dashboard on page 640.Command: Use this to add command-based steps. This lets you perform command scripting as part of the orchestration.
- **Deployer**: This is a special type of task used when you want to deploy applications or microservices in a release but will provide the details in the release definition. When using deployer tasks, the system automatically knows the applications or microservices to deploy, which are provided as part of a release definition instead of separate deploy tasks for each application or microservice. For details, see Release Management on page 619. You can use one or more deployer tasks per stage.

- **Workflow**: Use this to run a set of procedures in CI life cycles at the platform level. A workflow can combine all procedures for a build-test-deploy life cycle. For details, see Workflow Overview on page 1570

  After the workflow is completed, it returns the workflow state to the pipeline as `ERROR` or `SUCCESS`:

  - `ERROR`: This means that the `ERROR_LOG` has entries, `/myWorkflow/outcome` or `/myWorkflow/result` is `error`, the workflow ended in a state that includes *Error* or *Fail* in its name, or it has any failed jobs.
  - `SUCCESS`: `/myWorkflow/outcome` or `/myWorkflow/result` exist and are not equal to `error`.
- **Plugin**: Use this to run third-party procedures that interface with external systems. Plugin tasks let you integrate and orchestrate third-party tools. For example, you can run Selenium automation at the right time by using the Selenium plugin. For details about plugin tasks, see Plugin Pipeline Tasks on page 563.
- **Utility**: Use this to run a utility function, which is a higher-order operation than a third-party plugin, as a stage task. You can also define an application process step, service process step, or component process step as a utility function. For information about defining an application, service, or component process step as a utility function, see Authoring Application Processes on page 268 or Authoring Component Processes on page 256.
- **Pipeline**: Use this to trigger another pipeline to run, either synchronously or asynchronously (default). See Pipeline Hierarchies on page 429
- **Release**: Use this to trigger another pipeline to run in a release context, referred to as a sub-release, either synchronously or asynchronously (default). See Pipeline Hierarchies on page 429

Task Skipping

You can skip specific tasks at runtime by using the **Enabled** check box for those tasks in a task list. This check box is checked by default.

Parallel or Serial Tasks

All tasks, including manual tasks, can be run in serial, in parallel, or in a combination of both. By default, tasks in a pipeline stage or within a deployer task run serially in the order in which they were defined. In complex application or microservice processes with numerous pre- and post-deployment tasks, sometimes tasks without dependencies should be run in parallel to reduce runtimes.

Parallelizing tasks shortens the time to complete a stage or deployment. In a short deployment window to complete large numbers of deployments, this feature can dramatically impact how many deployments can be completed in that window. For example, in a stage with two manual tasks, each of which requires a separate user to perform actions that do not depend on the other user, these tasks can run at the same time. For a release with 80 applications or microservices, in which each one has two manual tasks to prompt users, and each user response takes 15 minutes, the total wait time for these tasks is 40 hours with serial tasks and 20 hours with parallel tasks.

For information about creating parallel tasks, see Grouping Pipeline Tasks for Running in Parallel or Serial Order on page 479

Multiple Deployers

Using more than one deployer task in a pipeline stage lets you deploy applications or microservices in parallel rather than in series. This lets you control deployer behavior depending on the applications or

microservices to be deployed. This feature is useful for complex releases where you must coordinate the release of applications or microservices. Multiple deployers provide the flexibility to run other types of tasks between deployments such as infrastructure configuration tasks.

Following are examples of how multiple deployers are used:

- A release has ten applications. The first four are infrastructure applications (RPMs) that must run. After deployment of these four applications, a test is run to confirm successful infrastructure deployment. The remaining six applications can then be deployed.
- A release has ten microservices. Microservices one through five can run in parallel, and microservices six through ten can run in parallel, but six through ten have a dependency that one through five must be installed first. The microservices can be broken into two deployers that run serially with microservice deployments within them that run in parallel.

How Release Mappings Determine Multiple Deployer Behavior

If a deployer has a mapping to a release, then it deploys applications based only on that release application mapping. If it does *not* have a mapping, then it deploys all applications that are part of the release configuration.

Simple Use Case

Let's say you have two deployers, and neither deployer is mapped to a release. In this case, both deployers will deploy all applications.

Complex Use Case

Let's say that a release has three applications:

- Application1
- Application2
- Application3

And that the release contains a pipeline stage with deployers as follows:

- Deployer1—serial
- Deployer2—expression (take all applications from the project)
- Deployer3—parallel

And that the release has the following configuration:

- Application1 -> Deployer3
- Application2 -> Deployer3
- Application3 -> Deployer3

Then the applications will be deployed as follows:

- Deployer1 deploys all applications (because there is no explicit mapping for Deployer1).
- Deployer2 deploys any applications that meet the expression criteria.
- Deployer3 deploys Application1, Application2, and Application3 (because a user-defined mapping exists for Deployer3 to Application1, Application2, and Application3).

Error Handling Behavior for Deployer Tasks Running in Parallel in a Group

Following are a few scenarios and their behaviors:

Scenario	Outcome
The deployer task has `stopOnError`, and one of its applications with `stopOnError` fails.	All deployer tasks in the group are aborted, and the next task (if any) is activated.
The deployer task has `continueOnError`, and one of its applications with `continueOnError` fails.	Other deployer applications continue to run. It has no impact on other deployer tasks in the group.
The deployer task has `stopOnError`, but all of its applications have `continueOnError`.	The applications continue to run. If any of them fails, the deployer task aborts other deployer tasks in the group once all applications are completed.
The deployer task has `stopOnError`, one of its applications has `stopOnError`, and the rest have `continueOnError`.	If the application with `stopOnError` fails, the deployer task aborts all applications in the current deployer task and other deployer tasks in the group. However, if an application with `continueOnError` fails, the deployer task aborts other deployer tasks in the group once all applications are completed.
The deployer task has `stopOnError` and all of its applications also have `stopOnError`.	If an application fails, the deployer task aborts all applications in the current deployer task and other deployer tasks in the group.
The deployer task has `continueOnError`, and all of its applications have `stopOnError`.	If any application fails, the remaining applications in the deployer task are aborted, but other deployer tasks are not impacted.

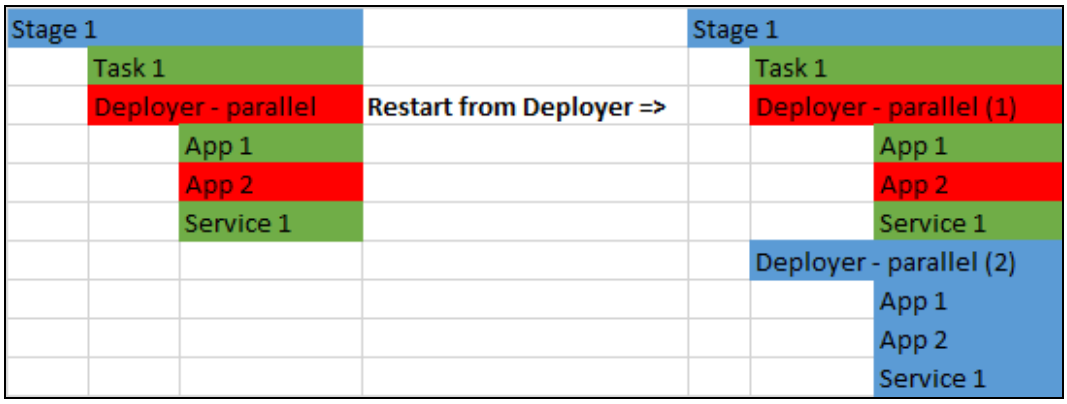
By default, all group tasks (including deployer group tasks) use `stopOnError` error handling.

Behavior for Restarting Parallel and Serial Deployer Tasks and Task Groups with Different Error Handling

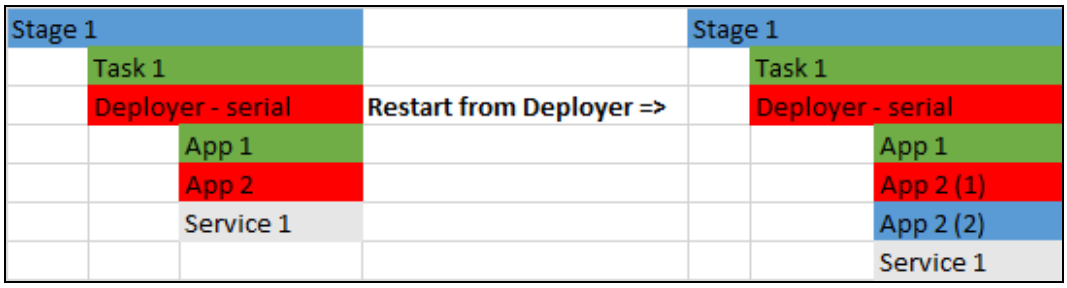
This section describes the various cases for restarting parallel and serial deployer tasks and groups with different error handling.

stopOnError Error Handling

When you restart a parallel deployer task or group, ElectricFlow restarts the entire deployer task or group:

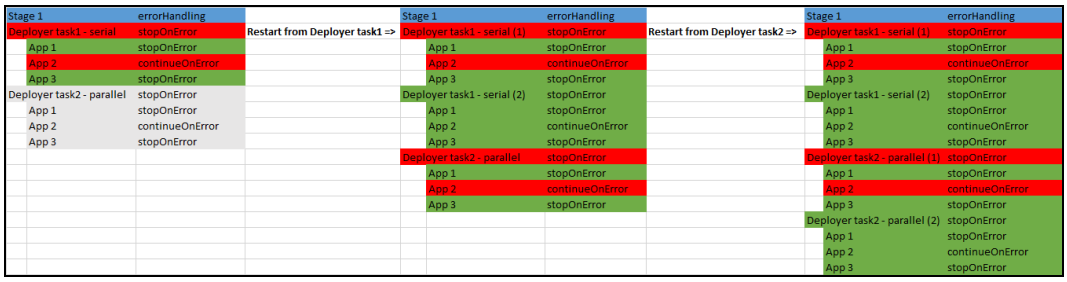


When you restart a serial deployer task or group, ElectricFlow restarts the run for the failed deployer application or service in the deployer task or from the failed subtask in the group:



continueOnError Error Handling

If a serial group or deployer task has a subtask or deployer application (or service) with `continueOnError`, ElectricFlow restarts the entire deployer task or group:



Error Handling in Tasks and Serial or Parallel Task Groups

When defining a task or a group of parallel or serial tasks, you must specify whether the pipeline continues or stops if the task or task group fails because of an error. To do so, you must select either **Continue on Error**, **Stop on Error**, or (additionally for pipeline tasks except manual and group tasks), **Manual Retry on Error** or **Automated Retry on Error**.

Manual Retry on Error or **Automated Retry on Error** let you enable a retry capability to rerun specific pipeline tasks that have failed. For example, if a hard drive fills up and causes a pipeline failure during tests near the end of that pipeline, you do not need to rerun the entire pipeline.

Following are the available error handling choices for tasks and serial or parallel task groups.

Error Handling	Available in Individual Non-Manual) Tasks?	Available in Manual Tasks?	Available in Parallel Group Tasks?	Available in Serial Group Tasks?	Description
Continue on Error	Yes	Yes	Yes	Yes	If any task fails or is rejected, the pipeline will still continue, allowing any other parallel tasks to complete before moving onto the next stage task or gate.
Stop on Error	Yes	Yes	Yes	Yes	If any task fails, the pipeline aborts. This is the default behavior. This behavior is usually used in Continuous Delivery pipelines.
Manual Retry on Error	Yes	No	No	Yes	An approver or assignee reviews the error and retries, skips, or fails the task. The test failure is caught (and the name of the task turns red to indicate the failure). Then, the pipeline prompts an operator with `Test failed. Do you want to fail the pipeline or rerun the tests?`. If the operator chooses "rerun," the pipeline reruns the last task.

Error Handling	Available in Individual Non-Manual) Tasks?	Available in Manual Tasks?	Available in Parallel Group Tasks?	Available in Serial Group Tasks?	Description
Automated Retry on Error	Yes	No	No	Yes	The task is automatically retried at specific time intervals for a specific number of times, after which you can specify whether the pipeline stops or continues.

Always-Run Capability for Tasks and Task Groups

You can define an "always run" task, which runs regardless of the error handling behavior for preceding tasks. This capability is most commonly used for cleanup operations at the end of a stage or gate.

You can also define an "always run" task group. Enabling this option guarantees that the tasks will run before the pipeline execution ends. Note that if this option is selected, and any individual task in the group that has error handling set to **Stop on Error** fails, then all the running tasks will be aborted.

Restart Capability for Failed Pipeline Runs

A restart capability for failed pipeline runs lets you configure a pipeline run to restart from the last failed task after you update the task definition and fix the issues that caused the failure.

A pipeline run is restartable if the last executed task in the run failed. (If it is also the last task in a stage or gate, then its error handling must not be "continue on error.") Any prior failed tasks (in the same or previous stages or gates) with "continue-on-error" error handling enabled are ignored.

Restarting a Pipeline from Any Stage Without a New Run

You can restart pipelines from any stage or task for better control when recovering from failures. This feature provides more flexibility and granularity over pipeline restarts. You can restart pipelines from any stage or even from the task level. This provides finer control to determine exactly which set of tasks needs to be restarted and from where, while maintaining the context of the entire pipeline. The pipeline view lets you easily see all runs of stages and tasks for auditability into the entire process.

This feature lets you restart the same pipeline run from any previously-run stage rather than creating a new run. The previous run could have completed successfully, failed, or could be still active (such as when it is waiting on a manual task).

This feature includes the ability to restart a pipeline from the point of failure manually or automatically at a point earlier than the failure point. This lets you restart an entire stage using the parameter values and output properties from the prior stages in the pipeline to maintain the context of the entire pipeline run. This functionality provides a single view for all runs if you have multiple runs for each stage, so that you do not need to know the specific runs where a stage was successful.

Usage Scenarios

Following are a couple of examples that illustrate the benefits of this feature:

- In a pipeline with Dev, QA, and Prod stages where the fourth task on the QA stage failed, you could restart the same pipeline run from that task. This feature extends this capability by letting you restart the run from the beginning of the QA stage, for example. This means that you need not create a new pipeline run from the QA stage, which avoids losing the parameter values and other runtime context from the earlier run.
- Let's say you have a pipeline for mainframe automation with Stage, UAT, Pre, Blue, and Prod stages, and each stage contains tasks. When the pipeline starts, user inputs are gathered via parameters.

  The UAT stage contains the first few tasks that drive automation in ISPW such as Promote the Release, Get Information from ISPW, and Create Evidence Report. After those tasks, a manual task in UAT seeks manual approval to go forward. In some cases, when a pipeline run reaches the approval task in UAT, the approver might notice several details to be tweaked on the ISPW side before the pipeline can proceed. But when those details are tweaked in ISPW, the entire UAT stage must be rerun without losing the context of all input parameters and so on until the UAT stage begins.

Stage Restart Details

- If at restart time, the previous run for the stage is still active, then all running tasks are aborted, and remaining tasks are skipped.
- When you restart a pipeline run from a specific stage, ElectricFlow injects a new stage between the failed stage and the next stage (and leaves the failed task as-is for your examination later). This lets you view the new run as well as all previous runs. Every new run for the stage or task will be identified with a new run number.
- When multiple runs exist for a stage, the context-relative property reference always looks up the properties (such as `/myPipelineRuntime/QA_stage/prop1`) on the latest run.
- The stage count and progress percentage computation are based on the latest stage runs.

Task Restart Details

- As with the stage restart functionality described above, instead of resetting and rerunning the failed task, ElectricFlow injects a new task (with the same name) between that task and the next task.
- Restarting a pipeline from a failed task is as described in Restart Capability for Failed Pipeline Runs on page 425.
- The restart behavior for group tasks is as follows:
  - Restart the entire task group if the task is parallel.
  - Restart from any task in the group, if the task group is serial.

  For more information, see Error Handling in Tasks and Serial or Parallel Task Groups on page 423.

Pipeline Restart Options

You can choose from the following main options for a pipeline restart:

Restart Option	Restrictions	API Command	Result
Restart a pipeline from a failed task	This is for failed pipeline runtimes only. It does not apply if the last completed task did not fail (when the failed task has `continueOnError` error handling).	`ectool restartPipelineRun <flowRuntimeId>`	• If the stage failed on condition or precondition evaluation, a new flow run-time state (`flowRuntimeState`) for a stage is created under the pipeline runtime. • If a high-level task (not a group or deployer task) failed, a new flow run-time state is created for a task under the stage or gate runtime. • If a subtask in a serial group or deployer task failed (`stopOnError` error handling), a new flow run-time state for a subtask is created under a group or deployer flow runtime. • If in the release configuration, the order or number of deployer items for a serial deployer task (applications or services) was changed, then a new ElectricFlow runtime state for the deployer task is created on restart. • If a subtask in a parallel group or deployer task failed or a subtask with `continueOnError` error handling in a serial group or deployer task failed, a new flow run-time state for a group or deployer task is created under the stage runtime.

Restart Option	Restrictions	API Command	Result

Restart a pipeline from a specific task	This is only for active stages or gates (for running pipelines) or last completed stages or gates (for failed pipelines). You cannot restart from a task that was not started yet.	`ectool restartPipelineRun <flowRuntimeId> --taskName <taskName>`	If the pipeline is running, the active task is aborted first, and then the following restart logic is used:
			• If a specified task coincides with the last failed (or aborted) task, then the results are as follows:
			  ◦ If a subtask in a serial group or deployer task failed (`stopOnError` error handling), a new flow run-time state (`flowRuntimeState`) for a subtask is created under a group or deployer flow runtime.
			  ◦ If a subtask in a parallel group or deployer task failed or a subtask with `continueOnError` error handling in a serial group or deployer task failed, a new flow run-time state for a group or deployer task is created under the stage runtime.
			• If a specified task is before the last completed task, then the new flow run-time states are created for all tasks from specified to the last completed.
			• If in the release configuration, the order or number of deployer items for a serial deployer task (applications or services) was changed, then a new ElectricFlow runtime state for the deployer task is created on restart.

Restart Option	Restrictions	API Command	Result
			• No specific logic is used for group or deployer tasks (the entire task is recreated).
Restart a pipeline from a specific stage	You cannot restart from a stage that has not run yet.	`ectool restartPipelineRun <flowRuntimeId> --stageName <stageName>`	If the pipeline is running, the active stage is aborted first, and then the following restart logic is used: For each stage from the specified one and to the last completed one, new flow run-time states (`flowRuntimeState`) under the pipeline are created.

Assigning a Resource or Pool to a Stage or Task for Commands, Procedures, or Plugins

By default, pipeline command, procedure, or plugin tasks run in the default pool or under a resource or pool specified within the `runCommand` procedure in the EC-Core plugin. You can specify a resource or pool for these types of tasks to execute on. You can also make this selection at the stage level, which will apply to all applicable tasks unless overridden there. This feature lets you run tasks on specific deployment environments.

Pipeline Hierarchies

Use pipeline and release tasks to create a pipeline hierarchy comprised of first-level and dependent pipelines. Consider an organization having pipelines that are dependent on each other at different stages. These pipelines deploy into shared environments, some stages may actually provision environments used by the release or other releases. So if a stage misses its deadline it can cause a cascading effect delaying other releases. Pipeline hierarchies provide a way to organize pipelines in order to minimize risks, bottlenecks, and delays in the release.

When defining pipeline and release tasks, you can:

- Create a task that triggers a dependent pipeline, either directly or in a release context.
- Run the dependent pipeline synchronously or asynchronously.
- Configure the behavior of the triggering pipeline run when the dependent pipeline fails (error handling).
- Configure a stage, task, or gate of the triggering pipeline to wait for the triggered dependent pipeline or additional, non-triggered pipelines to complete before it runs.

Setting up a Pipeline Hierarchy

1. For each triggering task: set task type to either **Pipeline** or **Release**at task definition time, based on the runtime context of the triggered pipeline—either outside of a release or in a release context, respectively.
    - Specify the triggering pipeline behavior in the event of a error or rejection by the triggered pipeline .
    - Specify how triggering pipeline is run: asynchronously or synchronously.

2. For each pipeline object: set wait conditions. See Wait Dependencies for details.

Pipeline Hierarchy Example

Consider three pipeline definitions A, B, and C, each with DEV, QA, and PROD stages.

Pipeline A:

- DEV Task 1 is a pipeline task that triggers Pipeline C, configured to run synchronously.
- QA Task 3 is a pipeline task that triggers Pipeline B, configured to run synchronously.
- Stage DEV sets **Wait for all triggered pipelines**.
- Triggered pipelines B and C run in Pipeline A's runtime context. Other instances of triggered pipelines B and C run in the context of their triggering pipeline.

Pipeline B:

- DEV Task 1 is a pipeline task that triggers Pipeline C, configured to run synchronously.
- Stage DEV sets **Wait for all triggered pipelines**.
- PROD Task 3 is a manual task: it requires manual intervention to complete. Any upstream triggering pipeline is blocked until the task is marked as complete.
- Triggered Pipeline C runs in Pipeline B's context. Other instances of triggered Pipeline C run in the context of their triggering pipeline.

Pipeline C:

- PROD Task 9 is a manual task: it requires manual intervention to complete. Any upstream triggering pipeline is blocked until the task is marked as complete.
- Triggered Pipeline C runs in the context of its triggering pipeline.

Using the above pipeline definitions, consider the two separate hierarchies as follows:

- Hierarchy 1: Pipeline B Stage DEV Task 1 triggers Pipeline C.
  Pipeline Portfolio view:

- Hierarchy 2: Pipeline A Stage DEV Task 1 triggers Pipeline C and Stage QA Task 3 triggers Pipeline B. Because Pipeline B is triggered, this hierarchy inherits hierarchy 1. However, the runtime context for pipelines B and C is separate from the context in hierarchy 1.
  Pipeline Portfolio view:

Running the Hierarchy

Pipeline B run is initiated to start Hierarchy 1. After the run starts, look at the the Pipeline Portfolio for this run to get status:

- Click the  for Pipeline B to see information about this pipeline.

- Click the the represents the first stage for Pipeline B: this opens a status panel on the right. In this instance, you can see that This stage is waiting on a manual response from Pipeline C.

Looking at the current run instance for Pipeline C (Pipeline C_17_20181108145753) on the **Pipeline Runs** page, Task 9 in Stage PROD is pending manual confirmation.

Now, Pipeline A run is initiated to start Hierarchy 2. It triggers Pipeline C (Pipeline C_18_20181108151821), which is a different runtime instance than that for Hierarchy 1.

After the manual approval is completed for Pipeline C Stage PROD Task 9, Pipeline A triggers Pipeline B, which in turn triggers Pipeline C. And again, the hierarchy is pending on Pipeline C Stage PROD Task 9. Notice the first Pipeline C is complete.

The run progresses to Pipeline B Stage PROD, where another manual task is encountered.

After the manual approval is completed for Pipeline B Stage PROD, the hierarchy completes, as can be seen in the following image: all stages are now green.

Entry and Exit Gates

Gates at the entries and exits of the stages control the pipeline's progress:

- An entry gate controls whether a stage may begin and its tasks may be executed.
- An exit gate controls whether the stage may be completed and the pipeline may progress.

Gate approvals are optional; if a gate has no approvals, the pipeline's progress continues. Gate approvals can be either manual (where the approver must physically click the approval to move to the next stage) or automated (based on whether specific criteria are met) or a combination of both.

Manual Gate Approvals

During a pipeline run, you can use the pipeline stage summary to see pipeline progress at each stage. You can review the user-generated information in the summary to make approval or rejection decisions at a gate. If a gate has approvals, one or more of these actions occurs:

- Email notifications are sent to the specified approvers.
- If the approver approves the request, the pipeline's progress continues.
- If the approver rejects it, the pipeline ends with an appropriate error (except if **On Rejection** is set to **Continue running**).

You can configure specific users or groups to be allowed to manually approve the criteria at a gate as follows:

- You can set the approvals so that approval from only approver is sufficient for the pipeline to progress. In this case, you can have one approval rule and have all the approvers assigned to that rule.
- You can set the minimum number of people required to approve from a group.

  If you want all the key individuals to approve sequentially, you can specify multiple approval rules, each for a specific user.
- In addition to specific users, you can apply approval rules to groups. Approval or rejection from any user in the group lets the pipeline progress or stops it.
- When defining a gate, you must specify whether the pipeline continues or stops if the approval is rejected. To do so, you must select one of the following under the **On Error** field:
  - **Continue running**: If the user rejects the gate approval, the pipeline completes the task and then continues to the next gate or stage task with a known error.
  - **Stop running**: If the user rejects the gate approval, then the pipeline aborts. This behavior usually occurs in continuous delivery pipelines.

For example, if your pipeline has multiple stages for testing, you might want to set entry and exit criteria for each stage to ensure that the code is testing properly in the software delivery life cycle.

- In the system test stage, the QA engineer must approve the start of the system testing workflow. That user or someone else in the QA group can review the system test results before approving the exit criteria.
- Before anything can enter the production stage, you should have entry gate approvals by various individuals.

For information about adding manual gate approvals, see Creating a Manual Approval Rule in a Gate on page 519.

Automated Gate Approvals

Automated approvals lets you set criteria that must be met before code is promoted to later stages. For audits, automated approvals let you show that these criteria were met. You can specify promotion criteria that are fully automated based on whether a condition is met or a combination of automated conditions and manual approvals. You can specify automated criteria that are based either on the outcome of a procedure of your choice or the outcome of a condition that you specify.

Procedure-Based Criteria

Procedure-based criteria allow you to call any procedure. These criteria let you query third party systems and implement complex gating rules. The job status corresponding to the procedure being executed determines the gate task outcome. A job outcome of `Success` (finished with no errors) is considered to be approved. A job outcome of `Warning`, `Failure`, or any other non-success job status is considered to be rejected. You can call procedures and pass in all the required parameters. For information about adding procedure-based gate approvals, see Creating a Procedure-Based Approval Rule in a Gate on page 522.

Condition-Based Criteria

You can create criteria based on custom conditions of your choice that evaluate to true or false to approve or reject gates. A condition is a Javascript expression that leverages properties. You can specify criteria based on any intrinsic properties that are appropriate for a release pipeline and a pipeline runtime. For example, you can create a property on the pipeline runtime based on the output of a test suite and write an expression to evaluate whether the property is 50 or greater (pass) or less then 50 (fail). For examples of Javascript expressions, see the *KBEC-00360 - Using Context-Relative Shortcuts to Properties on Pipelines and related objects* KB article.

For information about adding condition-based gate approvals, see Creating a Condition-Based Approval Rule in a Gate on page 524.

Automated Gate Approval Examples

- Simple automated gate criteria—The previous stage finishes with a successful status, and the next stage should proceed.
- Sophisticated automatic gate criteria—Based on a custom calculation or script result, the next stage should proceed. For example, if the performance test results show no more than a 5% degradation over the last main release or if all failed unit tests are marked exempt.
- Combination of automated criteria and manual approval—Based on performance tests passed and QA manual approval. For example:
    - Test results-based condition—Move to the next stage if greater than 90 percent of tests have passed.
    - If 80-90 percent of tests passed, require manual approval.
    - If less than 80 percent of tests passed, reject the gate.
- External trigger. For example, wait until a ServiceNow ticket is approved to trigger gate approval and promote to the next stage.

Pipeline "Run if" and "Wait until" Conditions

Before ElectricFlow 6.5, pipelines ran sequentially from the first stage to the last stage, without skipping a stage, task, or approval gate. When there was an error, the pipeline had to restart from the beginning. Starting in ElectricFlow 6.5, the pipeline runs can start or end at any point as well as skip a specific stage, task, or gate using **Run if** and **Wait until** conditions.

In the UI, you can set run the **Run if** and **Wait until** conditions on pipeline stages, tasks, and gates.

You can also set the **Run if** and **Wait until** conditions on pipeline stages, tasks, and gates using API commands through the command-line interface or through the DSL IDE while authoring pipelines.

- A **Run if** (run condition) is a property reference that the pipeline evaluates before executing the next stage, task, or approval gate. The pipeline waits until one or more dependent run conditions are met.

  In an API command, this condition is "fixed text" or text embedding property references that are evaluated into a logical `TRUE` or `FALSE`. An \"0\" or \"false\" is interpreted as `FALSE`. An empty string or any other result is interpreted as `TRUE`.By default, this condition is not set and is `FALSE`. The property reference can be a JavaScript expression, for example, this expression could test whether the name of a step is equal to the value of a property called "restartStep":
  `$[/javascript (myStep.stepName == myJob.restartStep)]`

- A **Wait until** condition (precondition or *wait condition*), is a property that allows a stage, task, or gate to be created with "pause", which then pauses the pipeline at that point. The pipeline waits until one or more dependent conditions are met. When the pipeline status is eligible to transition from pending to runnable, this condition is evaluated.

  In an API command, this condition is fixed text or text embedding a property reference that is evaluated to `TRUE` or `FALSE`. An empty string, a \"0\" or \"false\" is interpreted as `FALSE`. Any other result string is interpreted as `TRUE`. The pipeline does not progress until the condition is `TRUE`. By default, this condition is not set and is `FALSE`.

  You cannot use timestamps in preconditions on any object that supports preconditions. This includes stages, gates, and tasks as well as procedures and process steps.

Run if and **Wait until** conditions can be used to:

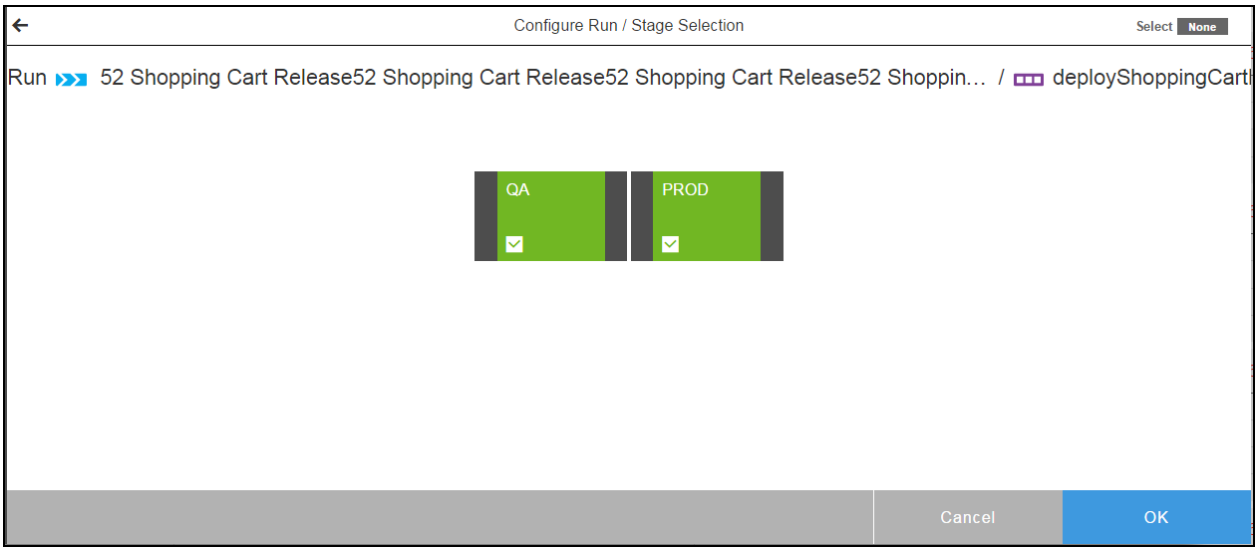
- Control a stage task based on the previous task.
- Skip some pipeline states based on some specific conditions, such as if a Linux installer file was not successfully built, tested, and verified.
- Execute certain approvals only if certain conditions are met. For example, during a functional test suite, the pipeline meets a rule or gate approval criteria if 90% of the functional tests have passed. If not, the pipeline waits for manual approval by assigned users before it can proceed.

For examples of how to set **Run if** and **Wait until** conditions on pipeline objects, see Pipeline Objects and Conditions on page 576.

Pipeline Start and End Stages and Stage Skipping

You can choose the start and end stages in a pipeline, as well as skip intervening pipeline stages in a sequence. Stage skipping is subject to skip restrictions configured by the pipeline owner. Additionally, pipeline access control can be configured down to the stage and gate level.

Following are specific examples of how you can use this feature.



Running or Re-Running a Stage Independently

In traditional release styles, when a deployment to a release stage fails, release management or operations often must fix the issue (such as with the code or environment) and start again from the failed stage to avoid running earlier stages unnecessarily. For example, stage 2 requires hours of testing, but stage 3 fails because a required server went offline. This functionality lets them determine if the error at stage 3 requires an entire pipeline rerun. Other examples are as follows:

- You can rerun a stage independently to redeploy a stage to update to a newer application or microservice release candidate that came out since the last stage run (successfully or not).
- Release management or operations running a release stage can to rerun a failed stage to recover from stage errors (such as a bad process step, failed server, or unavailable resource) that do not affect the final output, so that they can fix the problem and move forward without potentially rerunning potentially expensive stages that do not require a rerun.
- A release stage owner participating in a release with traditional characteristics can rerun the stage that they own with a different application or microservice version, because they received an update from the application or microservice team that needs testing in the release stage owner's environments.

Selecting a Subset of Pipeline Stages to Run

This functionality lets you start a pipeline and run stages only to a certain point or from a certain point without it being considered failed or left running and requiring an abort. For example:

- Beginning to middle—Consider a release pipeline that starts with continuous output of release candidates from the development team. In the first two weeks of a release, they continuously deploy different development outputs and test them in the first few stages of the pipeline, but they want it to stop and be considered a success after the test stage. Once the results are satisfactory, they change the target state to "production" and rerun all stages.
- Middle to end—You can recover from a stage failure by fixing the problem and then rerun the fixed stage to continue to production from the failure point. You can also start after the stage that you fixed manually (such as when a failed manual step to update the documentation was fixed).

Running Multiple Pipeline Stages at Once

This feature allows stages to run at the same time. With traditional releases, separate teams working on a release will work on their stages at the same time (and often in parallel), but their work still must be tracked and aggregated into the final release status. For example, if stage 4 is production, the operations department can start a time-consuming database backup before stage 3 (UAT) even begins. Other examples are as follows:

- The integration stage is being run to deploy and test the latest work from the developers, who might not test it. At the same time, the test stage is already running functional testing on the previous release candidates.
- The testing stage runs performance tests at the same time that the UAT stage is running user acceptance tests.The release manager examines the outcome of both before approving to move further down the pipeline.

Pipeline Scheduling

You can create schedules for pipeline runs. You can set a start date and can also make a run recurring (for example, daily or weekly).

Pipeline UI

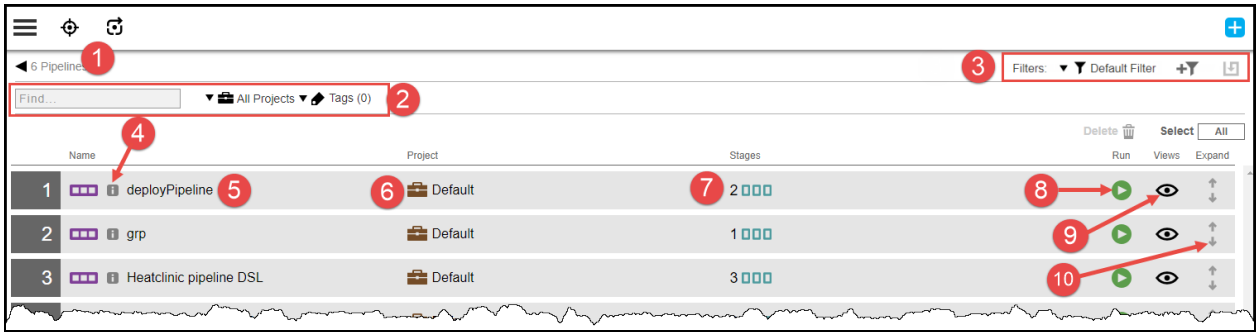
This section shows how pipelines are represented in the ElectricFlow UI. All of the information described in the previous sections is summarized in the pipeline definition in the Pipelines List.

ElectricFlow has two ways to view and edit pipelines: The Release Kanban View and the Pipeline Stage View. This section describes the two key differences between these views: the Hierarchy Menu and the process for plugin task creation and definition.

Pipelines List

The Pipelines List shows all the pipelines that you have permission to view. To open it from the Home page (https://<*ElectricFlow_server*>/flow), click **Pipelines** > **All Pipelines**.

This is a Pipelines List:



Each row in the list is a *pipeline definition*, consisting of the pipeline name, the project to which the pipeline belongs, and the number of stages in the pipeline. Selecting **All** or **None** and then clicking the **Delete** button removes the selected pipelines from the list. Clicking the blue **+** button in the upper right corner adds a pipeline.

1	Breadcrumb identifying the object you are viewing and the total number of pipelines.

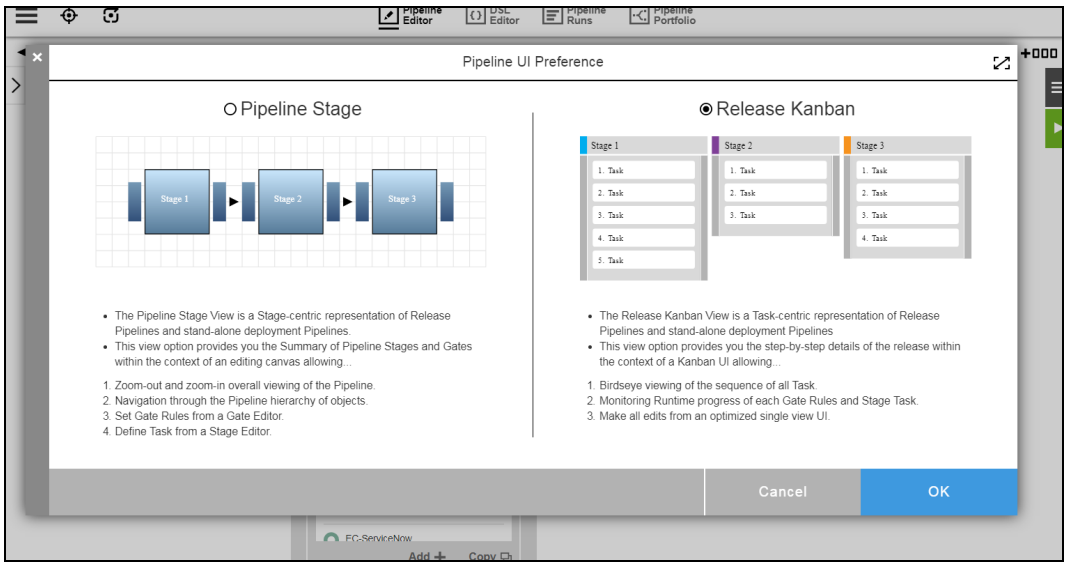
2	Default filter criteria—Configure the default filter here. See Searching and Filtering on page 19 for filter details; see Object Tags on page 45 for configuring tags. The default is to show the applications and microservices for all projects. To see only the objects for a specific project, click the down arrow in the **All projects** field and then select one or more projects in one of these ways: • Click on the name of one or more projects. • Enter the search criteria in the Search field. The projects that match the criteria appear in the list. If there are no matches, a message appears stating that there are no resource templates in the selected projects.
3	Filter selector—Select the filter to use on this page. Create custom filters here; see Searching and Filtering on page 19 for details.
4	Clicking **i** shows information about the object. In this example, clicking **i** shows the pipeline description: 
5	Name of the pipeline.
6	The project to which the pipeline belongs.
7	Number of stages in the pipeline.
8	Click the **Run pipeline** button to run the pipeline when the button is green.
9	Click the **View** button from which you can select the Pipeline Editor.
10	Click the **Expand** button to view the previous pipeline runs.

Pipeline Views

ElectricFlow provides two ways to view and edit a pipeline:

- Release Kanban View (this is the default view)
- Pipeline Stage View

Tip: You can change the system-wide default view with the **Use Release Kanban view for pipelines** ElectricFlow server setting. You can also prevent users from changing their pipeline view by disabling the **Allow users to change the Pipeline UI Preference** server setting (which is enabled by default). To change these settings, open the Automation Platform at https://<*ElectricFlow_ server*>/commander/ and choose **Administration** > **Server** > **Settings**.

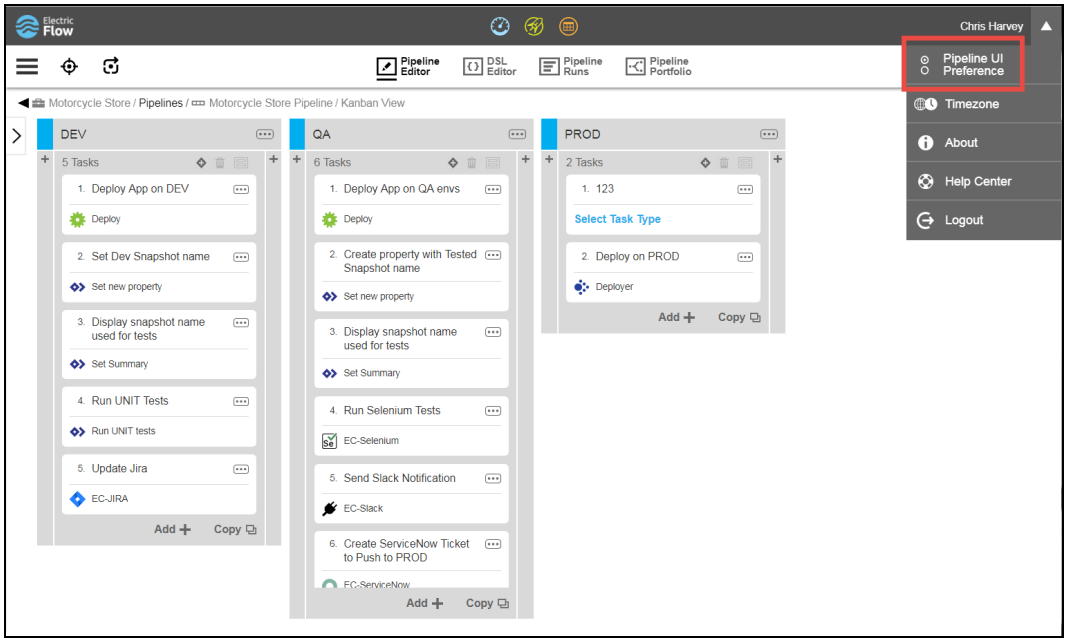


Changing Your Pipeline UI Preference to Another View

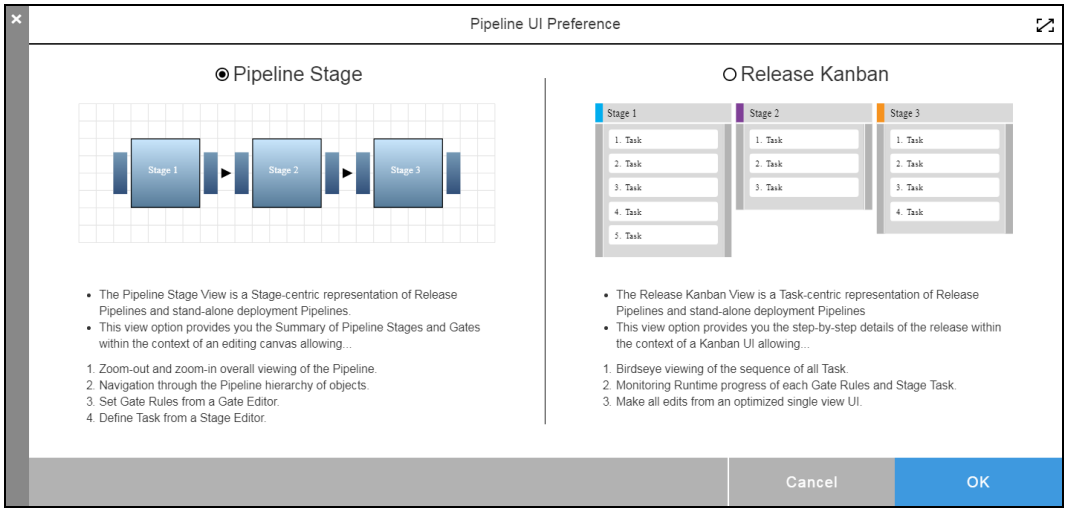
To change your pipeline UI preference to another view:

1. Open the home page of the ElectricFlow web UI by browsing to https://<*ElectricFlow_ server*>/flow/.

2. Click the home page pulldown menu and choose **Pipeline UI Preference**:



The **Pipeline UI Preference** dialog appears:



3. Click the **Pipeline Stage** radio button or the **Release Kanban** radio button, and then click **OK**.

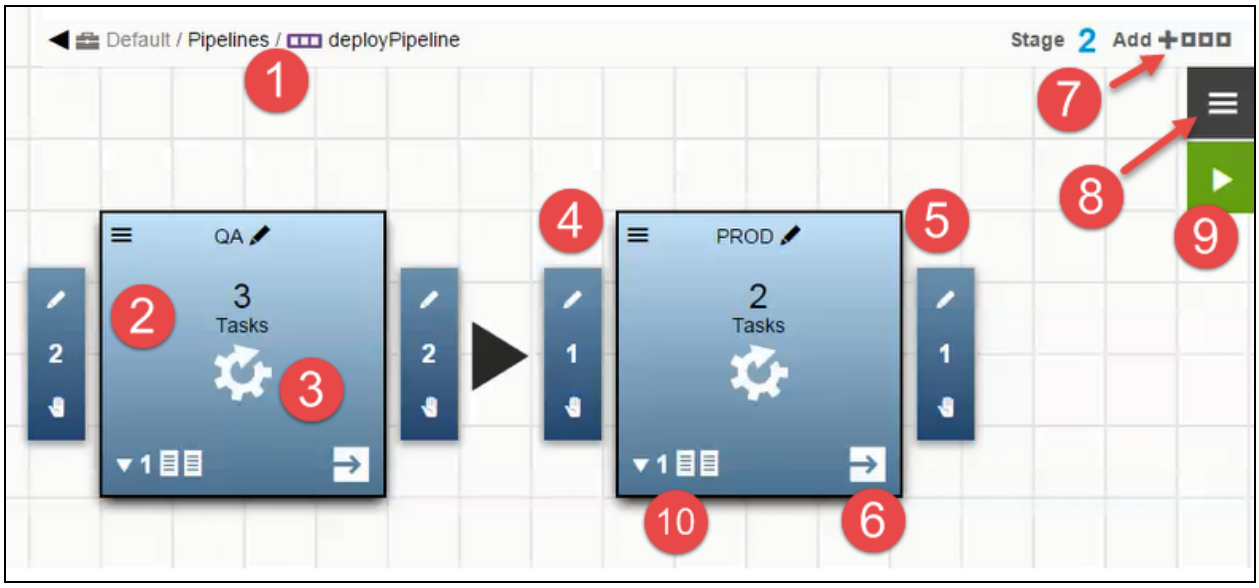
   The pipeline view changes to the type that you just selected.

Pipeline Stage View

The Pipeline Stage View is a Stage-centric representation of Release Pipelines and stand-alone deployment Pipelines. This view option provides you the Summary of Pipeline Stages and Gates within the context of an editing canvas to let you:

- Zoom out and zoom in overall viewing of the Pipeline.
- Navigate through the Pipeline hierarchy of objects.
- Set Gate Rules from a Gate Editor.
- Define a Task from a Stage Editor.

In the Pipeline Editor, you can author and edit a pipeline as well as view the underlying platform-level details. This is a pipeline with two stages: QA and PROD:



You can perform these actions:

- Clicking the **Add new stage** button (number 7) will add a new stage to the pipeline.
- Clicking the **Menu** button (number 8) will open a menu where you can get more details about the pipeline and take actions on the pipeline.
- Clicking the **Run pipeline** button (number 9) will start a pipeline run.
- In a stage, clicking the **Define stage tasks** button (number 6) will open a dialog box where you can define tasks for the stage.

1	Breadcrumb showing the object hierarchy, path to the pipeline, and the pipeline name.
2	Stage in the pipeline, a milestone or checkpoint in the software delivery process.
3	Number of tasks in a stage, which is the automation logic for a stage. See Pipeline Tasks on page 419. To access the tasks, click the (Define stage tasks) button in the bottom right corner (number 6 in the figure above).
4	Entry gate, consisting of the approvals to enter a stage. See Entry and Exit Gates on page 432.

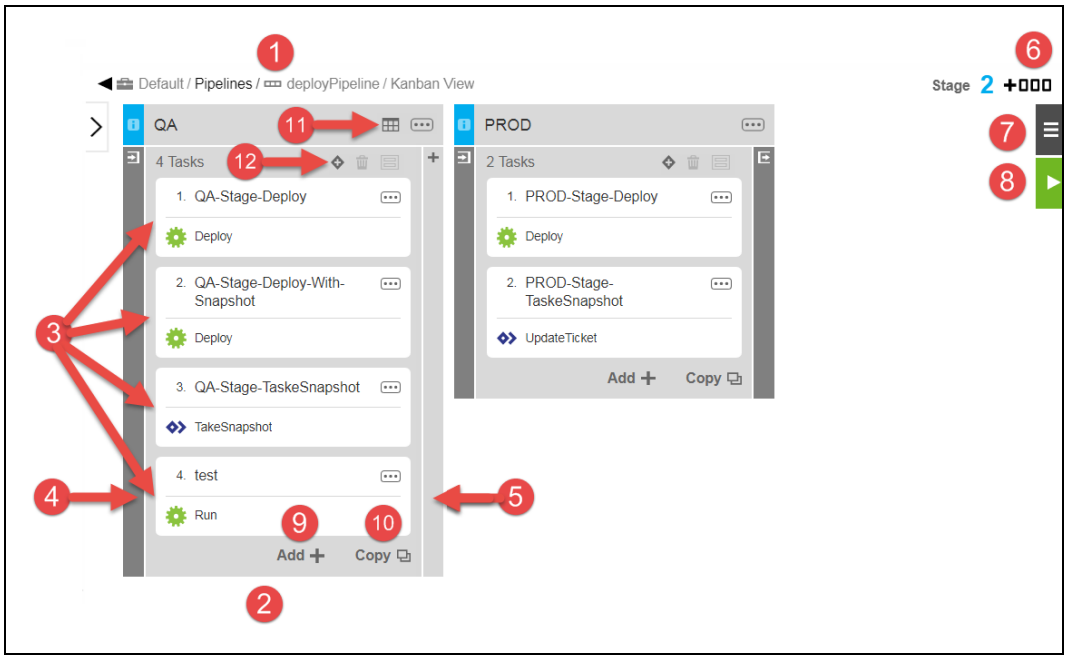
5	Exit gate, consisting of the approvals to exit a stage. See Entry and Exit Gates on page 432.
6	Click the  (Define stage tasks) button to open the task list for the stage.
7	Click the **Add new stage** button to add a stage.
8	Click the **Menu** button to open the menu where you get more details about the pipeline and take actions on the pipeline. • Clicking **Details** opens the **Pipeline Details** dialog box where you can edit the pipeline name and description. • Clicking **Properties** opens the **Properties** dialog box where you can view, add, and delete the properties associated with the pipeline. • Clicking **Parameters** opens the **Parameters** dialog box where you can view, add, and delete the parameters attached to the pipeline. • Clicking **Access Control** opens the Access Control page where you can view, add, and edit the privileges for the pipeline. • Clicking **Track Changes** shows the Change History for the pipeline and the objects in it. ElectricFlow tracks the changes to tracked objects including applications and microservices, procedures, workflows, workspaces, resources, and project-owned components such as library components and records a Change History of the historical states of the system and the changes between them. • Clicking **Delete** deletes the pipeline after you verify that you want to do this.
9	Click the **Run pipeline** button to run the pipeline. The Pipeline Runs List shows the real-time progress of the pipeline run. Clicking in a stage will open the Stage Summary where you can view task details and progress of each task. You can also view the results of previous pipeline runs.
10	Click to see the list of environments that the stage will be operating on. This pulldown menu appears only if there are deployment tasks (application or service process tasks) in that environment.

Release Kanban View

The Release Kanban View is a Task-centric representation of Release Pipelines and stand-alone deployment Pipelines. This view option provides you the step-by-step details of the release within the context of a Kanban UI to let you:

- See a birds-eye view of the sequence of all Tasks.
- Monitor the runtime progress of each Gate Rule and Stage Task.
- Make all edits from an optimized, single-view UI.

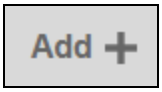
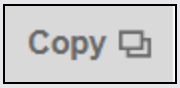
In the Release Kanban Editor, you can author and edit a pipeline as well as view the underlying platform-level details. This is a pipeline with two stages: QA and PROD:



You can perform these actions:

- Clicking the **Add new stage** button (number 7) will add a new stage to the pipeline.
- Clicking the **Menu** button (number 8) will open a menu where you can get more details about the pipeline and take actions on the pipeline.
- Clicking the **Run pipeline** button (number 9) will start a pipeline run.
- In a stage, clicking the **Define stage tasks** button (number 6) will open a dialog box where you can define tasks for the stage.

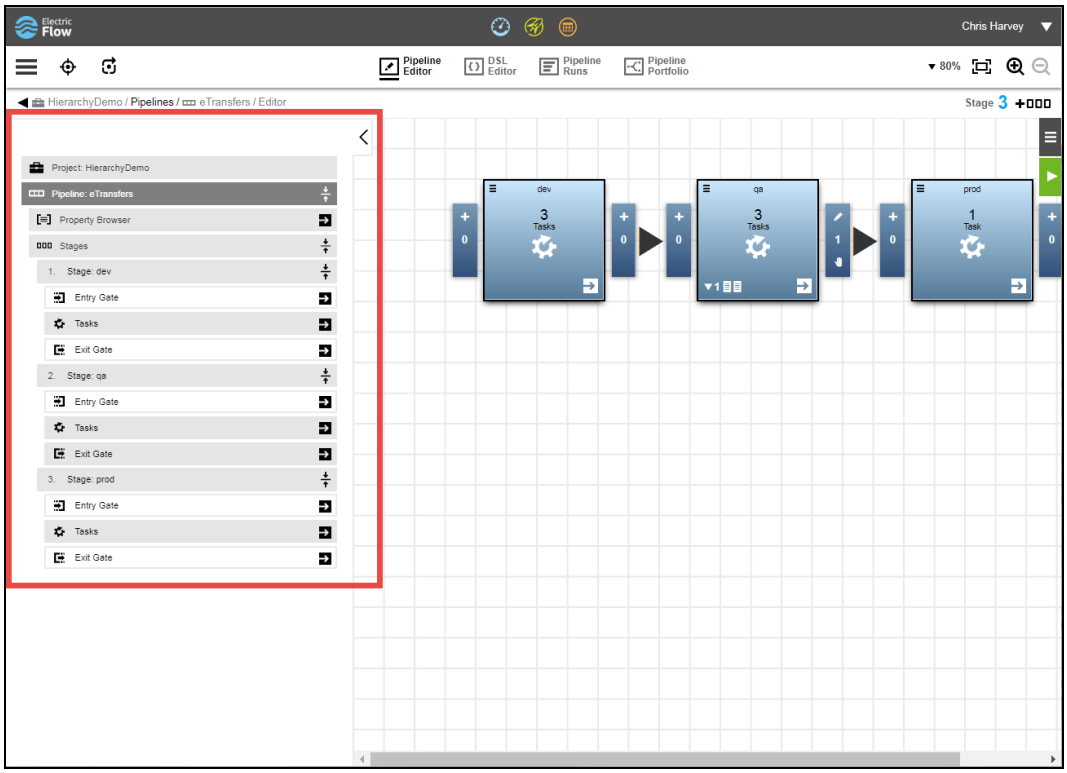
1	Breadcrumb showing the object hierarchy, path to the pipeline, and the pipeline name.
2	Stage in the pipeline, a milestone or checkpoint in the software delivery process.
3	Number of tasks in a stage, which is the automation logic for a stage. See Pipeline Tasks on page 419. To access the tasks, click the Add + (Add Task) button at the bottom of the stage (number 9 in the figure above).
4	Entry gate, consisting of the approvals to enter a stage. If a gate contains rules, it appears in gray and becomes expandable via mouse click so that you can view or edit the rules. If it has no rules, it appears in white (and is not expandable). In See Entry and Exit Gates on page 432.
5	Exit gate, consisting of the approvals to exit a stage. If a gate contains rules, it appears in gray and becomes expandable via mouse click so that you can view or edit the rules. If it has no rules, it appears in white (and is not expandable). See Entry and Exit Gates on page 432.

6	Click the **Add new stage** button to add a stage.
7	Click the **Menu** button to open the menu where you get more details about the pipeline and take actions on the pipeline. • Clicking **Details** opens the **Pipeline Details** dialog box where you can edit the pipeline name and description. • Clicking **Properties** opens the **Properties** dialog box where you can view, add, and delete the properties associated with the pipeline. • Clicking **Parameters** opens the **Parameters** dialog box where you can view, add, and delete the parameters attached to the pipeline. • Clicking **Access Control** opens the Access Control page where you can view, add, and edit the privileges for the pipeline. • Clicking **Track Changes** shows the Change History for the pipeline and the objects in it. ElectricFlow tracks the changes to tracked objects including applications and microservices, procedures, workflows, workspaces, resources, and project-owned components such as library components and records a Change History of the historical states of the system and the changes between them. • Clicking **Delete** deletes the pipeline after you verify that you want to do this.
8	Click the **Run pipeline** button to run the pipeline. The Pipeline Runs List shows the real-time progress of the pipeline run. Clicking in a stage will open the Stage Summary where you can view task details and progress of each task. You can also view the results of previous pipeline runs.
9	 Click the (Add Task) button to create a new task from the beginning.
10	 Click the (Copy Task) button to create a task by copying another task.
11	 Click the (calendar) button to view or set the start and end dates for this stage. This button appears only if there are start and end dates set.
12	 Click the (Stage Conditions) button to view or set "run if" and "wait until" conditions and pipeline or release wait dependencies for this stage.

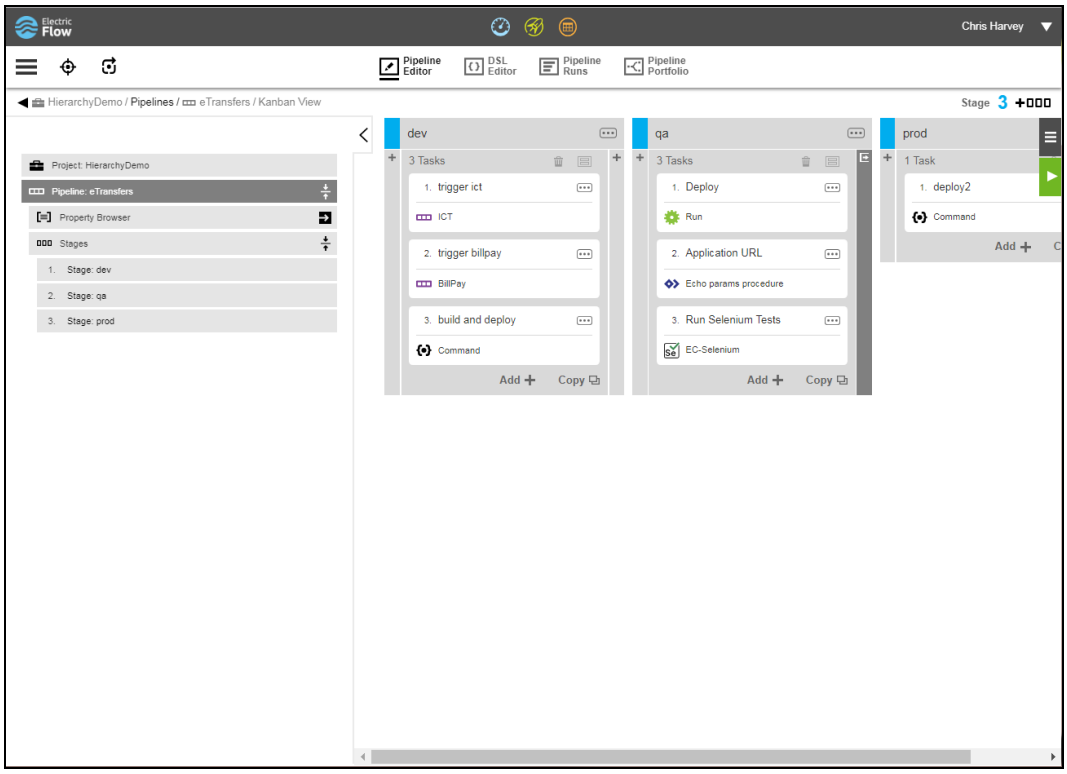
Hierarchy Menu in the Pipeline Stage View

The Hierarchy Menu in the Kanban View differs from the Pipeline Stage Task view as illustrated below. For details about these differences and using the Hierarchy Menu, see Hierarchy Menu on page 22.

Pipeline Stage View of example:



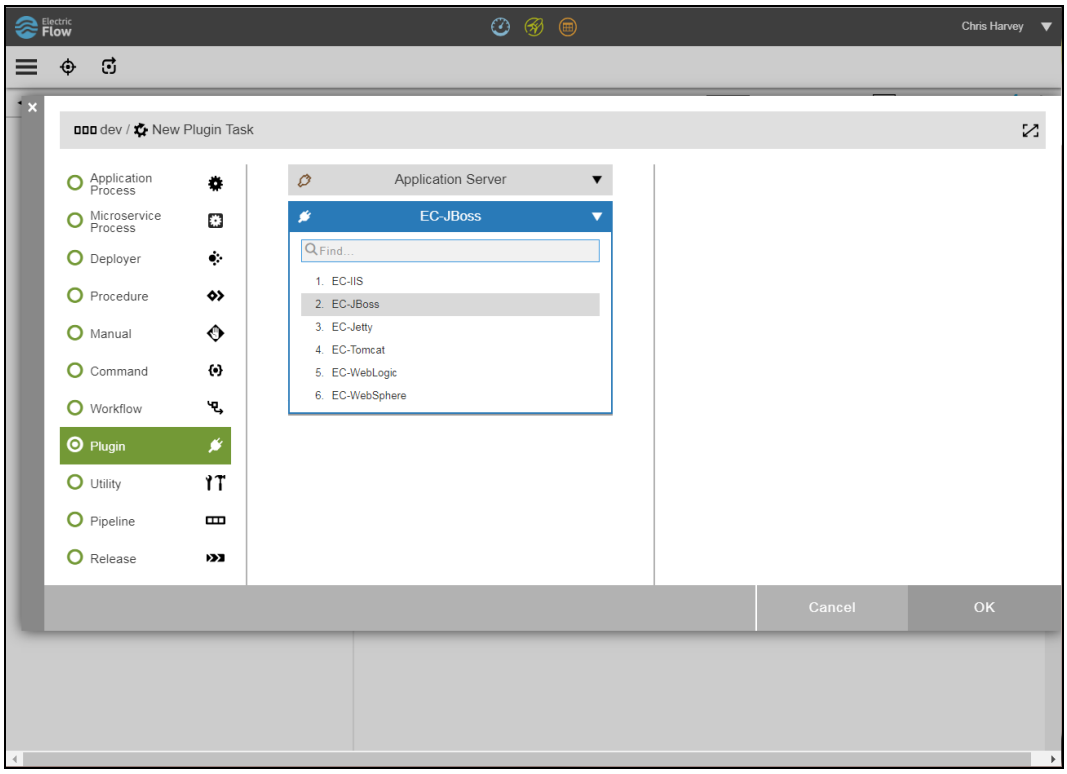
Release Kanban View of example:



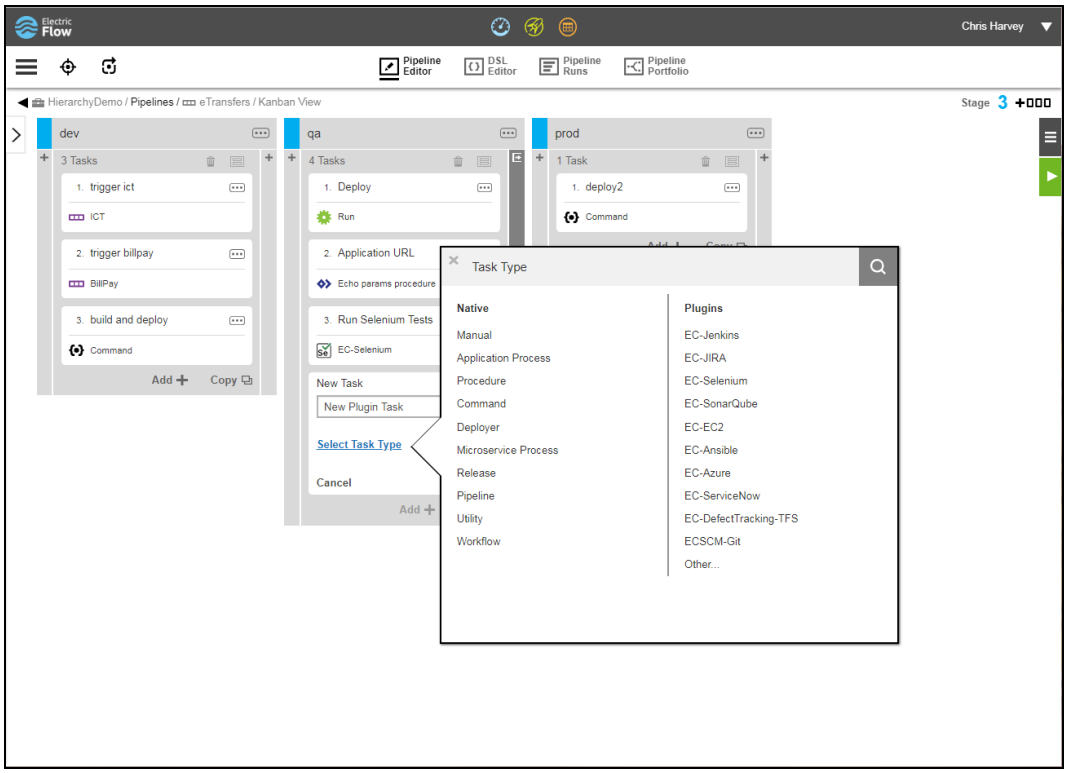
Hierarchy Menu in the Release Kanban View

Plugin task creation is easier than in the Pipeline Stage View, because you can select the plugin directly from the Kanban View:

Pipeline Stage View of example:

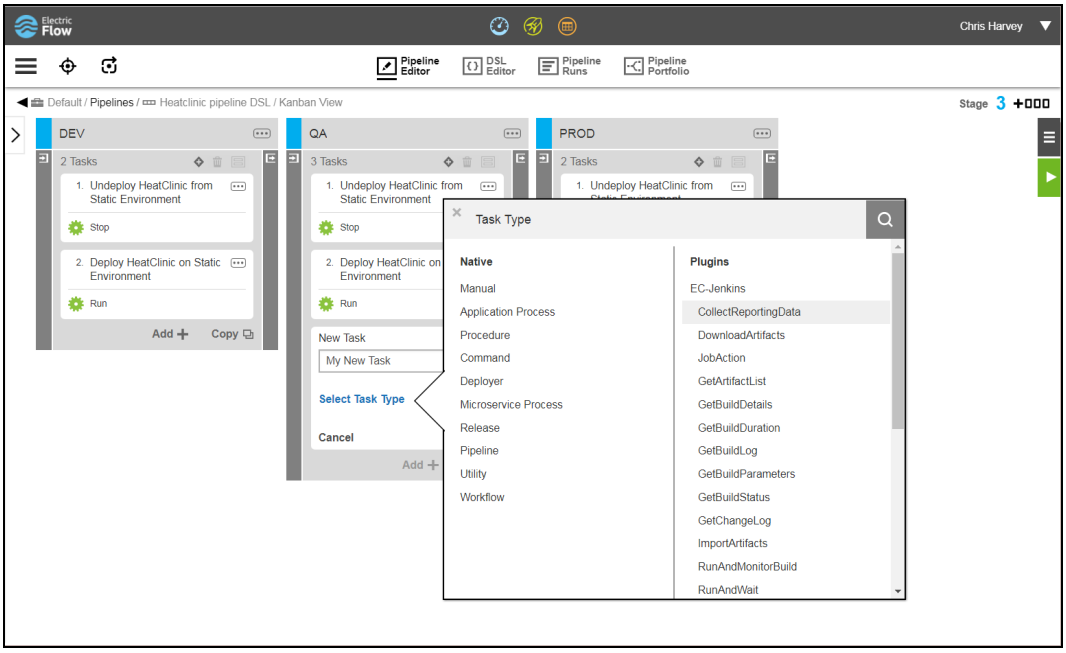


Release Kanban View of example:



The **Task Type** dialog, which contains the list of task types to select is divided into two categories: Native and Integrations.

- **Native**—List of standard task types such as Command or Application Process. Because the Release Kanban view lets you postpone the configuration of some details, you must add those details later to completely define the task by clicking its **Config** button once the task appears in the Kanban View. For example, if you select the Procedure task, you must then click its **Config** button to choose the actual procedure to invoke.
- **Integrations**List of plugins that are available for selection. When you select a plugin, you must also drill down to select one of its procedures to run. For example, the EC-Jenkins plugin and its CollectReportingData procedure:



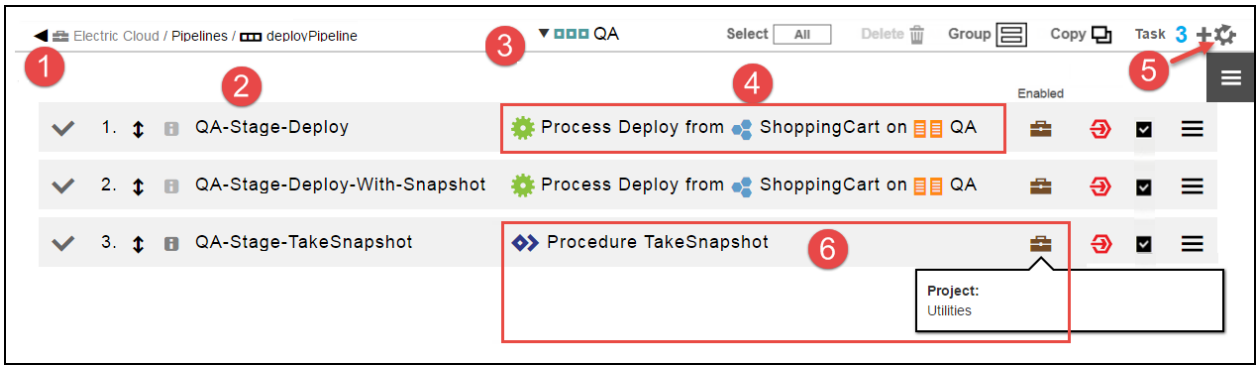
By default, ten commonly-used plugins appears in this list. You can expand the list to see all plugins by clicking **Other...**.

The ec_preferred_integrations property sheet determines the plugins to appear in this list and their order. To modify this list, open the Automation Platform at https://<*ElectricFlow_server*>/flow/, then select **Administration** > **Server** > **ec_deploy** > **ec_preferred_integrations**, and then edit the list.

For more information about property sheets, see Properties on page 1422.

Pipeline Stage List of Tasks

The task list for the stage shows all the tasks for the stage that are executed in sequential order.



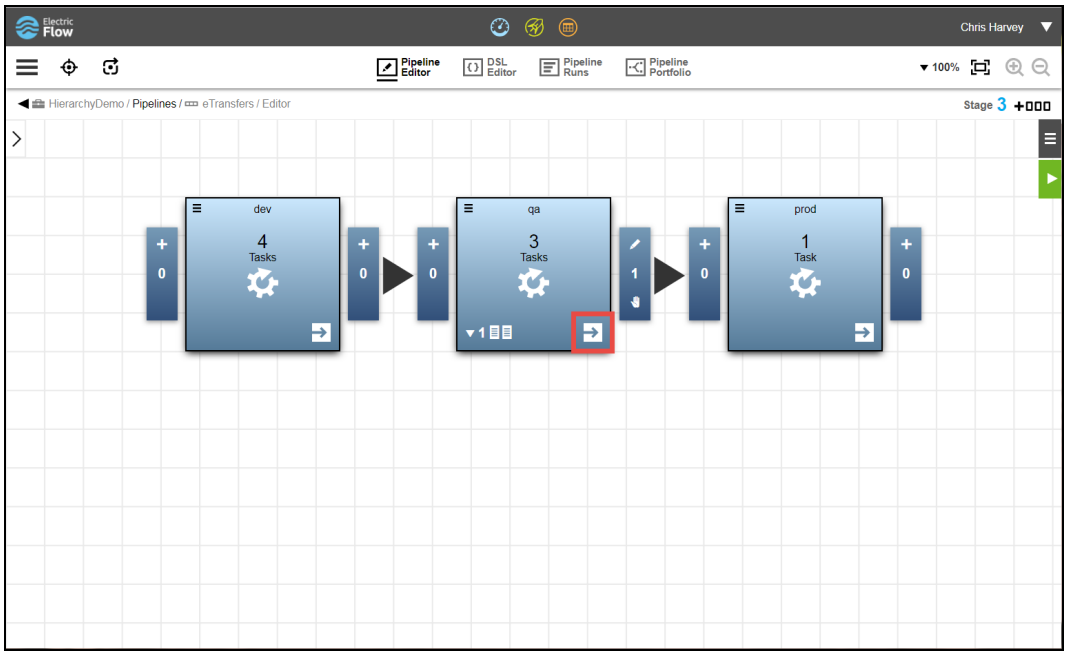
1	Breadcrumb showing the object hierarchy, path to the pipeline, and the pipeline name.
2	Name of the task.
3	Name of the stage.
4	The task type and the specific details for it. See Pipeline Tasks on page 419. This task is defined by an application process called Deploy from the "Shopping Cart" application. The application process is deployed in the QA environment.
5	Click the **Add task** button to add a task to the stage.
6	The task type and the specific details for it. See Pipeline Tasks on page 419. This task is defined by a procedure called TakeSnapshot in the Default project. Clicking on the **Project** button shows the project to which the procedure belongs.

Pipeline Task Definitions in the UI

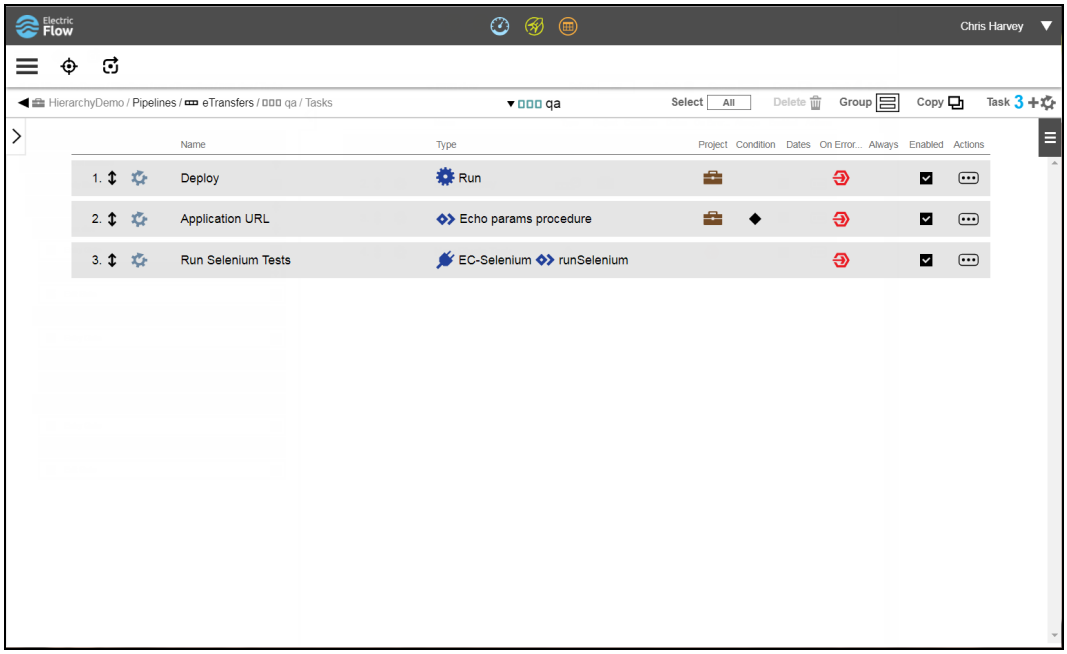
Tasks are defined in the **Task Definitions** dialog box described in this section. There are three ways to get to this dialog box to create or edit a task as described below.

Creating a Task in the Pipeline Stage View

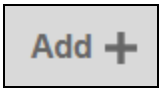
To create a task in the Pipeline Stage View, first click the (Details) button on the stage:



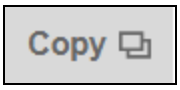
Then click the  (New Stage Task) button, then follow the prompts:



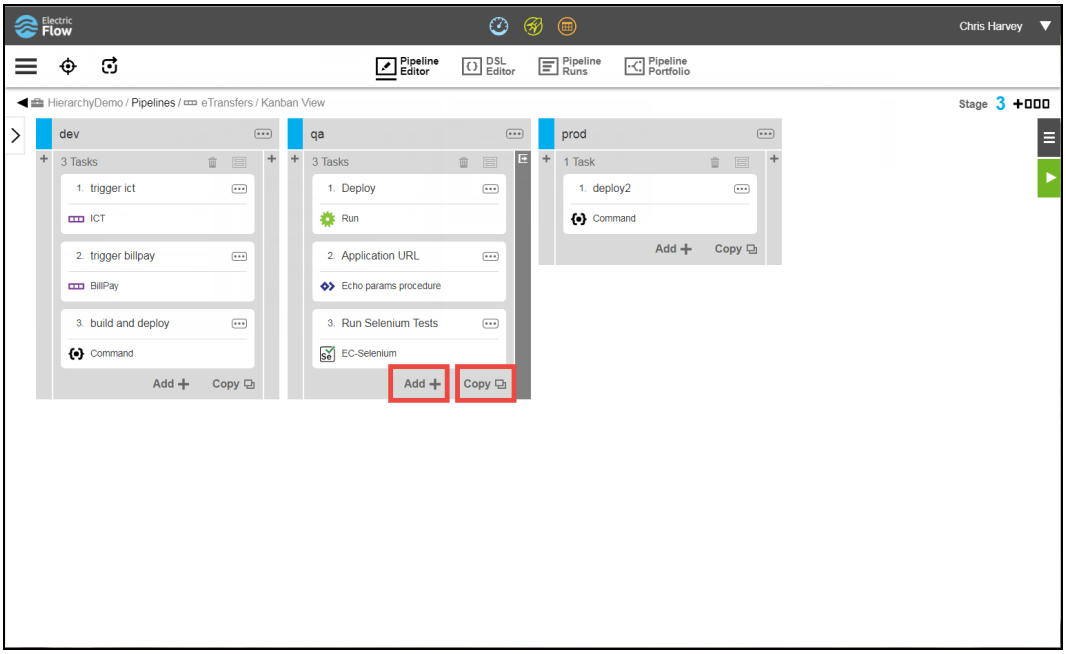
Creating a Task in the Release Kanban View



To create a task in the Release Kanban View, click the (New Task) button or the



(Copy from Existing Task) button, then follow the prompts:



Viewing or Editing a Task in a Task List

To view or edit a task in a task list, click the (Actions) menu in its row and select the appropriate action. Depending on the task type that you select, different fields appear on the right side of the dialog box.

Deploying an Application or Microservice

If you want to deploy an application or microservice, select **Application Process** or **Service Process** respectively. You need to select an application or microservice, a snapshot, an application process, and the environment where the process will run. If the parameters were created for the process, the Parameters field is enabled.

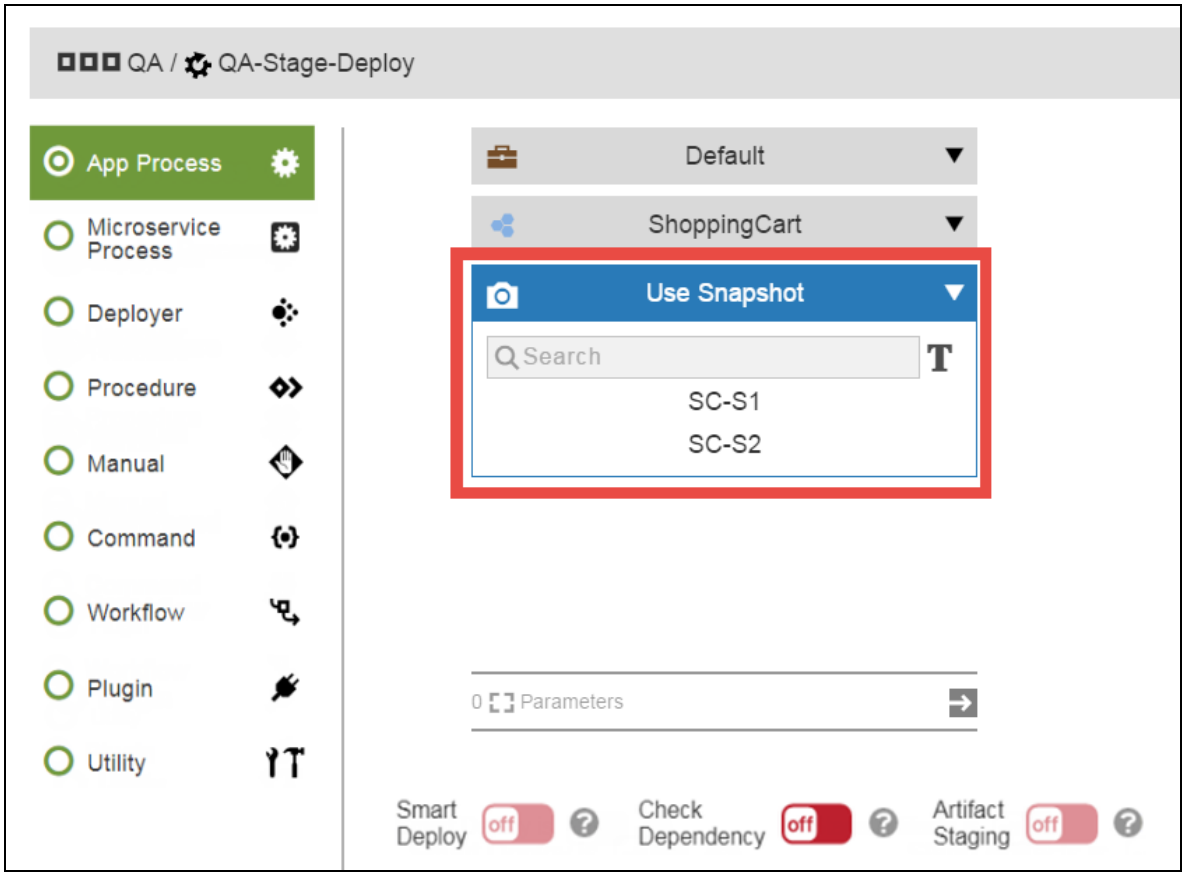
For applications, you can also enable or disable:

- **Smart Deploy**, where the system deploys only the artifacts that are not yet deployed to a resource, specific versions of the artifacts, or artifacts that are not yet deployed to a new resource.

- **Check Dependency**, which does an application dependency check before deployment. When this is enabled (the default), deployment will not start if the dependency check fails.
- **Artifact Staging**, which automatically downloads the required artifacts to the targets when the process starts. This facilitates artifacts to be staged locally on the targets instead of downloading during an active deployment and reduces downtime because the required artifacts for deployment are available locally.

For microservices, you can also enable or disable **Check Dependency**.

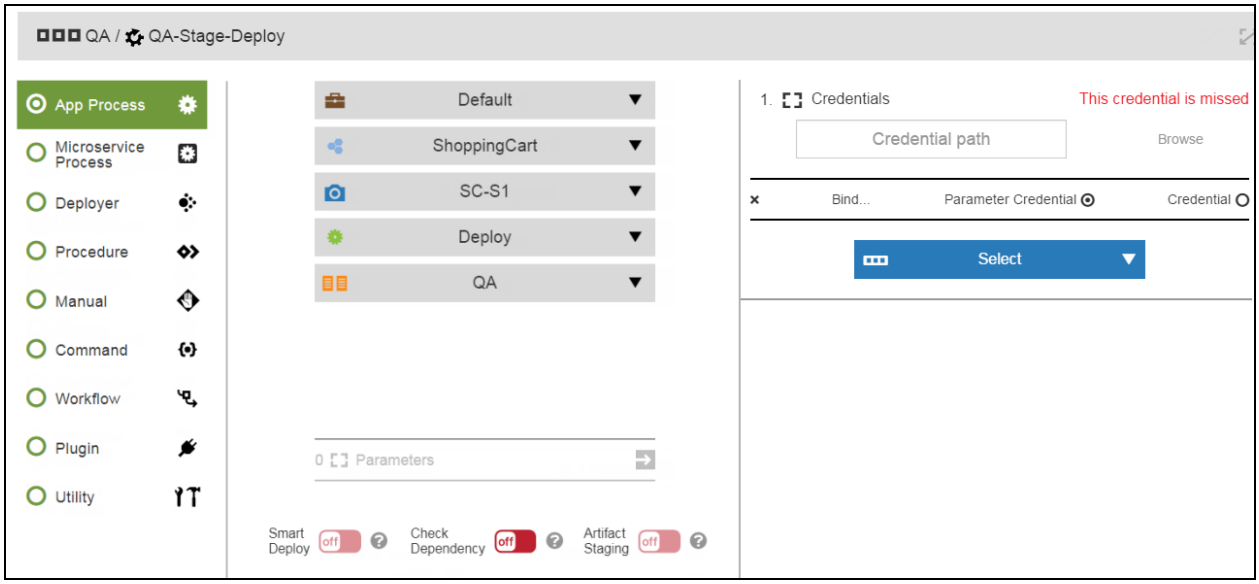
This example shows how to use a parameter to specify the snapshot. When you click in the "Select a Snapshot" field, these options appear:



Specifying a Snapshot

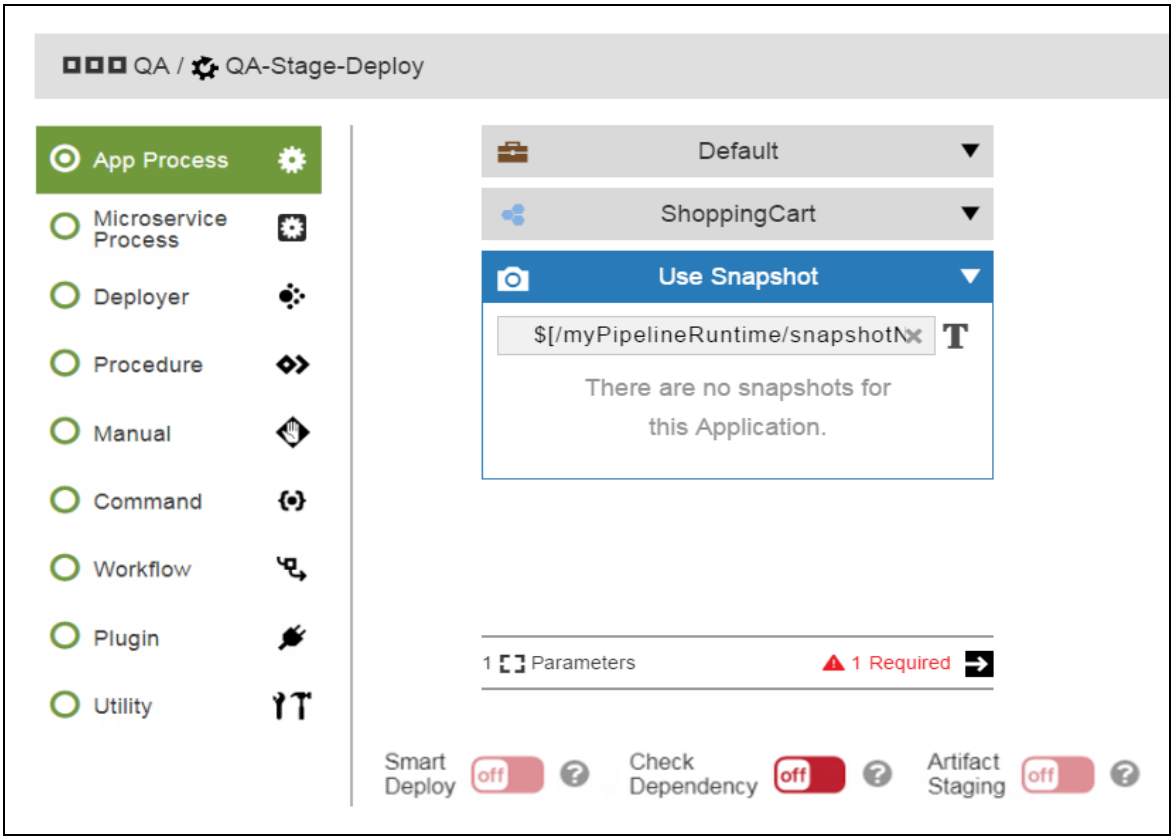
There are two ways to specify a snapshot:

- Select a specific snapshot for the application or microservice, such as **SC-S1**.



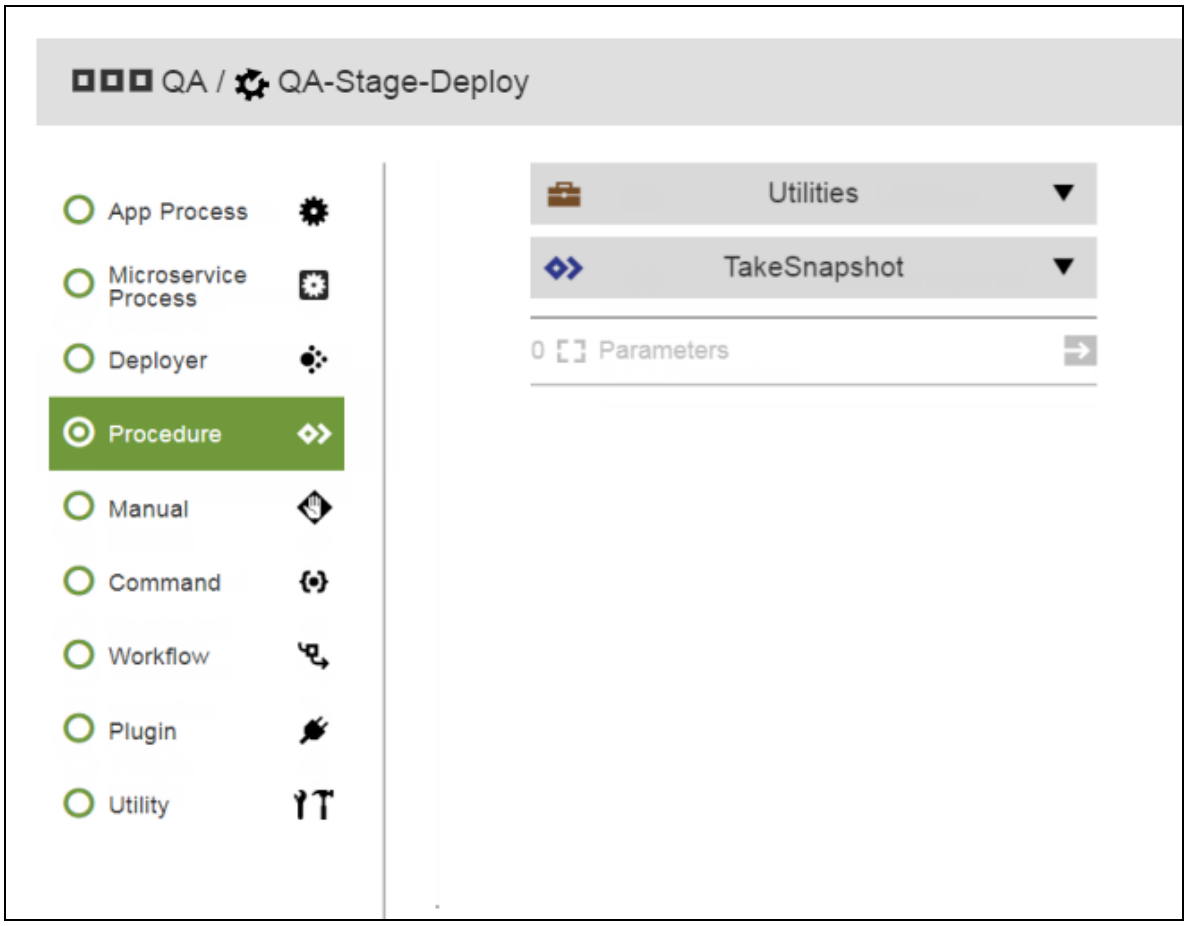
- Enter the snapshot as a parameter.

  Enter this parameter: `$[/myPipelineRuntime/snapshotName]`.



If you want to run a set of best practices, subroutines, modules, or functions that you can create and reuse (such as a script), select **Procedure**. You need to select a project and a procedure in that project. If the parameters were created for procedure, the Parameters field is enabled.

This example shows that the task is a procedure called TakeSnapshot in the Utilities project.



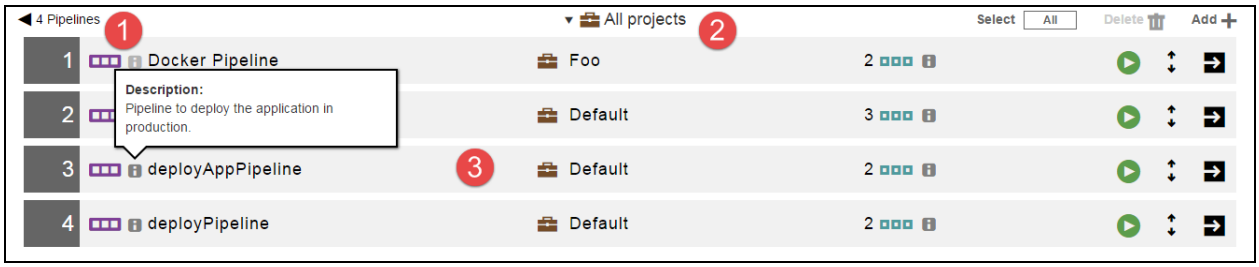
Example: Authoring and Running Pipelines

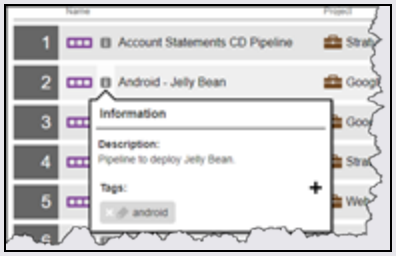
This example shows how to define and run a pipeline and view the results. This example covers the following topics:

- Parts of a Pipelines List on page 455
- Parts of a Pipeline on page 455
- Creating a Pipeline on page 457
- Editing Pipeline Stage Details on page 462
- Defining the Tasks in a Pipeline Stage Using the Pipeline Stage View on page 468
- Defining the Tasks in a Pipeline Stage Using the Release Kanban View on page 488
- Editing a Pipeline Definition on page 516
- Defining Gate Approvals on page 518
- Running Pipelines on page 536
- Viewing Pipeline Runs on page 542
- Troubleshooting Pipelines on page 549

Parts of a Pipelines List

Following are the key parts of a list of pipelines:

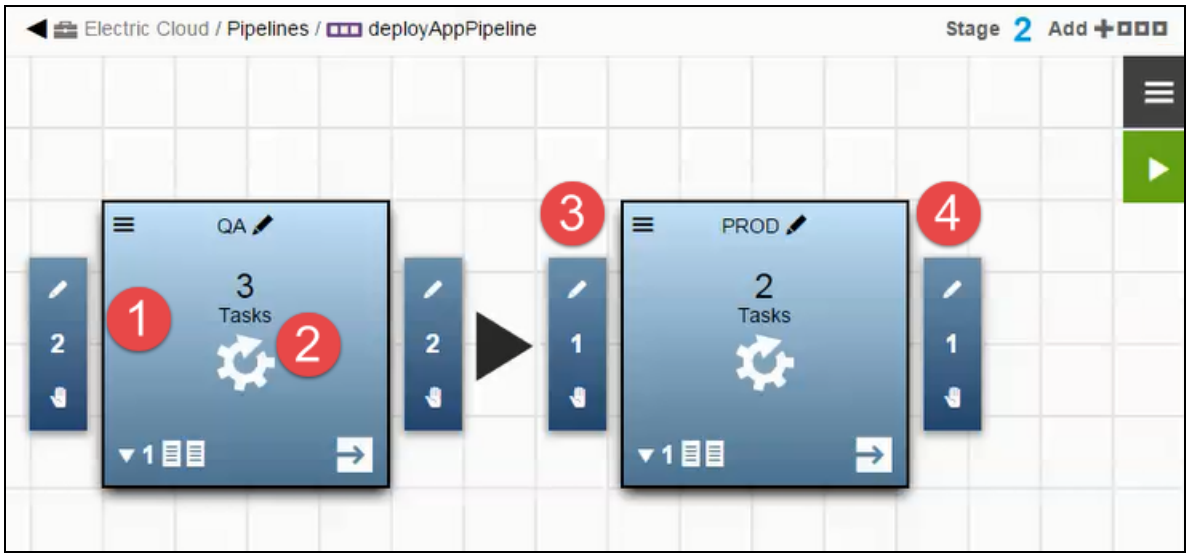


1	Breadcrumbs—Provide context for the current page and links to previous pages.
2	Default filter criteria—Configure the default filter. See Searching and Filtering on page 19 for filter details; see Object Tags on page 45 for configuring tags.
3	Filter selector—Select the filter to use on this page. Create custom filters here; see Searching and Filtering on page 19 for details.
4	Pipeline definition—a description of the pipeline, the project to which it belongs (**Stratus**), and the number of stages in the pipeline. • Clicking **i** next to the pipeline name shows a description of the pipeline and tags configured for the pipeline. See Object Tags on page 45for more information about tags.  • Clicking  opens the view selector. From here, you can open the pipeline editor, view pipeline runs, access the DSL editor, and view the pipeline portfolio.
5	Pagination controls—See Pagination on page 19 for details.

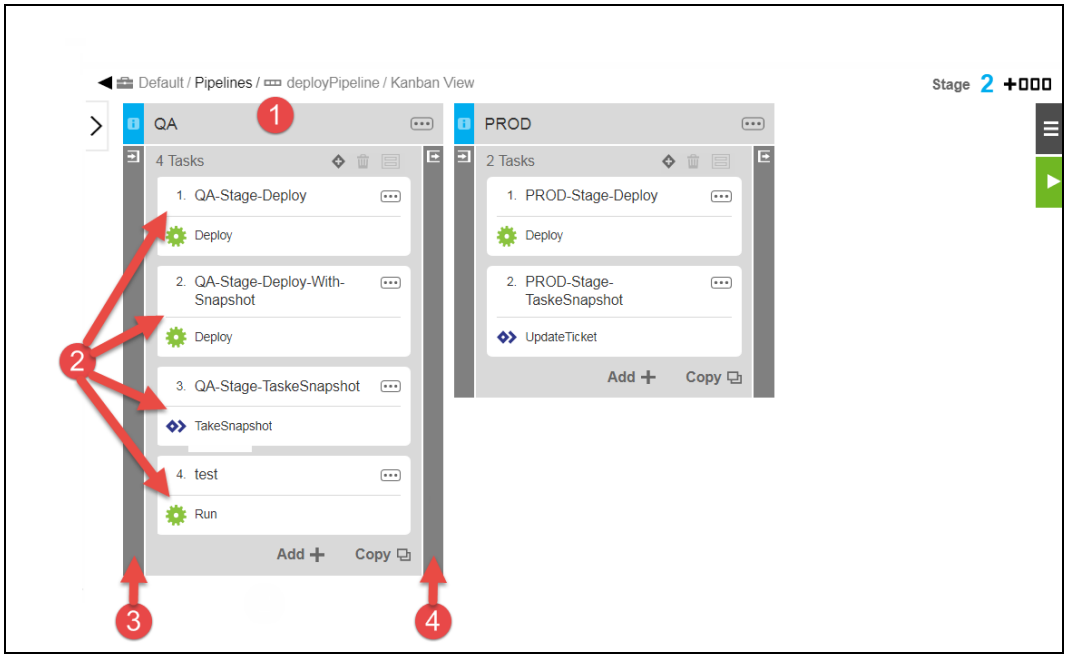
Parts of a Pipeline

Following are the key parts of a pipeline. This pipeline has two stages (QA and PROD).

Following is an example of a pipeline in Pipeline Stage View:



Following is an example of a pipeline in Release Kanban View:



Item	Description
1	Stage
2	Tasks
3	Entry gate
4	Exit gate

Important: When an application, a pipeline, or a release is cloned across different projects, you might need to fix the references after the objects are cloned. For example, if a pipeline with an application reference in the "default" project is cloned to a different project, the application reference needs to be fixed after the pipeline is cloned.

Creating a Pipeline

You can create a pipeline from scratch or based on an existing pipeline. To create a pipeline component from scratch:

Creating a Pipeline from Scratch

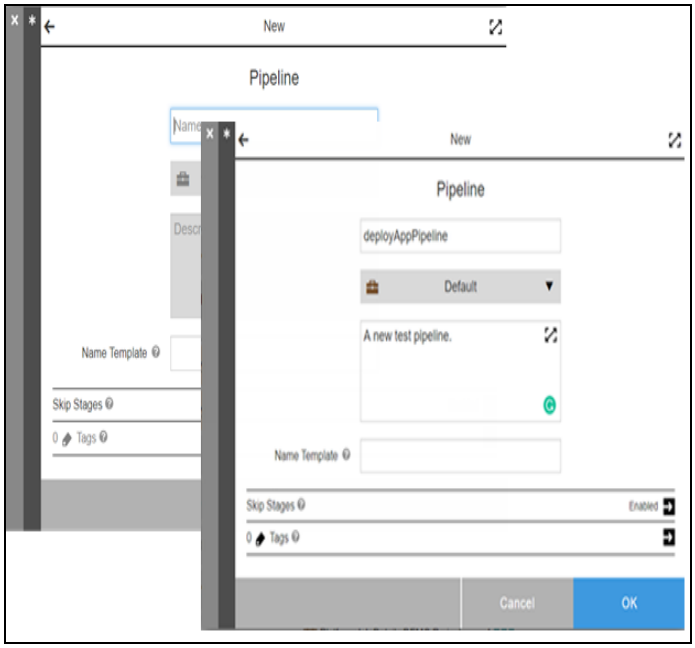
1. Open the home page of the ElectricFlow web UI by browsing to https://<*ElectricFlow_server*>/flow/.
2. Go to the pipelines list in one of these ways:
   - Click **Pipelines**.
   - Click the **Main menu** button in the upper left corner and then select **Pipelines** > **All Pipelines**.

   The pipelines list opens.

3. Click the (New Pipeline) button.
4. Click **Create New**.

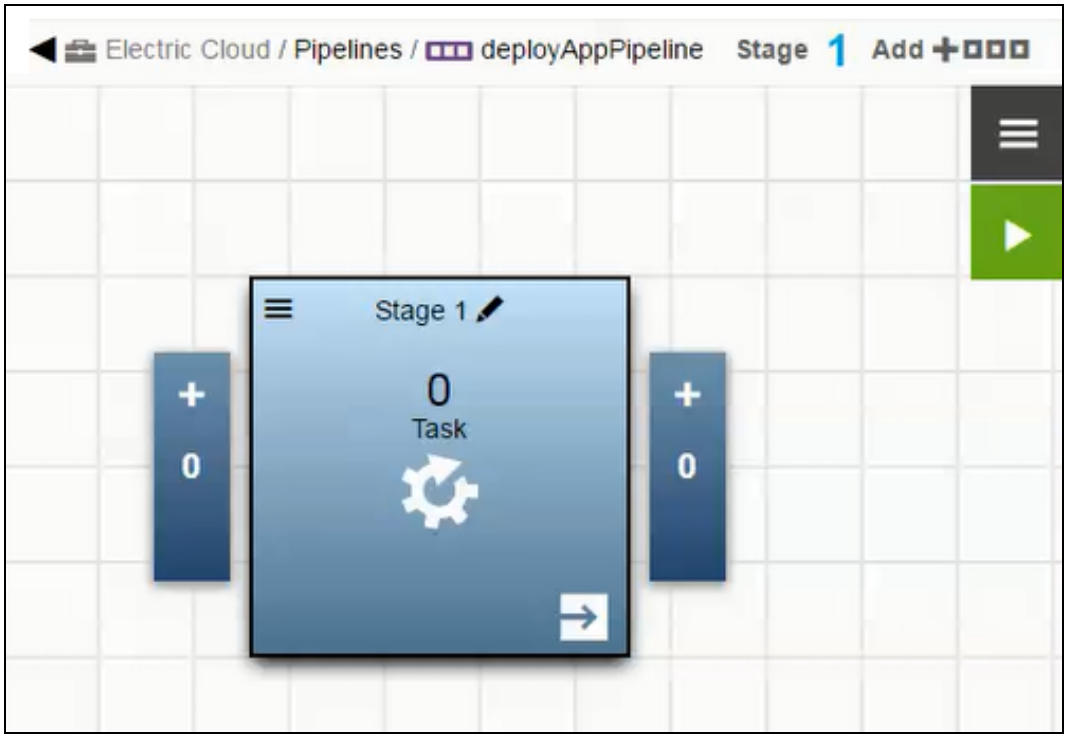
5. Enter the pipeline name, select a project where the applications in the pipeline will be deployed, and enter a description. Optionally, configure stage skipping and tags.

   For example:

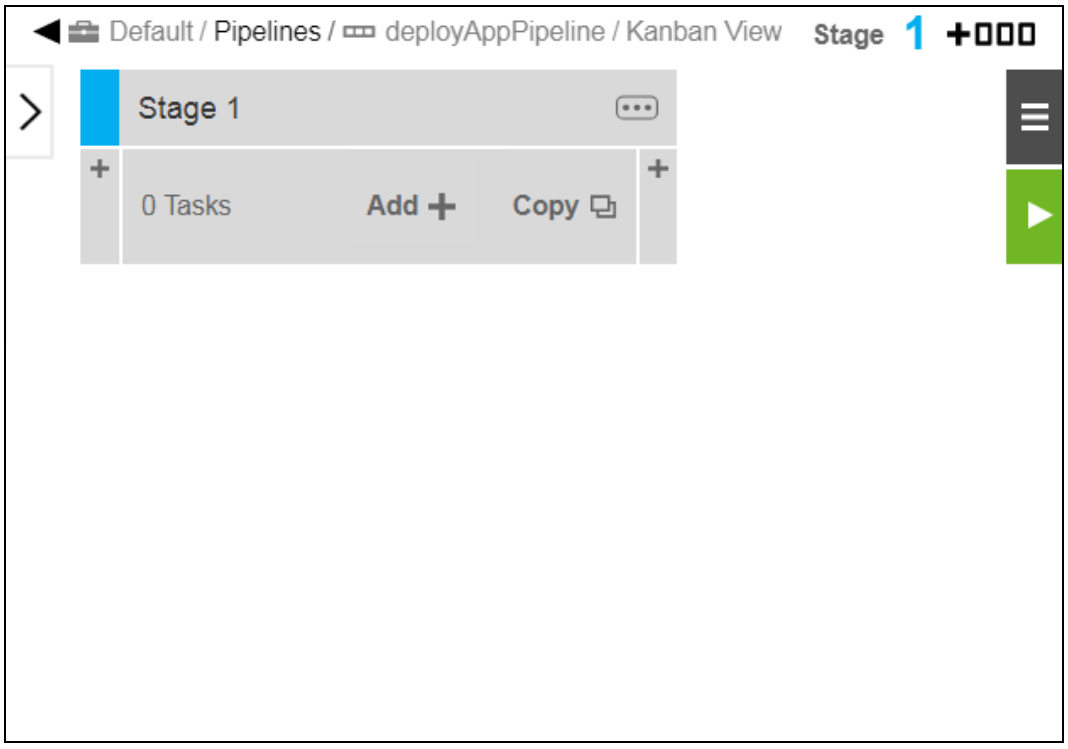


6. Click **OK**.

   The Pipeline Editor appears. It shows a pipeline with one stage. For example, in Pipeline Stage View:



   For example, in Release Kanban View:



Creating a Pipeline by Copying an Existing Pipeline

1. Open the home page of the ElectricFlow web UI by browsing to https://<*ElectricFlow_server*>/flow/.
2. Go to the pipelines list by either:
   - Clicking **Pipelines**.
   - Clicking the **Main menu** button in the upper left corner, and then selecting **Pipelines** > **All Pipelines**.

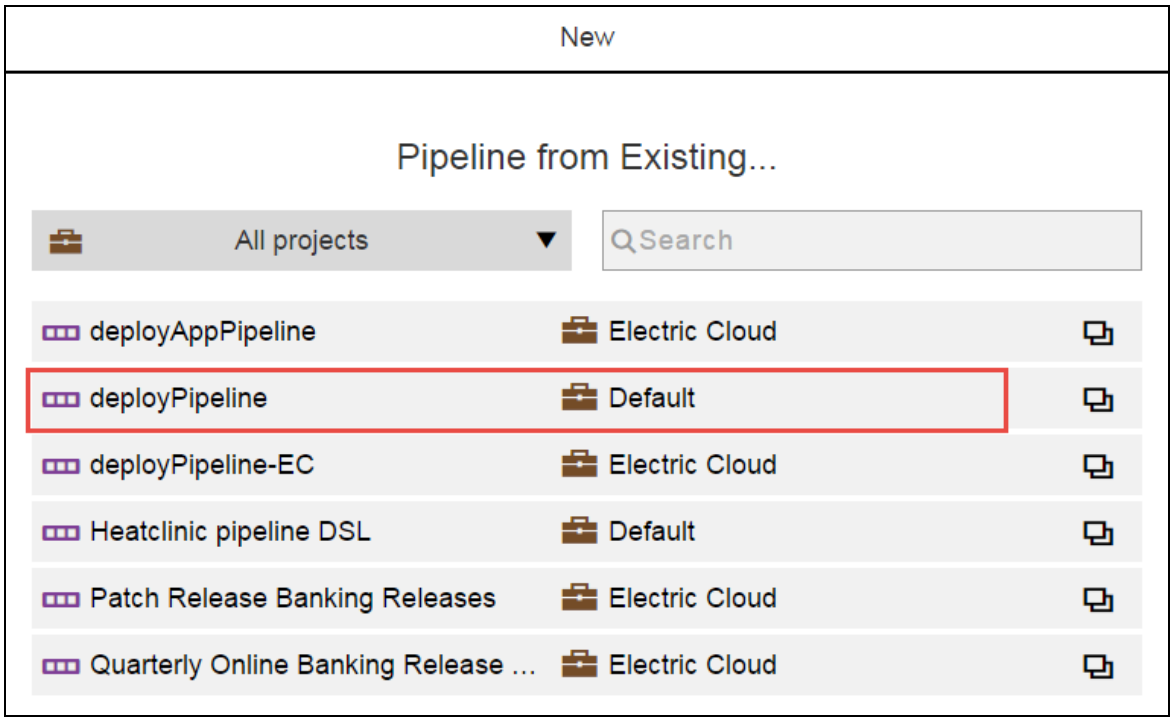
   The pipelines list opens.
3. Click the  (New Pipeline) button.

4. Click **Copy Existing**.

   The list of existing pipelines opens.

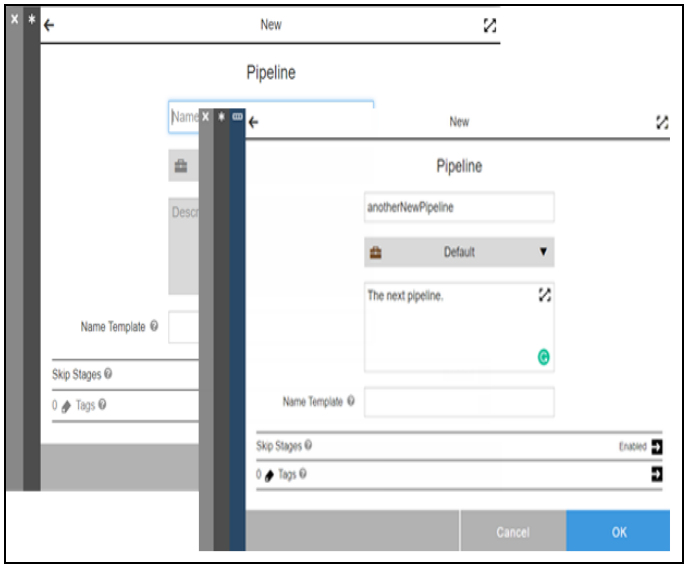
   It shows the list of available pipelines. The default is to show the pipelines for all projects. To see only the pipelines for a specific project, click the down arrow in the **All projects** field to select one or more projects in one of these ways:Click on the name of one or more projects.Enter the search criteria in the Search field. The projects that match the criteria appear in the list.You can also search for one or more pipelines by entering the search criteria in the **Search** field next to the **All projects** field. If there are no matches, a message appears stating that there are no pipelines in the selected projects.

   For example:



5. Select a pipeline to copy.

6. Enter the pipeline name, select a project where the applications in the pipeline will be deployed, and enter a description. Optionally, configure stage skipping and tags.

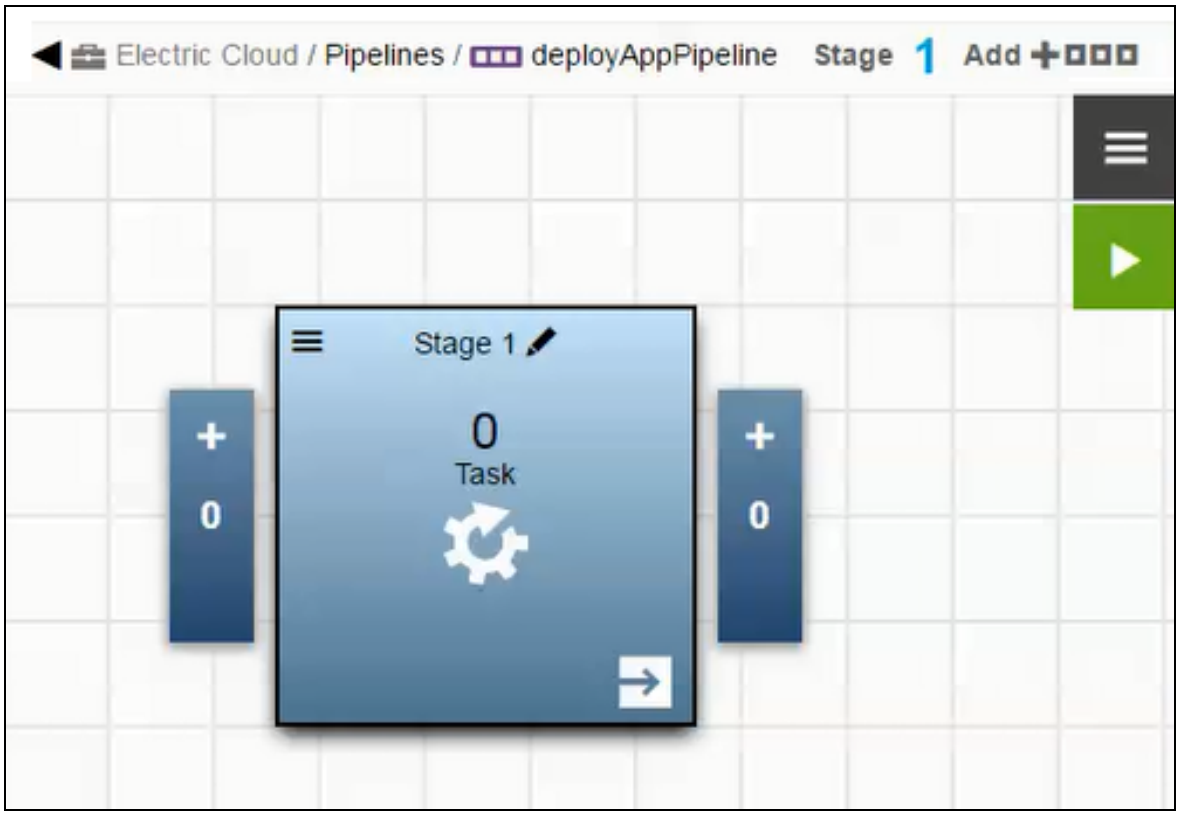


7. Click **OK**.

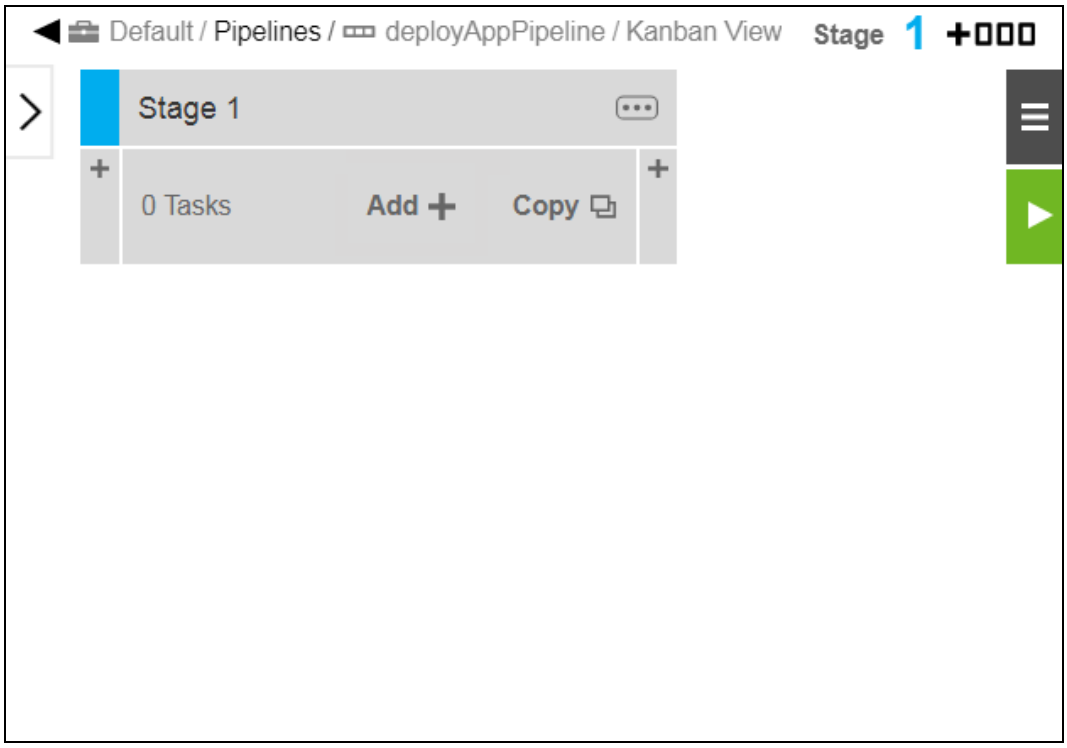
   The Pipeline Editor appears and shows a pipeline with the same stages, tasks, and gates as the existing pipeline.

Editing Pipeline Stage Details

When you create a pipeline from scratch, it adds a stage automatically. For example, in Pipeline Stage View:

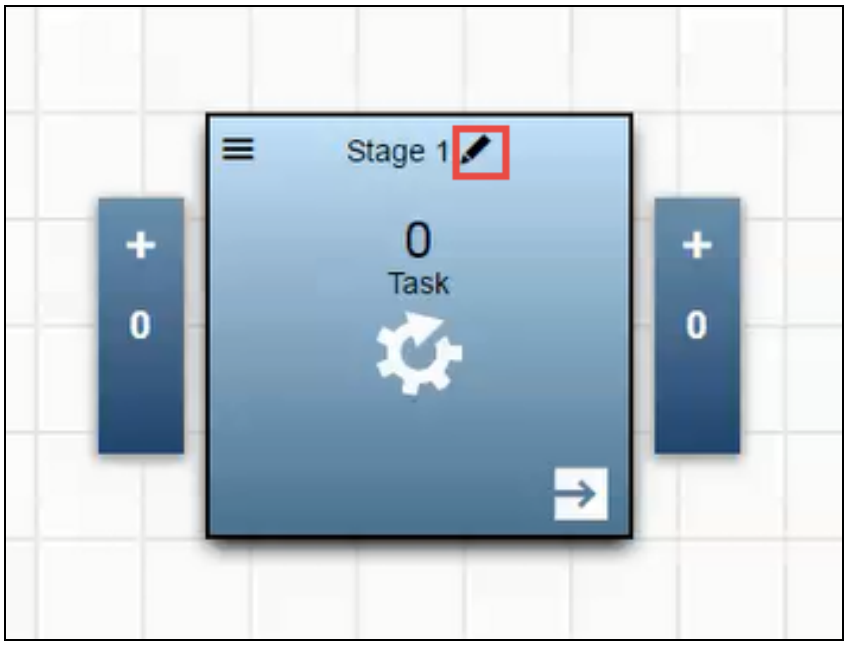


For example, in Release Kanban View:



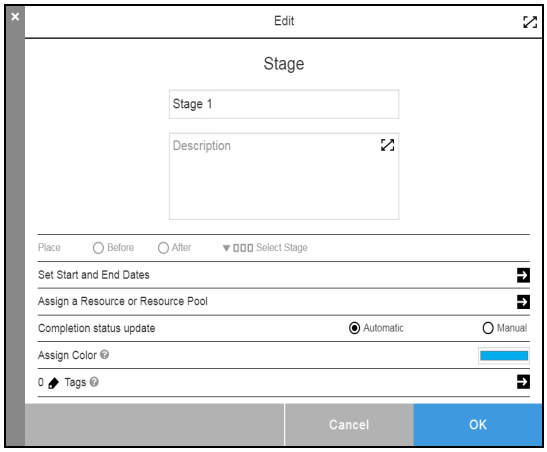
You can change the stage name and add a description of the stage. To do so:

1. Open the **Pipeline Stage Details** dialog in either of the following ways.
    - In Pipeline Stage View, click the following button:



- In Release Kanban View, click the (Actions) button and choose **Details**:

2. Enter information about the stage. the stage name and description.



where:

Place: The position of this stage with respect to other stages in the pipeline.

Set Start and End Dates: The start and end dates and times for this stage.

Assign a Resource or Resource Pool: Resource where this stage executes. See Setting Up Resources on page 1031 for details.

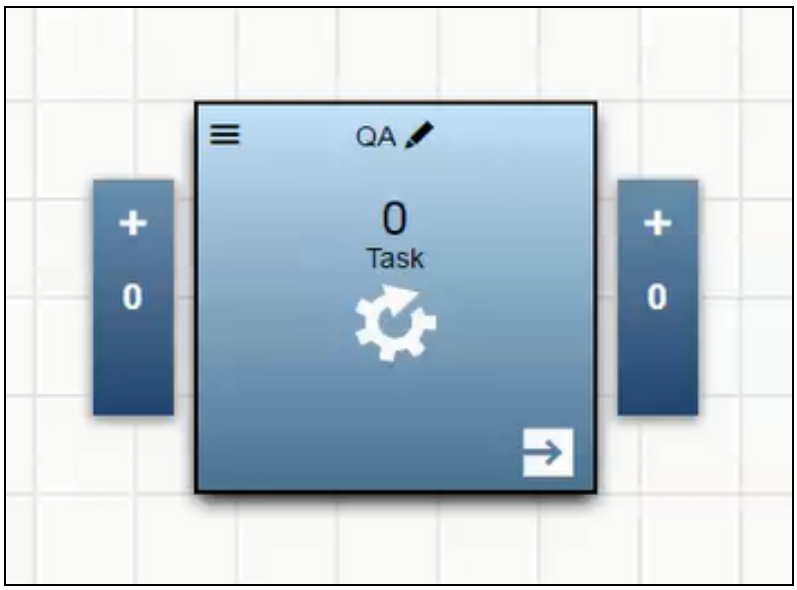
Completion status update: The manner in which the stage is marked as complete. See Release Concepts on page 622 for more details.

Assign Color: The color in which a stage appears in the Planned vs Actual view. See Release Dashboard on page 669 for details.

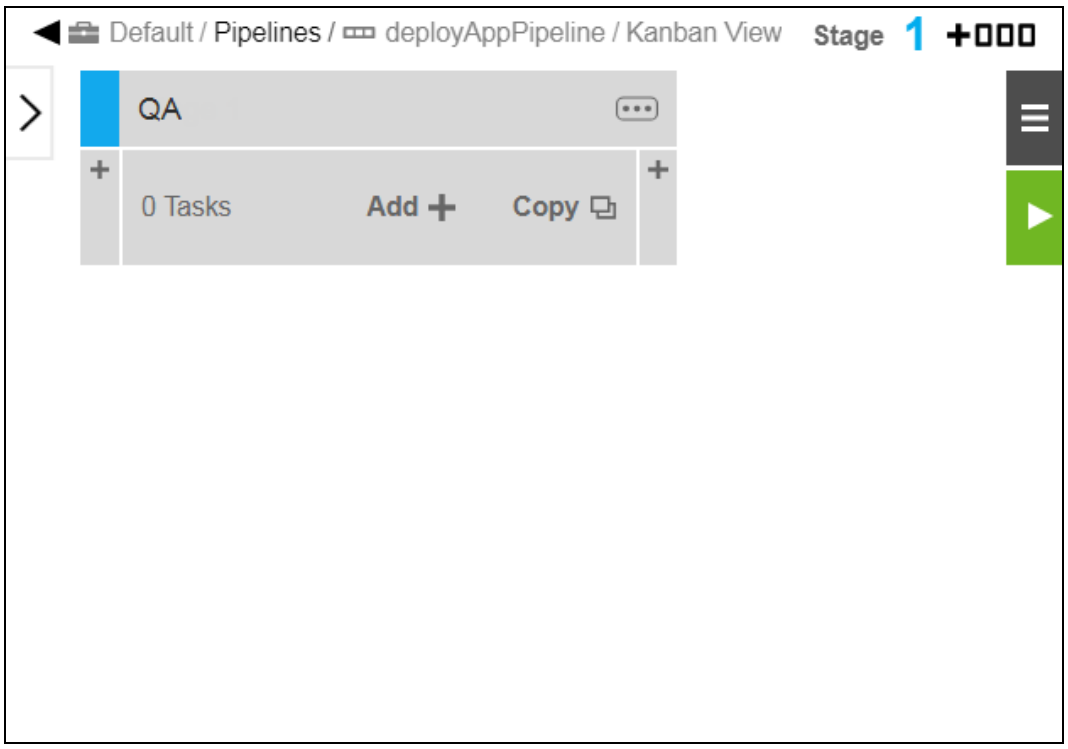
Tags: Add one or more tags to this stage object. See Object Tags on page 45 for details.

3. Click **OK**.

   The updated stage name appears. For example, in the Pipeline Stage View:



   For example, in the Release Kanban View:

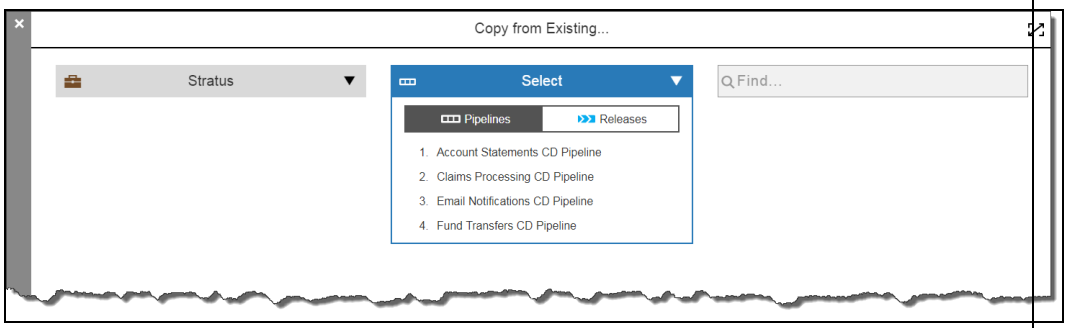


Defining the Tasks in a Pipeline Stage Using the Pipeline Stage View

This section describes how to define the tasks in a stage. You can add any number of tasks, reorder tasks, run tasks in parallel, delete tasks, and so on.

Parts of a Task List in a Pipeline Stage

Following are the key parts of a task list:

1	Breadcrumbs showing the path to the pipeline.
2	Name of the stage. When you click the down arrow, a list of the stages in the pipeline opens. You can select a stage to go to a different stage and view its task list.
3	• Click the **Select** button to select **All** or **None** of the items in the list. • Click the **Delete** button to delete the selected items. • Click the **Group** button to put multiple tasks into a group. • Click the **Copy** button to create a task by copying an existing task. The **Copy from Existing** dialog box appears, which lets you select an existing task to copy: 
4	Number of tasks in the stage. Click the  (Add Task) button to add a task to the stage.

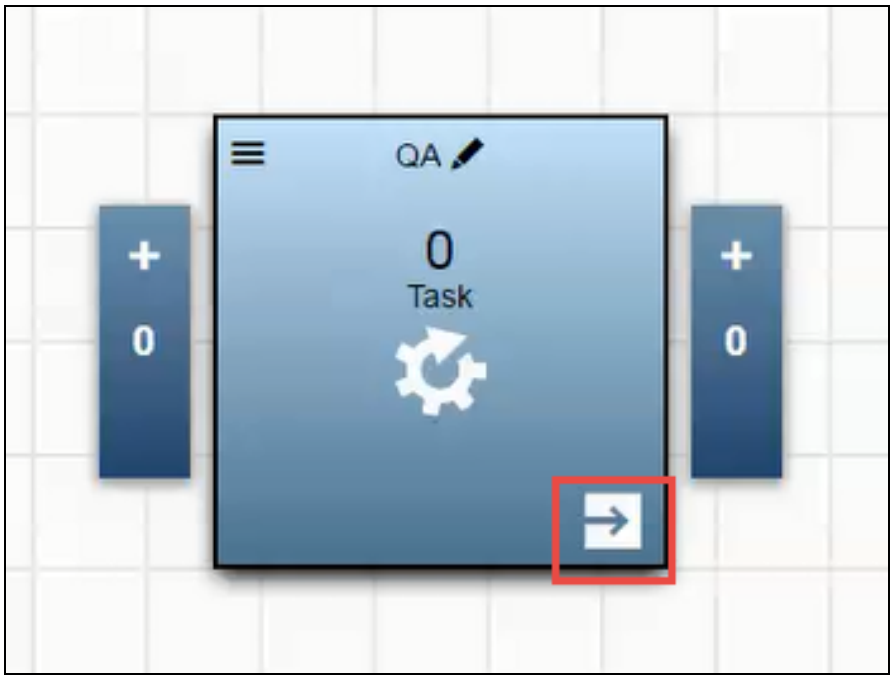
5	Button to open the menu to see details about the stage or to take action on the stage.
6	Task definition consisting of the task defined by an application process. This example consists of a task named QA-Stage-Deploy and an application process called Deploy, which is part of the ShoppingCart application that is mapped to the QA environment. Clicking the  (Actions) menu button lets you see details about the task and take actions on the task.
7	Task definition consisting of the task defined by a procedure. In this example, it consists of a task named QA-Stage-TakeSnapshot and a procedure named TakeSnapshot in the Utilities project. Clicking the project button shows the project to which the procedure belongs.  Clicking the **Actions** menu button lets you see details about the task and take actions on the task.
8	Determines whether to skip the task at run time. This lets you skip tasks that were executed successfully in the prior run. This check box is checked by default.
9	Determines whether to always run a task regardless of the error handling behavior for any preceding tasks. Enabling this option guarantees that the task runs before the pipeline execution ends. However, if one or more tasks in a parallel task group with Stop on Error enabled fails, all tasks in the same group are aborted (including tasks with Always Run enabled).

10	Determines the retry type. You can choose one of the following types: •  —If the task fails, the pipeline continues to the next gate or stage task. •  — If the task fails, the pipeline aborts. This is the default behavior. This behavior is usually used in Continuous Delivery pipelines. •  —An approver or assignee reviews the error and retries, skips, or fails the task. •  —The task is automatically retried at specific time intervals for a specific number of times, after which you can specify whether the pipeline stops or continues.

Adding Pipeline Tasks

Starting in the Pipeline Editor:

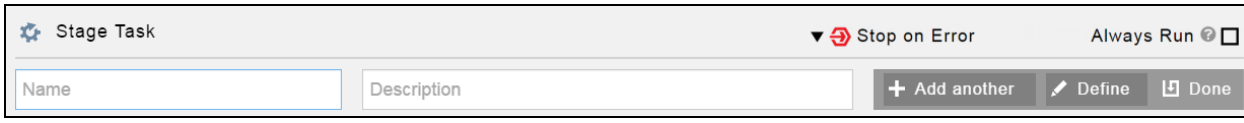
1. Click the  button in the lower right corner of a stage:



The task list opens. When the task list has one or more items, it shows the tasks in sequential order for the stage.

2. If the stage has no tasks, do the following in an empty row:
   a. Enter the name of the task in the **Name** field.
   b. (Optional) Enter a description for the task in the **Description** field.
   c. Select the **On Error** condition.

      When the stage task fails, the pipeline either continues running and proceeds to the next task (**Continue On Error**), stops running and aborts (**Stop On Error**), asks an approver to review the error and retry, skip, or fail the task (**Manual Retry on Error**), or retries the task automatically a specific number of times (**Automated Retry on Error**). The default is **Stop On Error**.



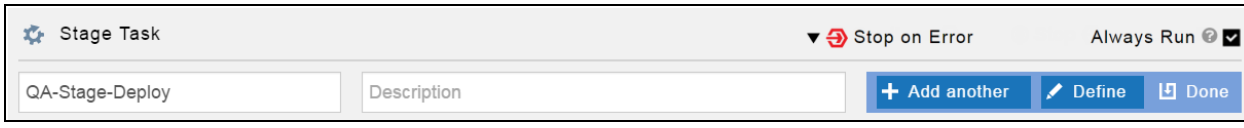
      Enter **QA-Stage-Deploy** in the **Name** field.

   d. Select the **Always Run** condition.

      Toggle this option when you want to guarantee that the task runs before the pipeline execution ends regardless of the error handling behavior for any preceding tasks. This gives you more control in the pipeline. **Always Run** is disabled by default.

      However, if one or more tasks in a parallel task group with **Stop on Error** enabled fails, all tasks in the same group are aborted (including tasks with **Always Run** enabled).

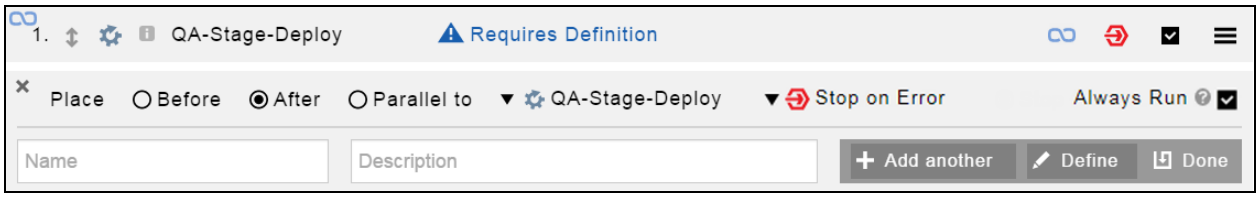
      Enter **QA-Stage-Deploy** in the **Name** field.



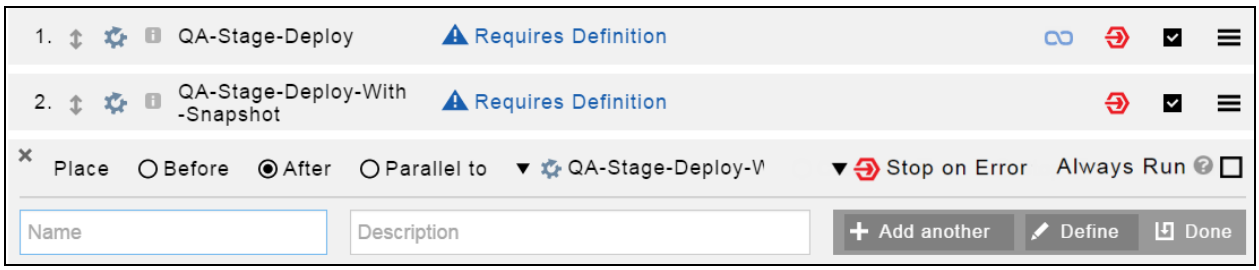
   e. Click **+ Add another**.

      The task you just added is saved, and a new task is automatically added after it.

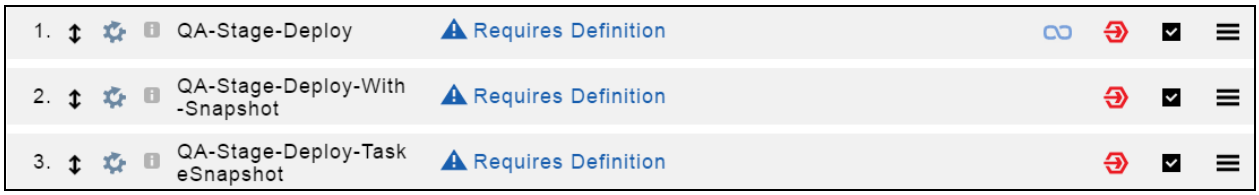
3. Enter the name of the new task in the **Name** field.
4. (Optional) Enter a description for the task in the **Description** field.
5. Set the position of the task.
    a. Select **Before** or **After**.
    b. Click the down arrow to select the task that the new task will precede or follow.
6. Select the **On Error** condition as described above.



7. If you want to add another task, click **+ Add another** and repeat the previous three steps to add it. Otherwise, click **Done**.



The list shows the tasks in the order that they will be executed.



Defining New Pipeline Tasks

1. Click **Requires Definition** for the task that you created.

    The dialog box to define the task opens. For more information about the task types, see Pipeline Tasks on page 419.

2. Enter the settings to define the task, and click **OK**.

   Example:

   a. Select **Application Process** to define the task as an application process.

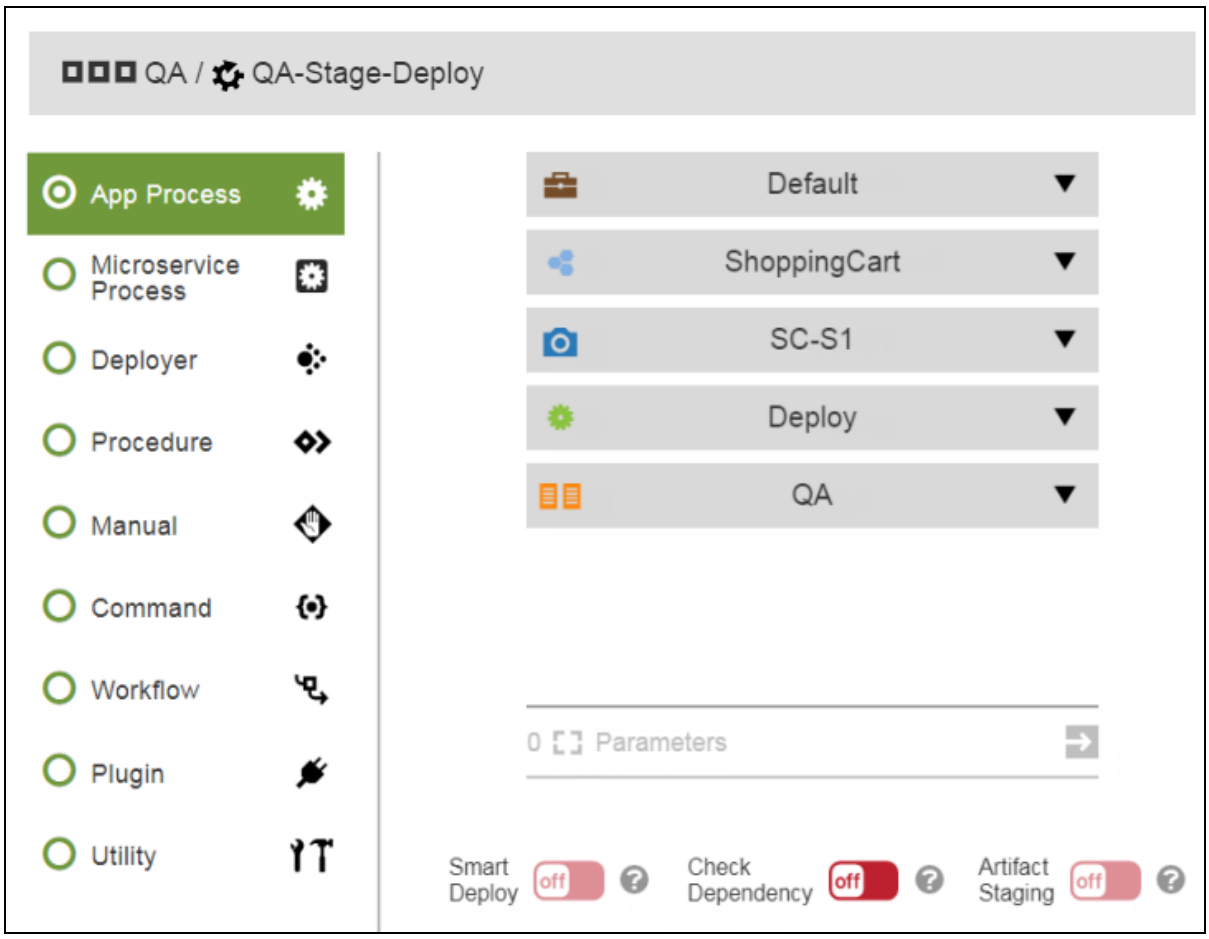
   b. In the **Select Project** field, select the **Default** project.

   c. Select the **ShoppingCart** application, and then select **Use Snapshot**

      When you click in the **Select a Snapshot** field, a **Search** field opens where you can enter the criteria to search for a snapshot as well as the names of existing snapshots.

   d. Select **SC-S1** as the snapshot to use.

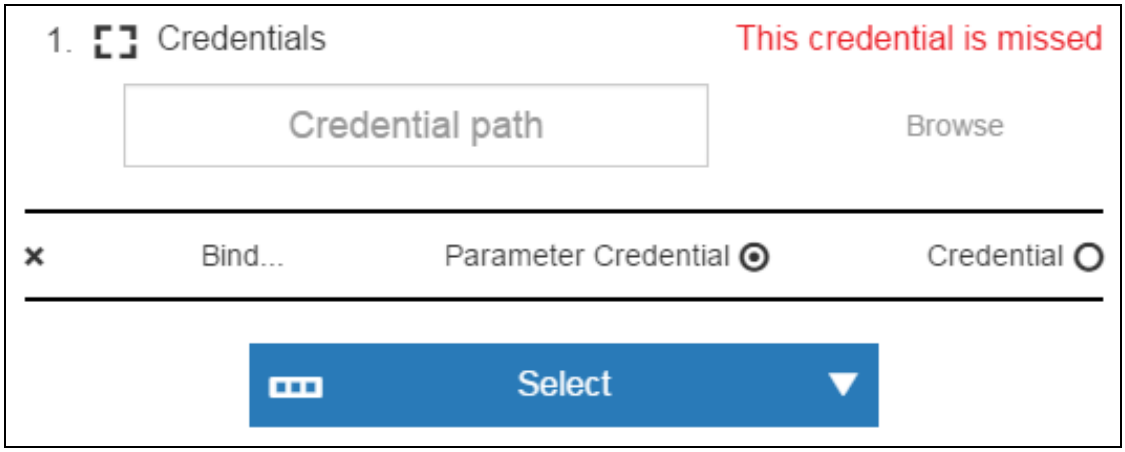
   e. Select the **Deploy** process and the **QA** environment where application will be deployed.



Go to Pipeline Tasks on page 419 for more information about the different ways to define a stage task.

f. The **Parameters** field and the **Output Parameters** field are disabled, because the **Deploy** process does not have any required parameters.

If the process has a required parameter for the pipeline run, such as a credential parameter, a message with the number of required parameters (such as **1 Required**) appears in the **Parameters** field. You can enter the path to the credential, browse to it, select the **Parameter Credential** or **Credential** binding, or select a user-defined credential that is attached to the project associated with the pipeline.



3. Click **Define** for the second task in the list, define the task, and click **OK**.

   The dialog box to define the task opens.

   Example:

   a. Select **Command** to define the task as a command or script to run.

      This command or script is passed to the task's shell for execution.

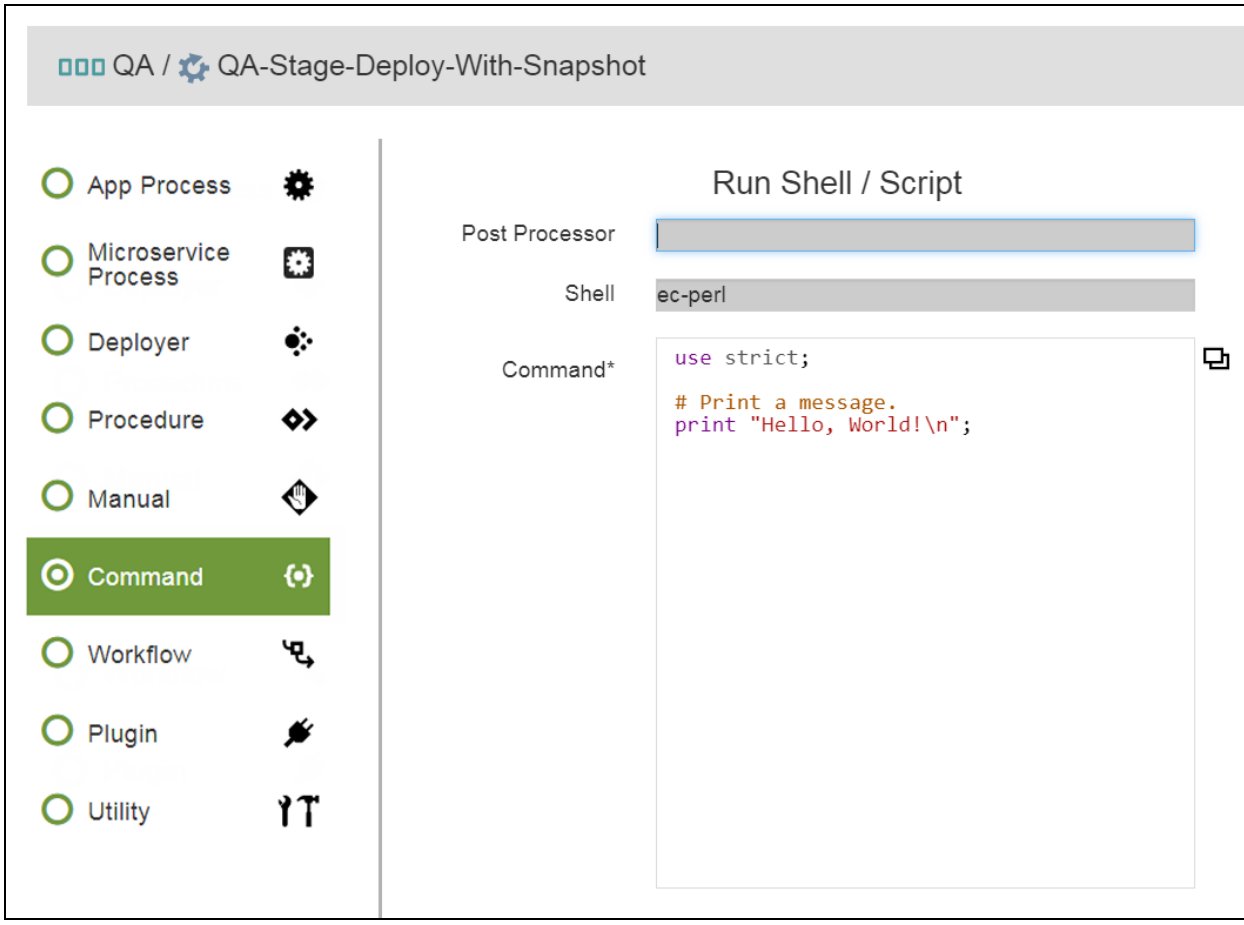
   b. (Optional) In the **Post Processor** field, enter the name of the postprocessor that you want to use.

      This field specifies a command (passed to the task's shell for execution) that analyzes the log file for the step and collects diagnostic information for reporting. If this field is blank, no postprocessor runs for the step. The default packaged postprocessor simply requires you to enter "postp," but for details on modifications to this default or other aspects of using a postprocessor, see Postprocessors.

   c. (Optional) In the **Shell** field, enter the shell to be used to execute the commands.

      For example, `ec-perl`. If you do not specify a shell on a task, at task run-time the server looks at the resource shell. If a resource shell is not defined, the shell line used by the agent is platform dependent:
      Windows: `cmd /q /c "{0}.cmd"`
      Linux or UNIX: `sh -e "{0}.cmd"`

d. In the **Command** field, enter the command or script to execute.



You can also click the (command code editor) button to invoke a text editor for entering the command or script.

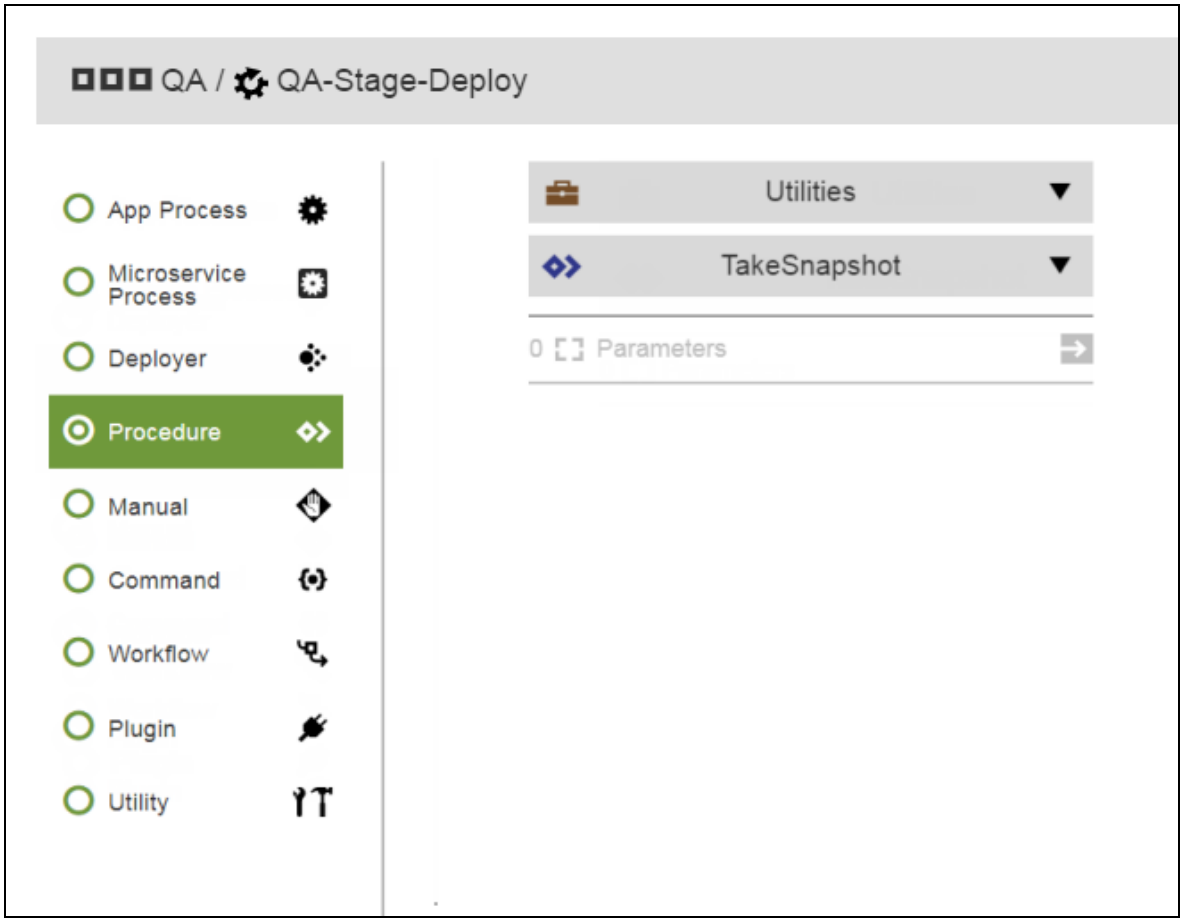
4. Click **Requires Definition** for the third task in the list, define the task, and click **OK**.

   Example:

   a. Select **Procedure**.

   b. When you click in the **Select Project** field, a drop-down list of available projects opens. Select the **Utilities** project.

   c. When you click in the **Select Procedure** field, a drop-down list of available procedures opens. Select the **TakeSnapshot** procedure.

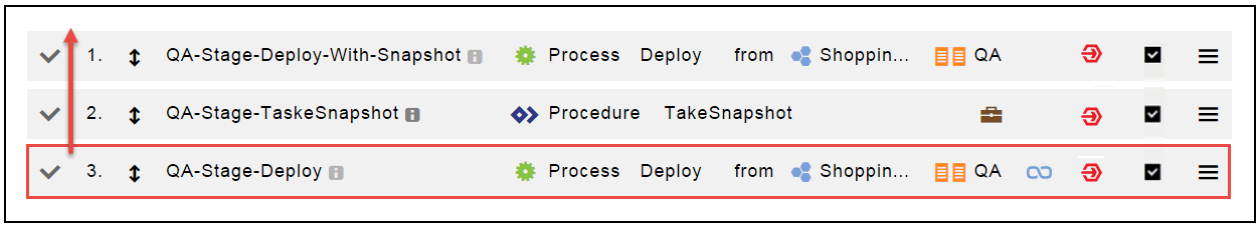


   If the **TakeSnapshot** procedure has required parameters for the pipeline run, the **Parameter** field is enabled. Go here for more information.

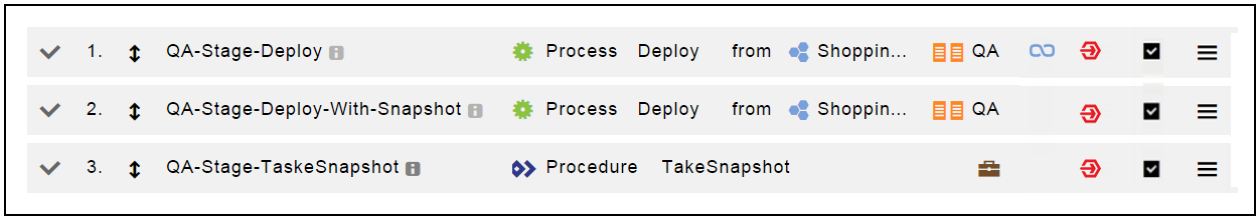
Changing the Order in Which Pipeline Tasks Are Executed

This section describes how to change the order in which the tasks are executed. Starting in the task list:

1. Choose the task that you want to move.
2. Click and drag the (up/down) button next to the task name to move the task to its new position.



The task is now in the first in the list.



You can also click the button on any task and click **Change Order** to reorder it.

Grouping Pipeline Tasks for Running in Parallel or Serial Order

By default, tasks are executed in order. You can group tasks without dependencies and run them in parallel to reduce runtimes and make a stage more efficient. You can also group tasks and run them in serial order.

To make a set of sequential tasks run in parallel:

1. Click to highlight at least two tasks that you want to run in parallel.

2. Click the Group button.

A parallel tasks group is created, which contains the tasks that will now run in parallel.

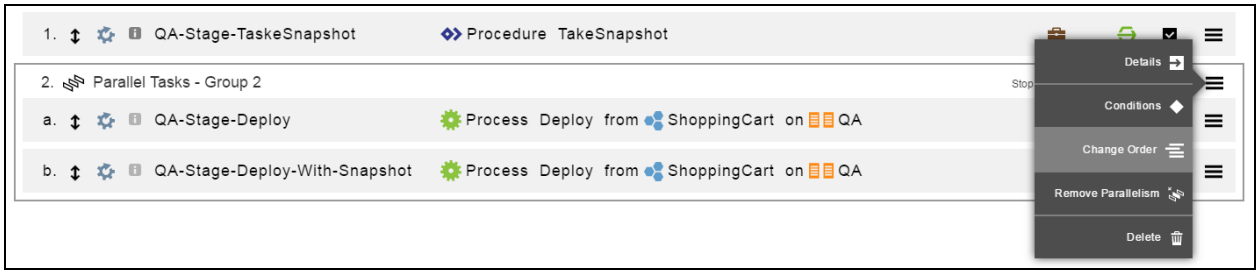
In this case, step 2 will not begin until both steps inside the parallel group have completed.

Tip: You can add new tasks to an existing parallel tasks group. To do so, drag the (up/down) button next to the task name to move the task to the group. You can rename a parallel tasks group by clicking the button for the group and then clicking **Details** to fill in a new name.

Changing the Order of Pipeline Tasks that Include a Group of Parallel Tasks

You can change the execution order of tasks that include a group of parallel tasks. To do so:

1. Click the button for the group and choose **Change Order**.



2. Click the either the **Before** or **After** radio button as appropriate, then click the pipeline rules ( ) menu, then click another task, and then click **Done.**

The tasks are now reordered.

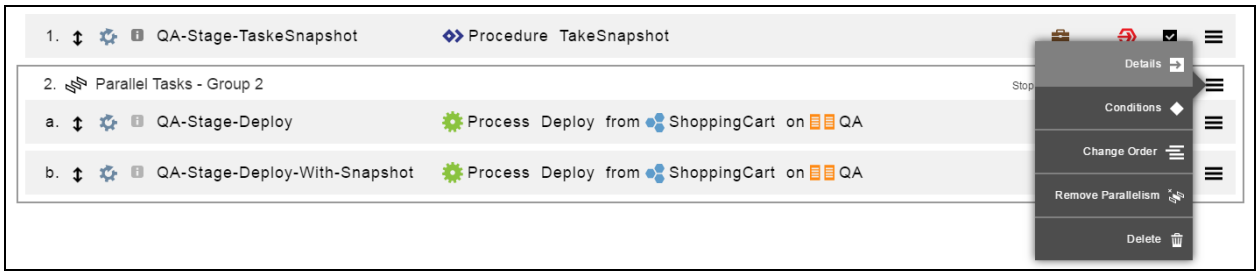
Tip: You can also click and drag the (up/down) button next to a group to reorder it.

Setting the Error or Rejection Handling for a Parallel Task Group

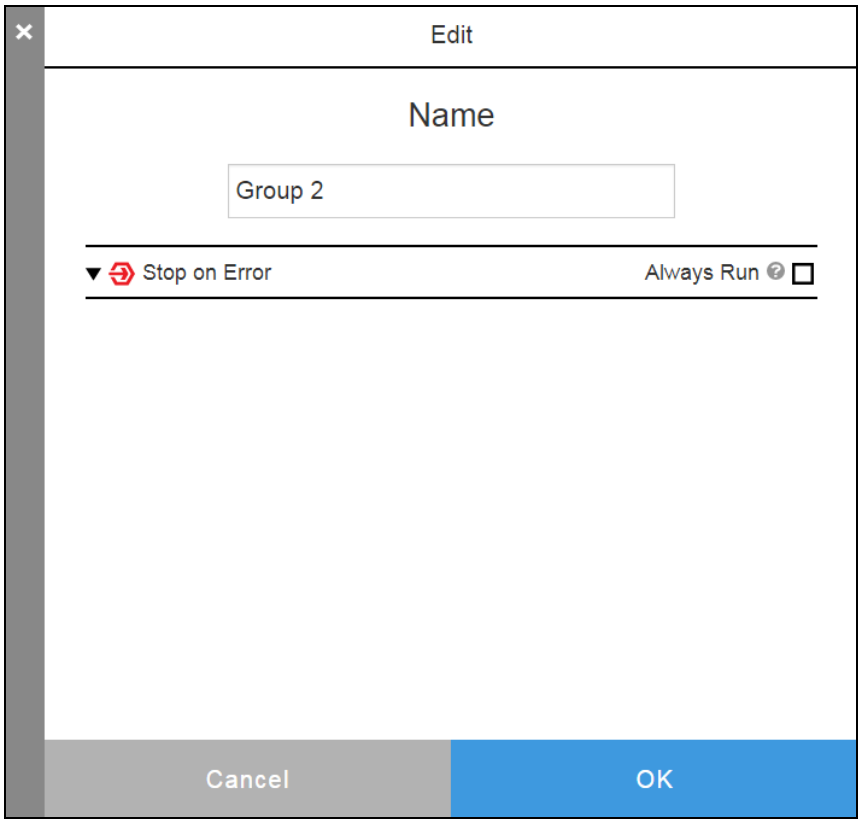
When defining a group of parallel tasks, you can specify whether the pipeline continues or stops if any task in the group fails because of an error or rejection. By default, if any task fails or is rejected, the pipeline aborts. This behavior is usually used in Continuous Delivery pipelines.

To set the error or rejection handling:

1. Click the button for the group and then click **Details**.



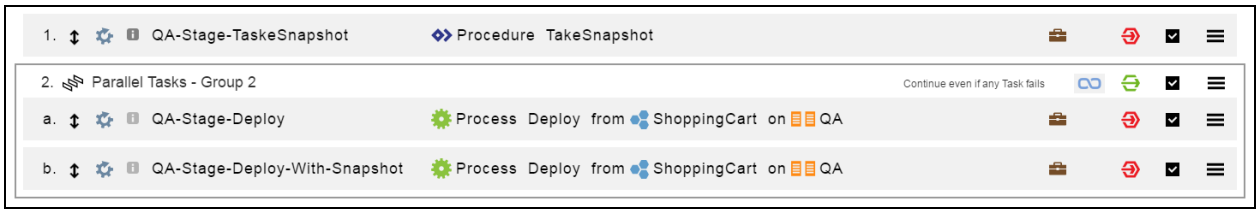
The **Name** dialog box appears.



2. Select one of the following options:
   - **Continue on Error**: If any task fails or is rejected, the pipeline will still continue, allowing any other parallel tasks to complete before moving onto the next stage task or gate.
   - **Stop on Error**: If any task fails or is rejected, the pipeline aborts.
3. Select **Always Run**. Enabling this option guarantees that the tasks will run before the pipeline execution ends. Note that if this option is selected, and any individual task in the group that has error handling set to **Stop on Error** fails, then all the running tasks will be aborted.

4. Click **OK.**

   If, for example, you changed from **Stop on Error** (the default) to **Continue on Error**, the Error or Rejection icon for the group changes to  .



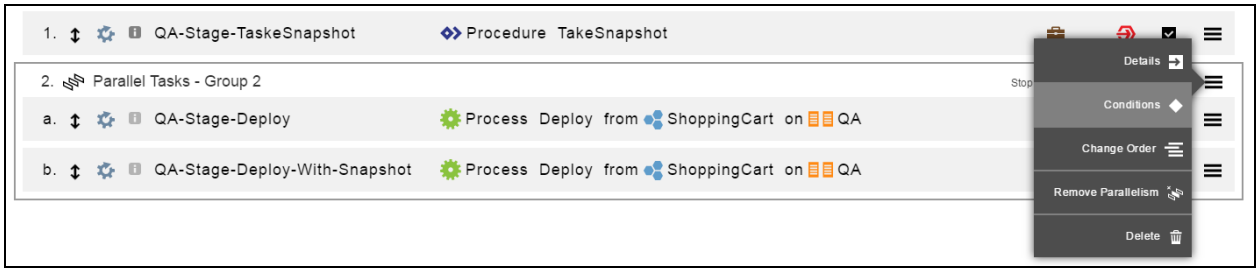
   If, for example, you changed from **Continue on Error** to **Stop on Error**, the Error or Rejection icon for the group changes to  .

Adding "Run if" and "Wait until" Conditions to a Parallel Task Group

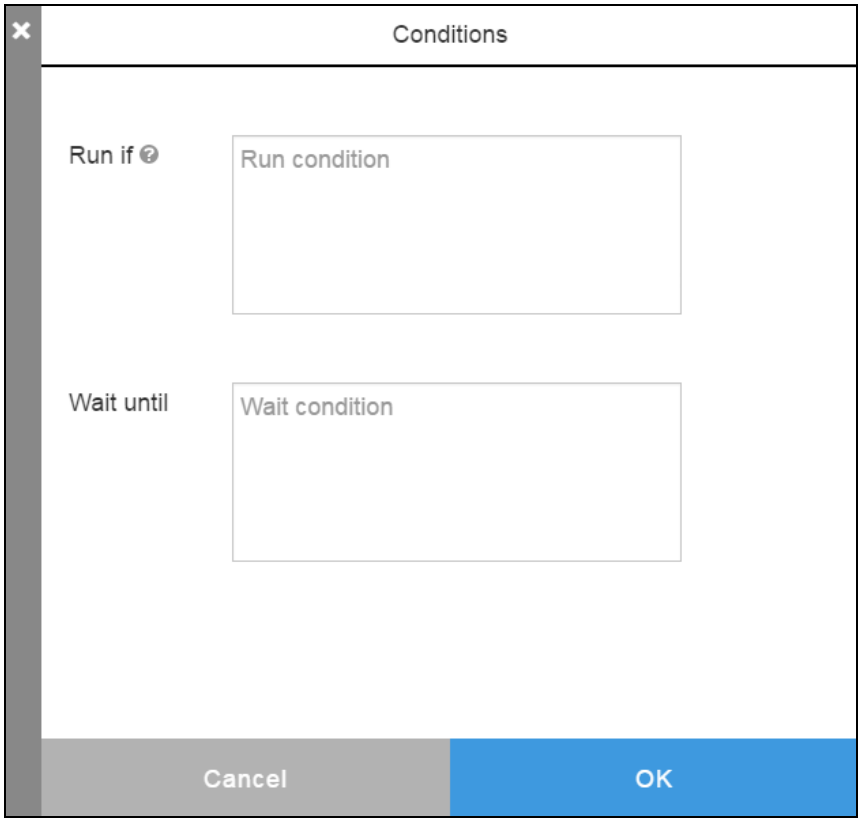
For a parallel task group, you can specify "Run if" and "Wait until" conditions to control the group behavior based on a precondition of your choice. Skipping a group causes all of its tasks to be skipped.

You can specify preconditions by entering a JavaScript expression. To do so:

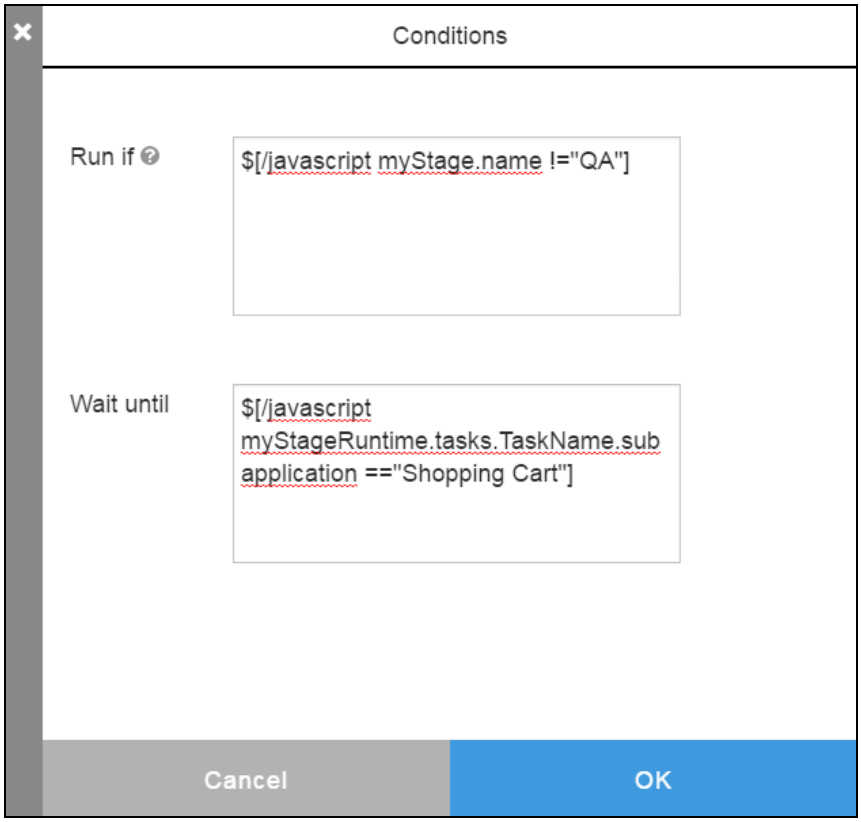
1. Click the  button for the group and then click **Conditions**.



The **Conditions** page appears.



2. Enter a JavaScript expression into the **Run if** field or the **Wait until** field (or both).



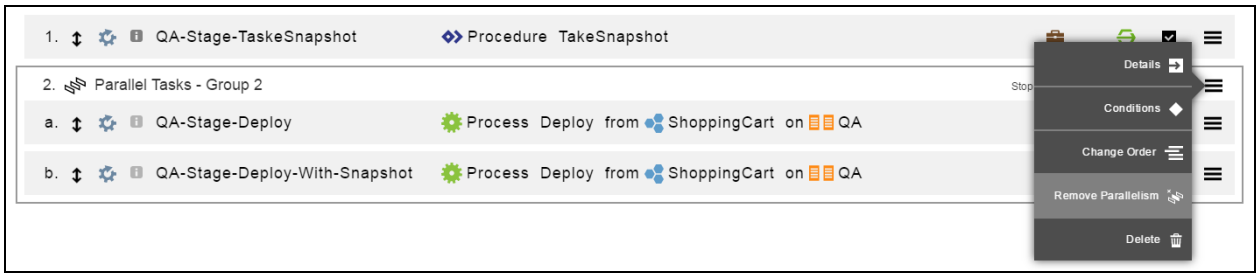
   For more examples of Javascript expressions for "Run if" and "Wait until" conditions, see the *KBEC-00360 - Using Context-Relative Shortcuts to Properties on Pipelines and related objects* KB article.

3. Click **OK**.

Unparallelizing Pipeline Tasks

You can unparallelize tasks, which means returning parallel tasks in a group to sequential execution. To do so:

1. Click the menu button for the group and then click **Remove Parallelism**.



A confirmation popup appears:

2. Click **OK**.

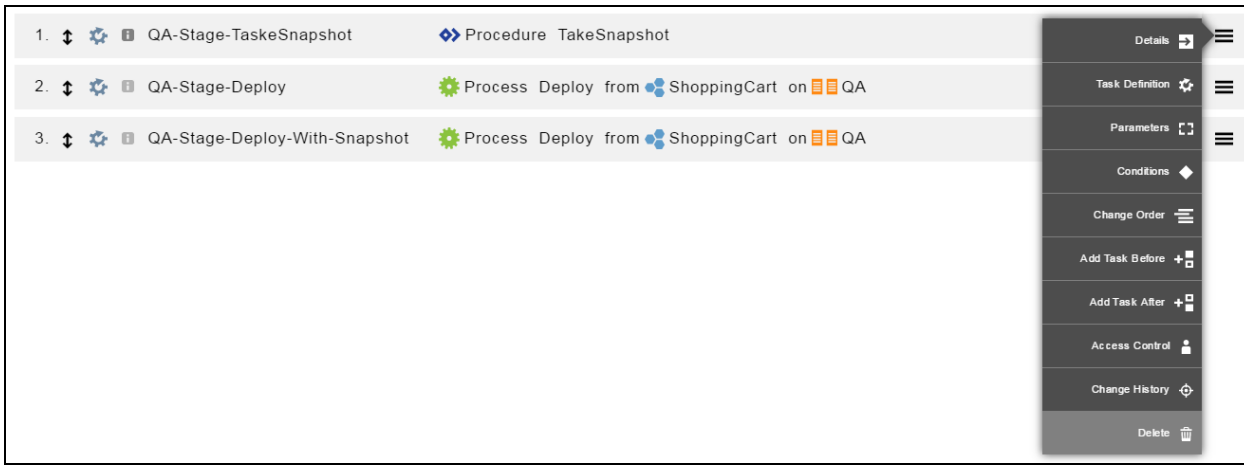
   The tasks will once again run in sequence.

Deleting a Pipeline Task from a Stage

You can delete a pipeline task from a stage. To do so:

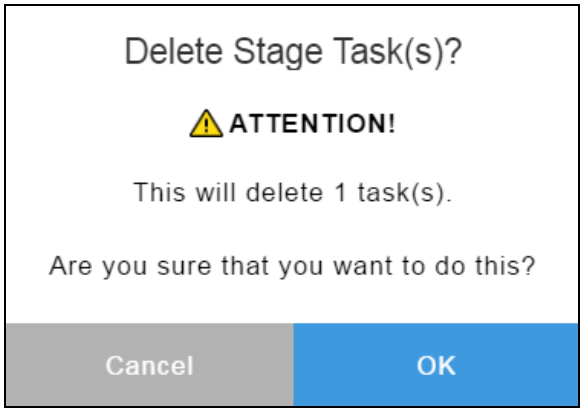
1. Do one of the following:

   - Click the menu button for the task, and then click **Delete**.



   - Click to highlight the task and then click the Delete button.

   A confirmation popup appears:



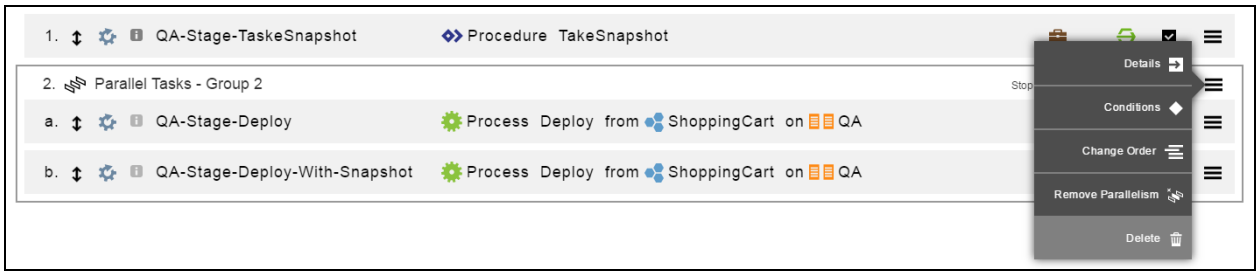
2. Click **OK**.

> **Tip:** You can also use this method to delete a parallel task group and all of its tasks.

Deleting a Parallel Task Group and All of Its Tasks from a Stage

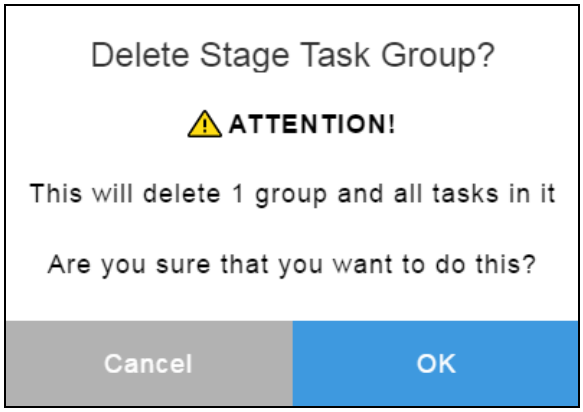
You can delete a parallel task group from a stage. This also deletes all of its tasks. To do so:

1. Click the menu for the parallel task group and then click **Delete**.



(You could also click to highlight the group and then click the Delete button.)

A confirmation popup appears:



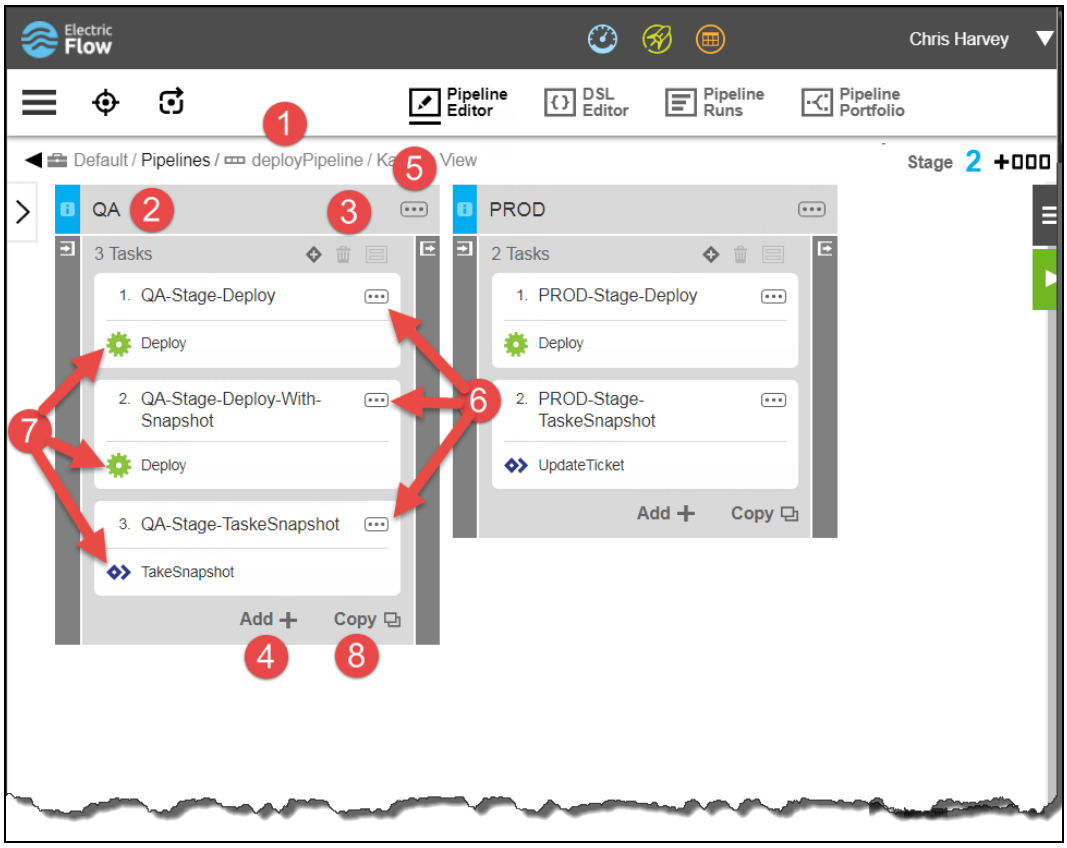
2. Click **OK**.

Defining the Tasks in a Pipeline Stage Using the Release Kanban View

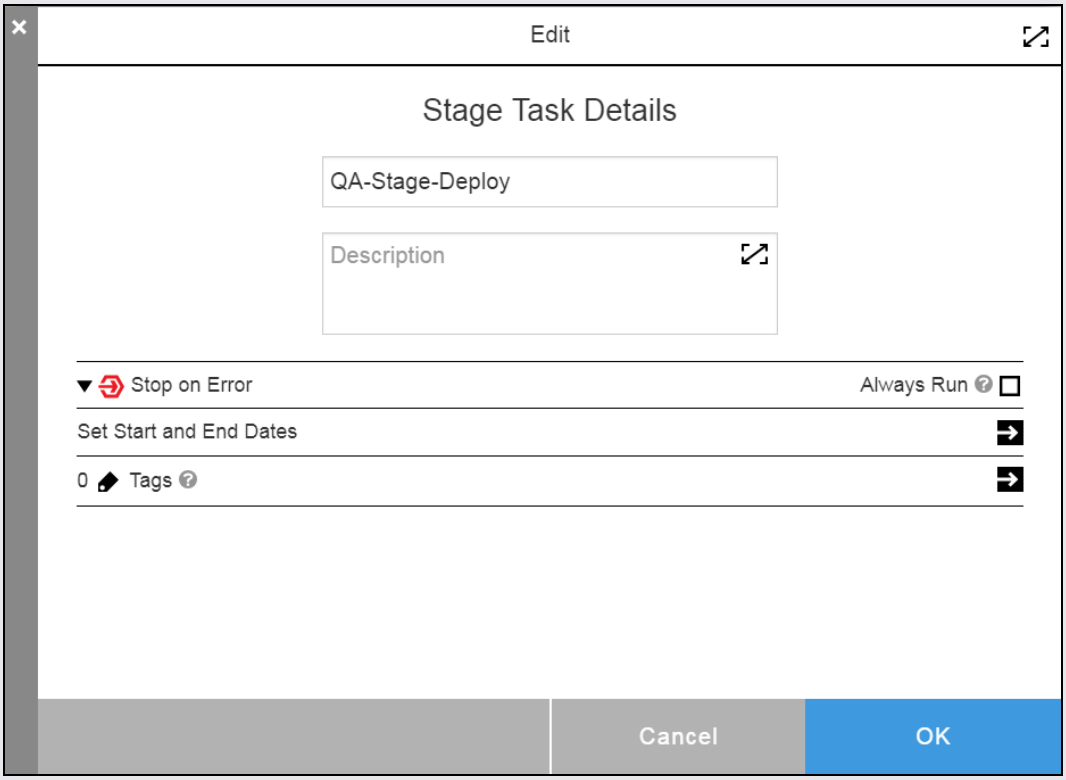
This section describes how to define the tasks in a stage. You can add any number of tasks, reorder tasks, run tasks in parallel, delete tasks, and so on.

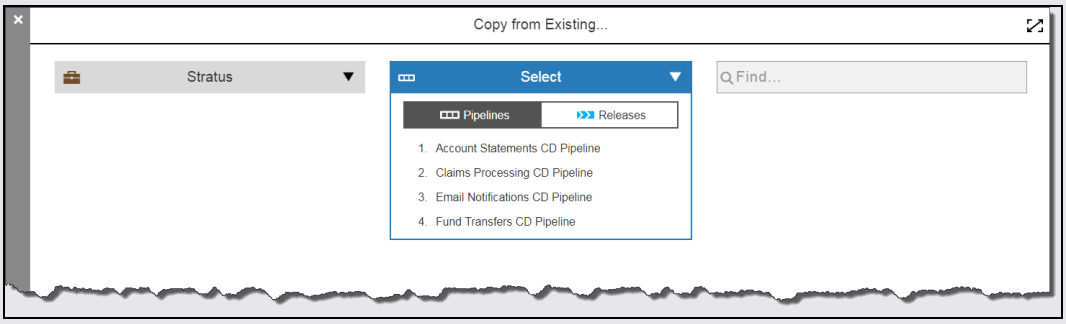
Parts of a Task List in a Pipeline Stage

Following are the key parts of a task list:



1	Breadcrumbs showing the path to the pipeline.
2	Name of the stage.
3	• Click the **+** (Stage Conditions) button to add a stage condition. • Click the **Delete** button to delete the selected items. • Click the **Group** button to put multiple tasks into a group.
4	Click the **Add +** button to add a task to the stage.
5	Clicking the (Actions) menu button lets you see details about the stage or to take action on the stage.

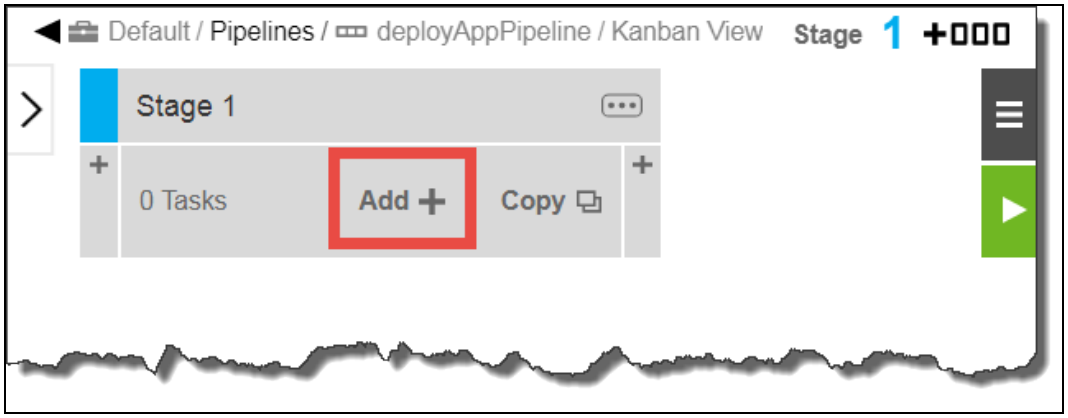
6	Clicking the (Actions) menu button lets you see details about the task or take action on the task.  **Stop on Error** pulldown menu—determines the error handling. You can choose one of the following types: • Continue on Error—If the task fails, the pipeline continues to the next gate or stage task. • Stop on Error— If the task fails, the pipeline aborts. This is the default behavior. This behavior is usually used in Continuous Delivery pipelines. • Manual Retry on Error—An approver or assignee reviews the error and retries, skips, or fails the task.

	• Automated Retry on Error—The task is automatically retried at specific time intervals for a specific number of times, after which you can specify whether the pipeline stops or continues. **Always Run** checkbox—determines whether to always run a task regardless of the error handling behavior for any preceding tasks. Enabling this option guarantees that the task runs before the pipeline execution ends. However, if one or more tasks in a parallel task group with Stop on Error enabled fails, all tasks in the same group are aborted (including tasks with Always Run enabled). **Set Start and End Dates** button—opens a dialog box to let you set task start and end dates, task duration, and "wait for start date" to specify whether you want this task to execute before the planned start date. **Tags** button—opens a dialgo box to let you tag this object or to modify or delete a tag. Tags let you group related objects by a user-defined term. For details, see Object Tags on page 45.
7	Icon signifying the type of task such as an Application Process task or a Procedure task.
8	Click the **Copy** button to create a task by copying an existing task. The **Copy from Existing** dialog box appears, which lets you select an existing task to copy: 

Adding Pipeline Tasks

Starting in the Pipeline Editor:

1. Click the **Add+** button:



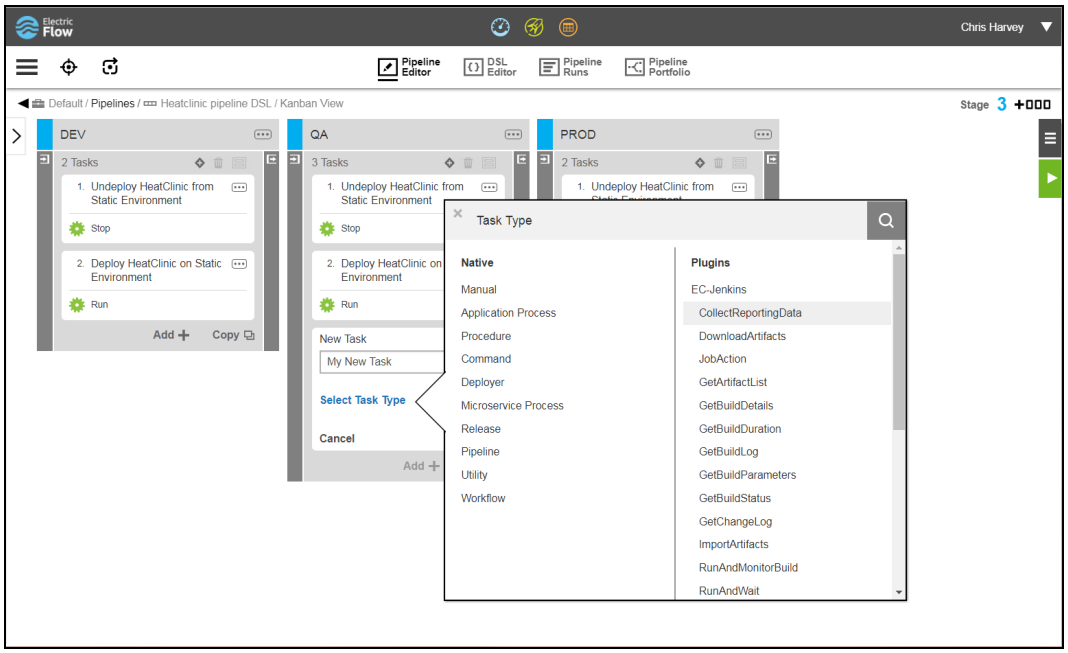
The task list opens. When the task list has one or more items, it shows the tasks in sequential order for the stage.

2. Enter the name of the new task in the **New Task** field.

3. Click **Select Task Type**.

   The **Task Type** dialog box appears, which contains the lists of task types from which to select:

   - **Native**—List of standard task types such as Command or Application Process. Because the Release Kanban view lets you postpone the configuration of some details, you must add those details later to completely define the task by clicking its **Config** button once the task appears in the Kanban View. For example, if you select the Procedure task, you must then click its **Config** button to choose the actual procedure to invoke.
   - **Integrations**—List of plugins that are available for selection. When you select a plugin, you must also drill down to select one of its procedures to run. For example, the EC-Jenkins plugin and its CollectReportingData procedure:



   Notice that plugin task creation is easier than in the Pipeline Stage View. This is because you can select the plugin directly from the Kanban View.

   By default, the top ten plugins appear in this list. You can see the complete list of plugins by clicking **Other...**.

   The ec_preferred_integrations property sheet determines the plugins to appear in this list and their order. To modify this list, open the Automation Platform at https://<*ElectricFlow_server*>/flow/, then select **Administration** > **Server** > **ec_deploy** > **ec_preferred_integrations**, and then edit the list.

4. Click the task type.

   The dialog box closes, and the undefined task type appears in the task.

5. If you want to add another task, click **Add +** and repeat the previous steps to add it.

   The list shows the tasks in the order that they will be executed.

Defining New Pipeline Tasks

1. Click **Define** for the first task in the list.

   The dialog box to define the task opens. For more information about the task types, see Pipeline Tasks on page 419.

2. Enter the settings to define the task, and click **OK**.

   Example:

   a. Select **Application Process** to define the task as an application process.

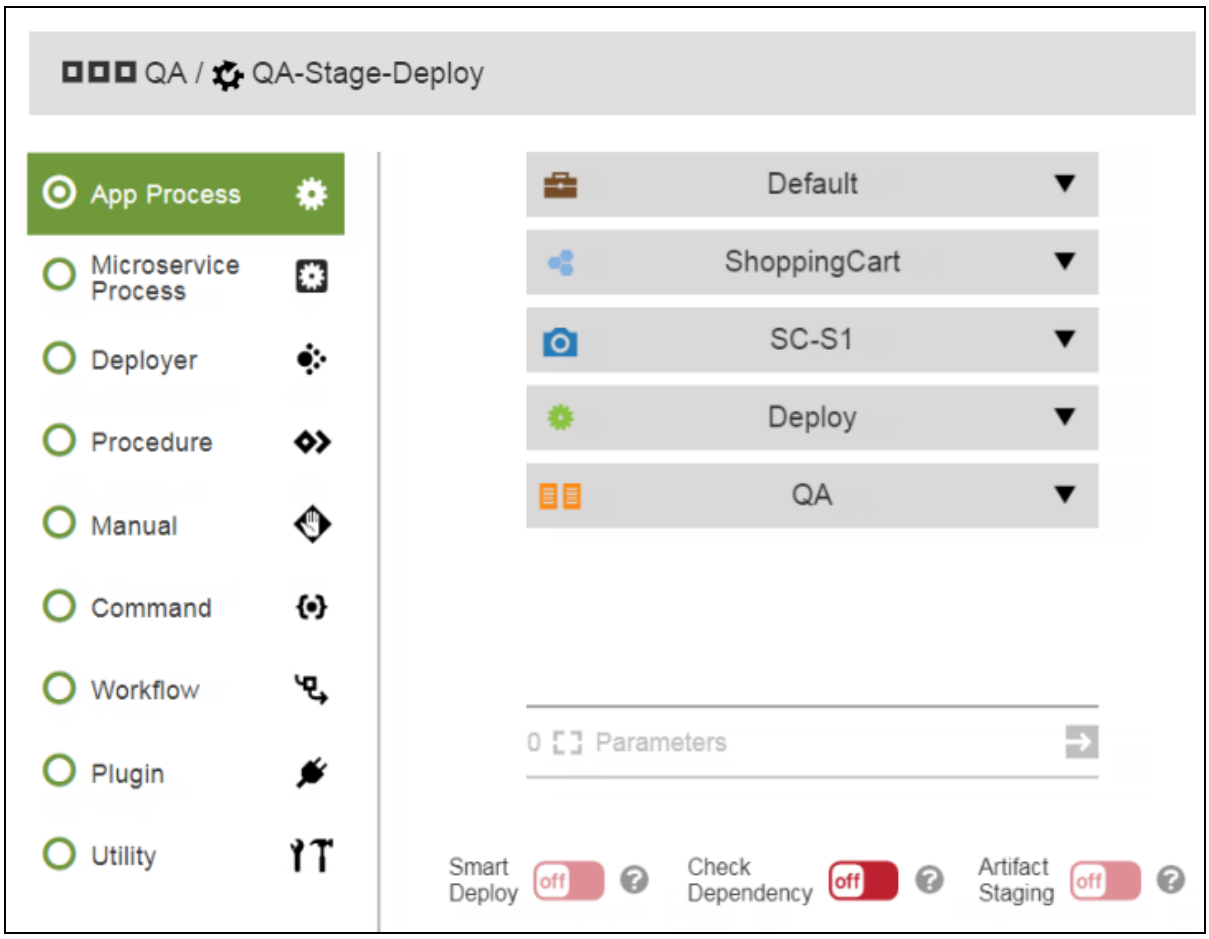
   b. In the **Select Project** field, select the **Default** project.

   c. Select the **ShoppingCart** application, and then select **Use Snapshot**

      When you click in the **Select a Snapshot** field, a **Search** field opens where you can enter the criteria to search for a snapshot as well as the names of existing snapshots.

   d. Select **SC-S1** as the snapshot to use.

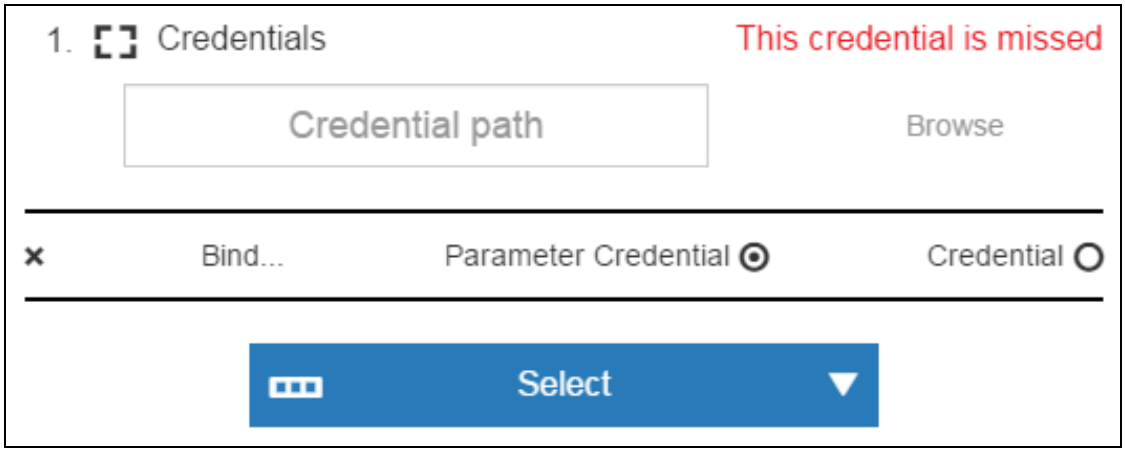
   e. Select the **Deploy** process and the **QA** environment where application will be deployed.



Go to Pipeline Tasks on page 419 for more information about the different ways to define a stage task.

f. The **Parameters** field is disabled because the **Deploy** process does not have any required parameters.

If the process has a required parameter for the pipeline run, such as a credential parameter, a message with the number of required parameters (such as **1 Required**) appears in the **Parameters** field. You can enter the path to the credential, browse to it, select the **Parameter Credential** or **Credential** binding, or select a user-defined credential that is attached to the project associated with the pipeline.



3. Click **Requires Definition** for the second task in the list, define the task, and click **OK**.

   The dialog box to define the task opens.

   Example:

   a. Select **Command** to define the task as a command or script to run.

      This command or script is passed to the task's shell for execution.

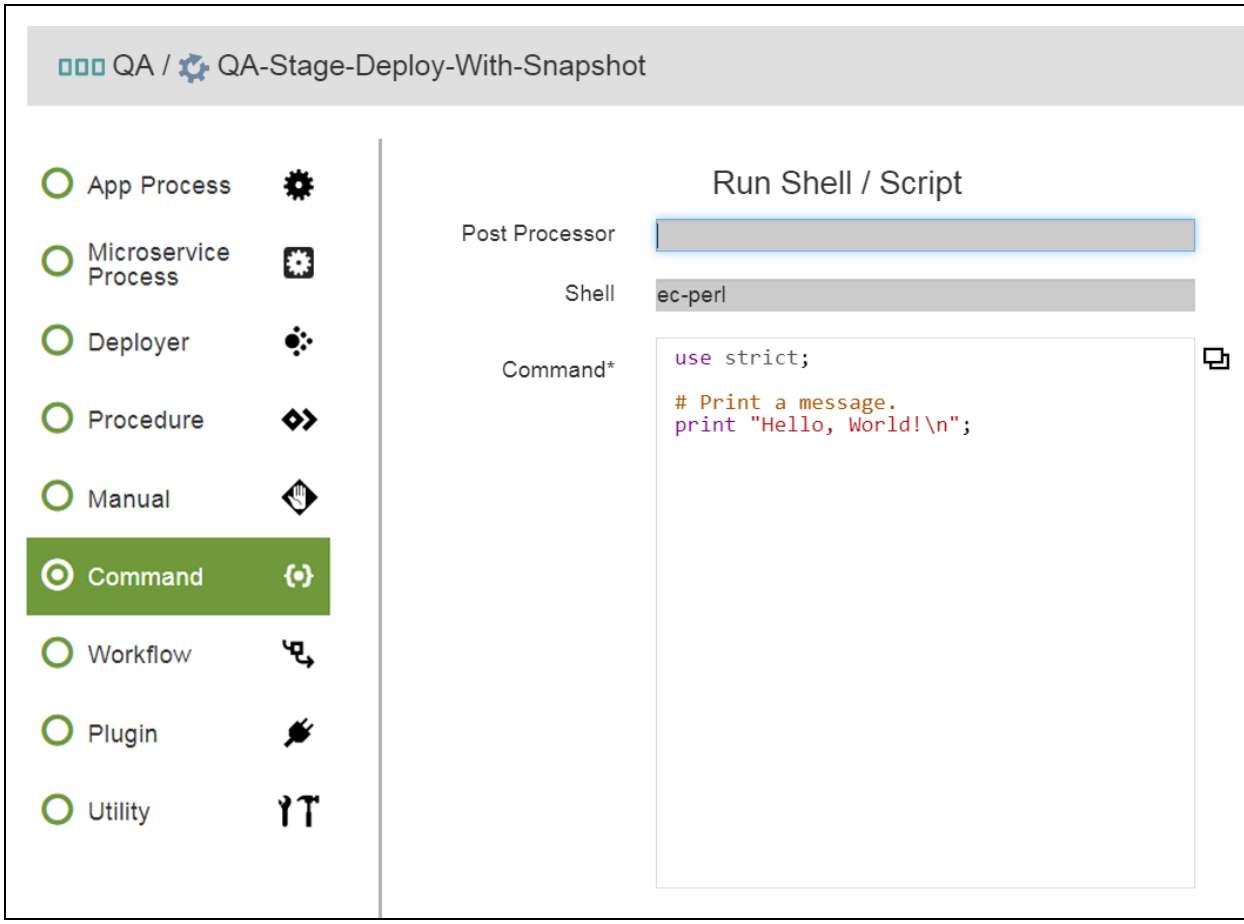
   b. (Optional) In the **Post Processor** field, enter the name of the postprocessor that you want to use.

      This field specifies a command (passed to the task's shell for execution) that analyzes the log file for the step and collects diagnostic information for reporting. If this field is blank, no postprocessor runs for the step. The default packaged postprocessor simply requires you to enter "postp," but for details on modifications to this default or other aspects of using a postprocessor, see Postprocessors.

   c. (Optional) In the **Shell** field, enter the shell to be used to execute the commands.

      For example, `ec-perl`. If you do not specify a shell on a task, at task run-time the server looks at the resource shell. If a resource shell is not set, the shell line used by the agent is platform dependent:
      Windows: `cmd /q /c "{0}.cmd"`
      Linux or UNIX: `sh -e "{0}.cmd"`

d. In the **Command** field, enter the command or script to execute.



You can also click the (command code editor) button to invoke a text editor for entering the command or script.

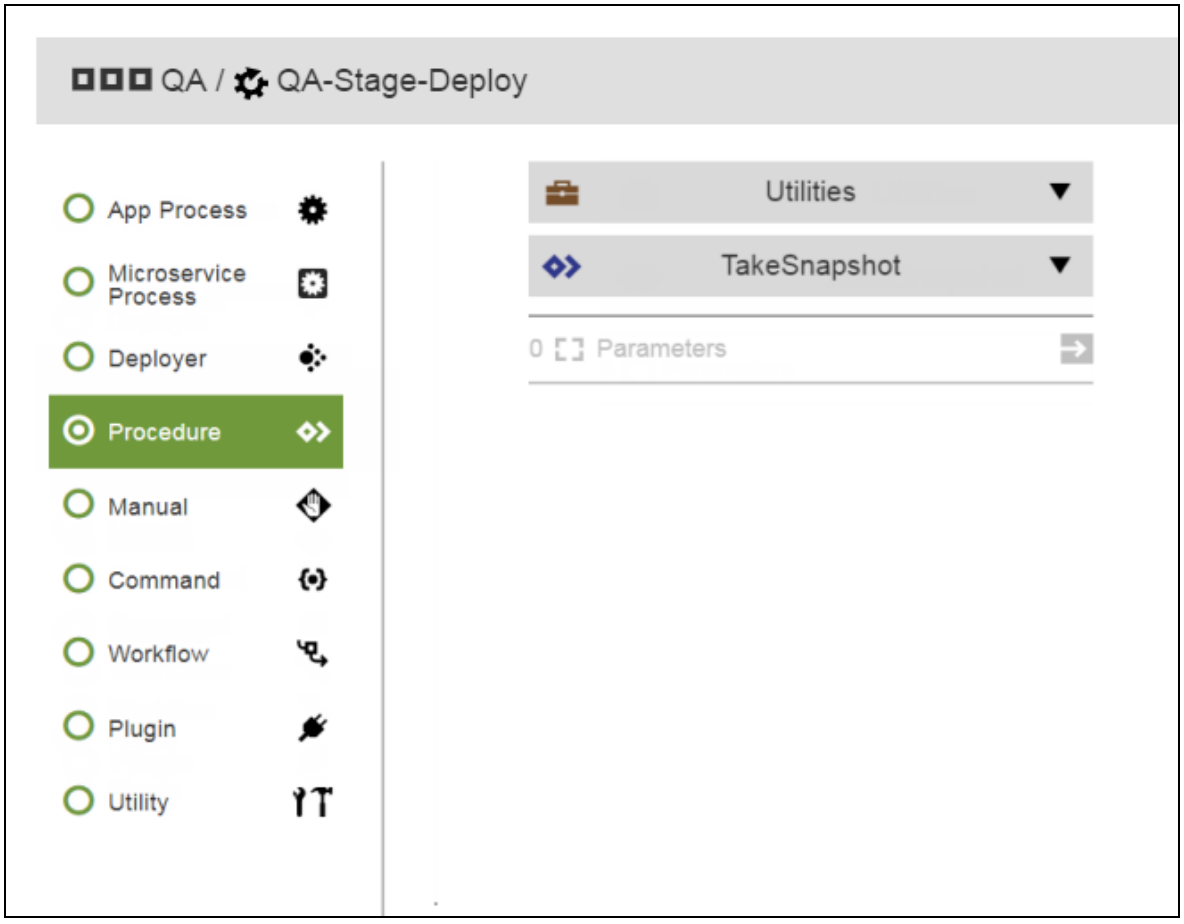
4. Click **Requires Definition** for the third task in the list, define the task, and click **OK**.

   Example:

   a. Select **Procedure**.

   b. When you click in the **Select Project** field, a drop-down list of available projects opens. Select the **Utilities** project.

   c. When you click in the **Select Procedure** field, a drop-down list of available procedures opens. Select the **TakeSnapshot** procedure.



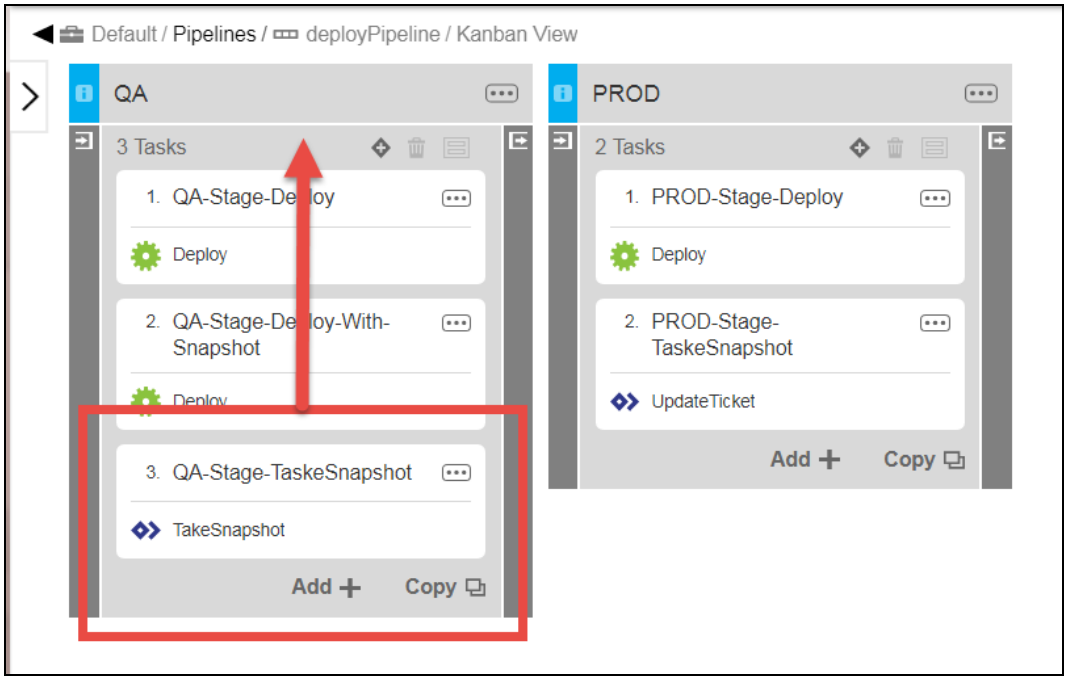
   If the **TakeSnapshot** procedure has required parameters for the pipeline run, the **Parameter** field is enabled. Go here for more information.

Changing the Order in Which Pipeline Tasks Are Executed

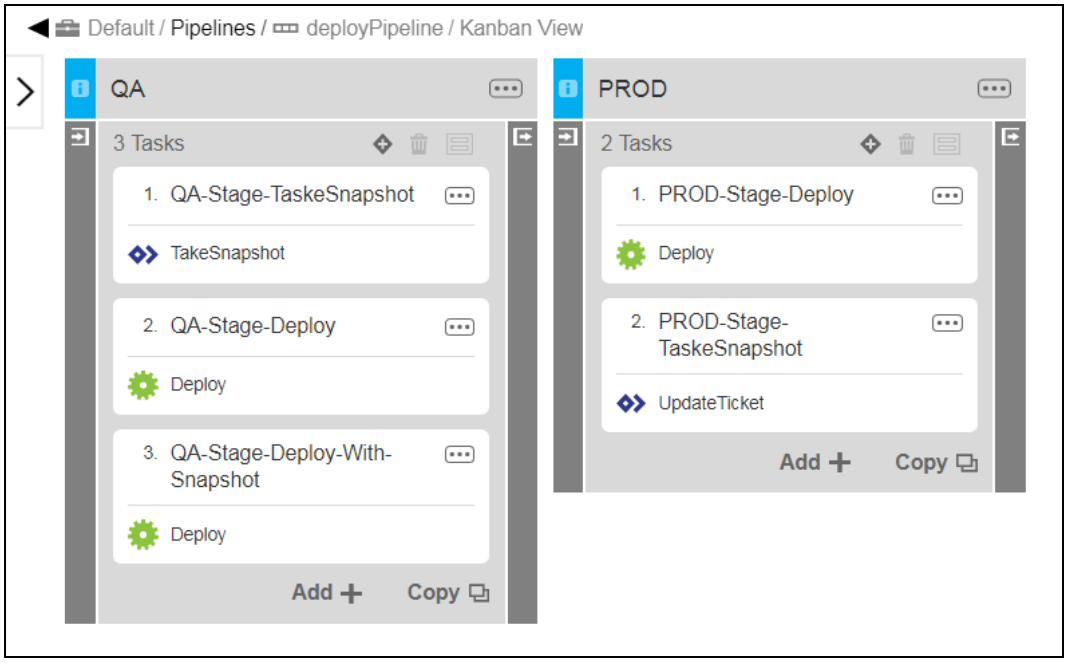
This section describes how to change the order in which the tasks are executed. Starting in the task list:

1. Choose the task that you want to move.

2. Click and drag the task to move the task to its new position:



The task is now in the first in the list:



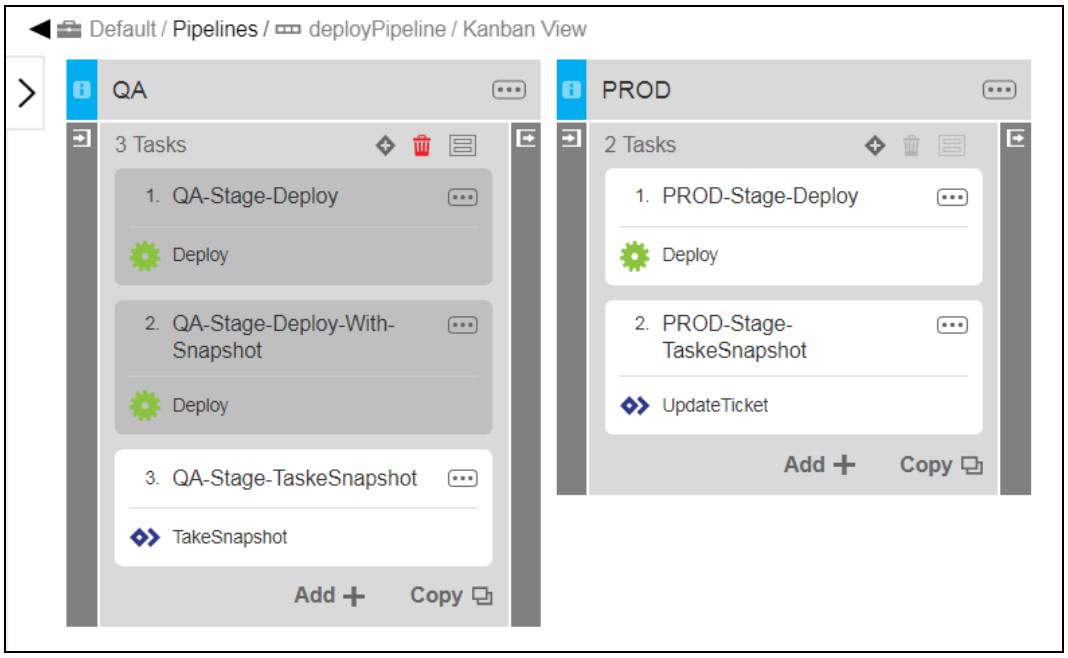
> **Tip:** You can also click **Reorder: After** or **Reorder: Before** from the (Actions) menu for the stage.

Grouping Pipeline Tasks for Running in Parallel or Serial Order

By default, tasks are executed in order. You can group tasks without dependencies and run them in parallel to reduce runtimes and make a stage more efficient. You can also group tasks and run them in serial order.

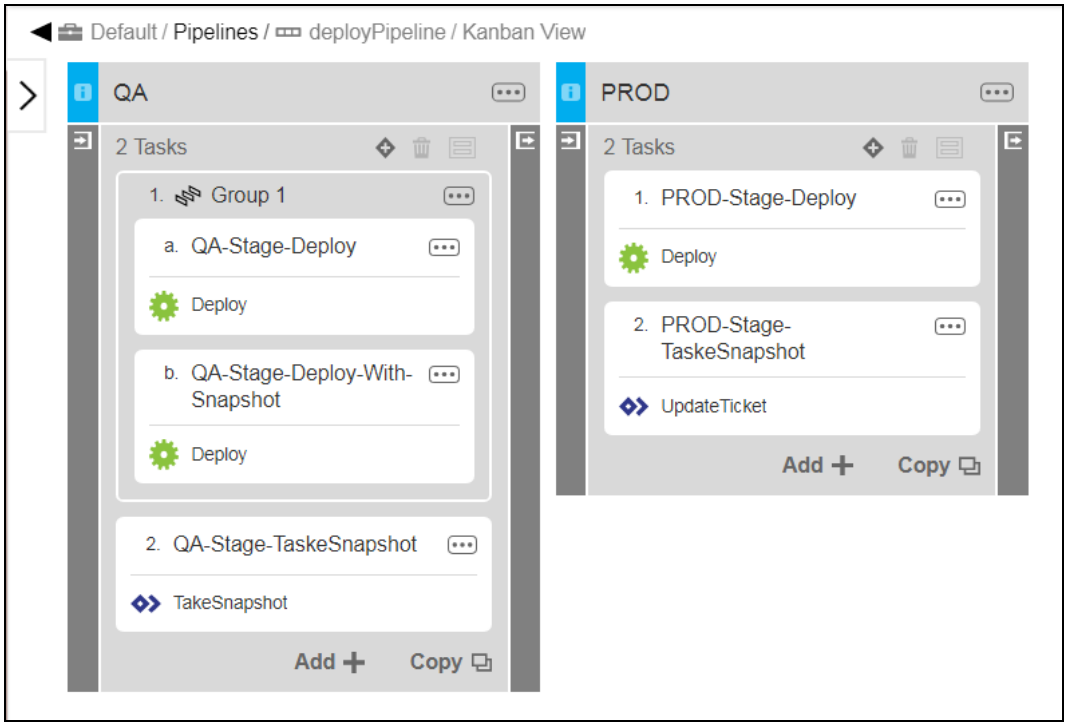
To make a set of sequential tasks run in parallel:

1. Click to highlight at least two tasks that you want to run in parallel.



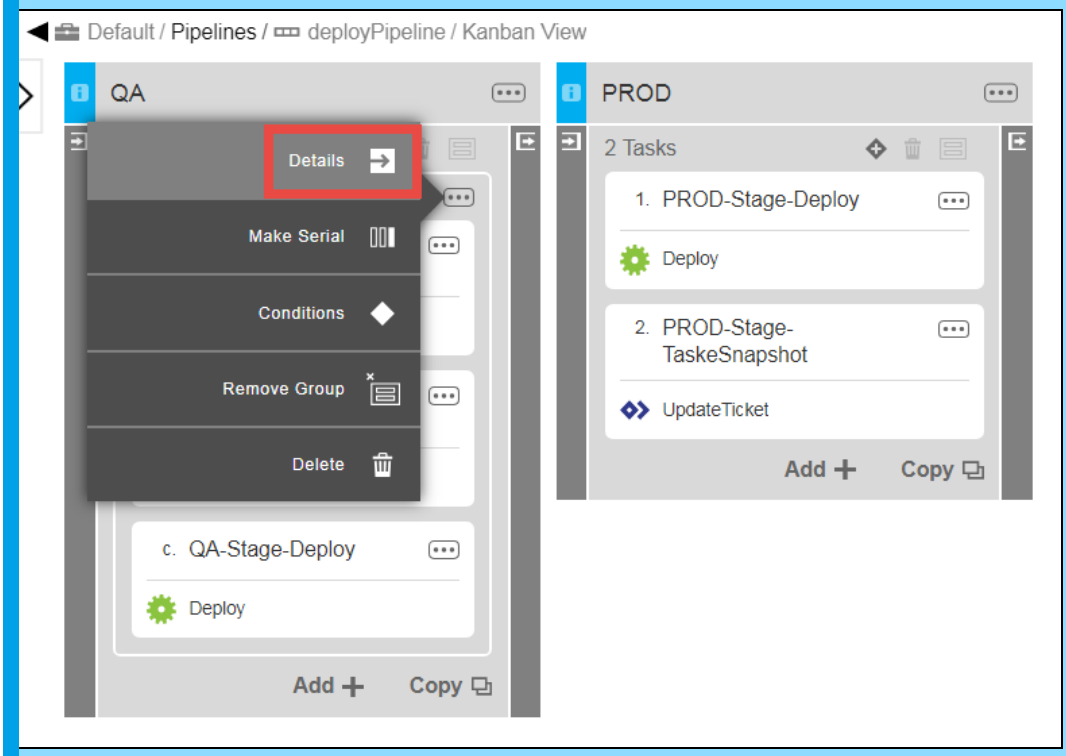
2. Click the Group button.

A parallel tasks group is created, which contains the tasks that will now run in parallel:



Tip: You can add new tasks to an existing parallel tasks group. To do so, drag the task to move the task to the group. You can rename a parallel tasks group by clicking the (Actions) button for

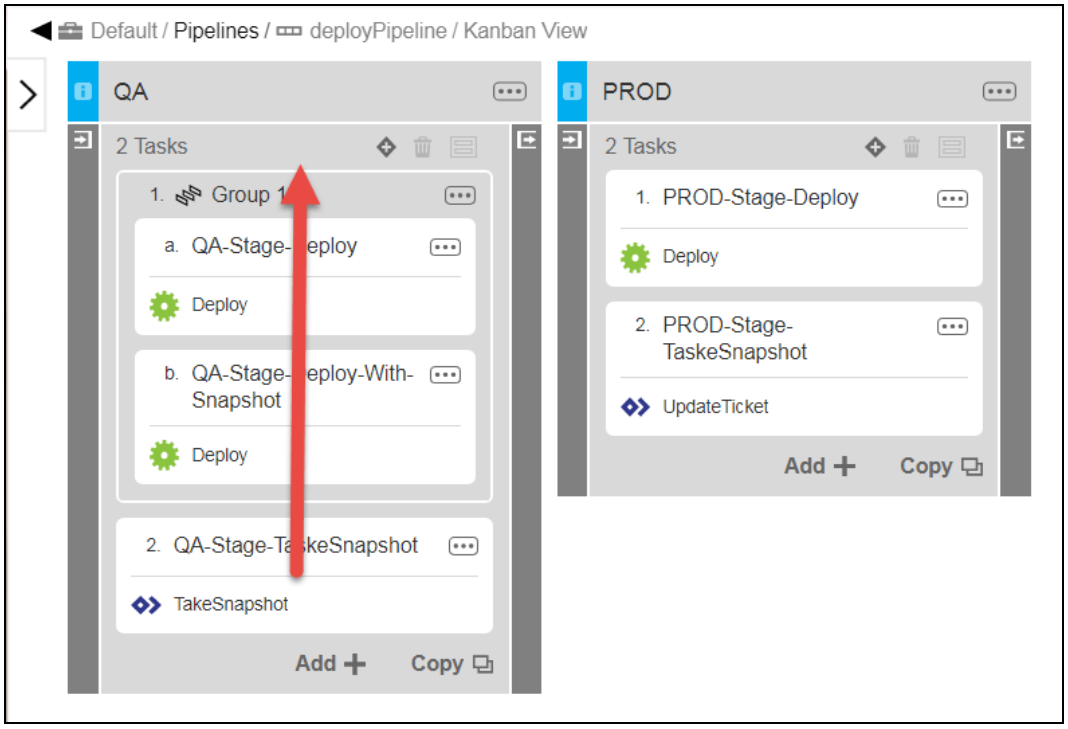
ne group and then clicking **Details** to fill in a new name in the **Edit** dialog box.



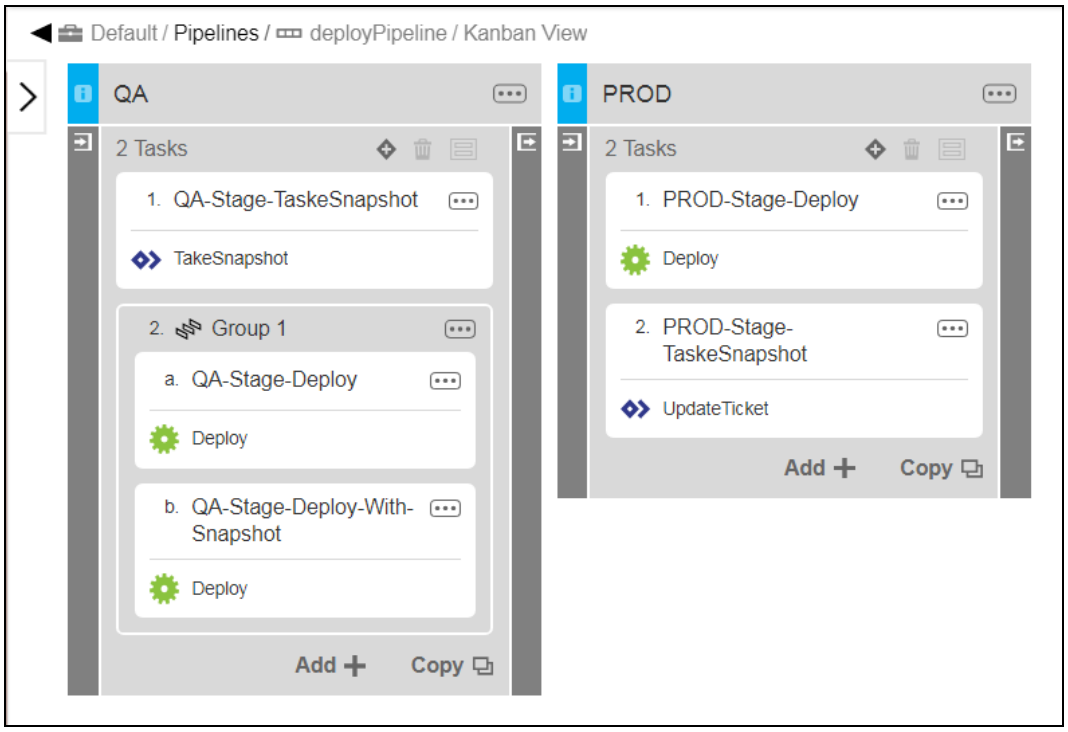
Changing the Order of Pipeline Tasks that Include a Group of Parallel Tasks

You can change the execution order of tasks that include a group of parallel tasks. To do so:

1. Click and drag a task to move it to its new position:



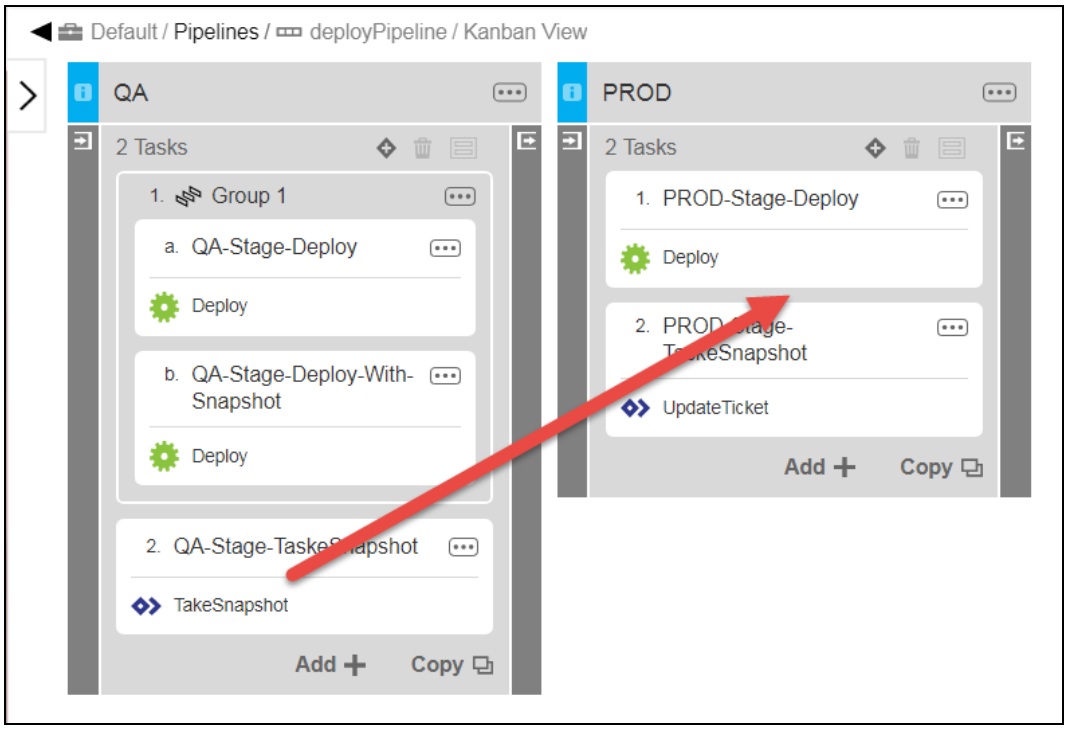
The task is now the first in the list:



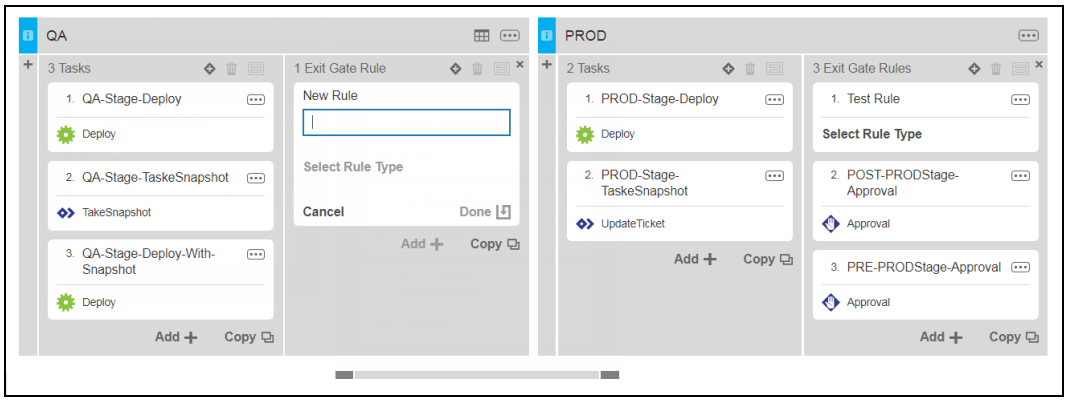
Tip: You can also click **Reorder: After** or **Reorder: Before** from the (Actions) menu for the stage.

Moving Pipeline Tasks from One Stage to Another Stage

You can move a task from a stage to another stage. To do so, click and drag the task to move it to its new position in the other stage:



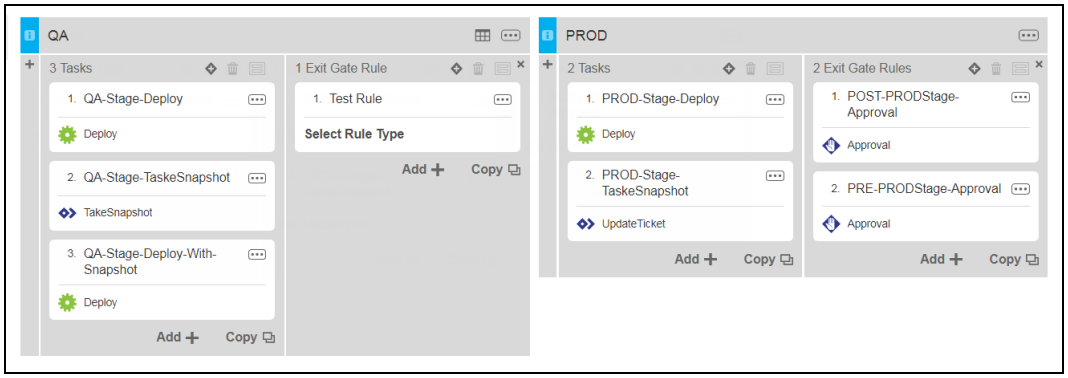
The task is now in the other stage:



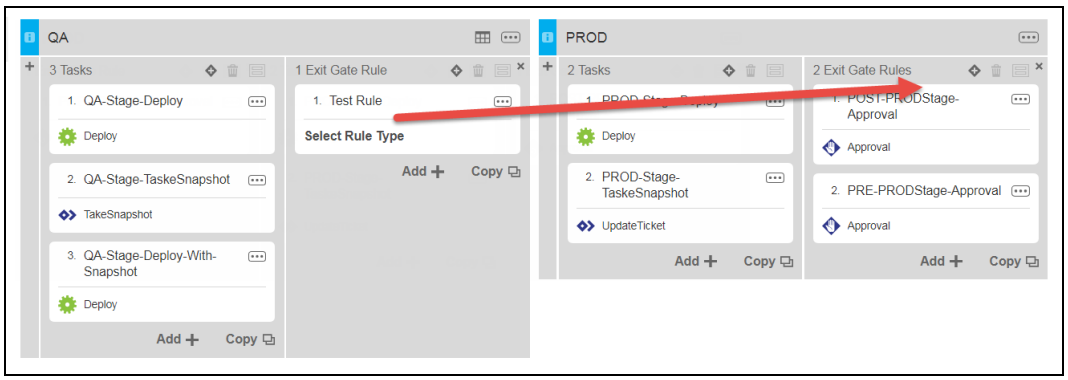
Moving Pipeline Gate Rules from One Gate to Another Gate

You can drag a pipeline gate rule from a gate to another gate. To do so:

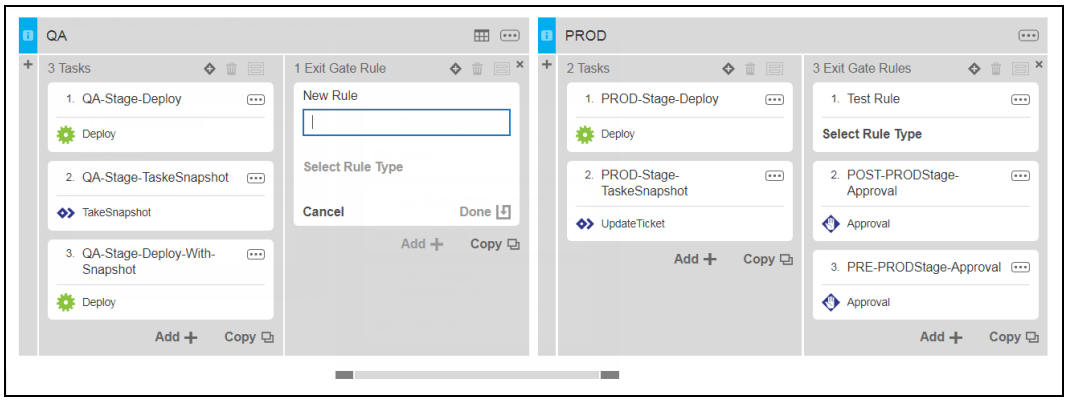
1. Click to open both gates:



2. Click and drag the rule to move it to its new position in the other gate:



The rule is now in the other gate:

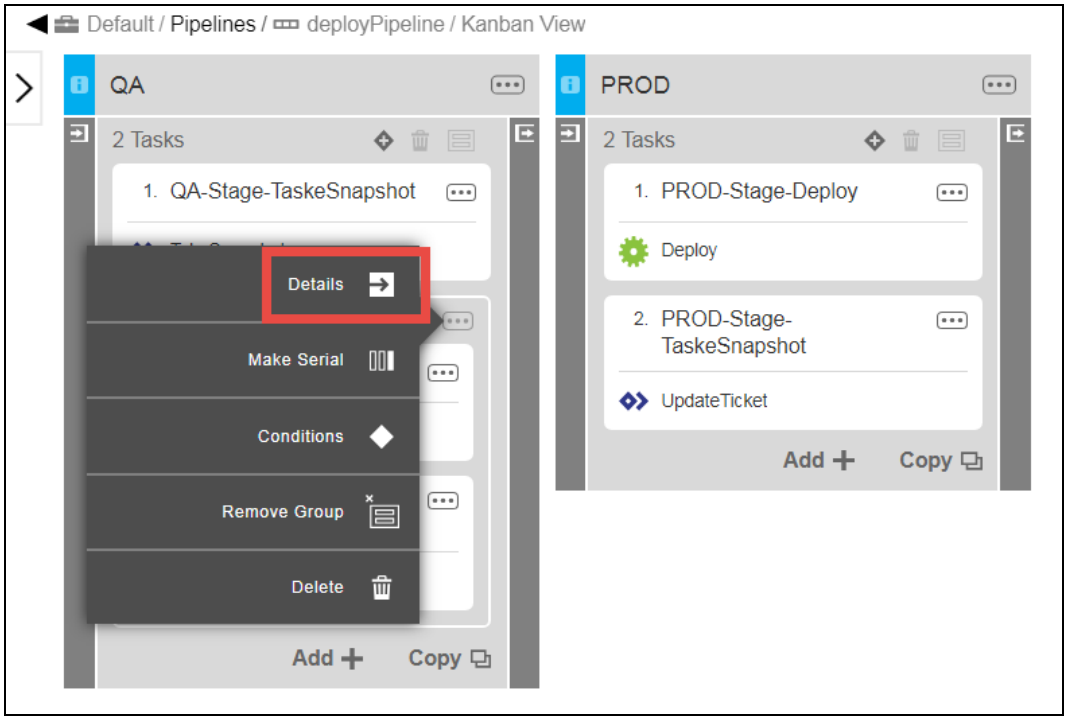


Setting the Error or Rejection Handling for a Parallel Task Group

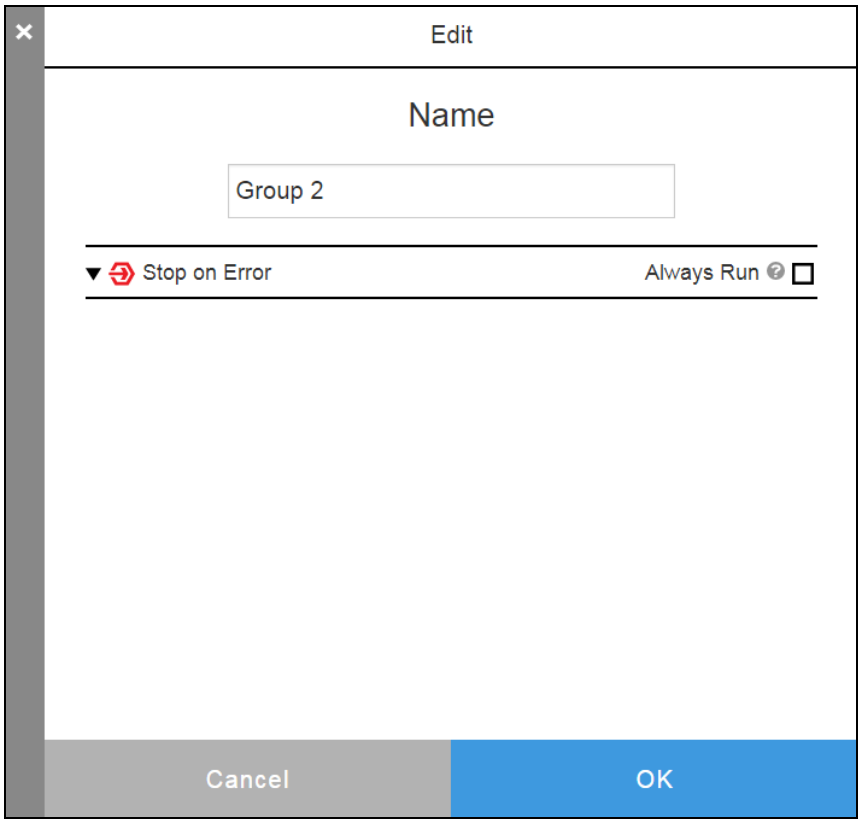
When defining a group of parallel tasks, you can specify whether the pipeline continues or stops if any task in the group fails because of an error or rejection. By default, if any task fails or is rejected, the pipeline aborts. This behavior is usually used in Continuous Delivery pipelines.

To set the error or rejection handling:

1. Click the (Actions) button for the group and then click **Details**.



The **Edit** dialog box appears.



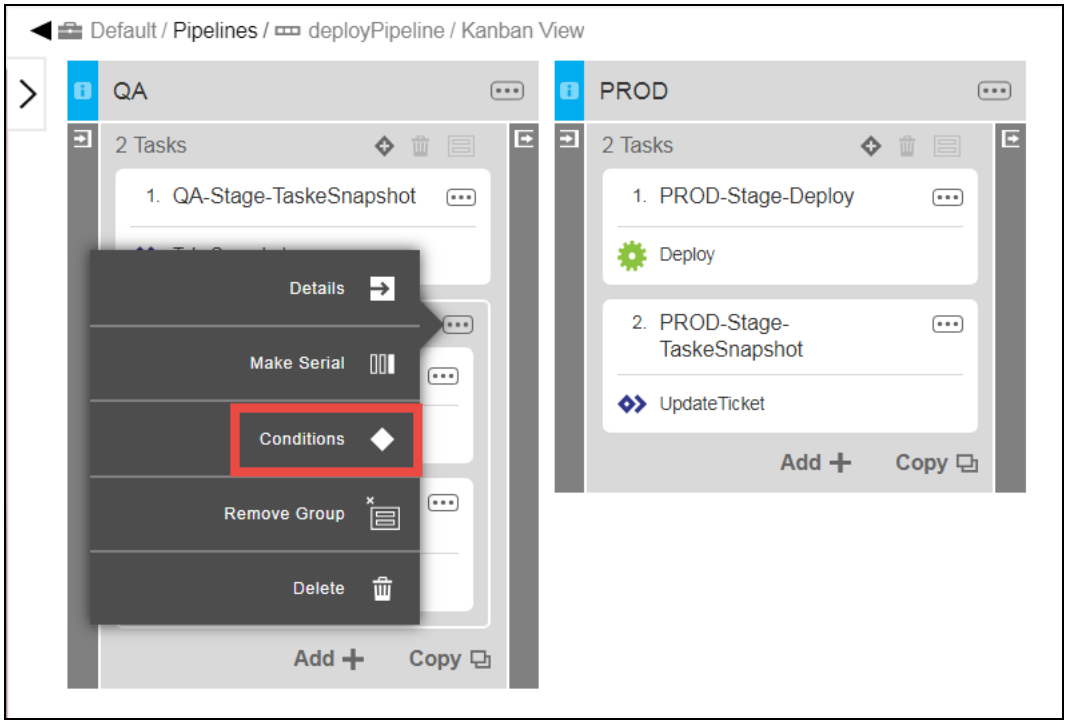
2. Select one of the following options:
   - **Continue on Error**: If any task fails or is rejected, the pipeline continues to the next gate or stage task.
   - **Stop on Error**: If any task fails or is rejected, the pipeline aborts.
3. Select **Always Run**. Enabling this option guarantees that the tasks will run before the pipeline execution ends. Note that if this option is selected, and any individual task in the group that has error handling set to **Stop on Error** fails, then all the running tasks will be aborted.
4. Click **OK.**

Adding "Run if" and "Wait until" Conditions to a Parallel Task Group

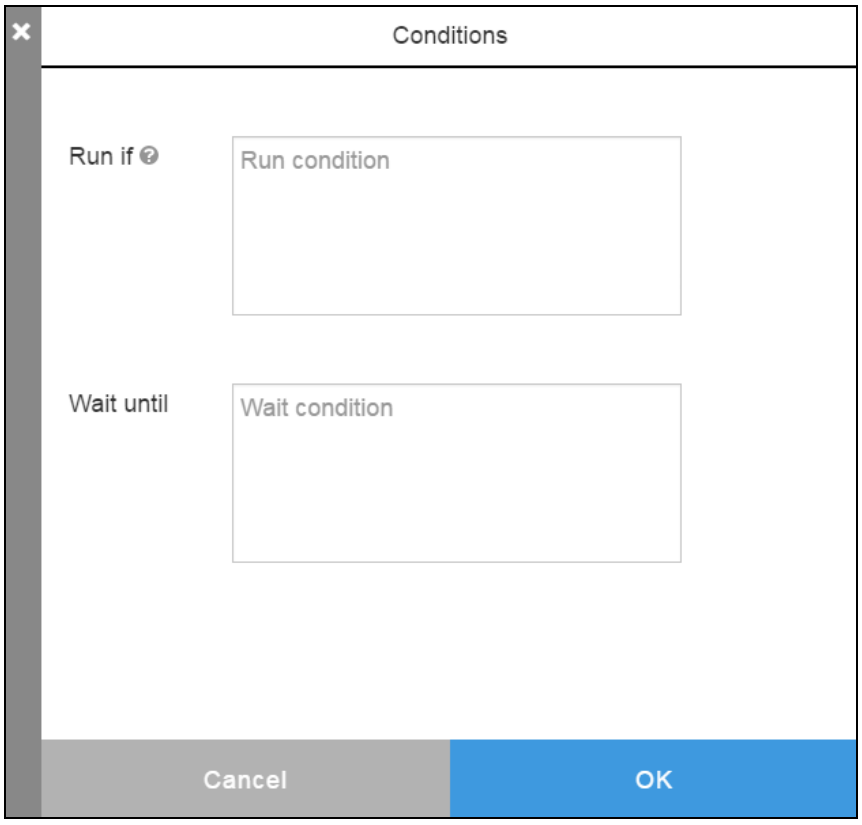
For a parallel task group, you can specify "Run if" and "Wait until" conditions to control the group behavior based on a precondition of your choice. Skipping a group causes all of its tasks to be skipped.

You can specify preconditions by entering a JavaScript expression. To do so:

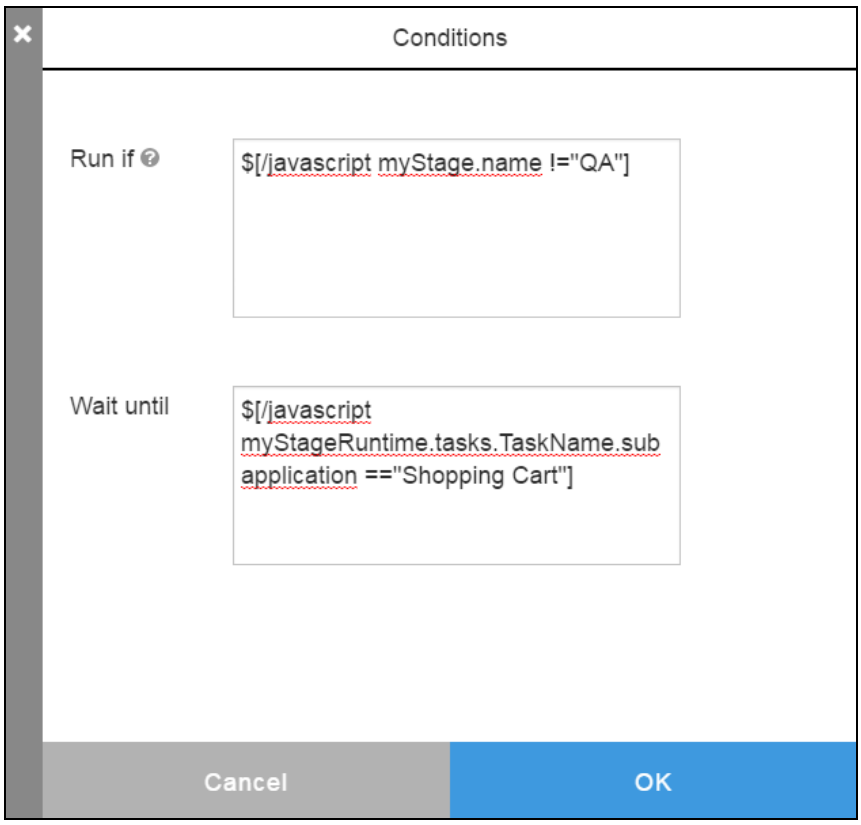
1. Click the (Actions) button for the group and then click **Conditions**.



The **Conditions** dialog box appears.


Conditions
Run if
Run condition
Wait until
Wait condition
Cancel
OK

2. Enter a JavaScript expression into the **Run if** field or the **Wait until** field (or both).



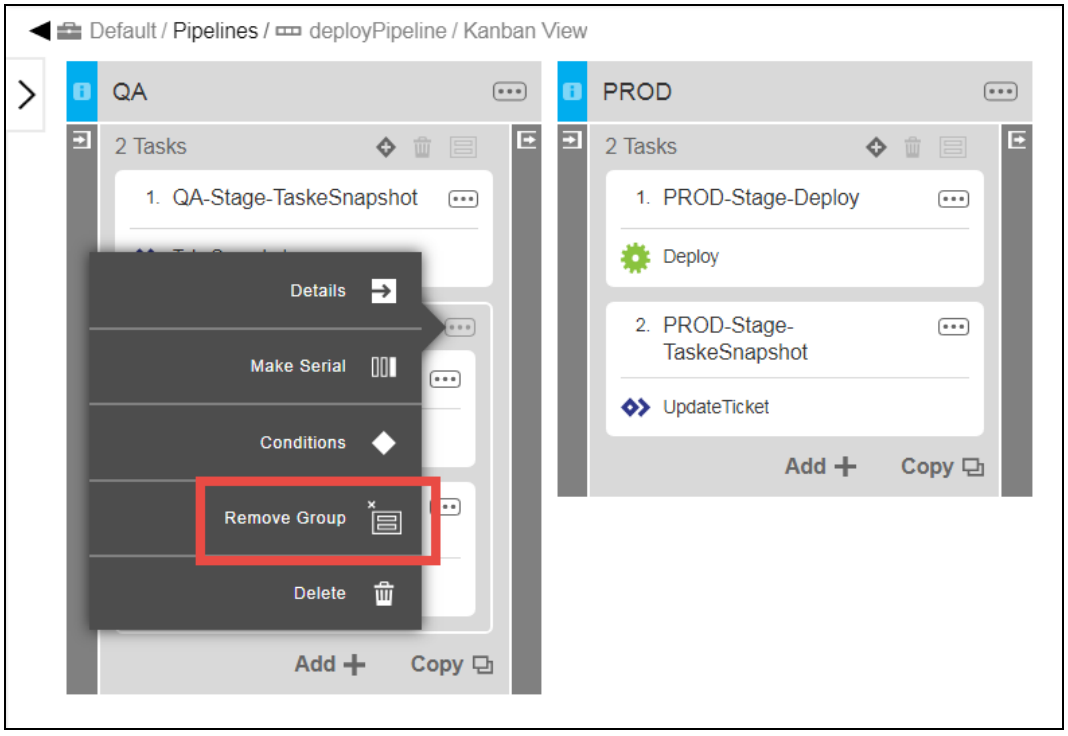
For more examples of Javascript expressions for "Run if" and "Wait until" conditions, see the *KBEC-00360 - Using Context-Relative Shortcuts to Properties on Pipelines and related objects* KB article.

3. Click **OK**.

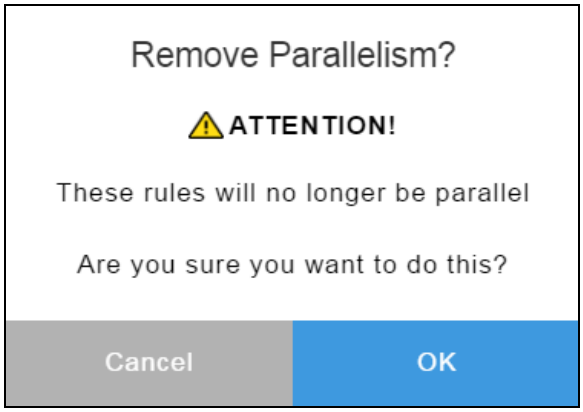
Unparallelizing Pipeline Tasks

You can unparallelize tasks, which means returning parallel tasks in a group to sequential execution. To do so:

1. Click the (Actions) button for the group and then click **Remove Group**.



A confirmation popup appears:



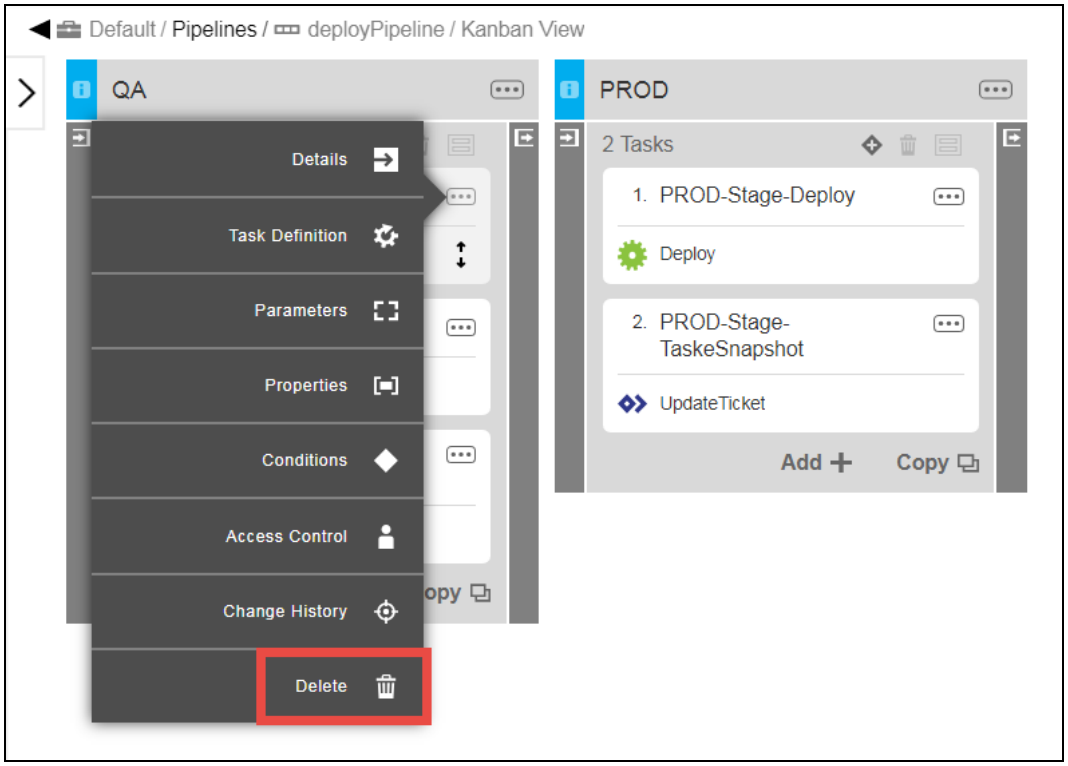
2. Click **OK**.

   The tasks will once again run in sequence.

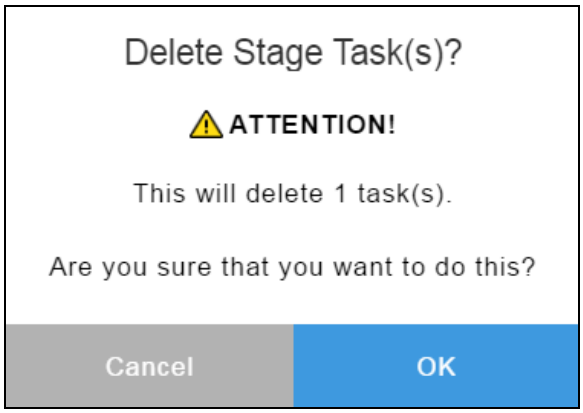
Deleting a Pipeline Task from a Stage

You can delete a pipeline task from a stage. To do so:

1. Click the (Actions) button for the task, and then click **Delete**.



A confirmation popup appears:



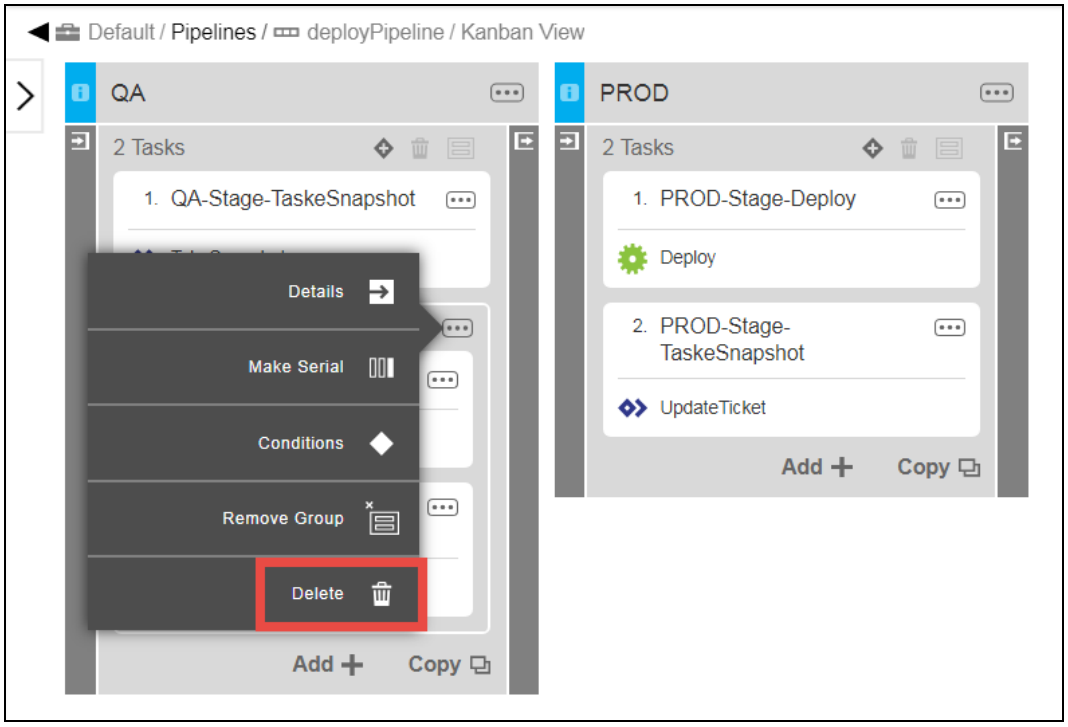
2. Click **OK**.

Tip: You can also use this method to delete a parallel task group and all of its tasks.

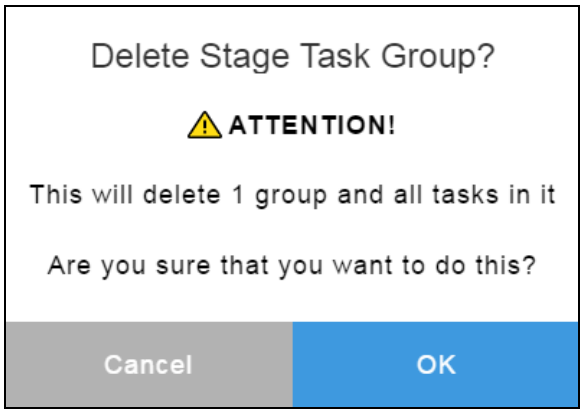
Deleting a Parallel Task Group and All of Its Tasks from a Stage

You can delete a parallel task group from a stage. This also deletes all of its tasks. To do so:

1. Click the (Actions) menu for the parallel task group and then click **Delete**.



A confirmation popup appears:



2. Click **OK**.

Editing a Pipeline Definition

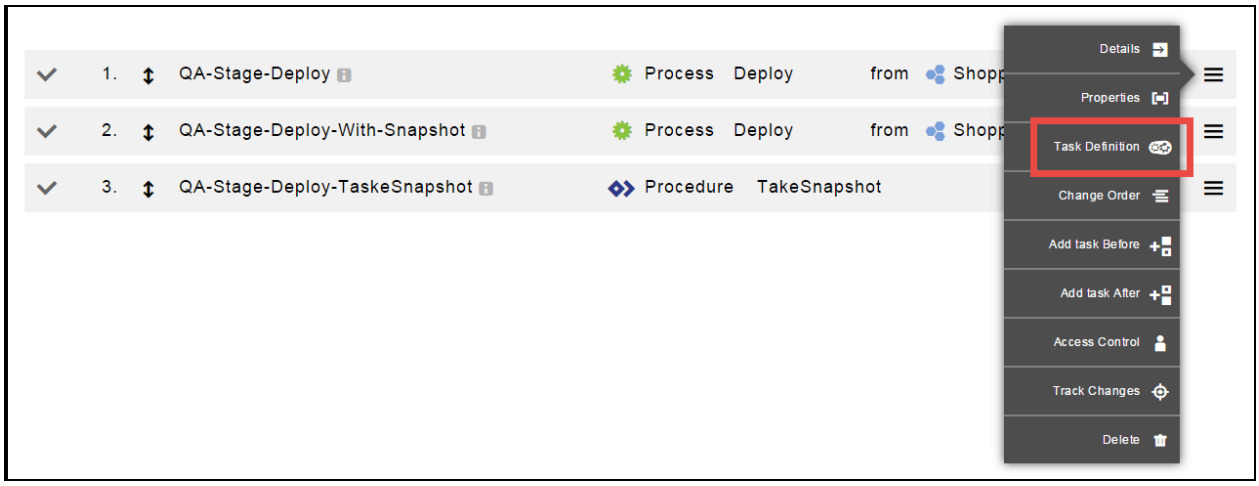
To edit a pipeline definition, you edit its tasks.

Editing a Pipeline Definition in Pipeline Stage View

To edit a pipeline definition in the Pipeline Stage View:

1. Go to the pipelines list.

2. Click the menu button at the end of the row for a task that you want to edit, and then click **Task Definition**:



   The task definition appears.

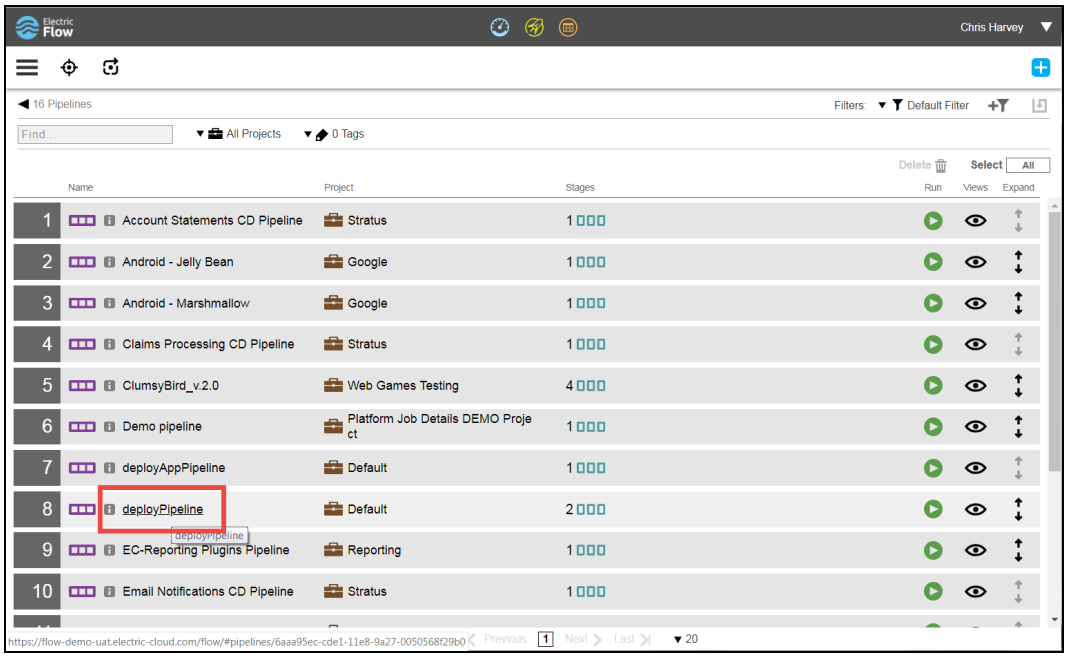
3. Change the settings that you want to modify as described in the sections above, and then click **OK**.

Editing a Pipeline Definition in Release Kanban View

To edit a pipeline definition in the Release Kanban View:

1. Go to the pipelines list.
2. Click the pipeline containing the task that you want to edit.

   For example:



   The pipeline appears.

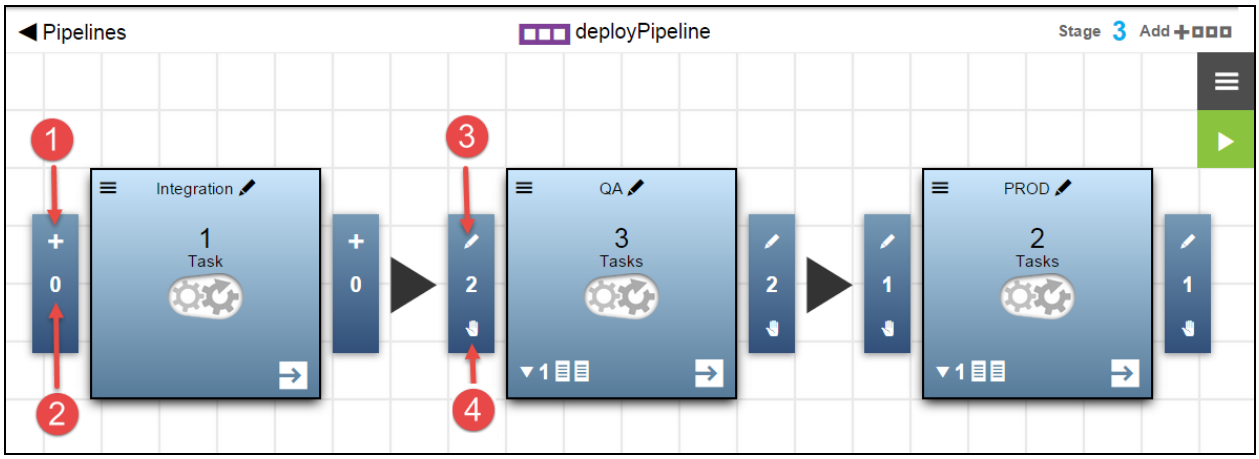
3. Change the settings that you want to modify as described in the sections above.

Defining Gate Approvals

This section describes how to create approval rules on the entry and exit gates. A gate approval can be either manual (where the approver must physically click the approval to move to the next stage) or automated (based on whether specific criteria are met). You can use a combination of both types of approvals on a gate. You can also change the order of approval rules, run approval rules in parallel, change the order of approval rules that include parallelized rules, and unparallelize approval rules.

Gates for Pipelines

The following example shows entry and exit gates in a pipeline in the ElectricFlow UI:



1	The button adds approvals to a gate that has none.
2	Number of approvals in the gate.
3	The button adds an approval to the gate or edits the approvals.
4	The hand icon means "needs attention." This means that the gate will require attention when the pipeline gets to this entry gate.

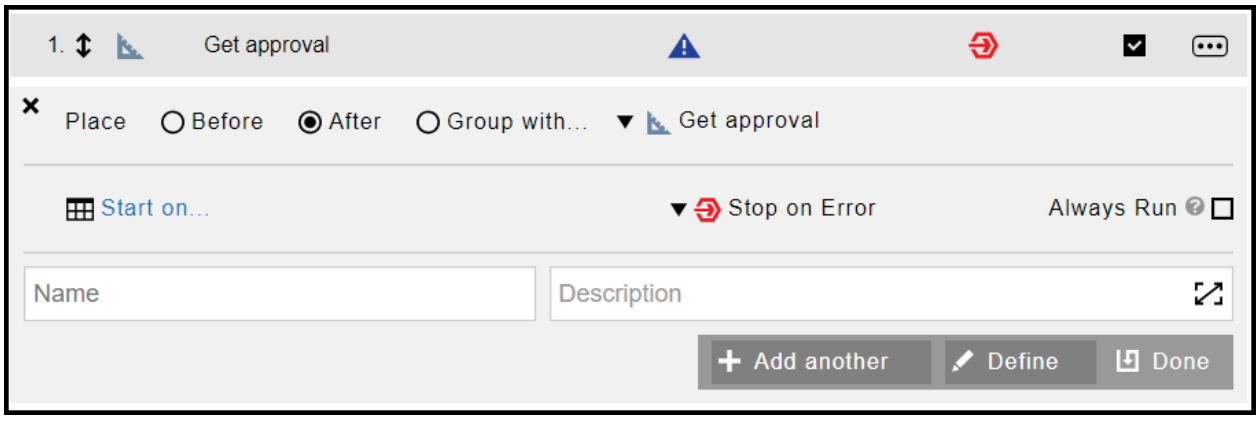
Creating a Manual Approval Rule in a Gate

This example shows how to create a single manual approval rule in which one approver needs to take action.

Starting in the Pipeline Editor:

1. Go to the gate where you want to add the approval rule.
2. Click the  button on the gate and select **Add Rule Before** or **Add Rule After** to add a gate approval rule.

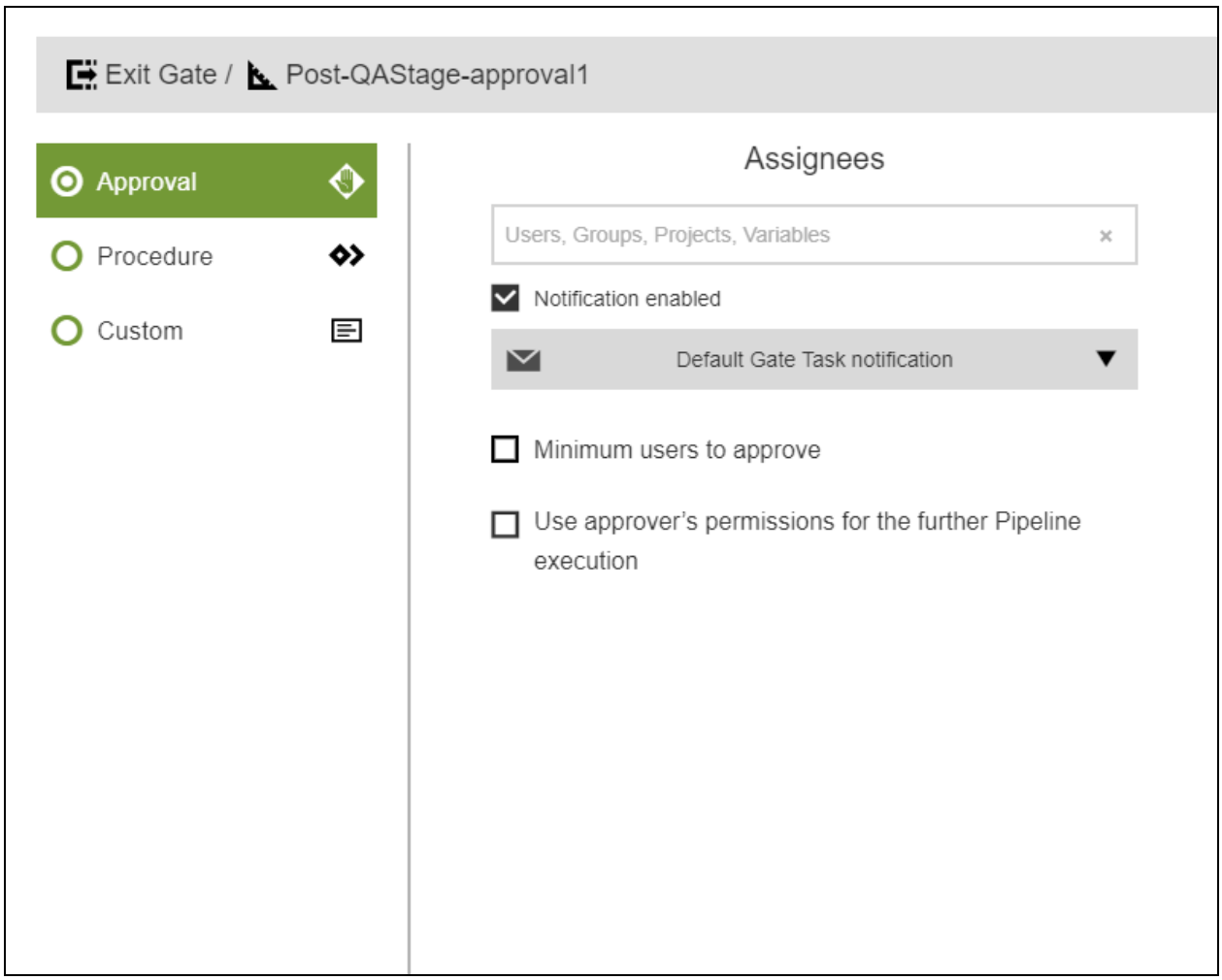
   The page to add or edit approval rules for the gate opens. Following is an undefined approval rule:



3. Select **Start on** date, action to take on error, and whether to always run the rule.
4. In the **Name** field, enter **POST-QAStage-Approval1**.
5. In the **Description** field, enter a suitable description of the rule.

6. Click **Define**.

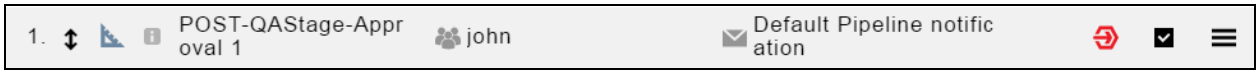
   The gate approval definition screen dialog appears:



7. Make sure that the **Approval** button is selected (the default).
8. In the **Assignees** field, add the user named **john**. Start typing the user or group name to find available users or groups.

   Keep **Default Pipeline Notification** as is. Note that you can add multiple approvers or groups.
9. Click **OK**.

For the gate, there is now one approval rule that user john must approve for the pipeline to progress:



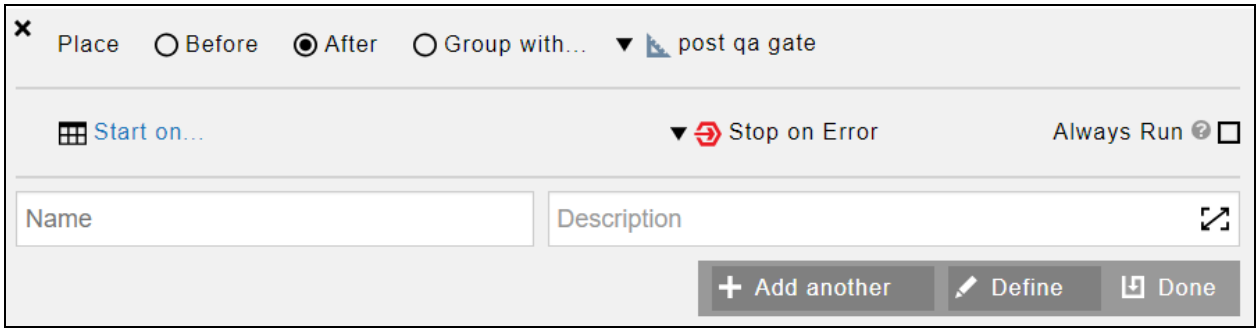
Creating a Procedure-Based Approval Rule in a Gate

This example shows how to create a procedure-based rule. This type of rule contains criteria that let you call any procedure. The job status corresponding to the procedure being executed determines the gate task outcome. A job outcome of `Success` (finished with no errors) is considered to be approved. A job outcome of `Warning`, `Failure`, or any other non-success job status is considered to be rejected.

Starting in the Pipeline Editor:

1. Go to the gate where you want to add the approval rule.
2. Click the  button to add another approval rule.

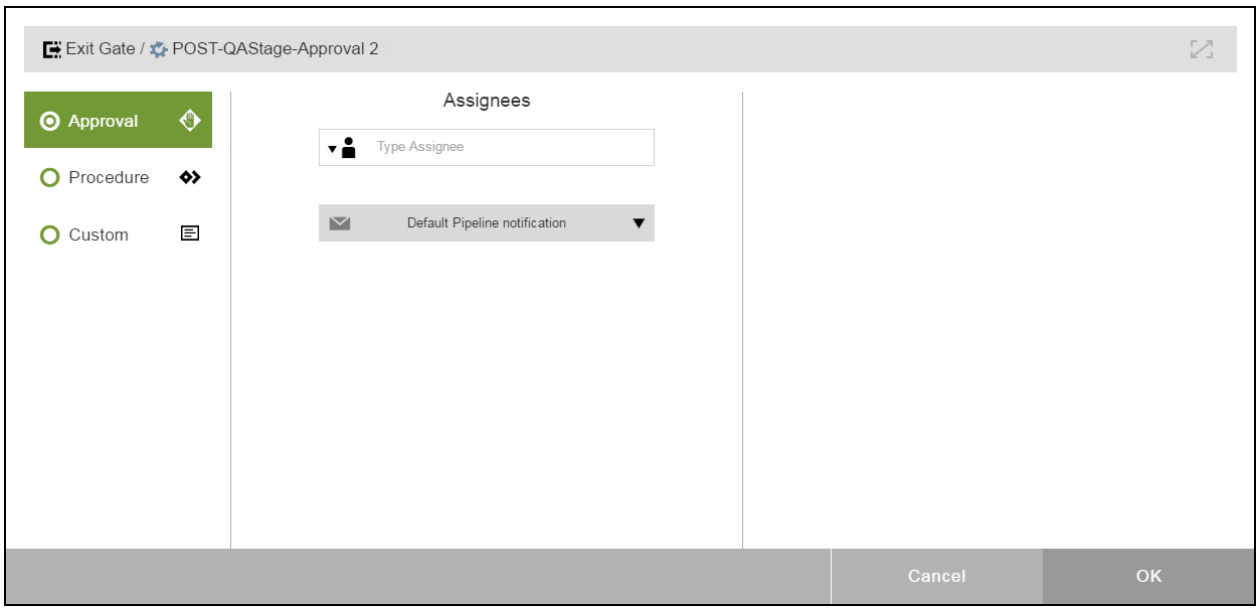
   The page to add or edit approval rules for the gate opens. This is an undefined approval rule for subsequently-added approval rules. You can choose the placement of this rule relative to the rule above it:



3. In the **Name** field, enter **POST-QAStage-Approval 2**.

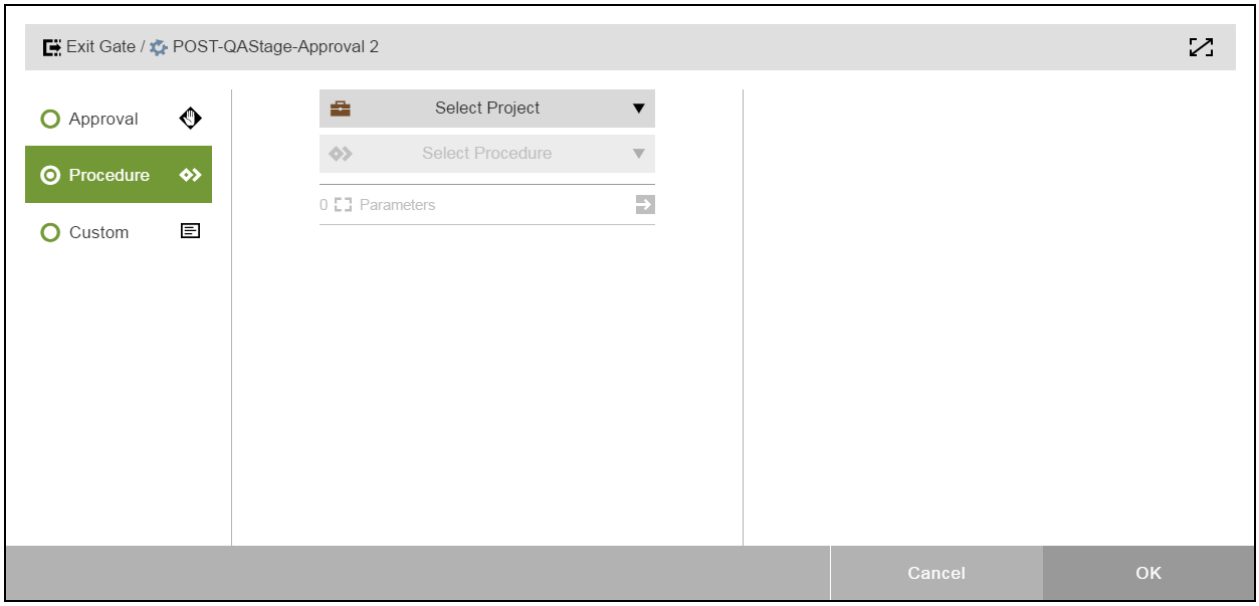
4. In the **Description** field, enter a suitable description of the rule and then click **Define**.

   The exit gate approval definition dialog appears:



5. Click the **Procedure** button.

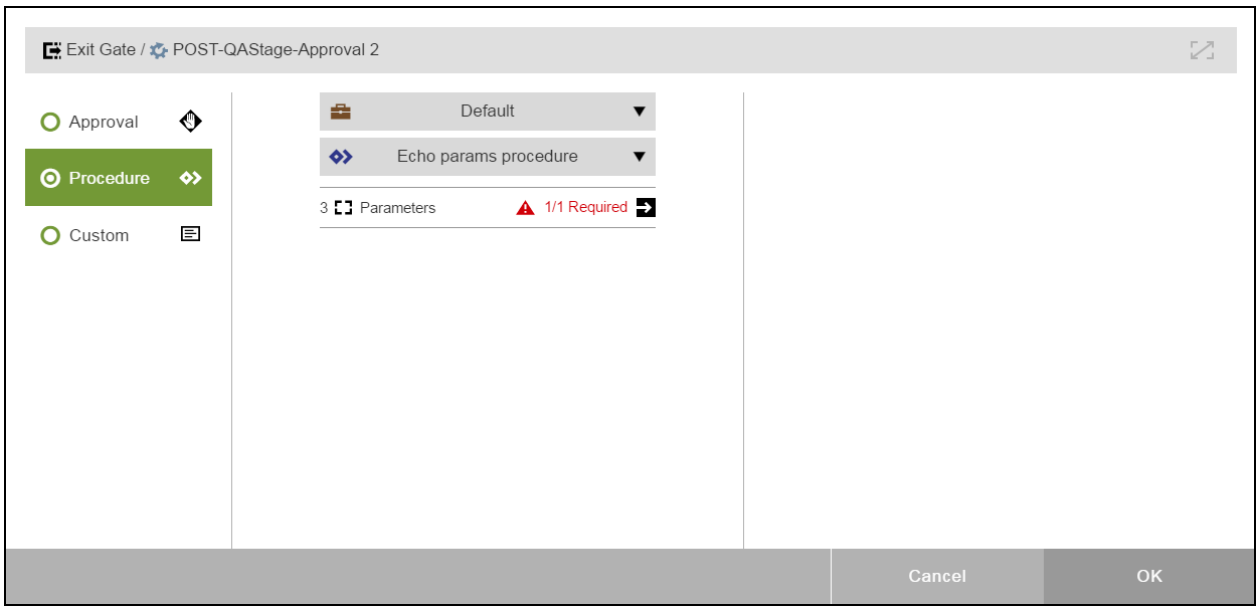
   The dialog box for defining a procedure-based approval appears:



6. Select the **Default** project from the **Select Project** menu.

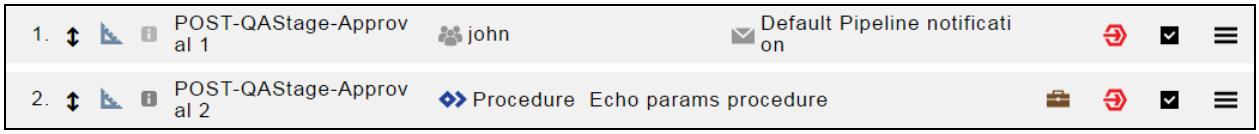
7. Select the **Echo params procedure** procedure from the **Select Procedure**menu.

   The Parameters button becomes active:



8. Click **Parameters**, enter values of your choice for **param1**, **param2**, and **param2**, and then click **OK**.

   The procedure-based approval rule appears in the page to add or edit approval rules for the gate:



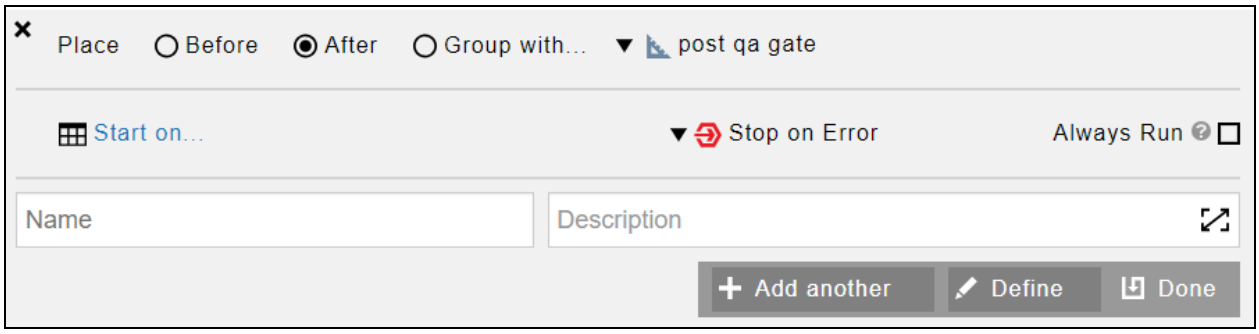
Creating a Condition-Based Approval Rule in a Gate

This example shows how to create a condition-based rule. You can create gate criteria based on custom conditions of your choice that evaluate to true or false to approve or reject gates. A condition is a Javascript expression that leverages properties. For example, you can create a property on the pipeline runtime based on the output of a test suite and write an expression to evaluate whether the property is 50 or greater (pass) or less then 50 (fail). You can specify criteria based on any intrinsic properties that are appropriate for a release pipeline and a pipeline runtime. For details about the intrinsic properties for each object in ElectricFlow Deploy, see the *Intrinsic Properties in ElectricFlow Deploy Objects* document at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html.

Starting in the Pipeline Editor:

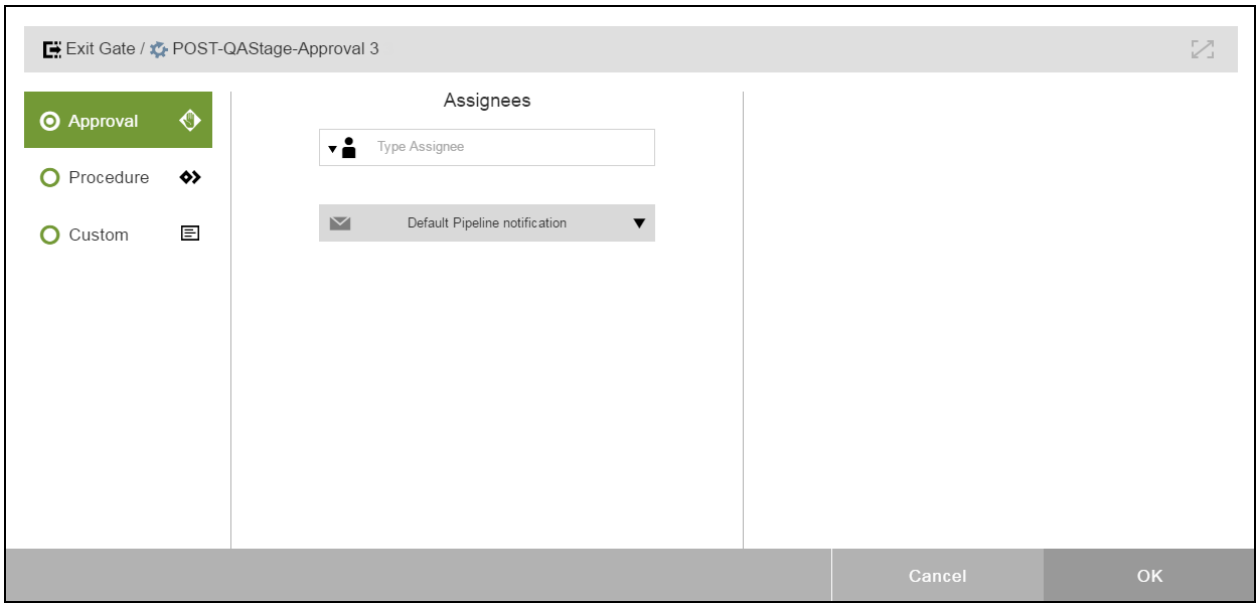
1. Go to the gate where you want to add the approval rule.
2. Click the button to add another gate approval rule.

   The page to add or edit approval rules for the gate opens:



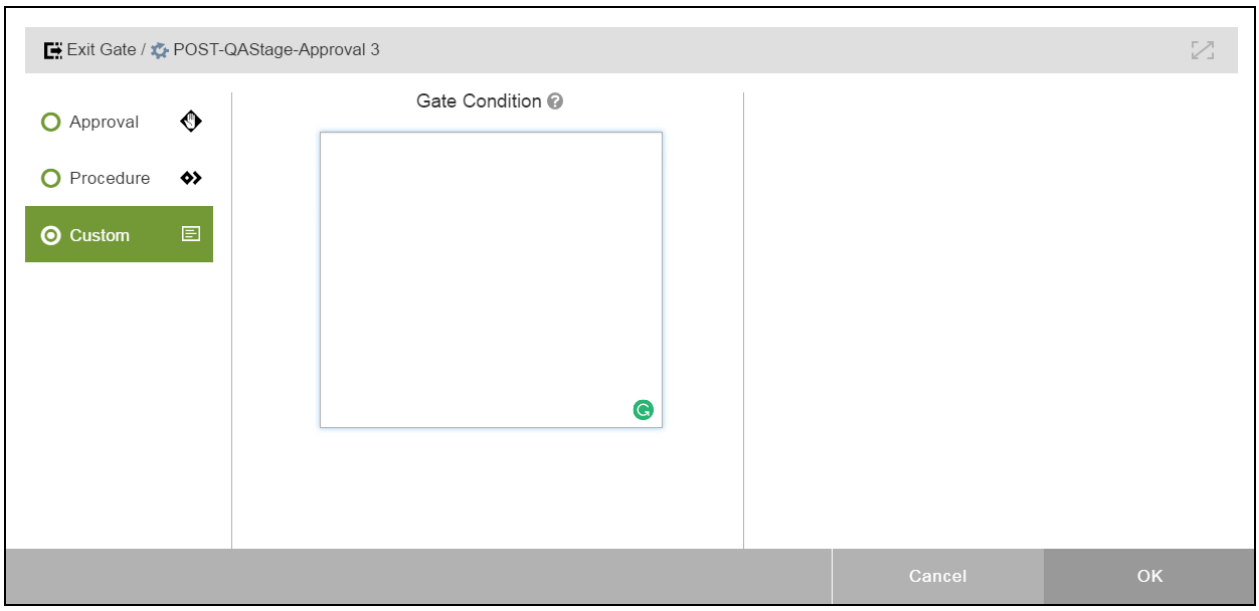
3. In the **Name** field, enter **POST-QAStage-Approval 3**.
4. In the **Description** field, enter a suitable description of the rule and then click **Define**.

   The exit gate approval definition dialog appears:



5. Click the **Custom** button.

   The dialog box for defining a condition-based approval appears:



6. In the **Gate Condition** field, enter one of the following conditions.

   - To check the outcome of the previous task in the same stage using the task name:

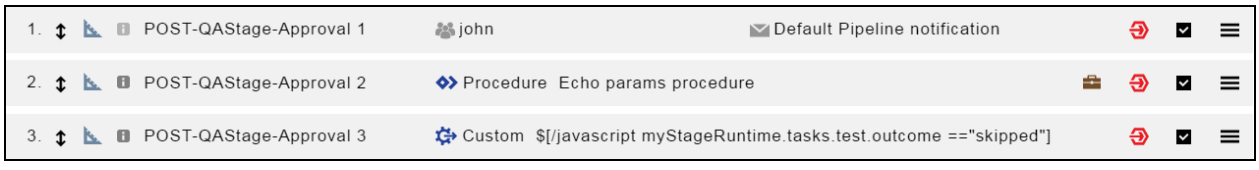
     ```
     $[/javascript myStageRuntime.tasks.test.outcome =="skipped"]
     ```

   - To set the evidence property in the task, which will appear in the comment after the task has completed:

     ```
     $[/javascript
     setProperty("/myTaskRuntime/evidence", "test value")
     true;
     ]
     ```

7. Click **OK**.

   The condition-based approval rule appears in the page to add or edit approval rules for the gate:



For more examples of Javascript expressions for conditions, see the *KBEC-00360 - Using Context-Relative Shortcuts to Properties on Pipelines and related objects* KB article.

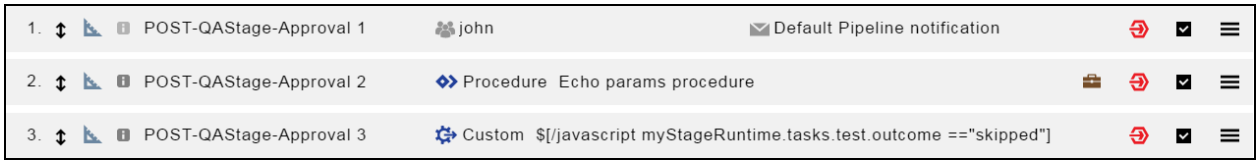
Changing the Order of Approval Rules

You can change the execution order of approval rules. To do so:

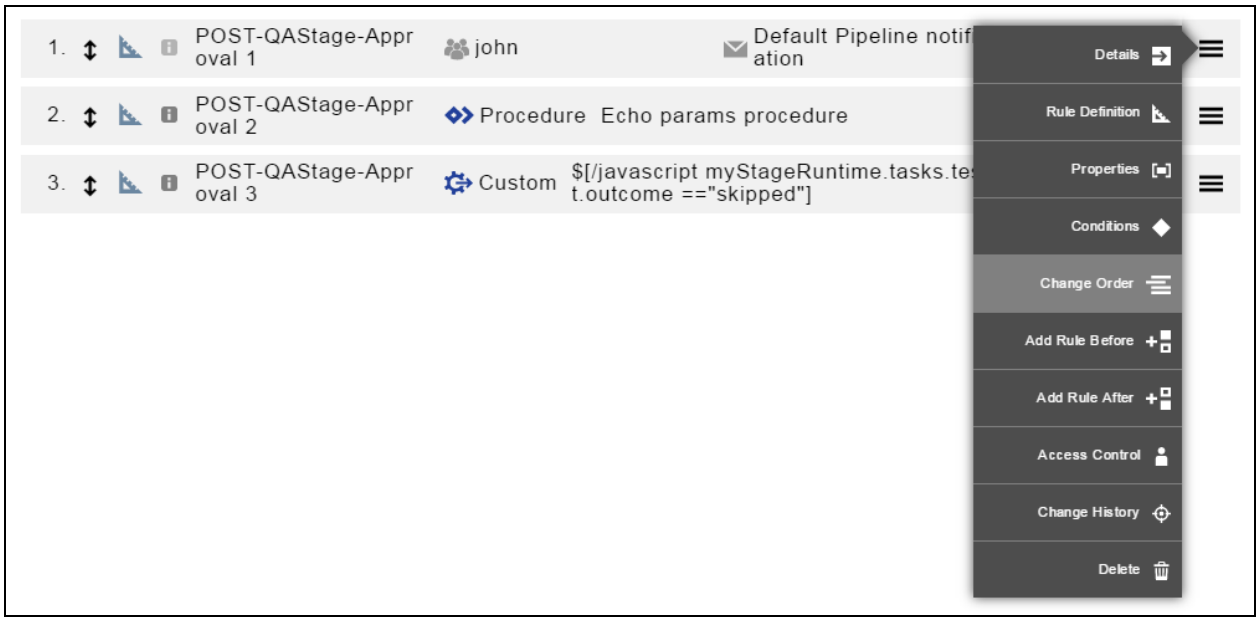
1. Go to the gate where you want to change the order of approval rules containing parallelized rules.
2. Click the button on the gate.

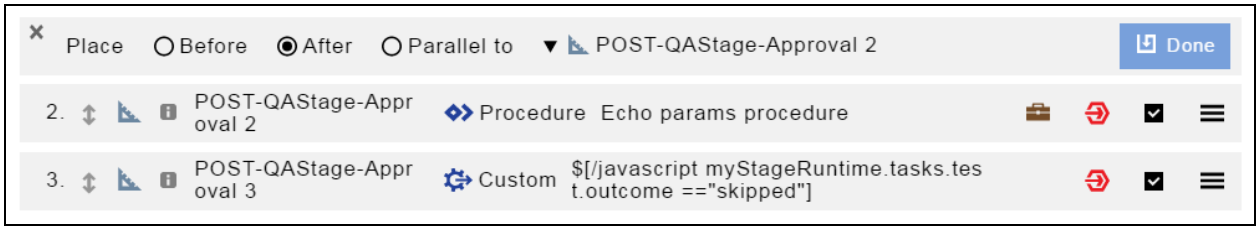
   The page to add or edit approval rules for the gate opens:



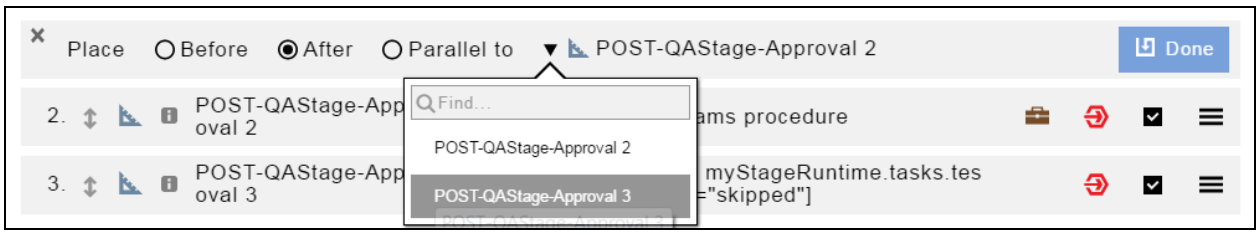
3. Click the menu for the **POST-QAStage-Approval 1** rule and choose **Change Order:**



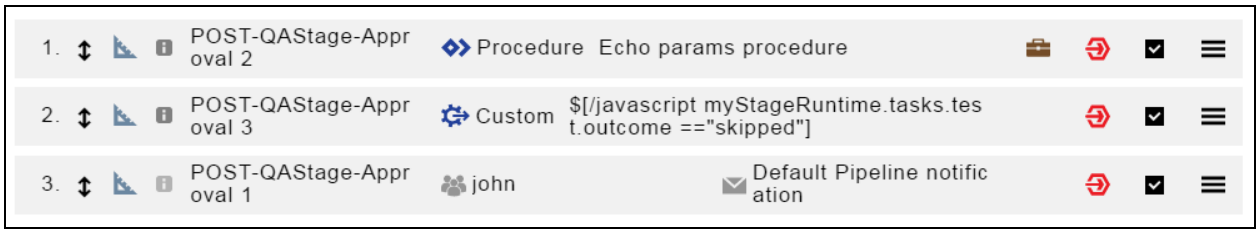
A menu appears that lets you reorder the group of parallel rules in relation to the other rules:



4. Click the pipeline rules (▼) menu, then click **POST-QAStage-Approval 4**, and then click **Done**:



The **POST-QAStage-Approval 1** rule will now run after the **POST-QAStage-Approval 3** rule:



Note that you can also click and drag the ↕ (up/down) button next to a rule name to reorder it.

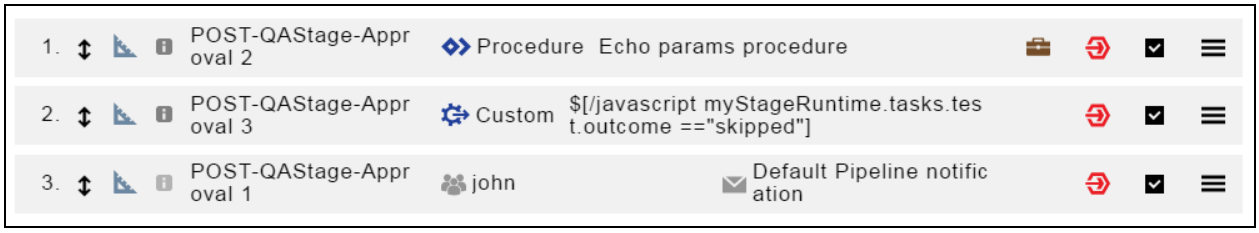
Running Approval Rules in Parallel

By default, approval rules are executed in numerical order. This means that gates must be approved in sequence, which can cause delays in pipeline execution if an approver does not act on an approval request in a timely manner. To mitigate this risk, you can configure any combination of sequential approval rules in parallel.

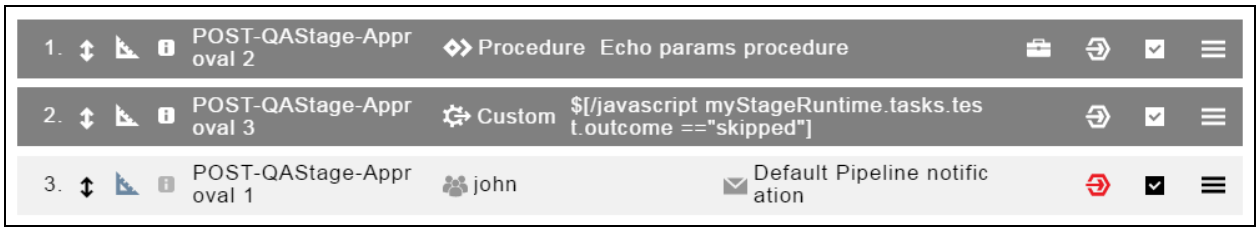
To make a set of sequential approval rules run in parallel:

1. Go to the gate where you want to parallelize the approval rules.
2. Click the button on the gate.

   The page to add or edit approval rules for the gate opens:



3. Click to highlight the **POST-QAStage-Approval 2** and **POST-QAStage-Approval 3** rules.



4. Click the button.

   The rules will now run in parallel:

Note that you can rename a parallel rules group by clicking the menu for the group and choosing **Details** to fill in a new name.

Changing the Order of Approval Rules that Include a Group of Parallelized Rules

You can change the execution order of approval rules that include parallelized rules. To do so:

1. Go to the gate where you want to change the order of approval rules containing parallelized rules.
2. Click the button on the gate.

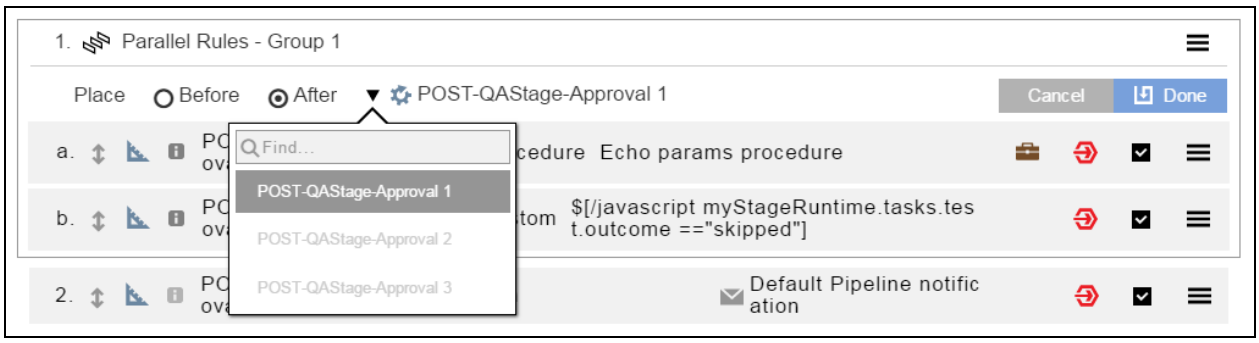
   The page to add or edit approval rules for the gate opens.

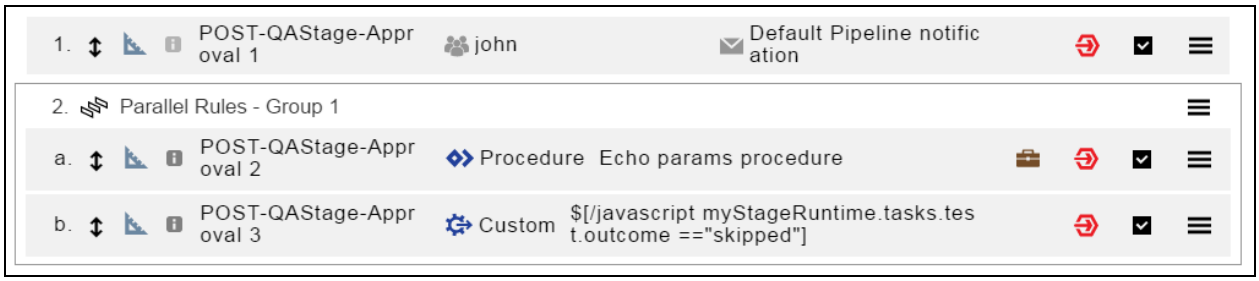
3. Click the menu for the **Parallel Rules—Group 1** group and choose **Change Order**:

A menu appears that lets you reorder the group of parallel rules in relation to the other rules:

4. Click the pipeline rules (▼) menu, then click **POST-QAStage-Approval 1**, and then click **Done**:



The rules in the **Parallel Rules—Group 1** group will now run after the **POST-QAStage-Approval 1** rule:



Note that you can also click and drag the ↕ (up/down) button next to a rule name group to reorder it.

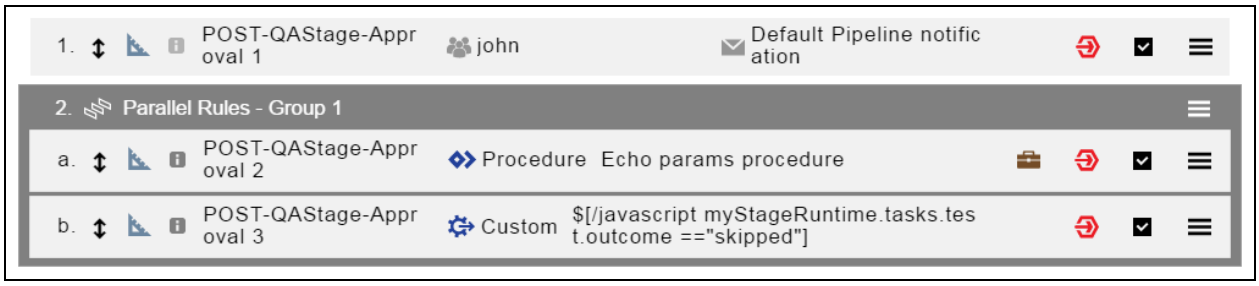
Unparallelizing Approval Rules

You can return parallel approval rules to sequential execution. To make a set of parallel rules run in sequence:

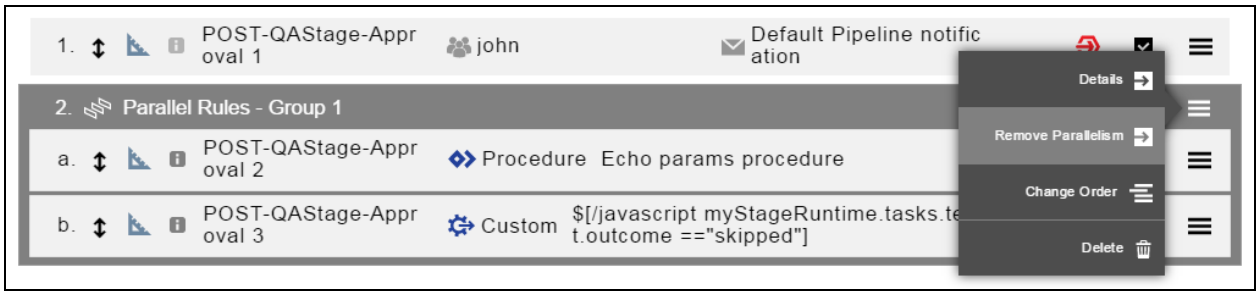
1. Go to the gate with the parallel rules.
2. Click the button on the gate.

The page to add or edit the approval rules for the gate opens:

3. Click to highlight the group of parallel rules.



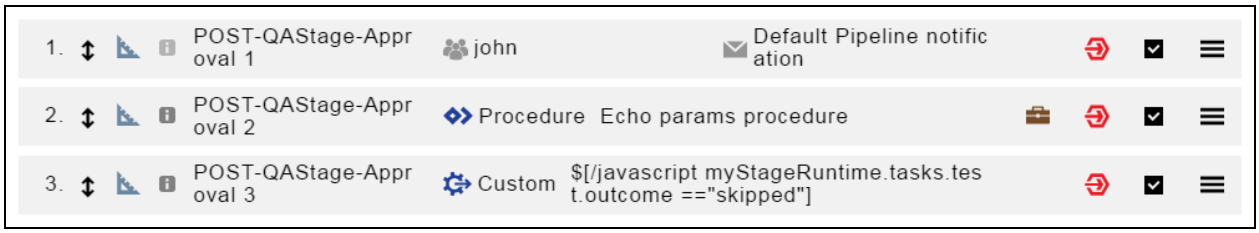
4. Click the menu ( ) button for the group, and then click **Remove Parallelism**.



A confirmation popup appears:

5. Click **OK**.

The rules will once again run in sequence:



Deleting an Approval Rule from a Gate

You can delete an approval rule from a gate.

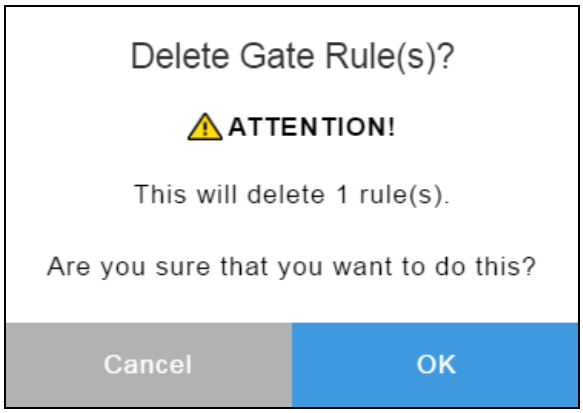
To do so:

1. Go to the gate with the parallel rules.
2. Click the button on the gate.

   The page to add or edit the approval rules for the gate opens.

3. Click the menu for the **POST-QAStage-Approval 1** rule and choose **Delete**.

A confirmation popup appears:



4. Click **OK**.

Running Pipelines

This section shows how to run a pipeline or restart a failed pipeline run. It also has examples showing different pipeline run scenarios.

Starting from the Pipeline Editor:

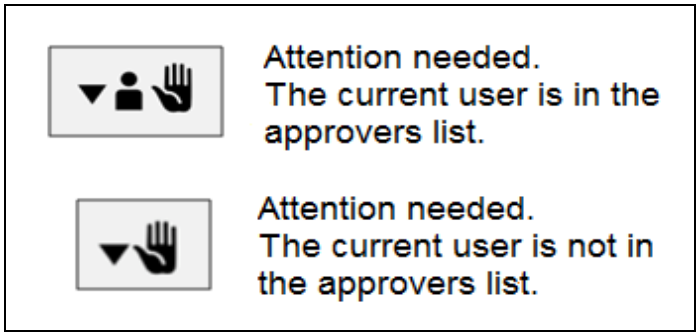
1. Click the **Run pipeline** button () to start a pipeline run.

   You can optionally select a previous pipeline run to use the parameters from that run. You can modify one or more of these parameters as described in the dialog box where the runtime parameters and settings are set.

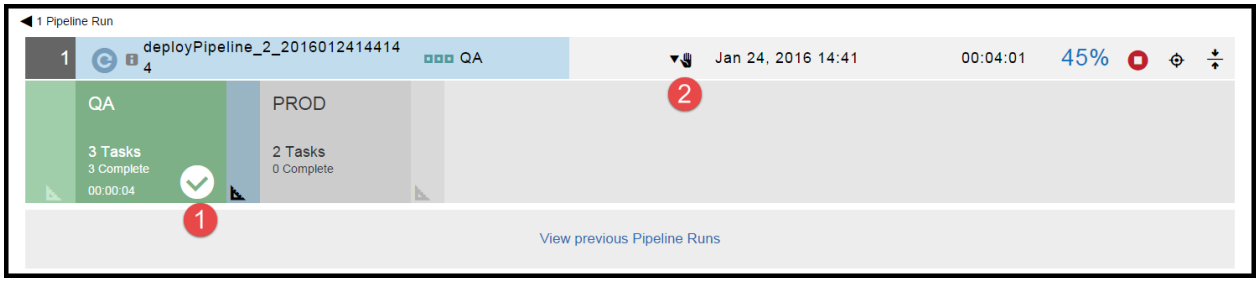
2. If the pipeline has required parameters to run the pipeline, enter the parameter values.

3. View the real-time progress of the pipeline run.

   Progress is show in percentage complete. These are the "Attention Needed" icons:

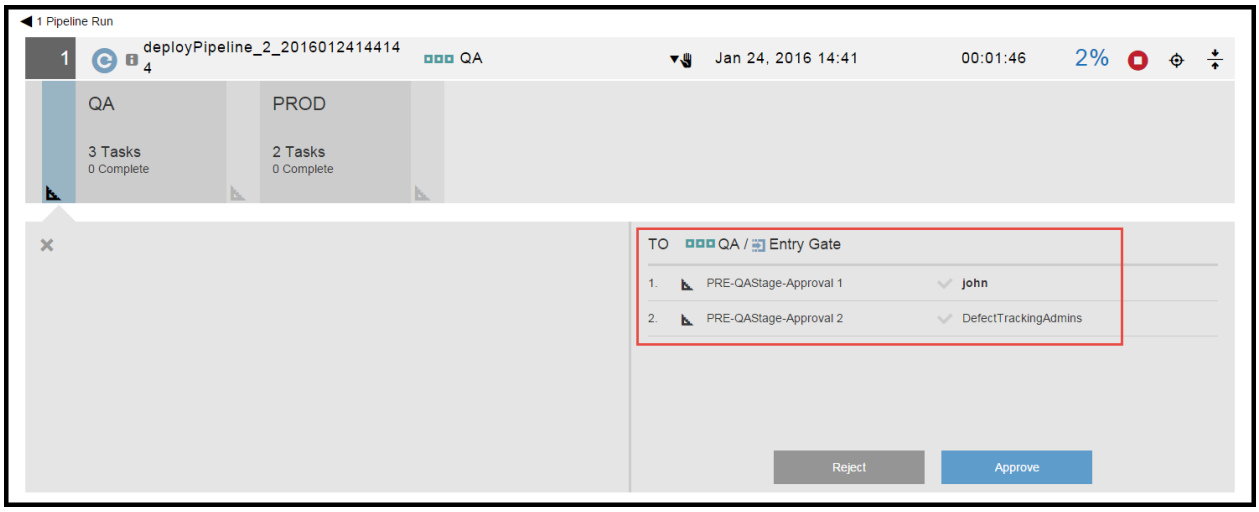


   The following example shows that exit gate of the QA stage and/or the entry gate of the PROD stage has gate approvals that need to be approved before the pipeline can enter the PROD stage.

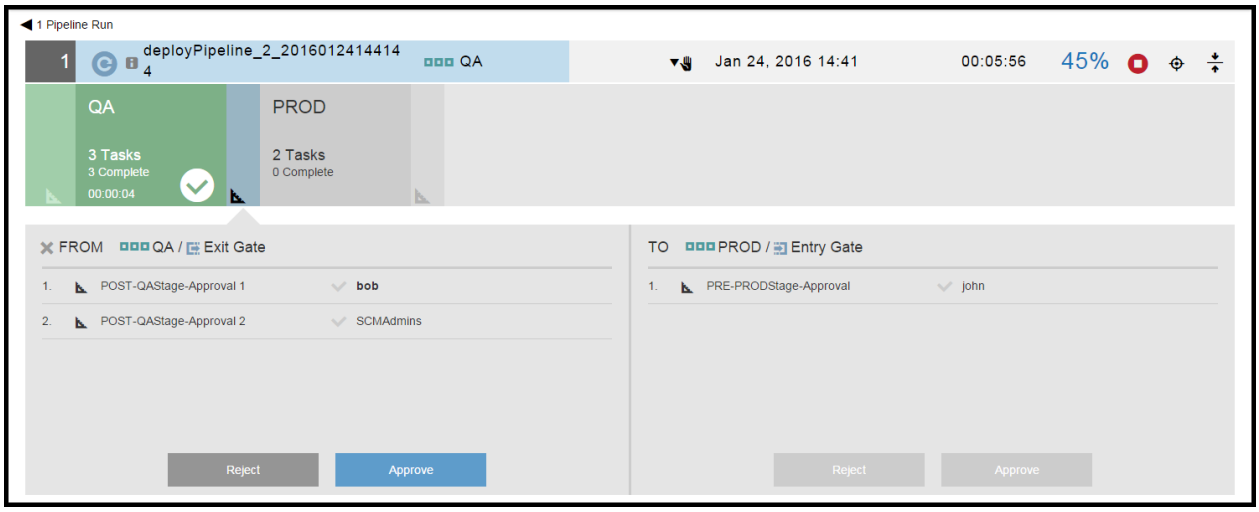


1	Click the  button to see the gate approvals and the required reviewers.
2	Click to see who the required reviewers are at the entry gate to the PROD stage.

   For the entry gate of the QA stage, the required approvers are the user named john and then the group named DefectTrackingAdmins.



For the exit gate of the QA stage, the required approvers are a user named bob (user) or a group named SCMAdmins and then a user named john.



4. Click in a stage to see the details of the pipeline stage summary.

   For examples of pipeline stage summaries, see Pipeline Stage Summary on page 603.

5. Click **View previous Pipeline Runs** to see the previous runs of this pipeline.

   For an example, see Example: Viewing Previous Pipeline Runs on page 540.

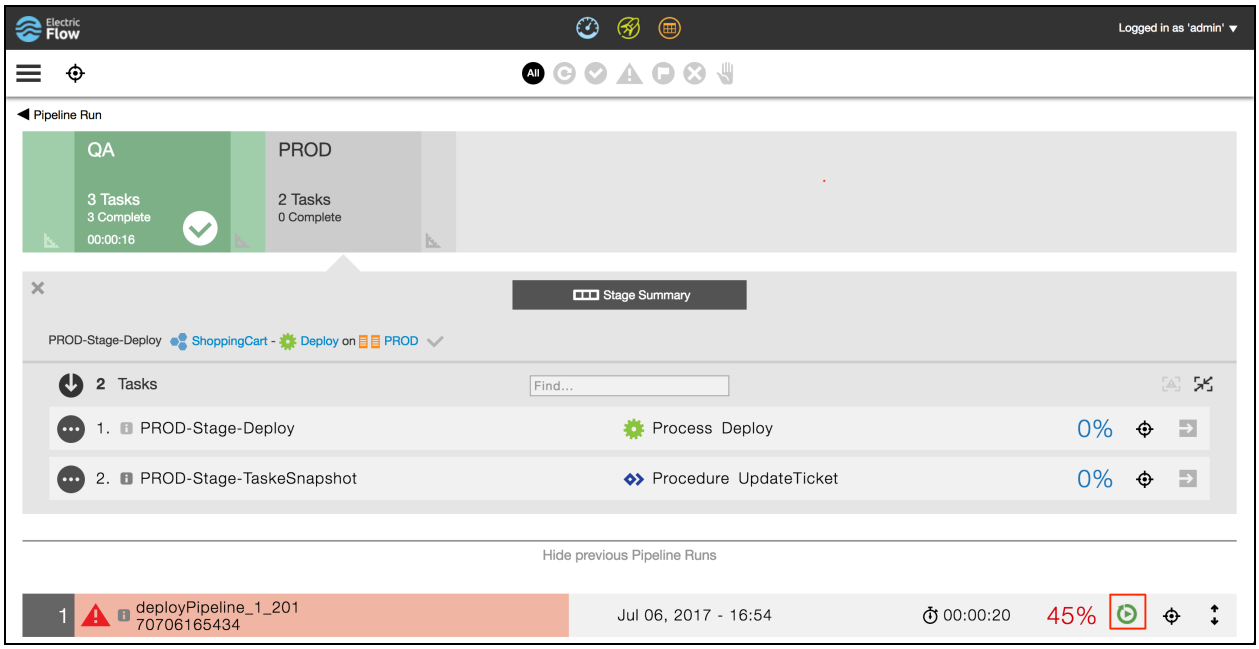
Restarting a Failed Pipeline Run

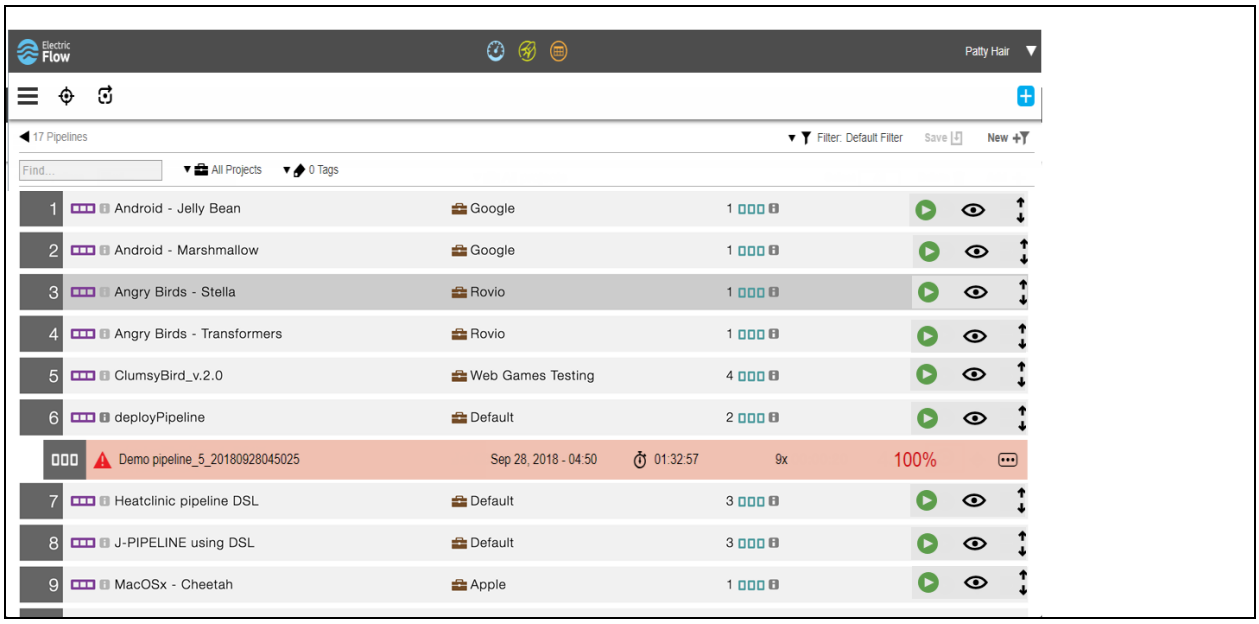
A restart capability for failed pipeline runs lets you configure a pipeline run to restart from the last failed task after you update the task definition and fix the issues that caused the failure.

A pipeline run is restartable if the last executed task in the run has a failure outcome. (If it is also the last task in a stage or gate, then its error handling must not be "continue on error.") Any prior failed tasks (in the same or previous stages or gates) with "continue-on-error" error handling enabled are

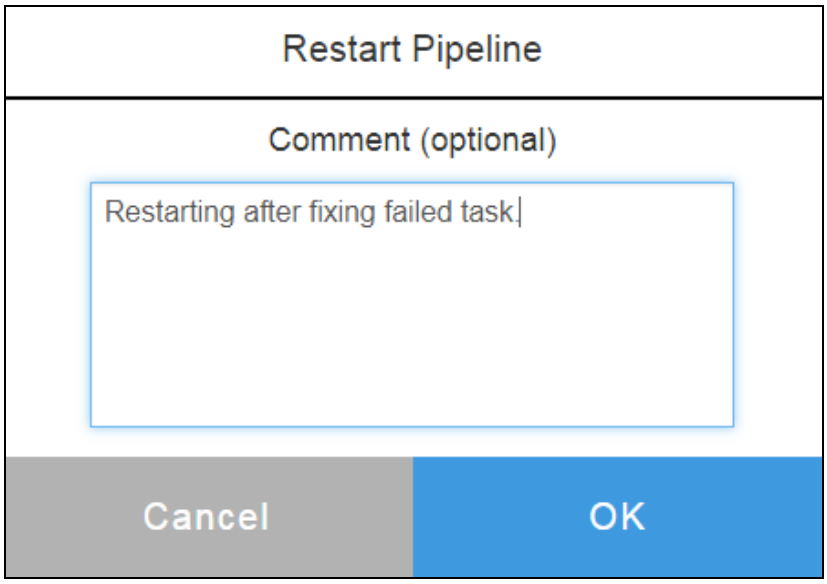
ignored. You can also restart a pipeline run if the last stage has no tasks in it and it is in error (because of a run condition/precondition evaluation error).

To restart a failed pipeline, you use the  (pipeline restart) button. For example:



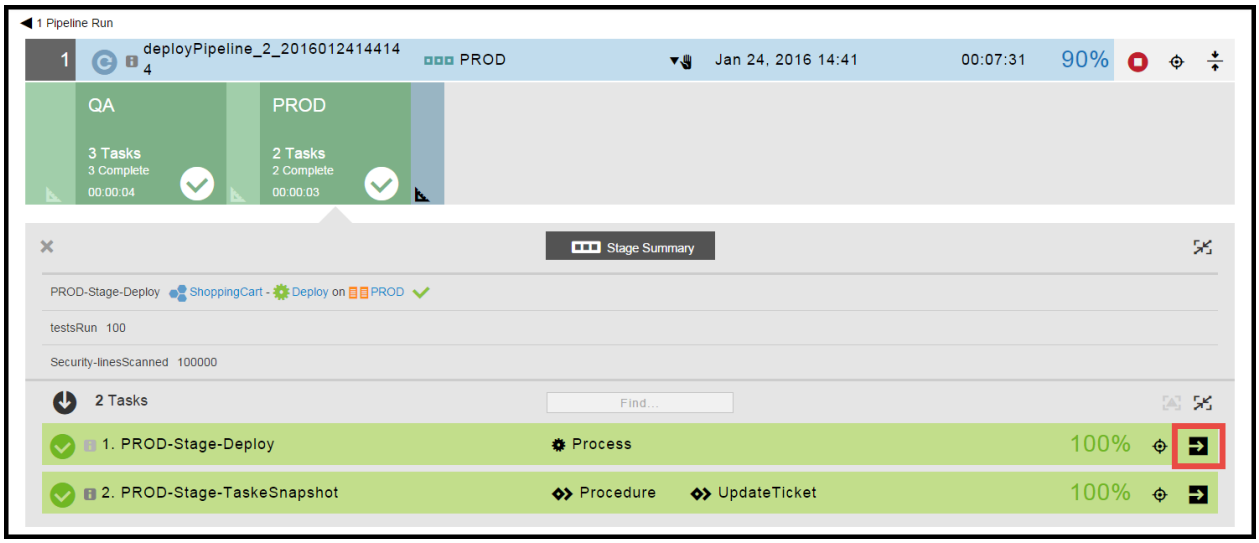


Restarting a pipeline prompts you enter a comment about the restart. For example:



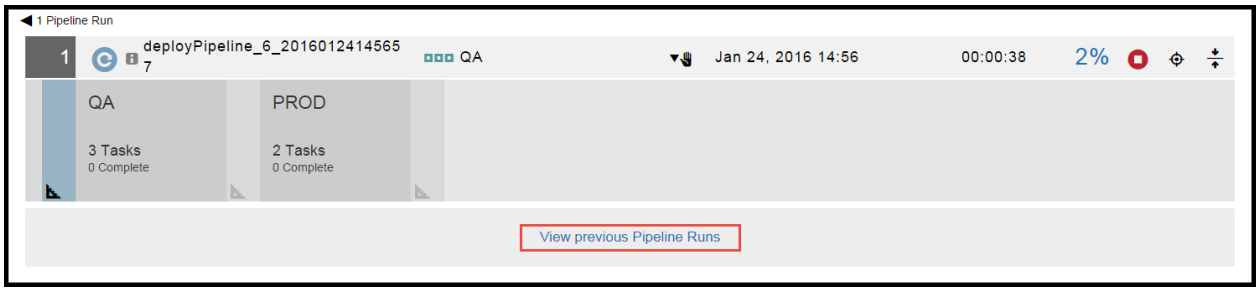
Viewing the Details of a Pipeline Run

For a task, click the ( ) button to see the Job Details page:

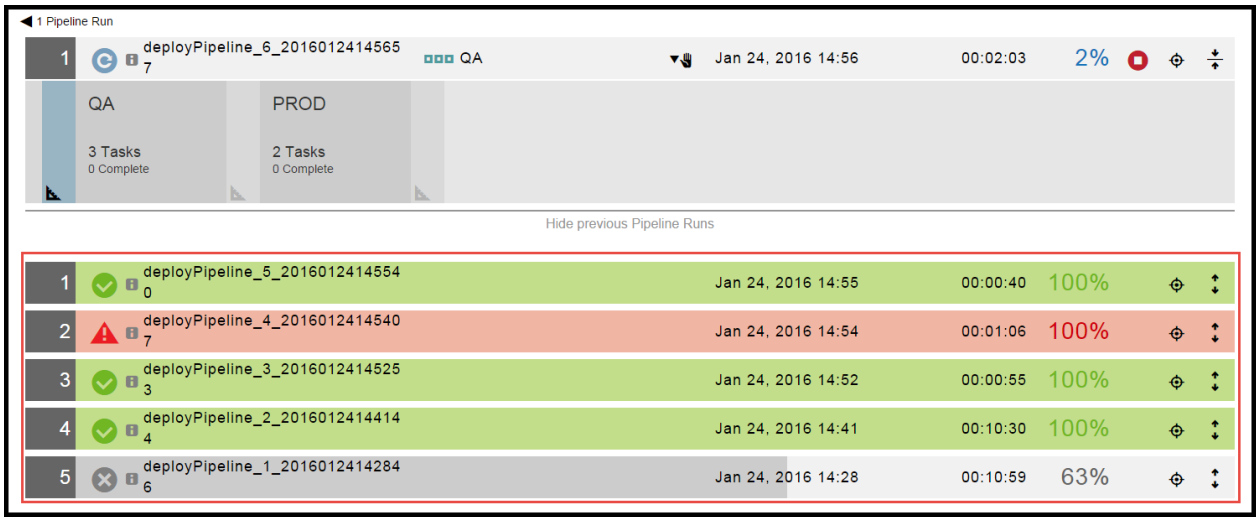


Example: Viewing Previous Pipeline Runs

Click **View previous Pipeline Runs** to see the previous runs of this pipeline.

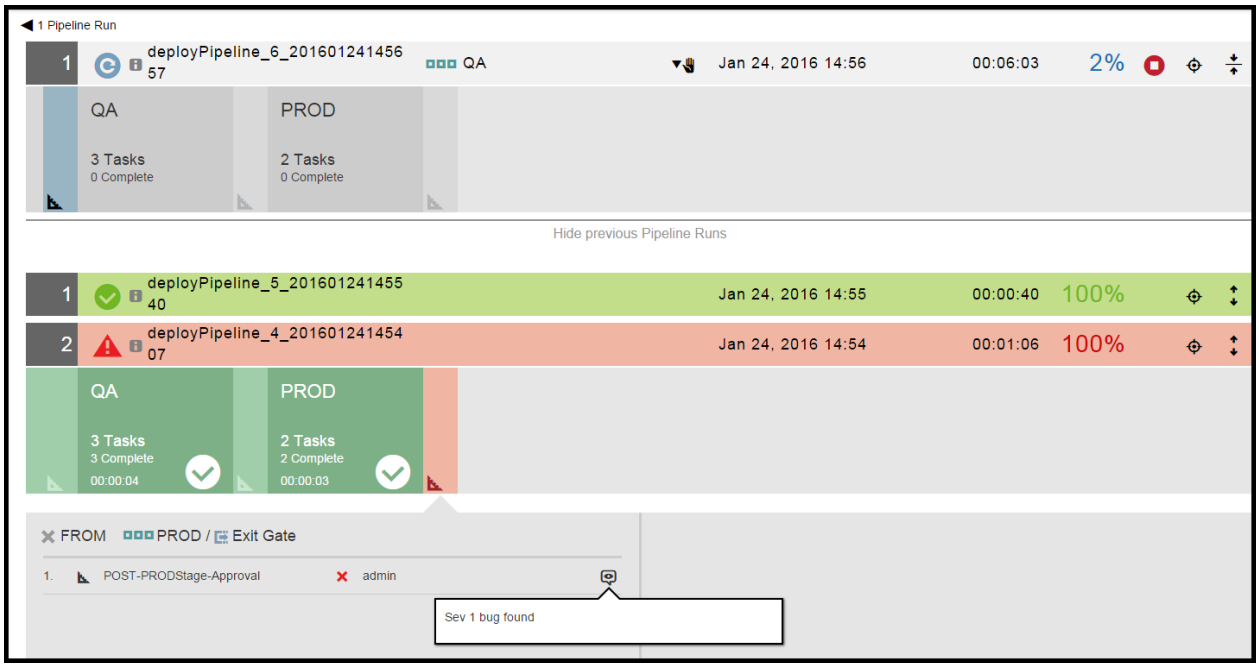


The previous pipeline runs appear.



Click the button for one or more previous pipeline runs.

In this example, the pipeline was not completed successfully. The approval at the exit gate of the PROD stage was rejected because a Sev 1 bug was found in the software release.



Viewing Pipeline Runs

The Pipeline Runs List summarizes the pipeline runs. Each row is an pipeline instance that has an icon and is color-coded to indicate the status. You can use filters and the Search field to find certain pipeline runs. See What's in the Pipeline Runs List on page 542 and Searching the Pipeline Runs List on page 545 for more information.

When you expand a row, you can view the details about the pipeline runs as well as the stage and task summaries for a pipeline instance. See Running Pipelines on page 536 for details.

Opening the Pipeline Runs List

1. Open the home page of the ElectricFlow web UI by browsing to https://<*ElectricFlow_server*>/flow/.
2. Go to the pipelines list in one of these ways:
   - Click **Pipelines**.
   - Click the **Main menu** button in the upper left corner and then select **Pipelines** > **All Pipelines**.

   The pipelines list opens.

By default, the list shows all the running or completed pipeline runs that you have the permission to view.

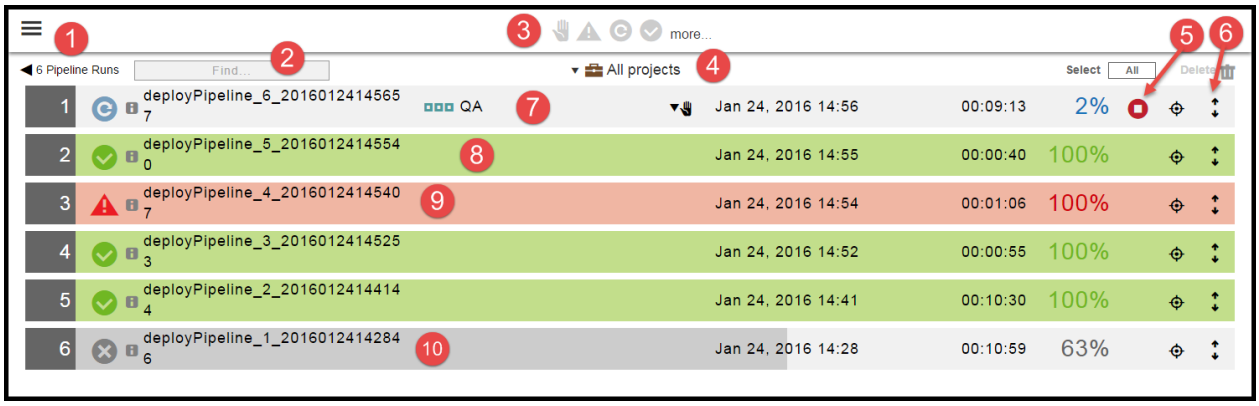
What's in the Pipeline Runs List

Each row shows the status of the pipeline run, the pipeline ID and name, the user running the pipeline, when the pipeline was run, and how long the pipeline run lasts.

For each pipeline run, you can take the these actions:

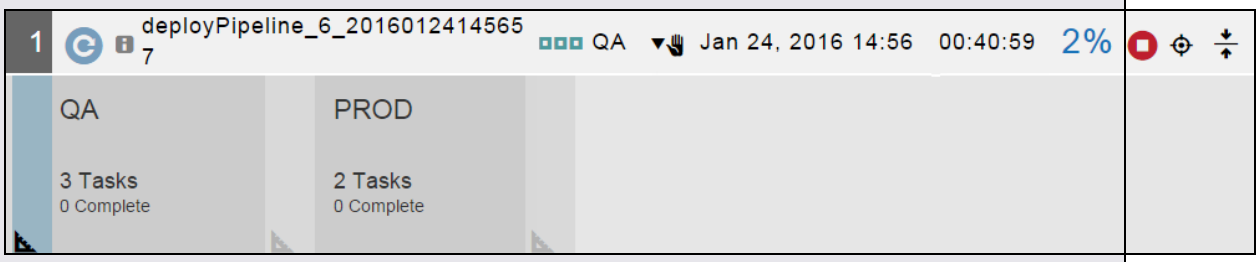
- Clicking the **Expand** button ( ) shows the pipeline run details and results of previous runs.
- Entering part or all of a pipeline name in the **Find** field (number 8 below) starts the search for pipelines matching the criteria. See Searching the Pipeline Runs List on page 545 for how to search in the Pipeline Runs List.
- Clicking on one of the filters (number 9 below) shows the pipeline runs whose status matches the filter. See Searching the Pipeline Runs List on page 545 for how to search in the Pipeline Runs List.

This is the Pipeline Runs List:



1	Breadcrumb identifying the object you are viewing and the total number of pipeline runs.
2	**Find** field. Enter the search criteria to find specific pipelines.

3	Filters presenting the status of the pipeline run. •  Needs attention. Manual approval may be required to complete the current stage and go to the next stage •  Completed with errors •  Running •  Completed with success •  Completed with warnings. The pipeline run could still be running, or it may have completed running •  Pipeline aborted. When you click the **Abort** button during a pipeline run, the pipeline will be aborted after the currently running tasks finish and no new tasks will be started.
4	Clicking the down arrow next to **All Projects** opens a drop-down menu. It shows the pipeline runs for a specific pipeline. The default is to show the pipeline runs for all projects. If you want to see only the pipeline runs for a specific project , click the down arrow in the **All projects** field to select one or more projects one of these ways: • Click on the name of one or more projects. • Enter the search criteria in the Search field. The projects that match the criteria appear in the list. You can also search for one or more pipeline runs by entering the search criteria in the **Search** field next to the **All projects** field. If there are no matches, a message appears stating that there are no pipeline runs in the selected projects.
5	Clicking the **Abort** button causes the current task to complete. No new tasks are started, and the current pipeline run stops.

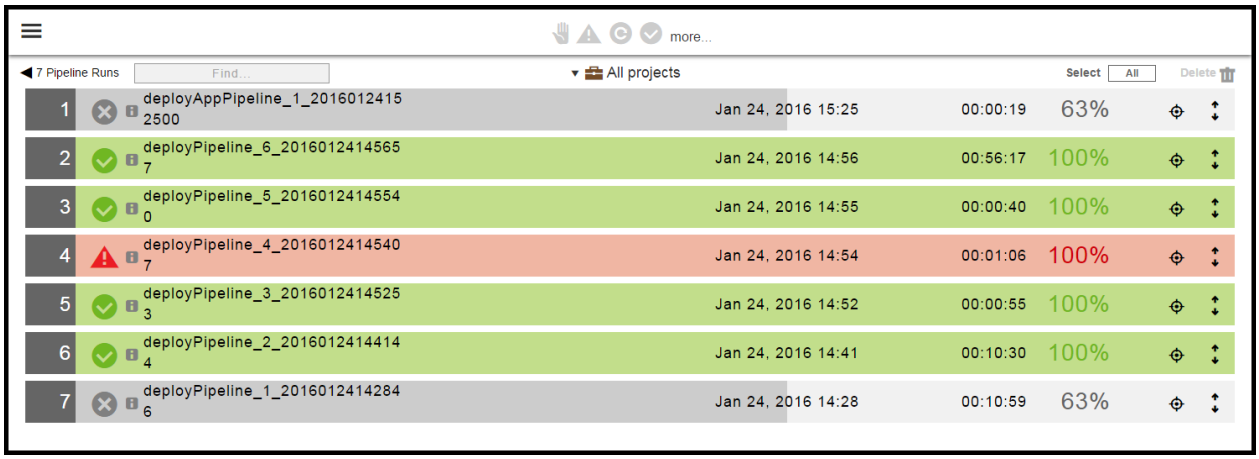
6	Click the  button to show more details about the pipeline run. Clicking it in this row shows more details about the pipeline run. 
7	This pipeline run is in progress. This shows that the pipeline is in the QA stage, an approver needs to do something to keep the pipeline running, when the pipeline is running and how long, and the percentage complete. You can abort the pipeline (number 5 above) and also see more details about this pipeline run (number 6 above).
8	The pipeline run was completed successfully.
9	The pipeline run was completed with errors.
10	The pipeline run was aborted.

Searching the Pipeline Runs List

To focus on specific pipeline runs, ElectricFlow provides several ways to quickly filter and search for pipeline runs, using a combination of filters and the **Find** field. You can also use these methods with the search operations on other ElectricFlow features.

- Filtering with Filter Icons on page 546
- Searching With the Find Field on page 547
- Filtering by Projects on page 547
- Filtering and Searching With the All Methods on page 549

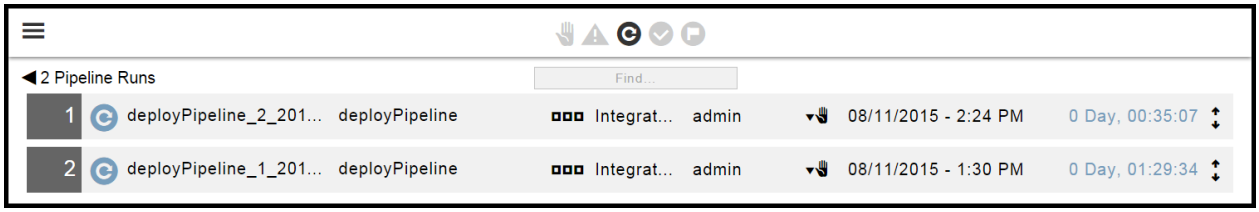
When no filter icons are selected, the **Find** field is empty, and **All projects** is selected, all of the pipeline runs appear in the list, such as:



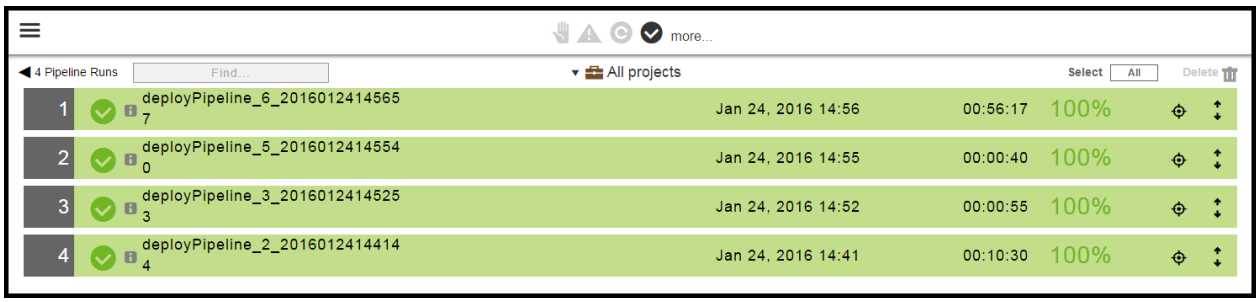
Filtering with Filter Icons

When you select a filter, the Pipeline Runs List is updated and shows only the pipeline runs with a status matching the filter. These examples show how to filter pipeline runs using the filter icons:

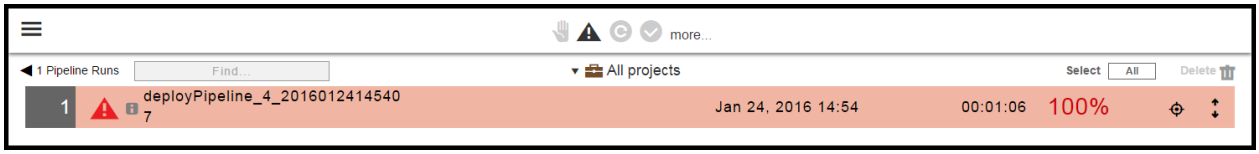
- Running



- Competed with success



- Completed with errors



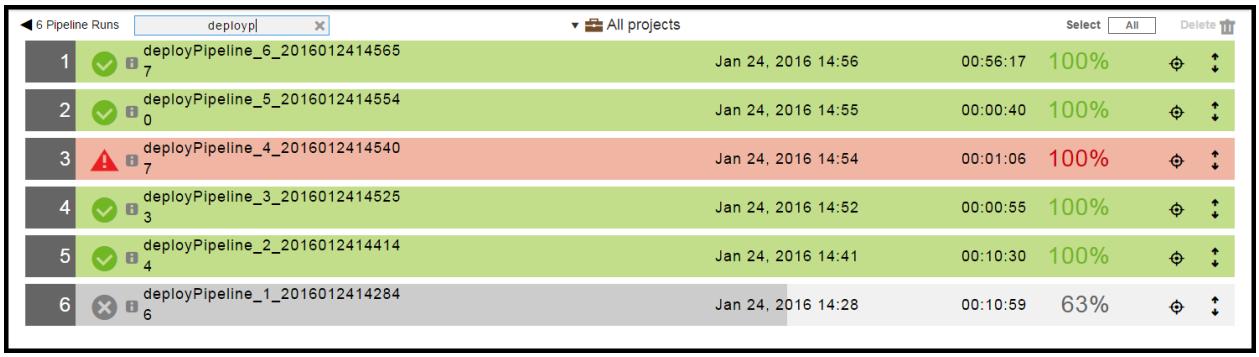
- Pipeline aborted. When you click the **Abort** button during a pipeline run, the pipeline will be aborted after the currently running tasks finish and no new tasks will be started.

Searching With the Find Field

Start typing the name of the pipeline in the Find field.

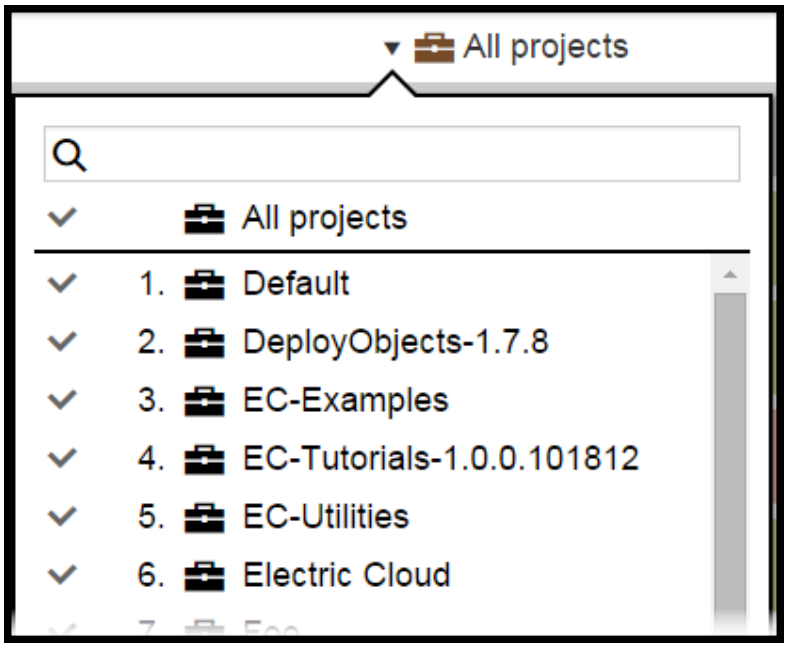
For example, if you enter "my" as the search criteria, there are no pipelines that match the search criteria. The Pipeline Runs Lists returns with nothing in the list.

To find a pipeline starting with "deploypip" in the Pipeline Runs List, start to enter "deploypip" and notice that ElectricFlow updates the list and shows the pipelines with names that match the search criteria.



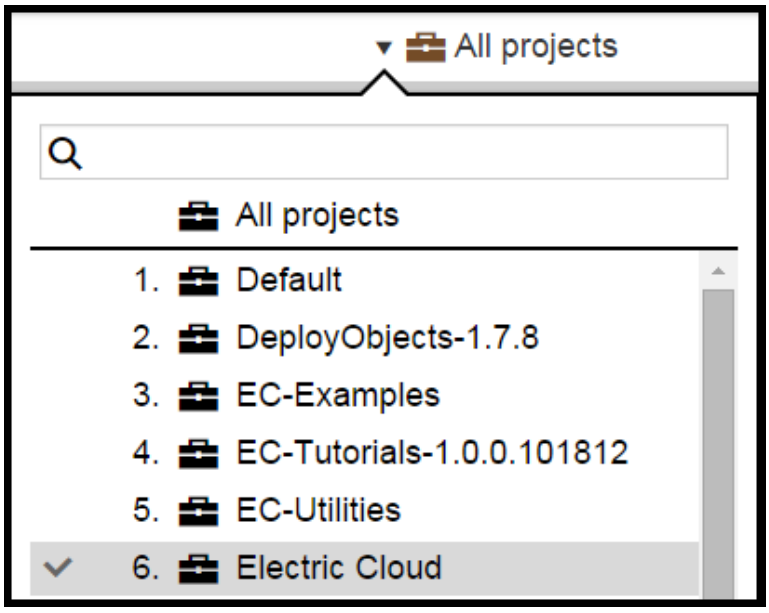
Filtering by Projects

Clicking the down arrow next to the **All projects** opens a drop-down list of all the projects you have permission to access.

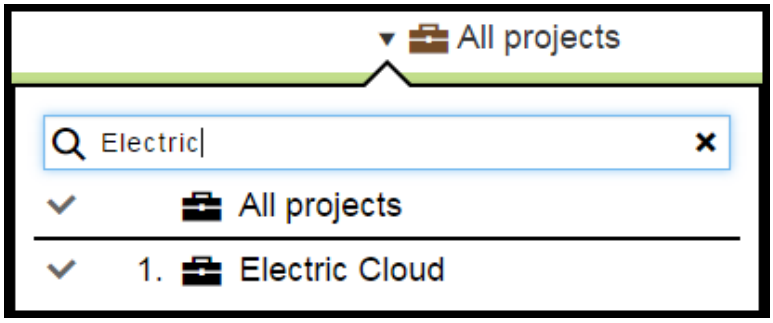


You can select the pipelines for specific projects one of these ways:

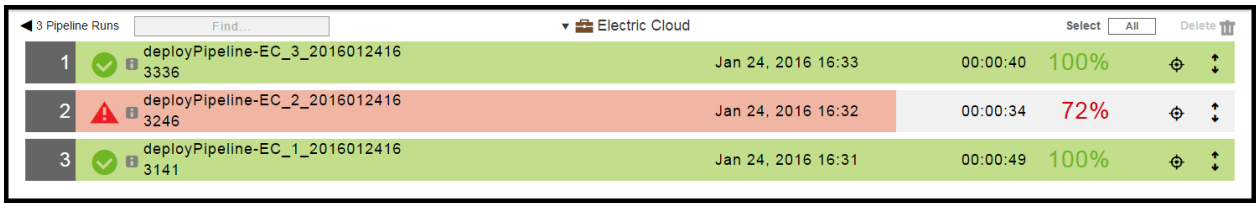
- Select one or more specific projects in the drop-down list, and then click outside the drop-down list.



- Enter search criteria to find a specific project, and then click outside the drop-down list.



The pipeline runs for the pipeline in the Electric Cloud project now appear in the Pipeline Runs List.



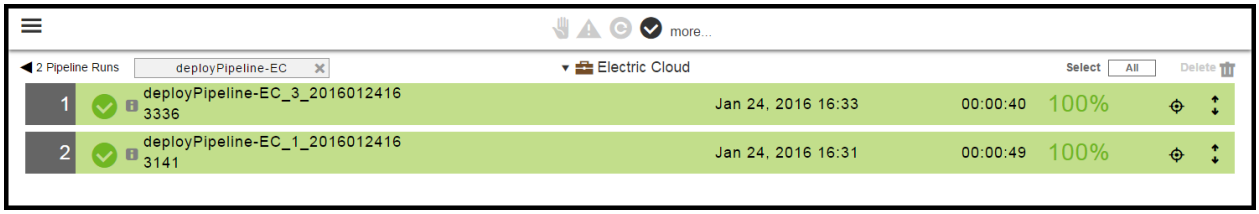
Filtering and Searching With the All Methods

You can use all three methods to search for specific pipeline runs.

This example shows how to do this:

- Select (Completed with success) .
- In the **Find** field, enter "deployPipeline-EC".
- Select the "Electric Cloud" project.

The pipeline runs that were completed successfully, start with "deployPipeline-EC", and are in the "Electric Cloud" project now appear in the Pipeline Runs List.



Troubleshooting Pipelines

This section shows how to troubleshoot your pipeline runs. If your pipeline result is not green, you may have a problem that you want to resolve. This topic describes some of the common steps that you may

need to use when determining where there are issues in the pipeline process.

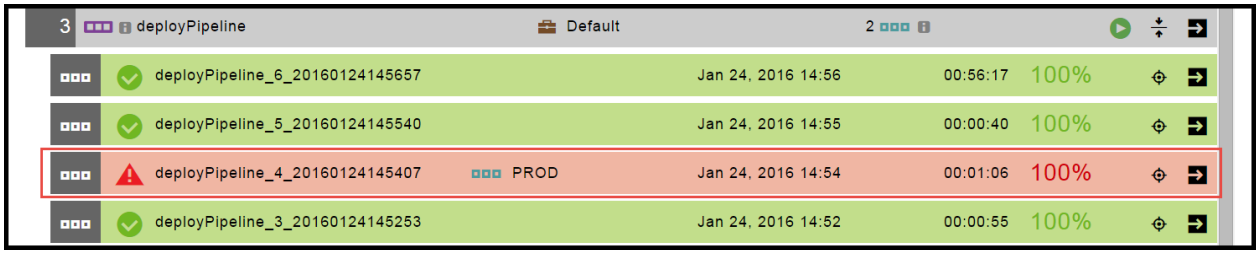
Example: Error Due to Gate Approvals

Starting in the Pipelines List:

1. Choose a pipeline to troubleshoot.

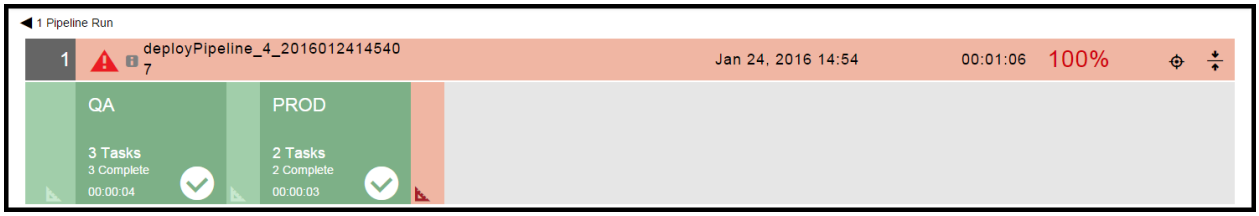
   In the example in this topic, choose the deployPipeline pipeline.

2. To view details about a specific pipeline run, choose a run and click the **View Details** button in the row.



Clicking the **View Details** buttons on any task will show you the details of the underlying work that was done to complete the task.

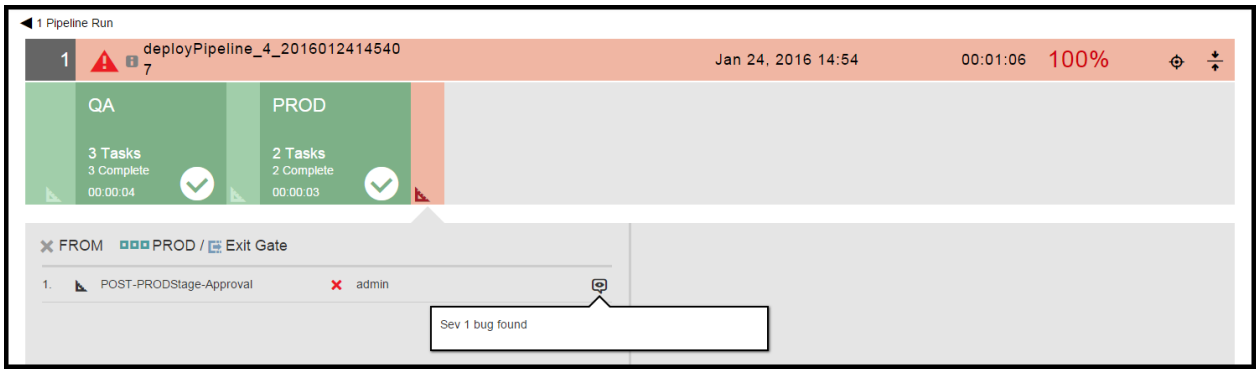
The pipeline run summary appears. There is an error in the exit gate of the PROD stage.



3. To get more information about the exit stage, click in the exit gate.

   Clicking in the exit gate will show the approval status of gate.

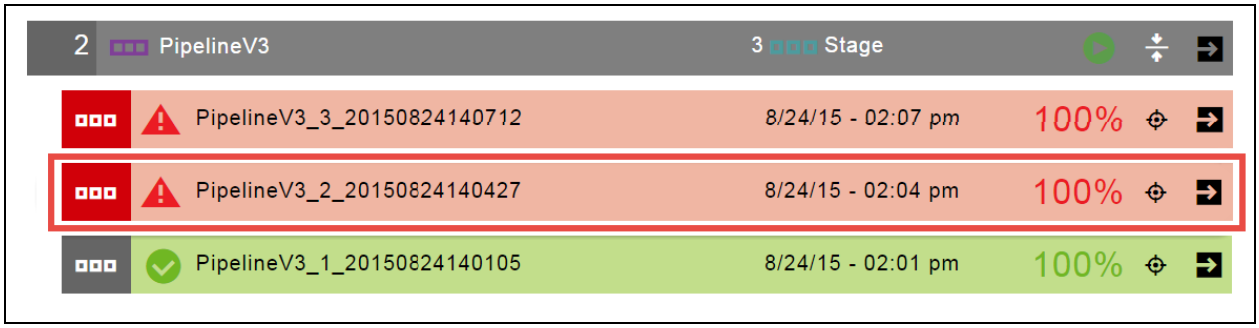
   In this example, the pipeline was not completed successfully. The approval at the exit gate of the PROD stage was rejected because a Sev 1 bug was found in the software release.



Example: Error in a Pipeline Stage

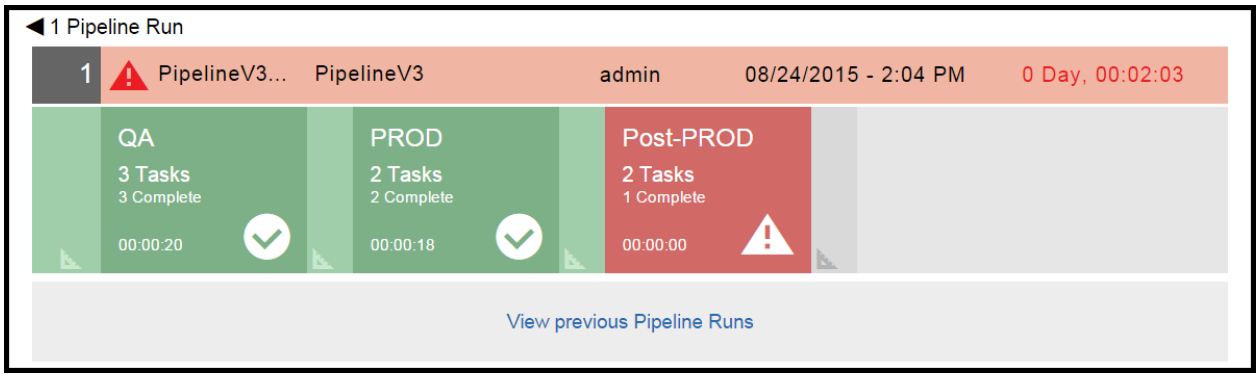
Starting in the Pipelines List:

1. Choose a pipeline to troubleshoot.
2. To view details about a specific pipeline run, choose a run and click the **View Details** button in the row.



Clicking the **View Details** buttons on any task will show you the details of the underlying work that was done to complete the task.

The pipeline run summary appears. There is an error in the Post-PROD stage.

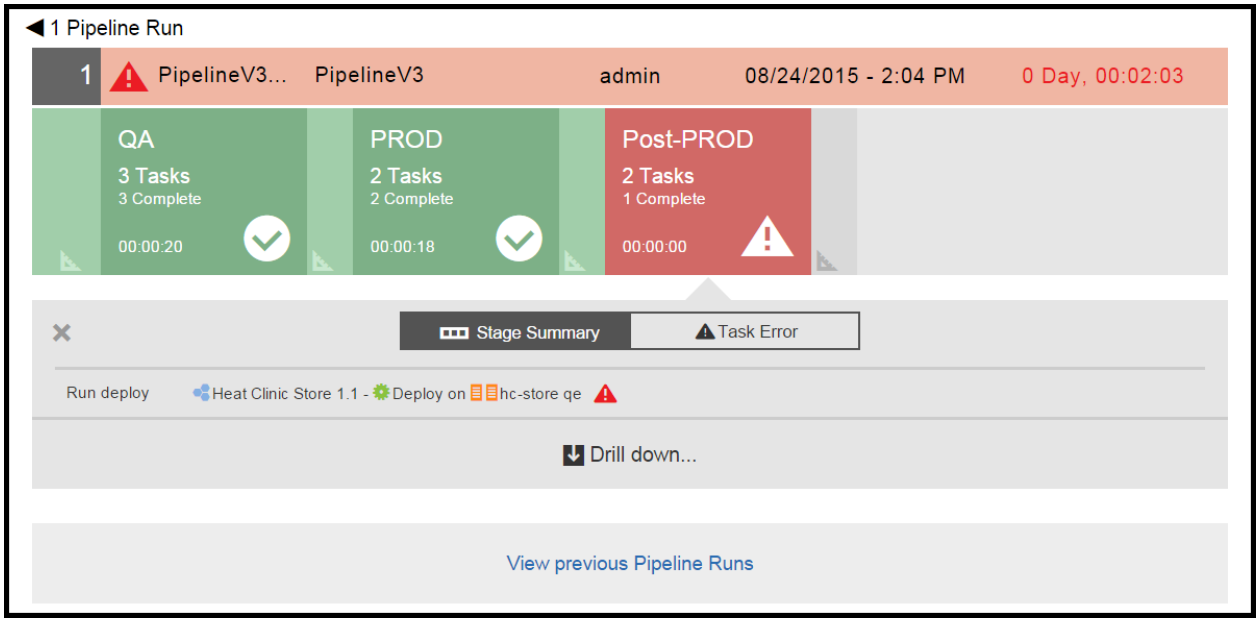


3. To get more information about the Post-PROD stage, click in the stage.

   Clicking in the Post-PROD stage will show the Stage Summary and Task Error details.

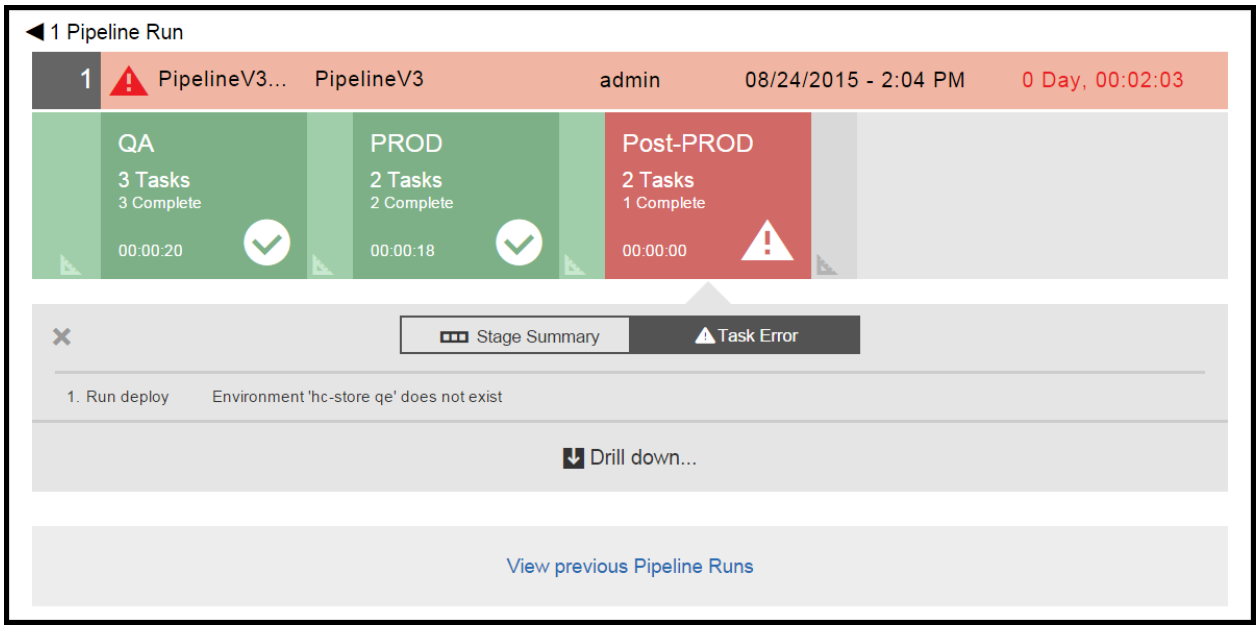
   The Stage Summary shows the task definition.

   In this pipeline, the task is an application process.

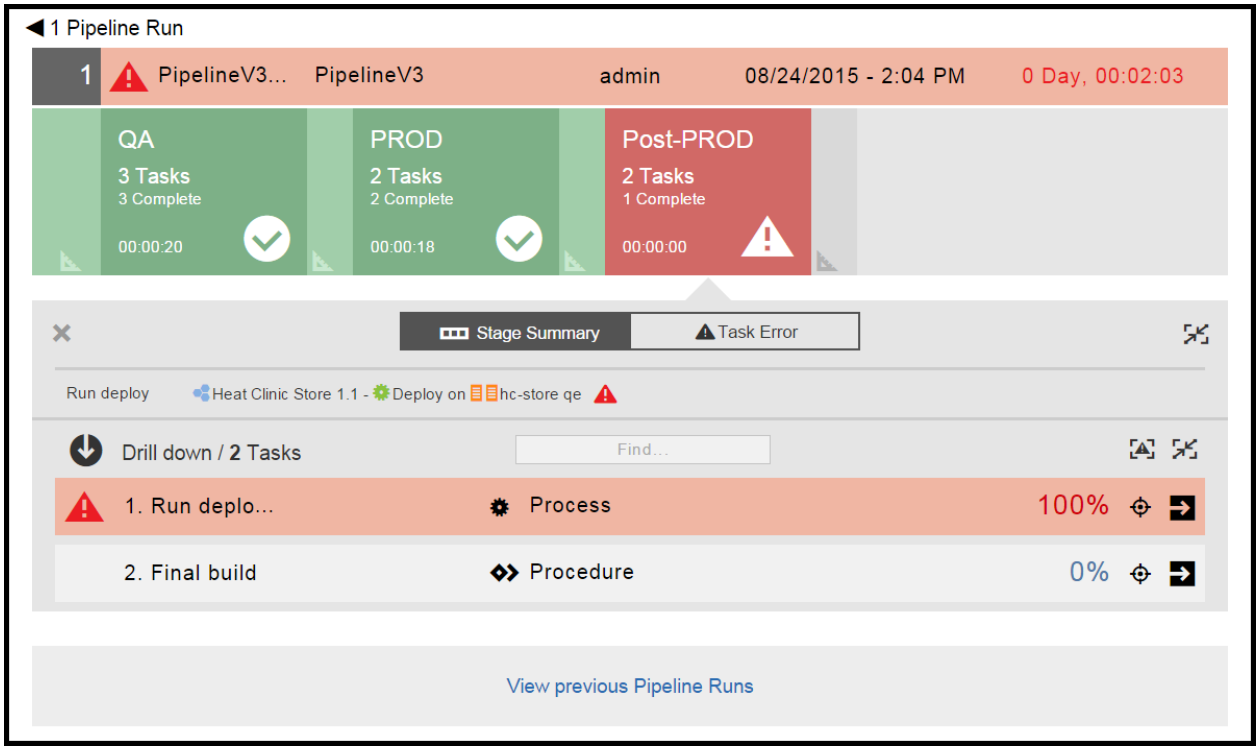


   The Task Error information appears.

   The error occurred in the "Run deploy" task. The problem is that the environment to which the application is mapped does not exist.



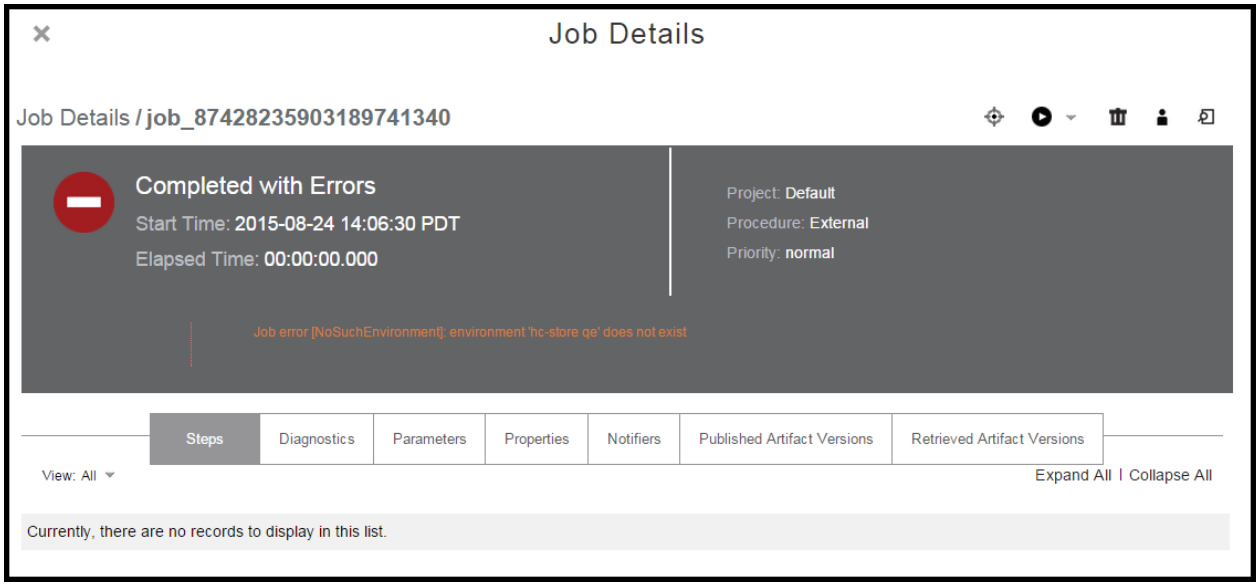
4. Click **Drill down** to see more stage details.



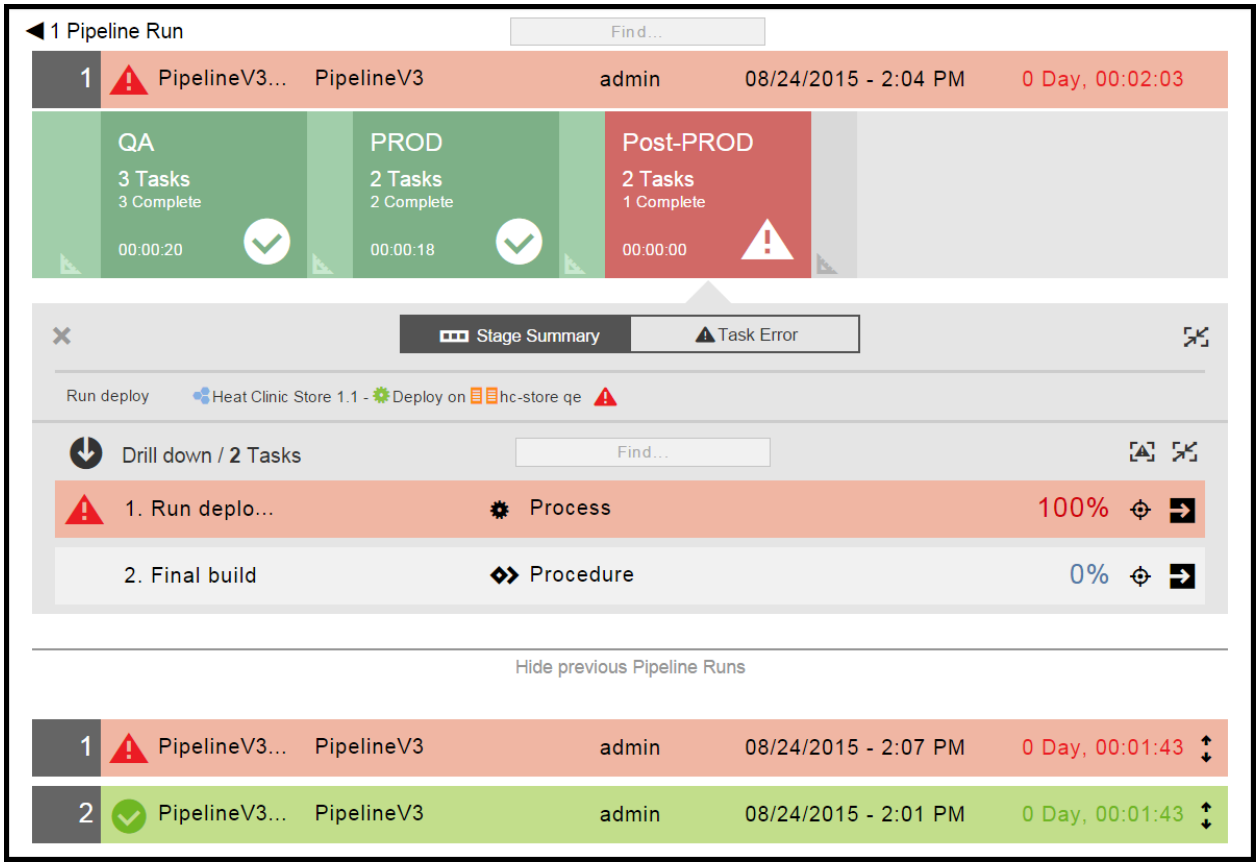
Notice that the "Run deploy" task failed and that the "Final build" task never ran because of the "Run deploy" task.

5. Click the **View details** button.

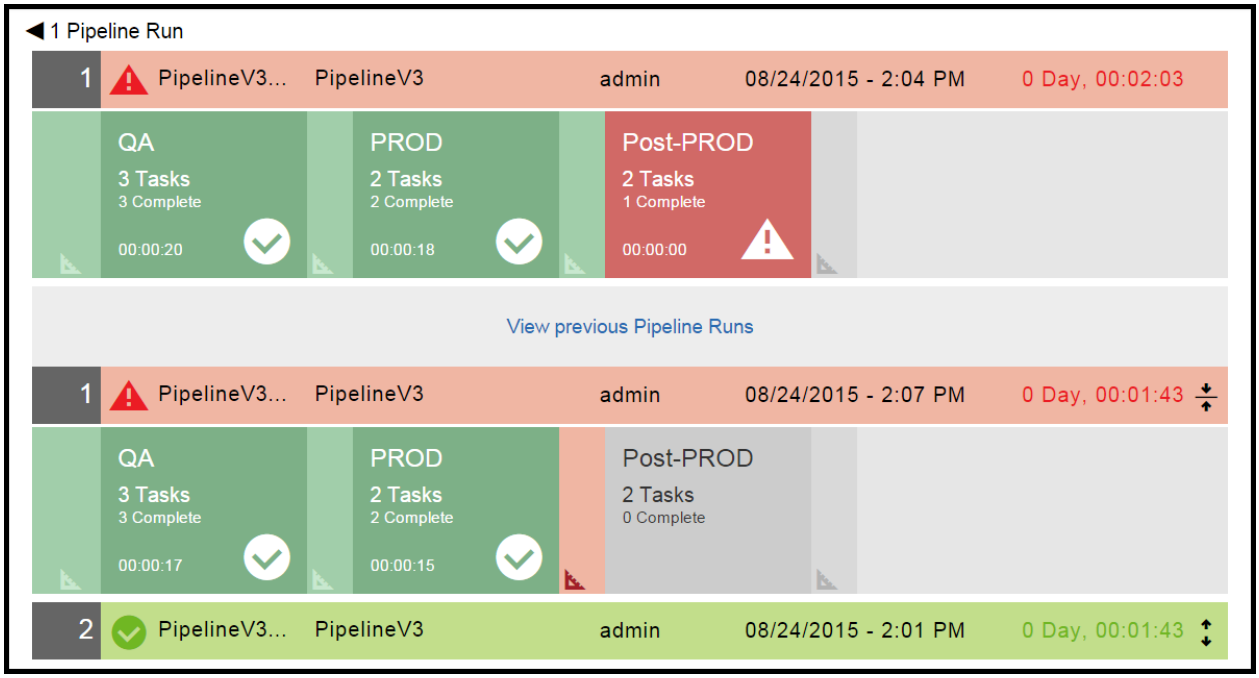
   The Job Details page opens and shows that the "Run deploy: tasks was completed with errors. The problem is that the environment in the task definition does not exist.



6. Click **View previous Pipeline Runs** to see the previous runs of this pipeline.



7. Choose a pipeline in the Previous Pipeline Runs list, and click the **Expand** button.



   The pipeline run summary for that pipeline appears.

8. Repeat Step 3 to Step 7 to do more troubleshooting.

Example: Authoring a Pipeline with Manual and Utility Tasks

This example shows how to add tasks to a pipeline while defining a stage. The tasks include a manual task and a utility task. You add a manual task to a pipeline while defining a stage.

Authoring a Pipeline with Manual and Utility Tasks

1. Choose a stage in the pipeline.
2. Click the **Define stage tasks** button in the lower right corner of the stage.

   The task list opens. When the task list has one or more items, it shows the tasks in sequential order for the stage.

3. Add a task to the stage.
4. Enter the name of the new task in the **Name** field, and enter an optional description in the **Description** field.

5. Set the position of the task.
   a. Select **Before** or **After**.
   b. Click the down arrow to select the task that the new task will precede or follow.

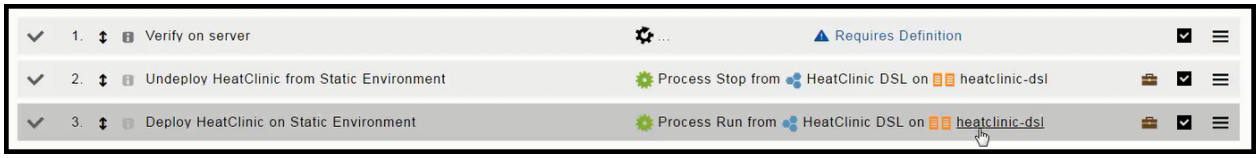
      Example:

6. (Optional) If you want to add another task, click **+ Add another** and repeat the previous two steps to add it.

   Otherwise, click **Done**.

   The list shows the tasks in the order that they will be executed.

   Example:



7. Click **Requires Definition** for the task that you added.

   The dialog box to define the task opens.

8. Enter the settings to define the task as a manual task, and click **OK**.

   Example:

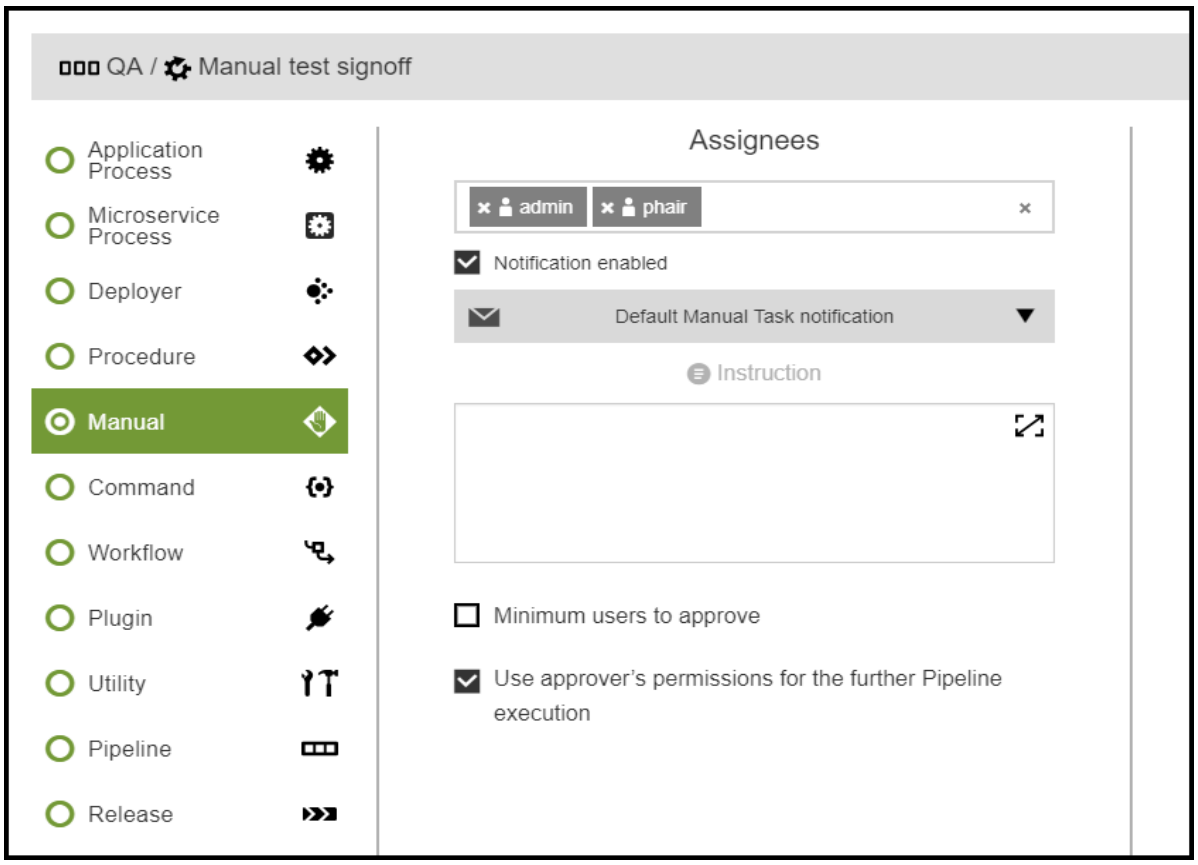
   a. Select **Manual** to define the task as a manual task requiring human action in the **Instruction** field.

      The instructions can include property references using the "`$[]`" notation for file links, which is needed when referencing a task instruction in an email notification template. For example, a manual task could have this instruction:

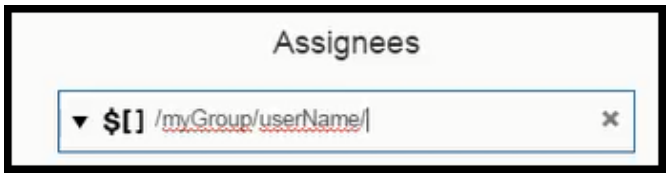
      Check the Clover report, and make sure there is at least 80% code coverage before you approve.
      http://myserver.com/reports/$[/myPipelineRuntime/ec_summary/report_name]

b. Click in the **Assignees** field and enter the name of the users or groups assigned to the task, and select one or more users or groups.

Example:



You can also use the $[ ] notation to specify the parameter



c. Check **Notification enabled**, as appropriate. If this is checked, select a **Default Manual Task notification** from the pulldown.

d. Check **Minimum users to approve**, as appropriate. Enter a positive integer.

e. Check **Use approver's permissions for the further Pipeline execution**. This overrides access control already defined for the task.

f. Click **OK**.

The task list now shows the task as a manual approval requiring human intervention.

Example:

9. Click  to add another task to the stage.
10. Enter the name of the new task in the **Name** field, and enter a description in the **Description** field.
11. Set the position of the task.

    Example:

12. Click **Done**.

    The task list now shows the new task.

    Example:

13. Click **Requires Definition** for the task you added.

14. Enter the settings to define the task as a utility task, and click **OK**.

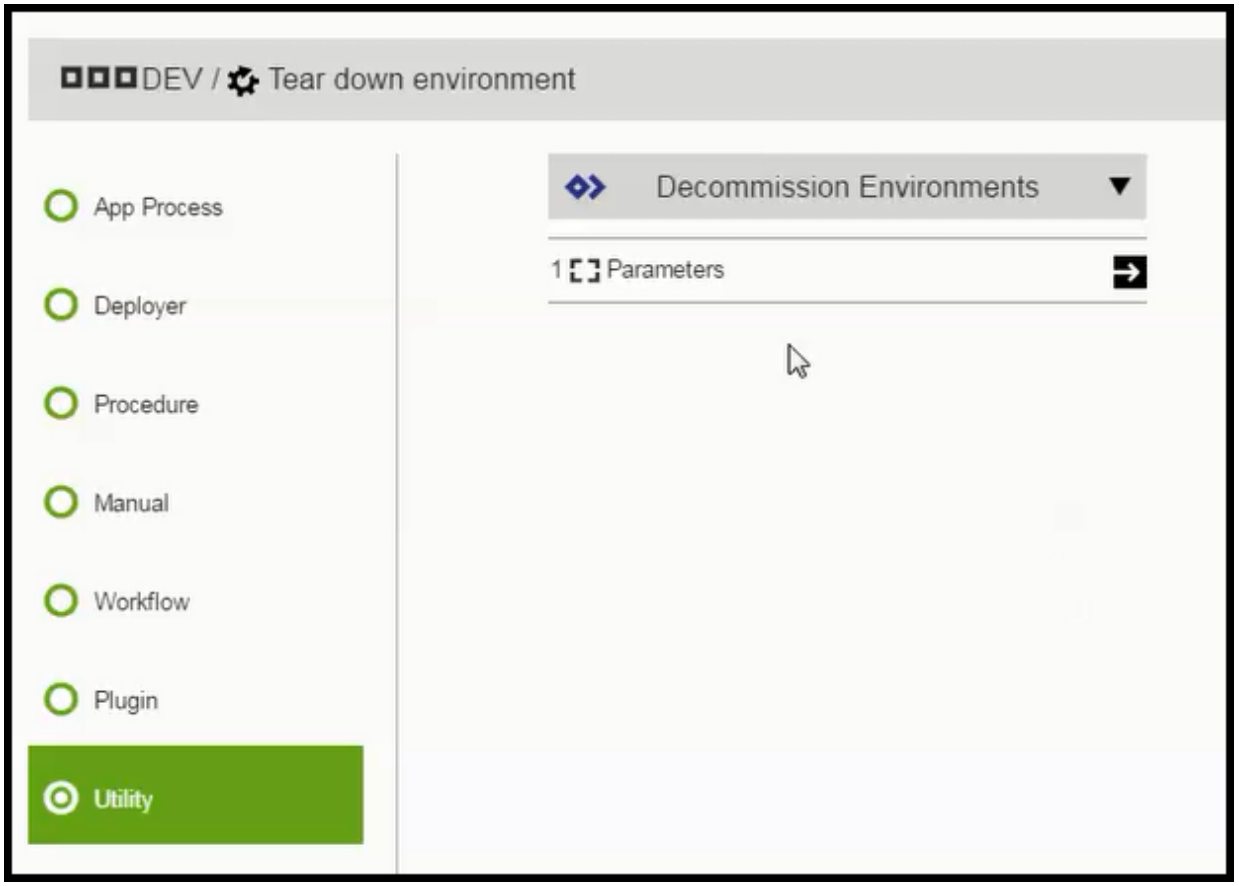
    A utility task, also called a utility function, is a higher-order operation than a third-party plugin that you can use in pipeline modeling.

    a. Select **Utility**.

    b. Click **Select procedure** > **Decommission Environments**.

    c. Click the  button the **Parameters** field to enter the parameters for the selected utility function.

    Example:



    The dialog box to enter the parameter values opens.

d. Enter the parameter values.

   Enter a comma-separated list of environments to tear down. You must enter each environment in the form of `/projects/<project_name>/environments/<environment_name>`.

   Example:

e. Click **OK**.

   The task list now shows a manual task as the first task and a utility task as the last task in the stage.

   Example:

Plugin Pipeline Tasks

Plugin tasks let you orchestrate third-party tools at the appropriate time in your pipeline. The following third-party tools can be used with ElectricFlow:

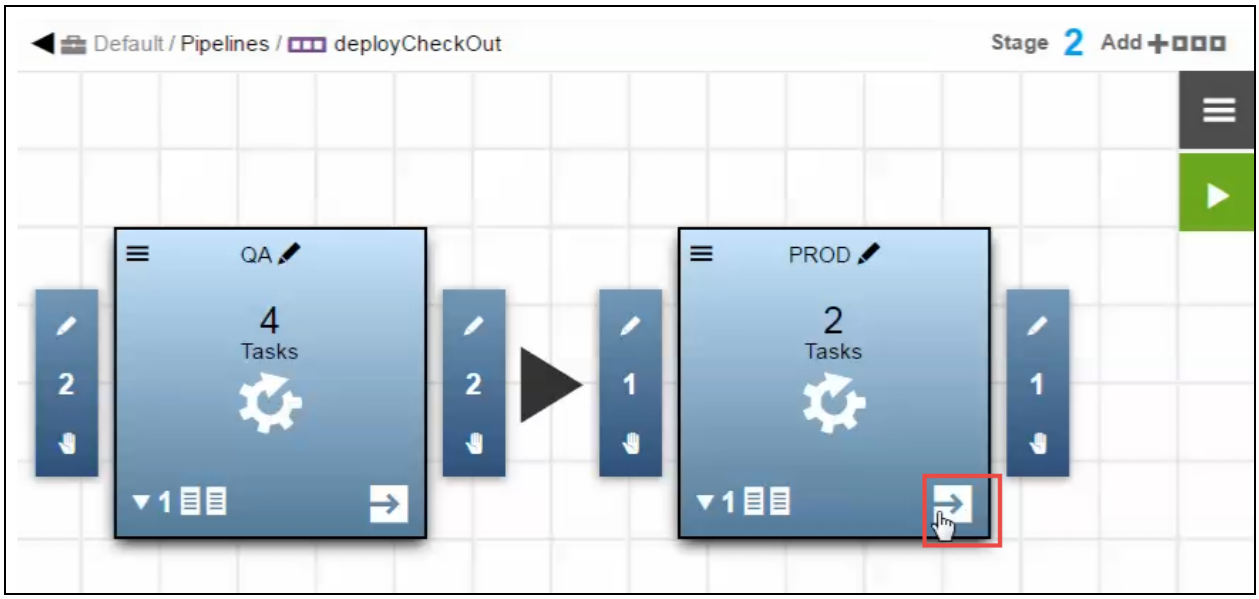
- Application Servers such as IIS, JBoss, and Tomcat
- Builds such as EMake, Maven, or Visual Studio
- Code Analysis such as Clover CMD, Coverity, or Klocwork
- Databases such as DBI, Oracle, or SQLServer
- Defect Tracking such as Bugzilla, JIRA, or Rally
- Notification such as Twitter
- Reporting such as Reports
- Resource Management such as Chef, EC2, or OpenStack
- Scripting/Shell operations such as Groovy, Python, or Ruby
- Source Code Analysis such as ECSCM-ClearCase, ECSCM-Git, or ECSCM-Perforce

- System such as Artifact or FileSysRepo
- Test such as HPQualityCenter, Jasmine, or Selenium
- Utility such as FileOps or SendEmail.

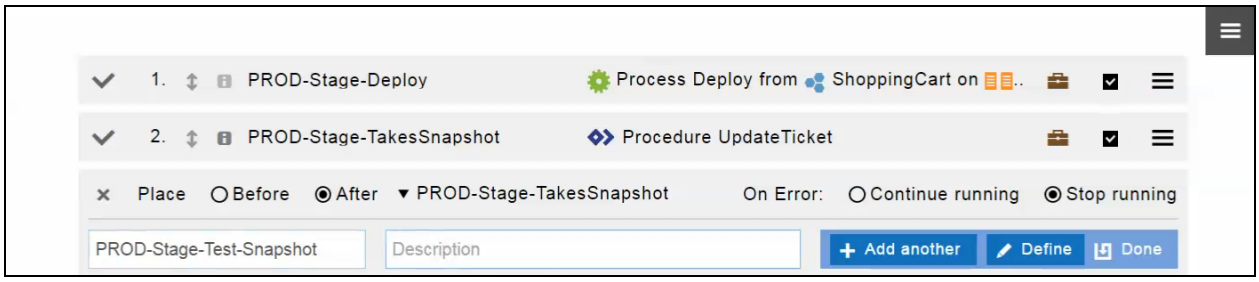
Plugin Tasks in the Pipeline Task View

This example shows how to use the Pipeline Task View to add a task to a pipeline stage and define it using a test plugin.

1. In the Pipeline Editor for a new pipeline, click the **Define stage tasks** button in the lower right corner of the PROD stage:

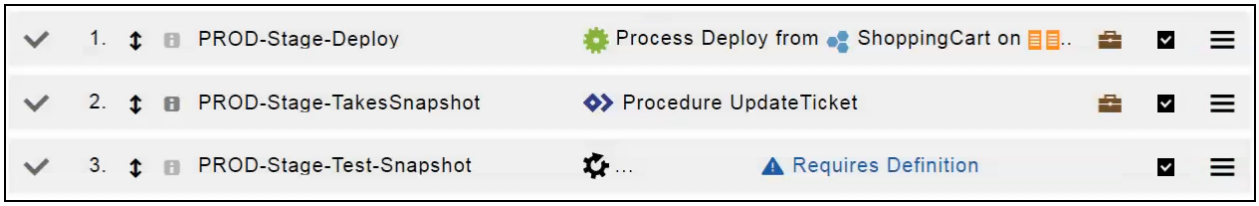


2. In the task list for the PROD stage, add a task named "PROD-Stage-Test-Snapshot" after the "PROD-Stage-Takes-Snapshot" task, and click **Done**:

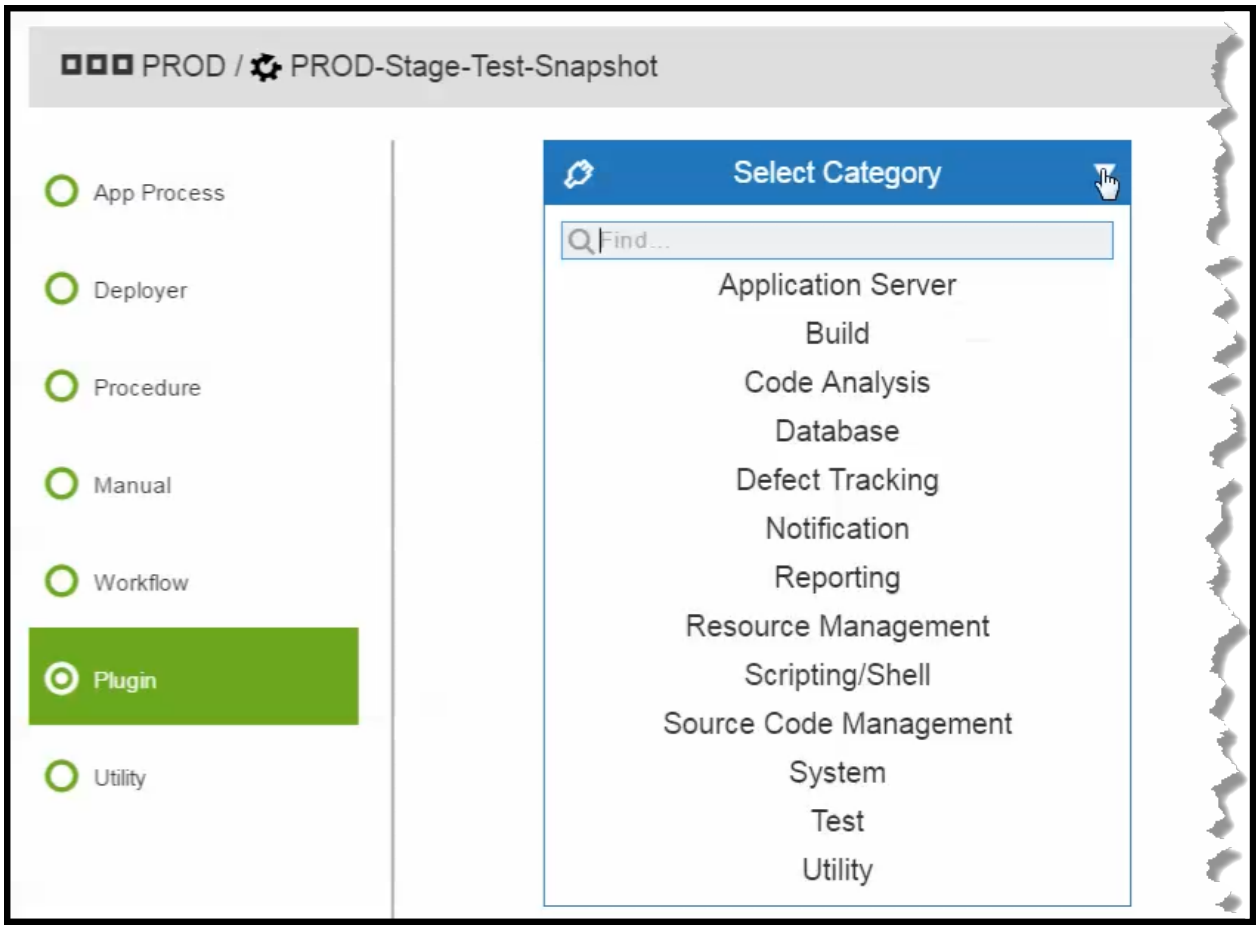


The task list is updated.

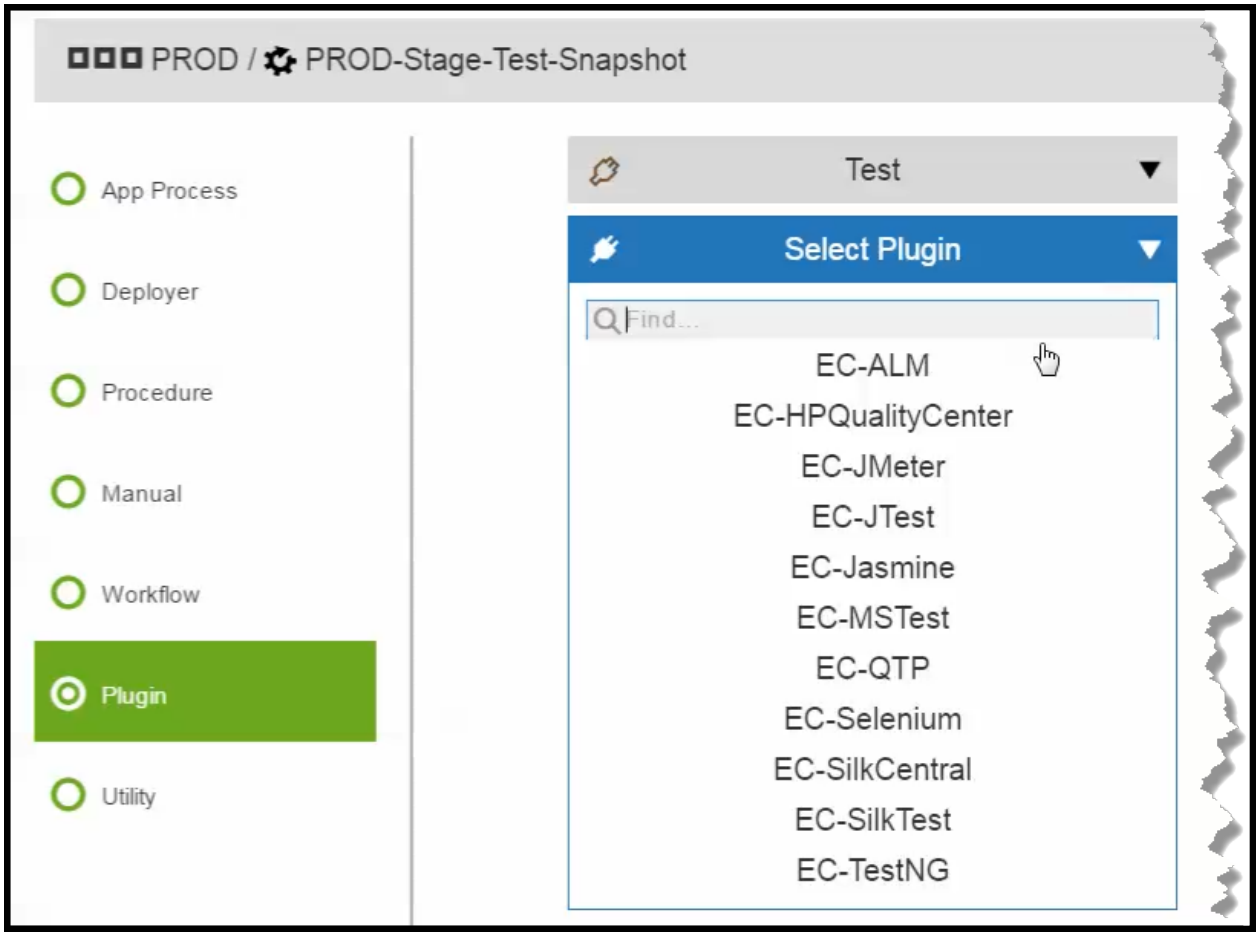
3. Click **Requires Definition** to define the task:



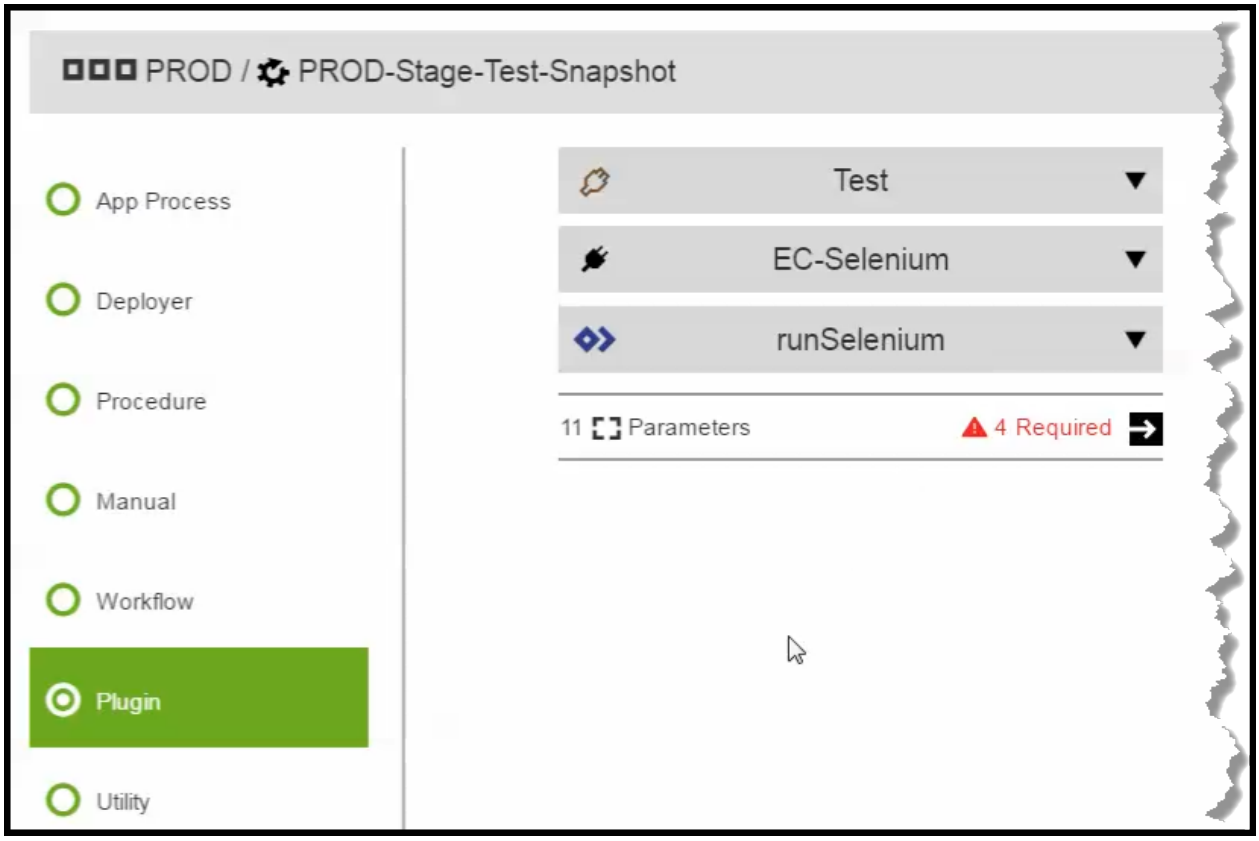
4. In the dialog box to define the task, select **Plugin** as the task type and then click in the **Select Category** field to view the list of available plugin types:



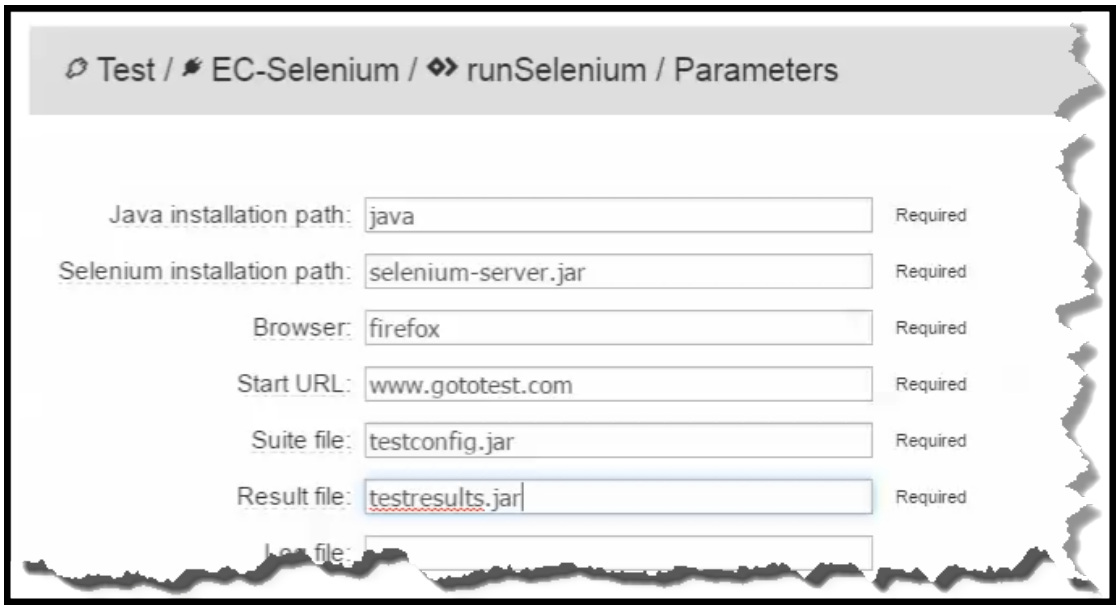
5. Select **Test**, and then click in the **Select Plugin** field to view the list of available test plugins:



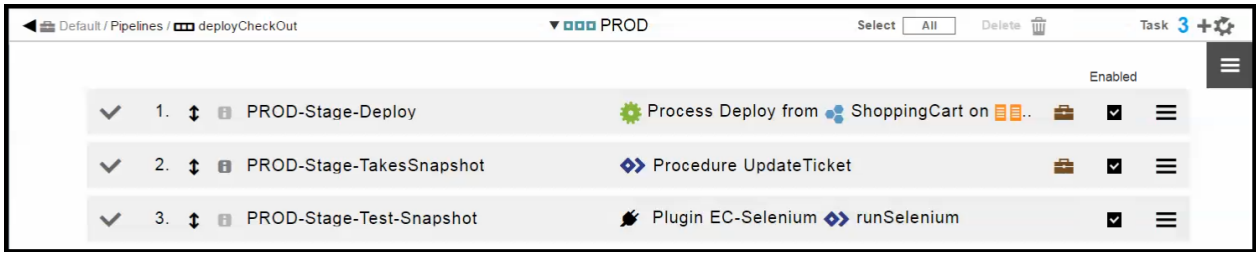
6. Select **EC-Selenium**, and then select **runSelenium** in the Procedure field. There are four required parameters for this procedure:



7. Click  to go to the dialog box where you enter the plugin parameters, enter the parameters, and click **OK**:

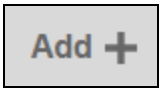


The task list for the PROD stage now shows three tasks, including the plugin task.

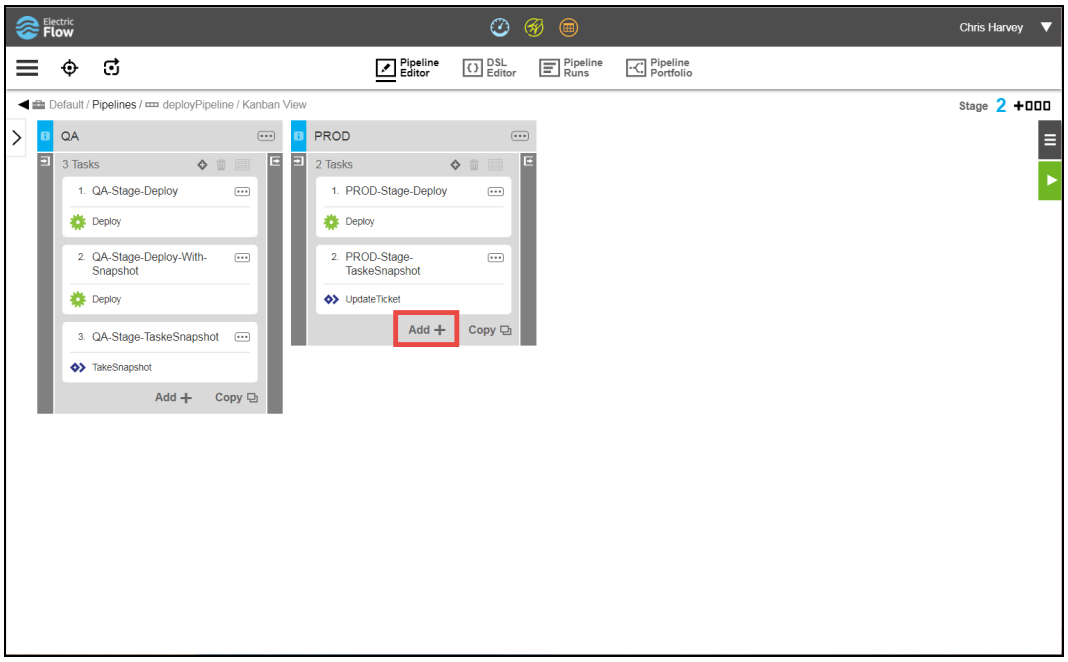


Plugin Tasks in the Release Kanban View

This example shows how to use the Release Kanban View add a task to a pipeline stage and define it using a test plugin.

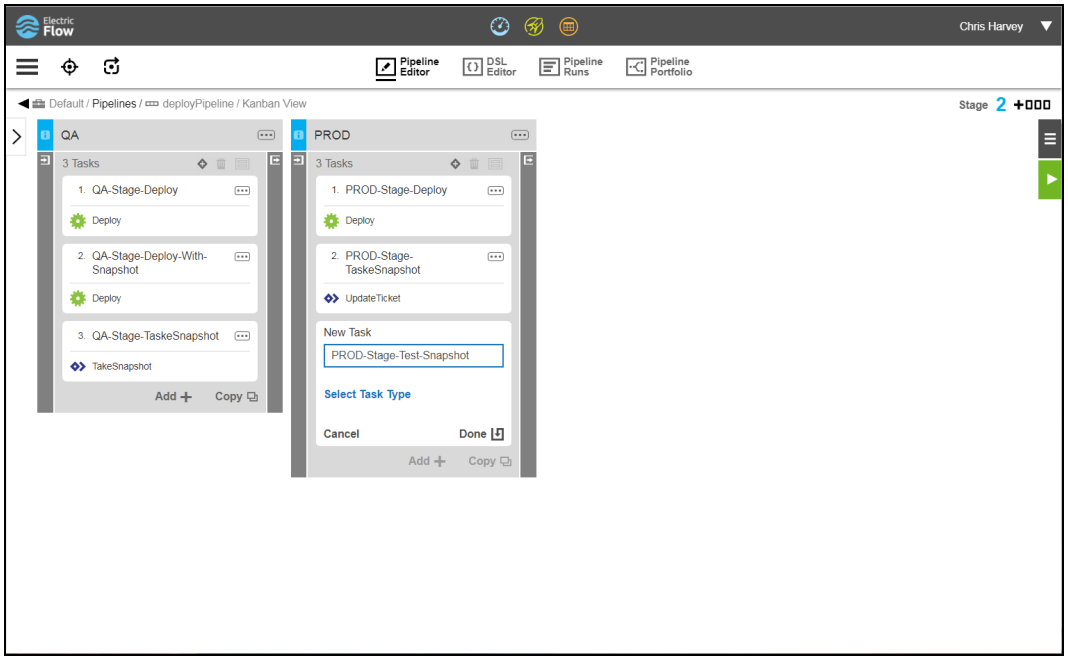


1. In the Pipeline Editor for a new pipeline, click the (New Task) button in the lower right corner of the PROD stage:

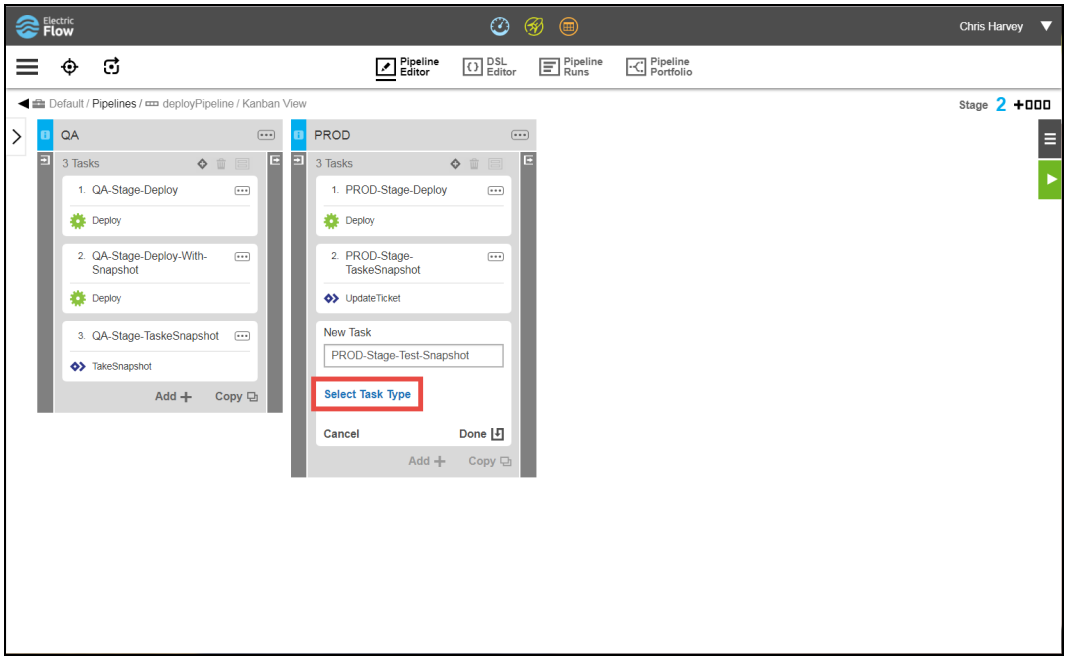


2. Add a task named "PROD-Stage-Test-Snapshot" after the "PROD-Stage-Takes-Snapshot" task:

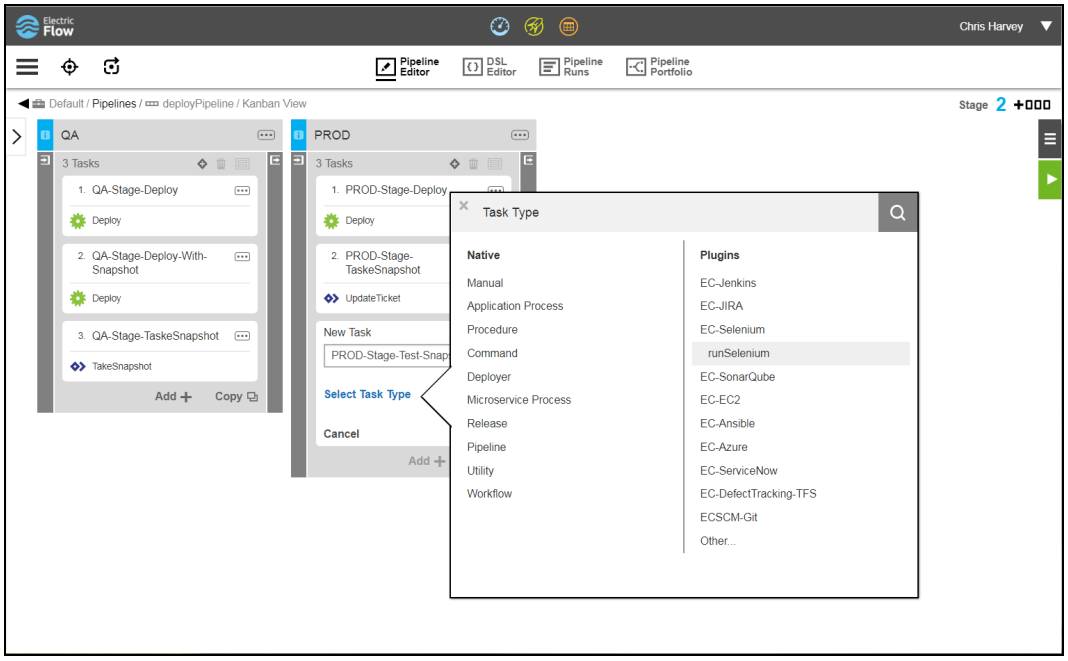
   The task list is updated.



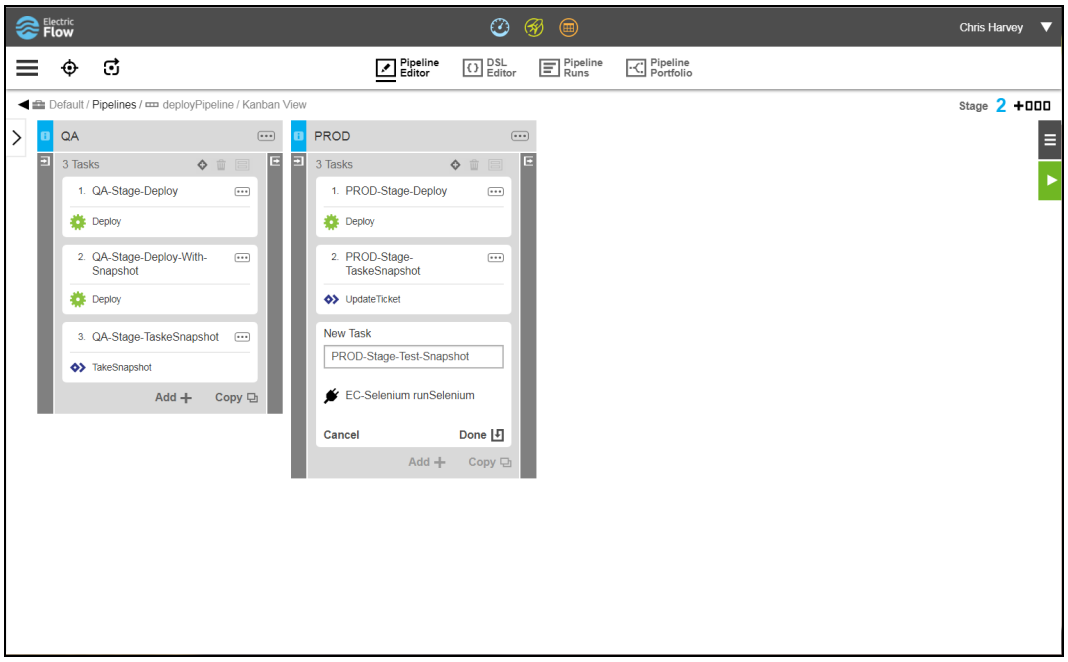
3. Click **Select Task Type** to define the task:



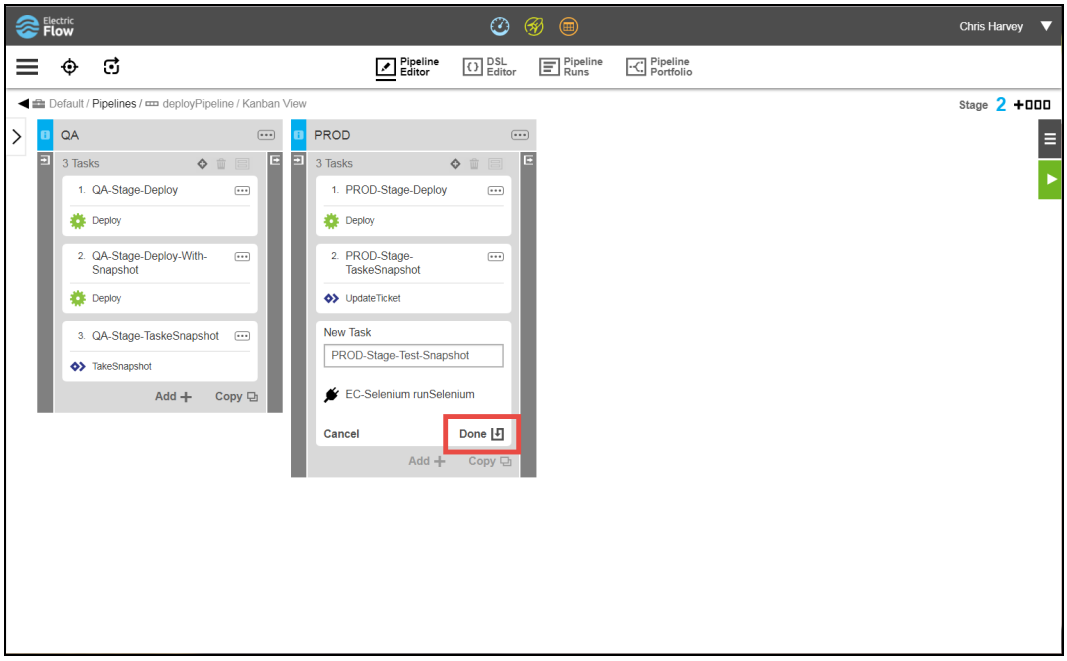
4. In the Task Type popup, click the **EC-Selenium** task type from the list of available plugin types, then click the **runSelenium** procedure:



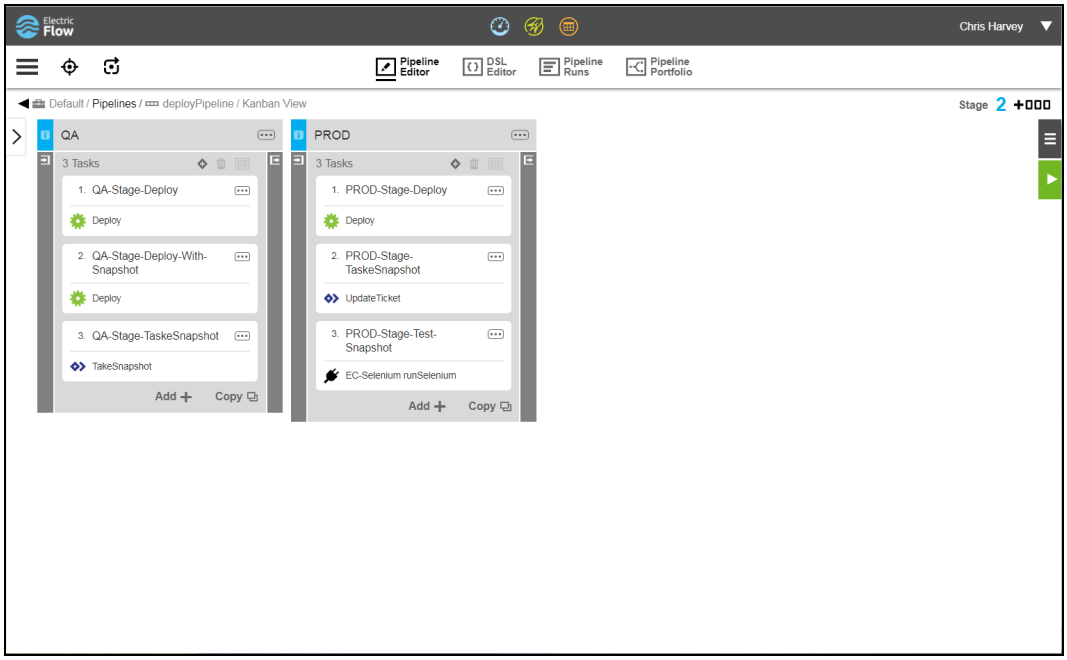
The task type and procedure appear on the task:



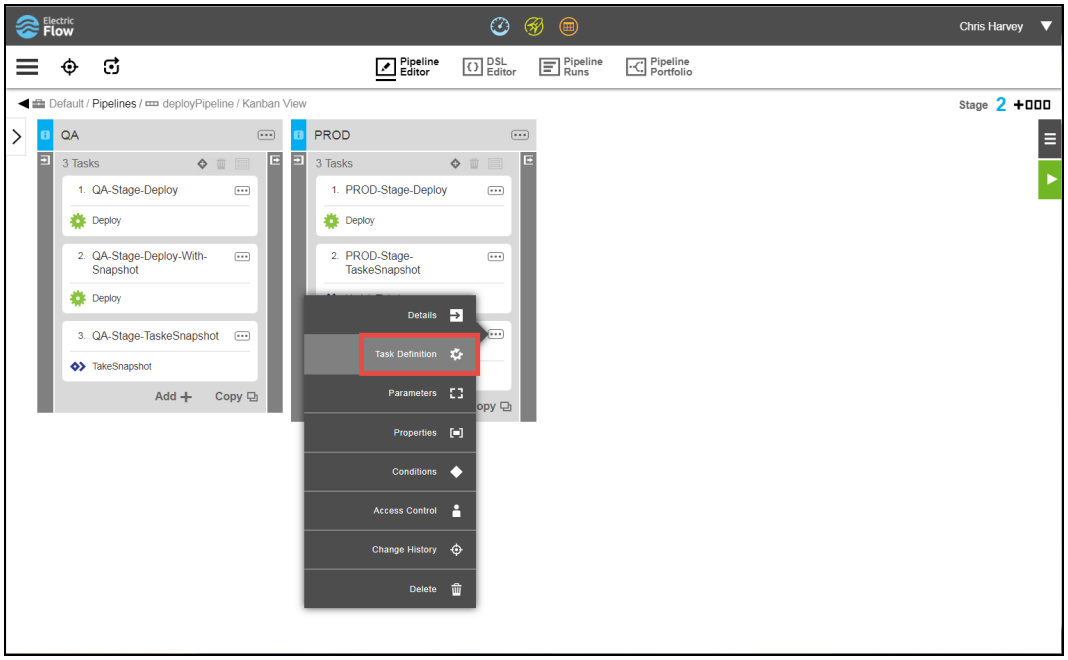
5. Click **Done** to define the task:



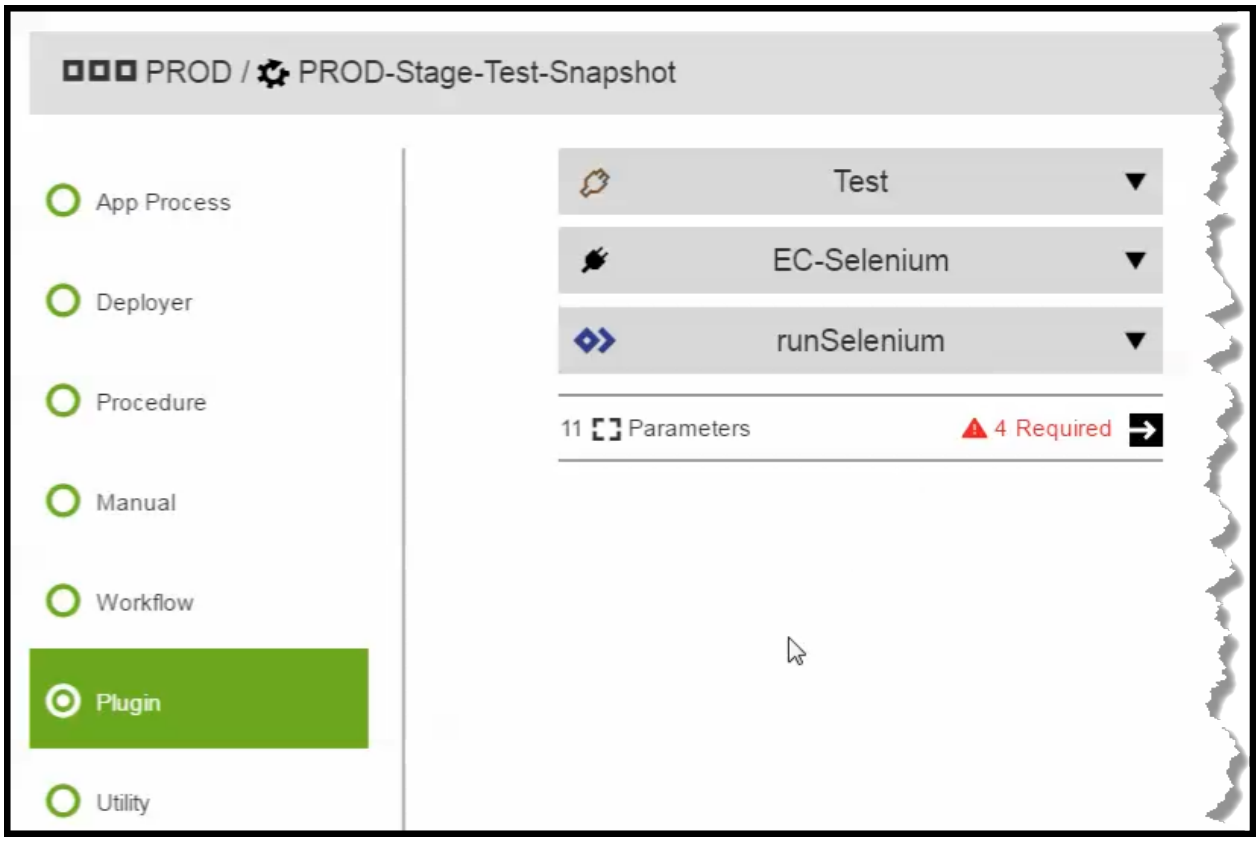
The task appears as follows:



6. Click the (Actions) menu, and then click the Task Definition button:

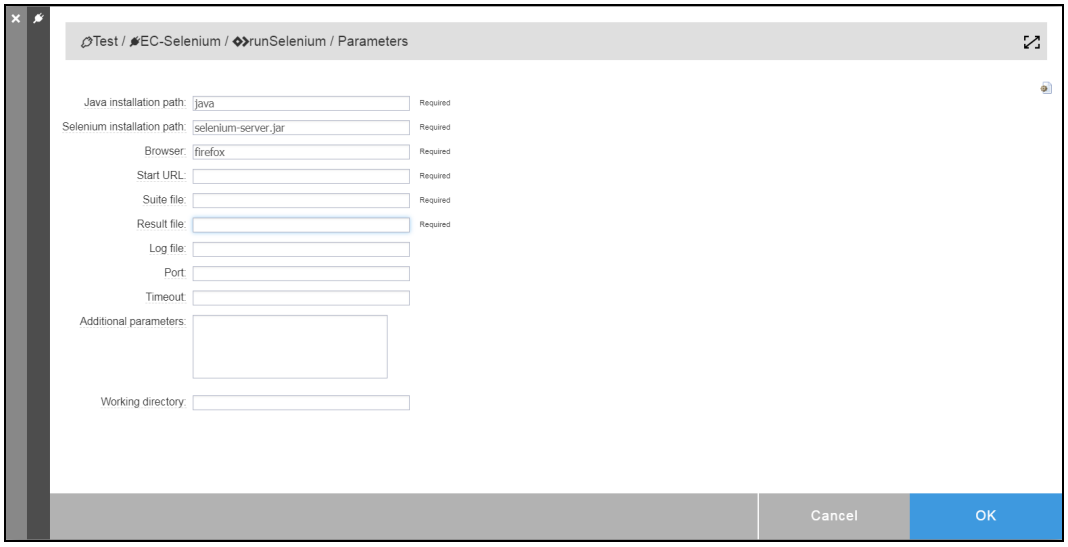


The task definition dialog box appears:


PROD / PROD-Stage-Test-Snapshot
App Process
Deployer
Procedure
Manual
Workflow
Plugin
Utility
Test
EC-Selenium
runSelenium
11 Parameters
4 Required

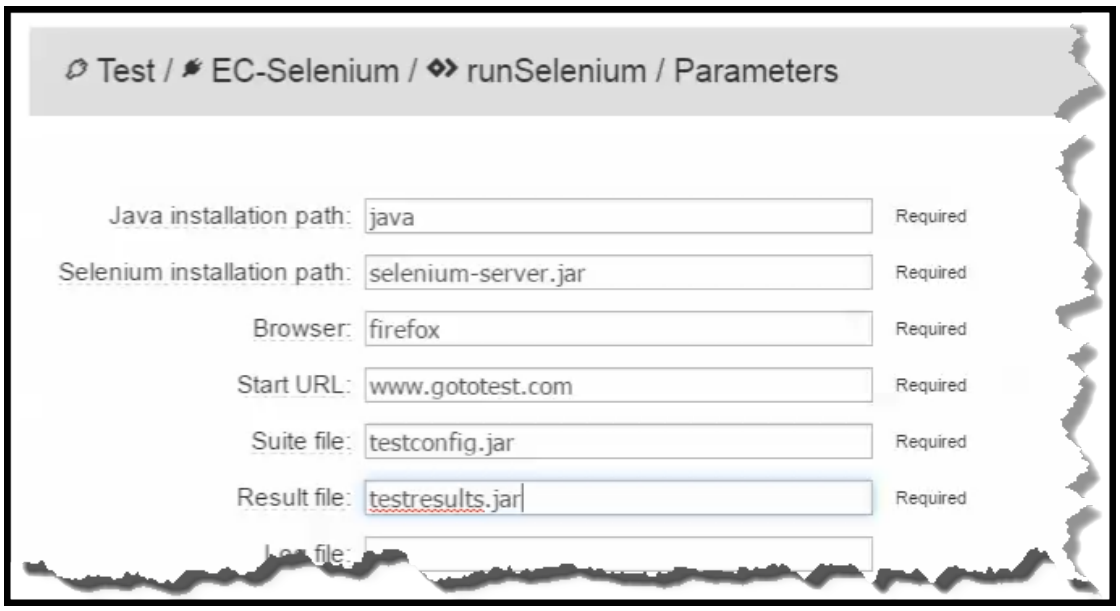
7. Click  to go to the dialog box where you enter the plugin parameters.

   The plugin parameters dialog box appears:

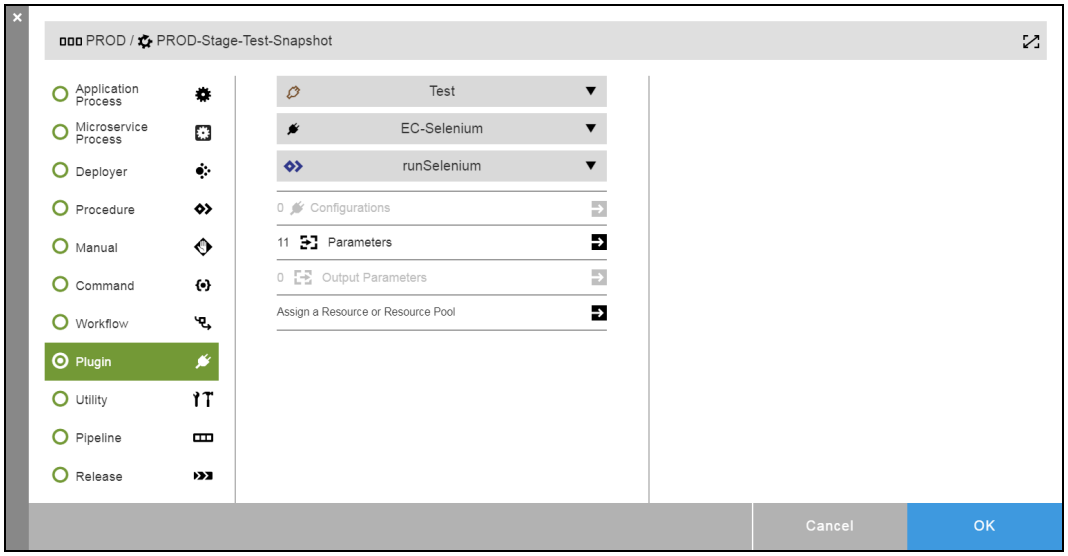


8. Enter the parameters, and click **OK**.

   For example:



   The task definition dialog box is updated to show that parameters are no longer required:



9. Click **OK** to exit the task definition dialog box.

Pipeline Objects and Conditions

You can set **Run if** conditions and **Wait until** preconditions in pipelines through the UI (via either the Pipeline Stage View or the Release Kanban View) or via API commands.

Types of Tasks on page 419

This section covers the following topics:

Pipeline Conditions

A pipeline condition, also known as a **Run if** condition, allows a stage, gate, or task to run only if the condition is satisfied. If not, the stage, gate, or task is skipped. For example:

- A quality gate has these **Run if**conditions:
  - If 90% or more of the test cases pass, the pipeline continues running.
  - If 70% to 90% of the test cases pass, human intervention is required.
  - If less than 70% of the test cases pass, the error handling condition of the gate (**Stop running** (the default) or **Continue running**) determines what happens next.
- The performance testing task can be skipped if it was executed in a specific environment.

> **Tip:** For examples of Javascript expressions for **Run if** and **Wait until** pipeline conditions, see the *KBEC-00360 - Using Context-Relative Shortcuts to Properties on Pipelines and related objects* KB article.

Pipeline Preconditions

A pipeline preconditions, also know as a **Wait until** precondition, restricts a stage, gate, or task from running until the condition is satisfied. For example, a stage, gate, or task cannot be run until approval from an external ticketing system or change management system is received.

You cannot use timestamps in preconditions on any object that supports preconditions. This includes stages, gates, and tasks as well as procedures and process steps.

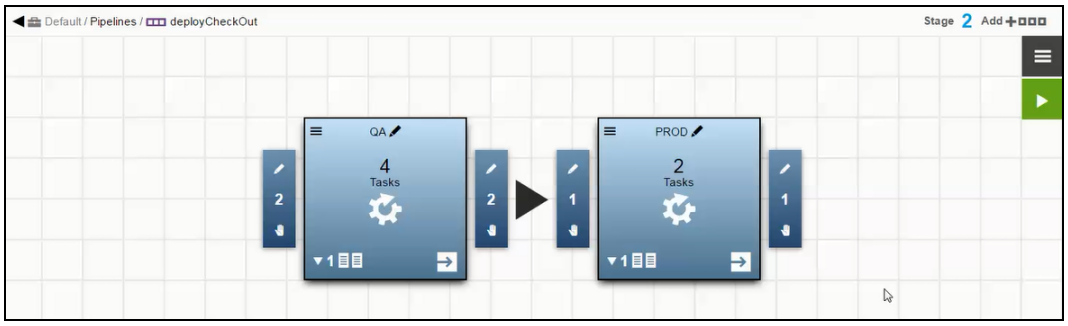
Using Pipeline Objects and Conditions in the Pipeline Stage View

You can set **Run if** and **Wait until** conditions for pipeline stages, gates, and tasks. Following are the overall steps for using pipeline objects and conditions in the Pipeline Stage View:

1. Modeling the Pipeline on page 578
2. Setting Conditions for the Stage on page 578
3. Setting the "Run If" and "Wait until" Conditions for a Task on page 581
4. Setting the "Run If" and "Wait until" Conditions for a Gate on page 582
5. Running the Pipeline on page 588

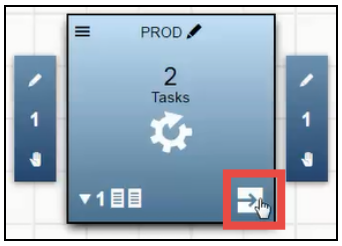
Modeling the Pipeline

1. Create a pipeline and define stages named QA and PROD.
2. Create tasks for each stage:

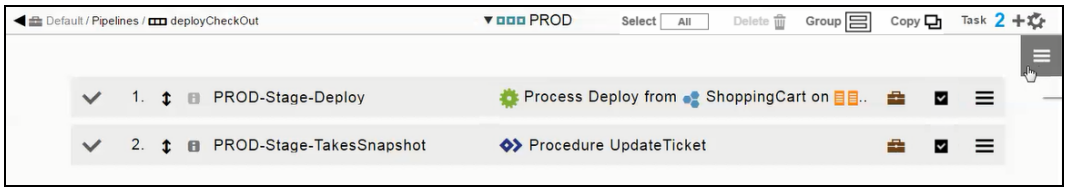


Setting Conditions for the Stage

1. In the PROD stage, click the **Define stage tasks** button in the lower right corner of the stage:

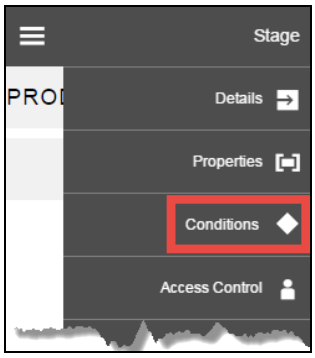


The task list opens:

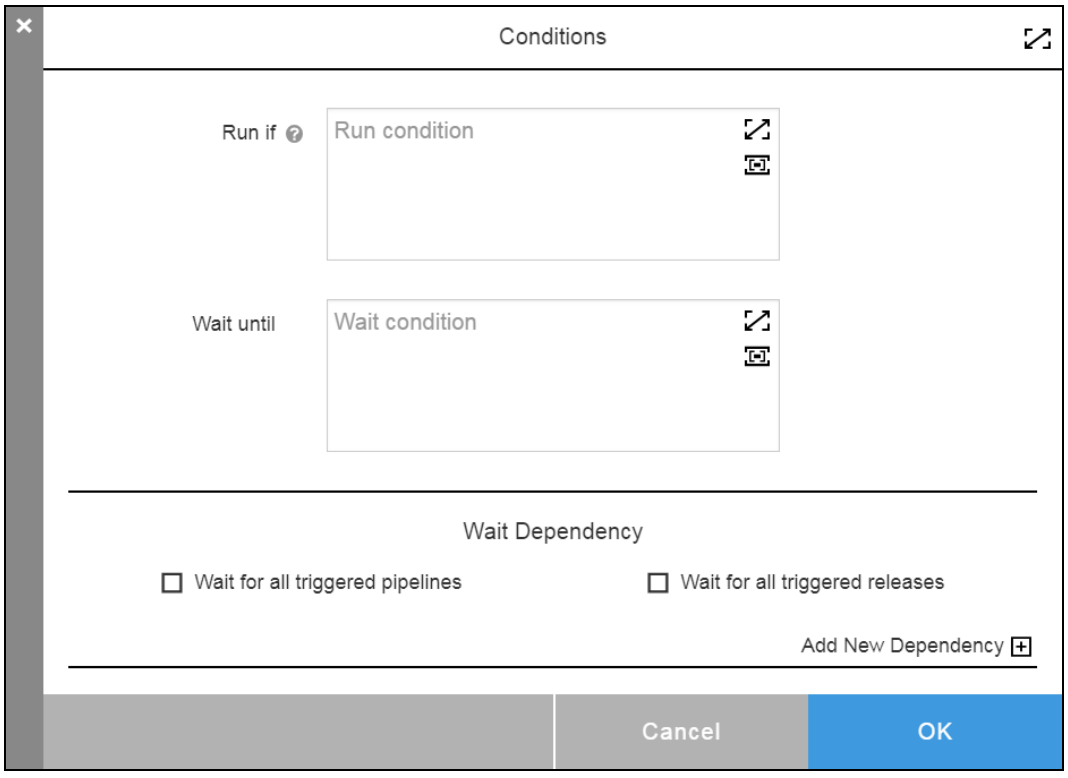


2. Set the **Run if** and **Wait until** conditions for the PROD stage.

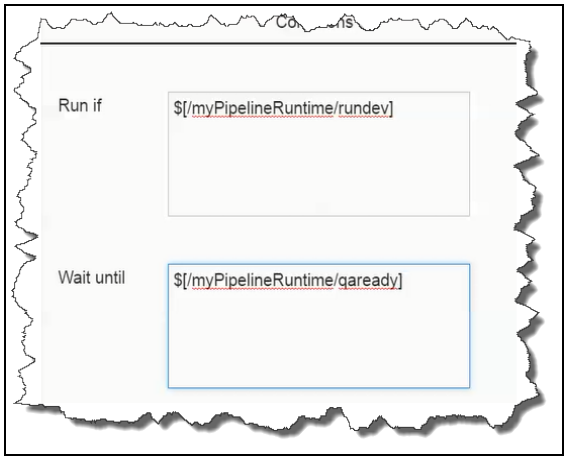
To do so, click  the upper right corner of the task list and select **Conditions**:



The **Conditions** dialog box appears:



3. Enter the Run condition in the **Run if** field and the Wait condition in the **Wait until** field:

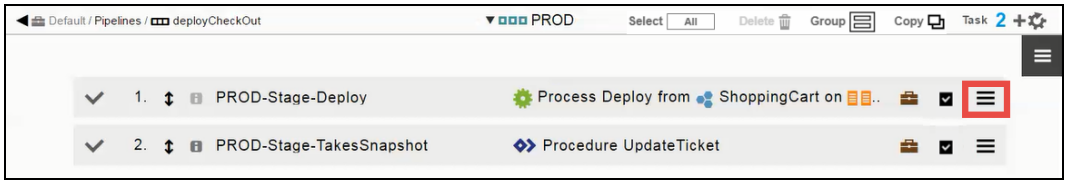


4. Click **OK** to save the conditions.

Setting the "Run If" and "Wait until" Conditions for a Task

1. Return to the task list for the PROD stage.

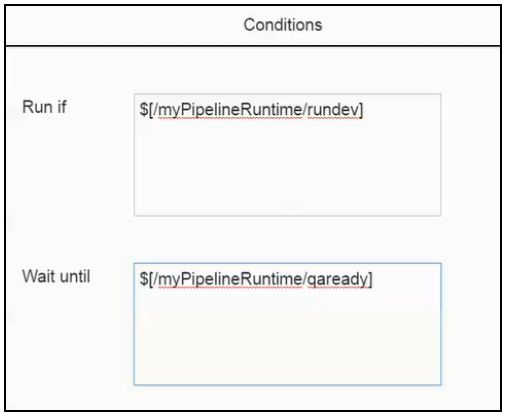
2. Click the menu button in the row for the first task (PROD-Stage-Deploy), and select **Conditions**:



The **Conditions** dialog box appears:

3. In the **Conditions** dialog box, enter the Run condition in the **Run if** field and the Wait condition in the **Wait until** field.

   For example:

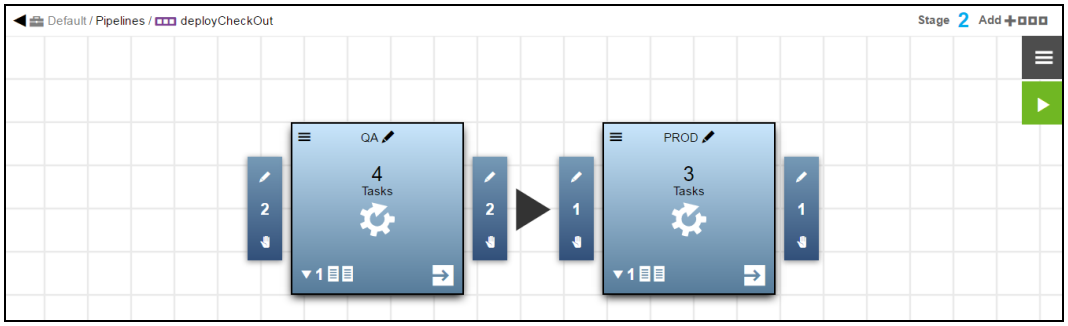


4. Click **OK** to save your entries.

Setting the "Run If" and "Wait until" Conditions for a Gate

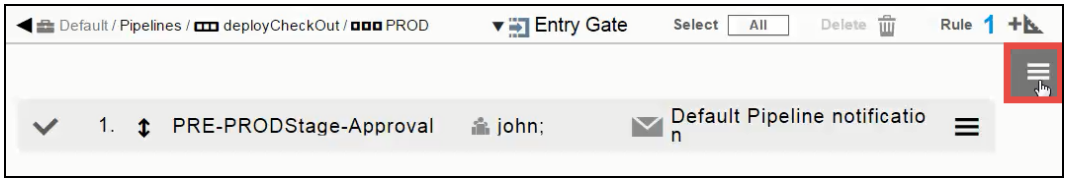
Setting Gate Conditions

1. Return to the Pipeline Editor.



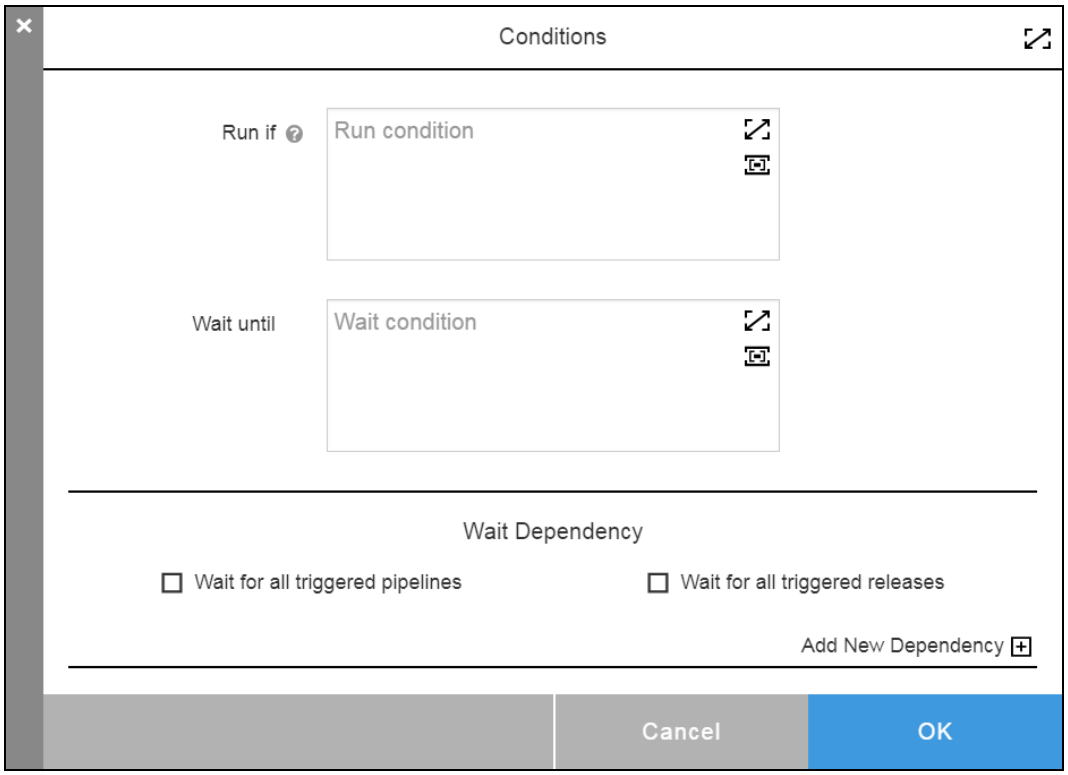
2. Click the **Edit** button ( ) in the entry gate to the PROD stage:

The list of rules for the entry gate appears:

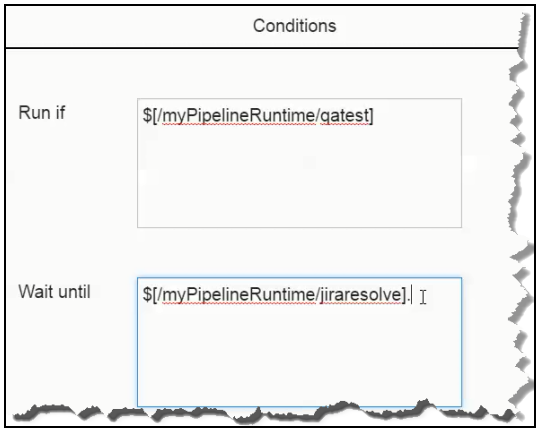


3. Click the menu button the upper right corner of the rule list, and select **Conditions**:

The **Conditions** dialog box appears:

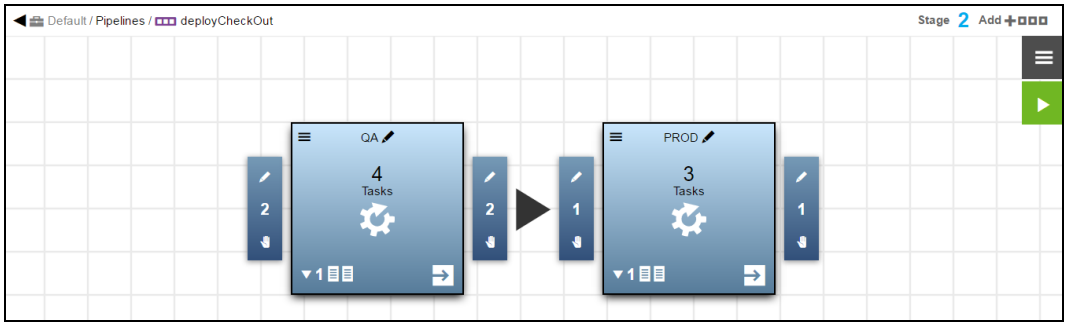


4. In the **Conditions** dialog box, enter the Run condition in the **Run if** field and the Wait condition in the **Wait until** field, and then click **OK**:



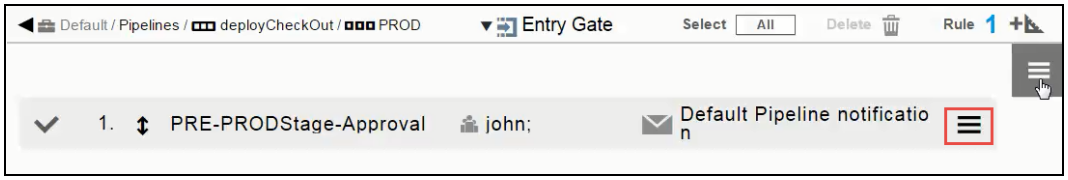
Setting the "Run If" and "Wait until" Conditions for a Rule in a Gate

1. Return to the Pipeline Editor.



2. Click the **Edit** button ( ) in the entry gate to the PROD stage:

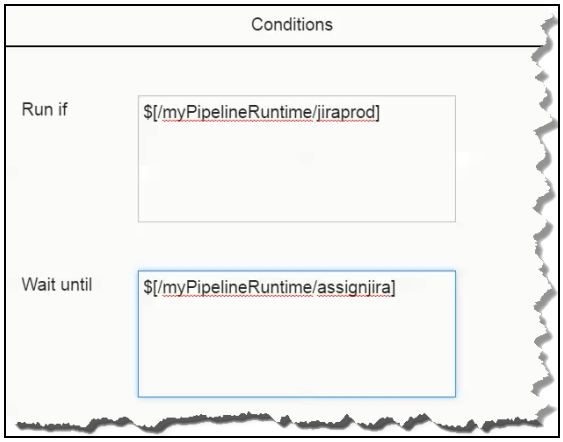
The list of rules for the entry gate opens:



3. Click the row for the PRE-PRODStage-Approval rule, and select **Conditions**:

The **Conditions** dialog box appears:

4. In the **Conditions** dialog box, enter the **Run if** condition in the **Run if** field and the **Wait until** condition in the **Wait until** field, and then click **OK**:



Running the Pipeline

When the pipeline runs, it waits until the **Run if** and **Wait until** conditions are met before the pipeline can progress to the PROD stage. Before the first task in the PROD stage starts, the pipeline again waits until the **Run if** and **Wait until** conditions are met before starting the task.

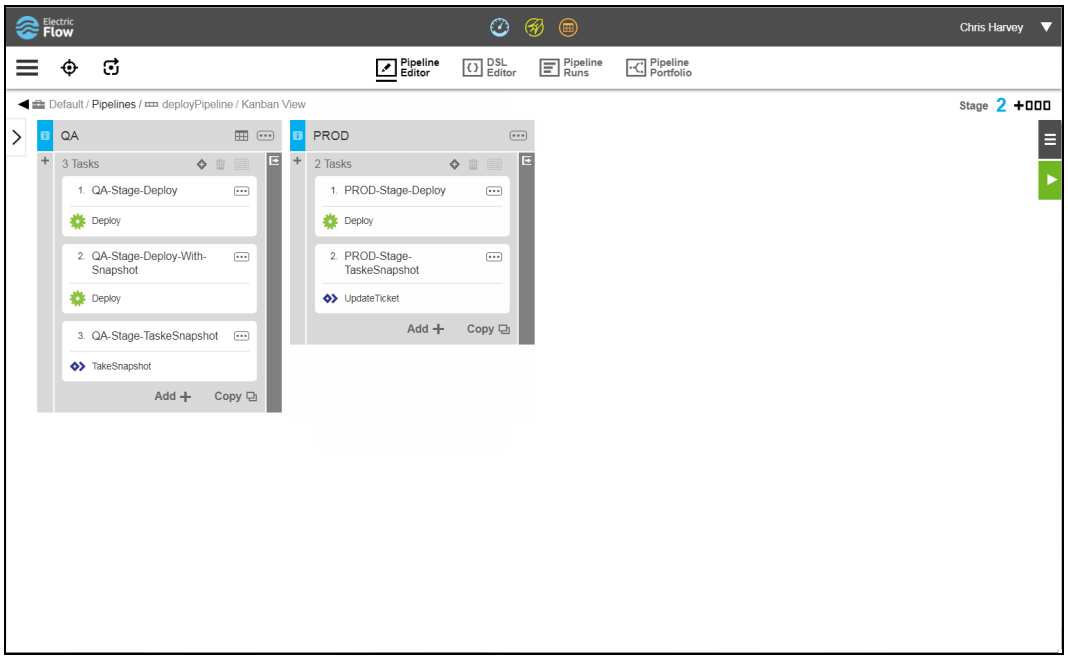
Using Pipeline Objects and Conditions in the Release Kanban View

You can set **Run if** and **Wait until** conditions for pipeline stages, gates, and tasks. Following are the overall steps for using pipeline objects and conditions in the Release Kanban View:

1. Modeling the Pipeline on page 589
2. Setting the "Run If" and "Wait until" Conditions for the Stage on page 590
3. Setting the "Run If" and "Wait until" Conditions for a Task on page 592
4. Setting the "Run If" and "Wait until" Conditions for a Gate on page 594
5. Running the Pipeline on page 599

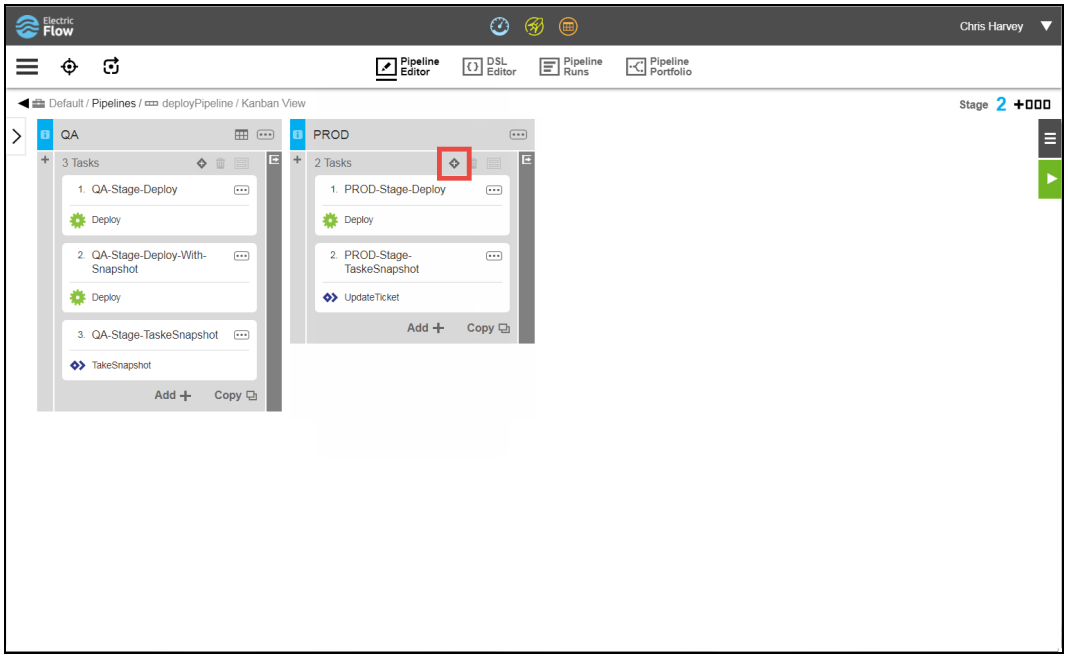
Modeling the Pipeline

1. Create a pipeline and define stages named QA and PROD.
2. Create tasks for each stage:



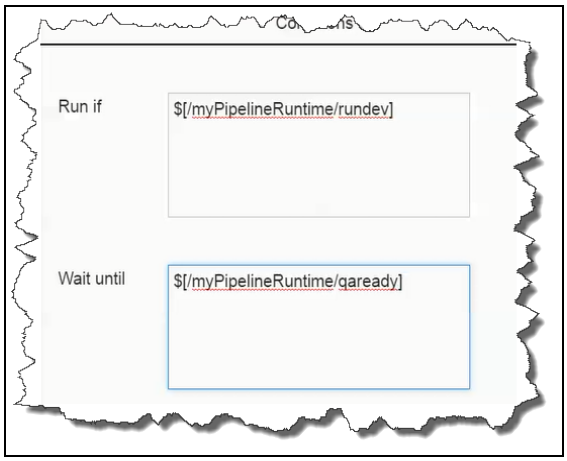
Setting the "Run If" and "Wait until" Conditions for the Stage

1. In the PROD stage, click the  (Stage Conditions) button in the top right corner of the stage:



The **Conditions** dialog box opens:

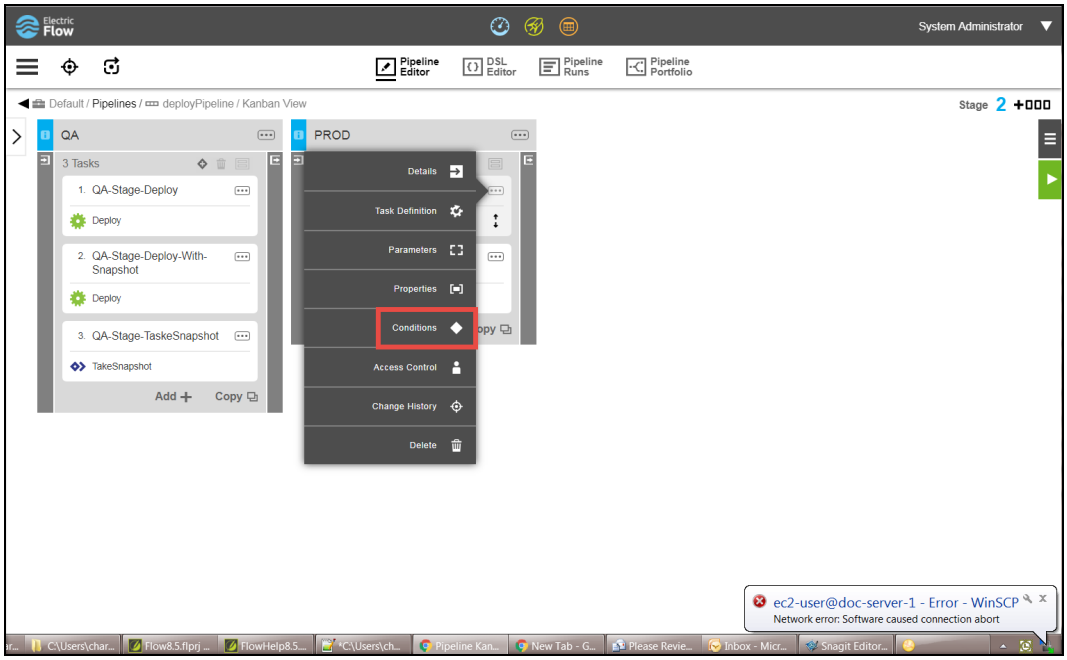
2. Enter the Run condition in the **Run if** field and the Wait condition in the **Wait until** field:



3. Click **OK** to save the conditions.

Setting the "Run If" and "Wait until" Conditions for a Task

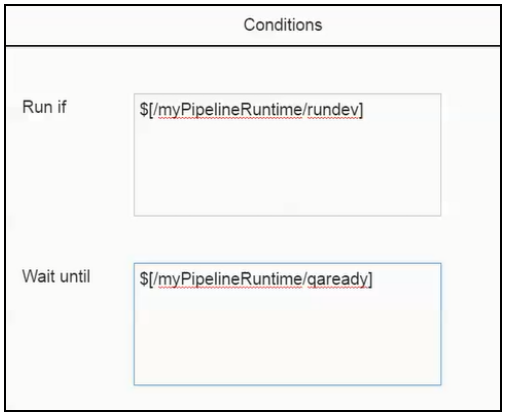
1. Click the (Actions) button for the first task (PROD-Stage-Deploy), and choose **Conditions**:



The **Conditions** dialog box appears:

2. In the **Conditions** dialog box, enter the Run condition in the **Run if** field and the Wait condition in the **Wait until** field.

   For example:

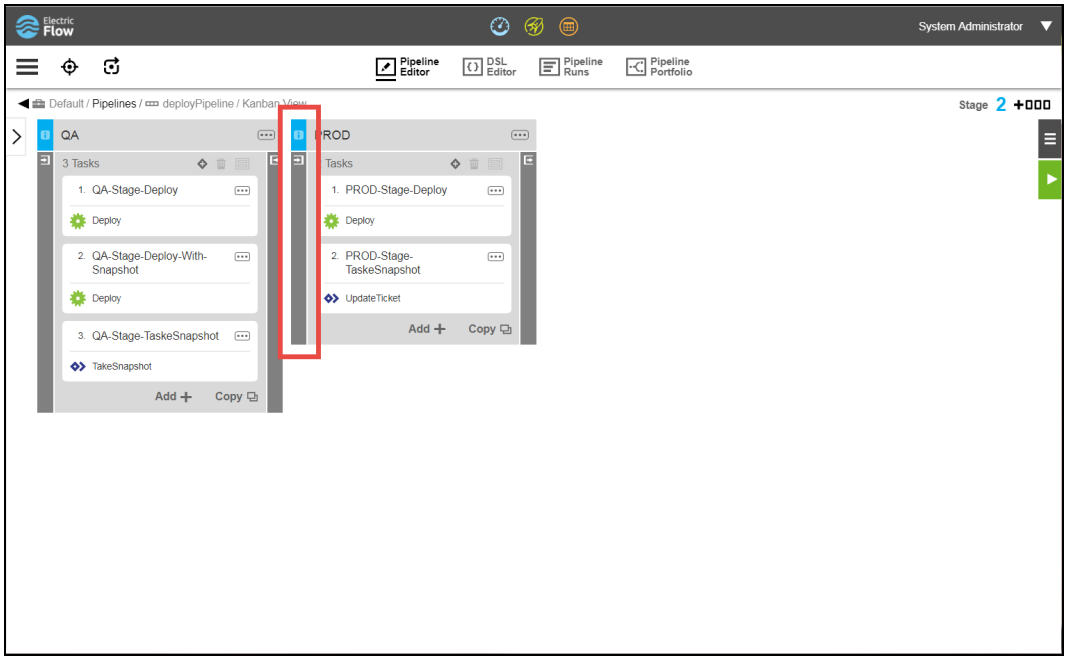


3. Click **OK** to save your entries.

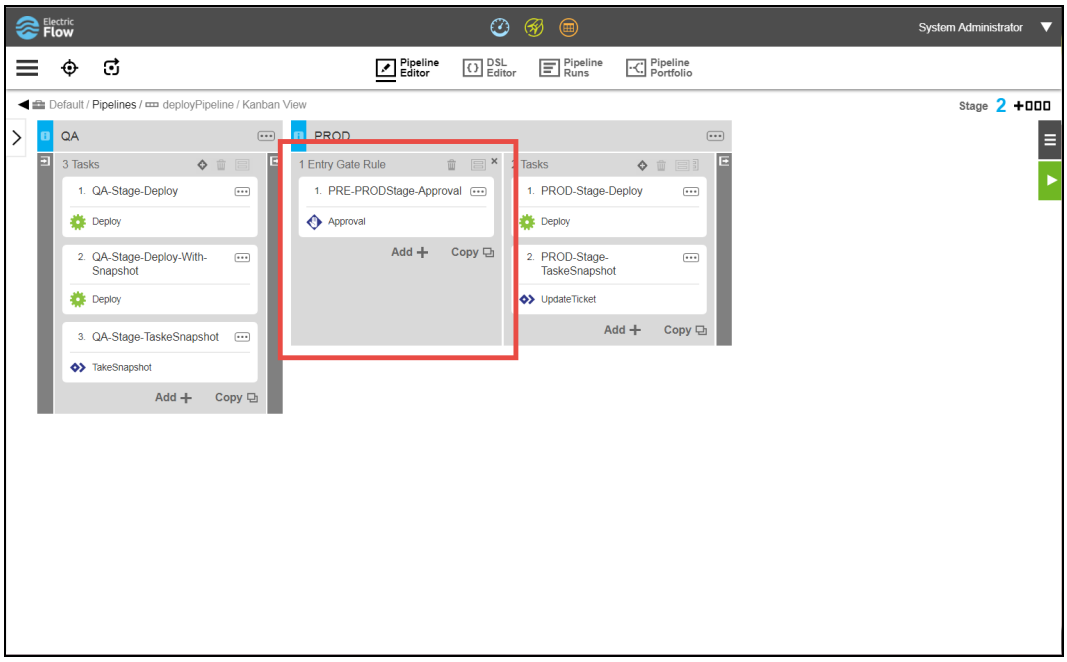
Setting the "Run If" and "Wait until" Conditions for a Gate

Setting Gate Conditions

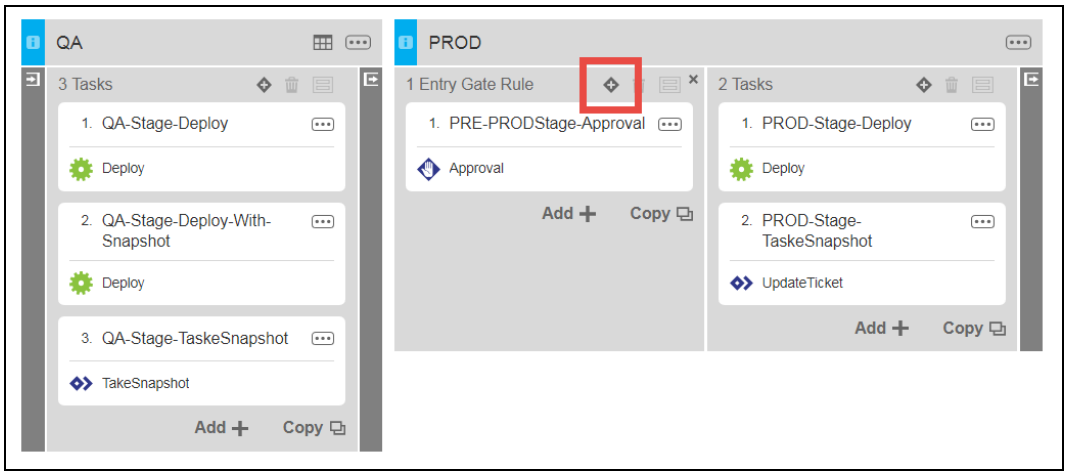
1. Click to expand the entry gate to the PROD stage:



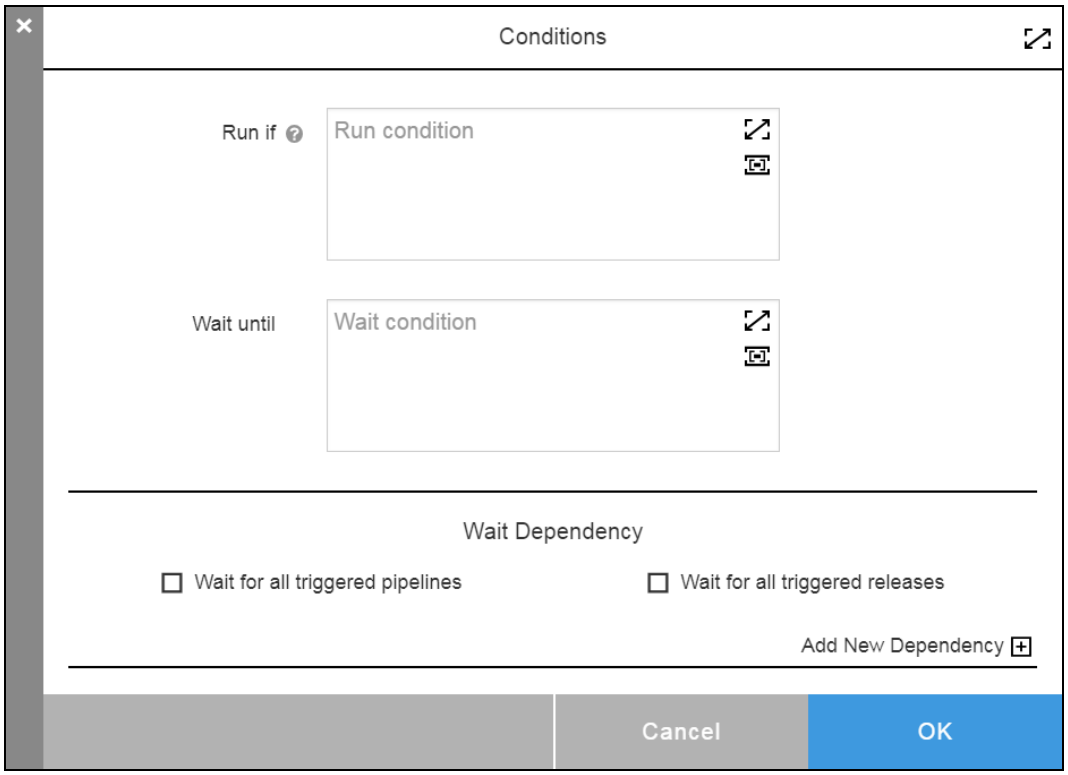
The list of rules for the entry gate opens:



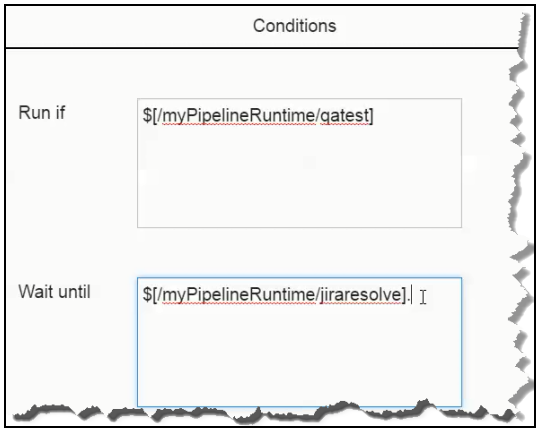
2. Click the  (Gate Conditions) button for the gate:



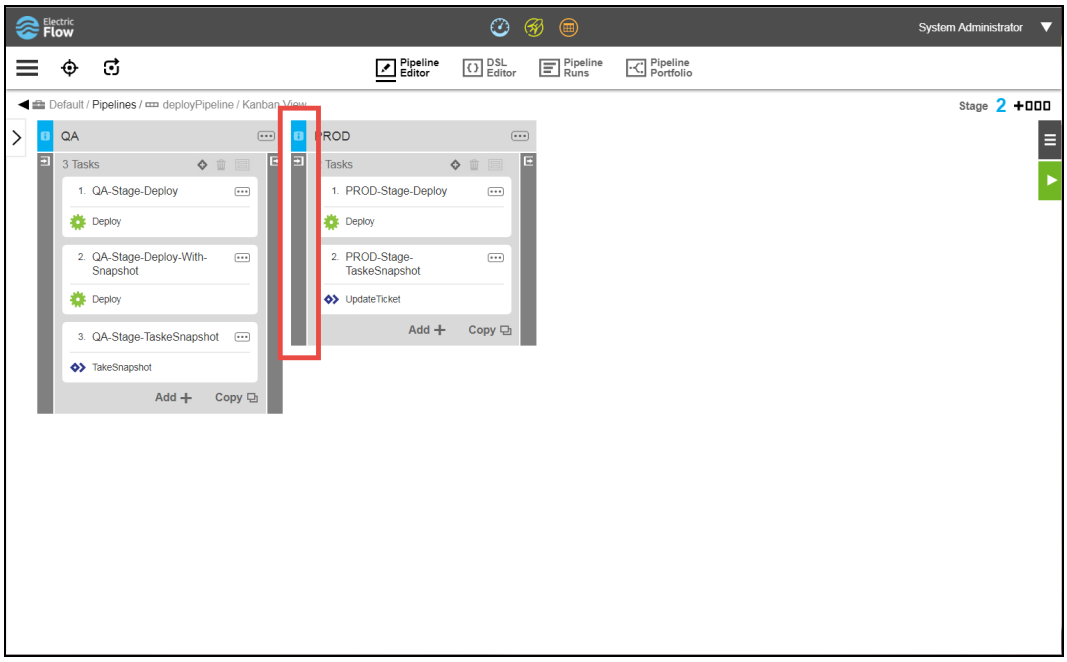
The **Conditions** dialog box appears:



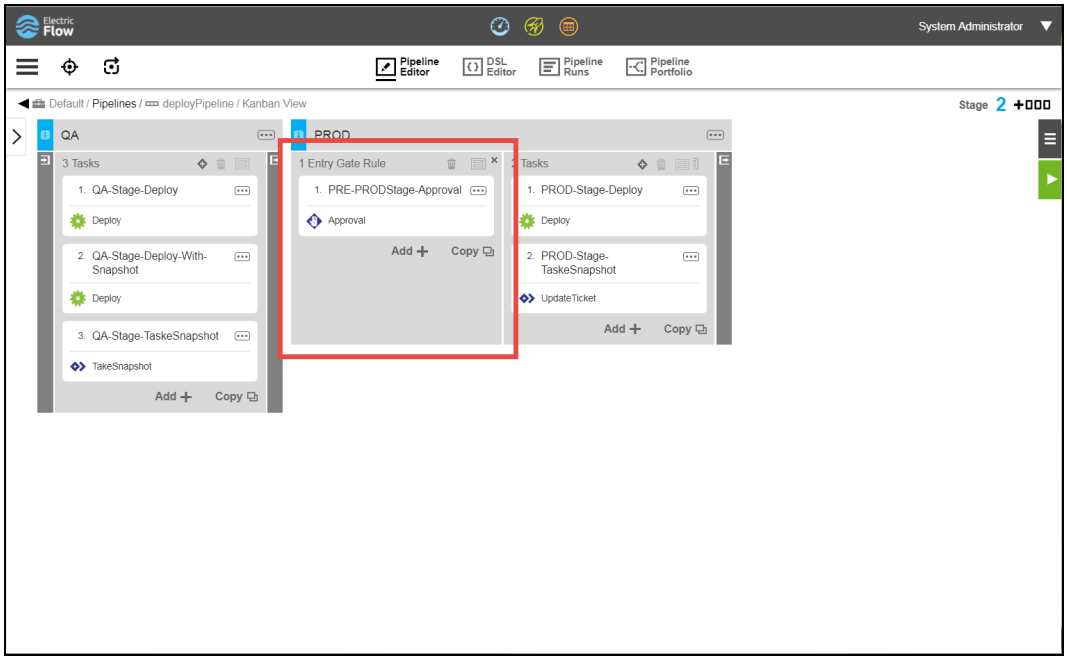
3. In the **Conditions** dialog box, enter the Run condition in the **Run if** field and the Wait condition in the **Wait until** field, and then click **OK**:



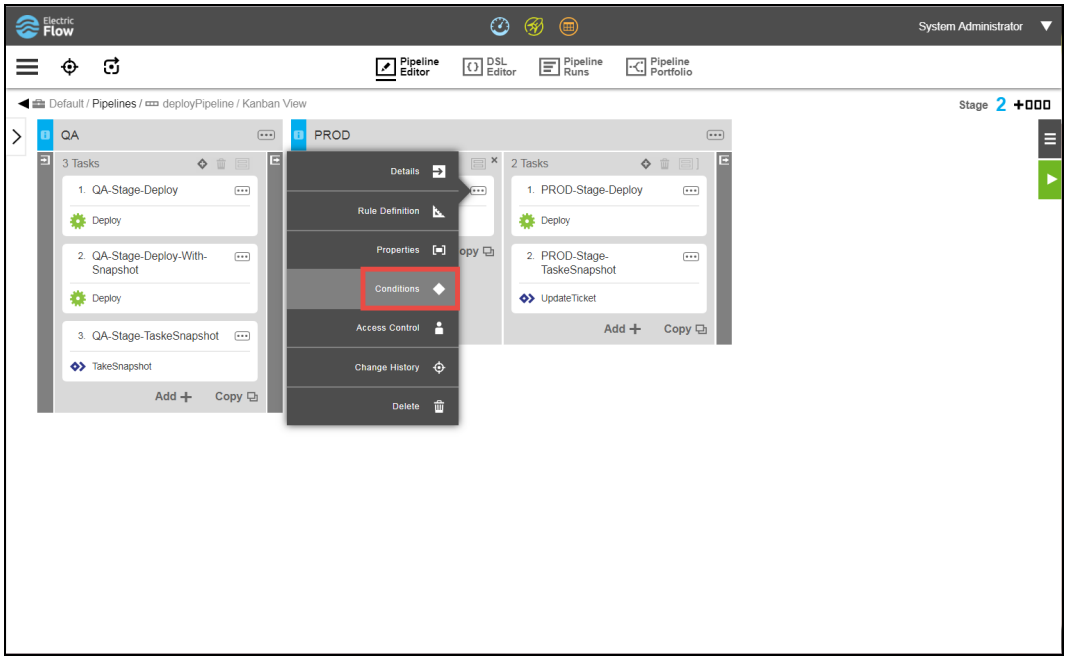
1. Click to expand the entry gate to the PROD stage:



The list of rules for the entry gate opens:

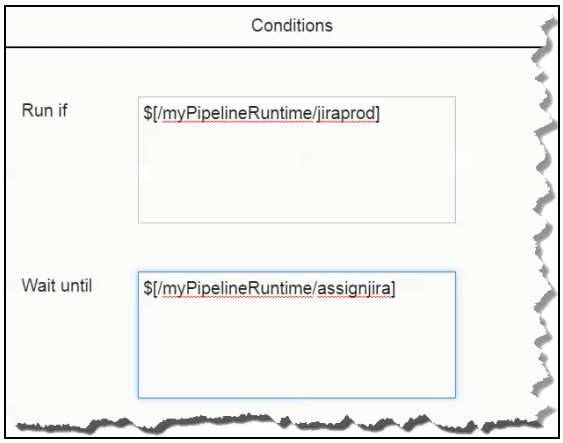


2. Click the  (Actions) button for the PRE-PRODStage-Approval rule, and select **Conditions**:



The **Conditions** dialog box appears:

3. In the **Conditions** dialog box, enter the **Run if** condition in the **Run if** field and the **Wait until** condition in the **Wait until** field, and then click **OK**:



Running the Pipeline

When the pipeline runs, it waits until the **Run if** and **Wait until** conditions are met before the pipeline can progress to the PROD stage. Before the first task in the PROD stage starts, the pipeline again waits until the **Run if** and **Wait until** conditions are met before starting the task.

Using Pipeline Objects and Conditions with API Commands

You can set the **Run if** and **Wait until** conditions using the API commands through the ectool command-line interface or through a DSL script on pipeline stages, tasks, and gates.

Stages

Create

- **Run if:** `ectool createStage <projectName> <stageName> [<condition>]`
- **Wait until:** `ectool createStage <projectName> <stageName> [<precondition>]`

Examples:

- Create a stage `QA` with a **Run if** condition to run only if `DEV.task1` completes successfully:

```
ectool createStage "Default" "QA" --condition "$[/javascript
myPipelineRuntime.stages["DEV"].tasks["task1"].outcome = "success"]"
```

- Create stage `QA` with a **Wait until** condition to wait until `DEV.task2` completes successfully:

```
ectool createStage "Default" "QA" --precondition "$[/javascript
myPipelineRuntime.stages["DEV"].tasks["task2"].outcome == "success"]"
```

Modify

- **Run if:** `ectool modifyStage <projectName> <stageName> [<condition>]`
- **Wait until:** `ectool modifyStage <projectName> <stageName> [<precondition>]`

Examples:

- Modify stage `QA` **Run if** condition to run only if `DEV.task1` fails:

```
ectool createStage "Default" "QA" --condition "$[/javascript
myPipelineRuntime.stages["DEV"].tasks["task1"].outcome = "error"]"
```

- Modify stage `QA` **Wait until** condition to wait until `DEV.task2` fails:

```
ectool createStage "Default" "QA" --precondition "$[/javascript
myPipelineRuntime.stages["DEV"].tasks["task2"].outcome == "error"]"
```

DSL script

When you use DSL to model the pipeline, the DSL script includes this content. The **Run if** and **Wait until** conditions are defined in the stage object. This example shows that the dev stage has a **Run if** conditon and a **Wait until** condition. The **Run if** condition is that `$[/myPipelineRuntime/rundev]` is `TRUE` for the dev stage to start. The **Wait until** condition is that `$[/myPipelineRuntime/qaready]` is `TRUE` for the pipeline to progress to the next stage.

```
pipeline ('heatclinic_pipeline'){
   ...
   stage ('dev'){
      ...
      condition = '$[/myPipelineRuntime/rundev]'
      precondition = '$[/myPipelineRuntime/qaready]'
      ...
      task ('uninstall'){
        ...
      }
      ...
   }
}
```

Tasks

Create

- **Run if:** `ectool createTask <projectName> <taskName> [<condition>]`
- **Wait until:** `ectool createTask <projectName> <taskName> [<precondition>]`

Examples:

- Create task `Deploy WAR file` with a **Run if** condition to run if `dev.task1` was skipped:

  ```
  ectool createTask "Default" "Deploy WAR file" --condition "$[/javascript
  myStageRuntime.stages["dev"]tasks["task1"].outcome =="skipped"]"
  ```

- Create task `Deploy WAR file` with a **Wait until** condition to wait until task `dev.task2` was skipped:

  ```
  ectool createTask "Default" "Deploy WAR file" --precondition "$[/javascript
  myStageRuntime.stages["dev"]tasks["task2"].outcome =="skipped"]"
  ```

Modify

- **Run if:** `ectool modifyTask <projectName> <taskName> [<condition>]`
- **Wait until:** `ectool modifyTask <projectName> <taskName> [<precondition>]`

Examples:

- Modify task `Deploy WAR file` **Run if** condition:

  ```
  ectool createTask "Default" "Deploy WAR file" --condition "$[/javascript
  myStageRuntime.stages["dev"]tasks["task1"].outcome =="skipped"]"
  ```

- Modify task `Deploy WAR file` **Wait until** condition for task `Deploy WAR file`:

  ```
  ectool createTask "Default" "Deploy WAR file" --precondition "$[/javascript
  myStageRuntime.stages["dev"]tasks["task2"].outcome =="skipped"]"
  ```

DSL script

When you use DSL to author the pipeline, the DSL script includes this content. The **Run if** and **Wait until** conditions are defined in the task object.

This example shows that the task `uninstall` has a **Run if** condition and a **Wait until** condition defined within the `dev` stage task:

```
pipeline ('heatclinic_pipeline'){
   ...
   stage ('dev'){
      ...
      task ('uninstall'){
         condition = '$[/javascript myFlowRuntime.flowRuntimeStates[taskName=t1].param == 1]'
         precondition = '$[/javascript new Date().getHours() > 15]'
         subprocedure = 'build'
         subproject = 'default'
      }
      ...
   }

}
```

Gates

Create

- **Run if:** `ectool createGate <projectName> <stageName> <gateType> [<condition>]`
- **Wait until:** `ectool createGate <projectName> <stageName> <gateType> [<precondition>]`

Example:

To create an exit gate including both **Run if** and **Wait until** conditions. The example below sets both the `--condition` and `--precondition` in the same `ectool` command.

```
ectool createGate "Default" "dev" "POST" --condition "$[/javascript
myGateRuntime.tasks["auto task"].outcome =="skipped"]" --precondition
"$[/myGate/autoTestCompleted]"
```

Modify

- **Run if:** `ectool createGate <projectName> <stageName> <gateType> [<condition>]`
- **Wait until:** `ectool createGate <projectName> <stageName <gateType> [<precondition>]`

Example:

Modify both **Run if** and **Wait until** conditions on the entry gate of stage `dev`. The example below sets both the `--condition` and `--precondition` in the same `ectool` command.

```
ectool modifyGate "Default" "dev" "PRE" --condition "$[/javascript
myGateRuntime.tasks["auto task"].outcome =="success"] --precondition
"$[/myGate/autoTestCompleted]"
```

DSL script

When you use DSL to author the pipeline, the DSL script includes this content. The **Run if** and **Wait until** conditions are defined in the gate object. This example shows that the build stage has two gates, an entry gate (PRE) and an exit gate (POST). The tasks for each gate are embedded within each one.

```
pipeline ('heatclinic_pipeline') {
  ...
   stage('build') {
       ...
       gate ('PRE') {
          condition =  '$[/myPipelineRuntime/runqa]'
          task ('pre-gate-task1') {
            ...
          }
       }

       gate ('POST') {
          precondition = '$[/myPipelineRuntime/qaready]'
          task ('post-gate-task1') {
            ...
          }
       }

       task ('stage-task1') {
       }
       ...
   }
}
```

Note: Before ElectricFlow 6.5, gates are implicitly created when you create a stage. You can define tasks for a stage. If you want to define an approval task for a gate, you must define a task with a gate type (`PRE` or `POST`) and a task type of `APPROVAL`. DSL scripts created before ElectricFlow 6.5 will work, but **Run if** and **Wait until** conditions cannot be applied to them.

Pipeline Stage Summary

A pipeline stage summary shows various status and metrics relevant to the pipeline when a pipeline stage is executed. For example, the summary could be the scan results from the code coverage tool or links to reports that are generated during the pipeline run. You can view the pipeline stage summaries during the pipeline run or for completed pipeline runs.

From the Pipeline Runs view, you can click on a specific stage to see that stage's summary. The stage summary shows two sets of stage-specific information.

- Deployment summary consisting of:
  - A system-generated deployment summary.
  - Applications deployed through the pipeline and the environments where they are deployed.
- User-generated summary consisting of:
  - Links to web sites for reports and other information about the deployed artifacts.
  - Property information about the pipeline stage and the objects in it.

Creating User-Generated Data for the Stage Summary

If the task in a stage is defined by a procedure, a workflow, or an application process that includes a procedure as a process step, you can create user-generated data using ectool.

Stage Summary Information as Links to a Web Site

To create links to web site:

- Create a procedure and the underlying steps with commands like the following:

```
ectool setProperty "/myPipelineStageRuntime/ec_summary/report" --value "<html><a
href=\"http://www.myreportserver.com\" target=\"_blank\">Global
Report</a></html>"
```

- Call this procedure from a task in the stage.

Generic Property Information

- Create a procedure and the underlying steps with commands like the following.

  To create a summary property:

```
ectool setProperty "/myPipelineStageRuntime/ec_summary/testCoverage "test value"
```

  To create a property entry:

```
ectool setProperty "/myPipelineStageRuntime/ec_summary/testCoverage" --value
"80%"
```

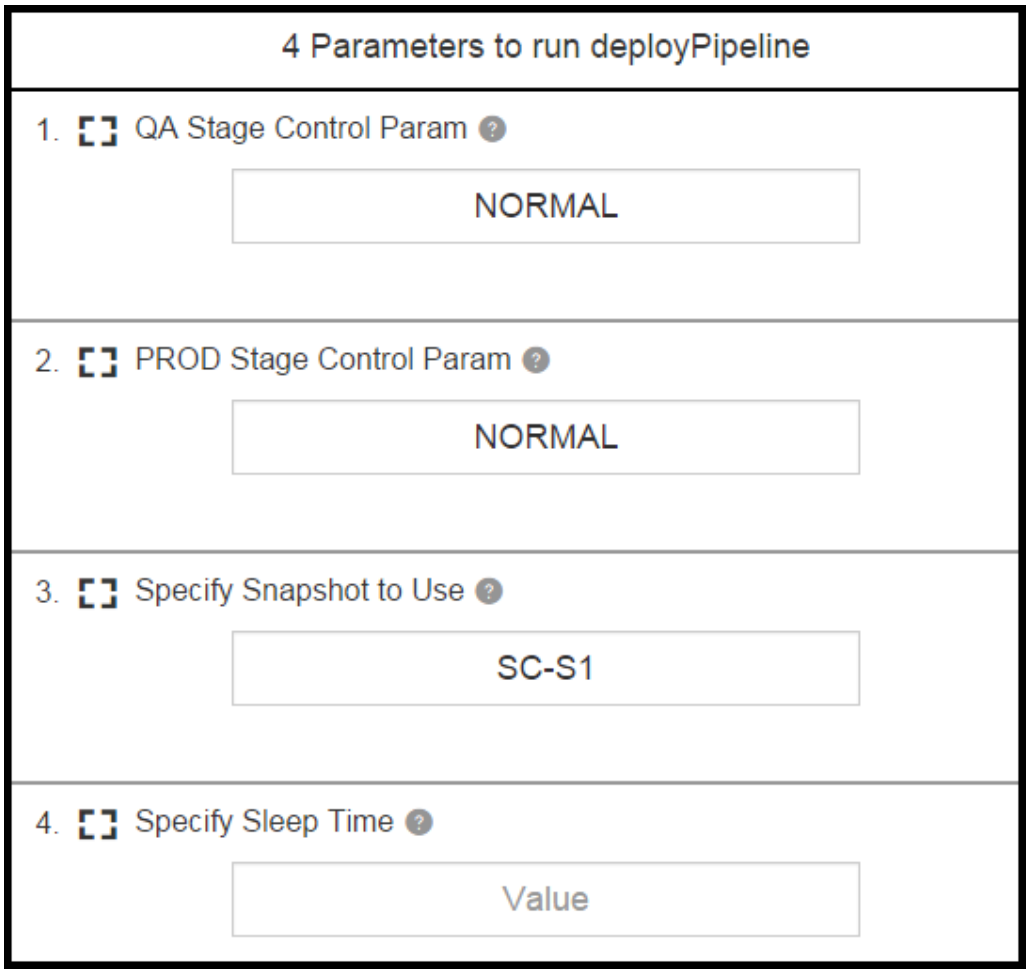
- Call this procedure from a task in the stage.

The following sections provide examples of user-generated data in the stage summary.

Viewing the Stage Summary During a Pipeline Run

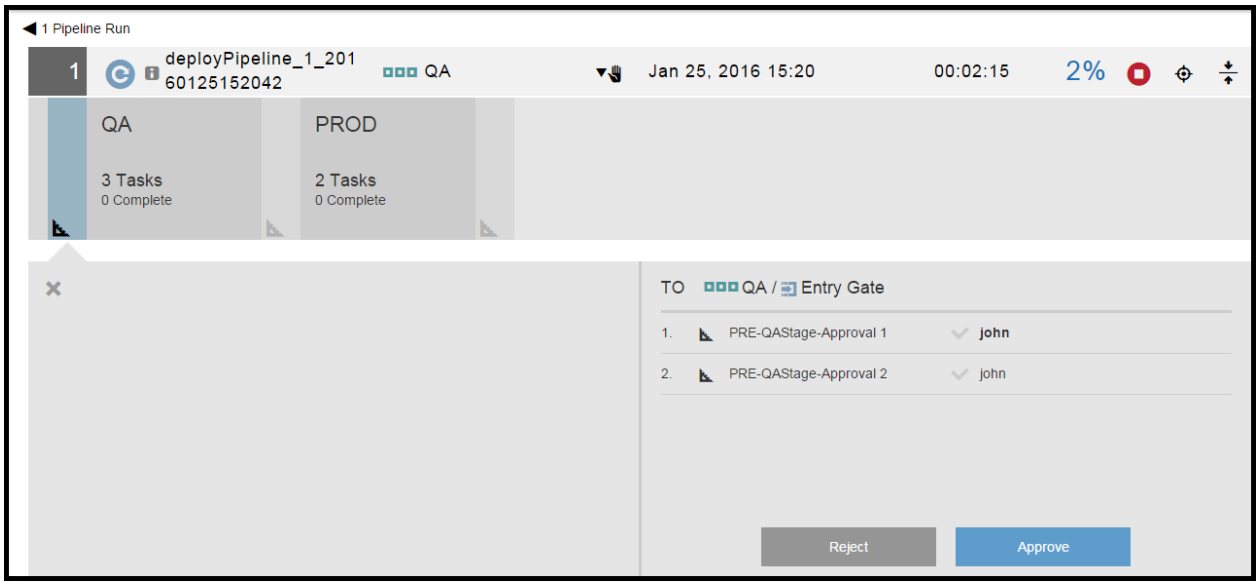
To view the stage summary of a stage during its pipeline run, start the pipeline run by selecting a pipeline in the Pipelines list and clicking the  button:

This pipeline requires input before the applications to be deployed through the pipeline. Enter the required parameters for pipeline, and click **OK**.

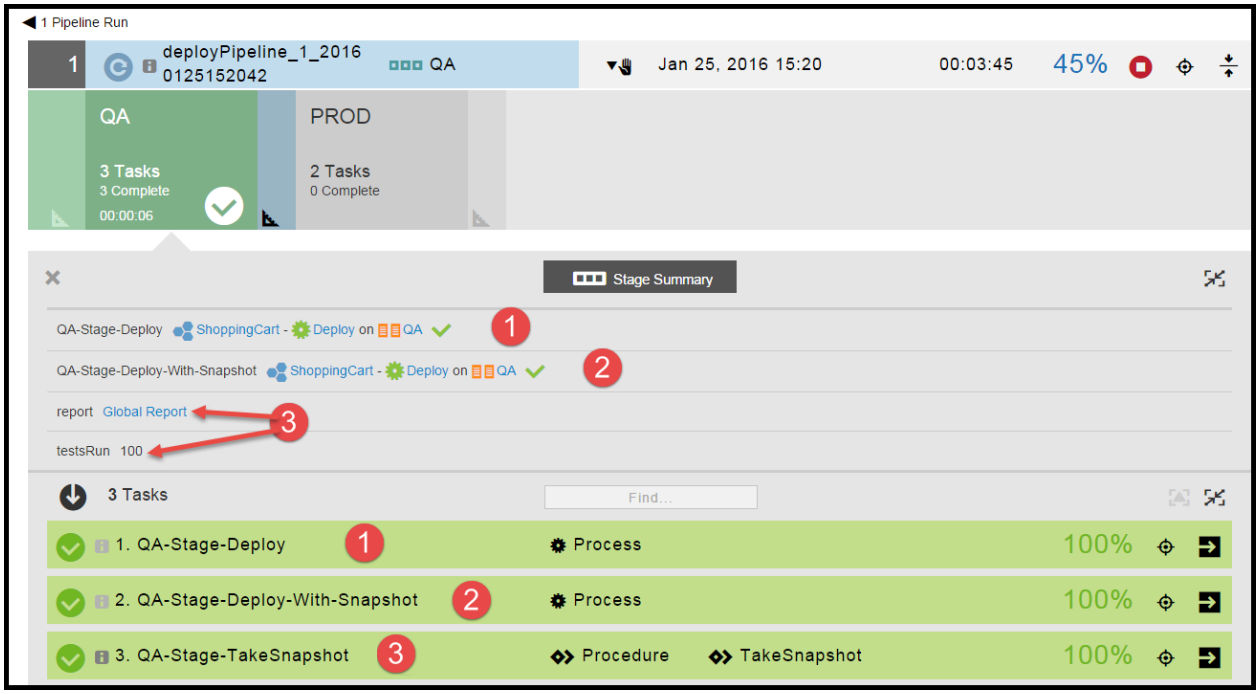


The pipeline runs. You can view its progress on the Pipeline Run page.

When manual approvals are required at the entry and exit gates to a stage, the system waits for the approvers to approve or reject before proceeding to the next step in the pipeline.



As the pipeline progresses through a stage, you can view its progress by clicking in the stage, which opens the pipeline stage summary.



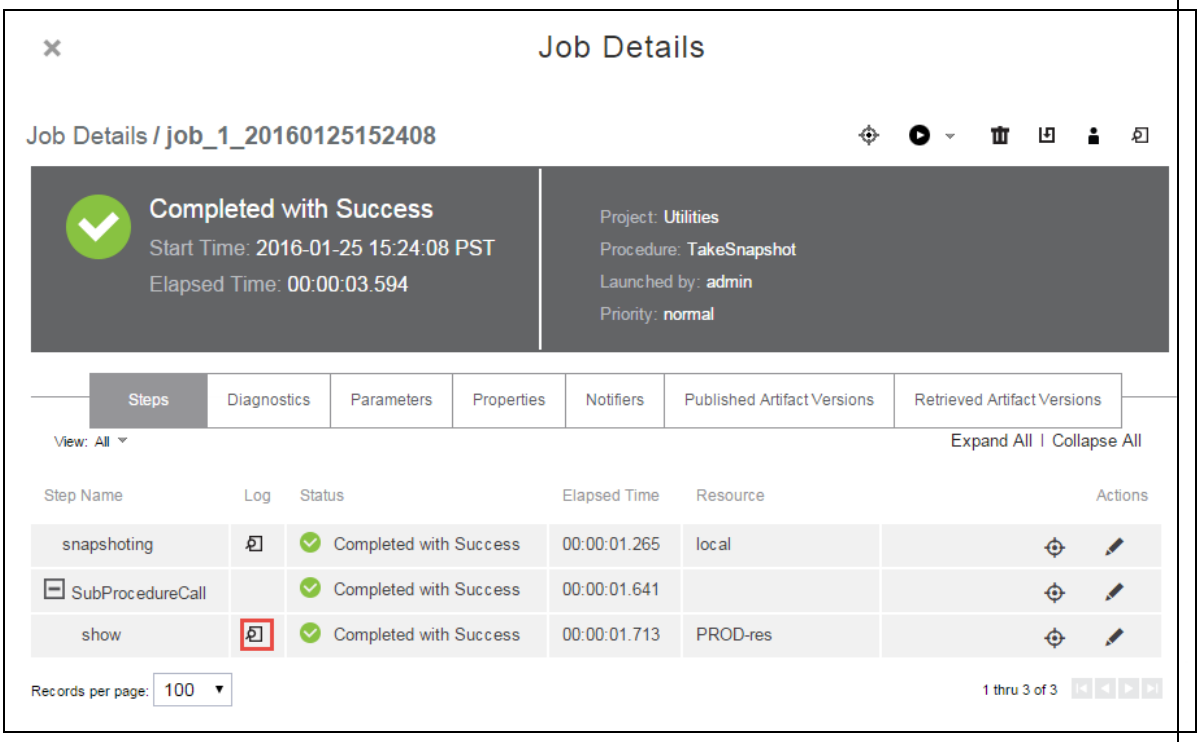
For the QA stage, the stage summary shows this information:

- Environment summary—The first two tasks (numbers 1 and 2 above) show what application processes were deployed and where they were deployed. The summary consists of the application, the application process being deployed, and the environment to which the application is deployed.

- User-generated summary–You can create two types of summary data, property information (consisting of text and/or numbers) and links to web sites (number 3 above).

  In this example, the property information is testCoverage at 80%, and the link to the report is Global Report. When you click Global Report, the web site *http://www.myreportserver.com* opens.

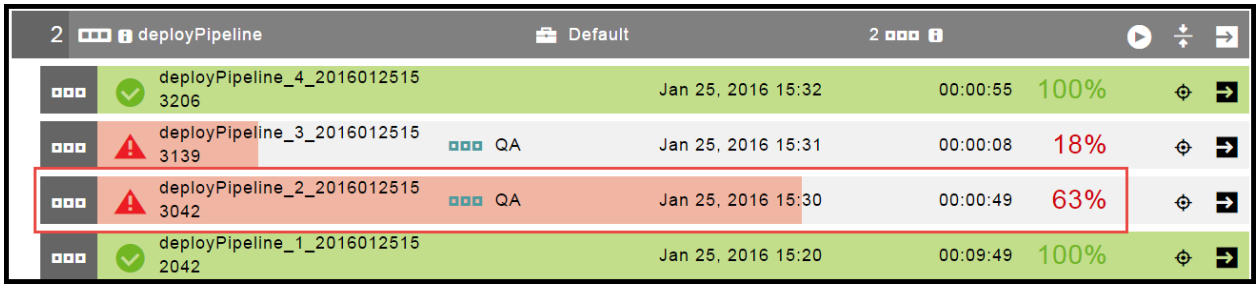
The stage summary also shows information about the tasks:

1	The first task is QA-Stage-Deploy. It is defined by the ShoppingCart application. The Deploy application process was deployed in the QA environment. The green check mark means that the task was successfully completed.
2	The second task is QA-Stage-Deploy-with-Snapshot. It is also defined by the ShoppingCart application. The "Deploy" application process was deployed in the QA environment. The green check mark means that this task was also successfully completed.
3	The third task is QA-Stage-TakeSnapshot. It is defined by a procedure called TakeSnapshot. When you click on the task name or click the button, the Job Details page opens, showing that the procedure consists of two steps. Both steps were successfully completed. You can dig deeper by clicking on the Log button in the Log column on the Job Details page. 

Viewing the Stage Summary for a Completed Pipeline Run

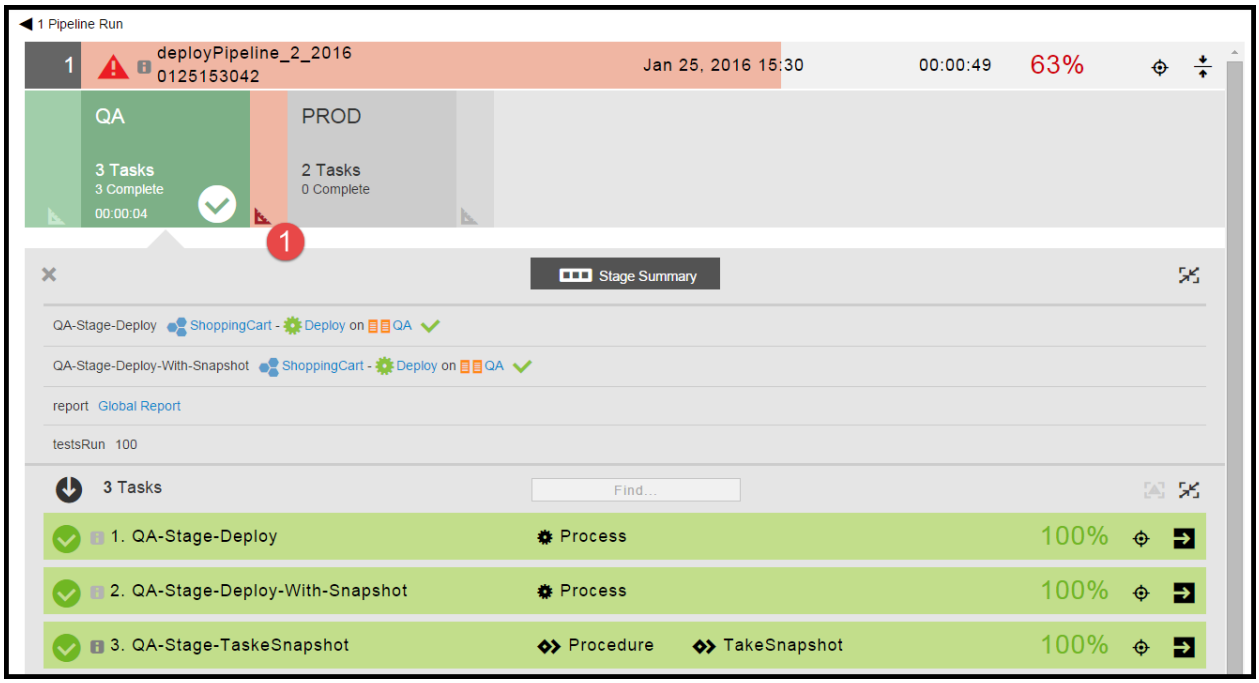
To view the stage summary of a stage in a completed pipeline run, start by selecting a pipeline in the Pipelines List and clicking the  button:

Clicking on the name of pipeline shows the list of current and completed pipeline runs.

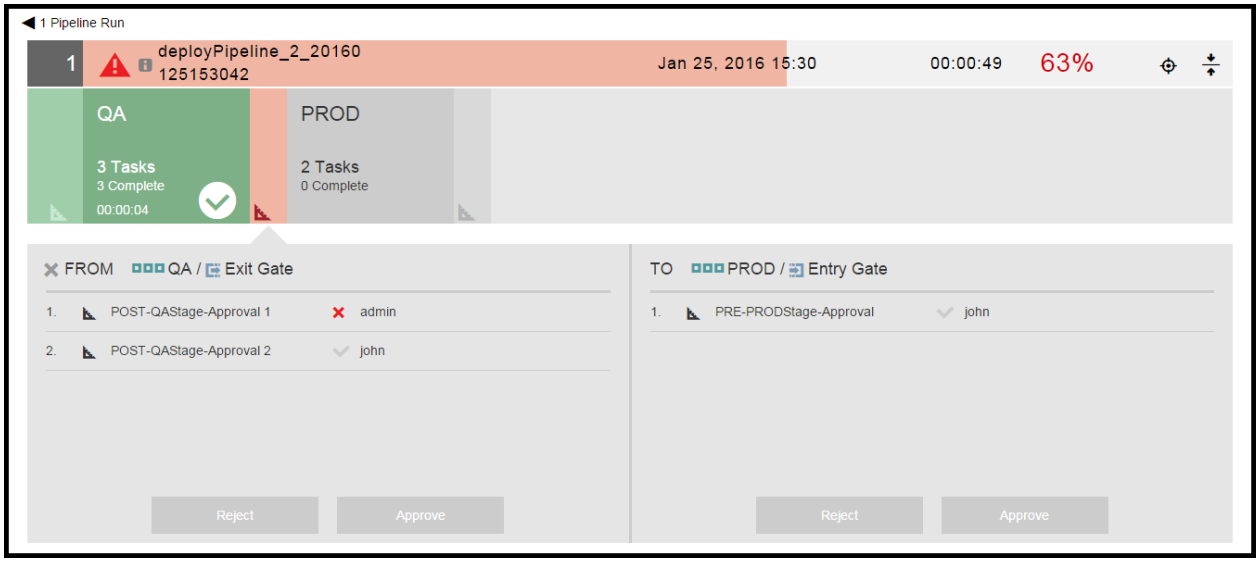


To troubleshoot the pipeline, select the pipeline run that completed with errors. Click the name of the pipeline run to view a representation of the pipeline, and then click in a stage to view the stage summary for that stage.

This is an example of a stage summary:



To find out why the pipeline was completed with errors, click the button (number 1 above) in the stage summary. The summary of the Exit Gate for the QA stage shows that one of the required approvals were rejected, so the pipeline stopped with errors.



Credentials in Pipelines

Credentials are passed to pipelines through applications or microservices in pipeline stages. If an application or microservice has required inputs (parameters) and any of the inputs is a credential parameter, you can enter the path to the credential, browse to it, select a credential parameter (**Parameter Credential**) or a credential binding to a pipeline task (**Credential binding**), or select a user-defined credential that is attached to the project associated with the pipeline.

A credential must be explicitly attached to the object using it so the server can perform an access control list (ACL) check at the definition time and limit the visibility of the password. To support accessing credentials at the pipeline task level, these tasks need to have the appropriate credentials attached. This is done implicitly by the UI when you are binding credentials to the pipeline tasks. If you are using ectool, use the `attachCredential` API command to perform the same operation.

Example: Integrating Test Automation in Release Pipelines

Automated testing is critical to identifying errors early in your pipeline and ensuring quality software is delivered to production. The results from automated tests can determine whether a pipeline should be promoted to the next stage, or if the release should be stopped from moving forward. Driving test automation flows at the right time in the pipeline becomes even more critical in order to meet the business desires of releasing more frequently.

ElectricFlow integrates with many testing tools and frameworks to run automated tests as part of the delivery pipeline. The results of these tests can be linked into the pipeline summary to have a central view of all of the data associated with the release. Test logs can be parsed to extract results and specific

metrics or data, and then this data can be bubbled up for visibility and to control promotion to the next stage using automated gates.

Acting on Test Results

Your pipeline should be data driven and should respond based on the outcome of testing with a "go or no go" on continuing the release. In line with the goal of reducing cycle times and improving quality and efficiency in your releases, these checks of test results should be an automated part of your pipeline rather than requiring a manual check of the results. After running the tests and gathering the results, the pipeline can have policy-based approvals to ensure that the test results meet a certain threshold before promoting further. Use automated gates and run conditions in ElectricFlow pipelines to check test outcomes before continuing.

When going from Dev to Staging, an automated check of test results may be a sufficient gate between the stages. As you move further right in the pipeline, it may require a combination of automated checks with manual approvals before the pipeline can be promoted to controlled environments.

Collecting and Parsing Test Data

ElectricFlow stores the standard output from any commands in log files, which can easily be drilled into to see the details of what was run and troubleshoot any issue. When running large test suites, it is not always easy to scan a long log file and find the data you need. You may need to extract certain results or numbers for easy visibility and for use later in the pipeline. In fact, many times you need to understand the log as it is generated in real time and take corrective action immediately instead of waiting for the entire process to finish.

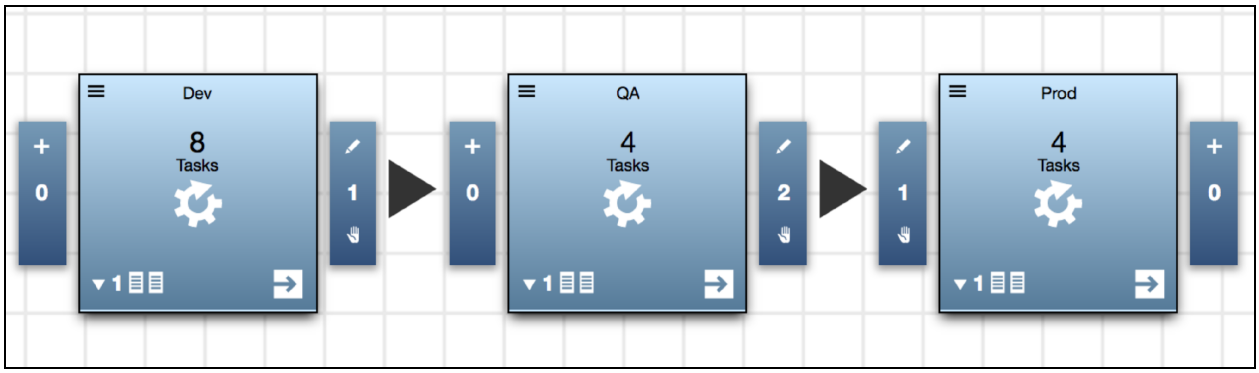
The ElectricFlow PostP (or PostProcessor) feature is a powerful and flexible way to extract important details and values from the job's log file in real time. With customizable pattern matching, PostP parses the log file as the job is running to find interesting test data that matches your regular expressions.This data can then be stored in properties that can be retrieved and acted on later in the pipeline. There are a variety of other actions that can be taken, such as extracting sections of the log file or counting the number of occurrences of a pattern. For example, PostP can be used to fail a pipeline fast if you realize that first few critical test cases of a normally long-running functional test have failed and that there is no need to continue the tests.

See Postprocessors: Collecting Data for Reports on page 1506 for more detail on how to write your custom matchers as well as the functions available for parsing and extracting data with PostP.

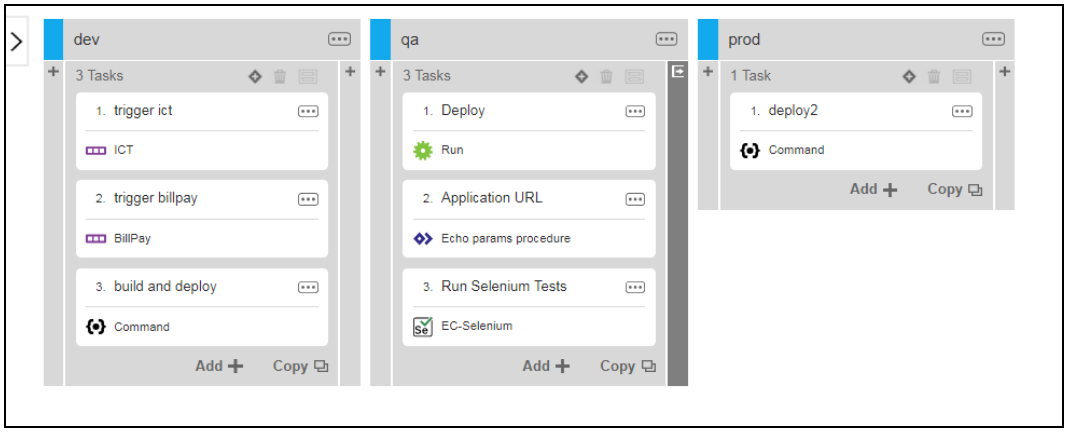
Example: Test Automation Driving the ElectricFlow Pipeline

This example illustrates how you can aggregate your automated testing data, extract the details, and act on results to drive your pipeline.

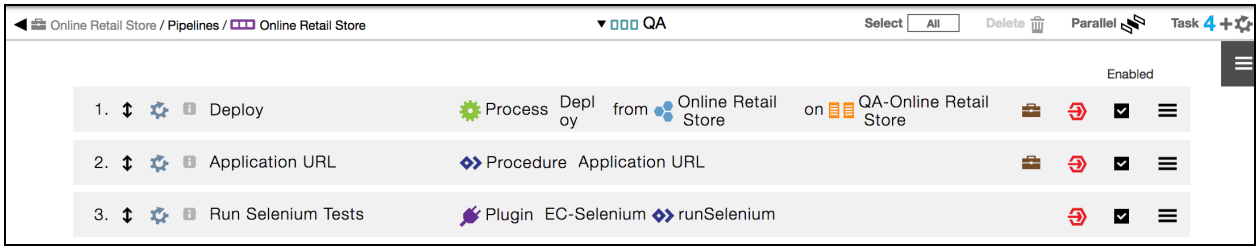
The following figure shows the pipeline discussed in this example as seen in the Pipeline Stage View:



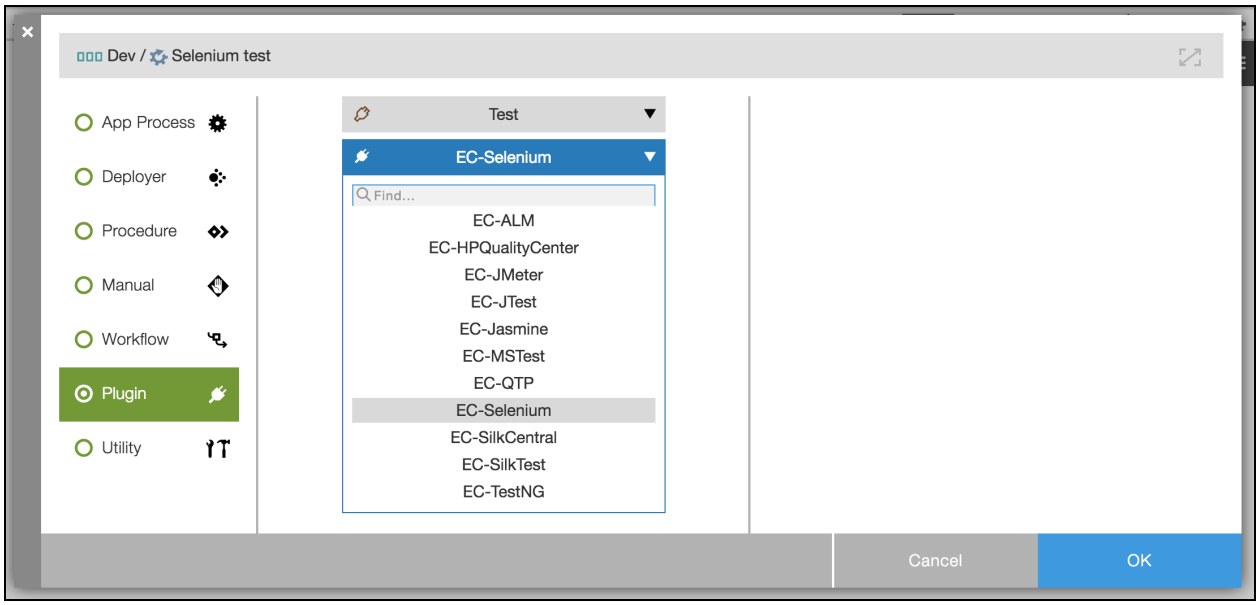
The following figure shows the same pipeline as seen in the Release Kanban View:



The pipeline has three stages, Dev, QA, and Prod. The Dev and QA stages run the application deployment process and then start a suite of Selenium Tests, calling the EC-Selenium plugin to invoke the tests.



You can replace the testing task with any testing tool by using one of the test plugins available in ElectricFlow.



A link to the test results is stored in the pipeline summary. Anyone viewing or approving the pipeline has direct access to the full details of the tests that were run for the environment. All data is aggregated centrally. For details on adding links to a stage summary, see Pipeline Stage Summary on page 603.

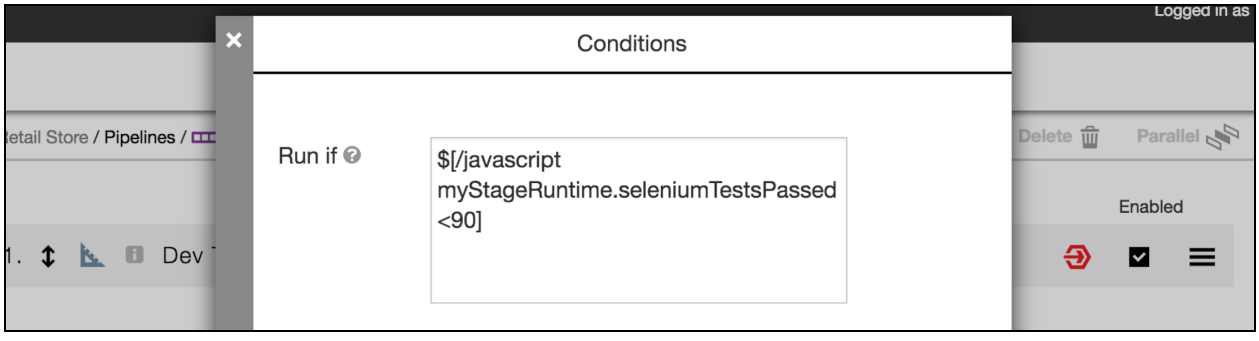
The selenium test steps leverage PostP to extract the data on successful tests passed and store them in a property in the pipeline stage named seleniumTestsPassed. These properties are then used in the automated gates between pipeline stages.

For details on defining automated gate and adding conditions, see Pipeline Concepts on page 413 and Creating a Condition-Based Approval Rule in a Gate on page 524

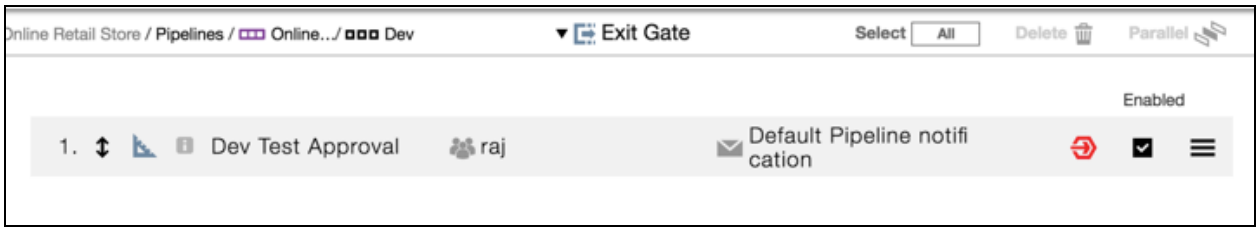
Dev to QA Promotion

There is one gate between the Dev and QA stages. This gate has a combination of automation and manual intervention. Before entering the gate, there is an automated condition that checks the percentage of Selenium tests that passed. If the percent is greater than 90, the manual check is skipped, and the pipeline starts the QA stage. If less than 90% of tests passed, the manual gate is invoked, and the QA engineer must view the test results to see whether the application can be promoted to QA.

Condition on the gate:

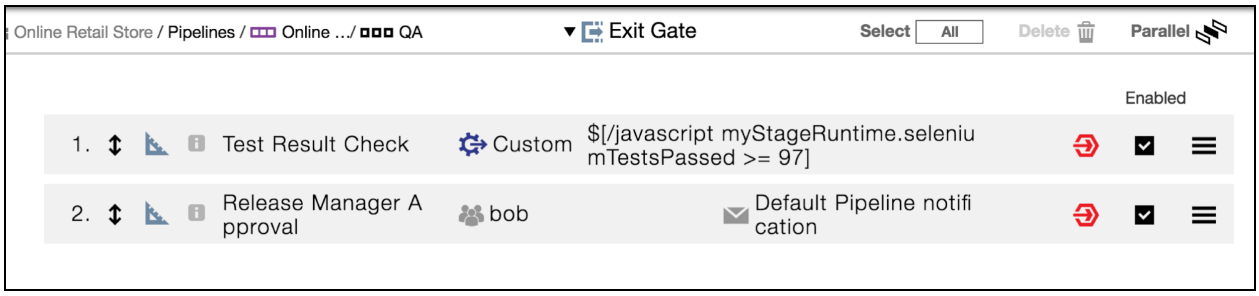


The manual gate is triggered only if the condition is met:



QA to Production Promotion

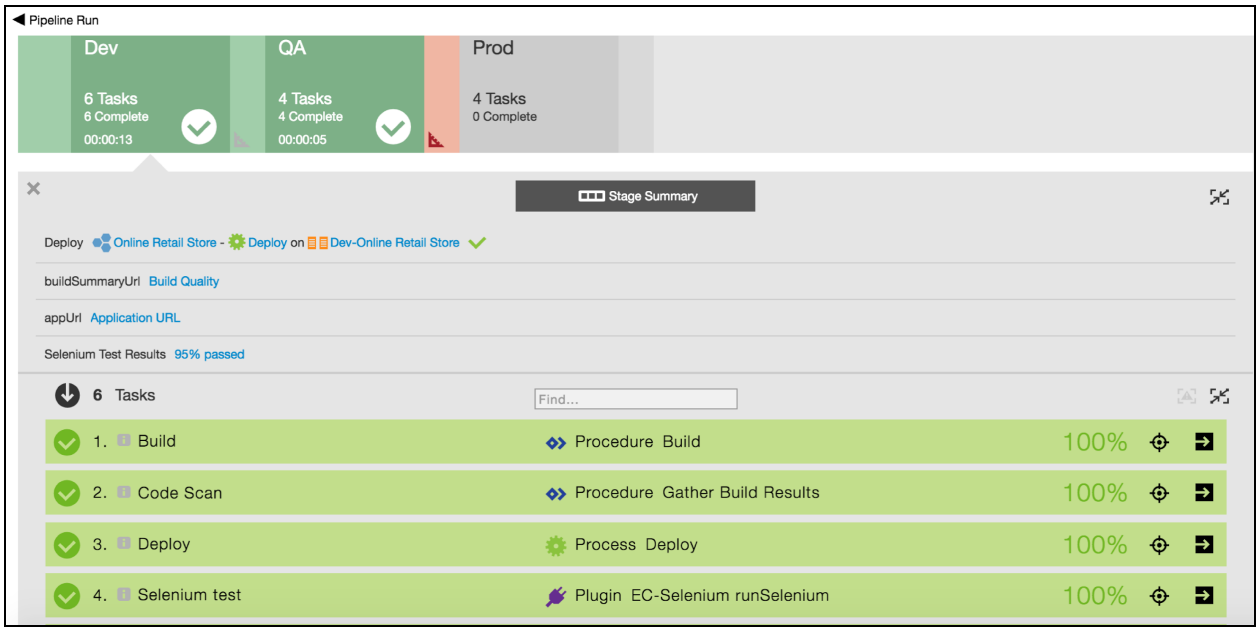
The promotion from QA to Production requires more diligence than the promotion to QA. Here there are two gates, one automated and a second manual approval, required for promotion. The first automated gate, as with the Dev stage, is based on the test results but requires that 100% of the tests passed in order to continue. If the automated gate is rejected, then the pipeline is aborted. If the automated gate is approved, then a manual gate is triggered, where the release manager will view the pipeline details with quick access to the testing results linked to the stage summary and then manually approve or reject the gate.



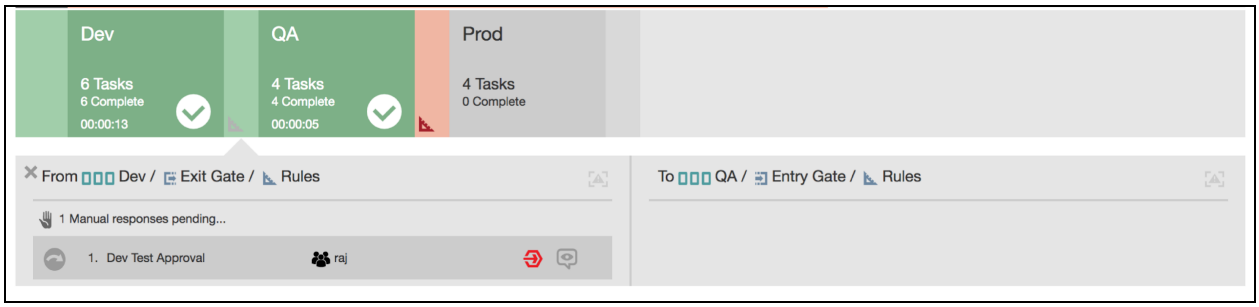
Running the Pipeline

Following is a walk-through of a pipeline run.

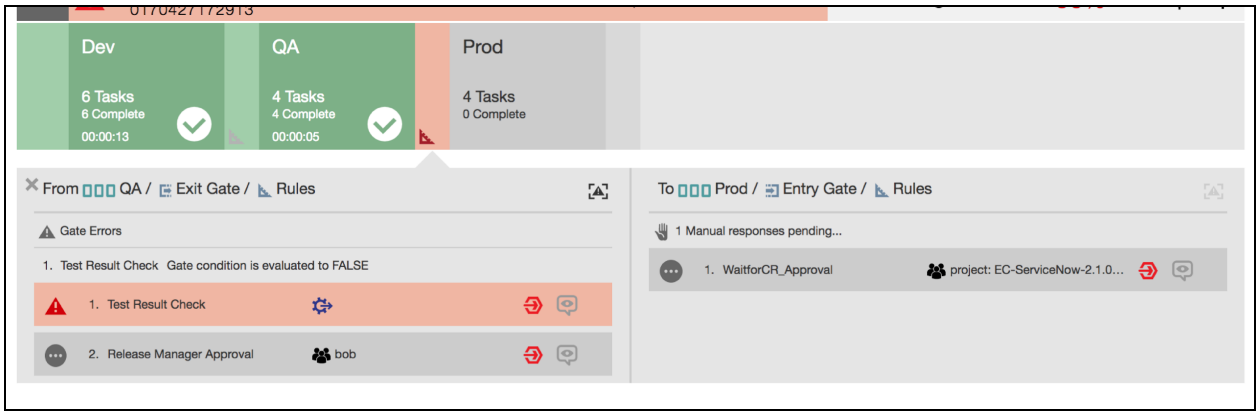
In the Dev stage below, 95% of Selenium tests passed. The test results are shown and linked in the stage summary.



Since more than 90% of the Selenium tests passed, the manual gate was skipped and the pipeline automatically promoted to the QA stage.



In the QA stage, again, 95% of Selenium tests passed. This did not meet the required 100% of test completion in the first automated gate. Since the automated gate failed, the pipeline was aborted before entering the manual gate to promote to Production.



This example shows one scenario and uses automated and manual gates to control progression of the pipeline. ElectricFlow provides the flexibility to combine automated, procedure based, and manual gates as needed to ensure that your pipelines include the rigor required to meet your quality and compliance standards.

Leveraging Test Data Management and Service Virtualization in Release Pipelines

Automating your release pipelines makes the path to production seamless, efficient and reduces risk. To maximize these benefits in your pipelines, it is imperative to include effective automated testing strategies. Tests in lower environments, like development, testing and QA, should mimic the production environment as closely as possible. Waiting on having the right components ready or incomplete and unrealistic data can slow down the entire process and increase the your cycle time, impacting your time and cost per release. Integrating test data management and service virtualization directly into your release pipeline means efficient test cycles while providing reliability and ensuring production readiness.

Test data management allows testing against production-like data in preproduction environments. Test data management tools provide the ability to create and manage test data that is anonymized and random, but also provides coverage of all test cases—and then support the use of that data in testing.

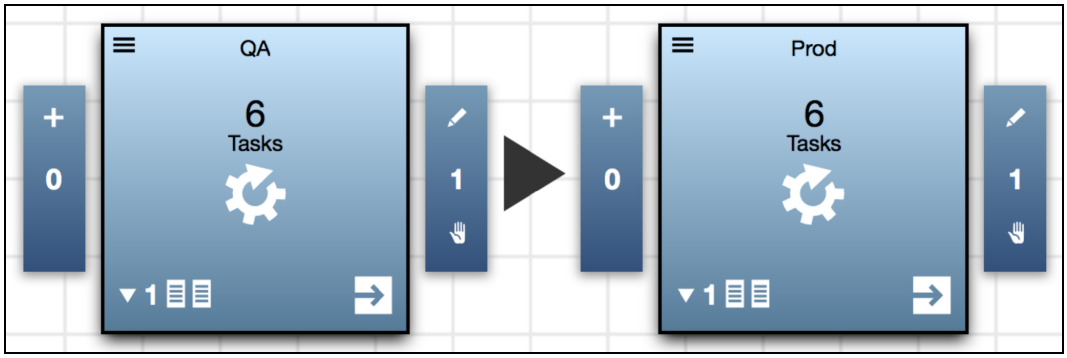
Service virtualization enables use of virtual services instead of depending on readiness of production services that might not be ready or available in lower environments. The virtual services emulate the components in production and allow comprehensive testing that can happen more frequently and earlier in the pipeline.

Example: ElectricFlow Pipeline with Test Data Management and Service Virtualization

Test data management and service virtualization strategies should be automated and integrated into applications deployments and release pipelines to enable comprehensive and reliable system testing. This means provisioning the virtual services and representative virtual data to create environments that provide full coverage of any scenario or corner case.

This example shows strategies for using these practices in a release pipeline in ElectricFlow. In this example, the EC-Parasoft plugin is used to integrate with Parasoft as the tool to manage and import testing data and create environments with virtual services for testing. The same patterns and strategies can be used to model pipelines using other test tools by replacing calls to Parasoft with the appropriate calls to the other tool.

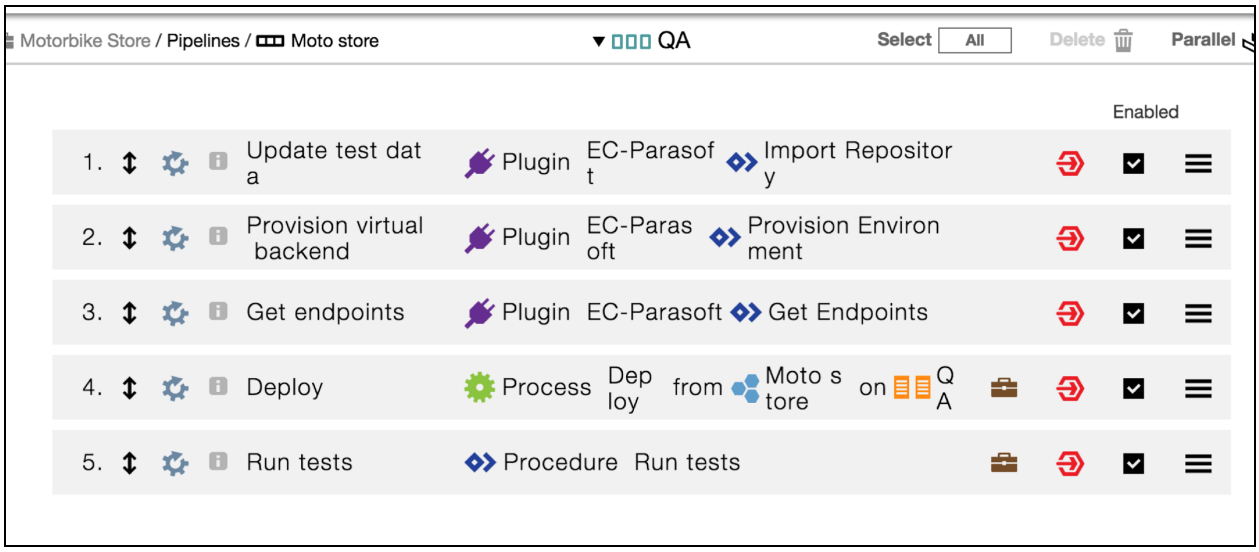
The pipeline has two stages, QA and Prod, for deploying the motorbike store application in each environment and then running system tests. In the QA stage, test data must be imported, and virtual services must be generated in order to create a complete environment for system testing. Test data and service virtualization are not required in the Production stage because the production environment has all production components and data available.



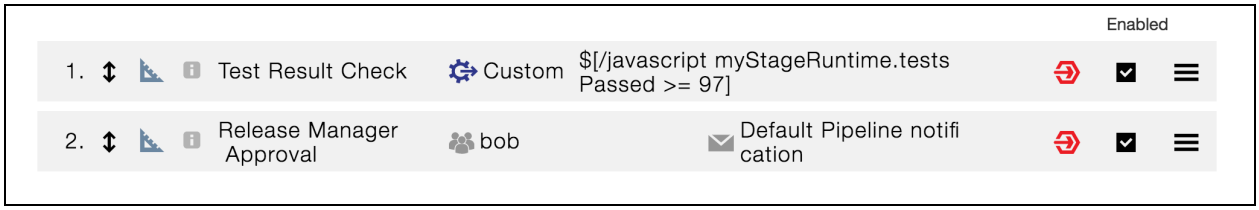
QA Stage

The QA environment does not have access to all required production services and requires test data to run full system tests. Before starting the application deployment or tests, the first tasks in the pipeline prepare the test data and virtual services for the environment.

- The first step imports the test data to the server using the EC-Parasoft plugin action to import a repository.
- Next, the "provision environment" plugin action is called with parameters to provision a Parasoft environment with the required virtual services.
- The endpoints for the virtual services are retrieved from the newly provisioned environment to be used in the tests.
- Now that the environment and data are prepared, the motorbike store application is deployed by calling the deploy process for the model.
- Finally, system tests are run for the application using the test data and provisioned environment with virtual services.



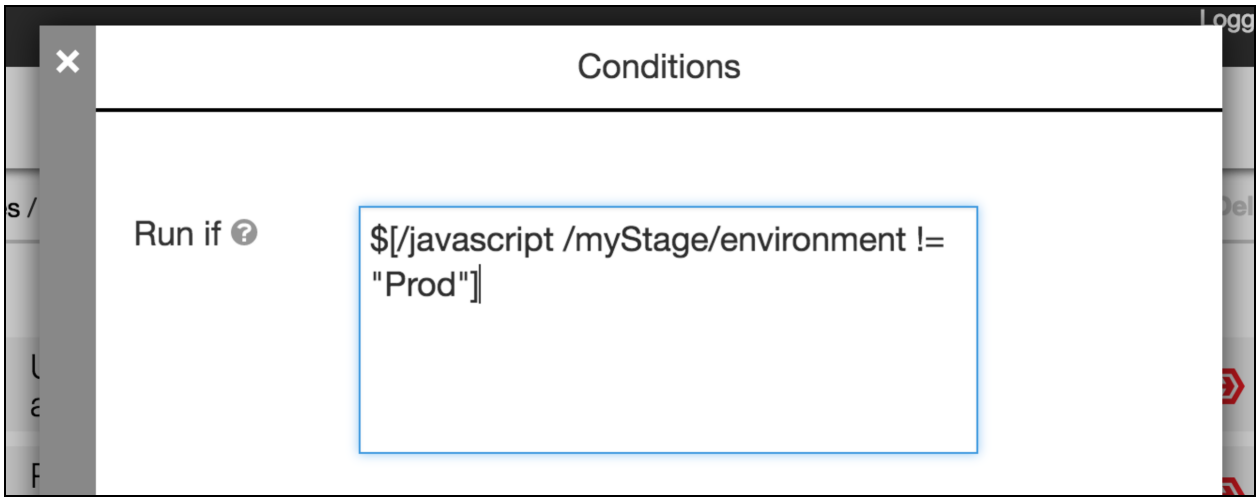
After the QA stage is complete, an automated gate is based on a condition that checks the test results. If greater than 97% of tests passed, the gate is automatically approved, and the pipeline is promoted to the Production environment. If the tests results do not meet this threshold, the gate is rejected, and the pipeline stops execution.This is just an example of automated quality gates. You can add any conditional check to automatically control the pipeline progress.



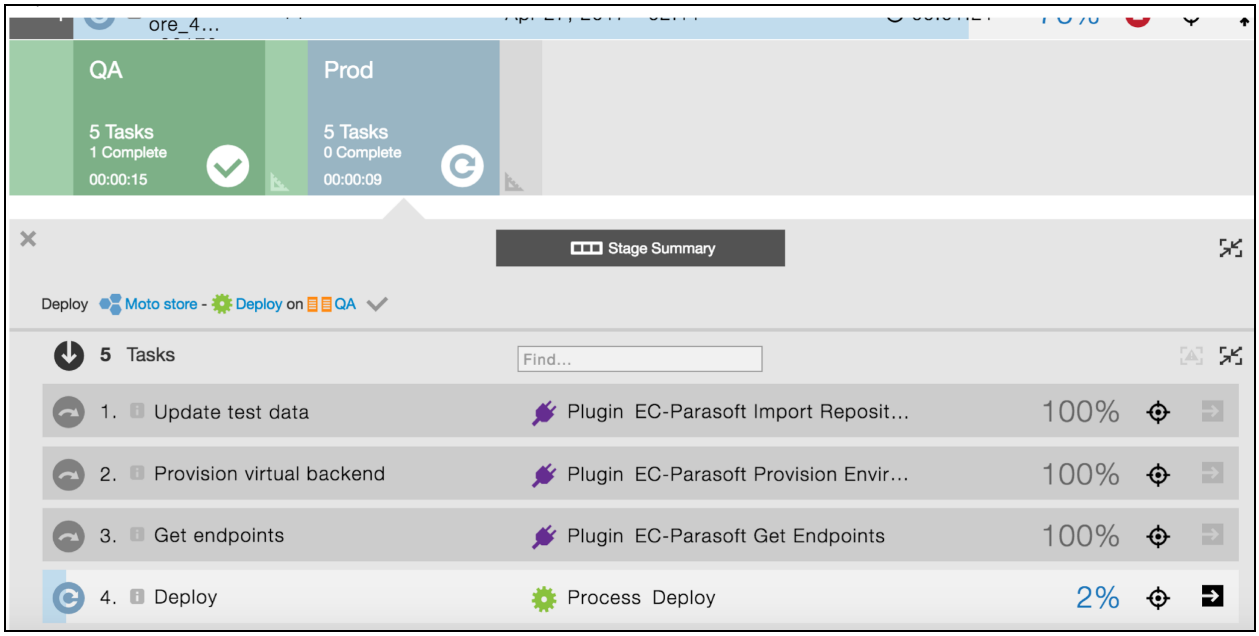
Production Stage

The Production stage does not require test data or virtual services. Conditions on the tasks are used to skip the steps for importing test data and provisioning virtual services. The condition checks which environment is being run in the stage by checking a property stored on the stage. If it is the production environment, the test data management and service virtualization steps are skipped for the stage.

This example uses Parasoft for test data management and service virtualization. Other tools can be leveraged with the same approach in modeling the pipeline



When the pipeline is run, all of the test preparation steps are skipped, and the application deploy process is run directly:



Other Strategies and Actions

Instead of calling test data management and service virtualization from pipeline tasks, these actions could also be called from within the application deployment process itself. Similar conditions can be used to skip these steps for production deployments. The condition can reference the current environment being deployed to, and can be set on the entry into the steps that need to be skipped.

Aside from importing test data as show in the example, integrations with Test Data Management systems can also be used to manage and update data set. This is done with the EC-Parasoft plugin using the "Update Data Set" and "Update Record" procedures.

See the EC-Parasoft plugin help file for details: http://electric-cloud.com/plugins/directory/p/parasoft.

Chapter 4: Release Management

The Release module captures, executes, visualizes, and controls the life cycle of multiple application or microservice enterprise releases. Using the Release capability, outputs from multiple teams can be coordinated to produce a final release to be pushed to production. You can create a Release model consisting of multiple applications or microservices deployed on different systems, such as traditional, cloud, mainframes, and remote servers. You can then run the Release pipeline reliably and repeatedly until the software release is complete.

You can use the Release feature to deliver software releases through traditional methods where changes from multiple applications or microservices are bulked up and are released together as a unit through a pipeline on a regular cadence (schedule) such as weekly, monthly, quarterly, or other frequency. In such traditional scenarios, the Release capability allows you to manage dependencies between multiple applications or microservices. In addition, you can even use the Release feature to cater to Continuous Delivery-style releases where code check-in from a developer can traverse all the way to production through various pipeline stages and approval gates. The Release feature allows ElectricFlow to manage all the releases in one platform, regardless of the release methodologies used by the release team. Anyone can quickly get a status of a release and then access the Release details. This information can also be used to troubleshoot and improve the software release process.

- Visibility and coordination

  The *Release Dashboard*, also referred to as the *Release List*, allows the release team to get a bird's eye view of all the releases that are planned, active, or completed. For a specific release, you can easily see its status, what the milestones are, if the release is blocked for some human intervention, and the Release's progress.

- Path to Production (or what is where)

  For each stage in the Release pipeline, the Path-to-Production view shows the applications or microservices deployed in the release, the deployed versions, and the environments to which they are deployed. The details in this view include the application or microservice versions, the snapshots, and the environments that are not in compliance with the bill of materials throughout the Release.

- Control

  ElectricFlow keeps track of the data from the multiple teams involved in the release. The Release feature captures, validates, coordinates, and tracks all the details in one place, where everyone can access the *Release definition*, bill of materials (which applications or microservices to release),the pipeline controlling the release process, and environment to use across stages, approval conditions, release configurations, and so on.

  When the Release is run, ElectricFlow automatically coordinates the pipeline information needed to deploy applications or microservices in each stage. If you need to add an application or microservice, change an application or microservice version, or deploy to a different environment, you need to change only the bill of materials. You do not need to revise the pipeline definition or pipeline tasks.

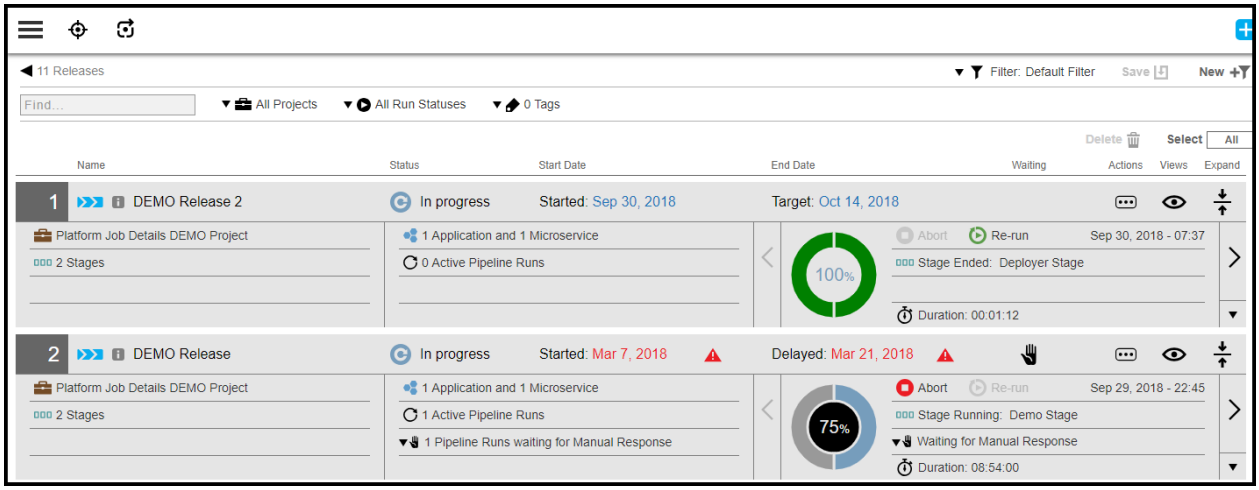
  You can reuse the pipeline to deliver multiple releases of the same software.

The *Release manifest*, also referred to as the *Release definition*, specifies the release process flow for the applications or microservices to be released, which include:

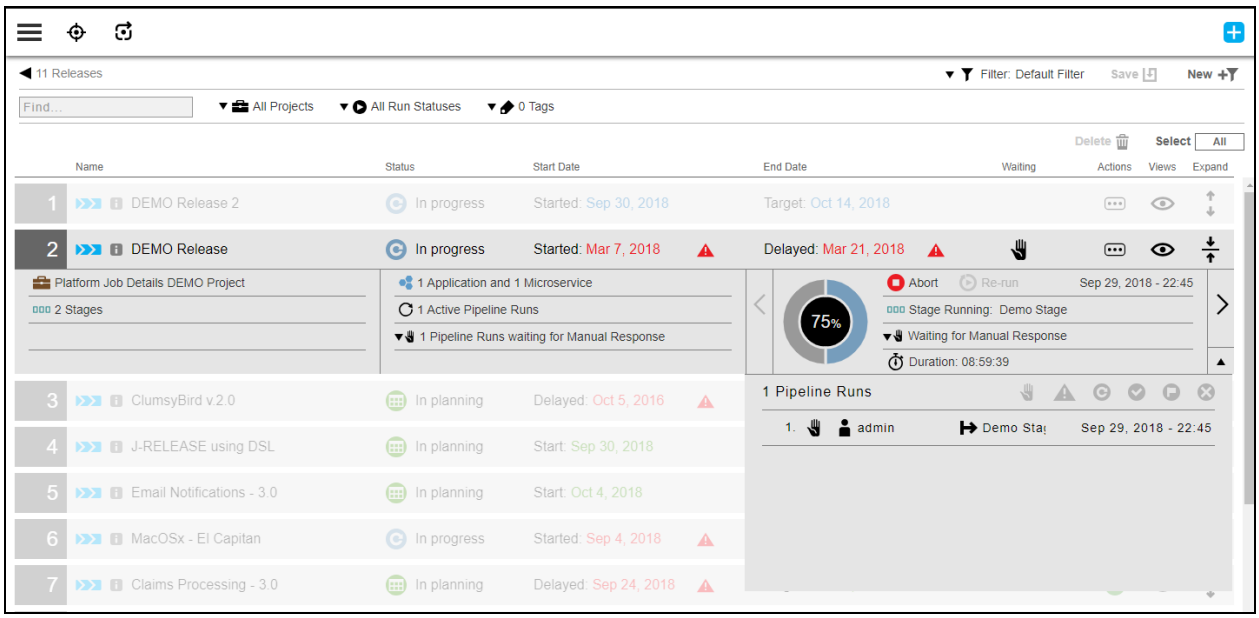
- The pipeline controlling the process.
- The applications or microservices in the pipeline for multiple-tiered deployments.
- How to deploy tasks and processes using run-time parameters and other settings, such as smart deploy and artifact staging.
- Where versioned objects will be deployed (multiple-tiered environments).
- The approvers for manual process steps, manual tasks, and pipeline gates.

Multiple Pipeline Runs in a Release

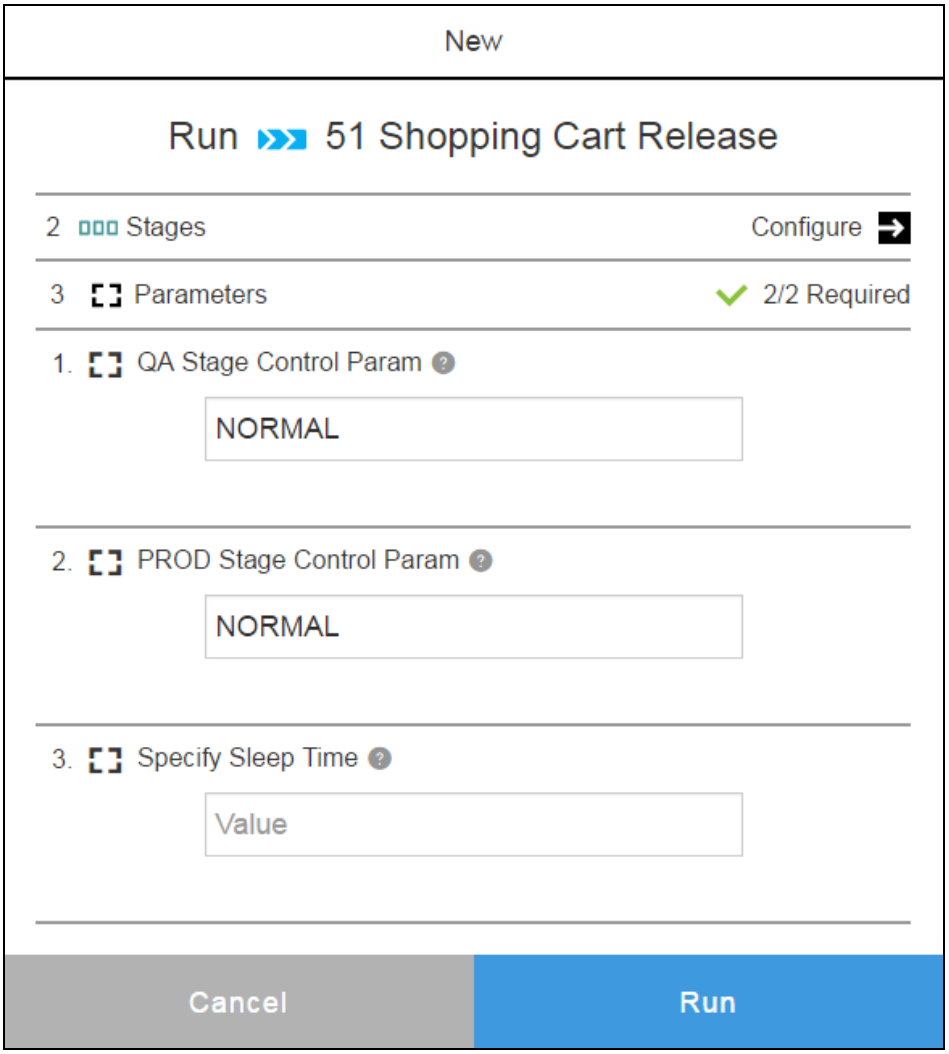
You can launch multiple instances of a pipeline associated to a release:



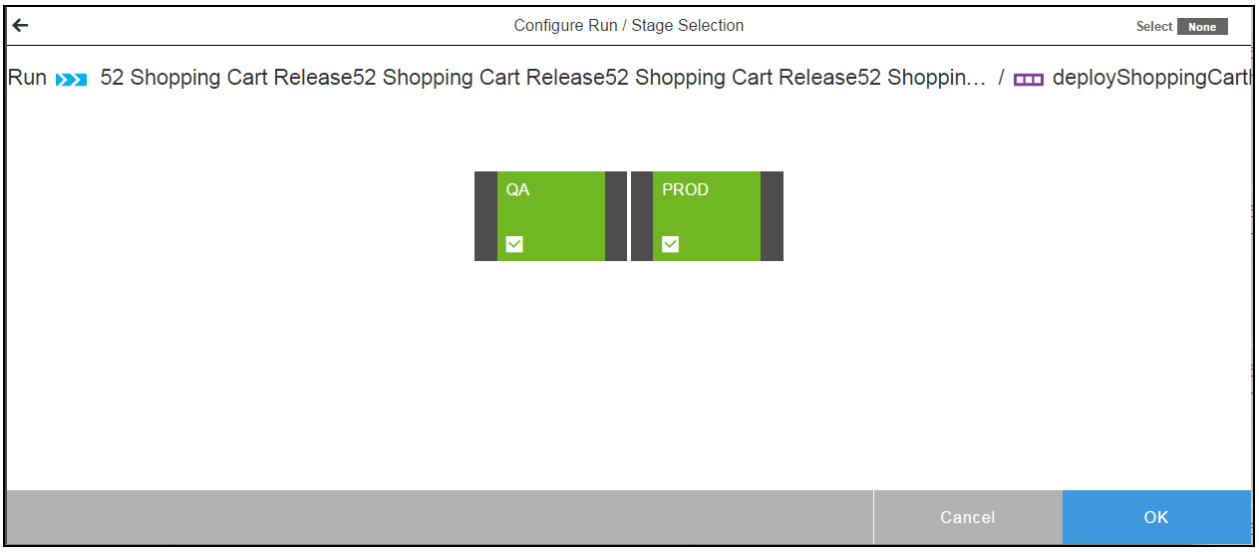
You can navigate through multiple pipeline runs:



You can pass pipeline parameters when starting a release:



You can also select the pipeline stages to run in a release:



Release Scheduling

You can create schedules for releases. For example, where teams are responsible for each stage and they want to execute only the releases for their stage. With the ability to schedule releases and configure stages to run, each team can set up their own independent schedules that run on a particular stage. They can set a start date and can also make a release recurring (for example, daily or weekly).

Release Concepts

Releases allow you to orchestrate multiple applications or microservices across multiple environments and to manage the software release life cycle using one platform. It brings together everything you need to take your software through its journey to production, regardless of your software release processes. You can manage traditional release processes (culminating in one big software release at the end of the software delivery life cycle), CI scenarios (build-test automation), CD scenarios (deployment automation), or a combination of these methods using a single platform. Dependencies can also be managed and controlled with this platform.

Modeling Releases

You can model a Release by defining the Release process that consists of

- The bill of materials consisting of the multiple applications or microservices that you want to deploy.
- One or more environments where the applications or microservices will be deployed or installed during the Release run.
- Pipelines and any inputs to the applications or microservices (parameters created for them) specifying how the applications or microservices will be deployed. Pipelines are the paths to production, and all the other objects in the Release are linked to the selected pipeline.
- The pipeline approval gates that determine when tasks in the stage can start or when the pipeline can enter or exit a stage.
- Approvers and task assignees perform specific activities as the Release runs. Individual users and teams are notified when an action needs to be taken.

The Release feature leverages the pipeline capability and a special type of pipeline task called a *deployer* to deliver an *auto-flexing* mechanism where the same pipeline can be used across multiple releases. For example, you have a standardized process for your major releases, and the only things that normally change are the applications or microservices and their released versions. In this case, you do not want to specify the exact applications or microservices to deploy as part of the pipeline tasks, which do not allow for pipeline reusability. This is where a deployer task in the pipeline and Release capability work together to provide reusability.

The deployer task, previously mentioned in Pipelines on page 413, is a special type of task specified in the task list of a pipeline stage. It gets the actual list of applications or microservices (and the number of applications or microservices) to deploy from the bill of materials in the Release definition. This list is considered the payload of the Release. At runtime, the list of applications or microservices to deploy is passed to the pipeline through the Release object. The pipeline then deploys the applications or microservices in sequential order. So you can create one pipeline that works for different software releases.

You can have multiple deployer tasks in a pipeline stage. Using multiple deployer tasks in a pipeline stage lets you deploy applications or microservices in parallel rather than in series.

In a pipeline stage, you can define a task using out-of-the-box third-party integrations (plugins) such as:

- Artifact repositories such as Git, Perforce, Subversion, Microsoft Team Foundation Server (MS TFS), Accrue, ClearCase, Repo, and Vault
- Issue tracking products that provide application or microservice developers with requirements tracking such as HP Application Lifecycle Management (HP ALM), Bugzilla, ClearQuest, Fortress, JIRA, HP Quality Center (HPQC), Rally, IBM Rational Team Concert, TeamForge, and TestTrack
- IT service management (ITSM) products such as ServiceNow
- Project management products such as Rally

For a complete list of supported plugins, go to the Electric Cloud Plugins Directory.

For details about tasks in pipeline stages, see Pipeline Tasks on page 419, Defining the Tasks in a Pipeline Stage Using the Pipeline Stage View on page 468, and Editing a Pipeline Definition on page 516.

ElectricFlow provides multiple project support for objects in application deployments, pipelines, and releases. These objects as well as the objects belonging to them can be in any project within ElectricFlow.

All objects in ElectricFlow have these capabilities for better management and maintenance of end-to-end software releases:

- Projects are top-level security containers that provide access control at the project level. All objects in a project inherit the access control settings from the project to which they belong.
- Projects are also ways to provide logical groupings of objects. If your deployment is large and has many production environments, you can organize production environments by project, with each project having its own access control settings. This makes it easier to manage and maintain multiple environments during the software delivery process.

Setting Up Releases

This section describes how to set up releases. To get started defining the Release process, you select a pipeline on which the process is based. This pipeline needs to have one or more deployer tasks in its stages. Then you select the applications or microservices that you want to deploy (the bill of materials),

the environment where the applications or microservices will be deployed, and any inputs to the applications or microservices. For instructions, see Release Definition on page 646.

To get a bird's eye view of all the releases that are planned, active, or completed, see Release Dashboard on page 669. To view the bill of materials, see Path-to-Production View on page 680 To view the pipeline's progress during the Release run, see Pipeline Stage Summary on page 603. For instructions on how to set up objects in a release:

- See Examples: Modeling and Deploying Applications on page 255 to set up applications or microservices.
- See Master Component Examples on page 68 for examples of how to create and add parameters to your applications or microservices. The steps apply to applications or microservices with master and normal components.
- See Pipelines on page 413 to set up pipelines.

  ElectricFlow supports resource exclusivity at the platform level. This allows a procedure to lock a resource for a job step and execute the tasks. The job step acquires and retains exclusive use of a resource, ensuring that the resource is always available for the step. See Reserving a Resource for Job Step Duration on page 1608 for more information.

Important: When an application, a pipeline, or a release is cloned across different projects, you might need to fix the references after the objects are cloned. For example, if a pipeline with an application reference in the "default" project is cloned to a different project, the application reference needs to be fixed after the pipeline is cloned.

Important: When you export a project while a pipeline is in progress, only the full export includes flow runtimes that have been completed. If you want to include in-progress pipeline runs in the path-to-production view and the visual indicators showing their percentage completed in the Release Dashboard, set the `excludeJobs` argument to `0` or `false` in the `export` command. When the XML file is imported, the in-progress pipeline runs in the imported project are displayed in the path-to-production view and the Release Dashboard. The jobs might be incomplete once the XML is imported.

Controlling the Releases

ElectricFlow provides these ways to ensure that the Release runs are successful:

- Manual gates:

  For each stage in the Release pipeline, an entry gate controls when the tasks can start. If the conditions and approvals are met, the process flow can proceed and the tasks can be executed. The process flow can complete the stage and proceed to the next stage when the exit conditions and approvals are met.

  ElectricFlow sends notifications to the appropriate approvers, users, and/or groups.

  See Visibility and Status of Release Pipelines on page 638 for an example process flow.

- Preconditions:

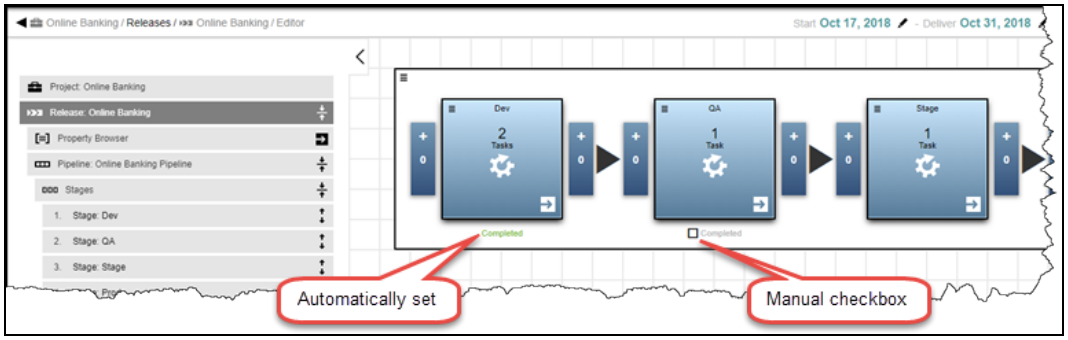
  You can set **Run if** and **Wait until** conditions on stages, tasks, and gates in pipelines. See Pipeline Concepts on page 413 for more information.

  You can set conditions in process branching when authoring application, microservice, and component processes. These conditions determine the path that the Release takes through the process flow. See Process Branching on page 112 for how to set conditions on the transitions between process steps.

  You can set preconditions on procedure steps. See Step—create new or edit existing step on page 1171 for how to set preconditions in procedure steps.

- Completion status:
  Completion status of each stage is recorded with a completed flag. This flag is set automatically or manually, based on the **Completion Status Update** setting in the stage details.
  - Automatic completion: The completed flag is automatically set the the stage's post gate completes, or if no post gate, when the last task in the stage completes. If a stage is rerun after it has been marked completed, it is automatically unset (stage now marked as incomplete).
  - Manual completion: After the stage's post gate completes, or if no post gate, when the last task in the stage completes, the completion status must me manually set. In order to rerun after it has been marked completed, it must be manually unset (stage now marked as incomplete).

  View a stage's completion status from the **Release Editor**. Stages pending manual completion update are noted with a checkbox.



- ServiceNow integration: Use the ServiceNow plugin to integrate IT service management (ITSM) with ElectricFlow.
- Plugins: You can use third-party plugins to execute the conditions and rules for approvals and quality gates.
- Postp: You can use postp to get data from the log files, make it appear on the Job Details page, and use it to take action on the Release run. See Postp on page 639 for more details.

Release Planning and Tracking

ElectricFlow can capture planning data beyond just the planned start date and end date of the Release. For better planning and tracking, teams involved in a Release can capture planning data at a granular

level. Release managers can analyze the release pipelines, get status of different stages and tasks, identify bottlenecks, and reduce failures.

Hierarchical Releases and Pipelines with Portfolio Views

Enterprise releases can be complex with many moving parts, nested hierarchies, and dependencies. Enterprise releases are often composed of other "subreleases" and groups of applications, each with its own pipeline and timelines.

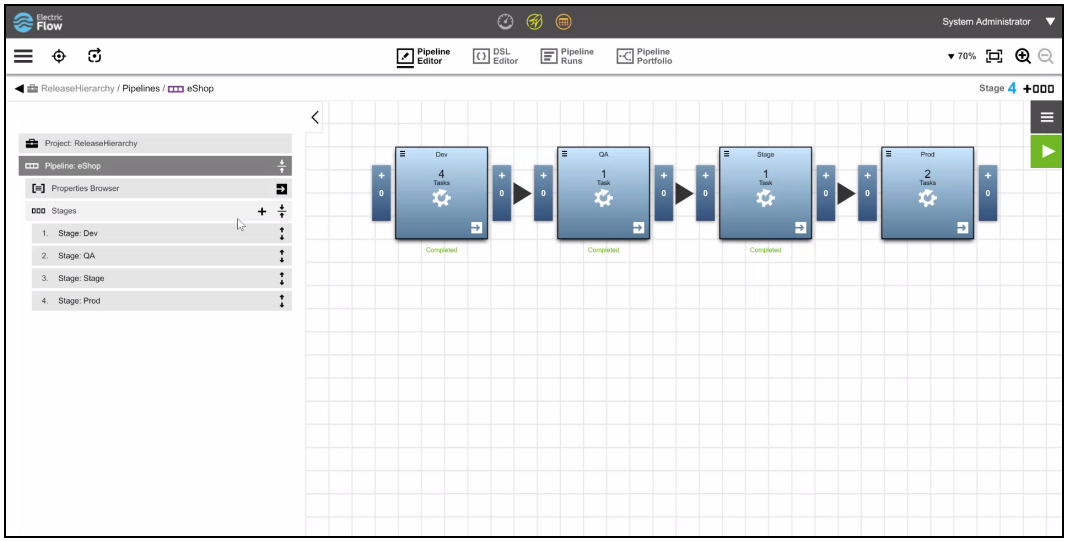
With support for hierarchical releases and pipelines in ElectricFlow, you can model, automate, and visualize dependencies across releases. You can use ElectricFlow to set up releases or pipelines that trigger subreleases, with the ability to model multilevel hierarchies of nested releases. Triggers and dependencies can be configured anywhere, with configuration options to provide needed flexibility.

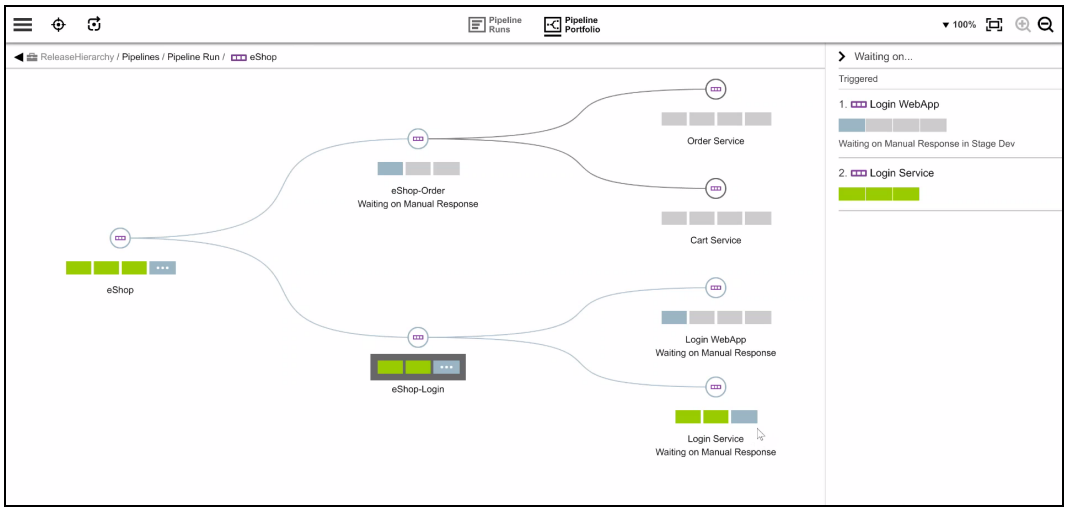
In addition to dependency modeling, the Pipeline Portfolio View and the Release Portfolio View in ElectricFlow provide top-down visibility into the entire dependency hierarchy. At runtime, this view gives detailed insight into the status of each pipeline in the hierarchy, down to the stage level to show successes, failures, and waits for manual tasks or dependencies. This view provides the status of all related releases at a glance, including drill-downs into dependencies so you can easily understand progress and identify delays, which significantly simplifies enterprise release management.

The Pipeline Portfolio view and the Release Portfolio view show duration information for each pipeline in the hierarchy and its stages visible at the top level.

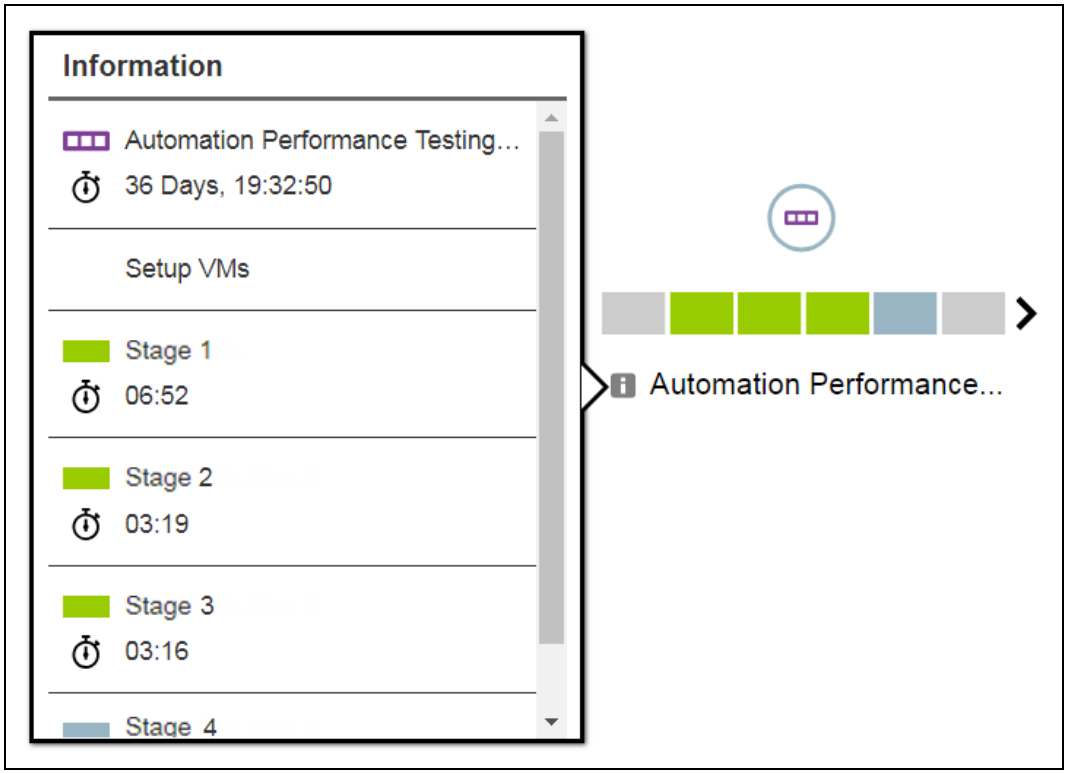
Pipeline Portfolio View

To view the Portfolio View for a pipeline, click the (Pipeline Portfolio) button at the top of the pipeline editor. The following screenshots provide an example of hierarchical pipelines with a master pipeline named eShop and its Pipeline Portfolio View at runtime:





You can view the duration of a pipeline or a stage by clicking the (information) button, which opens a popover that displays the duration times. For example:



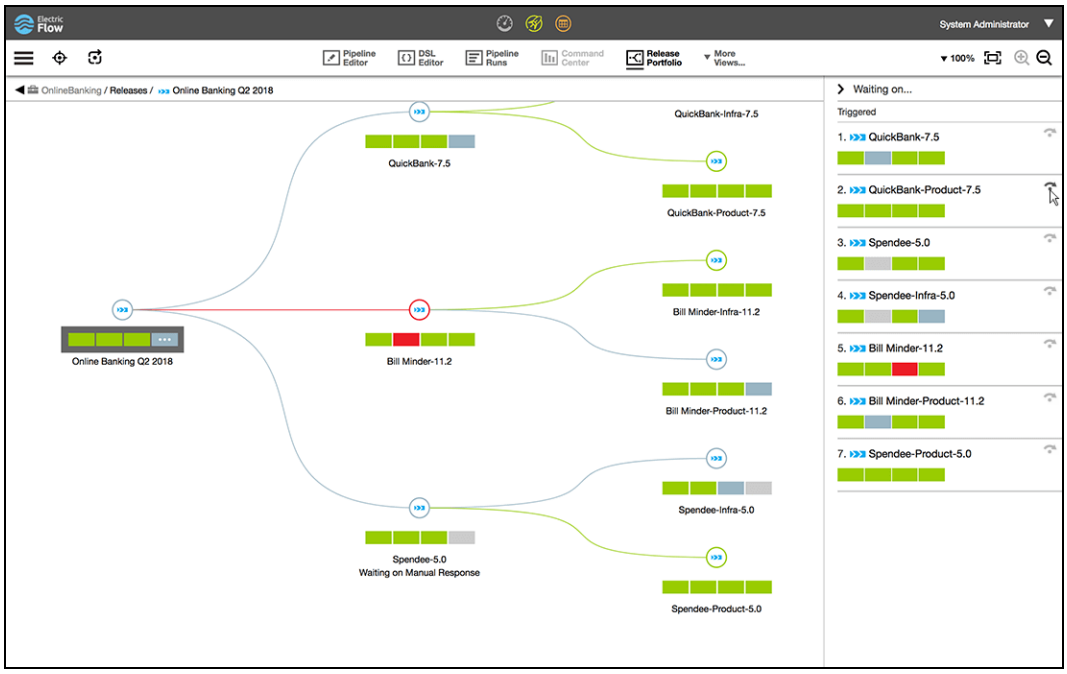
You can go to a parent pipeline portfolio view (navigating up and down from the portfolio). To do so, go to a main pipeline runtime page, then click **Pipeline Portfolio**, and then click the "description" icon for

a "child" pipeline. This displays information about when a PRE/POST gate is waiting for a dependency or manual action.

Release Portfolio View

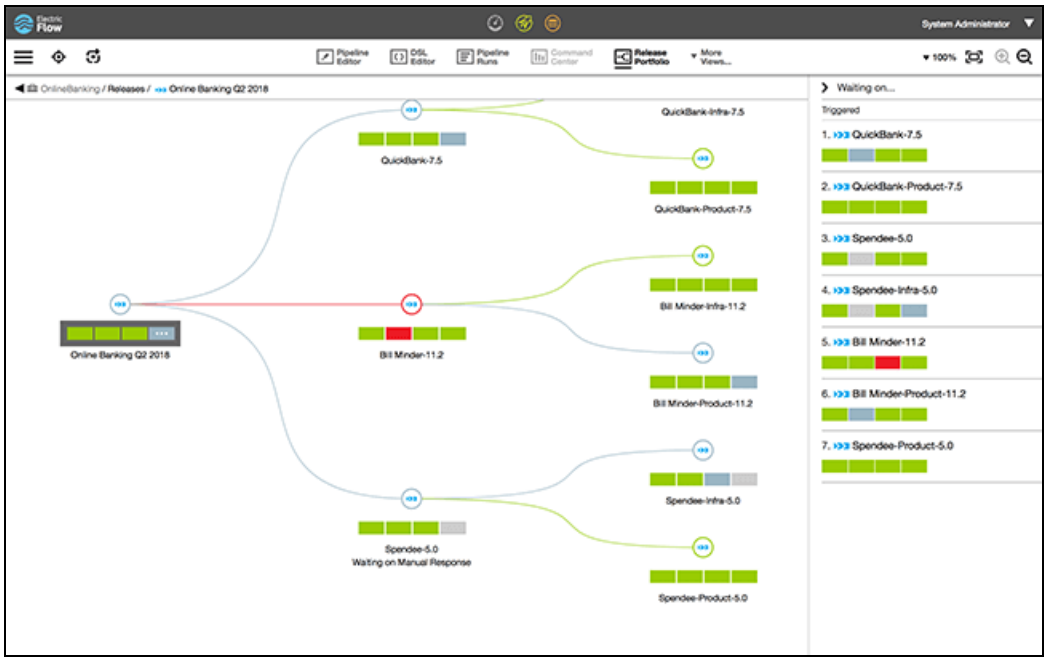
The Release Portfolio View provides a top-level view of the entire hierarchy of releases. As you are authoring, this view provides a clear understanding of the connections and dependencies between releases down to specific stages in each release pipeline.

To view the Portfolio View for a release, click the (Release Portfolio) button at the top of the screen. The following screenshot provides an example of hierarchical releases with a master release named Online Banking Q2 2018 and its Release Portfolio View at runtime:

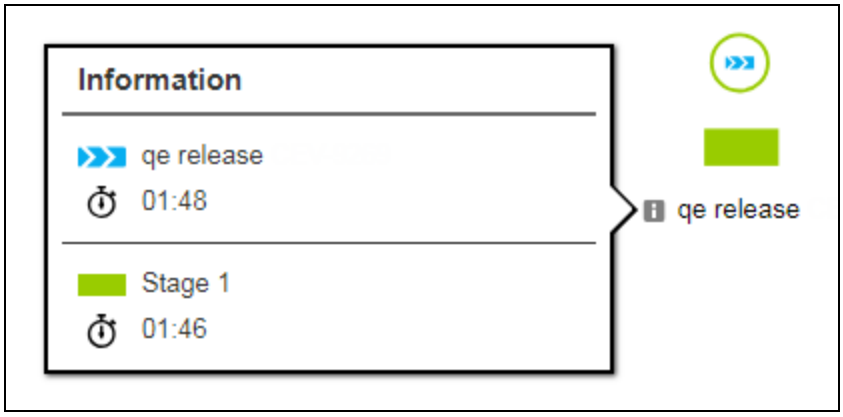


At runtime, as releases are in progress, this top-down view gives detailed insight into the status of the release hierarchy with live status on each pipeline down to the stage level. This shows whether the stage is actively running, completed successfully, or has failed. This visibility down to stage-level status provides detailed information for each release from the Portfolio View, so you do not need to open each release to check for progress through the stages.

You can drill down into dependency details to identify delays. Stages waiting for dependencies are highlighted, and you can drill down into the details to see a list of dependencies that the release is waiting for. This visibility lets you identify inefficiencies or process bottlenecks and take appropriate action:



You can view duration information by clicking the (information) button, which opens a popover that displays the duration times. For example:

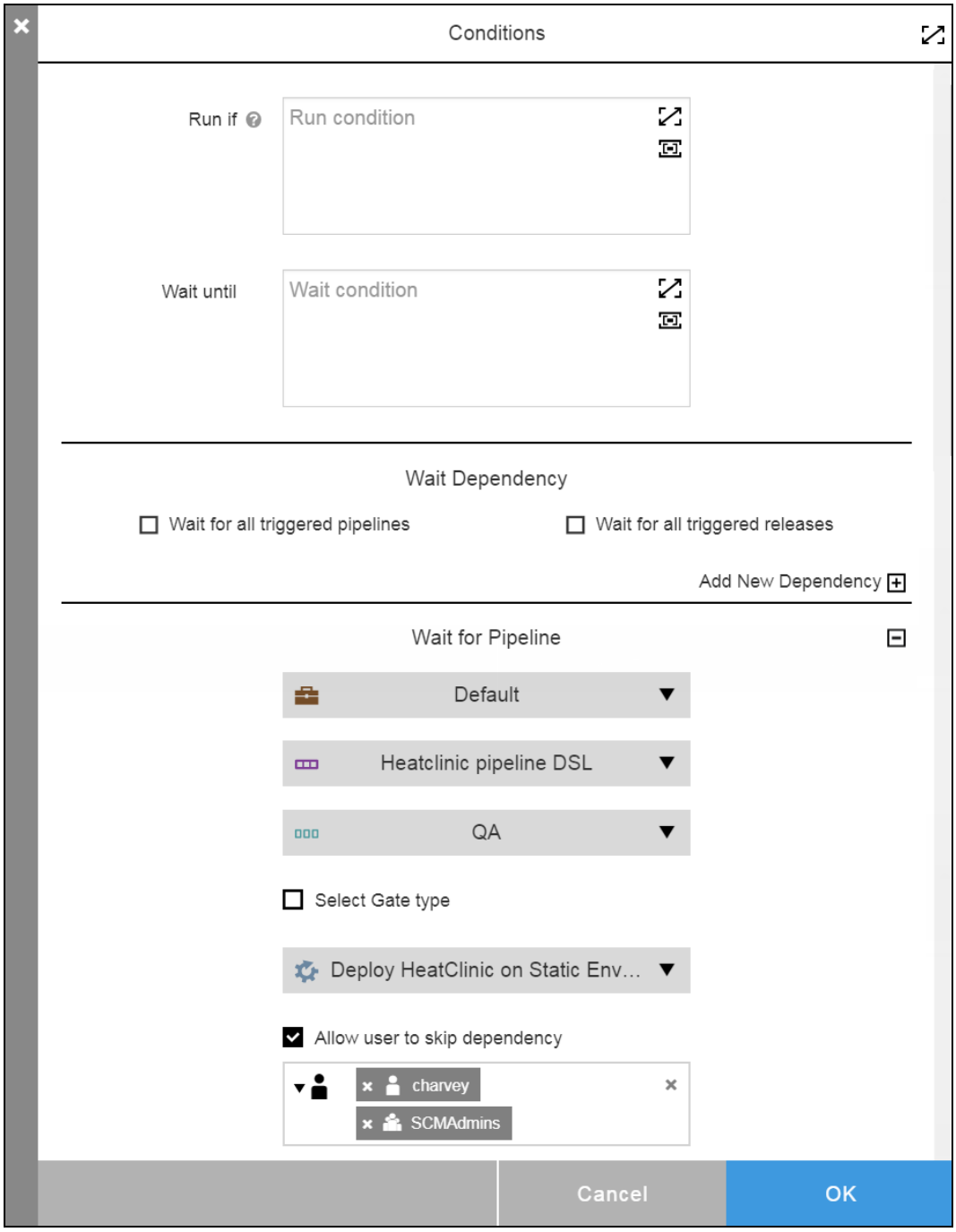


Dependencies and Dependency "Skipping Wait"

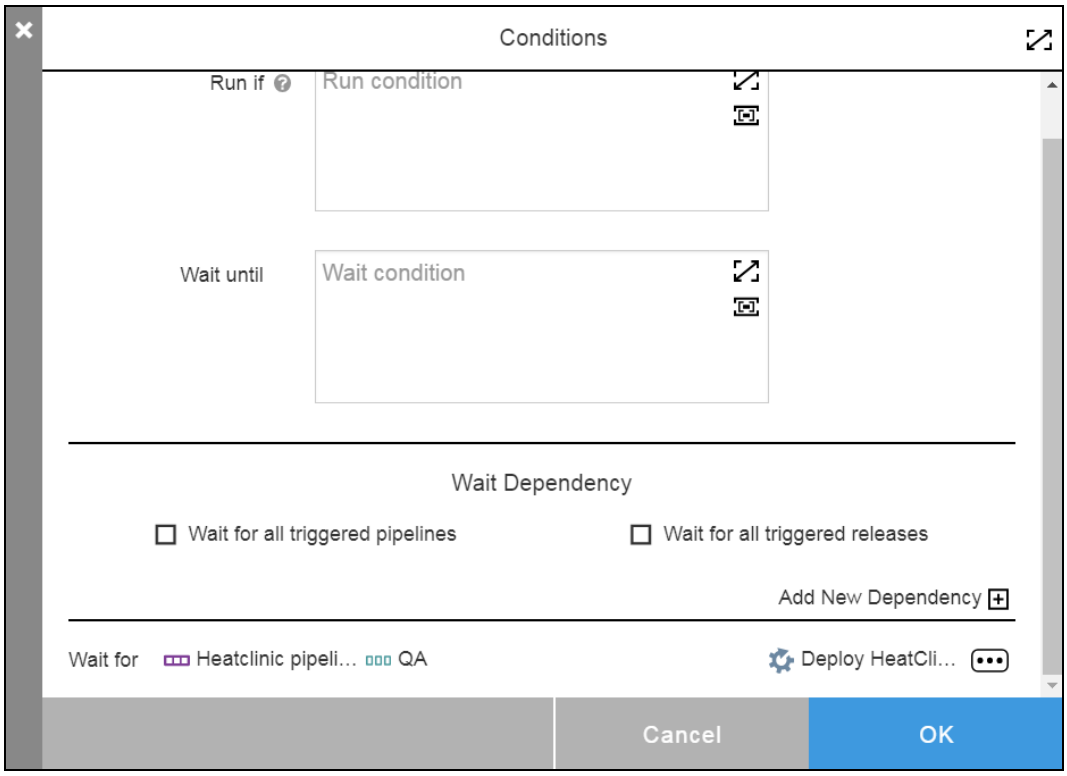
You can set dependencies on any stage or task in a pipeline or release to wait for subreleases to complete. Invoking a parent pipeline or release will continue only when the wait dependency is satisfied (such as when a stage, a task, or the subpipeline itself completes). These wait dependencies can be set to wait for entire subpipelines or releases to complete or to wait for specific stages or gates. At runtime, all dependencies are automatically enforced to reduce the need for manual intervention or checks.

When defining pipeline or release wait dependencies, you can configure whether skipping is allowed and the users who are allowed to skip. This lets you make your hierarchy more secure during runtime.

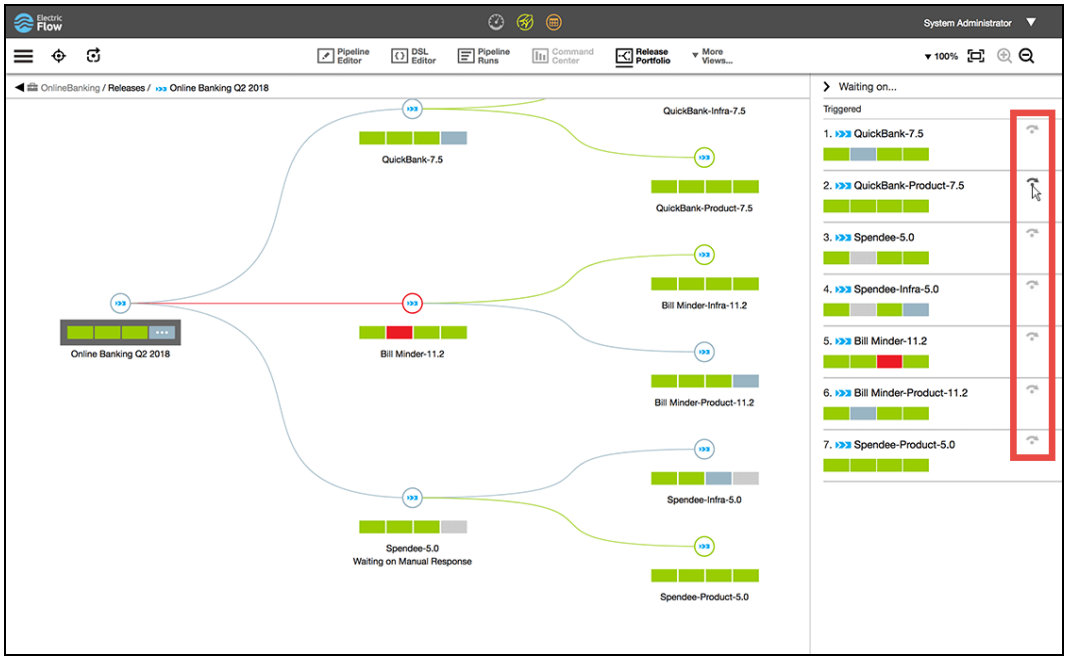
Following is an example of how to configure wait skipping:



Following is an example that shows a configured "wait skip:"

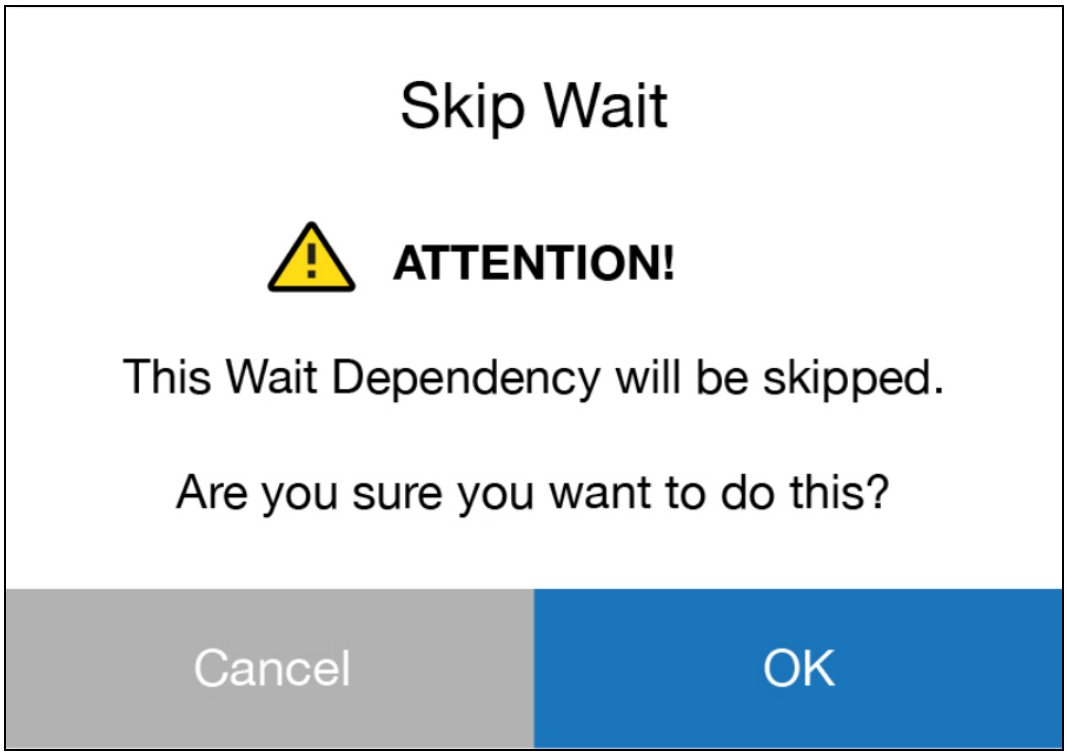


The following figure shows examples of releases that are configured that allow skipping and the buttons used to request a skip:



These buttons are visible only to the users who are allowed to skip the wait dependencies.

You are asked for confirmation after you click a "skip wait" button:



Usage Scenarios for Hierarchical Releases and Pipelines

The following scenarios illustrate several benefits of using this feature:

- Several release pipelines depend on each other at different stages. These release pipelines deploy into shared environments, and some stages might provision environments used by the release or other releases. If a stage misses its deadline, it can cause a cascading effect to delay other releases. Hierarchical releases provide a high-level view into all pipelines to assess risk in the release and bottlenecks or delays.

  A specific example is a project that contains three pipelines. Pipeline A spawns pipeline B in STAGE 1 and continues to run, and STAGE 3 depends on pipeline B completing, so the paths merge. Pipeline B spawns Pipeline C at STAGE 1 and continues to run, and STAGE 3 waits for Pipeline C to complete before continuing. In this way at STAGE 3 of Pipeline A, all three pipelines merge to complete the release.

- A release contains three separate independent applications, and each pipeline is triggered autonomously by its team. For the release to complete, these independent pipelines must converge when completed.

Release Planning, Scheduling, and Tracking

This functionality allows release pipelines to capture planning data at the stage and task level. It also provides views that help you analyze the release status. This functionality simplifies release creation for first-time users to onboard faster.

With the overall set of capabilities in this functionality, you can:

- Create and access a pipeline from within a release. Pipelines in a release are local to the release. (Pipelines that belong to releases do not appear in the pipelines list)
- Capture or view planning data. You can capture the planned start and end date at the stage/gate and task level
- Capture planning data at authoring (this supports editing dates during runtime)
- Control release execution
- Use the following views to analyze a release:
    - Planned versus actual release progress views. You can compare the planned and actual dates to get the release status
    - Calendar view to view all the releases on a calendar (definition view)

The release object contains these features:

- Ability to add start and end dates on tasks and stages
- Statistics such as lead time and process time for each stage
- Ability to complete a stage (automatically or manually)
- Release calendar across all releases with environment reservation

Release and Environment Reservations Calendar

The Release Calendar provides a top-down perspective of all releases and environment reservations in one view. This calendar lets you view the schedule for releases that are planned for one or more projects so you can plan accordingly to avoid conflicts. This calendar also shows the schedule for releases that are in progress or have completed as well as environment reservations with details (including blackout periods). You can switch between the monthly, weekly, or daily views and show all pipelines scheduled within the selected time period.

For organizations managing multiple releases across multiple teams and shared environments at the same time, it is critical to have centralized visibility to identify schedule and resource conflicts. The Release Calendar gives a top down perspective of all releases and environment reservations in one view. With this calendar view you can:

- Track completed, in-progress, and planned releases in one view
- Visualize delivery timelines, environment availability, and blackout periods together to zero in on conflicts

Usage Scenarios

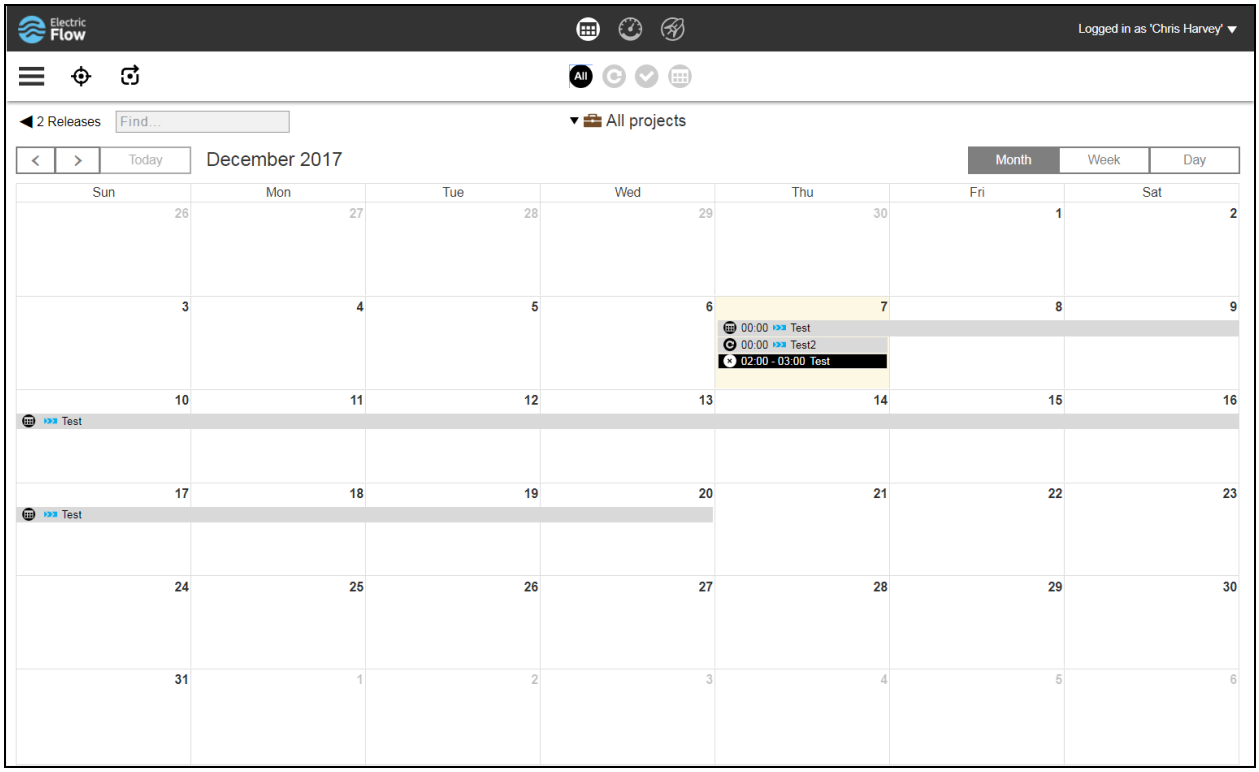
A Release has two stages, QA and PROD.

- The QA stage has the window June 1-15. The PROD stage has the time window June 10- June 30th.
- For the QA stage, once the stage is entered, run the stage tasks daily at 3 P.M. while in the window for the stage.
- For the PROD stage, do not allow entry to the stage before June 10th. Once entering the stage, run the deployment once.
- If the release pipeline is started before June 10th, the first stage will not be entered until the entry date is met.

- If the release pipeline is started after June 15th, the DEV stage will be skipped, and the pipeline will start in PROD.

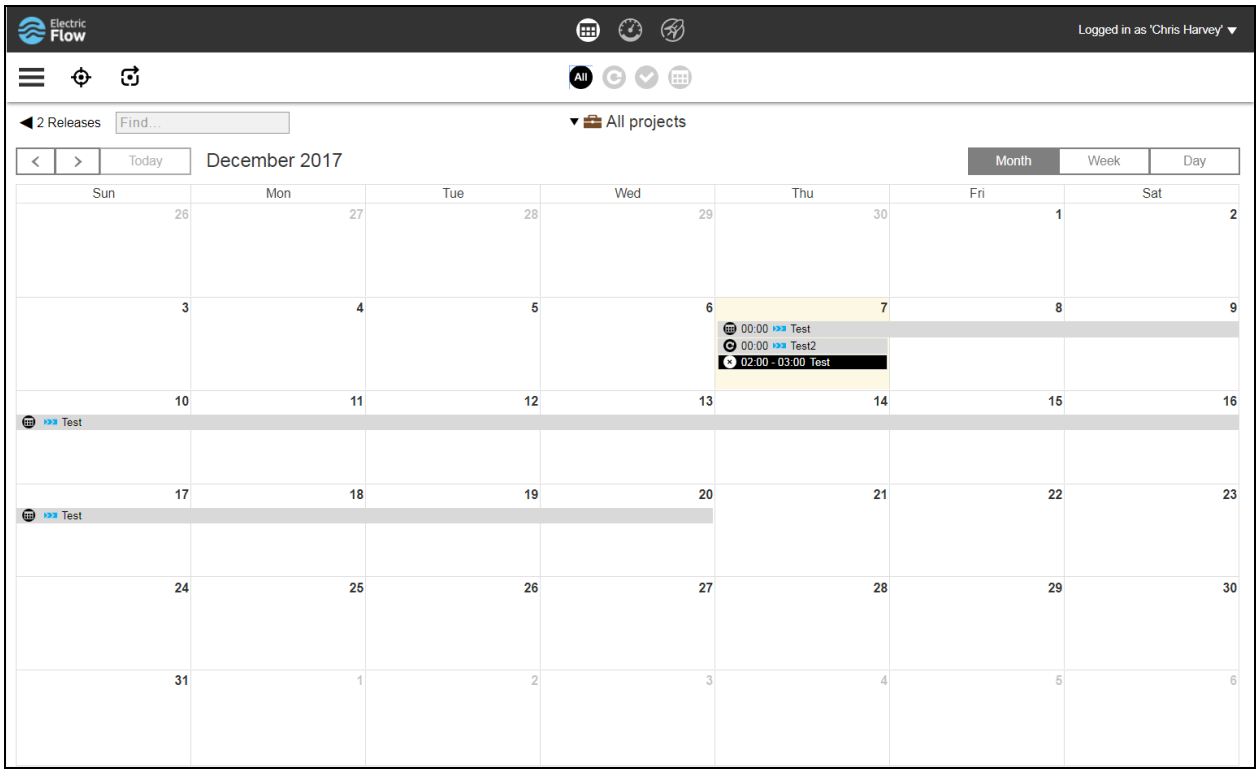
To open the release calendar, click the (Release Calendar) button at the top of the page.

Following is a release calendar with two releases (a running release and a release in planning) and an environment blackout period.



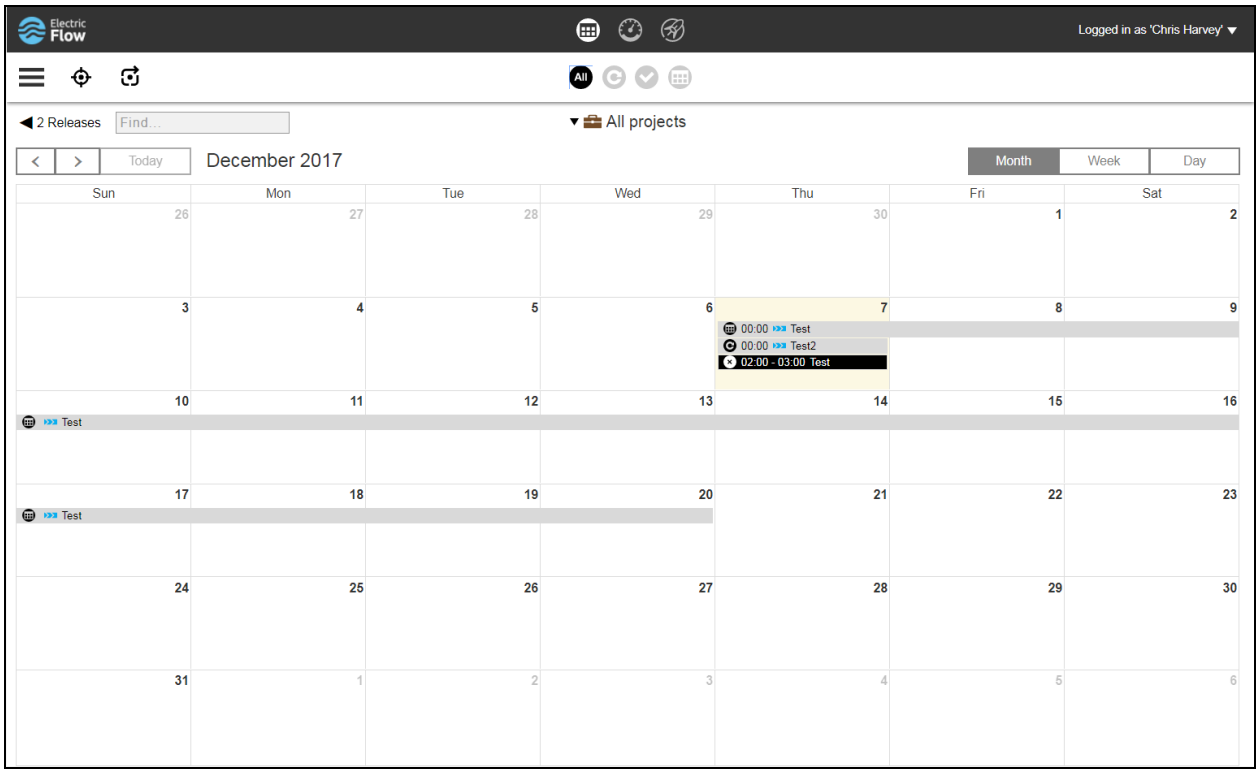
You can click a release or environment reservation or a blackout period to view a details popup. Blackout periods might represent company holidays or other significant dates.

For example, you can click a release to see the start and end dates and times:



Note that a warning might appear if you have not yet set your time zone.

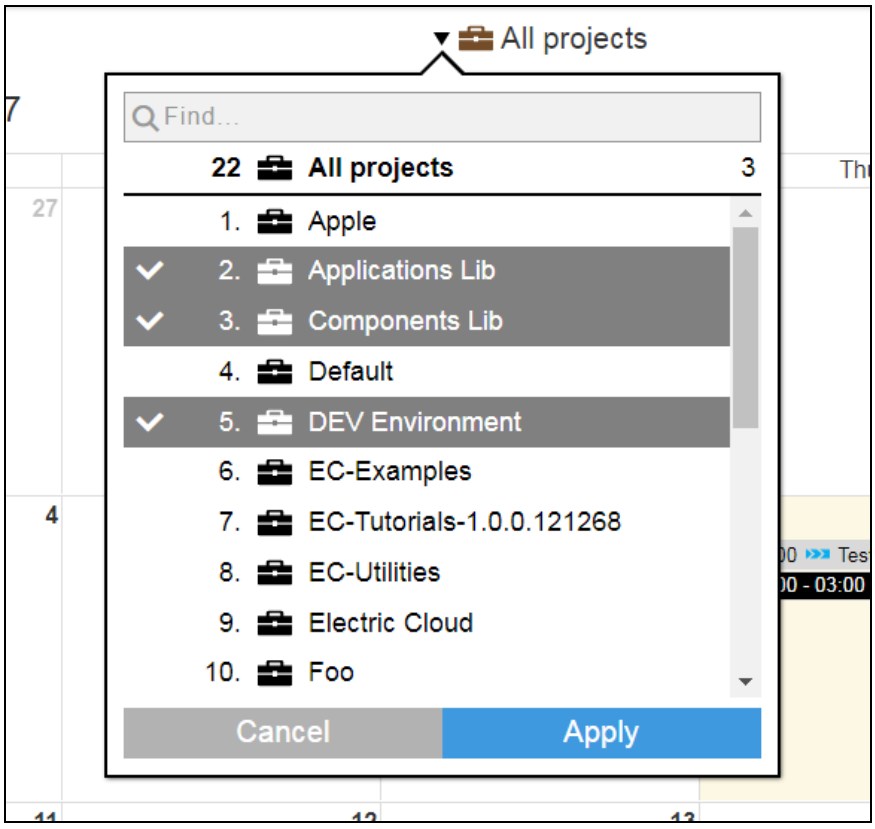
Also, for example, you can click an environment blackout to see the start and end dates and times:



You can click **Month**, **Week**, and **Day** buttons to toggle among those views.

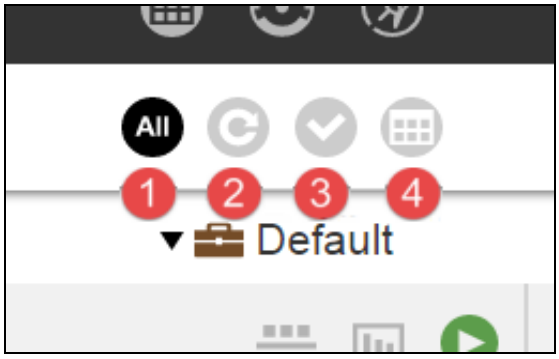
Filtering Releases by Project

By default, the release calendar displays releases from all projects. You can choose one or more specific projects from the projects menu. The following example shows three projects selected:



Filtering Releases by Status

By default, the release calendar displays all releases regardless of their statuses such as running releases or releases in planning. You can filter the list of releases according to status by using the following buttons:

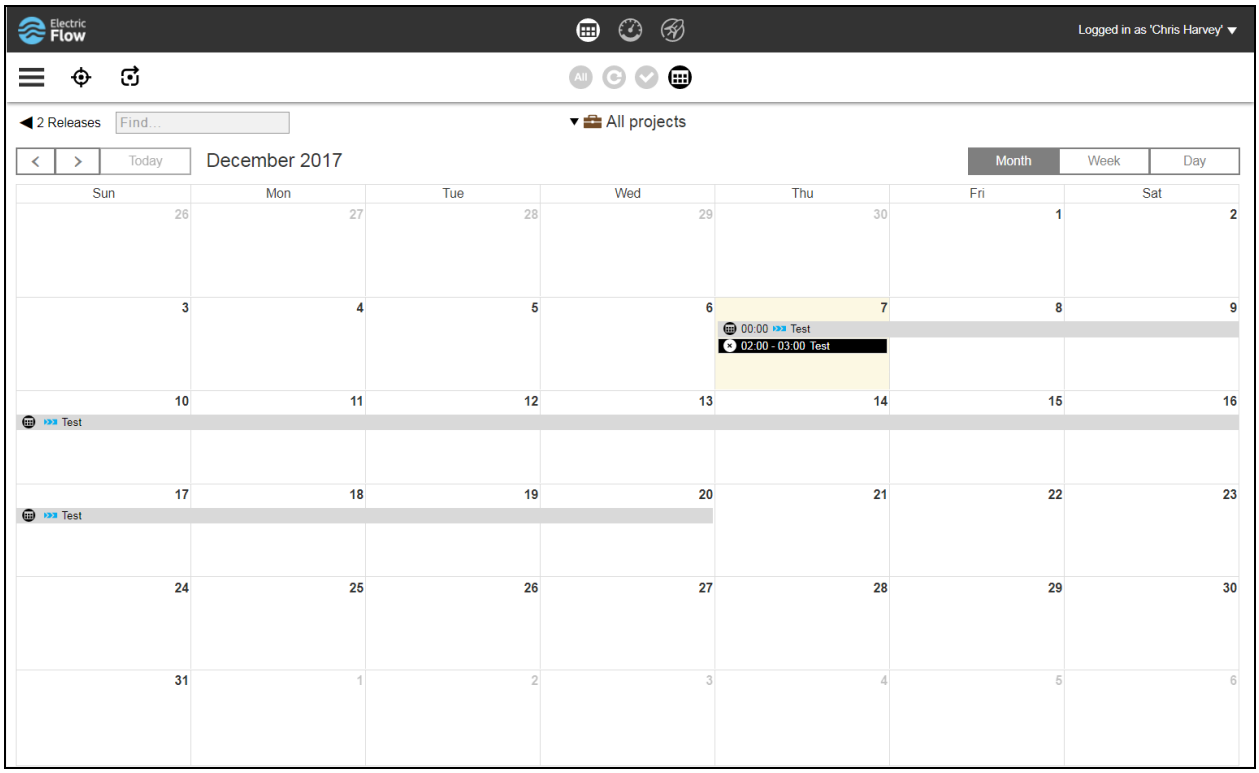


Following are descriptions of the filters:

1	All releases

2	Running releases. A running release is one that is in progress, which means that its pipeline is still running. For details about running and ending (completing) a release, see Running and Completing Releases on page 683.
3	Successful releases. A successful release is one with a pipeline that ran with no errors that you have subsequently completed. For details about running and ending (completing) a release, see Running and Completing Releases on page 683.
4	Releases in planning. A release in planning is one whose release run has not started.

For example, you can view only the releases in planning by clicking the  (Planning) filter button:



To view the details of a release, click the release name, which opens the **Applications / View Run** page. For more information, see Deploying and Troubleshooting Applications on page 287.

Visibility and Status of Release Pipelines

From the Release Dashboard

The Release Dashboard orchestrates all the information about the planned, active, or completed releases and displays the releases that you have permission to view in one place. At any time, you can

quickly see the current status of the release, where it is in the process flow, any errors during the runtime, when human intervention is required to take some action or perform a task, approvals, and the start and end dates for the release. See Release Dashboard on page 669 for more information, and See Example: Visibility in a Release Pipeline from the Release Dashboard on page 640 for an example how to drill down from the top Pipeline Run view to the specific job details of the release tasks.

For a release that is in progress or has been completed, clicking the Release status opens the Pipeline Run view. You can view the Stage Summary for the latest release or view previous pipeline runs. The Stage Summary shows the pipeline's progress through each stage and evidence that the tasks were executed. Clicking in a stage task or a gate displays its Stage Summary. See Pipeline Stage Summary on page 603 for more information.

- When the Release is in progress, you can view the progress of the pipeline as it runs. See Viewing the Stage Summary During a Pipeline Run on page 604 for an example of this.
- You can also view a completed pipeline run. See Viewing the Stage Summary for a Completed Pipeline Run on page 608 for an example of how to do this.
- Clicking **View previous Pipeline Runs** displays the last five pipeline runs. After selecting one of these runs, you can view the Stage Summary to view more details about the pipeline run.
- If the pipeline has errors, you can also view what the error is and where it occurred in the pipeline in the Task Error view.

For example, you can view the progress and evidence of the deployer task run. Clicking displays the payload of the release. This is the actual list of applications or microservices (and the number of applications or microservices) to deploy that is specified in the bill of materials in the release definition.

From the Release dashboard, you can go to the Path-to-Production view for a running or completed release. This view shows the bill of materials for the release. You can easily see the artifacts and snapshots that are in compliance with the release manifest. See Path-to-Production View on page 680 for more information.

From the Environment Inventory

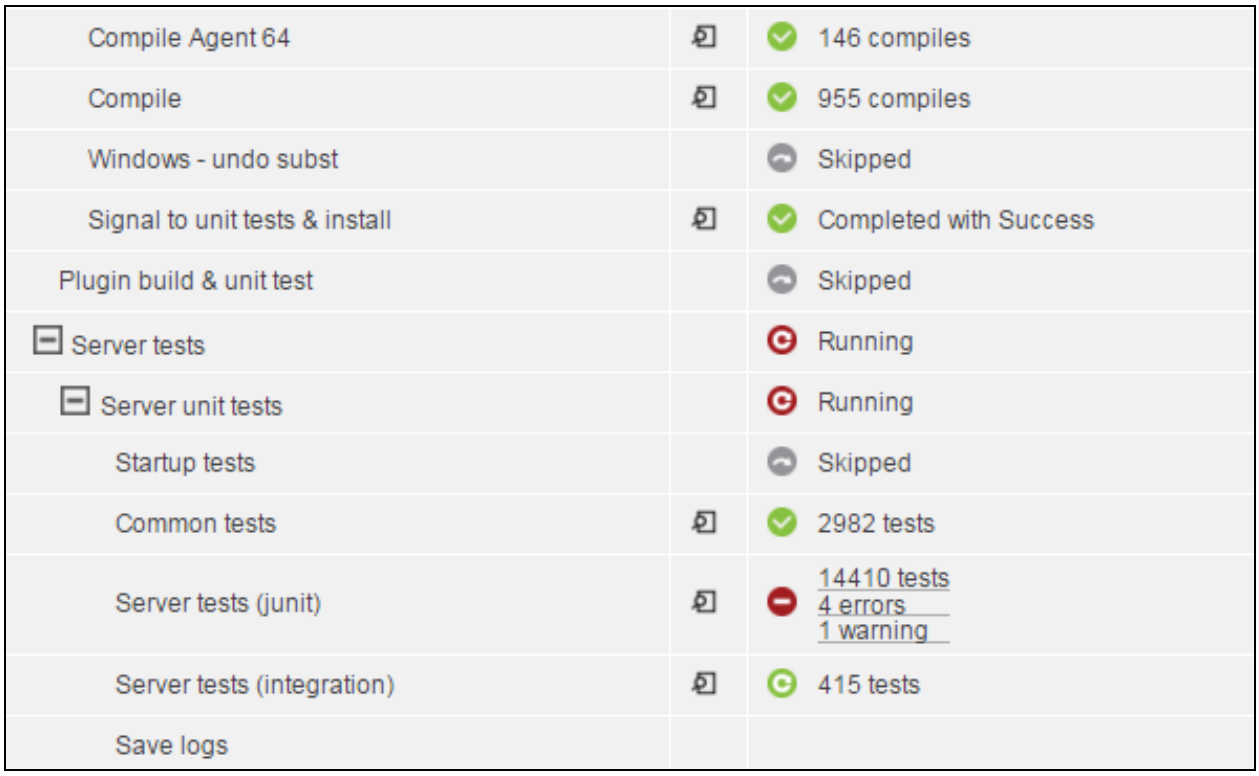
The environment inventory shows what is deployed in an environment at any time. See Environment Inventory on page 131 for more information.

From the Change History

The Change Tracking feature tracks the changes between every state of non-runtime objects, such as environments and resources. ElectricFlow records a *Change History* of the historical states of environment objects, including environment tiers and resources, and records changes between them. See Change Tracking on page 1374 for more information.

Postp

Postp, a postprocessor included with ElectricFlow, is used for data collection and for report generation. For visibility in a Release run, you can get live streaming data from the log files using the standard output settings. Using postp commands in pipeline tasks (defined by application or microservice processes, procedures, or workflows), you can make this data appear in the Job Details page in ElectricFlow.



This example shows:

- During the Compile step, there were 955 compiles.
- In the "Command tests" step, 2982 tests were run.
- For the "Server tests (junit)" step, 14410 tests were run with 4 errors and 1 warning. Clicking on the links in this step provides details and evidence about how the job ran and what actions to take next.

Clicking on the **log** button for any of these steps will display the log file details.

For more information about using postp:

- See Defining the Tasks in a Pipeline Stage Using the Pipeline Stage View on page 468 and Editing a Pipeline Definition on page 516 to define, view, and edit pipeline tasks in the pipeline definition.
- See Postprocessors: Collecting Data for Reports on page 1506 and Postp extension on page 1604 to set up data collection, generate reports, and display data from the log files.

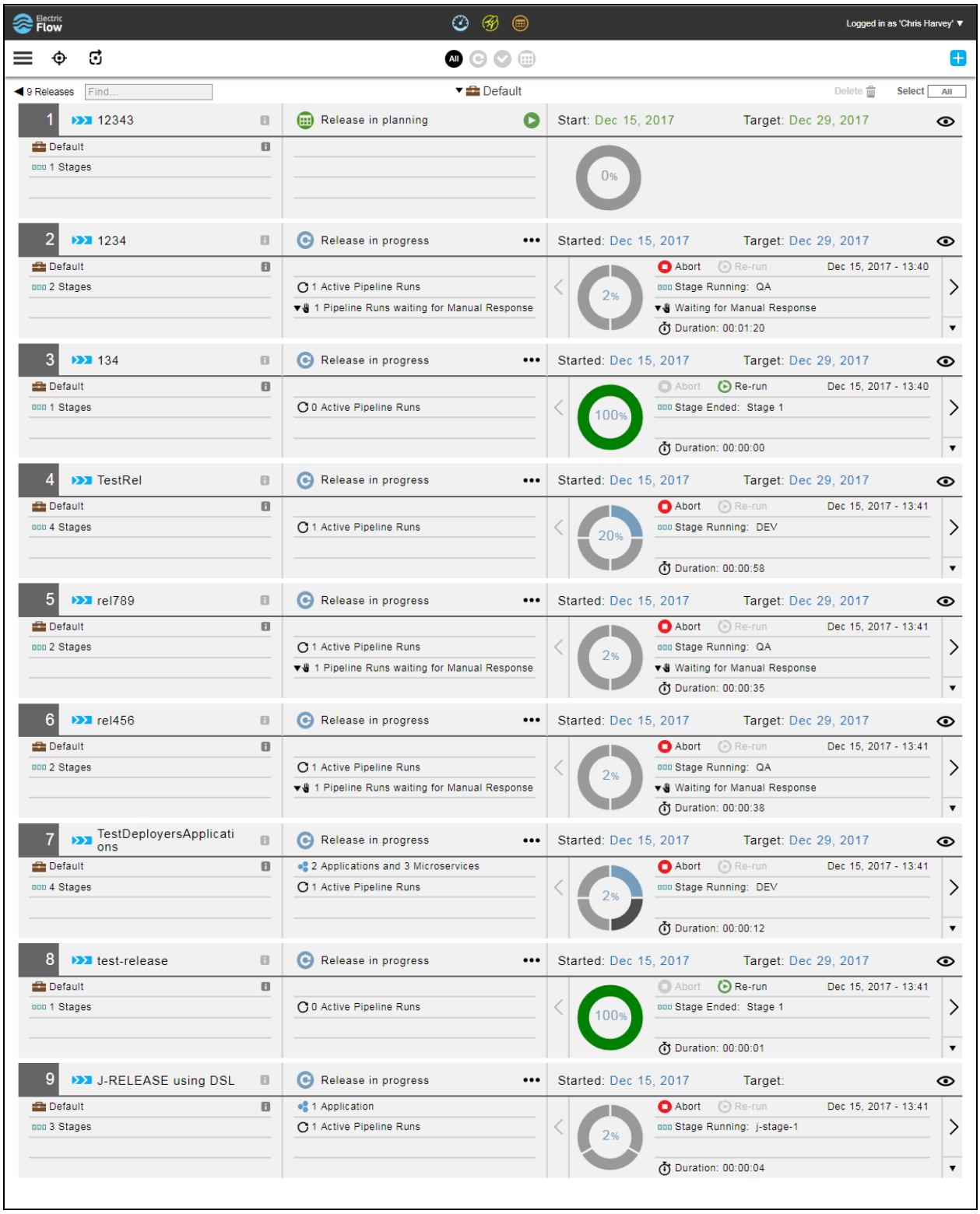
Example: Visibility in a Release Pipeline from the Release Dashboard

For an In-Progress Release

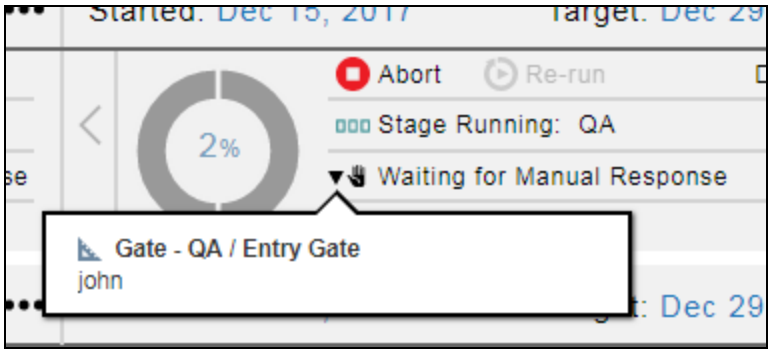
You can view the progress of the pipeline during the Release run starting in the Release Dashboard:

- In the Release dashboard, clicking on the status of an in-progress or completed Release opens the Pipeline Run view for the Release pipeline.

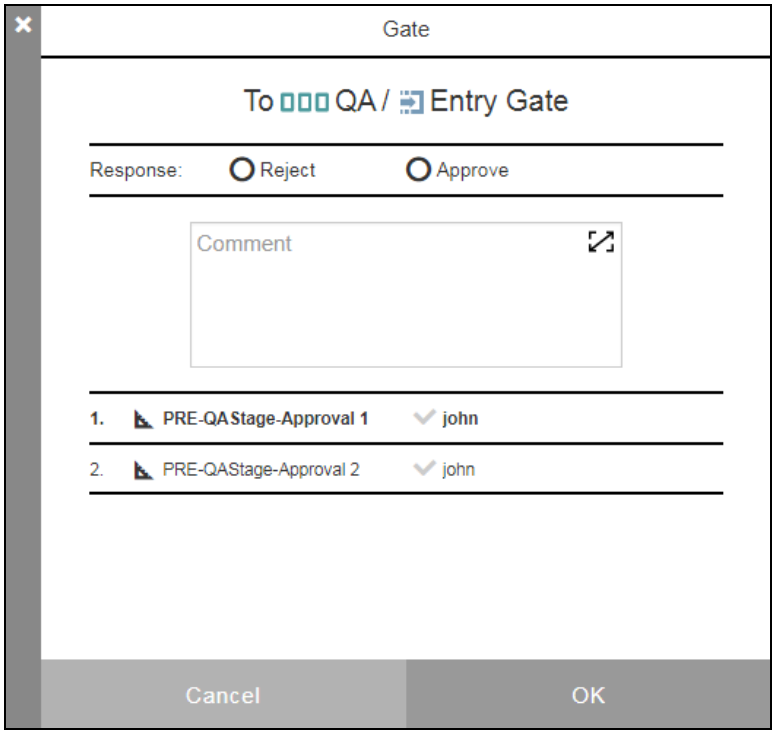
Example: Clicking **Release in progress** for the second pipeline shows the progress of a currently running Release.


Electric Flow
Logged in as 'Chris Harvey'
9 Releases
Find...
Default
Delete
Select
All
1 12343
Release in planning
Start: Dec 15, 2017
Target: Dec 29, 2017
Default
1 Stages
0%
2 1234
Release in progress
Started: Dec 15, 2017
Target: Dec 29, 2017
Default
2 Stages
1 Active Pipeline Runs
1 Pipeline Runs waiting for Manual Response
2%
Abort
Re-run
Dec 15, 2017 - 13:40
Stage Running: QA
Waiting for Manual Response
Duration: 00:01:20
3 134
Release in progress
Started: Dec 15, 2017
Target: Dec 29, 2017
Default
1 Stages
0 Active Pipeline Runs
100%
Abort
Re-run
Dec 15, 2017 - 13:40
Stage Ended: Stage 1
Duration: 00:00:00
4 TestRel
Release in progress
Started: Dec 15, 2017
Target: Dec 29, 2017
Default
4 Stages
1 Active Pipeline Runs
20%
Abort
Re-run
Dec 15, 2017 - 13:41
Stage Running: DEV
Duration: 00:00:58
5 rel789
Release in progress
Started: Dec 15, 2017
Target: Dec 29, 2017
Default
2 Stages
1 Active Pipeline Runs
1 Pipeline Runs waiting for Manual Response
2%
Abort
Re-run
Dec 15, 2017 - 13:41
Stage Running: QA
Waiting for Manual Response
Duration: 00:00:35
6 rel456
Release in progress
Started: Dec 15, 2017
Target: Dec 29, 2017
Default
2 Stages
1 Active Pipeline Runs
1 Pipeline Runs waiting for Manual Response
2%
Abort
Re-run
Dec 15, 2017 - 13:41
Stage Running: QA
Waiting for Manual Response
Duration: 00:00:38
7 TestDeployersApplications
Release in progress
Started: Dec 15, 2017
Target: Dec 29, 2017
Default
4 Stages
2 Applications and 3 Microservices
1 Active Pipeline Runs
2%
Abort
Re-run
Dec 15, 2017 - 13:41
Stage Running: DEV
Duration: 00:00:12
8 test-release
Release in progress
Started: Dec 15, 2017
Target: Dec 29, 2017
Default
1 Stages
0 Active Pipeline Runs
100%
Abort
Re-run
Dec 15, 2017 - 13:41
Stage Ended: Stage 1
Duration: 00:00:01
9 J-RELEASE using DSL
Release in progress
Started: Dec 15, 2017
Target:
Default
3 Stages
1 Application
1 Active Pipeline Runs
2%
Abort
Re-run
Dec 15, 2017 - 13:41
Stage Running: j-stage-1
Duration: 00:00:04

- When human intervention is needed at a gate (indicated by the  hand menu), clicking the menu displays the required approvals.

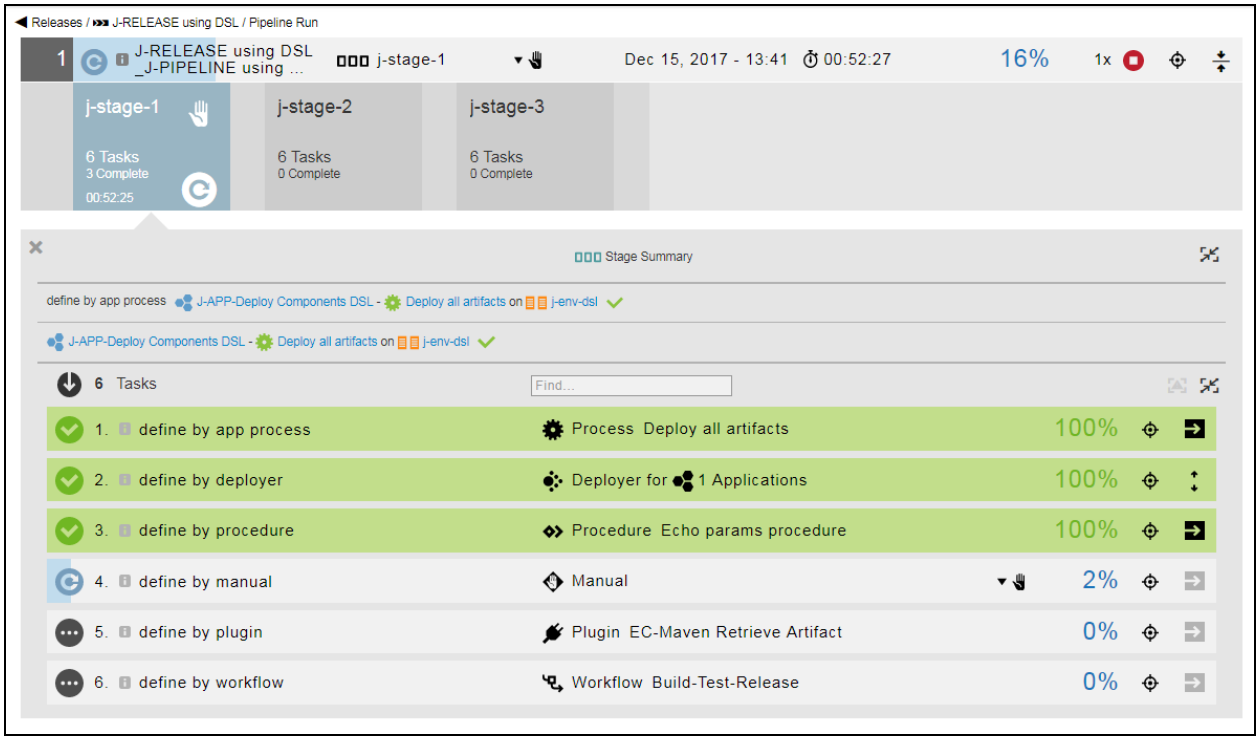


Clicking the specific approval (john in this case) opens the gate approval dialog.



After the approval rules or conditions are approved, the pipeline continues.

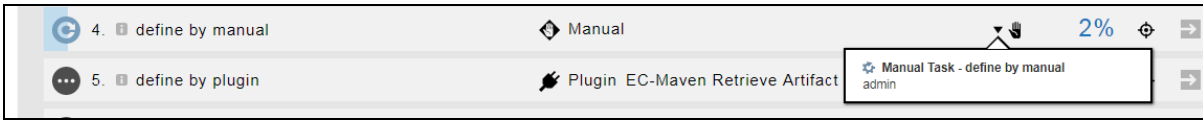
- When a pipeline stage is running, it is enabled. Clicking in it displays the Stage Summary, which shows the tasks in the stage and their progress.



1. In the Stage Summary, the top section shows information about the tasks deployed in the pipeline stage, which you can show or hide. You can also display user-generated information such as:
    - Links to web sites for reports and other information about the deployed artifacts
    - Property information about the pipeline stage and the objects in it
2. The next section shows the tasks in the pipeline stage. If the pipeline is in progress,

    click in the row to view details for the manual task:



3. After completing the manual task, click **Complete** as the outcome, add a comment, and click **OK** in the **Manual** dialog box.

To view details about a completed manual task, click to view the details. The

Manual dialog box opens. When the manual task has required input, click in the Parameters row to view the manual task details. The **Manual** dialog box now shows the **action** and **evidence** fields. The **action** field describes what was done to approve the manual task, and the **evidence** field describes what was done to approve it.

- When human intervention is needed at the exit gate (indicated by the hand icon), an exit gate is enabled. Clicking in the exit gate displays the approvals required to complete the pipeline stage.
- If the entry gate to the next stage has approval conditions before the tasks in it can start, this entry gate is enabled. Clicking in the entry gate displays the approvals required to start the tasks in a pipeline stage.
- The sequence continues as the pipeline progresses.
- Click **View previous Pipeline Runs** to view the previous five runs.

Clicking displays the pipeline, where you can click in a gate or stage to view more details.

Clicking hides this pipeline.

To view details about a manual task, click to view the details of the manual task. The

Manual dialog box opens. When the manual task has required input, click in the Parameters row to view the manual task details. The **Manual** dialog box now shows the **action** and **evidence** fields. The **action** field describes the what was done to approve the manual task, and the **evidence** field describes what was done to approve it.

To view details about the other pipeline tasks, click to view the Job Details.

For a Completed Release

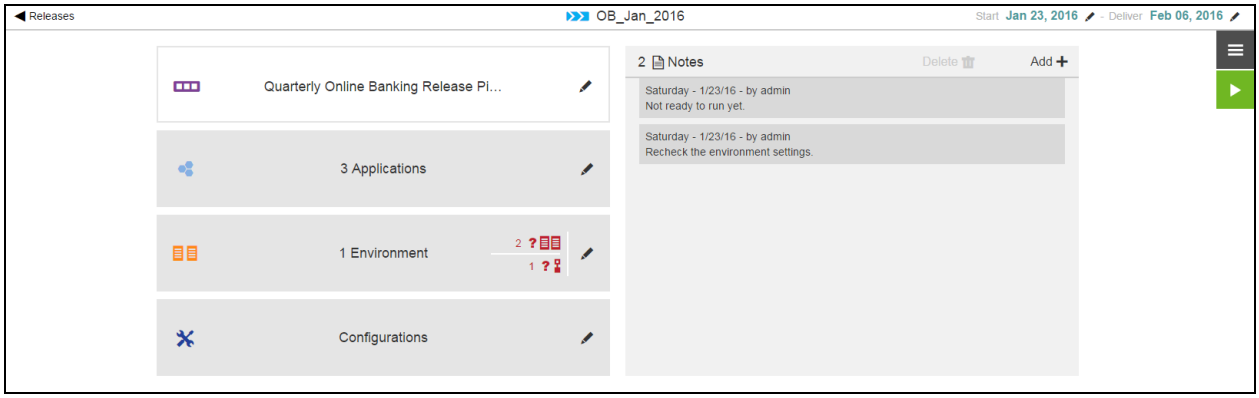
See Running and Completing Releases on page 683 for an example of running and completing a Release.

Release Definition

The release process is defined in the release definition, which consists of the underlying pipeline, the applications to deploy (the payload), where the applications will be deployed (the environments), and any inputs (parameters) to the application (referred to as configurations). The release definition also includes notes that can be entered by team members, task assignees, and approvers as well as planned start and end dates for releases.

This section has examples of release definitions and shows how to define a release.

Example: Completed Release Definition



Example: Traditional Multiple Application Release

This example shows how to define a multiple application release. You can create a new release or copy an existing release.

A copy of a release is the same as the existing release with the following exceptions:

- The status in the copy set to `Release in planning`.
- if the start date of the existing release is in the past, then the start date and end date in the copy are set to the current date and two weeks later respectively.
- The actual start date and actual end date fields in the copy are cleared. This is because the Release is in planning and has not started (and hence there is no end date as well).

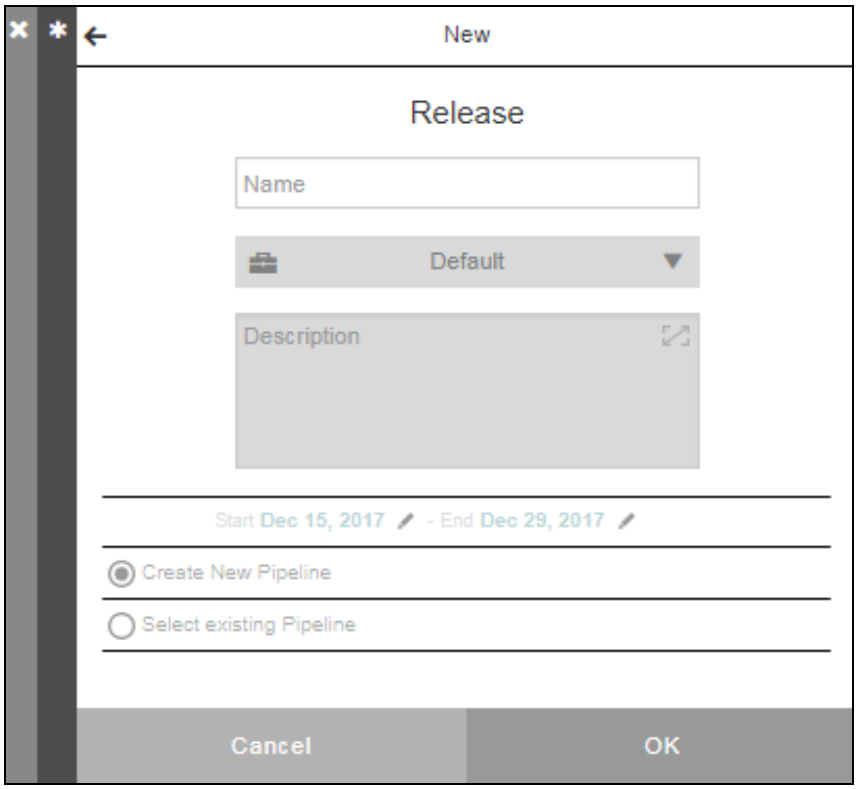
The example in this section shows how to create a new release. To do so:

1. From the Home page (https://<*ElectricFlow_server*>/flow), click **Releases**.
2. In the Release Dashboard, click **Add+** to add a release.

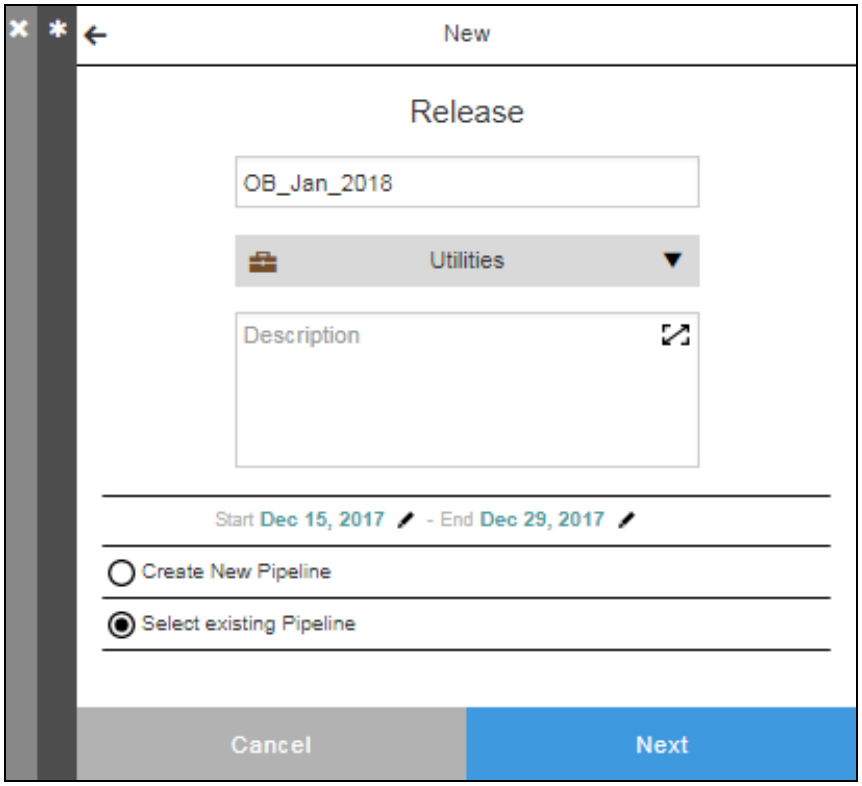
   The **New** dialog box appears:

3. In the **New** dialog box, click **Create new release**.

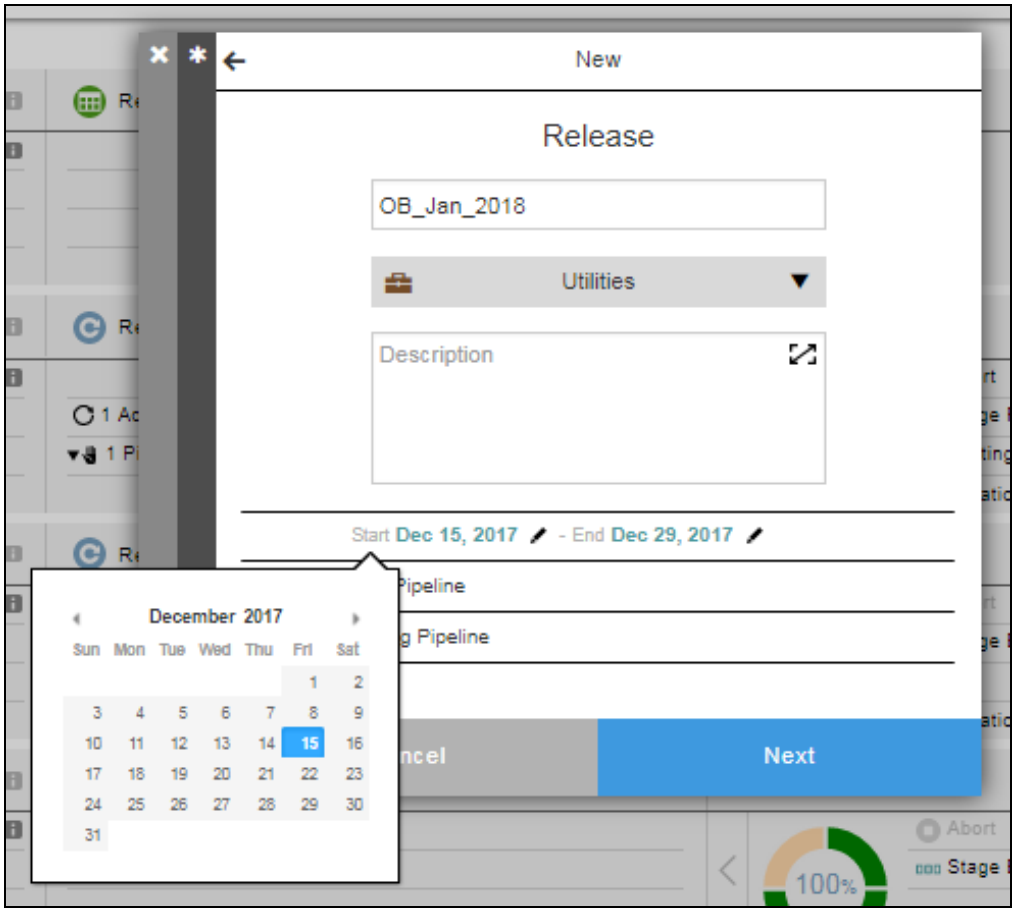
> **Tip:** You can include hyperlinks as part of an object description for any ElectricFlow object.



4. Enter the name of the new release, select a project to which the release belongs, and click **Select existing Pipeline**.

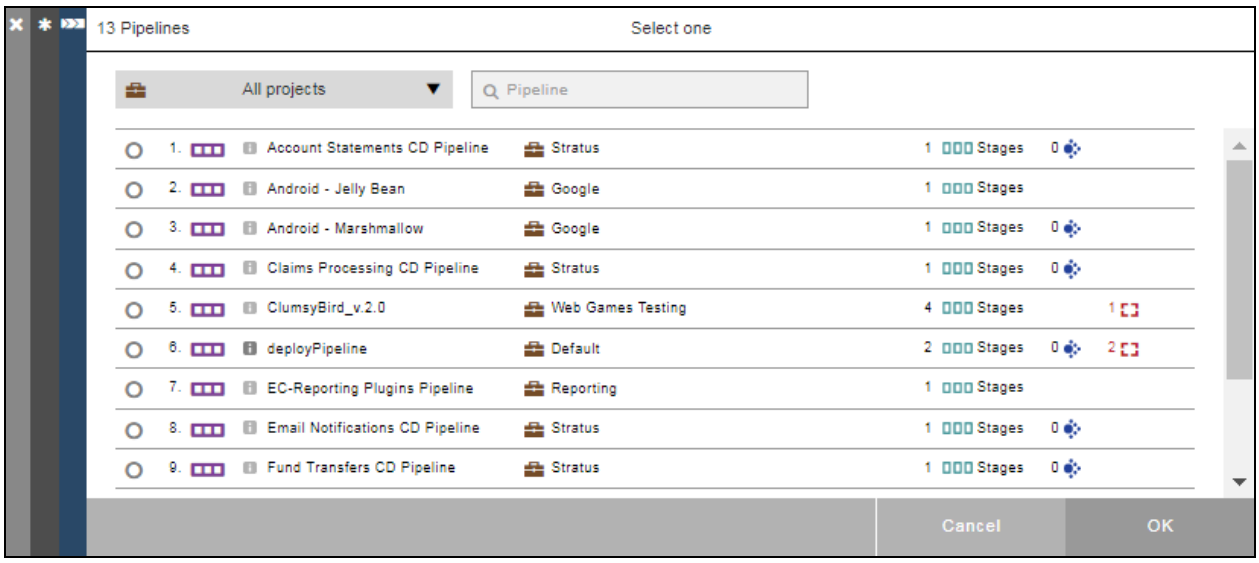


5. (Optional) Select start and end dates by clicking either date and choosing a new date from the calendar popup.



6. Click **Next**.

   The **Select on** dialog box with a list of pipelines appears.



   It shows the list of available pipelines. The default is to show the pipelines for all projects. To see only the pipelines for a specific project, click the down arrow in the **All projects** field to select one or more projects in one of these ways:

   - Click on the name of one or more projects.
   - Enter the search criteria in the Search field. The projects that match the criteria appear in the list.

   You can also search for one or more pipelines by entering the search criteria in the **Search** field next to the **All projects** field. If there are no matches, a message appears stating that there are no pipelines in the selected projects.

After the pipeline is selected, the **Next** button is enabled because the applications in the pipeline

have required inputs (parameters), as indicated by . This means that there are two required inputs to deploy the applications through the pipeline. If any input is a credential parameter, you can enter the path to the credential, browse to it, select the Parameter Credential or Credential binding, or select a user-defined credential that is attached to the project associated with the pipeline.

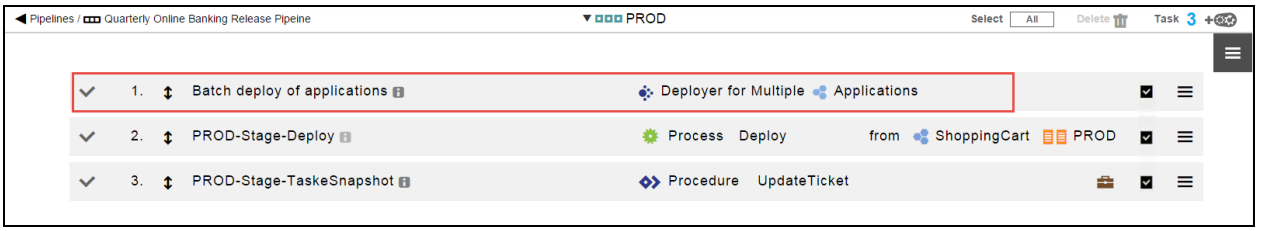
Enter the required inputs, and click **OK**.

If the applications did not have any required inputs, the **OK** button would be enabled instead.

More about this pipeline:

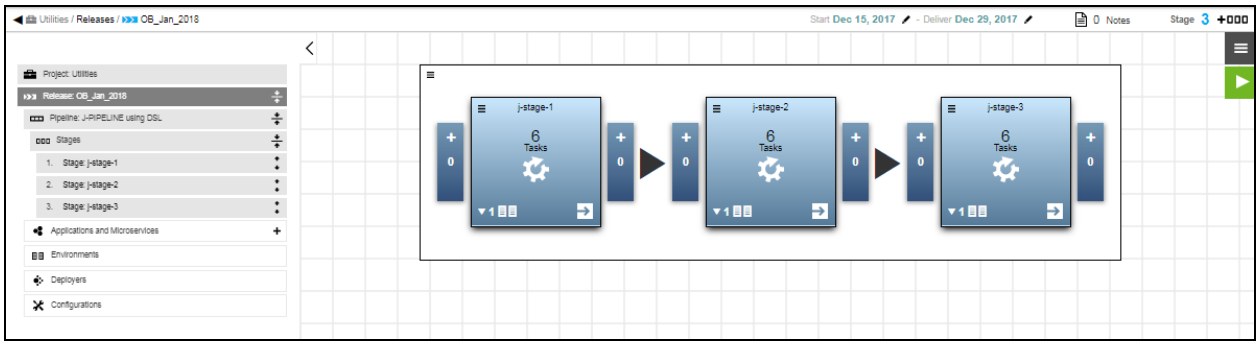
This pipeline has one deployer task, as shown in the task list for the pipeline.

Remember that a deployer task is specified in the task list of a pipeline stage but gets the actual list of applications to deploy from the release payload (the bill of materials of the release definition). At runtime, this list is passed to the pipeline through the release object, and the pipeline deploys the applications in sequential order.



7. Click a pipeline and click **OK**.

   The new release appears. Note that the hierarchy menu on the left side shows the pipeline that you selected.



8. Select a pipeline.

   The pipeline called "Quarterly Online Banking Release Pipeline" is selected.

   After the pipeline is selected, the **Next** button is enabled because the applications in the pipeline have required inputs (parameters), as indicated by 

. This means that there are two required inputs to deploy the applications through the pipeline. If any input is a credential parameter, you can enter the path to the credential, browse to it, select the Parameter Credential or Credential binding, or select a user-defined credential that is attached to the project associated with the pipeline.

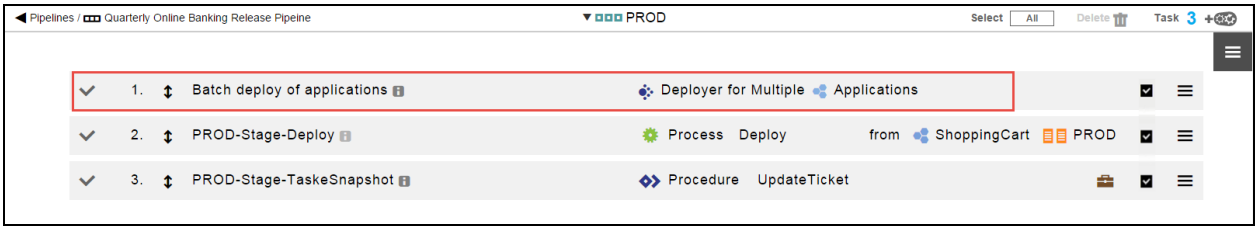
   Enter the required inputs, and click **OK**.

   If the applications did not have any required inputs, the **OK** button would be enabled instead.

   ## More about this pipeline:

   This pipeline has one deployer task, as shown in the task list for the pipeline.

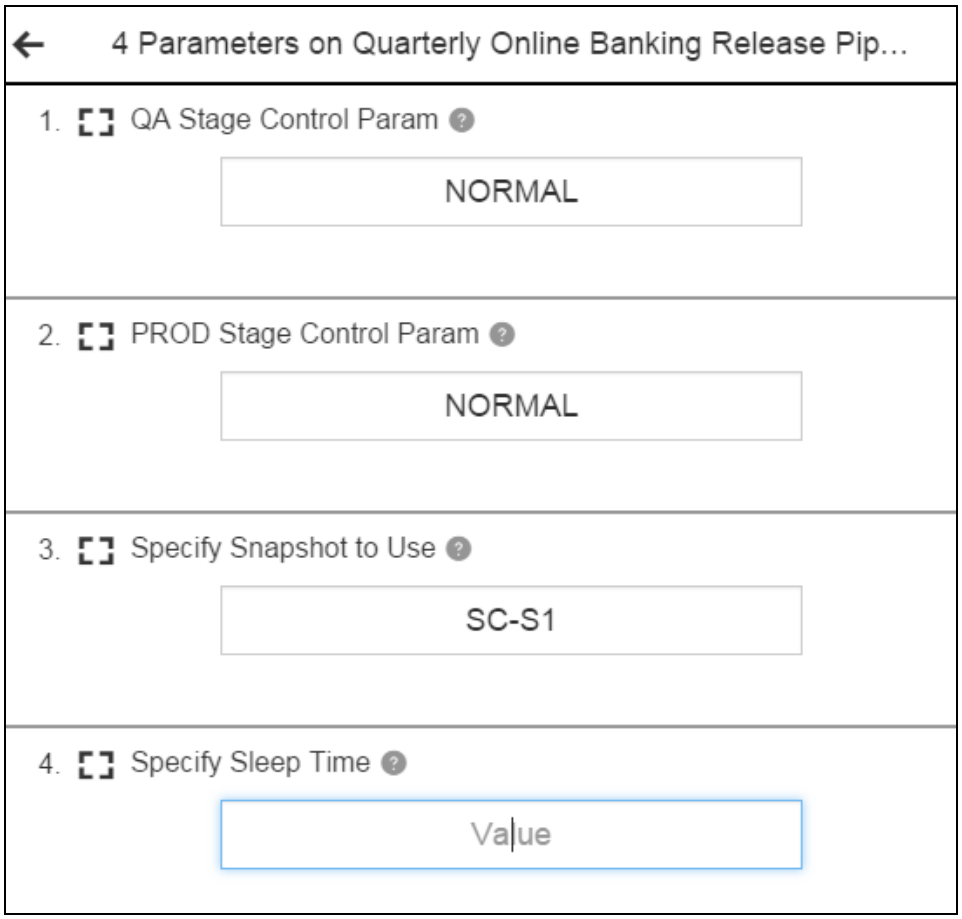
   Remember that a deployer task is specified in the task list of a pipeline stage but gets the actual list of applications to deploy from the release payload (the bill of materials of the release definition). At runtime, this list is passed to the pipeline through the release object, and the pipeline deploys the applications in sequential order.



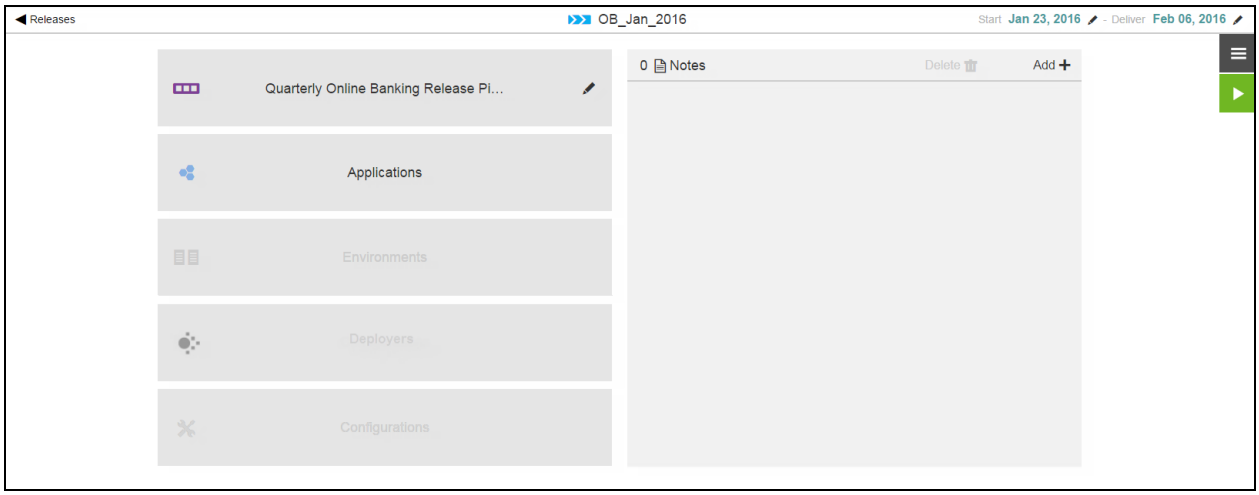
9. If the applications in the selected pipeline have inputs, a dialog box showing the inputs to the pipeline will be displayed.

   If any input is a credential parameter, you can enter the path to the credential, browse to it, select the Parameter Credential or Credential binding, or select a user-defined credential that is attached to the project associated with the pipeline.

   If any of required parameters do not have values, enter the appropriate values and click **OK**.



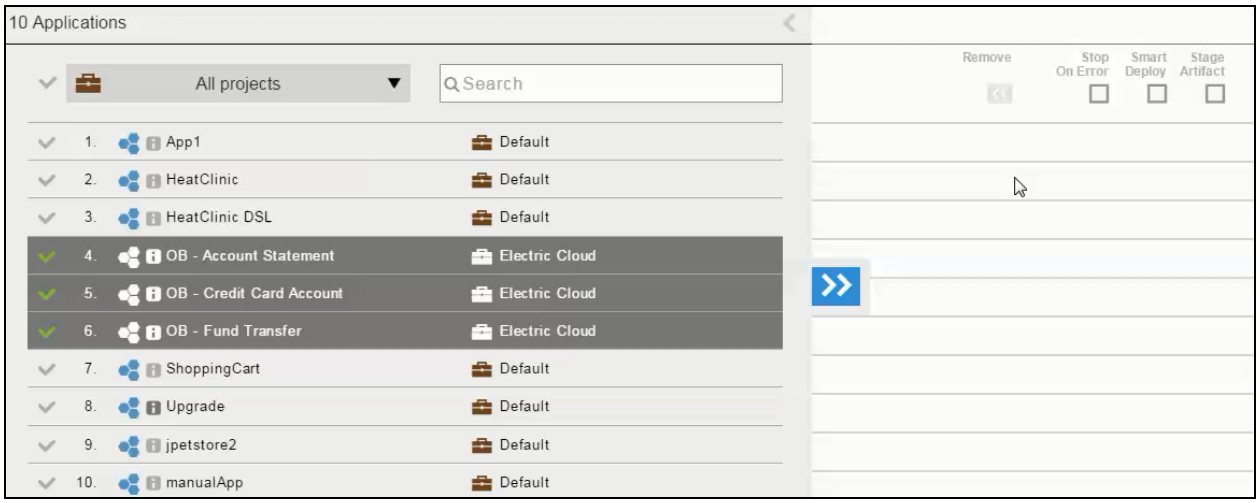
   The Application field is now enabled in the Release Definition page.



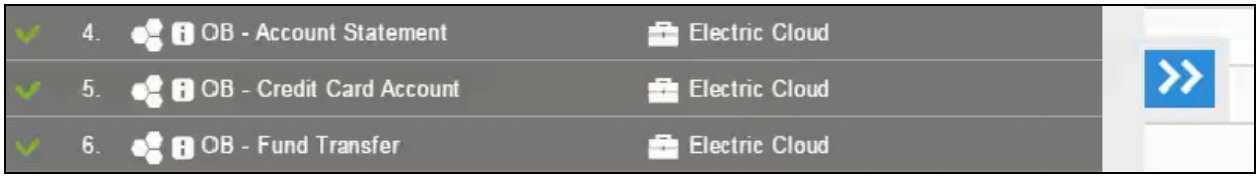
10. Click **Applications**.

    The dialog box where you specify the applications to be deployed through the pipeline opens.

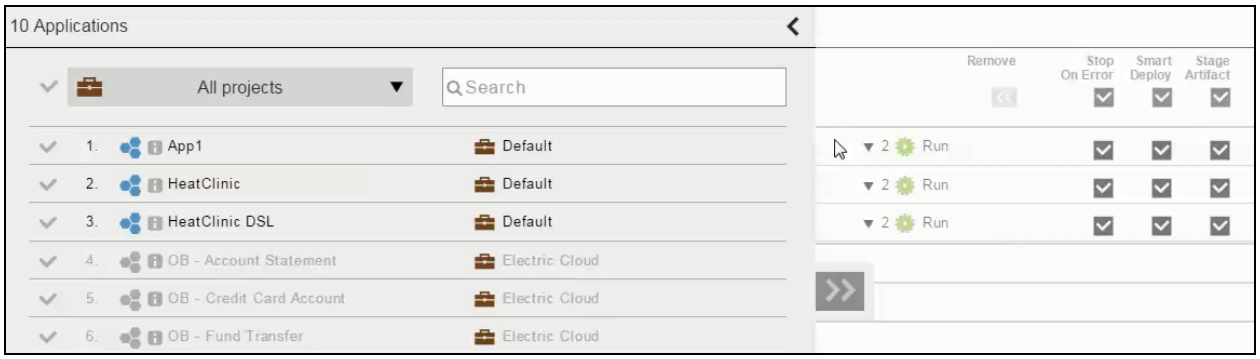
11. Select the applications that you want to deploy in the release, and then click the  button to move them to the right side of the dialog box, which is the list of applications to be deployed through the pipeline.



These applications are selected:



Clicking button moves the selected applications to the right side and displays the number of required parameters at run time and the application process to be deployed.



It shows the list of available applications. The default is to show the applications for all projects.

If you want to see only the applications for a specific project, click the down arrow in the **All projects** field to select one or more projects one of these ways:

- Click on the name of one or more projects.
- Enter the search criteria in the Search field. The projects that match the criteria appear in the list.

You can also search for one or more applications by entering the search criteria in the **Search** field next to the **All projects** field. If there are no matches, a message appears stating that there are no applications in the selected projects.

12. Click **<** to set how you want the applications to be deployed during the release run.

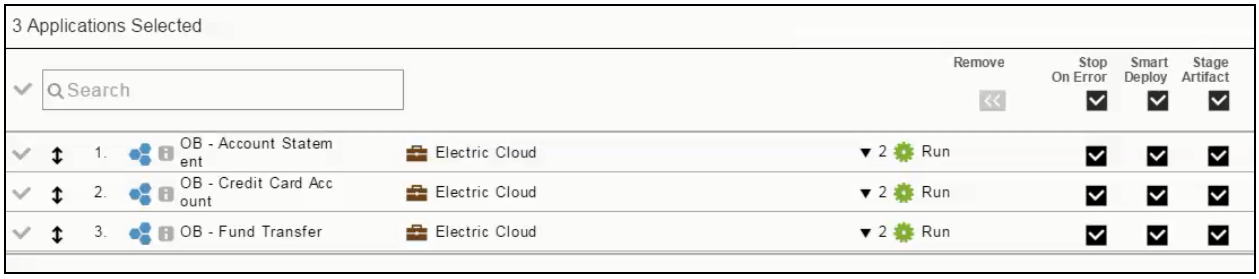
    For each application, you can select these settings:

    - **Stop on Error**: Select this error handling setting to abort the release run when the deployer task fails. The default is **Stop on Error**. If the application fails, the release pipeline aborts.

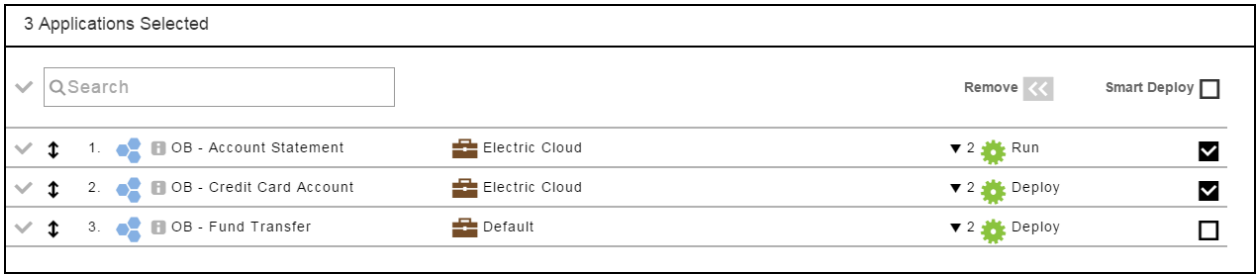
      When this setting is not selected (continue on error, which is similar to the **Continue running** option in a pipeline stage or gate), the deployer continues to the next application with a known error if the current application fails. This allows the release to continue running when an application run has errors and fails. For example, in a release with multiple applications, some applications can fail and not be deployed with other that were successfully deployed as delivered in the release.

    - **Smart Deploy**: Select this setting if you want the release to deploy the application only with artifacts that have not been deployed to a resource or with selected versions of the artifact have not been deployed to new resources since a previous run.
    - **Stage Artifacts**: Select this setting if you want the release to retrieve artifacts that will be deployed in an application run before the deployment starts.

    In this example, all the settings are selected for the selected applications. The release will stop if any of the applications fails.



Note: The default is that ElectricFlow uses smart deploy to deploy the applications in the pipeline. If you want to use smart deploy for all the applications, you do not need to change the settings.



13. If the applications have snapshots, click the down arrow next to  , and select a specific snapshot for each application, and click **OK**.

    Using a snapshot as part of the release definition is a best practice. Using snapshots, you can lock the exact versions of the artifacts to be deployed, which provides consistency and predictability. But you do not have to select a specific snapshot for an application.

    If a snapshot is not selected for an application here, the exact artifact versions to use would be based on how that application is modeled. That is, if the application model uses the *latest* artifact version for a component, the latest version would be used every time the release runs when a snapshot is not selected.

    When the release runs, this list of applications (the payload) is passed to the deployer task in the pipeline, and the applications are deployed through the pipeline sequentially. You can change the order in which the applications will be deployed by reordering the application input in the payload.

14. Click **Environments** to select the where the applications in each stage will be deployed.

    The "Environment selection" dialog box opens.

    Each cell in the table represents an environment where application in a stage will be deployed.

    ## About the filters in the "Environment selection" dialog box:

    When you mouse over the filter icons, the tool tip text appears, from left to right:

    - View only cells with missing environments.
    - View only cells with missing tier maps.
    - View only cells marked "N/A".
    - View only empty cells.

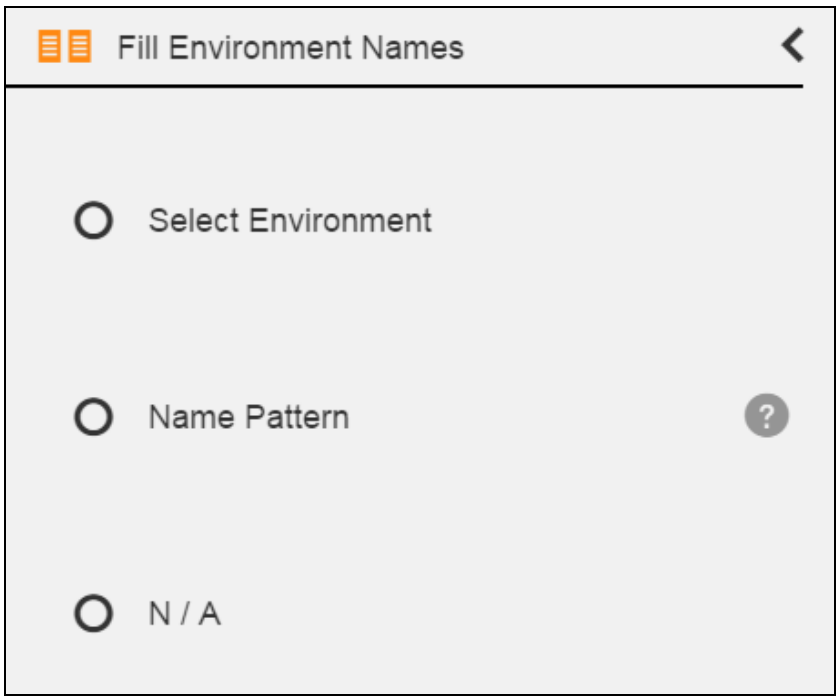
15. Select one or more cells in the table one of these ways:

    - For one cell (one application-stage combination)–You can select one cell by clicking in it. This allows you to select the environment for a specific application-stage combination.
    - For multiple cells (all the applications in a stage)–If the environment in a stage is the same for all the applications, you can select all the cells in a column (stage) by clicking in the header cell with the stage name.

    After one or more cells is selected, the **Fill** button in the upper right corner is enabled.

16. Click the **>** button in the upper left corner of the dialog box..

    The "Fill Environment Names" section opens on the left side of the dialog box.



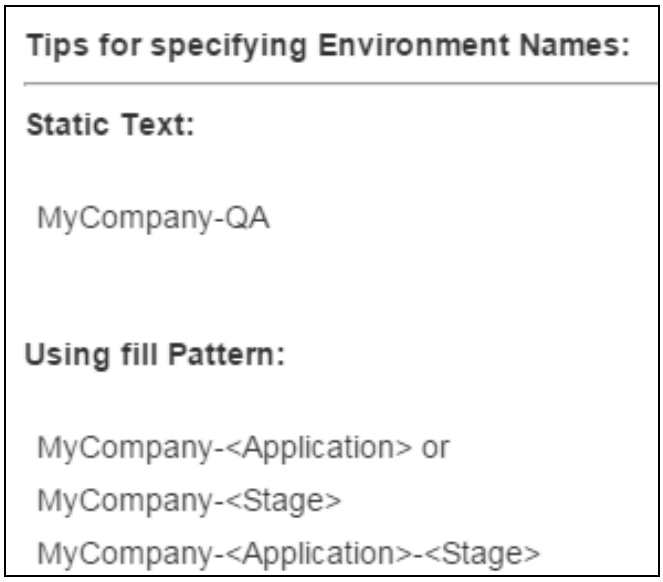
17. Select an environment using one of these options:

    - **Select Environment**–Click the down arrow to select an environment or environment template.

      Clicking **Select Environment** opens the list of available environments in the **Environments** tab. These are static environments that are authored before deploying the application in the stage.

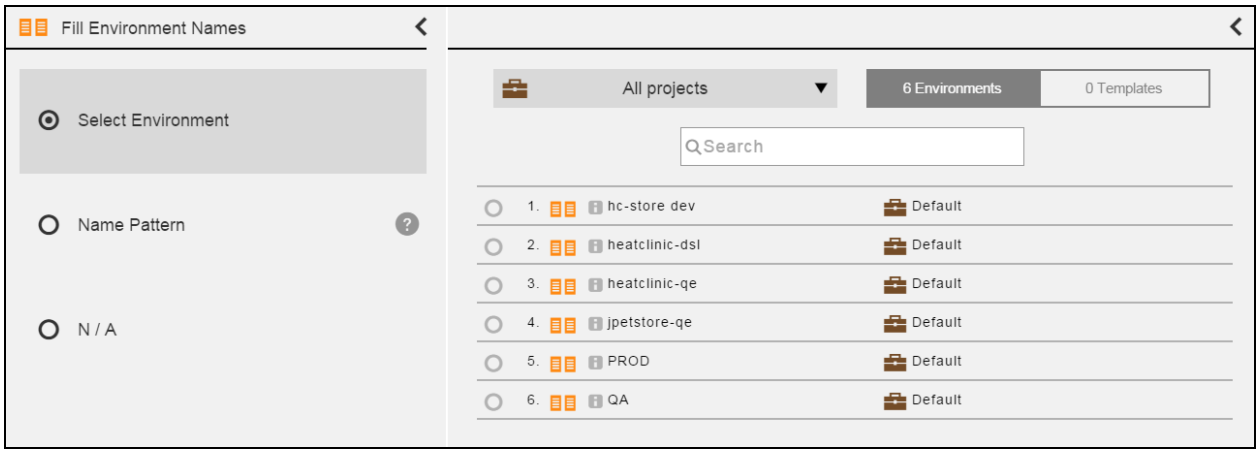
      Clicking on the **Templates** tab opens a list of environment templates. These are dynamic environments with provisioned cloud resources that can be spun up during the application deployment. For more information about dynamic environments, see Modeling Dynamic Environments on page 344.

    - **Name Pattern**–Click **?** for tips on how to specify environment names. The tips describe how to specify environment names using static text or a fill pattern.

    - **N/A**–Select this to not run the application in the stage. You should set environment mapping to **N/A** for stages where you do not want an active deployment even though the deployer is attached to a stage.

    In this example, the "hc-store dev" environment is selected from the Environments list.



It shows the list of available environments. The default is to show the environments for all projects.

If you want to see only the environments for a specific project, click the down arrow in the **All projects** field to select one or more projects one of these ways:

- Click on the name of one or more projects.
- Enter the search criteria in the Search field. The projects that match the criteria appear in the list.

You can also search for one or more environments by entering the search criteria in the **Search** field next to the **All projects** field. If there are no matches, a message appears stating that there are no environments in the selected projects.

18. Click **OK** to close the "Fill Environment Names" section.
19. Click **OK** to save the settings.
20. Click **Configurations** to enter the inputs (parameters) for the applications in a stage.

    If any of the required parameters have not been entered on one of the cells for an application in a stage, you will see this:  .

    You can enter the value for it now or at runtime.
21. Click **OK**.

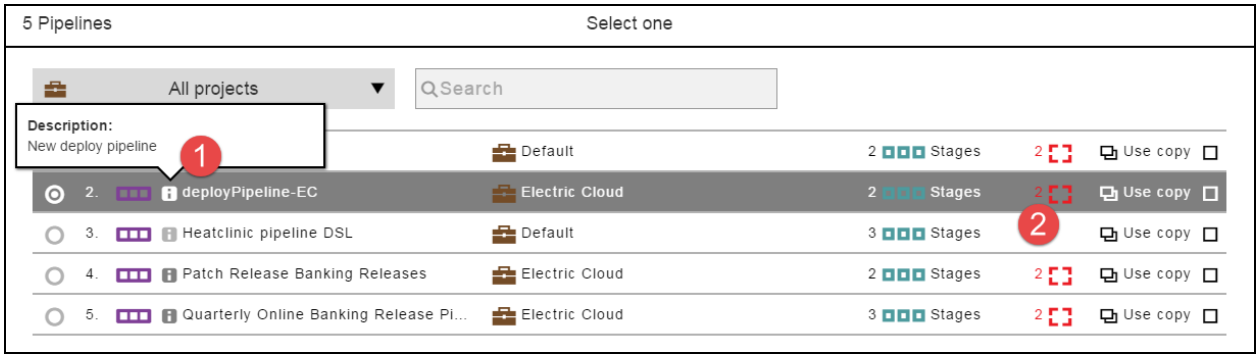
The next step is to run the release. See Running and Completing Releases on page 683 to learn more about running and completing the release.

Example: Continuous Delivery Style Releases for One Application

This example shows that when you define a release, you do not have to enter all the details. However, any configuration issues have to be resolved before the release run.

After a release is added, the release definition opens:

Click in the **Pipeline** field. A dialog box opens where you can select a pipeline:



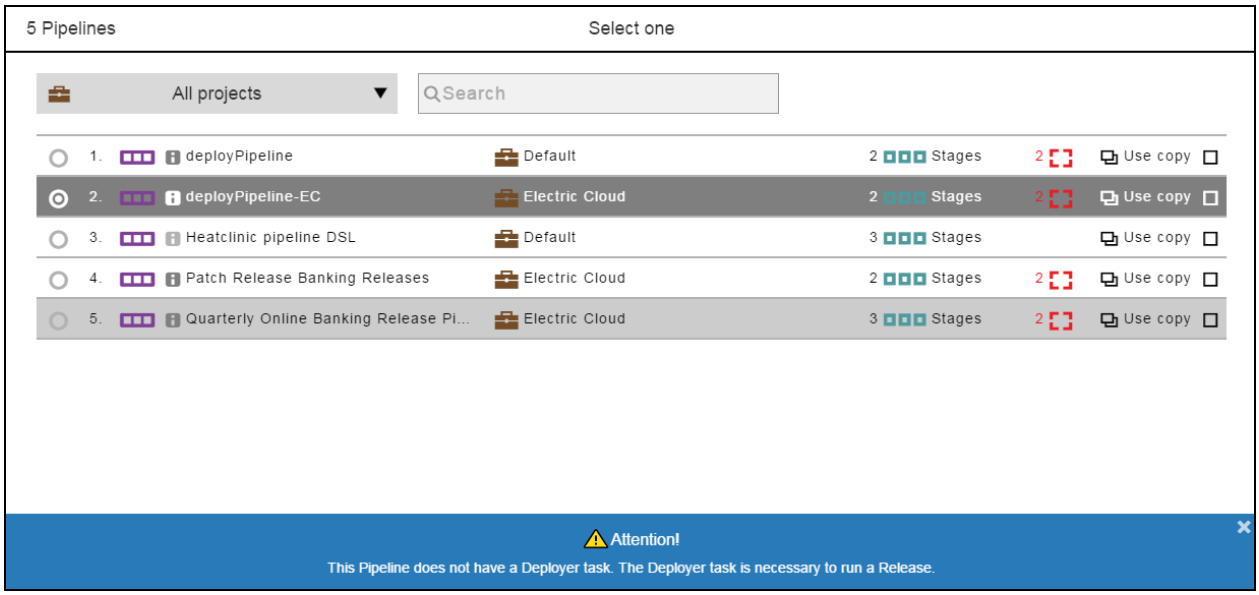
When you click the button (number 1 above), a description of the deployPipeline appears. The icon on the right (number 2 above) means that two required parameters are attached to the pipeline.

It shows the list of available pipelines. The default is to show the pipelines for all projects. To see only the pipelines for a specific project, click the down arrow in the **All projects** field to select one or more projects in one of these ways:

- Click on the name of one or more projects.
- Enter the search criteria in the Search field. The projects that match the criteria appear in the list.

You can also search for one or more pipelines by entering the search criteria in the **Search** field next to the **All projects** field. If there are no matches, a message appears stating that there are no pipelines in the selected projects.

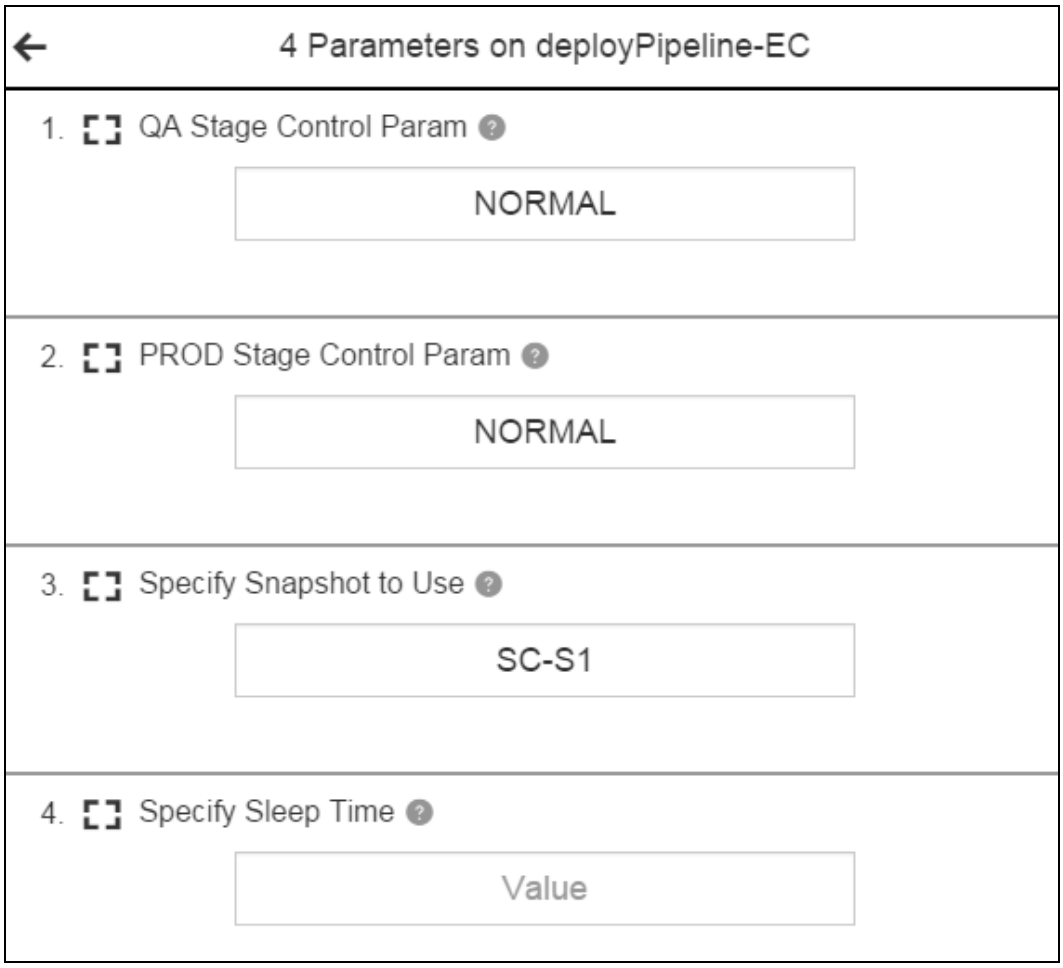
Select the **deployPipeline** application, and click **Next**.



If the pipeline does not have a deployer task, a message appears reminding you that you need to define a deployer task to run the release. Close the message to continue, and click **Next**.

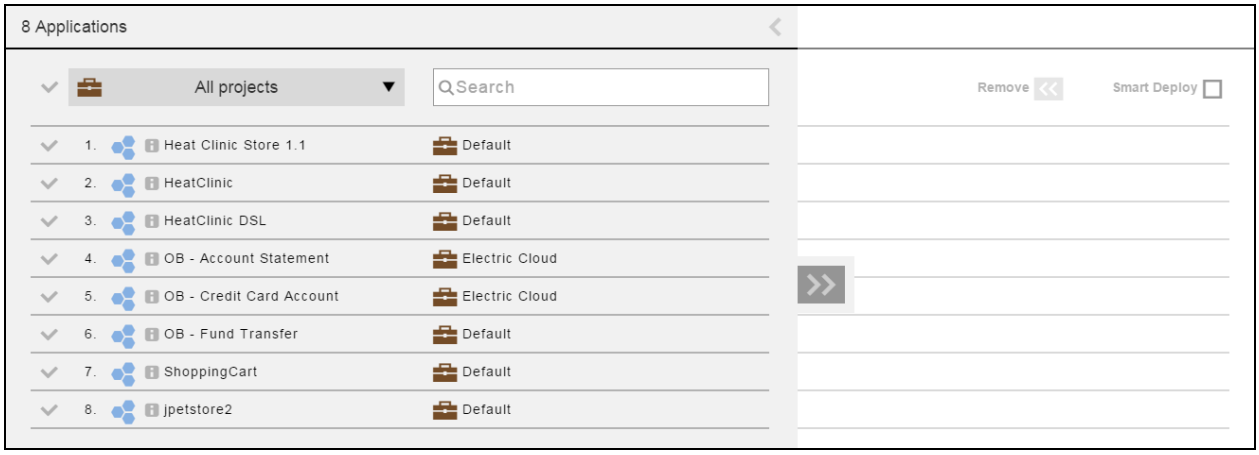
If the pipeline has a deployer task, the message does not appear so you need to only click **Next**.

In the example, the pipeline has four required parameters. Some of them have default values that you can keep or change. Others do not, and you have to enter a value for them.



After you enter the values, click **OK**.

Now click in the **Applications** field. The dialog box to select one or more applications opens:



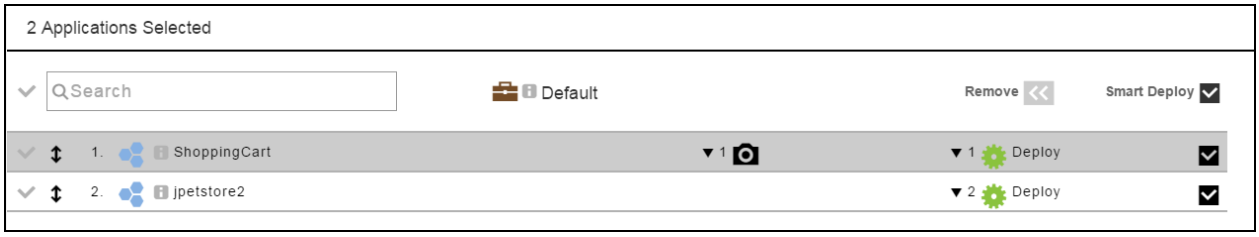
It shows the list of available applications. The default is to show the applications for all projects.

If you want to see only the applications for a specific project, click the down arrow in the **All projects** field to select one or more projects one of these ways:

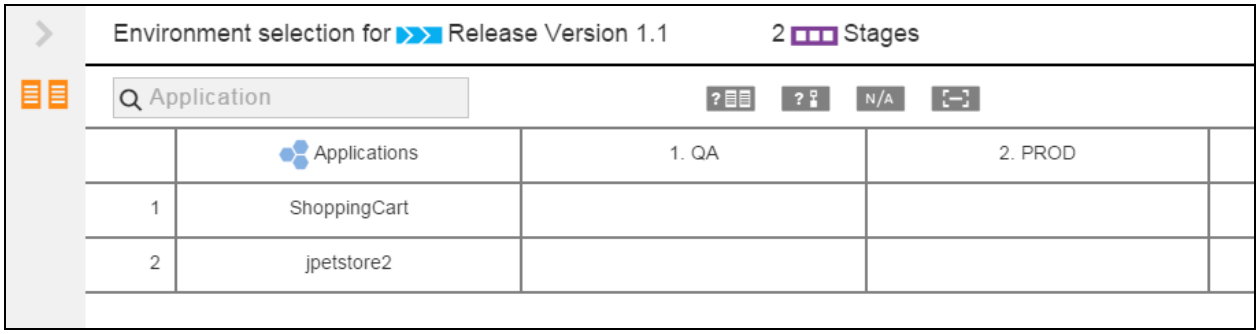
- Click on the name of one or more projects.
- Enter the search criteria in the Search field. The projects that match the criteria appear in the list.

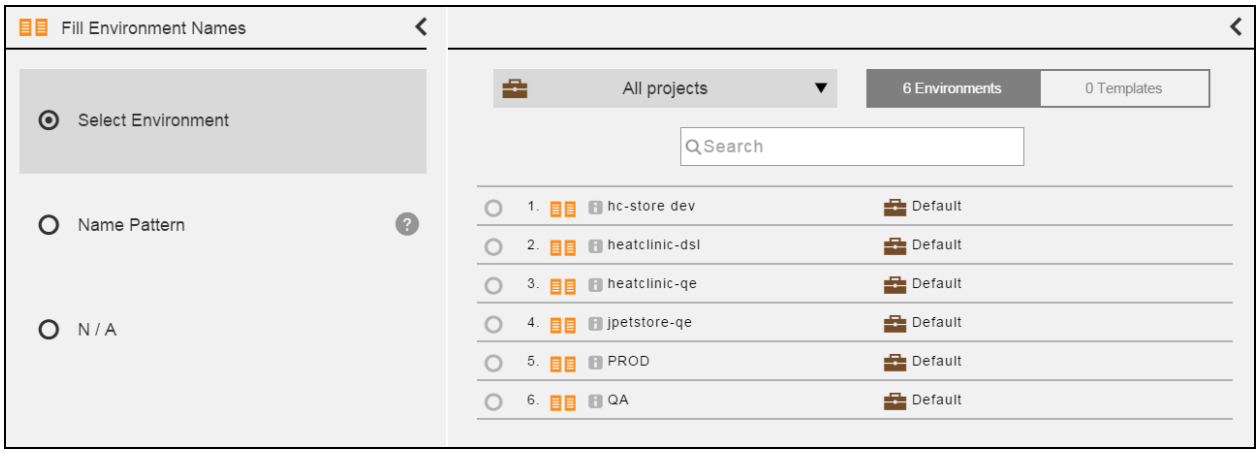
You can also search for one or more applications by entering the search criteria in the **Search** field next to the **All projects** field. If there are no matches, a message appears stating that there are no applications in the selected projects.

Select one or more applications, click the  button to add them to the pipeline, click **OK**, and click **<**:



Now click in the **Environments** field. The dialog box to select an environment where the applications in each stage will be deployed opens:





It shows the list of available environments. The default is to show the environments for all projects.

If you want to see only the environments for a specific project, click the down arrow in the **All projects** field to select one or more projects one of these ways:

- Click on the name of one or more projects.
- Enter the search criteria in the Search field. The projects that match the criteria appear in the list.

You can also search for one or more environments by entering the search criteria in the **Search** field next to the **All projects** field. If there are no matches, a message appears stating that there are no environments in the selected projects.

About the filters in the "Environment selection" dialog box:

When you mouse over the filter icons, the tool tip text appears, from left to right:

- View only cells with missing environments.
- View only cells with missing tier maps.
- View only cells marked "N/A".
- View only empty cells.

For detailed instructions about how to select the environments in which the applications in each stage will be deployed, go here.

In this example, the environments are selected as follows:

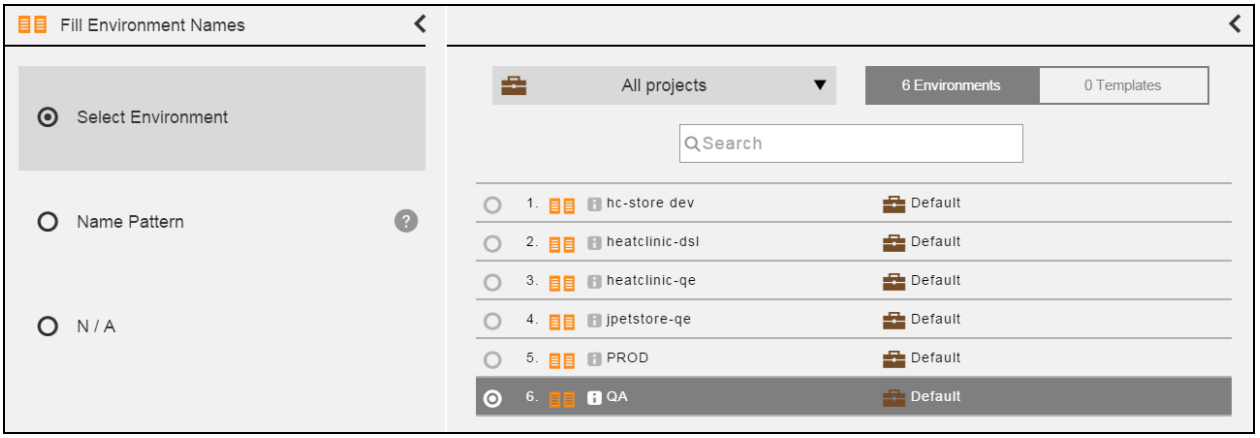
- ShoppingCart application in the QA stage:

  In the ShoppingCart row, select the cell in the QA column and double-click in it. Notice that the **Fill** button is now enabled.

  A panel opens to the left where you can select the environment where the ShoppingCart application will be deployed.

  Select the environment to which the ShoppingCart application is deployed: **QA**.

Click **OK** to save your selection.

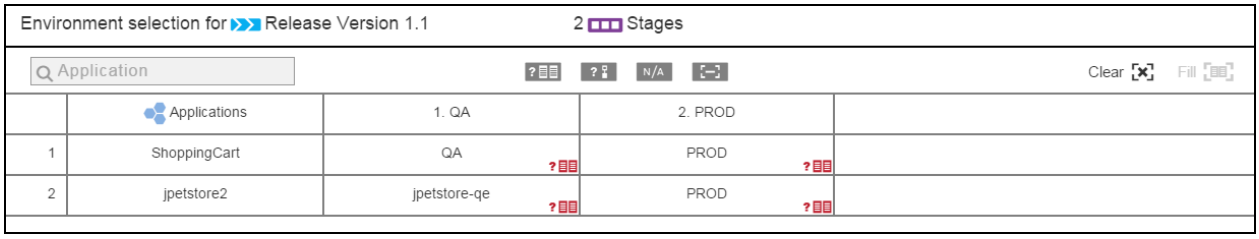


- jpetstore2 application in the QA stage:

  In the jpetstore2 row, select the cell in the QA column and click in it. Notice that the **Fill** button is now enabled.

  A panel opens to the left where you can select the environment where the jpetstore2 application will be deployed.

  Select the environment to which the jpetstore2 application is deployed: **jpetstore-qe**.



  Click **OK**.

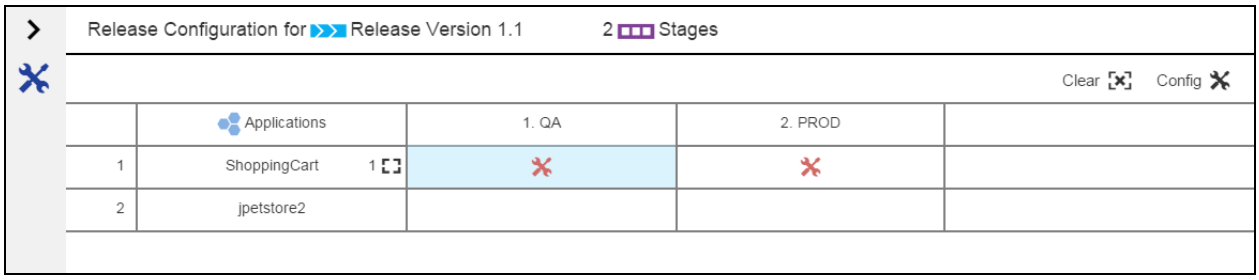
- All the applications in the PROD stage. Click in the header cell for the PROD stage.

  A panel opens to the left where you can select the environment where the applications will be deployed.

  Select the environment to which the applications will be deployed: **PROD**.

  Click **OK**.

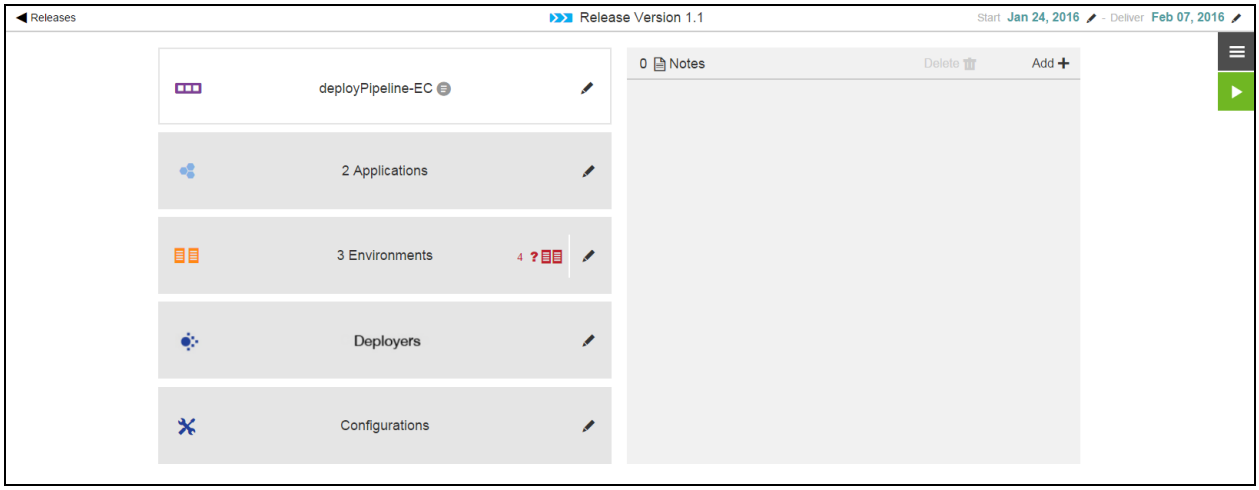
Now click in the **Configurations** field. The dialog box to enter the required parameters for the pipeline to run opens:



If any of the required parameters have not been entered in a cell (an application to be deployed in a

stage) with in it, you can click in the cell to enter the inputs (parameters) for the application in a specific stage. You can enter the value for it now or at runtime. Then click **OK** to save your entries.

The release definition now looks like this:



The next step is to run the release. See Running and Completing Releases on page 683 to learn more about running and completing the release.

Release Dashboard

The Release Dashboard, also referred to as the Release List, provides a way for the release team, task assignees, and approvers to quickly see the status of all the planned, active, or completed releases at any time. They can quickly see the current status, specific tasks and who is assigned to them, and the start and end dates for the stages in one place on one platform.

Accessing the Release Dashboard

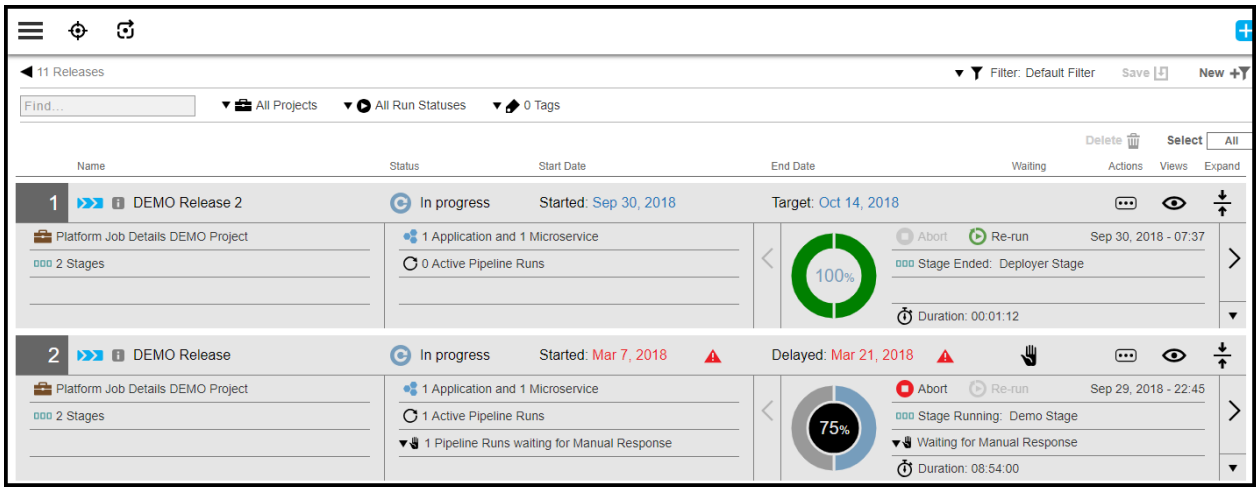
To open the Release Dashboard from the Home page (https://<*ElectricFlow_server*>/flow/), do one of the following actions:

- Click **Releases**.

- Click the **Main Menu** button ( ) and select **Releases**.

Release Dashboard

The Release Dashboard shows all the Releases that you have permission to view:



By default, the releases for all projects are displayed.

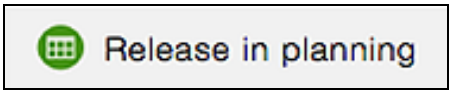
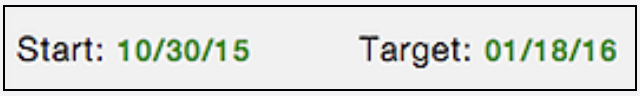
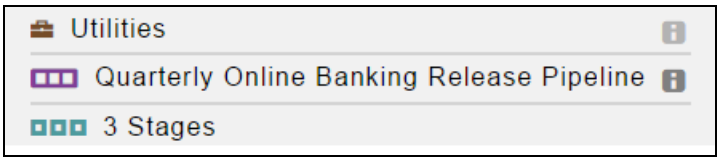
To see only the objects for a specific project, click the down arrow in the **All projects** field and then select one or more projects in one of these ways:

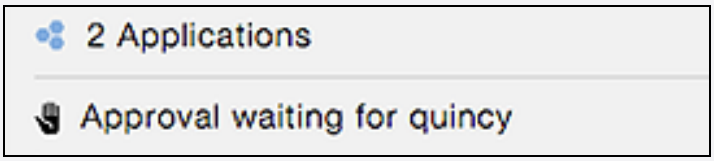
- Click on the name of one or more projects.
- Enter the search criteria in the Search field. The projects that match the criteria appear in the list.

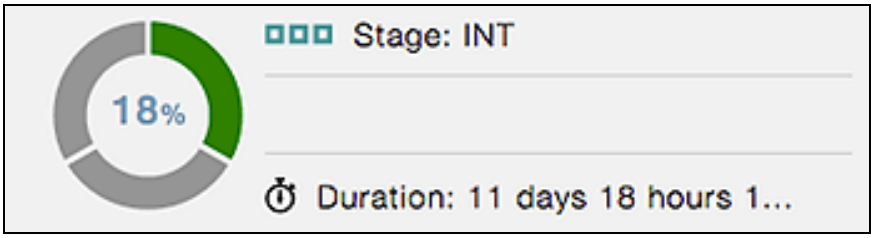
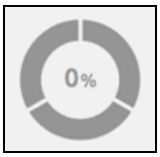
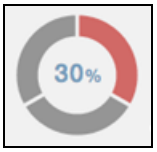
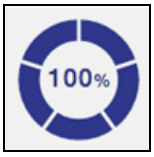
If there are no matches, a message appears stating that there are no resource templates in the selected projects.

The Release Dashboard has this information:

Each row in the Release Dashboard contains this information:	
Release name	Example: 

Release status	The status can be: • Release in planning • Release in progress • Release completed Examples: 
Start and target (end) dates for the Release	Example:  The default time frame between the start and target (end) dates is two weeks. The color of the dates indicates the Release status. • Green: The Release has not started yet, and the start and end dates are within the planned time frame. • Medium blue: The Release is in progress, and the dates are the actual start and end dates. • Dark blue: The Release is completed, and the dates are the actual start and end dates. • Red: The date is past the planned end date.
Project to which the pipeline belongs, name of the pipeline underlying the software release, and the number of stages in this pipeline	Example: 

Number of applications in the pipeline and any notifications for actions to be taken before the pipeline progresses	Example:  This pipeline requires human intervention to progress.The user named quincy needs to approve something at an entry or exit gate of a stage before the pipeline progresses.

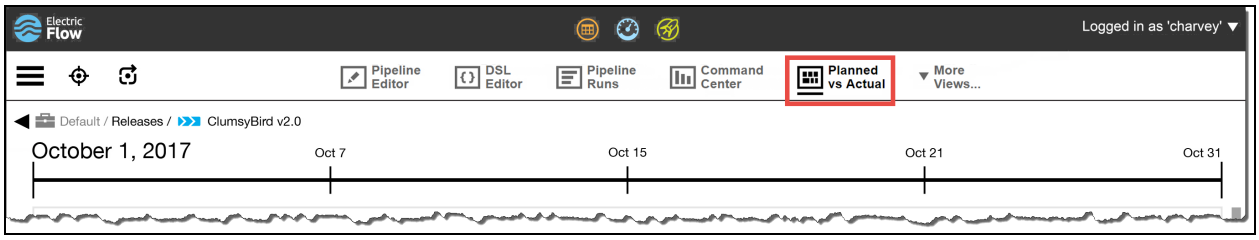
Visual indicator showing how far the Release has progressed with the percentage completed, the current stage, and how long the Release has been running.	The visual indicator is split into sections equal to the number of stages in the pipeline. If pipeline has three stages, it is split into three segments. This example shows that the Release is running without errors, is at the INT stage, and has been running for over 11 days. The duration is the total time that the Release has been running before the Release is ended.  If the Release is still in the planning phase, the percentage completed is 0%, and the segments are all gray.  If the pipeline is running with errors, the part of the segment representing the current completed stage is red.  If the pipeline run is completed, the visual indicator is blue and the percentage completed is 100%. 
Buttons and Icons in the Release Dashboard	

Entering part or all of a pipeline name in the **Find** field on the top of the dashboard starts the search for Releases matching the search criteria.	
Selecting **All** or **None** and then clicking the **Delete** button removes the selected pipelines from the list.	
Clicking the **Add+** button adds a Release.	
Clicking this button opens to Path-to-Production view, which displays the applications being deployed in the Release and the environments where they are deployed. The details in this view include the application versions and the environments that are not in compliance with the bill of materials throughout the Release. See Path-to-Production View on page 680for more information.	
Clicking this button opens the Release Definition dialog box. See Release Definition on page 646.	

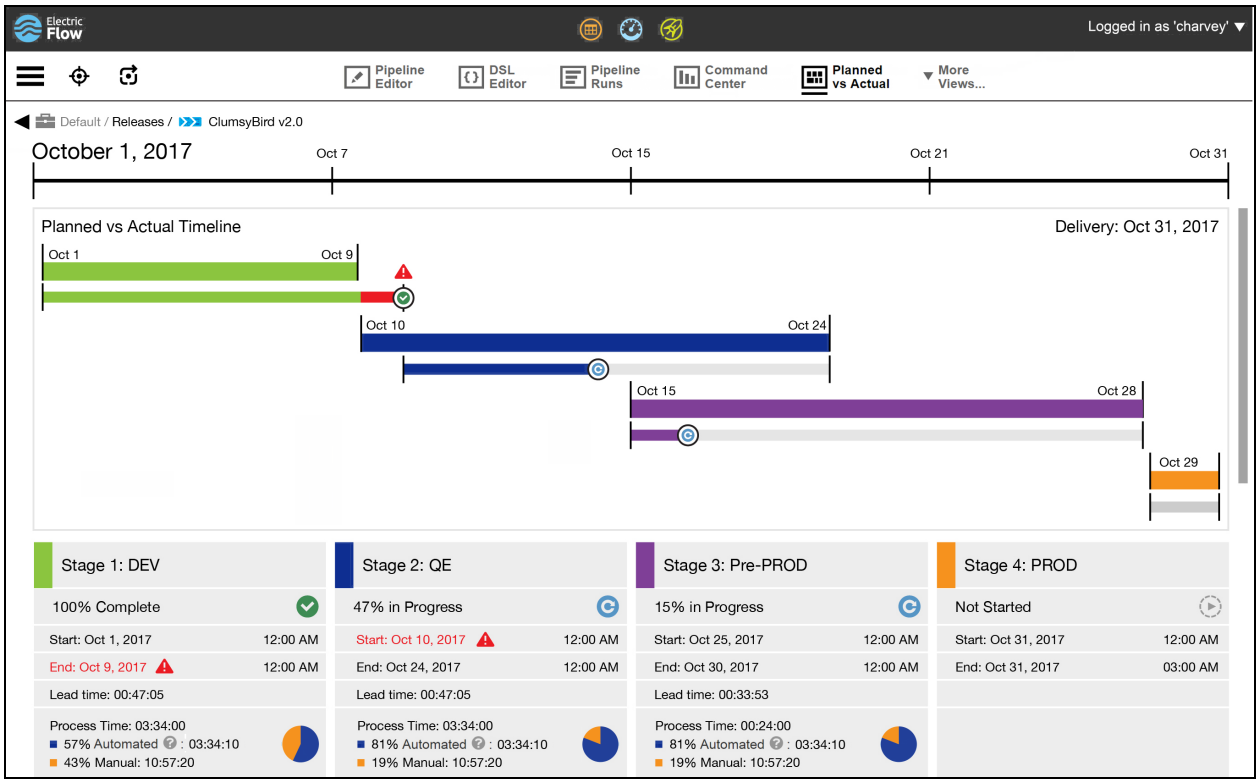
Clicking this button starts a Release run. You can optionally select a previous release run to use the parameters from that run. You can modify one or more of these parameters as described in the dialog box where the runtime parameters and settings are set.	
Clicking this button allows you to end, re-run, or abort the Release pipeline. To go to the Pipeline Run page, click the  button in the upper left corner of the Release Dashboard, and select **Pipelines** > **Pipeline Runs**. For more information about the Pipelines Runs page, see Running Pipelines. For more information about running and ending (completing) the Release, see Running and Completing Releases on page 683.	 Select one of these actions: • **End Release**–Clicking this ends the Release. You cannot end the pipeline that is currently running. If you want to end it, you must wait for this run to complete or abort it in the Pipeline Run page. When the pipeline is aborted, tasks that are currently running will finish, and no new tasks will be started. • **Run Pipeline**–Clicking this starts a new pipeline run. • **Abort All Runs**–Clicking this aborts all pipeline runs. • **Schedules**–Clicking this lets you create a release schedule.

Planned Versus Actual View to Analyze Pipeline Status

The Planned vs Actual view shows a bar-chart timeline of each pipeline stage versus your plan and the percentage complete of each stage to show where you are on your milestones. This view also shows the total time spent per stage and the time spent in automated versus manual activities. To see this view, click the **Planned vs Actual** button in the menu bar:

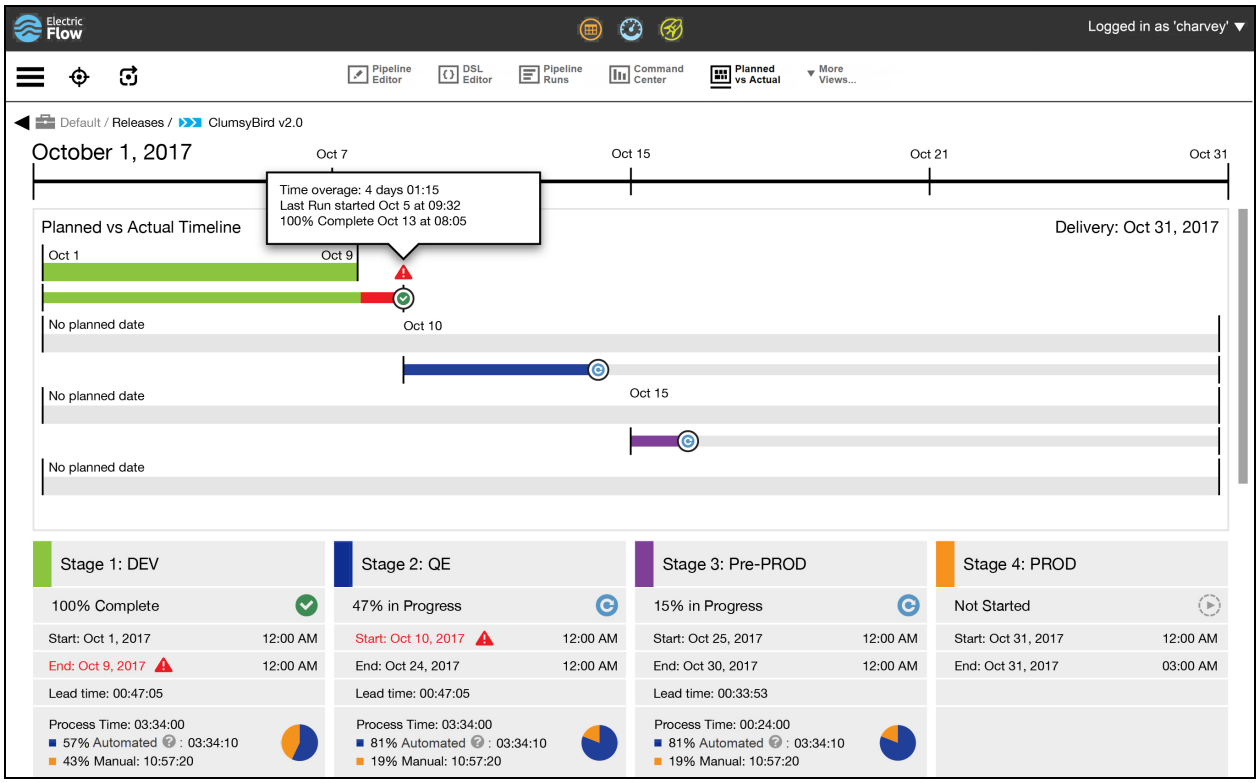


Following is an example:



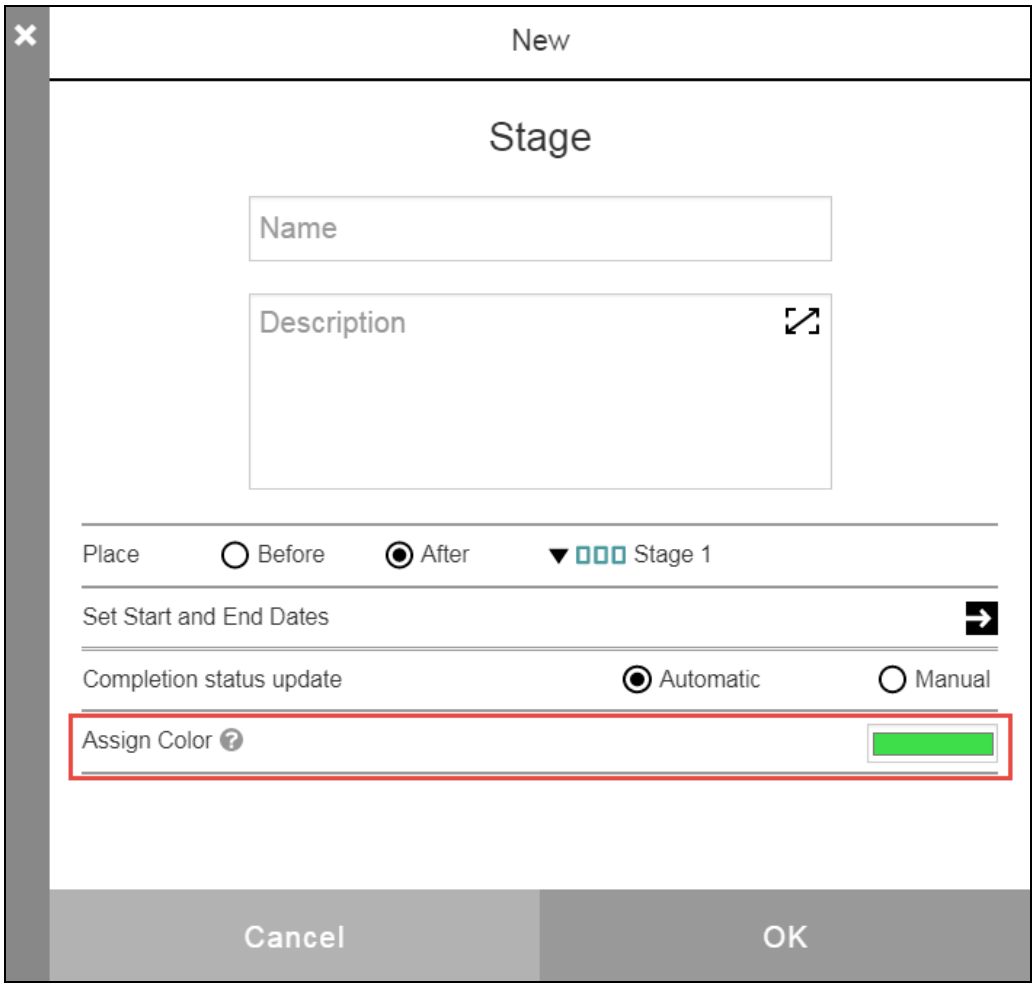
Viewing Key Details for a Pipeline Stage

You can click the end point of an "actual" progress bar for a pipeline stage to view key details. For example:

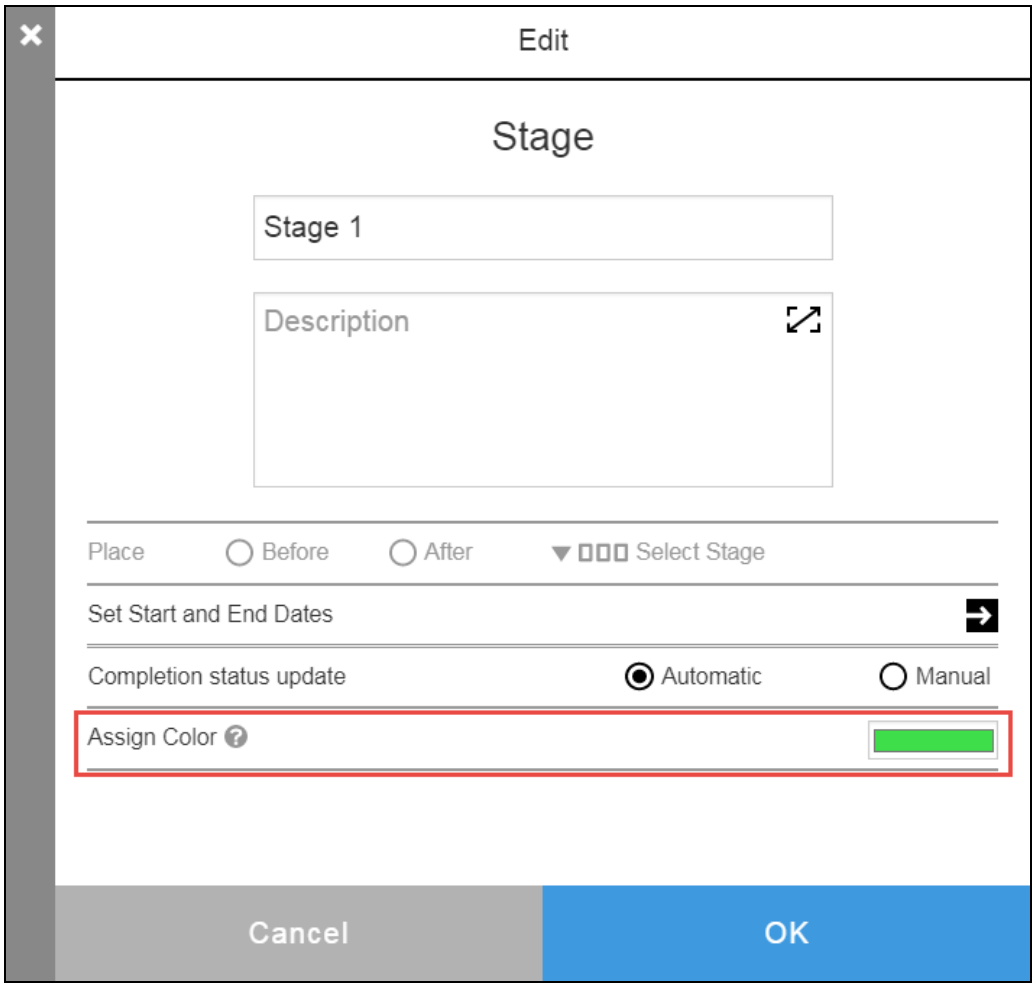


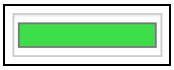
Changing the Color Coding for Pipeline Stages

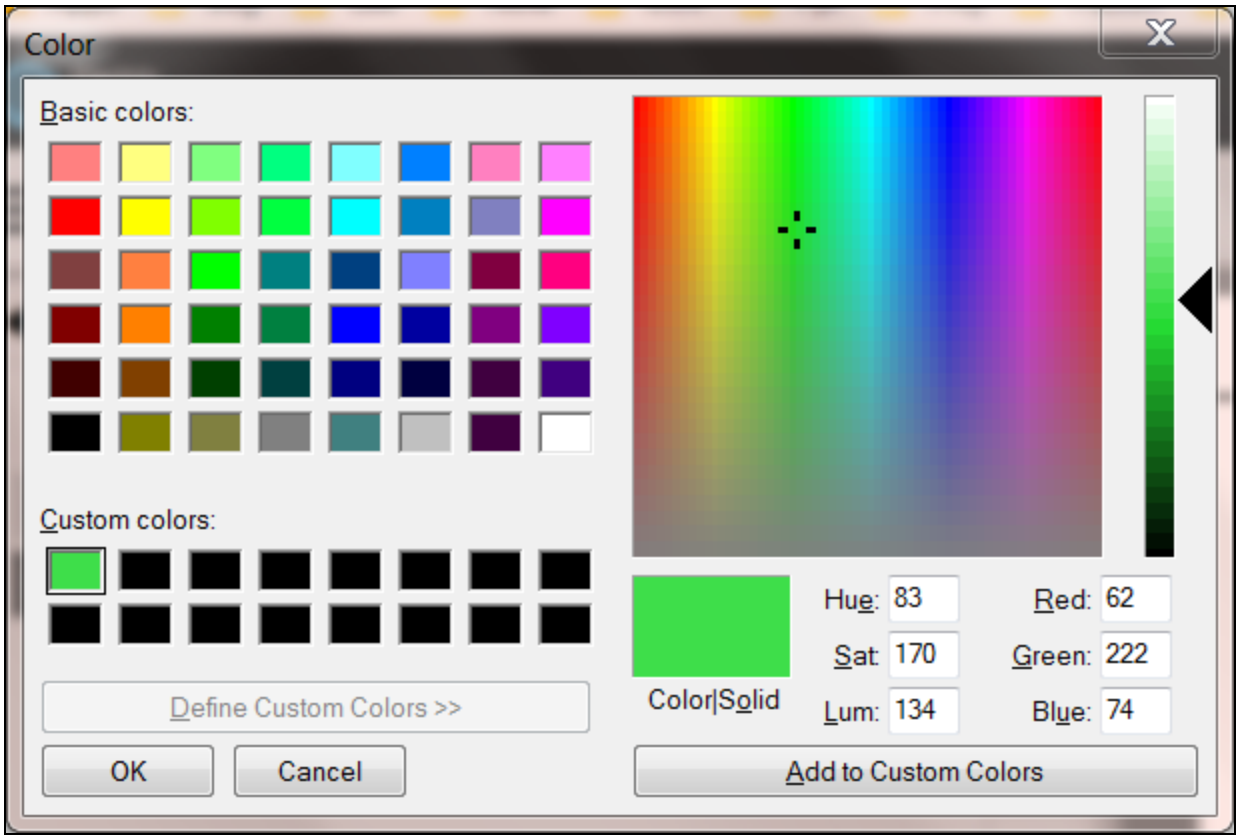
You can customize the color in which a new stage will appear in the Planned vs Actual view by clicking the **Assign Color** button in the pipeline stage details dialog box as in the following screenshot and then choosing a color from the palette that appears:



You can also change the color coding for an existing stage. To do so, start by clicking the button in the stage and then clicking **Details**. The details dialog box for the stage appears:



Then click the  (Assign Color) button. The color palette dialog box appears:



Finally, click a color in the dialog box and click **OK** to save your change.

You can optionally create custom colors and add them to the **Custom colors** palette for later selection via the **Add to Custom Colors** button.

Path-to-Production View

The *Path-to-Production view* shows the applications deployed in the release, which versions are deployed, and the environments to which they are deployed. The details in this view include the application versions, snapshots, and the environments that are not in compliance with the *bill of materials* throughout the release.

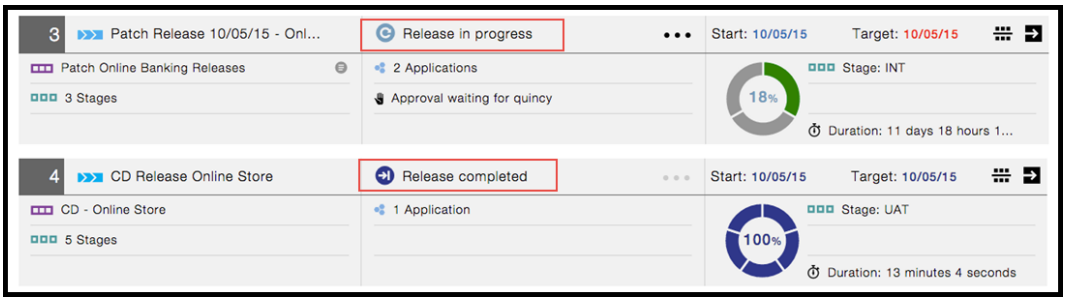
To go to the Path-to-Production view:

1. Go to the Release Dashboard.
2. Find a release that is in progress or completed.

   The button should be enabled.

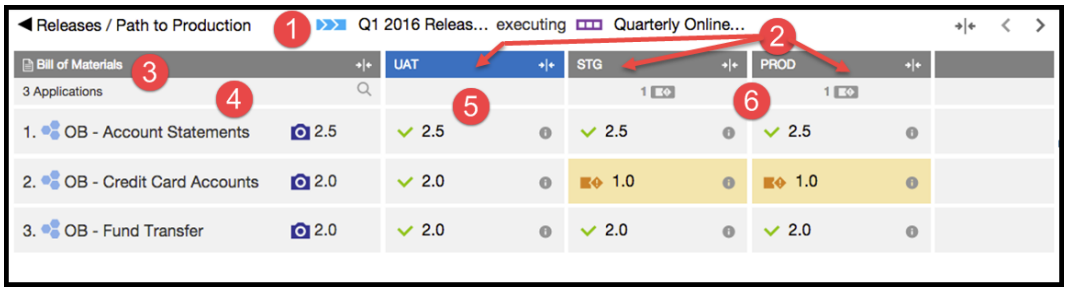
   Example:

   Select one of these releases:

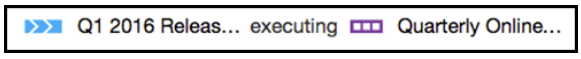


3. Click the button to open to Path-to-Production view.

This is a Path-to-Production view showing the bill of materials.



This information is in the bill of materials:

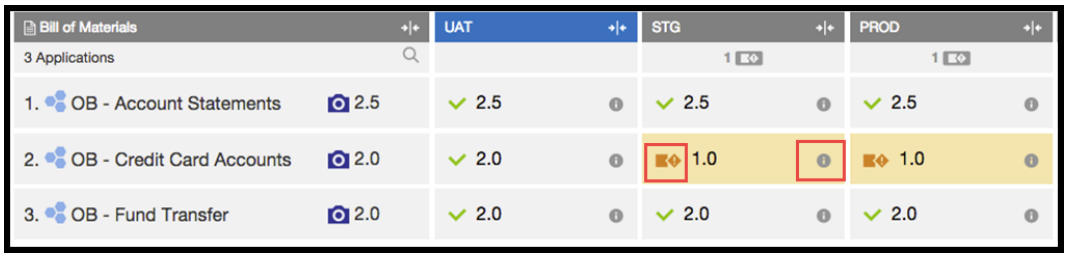
1	Release name and pipeline name	Example: 
2	Stage	Each column shows the details for each stage in all the applications. Stage names are defined from the associated Pipeline. The stages in this example are *UAT*, *STG*, and *PROD*.

3	Summary	The bill of materials has three applications. In this example, the STG and PROD stages each have a  compliance issue indicated by .
4	Application	Each row shows the details for each application across all the environments. The application names are defined from the associated pipeline. The applications in this example are: • OB—Account Statements • OB—Credit Card Statements • OB—Fund Transfer
5	UAT stage	In this example, the UAT stage has no compliance issues. The versions of the applications deployed in the release match what is in the bill of materials.
6	STG and PROD stages	In this example, the STG and PROD stages have compliance issues. The "OB—Credit Card Accounts" application version 2.0 is supposed to be deployed in the STG and PROD stages. However, version 1 of the application is deployed instead.

To get more details about compliance issues:

- Mousing over shows a message explaining the noncompliance. Use this information to troubleshoot and resolve the noncompliance.

- Clicking shows the environment to which the application is deployed, the deployment date, and information about the deployed components and artifacts.



You can take these actions in the bill of materials:

- Clicking the  button shows less detail.
- Clicking the  button shows all the details.
- Clicking the  button starts opens to Search field, where you can enter the part of an application name to find specific applications, which is especially helpful when you have a long list.

Release Summary

The Release summary feature allows Release-specific information from pipeline stage tasks to be displayed in the Release Dashboard. You can add Release summary information by creating properties under the `ec_releaseSummary` property sheet that exists on the run time associated with the Release.

These examples show how to do this from ectool:

```
ectool setProperty /myReleaseRuntime/ec_releaseSummary/summary1 --value "<property value in plain text>"
```

or

```
ectool setProperty /myReleaseRuntime/ec_releaseSummary/summary2 --value "<html><a href=\"http://electric-cloud.com/\">Electric Cloud</a></html>"
```

For a pipeline that is run from a release, after the task in a pipeline stage is completed, you can view the summary property in the Release Dashboard:

Running and Completing Releases

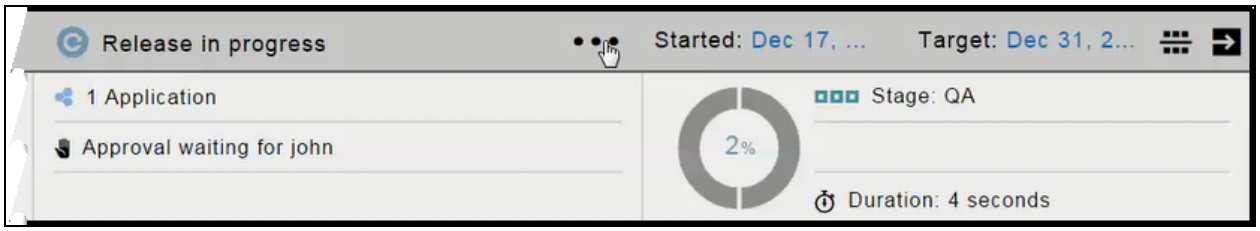
After a Release is defined, you can run it multiple times on the journey to production. At the end of the software release life cycle, when the Release process has been successfully completed and the software is ready for production, you can end the Release. Once the Release has ended, it cannot be run again.

Running the Release

This example shows how to run a Release called Q1_TimeBaseRelease.

You can start the Release run when the Release status is "Release in planning" and the **Run** button () is enabled.

The Release status changes to "Release in progress" and the **Run** button () has been replaced by the  button. Clicking on it shows a pop-up window with more actions. The Start and Target dates are in blue because the Release is in progress.

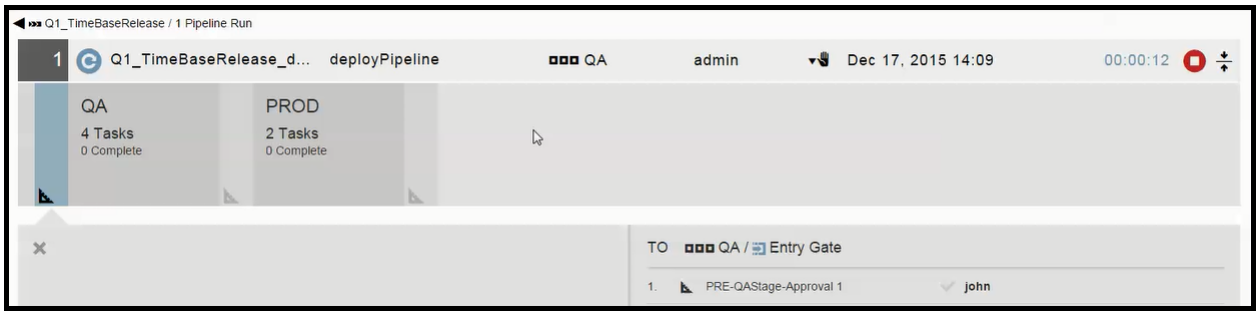


When 

is displayed, human intervention is required for the pipeline to progress to the first stage.

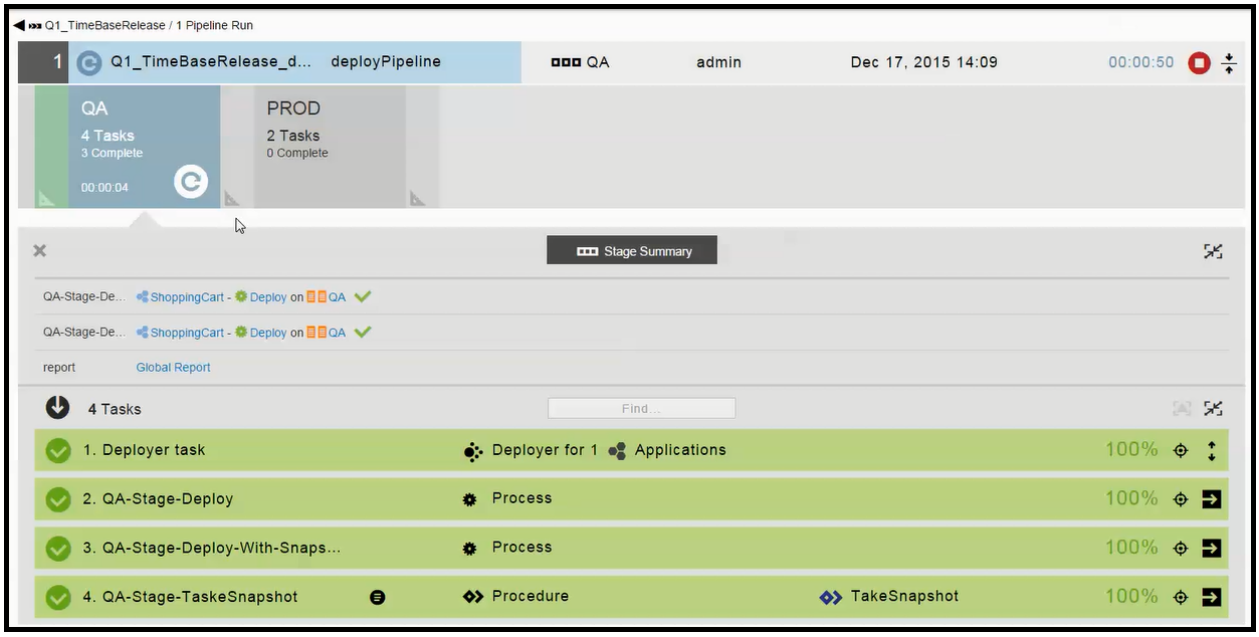
Clicking on 

causes the Pipeline Run view for the deployPipeline to be displayed.



The pipeline can progress after the user named john approves the entry to the QA stage of the pipeline.

As the pipeline progresses, you can click in the QA stage to view the pipeline stage summary:



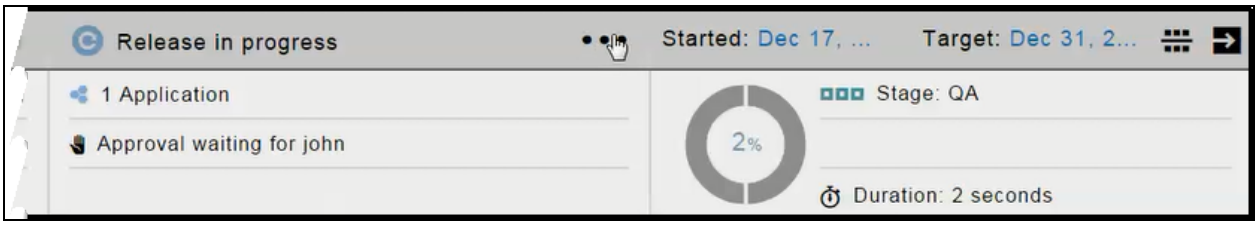
The pipeline continues to progress to the PROD stage after the required approvers respond at the exit gate. Progress continues until this Release run is completed.

After the Release is completed, you can start a new Release run by clicking the ••• button and selecting **Re-run Pipeline**. You can re-run the Release as many as times as you want during the software release life cycle.

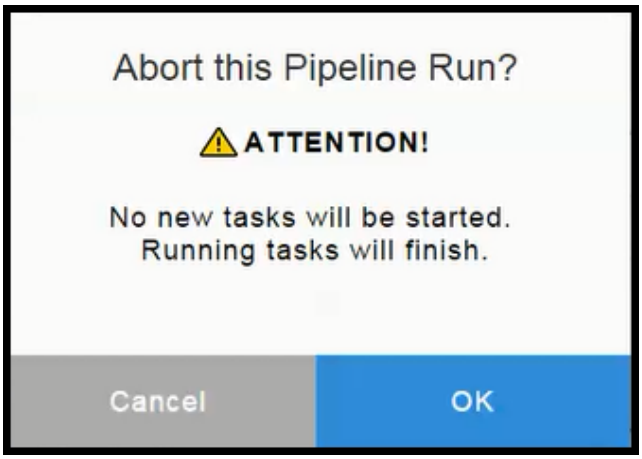
Aborting a Release Run

There are two ways to abort a Release run.

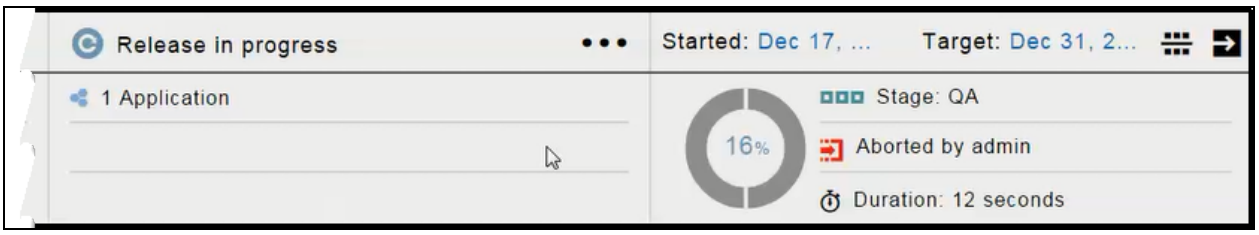
- During the first Release run, you can abort it from the Release Dashboard.



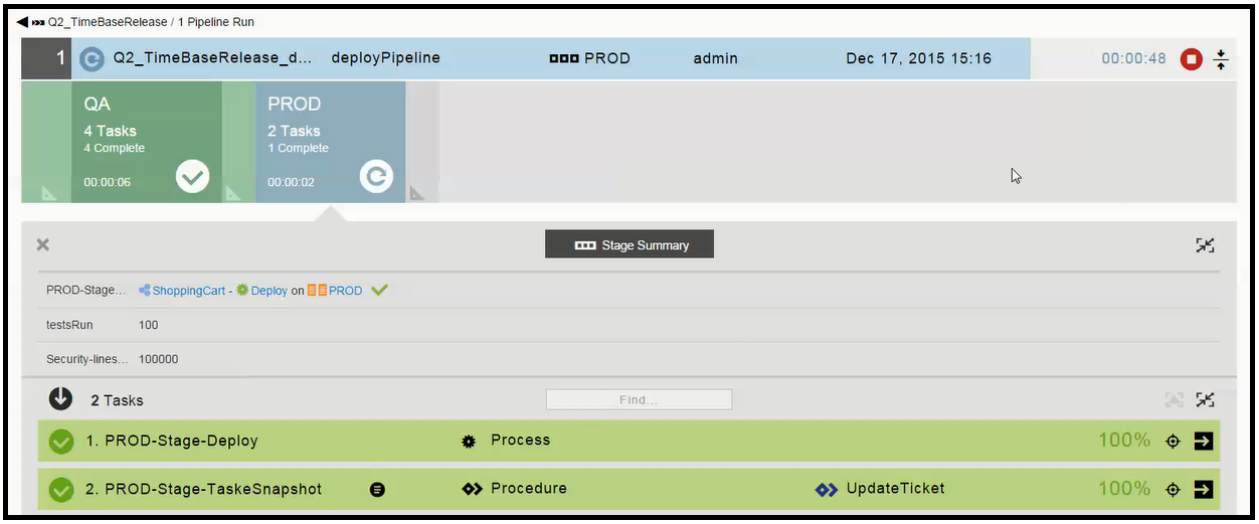
Clicking the ••• button and selecting **Abort Pipeline** will abort the pipeline. The tasks that are currently running will finish, and no new tasks will be started. This message appears:



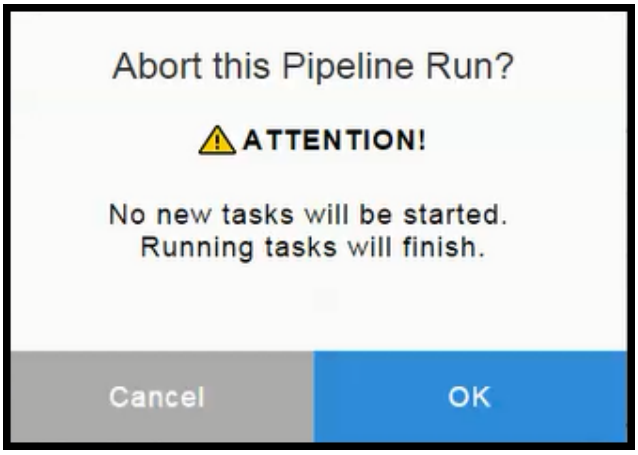
This is the Release status after the Release run is aborted.



- During subsequent Release runs, you can abort the current Release run from the Pipeline Run view.



Clicking the Abort ( ) button will abort the pipeline. The tasks that are currently running will finish, and no new tasks will be started. This message appears:



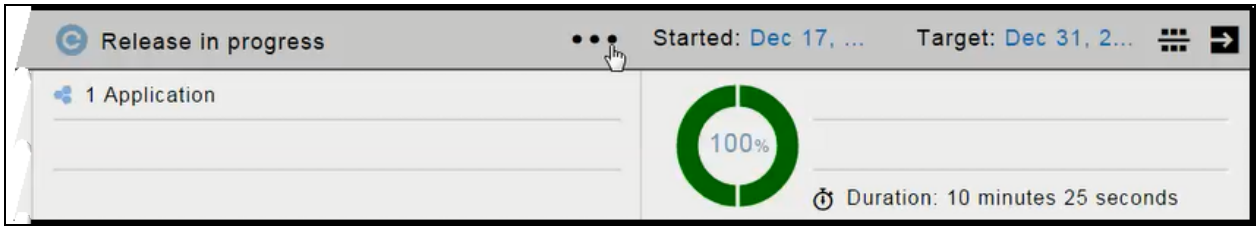
After you click **OK**, the Pipeline Run view now displays this status:

Completing the Release

When you complete the Release, no other actions can take place on it, and this action cannot be undone (reversed). The Release cannot be run again.

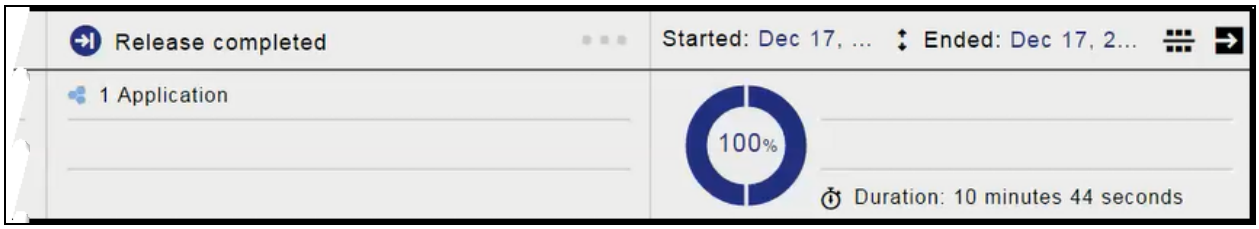
You cannot end the pipeline that is currently running. If you want to end it, you must wait for this run to complete or abort it in the Pipeline Run view. For more information, see Aborting a Release Run on page 685.

If the Release run is complete, you can complete the Release from the Release Dashboard.



Clicking the ... button and selecting **End Release** ends the Release. This message appears:

After you click **OK**, the Pipeline Run view now displays this status:



The Release status is now:

- The icon next to "Release completed" indicates that the Release has been completed (ended).
- The "percentage complete" icon on the right is split into two sections because the pipeline has two stages. It shows that the Release run is 100% complete.
- The Target date has been replaced by the Ended date.
- The Start and Ended dates are in dark blue because the Release is completed.
- The Duration is the total time for the Release.

Chapter 5: DevOps Insight

DevOps Insight provides dashboards that give you insights into deployment and release activities over time. The ability to visualize this information as dashboards enables enterprises to understand the overall status of their processes, identify hotspots that need action, understand trends, and find opportunities for further improvement. DevOps Insight provides several dashboards as well as the ability to create custom dashboards.

These dashboards include the Release Command Center dashboard, which provides a centralized view of all activities and processes related to a release. These include key metrics from the tools used across the end-to-end process, from work item tracking and build automation to test and operations tools. The Release Command Center dashboard helps you to understand how many user stories are planned, how many of them are "dev complete," how many are tested, what is the success rate of each of them, and other metrics that can help you better manage software production.

This chapter covers the following topics:

About DevOps Insight Dashboards

DevOps Insight dashboards let you focus on key metrics across your releases and CI/CD pipelines from code commit to deployments and more. ElectricFlow includes the following dashboards:

- Releases Dashboard on page 698
- Application Deployments Dashboard on page 705
- Microservice Deployments Dashboard on page 713
- Release Command Center Dashboard on page 720
- Continuous Integration Dashboard on page 751
- Code Commit Trends Dashboard on page 760

You can also create your own dashboards and reports based on the key metrics that are important to your organization. For details about creating or modifying dashboards, see Authoring DevOps Insight Dashboards. For details about creating or modifying reports, see Authoring DevOps Insight Reports on page 764.

Configuring the DevOps Insight Server

You use the **Administration > DevOps Insight Server > DevOps Insight Server Configuration** page in the Automation Platform to set up connectivity and authentication for the DevOps Insight server. The data that populates the dashboards is transmitted from the DevOps Insight server to the ElectricFlow server.

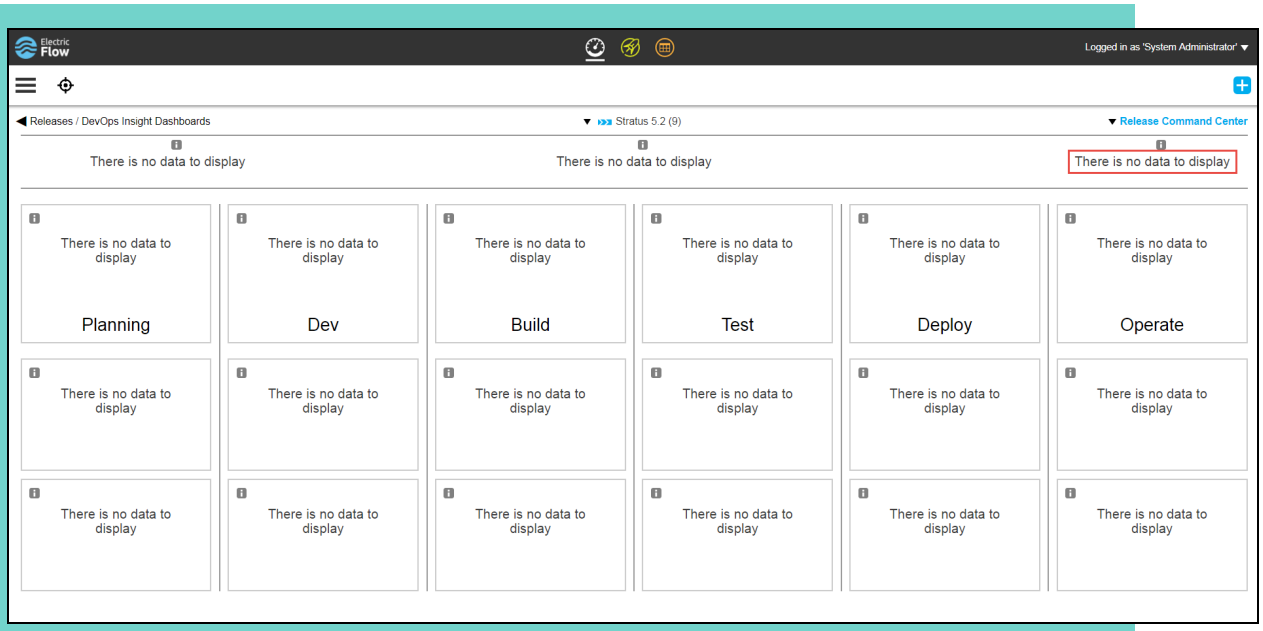
Setting Up DevOps Insight Server Connectivity and Authentication

Use the following settings to set up connectivity and authentication.

Checkbox or field	Description
Enable DevOps Insight	Specifies whether to enable DevOps Insight. This is enabled by default.
URL for Logstash service on the DevOps Insight Server	URL where Logstash receives data from the ElectricFlow server. This is required if DevOps Insight is enabled.
URL for Elasticsearch service on the DevOps Insight Server	URL where the ElectricFlow server retrieves data from Elasticsearch. This is required if DevOps Insight is enabled.
Authentication Credentials	User name and password for authenticating with the DevOps Insight server. The default user name is `reportuser`. Use the password that was defined during DevOps Insight installation.
Test Connection	Specifies whether to test the connection to the DevOps Insight server before you save the configuration. This is disabled by default.

The DevOps Insight server uses the Elasticsearch search engine and the Logstash data-collection and log-parsing engine to gather data from the ElectricFlow server. This data is used by the Application Deployments, Microservice Deployments, Releases, and Release Command Center dashboards.

Note:
Only the data that is created or updated in ElectricFlow after you configure DevOps Insight is synced with the DevOps Insight server. For example, for an existing release, the **Days to Delivery** field in the Release Command Center is blank until you update the release in some way after configuring the DevOps Insight server:



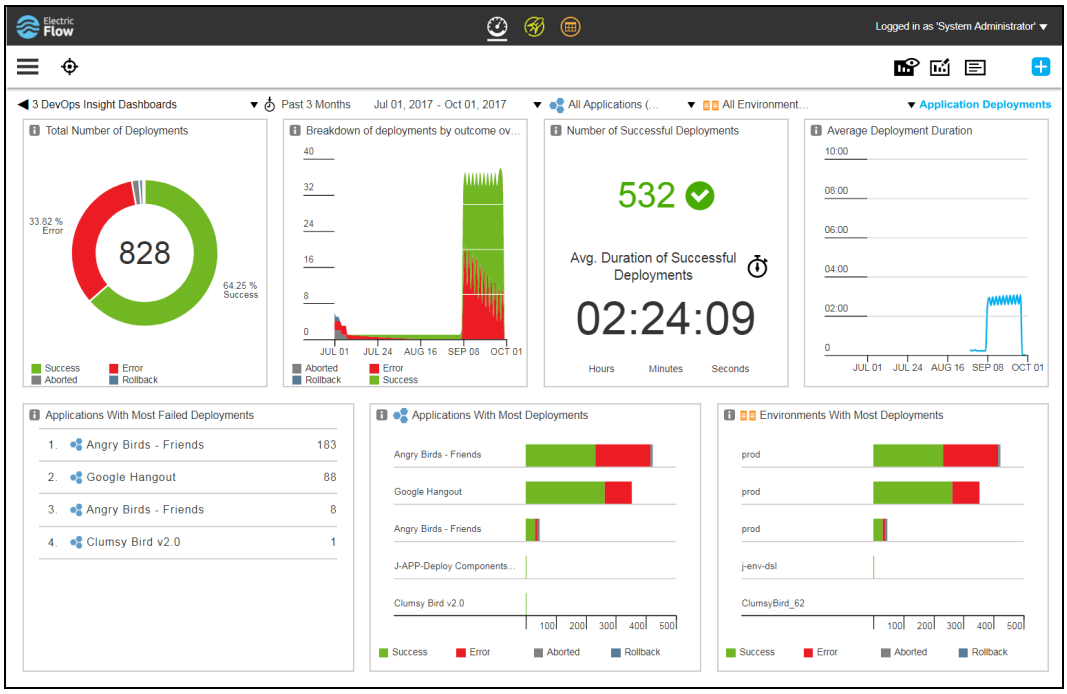
This restriction does not apply to existing data that ElectricFlow does not create or update (for example, data from plugins such as JIRA or HP ALM).

Ensuring that the DevOps Insight Server Is Installed

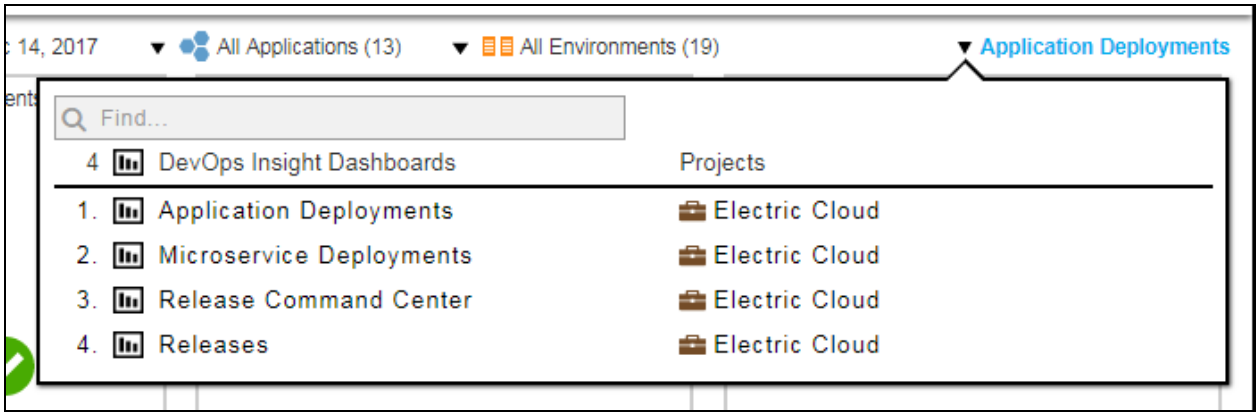
To ensure that your dashboards are populated with data, you must make sure that the DevOps Insight server is installed. For details, see the *ElectricFlow Installation Guide* at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html.

Viewing a DevOps Insight Dashboard

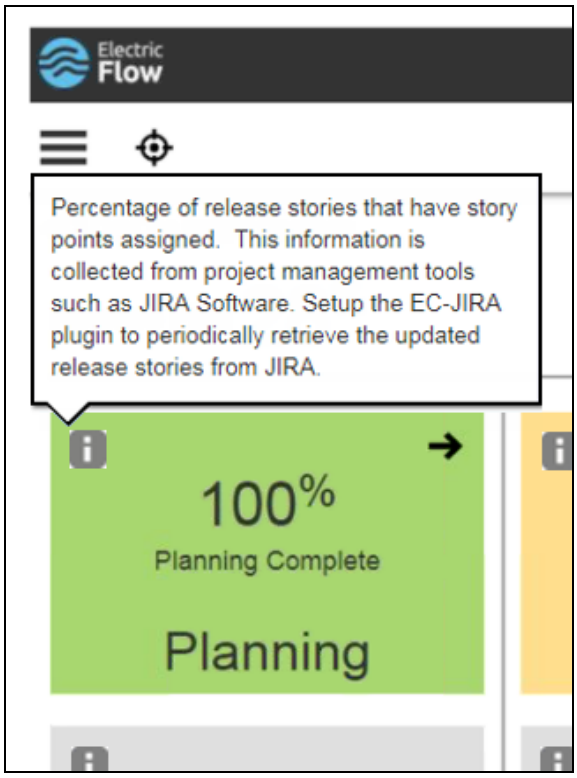
Click the  (DevOps Insight dashboards launch) button to open DevOps Insight. Following is an example of a dashboard that appears after you open DevOps Insight:



You can view a different dashboard by selecting it from the pulldown menu of available dashboards at the top right corner of the dashboards page:



Every cell in a dashboard contains an info ( ) button, which displays a tooltip that describes the data that appears in that cell. For dashboards that you create, you add the text for the tooltip that is displayed. The following example shows a clicked info button in the Release Command Center:



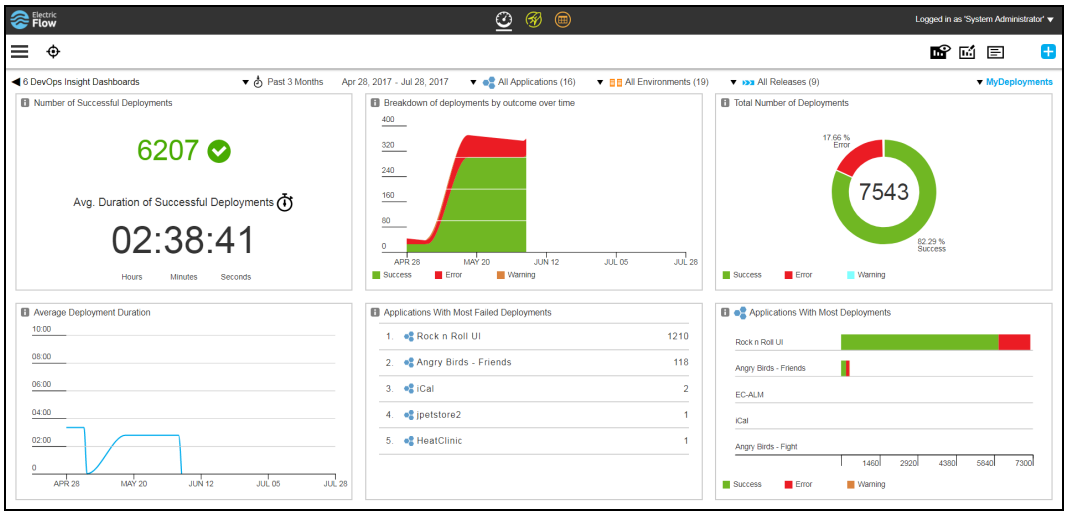
Viewing Modes

A dashboard has three viewing modes. You can switch among them by using the Graphical, edit, and DSL buttons (from left to right):

Following are examples of the three viewing modes.

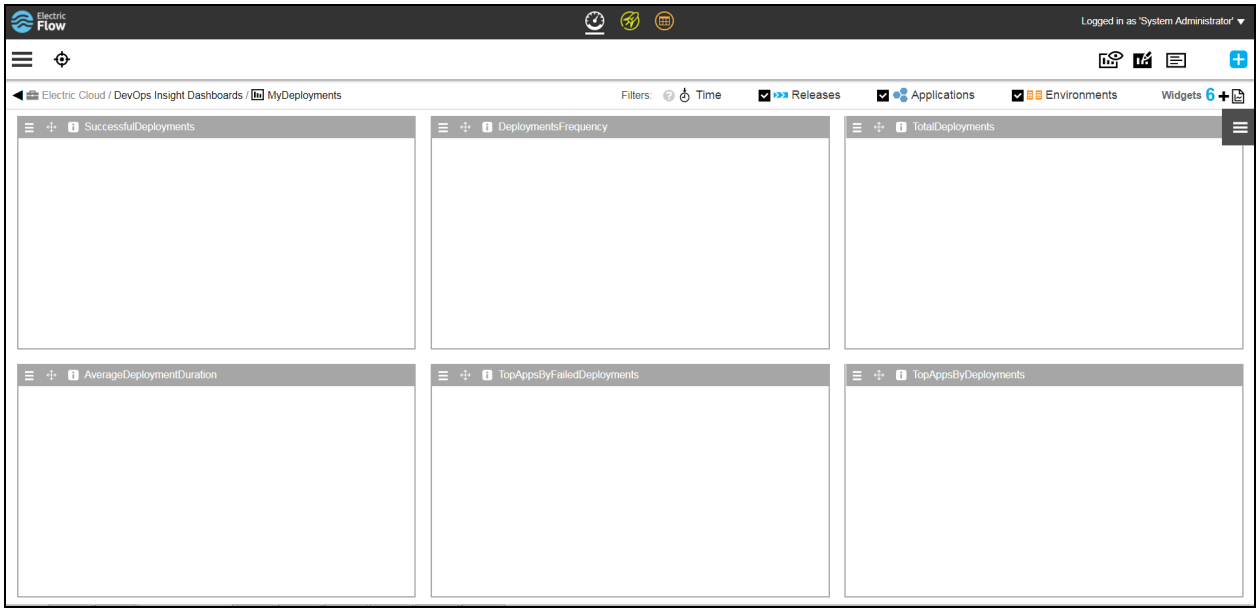
Graphical ("View Dashboard") Mode

This mode ( ) provides the data and visualizations for which the dashboard was created.



Edit ("Dashboard Editor") Mode

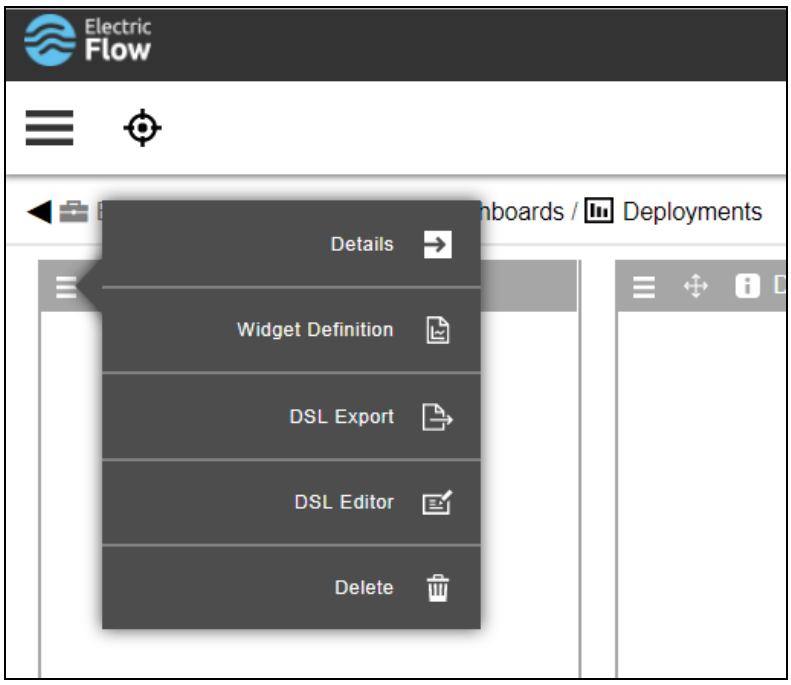
This mode ( ) lets you make changes to any dashboard that you create. It also lets you make layout changes to the Application Deployments dashboard, the Microservice Deployments dashboard, and the Releases dashboard. The following example shows edit mode for the Application Deployments dashboard:



Dashboards that you create are fully editable. The Deployments and Releases dashboards are editable in one way: You can move the widgets to change the layout of their corresponding cells. The Release Command Center dashboard is not editable in any way.

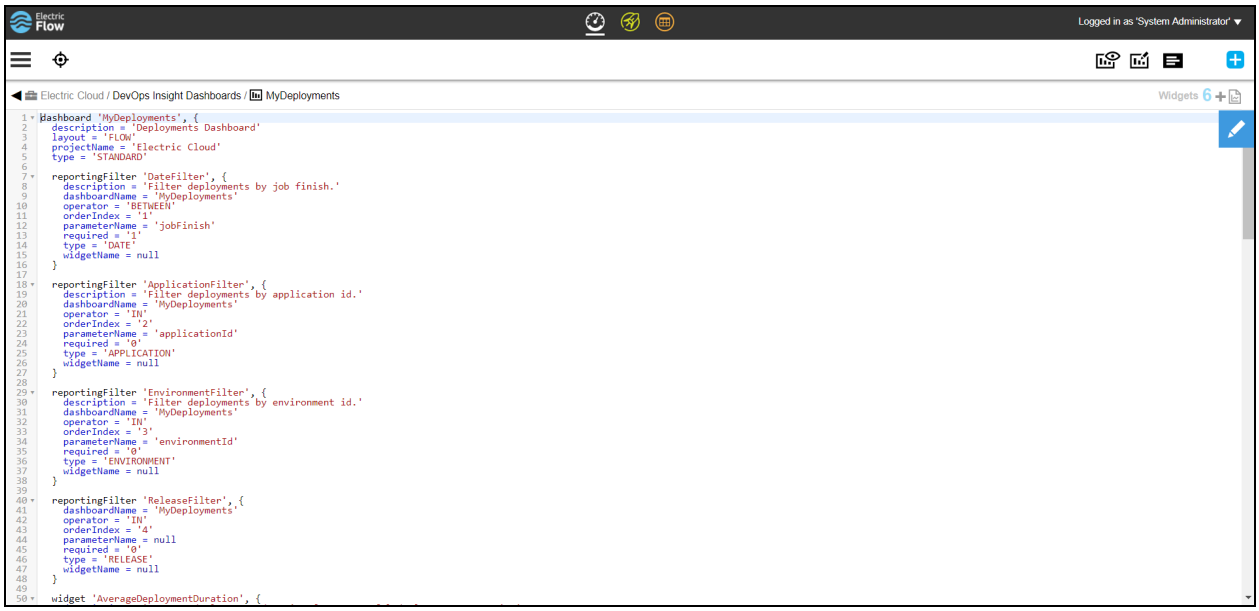
To move a widget, click the  (move widget) button and drag the widget to a new position.

In edit mode, a  menu appears in every widget. For example:



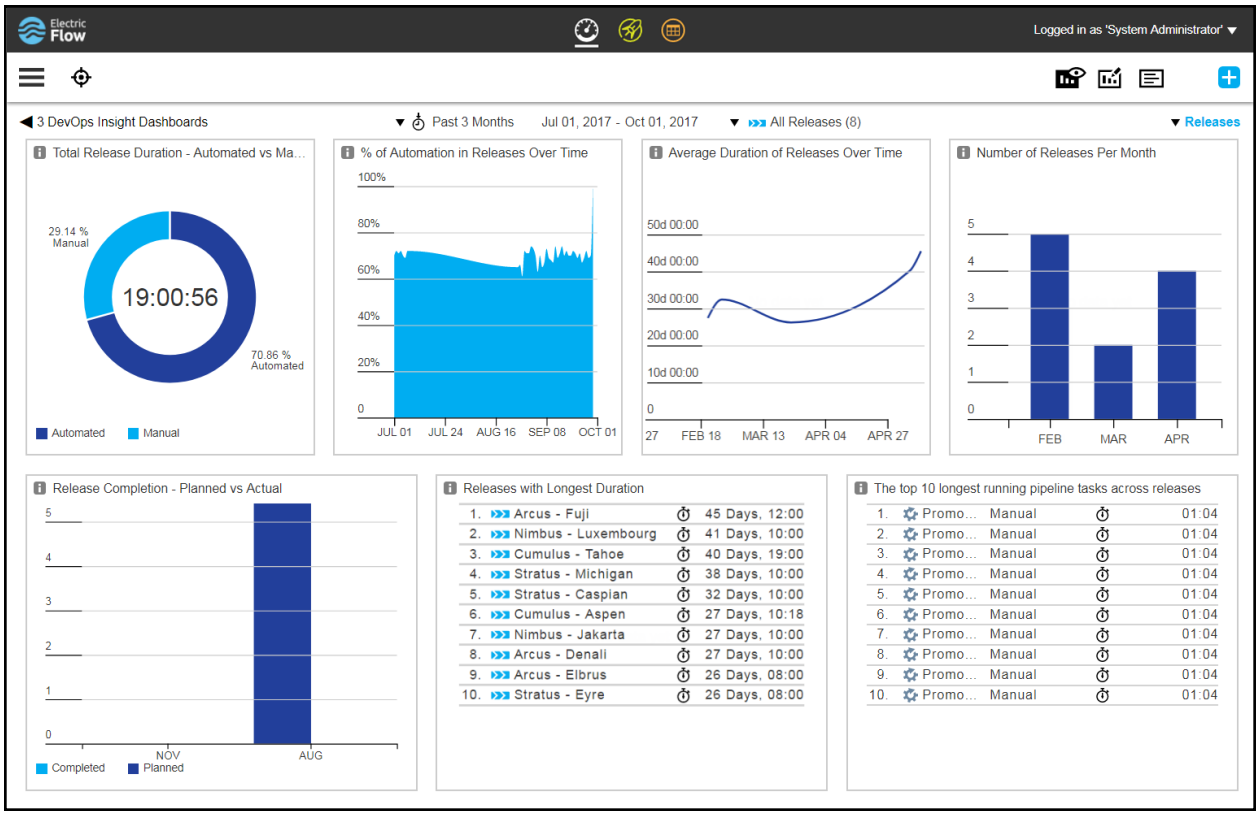
DSL ("DSL Editor") Mode

For bundled dashboards, this mode ( ) lets you view their Domain Specific Language (DSL). For dashboards that you create, this mode lets you edit their DSL.



Releases Dashboard

The Releases dashboard displays key metrics across releases that you can use to gain insight into your organization's throughput and to bubble up any potential bottlenecks in your release delivery pipelines so that they can be addressed quickly. The metrics displayed by the Releases dashboard can be filtered by date, project, releases, and tags via the drop-down menus at the top of the dashboard.

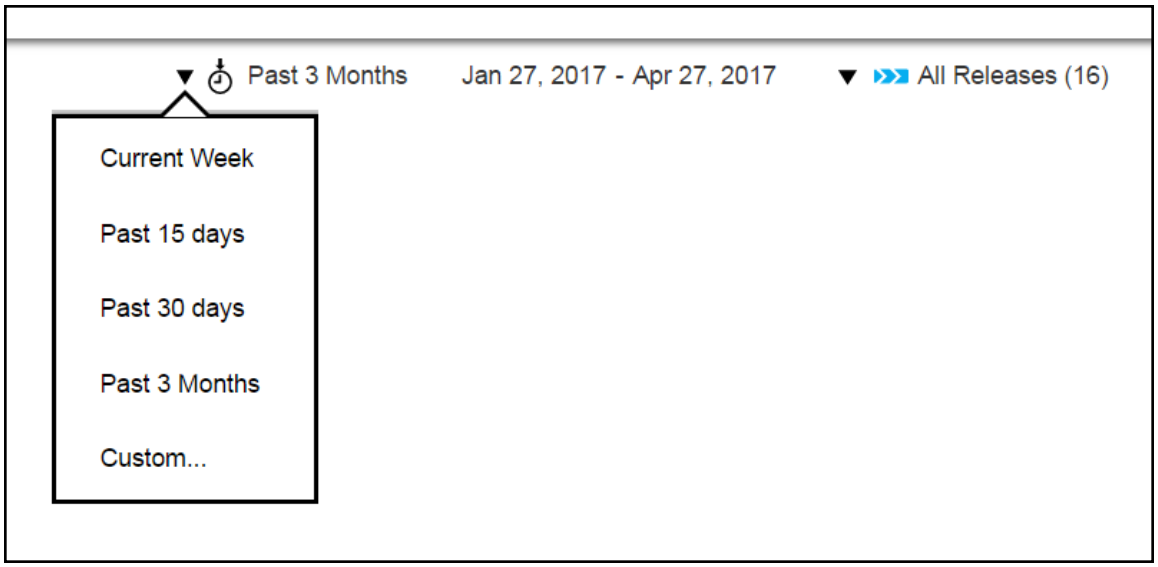


Filters

This dashboard provides drop-down menus that let you filter by time period, project, release, and tags:

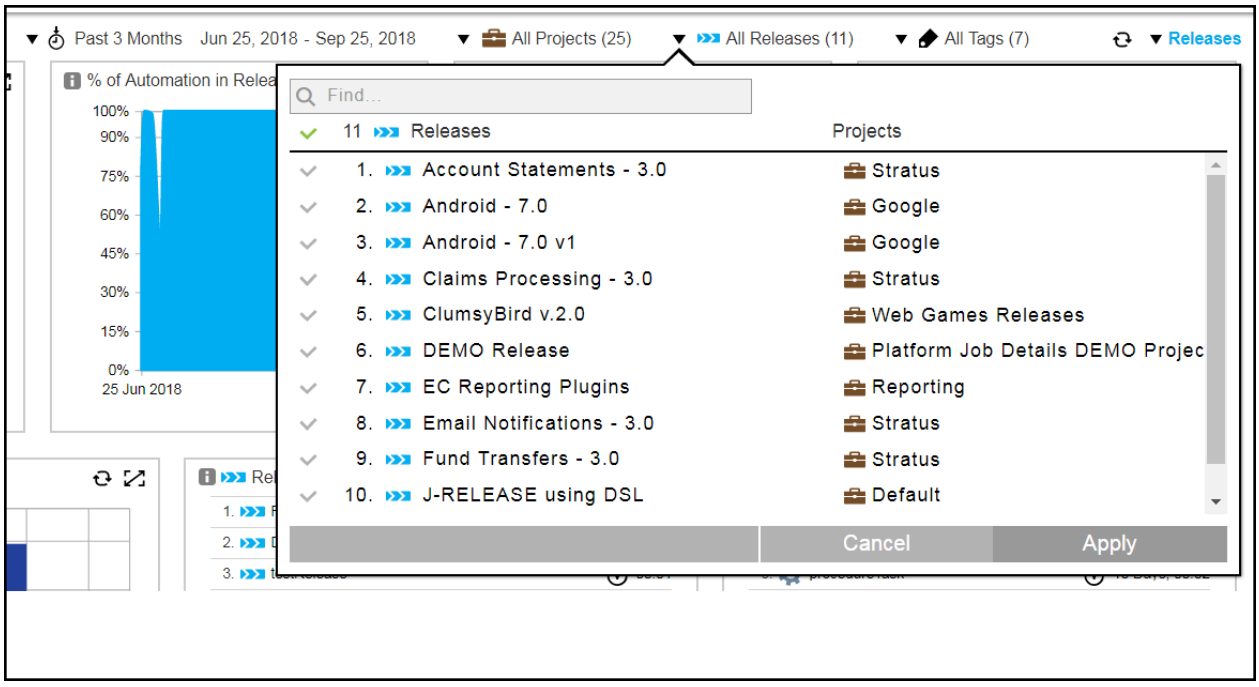
Time Filter

By default, release metrics for the past three months are displayed. You can change the date range by selecting the drop-down menu on the top of the dashboard. In addition to the preset ranges such as Current Week and Past 15 days, you can also specify a custom date range to use by using the **Custom...** option.



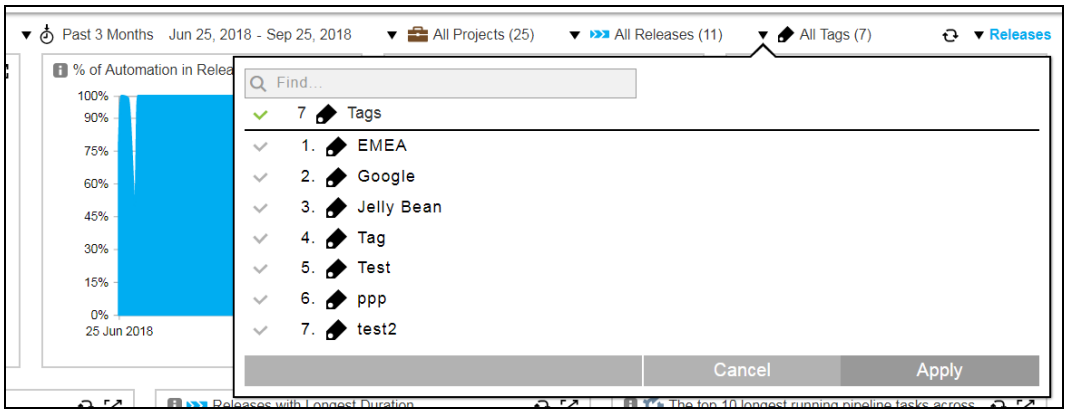
Releases Filter

By default, metrics for all releases appear. You can choose specific releases by selecting the drop-down menu for releases at the top of the dashboard:



Tags Filter

Filters the builds based on the tags marked on the releases. By default, release metrics for all builds appear. You can choose specific releases marked with specific tags by selecting the drop-down menu for tags at the top of the dashboard. The release metrics are filtered based on those tags.

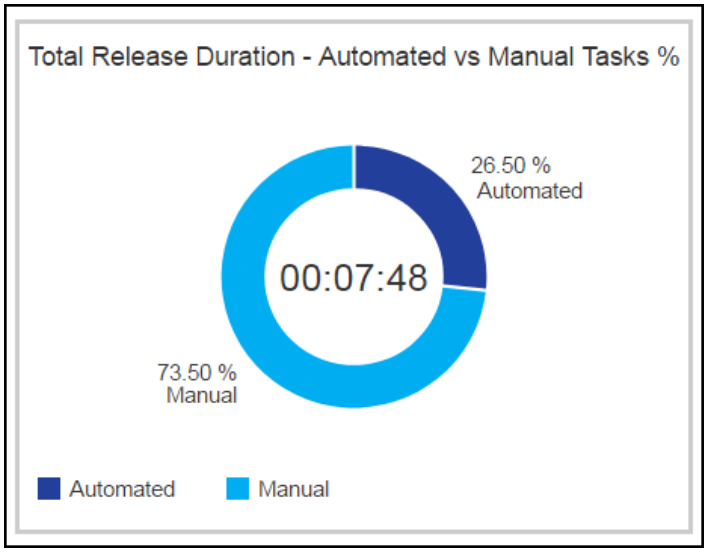


Examples are deployment tags such as *prod* or *UAT* and feature tags from JIRA such as *webapp*. For more information about tags, see the "Object Tags" section in the "Introduction to ElectricFlow" chapter.

Visualizations

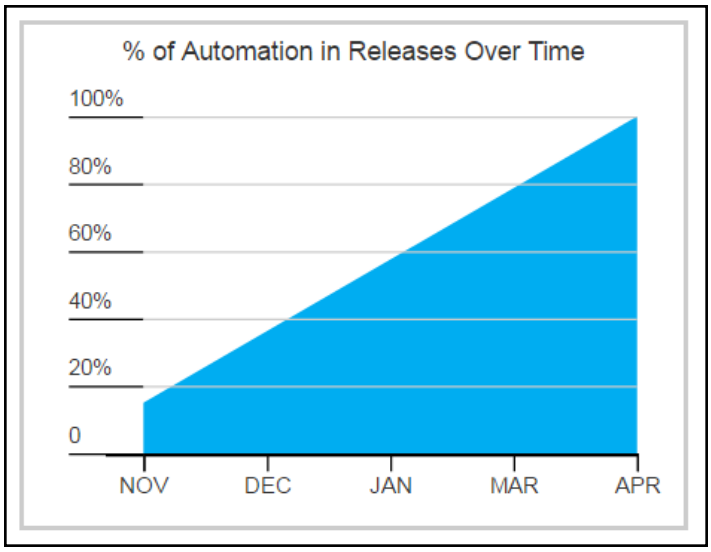
Total Release Duration - Automated vs Manual Tasks %

This chart shows the total time spent on releases (the number in the center) and what percentage of that was spent on automated tasks versus manual tasks. This clearly points to your release efficiency by showing how much of the release time is being spent on manual tasks and the time savings you would achieve by automating more release processes.



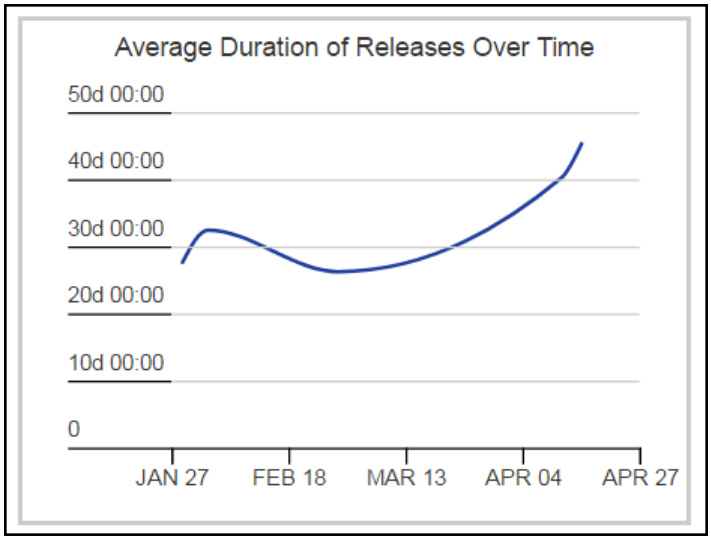
% of Automation in Releases Over Time

This chart shows the progress in release process automation over time. This tells you how your organization is improving its application delivery and deployment processes across releases over time by adding more automation and reducing manual actions wherever possible.



Average Duration of Releases Over Time

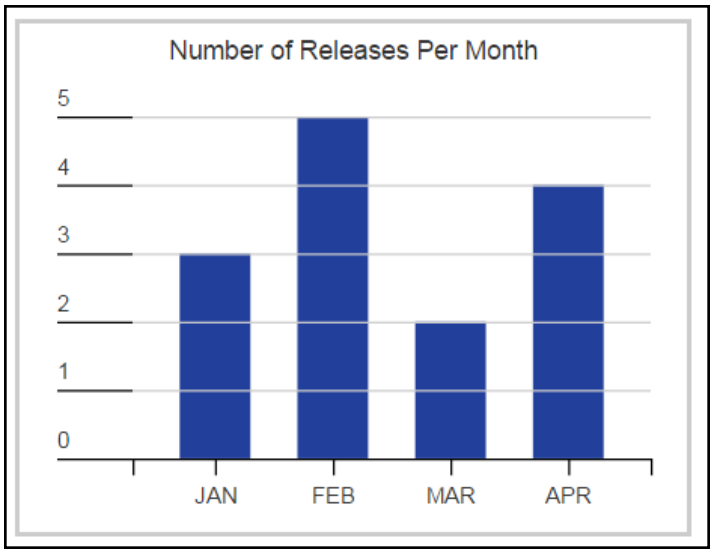
This chart shows the trend for how long different releases took over time. It lets you visualize your organization's throughput in terms of release efficiency over time. This helps you understand if you are spending less or more time on average for your releases.



You can drill down into the Average Duration of Releases Over Time cell by clicking and dragging to select a region. The Releases list page appears and shows the releases that completed in the selected region.

Number of Releases Per Month

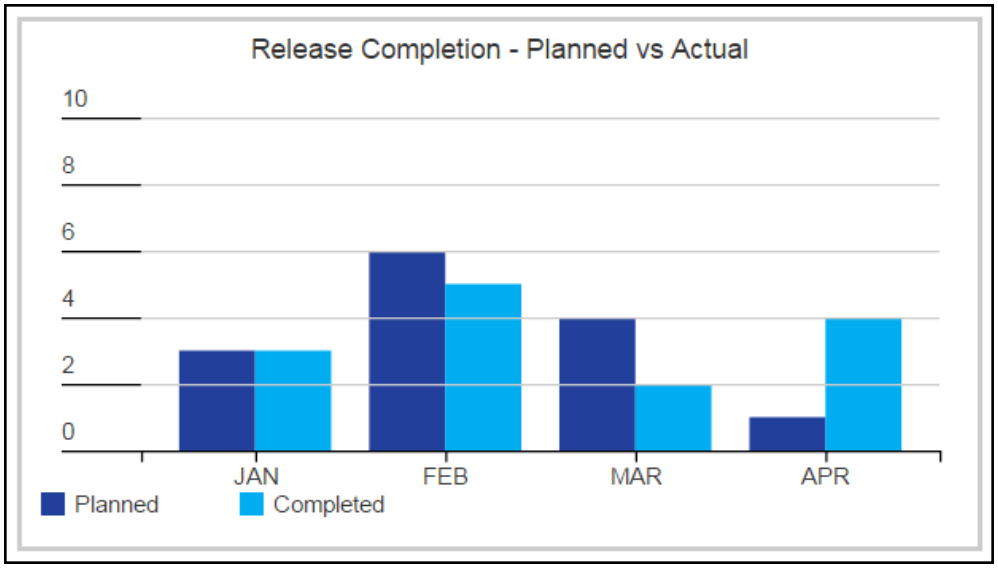
This chart shows the number of releases completed in the past several months, which lets you gauge the efficiency and throughput of your organization.



You can drill down into the Number of Releases Per Month cell by clicking a vertical bar. The Releases list page appears and shows the releases that completed in the month that you clicked.

Release Completion - Planned vs Actual

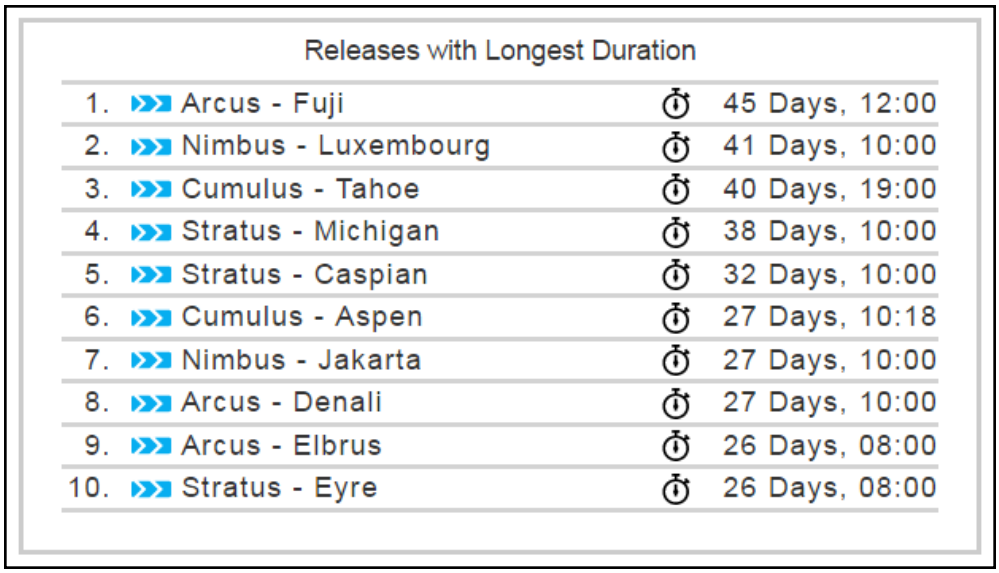
This chart compares the planned completion dates against their actual completion dates. This lets you visualize how well your organization adheres to the plan. In case there are deviations, this report can help show any patterns when the deviations occur.



You can drill down into the Release Completion - Planned vs Actual cell by clicking a vertical bar. The Releases list page appears and shows the releases that are either planned to be completed in that month or have completed in that month.

Releases with Longest Duration

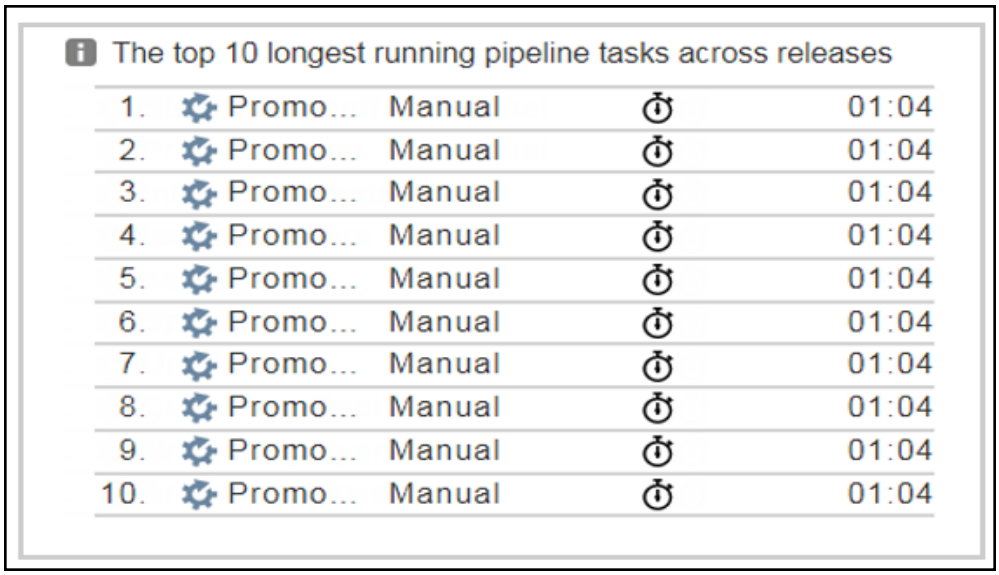
This report lists the top ten releases that took the most time to complete.



You can drill down into the Releases with Longest Duration cell by clicking a row for a release. The Releases list page appears and shows the selected release.

Top 10 Longest Running Pipeline Tasks across Releases

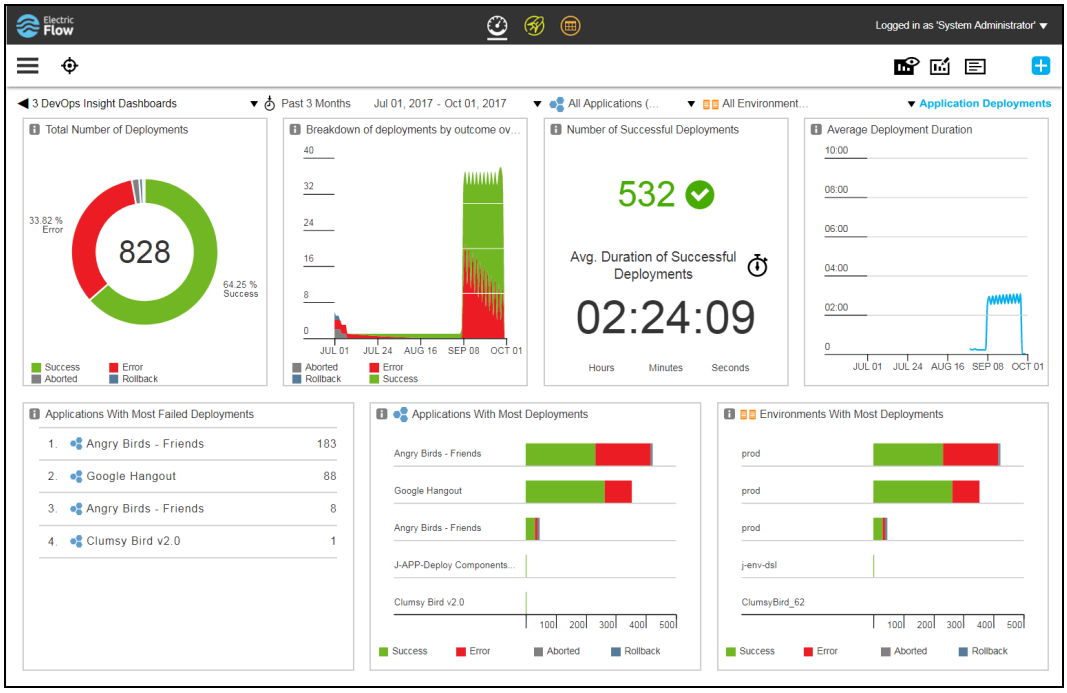
This report lists the top ten pipeline tasks that were run in the context of releases that took the most time to complete. This report helps find certain tasks (especially manual tasks) that need attention to improve your ability to deliver faster and on time.



You can drill down into the Top 10 Longest Running Pipeline Tasks Across Releases cell by clicking a pipeline task. The Pipeline Run Details page appears and shows an expanded view of the pipeline run that includes the task.

Application Deployments Dashboard

The Application Deployments dashboard displays key metrics for application deployments that can be used to gain insight into your organization's throughput and to bubble up any potential bottlenecks in your application deployment processes so that they can be addressed quickly. The Application Deployments dashboard provides indicators to measure agility of development and reliability of application deployments. This dashboard does not include microservice deployments.

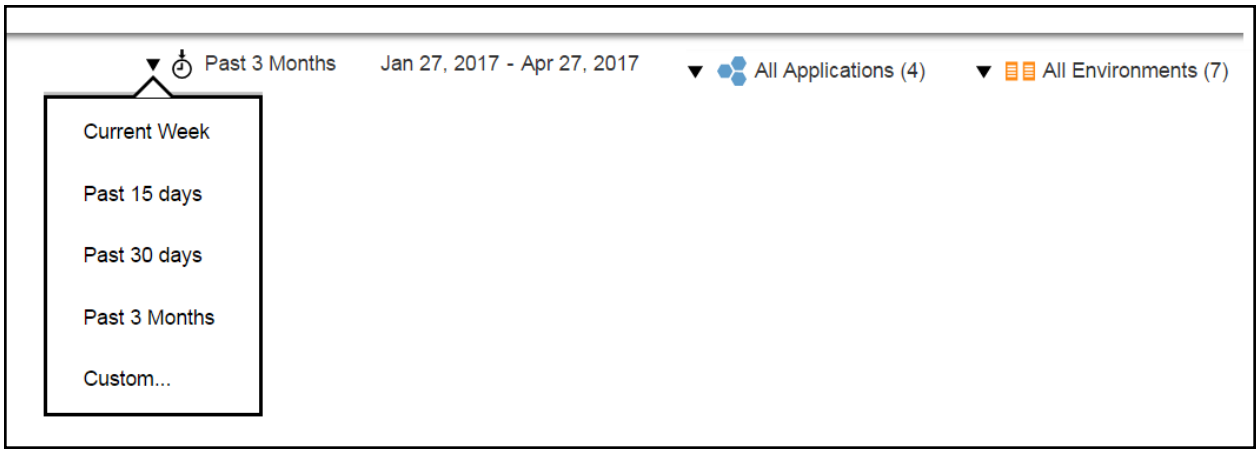


Filters

This dashboard provides drop-down menus that let you filter by time period, application, and environment:

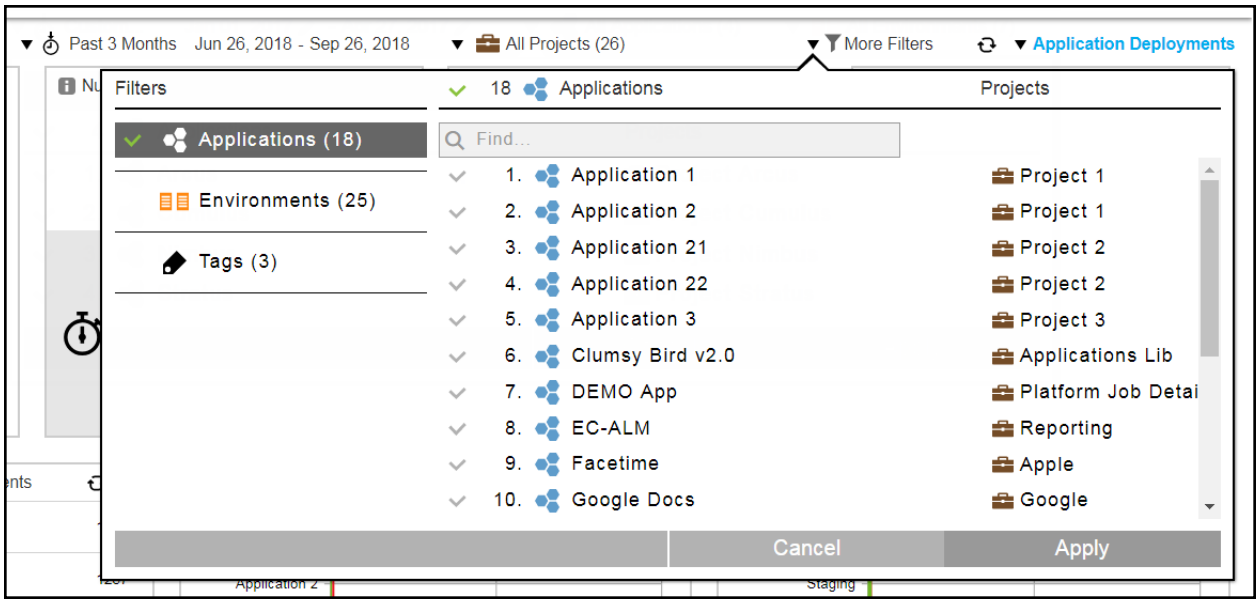
Time Filter

By default, the application deployment metrics for the past three months appear. You can change the date range by selecting the drop-down menu on the top of the dashboard. In addition to the preset ranges such as Current Week and Past 15 days, you can also specify a custom date range by using the **Custom...** option.



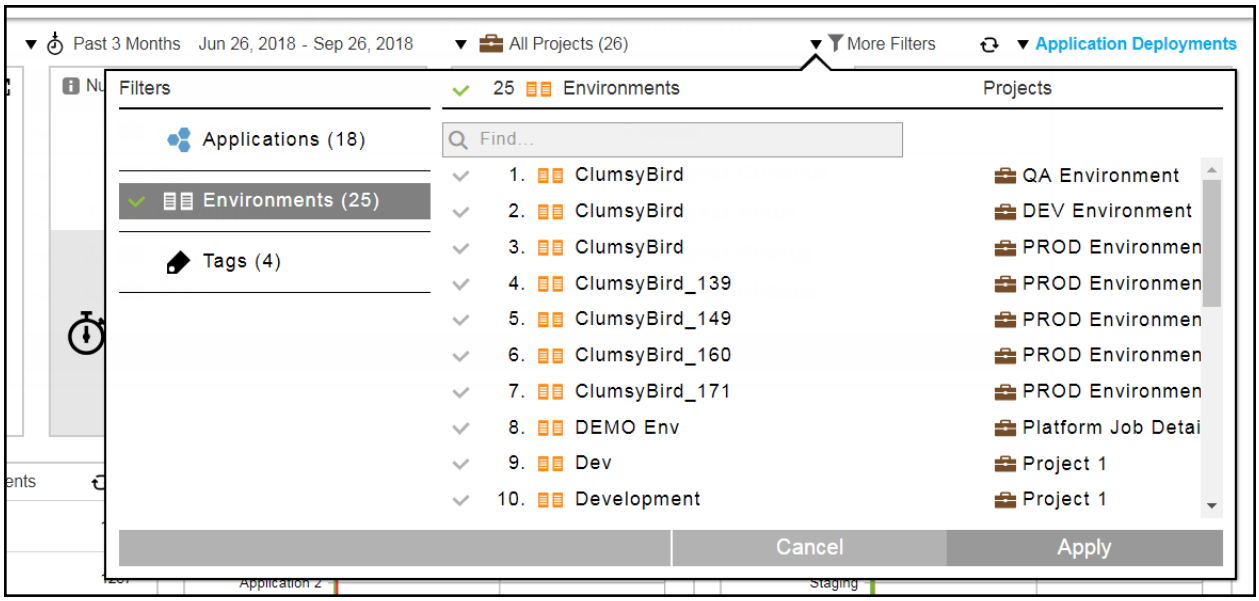
Applications Filter

By default, deployment metrics for all applications appear. You can choose specific applications by selecting the drop-down menu for applications at the top of the dashboard. The deployment metrics are filtered based on those applications.



Environments Filter

By default, application deployment metrics for all environments appear. You can choose specific environments by selecting the drop-down menu for environments at the top of the dashboard. Then the application deployment metrics will be filtered based on the selected environments that were used for application deployments.



Tags Filter

Filters the builds based on the tags marked on the builds. For ElectricFlow builds, this is the tags set on the build job and tags set on the build procedure, which are passed on to the build job.

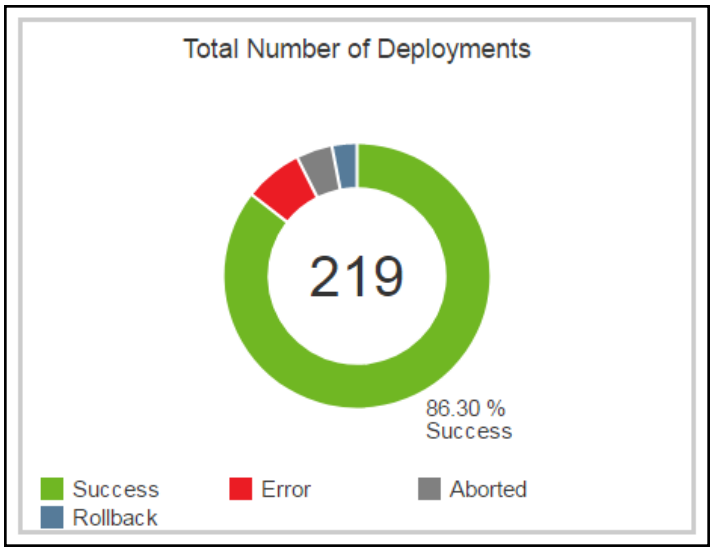
By default, build metrics for all builds appear. You can choose specific builds marked with specific tags by selecting the drop-down menu for tags at the top of the dashboard. The build metrics are filtered based on those tags.

Examples are deployment tags such as *prod* or *UAT* and feature tags from JIRA such as *webapp*. For more information about tags, see the "Object Tags" section in the "Introduction to ElectricFlow" chapter.

Visualizations

Total Number of Deployments

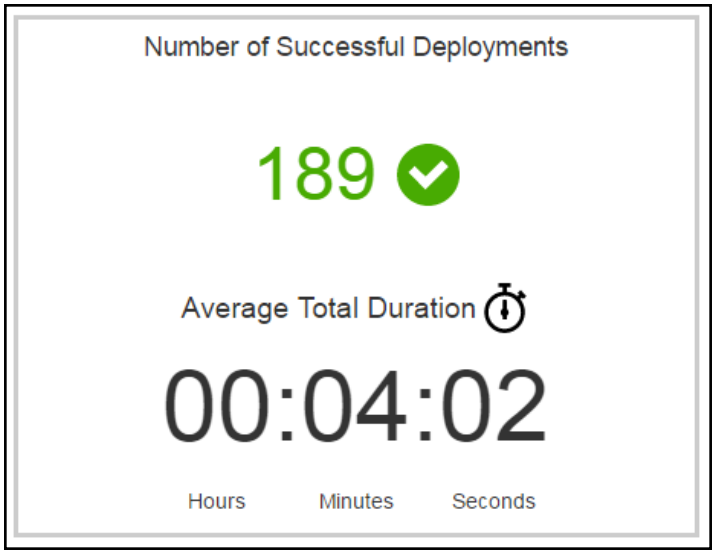
This chart provides a breakdown of all the application deployments by their outcome, which lets you measure the reliability of your application deployments.



You can drill down into the Total Number of Deployments cell by clicking a slice or clicking on the legend (useful if the slice is too small to click on). The application deployments for the outcome in the selected slice appear on the Application Deployments page. For example, if you click on Error, only the failed deployments are listed in the Application Deployments page. Examples of possible outcomes are Success, Error, Warning, and Aborted.

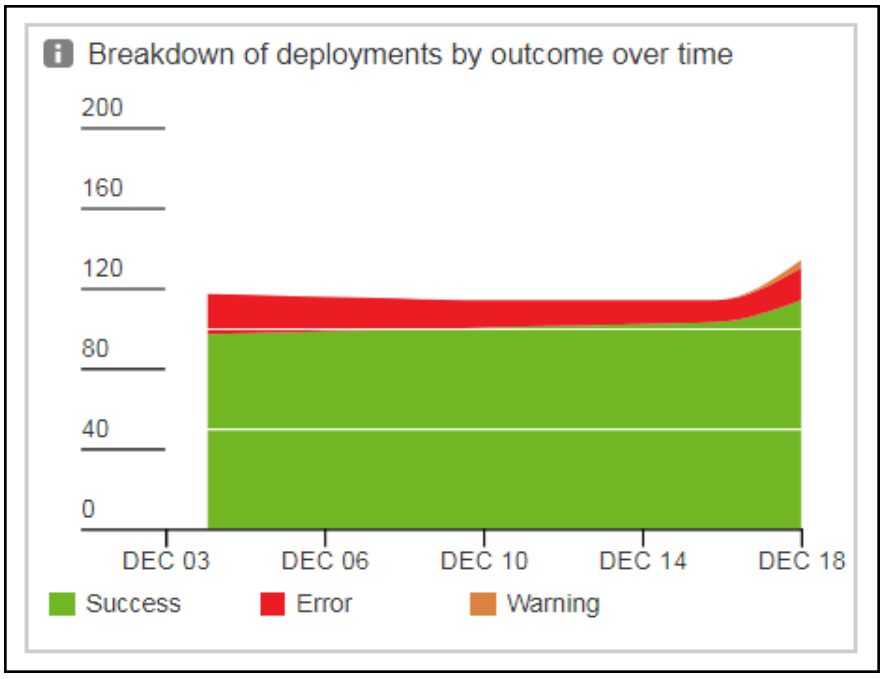
Number of Successful Deployments

This chart provides two powerful metrics as measures of your application deployment's health and efficiency: the total number of successful application deployments and their average deployment time.

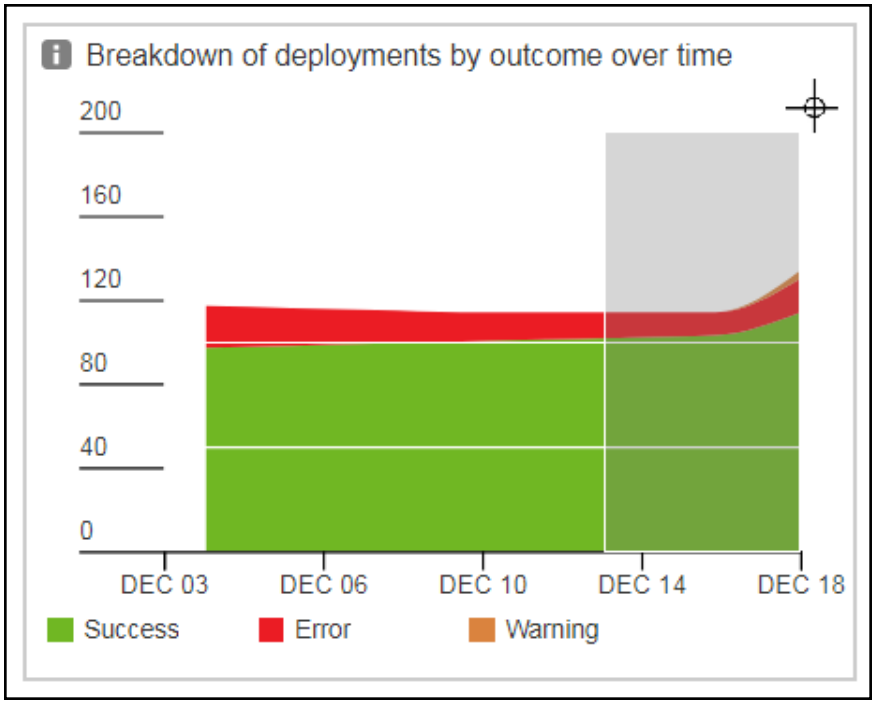


Breakdown of Deployments by Outcome Over Time

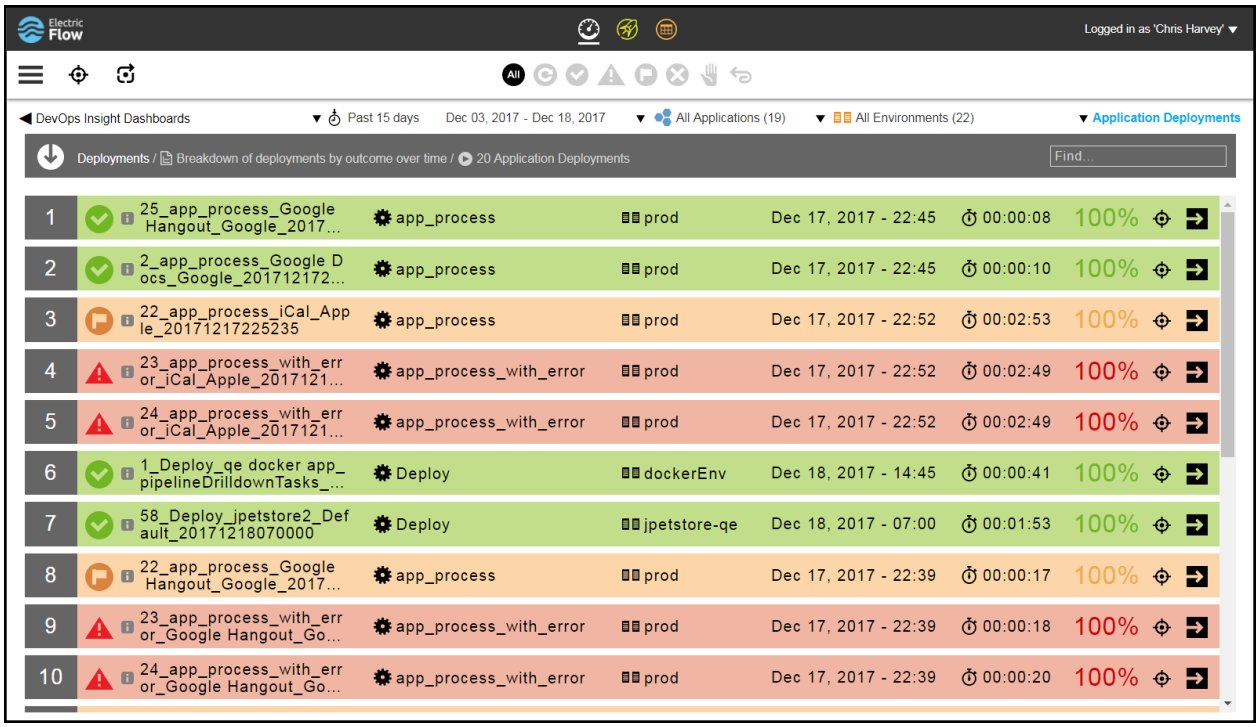
This chart shows you the application deployment trend over time. You can use it to gauge the efficiency of your organization in delivering applications and features over time.



You can drill down into the Breakdown of Deployments by Outcome Over Time cell by clicking and dragging to select a date range. For example:

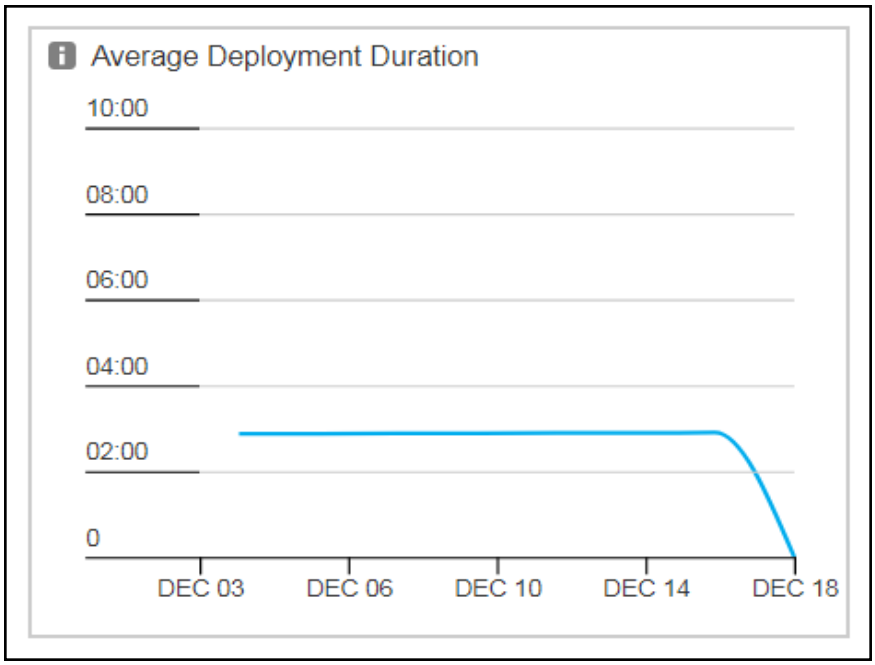


The application deployments for the selected date range appear on the Application Deployments page. For example:

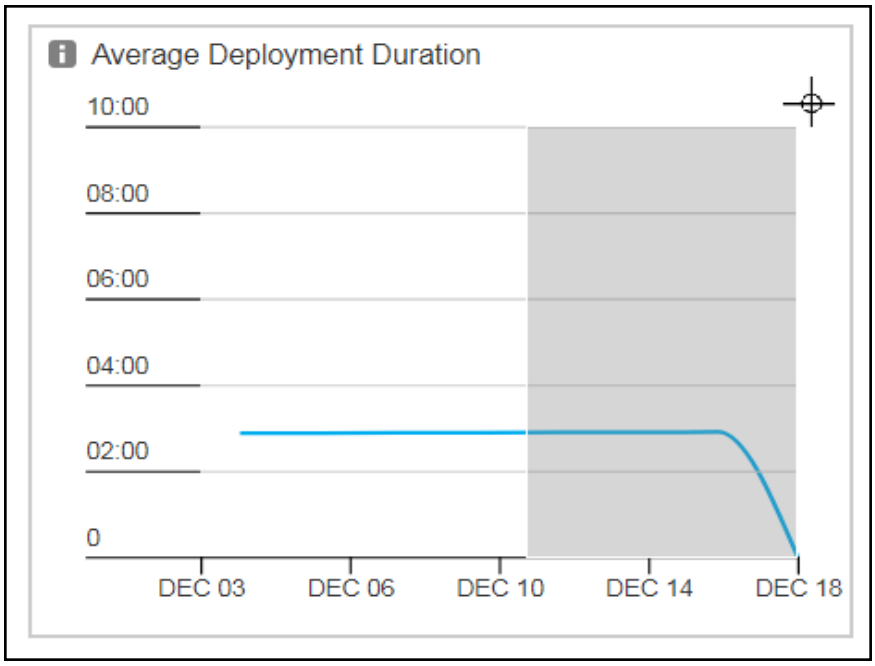


Average Deployment Duration

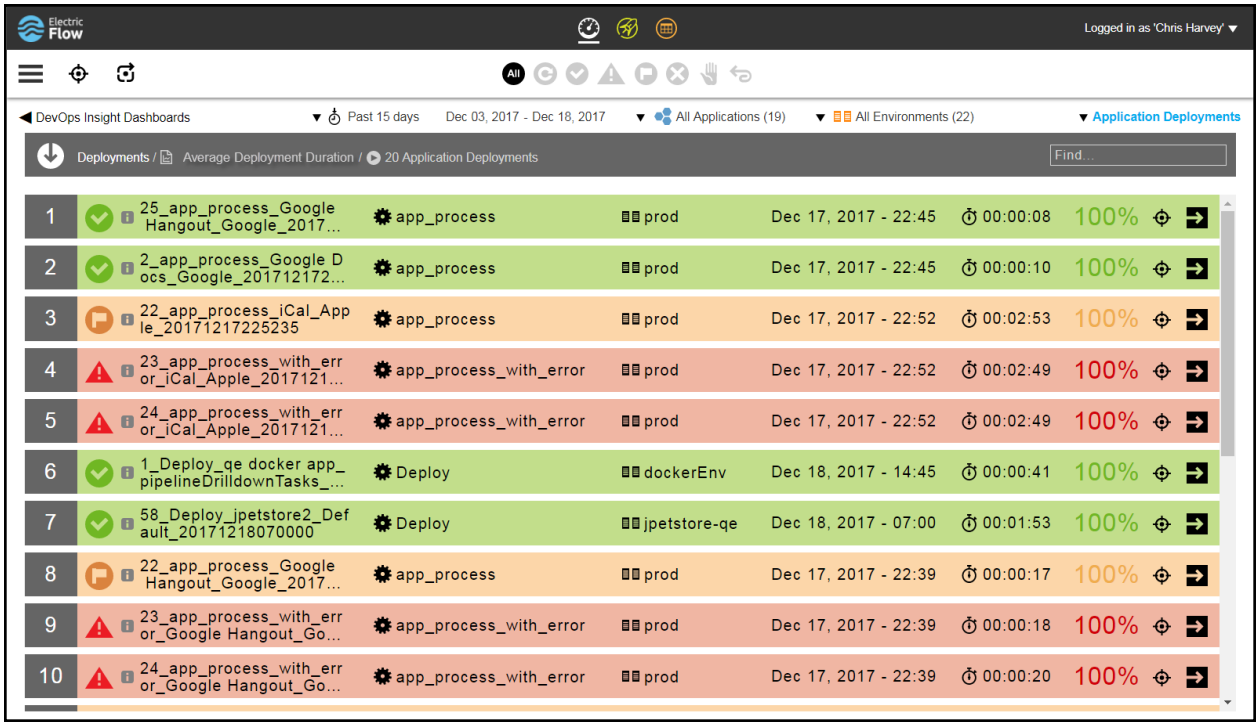
This chart shows the average duration of successful application deployments over time. This lets you see whether your organization's application deployment efficiency is improving over time.



You can drill down into the Average Deployment Duration cell by clicking and dragging to select a date range. For example:

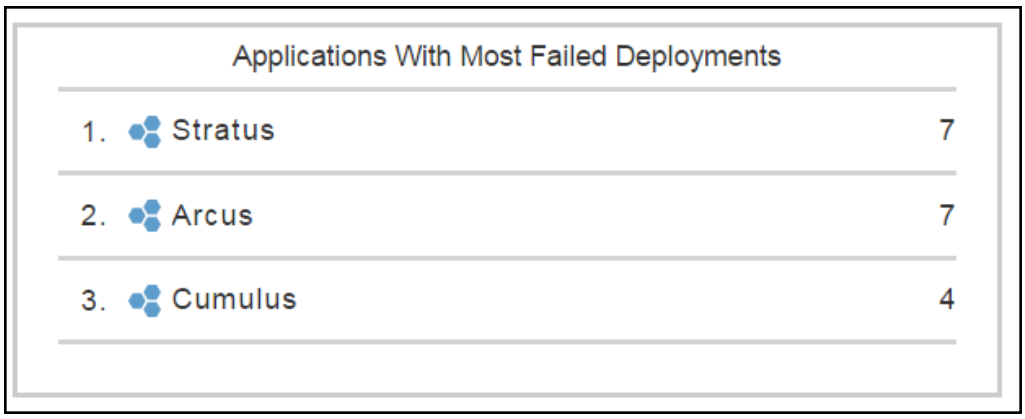


The application deployments for the selected date range appear on the Application Deployments page. For example:



Applications With Most Failed Deployments

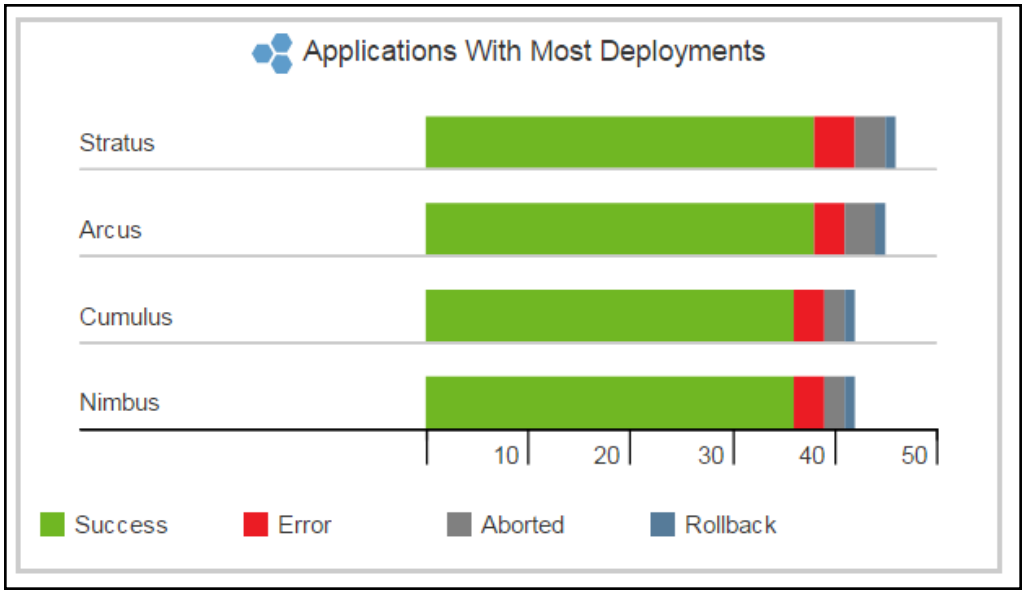
This report lists the applications with the largest number of failed deployments. The list is limited to five applications, which lets you focus on application teams that might need the most help.



You can drill down into the Applications With Most Failed Deployments cell by clicking an application. The Application Deployments page shows the failed deployments for the selected application.

Applications With Most Deployments

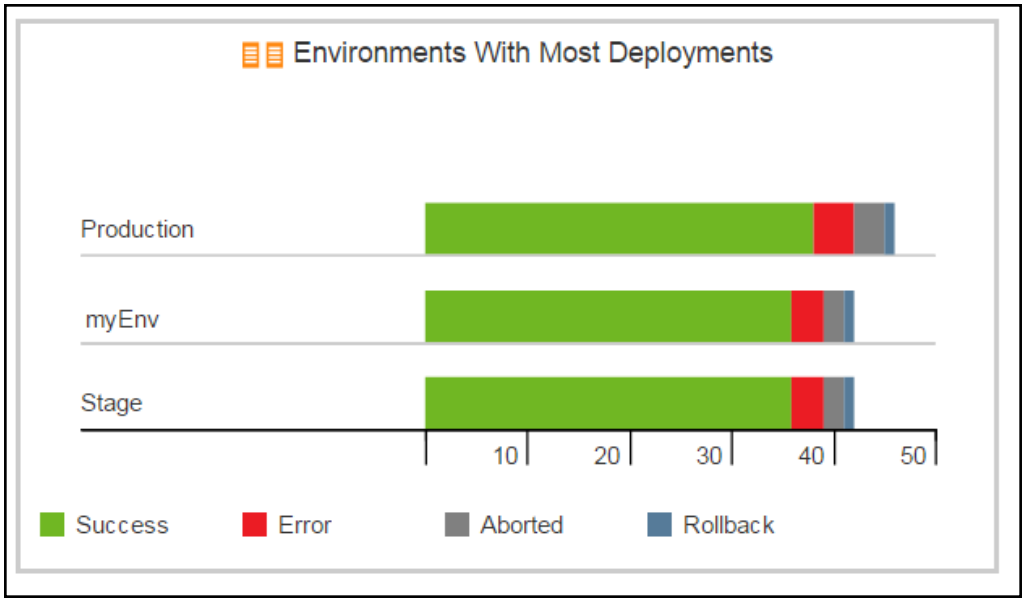
This chart shows the top applications with the most deployments with a breakdown by deployment outcome. The information in this report along with the earlier "Applications With Most Failed Deployments" report lets you gain insight into the productivity of different application teams along with the reliability of delivery.



You can drill down into the Applications With Most Deployments cell by clicking an application. The Application Deployments page shows the deployment breakdown based on the selected application and deployment outcome.

Environments With Most Deployments

This chart shows the environments with the most application deployments with a breakdown by deployment outcome. This presents information similar to the "Applications With Most Deployments" report except it uses the application deployment environments dimension.



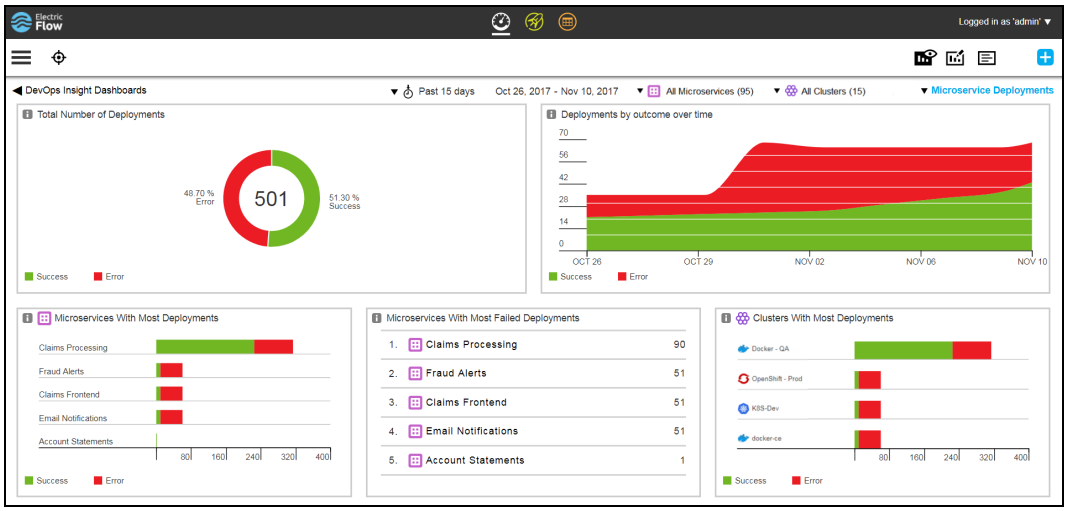
You can drill down into the Environments With Most Deployments cell by clicking an environment. The Application Deployments page shows the deployments based on the selected application and deployment outcome.

Microservice Deployments Dashboard

The Microservice Deployments dashboard displays key metrics for microservices deployed to different container orchestration platforms. These metrics can be used to gain insight into your organization's throughput and to bubble up any potential bottlenecks in your microservice deployment processes so that they can be addressed quickly.

The Microservice Deployments dashboard can be used to identify problem hotspots and measure the reliability of microservices running on your container clusters. This dashboard provides indicators to measure agility of development and reliability of microservice deployments.

This dashboard does not include application deployments. This dashboard includes metrics only for independent (project-level) microservices and does not include metrics for microservices within applications.

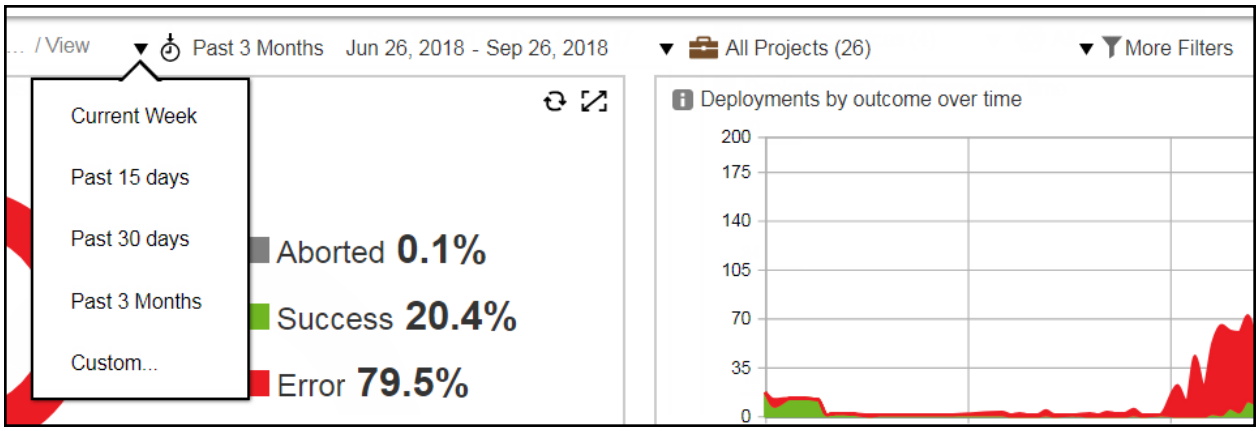


Filters

This dashboard provides drop-down menus that let you filter by time period, microservice, cluster, and tags:

Time Filter

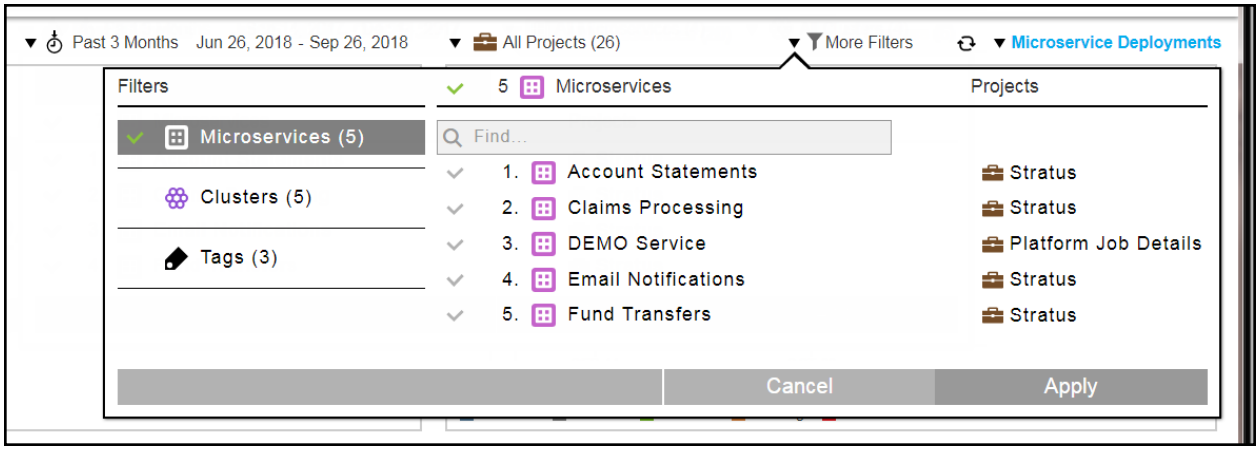
By default, the microservice deployment metrics for the past three months appear. You can change the date range by selecting the drop-down menu on the top of the dashboard. In addition to the preset ranges such as Current Week and Past 15 days, you can also specify a custom date range by using the **Custom...** option.



Microservices Filter

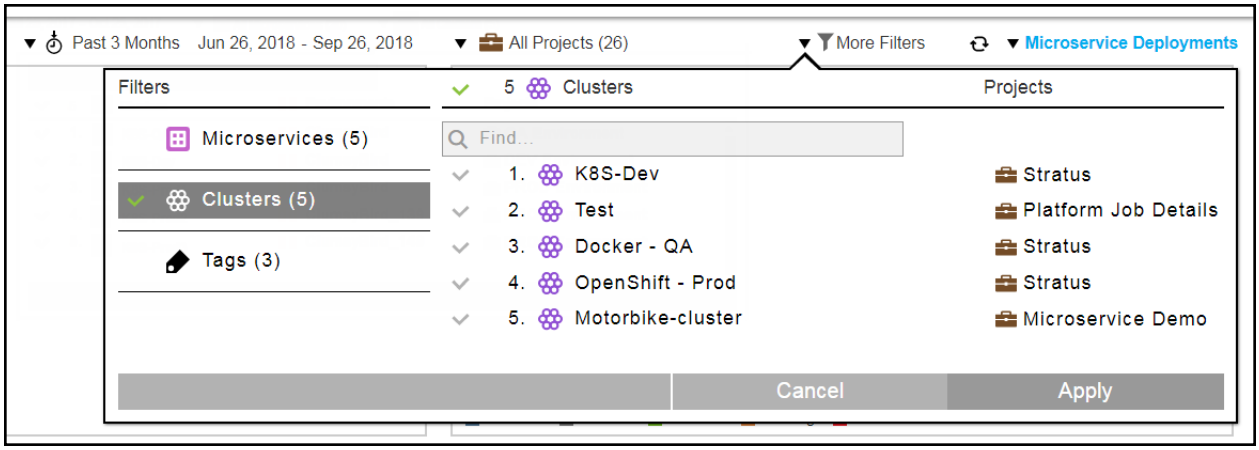
By default, microservice deployments metrics for all microservices appear. You can choose specific microservices by selecting the drop-down menu for microservices at the top of the dashboard. The

microservice deployments metrics are filtered based on those microservices.



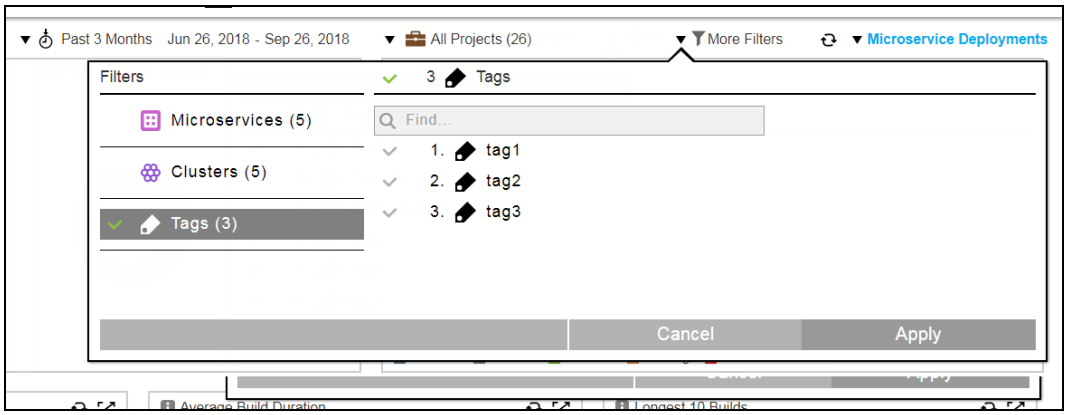
Clusters Filter

By default, microservice deployments metrics for all clusters appear. You can choose specific clusters by selecting the drop-down menu for clusters at the top of the dashboard. Then the microservice deployments metrics will be filtered based on the selected clusters that were used for microservice deployments.



Tags Filter

Filters the microservice deployments based on the tags marked on the builds. By default, build metrics for all microservice deployments appear. You can choose specific microservice deployments marked with specific tags by selecting the drop-down menu for tags at the top of the dashboard. The microservice deployments are filtered based on those tags.

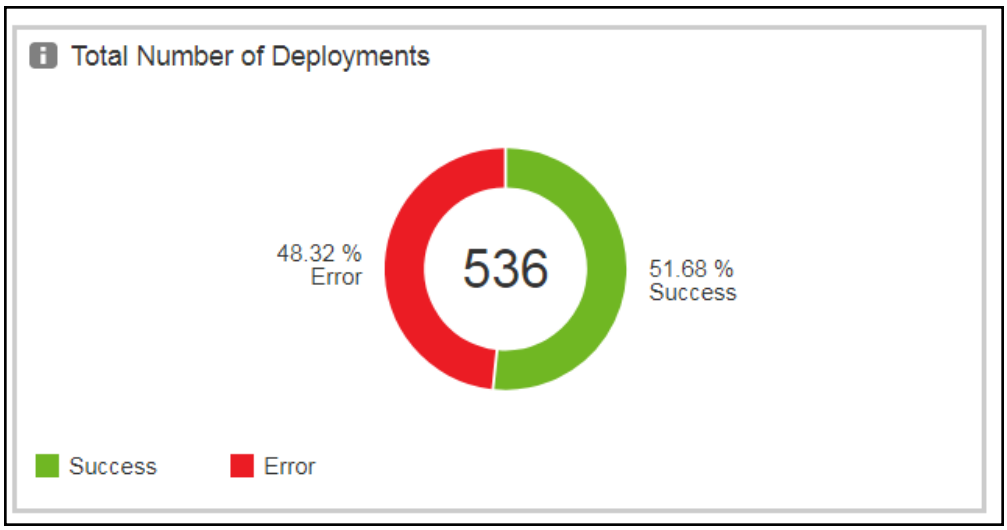


Examples are deployment tags such as *prod* or *UAT* and feature tags from JIRA such as *webapp*. For more information about tags, see the "Object Tags" section in the "Introduction to ElectricFlow" chapter.

Visualizations

Total Number of Deployments

This chart provides a breakdown of all microservice deployments by their outcome. This lets you measure the reliability of your microservice deployments.

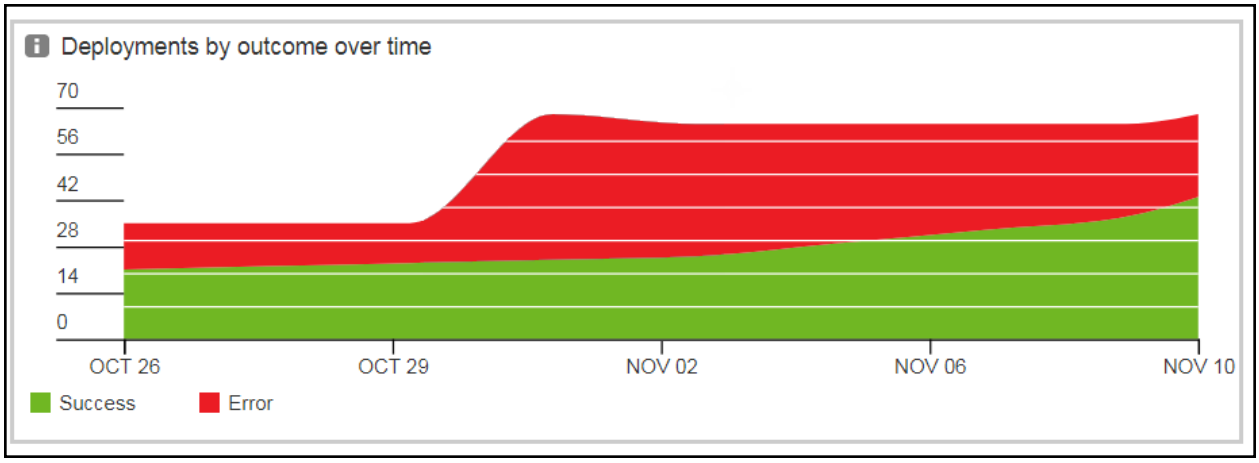


You can drill down into the Total Number of Deployments cell by clicking a slice or clicking on the legend (useful if the slice is too small to click on). The microservice deployments for the outcome in the selected slice appear on the Microservice Deployments page. For example, if you click on Error, only the failed deployments are listed in the Microservice Deployments page. Examples of possible outcomes are Success, Error, Warning, and Aborted.

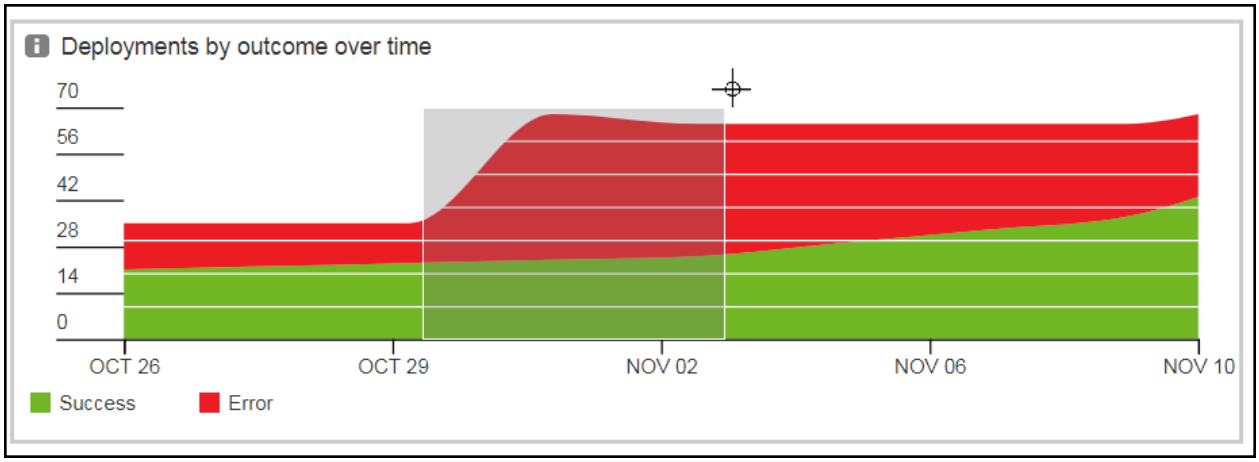
Deployments by Outcome Over Time

This chart provides two powerful metrics as measures of health and efficiency of your microservice deployments: the total number of successful deployments over time and total number of deployments

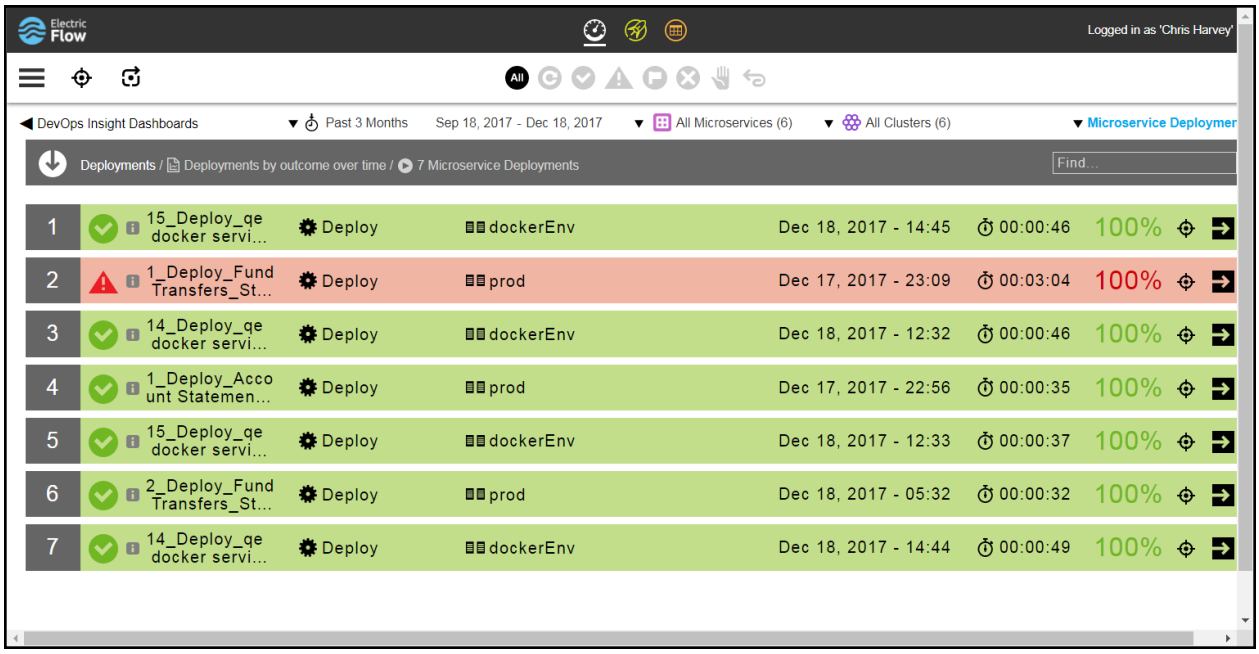
with errors over time.



You can drill down into the Deployments by Outcome Over Time cell by clicking and dragging to select a date range. For example:

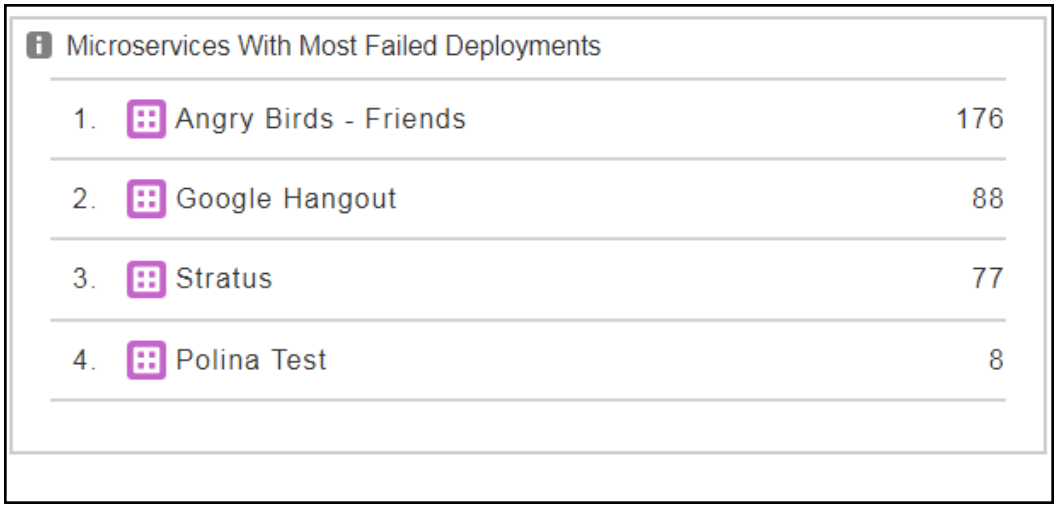


The microservice deployments for the selected date range appear on the Microservice Deployments page. For example:



Microservices With Most Failed Deployments

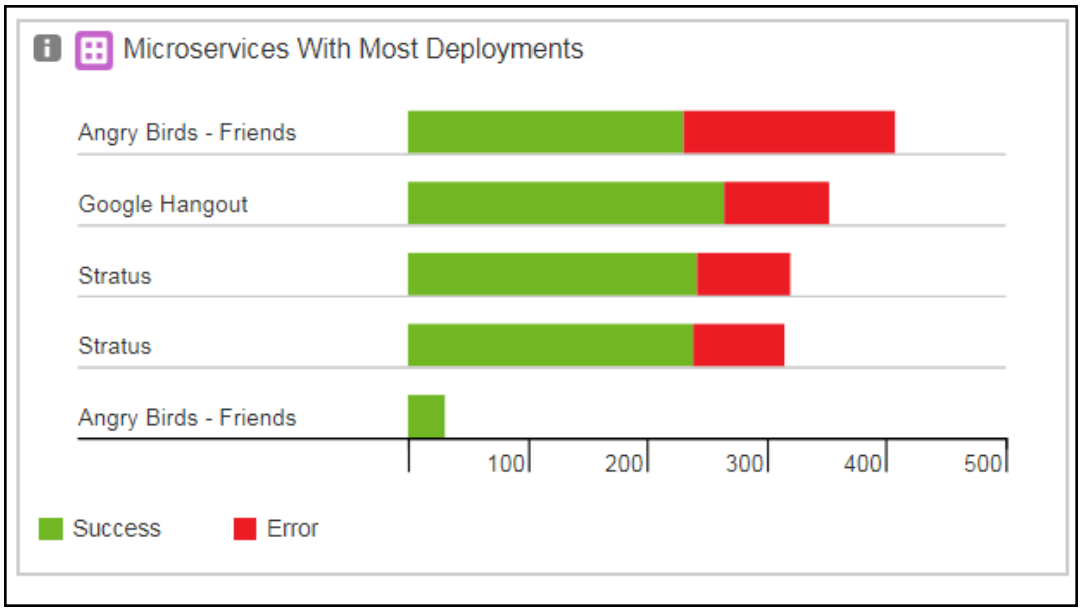
This report lists the microservices with the largest number of failed deployments. The list is limited to five microservices, which lets you focus on microservice teams that might need the most help.



You can drill down into the Microservices With Most Failed Deployments cell by clicking a microservice. The Microservice Deployments page shows the failed deployments for the selected microservice.

Microservices With Most Deployments

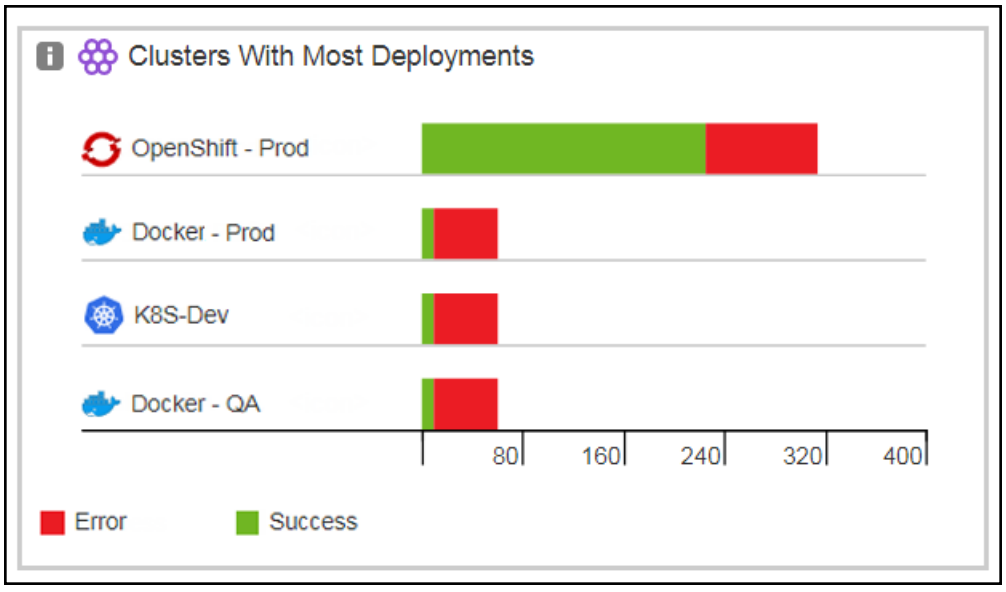
The information in this report along with the earlier report "Microservices With Most Failed Deployments" lets you gain insight into the productivity of different microservice teams along with the reliability of delivery. The list is limited to five microservices.



You can drill down into the Microservices With Most Deployments cell by clicking a microservice. The Microservice Deployments page shows the deployment breakdown based on the selected service and deployment outcome.

Clusters With Most Deployments

This chart shows the clusters with the most microservice deployments with a breakdown by deployment outcome. This presents information similar to the "Microservices With Most Deployments" report except it uses the microservice deployment clusters dimension.



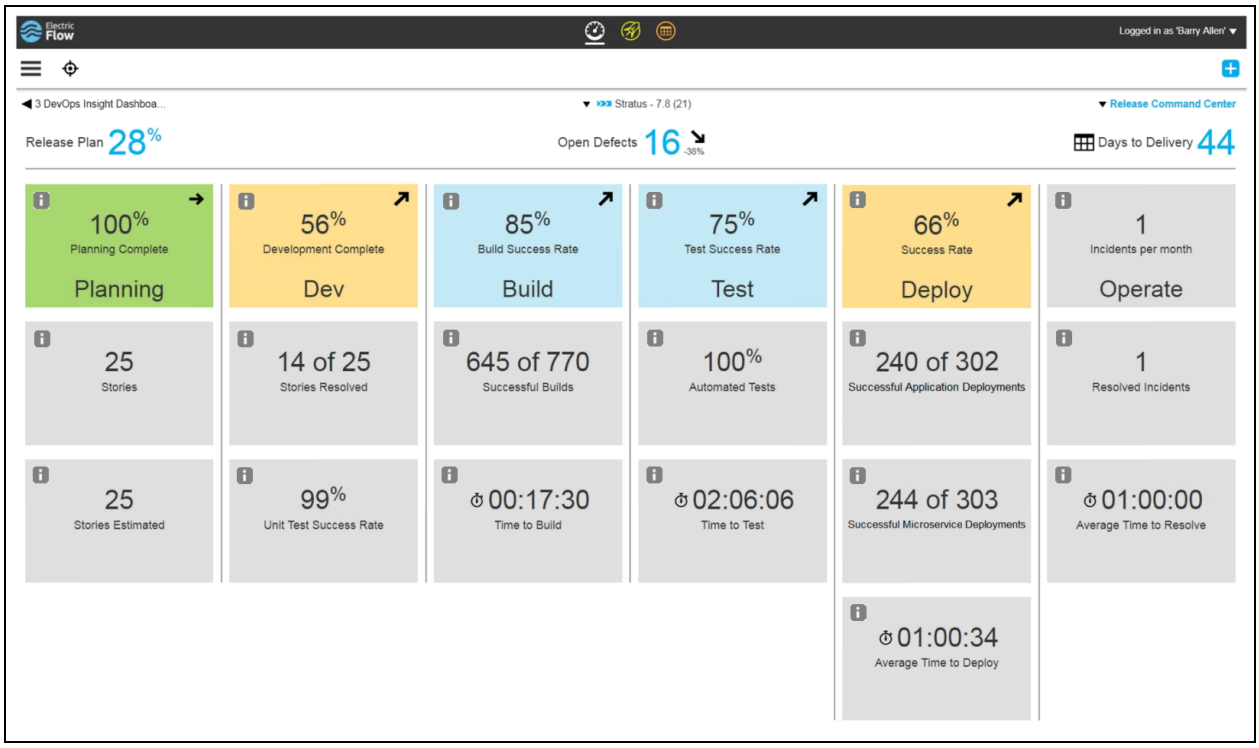
You can drill down into the Clusters With Most Deployments cell by clicking a cluster. The Microservice Deployments page shows the deployments based on the selected cluster and deployment outcome.

Release Command Center Dashboard

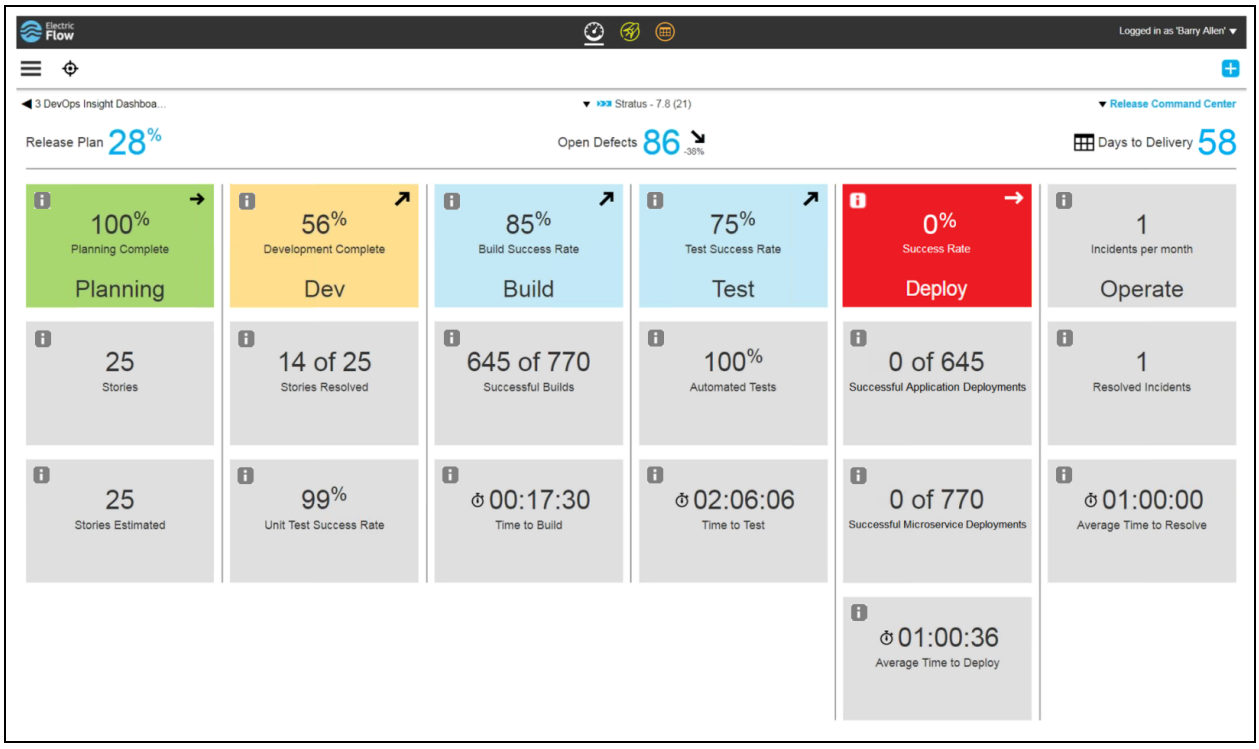
The Release Command Center dashboard provides a "birds-eye" view of a product or release across all of its pipeline phases. These phases include initial planning with tools (such as JIRA or Rally) to groom the backlog, target stories, and tasks.

This dashboard includes metrics for application deployments and independent (project-level) microservice deployments. This means that it does not include metrics for microservices within applications.

Following is a typical example of the dashboard. This example is for a release that is in progress with no major issues:

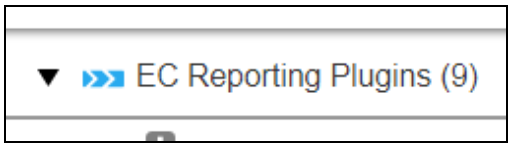


Following is a typical example for a release that is in progress but has issues with deployments. In this example, the **Deploy** cell is colored red, which indicates that too many deployments are failing in the last ten days for the release to progress satisfactorily:



Filters

This dashboard provides drop-down menus that let that lets you filter by release:



Release Plan Metric

This metric displays the percentage of release stories that are complete. The percentage is calculated using the latest feature data associated with the release for type "story". Records with status "closed" are considered as completed. This metric includes a trend indicator arrow.

Open Defects Metric

This metric displays the number of open defects for the release. The open defects are calculated using the latest defect data associated with the release. Records with resolution that is not set or empty are considered as open. This metric includes a trend indicator arrow.

Days to Delivery Metric

This metric displays the number of days remaining in the release.

Pipeline Phases and Cells

Each phase displays a particular category of metrics. This lets you estimate the work in terms of story points or time all the way through deploying and running in production. Data is presented at the following levels:

- Phase (such as Dev, Build, or Test)
- Phase level summary (measures in upward or downward trends)
- Phase level details

The color thresholds are defined for the phase summary widgets for the Planning, Dev, Build, Test , and Deploy phases. By default, the thresholds are the same based on the percentage value displayed for their corresponding widgets:

- Less than 50%: Red
- Between 50 and 75%: Yellow
- Between 75 and 90%: Blue
- 90% or greater: Green

Below is a list of each phase with a description of each cell in that phase.

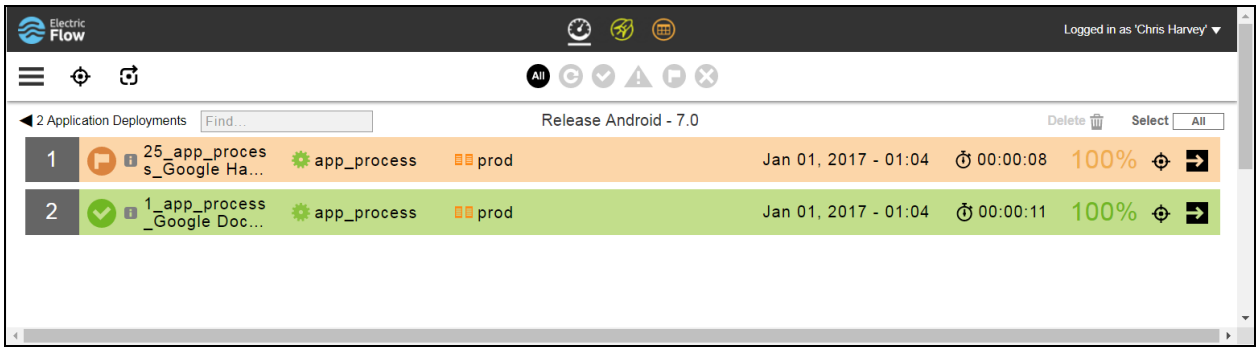
Drilling Down into Phases

Drill-down lets you take a "closer look" at the data in the Release Command Center dashboard. You can drill down into each of the pipeline phases. Drill-down for phases is implemented using either bundled plugins such as EC-ALM or EC-ServiceNow or using the ElectricFlow Deploy UI itself. Drill-down details are provided below.

Planning Phase

Cell	Description
Planning Complete	Percentage of release stories that have story points assigned. This metric includes a trend indicator arrow.
Stories	Total count of release stories.
Stories Estimated	Total count of release stories with story points.

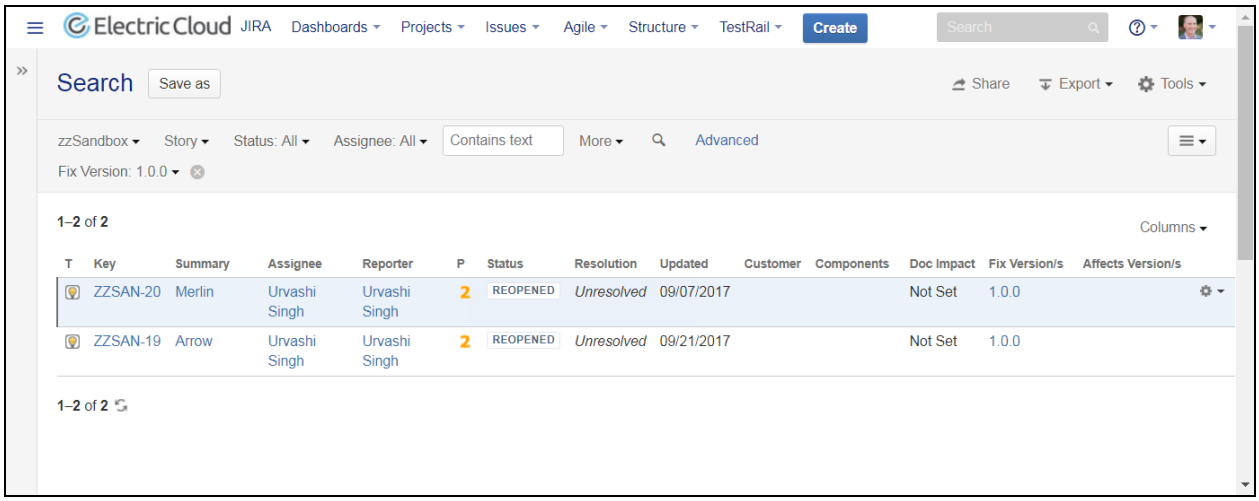
Drill-down in the Planning phase collects data from the EC-JIRA plugin. Clicking any cell in this phase opens a customized JIRA page. For example:



Dev Phase

Cell	Description
Development Complete	Percentage of release stories that have exited development.
Stories Resolved	Percentage of release stories that are development complete.
Unit Test Success Rate	Percentage of successful unit tests out of the total unit tests run through CI builds for the release in the last 10 days.

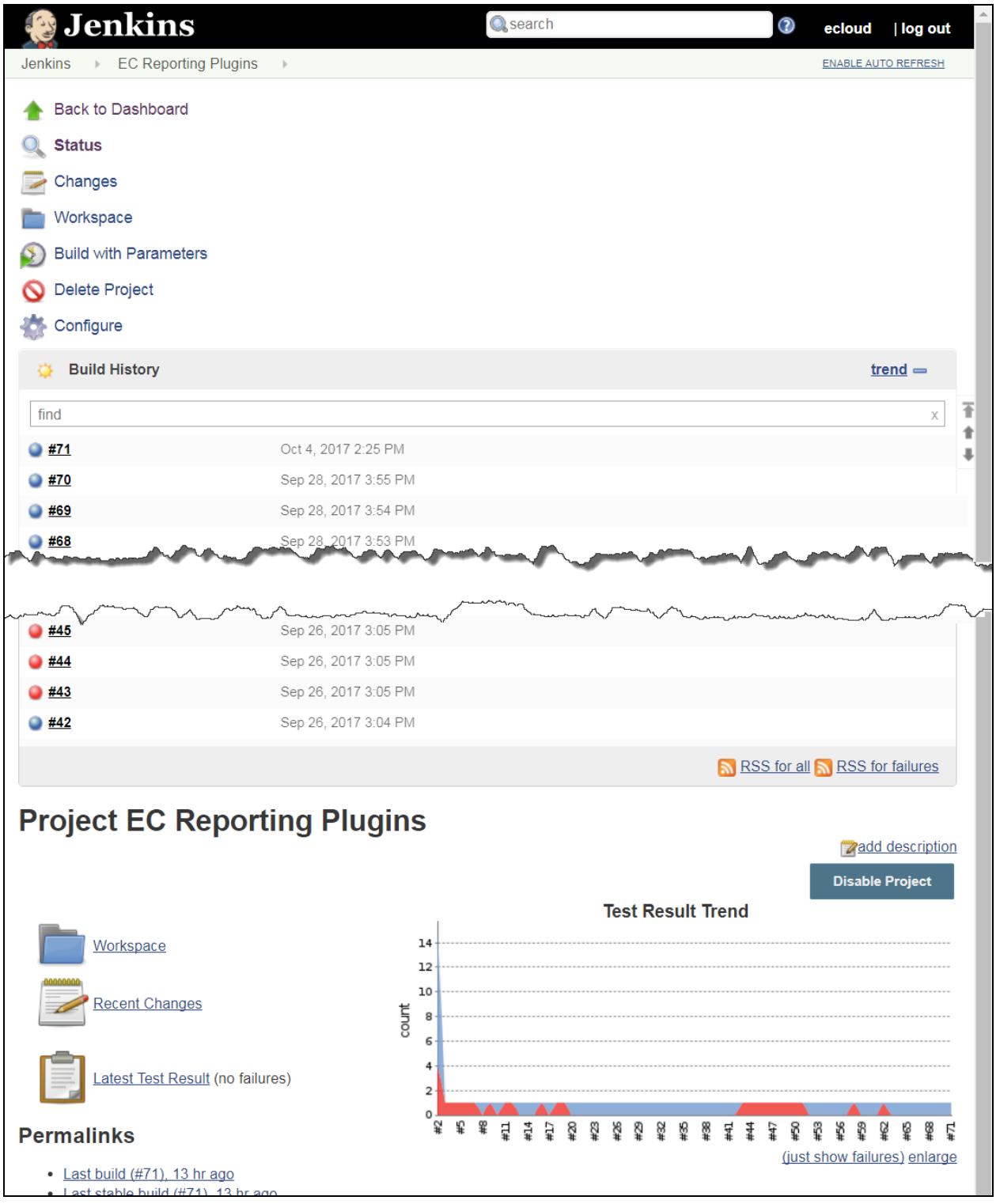
Drill-down in the Dev phase collects data from the EC-JIRA plugin. Clicking any cell in this phase opens a customized JIRA page. For example:



Build Phase

Cell	Description
Build Success Rate	Percentage of successful builds out of the total builds for the release in the last 10 days. This metric includes a trend indicator arrow.
Successful Builds	Number of successful builds out of the total builds for the release in the past 10 days.
Time to Build	Average time taken by builds associated with the current release to complete in the past 10 days.

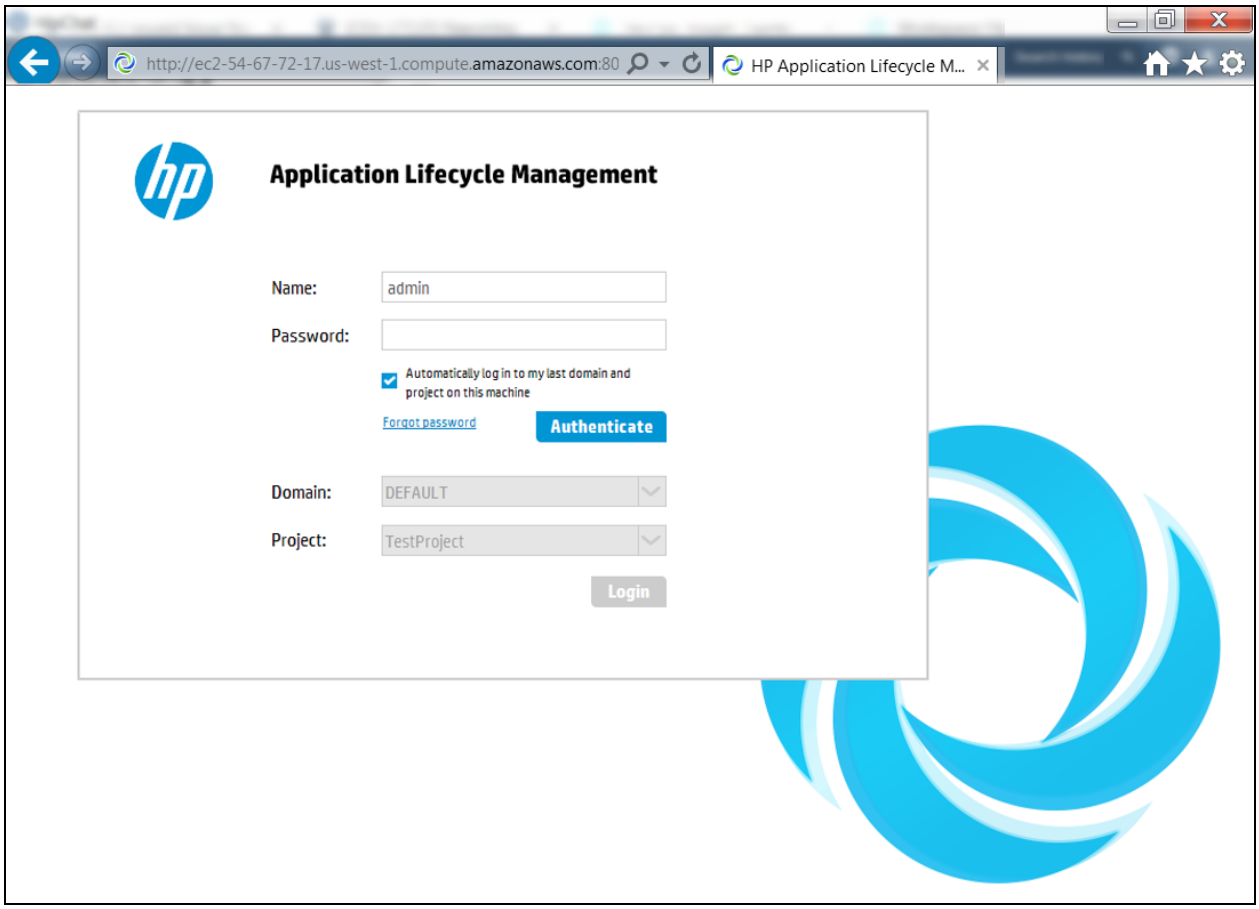
Drill-down in the Build phase collects data from the EC-Jenkins plugin. Clicking any cell in this phase opens a customized Jenkins jobs page in another browser tab to show all the builds. For example:


Jenkins
search
ecloud | log out
Jenkins EC Reporting Plugins
ENABLE AUTO REFRESH
Back to Dashboard
Status
Changes
Workspace
Build with Parameters
Delete Project
Configure
Build History
trend
find
#71 Oct 4, 2017 2:25 PM
#70 Sep 28, 2017 3:55 PM
#69 Sep 28, 2017 3:54 PM
#68 Sep 28, 2017 3:53 PM
#45 Sep 26, 2017 3:05 PM
#44 Sep 26, 2017 3:05 PM
#43 Sep 26, 2017 3:05 PM
#42 Sep 26, 2017 3:04 PM
RSS for all RSS for failures
Project EC Reporting Plugins
add description
Disable Project
Workspace
Recent Changes
Latest Test Result (no failures)
Test Result Trend
count
Permalinks
Last build (#71), 13 hr ago
(just show failures) enlarge

Test Phase

Cell	Description
Test Success Rate	Percentage of successful tests out of the total tests that were run for the release in the last 10 days. This metric includes a trend indicator arrow.
Automated Tests	Percentage of automated tests out of total tests that ran for the release.
Time to Test	Average test duration in the last 10 days.

Drill-down in the Test phase collects data from the EC-ALM plugin. Clicking any cell in this phase opens HP ALM in a browser window, from which you can drill down for more information. For example:

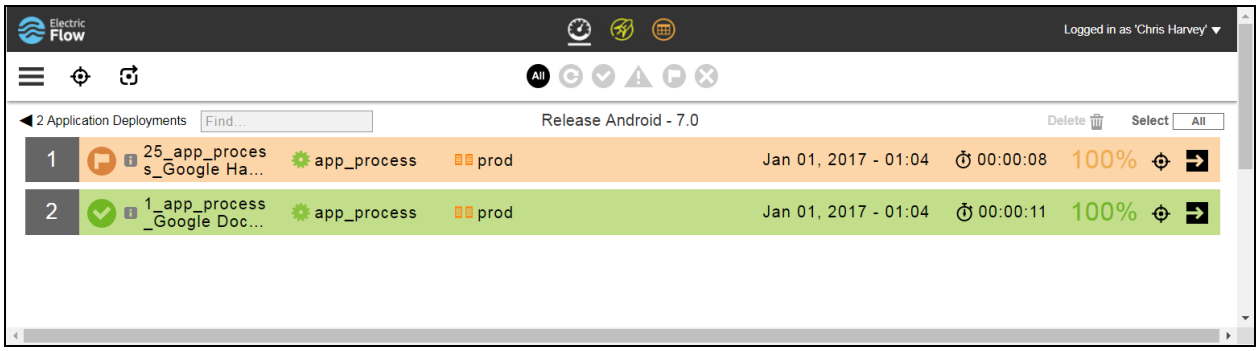


Deploy Phase

Cell	Description
Successful Application Deployments	Number of successful application deployments out of the total completed deployments in the past 10 days.
Successful Microservice Deployments	Number of successful microservice deployments out of the total completed deployments in the past 10 days.

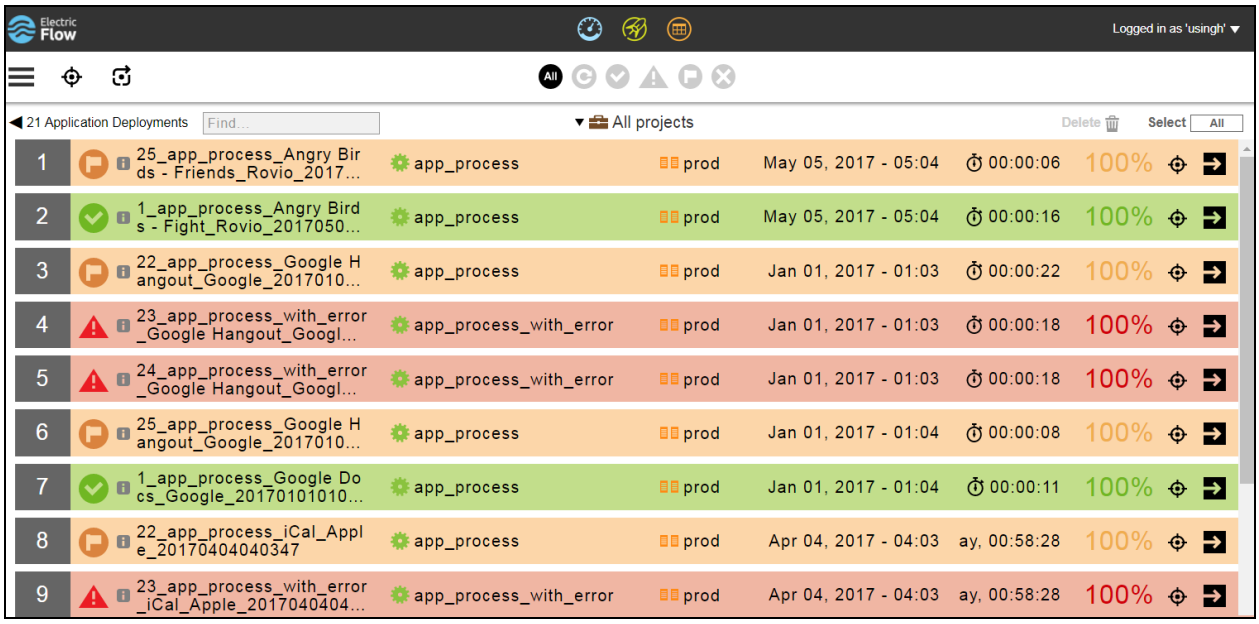
Clicking to drill down into **Successful Application Deployments** opens the **Application Deployments** page, which shows application deployments for the selected release. Note that you can also open the **Application Deployments** page by clicking **Applications** > **Application Deployments** from the **Home** menu.

Following is an example of the **Application Deployments** page when opened from the Release Command Center:

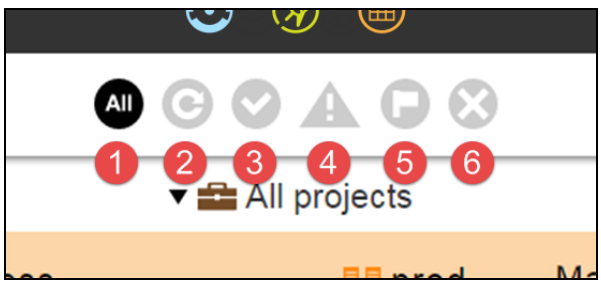


This is a page that is filtered by successful deployments for the selected application.

Following is an example of the **Application Deployments** page when opened from the Home menu:



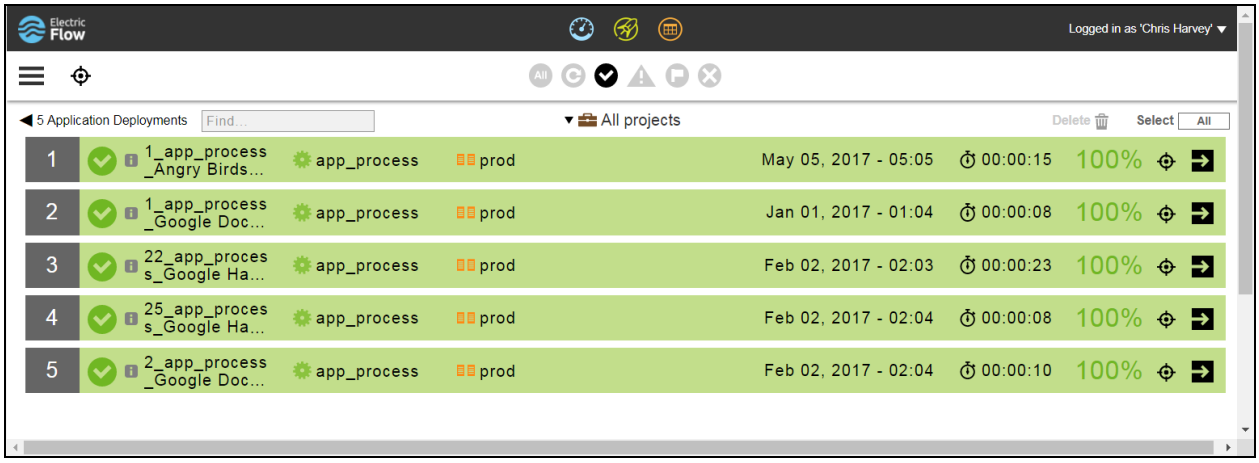
You can filter the list of deployments according to status by using the following buttons:



Following are descriptions of the filters:

1	All deployments
2	Running deployments (gray highlighting)
3	Successful deployments (green highlighting)
4	Deployments that stopped because of one or more errors (red highlighting)
5	Deployments that finished with one or more warnings (yellow highlighting)
6	Aborted deployments (gray highlighting)

For example, you can view only the successful deployments by clicking the  (Successful) button:



To view the details of a deployment, click the deployment, which opens the **Applications / View Run** page. For more information, see Deploying and Troubleshooting Applications on page 287.

Clicking to drill down into **Successful Microservice Deployments** opens the **Microservice Deployments** page, which shows independent (project-level) microservice deployments for the selected release. Note that you can also open the **Microservice Deployments** page by clicking **Microservices** > **Microservice Deployments** from the **Home** menu.

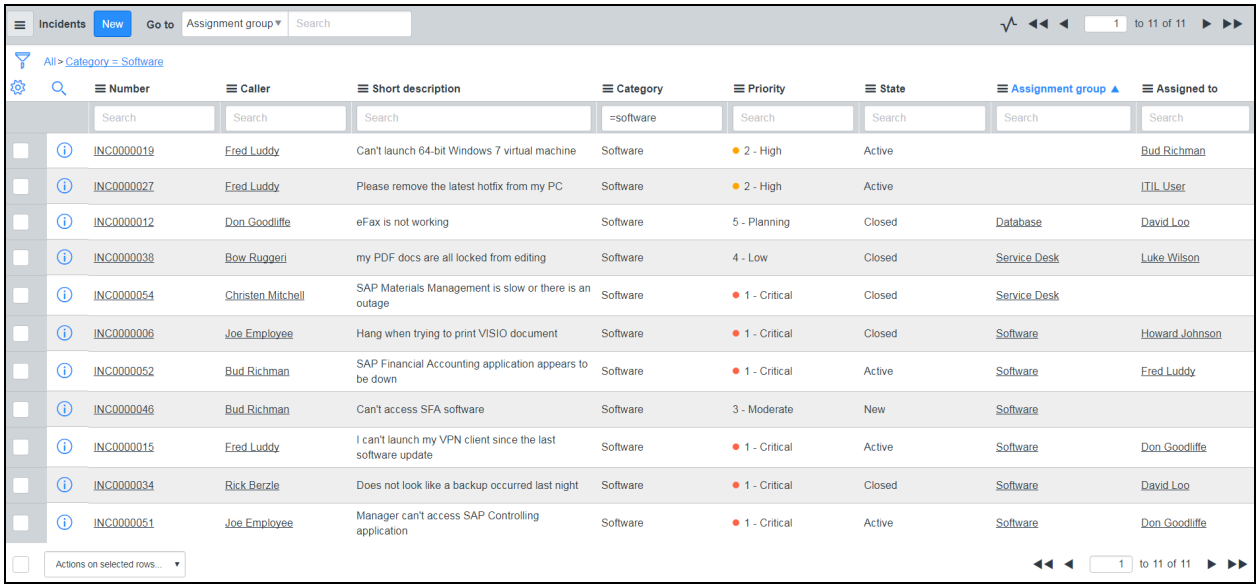
The filter buttons for microservice deployments work in a manner similar to the filter buttons for application deployments as described above.

Operate Phase

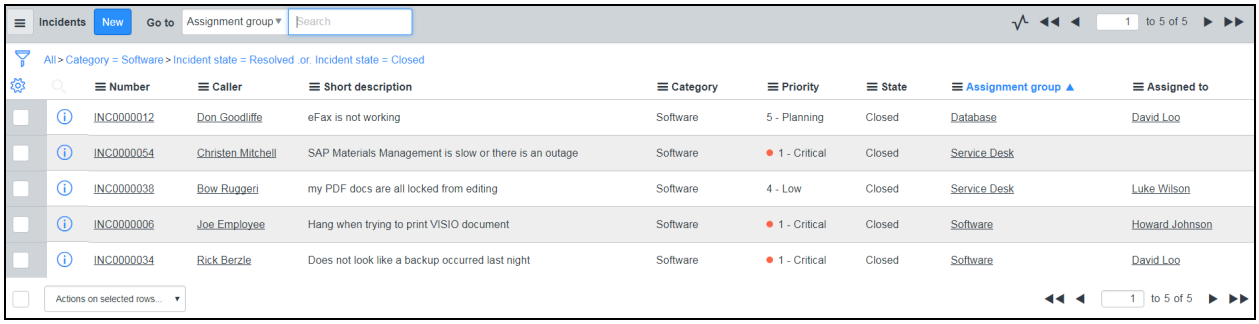
The Operate phase shows incident metrics associated with the deployed release (post-production). This phase collects data periodically from the EC-ServiceNow plugin. The plugin procedure provides a general field mapping parameter (as do other DevOps Insight enabled plugins), which lets you map any field from the ServiceNow incident table to the incident report object type in the DevOps Insight server.

Cell	Description
Total Incidents	Total number of incidents opened against the release.
Resolved Incidents	Number of resolved incidents out of total incidents for the release.
Average Time to Resolve	Average time to resolve an incident for the release.

You can drill down into this phase. Clicking the **Total Incidents** cell in this phase opens a ServiceNow page that lists all incidents associated with the release. For example:



Clicking the **Resolved Incidents** cell or the **Average Time to Resolve** cell in this phase opens a ServiceNow page that lists all resolved incidents associated with the release. For example:

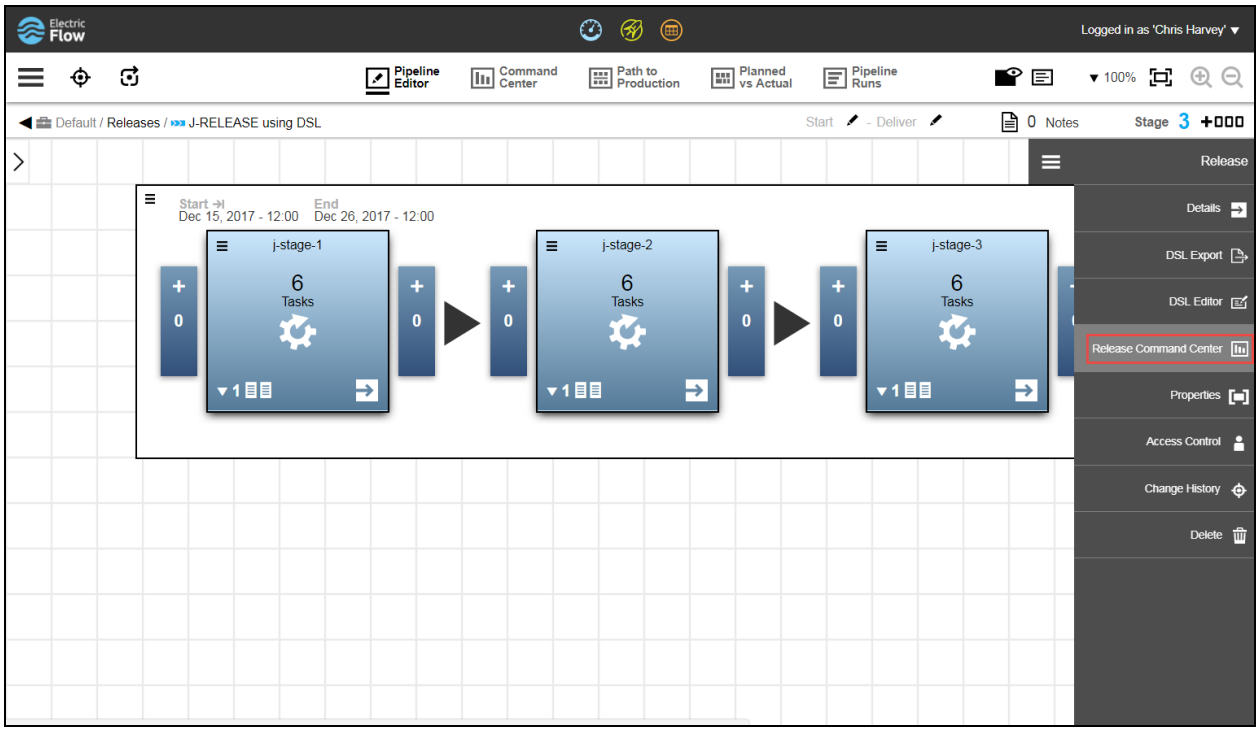


Opening the Dashboard

There are three ways to access the Release Command Center dashboard:

- From the dashboard UI by clicking the  (DevOps Insight dashboards launch) button and then selecting **Release Command Center** from the dashboards drop-down list.

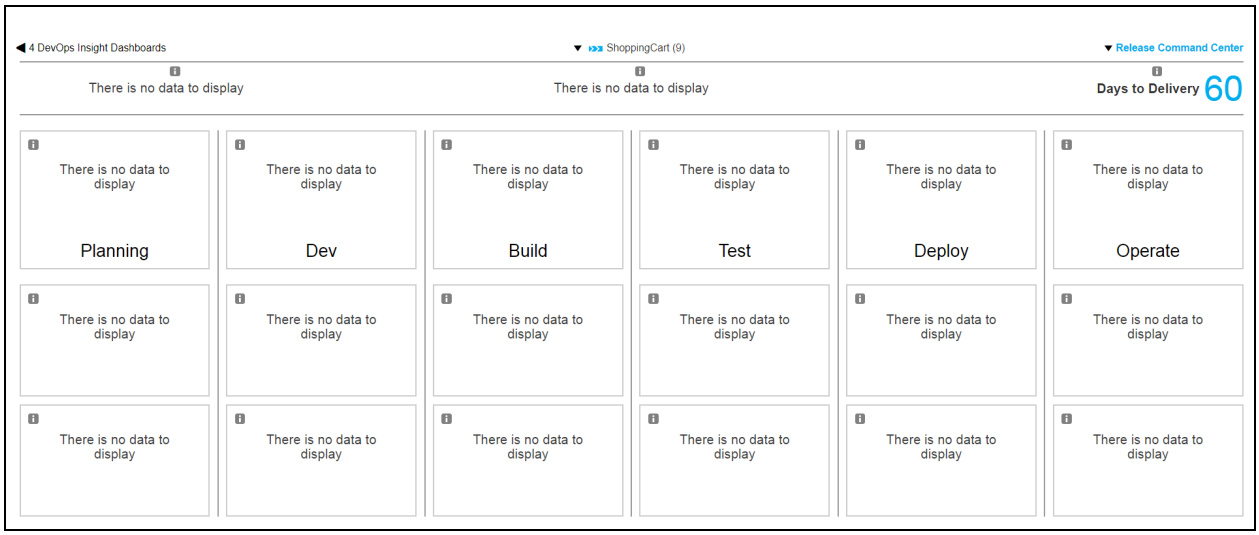
- From the Application Editor by clicking the (Command Center) button.
- From the **Release Command Center** context menu within a release details page. For example:



Enabling the Plugins to Populate the Dashboard

To help you get started quickly with the Release Command Center dashboard for different plugins, ElectricFlow bundles certain content in the form of procedures that provide a more prescribed set.

The Release Command Center dashboard shows release data that it collects and aggregates for tools such as JIRA and Jenkins. When you first view the Release Command Center for a release, each cell contains a message indicating that there is no data to display:



ElectricFlow includes several plugins that you can configure to periodically collect data from third-party tools and send it to the DevOps Insight server for displaying in the Release Command Center dashboard. The following table shows the sections and phases of the Release Command Center and the plugins that you configure to collect data for each cell in the different sections and phases.

Phase or section	Third-party tool	Plugin	Type of data collected
Summary section	JIRA	EC-JIRA	Feature (JIRA stories) Defect (JIRA defects)
Planning phase	JIRA	EC-JIRA	Feature (JIRA stories)
Dev phase (other than "Percentage of successful unit tests")	JIRA	EC-JIRA	Feature (JIRA stories)
Dev phase (for "Percentage of successful unit tests")	Jenkins	EC-Jenkins	Quality (test result from Jenkins builds)
Build phase	Jenkins	EC-Jenkins	Build (Jenkins builds)
Test phase	HP ALM	EC-ALM	Quality (test runs from HP ALM)
Operate phase	ServiceNow	EC-ServiceNow	Incident (ServiceNow incidents)

ElectricFlow automatically sends data for the Deploy phase to the DevOps Insight server as deployments that are orchestrated by ElectricFlow. The deployments begin to appear in the Release Command Center as soon as deployments triggered through your release pipeline are complete.

Each plugin supported by the Release Command Center has a procedure named CollectReportingData, which you can set up to run periodically to collect data for one or more releases. ElectricFlow includes a utility procedure in the "Electric Cloud" project for each of these plugins to make plugin setup easier. The following sections describe the utility procedure provided for each plugin.

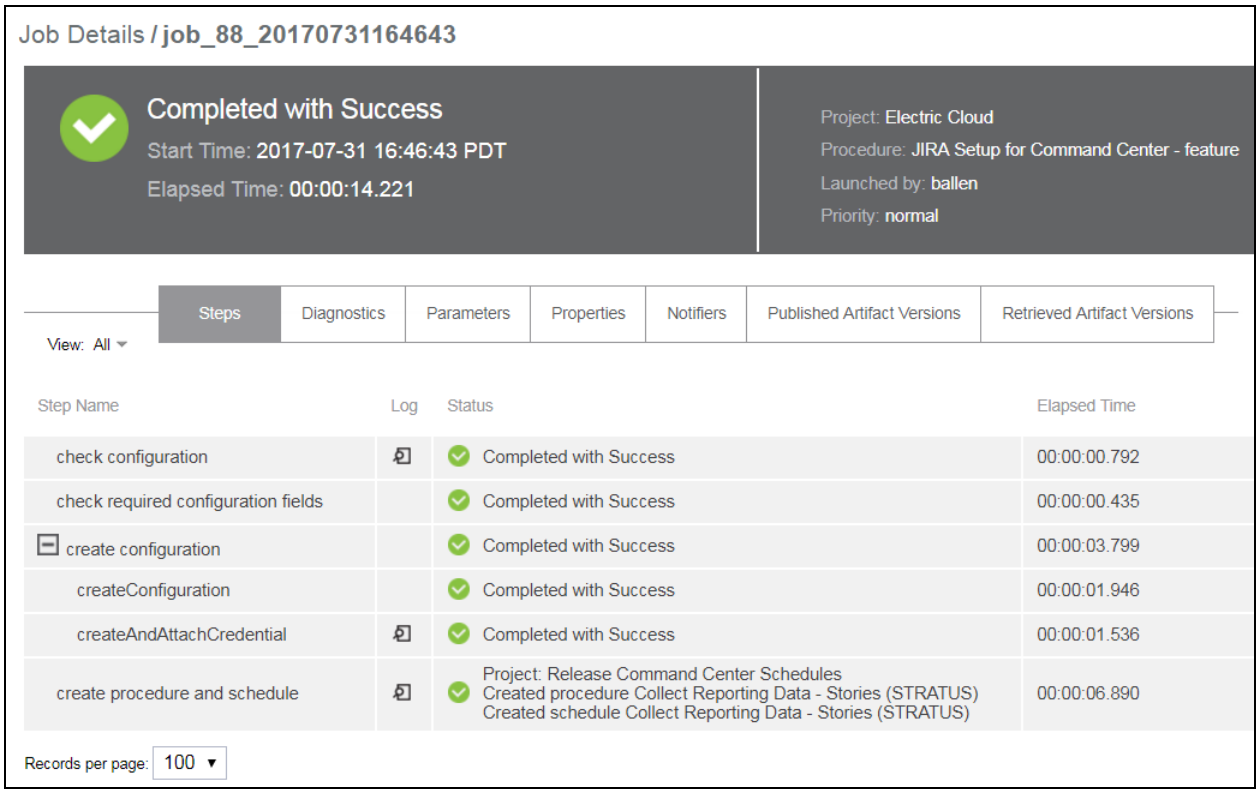
"JIRA Setup for Command Center - feature" Procedure

This procedure creates a schedule that runs periodically to retrieve updated stories from JIRA and send the data to the DevOps Insight server. This occurs via the sendReportingData API.

Parameters

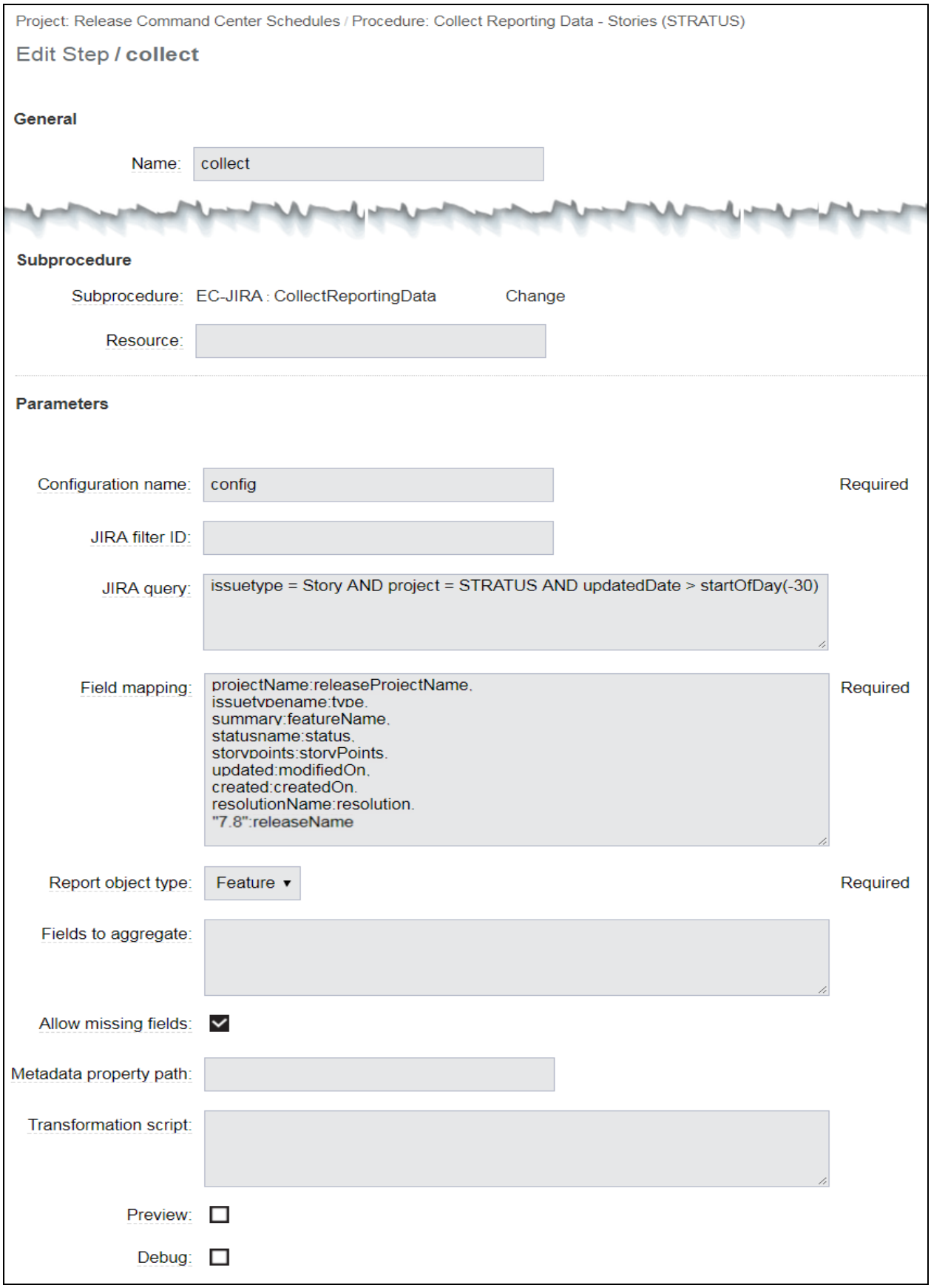
Parameter	Required	Description
Configuration Name	Yes	Name of the JIRA plugin configuration to use. This plugin configuration is created if it does not exist.
JIRA Server URL	No	JIRA server URL, such as http://jira.mycompany.com. Required if the plugin configuration does not exist and must be created.
JIRA Credentials	No	User name and password to connect to JIRA. Required if the plugin configuration does not exist and must be created.
JIRA Project	Yes	JIRA project key to collect JIRA issues from.
Preview Only	No	If checked, no data is sent to the DevOps Insight server. Use this option to preview gathered data.
JIRA Fix Version	No	This parameter will map to the name of the release that the feature is associated with. Set this parameter if your release name does not map directly to the fixVersion in JIRA.
Project Name	No	Name of the project where the schedule for gathering feature data for the command center will be created. The project is created if it does not exist.

Parameter	Required	Description
Schedule and procedure name to use	Yes	Name of the schedule and the procedure that will be created for gathering feature data for the command center.
Schedule Frequency	Yes	Frequency (in minutes) for the schedule that is created for gathering data. The default is 30 minutes.



This procedure also creates a procedure in the same project as the schedule. This newly-created procedure is triggered by the schedule, which then invokes the CollectReportingData plugin procedure.

The procedure maps the fields from JIRA to the fields defined in the feature report object type based on a predetermined default mapping. You can update this mapping by editing the newly-created procedure if required to suit your specific requirements.


Project: Release Command Center Schedules / Procedure: Collect Reporting Data - Stories (STRATUS)
Edit Step / collect
General
Name: collect
Subprocedure
Subprocedure: EC-JIRA : CollectReportingData Change
Resource:
Parameters
Configuration name: config Required
JIRA filter ID:
JIRA query: issuetype = Story AND project = STRATUS AND updatedDate > startOfDay(-30)
Field mapping: projectName:releaseProjectName.
issuetypename:type.
summary:featureName.
statusname:status.
storypoints:storyPoints.
updated:modifiedOn.
created:createdOn.
resolutionName:resolution.
"7.8":releaseName
Required
Report object type: Feature Required
Fields to aggregate:
Allow missing fields:
Metadata property path:
Transformation script:
Preview:
Debug:

See the figure above for the default mapping used in the procedure created by running the "JIRA Setup for Command Center - feature" procedure. You can update this mapping manually based on your specific requirements. See the EC-JIRA plugin help file for details on field mappings and other input parameters for the CollectReportingData procedure.

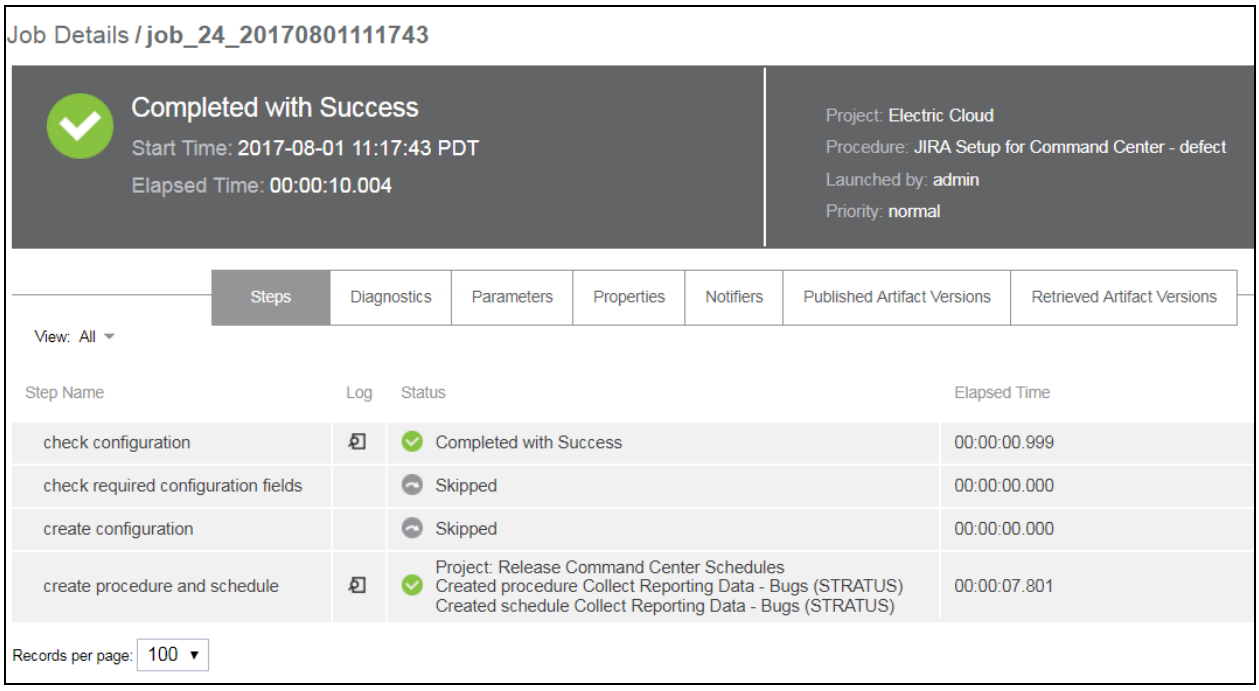
"JIRA Setup for Command Center - defect" Procedure

This procedure creates a schedule that runs periodically to retrieve updated defects from JIRA and send the data to the DevOps Insight server. This occurs via the sendReportingData API.

Parameters

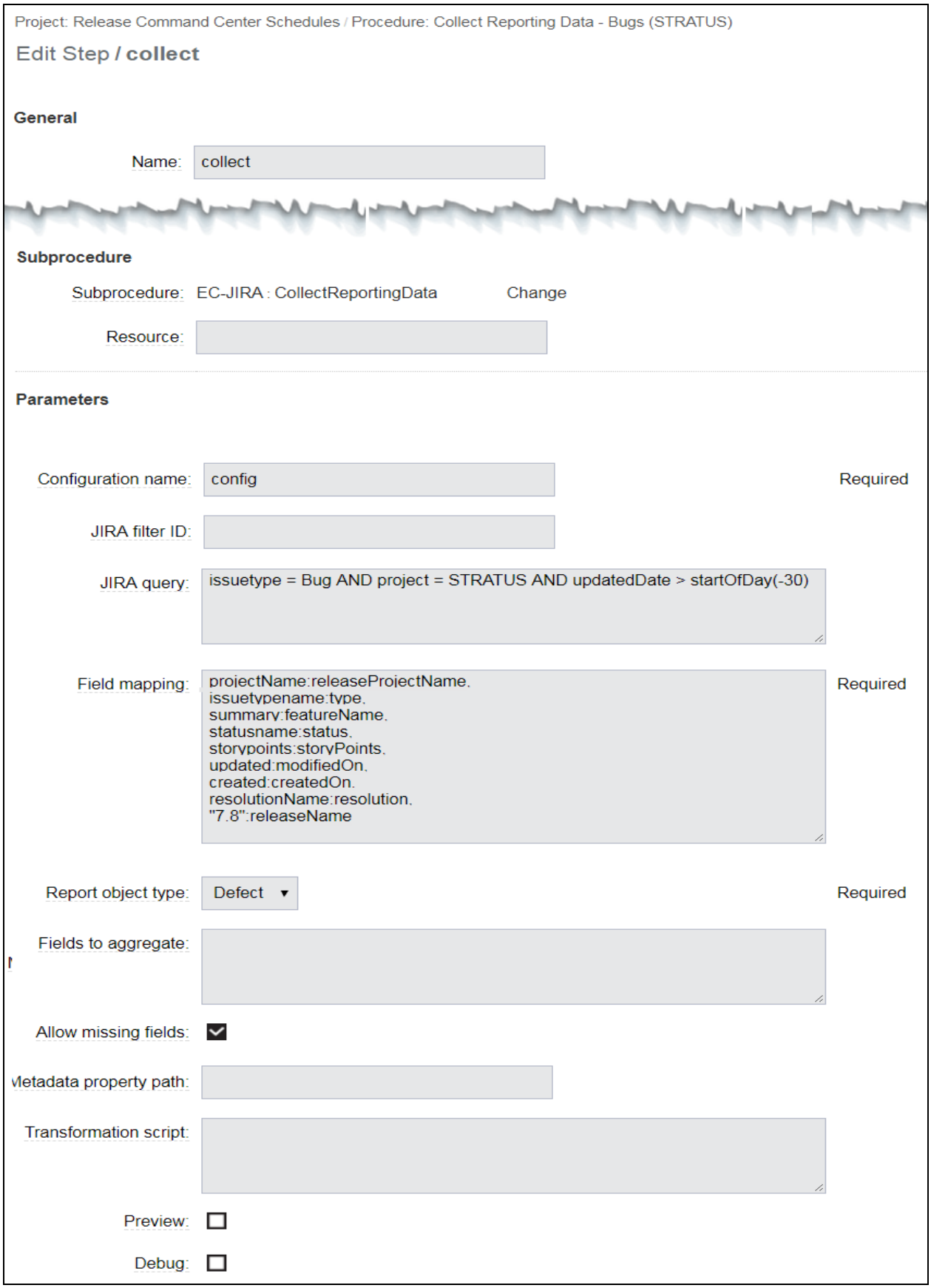
Parameter	Required	Description
Configuration Name	Yes	Name of the JIRA plugin configuration to use. This plugin configuration is created if it does not exist.
JIRA Server URL	No	JIRA server URL, such as http://jira.mycompany.com. Required if the plugin configuration does not exist and must be created.
JIRA Credentials	No	User name and password to connect to JIRA. Required if the plugin configuration does not exist and must be created.
JIRA Project	Yes	JIRA project key to collect JIRA issues from.
Preview Only	No	If checked, no data is sent to the DevOps Insight server. Use this option to preview gathered data.
JIRA Fix Version	No	This parameter will map to the name of the release that the defect is associated with. Set this parameter if your release name does not directly map to the fixVersion in JIRA.
Project Name	No	Name of the project where the schedule for gathering defect data for the command center will be created. The project is created if it does not exist.

Parameter	Required	Description
Schedule and procedure name to use	Yes	Name of the schedule and the procedure that will be created for gathering defect data for the command center.
Schedule Frequency	Yes	Frequency (in minutes) for the schedule that is created for gathering data. The default is 30 minutes.



This procedure also creates a procedure in the same project as the schedule. This newly-created procedure is triggered by the schedule, which then invokes the CollectReportingData plugin procedure.

The procedure maps the fields from JIRA to the fields defined in the defect report object type based on a predetermined default mapping. You can update this mapping manually by editing the newly-created procedure if required, based on your specific requirements.


Project: Release Command Center Schedules / Procedure: Collect Reporting Data - Bugs (STRATUS)
Edit Step / collect
General
Name: collect
Subprocedure
Subprocedure: EC-JIRA : CollectReportingData Change
Resource:
Parameters
Configuration name: config Required
JIRA filter ID:
JIRA query: issuetype = Bug AND project = STRATUS AND updatedDate > startOfDay(-30)
Field mapping: projectName:releaseProjectName,
issuetypename:type,
summary:featureName,
statusname:status,
storypoints:storyPoints,
updated:modifiedOn,
created:createdOn,
resolutionName:resolution,
"7.8":releaseName
Required
Report object type: Defect Required
Fields to aggregate:
Allow missing fields:
Metadata property path:
Transformation script:
Preview:
Debug:

See the figure above for the default mapping used in the procedure created by running the "JIRA Setup for Command Center - defect" procedure. You can update this mapping manually based on your specific requirements. See the EC-JIRA plugin help file for details on field mappings and other input parameters for the CollectReportingData procedure.

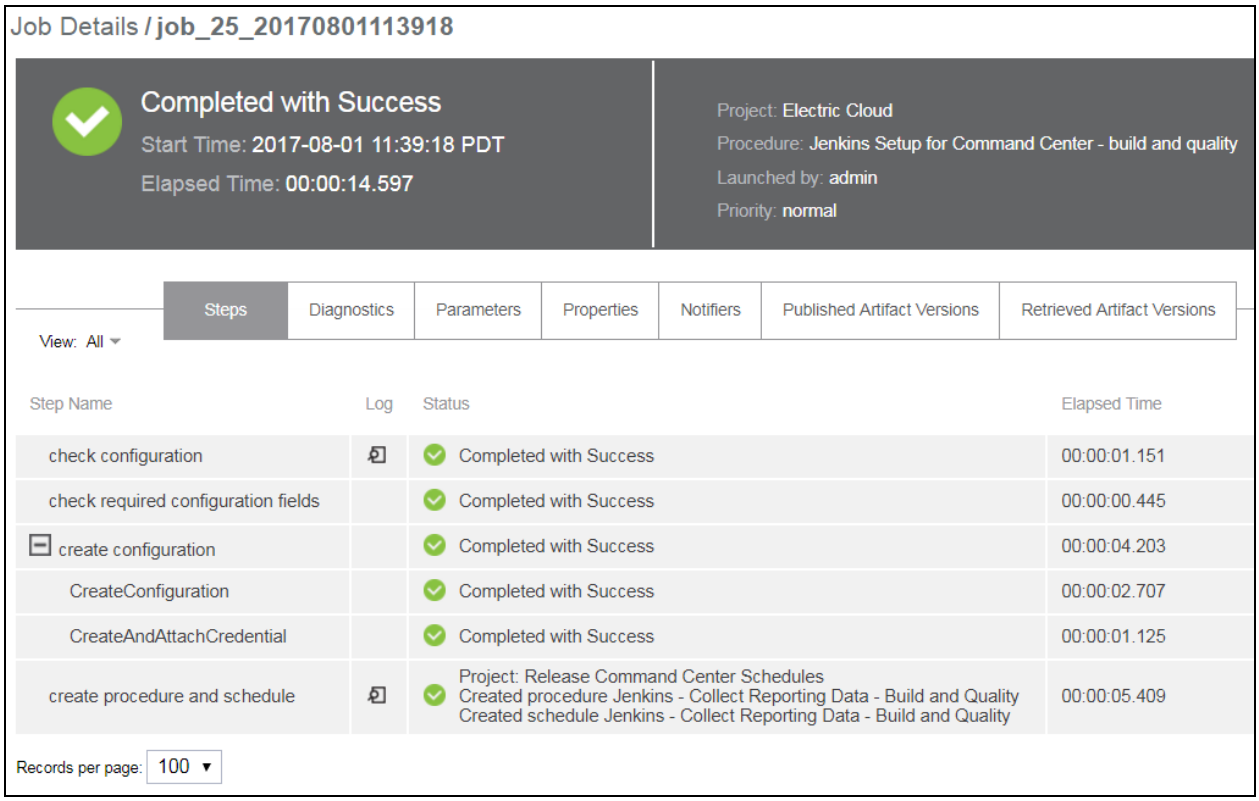
"Jenkins Setup for Command Center - build and quality" Procedure

This procedure creates a schedule that runs periodically to retrieve build data from Jenkins and send it to the DevOps Insight server. This occurs via the sendReportingData API.

Parameters

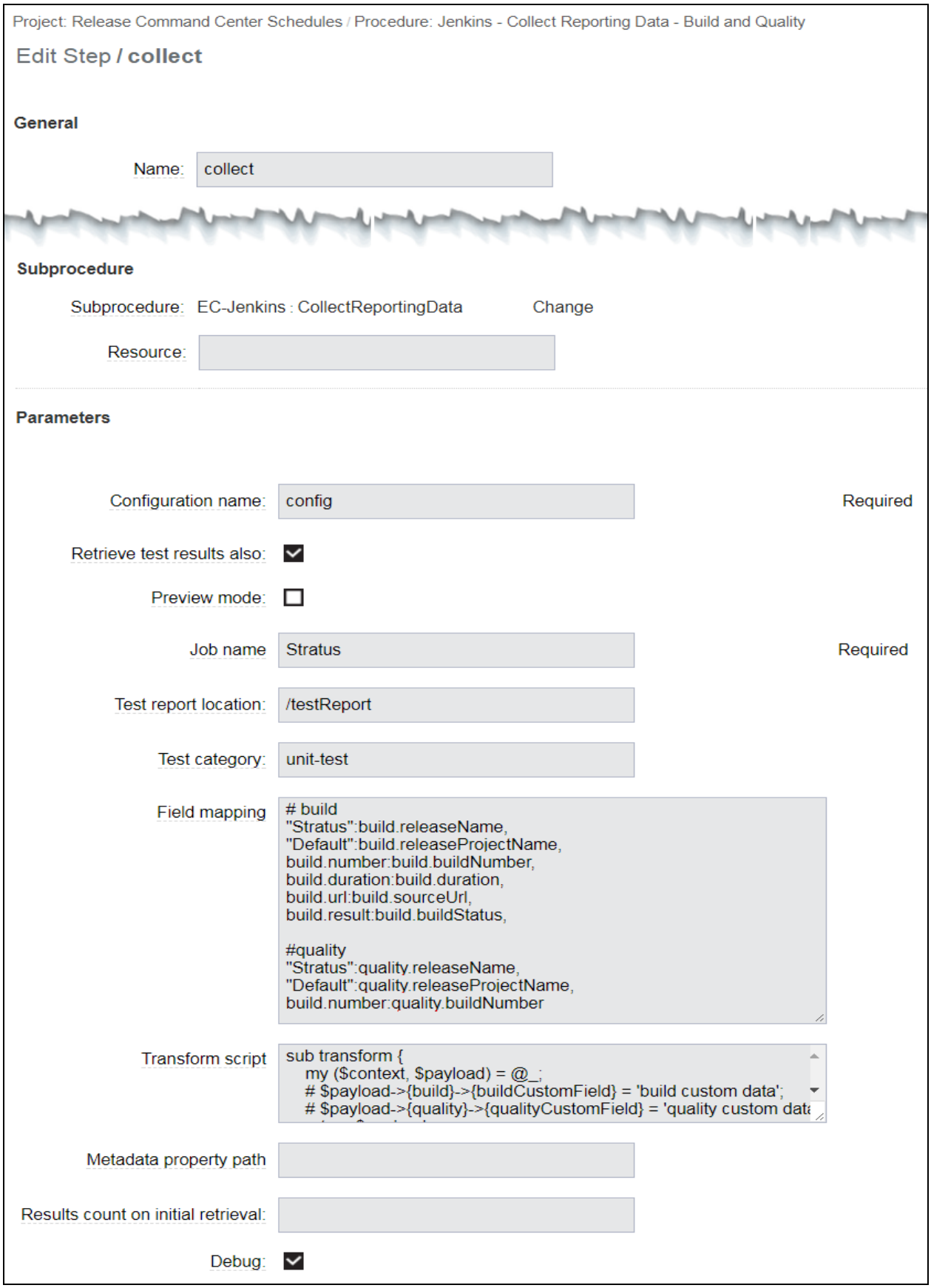
Parameter	Required	Description
Configuration Name	Yes	Name of the Jenkins plugin configuration to use. This plugin configuration is created if it does not exist.
Jenkins URL	No	The Jenkins server URL, such as http://jenkins:8080. Required if the plugin configuration does not exist and must be created.
Jenkins Credentials	No	User name and password to connect to Jenkins. Required if the plugin configuration does not exist and must be created.
Jenkins Job	Yes	Name of the Jenkins job for while the build data will be collected and sent to the DevOps Insight server.
Retrieve test results also	No	If this is checked, test results are also retrieved for the builds and sent as quality data to the DevOps Insight server along with the build data.

Parameter	Required	Description
Test result URL	Yes	Relative URL of test results in Jenkins.
Release Name	Yes	Name of the release that the retrieved builds are associated with.
Release Project Name	Yes	Name of the project that the release belongs to that the retrieved builds are associated with.
Preview Only	No	If this is checked, no data is sent to the DevOps Insight server. Use this option to preview gathered data.
Project Name	No	Name of the project where the schedule for gathering build data for the command center will be created. The project is created if it does not exist.
Schedule and procedure name to use	Yes	Name of the schedule and the procedure that will be created for gathering build and quality data for the command center.
Schedule Frequency	Yes	Frequency (in minutes) for the schedule that is created for gathering data. The default is 30 minutes.



This procedure also creates a procedure in the same project as the schedule. This newly-created procedure is triggered by the schedule, which then invokes the CollectReportingData plugin procedure.

The procedure maps the fields from Jenkins to the fields defined in the build report object type and quality report object type based on predetermined default mappings. You can update this mapping manually by editing the newly-created procedure for your specific requirements.


Project: Release Command Center Schedules / Procedure: Jenkins - Collect Reporting Data - Build and Quality
Edit Step / collect
General
Name: collect
Subprocedure
Subprocedure: EC-Jenkins : CollectReportingData Change
Resource:
Parameters
Configuration name: config Required
Retrieve test results also:
Preview mode:
Job name Stratus Required
Test report location: /testReport
Test category: unit-test
Field mapping
build
"Stratus":build.releaseName,
"Default":build.releaseProjectName,
build.number:build.buildNumber,
build.duration:build.duration,
build.url:build.sourceUrl,
build.result:build.buildStatus,
#quality
"Stratus":quality.releaseName,
"Default":quality.releaseProjectName,
build.number:quality.buildNumber
Transform script
sub transform {
my ($context, $payload) = @_;
# $payload->{build}->{buildCustomField} = 'build custom data';
# $payload->{quality}->{qualityCustomField} = 'quality custom data
Metadata property path
Results count on initial retrieval:
Debug:

See the figure above for the default mapping used in the procedure created by running the "Jenkins Setup for Command Center - build and quality" procedure. You can update this mapping manually based on your specific requirements. See the EC-Jenkins plugin help file for details on field mappings and other input parameters for the CollectReportingData procedure.

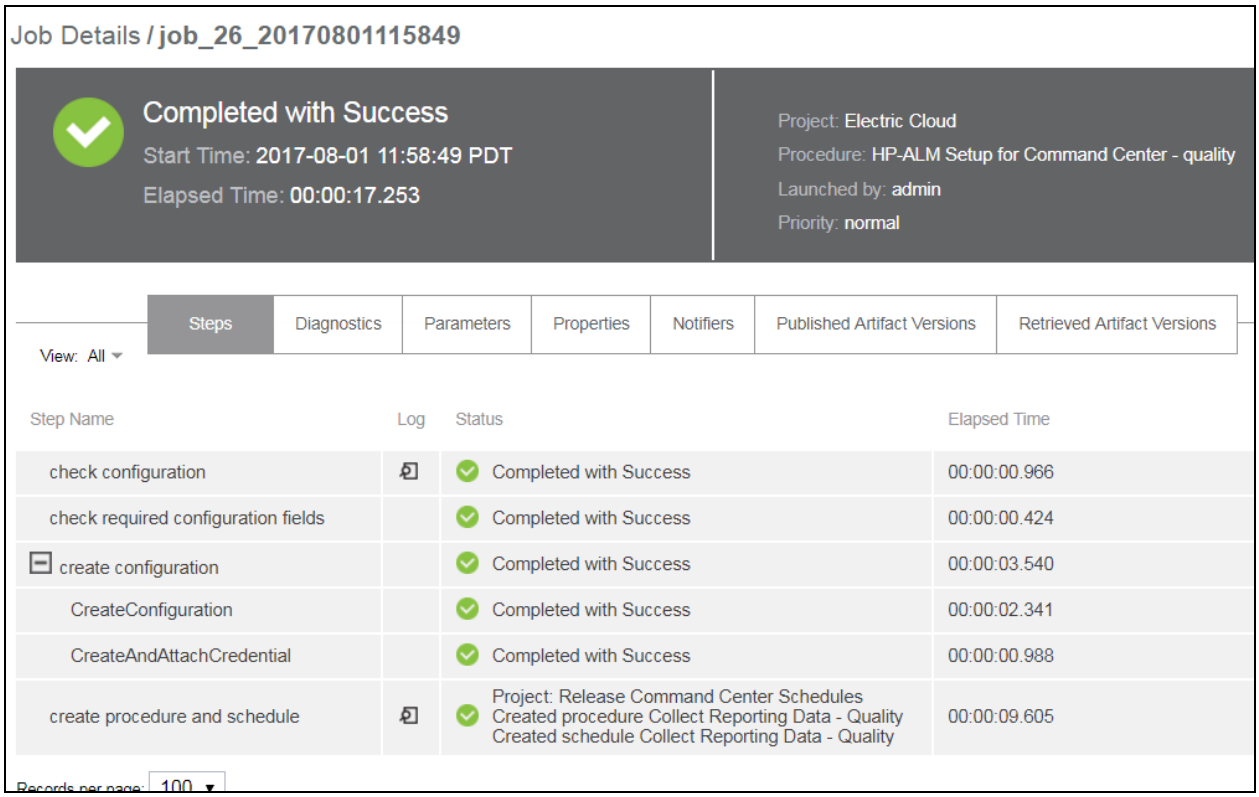
"HP-ALM Setup for Command Center - quality" Procedure

This procedure creates a schedule that runs periodically to retrieve updated stories from HP ALM and send quality data to the DevOps Insight server. This occurs via the sendReportingData API.

Parameters

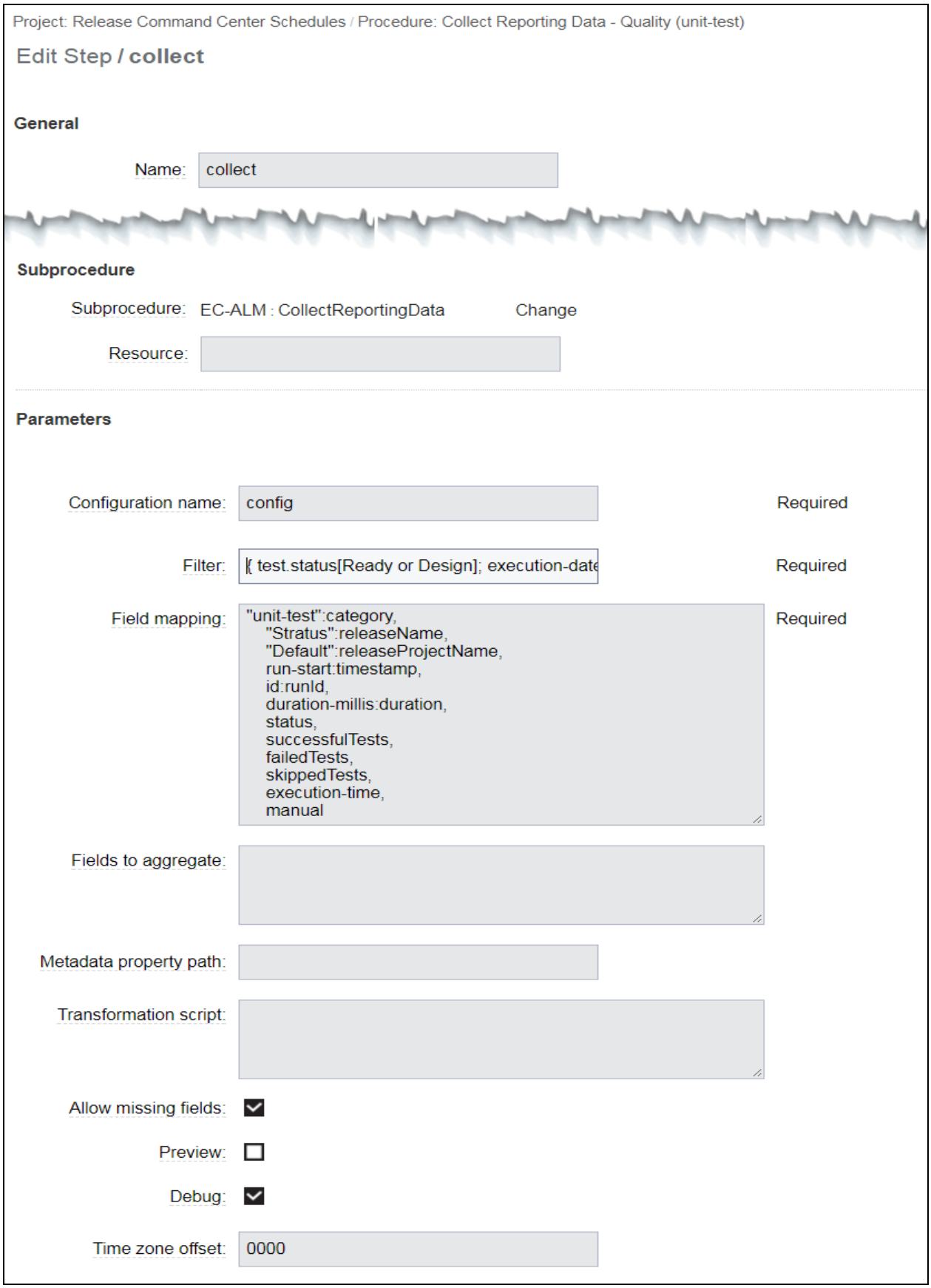
Parameter	Required	Description
Configuration Name	Yes	Name of the ALM plugin configuration to use. This plugin configuration is created if it does not exist.
HP ALM Server URL	No	HP ALM server URL, such as http://mycompany:8080/qcbin. Required if the plugin configuration does not exist and must be created.
HP ALM Domain	No	Domain name for HP ALM. Required if the plugin configuration does not exist and must be created.
HP ALM Project	No	Project name for HP ALM. Required if the plugin configuration does not exist and must be created.
HP ALM Credentials	No	User name and password to connect to the HP ALM instance. Required if the plugin configuration does not exist and must be created.
HP ALM Server Timezone offset	No	Timezone offset for HP ALM server; 0000 stands for GMT. Used for converting datetime fields retrieved from the HP ALM server to UTC before sending them to the DevOps Insight server.
Preview Only	No	If this is checked, no data is sent to the DevOps Insight server. Use this option to preview gathered data.

Parameter	Required	Description
Release Name	Yes	Name of the release that the retrieved test runs (stored as quality data in DevOps Insight Server) are associated with.
Release Project Name	Yes	Name of the project that the release belongs to, that the retrieved test runs (stored as quality data in DevOps Insight Server) are associated with.
Project Name	No	Name of the project where the schedule for gathering quality data for the command center will be created. The project is created if it does not exist.
Schedule and procedure name to use	Yes	Name of the schedule and the procedure that will be created for gathering quality data for the command center.
Schedule Frequency	Yes	Frequency (in minutes) for the schedule that is created for gathering data. The default is 30 minutes.



This procedure also creates a procedure in the same project as the schedule. This newly-created procedure is triggered by the schedule, which then invokes the CollectReportingData plugin procedure.

The procedure maps the fields from HP ALM test runs to the fields defined in the quality report object type based on a predetermined default mapping. You can update this mapping manually by editing the newly-created procedure for your specific requirements.


Project: Release Command Center Schedules / Procedure: Collect Reporting Data - Quality (unit-test)
Edit Step / collect
General
Name: collect
Subprocedure
Subprocedure: EC-ALM : CollectReportingData Change
Resource:
Parameters
Configuration name: config Required
Filter: { test.status[Ready or Design]; execution-date Required
Field mapping: "unit-test":category, "Stratus":releaseName, "Default":releaseProjectName, run-start:timestamp, id:runId, duration-millis:duration, status, successfulTests, failedTests, skippedTests, execution-time, manual Required
Fields to aggregate:
Metadata property path:
Transformation script:
Allow missing fields:
Preview:
Debug:
Time zone offset: 0000

See the figure above for the default mapping used in the procedure created by running the "HP-ALM Setup for Command Center - quality" procedure. You can update this mapping manually based on your specific requirements. See the EC-ALM plugin help file for details on field mappings and other input parameters for the CollectReportingData procedure.

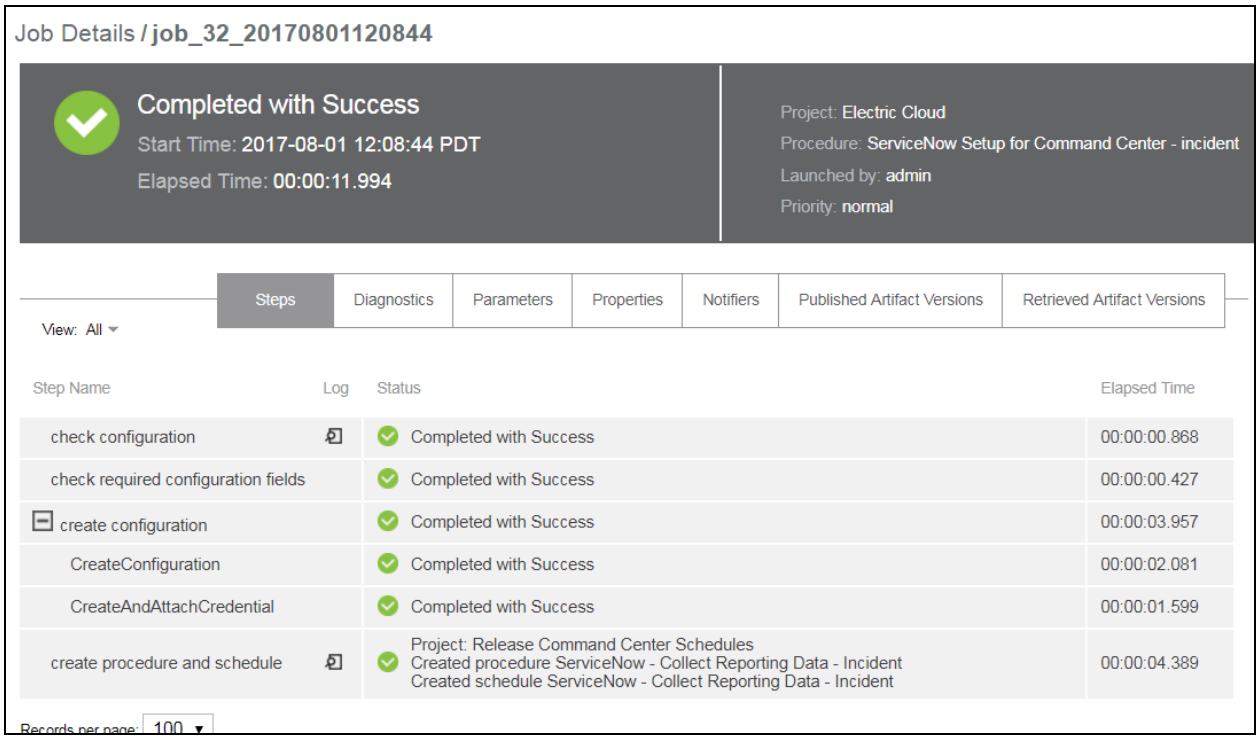
"ServiceNow Setup for Command Center - incident" Procedure

This procedure creates a schedule that runs periodically to retrieve updated incidents from ServiceNow and send the data to the DevOps Insight server. This occurs via the sendReportingData API.

Parameters

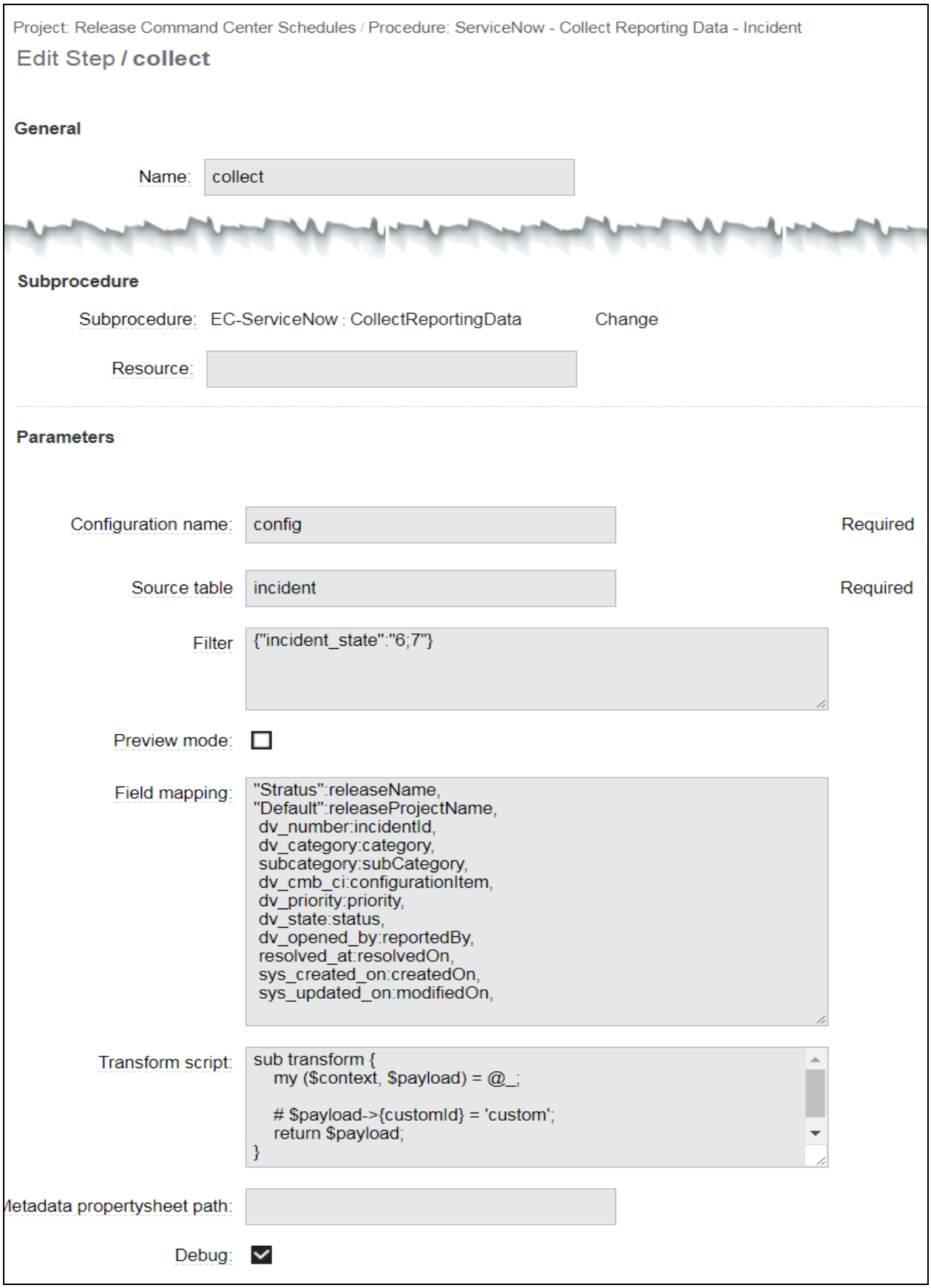
Parameter	Required	Description
Configuration Name	Yes	Name of the ServiceNow plugin configuration to use. This plugin configuration is created if it does not exist.
ServiceNow Instance URL	No	ServiceNow instance URL, such as https://instance.service-now.com. Required if the plugin configuration does not exist and must be created.
ServiceNow Credentials	No	User name and password to connect to the ServiceNow instance. Required if the plugin configuration does not exist and must be created.
ServiceNow Source Table	Yes	ServiceNow source table. Set to 'incident' to retrieve updated incidents from ServiceNow.
Release Name	Yes	Name of the release that the retrieved incidents are associated with.
Release Project Name	Yes	Name of the project containing the release.

Parameter	Required	Description
Preview Only	No	If this is checked, no data is sent to the DevOps Insight server. Use this option to preview gathered data.
Project Name	No	Name of the project where the schedule for gathering incident data for the command center will be created. The project is created if it does not exist.
Schedule and procedure name to use	Yes	Name of the schedule and the procedure that will be created for gathering incident data for the command center.
Schedule Frequency	Yes	Frequency (in minutes) for the schedule that is created for gathering data. The default is 30 minutes.



This procedure also creates a procedure in the same project as the schedule. This newly-created procedure is triggered by the schedule, which then invokes the CollectReportingData plugin procedure.

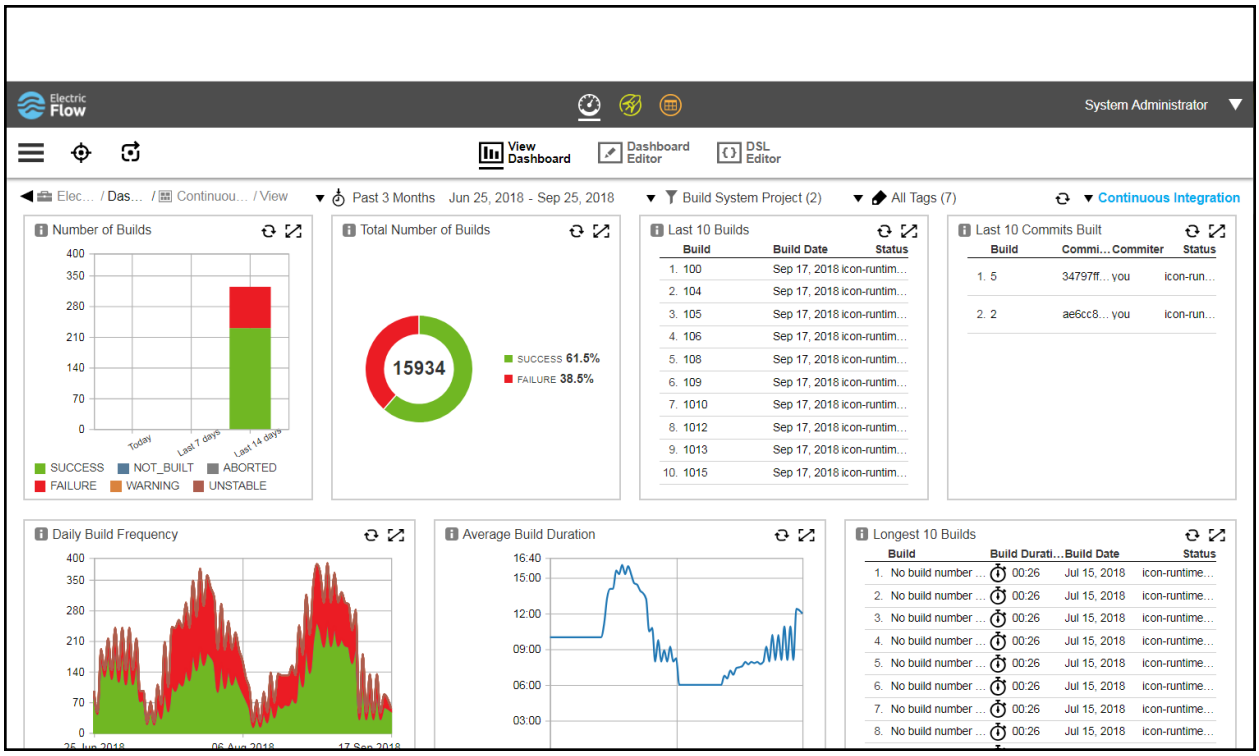
The procedure maps the fields from ServiceNow incident records to the fields defined in the incident report object type based on a predetermined default mapping. You can update this mapping manually by editing the newly-created procedure for your specific requirements.


Project: Release Command Center Schedules / Procedure: ServiceNow - Collect Reporting Data - Incident
Edit Step / collect
General
Name: collect
Subprocedure
Subprocedure: EC-ServiceNow : CollectReportingData Change
Resource:
Parameters
Configuration name: config Required
Source table incident Required
Filter {"incident_state":"6;7"}
Preview mode:
Field mapping: "Stratus":releaseName, "Default":releaseProjectName, dv_number:incidentId, dv_category:category, subcategory:subCategory, dv_cmb_ci:configurationItem, dv_priority:priority, dv_state:status, dv_opened_by:reportedBy, resolved_at:resolvedOn, sys_created_on:createdOn, sys_updated_on:modifiedOn,
Transform script: sub transform { my ($context, $payload) = @_; # $payload->{customId} = 'custom'; return $payload; }
Metadata propertysheet path:
Debug:

See the figure above for the default mapping used in the procedure created when the "ServiceNow Setup for Command Center - incident" procedure runs. You can update this mapping manually for your specific requirements. See the EC-ServiceNow plugin help file for details on field mappings and other input parameters for the CollectReportingData procedure.

Continuous Integration Dashboard

The Continuous Integration dashboard displays key metrics for builds that can be used to gain insight into your organization's build throughput and to bubble up any potential build issues. The Continuous Integration dashboard provides indicators to measure agility of development.



Filters

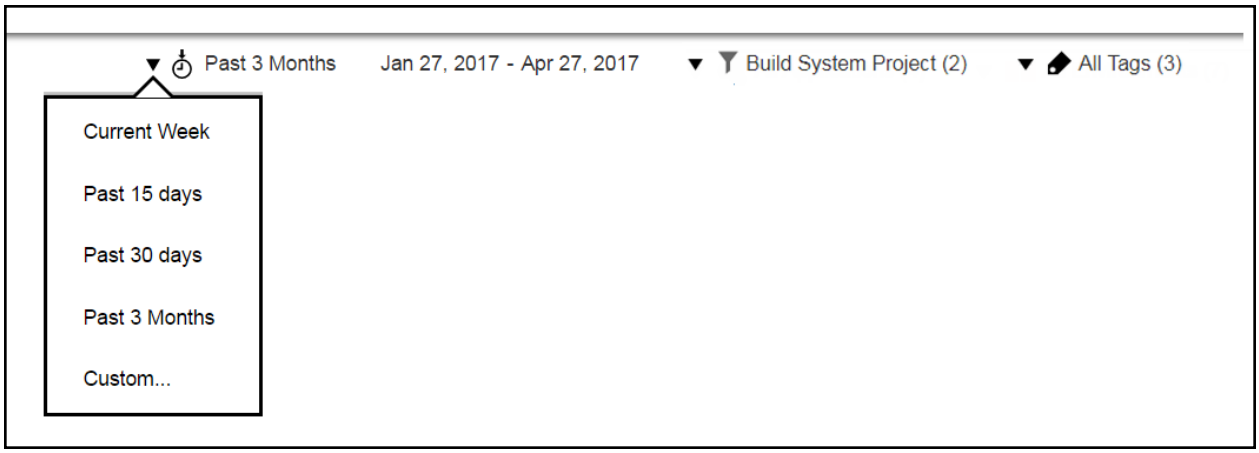
This dashboard provides drop-down menus that let you filter by time period, project, and tags:

Time Filter

Filters the builds based on the selected time range. The time filter applies to the build end time, which is set as the timestamp field value for the build records.

By default, the build metrics for the past three months appear. You can change the date range by selecting the drop-down menu on the top of the dashboard. In addition to the preset ranges such as

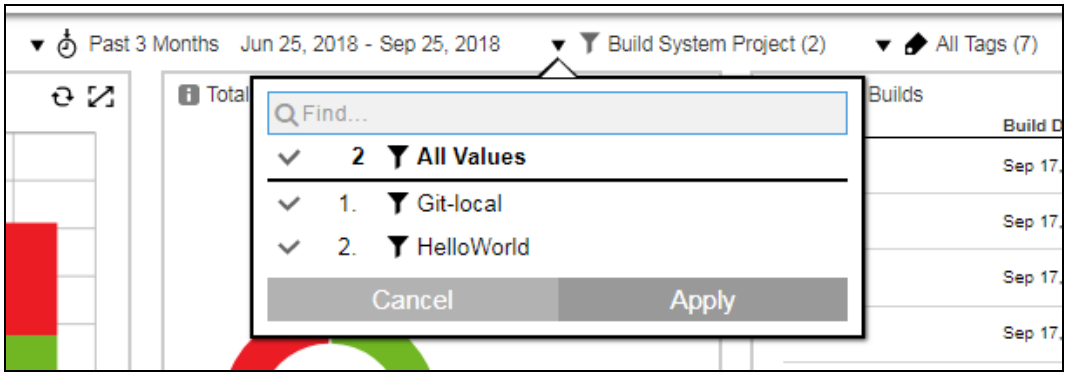
Current Week and Past 15 days, you can also specify a custom date range by using the **Custom...** option.



Build System Project Filter

Filters the builds based on the project that the builds are associated with in the native build system using the projectName field. For ElectricFlow builds, this is the project name of the build procedure, while for Jenkins, this is the Jenkins project name.

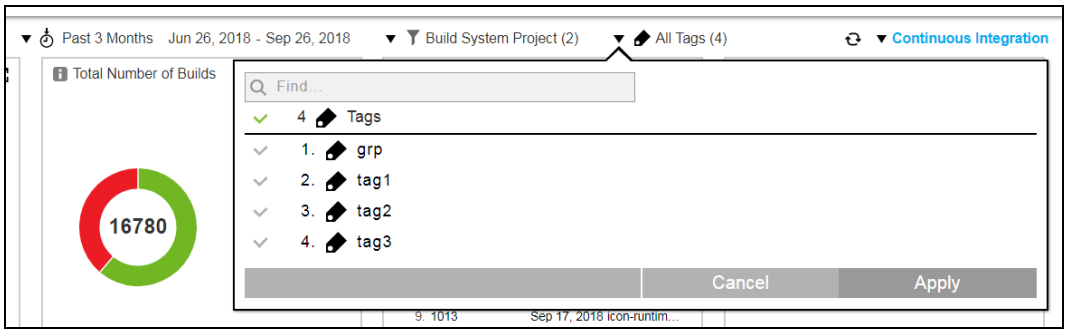
By default, build metrics for all build system projects appear. You can choose specific build system projects by selecting the drop-down menu for projects at the top of the dashboard. The build metrics are filtered based on those projects.



Tags Filter

Filters the builds based on the tags marked on the builds. For ElectricFlow builds, these are the tags set on the build job and tags set on the build procedure, which are passed on to the build job.

By default, build metrics for all builds appear. You can choose specific builds marked with specific tags by selecting the drop-down menu for tags at the top of the dashboard. The build metrics are filtered based on those tags.

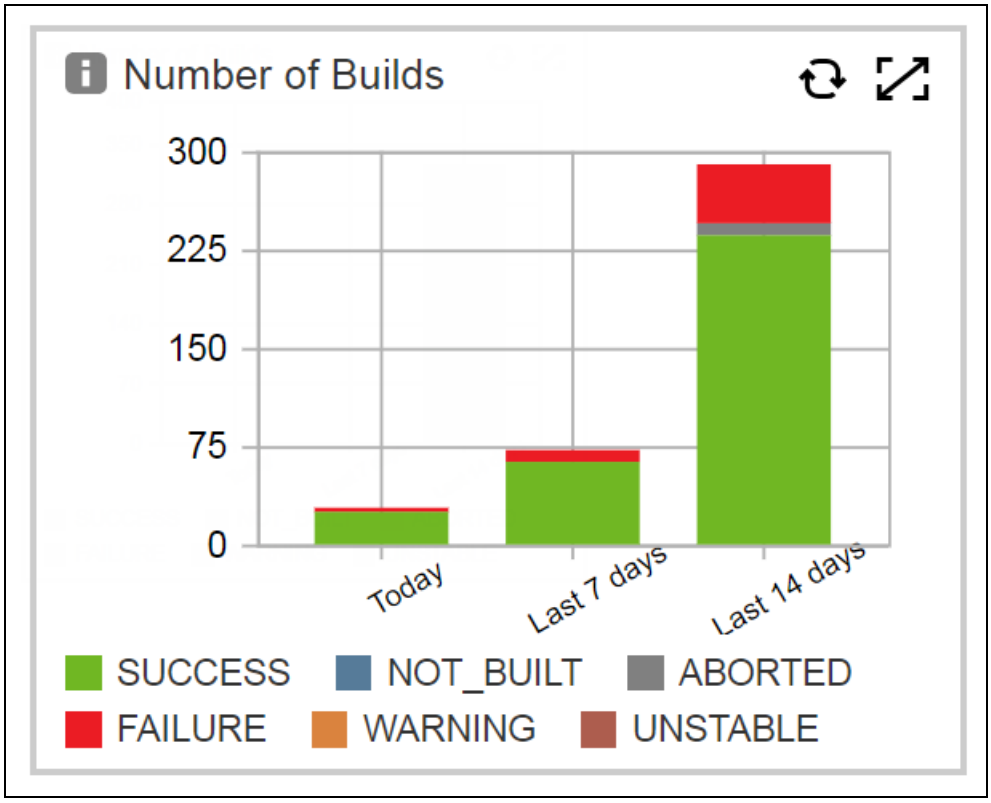


Examples are deployment tags such as *prod* or *UAT* and feature tags from JIRA such as *webapp*. For more information about tags, see the "Object Tags" section in the "Introduction to ElectricFlow" chapter.

Visualizations

Number of Builds

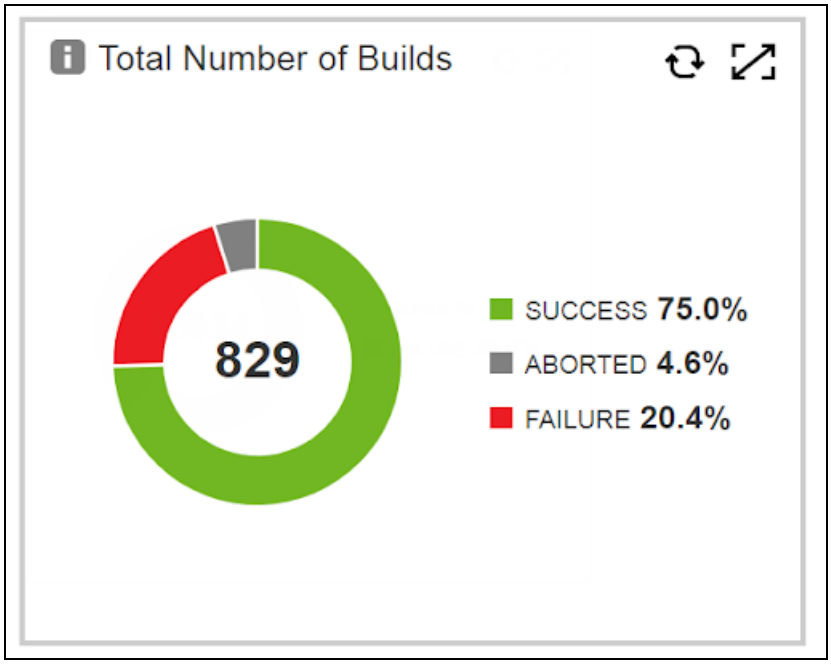
This chart provides the number of builds that have occurred today, during the last seven days, and during the last 14 days:



These durations are fixed and are therefore unaffected by the time filter.

Total Number of Builds

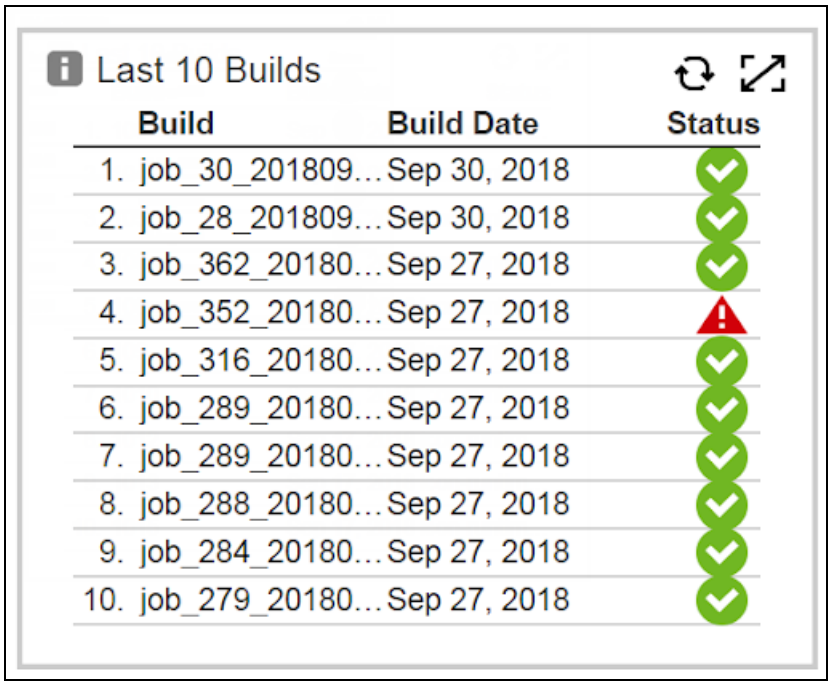
This chart provides the total number of builds that have occurred and shows a breakdown by different build outcomes. These are the possible outcomes: SUCCESS, FAILURE, UNSTABLE, NOT_BUILT, ABORTED, and WARNING.



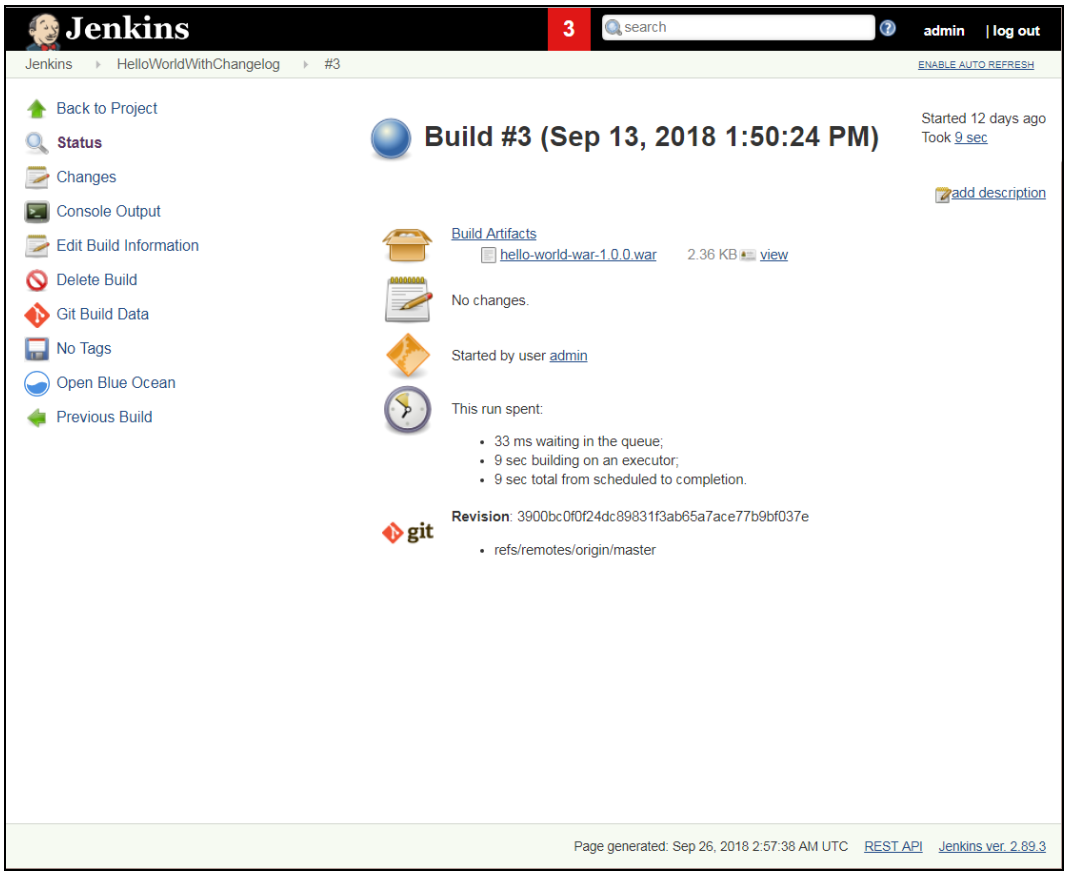
The duration depends on the time range selected using the Time Reference filter at the top of the dashboard.

Last 10 Builds

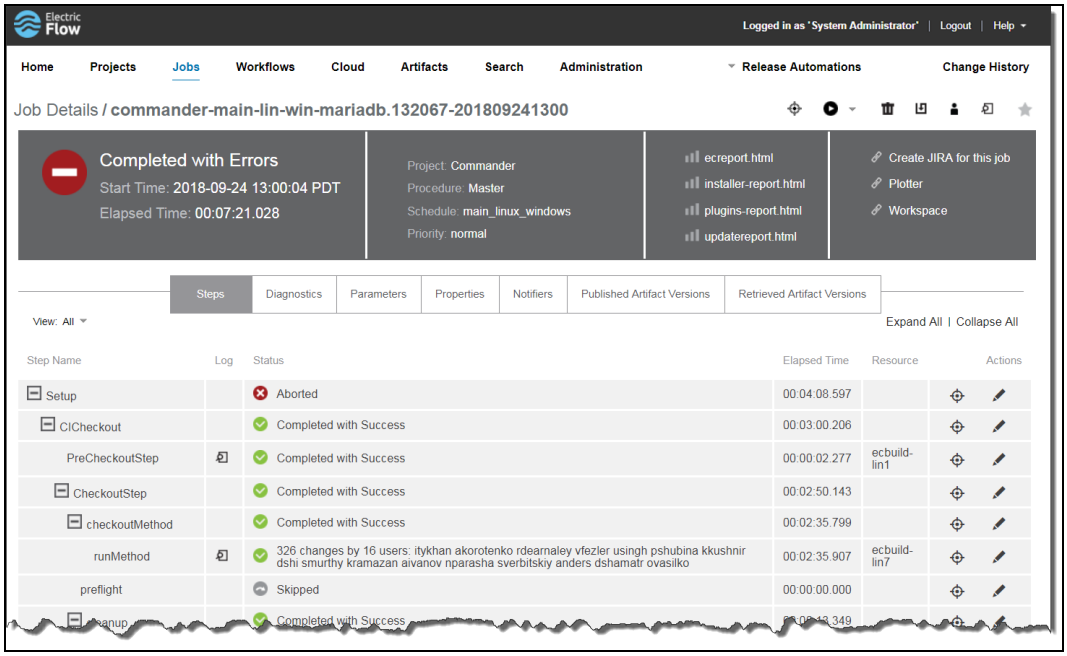
This report lists the last ten builds:



You can drill down into a build by clicking that build. This opens your CI system such as Jenkins or ElectricFlow. For example, if you are using Jenkins:

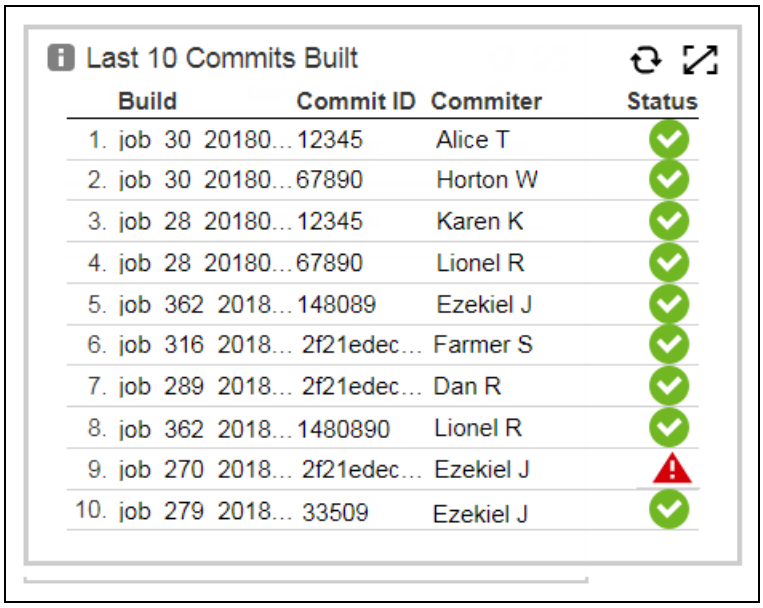


Also, for example, if you are using ElectricFlow:



Last 10 Commits Built

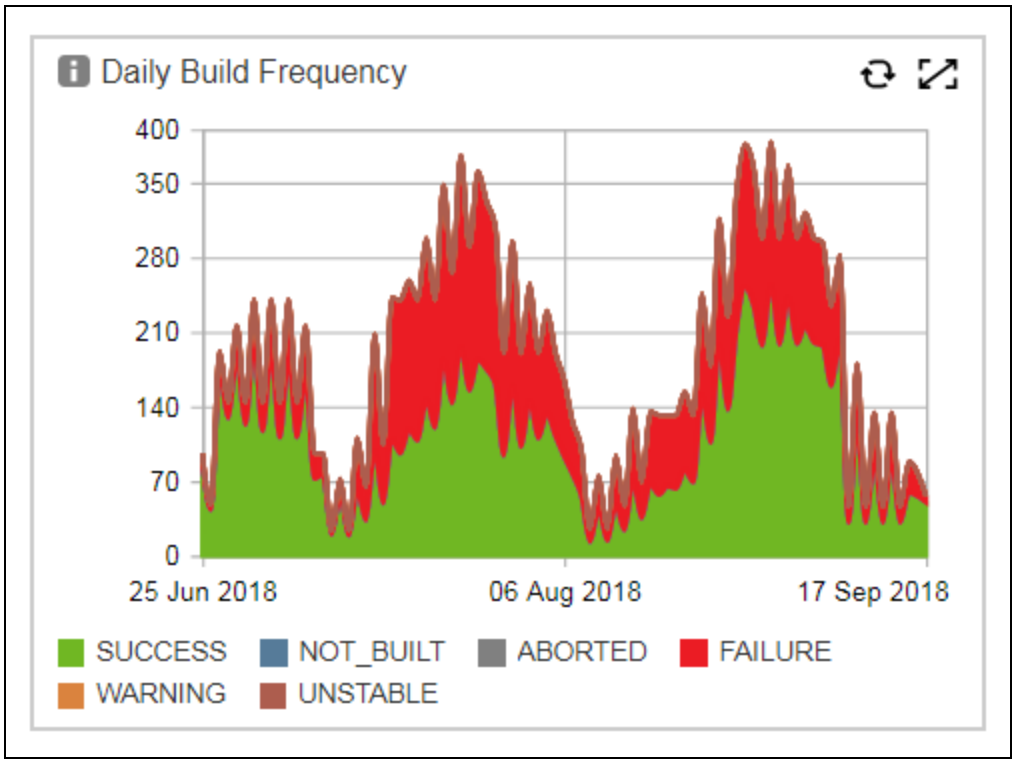
This chart shows the last ten code commits that have been built:



You can drill down into a build by clicking that build to open the build details page in your CI system such as Jenkins or ElectricFlow as described above.

Daily Build Frequency

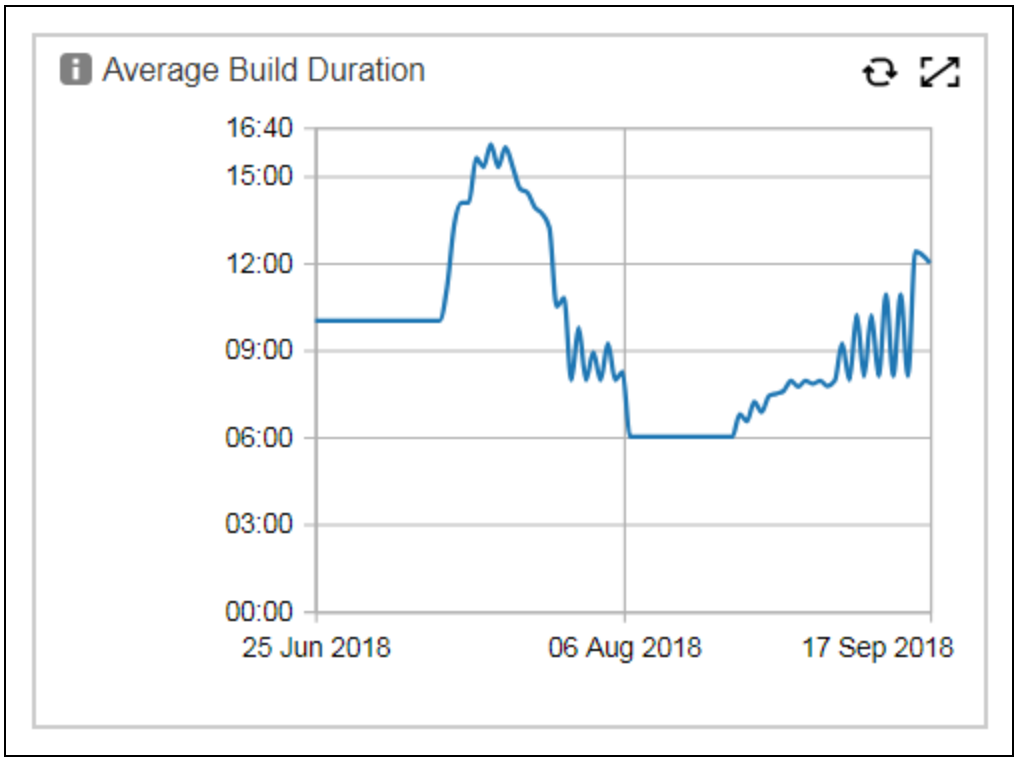
This chart shows the number of builds that have occurred over time:



The duration depends on the time range selected using the Time Reference filter at the top of the dashboard.

Average Build Duration

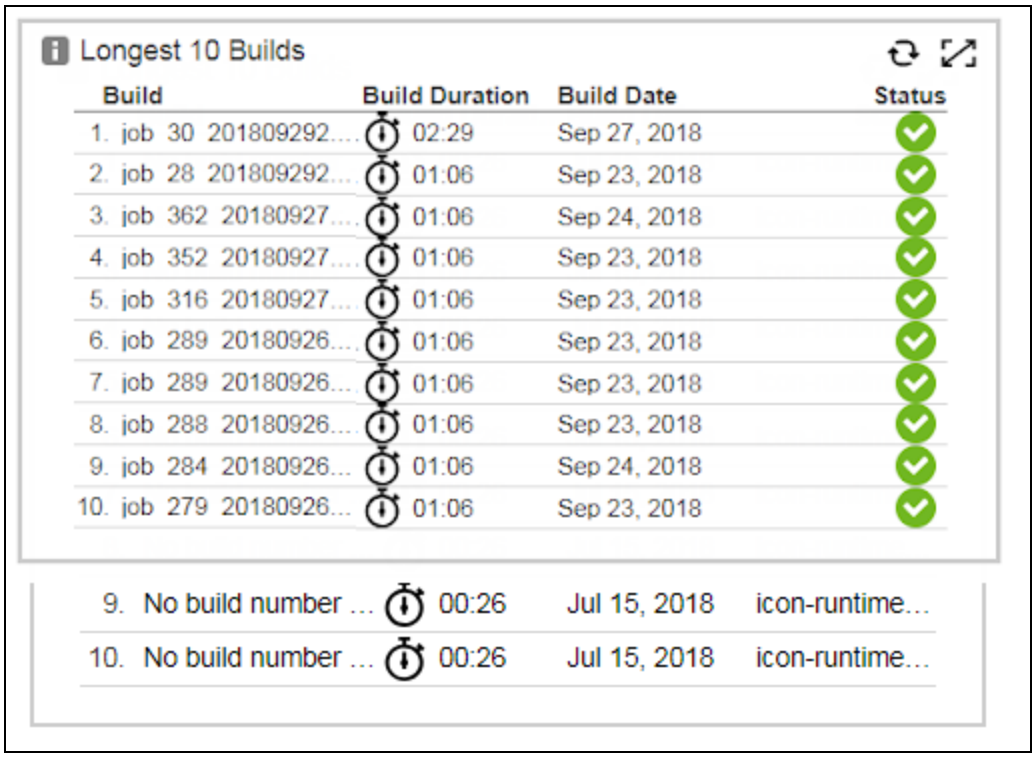
This chart shows the average build duration over time:



The duration depends on the time range selected using the Time Reference filter at the top of the dashboard.

Longest 10 Builds

This report lists the ten longest builds that have occurred:

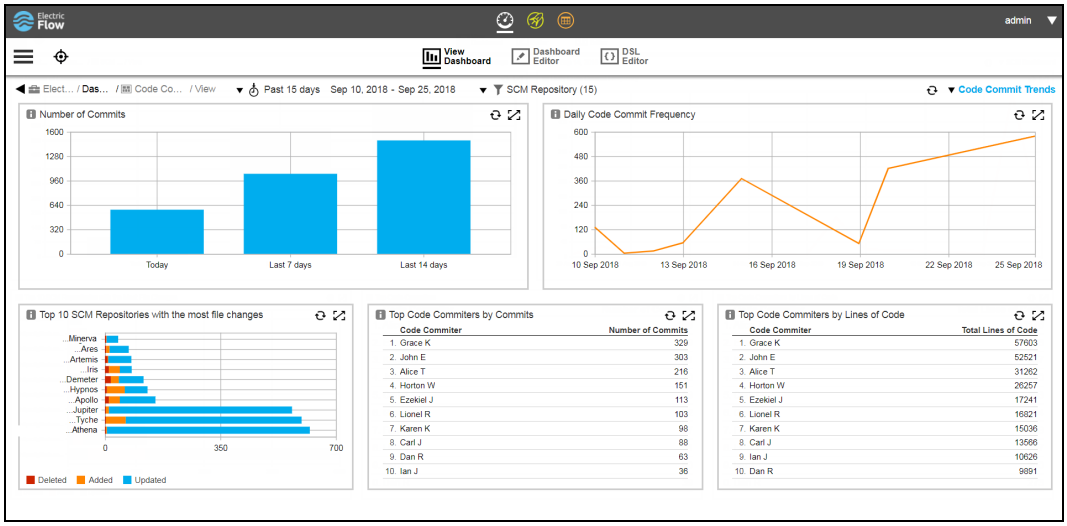


The duration depends on the time range selected using the Time Reference filter at the top of the dashboard.

You can drill down into a build by clicking that build to open the build details page in your CI system such as Jenkins or ElectricFlow as described above.

Code Commit Trends Dashboard

The Code Commit Trends dashboard lets you monitor the code velocity over time across teams and SCM repositories.



Filters

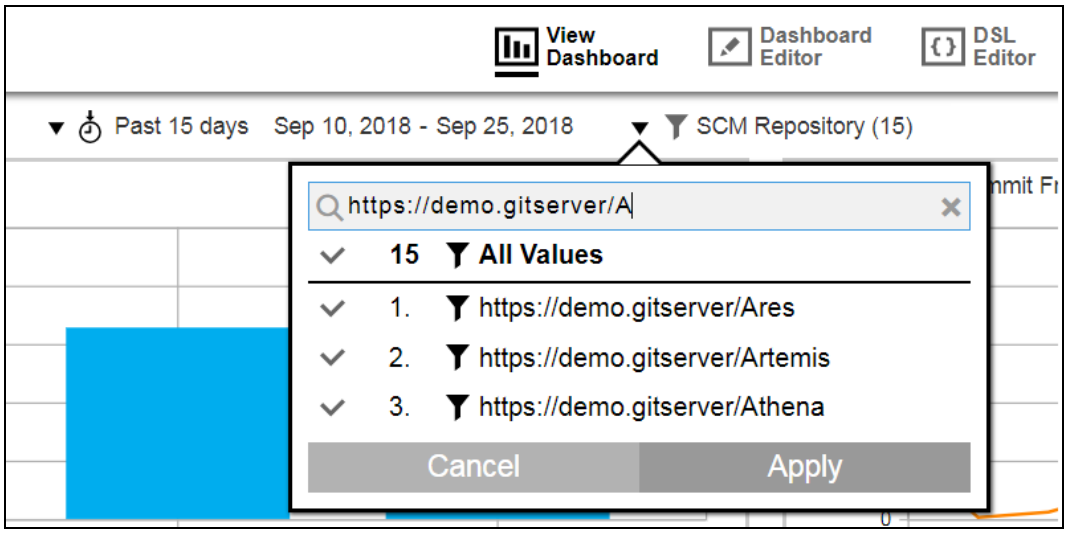
This dashboard provides drop-down menus that let you filter by time period and SCM Repository.

Time Filter

Filters the commits based on the selected time range. The time filter applies to the commit date.

SCM Repository Filter

Filters the commits based on specific repositories.



Adding Repositories to the SCM Repository Filter List

The Code Commit Trends dashboard shows metrics for code commits made to your SCM repositories. Out-of-the-box, the ECSCM-Git plugin is enabled to collect the code commit data from Git and GitHub repositories and send it to the DevOps Insight server. The Git Setup for DevOps Insight procedure lets you configure the plugin and set up schedules to get the details periodically on the latest commits that were submitted by developers to the Git repositories.

The SCM Repositories filter shows the repositories for which the DevOps Insight server has collected commit data. By default, a maximum of 20 repositories appear. You can type in the search text box to retrieve repositories that start with the typed-in text.

To run the Git Setup for DevOps Insight procedure:

1. Go to https://<*ElectricFlow_server*>/commander/.
2. Click **Administration** > **Projects** > **Electric Cloud**.

   The list of procedures for the Electric Cloud project appears.

3. Click the  (Run Immediately) button for the Git Setup for DevOps Insight procedure.

4. Enter your parameter values into the dialog box.

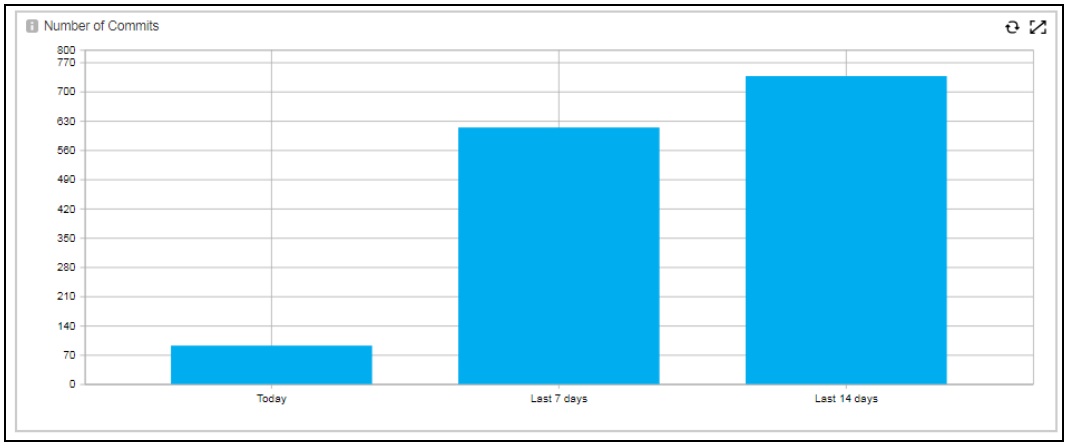
   For descriptions of the parameters for this procedure, see the online Help file by clicking **Administration** > **Projects** > **Electric Cloud** > **Git Setup for DevOps Insight** > **Help**.

5. Click **OK**.

Visualizations

Number of Commits

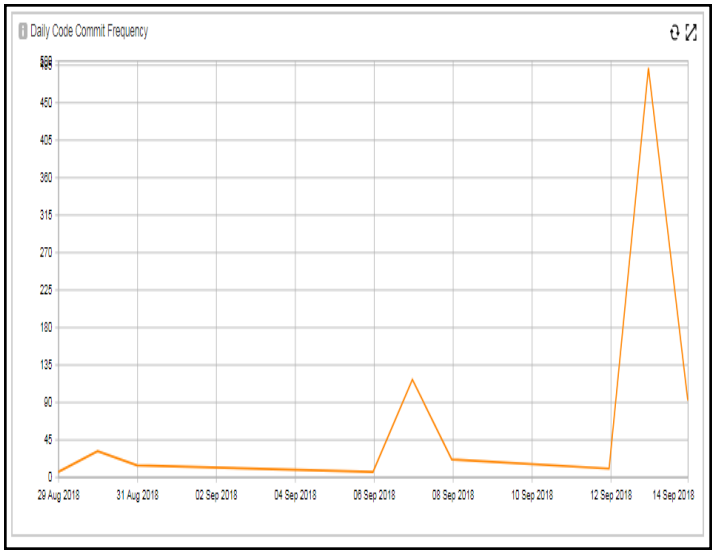
This chart provides the total number of commits today, during the last seven days, and during the last 14 days:



These durations are fixed and are therefore unaffected by the time filter.

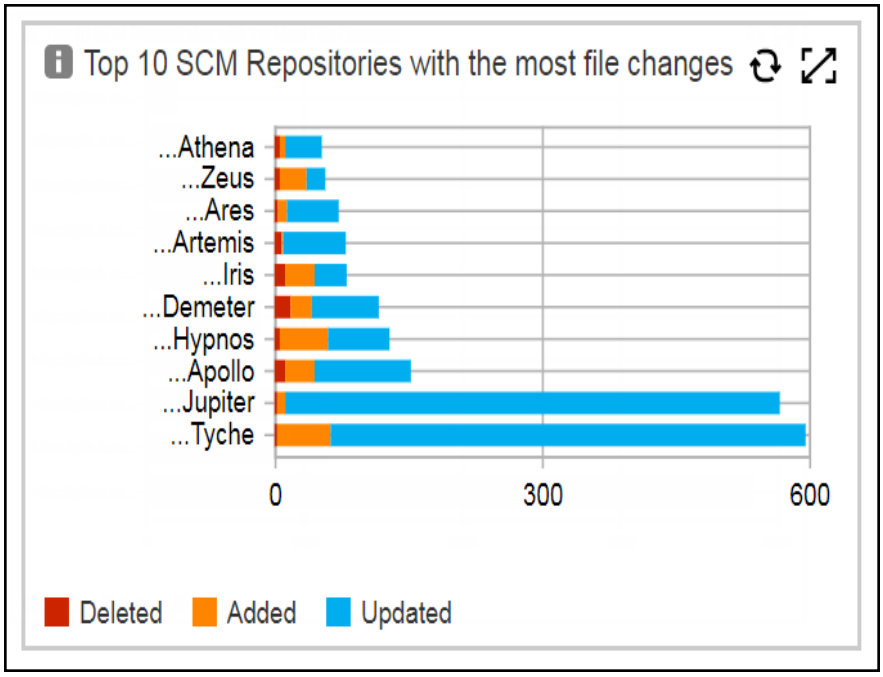
Daily Code Commit Frequency

This chart provides the number of code commits made per day:



Top 10 SCM Repositories with the Most File Changes

This list shows you the top 10 SCM repositories with the most number of files added, removed, or updated:



Top Code Commiters by Commits

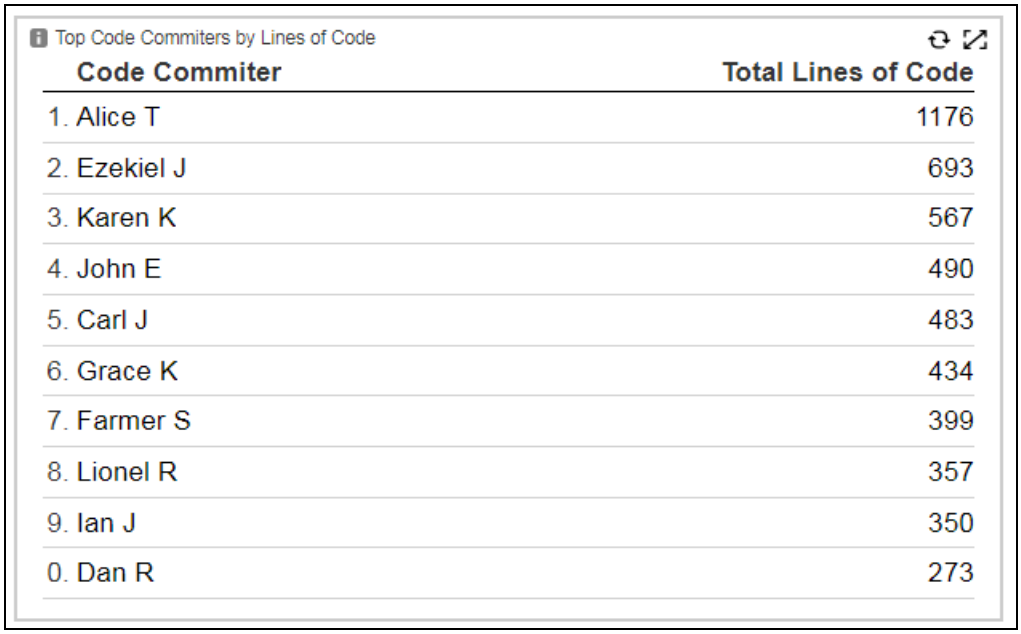
This list shows the top 10 code commiters with the highest number of commits:

Top Code Commiters by Commits

Code Commiter	Number of Commits
1. Alice T	168
2. Ezekiel J	99
3. Karen K	81
4. John E	70
5. Carl J	69
6. Grace K	62
7. Farmer S	57
8. Lionel R	51
9. Ian J	50
0. Dan R	39

Top Code Commiters by Lines of Code

This list shows the top 10 code commiters with the most number of lines of code added, removed, and updated:


Top Code Commiters by Lines of Code

Code Commiter	Total Lines of Code
1. Alice T	1176
2. Ezekiel J	693
3. Karen K	567
4. John E	490
5. Carl J	483
6. Grace K	434
7. Farmer S	399
8. Lionel R	357
9. Ian J	350
0. Dan R	273

Authoring DevOps Insight Reports

The DevOps Insight Report Editor lets you define a report based on criteria such as filters (search criteria), aggregations, and sorting. You create the query definition for a report by using the Report Editor.

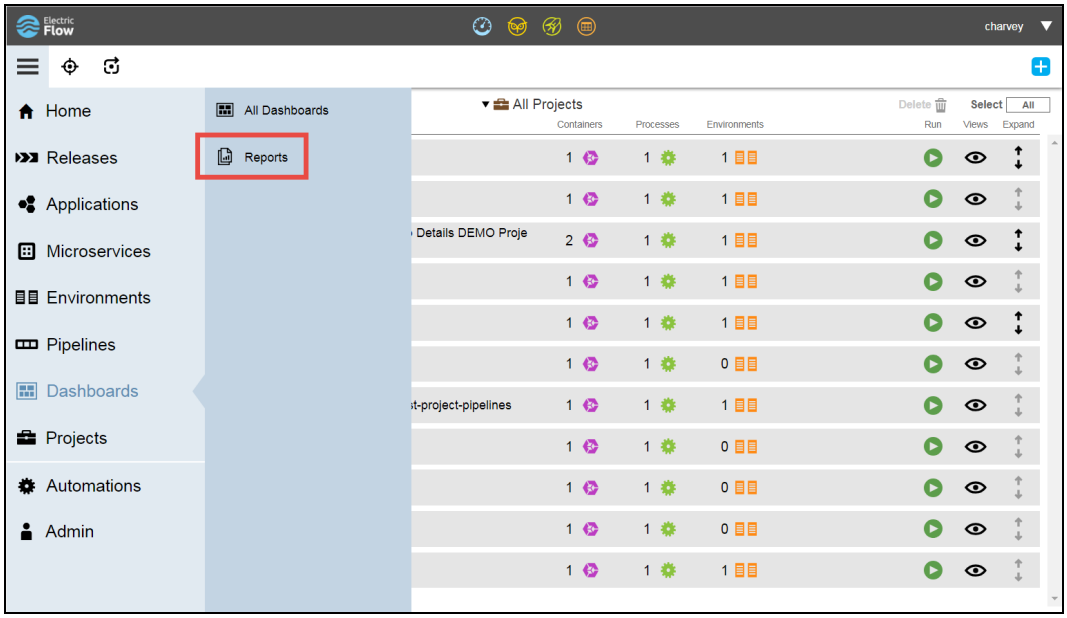
These reports are based on Elasticsearch Query DSL and provide an interface to common Elasticsearch queries. The Report Editor includes an Advanced mode, which lets you edit the query DSL directly.

You select from a list of available reports when you create a DevOps Insight dashboard widget. This is described in Defining the Dashboard Widgets on page 795 and Adding Widgets to an Existing DevOps Insight Dashboard.

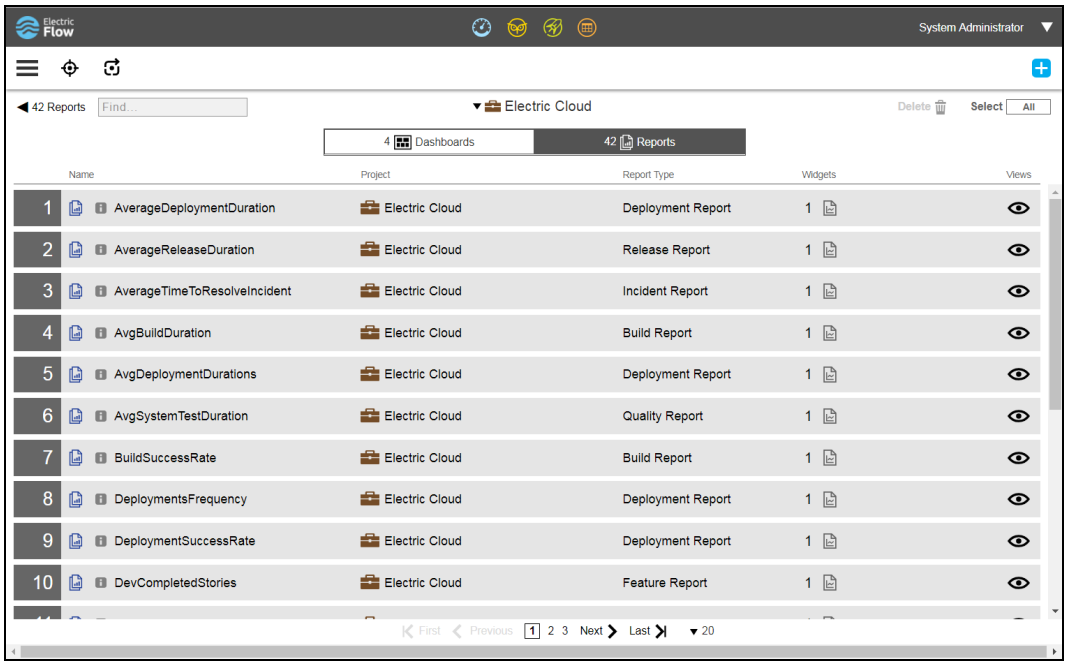
This section covers the following topics:

Viewing the List of Reports

To open the Reports list, click **Dashboards** > **Reports** from the (main) menu in the upper left corner of the Deploy UI:

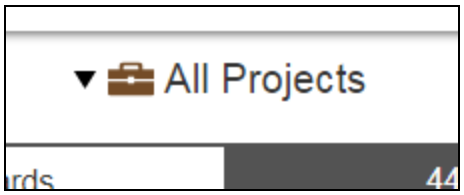


The Reports list opens. The Reports list shows all of the reports that are available in the currently-selected project or projects. For example, the following Reports list shows the reports in a project named Electric Cloud:



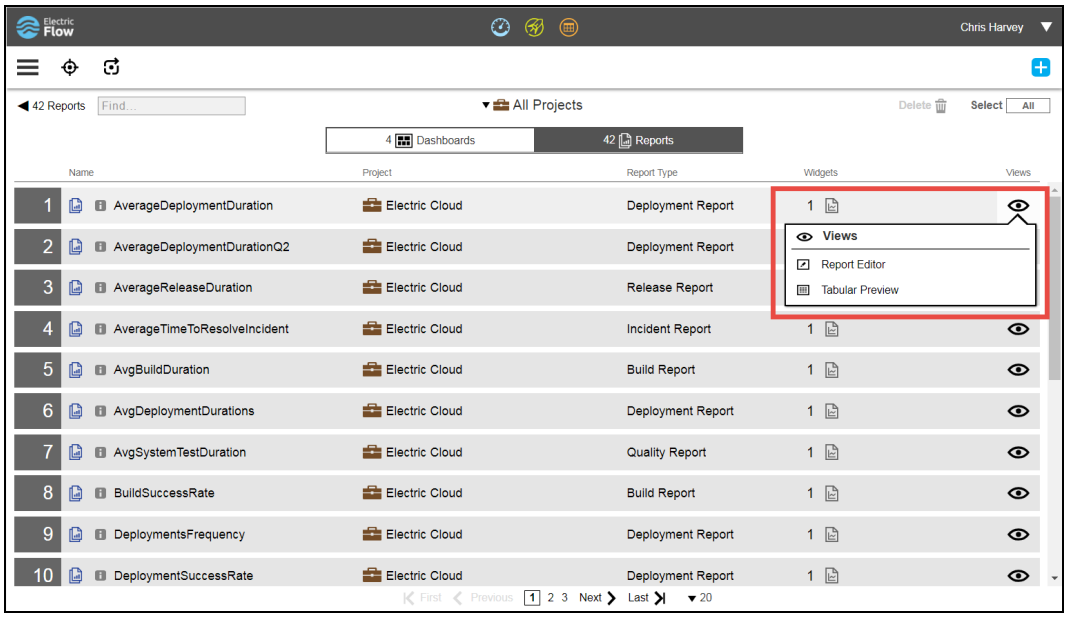
This list comprises reports included with ElectricFlow as well as any custom reports that you have created.

By default, the reports are listed for all projects. You can filter this list by any combination of projects via the projects menu:



A variety of "stock" reports are included with ElectricFlow. These reports can help you learn about the reporting functionality and how to create your own reports. You can also make copies of these reports to use as templates for creating reports. These reports are in the Electric Cloud project and are not editable.

The (Views) button provides quick access to the Report Editor or the Tabular Preview for a specific report:



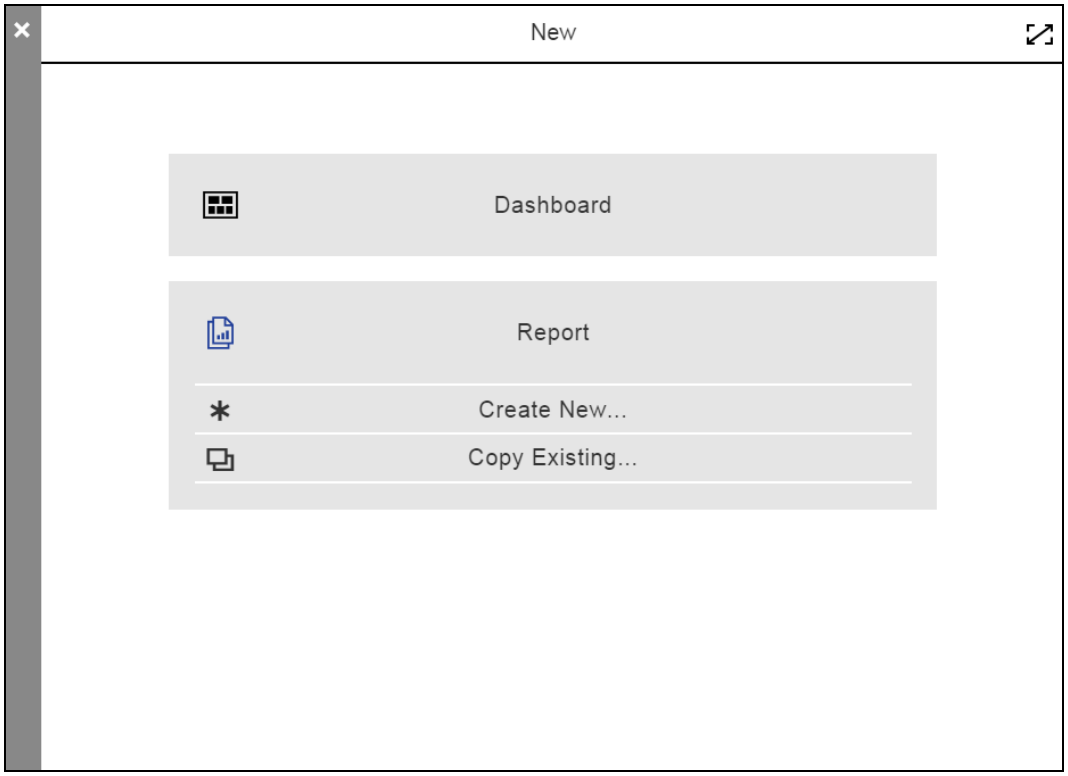
Creating a Custom Report

To create a report:

1. From within the Reports list, click the (new DevOps Insight dashboard or report) button.

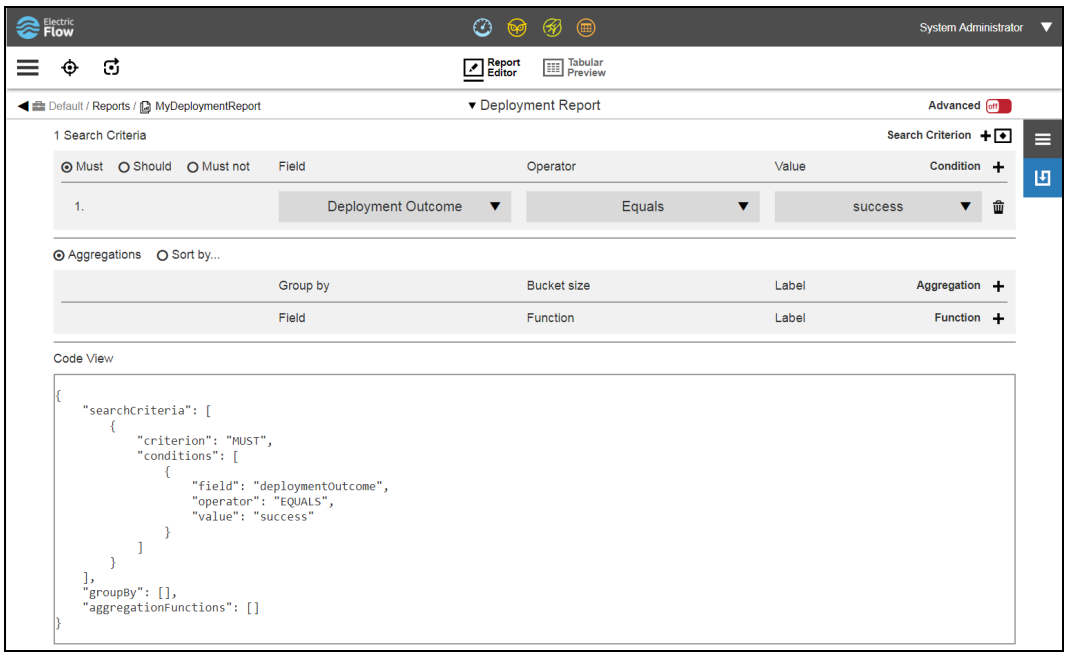
   Tip: This button is also available in the DevOps Insight Dashboards list and in any DevOps Insight dashboard.

The following dialog box appears:



2. Click **Report** > **Create New...**.

   The DevOps Insight Report Editor appears:

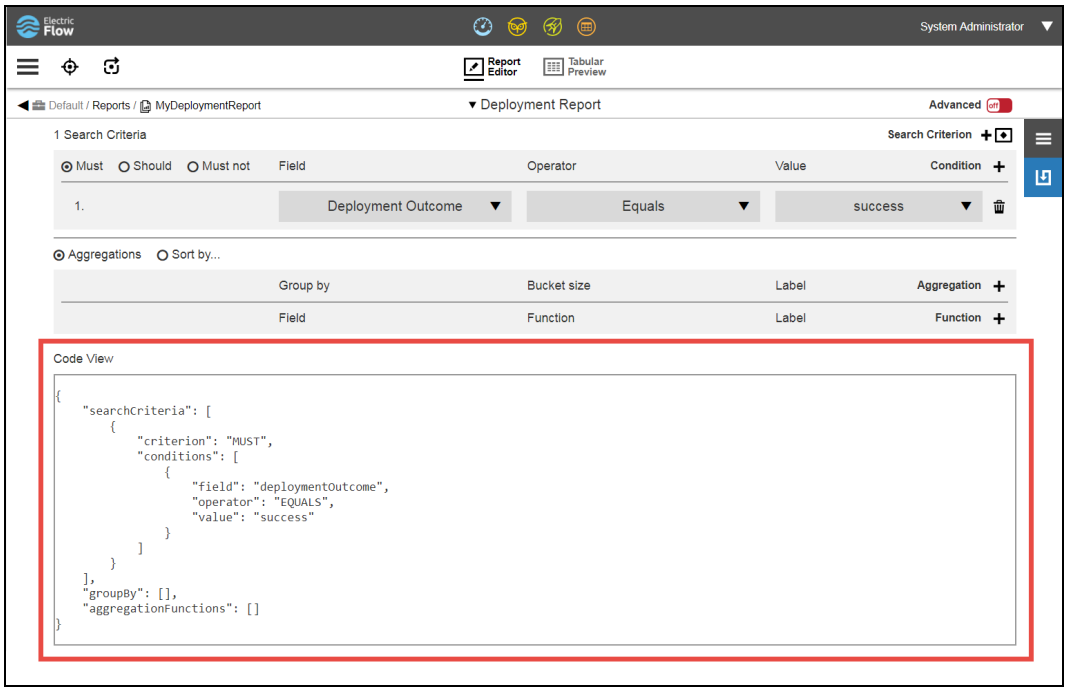


   This shows the standard editing mode, which is the default mode when you open the Report Editor. The editor lets you define any number of search criteria and aggregation types for creating custom reports.

Editing a Report's Query by Using Standard Mode

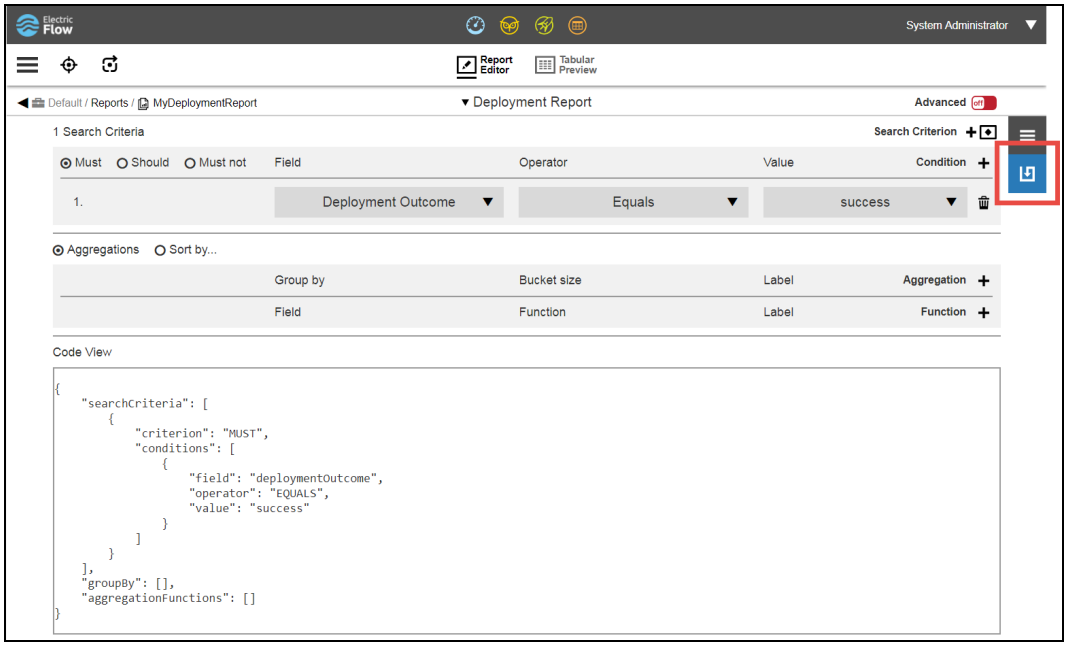
1. Use the Report Editor to create your report's query.

   The **Code View** window shows a read-only view of the underlying DSL code for your query. For example:



This view changes dynamically as you modify the report.

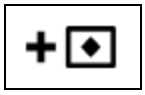
2. Click the  (Save) button on the right side of the page to save your changes:



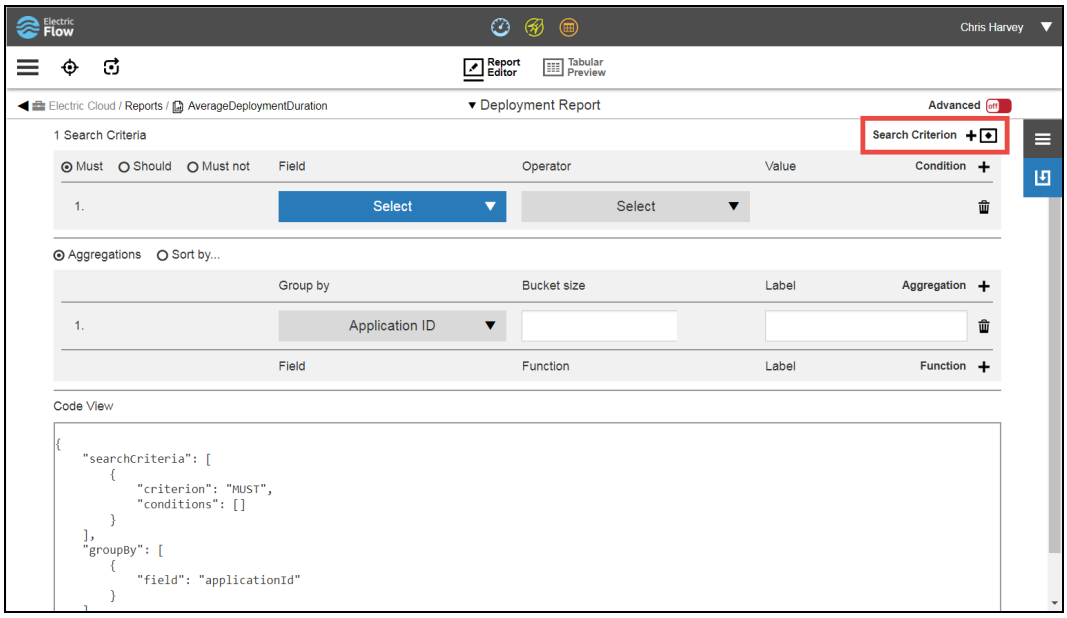
You must save the report if you want to subsequently edit the underlying DSL code directly (described in Editing a Report's Query DSL Code Directly by Using Advanced Mode on page 776).

Report Search Criteria

The list of search criteria determines the matching for specific object attribute values. You can add one



or more search criteria by clicking the (add search criterion) button in the Report Editor:



Object Attribute Fields

The **Fields** pulldown displays the list of available object attributes. Examples are Application ID, Job Name, Planned Start Time, Task Type, and Wait for Planned Start.

Search Operators

The following search operators are available:

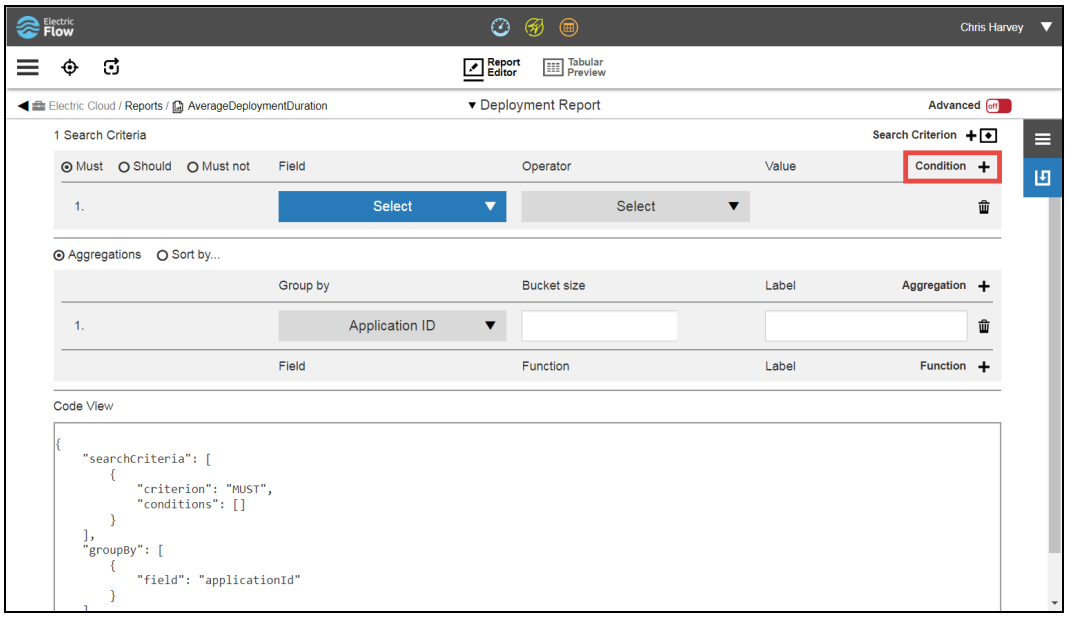
Operator	Applicable Attribute Data Types
Equals	All
Exists	All
Not Exists	All
Greater than	All data types except BOOLEAN
Greater than or Equals	All data types except BOOLEAN
Less than	All data types except BOOLEAN
Less than or Equals	All data types except BOOLEAN
Between	All data types except BOOLEAN

Values to Match

The field in the **Value** column represents the particular search value to match.

Adding Search Conditions

You can add conditions to the search criteria by clicking the (add condition) button:

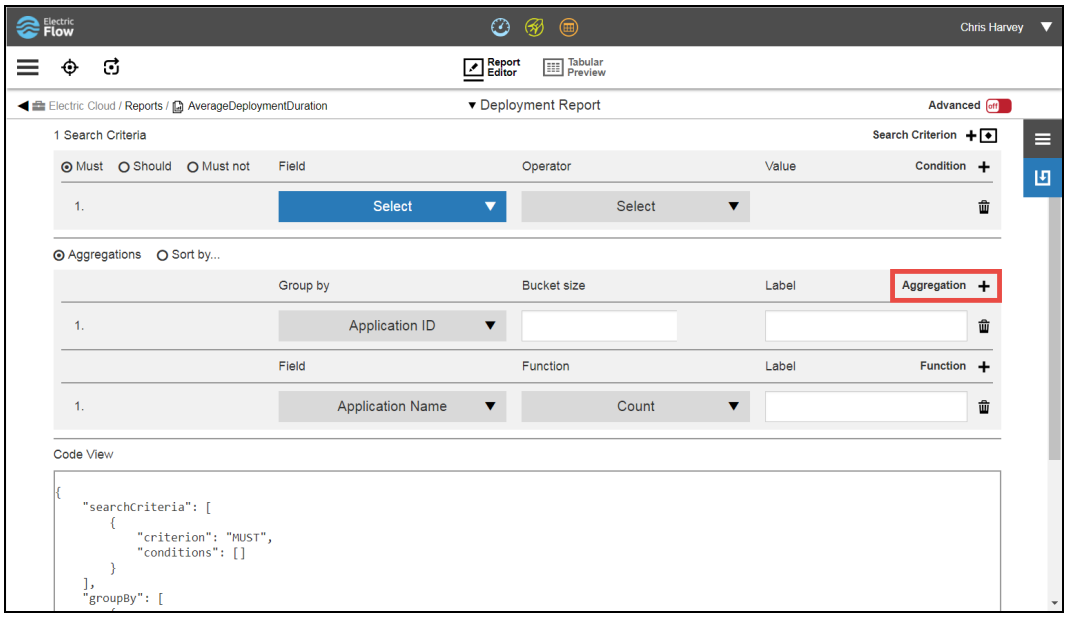


Aggregations or Sorting

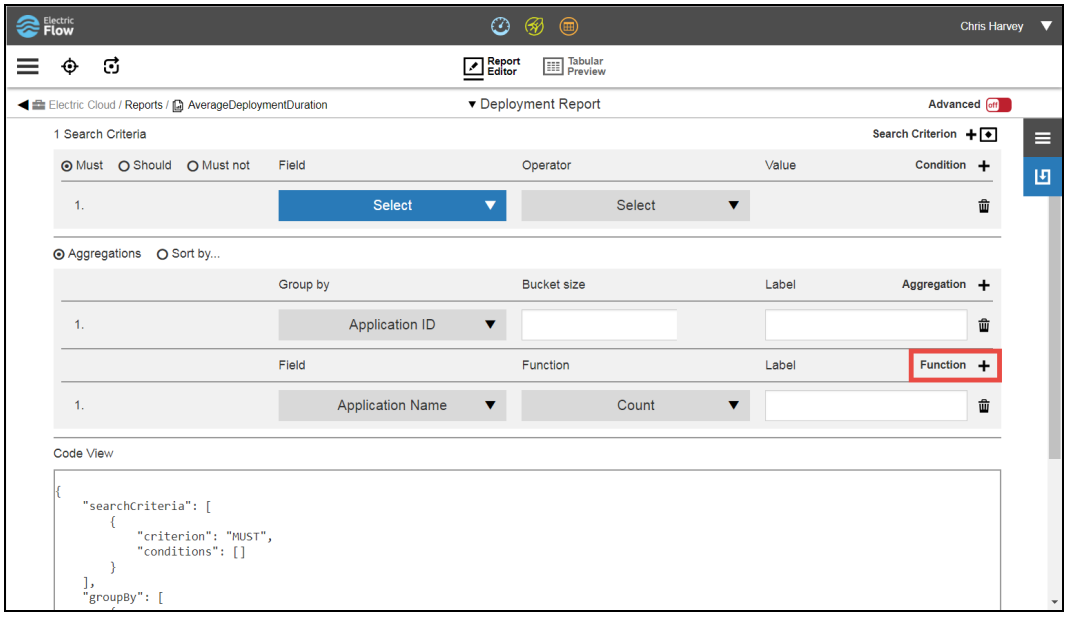
Aggregations or sorting determine the grouping or order of the query results. You can choose either aggregation or sorting (but not both).

Query Result Aggregations

You can add aggregations by clicking the (add aggregation) button:



You can add functions by clicking the (add function) button:



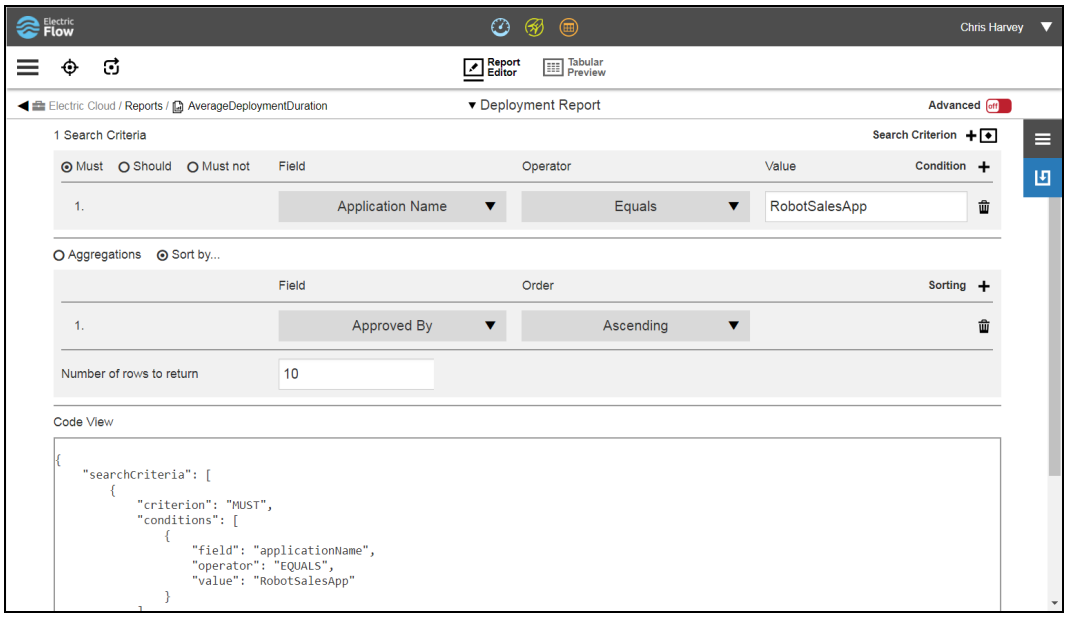
The following functions are available:

Function	Applicable for Attribute Types
Average	NUMERIC, PERCENT

Function	Applicable for Attribute Types
Count	All
Distinct Count	All
Max	NUMERIC, PERCENT, DATE, DATETIME
Min	NUMERIC, PERCENT, DATE, DATETIME
Sum	NUMERIC, PERCENT

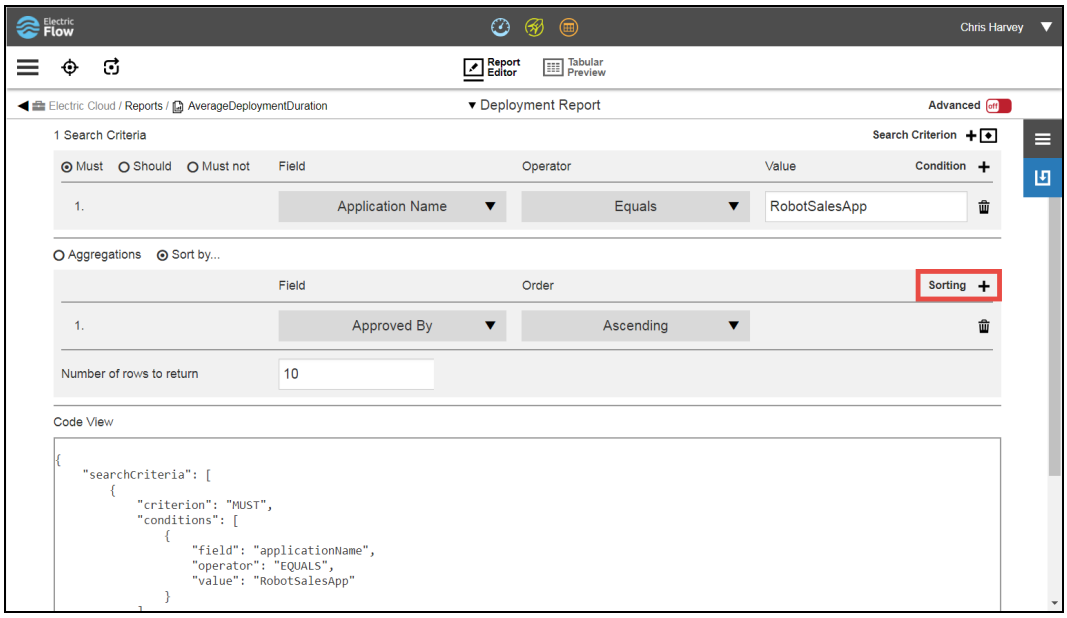
Query Result Sorting

As an alternative to aggregating the query results, you can sort them. For example:



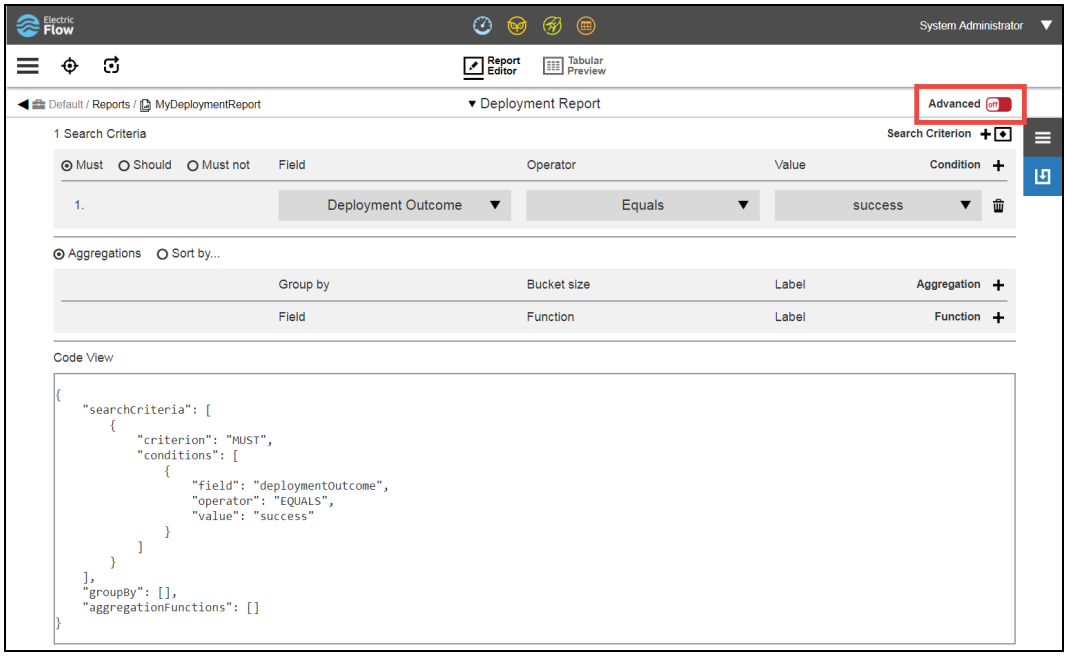
You can sort the query results by a specific object attribute field in ascending or descending order. You can specify the number of rows for the query to return.

To add a secondary sorting field, click the  (add sorting field) button:

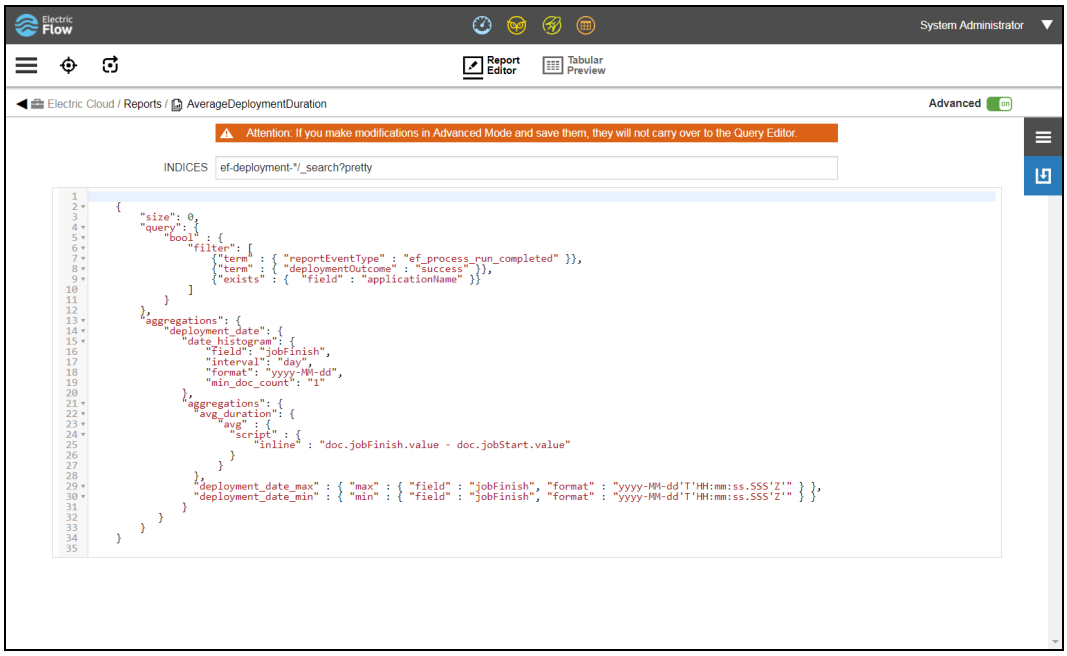


Editing a Report's Query DSL Code Directly by Using Advanced Mode

Advanced mode shows a view of the underlying query DSL code. You enter and exit Advanced mode by clicking the **Advanced** slider (toggle) button:



Advanced mode lets you edit the query DSL code directly. For example:



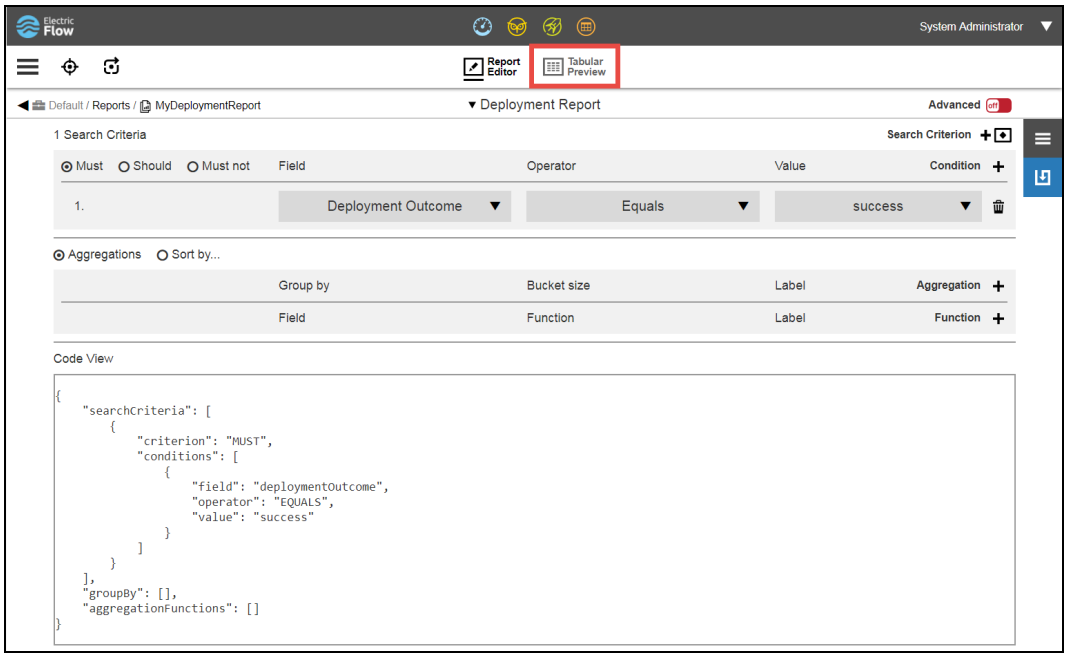
Note that editing the query DSL code makes it incompatible with Standard mode in the Report Editor. The following warning is always present in Advanced mode as a reminder:

Attention: If you make modifications in Advanced Mode and save them, they will not carry over to the Query Editor.

For details about Elasticsearch Query DSL, see the Elastic Stack and Product Documentation web page.

Previewing Report Query Results by Using Tabular Preview

To check your query results, click the (Tabular Preview) button at the top of the page:

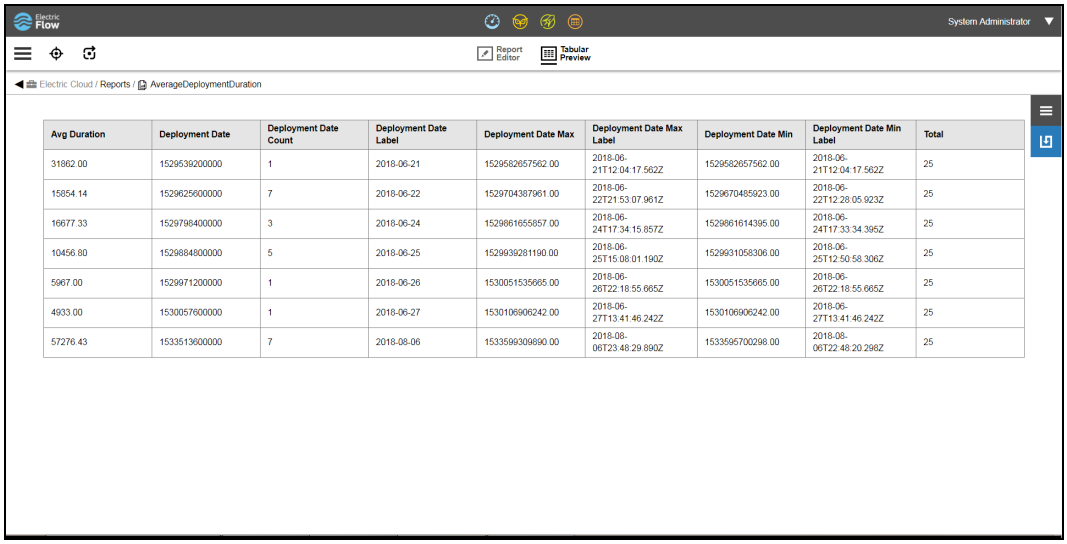


The Tabular Preview shows the first 20 rows of output so that you can test your report criteria. The Tabular Preview provides a preview of the report data from the DevOps Insight server.

The tables in the following examples describe some of the columns that might be returned based on the report that you have defined.

Example Report: AverageDeploymentDuration

This report is included in DevOps Insight.



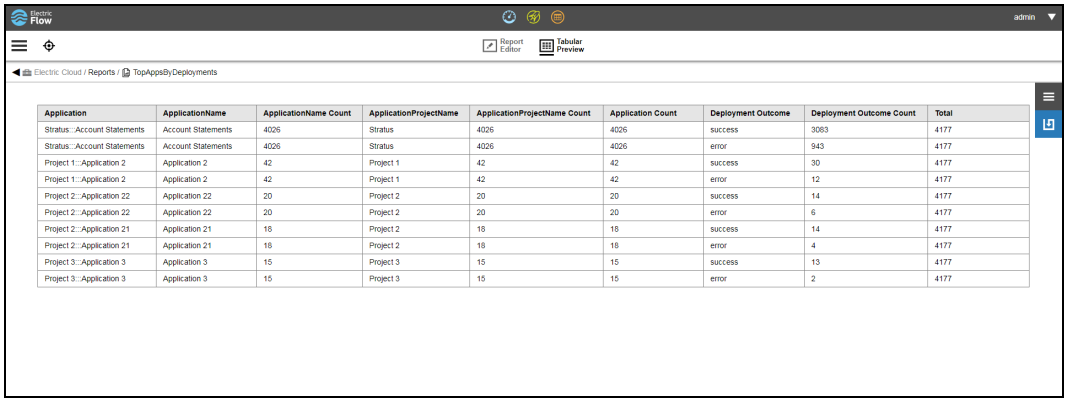
Avg Duration	Deployment Date	Deployment Date Count	Deployment Date Label	Deployment Date Max	Deployment Date Max Label	Deployment Date Min	Deployment Date Min Label	Total
31862.00	1529539200000	1	2018-06-21	1529582657562.00	2018-06-21T12:04:17.562Z	1529582657562.00	2018-06-21T12:04:17.562Z	25
15854.14	1529625600000	7	2018-06-22	1529704387961.00	2018-06-22T21:53:07.961Z	1529670485923.00	2018-06-22T12:28:05.923Z	25
16677.33	1529798400000	3	2018-06-24	1529861655857.00	2018-06-24T17:34:15.857Z	1529861614395.00	2018-06-24T17:33:34.395Z	25
10456.80	1529884800000	5	2018-06-25	1529939281190.00	2018-06-25T15:08:01.190Z	1529931058306.00	2018-06-25T12:50:58.306Z	25
5967.00	1529971200000	1	2018-06-26	1530051535665.00	2018-06-26T22:18:55.665Z	1530051535665.00	2018-06-26T22:18:55.665Z	25
4933.00	1530057600000	1	2018-06-27	1530106906242.00	2018-06-27T13:41:46.242Z	1530106906242.00	2018-06-27T13:41:46.242Z	25
57276.43	1533513600000	7	2018-08-06	1533599309890.00	2018-08-06T23:48:29.890Z	1533595700298.00	2018-08-06T22:48:20.298Z	25

The following table shows some of the date-based fields stored in Elasticsearch for this report and their corresponding formatted fields.

Column	Description
Total	Number of records that matched the report search criteria
<given date field>. such as Deployment Date	Date in milliseconds since the epoch. This is the raw data as stored in Elasticsearch. For aggregated data, the column representing this data also has an equivalent column (named **Deployment Date Label**, for example) in which this data is converted to a user-friendly date
<given date field>`Label` such as Deployment Date Label	Date formatted as `yyyy-MM-dd`

Example Report: TopAppsByDeployments

This report is included in DevOps Insight. This is an example of a report containing aggregations.



Application	ApplicationName	ApplicationName Count	ApplicationProjectName	ApplicationProjectName Count	Application Count	Deployment Outcome	Deployment Outcome Count	Total
Stratus::Account Statements	Account Statements	4026	Stratus	4026	4026	success	3083	4177
Stratus::Account Statements	Account Statements	4026	Stratus	4026	4026	error	943	4177
Project 1::Application 2	Application 2	42	Project 1	42	42	success	30	4177
Project 1::Application 2	Application 2	42	Project 1	42	42	error	12	4177
Project 2::Application 22	Application 22	20	Project 2	20	20	success	14	4177
Project 2::Application 22	Application 22	20	Project 2	20	20	error	6	4177
Project 2::Application 21	Application 21	18	Project 2	18	18	success	14	4177
Project 2::Application 21	Application 21	18	Project 2	18	18	error	4	4177
Project 3::Application 3	Application 3	15	Project 3	15	15	success	13	4177
Project 3::Application 3	Application 3	15	Project 3	15	15	error	2	4177

Column	Description
Total	Number of records that matched the report search criteria
<aggregation field>`Count` such as Deployment Outcome Count	Number of records that fall into the aggregation bucket. For example, if you define a group by aggregation on deployment outcome, then for each unique deployment outcome such as `success` and `warning`, the **Deployment Outcome Count** column will return the number of deployments that completed with that deployment outcome

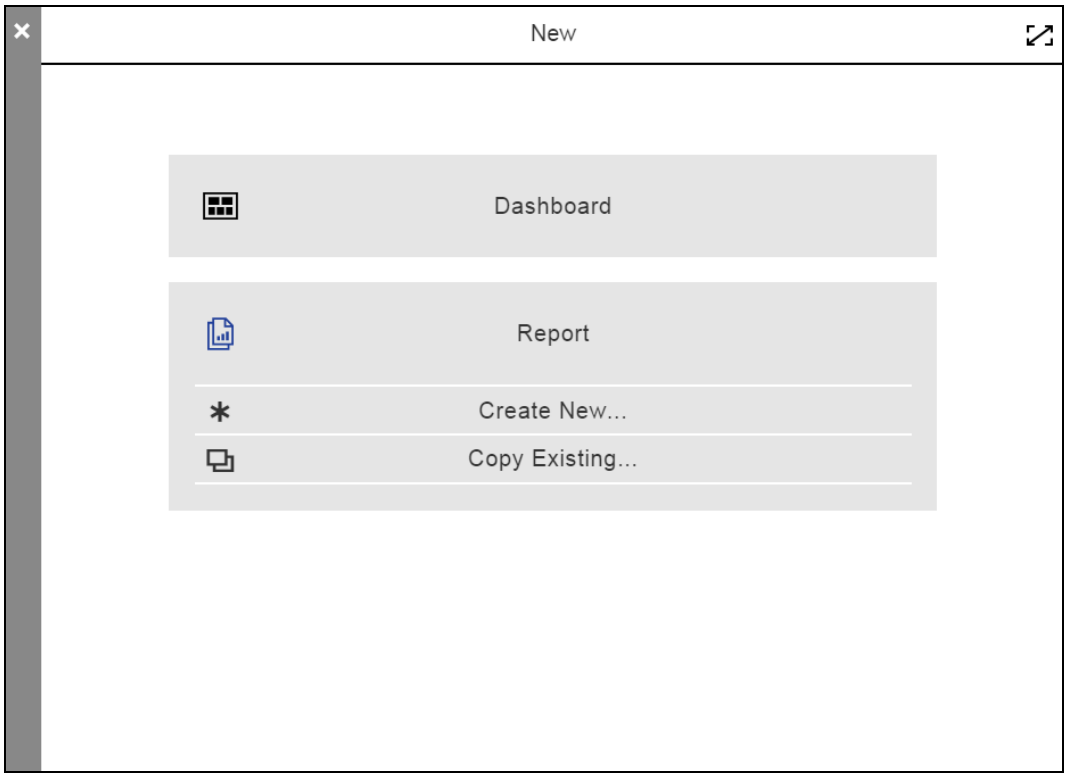
Creating a Custom Report by Copying an Existing Report

Copying another report lets you create a custom report more quickly. You can copy your custom reports as well as reports that are included with ElectricFlow. You cannot edit reports that are included with ElectricFlow, but copying them provides a substitute for customizing their functionality while preserving them to use as templates for future custom reports.

1. From within the Reports list, click the (new DevOps Insight dashboard or report) button.

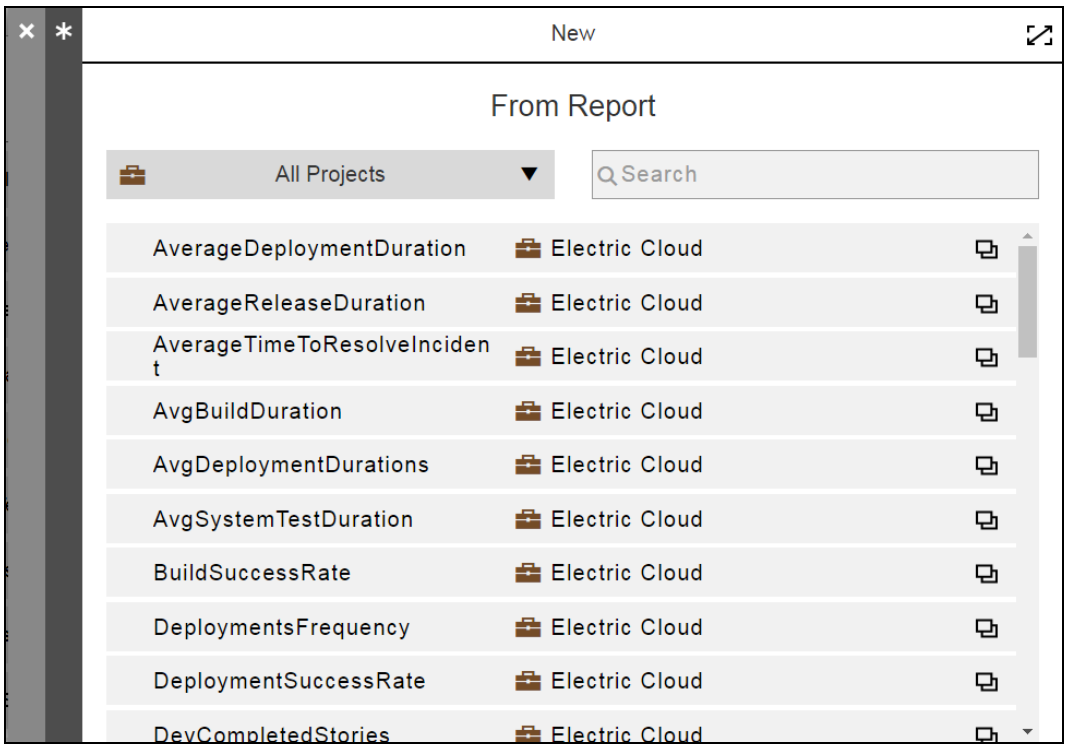
   Tip: This button is also available in the DevOps Insight Dashboards list and any DevOps Insight dashboard.

   The following dialog box appears:



2. Click **Report** > **Copy Existing...**.

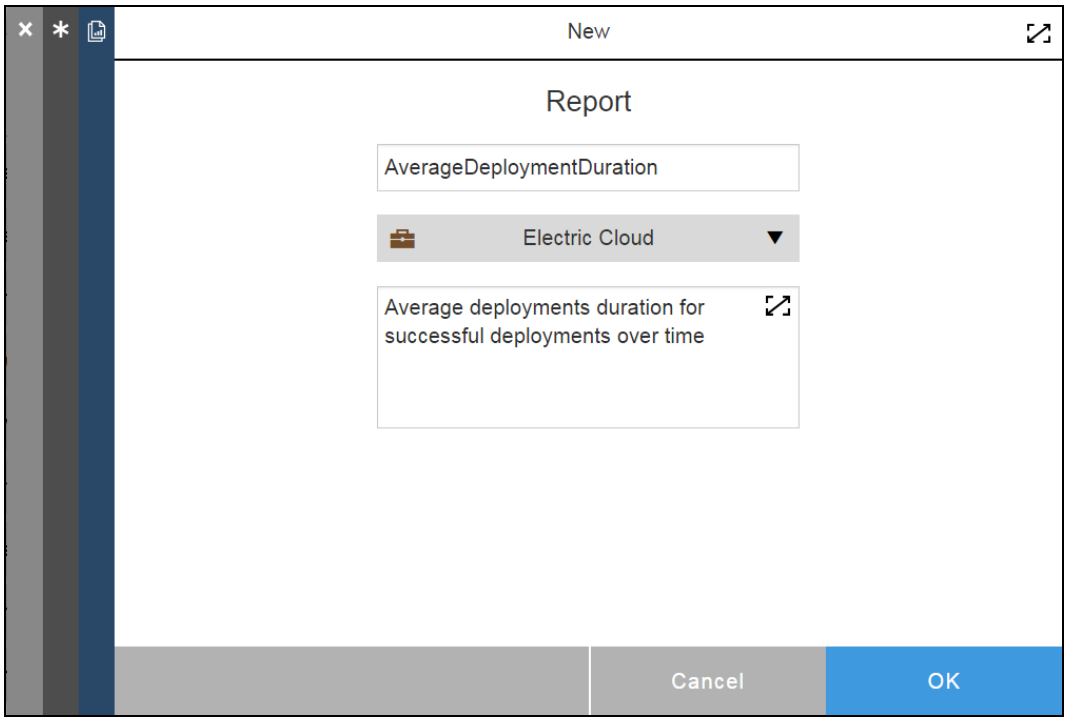
   The **From Report** dialog box appears. For example:



3. Click the name of the report that you want to copy.

   You can narrow down the list of projects by selecting one or more projects from the projects pulldown menu or by entering search terms into the **Search** field.

   The **New Report** dialog box appears. For example:



4. Enter the new report name into the dialog box, and then select the project that will contain the report.

   By default, new reports are added to the Electric Cloud project. You can enter a report description into the comments field if needed.

5. Click **OK** to save your changes.

   The new report appears in the **Report Editor**.

6. Customize the new report's query as described in Authoring DevOps Insight Reports on page 764.

Editing a Custom Report

Any custom report is editable. To edit a report:

1. Open the Reports List as described in Authoring DevOps Insight Reports on page 764.
2. Do one of the following to open the Report Editor:
   - Click the report.
   - Click the corresponding  (Views) buttons for the report, and then click **Report Editor**.
3. Edit the report as described in Authoring DevOps Insight Reports on page 764 and Authoring DevOps Insight Reports on page 764.

Adding Input Parameters to a Report

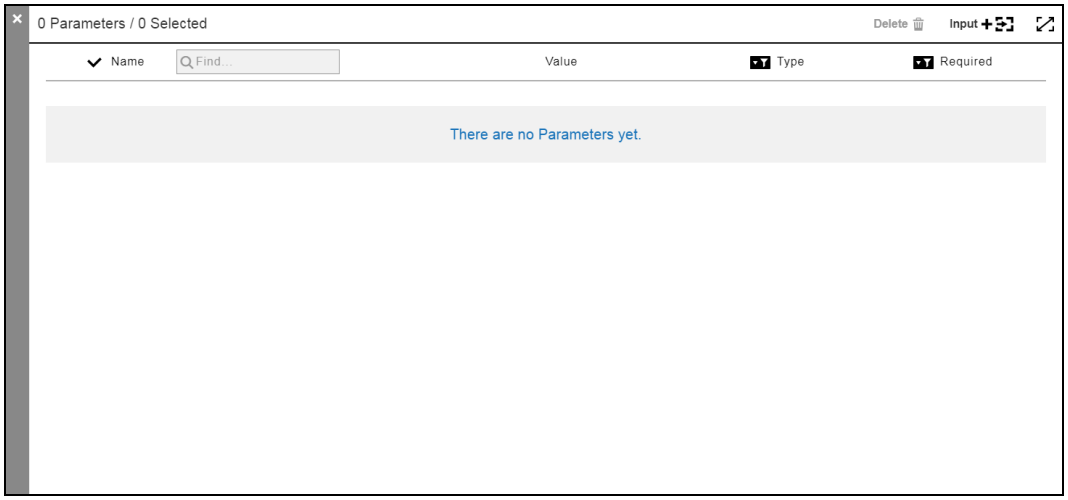
Parameters are free-form input values that can be fed to a dashboard or report to "slice and dice" the data based on the parameter value. For example, the Release Command Center dashboard shows metrics and trends over the last 10 days.

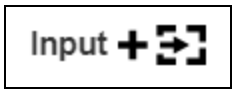
You might want to view the trend over a different time interval such as the last month or last two weeks based on the duration of your releases. Parameters let you define the time interval as a parameter that can be used by the dashboard or report viewer to change the time interval for viewing the Release Command Center metrics.

To add a parameter:

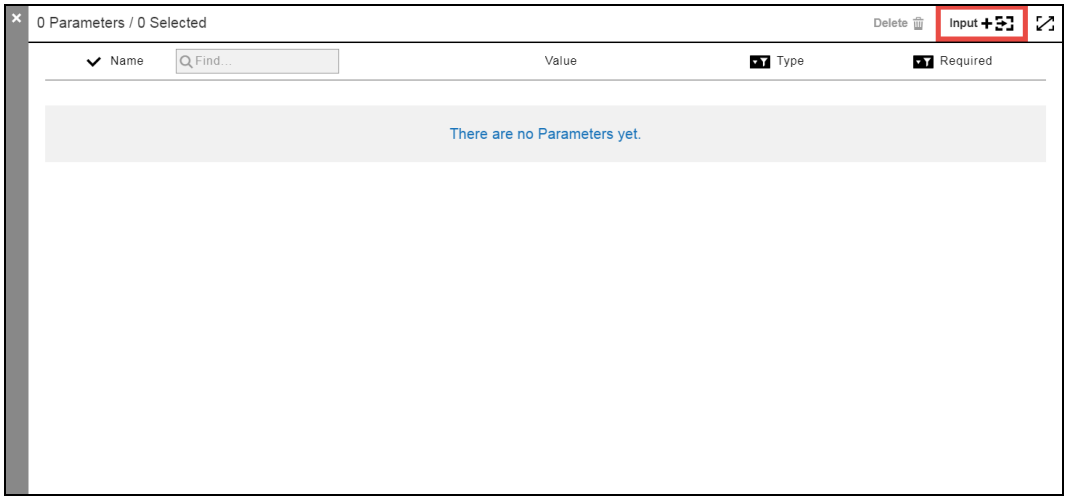
1. Click the  ("hamburger") menu button (at the top right of the page), and then click the  (Parameters) button.

   The parameters editor dialog box appears:

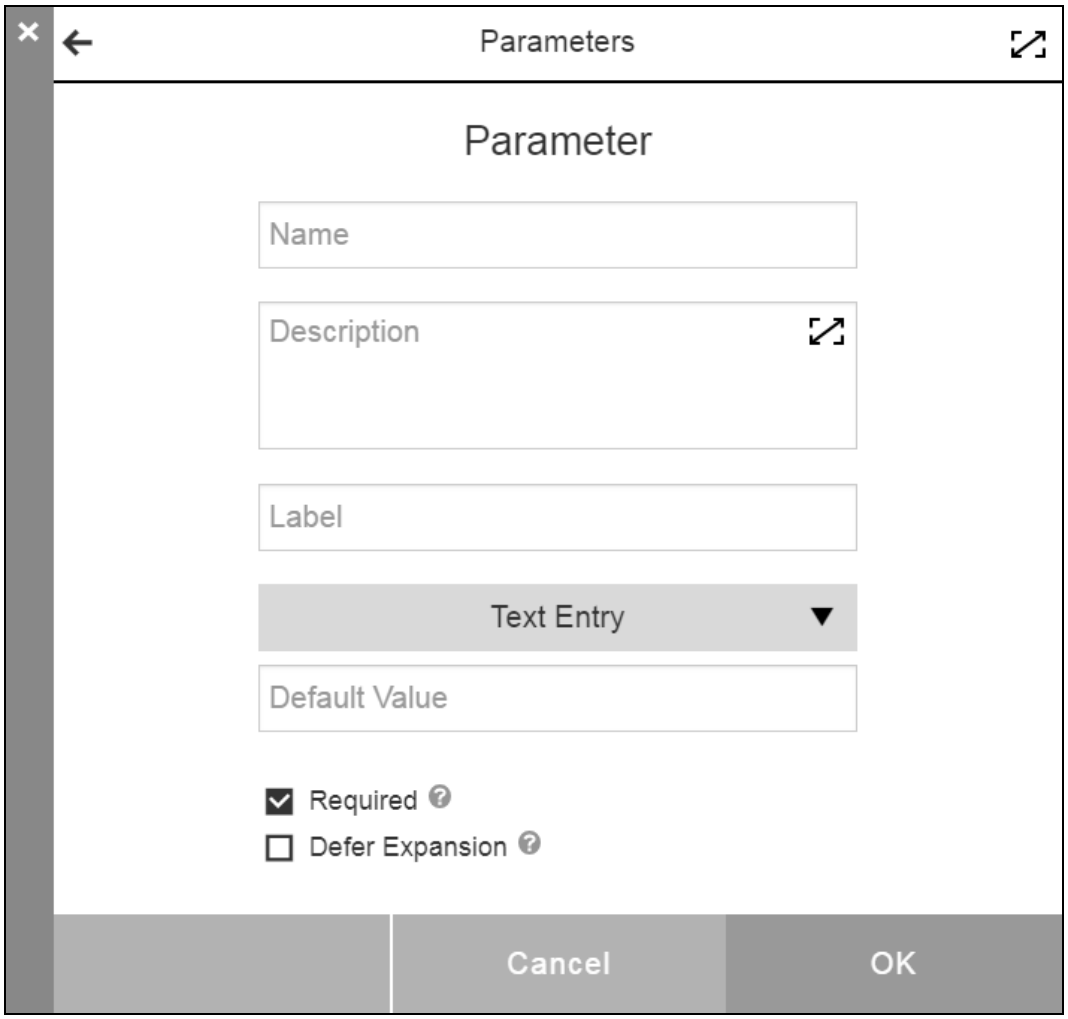




2. Click the (add input parameter) button:



The **Parameter** dialog box appears:


Parameters
Parameter
Name
Description
Label
Text Entry
Default Value
Required
Defer Expansion
Cancel
OK

3. Fill out the dialog box as follows:

Field or Option	Description
Name	Enter a unique name to specify the parameter when a dashboard or report is rendered.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Label	(Optional) Label for the parameter. If you enter a label, it appears in the UI form. If you do not enter a label, the parameter name appears in the UI form.
Parameter type	Select the parameter type from the drop-down menu. The "types" described below are available.
Text entry	Allows a short text entry in the Default Value field.
Text area	Expands the Default Value field to allow adding a longer entry.

Field or Option	Description

Dropdown menu	Creates a drop-down menu from which to select a value when the parameter is presented.
	• Enter options—Click **+ Add Option** to add a new row. Type-in the text and value for each option. The text is what will be displayed in the menu, and the value is the parameter value if that option is selected.
	• Load options from list—Enter a pipe-separated list of options (for example, `foo|bar|baz`). The text and value for the options will be the same.
	• Load options from property sheet—Enter the path to property sheet that contains options for the parameter. The property sheet must be created in a specific format:
	  • An `optionCount` property must exist whose value is the number of options.
	  • For each option, create a nested property sheet called `option`N, where N is the option number, starting with `1`.
	  • In each nested sheet, create two properties—`text` and `value`. The value of the property `text` will be displayed in the menu. The value of the property `value` is the parameter value if that option is selected.
	  • If `optionCount` is set to `3`, you must create three nested sheets—`option1`, `option2`, and `option3`.

Field or Option	Description

Radio selector	Creates "radio" buttons to select an entry when the parameter is presented.
	• Enter options—Click **+ Add Option** to add a new row. Type-in the text and value for each option. The text is what will be displayed in the menu, and the value is the parameter value if that option is selected.
	• Load options from list—Enter a pipe-separated list of options (for example, `foo|bar|baz`). The text and value for the options will be the same.
	• Load options from property sheet—Enter the path to property sheet that contains options for the parameter. The property sheet must be created in a specific format:
	  • An `optionCount` property must exist whose value is the number of options.
	  • For each option, create a nested property sheet called `option`N, where N is the option number, starting with `1`.
	  • In each nested sheet, create two properties—`text` and `value`. The value of the property `text` will be displayed in the menu. The value of the property `value` is the parameter value if that option is selected.
	  • If `optionCount` is set to `3`, you must create three nested sheets—`option1`, `option2`, and `option3`.

Checkbox	Creates a checkbox for a value to select (or not) when the parameter is presented. • Value when unchecked—The value of the parameter when the checkbox is unchecked. The default is "true." • Value when checked—The value of the parameter when the checkbox is selected. The default is "false." • Initially checked—Whether or not the checkbox should be checked initially. If true, set the "default value" to match the "Value when checked." If false, set the "Default value" to match the "Value when unchecked."

Field or Option	Description
Project	Creates a project selector field to choose a different project name where the parameter needs to find additional information.
Default Value	Default value to assign to the parameter if no explicit value is provided. The default value will be used when the dashboard is rendered so that you are not forced to specify the parameter values before viewing the dashboard or report.
Required	Click the checkbox to select the parameter as required. The check box is checked by default (meaning that the parameter is required).
Defer Expansion	Determines when to expand the parameter value. The check box is unchecked by default (meaning that expansion is not deferred).

4. Click **OK**.

   The new parameter is added to the parameters editor dialog box. For example:

5. Click the **x** to close the dialog box.

Authoring DevOps Insight Dashboards

In addition to using the Releases, Application Deployments, Microservice Deployments, Release Command Center, Continuous Integration, and Code Commit Trends dashboards that are included in ElectricFlow, you can create your own dashboards and reports to visualize key metrics that are

important to your organization. You can author dashboards that contain cells that display metrics for releases, deployments, features, builds, and so on.

This section covers the following topics:

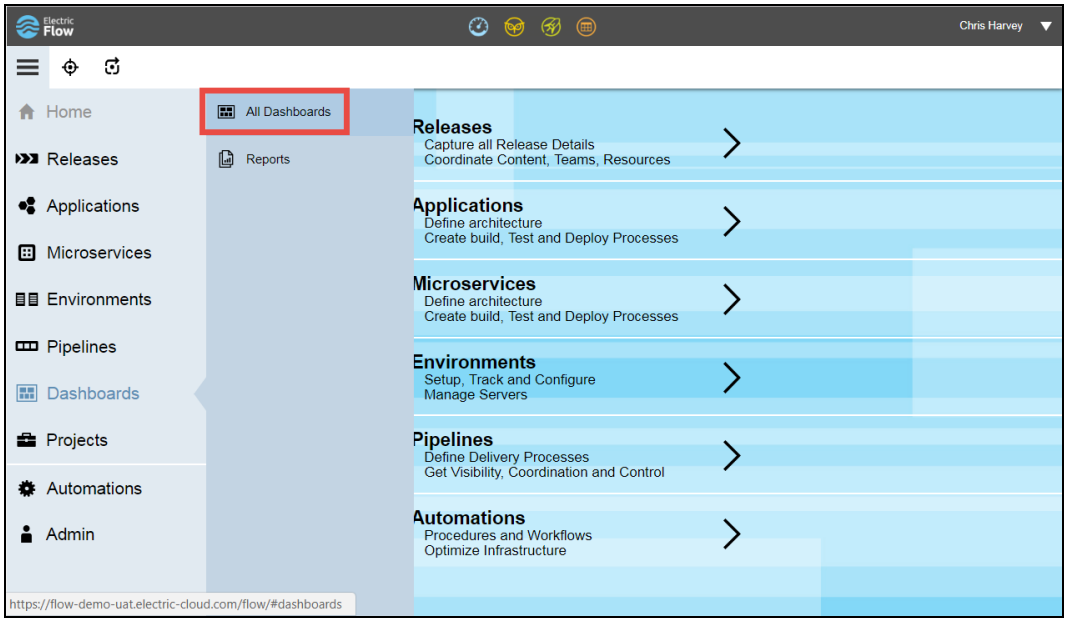
Creating a New DevOps Insight Dashboard Using the UI

You select a project to contain the dashboard, name the dashboard, and add one or more widgets, which will display the kinds of data that you select. You can also add input parameters to a dashboard.

You can create a widget or copy a widget from an existing widget for placement in a dashboard. For example, you might want to create a new dashboard consisting of two widgets from one dashboard and two widgets from a second dashboard.

Opening the Dashboards List

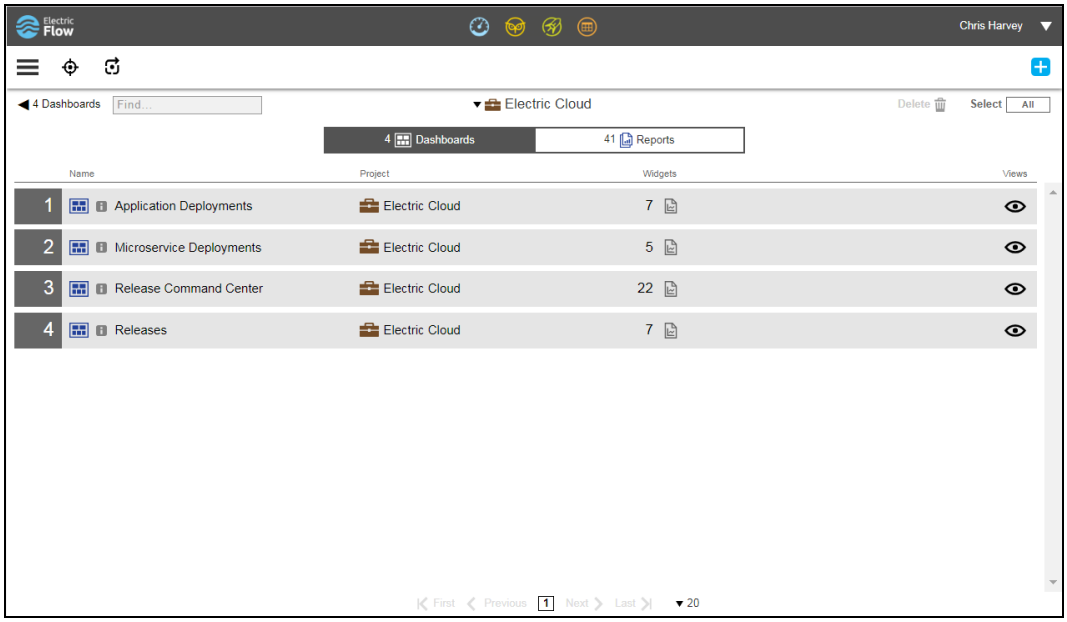
To open the Dashboards list, click **Dashboards** > **All Dashboards** from the (main) menu in the upper left corner of the Deploy UI:



The Dashboards list opens.

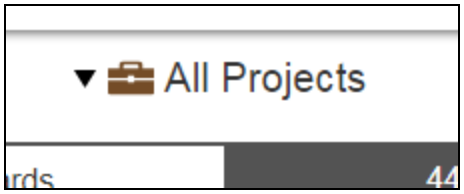
Viewing the List of Dashboards

The Dashboards list shows all of the DevOps Insight dashboards that are available in the currently-selected project or projects. For example, the following Dashboards list shows the reports in a project named Electric Cloud:



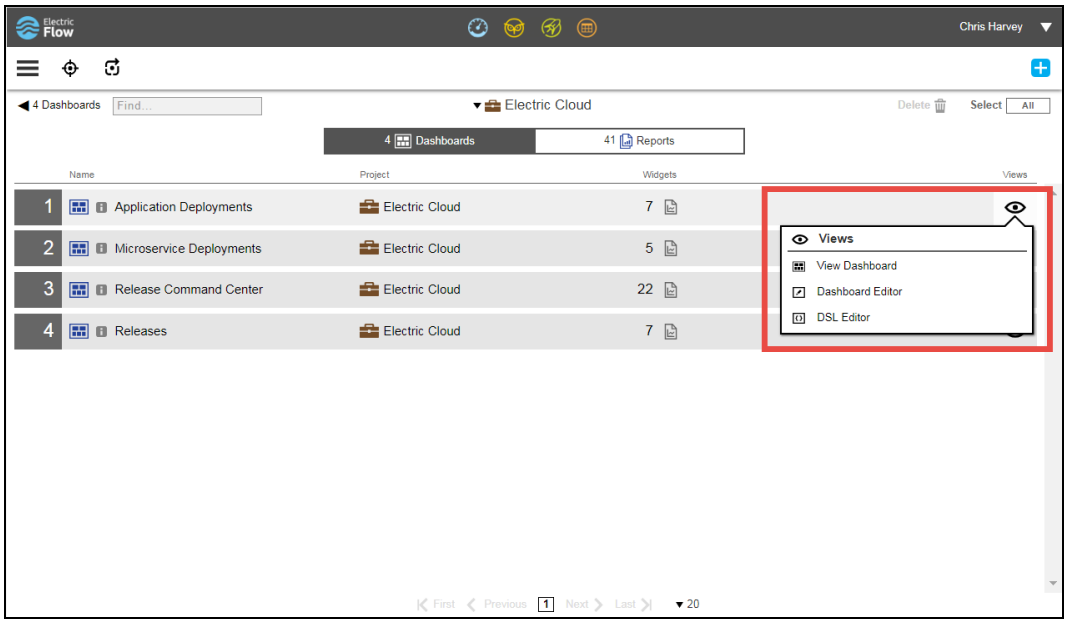
The Dashboards list comprises dashboards included with ElectricFlow and as well as custom dashboards that you have created.

By default, the dashboards are listed for all projects. You can filter this list by any combination of projects via the projects menu:



A variety of "stock" dashboards are included with ElectricFlow. These dashboards can help you learn about the dashboard functionality and how to create your own dashboards. You can also make copies of these dashboards to use as templates for creating dashboards. These dashboards are in the Electric Cloud project and are not editable.

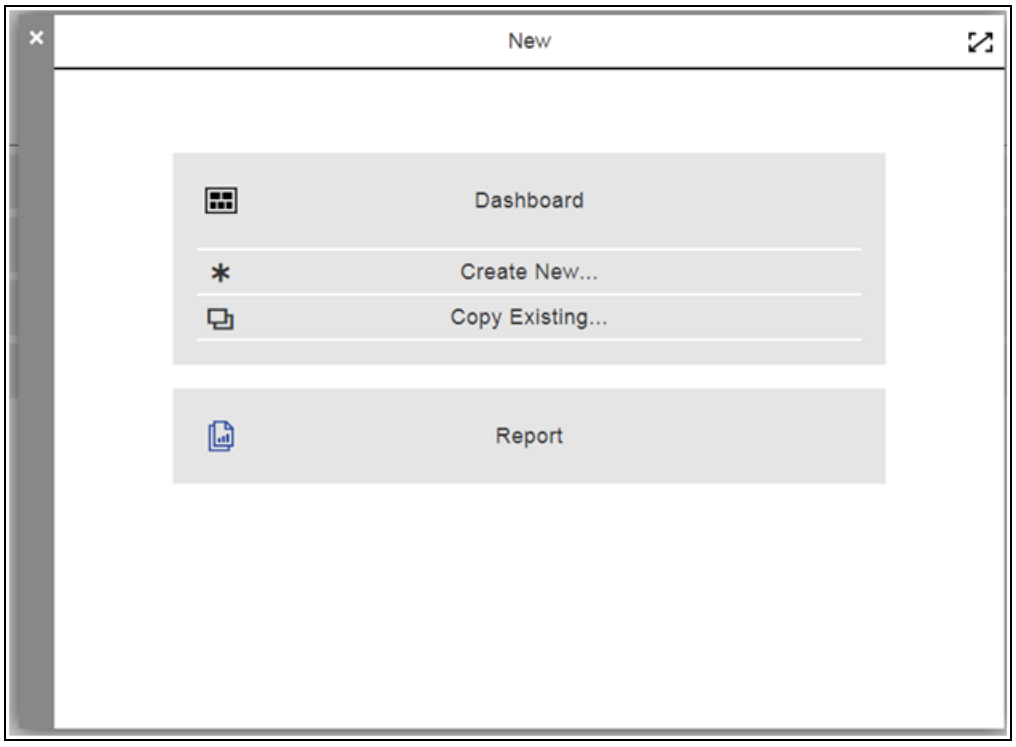
The (Views) button provides quick access to the Dashboard Editor for a specific dashboard:



Naming a Dashboard and Assigning it to a Project

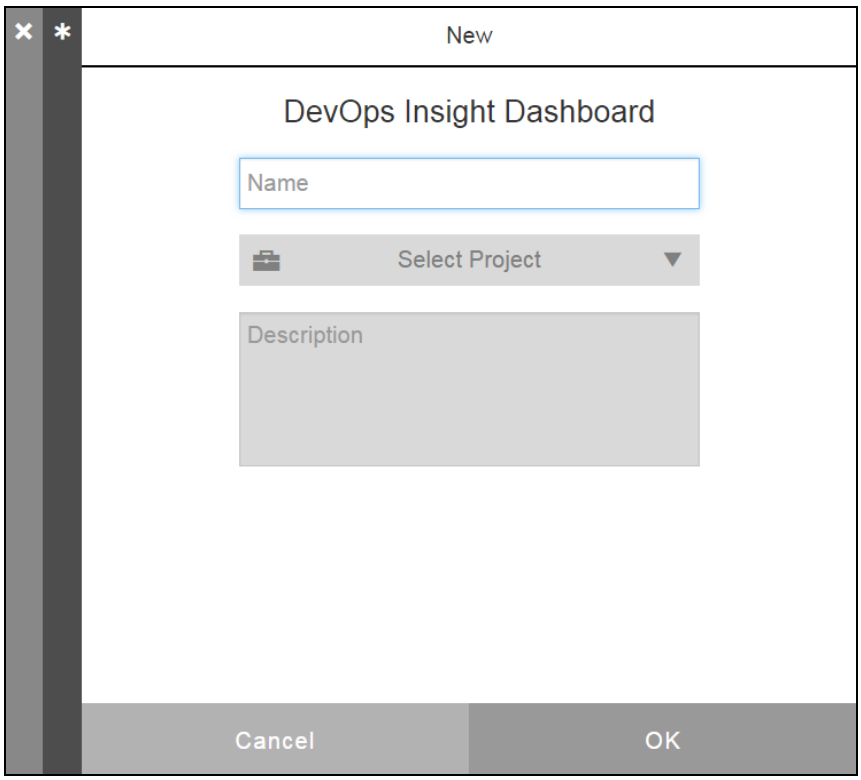
1. Click the (new DevOps Insight dashboard or report) button.

   The **New** dialog box appears:


New
Dashboard
Create New...
Copy Existing...
Report

2. Click **Create New...**.

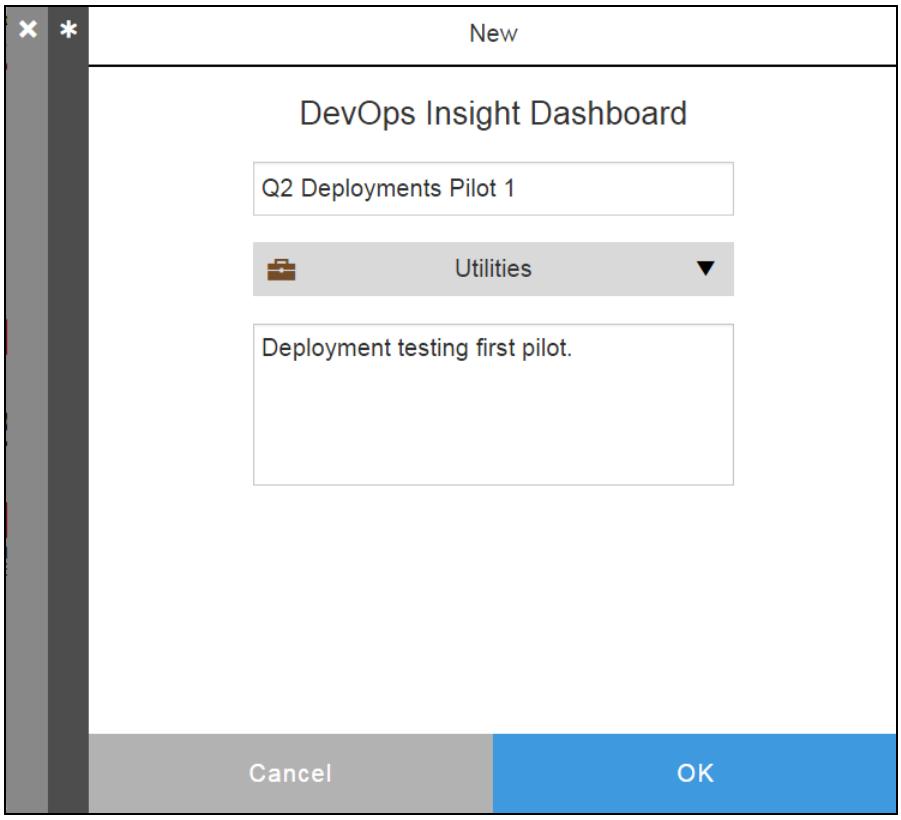
   The **New DevOps Insight Dashboard** dialog box appears:



3. Enter the following details:
   - **Name**—Name of the dashboard to create.

     You should avoid using special characters, which are described in Using Special Characters in ElectricFlow Object Names on page 1.
   - **Select Project**—The ElectricFlow project to which this dashboard will belong.
   - (Optional) **Description**—Description of the dashboard.

   For example:

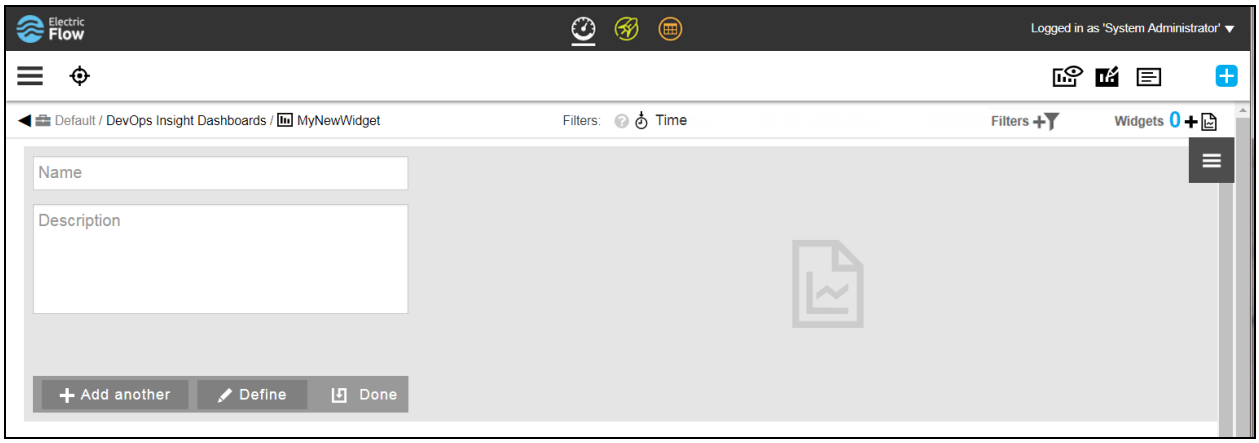


Defining the Dashboard Widgets

To create a cell, you define its underlying widget.

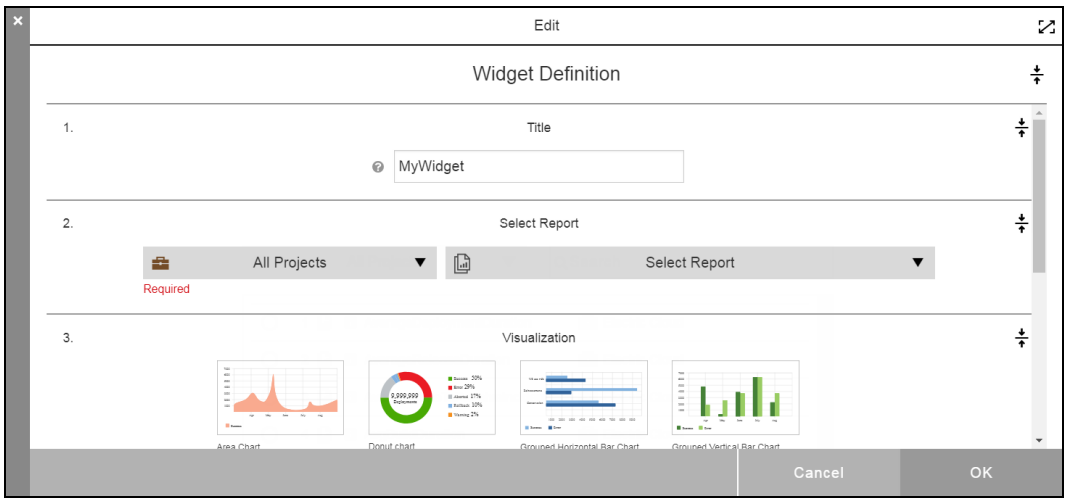
1. Click **OK**.

   A dialog box for creating the first widget for the new dashboard appears:



2. Click **Define**.

   The **Edit Widget Definition** dialog box for the new widget appears. For example:



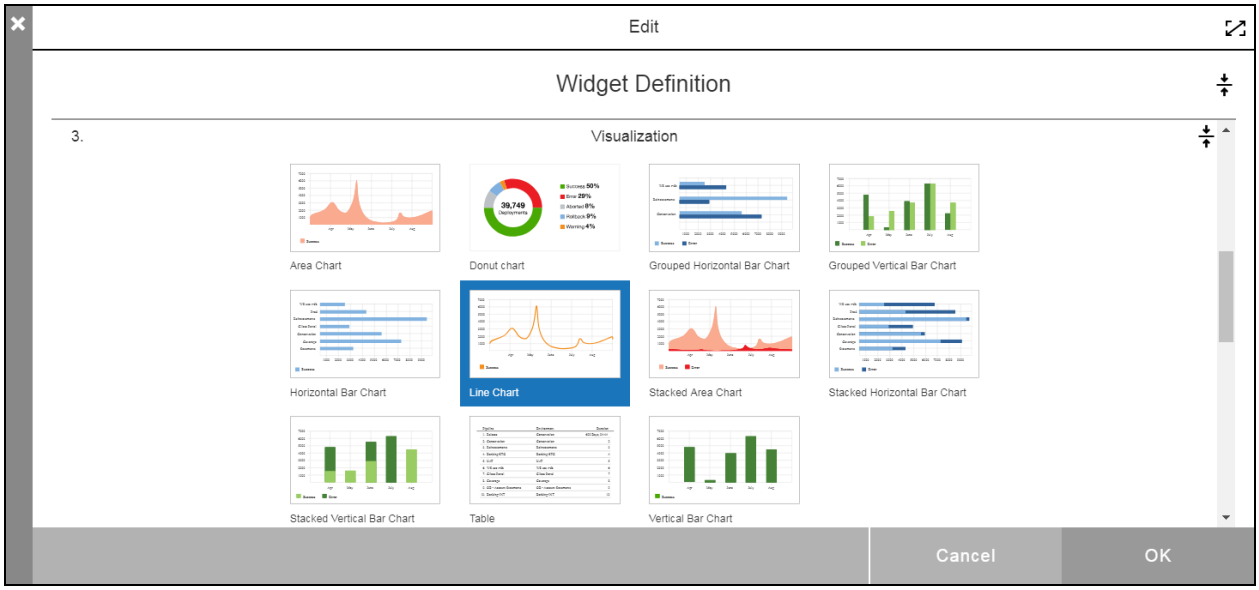
   The **Select Report** section lets you choose a DevOps Insight report upon which to base the widget.

3. Choose a report from the **Select Report** section.

   For details about creating reports, see Creating a Custom Report on page 767.

4. Scroll down and then click the (expand) button to expand the **Visualization** section.

   The **Visualization** section appears:



The **Visualization** section lets you select from a variety of widget visualizations such as donut charts, line charts, and tables.

The set of thumbnails that appears in the color map menus is determined by the report definition as well as the visualization type. For example, donut charts require that you map colors for the slices, and line charts require that you choose the color of the lines.

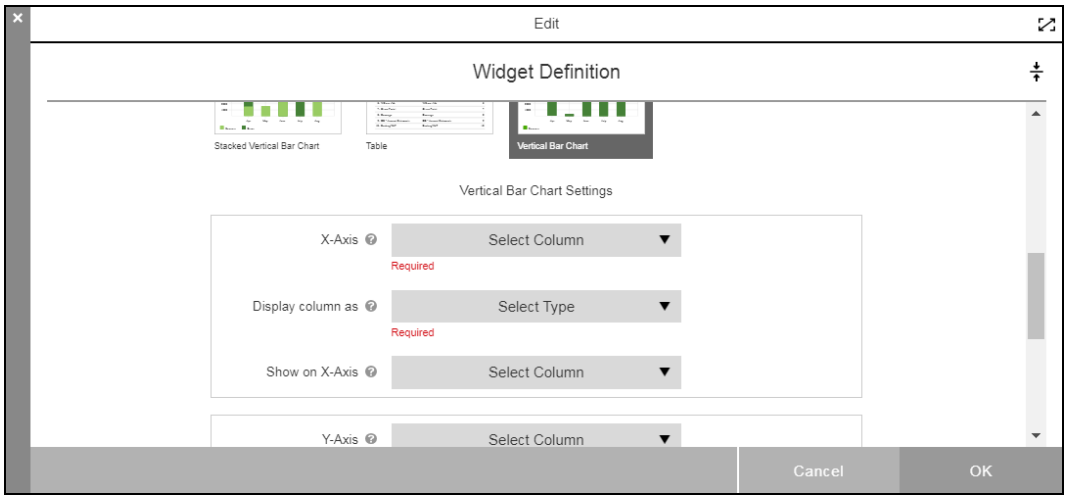
The widget fields and widget properties let you define details for the visualization. These details vary with the type of selected visualization and include columns to display in a table visualization and the color coding to use in a donut chart.

5. Complete the fields and menu options as follows.

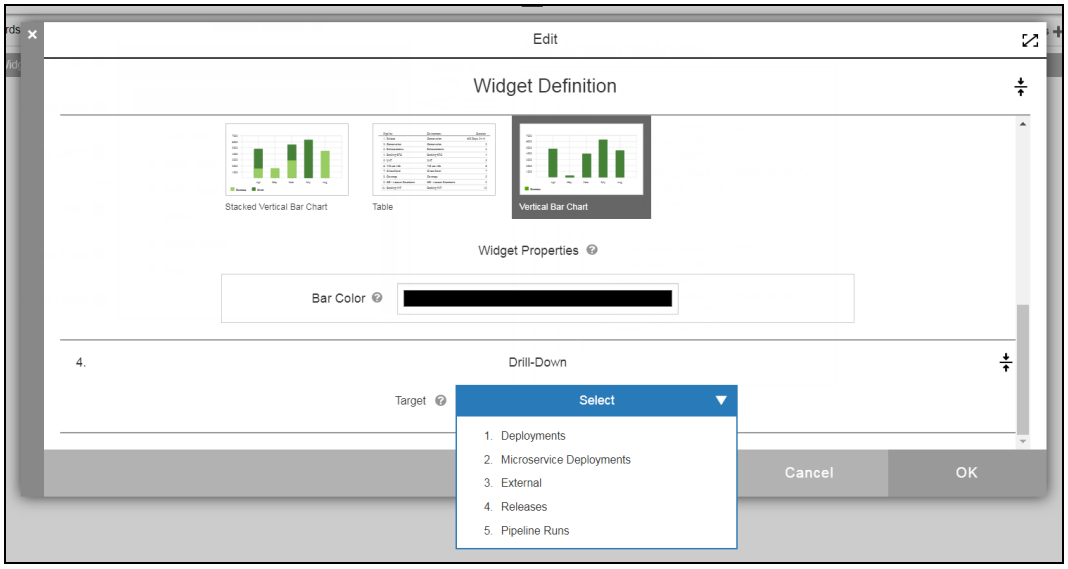
   The exact set of fields in this section varies with the type of visualization that you selected.

Menu or Field	Description
Visualization	The type of visualization to use for rendering the data returned by the widget's backing report
Widget Fields	The fields used for rendering the widget based on the visualization type. These fields should map to the fields returned by the report.
Widget Fields displayed for Donut Chart	
Slice	Slice or bucket
Slice Type	Expected data type for the slice
Slice Label	Field to use for display instead of the Slice value
Slice Size	Slice size
Slice Size Type	Expected data type for the slice size
Slice Size Label	Field to use for display instead of the slice size value
Total	Total value to display in the doughnut hole
Total Type	Expected data type for Total
Total Label	Field to use for display instead of the Total value
Colors Map	Colors used when rendering the widget based on values returned by the widget's backing report. The usage of the color map is determined by the visualization type

   The available options in the widget fields drop-down menus are based on the selected report. Following is an example of the widget fields displayed for a Vertical Bar Chart visualization:



The drop-down menus for the widget fields (**X-Axis** and **Y-Axis**) are populated based on the columns that are returned by the selected report:



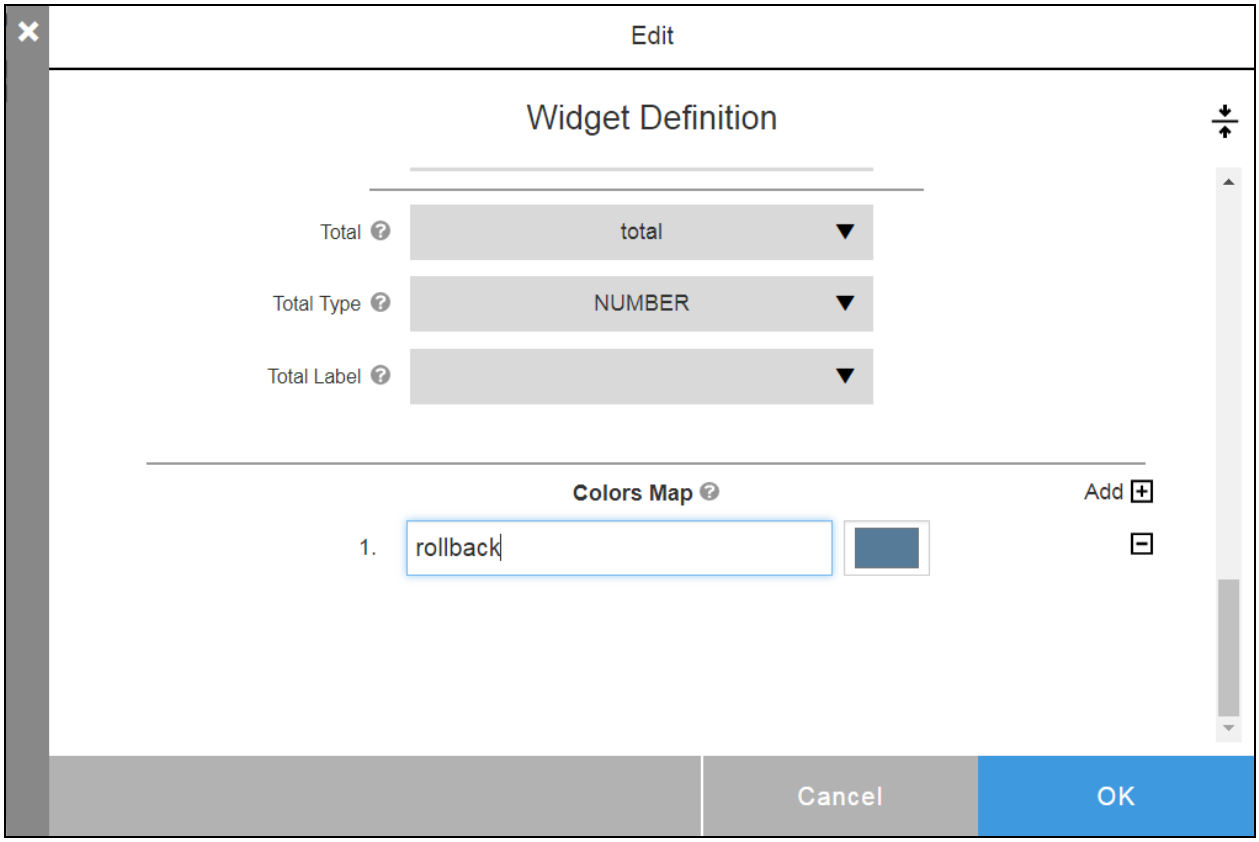
This means that an applicable report (either bundled with ElectricFlow or custom) must exist. For details about creating custom reports, see Authoring DevOps Insight Reports on page 764.

You can also type in custom field names.

The "Type" drop-down value is set automatically if the selected column could be determined based on the selected report.

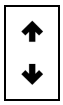
6. Scroll down and then choose the color coding (displayed if the visualization type supports it) used for the first value from the widget's backing report to be used when rendering the widget.

   For example:



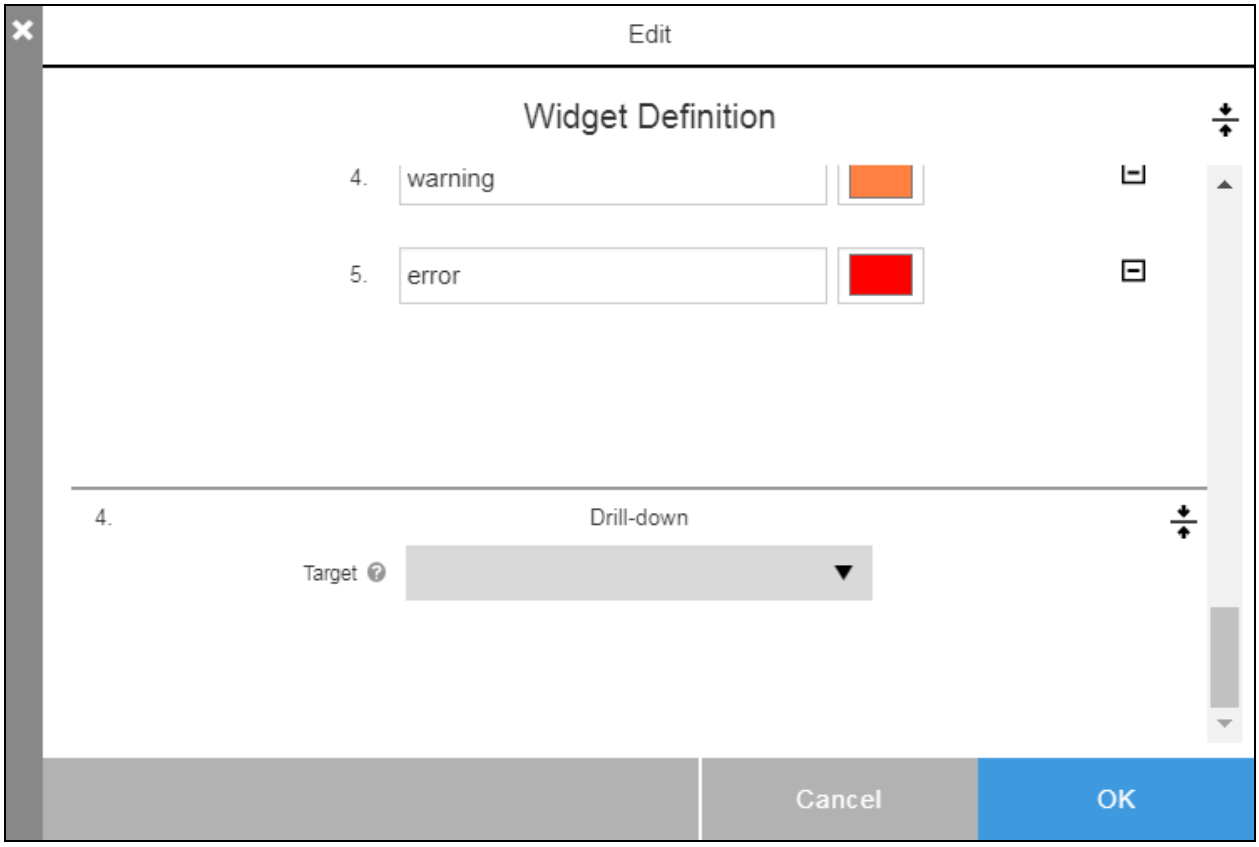
7. Repeat the prior step for each of the remaining values from the widget's backing report.

   For example:



8. Scroll down and then click the (expand) button to expand the **Drill-down** section.

   The **Drill-down** section appears. For example:



9. Complete the fields and menu options in the **Drill-down** section.

   The exact set of menu options or fields in this section varies with the type of target that you select from the **Target** pull-down.

Menu or Field	Description
Target	Page to navigate to when you drill down through the dashboard widget
Deployment Outcome	Deployment outcome
Application Project Name	Application project name
Application Name	Application name
Environment Project Name	Environment project name
Environment Name	Environment name
Deployment Date Range - Minimum	Minimum deployment date
Deployment Date Range - Maximum	Maximum deployment date
Microservice Project Name	Microservice project name
Microservice Name	Microservice name
Cluster Name	Cluster name
External URL	External URL to navigate to
Release Project Name	Release project name
Release Name	Release name

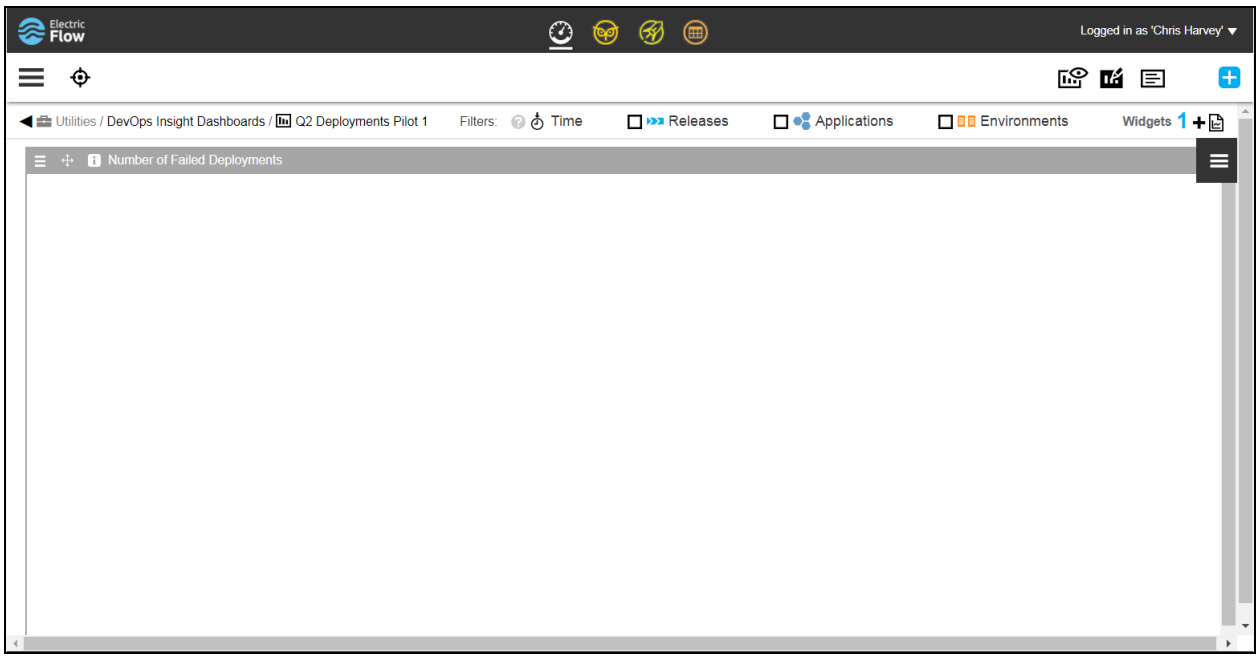
Menu or Field	Description
Release Actual End Date Range - Minimum	Minimum actual end date and time for the release
Release Actual End Date Range - Maximum	Maximum actual end date and time for the release
Release Planned End Date Range - Minimum	Minimum planned end date and time for the release
Release Planned End Date Range - Maximum	Maximum planned end date and time for the release
Pipeline Project Name	Pipeline project name
Pipeline Runtime Name	Pipeline runtime name
Pipeline Stage Name	Pipeline stage name
Path to the Pipeline Task	Field containing the '/' separated path to the pipeline task. For example, `group1/parentTask1/task1`

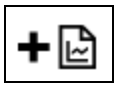
Each of these fields represents a field or column from the corresponding report definition that you created above. For example, in the **Application Name** field, you could enter `${applicationName}`, where `applicationName` is a column in the report definition that you created in the **Report** section above.

Also, for example, in the **External URL** field, you could enter `${drilldown_base_url}^incident_state=6^ORincident_state=7`, where `drilldown_base_url` is a column in the report definition that you created in the **Report** section above.

10. Click **OK**.

    The new widget appears in the new dashboard in "Editing Dashboard" mode. For example:



11. (Optional) Click the  button to add additional widgets to the dashboard, and then repeat the prior steps to create the widgets.

> **Tip:** You can also create a widget by copying a widget from another dashboard. For details, see Copying a Widget from Another Dashboard and Adding It to a Dashboard on page 822

Adding Input Parameters to a Dashboard

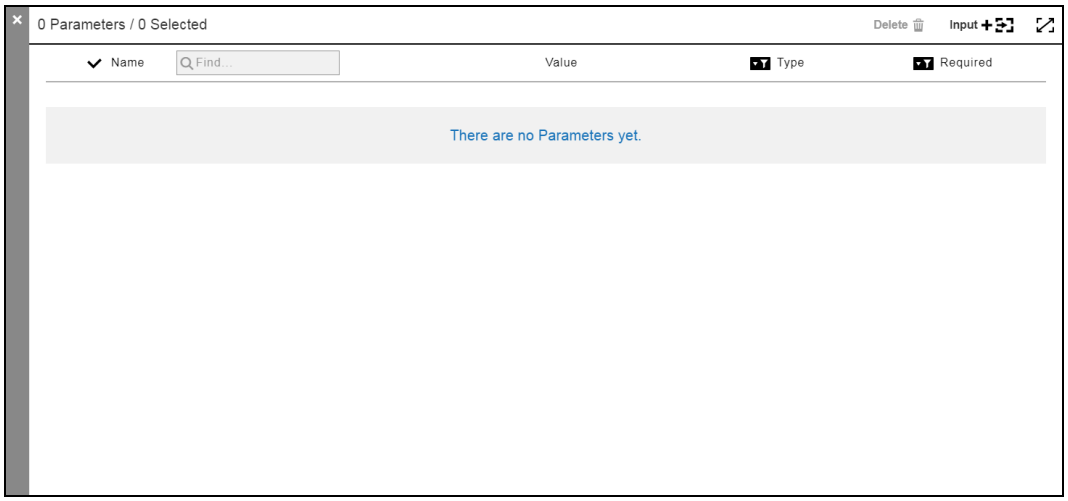
Parameters are free-form input values that can be fed to a dashboard or report to "slice and dice" the data based on the parameter value. For example, the Release Command Center dashboard shows metrics and trends over the last 10 days.

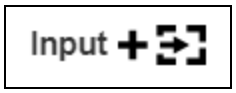
You might want to view the trend over a different time interval such as the last month or last two weeks based on the duration of your releases. Parameters let you define the time interval as a parameter that can be used by the dashboard or report viewer to change the time interval for viewing the Release Command Center metrics.

To add a parameter:

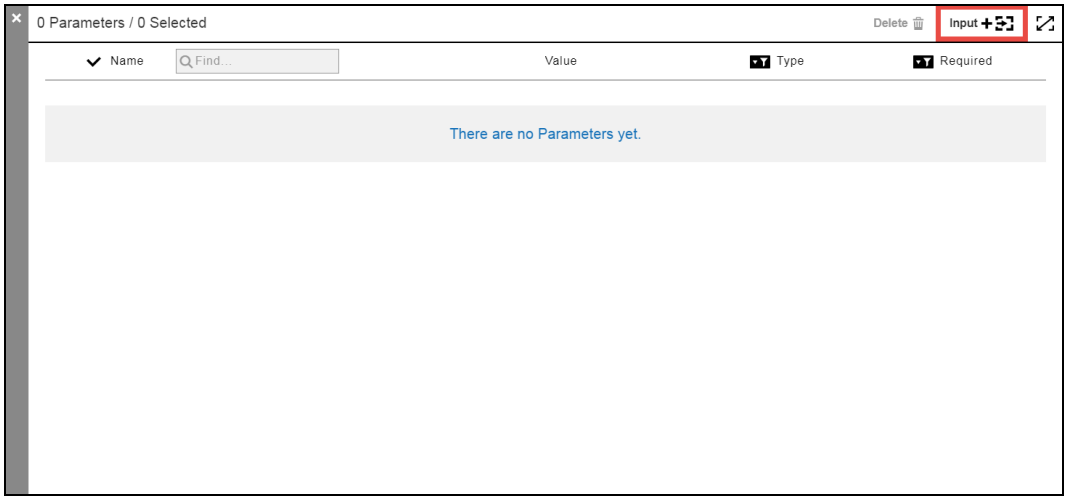
1. Click the  ("hamburger") menu button (at the top right of the page), and then click the  (Parameters) button.

   The parameters editor dialog box appears:

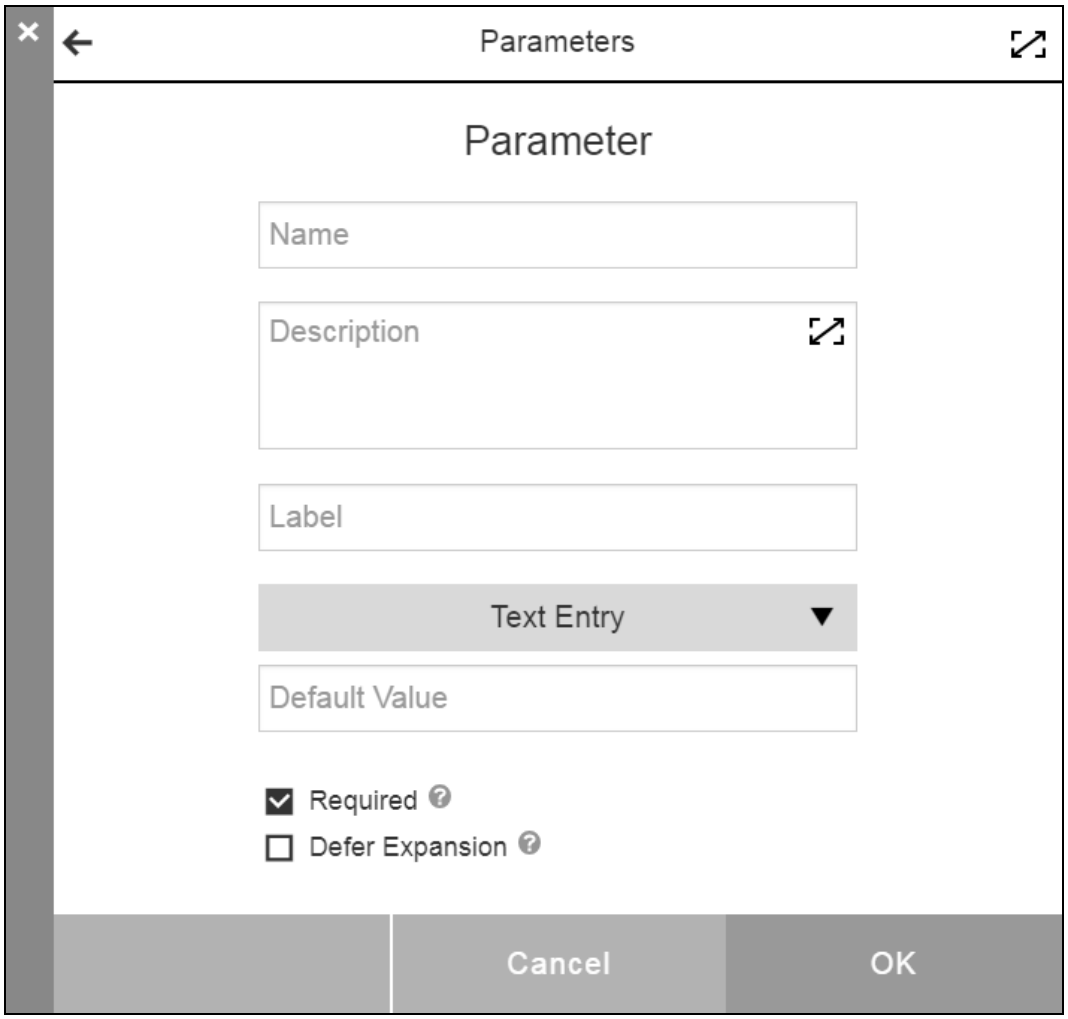




2. Click the (add input parameter) button:



The **Parameter** dialog box appears:


Parameters
Parameter
Name
Description
Label
Text Entry
Default Value
Required
Defer Expansion
Cancel
OK

3. Fill out the dialog box as follows:

Field or Option	Description
Name	Enter a unique name to specify the parameter when a dashboard or report is rendered.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Label	(Optional) Label for the parameter. If you enter a label, it appears in the UI form. If you do not enter a label, the parameter name appears in the UI form.
Parameter type	Select the parameter type from the drop-down menu. The "types" described below are available.
Text entry	Allows a short text entry in the Default Value field.
Text area	Expands the Default Value field to allow adding a longer entry.

Field or Option	Description

Dropdown menu	Creates a drop-down menu from which to select a value when the parameter is presented.
	• Enter options—Click **+ Add Option** to add a new row. Type-in the text and value for each option. The text is what will be displayed in the menu, and the value is the parameter value if that option is selected.
	• Load options from list—Enter a pipe-separated list of options (for example, `foo|bar|baz`). The text and value for the options will be the same.
	• Load options from property sheet—Enter the path to property sheet that contains options for the parameter. The property sheet must be created in a specific format:
	  • An `optionCount` property must exist whose value is the number of options.
	  • For each option, create a nested property sheet called `option`N, where N is the option number, starting with `1`.
	  • In each nested sheet, create two properties—`text` and `value`. The value of the property `text` will be displayed in the menu. The value of the property `value` is the parameter value if that option is selected.
	  • If `optionCount` is set to `3`, you must create three nested sheets—`option1`, `option2`, and `option3`.

Field or Option	Description

Radio selector	Creates "radio" buttons to select an entry when the parameter is presented.
	• Enter options—Click **+ Add Option** to add a new row. Type-in the text and value for each option. The text is what will be displayed in the menu, and the value is the parameter value if that option is selected.
	• Load options from list—Enter a pipe-separated list of options (for example, `foo|bar|baz`). The text and value for the options will be the same.
	• Load options from property sheet—Enter the path to property sheet that contains options for the parameter. The property sheet must be created in a specific format:
	  • An `optionCount` property must exist whose value is the number of options.
	  • For each option, create a nested property sheet called `option`N, where N is the option number, starting with `1`.
	  • In each nested sheet, create two properties—`text` and `value`. The value of the property `text` will be displayed in the menu. The value of the property `value` is the parameter value if that option is selected.
	  • If `optionCount` is set to `3`, you must create three nested sheets—`option1`, `option2`, and `option3`.

Checkbox	Creates a checkbox for a value to select (or not) when the parameter is presented. • Value when unchecked—The value of the parameter when the checkbox is unchecked. The default is "true." • Value when checked—The value of the parameter when the checkbox is selected. The default is "false." • Initially checked—Whether or not the checkbox should be checked initially. If true, set the "default value" to match the "Value when checked." If false, set the "Default value" to match the "Value when unchecked."

Field or Option	Description
Project	Creates a project selector field to choose a different project name where the parameter needs to find additional information.
Default Value	Default value to assign to the parameter if no explicit value is provided. The default value will be used when the dashboard is rendered so that you are not forced to specify the parameter values before viewing the dashboard or report.
Required	Click the checkbox to select the parameter as required. The check box is checked by default (meaning that the parameter is required).
Defer Expansion	Determines when to expand the parameter value. The check box is unchecked by default (meaning that expansion is not deferred).

4. Click **OK**.

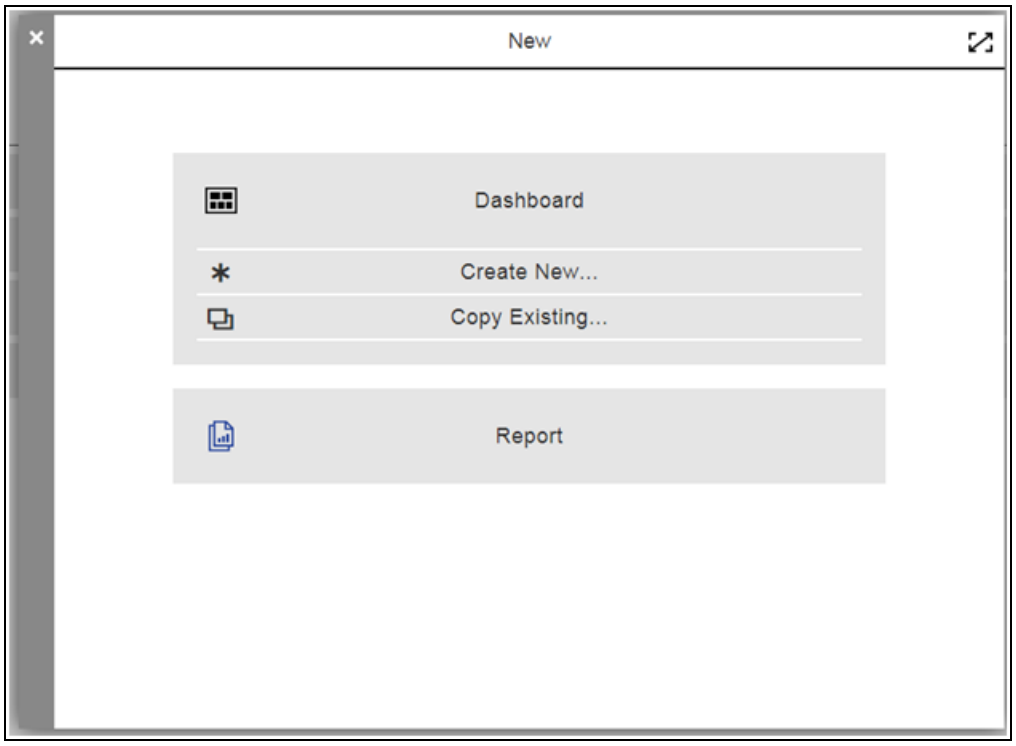
   The new parameter is added to the parameters editor dialog box. For example:

5. Click the **x** to close the dialog box.

Creating a DevOps Insight Dashboard by Copying an Existing Dashboard

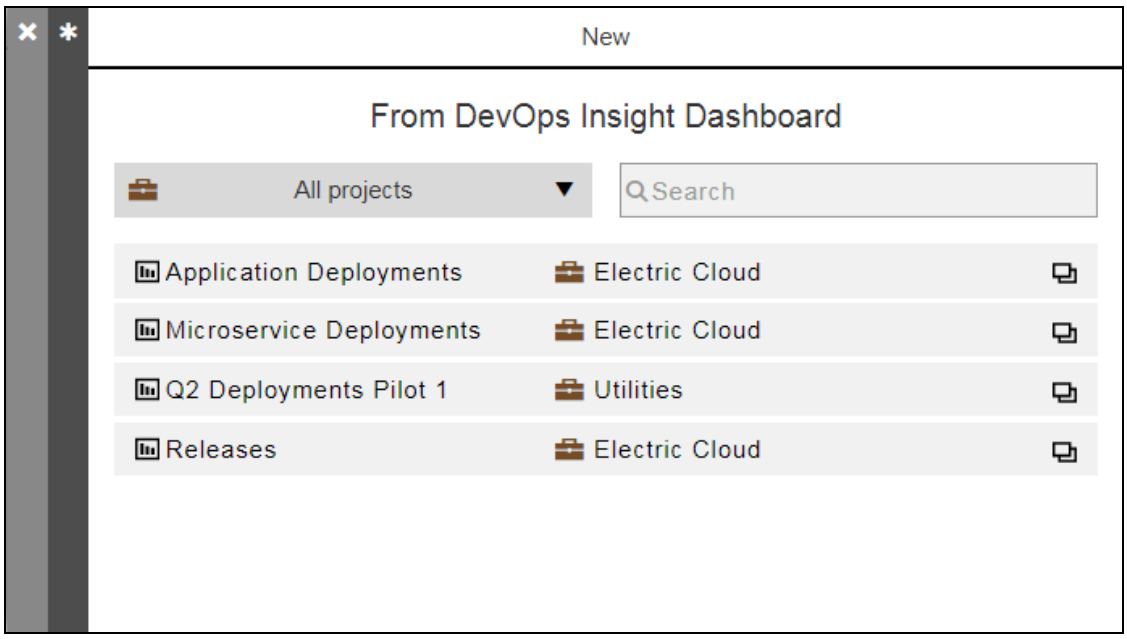
1. Click the  (new DevOps Insight dashboard) button.

   The **New** dialog box appears:



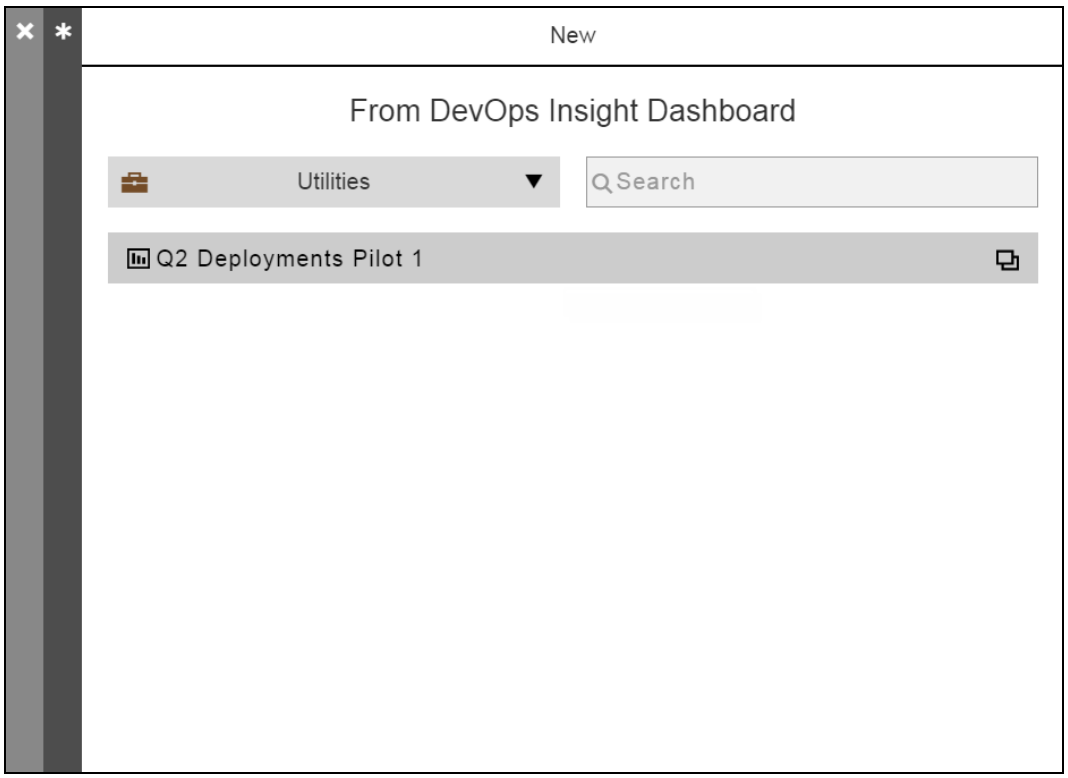
2. Click **Copy Existing...**.

   The **New From DevOps Insight Dashboard** dialog box appears:

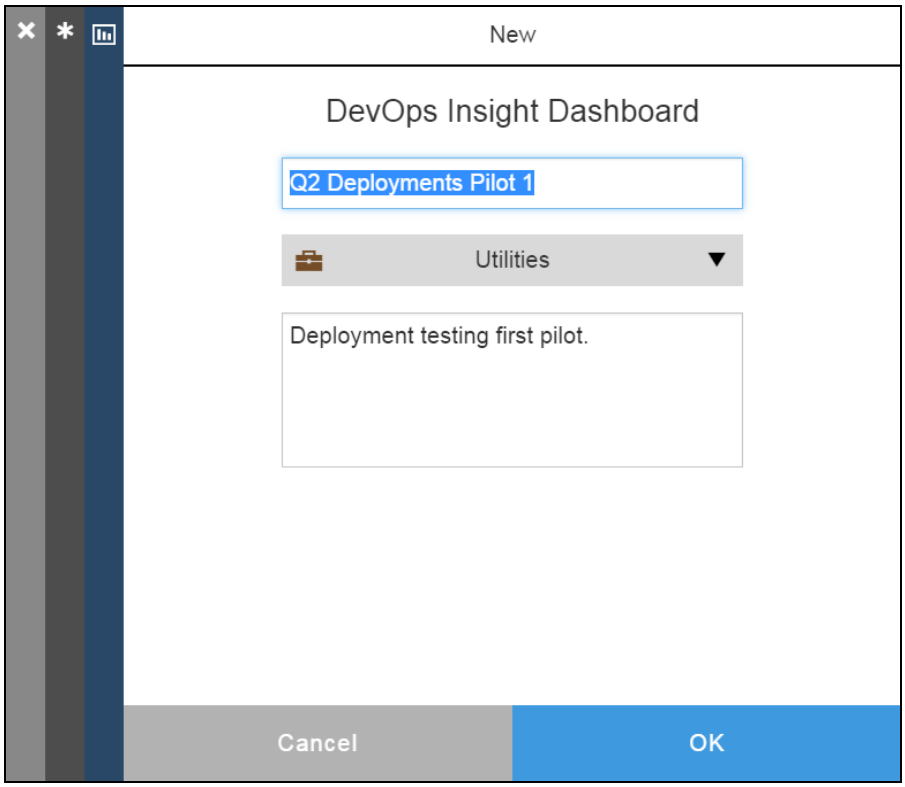


3. Browse to the dashboard that you want to copy and click to select it.

   For example:



   The **New DevOps Insight Dashboard** dialog box appears. The dashboard name and description fields are populated with the values from the dashboard that you copied. For example:


New
DevOps Insight Dashboard
Q2 Deployments Pilot 1
Utilities
Deployment testing first pilot.
Cancel
OK

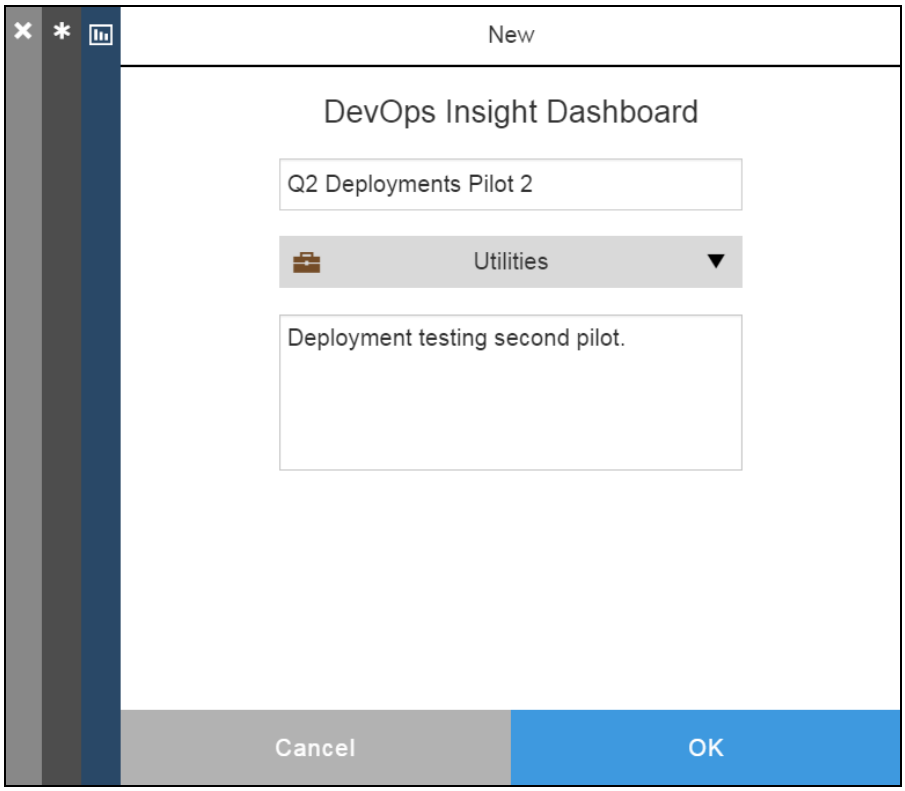
4. Modify the dashboard name and description fields as needed:

   - **Name**—Name of the dashboard to create.

     You should avoid using special characters, which are described in Using Special Characters in ElectricFlow Object Names on page 1.

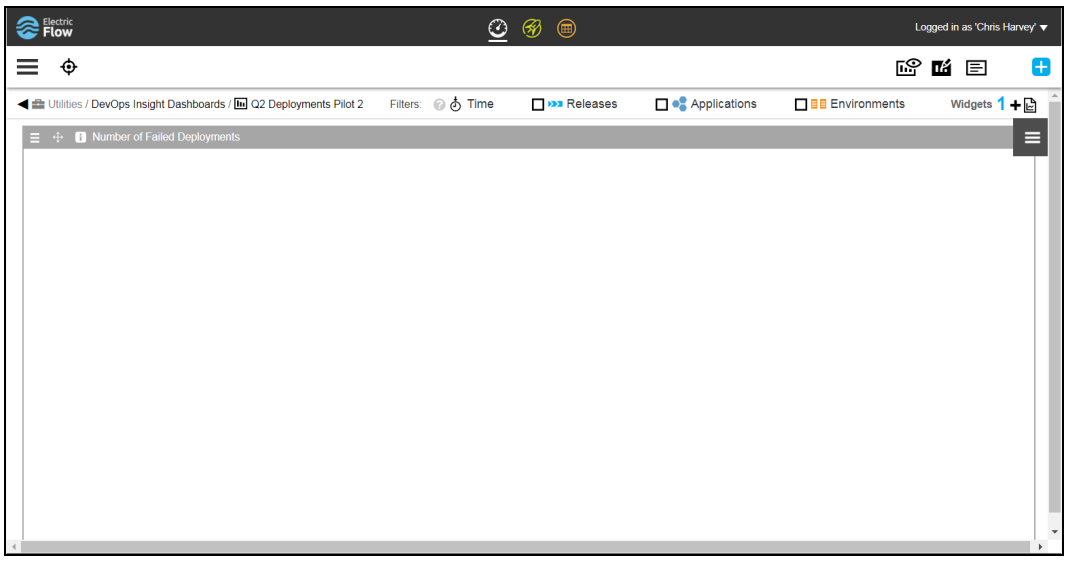
   - **Select Project**—The ElectricFlow project to which this dashboard will belong.
   - (Optional) **Description**—Description of the dashboard.

   For example:



5. Click **OK**.

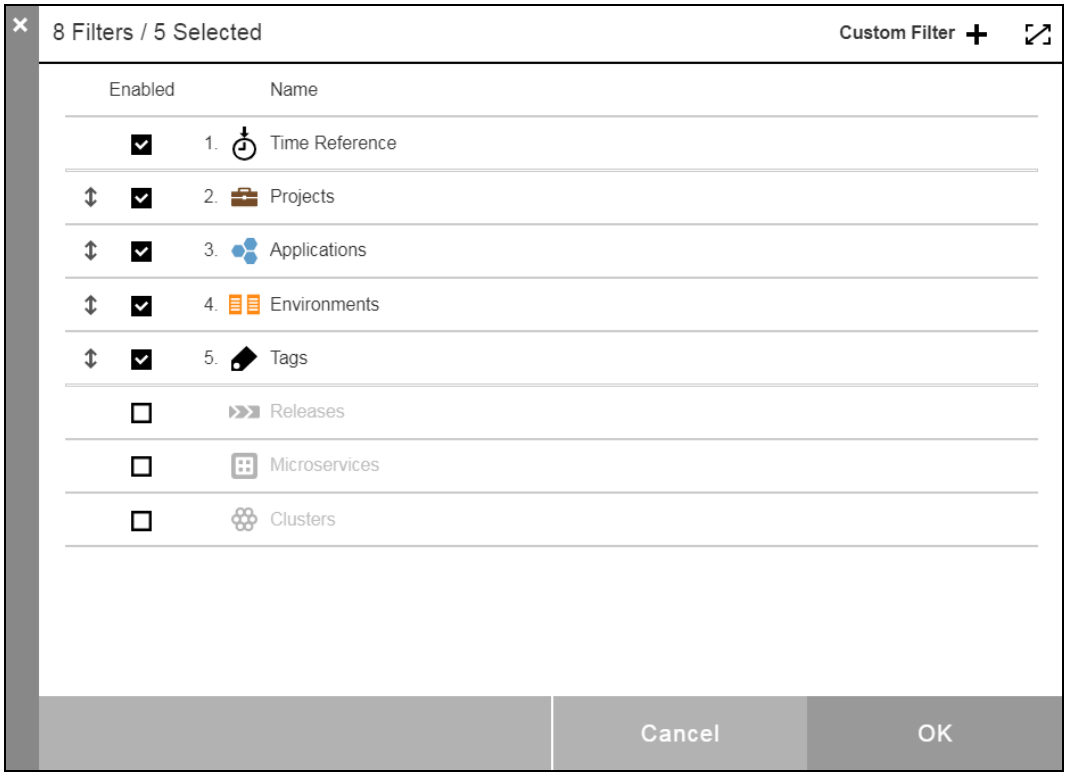
   The new dashboard appears in "Dashboard Editor" mode. For example:



6. (Optional) Click the 

(Add Filters) button to add a filter.

   The Time Reference filter is added by default when you create a new dashboard via the dashboard editor. A dialog box for selecting filters for the new dashboard appears:



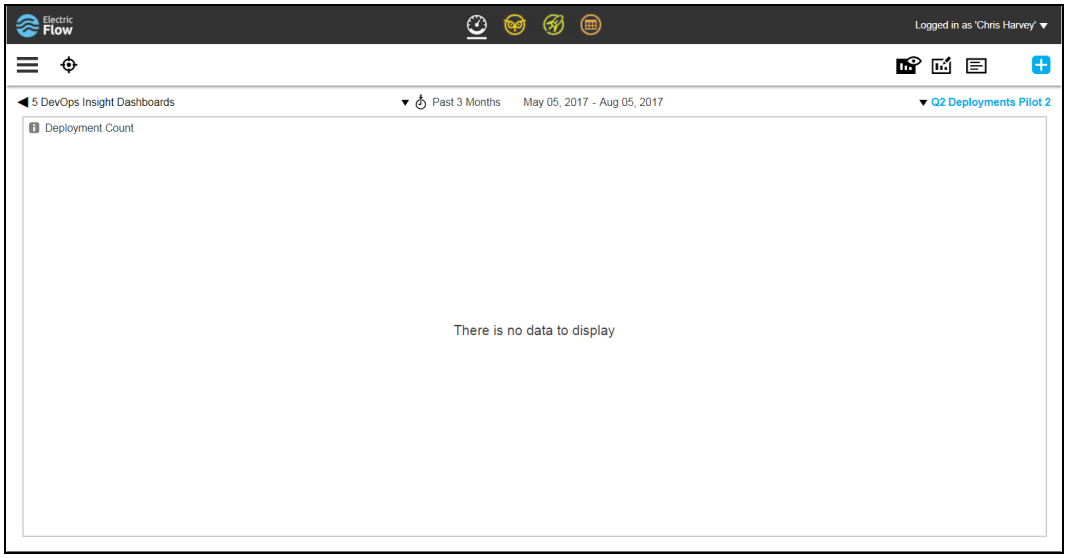
   You can add your own filters by choosing the 

button. For details, see

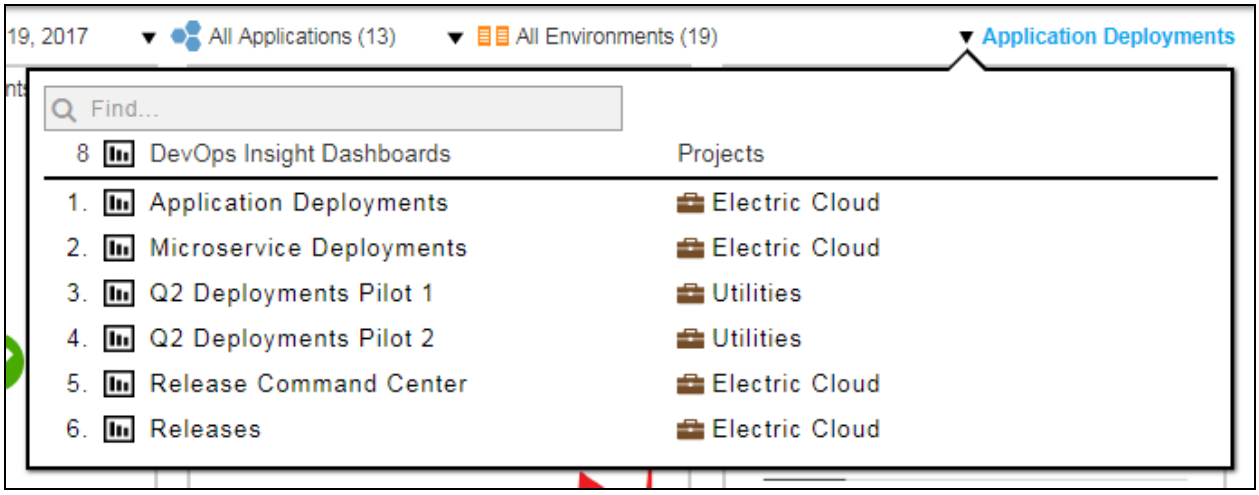
7. Check the check boxes for one or more filters and click **OK**.

8. (Optional) Click the  button to see the new dashboard in "View Dashboard" mode.

   For example:



   The new dashboard appears in the pulldown menu of available dashboards at the top right corner of the dashboards page. For example:



9. Add one or more widgets as needed.

   For details, see Defining the Dashboard Widgets on page 795 above.

Adding Input Parameters to a Dashboard

See Adding Input Parameters to a Dashboard on page 805

Adding Widgets to an Existing DevOps Insight Dashboard

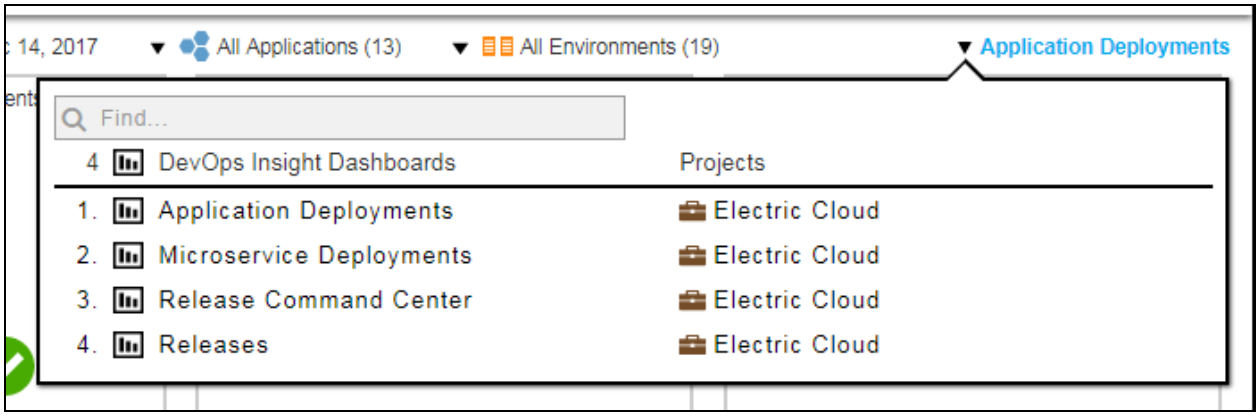
You can add widgets to an existing dashboard to extend its functionality. You can create a widget or copy a widget from an existing widget. For example, you might want to create two entirely new widgets and also add copies of two widgets that exist in a second dashboard.

Creating a Widget and Adding It to a Dashboard

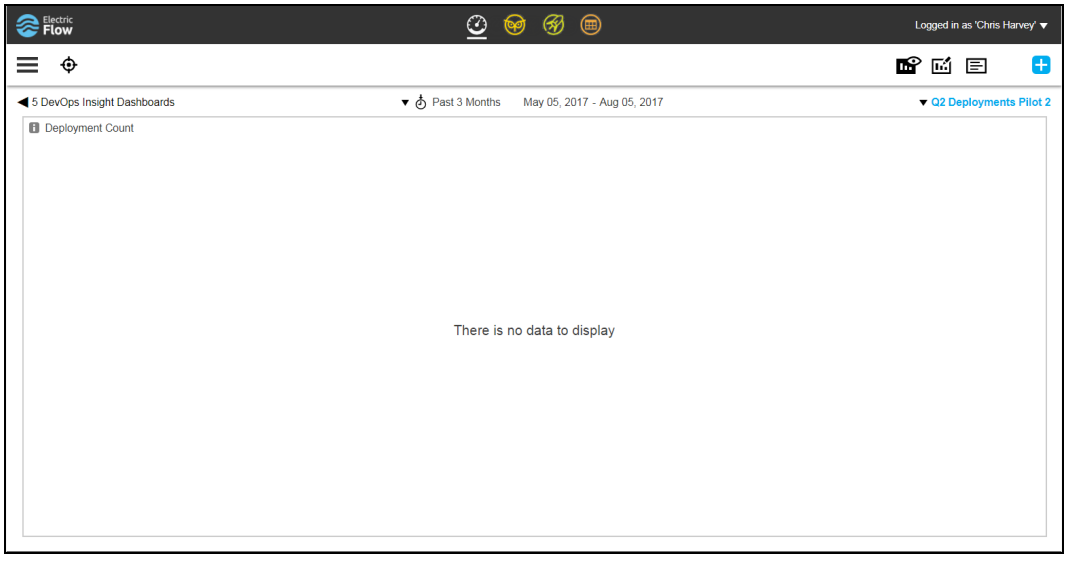
To create a widget and add it to a dashboard:

1. Select the dashboard from the dashboards pull-down menu.

   For example:



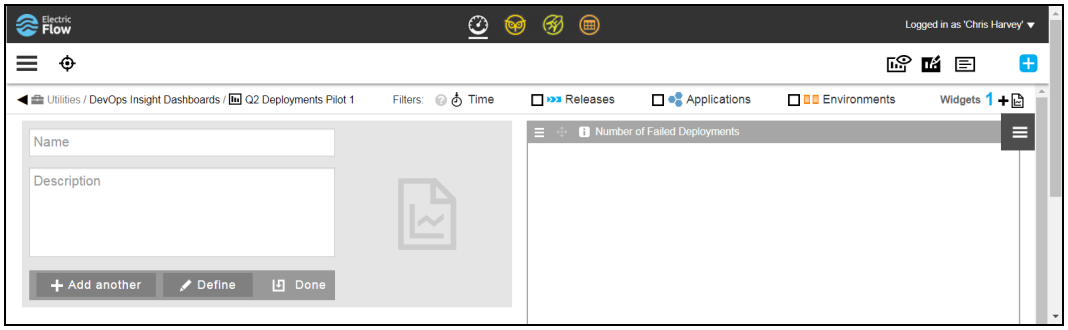
   The dashboard appears in Graphical viewing mode. For example:



2. Click the  button to switch to dashboard edit mode.

3. Click the  (add widget) button and then click **Create New**.

   A dialog box for creating the widget appears. For example:



4. Enter the following details into the dialog box:

   - **Name**—Name of the widget to create. This name appears as the title above the widget in the dashboard.

     You should avoid using special characters, which are described in Using Special Characters in ElectricFlow Object Names on page 1.

   - (Optional) **Description**—Description of the widget.

5. Click **Define** and then edit the definition of the widget.

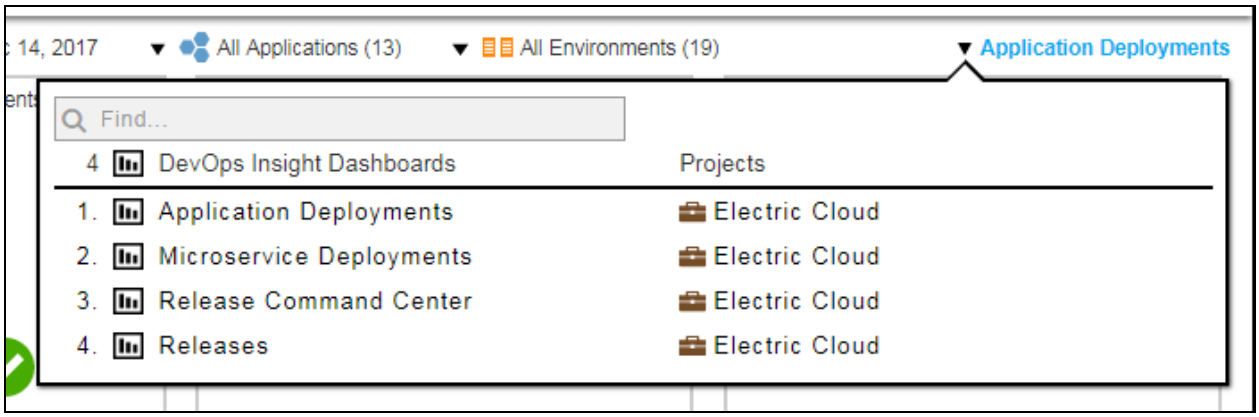
   For details, see Defining the Dashboard Widgets on page 795

Copying a Widget from Another Dashboard and Adding It to a Dashboard

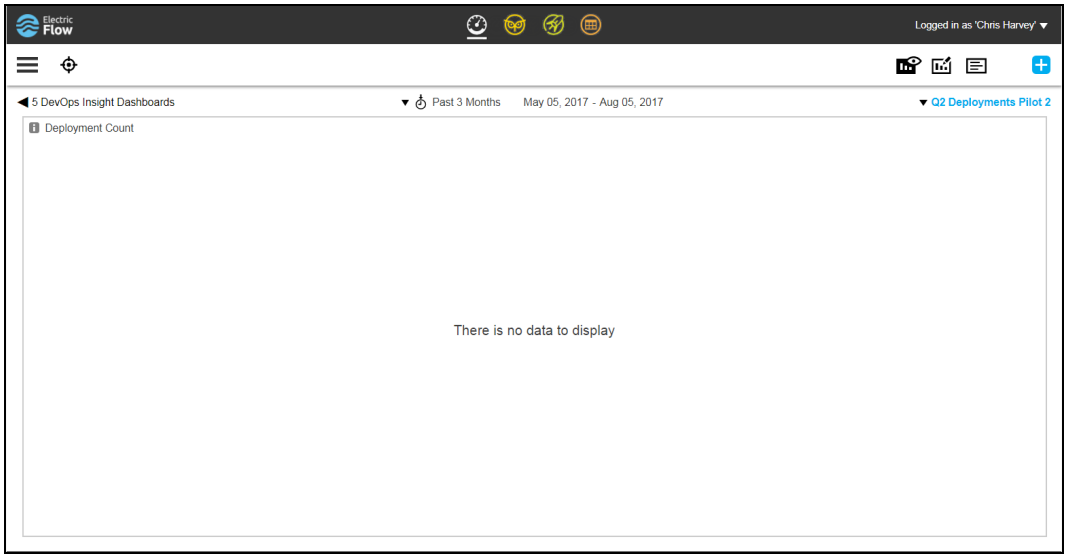
To copy a widget from another dashboard and add it to dashboard:

1. Select the dashboard to contain the new widget from the dashboards pull-down menu.

   For example:



The dashboard appears in Graphical viewing mode. For example:



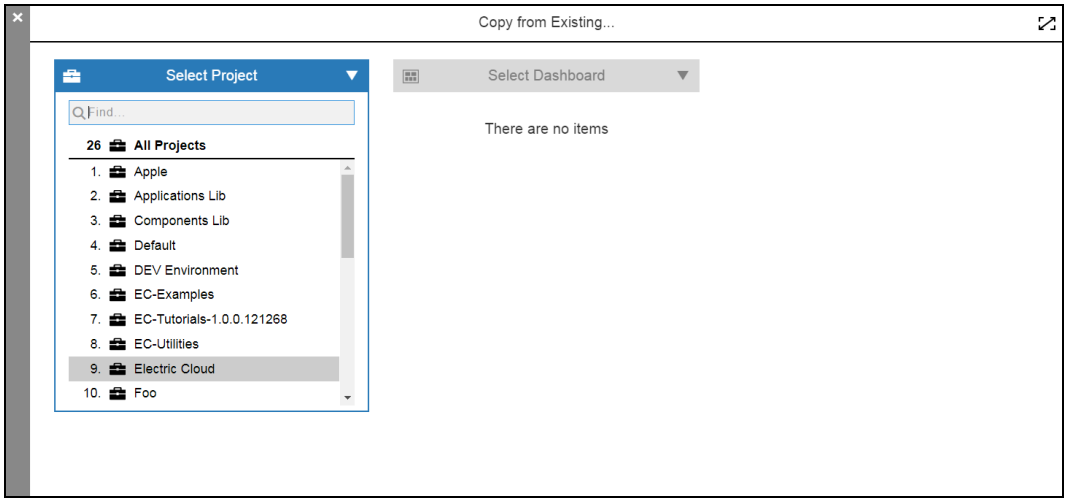
2. Click the  (edit dashboard) button to switch to edit mode.

3. Click the  (add widget) button, and then click **Copy Existing**.

   The **Copy from Existing...** dialog box appears:

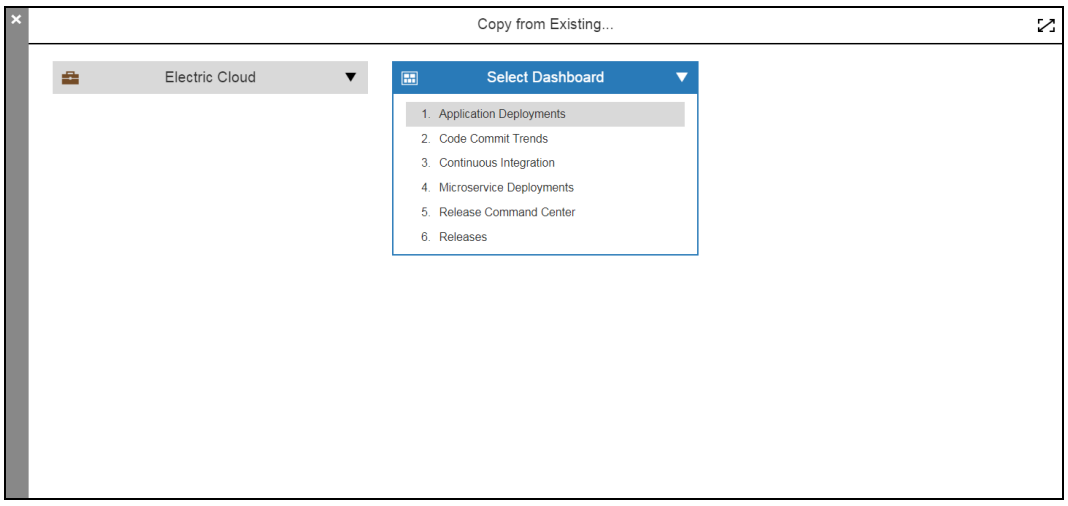
4. Choose a project from the **All Projects** menu.

   For example:

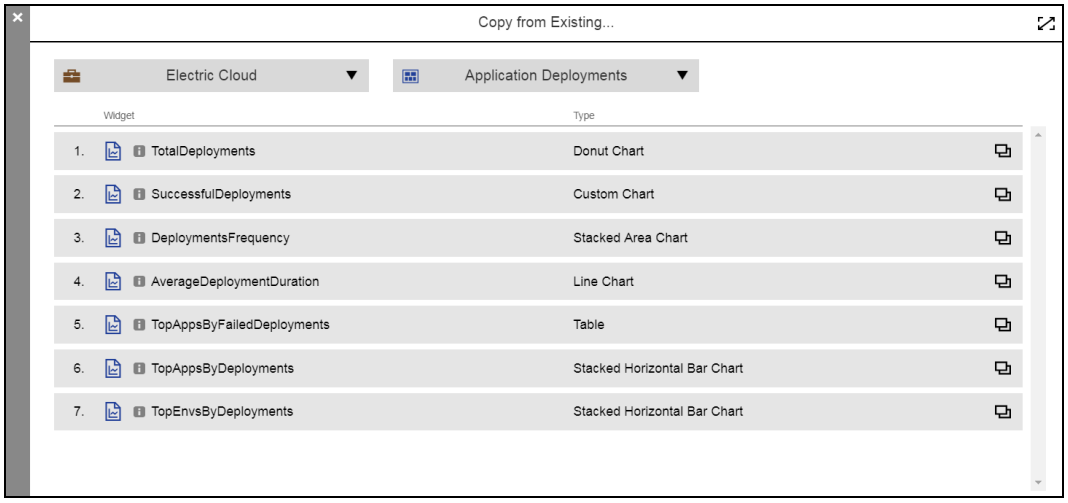


5. Choose a dashboard from the **Select Dashboard** pulldown menu.

   For example:

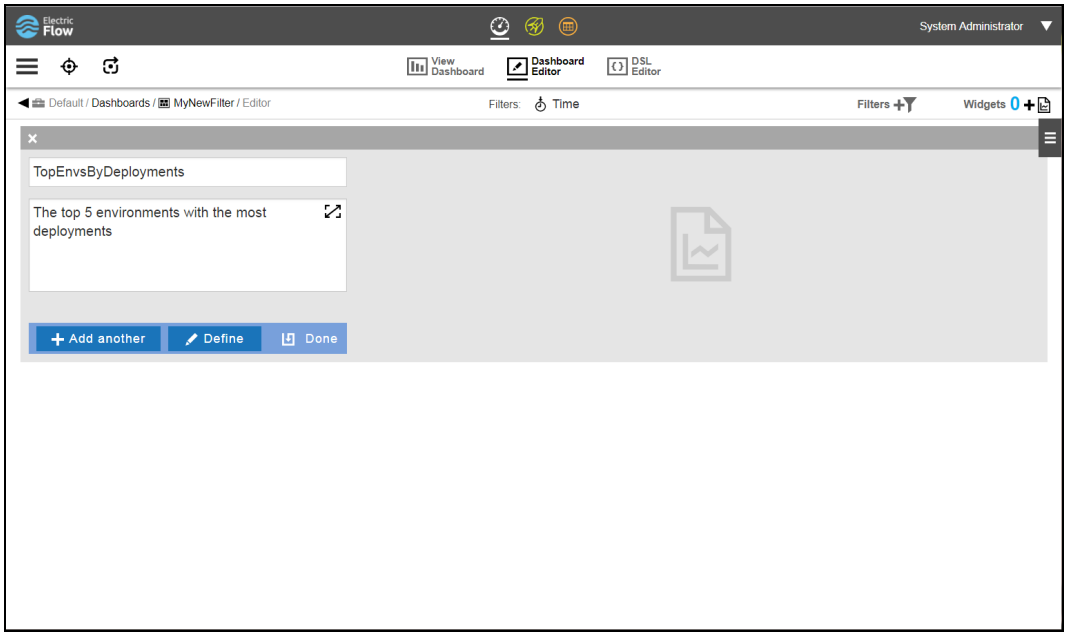


The list of available widgets to copy appears in the dialog box. For example:



6. Click the widget that you want to copy.

   A dialog box for editing the new widget appears. The name and description from the source widget are prefilled. For example:



7. (Optional) Modify the following details in the widget editor dialog box:

   - **Name**—Name of the new widget. This name appears as the title above the widget in the dashboard.

     You should avoid using special characters, which are described in Using Special Characters in ElectricFlow Object Names on page 1.

   - (Optional) **Description**—Description of the new widget.

8. Click **Define** and then edit the definition of the widget.

   For details, see Defining the Dashboard Widgets on page 795

Authoring New Dashboards Using DSL

This section describes how to use DSL to create your own report definitions and use them to build dashboards for visualizing the report data. ElectricFlow also provides REST and perl APIs for creating the reports and dashboards in addition to DSL.

To build a dashboard, you start by defining the reports to be used by your dashboard. A dashboard is the overall page that contains all of your charts and reports. For example, the Releases dashboard has all the charts and reports for Releases. You can create your own dashboard pages as needed. Then you can define which custom reports and charts to show in a dashboard page.

Here is an example of a report definition that gets the total number of application deployments that deployed microservices on a container orchestration platform:

```
projectName 'MyProject', {
    report 'TotalContainerDeployments', {
```

```
        title = 'Total Number of Container Deployments'
       reportObjectTypeName = 'deployment'
       uri = 'ef-deployment-*/_search?pretty'
       definition = '''
{
"size": 0,
"query": {
    "bool" : {
        "filter": [
            {"term" : {  "reportEventType" : "ef_process_run_completed" }},
            {"term" : { "jobStepType" : "service" }}
        ],
        "filter": [
            @@ELECTRIC_FLOW_REPORT_FILTERS@@
        ]
    }
  },
   "aggregations": {
      "deployment": {
         "terms": {
            "field": "deploymentOutcome"
         }
      }
   }
}'''
      }
}
```

Now, you can refer to this report definition in your dashboard. Here is sample DSL for adding this report to 'My New Dashboard':

```
projectName 'MyProject', {
 dashboard 'My New Dashboard', {


  reportingFilter 'DateFilter', {
    parameterName = 'jobFinish'
    type = 'DATE'
    operator = 'BETWEEN'
    required = '1'
  }

  reportingFilter 'ApplicationFilter', {
    type = 'APPLICATION'
    operator = 'IN'
    required = '0'
  }

  reportingFilter 'EnvironmentFilter', {
    type = 'ENVIRONMENT'
    operator = 'IN'
    required = '0'
  }


  widget 'TotalContainerDeployments',
            clearAttributeDataTypes: true,
```

```
            clearAttributePaths: true,
            clearColors: true, {

    description = 'Total container deployments by outcome'
    reportName = 'TotalContainerDeployments'
    visualization = 'DONUT_CHART'
    attributePath = ['xAxis': 'deployment',
                     'yAxis': 'deployment_count',
                     'total': 'total',
                ]
    attributeDataType = ['xAxis': 'STRING',
                         'yAxis': 'NUMBER',
                         'total': 'NUMBER',
                         ]
    color = ['success': '#70b723',
             'warning': '#DA833E',
             'error':   '#eb1c24',
             'aborted': '#808080',
             'rollback':'#567b99']
  }
 }
}
```

DevOps Insight Report Object Types and Attributes

You use attributes for DevOps Insight report objects when creating custom reports and dashboards. The following tables list the attributes for the DevOps Insight report object types as follows:

- Build on page 829
- Build Commit on page 831
- Code Commit on page 833
- Defect on page 835
- Deployment on page 837
- Feature on page 843
- Incident on page 845
- Job on page 848
- Pipeline Run on page 850
- Quality on page 856
- Release on page 858

Build

Object Type: Build Report object name: build Description: Build data from a continuous integration system such as Jenkins			
Attribute	**Display Name**	**Type**	**Description**
baseDrilldownUrl	Base Drilldown Url	STRING	Base URL used to construct the full URL for drilling down from a widget into an external build system such as Jenkins. For example, https://10.200.1.171:8080/job/CrossWing.
buildNumber	Build Number	STRING	Unique build number or identifier assigned to the build by the build system.
buildStatus	Build Status	STRING	Current status of the build: SUCCESS, FAILURE, UNSTABLE (Jenkins only), NOT_BUILT, ABORTED, or WARNING.
duration	Duration	NUMBER	Duration of the build in milliseconds
endTime	End Time	DATETIME	End date and time for the build. The format is ISO 8601 in UTC time. For example, 2017-01-01T11:54:58.569Z.
launchedBy	Launched By	STRING	User or process that launched the build.
pluginConfiguration	Plugin Configuration	STRING	Name of the plugin configuration if the data was populated through EC-Jenkins.
pluginName	Plugin Name	STRING	Name of the plugin if the data was populated through a plugin such as EC-Jenkins. For example, EC-Jenkins.
projectName	Build System Project Name	STRING	Name of the project in the build system. For example, the Jenkins project name for Jenkins builds, the ElectricFlow project name for ElectricSentry builds, or the build project name in TFS.
releaseName	Release Name	STRING	Name of the ElectricFlow release to which the build belongs.
releaseProjectName	Release Project Name	STRING	Name of the ElectricFlow project containing the release to which the build belongs.

Object Type: Build **Report object name: build** **Description: Build data from a continuous integration system such as Jenkins**			
Attribute	**Display Name**	**Type**	**Description**
releaseUri	Release URI	STRING	Partial URL that can be used along with baseDrilldownUrl to construct the drill-down URL to navigate to builds associated with the current release. The continuous integration system must allow such URLs. For Jenkins, DevOps Insight does not use this field, because Jenkins does not provide this ability. You can drill down only into the builds of a specific project in Jenkins.
source	Build System Type	STRING	Name of the continuous integration system. For example, Jenkins.
sourceUrl	Build URL	STRING	URL for the build record. For example, https://10.200.1.171:8080/job/CrossWing.
startTime	Start Time	DATETIME	Starting date and time for the build. The format is ISO 8601 in UTC time. For example, 2017-01-01T11:54:58.569Z.
tags	Tags	STRING	Tags or labels associated with the build.
timestamp	Timestamp	DATETIME	Date and time that the build completed. The system uses this field to determine the time-based index for storing the build record by year. For example, builds that completed in 2017 are stored in the index ef-build-2017. The format is ISO 8601 in UTC time. For example, 2017-01-01T11:54:58.569Z.
jobName	Job Name	STRING	Name of the Job. For ElectricFlow CI builds, this will be set using the name of the job started by ElectricSentry to do the build. Not used for Jenkins builds.

Build Commit

Object Type: Build Commit **Report object name: build_commit** **Description: Source code changesets included in builds done by ElectricFlow or continuous integration systems such as Jenkins**			
Attribute	**Display Name**	**Type**	**Description**
buildUrl	Build URL	STRING	URL for the build record. For example, https://10.200.1.171:8080/job/CrossWing
buildNumber	Build Number	STRING	Unique build number or identifier assigned to the build by the build system
startTime	Start Time	DATETIME	Start date and time for the build. The format is ISO 8601 in UTC time. For example, 2017-01-01T11:54:58.569ZÂ
endTime	End Time	DATETIME	End date and time for the build. The format is ISO 8601 in UTC time. For example, 2017-01-01T11:54:58.569ZÂ
buildStatus	Build Status	STRING	Current status of the build: SUCCESS, FAILURE, UNSTABLE (Jenkins only), NOT_BUILT, ABORTED, or WARNING
projectName	Build System Project Name	STRING	Name of the project in the build system. For example, the Jenkins project name for Jenkins builds, the ElectricFlow project name for ElectricFlow CI builds, or the build project name in TFS
tags	Tags	STRING	Tags or labels associated with the build
buildSource	Build System Type	STRING	Name of the continuous integration system. For example, Jenkins
commitDate	Commit Date	DATETIME	Date when the changeset was committed or checked into the source control system.
commitId	Commit ID	STRING	Unique identifier for the changeset in the source control management system. For example, SHA number in Git or changelist number in Perforce
commitUrl	Commit URL	STRING	URL to the changeset in the SCM repository

Object Type: Build Commit Report object name: build_commit Description: Source code changesets included in builds done by ElectricFlow or continuous integration systems such as Jenkins			
Attribute	**Display Name**	**Type**	**Description**
`jobName`	Job Name	`STRING`	Name of the Job. For ElectricFlow CI builds, this will be set using the name of the job started by ElectricSentry to do the build. Not used for Jenkins builds
`scmRepoBranch`	SCM Repository Branch	`STRING`	Repository branch—Specific to Git or Git-based SCMs. For example, `'master'`
`scmSource`	SCM Type	`STRING`	Name of the source control management system. For example, `Git` or `Perforce`
`scmUrl`	SCM Server URL	`STRING`	URL to the SCM repository or the SCM server name. For example, `'git://git.apache.org/zookeeper.git'`

Code Commit

Object Type: Code Commit **Report object name: code_commit** **Description: Code commit or changeset data from source code management systems such as Git or GitHub**			
Attribute	**Display Name**	**Type**	**Description**
`pluginConfiguration`	Plugin Configuration	`STRING`	Name of the plugin configuration if the data was populated through a plugin such as ECSCM-Git
`pluginName`	Plugin Name	`STRING`	Name of the plugin if the data was populated through a plugin. For example, ECSCM-Git
`source`	SCM Type	`STRING`	Name of the source control management system. For example, `Git` or `Perforce`
`sourceUrl`	Commit URL	`STRING`	URL to the changeset in the SCM repository
`baseDrilldownUrl`	BaseDrilldown Url	`STRING`	Base URL used to construct the full URL for drilling down from a widget into an external source control management system such as Git. For example, `'git://git.apache.org/zookeeper.git'`
`commitDate`	Commit Date	`DATETIME`	Date when the changeset was committed or checked into the source control system
`scmUrl`	SCM Server URL	`STRING`	URL to the SCM repository or the SCM server name. For example, `'git://git.apache.org/zookeeper.git'`
`scmRepoBranch`	SCM Repository Branch	`STRING`	Repository branch. This is specific to Git or Git-based SCMs. For example, `'master'`
`commitId`	Commit ID	`STRING`	Unique identifier for the changeset in the source control management system. For example, the SHA number in Git or the changelist number in Perforce
`commitAuthor`	Commit Author	`STRING`	Changeset author name

Object Type: Code Commit

Report object name: code_commit

Description: Code commit or changeset data from source code management systems such as Git or GitHub

Attribute	Display Name	Type	Description
commitAuthorId	Commit Author ID	STRING	Unique identifier for the changeset author in the source control management system. For example, top-dev@my-org.com
commitMessage	Commit Message	STRING	Commit message provided with the commit or changeset
codeLinesAdded	Code Line Added	NUMBER	Number of code lines added in the commit or changeset
codeLinesUpdated	Code lines Updated	NUMBER	Number of code lines updated in the commit or changeset
codeLinesRemoved	Code Lines Removed	NUMBER	Number of code lines deleted in the commit or changeset
filesAdded	Files Added	NUMBER	Number of files added in the commit or changeset
filesUpdated	File Updated	NUMBER	Number of files updated in the commit or changeset
filesRemoved	Filed Removed	NUMBER	Number of files removed in the commit or changeset

Defect

Object Type: Defect **Report object name: defect** **Description: Defects logged in defect tracking systems such as Atlassian JIRA**			
Attribute	**Dis-play Name**	**Typ-e**	**Description**
`baseDrilldownUrl`	Base Drilldown Url	`STRING`	Base URL used to construct the full URL for drilling down from a widget into a defect-tracking system such as Atlassian JIRA. For example, `http://jira101.myco.com?jql=project%3DTEST%20AND%20issuetype%3DBug`
`createdOn`	Created On	`DATETIME`	Date and time that the defect was created. The format is ISO 8601 in UTC time. For example, `2017-01-01T11:54:58.569Z`
`defectName`	Defect Name	`STRING`	Name of the defect
`key`	Unique Defect ID	`STRING`	Unique identifier for the defect in the defect tracking system
`modifiedOn`	Modified On	`DATETIME`	Date and time that the defect was last modified. The format is ISO 8601 in UTC time. For example, `2017-01-01T11:54:58.569Z`
`pluginConfiguration`	Plugin Configuration	`STRING`	Name of the plugin configuration if the data was populated through EC-JIRA
`pluginName`	Plugin Name	`STRING`	Name of the plugin if the data was populated through EC-JIRA. For example, `EC-JIRA`
`releaseName`	Release Name	`STRING`	Name of the release containing the defect
`releaseProjectName`	Release Project Name	`STRING`	Name of the project containing the defect

Object Type: Defect Report object name: defect Description: Defects logged in defect tracking systems such as Atlassian JIRA			
Attribute	**Dis-play Name**	**Typ-e**	**Description**
releaseUri	Release URI	STRING	Partial URL that can be used along with baseDrilldownUrl to construct the drill-down URL to navigate to the defects associated with the current release in the project management system. For example, for JIRA, if baseDrilldownUrl is set to http://jira101.myco.com?jql=project%3DTEST%20AND%20issuetype%3DBug, and the defects associated with the current release are assigned fixVersion value 7.5, then the release URI can be set as fixVersion=7.5 to then construct the external URL in the widget definition as ${baseDrilldownUrl}%20AND%20${releaseUri}. This is expanded when the widget is rendered in the UI to http://jira101.myco.com?jql=project%3DTEST%20AND%20issuetype%3DBug%20AND%20fixVersion=7.5 as a drill-down link
resolution	Resolution	STRING	Current resolution of the defect. The possible values depend on the defect tracking system
resolvedOn	Resolved On	DATETIME	Date and time that the defect was resolved. The format is ISO 8601 in UTC time. For example, 2017-01-01T11:54:58.569Z
source	Source	STRING	Name of the defect tracking system. For example, JIRA
sourceUrl	Source Url	STRING	URL for the defect record. For example, http://jira101.myco.com?jql=project%3DTEST%20AND%20issuetype%3DBug
status	Status	STRING	Current status of the defect: Open, Inprogress, Resolved, or Closed
timestamp	Timestamp	DATETIME	Date and time when the defect record was last updated. The system uses this field to determine the time-based index for storing the defect records by year. For example, defects updated in 2017 are stored in the index ef-defect-2017. The format is ISO 8601 in UTC time. For example, 2017-01-01T11:54:58.569Z
type	Type	STRING	Defect type. The possible values depend on the defect tracking system. The possible value for JIRA is Bug.

Deployment

Object Type: Deployment Report object name: deployment Description: Data from application or microservice deployments in ElectricFlow such as deployment status and outcome			
Attribute	**Display Name**	**Type**	**Description**
applicationId	Application ID	STRING	Unique ID of the application that was deployed
applicationName	Application Name	STRING	Name of the application
approvedBy	Approved By	STRING	User who approved the manual step
clusterName	Cluster Name	STRING	Name of the cluster to which the service was deployed
comment	Comment	STRING	Comment provided when the user completed the manual step
deploymentOutcome	Deployment Outcome	STRING	Outcome of the deployment: Success, Error, Warning, or Skipped (check the job outcome)
elapsedTime	Elapsed Time	STRING	Time taken by the deployment to complete (in milliseconds)
environmentId	Environment ID	STRING	Unique ID of the environment to which the application or service was deployed
environmentName	Environment Name	STRING	Name of the environment
environmentProjectName	Environment Project Name	STRING	Name of the environment project
flowRuntimeId	Flow Runtime ID	STRING	ID of the pipeline run that started or triggered the deployment
flowRuntimeName	Flow Runtime Name	STRING	Name of the pipeline run that started deployment

Object Type: Deployment Report object name: deployment Description: Data from application or microservice deployments in ElectricFlow such as deployment status and outcome			
Attribute	**Display Name**	**Type**	**Description**
flowRuntimeStateName	Flow Runtime State Name	STRING	Name of the stage or task that started the deployment
groupName	Group Name	STRING	Name of the group containing the task
jobAbortStatus	Job Abort Status	STRING	Abort status if the job was aborted: ABORT or FORCEABORT
jobAbortedBy	Job Aborted By	STRING	User who aborted the job
jobFinish	Job Finish	DATETIME	Job completion time
jobId	Job ID	STRING	Unique job ID
jobName	Job Name	STRING	Name of the job
jobOutcome	Job Outcome	STRING	Outcome of the job: Success, Error, Warning, or Skipped
jobStart	Job Start	DATETIME	Job start time
jobStatus	Job Status	STRING	Status of the job: pending, runnable, running, or scheduled
jobStepAbortStatus	Job Step Abort Status	STRING	Abort status of the job step: ABORT or FORCE_ABORT
jobStepAbortedBy	Job Step Aborted By	STRING	User or project that aborted the job step
jobStepName	Job Step Name	STRING	Name of the job step

Object Type: Deployment Report object name: deployment Description: Data from application or microservice deployments in ElectricFlow such as deployment status and outcome			
Attribute	**Display Name**	**Type**	**Description**
jobStepOutcome	Job Step Outcome	STRING	Outcome of the job step: Success, Error, Warning, or Skipped
jobStepStatus	Job Step Status	STRING	Status of the job step: Pending, Runnable, Scheduled, Running, or Completed
jobStepType	Job Step Type	STRING	Type of the job step
jobType	Job Type	STRING	Type of the job: rollback (set only for rollback jobs)
launchedByUser	Launched By User	STRING	User who launched the job
manual	Manual	BOOLEAN	Boolean flag to indicate if a step is manual
parentTaskId	Parent Task ID	STRING	Id of parent task.
parentTaskName	Parent Task Name	STRING	Name of the parent task
parentTaskType	Parent Task Type	STRING	Task type of the parent task
pipelineId	Pipeline ID	STRING	ID of the pipeline containing the task that launched the job
pipelineName	Pipeline Name	STRING	Name of the pipeline containing the task that launched the job
pipelineRunOutcome	Pipeline Run Outcome	STRING	Outcome of the pipeline run: Success, Error, Warning, Skipped, or Aborted
plannedEndTime	Planned End Time	DATETIME	Planned end date of the task that triggered the deployment.

Object Type: Deployment Report object name: deployment Description: Data from application or microservice deployments in ElectricFlow such as deployment status and outcome			
Attribute	**Display Name**	**Type**	**Description**
plannedStartTime	Planned Start Time	DATETIME	Planned start date of the task that triggered the deployment.
pluginDisplayName	Plugin Display Name	STRING	Display name of the plugin
pluginKey	Plugin Key	STRING	Key of the plugin
processId	Process ID	STRING	ID of the process that launched the job
processName	Process Name	STRING	Name of the process that launched the job
processStepName	Process Step Name	STRING	Name of the process step in the process
projectId	Project ID	STRING	ID of the project
projectName	Project Name	STRING	Name of the project
releaseId	Release ID	STRING	ID of the release to which the pipeline runs belongs
releaseName	Release Name	STRING	Name of the release to which the pipeline run belongs
releaseProjectName	Release Project Name	STRING	Name of the project containing the release

Object Type: Deployment Report object name: deployment Description: Data from application or microservice deployments in ElectricFlow such as deployment status and outcome			
Attribute	**Display Name**	**Type**	**Description**
reportEventType	Report Event Type	STRING	Type of report event: ef_pipeline_run_completed ef_pipeline_run_gate_completed ef_pipeline_run_gate_started ef_pipeline_run_stage_completed ef_pipeline_run_stage_started ef_pipeline_run_started ef_pipeline_run_task_completed ef_pipeline_run_task_started ef_process_run_completed ef_process_run_rollback_step_completed ef_process_run_rollback_step_started ef_process_run_started ef_process_run_step_retry_completed ef_process_run_step_retry_started ef_process_run_waiting_on_manual ef_process_run_waiting_on_manual_completed
retries	Retries	NUMBER	Number of times to retry a step upon an error
retryType	Retry Type	STRING	Type of retry: auto or manual
serviceId	Service ID	STRING	ID of the service associated with the job
serviceName	Service Name	STRING	Name of the service associated with the job
stageName	Stage Name	STRING	Name of the stage containing the task that launched the job
taskName	Task Name	STRING	Name of the task that launched the job

Object Type: Deployment **Report object name: deployment** **Description: Data from application or microservice deployments in ElectricFlow such as deployment status and outcome**			
Attribute	**Display Name**	**Type**	**Description**
taskStatus	Task Status	STRING	Status of the task
taskType	Taks Type	STRING	Type of the task: APPROVAL COMMAND CONDITIONAL DEPLOYER GROUP MANUAL PIPELINE PLUGIN PROCEDURE PROCESS RELEASE UTILITY WORKFLOW
tierMapName	Tier Map Name	STRING	Name of the tier map
waitForPlannedStart	Wait For Planned Start	BOOLEAN	Boolean flag that represents whether a task should wait for a planned start date before running.

Feature

Object Type: Feature **Report object name: feature** **Description: Features or stories tracked in project management tools such as Atlassian JIRA**			
Attribute	**Dis-play Name**	**Typ-e**	**Description**
`baseDrilldownUrl`	Base Drilldown Url	`STRING`	Base URL used to construct the full URL for drilling down from a widget into a system such as JIRA. For example, `http://jira101.myco.com?jql=project%3DTEST%20AND%20issuetype%3DStory`
`createdOn`	Created On	`DATETIME`	Date and time that the feature was created. The format is ISO 8601 in UTC time. For example, `2017-01-01T11:54:58.569Z`
`featureName`	Feature Name	`STRING`	Name of the feature
`featuretype`	Feature Type	`STRING`	Type of the feature : `Improvement`, `New Feature`, or `Story`
`key`	Unique Feature ID	`STRING`	Unique identifier for the feature in the project management tool
`modifiedOn`	Modified On	`DATETIME`	Date and time that the feature was last modified. The format is ISO 8601 in UTC time. For example, `2017-01-01T11:54:58.569Z`
`pluginConfiguration`	Plugin Configuration	`STRING`	Name of the plugin configuration if the data was populated through EC-JIRA
`pluginName`	Plugin Name	`STRING`	Name of the plugin if the data was populated through EC-JIRA. For example, `EC-JIRA`
`releaseName`	Release Name	`STRING`	Name of the release containing the feature
`releaseProjectName`	Release Project Name	`STRING`	Name of the project containing the feature

Object Type: Feature **Report object name: feature** **Description: Features or stories tracked in project management tools such as Atlassian JIRA**			
Attribute	**Display Name**	**Type**	**Description**
releaseUri	Release URI	STRING	Partial URL that can be used along with baseDrilldownUrl to construct the drill-down URL to navigate to the features associated with the current release in the project management system. For example, for JIRA, if baseDrilldownUrl was set to http://jira101.myco.com?jql=project%3DTEST%20AND%20issuetype%3DStory, and if the features associated with the current release were assigned fixVersion value 7.5, then releaseUri can be set as fixVersion=7.5 to then construct the external URL in the widget definition as ${baseDrilldownUrl}%20AND%20${releaseUri}. This is expanded when the widget is rendered in the UI to http://jira101.myco.com?jql=project%3DTEST%20AND%20issuetype%3DStory%20AND%20fixVersion=7.5 as a drill-down link
resolution	Resolution	STRING	Current resolution of the feature. Possible values include: Fixed, Duplicate, Won't Fix, Incomplete, or Cannot Reproduce.
resolvedOn	Resolved On	DATETIME	Date and time that the feature was resolved. The format is ISO 8601 in UTC time. For example, 2017-01-01T11:54:58.569Z
source	Source	STRING	Name of the project management tool. For example, JIRA
sourceUrl	Source Url	STRING	URL for the feature record. For example, http://jira101.myco.com/browse/ARR-18001
status	Status	STRING	Current status of the feature: Open, Inprogress, Resolved or Closed
storyPoints	Story Points	NUMBER	Number of story points assigned to the feature. This can be any integer
tags	Tags	STRING	Tags or labels associated with the feature.

Object Type: Feature Report object name: feature Description: Features or stories tracked in project management tools such as Atlassian JIRA			
Attribute	**Dis-play Name**	**Typ-e**	**Description**
timestamp	Timestamp	DATETIME	Date and time that the feature record was last updated. The system uses this field to determine the time-based index for storing the feature records by year. For example, features updated in 2017 are stored in the index ef-feature-2017. The format is ISO 8601 in UTC time. For example, 2017-01-01T11:54:58.569Z

Incident

Object Type: Incident Report object name: incident Description: Incidents logged and managed in ITSM systems such as ServiceNow for the deployed releases (post-production)			
Attribute	**Display Name**	**Type**	**Description**
baseDrilldownUrl	Base Drilldown Url	STRING	Base URL used to construct the full URL for drilling down from a widget into a system such as ServiceNow. For example, https://ven02149.service-now.com/incident_list.do?sysparm_query=^incident_state=6^ORincident_state=7
category	Category	STRING	Category containing the incident. The possible values depend on the ITSM system
configurationItem	Configuration Item	STRING	Configuration item name for the incident in the ITSM system
createdOn	Created On	DATETIME	Date and time that the incident was created. The format is ISO 8601 in UTC time. For example, 2017-01-01T11:54:58.569Z
incidentId	Unique Incident ID	STRING	Unique ID of the incident

Object Type: Incident **Report object name: incident** **Description: Incidents logged and managed in ITSM systems such as ServiceNow for the deployed releases (post-production)**			
Attribute	**Display Name**	**Type**	**Description**
`modifiedOn`	Modified On	`DATETIME`	Date and time that the incident was last modified. The format is ISO 8601 in UTC time. For example, `2017-01-01T11:54:58.569Z`
`pluginConfiguration`	Plugin Configuration	`STRING`	Name of the plugin configuration if the data was populated through EC-ServiceNow
`pluginName`	Plugin Name	`STRING`	Name of the plugin if the data was populated through EC-ServiceNow. For example, `EC-ServiceNow`
`priority`	Priority	`STRING`	Incident priority. The possible values depend on the ITSM system
`releaseName`	Release Name	`STRING`	Name of the release for which the incident occurred
`releaseProjectName`	Release Project Name	`STRING`	Name of the project containing the release for which the incident occurred

Object Type: Incident **Report object name: incident** **Description: Incidents logged and managed in ITSM systems such as ServiceNow for the deployed releases (post-production)**			
Attribute	**Display Name**	**Type**	**Description**
releaseUri	Release URI	STRING	Partial URL that can be used along with baseDrilldownUrl to construct the drill-down URL to navigate to the incidents associated with the current release in the ITSM system. For example, for ServiceNow, if baseDrilldownUrl is set to show all resolved or closed incidents: https://ven02149.service-now.com/incident_list.do?sysparm_query=^incident_state=6^ORincident_state=7, and if the incidents associated with the current release belonged to the online-portal category, then the release URL can be set as category=software to construct the external URL in the widget definition as ${baseDrilldownUrl}^${releaseUri}. This is expanded when the widget is rendered in the UI to https://ven01735.service-now.com/incident_list.do?sysparm_query=^incident_state=6^ORincident_state=7^category=software as a drill-down link
reportedBy	Reported By	STRING	User name of the reporter of the incident
resolvedOn	Resolved On	DATETIME	Date and time that the incident was resolved. The format is ISO 8601 in UTC time. For example, 2017-01-01T11:54:58.569Z
source	Source	STRING	Name of the ITSM system. For example, ServiceNow
sourceUrl	Source URL	STRING	URL for the incident record in the source system. For example, https://ven02149.service-now.com/nav_to.do?uri=incident.do?sys_id=46ee0924a9fe198100f1cf78c198454a

Object Type: Incident Report object name: incident Description: Incidents logged and managed in ITSM systems such as ServiceNow for the deployed releases (post-production)			
Attribute	**Display Name**	**Type**	**Description**
status	Status	STRING	Current status of the incident. Possible values include New, Active, Resolved, or Closed. The actual values depend on the ITSM system
subCategory	Sob Category	STRING	Subcategory containing the incident. The possible values depend on the ITSM system
timestamp	Timestamp	DATETIME	Date and time when the incident was last updated. The system uses this field to determine the time-based index for storing the incident records by year. For example, updated in 2017 are stored in the index ef-incident-2017. The format is ISO 8601 in UTC time. For example, 2017-01-01T11:54:58.569Z

Job

Object Type: ElectricFlow Job Report object name: job Description: Data from ElectricFlow jobs (the output associated with invoking an ElectricFlow procedure or process)			
Attribute	**Display Name**	**Type**	**Description**
applicationId	Application ID	STRING	Unique ID of the application process for which the job was run
applicationName	Application Name	STRING	Name of the application
elapsedTime	Elapsed Time	STRING	Time taken by the job to complete (in milliseconds)

Object Type: ElectricFlow Job

Report object name: job

Description: Data from ElectricFlow jobs (the output associated with invoking an ElectricFlow procedure or process)

Attribute	Display Name	Type	Description
environmentId	Environment ID	STRING	Unique ID of the environment to which the application or service was deployed
environmentName	Environment Name	STRING	Name of the environment
environmentProjectName	Environment Project Name	STRING	Name of the environment project
flowRuntimeId	Pipeline Run ID	STRING	ID of the pipeline run that started or triggered the job
jobFinish	End Time	DATETIME	Job completion time
jobID	Job ID	STRING	Unique job ID
jobName	Job Name	STRING	Name of the job
jobStart	Start Time	DATETIME	Job start time
combinedStatus	Job Result	STRING	Set using combinedStatus. Job details are sent for completed jobs only, so possible values based on terminal combined status are Aborted, Error, Success, Warning, or Skipped
launchedByUser	Launched By User	STRING	User who launched the job
pipelineId	Pipeline ID	STRING	ID of the pipeline containing the task that launched the job
projectName	Project Name	STRING	Name of the project
releaseId	Release ID	STRING	ID of the release to which the pipeline runs belongs

Object Type: ElectricFlow Job Report object name: job Description: Data from ElectricFlow jobs (the output associated with invoking an ElectricFlow procedure or process)			
Attribute	**Display Name**	**Type**	**Description**
releaseName	Release Name	STRING	Name of the release to which the pipeline run belongs
releaseProjectName	Release Project Name	STRING	Name of the project containing the release
serviceId	Service ID	STRING	ID of the service associated with the job
serviceName	Service Name	STRING	Name of the service associated with the job
tags	Tags	STRING	Tags associated with the job

Pipeline Run

Object Type: Pipeline Run Report object name: pipelinerun Description: Data from pipeline runs in ElectricFlow such as pipeline run outcome and runtime start and end times			
Attribute	**Display Name**	**Type**	**Description**
action	Manual task/step action	STRING	Action the user took to complete the manual task or step: `completed` or `failed`
applicationName	Application Name	STRING	Name of the application
applicationProjectName	Application Project Name	STRING	Name of the project containing the application
approvedBy	Approved By	STRING	Name of the user who approved the stage or gate task
approvers	Approvers	STRING	List of approvers for the stage or gate task

Object Type: Pipeline Run

Report object name: pipelinerun

Description: Data from pipeline runs in ElectricFlow such as pipeline run outcome and runtime start and end times

Attribute	Display Name	Type	Description
elapsedTime	Elapsed Time	STRING	Time taken by the pipeline run to complete (in milliseconds)
evidence	Evidence	STRING	Evidence that the task was run
flowRuntimeAbortStatus	Pipeline Run Abort Status	STRING	Abort status of the pipeline run if it was aborted: ABORT or FORCEABORT
flowRuntimeAbortedBy	Pipeline Run Aborted By	STRING	User who aborted the pipeline run
flowRuntimeCompleted	Pipeline Run Completed	BOOLEAN	Boolean flag to indicate if the pipeline run is completed
flowRuntimeFinish	Pipeline Run Finish Time	DATETIME	Time when the pipeline run was completed
flowRuntimeId	Pipeline Run ID	STRING	ID of the pipeline run
flowRuntimeName	Flow Runtime Name	STRING	Name of the pipeline run
flowRuntimeStart	Pipeline Run Start Time	DATETIME	Time when the pipeline run started
flowRuntimeStateAbortStatus	Task/Stage Run abort status	STRING	Abort status of a stage, task, or gate associated with the pipeline run: ABORT or FORCEABORT

Object Type: Pipeline Run **Report object name: pipelinerun** **Description: Data from pipeline runs in ElectricFlow such as pipeline run outcome and runtime start and end times**			
Attribute	**Display Name**	**Type**	**Description**
flowRuntimeStateAbortedBy	Task/Stage Run aborted by	STRING	User who aborted the stage, task, or gate in the pipeline run
flowRuntimeStateFinish	Task/Stage Run finish time	DATETIME	Time when the stage, task, or gate finished execution in the pipeline run
flowRuntimeStateId	Task/Stage Run ID	STRING	ID of the runtime object associated with the stage, task, or gate in the pipeline run
flowRuntimeStateName	Flow Runtime State Name	STRING	Name of the runtime object associated with the stage, task, or gate in the pipeline run
flowRuntimeStateOutcome	Flow Runtime State Outcome	STRING	Outcome of execution of the runtime object associated with the stage, task, or gate in the pipeline run: Success, Error, Warning, or Skipped
flowRuntimeStateStart	Task/Stage Run start time	DATETIME	Time when the stage, task, or gate started execution in the pipeline run
gateType	Gate Type	STRING	Type of gate: PRE or POST
groupName	Group Name	STRING	Name of the group task
manual	Manual	BOOLEAN	Boolean flag to indicate if it is a manual or an approval task
parentTaskId	Parent Task ID	STRING	Id of parent task.

Object Type: Pipeline Run

Report object name: pipelinerun

Description: Data from pipeline runs in ElectricFlow such as pipeline run outcome and runtime start and end times

Attribute	Display Name	Type	Description
parentTaskName	Parent Task Name	STRING	Name of the parent task (either a deployer task or a group task)
parentTaskType	Parent Task Type	STRING	Type of the parent task
pipelineId	Pipeline ID	STRING	Unique ID of the pipeline
pipelineName	Pipeline Name	STRING	Name of the pipeline
pipelineRunOutcome	Pipeline Run Outcome	STRING	Outcome of the pipeline run: Success, Error, Warning, or Skipped
projectId	Project ID	STRING	ID of the project containing the pipeline run
projectName	Project Name	STRING	Name of the project containing the pipeline run
releaseId	Release ID	STRING	ID of the release associated with the pipeline run
releaseName	Release Name	STRING	Name of the release associated with the pipeline run
releaseProjectName	Release Project Name	STRING	Name of the project containing the release associated with the pipeline run

Object Type: Pipeline Run Report object name: pipelinerun Description: Data from pipeline runs in ElectricFlow such as pipeline run outcome and runtime start and end times			
Attribute	**Display Name**	**Type**	**Description**
reportEventType	Report Event Type	STRING	Type of the report event: ef_pipeline_run_completed ef_pipeline_run_gate_completed ef_pipeline_run_gate_started ef_pipeline_run_stage_completed ef_pipeline_run_stage_started ef_pipeline_run_started ef_pipeline_run_task_completed ef_pipeline_run_task_started ef_process_run_completed ef_process_run_rollback_step_completed ef_process_run_rollback_step_started ef_process_run_started ef_process_run_step_retry_completed ef_process_run_step_retry_started ef_process_run_waiting_on_manual ef_process_run_waiting_on_manual_completed
retries	Retries	NUMBER	Number of times to retry the step after an error
retryType	Retry Type	STRING	Type of retry: auto or manual
serviceName	Service Name	STRING	ID of the service
serviceProjectName	Stage Project Name	STRING	Name of the project that contains the service

Object Type: Pipeline Run Report object name: pipelinerun Description: Data from pipeline runs in ElectricFlow such as pipeline run outcome and runtime start and end times			
Attribute	**Display Name**	**Type**	**Description**
stageName	Stage name	STRING	Name of the stage that contains the task
tags	Tags	STRING	Tags associated with a pipeline or release run.
taskName	Task Name	STRING	Name of the task
taskStatus	Task Status	STRING	Status of the task
taskType	Task Type	STRING	Type of the task: APPROVAL COMMAND CONDITIONAL DEPLOYER GROUP MANUAL PIPELINE PLUGIN PROCEDURE PROCESS RELEASE UTILITY WORKFLOW
plannedEndTime	Task/Stage planned end time	DATETIME	Planned end date of the task.

Object Type: Pipeline Run Report object name: pipelinerun Description: Data from pipeline runs in ElectricFlow such as pipeline run outcome and runtime start and end times			
Attribute	**Display Name**	**Type**	**Description**
plannedStartTime	Task/Stage planned start time	DATETIME	Planned start date of the task.
waitForPlannedStart	Wait For Planned Start	BOOLEAN	Boolean flag indicating if a stage or task should wait for a planned start time

Quality

Object Type: Quality Report object name: quality Description: Test results obtained from Application Life System Management systems such as HP ALM or Continuous Integration systems such as Jenkins			
Attribute	**Display Name**	**Type**	**Description**
baseDrilldownUrl	Base Drilldown Url	STRING	Base URL used to construct the full URL for drilling down from a widget into a system such as HP ALM or Jenkins. For example, http://10.200.1.210:8080/qcbin/domain
buildNumber	Build Number	NUMBER	Number assigned to the build by the build system. This field is applicable only for test results obtained from a build. This can be any integer
category	Category	STRING	Category for the tests. For example, unit-test for JUnit test results obtained from Jenkins by the EC-Jenkins plugin and system-test for test results from HP ALM by the EC-ALM plugin
duration	Duration	NUMBER	Duration of the tests in milliseconds
failedTests	Number of Failed Tests	NUMBER	Number of tests that failed. This can be any integer
manual	Manual	BOOLEAN	Whether the tests were run manually

Object Type: Quality

Report object name: quality

Description: Test results obtained from Application Life System Management systems such as HP ALM or Continuous Integration systems such as Jenkins

Attribute	Display Name	Type	Description
pluginConfiguration	Plugin Configuration	STRING	Name of the plugin configuration if the data was populated through EC-Jenkins or EC-ALM
pluginName	Plugin Name	STRING	Name of the plugin if the data was populated through EC-Jenkins or EC-ALM. For example, EC-Jenkins
releaseName	Release Name	STRING	Name of the release to which the test result belongs
releaseProjectName	Release Project Name	STRING	Name of the project containing the release to which the test result belongs
releaseUri	Release URI	STRING	Partial URL that can be used along with baseDrilldownUrl to construct the drill-down URL to navigate to test results associated with the current release. The external system must allow such URLs. For both Jenkins and HP ALM, DevOps Insight does not use this field, because neither system provides this ability. You can drill down only into the builds of a specific job in Jenkins. With HP ALM, you can drill down only into the project in HP ALM
runId	Test Run ID	STRING	Unique ID of the test run
skippedTests	Number of Skipped Tests	NUMBER	Number of skipped tests. This can be any integer
source	Source	STRING	Name of the source system from which the test results were obtained. For example, Jenkins
sourceUrl	Source Url	STRING	URL for the test result record in the source system. For example, https://10.200.1.171:8080/job/CrossWing
status	Status	STRING	Status of the test runs. The possible values depend on the source system

Object Type: Quality Report object name: quality Description: Test results obtained from Application Life System Management systems such as HP ALM or Continuous Integration systems such as Jenkins			
Attribute	**Display Name**	**Type**	**Description**
successfulTests	Number of Successful Tests	NUMBER	Number of tests that have succeeded. This can be any integer
timestamp	Timestamp	DATETIME	Date and time that the tests completed. The system uses this field to determine the time-based index for storing the test quality record by year. For example, tests that completed in 2018 are stored in the index ef-quality-2018

Release

Object Type: Release Report object name: release Description: Data from application or microservice releases in ElectricFlow such as planned and actual start and end times			
Attribute	**Display Name**	**Type**	**Description**
actualEndTime	Actual End Time	DATETIME	Actual end time for the release
actualStartTime	Actual Start Time	DATETIME	Actual start time for the release
plannedEndTime	Planned End Time	DATETIME	Planned end date for the release
plannedStartTime	Planned Start Time	DATETIME	Planned start date for the release
releaseId	Release ID	STRING	Unique ID for the release
releaseDuration	Release Duration	NUMBER	Duration of the release in milliseconds
releaseName	Release Name	STRING	Name of the release

Object Type: Release **Report object name: release** **Description: Data from application or microservice releases in ElectricFlow such as planned and actual start and end times**			
Attribute	**Display Name**	**Type**	**Description**
releaseProjectName	Release Project Name	STRING	Name of the project containing the release
tags	Tags	STRING	Tags associated with the release

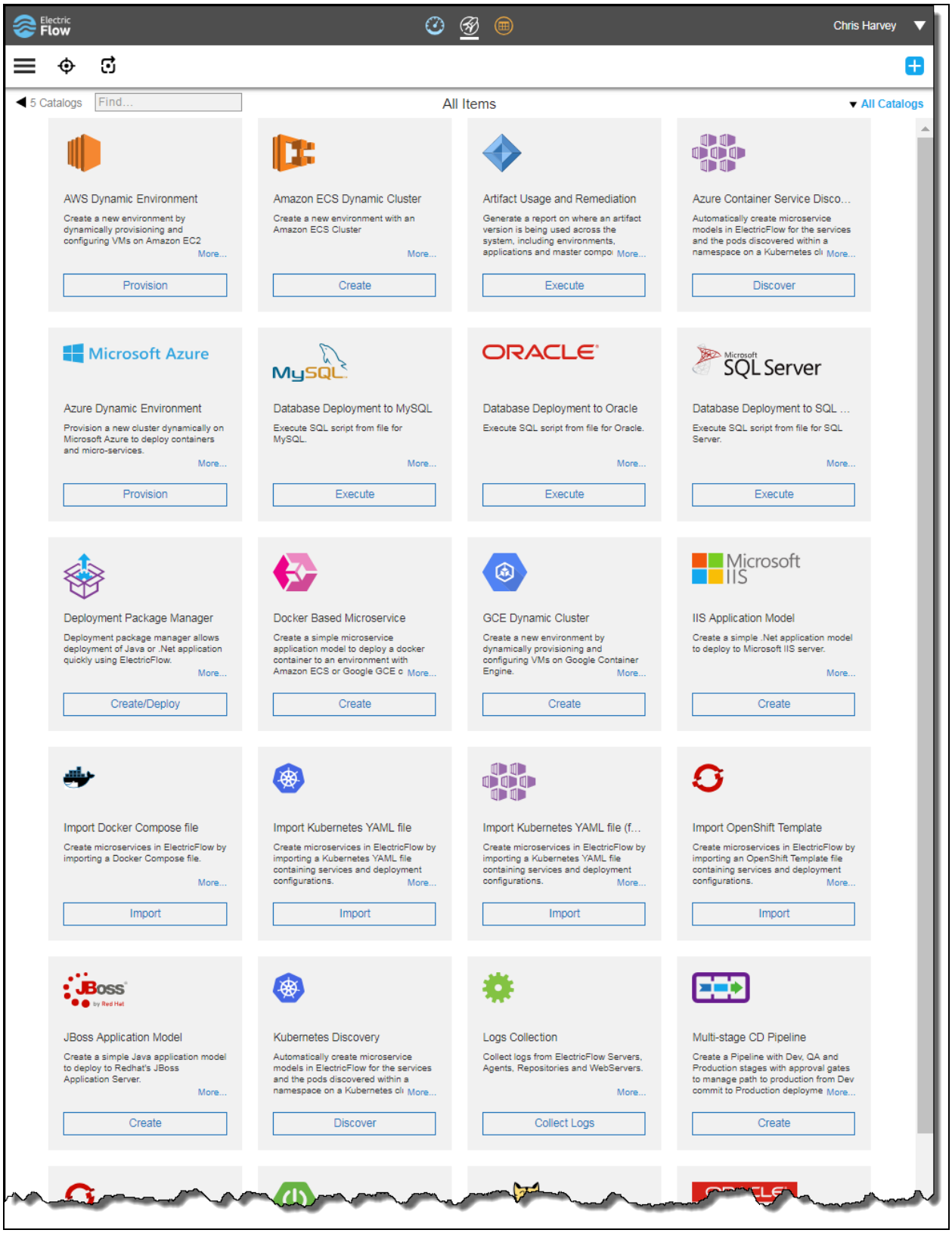
Chapter 5: Self-Service Catalogs

The ElectricFlow Self-Service Catalogs help you accelerate application deployment. This feature lets you automate routine deployment tasks and publish them as templates so that end users can use them by providing minimal information and without learning to use ElectricFlow.

How End Users Use the Self Service Catalogs

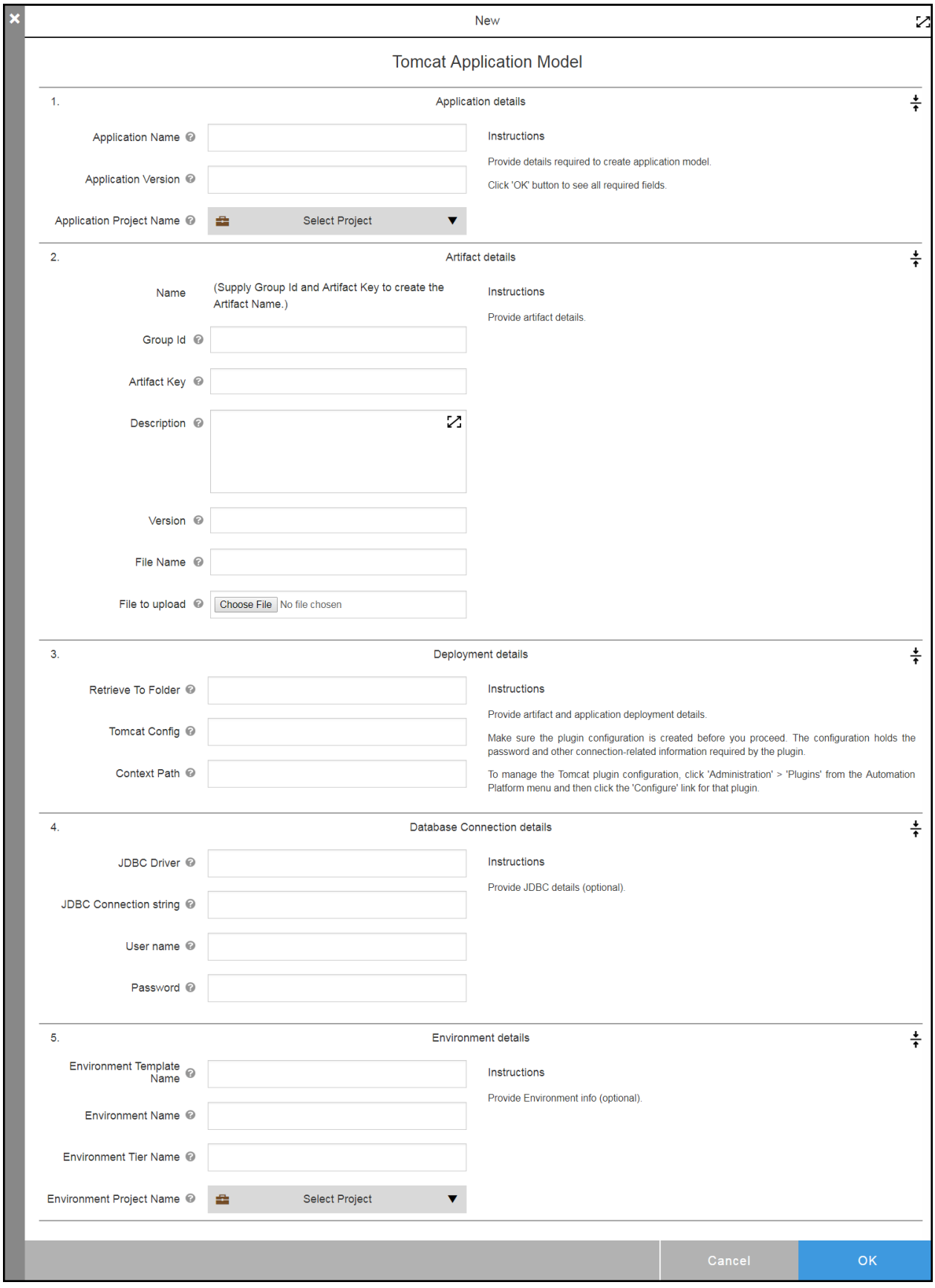
End users click the (Self Service Catalogs) button to open the Self-Service Catalog UI to see all the available catalog items. By default, ElectricFlow includes four categories of catalog item: Applications, Pipelines, Containers, and Cloud Provisioning. All catalogs that are included with ElectricFlow are in the Electric Cloud project.

Following is an example of the Self-Service catalog UI. The catalogs in this example are bundled with ElectricFlow:


Electric Flow
Chris Harvey
5 Catalogs
Find...
All Items
All Catalogs
AWS Dynamic Environment
Create a new environment by dynamically provisioning and configuring VMs on Amazon EC2
More...
Provision
Amazon ECS Dynamic Cluster
Create a new environment with an Amazon ECS Cluster
More...
Create
Artifact Usage and Remediation
Generate a report on where an artifact version is being used across the system, including environments, applications and master compo
More...
Execute
Azure Container Service Disco...
Automatically create microservice models in ElectricFlow for the services and the pods discovered within a namespace on a Kubernetes cl
More...
Discover
Microsoft Azure
Azure Dynamic Environment
Provision a new cluster dynamically on Microsoft Azure to deploy containers and micro-services.
More...
Provision
MySQL
Database Deployment to MySQL
Execute SQL script from file for MySQL.
More...
Execute
ORACLE
Database Deployment to Oracle
Execute SQL script from file for Oracle.
More...
Execute
Microsoft SQL Server
Database Deployment to SQL ...
Execute SQL script from file for SQL Server.
More...
Execute
Deployment Package Manager
Deployment package manager allows deployment of Java or .Net application quickly using ElectricFlow.
More...
Create/Deploy
Docker Based Microservice
Create a simple microservice application model to deploy a docker container to an environment with Amazon ECS or Google GCE c
More...
Create
GCE Dynamic Cluster
Create a new environment by dynamically provisioning and configuring VMs on Google Container Engine.
More...
Create
Microsoft IIS
IIS Application Model
Create a simple .Net application model to deploy to Microsoft IIS server.
More...
Create
Import Docker Compose file
Create microservices in ElectricFlow by importing a Docker Compose file.
More...
Import
Import Kubernetes YAML file
Create microservices in ElectricFlow by importing a Kubernetes YAML file containing services and deployment configurations.
More...
Import
Import Kubernetes YAML file (f...
Create microservices in ElectricFlow by importing a Kubernetes YAML file containing services and deployment configurations.
More...
Import
Import OpenShift Template
Create microservices in ElectricFlow by importing an OpenShift Template file containing services and deployment configurations.
More...
Import
JBoss by Red Hat
JBoss Application Model
Create a simple Java application model to deploy to Redhat's JBoss Application Server.
More...
Create
Kubernetes Discovery
Automatically create microservice models in ElectricFlow for the services and the pods discovered within a namespace on a Kubernetes cl
More...
Discover
Logs Collection
Collect logs from ElectricFlow Servers, Agents, Repositories and WebServers.
More...
Collect Logs
Multi-stage CD Pipeline
Create a Pipeline with Dev, QA and Production stages with approval gates to manage path to production from Dev commit to Production deployme
More...
Create

To use a catalog item, an end user clicks the button (such as **Create**, **Execute**, **Import**, **Discover**, or **Provision**) on an item, then enters the required parameters, and then clicks **OK** to process the underlying catalog template.

Following is an example form for how end users will enter parameters after clicking **Create**. This example is for the **Tomcat Application Model** catalog item, which is bundled with ElectricFlow:


New
Tomcat Application Model
1.
Application details
Application Name
Application Version
Application Project Name
Select Project
Instructions
Provide details required to create application model.
Click 'OK' button to see all required fields.
2.
Artifact details
Name
(Supply Group Id and Artifact Key to create the Artifact Name.)
Group Id
Artifact Key
Description
Version
File Name
File to upload
Choose File
No file chosen
Instructions
Provide artifact details.
3.
Deployment details
Retrieve To Folder
Tomcat Config
Context Path
Instructions
Provide artifact and application deployment details.
Make sure the plugin configuration is created before you proceed. The configuration holds the password and other connection-related information required by the plugin.
To manage the Tomcat plugin configuration, click 'Administration' > 'Plugins' from the Automation Platform menu and then click the 'Configure' link for that plugin.
4.
Database Connection details
JDBC Driver
JDBC Connection string
User name
Password
Instructions
Provide JDBC details (optional).
5.
Environment details
Environment Template Name
Environment Name
Environment Tier Name
Environment Project Name
Select Project
Instructions
Provide Environment info (optional).
Cancel
OK

Once the DSL template processing is complete, an end user navigates to the ElectricFlow UI to see the object that was created. For the Tomcat catalog item example, an end user can navigate to the new application, complete the environment mapping, and then deploy the application.

You can extend a catalog by creating catalog items that are specific to your business. Sample DSL to create catalog items is included in ElectricFlow as a starting point.

Key Benefits of Self Service Catalogs

ElectricFlow Self Service Catalogs have the following key benefits to your organization:

- Promotes best practices in your organization
- Accelerates ROI from your deployment solution
- Improves deployment efficiency by reducing the learning curve

Catalog Items

A catalog item is made up of DSL templates or procedures accompanied by UI form metadata to capture the user input required by the DSL or the procedure. ElectricFlow includes four major classes of Service Catalog item:

- DSL-based
- Plugin-based
- Procedure-based
- Deployment package manager

DSL-Based Catalog Items

DSL-based catalog items are the easiest to create. You use the ElectricFlow UI to author a model, then export the DSL, and then convert the DSL to a template before exposing it to end users as a catalog item.

Following is a simple DSL template. This template creates an application in the Default project folder:

```
 /* args.appName is user entered value passed in as argument */
def appName = args.appName;

application appName, {
   projectName = "Default"
}
```

You populate the `dslParamForm` attribute of the catalog item with UI form metadata to collect the user input for the application name parameter:

```
 {"sections": {"section": [{"name": "Required Params","ec_parameterForm":
"<editor><formElement><property>appName</property><label>ApplicationName</label><ty
pe>entry</type><required>1</required</formElement></editor>"}],"endTarget": {..}}}
```

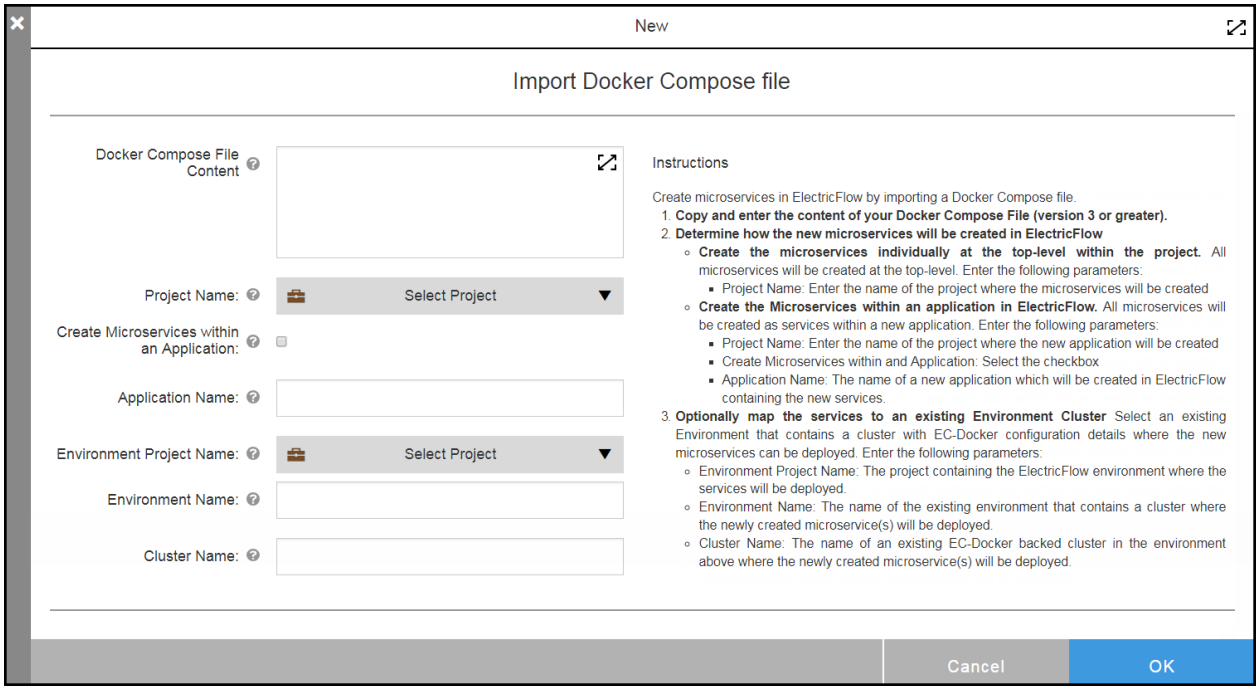
- `section` allows groupings of user input parameters into different sections.
- `ec_parameterForm` specifies the form XML that defines the user input form relevant to a section.
- `endTarget` specifies where to navigate the user after the catalog item is processed. Valid targets are the `application`, `environment`, `release`, `pipeline`, or `jobs` listing pages.

Plugin-Based Catalog Items

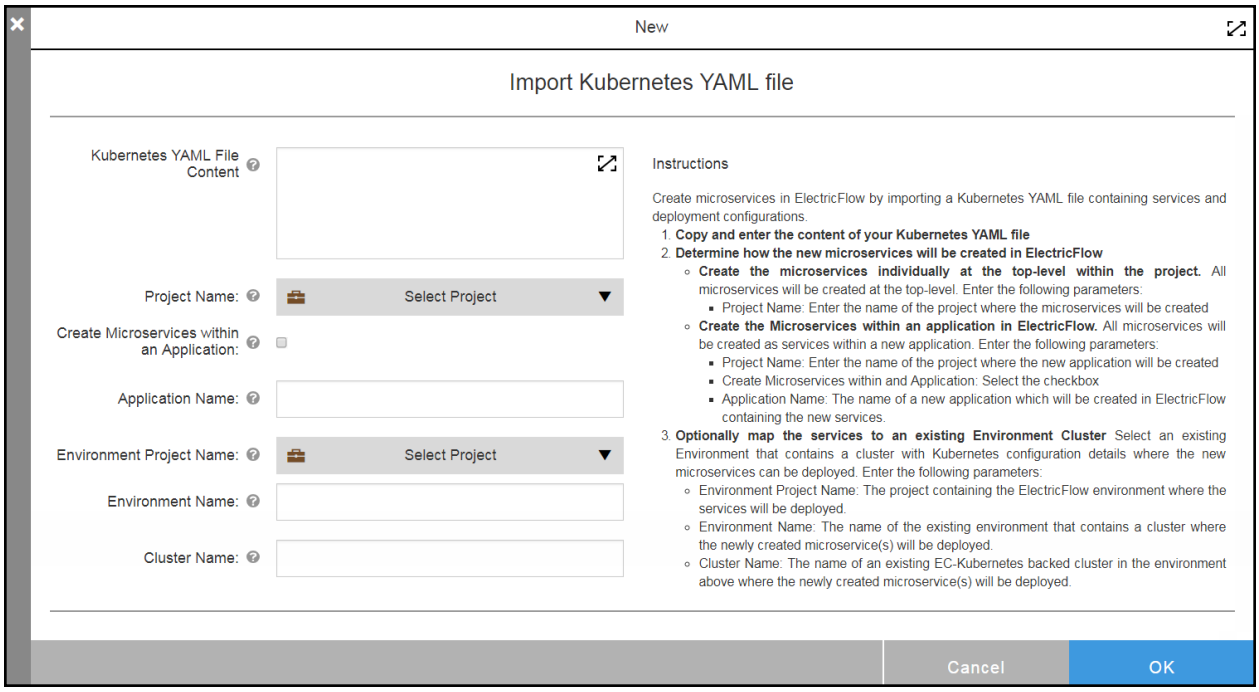
With plugin-based catalog items, you can expose any existing plugin procedure as a Service Catalog item. You can expose any useful procedure or plugins directly to end users.

As with DSL, for each procedure to expose in a Service Catalog, you must create additional metadata. For a simple use case, you just create the `ec_customEditorData/selfServiceUIForm` property. This property tells the catalog where to find the UI form to render that corresponds to the procedure. The Service Catalog UI uses this information to render the form. When the user fills in the form, that information is passed to the procedure.

Following is an example form for entering parameters for the bundled **Import Docker Compose File** catalog item, which is plugin-based:



Following is an example form for entering parameters for the bundled **Import Kubernetes YAML File** catalog item, which is also plugin-based:



Note: If the `ec_customEditorData/selfServiceUIForm` property is not found, then the system will try to render the UI form based on the parameter definition with the exception for Credentials, Credential Parameters, and Projects types.

Procedure-Based Catalog Items

With procedure-based catalog items, you can expose any existing procedure as a Service Catalog item. You can expose any useful procedure or plugins directly to end users.

As with DSL, for each procedure to expose in a Service Catalog, you must create additional metadata. For a simple use case, you just need to create the `ec_customEditorData/selfServiceUIForm` property. This property tells the catalog where to find the UI form to render that corresponds to the procedure. The Service Catalog UI uses this information to render the form. When the user fills in the form, that information is passed to the procedure.

Note: If the `ec_customEditorData/selfServiceUIForm` property is not found, then the system will try to render the UI form based on the parameter definition with the exception for Credentials, Credential Parameters, and Projects types.

Deployment Package Manager Catalog Items

This is a special type of catalog item that lets you deploy Java or .NET applications quickly. This item is included in ElectricFlow.

Before you create this type of item, you must create the deployment package outside of ElectricFlow. A deployment package is a `manifest.json` file plus one or more components that make up an application. After you create the deployment package, you can use it to create an application model that

.

Sample Catalog Item DSL

Here is sample code to create a catalog item in a catalog named Custom. This is a DSL-based catalog item that creates a simple application object:

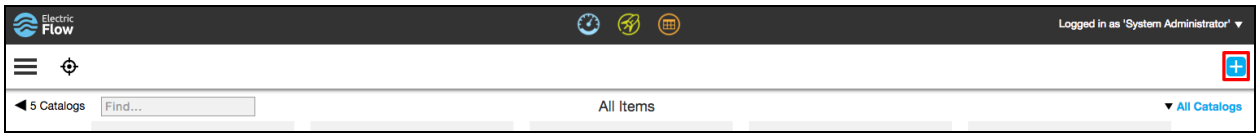
```
catalog 'Foo', {
  description = 'Catalog item description'
  iconUrl = 'icon'
  projectName = 'Electric Cloud'
  catalogItem 'Sample Catalog Item', {
    description = '''<xml>
         <title>This is the short description that is shown in the Service catalog UI
as part of this catalog item</title>
         <htmlData>
           <![CDATA[<div>
              This is the html stylized text based on catalog item details that gets
show when the user clicks the more link.
           </div>]]>
         </htmlData>
    </xml>'''
    buttonLabel = 'Create'
    catalogName = 'Custom'
    dslParamForm = '''{
      \"sections\": {
        \"section\": [{
          \"name\": \"Application details\",
          \"instruction\": \"Provide Application details\",
          \"ec_parameterForm\": \"<editor> <formElement> <label>Application
Name</label> <property>appName</property> <documentation>Name of the application to be
created.</documentation> <type>entry</type> <required>1</required> </formElement>
</editor>\"
        }],
        \"endTarget\": {
          \"object\": \"application\",
          \"formValue\": \"appName\"
        }
      }
    }'''
    dslString = '''def appName = args.appName;
                application appName, {
                projectName = "Default"
     }'''
    iconUrl = 'public/app/assets/img/taskicon.png'
    projectName = 'Electric Cloud'
    subpluginKey = null
    subprocedure = null
    subproject = null
  }
}
```

Managing the Service Catalogs

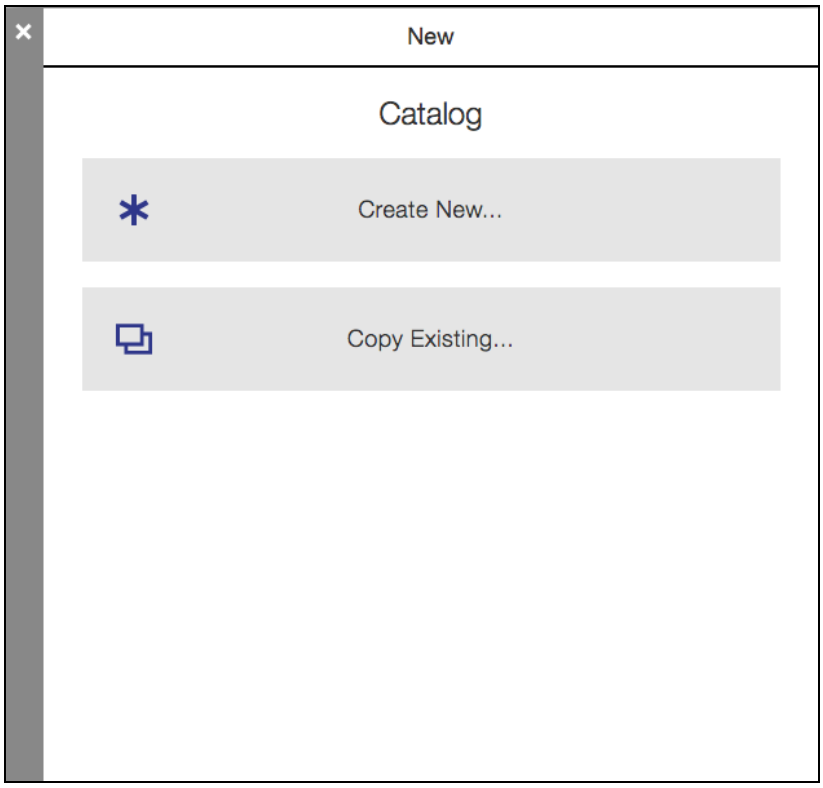
You can create self-service catalogs and catalog items directly from the UI.

Creating a Catalog

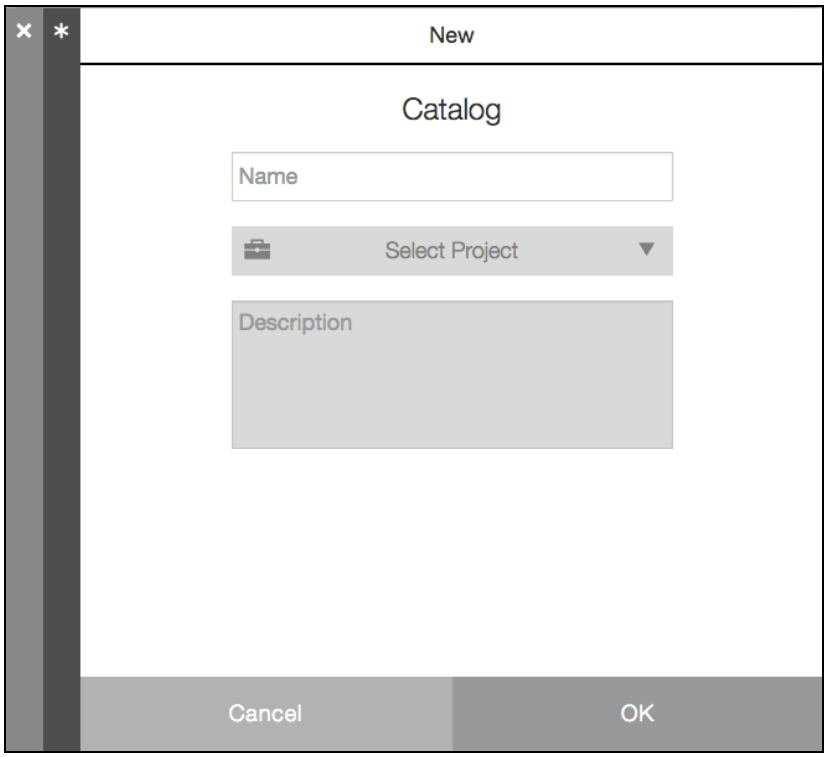
In the self-service catalog UI, click the (Add) button to create a new catalog:



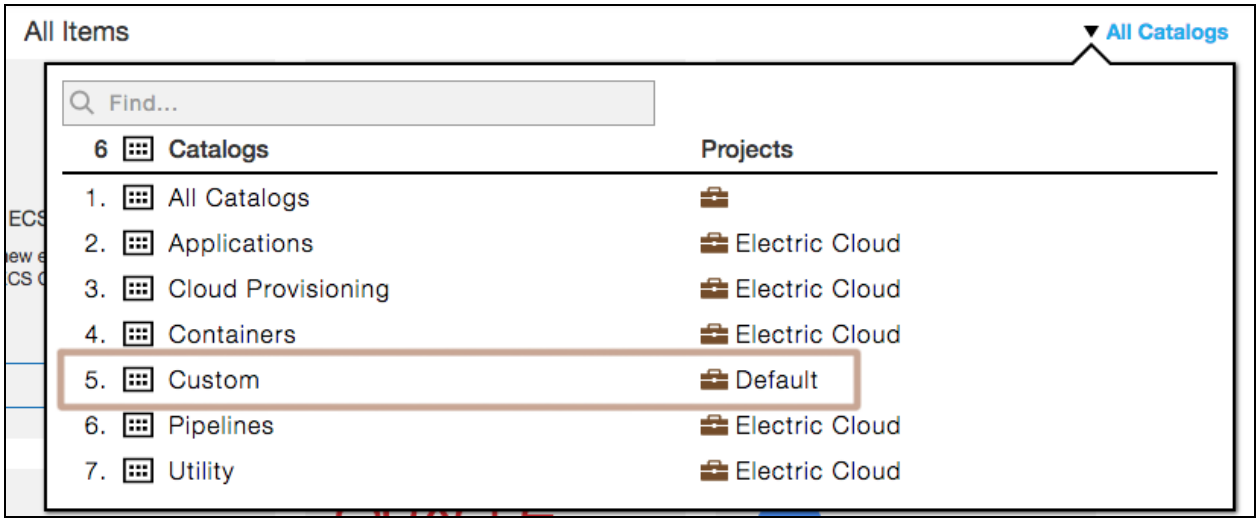
Then from the new **Catalog** dialog, select **Create New**.



Then enter a name, an optional description, and select a project where you want to create the new catalog item and click **OK**:



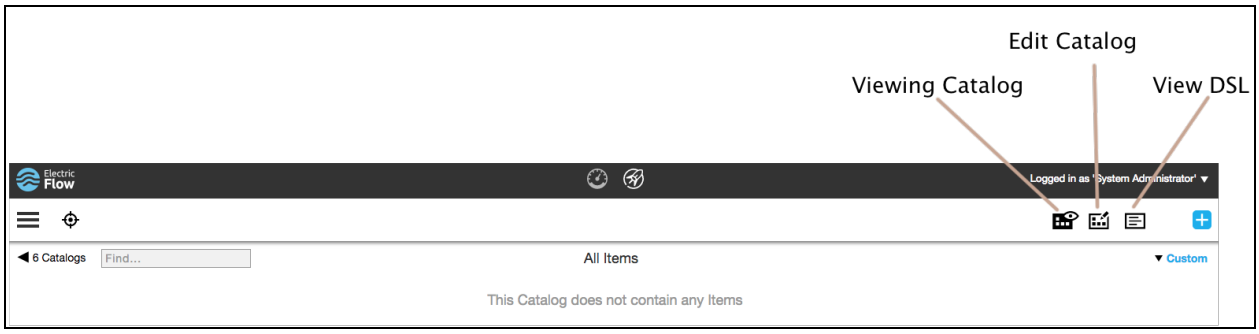
Once the new catalog item is created, the new catalog item appears in the **Catalogs** drop-down menu:



Note:
You cannot edit catalogs or catalog items that are bundled with ElectricFlow. However, you can use the **Copy Existing** option to create new catalogs from catalogs bundled with ElectricFlow.

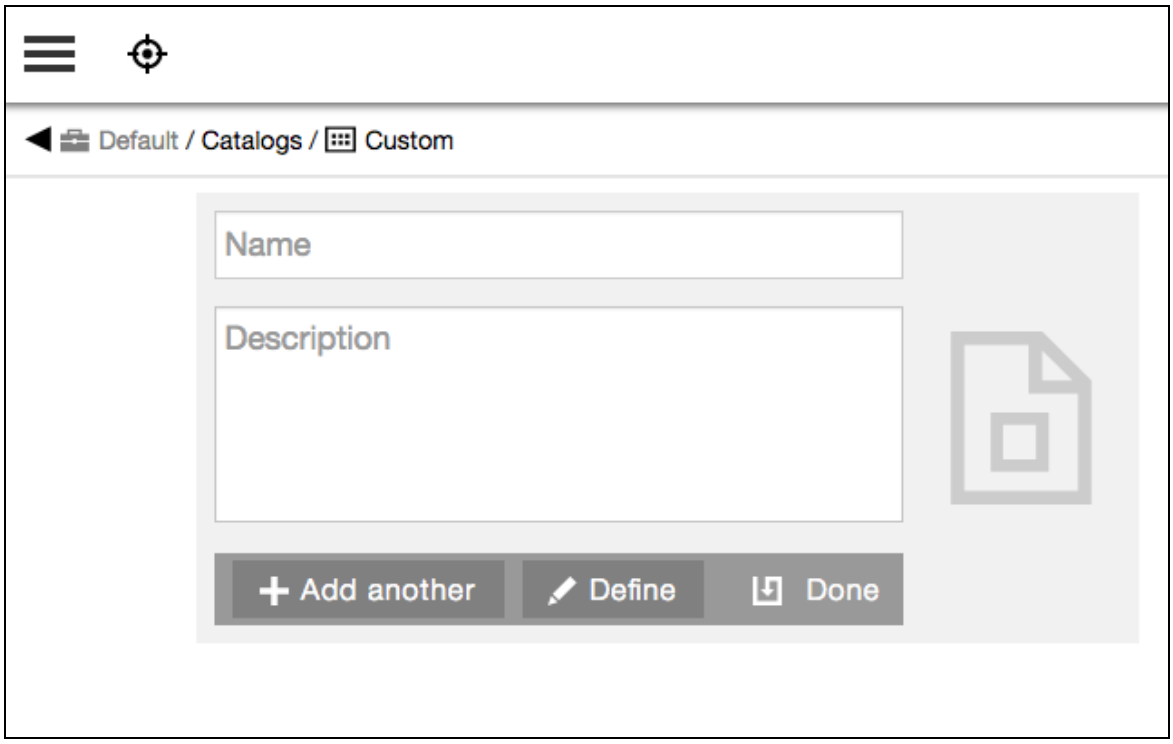
Creating Catalog Items

Select the catalog that you want to edit from the catalog drop-down menu. There are three options on this screen:



Select **Edit Catalog** to switch to the edit mode. After editing catalog items, you can use the **Viewing Catalog** mode to switch the UI back to the view mode to test the new items.

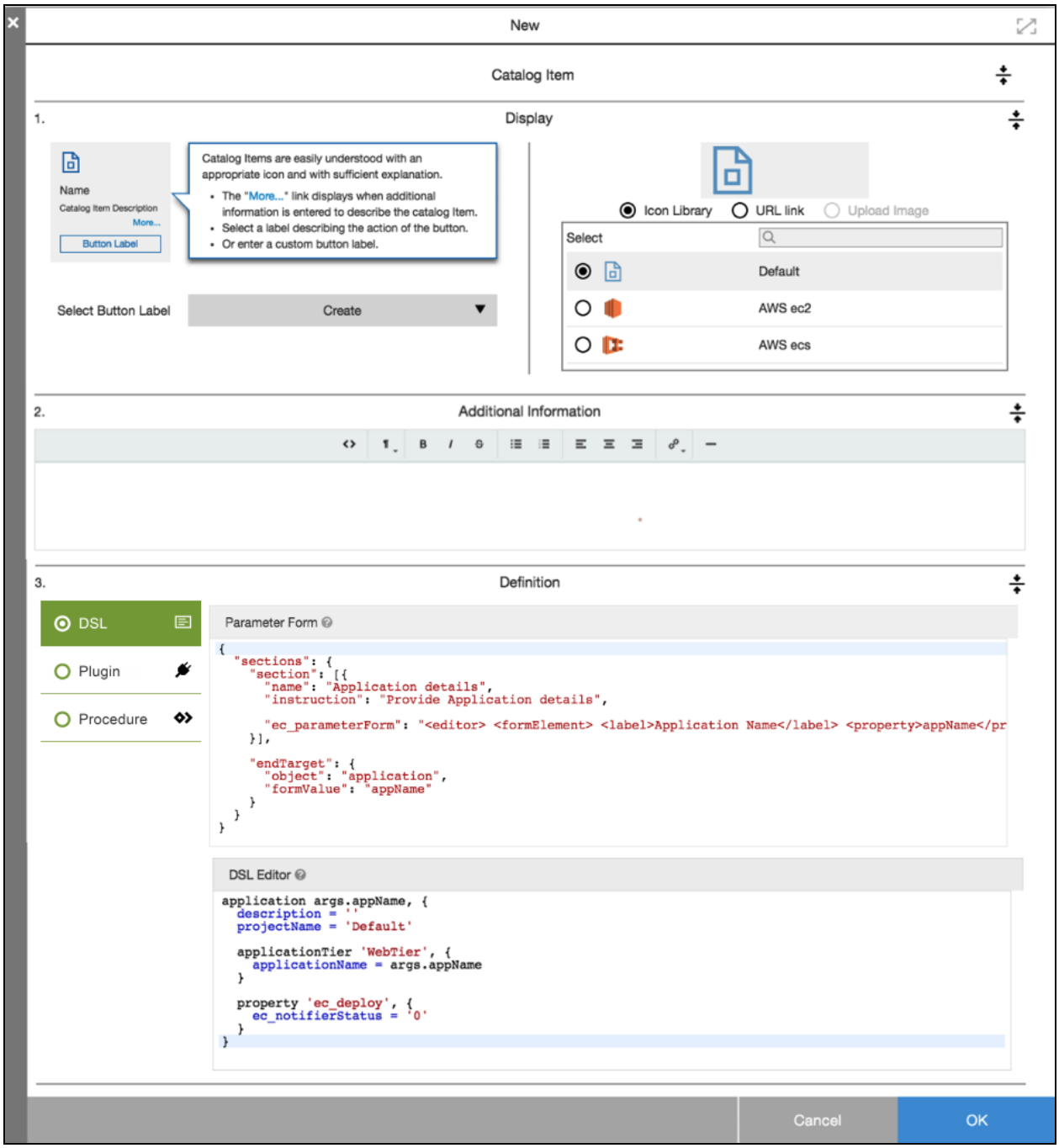
Enter a **Name** and an optional **Description** for the new item that you are creating. You can create another item or continue with defining the new item:



Note: Icon upload is not supported.

Click **Define** and enter the relevant information as follows. The key attributes that are relevant to a catalog item definition are:

- Name and description
- Icon that represents the catalog item
- Additional information that appears when the user clicks the **more** button. This gives the end user additional details regarding the usage of the catalog item
- Button Label: Label corresponding to the action button
- Parameter form: See Catalog Items on page 864 above for the details on the UI form metadata
- DSL Editor: DSL template corresponding to the catalog item



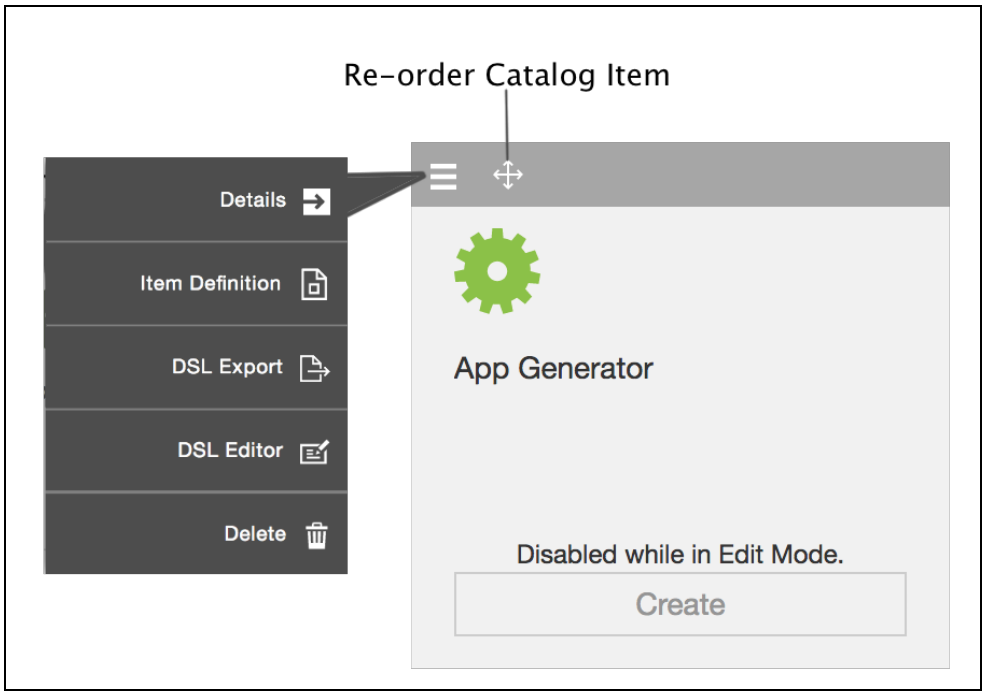
Creating Procedure-Based Catalog Items

You can create procedure-based catalog items. With a procedure-based catalog, you pick a project and a procedure to create the catalog item. See Procedure-Based Catalog Items on page 866 above for details.

At runtime, the system looks for the `ec_customEditorData/selfServiceUIForm`. If not found, then the UI is rendered based on the parameter definition on the procedure.

Editing Catalog Items

Once a catalog item is created, you can use the cross bar icon to reorder the catalog item. A menu associated with the catalog item provides additional options to edit item details or item definitions or to delete an existing catalog item. Options to export DSL or edit DSL are also available from the menu:



For More Information

See the "Self Service Catalog" video and the "The Self-Service Catalog Launching your DevOps Automation" video in the Electric Cloud YouTube channel.

Chapter 6: Security

ElectricFlow provides security by assigning roles and privileges to specific users and groups on:

- System objects including applications, microservices, environments, projects, jobs, and schedules.
- Actions performed on deployment models.

ElectricFlow uses access control, project-level security, and credentials and impersonation to enforce roles and privileges when executing deployment steps. For an example of how to define roles and privileges when a specific user is allowed to deploy an application or microservice only to a specific environment, see Use Case: Attaching Credentials in Deployment Automation on page 887.

Access Control

ElectricFlow uses access control to provide security for all system objects. This mechanism controls how users and groups use the system. Users must log in to view information or to perform operations (actions). After users log in, their system access is limited based on:

- The user name
- The groups to which the user belongs
- The permissions specified for various ElectricFlow objects

Go to Access Control for more information about how ElectricFlow enforces access control and for security examples using access control. For instructions on how to set up access control, go to these topics:

- Access Control on page 1089 GUI page
- Access Control—defining entries on page 1091
- Privileges—create new or edit existing privileges on page 1092

Project-Level Security

Multiple Project support is available on applications or microservices, pipelines, releases, environments, Master Components, resources, and environment templates as well as platform objects (such as artifacts, procedures, jobs, schedules, and workflows) These objects, as well as the objects belonging to them, can be in any project within ElectricFlow.

This significantly improves object management at scale by allowing:

- ACL inheritance—All objects in a project inherit the access control settings from the project, providing better security for all the objects. Objects such as applications or microservices, environments, pipelines, and releases can be managed in their own projects and will inherit the ACLs setup at the project level. This significantly simplifies permissions management.
- Logical grouping—This allows users to better manage deploy and release objects under various projects that are logically mapped by users, roles, geography, department, and so on, resulting in easier maintenance.

For an example of how to select a project for an application or microservice, see Examples: Modeling and Deploying Applications on page 255. You can also use API commands to do this:

- Use the `createApplication` API command to create a new application for a specific project.
- Use the `createService` API command to create a new microservice for a specific project.
- Use the `createProcess` command to create an application, microservice, or component process for a specific project.

For details about these commands, see the *ElectricFlow API Guide* at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html For details about authoring and deploying an application or microservice, see Examples: Modeling and Deploying Applications on page 255. For an example of how to select a project for a Release, see Release Definition on page 646. You can also use the `createRelease` API command to define a Release for a specific project.

Credentials and User Impersonation

What is a Credential?

In ElectricFlow, a "credential" is an object that stores a user name and password for later use. Two credential types are available, *stored* or *dynamic*:

- ***stored*** credentials – These credentials are given a name and stored in encrypted form in the database. Each Project has a list of stored credentials it owns. These credentials are managed from the Project Details page.
- ***dynamic*** credentials – These credentials are captured when a job is created. Dynamic credentials are stored on the server temporarily until the job completes and then discarded.

Encryption of Stored Credentials in ElectricFlow

128-bit AES (Advanced Encryption Standard) is used to store credential data in ElectricFlow, including passwords.

Defining a Credential

After a credential is created, no one can view the password for the credential's account. This means one person can define a credential and enter the password, and other people can use the credential (and its account) without needing to know the password.

To create a new credential in the automation platform UI:

1. Click the Projects tab.
2. Select a project (first column) to see the Project Details page.

3. Select the Credentials tab and then click **Create Credential**.

   Click the **Help** link on the New Credential page if you need more information about what to enter in the fields.

To create a new credential in the Deploy UI:

1. From the Projects list, click the Actions selector for the desired project and select **Details**. The Project edit dialog appears.
2. Click the Manage Credentials right arrow button. The **Credentials** dialog appears.
3. Click **Add** and the **Credential** dialog appears. Enter data into the fields.
4. Click **OK** to save the credential.

Why Use Credentials?

Use ElectricFlow credentials for:

- User impersonation
- Saving passwords for use inside steps

When a step needs to run as a particular user, ElectricFlow can retrieve the user name and password from a stored credential. The credential is passed to the agent over an encrypted channel so the agent can authenticate itself to the operating system and set up a security context where the step runs with the user permissions in the credential.

In some cases, a step needs to enter credentials directly to an application or microservice.

- The credential may be a fixed application or microservice credential that needs to be presented every time the step runs. In this case, a stored credential provides a place to put this information without needing to embed the password in a script.
- Other times, the password is not known in advance. In this case, a dynamic credential created when a job is launched can prompt a user for information and then pass the credential to the step in a secure manner.

Credential Access Control

Because ElectricFlow stores passwords in a recoverable manner, it is important that credentials are accessible only under carefully controlled circumstances. To protect credentials, ElectricFlow uses two mechanisms: access control lists (ACL) and *attached* credentials.

To use a credential for any purpose, a user who attempts to reference the credential must have execute permission on the credential object—this allows you to explicitly control which users are allowed to use a credential anywhere in the system.

In addition to the user-based access control check, ElectricFlow requires a credential be explicitly *attached* to the object that is going to use it. After an object has a credential attached, object modifications are restricted to users who have both execute permission on the credential and modify permission on that object. This safeguard helps prevent credentials from accidentally being used by unauthorized users.

Attach a credential to an object in the automation platform UI one of these ways:

- Set the impersonation credential for a step, procedure, project, or schedule.

  You can also set the impersonation credential for an application or microservice process, component process, or a process step.

  Impersonation credentials are inherited, so attaching a credential to a container object like a procedure or a project implicitly attaches it to every object inside the container, such as all steps in a procedure.

- Explicitly attach the credential to a step or schedule using the UI or the `attachCredential` API.

  To access credentials from a step, the credential must be attached directly to the step, or the credential must be passed as a parameter to the containing procedure and have the parameter attached instead.

Using Credentials for Impersonation at the Platform Level

This information addresses the question, "When a job step is running on a resource, under which account is it running and how can I control that?" By default, a job step runs under the account used by the ElectricFlow agent that runs the step, which was chosen when the agent was installed. This approach works well in environments where it makes sense to run all jobs under a single user such as a "build" user. You install all agents to run as the desired user, and you do not have to worry about anything else—every step of every job runs as that user.

However, in other environments you may prefer to run different jobs, or even different job steps, under different accounts.

For example:

- If independent groups are sharing an ElectricFlow system, you may want each group to use a different account for its jobs.
- You may need to run jobs as particular users to access ClearCase views for those users.
- There may be certain steps that require special privileges.

  For example, as part of your builds you may need to run a step that generates a certificate using privileged corporate information; that step must run under a special high-privileged account, but you want the remainder of the steps in the build to use a less-privileged account.

ElectricFlow allows you to select accounts on a per-job or per-job-step basis. This mechanism is called *impersonation*, where the ElectricFlow agent impersonates a particular user for the duration of a job step, and this *impersonation* is implemented using credentials. The use of credentials creates special security challenges.

> **Note:** It is important to ensure privileged accounts can be used only for the purposes you intend and cannot be "hijacked" for other purposes. The ElectricFlow access control mechanism contains special facilities to ensure proper credential use.

Setting the Impersonation Credential in the Automation Platform

Attaching Impersonation Credentials to Steps, Procedures, and Projects

You can attach impersonation credentials to procedure steps, procedures, and projects before executing a job step.

ElectricFlow searches for a credential in the following order and uses the first credential it finds:

1. Procedure step
2. Procedure to which the step belongs
3. Project to which the project belongs

If the step is in a nested procedure, and no credential is found on the step, its procedure, or its project, ElectricFlow checks the calling step (and its procedure and project), its caller, and so on until it has worked up through the topmost procedure in the job. If no credential is found, the job step runs under the ElectricFlow agent account on its resource.

This approach makes it easy to manage your account usage. For example, if you want all jobs in one project to use a particular account, define a credential in that project for the account, and then attach the credential to the project. If you want all jobs in a project to use a particular account, except for one step that should use a different account, create two credentials in the project. Attach the first credential to the project and attach the second credential to the particular step.

Attaching Impersonation Credentials to Schedules

You can also attach a credential to a schedule. The credential in the schedule is used for steps where no other credential is available.

The schedule's credential receives the lowest priority.

Attaching Impersonation Credentials to Jobs

You can specify a credential when you launch a job manually one of these ways:

- Provide the name of an existing credential
- Enter an account and password.

If you choose the second approach, the account and password are stored only in ElectricFlow for the duration of the job.

For either way, the credential you specified when you launched the job is used as a last resort for steps only with no other credential.

Attaching a New Impersonation Credential

To attach a new impersonation credential for a procedure, step, or schedule:

- Click the Projects tab.
- Select a project.
- Go to the Project Details page for the project you selected.

- Do one of the following:
  - For a **procedure**:
    - Click **Create Procedure** to go to the New Procedure page.
    - On this page, select the Impersonation Credential drop-down box to select a credential to use for impersonation.
  - For a **step**:
    - Select a procedure to go to the Procedure Details page.
    - Select a step name to go to the Edit Step page.
    - On the Edit Step page, scroll down to the Impersonation section.
  - For a **schedule**:
    - Select the Schedules subtab.
    - Select a schedule to go to the Edit Schedule page.
    - Scroll down to Impersonation Credential section.

  For more information in the platform GUI, each of these web pages contains a **Help** link in the upper-right corner.

Accessing Credentials From a Step at the Platform Level

The ElectricFlow server supports an operation called `getFullCredential` that can return a password when called from inside a step. The ectool interface for this operation is:

```
getFullCredential <credential name>

   [--value <password|user>]
```

Without the `--value` option, the response is the same as that returned by `getCredential`, with the addition of a "password" element.
With `--value`, the simple text password or user name value is returned on *stdout* so it can be used directly without XML parsing.

The `getFullCredential` API is allowed only inside a running step, and is allowed to use credentials that were explicitly attached to the step only.

Attaching a Credential to a Procedure Step, Procedure, or Schedule

A credential must be explicitly attached to the object using it so the server can perform an access control (ACL) check at definition time and limit the visibility of the password. To support accessing credentials other than the one being used for impersonation, steps, procedures, and schedules contain a list of credential names.

In the platform GUI, to attach a credential (other than an impersonation credential) to a step, procedure, or schedule:

- Go to an "Edit" page.

  For example, to create a credential for a step, a step must already exist.
- Select a step that needs a credential. The Edit Step page opens.
- Scroll to the lower section of the page and click **Attach Credential**.

If you are an ectool command-line user, use the `attachCredential` API command. For more information on the command-line interface, go to the *ectool* Help topics within the online Help system.

Passing Credentials as Parameters

Sometimes you may not know which credential you need to use when you define a procedure step. In this situation, you can leave the credential choice up to the caller of the procedure.

The following example describes the procedure for passing credentials as parameters:

1. Create a project named InnerProj.
2. Within that project create a procedure named InnerProc with a parameter 'cred' of the type credential.
3. Within that procedure create a step named InnerStep with:
    - Command: ectool getFullCredential cred
    - Attached Parameter Credential: cred
4. Create a project named OuterProj with two credentials: cred1 and cred2.
5. Within that project create a procedure named OuterProc with two parameters, cred1 and cred2, of the type credential.
6. Within that procedure create two subprocedure steps that both call InnerStep from the InnerProj project:
    - OuterStep1 with Attached Parameter Credential cred1, and cred1 in the Parameters section
    - OuterStep2 with Attached Parameter Credential cred2, and cred2 in the Parameters section

If you are an ectool command-line user, you can use the `createFormalParameter` and `attachParameter` API commands. For more information on the command-line interface, go to the *ectool* Help topics within the online Help system.

Credential References

For schedules, the value for any actual parameter passed to a credential-type formal parameter is interpreted as a reference to a previously attached credential on the schedule.

For procedure steps, the value for any actual parameter passed to a credential type formal parameter is interpreted as a reference to a previously attached credential or credential parameter on the calling step.

When a job step is created, all credential references are resolved and the content is copied into transient credentials on the job step. Credential parameter references are resolved by looking for a credential in the calling job step's transient credential map—looking for a credential with the name specified as the actual parameter value.

Credential references in the API can be specified as a relative or an absolute credential path.

- A *relative* path is similar to "rootCred" and is interpreted as referring to a credential in the current project.
- An *absolute* path includes a project name like "`/projects/MyProject/credentials/rootCred`" and can refer to a credential in any project.

Anywhere a credential reference is accepted (for example, setting an impersonation credential or attaching a credential to a step), either form can be used.

Best Practices for Retrieving Credentials

The ElectricFlow impersonation mechanism provides powerful mechanisms for using different accounts in job steps, and it can be used to handle a variety of challenges. However, if misused, impersonation can open up dangerous security loopholes. This section discusses these risks and how to avoid them.

Use of credentials and impersonation tend to fall into two classes.

First Class

In the first class, your goal is to open up access to accounts, you want to make some accounts generally available, and you trust large groups of users (such as everyone who can access a particular project) to use the accounts appropriately. This usage type is simplest and relatively safe.

1. Define an ElectricFlow group containing all trusted users.
2. Set permissions on the Project so only trusted users can access the project.
3. Define credentials for accounts in the Project and grant execute permission to everyone in the group.

After completing this task, everyone in the group can use the accounts for arbitrary purposes. No one outside the group will be able to access the project or the credentials.

Second Class

In the second class, you want to provide tightly-controlled access to a highly privileged function such as generating a certificate or accessing sensitive information. This kind of usage requires careful thought to avoid allowing unintended access to the credential. The best way to implement a solution for this scenario is to restrict access to the credential and use it on individual steps only after careful analysis of issues such as the following:

Does the step's command use properties?

If so, someone could potentially change the property value to change the step behavior.
For example, suppose the step's command is defined like this:

```
echo $[/myProject/x]
```

Anyone with write access to property *x* can hijack the step by placing the value "`nothing; myCommand`" in the property.
After property substitution, the final command will be:

```
echo nothing; myCommand
```

that will cause "myCommand" to execute in the step.

Either restrict access to the property sheet or avoid substitutions.

Does the step execute code or commands that can be modified?

In most jobs that execute build and test sequences, the job extracts various files from a source code control system and some of those files contain code executed during the build. In this case, anyone with access to the source code control system can affect the executed code.

For example, suppose a step runs a Make or Ant command using a configuration file extracted from the source code control system. Anyone with access to the source code control system can modify the configuration file to introduce new commands executed during the Make or Ant step. In this situation, it is virtually impossible to control what happens during the step, so you should never attach a credential for a privileged account to such a step.

The higher you attach a credential, the greater the risk of uncontrolled access to the credential's account.

For example, if you attach a credential to a project, there is a good chance anyone with write access to the project, or even the ability to execute the project's procedures, can hijack the credential using one of the loopholes described above. If you attach a credential to a step that invokes a subprocedure, you effectively give anyone with write access to the subprocedure complete access to the credential's account. If you care about access to an account, you should use its credential in the narrowest possible fashion, such as attaching it to a single job step.

Avoid passing passwords on the command line

When using credentials, avoid passing the password on the command line because [on most platforms] those passwords will be visible to all users logged into the system. The secure way to pass a credential to an application or microservice is to use a secured file in the file system or to pipe the value into the application or microservice via `stdin`.

Credentials and Impersonation in Application, Microservice, and Component Processes

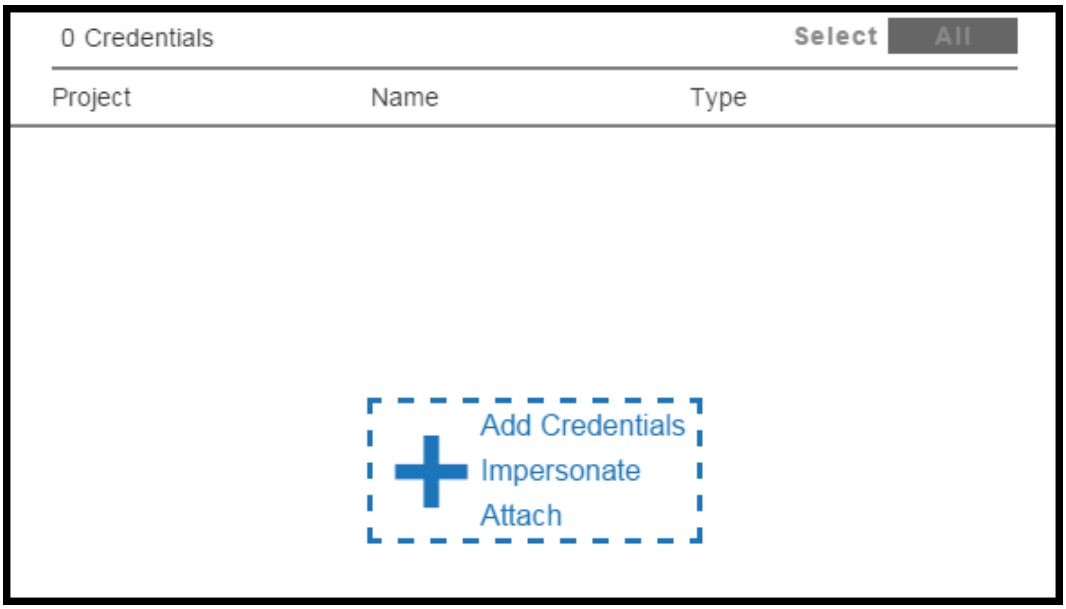
You apply credentials and impersonation to control who can deploy applications or microservices and where the applications or microservices are deployed.

- You can attach one or more credentials to component, application, or microservice process steps.
- You can use only one impersonation credential when running an application or microservice process, component process, or a process step.
- When you attach an impersonation credential in ElectricFlow, it specifies the user who can deploy the application or microservice and the environment in which the application or microservice is deployed.
- When you attach an impersonation credential to a platform-level object (such as a procedure), it specifies the account (user) that can run the job or job step. If you want to specify another condition, you have to attach another credential to the object.

Important: When you impersonate using a credential, make sure that the impersonated user has the absolute path to the `bin` directories in the `$PATH` environment. If you define a process step with a command, you must enter the absolute path in the **Post Processor** and **Shell** fields in the **Define Step** dialog box.

1. Click in the **Add Credentials** field.

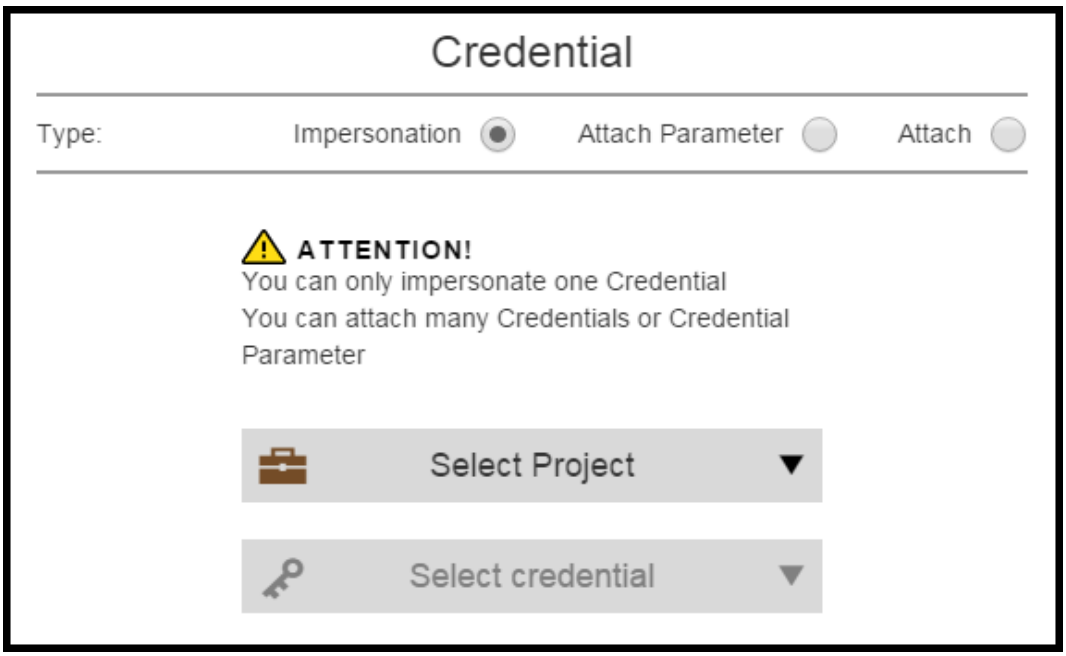
Example:



The **Component Process Step / Add** dialog box opens.

2. To impersonate one credential:

Example:

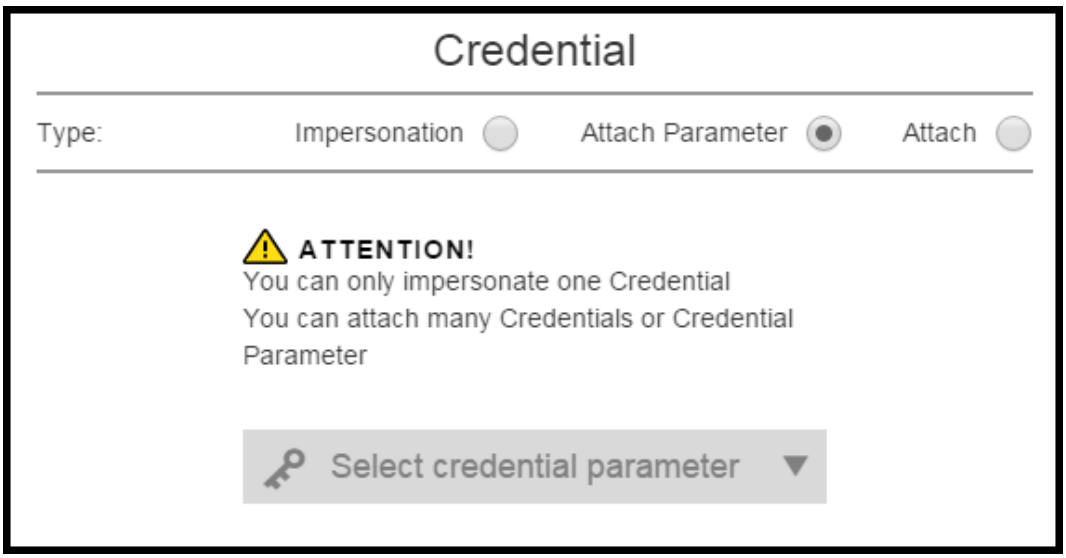


1. Select **Impersonate** in the **Type** field.
2. Click the **Select Project** field to select a project. You can select a credential from a project that is different than the one containing the application.
3. Click the **Select Credential** field to open a drop-down list of credentials for the process step.
4. Select a credential.
5. Click **OK**.

   The **Credentials** dialog box now shows the one credential for impersonation.

3. To attach one or more credential parameters to the process step:

Example:

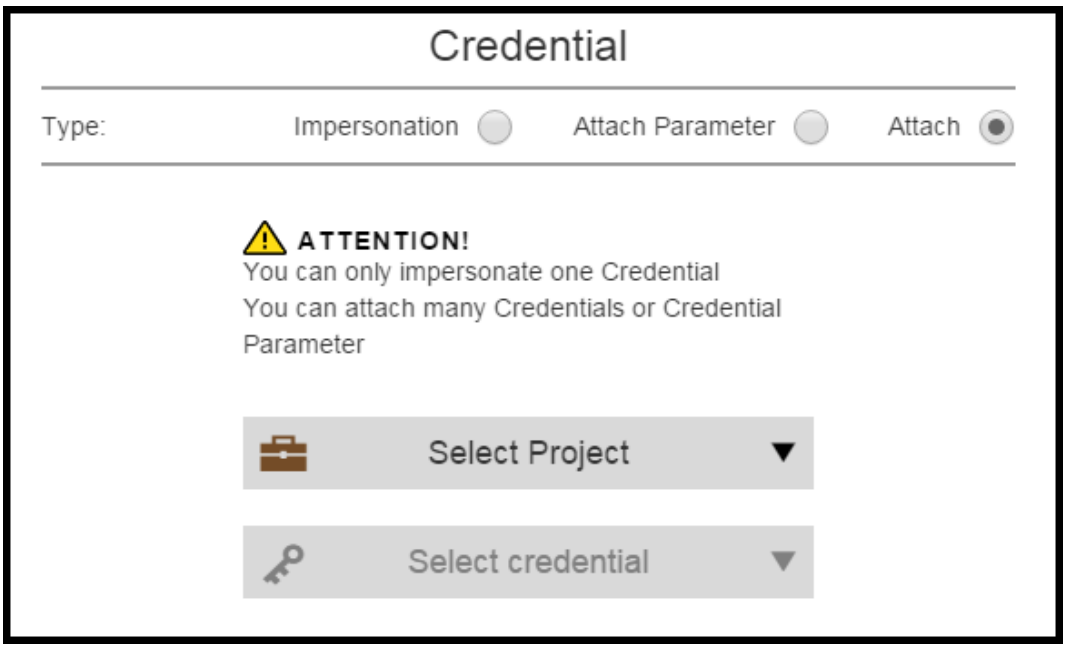


1. Select **Attach Parameter** in the **Type** field.
2. Click the **Select Credential Parameter** field to open a drop-down list of credentials for the process step.
3. Select a credential.
4. Click **OK**.

   The **Credentials** dialog box now shows the attached credentials

4. To attach one or more credentials to the process step:

Example:



1. Select **Attach** in the **Type** field.
2. Click the **Select Project** field to select a project. You can select a credential from a project that is different than the one containing the application.
3. Click the **Select Credential** field to open a drop-down list of credentials for the process step.
4. Select a credential.
5. Click **OK**.

   The **Credentials** dialog box now shows the attached credentials.

Use Case: Attaching Credentials in Deployment Automation

When modeling an application or microservice, you can attach credentials these ways:

- For credentials, attach one or more credentials to component, application, or microservice process steps.
- For credentials for impersonation, attach only one impersonation credential to these objects:
  - Component process
  - Component process step
  - application or microservice process
  - Application or microservice process step

When you deploy the application or microservice, ElectricFlow applies the credentials based on the user permissions and deploys the application or microservice in one or more environments.

Example

This example describes how to define roles and privileges when an application or microservice is deployed to more than one environment and specific users are limited to specific environments.

You can attach impersonation credentials to an application or microservice in the GUI. The application or microservice has these credentials:

- Development (dev)
- Quality Engineering (qe)
- Production (prod)

Users have these privileges:

- User A is allowed to deploy the application or microservice to build a MyQSQL database in any environment and has admin privileges.
- User B is allowed to only deploy the application or microservice in the quality (qa) environments and is not trusted in the development (dev) and production(prod) environments.

The following user permissions determine what users are allowed to do in ElectricFlow.You can configure these credentials only from the GUI.

- You configure User A's profile in the automation platform and give User A higher-order privileges than other users. User A has the following credentials, including a credential for impersonation:

  For each environment in ElectricFlow, set a property using a reference such as `$[myEnvironment/dbConfigName]` and define a unique value, which can be passed as a credential to a process or process step.

  - In the development (dev) environment, set `dbConfigName = dbUser_dev`.
  - In the quality engineering (qe) environment, set `dbConfigName = dbUser_qa`.
  - In the production (prod) environment, set `dbConfigName = dbUser_prod`.
- When you configure User B's profile in the automation platform, User B is only given the credentials to deploy in the quality (qa) environment. You do not need to set properties to be passed as credentials when the application or microservice is deployed.

Attaching the same credentials from the automation platform is more complicated. Instead of setting only one credential for User A and one for User B, you need to create three unique credentials for the environments in addition to credentials for the various user and environment combinations, such as User A and the development environment.

Credentials in Pipelines

Credentials are passed to pipelines through applications or microservices in pipeline stages. If an application or microservice has required inputs (parameters) and any of the inputs is a credential parameter, you can enter the path to the credential, browse to it, select a credential parameter (**Parameter Credential**) or a credential binding to a pipeline task (**Credential binding**), or select a user-defined credential that is attached to the project associated with the pipeline.

A credential must be explicitly attached to the object using it so the server can perform an access control list (ACL) check at the definition time and limit the visibility of the password. To support accessing credentials at the pipeline task level, these tasks need to have the appropriate credentials attached. This is done implicitly by the UI when you are binding credentials to the pipeline tasks. If you are using ectool, use the `attachCredential` API command to perform the same operation.

Credentials in Releases

In the Release definition, credentials appear in these objects:

- Pipelines: How the credential is applied to the Release depends on the selected pipeline.

  If the pipeline has required inputs (parameters) and any of the inputs is a credential parameter, you can enter the path to the credential, browse to it, select a credential parameter (**Parameter Credential**) or a credential binding to a pipeline task (**Credential binding**), or select a user-defined credential that is attached to the project associated with the pipeline.

- Applications or microservices: How the credential is applied to the Release depends on the applications or microservices in the selected pipeline.

  If any input to an application or microservice is a credential parameter, you can enter the path to the credential, browse to it, select a credential parameter (**Parameter Credential**) or a credential binding to a pipeline task (**Credential binding**), or select a user-defined credential that is attached to the project associated with the pipeline.

See the steps to select a pipeline and to select credential inputs for an application or microservice in the pipeline in the Release Definition on page 646for examples of these operations.

Cross-Site Request Forgery Protection

CSRF protection is disabled by default. You can enable or disable CSRF protection as follows:

- To enable CSRF protection, enter `ecconfigure --webCsrfProtection=true`.

  Examples using the default directory:

  - For Linux, enter

    ```
    /opt/electriccloud/electriccommander/bin/ecconfigure --
    webCsrfProtection=true
    ```

  - For Windows, enter

    ```
    C:\Program Files\Electric Cloud\ElectricCommander\bin>ecconfigure --
    webCsrfProtection=true
    ```

- To disable CSRF protection, enter `ecconfigure --webCsrfProtection=false`.

  Examples using the default directory:

  - For Linux, enter

    ```
    /opt/electriccloud/electriccommander/bin/ecconfigure --
    webCsrfProtection=false
    ```

  - For Windows, enter

    ```
    C:\Program Files\Electric Cloud\ElectricCommander\bin>ecconfigure --
    webCsrfProtection=false
    ```

Chapter 7: Change Tracking

Change Tracking is designed to track every change between every state of non-runtime ElectricFlow objects and to allow you to revert to any previous state of these objects. ElectricFlow tracks the changes to tracked objects including applications or microservices, procedures, workflows, workspaces, resources, and project-owned components such as library components and records a *Change History* of the historical states of the system and the changes between them. The change history has a copy of every state in which every non-runtime object has been, the object's current state, and indexing records for searching through the database. The tracked objects are not affected when ElectricFlow executes an object that is usually created or modified at execution time, such as a workflow or process.

Change tracking allows you to do the following:

- When debugging a failed job or looking for more information about a component in an ElectricFlow, see the Change History for the changes relevant to that object.
- When searching for specific change history records, filter the records by time frame, change type, entity type, or developer.
- Revert changes to an object or to an objects and its children.
- When you want to determine the differences between objects, export them at various levels in the object hierarchy.

In ElectricFlow, you can use Change Tracking in all aspects of the ElectricFlow end-to-end solution:

- In build-test automation, track the Change History of artifacts, resources, and ACLs.
- In deployment automation, track the Change History of components, environment tiers, and process steps (application, microservice, or component). You also use Change Tracking with snapshots to make it easier to deploy reliable and repeatable software for Continuous Delivery.
- In releases and Pipelines, track the Change History of stages and tasks.

Performance Consequences of Change Tracking

Change Tracking affects how ElectricFlow runs. The impact can be from low (minimal changes to the times to update the database) to high (significantly longer times to update the database).

ElectricFlow needs to store additional information in the database about the tracked objects in the Change History. In addition to storing normal information about ElectricFlow operations, ElectricFlow stores a separate archival copy of every state of every tracked object as well as the indexing information used to retrieve the Change History of tracked objects and the objects they own. This increase in database usage has performance consequences.

For most of ElectricFlow operations, the performance consequences of Change Tracking are insignificant, typically a few percent slower in one of these situations:

- None of the affected objects are tracked (job records and similar runtime-only objects are not tracked).
- Only a small number of tracked objects are changed by operations.

However, for operations that affect a very large number of tracked objects in a single operation (such as importing, cloning, or deleting a large project for which change tracking is enabled), the performance consequences *will be* significant. The performance consequences are similar when enabling or disabling Change Tracking for a large project, because both operations require the server to write to the database the records for every tracked object owned by the project.

For these Change Tracking-intensive operations, the system performance will typically be significantly slower because several times as much data is being read and written between the ElectricFlow server and the database. In our testing, we have seen a wide range of degrees of performance decrease from 1.25x to 10x slower when Change Tracking is enabled on a large project, compared to a project on which Change Tracking is disabled. The decrease in performance is typically from 2x to 6x.

Use Case 1: Importing and Cloning Large Projects with Change Tracking Enabled

ElectricFlow does the following to increase the performance of Change Tracking-intensive operations in a large project with this configuration:

- The project contains over 20,000 audited objects.
- Change Tracking is enabled.

To increase the performance when a large project is imported or cloned, ElectricFlow automatically decreases the amount of Change History indexing information that it saves, reducing the level of detail for these operations in the Change History. While this can improve the system performance, it can also make it harder when you want to revert an object to a specific state or to find information in the Change History while troubleshooting or debugging an issue. If you want suppress this behavior, use `--reducedDetailChangeHistory false` in the `import` or `clone` API command.

By default, the objects listed in the table below and the tracked objects that they own are always tracked when Change Tracking is enabled (except when Change Tracking is globally disabled).

During an upgrade from ElectricFlow 5.0.x to ElectricFlow 6.0, when the ElectricFlow server starts, the server writes to the database a record for every tracked object and the objects that it owns. The objects listed in the table get new audit records. If any of the objects owns many tracked objects (such as properties), the startup process will be slow.

Objects (Entities)	Path
server	/server
system objects	/systemObjects/*
users	/users/*
groups	/group/*
plugin	/plugins/*
email configuration	/emailConfigs/*
workspaces	/workspaces/*
zones	/zones/*
gateways	/gateways/*
agents	/agents/*

Objects (Entities)	Path
resources	/resources/*
resource pools	/resourcesPools/*
repositories	/repositories/*
directories	/directories/*

Use Case 2: Change Tracking of Non-Project-Owned Objects

In this context, *non-project-owned objects* include all objects not contained within a project. Such objects include resources, artifacts, users, groups, zones, and other objects as well as the property sheets, properties, access control lists (ACLs), and ACL entries (ACEs) for them. In releases earlier than ElectricFlow 6.2, Change Tracking was enabled by default for non-project-owned objects, and there was no way to disable Change Tracking for them. When users upgraded from earlier versions of ElectricFlow to ElectricFlow 6.0.x, the upgrade process initialized and created Change Tracking-related data for them in the database. The performance impact was usually low, except when there was a very large number (in the several hundred thousands) of non-project-owned objects, such as many properties under the `/server` directory; in this situation, the performance of the first startup was significantly slower. The best solution to this situation is to relocate most of the projects under a project and disable Change Tracking for that project, which may not be a practical short-term solution.

Starting in ElectricFlow 6.2, Change Tracking can be disabled for all non-project-owned objects. (Change Tracking for these objects is enabled by default.) Using this feature is not recommended as a long-term solution; instead we recommend moving most of the objects under a project that has Change Tracking disabled.(Repeatedly disabling and enabling Change Tracking for non-project-owned project is expensive and time-consuming.)

In ElectricFlow 6.2, when there are many non-project-owned objects, the best way to disable Change Tracking for all non-project-owned objects is to change the `database.properties`configuration file to disable Change Tracking for these objects before starting the server running a pre-ElectricFlow 6.2 release. In the `database.properties`file, make sure that it has this line:

```
COMMANDER_DB_NON_PROJECT_AUDITING_ENABLED=false
```

Then install ElectricFlow 6.2 and start the server, running the system this way until most of the non-project-owned objects are relocated to a non-tracked project, and re-enable Change Tracking for non-project-owned objects, which requires restarting the server.

Estimating Database Growth

Here is one way to estimate the rate at which the database grows for Change Tracking:

Most of the information needed to estimate the database growth rate can also be parsed from an exported XML file of the project (such as a full import), except for the version count.

The likely rate of database growth is based on these factors:

- How many objects of different types will be tracked
- How big they are
- How often their state changes

The rate of database consumption is then the sum over all different types of tracked objects of the product of these three factors.

Common Usage Patterns

For most customer usage patterns, the most common objects (the ones that dominate on first factor) are ACLs and their ACEs, properties, or all of these objects. Typically:

- ACLs and ACEs: ACLs are consistently small, and usually have only a few ACEs.
- Properties: While most of them are small, some properties can be very large. The average property size is large.

For most users, a good quick estimate can be based on properties, especially properties that are large or are modified often.

Estimating Process

If you expect to enable Change Tracking for all projects, determine the values for these factors and calculate the database growth rate:

Database growth rate = Factor 1 x Factor 2 x Factor 3

- Factor 1: How many objects of different types will be tracked

  The following procedure is only for properties:

  1. Join the ec_property table in the database joined to the ec_property_sheet table by ec_property.parent_sheet_id = ec_property_sheet.id.
  2. Search for properties where the ec_property_sheet.entity_type value is one of the entity types corresponding to a database table that has a matching ec_*_aud table.
  3. Determine the number of tracked objects per property type.

     For most usage patterns, the great majority of properties will belong to entity types like job and jobStep that are not change tracked and so have no corresponding ec_*_aud table.

  If you are already using ElectricFlow 5.3 or later, these properties and properties belonging to projects for which Change Tracking is disabled are also recognizable by having `ec_property_tracked` set to `false`.

- Factor 2: How big they are (size)

  The following procedure is only for properties.

  For small properties:

  1. Search for objects that are less than 450 bytes in the `ec_property.string` column.
  2. Add an overhead value of half a kilobyte per property for the other columns.
  3. Determine the average size of the tracked objects.

  For larger properties:

  1. Search for the length of the value in the `ec_clob` table linked to by the `ec_property.clob` column.
  2. Add an overhead value of one kilobyte per property for the other columns.
  3. Determine the average size of the tracked objects.

- Factor 3: How often new copies of the objects are generated

  The following procedure is only for properties:

  1. Use one of these methods:
     - Determine ratio of how long ago the ec_property.created and ec_property.modified datetimes are , which gives an estimate concentrating on only the most recent changes.

       and/or

     - Determine the value of the ec_property.version counter divided how long ago the ec_property.created datetime is, which gives a long-term average rate of changes to the property since it was created.
  2. Determine the average update frequency.

For properties that are frequently updated always to be a numerical value, it is possible to suppress Change Tracking of numerical-value updates by using one of these commands:
ectool: `ectool modifyProperty <projectName> -- path <propertyPath> -- counter true`

or

ec-perl: `$cmdr-> modifyProperty (<projectName>, {path => <propertyPath>, counter => true});`

When the `counter` flag is set for a property, Change Tracking does not track changes to the property if the only change was a change in the property value from one numeric value to another. All other forms of changes to the properties that have this flag set are tracked normally.

Important: Reverting an object that owns a counter property to a previous state will revert the value of the counter property to its value at the previous time that a change to this property was tracked–typically this is when its value is initialized. This may not be the desired behavior, so you may need to manually set a counter property if it is reverted.

Alternative Estimating Methods

Depending on your usage patterns, one of these methods may result in a more accurate estimate than the previous method of determining the average factor values and multiplying them

Split the set of properties of change tracked objects into different populations

For each object, perform the previous method of determining the average factor values and multiplying them per tracked object.

Add the values for each tracked objects together.

Example:

- If you have one set of properties (such as those in one project, or belonging to one user) containing very large properties that are never changed, and another set containing very small properties that change often, multiplying the average property size by the average rate of property update would result in an overestimate of the total database consumption.
- If you put the properties in separate projects and calculate the database growth rate for each project, the database growth rates are more accurate.

Other columns that may be of interest include ec_property.owner (who created the property) and ec_property.last_modified_by usernames (who last modified the property).

Best Practices for Change Tracking

Change Tracking allows you to troubleshoot and debug issues, find information about specific objects, and revert an object to a specific state. However, Change Tracking affects the system performance. In some situations, the performance consequences of Change Tracking can be significant.

This section describes how to improve the system performance.

When to Enable or Disable Change Tracking for a Project

In some situations, enabling or disabling Change Tracking on a project causes significant performance consequences. The ElectricFlow system performance becomes slower as the server writes to the database the records for all the tracked objects in the project.

To determine when to use Change Tracking on a project, do the following:

1. Export the project using this command:

   ```
   ectool export <project_name> --relocateable 0 --withAcls 1 – withNotifiers 1
   ```

2. Evaluate the results.
    - The size of the results file gives you a good estimate of how big the project is:
        - If the file size is in the hundreds of MB, the project is large.
        - If the file size is in the tens of MB or less, the project is small.
    - (Optional) Write a script to parse the results and determine these statistics:
        - Total number of objects that can be tracked in the project.

          Use this value to determine the project size (for example, small, medium, or large), which is used to determine how long it would take to enable, disable, copy, import, or delete a project on which Change Tracking is enabled.
        - The total size (in XML format) of the entities in the project that can be tracked by Change Tracking.

          This gives a rough estimate of how much the database usage will immediately grow when Change Tracking is enabled for the project.
        - The rate at which tracked entities are modified in the project.

          Look at the most recent last-modification dates of entities owned by the project for a pattern where certain entities are modified on a regular basis.

          If there are objects that appear to be being modified frequently, the script can also parse out their path and the user name they were last modified by.

Next Steps:

- If the project is small, and its contents are not frequently changed by an automated process, you can safely enable Change Tracking for it.
- If the project is large, and if you frequently export, import, or copy it, and if the performance of these operations is very important to you, enabling Change Tracking for it may be problematic (see Estimating Process on page 894 for a possible workaround).
- If significant parts of the contents of the project are frequently updated by an automated process, enabling Change Tracking for the project will consume significantly more database space, and tracking the changes performed by the automated process is unlikely to be of much value, so enabling Change Tracking for the project is inadvisable.

Splitting Tracked Objects Into Separate Projects

If you export the project, evaluate the results, and determine that enabling or disabling Change Tracking on the project will significantly affect the performance of the ElectricFlow system, one way to improve the system performance is to split the project into separate projects based how the objects are modified.

For example, if you parse the results and find that there are two types of entities in your current project:

- Objects that are manually modified over time by real users.

  These objects represent real changes in project and are modified as procedures, jobs, and applications or microservices are run.

- Objects that are frequently updated at regular intervals by an automated process using metaprogramming.

  These objects are automatically generated and updated on a regular basis. The project file should have a large list of modifications to tracked objects.

Then you should split the objects into separate projects.

- Put the objects that are manually modified by real users in one project and enable Change Tracking on this project.
- Put the objects that are updated by an automated process in a different project and disable Change Tracking on it.

Configuring Change Tracking

Change Tracking must be enabled when ElectricFlow starts for your system to track changes and record the Change History.

By default, Change Tracking is enabled for projects created in ElectricFlow 5.3 or later. It is disabled for projects created in ElectricFlow versions before ElectricFlow 5.3, but can be enabled manually.

Enabling Change Tracking Globally

When installing ElectricFlow:

1. Add this line to the `database.properties` file:

   ```
   COMMANDER_DB_AUDITING_ENABLED=true
   ```

2. Restart the ElectricFlow server.

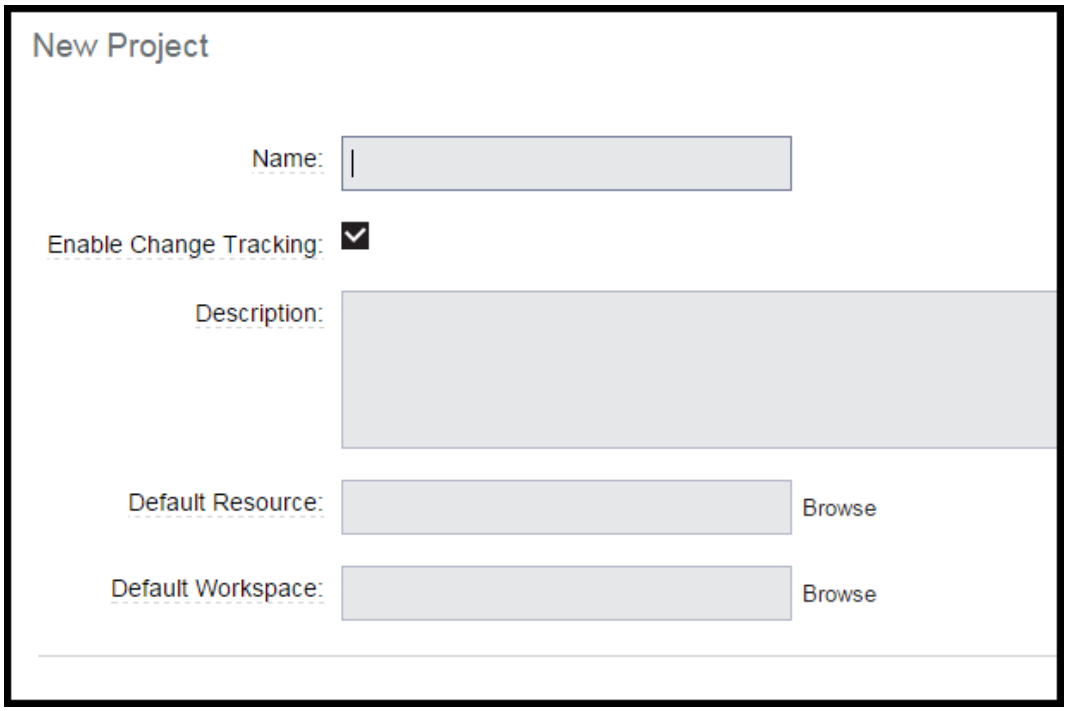
Enabling Change Tracking on a Per-Project Basis

> **Important:** Unless the `allowEnablingDisablingChangeTrackingProjectLevel` setting has been altered, only admin users are allowed to perform this task. The default setting is `adminOnly`. If the setting is changed to `anyoneWithAccess`, any user with access to the project can enable or disable Change Tracking for the project

You can enable Change Tracking one of the following ways:

In the ElectricFlow Platform UI

Change Tracking is enabled when the **Enable Change Tracking** check box is selected.



To disable Change Tracking, click the **Enable Change Tracking** check box to clear it. and click **OK**.

Using ectool

- Enter `ectool modifyProject <projectName> --tracked true` to enable Change Tracking.
- Enter `ectool modifyProject <projectName> --tracked false` to disable Change Tracking.

Using ec-perl:

- Enter `$cmdr->modifyProject(<projectName>, {tracked => true});` to enable Change Tracking.
- Enter `$cmdr->modifyProject(<projectName>, {tracked => false});` to disable Change Tracking.

For properties that are frequently updated always to be a numerical value, it is possible to suppress Change Tracking of numerical-value updates by using one of these commands:
ectool: `ectool modifyProperty <projectName> -- path <propertyPath> -- counter true`

or

ec-perl: `$cmdr-> modifyProperty (<projectName>, {path => <propertyPath>, counter => true});`

When the `counter` flag is set for a property, Change Tracking does not track changes to the property if the only change was a change in the property value from one numeric value to another. All other forms of changes to the properties that have this flag set are tracked normally.

Important: Reverting an object that owns a counter property to a previous state will revert the value of the counter property to its value at the previous time that a change to this property was

racked–typically this is when its value is initialized. This may not be the desired behavior, so you ay need to manually set a counter property if it is reverted.

Upgrading to ElectricFlow 6.x

Change Tracking is enabled when you upgrade to ElectricFlow 6.x. This can significantly increase the time it takes to complete the upgrade.

If you want to upgrade with Change Tracking disabled, add this line to the `database.properties` file before starting the upgrade:

```
COMMANDER_DB_AUDITING_ENABLED=false
```

Customizing the Change History Page

The performance of the Change Tracking feature is affected by number of records in the Change History as well as the number of entries being tracked. For example, a page showing 5000 entries may be slow to load and update and does not provide much useful information.

This ElectricFlow server uses the lowest of the following limits to determine the maximum amount of records to display in the Change History page:

- Maximum amount of records on the Change History page

  To change the maximum number of records in the Change History page:

  1. Set the `CHANGE_TRACKING_HARD_MAX_RECORDS` parameter in the `wrapper.conf` file to a new value.

     The default value is 1000.

  2. Restart the ElectricFlow server.

- Maximum number of records retreived

  Set the `TrackingMax Records` server setting to a new value not exceeding the `CHANGE_TRACKING_HARD_MAX_RECORDS` parameter in the `wrapper.conf` file.

  To set `TrackingMaxRecords`, do one of the following:

  Change the value in the UI. See the Server Settings page in the automation platform UI.

  Use ectool to change the value. For example, enter the following command to limit the number of records retrieved to 100:

  ```
  ectool setProperty /server/settings/changeTrackingMaxRecords --value 100
  ```

Searching the Change History

Follow these steps to start a search in the Change History.

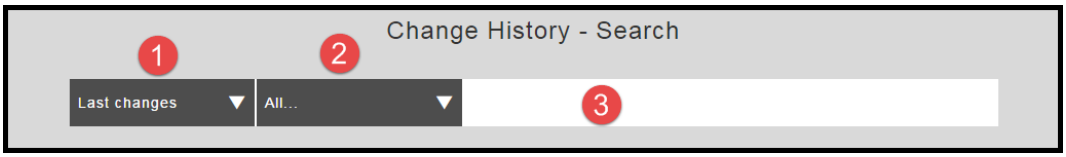
You can start a Change History search from most pages in the ElectricFlow UI.

- Click the **Search** button or click **Change History**.

Example:

The **Change History—Search** dialog box opens.

Example:

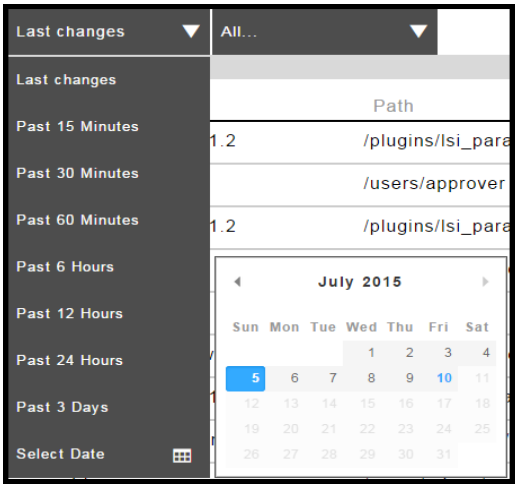


1	**Time range** field. Click the down arrow to open the drop-down list of start times. The end time is the current time.
2	**Objects** field. Click the down arrow to open the drop-down list of objects to include in the search. You can select **All** or specific objects. By default, seven of the most commonly tracked objects are selected.
3	Search criteria. After you type, the system starts searching for objects based on the time range and objects that you selected. The search results are in the Change History.

- To select a time range for the search:
  - Click the down arrow in the **Time range** field to open the drop-down list.
  - Select a time range.
  - If you want to use a time increment longer than three days, do the following:
    1. Click **Select Date**.

       The Date Picker opens.

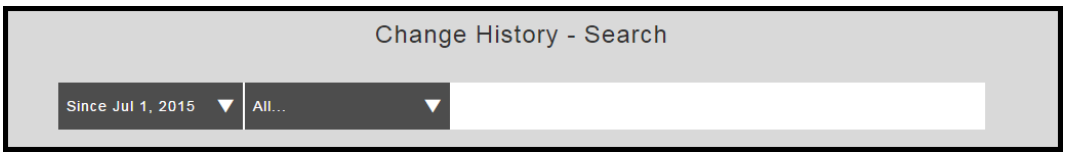
       Example:



    2. Select a date.

       The Date Picker closes and the date that you selected appears in the Time Increment field.

       Example:



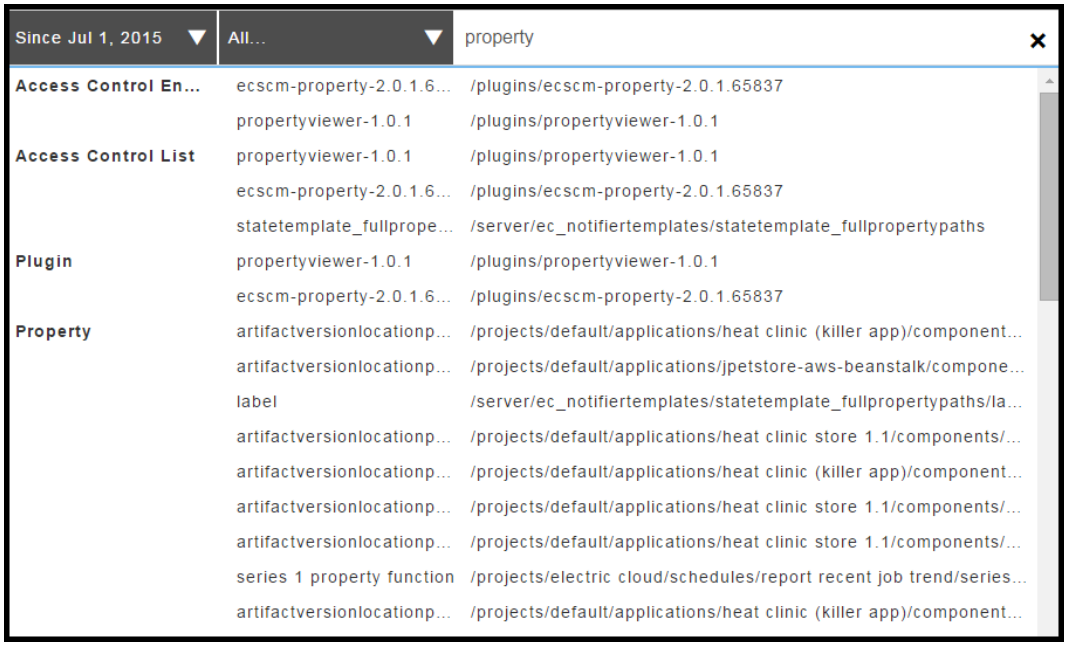
- To select an objects for the search:
  - Click the down arrow in the **Object** field to open the drop-down list.
  - Select the objects for the search.
- Enter the search criteria.

  As you type, the system starts to search for objects that match your search criteria.

  A list of objects matching your search criteria appears in the results section.

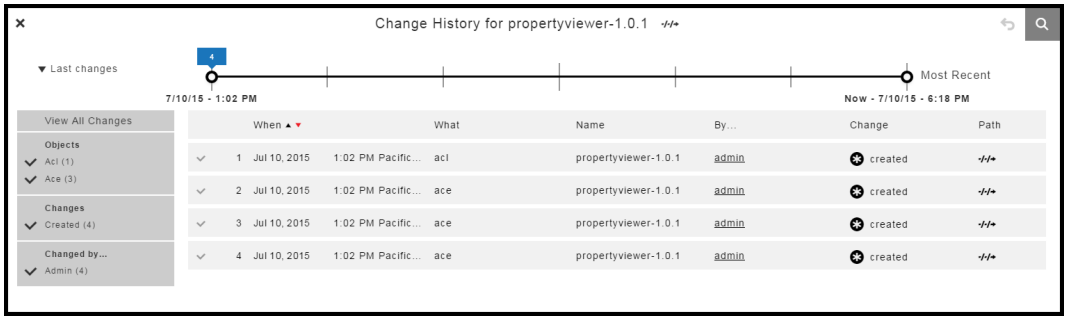
- Select an object in the list.

Example:



The change history for the object that you selected appears.

Example:



Viewing the Change History

You can open and view the change history from the following objects in ElectricFlow:

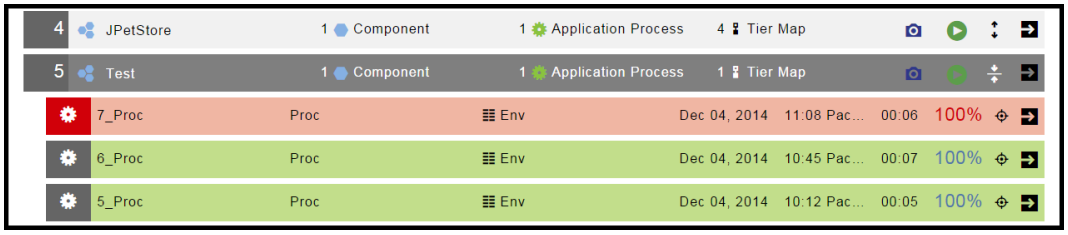
- Applications List and Microservices List
- Application Editor and Microservice Editor
- Application Process Designer and Microservices Process Designer
- Artifacts
- Components

- Component Process Designer
- Environments Designer
- Environment Tier
- Jobs
- Process Steps (Application, Microservice, and Component)
- Projects
- Resources
- Workflows

Viewing the Change History from the Applications List

You may want to view the Change History for these objects in an application:

- An application process that did not run successfully



- Component or component process in an application process that did not run successfully

- Resource that was not deployed successfully

1. Open the home page of the ElectricFlow web UI by browsing to https://<*ElectricFlow_server*>/flow/.
2. Go to the applications list in one of these ways:
   - Click **Applications**.
   - Click the **Main menu** button in the upper left corner and then select **Applications**.
3. Choose an application.

4. Click the **View** button.

   Example:

   A list of the application processes for the application appear.

5. To view the Change History of an application process:
   1. Choose a process.
   2. Click the **Change History** button.

      Example:

      The Change History for the application process opens.

6. To view the change history for an object in the application process,
   - Click the **View Details** button.

     A list of objects in the application process (components and component processes) appears, and the breadcrumb changes to *Applications/View Run*.
   - Choose an object.
   - Click the **Change History** button to see the change history for the object.

Viewing the Change History From an Application, Microservice, or Environment

You may want to view the Change History for these objects:

- Applications
- Independent microservices
- Application processes
- Component processes
- Environments
- Resources

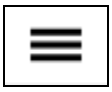
Application Editor

Viewing the Change History for an Application

1. Go to the Applications List.
2. Select an application.

   The Application Editor opens.
3. Click the **Menu** button.

   Example:

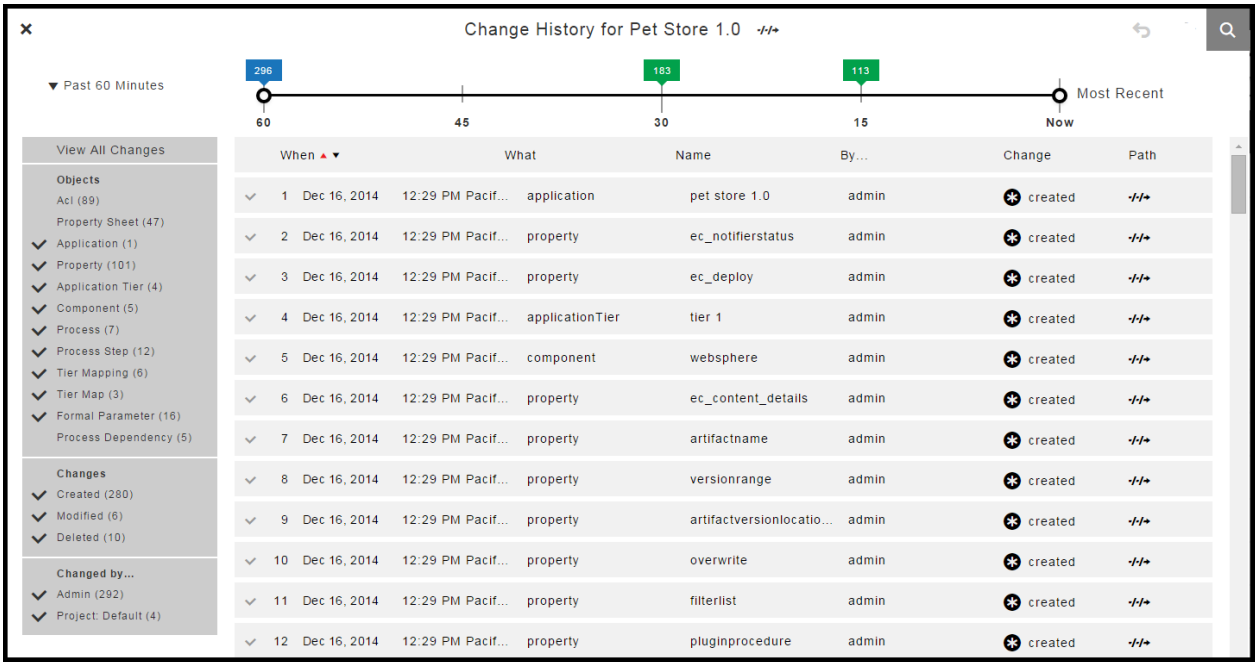
   

4. Select **Track Changes**.

   The Change History for the object opens.

   The default time increment is **Past 60 Minutes**.

   Example:



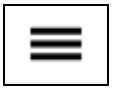
Viewing the Change History for an Application Process or Process Step

1. Go to the Applications List.
2. Select an application.

   The Application Editor opens.
3. Select an application process.

   The Application Process Visual Editor opens.
4. To view the Change History for the application process:
   - Click the **Menu** button.

     Example:

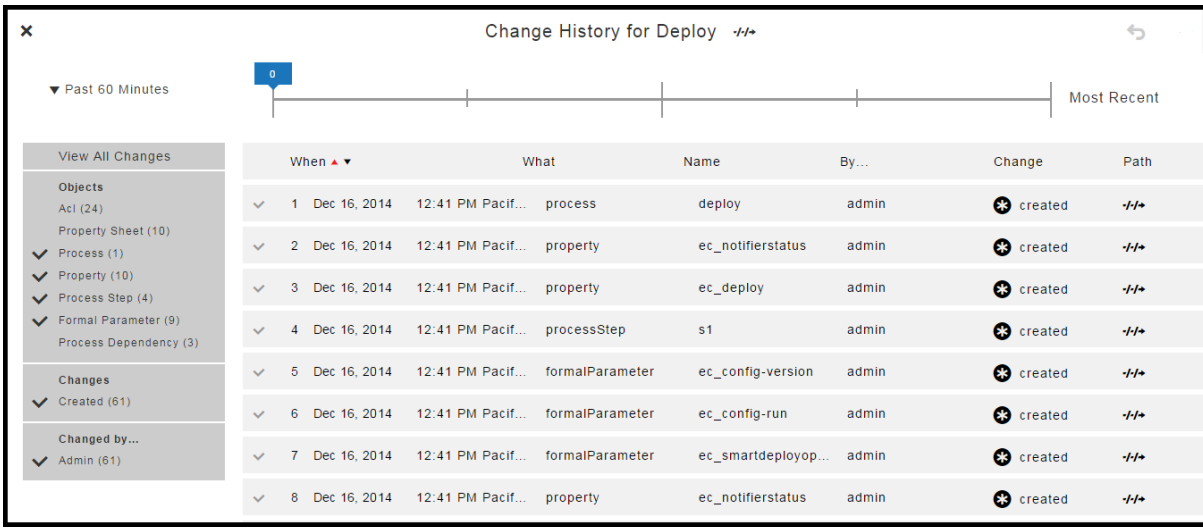
     

   - Select **Track Changes**.

     The Change History for the application process opens.

     The default time increment is **Past 60 Minutes**.

     Example:



- To view the Change History for the application process step:
  - Choose a step.
  - Click the **Edit** button in the step to open the context menu.

    Example:

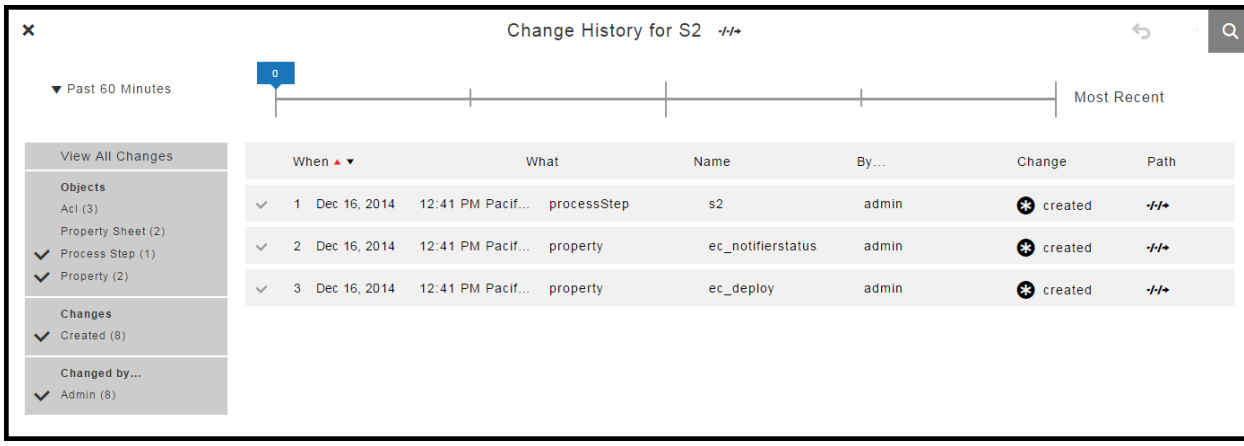
    

  - Select **Track Changes**.

    The Change History for the application process step opens.

    The default time increment is **Past 60 Minutes**.

    Example:



Component Process Visual Editor

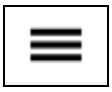
1. Go to the Applications List.
2. Select an application.

   The Application Editor opens.
3. Choose a component.
4. Select a component process for the component.

   The Component Process Visual Editor opens.

5. To view the Change History for the component process:
   - Click the **Menu** button.

     Example:

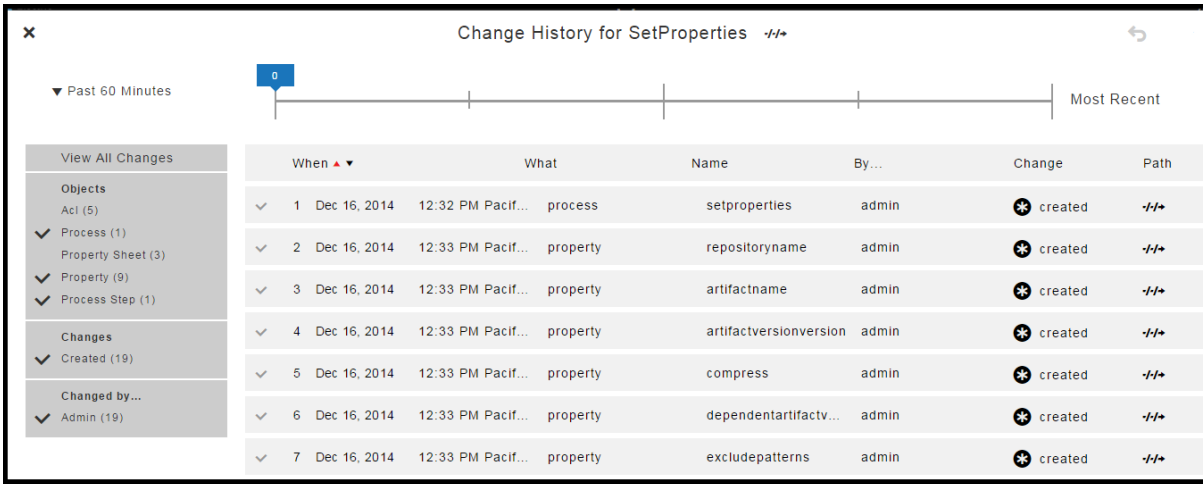
     

   - Select **Track Changes**.

     The Change History for the component process opens.

     The default time increment is **Past 60 Minutes**.

     Example:



- To view the Change History for the component process step:
  - Choose a step.
  - Click the **Edit** button in the step to open the context menu.

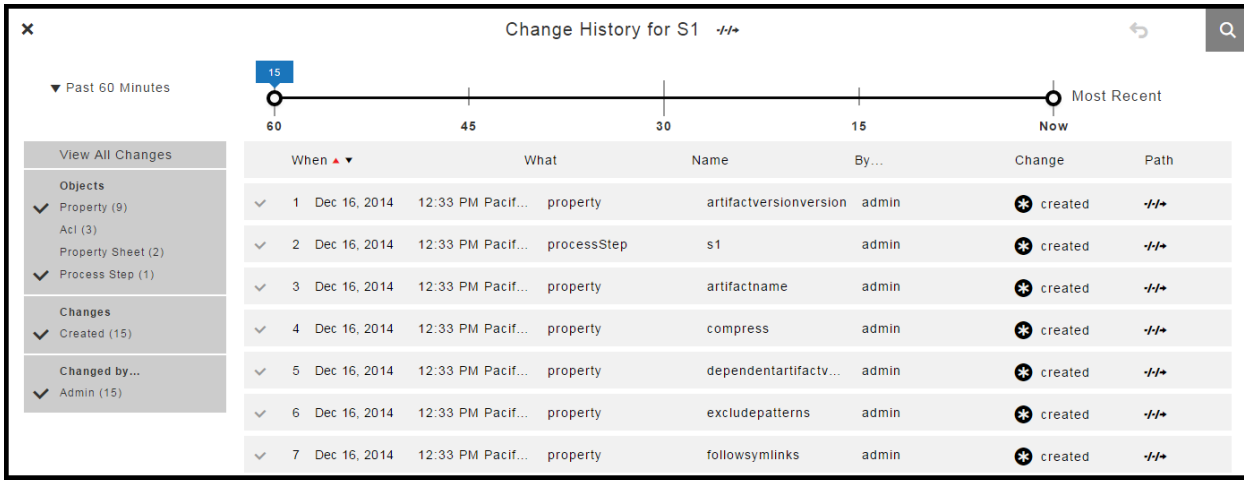
    Example:

  - Select **Track Changes**.

    The Change History for the component process step opens.

    Example:

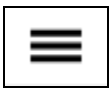


Environment Editor

1. Go to the Environments List.
2. Select an environment.

   The Environment Editor opens.
3. Click the **Menu** button.

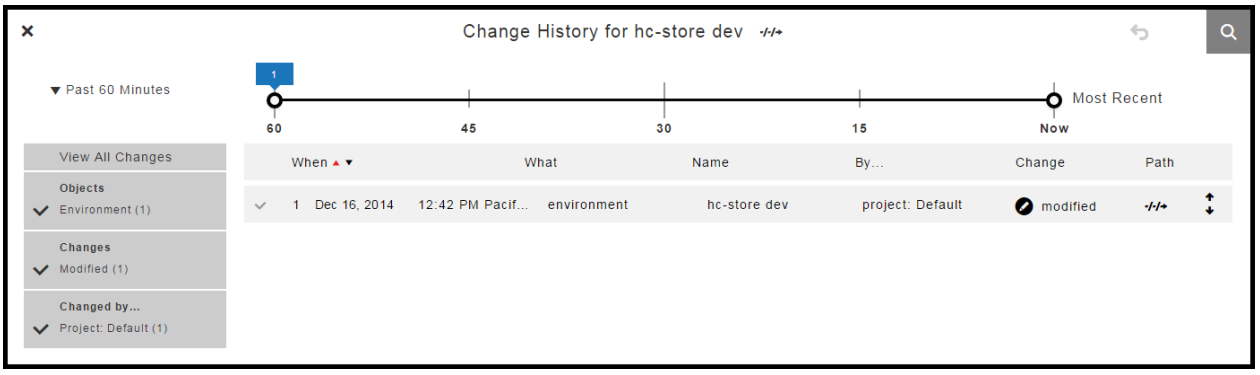
   Example:



4. Select **Track Changes**.

   The Change History for the object opens.

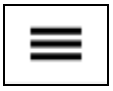
   Example:



Environment Tier

1. Go to the Environments Designer.
2. Choose an environment tier.
3. Click the **Menu** button.

   Example:

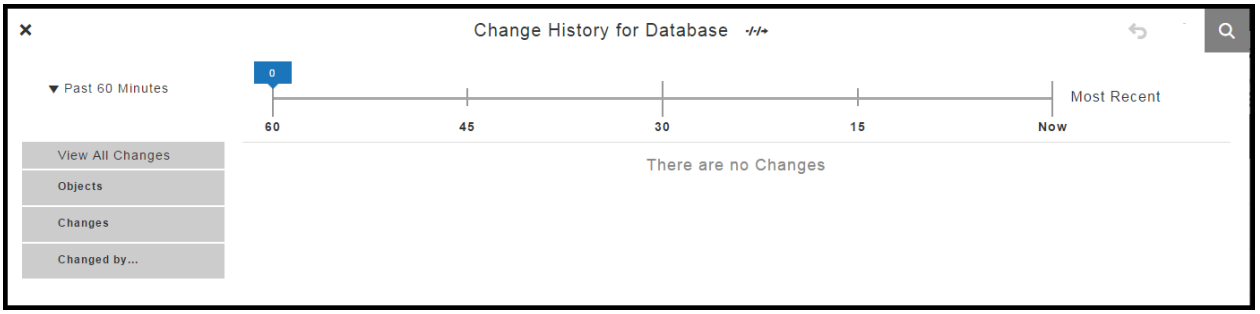


4. Select **Track Changes**.

   The Change History for the object opens.

   The default time increment is **Past 60 Minutes**.

   Example:



Viewing the Change History for Artifacts, Jobs, Projects, and Workflows

When troubleshooting why a job failed, you can view the Change History for artifacts, jobs, projects, and workflows in the automation platform.

Artifacts

1. Open the home page of the Automation Platform web UI by browsing to https://<*ElectricFlow_server*>/commander/.
2. Go to the Artifacts tab.
3. Choose an artifact.

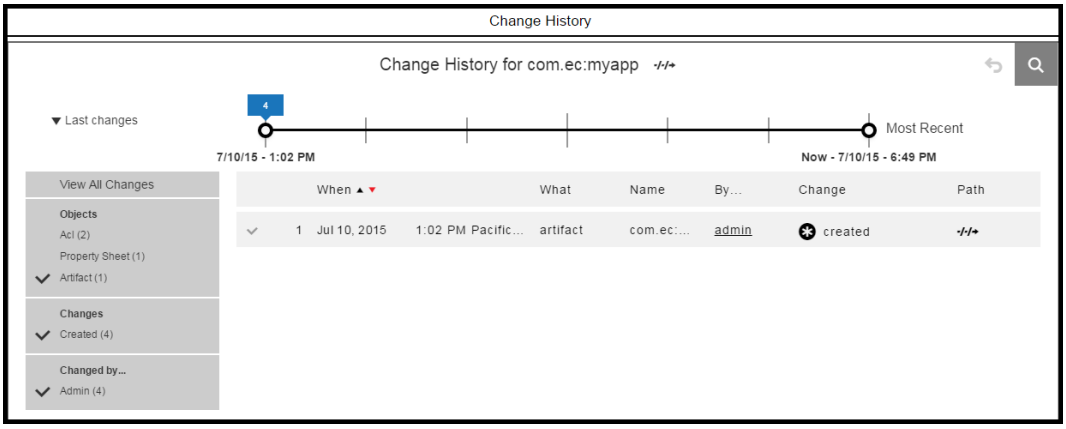
4. Click the **Track Changes**button.

   Example:

   The Change History for the selected artifact opens.

   The default time increment is **Past 60 Minutes**.

   Example:



Jobs

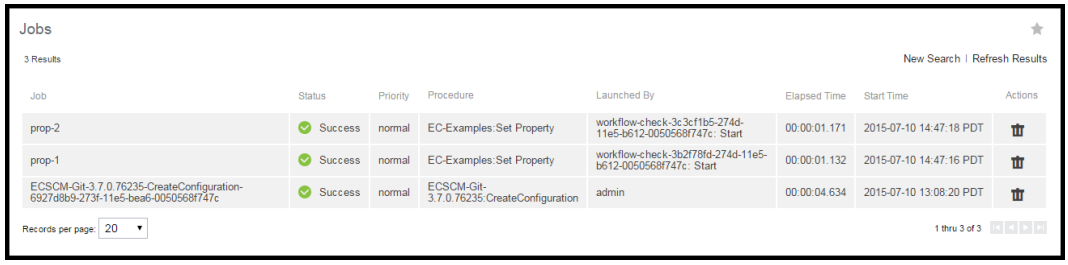
Starting from the Home page (https://<*ElectricFlow_server*>/commander/):

1. To go to the Job Details page, do one of the following:
   - Use the Jobs tab.
   - Use the Jobs Quick View list.

2. If you use the Jobs tab, follow these steps:
   - Click the Jobs tab.

     The Jobs page opens.

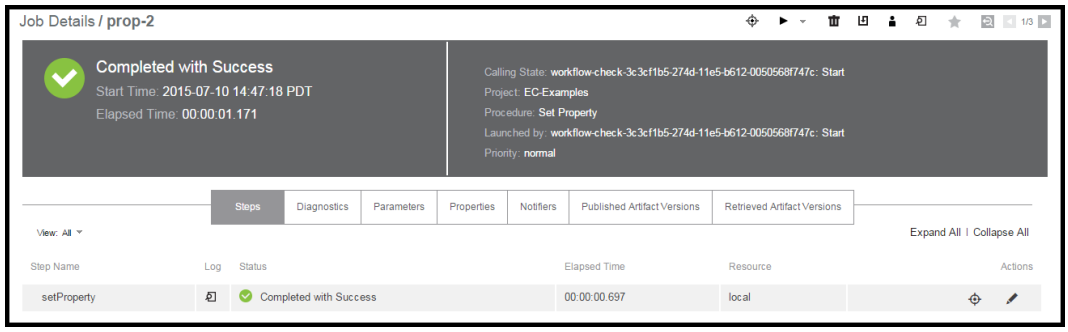
     Example:



   - Click a job name to select a job.

     The Job Details page opens.

     Example:



3. If you use the Jobs Quick View list, click a job name to select a job.

   The Job Details page opens.
4. Choose a job or job step.
5. Click **Track Changes** for the job or job step.

   The Change History for the job or job step opens.

   The default time increment is **Past 60 Minutes**.

Projects

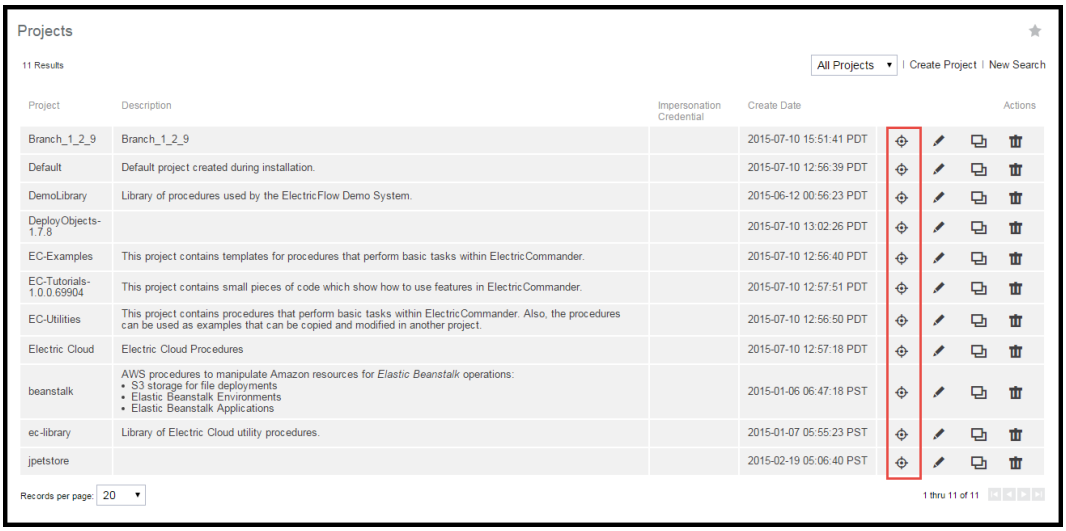
Starting from the Home page:

1. Go to the Projects tab.
2. Choose a project.

3. Click the **Track Changes** button.

   The Change History for the project opens.

   Example:



Workflows

Starting from the Home page:

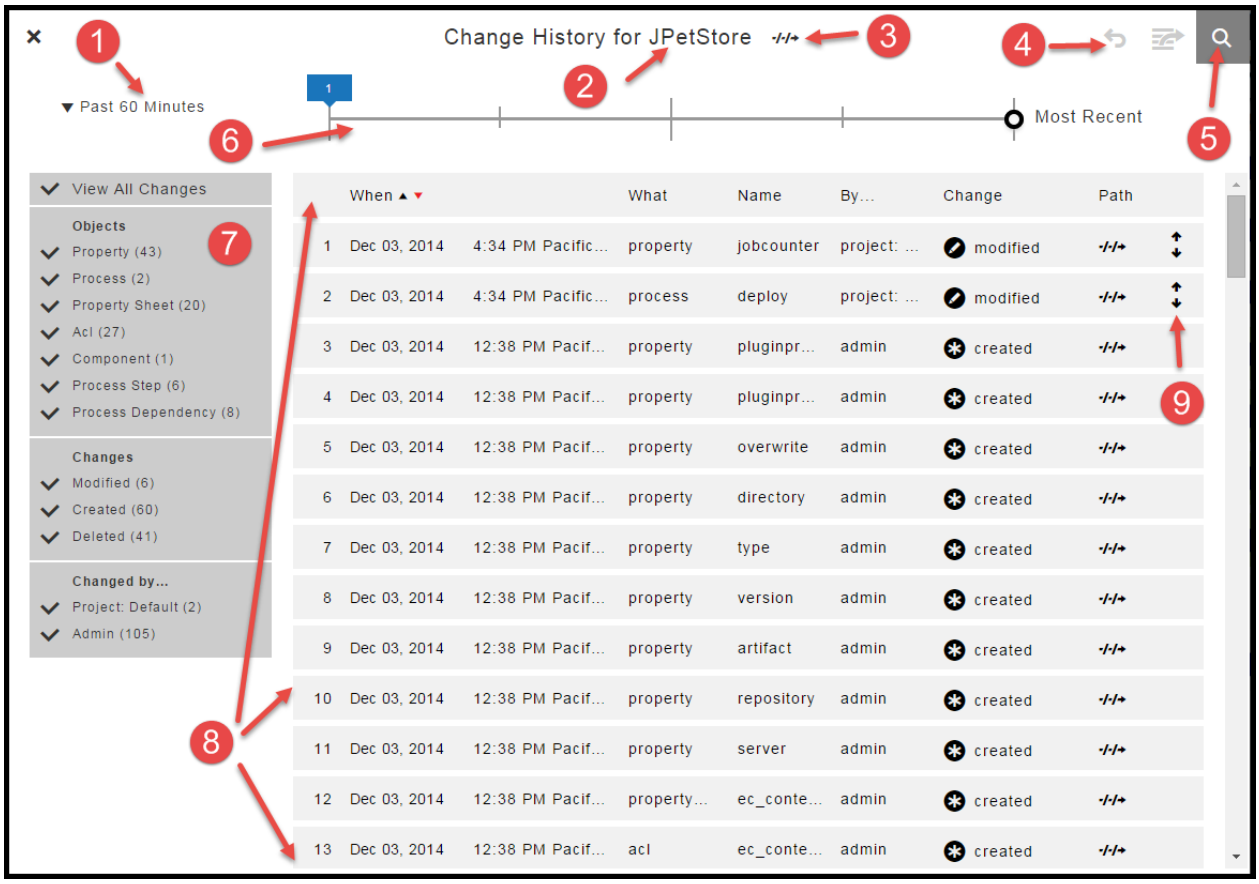
1. Go to the Workflows tab.
2. Choose a workflow.
3. Click **Track Changes**.

   The Change History for the workflow opens.

Change History Page

How to get here: Click the **Search** button to open the "Change History—Search" form, and enter the search criteria.

The search results appear in the Change History page and include the following information:

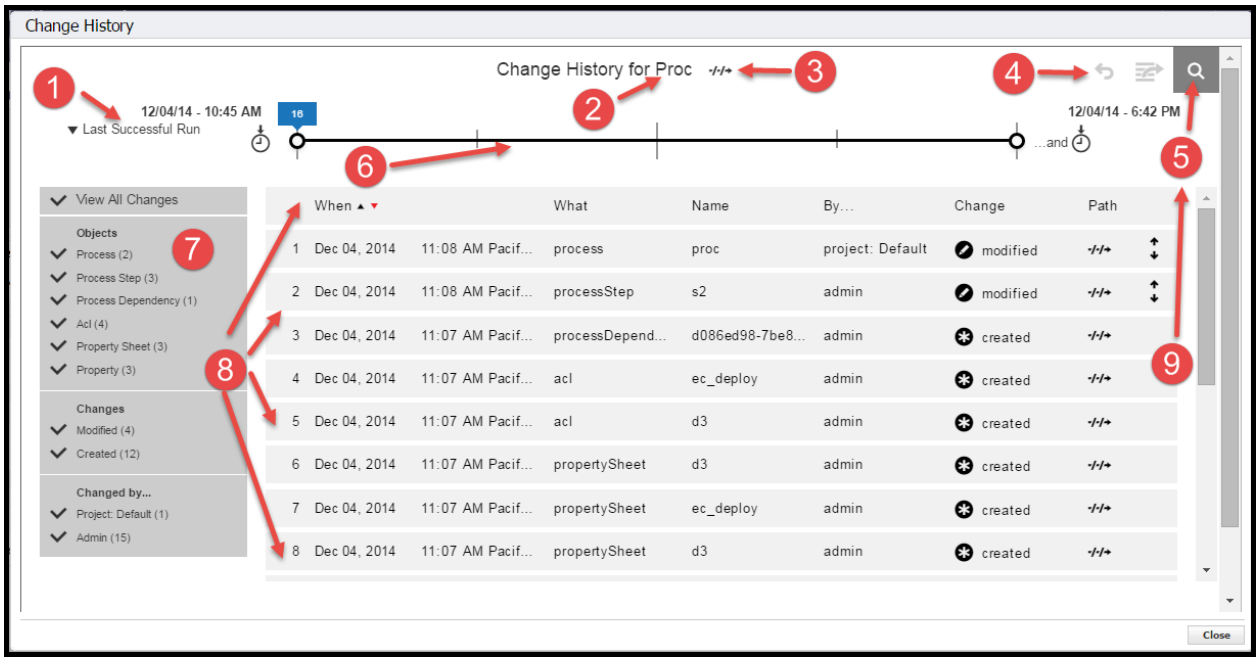


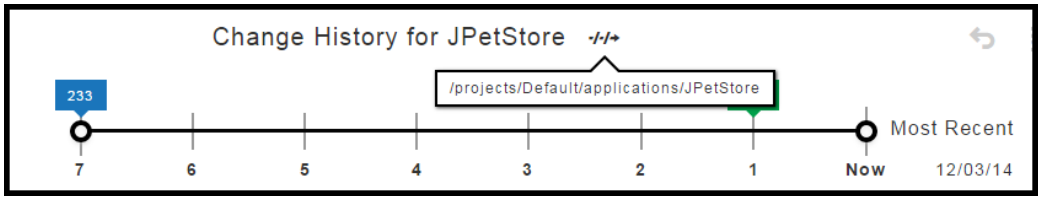
How to get here:

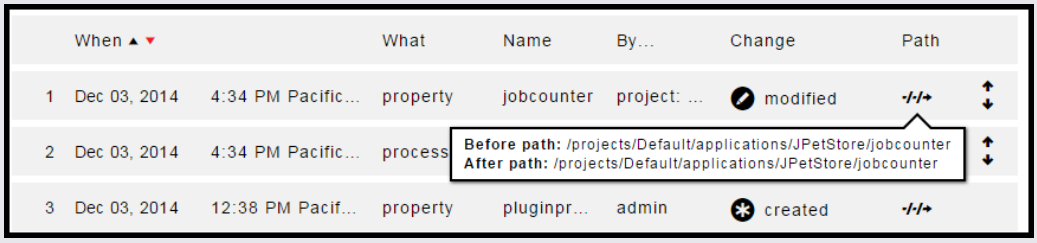
From the Home page in the Automation Platform (https://<*ElectricFlow_server*>/commander/), use one of the following methods to open and view the change history for jobs, projects, workflows, and artifacts:

- Select a job that failed > Click the job name to go to the Job Details page > Click **Track Changes** in the upper right of the page.
- Click the Jobs tab > Select a job that failed to the Job Details page > Click **Track Changes** in the upper right of the page.

The search results appear in the Change History page for the job and include the following information:



1	Time range. The end time is the current time. You can change the end time after you run the search and get the search results.
2	Name of the tracked object.
3	Path to the tracked object. Example: 
4	After making a change, you can revert or export the object.Click this to go to the undo or redo the last action on the page.
5	Click this to run a new change history search.

6	Time line. The start time is based on the time range that you selected. The end time is the current time. You can manually change the start and end times after you run the search and get the search results.
7	Filters for the change history. You can view all changes or view only selected changes. The objects in the list are the objects in the change history search results.
8	Change history for the selected object. • When—The date and time that the object changed. • What—The type of object. • Name—The name of the object. • By—The "user" that changed the object, which can be a project or a user. • Change—The type of change. • Path–Click the **View Path** button to see the path to the object. 
9	Click the **View** button to see more details about the object.

Time Line

The time line for the change history is at the top of the Change History page.

The time line is automatically separated into divisions based on the number of minutes, hours, or days between the end time and the start time.

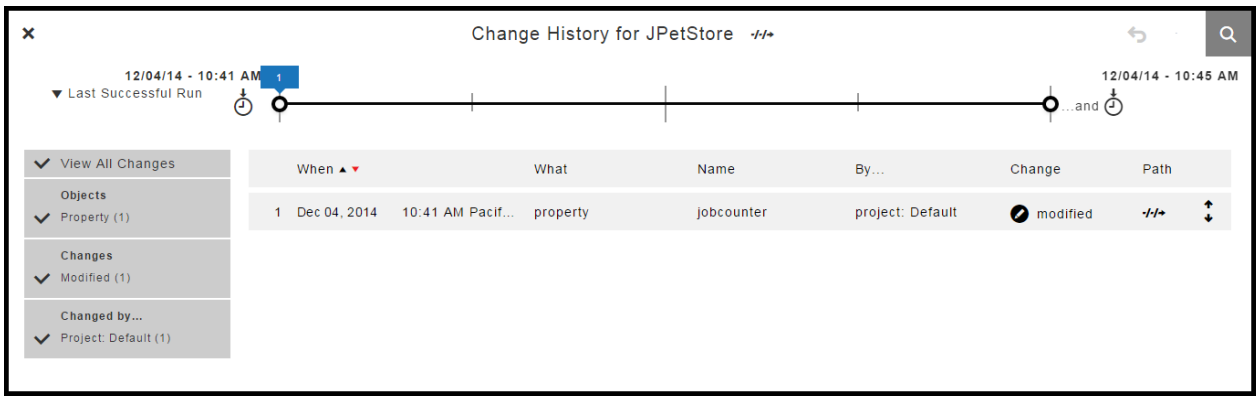
Default Settings

These are the default settings for the time line:

- The start time is the selected time range in the upper left of the page.
- The end time is **Most Recent** when the latest change to the object occurred.
- The default range is from the **Last Successful Run** to the **Most Recent**.
- The entire time line is displayed, and all the changes are in the list below the time line.

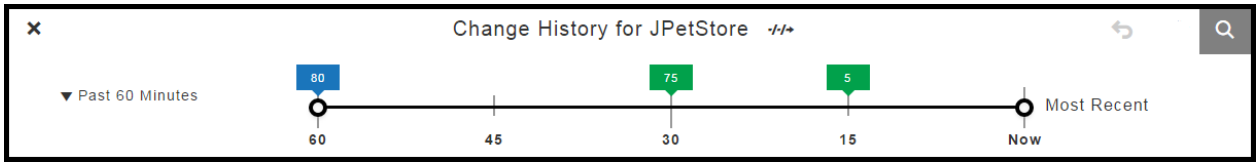
The time line on this page has the default range.

Note: The **Last Successful Run** range is available only after the first time that you run an application.

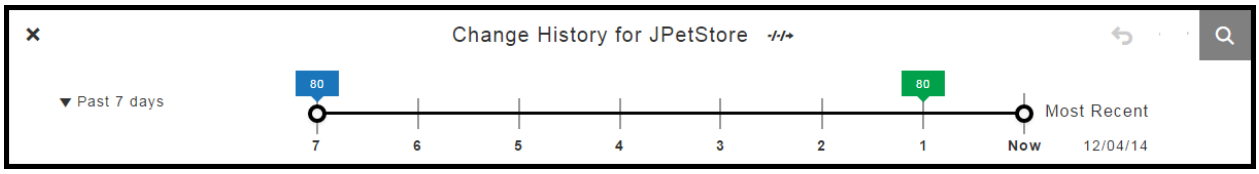


When the range is changed to **Past 60 Minutes**, the time line changes:

- The start time is 60 minutes from **Now**.
- The end time is when the **Most Recent** change occurred (**Now**).
- The time line has four divisions.



If the increment is **Past 7 Days**, the time line has seven one-day divisions.

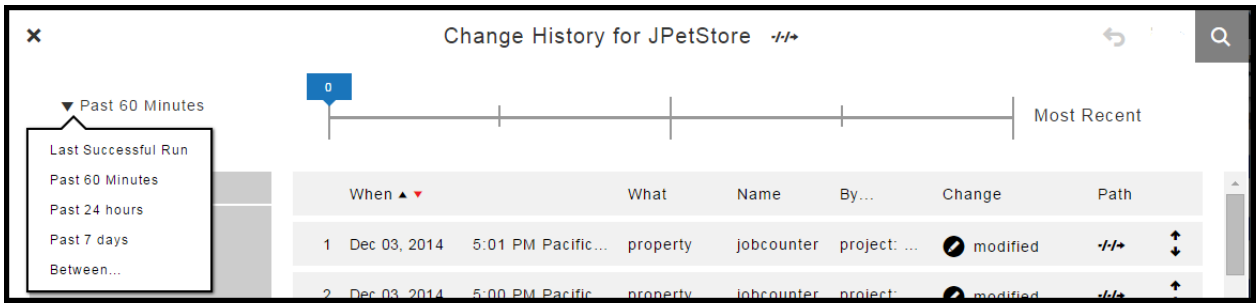


You can view the change history in different ways depending on the time range for the time line.

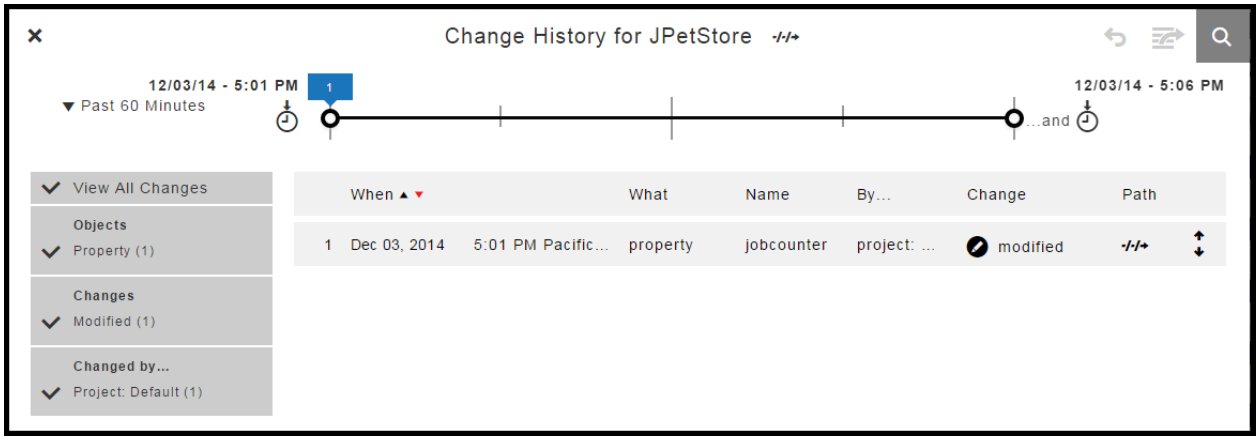
Selecting the Time Range

You can view the change history in different ways depending on the time range for the time line.

Click the down arrow next to the time range to select a different range for the change history.



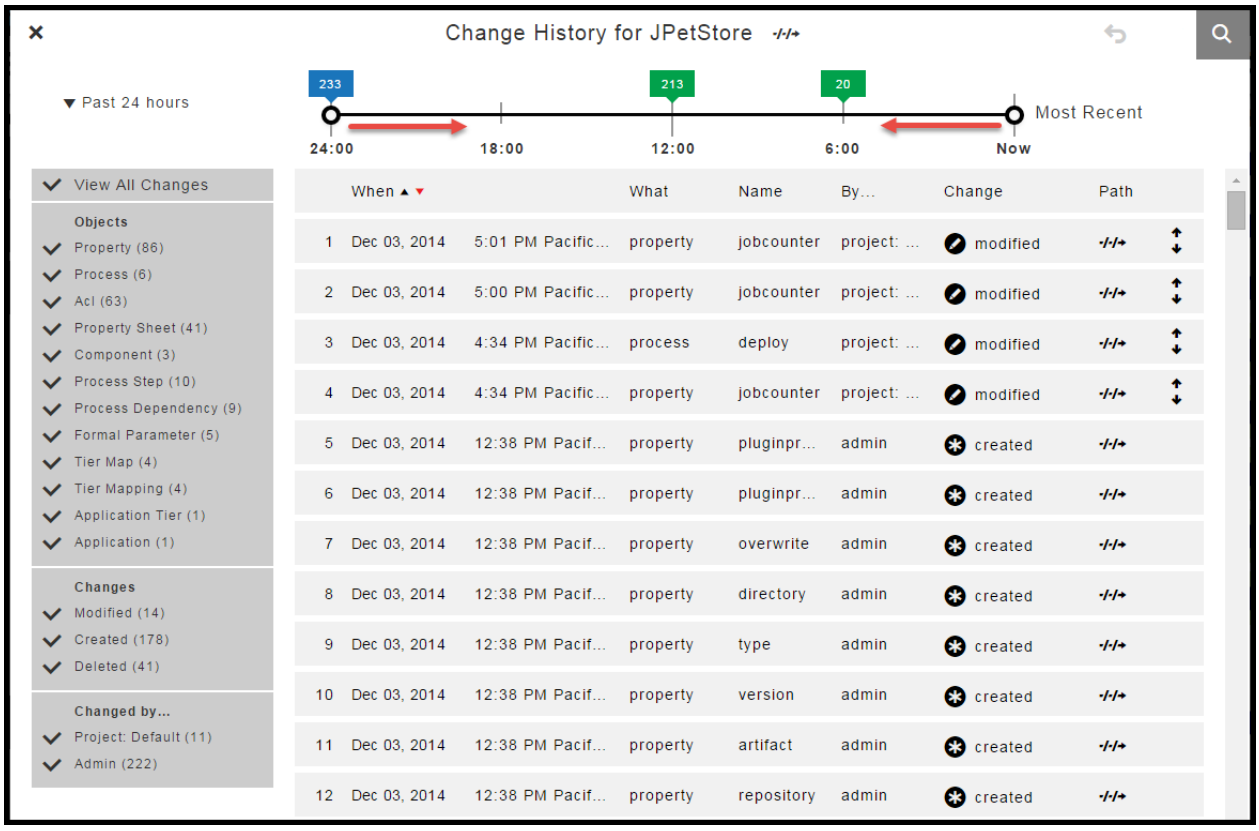
You can see the change history since the last successful run. Notice that the information on the left side shows a summary of the changes, which is a subset of the results that you got.



Moving the Start and End Times

You can manually move the start and end times on the time line.

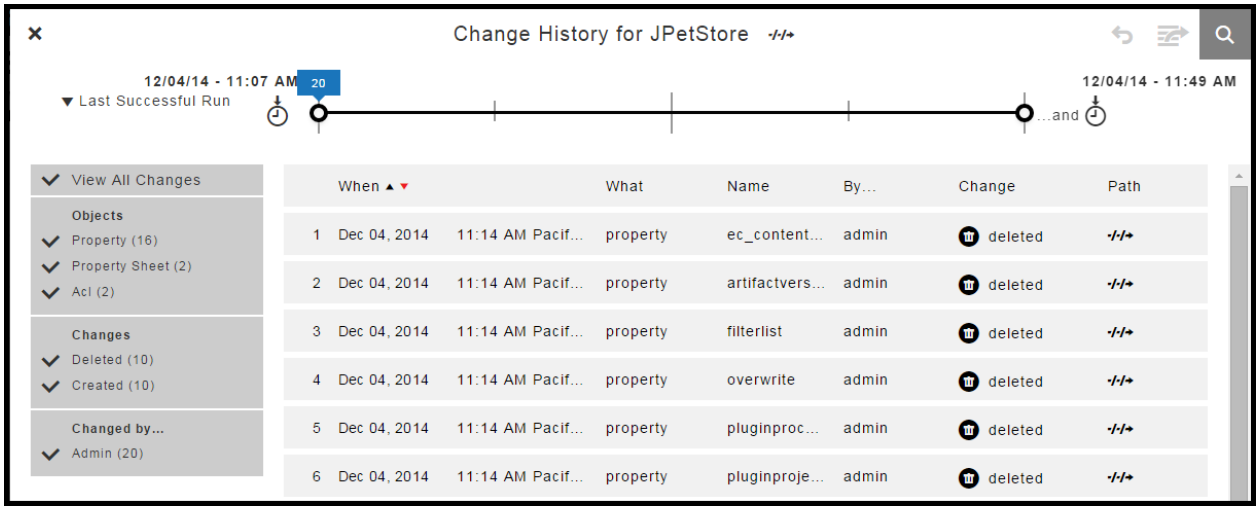
When you move the start time to 18:00 and the end time to 6:00, notice that the list of objects in the change history changes.



When you move the start time to 18:00 and the end time to 6:00, notice that the list of objects in the change history changes.

Specifying the Time Range

Example:

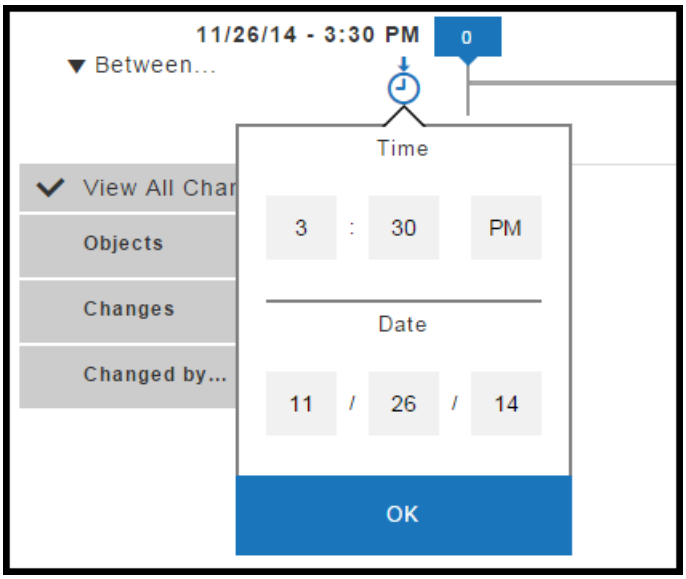


To manually set the times and dates for the start and end times:

1. Select **Between**.

   A drop down dialog box opens.

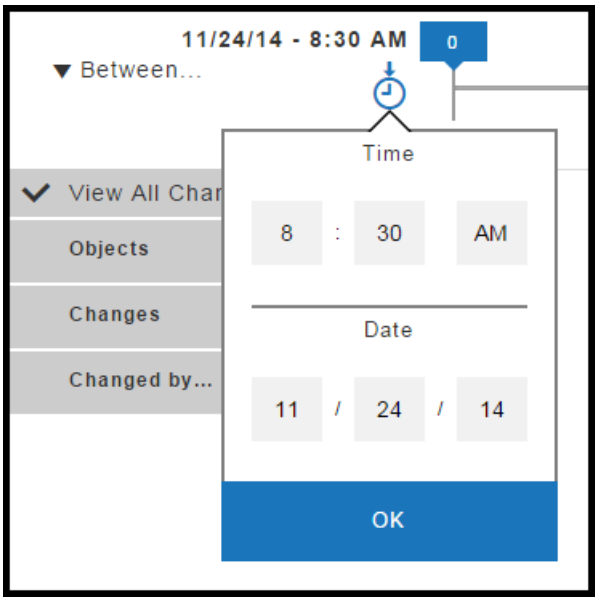
   Example:



2. Select the time and date for the start of the time line.

   The default settings are 3:30 pm and eight days before the current date.

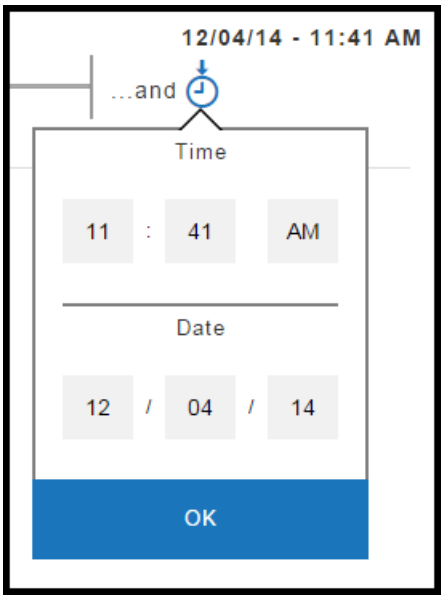
   Example:



3. Click **OK**.

   A drop down dialog box opens at the other end of the time line.

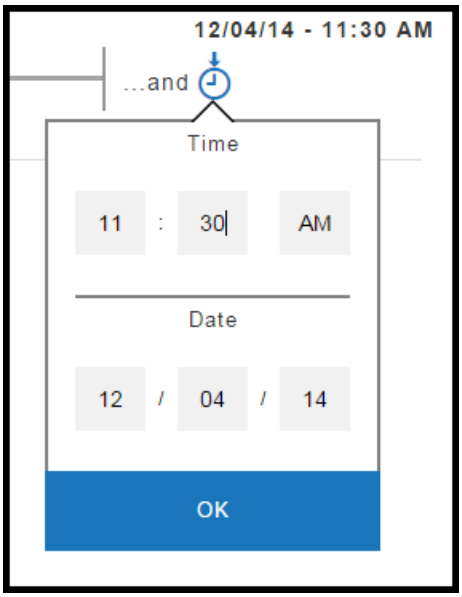
Example:



4. Select the time and date for the end of the time line.

   The defaults are 3:30 pm and the current date.

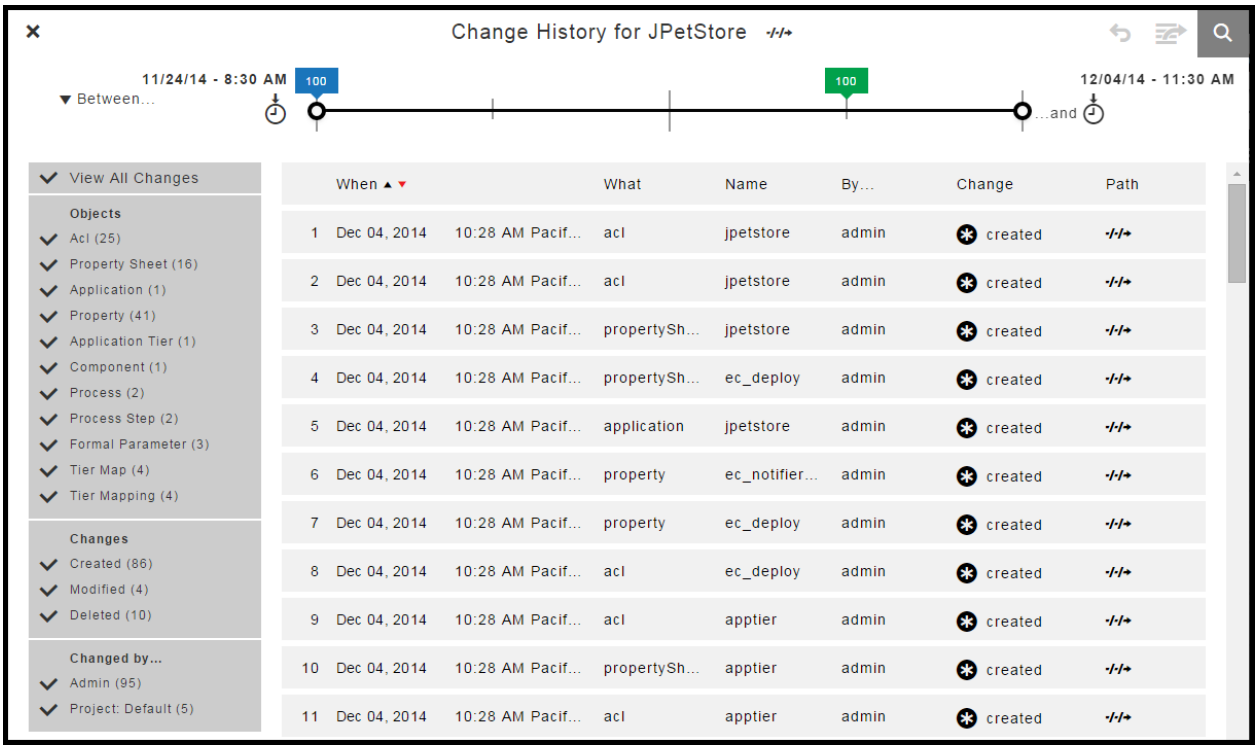
   ## Example:



5. Click **OK**.

   The time line changes to show only the changes from the start and end times and dates that you selected.

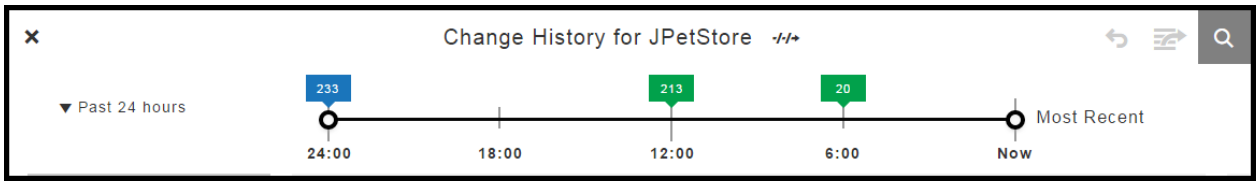
   Example:



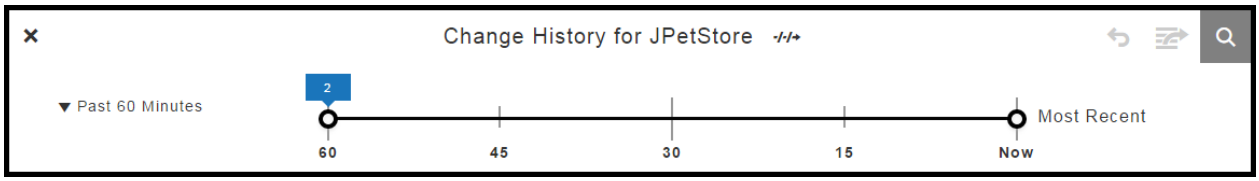
Number of Changes

The time range at the top of the change history shows the number of changes.

- There have been 233 changes in the last 24 hours.
- There have been 213 changes in the last 12 hours.
- There have been 20 changes in the last 6 hours.

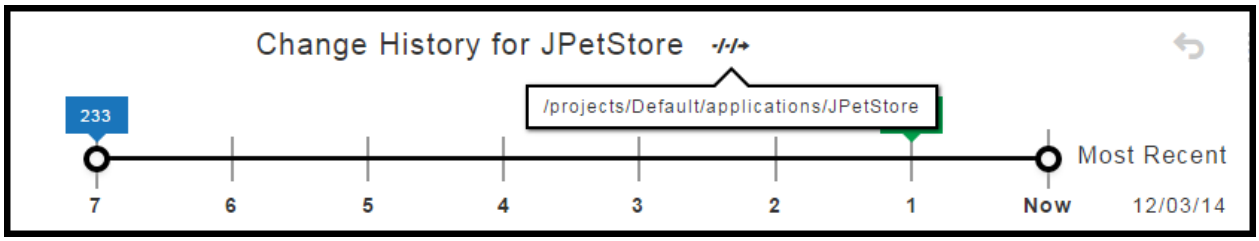


When you change the time range, the number of changes also changes. In the last 60 minutes, there have been only two changes.

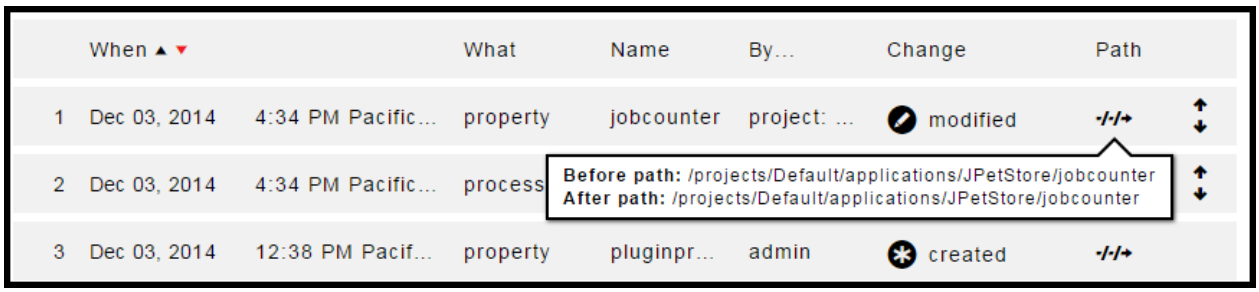


Paths to Objects

Click the **View Path** button next to the "Change History for JPetStore" title to see the path to the application.

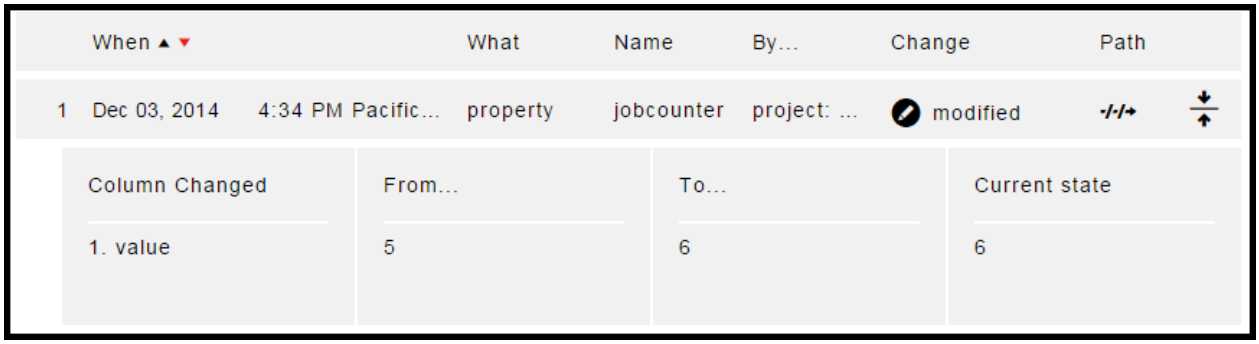


Click the **View Path** button to see the change in the path to the object before and after the change.



Detailed Object Changes

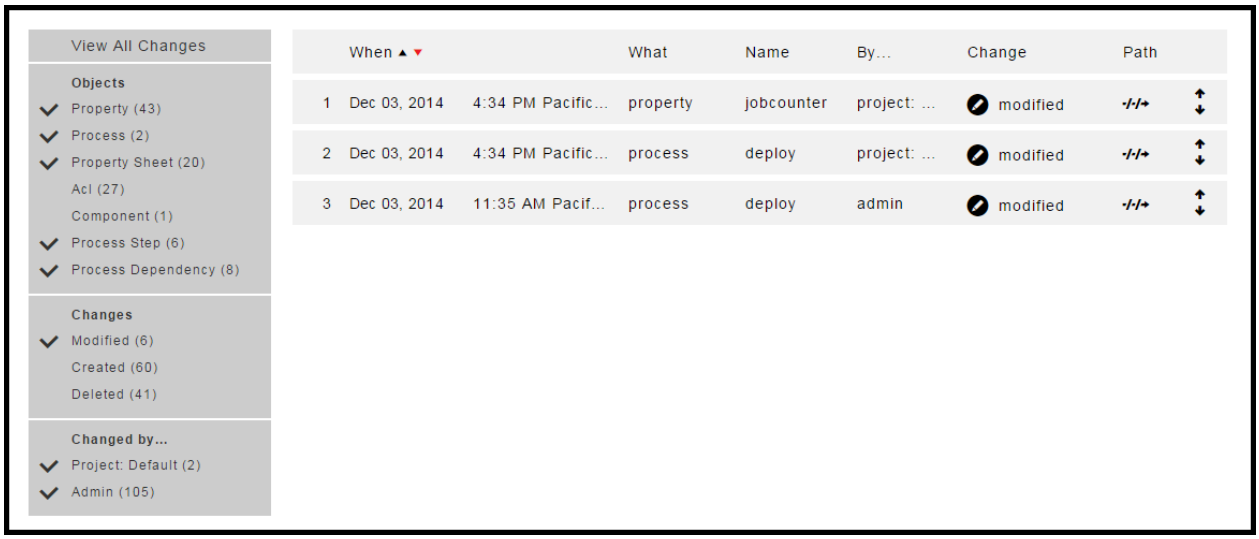
Click the **View** button to see the change in the property called jobcounter.



Filters

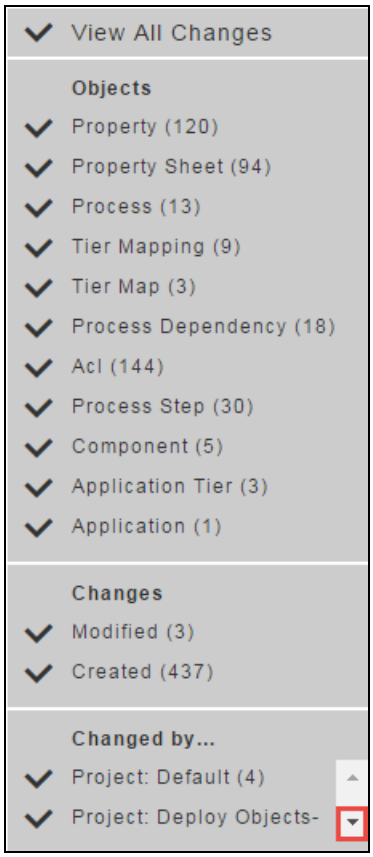
You can use filters to view changes to specific objects, the types of changes, and the users how made those changes.

Instead of selecting **View All Changes**, you can select specific objects, such as only properties, processes, property sheets, process steps, and process dependencies that have been modified by the Project:Default and Admin users.

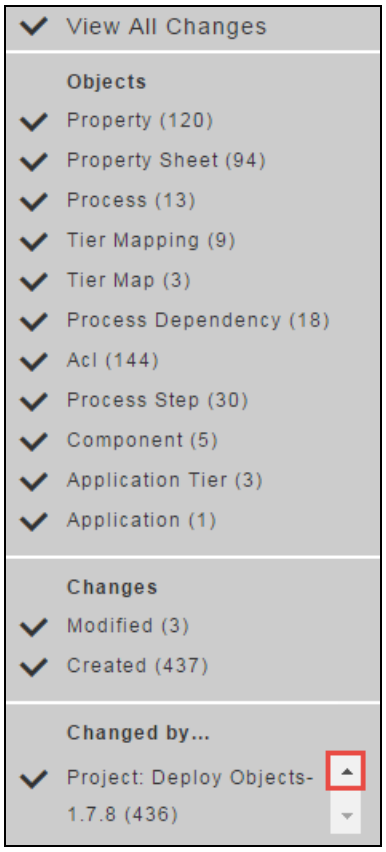


When the list of filter criteria is long, not all of the criteria may appear in the filter list. To see all of the criteria, use the up or down arrows to see all the options.

This list does not show all of the users. Use the up and down arrows to see all four of the users.



Click the down arrow to see the other users.



Modifying What You See the Change History

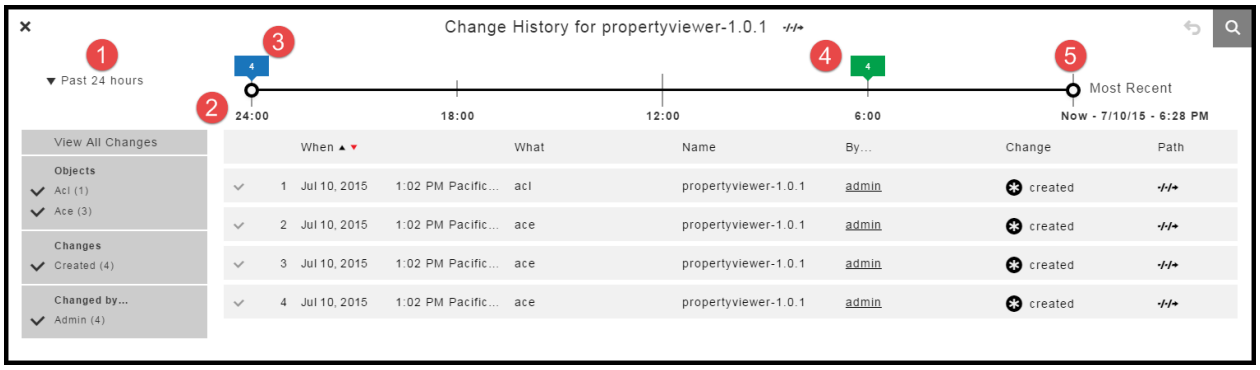
You can modify the information that appears in the Change History with these settings:

- Time line–See Change History Time Line on page 1380.
- Filters–See Change History Filters on page 1387.

Change History Time Line

The time line is at the top of the Change History page.

This example shows the Change History for a property called *propertyviewer-1.0.1*.



1	Time increment. The default is **Last Changes**. Click the down arrow to select another time increment.
2	The system automatically calculates the minutes, hours, and days since the last successful run. In the example, the last successful run occurred 24 hours ago. The time line is divided into four 6 hour subdivisions.
3	Total number of changes in the selected time increment.
4	The number of changes that occurred between 6 hours ago and now is 4. When you click the change number, the Change History is updated and shows only those changes.
5	Drag the start and end time markers to view specific changes.

Default Settings

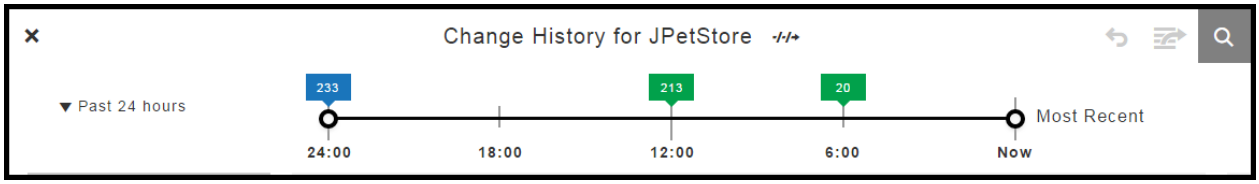
The default time increment is **Last Change**.

The entire time line is displayed, and all the changes are in the list below the time line.

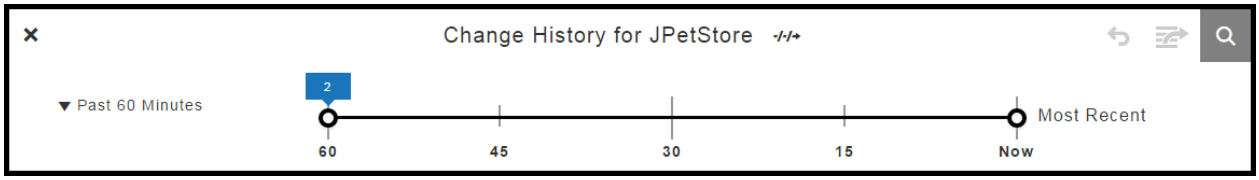
Number of Changes

The time line shows the number of changes throughout the time increment. In the following example:

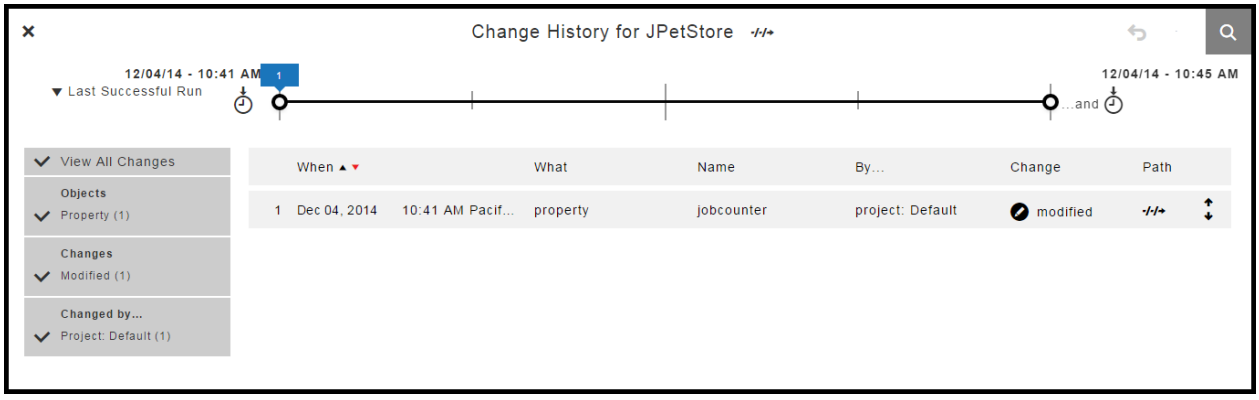
- There have been 233 changes in the last 24 hours.
- There have been 213 changes in the last 12 hours.
- There have been 20 changes in the last 6 hours.



When you change the time increment, there have been two changes in the previous 60 minutes.



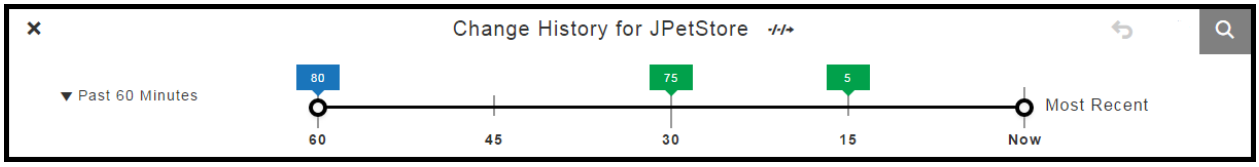
Time Increment



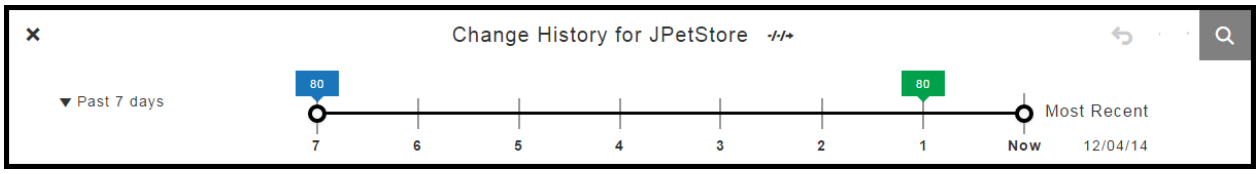
The system automatically determines how the time line is divided for the selected time increment.

When the range is changed to **Past 60 Minutes**, the time line changes:

- The start time is 60 minutes from **Now**.
- The end time is when the **Most Recent** change occurred (**Now**).
- The time line has four divisions.



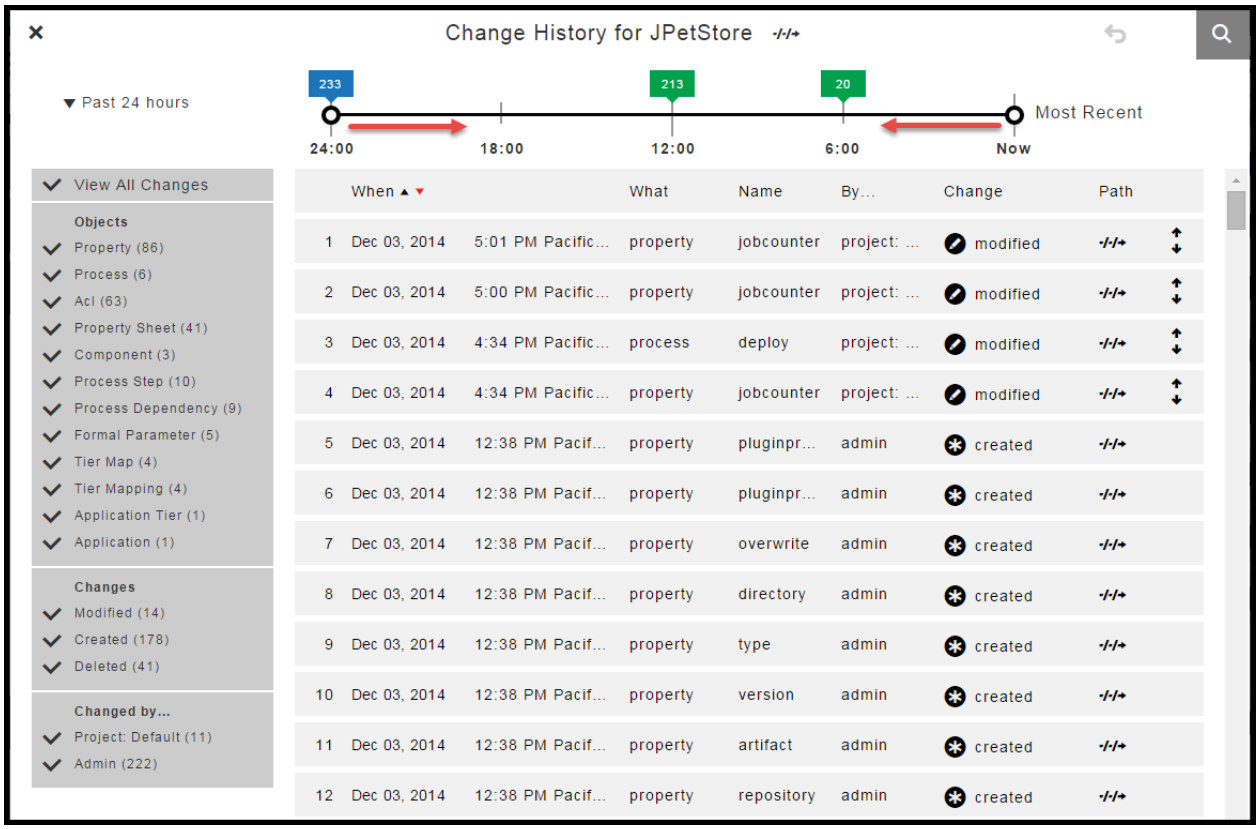
If the increment is **Past 7 Days**, the time line has seven one-day divisions.



Moving the Start and End Times Manually

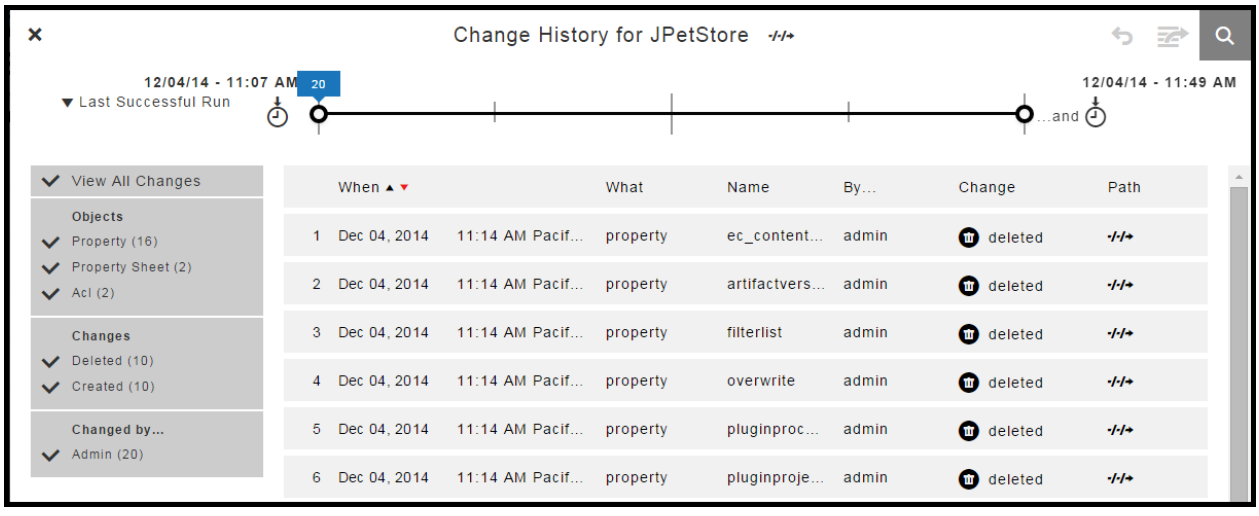
You can manually move the start and end times on the time line.

When you move the start time to 18:00 and the end time to 6:00, the list of objects in the change history changes.



Setting Custom Time Increments

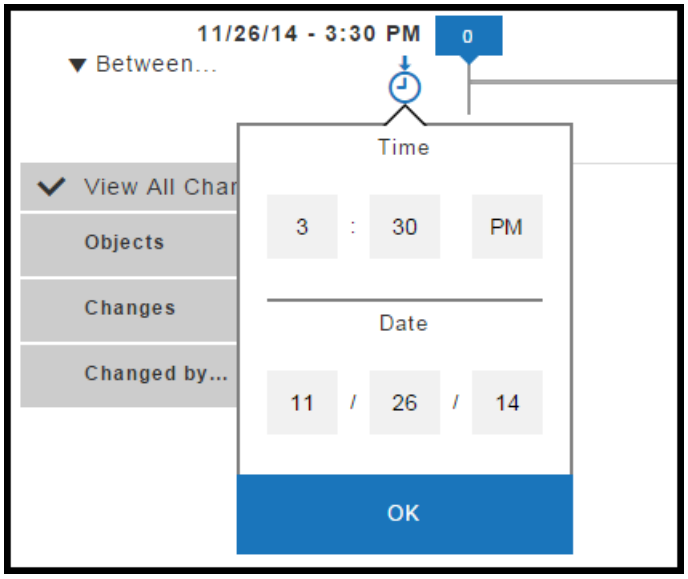
Example:



- Select **Between**.

  A drop down dialog box opens.

Example:



- Select the time and date for the start of the time line.

  The default settings are **3:30 PM** and eight days before the current date.

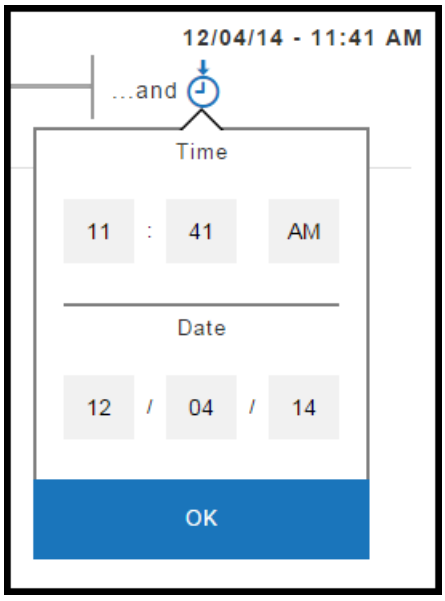
  ### Example:

  style="max-width: 6.5in;width: auto;height: auto;min-width: auto;min-height: auto;max-height: auto;" />

- Click **OK**.

  A drop down dialog box opens at the other end of the time line.

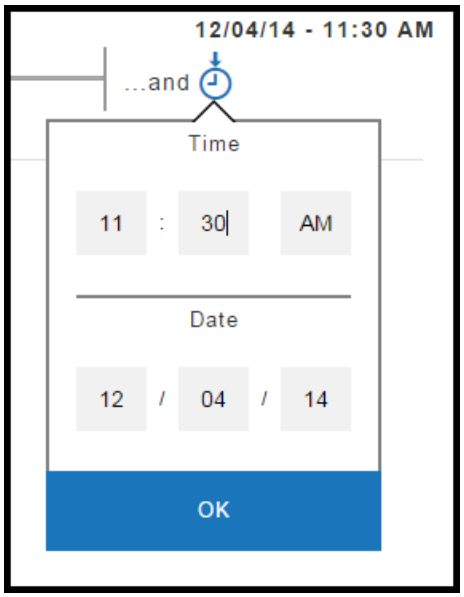
  ### Example:



- Select the time and date for the end of the time line.

  The defaults are **3:30 PM** and the current date.

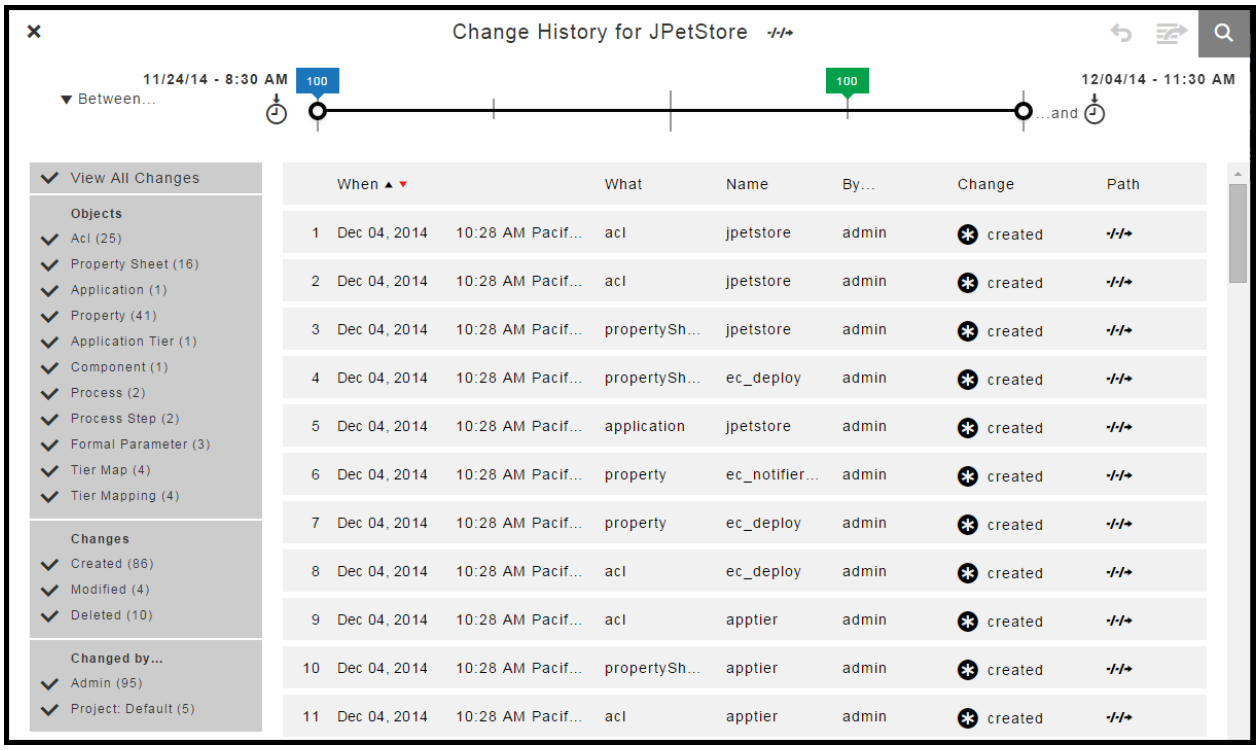
  ### Example:



- Click **OK**.

  The time line changes to show only the changes from the start and end times and dates that you selected.

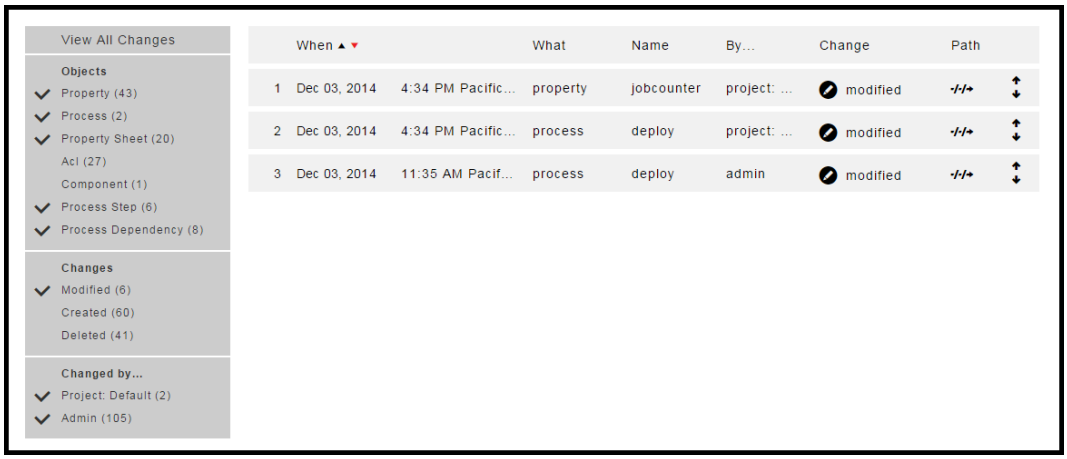
Example:



Change History Filters

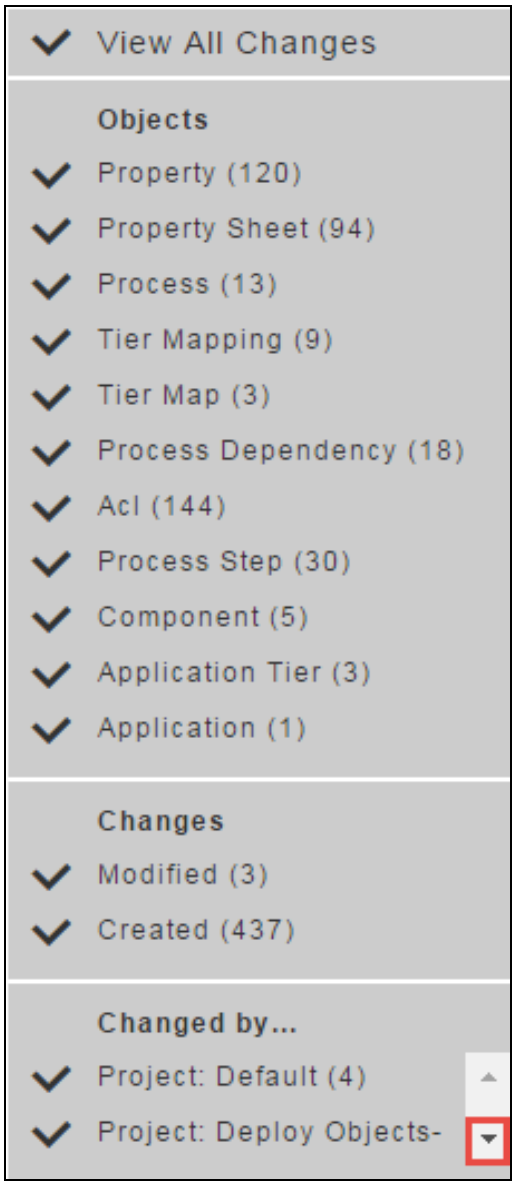
The Change History has records only for an object and its tracked entities, not all entities belonging to the object. You can use filters to view changes to specific objects, the types of changes, and the users how made those changes if they are are tracked.

In this example, instead of selecting **View All Changes**, you can select specific objects, such as only properties, processes, property sheets, process steps, and process dependencies that have been modified by the Project:Default and Admin users.

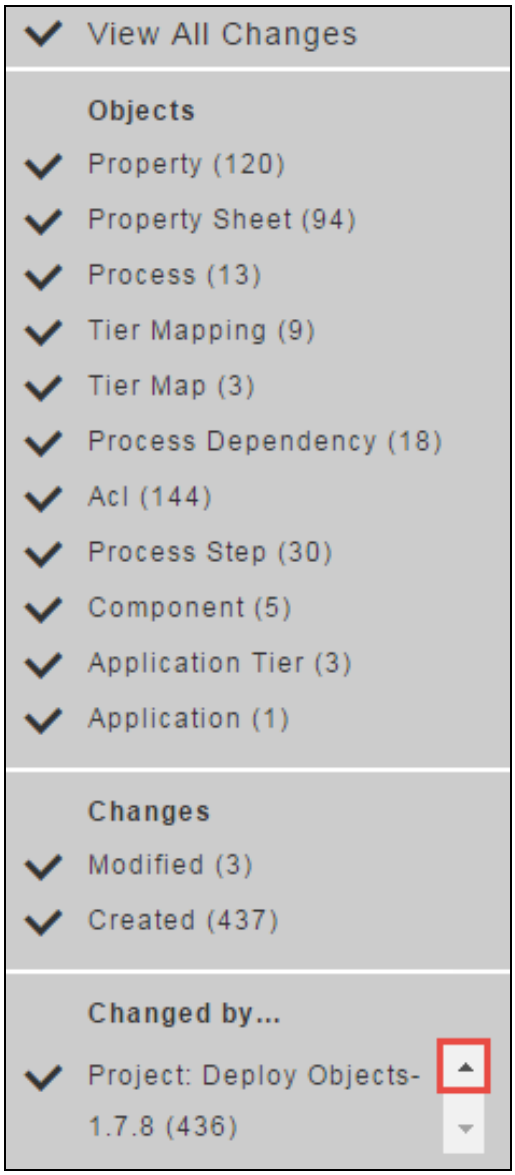


When the list of filter criteria is long, not all of the criteria may appear in the filter list. To see all of the criteria, use the up or down arrows to see all the options.

This list does not show all of the users. Use the up and down arrows to see all four of the users.



Click the down arrow to see the other users.



Reverting Changes to a Tracked Object and Its Tracked Contents

The Change History displays a list of changes in reverse chronological order (from latest to oldest) for tracked entities in objects such as applications, independent microservices, procedures, workflows, workspaces, resources, and project-owned components such as library components. Each row in the Change History or a tracked object is a record of a change made either to the tracked object or to a tracked object owned by the object (directly or indirectly). For example, the Change History of a project would show both changes to the project itself and changes to any tracked object in the project, such as its procedures, the steps for each procedure, or properties belonging to the procedure steps.

Some types of objects (mostly run-time objects such as jobs, because tracking those would be resource-intensive and seldom useful) are not tracked by the Change Tracking feature. For a few types

of tracked objects, some of their attributes are not tracked (for example, agents are tracked, but certain attributes of agents that vary frequently in real-time, such as their current status, are not).

Changes to untracked objects, or to untracked attributes of tracked objects, are not shown in the Change History and cannot be reverted. In general, objects that are normally created or changed manually are tracked, and objects that are normally created or modified during automated processing are not, However, depending on your specific usage patterns, this behavior may not be recorded in the Change History, such as when you frequently run jobs that use ectool or API commands to automate the creation of new procedures or steps.

> **Note:** You must have the *Read*, *Modify*, and *Execute* permissions to revert changes in the UI or to access to the `revert` API function.

Follow these steps to select a tracked object and its tracked entities that you want to revert:

- Go to the Change History for the object that you want to revert using one of these methods:

  - Click the **Track Changes** button ( )for the object.
  - From the Automation Platform Home page (https://<*ElectricFlow_server*>/commander/),

    click the **Change History**button ( ) to go to the "Change History—Search" page and search for the object.
- Locate the time period in which you are interested.

  A list of changes for the tracked object appears.
- If you decide to revert the changes for an object owned by the tracked object or for an object that owns the tracked object, use one of the methods to view the list of changes for this object:
  - Close the current Change History and navigate to the Change History for the object.

  - Click the **View Path** button ( ) on the top of the current Change History, copy the path, go to the "Change History—Search" page, and search for a suitable addition to or truncation of that path.

- Select a row (record) in the Change History corresponding to the earliest of the changes that you want to undo.

  You will be reverting the object and all its tracked contents to their state directly before the selected change was made.

  1. If the **Revert** button ( ) becomes enabled, ElectricFlow is capable of reverting the selected changes to the object. is tracking the changes to this object.

     Go to Step 5.

  2. If the **Revert** button is still disabled, ElectricFlow is not currently capable of reverting changes to the selected object. This could be due to tracking of this object being disabled and re-enabled, or the Change History feature being disabled and re-enabled. Also, not all types of tracked objects can currently be reverted (and due to mutual consistency requirements in the database format, certain types of objects cannot be reverted).

     If you are unable to revert the selected object, you may still be able to achieve what you need to do one of these ways:

     - Revert the parent object that owns the object you want to revert.
     - Separately revert the individual objects owned by the selected object or even nested partial-reversions-of-reversions, possibly combined with manually reversion (undoing) certain changes.

     Go to Step 3 to search for an for an owned or owning tracked object.

- Before clicking the **Revert** button, make certain that you know all the changes that will be reverted, and have confirmed that this is what you want to do. (While it is possible to re-revert the changes caused by a mistaken reversion operation, doing so leaves a lot of activity in the Change History.)

  Important: All of the changes for tracked entities will be reverted, even the ones that you do not see in the Change History because of the timeline and filter settings.

  1. You may need to modify the timeline and filter settings to display all records in the Change History before you verify that all of the records are changes you want to revert.
  2. Make sure that the end of the timeline is set to **Now** and that the filters are set to **View All Changes**.
  3. Look through the entire list of changes from the most recent to the one you have selected, expanding them to see the details if needed, and confirm that you really want to revert all of them.

  Note:
  If you want to revert some of the changes shown but not others, you may be able to achieve what you want to do with some combination of the following methods:

  - Reverting just certain objects owned by the object that you want to revert.
  - Using nested or overlapping sets of reverts of owned objects.
  - Undoing individual changes manually using the ElectricFlow UI.
  - Using nested partial-reverts-of-reverts.

  If you fully understand the XML format used for ElectricFlow exports, you can use the `export` API call to export the past and current states of a part of the object hierarchy, diff and merge them using a suitable tool for diffing and merging XML files, and then re-import the results. Caution is strongly suggested if you are using this approach to revert objects for which ElectricFlow does not currently support reversion, because:

  - There are some places in the ElectricFlow XML data hierarchy where peer objects refer to each other in ways that need to be kept consistent (such as by index of current position in a list).
  - For some types of objects, re-importing them can change their IDs and break references-by-id.

- Click the **Revert** button. Depending on the size of the object and every tracked object it owns, this operation could take some time (as long as exporting a past state of them and then re-importing it).

  If this operation is successful, the object and its tracked entities are reverted back to the state at the date and time of the record selected in Step 3. A message appears that the Revert operation was successful.

  If this operation is not successful, the object and its tracked entities are not reverted, and a message appears about the failed operation.

Chapter 8: Notifications

Application or Microservice Processes and Process Steps

ElectricFlow sends email notifications to users and groups at the application-process or microservice-process and application-process-step or microservice-process-step (type: process) levels. The notifications are triggered based on how the job finishes (the onCompletion event) and on the job outcome (success or failure).

When configuring email notifications:

- You specify users, groups, or email addresses as recipients of the notifications.
- You can also target notifications at specific environments.
- You can enable or disable notifications at the application-process-step or microservice-process-step levels.
- You can enable or disable email notifications for manual steps.

> **Note:** The user and group settings are defined and managed at the platform level. See Users and Groups for more information.

Go to Configuring Email Notifications in Application or Microservice Processes about how to set up email notifications. When notifications are enabled and a default template is used, you do not have to perform additional steps to set up an email notification.

ElectricFlow provides default templates for success and failure outcomes. All the templates have a name, subject, body, and type content that are stored as properties in ElectricFlow.

You can edit existing templates or create new email templates in ElectricFlow. See Email Notifier—create new or edit existing email notifier on page 1104 for details about editing HTML default notifications and to create new email templates.

ElectricFlow also sends notifications for assignees and approvers of manual process steps. See Manual Tasks and Steps on page 111 for more information.

Jobs, Workflow States, and Procedures

For jobs and job steps, two types of email notifiers:

- On Start Notifier – Sends an email when a job or job step starts.
- On Completion Notifier – Sends an email when a job or job step completes.

For workflow states, three types of email notifiers:

- On Enter Notifier – Sends an email when the state becomes the workflow's active state.
- On Start Notifier – Sends an email after the state's subjob or subworkflow starts. If no subjob or subworkflow is defined for that state, the notifiers will not be sent.
- On Completion Notifier – Sends an email after the state's subjob or subworkflow completes. If no subjob or subworkflow is defined for that state, the notifiers will not be sent.

Using email notifiers:

- Attaching an email notifier to a procedure results in a corresponding email notifier associated with the job that is created when the procedure is executed.
- Attaching an email notifier to a procedure step results in a corresponding email notifier associated with the job step created when the procedure referencing the procedure step is executed.
- Attaching an email notifier to a state definition results in a corresponding email notifier associated with the state that is created when the workflow is executed.

You must create an email configuration before you create and configure email notifiers at the platform level. Then you can send them to individuals and groups.

To create and edit email configurations, see Email Configurations on page 1103, and Email Configuration—create new or edit existing email configuration on page 1103.

To create or edit email notifiers, see Email Notifier—create new or edit existing email notifier on page 1104. See the topics starting with "Email Notifier Template" for examples of HTML templates that you can use as-is or copy and rename to create a new template.

Pipelines

Notifications for pipelines are automatically sent in the following scenarios. You do not have to perform any additional steps to send email notifications.

- For a manual task in a stage, the appropriate notifications are sent to the user or group who performs that task and to the user or group who responds when the task is completed.
- When entry and exit gates have specified approvers, ElectricFlow sends approval notifications to them.
    - You can set the approvals such that approval from only one of the approvers is sufficient for the pipeline to progress. In this case, you can have one approval rule and have all the approvers assigned to that rule.
    - Instead, if you want all the key individuals to approve sequentially, you can specify multiple approval rules, each for a specific user.
    - In addition to specific users, approval rules support groups. Approval or rejection from any user in the group will allow the pipeline to progress or stop.
- You can enable or disable email notifications for pipeline gate tasks and manual task approvals.

After approvers log into ElectricFlow, they can approve or reject the approval rules. The pipeline's progress will continue if the approval rules are approved. The pipeline will end with the appropriate error if the rules are rejected.

See Pipeline Concepts on page 413 for more information about notifications in pipelines.

Configuring Email Notifications

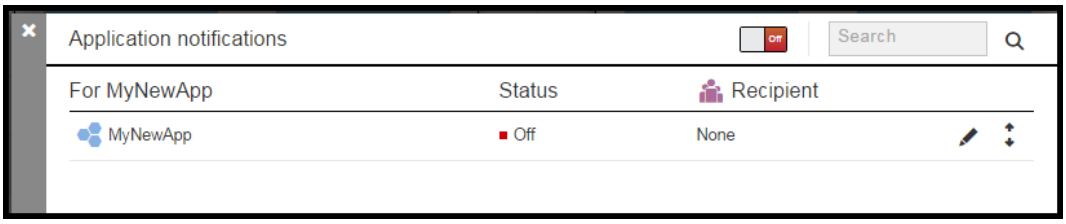
Configuring Email Notifications in Application or Microservice Processes

Review these guidelines before configuring email notifications:

- You can configure notifications for application or microservice processes and application or microservice process steps (type: process).

  When you open the Applications notifications or Microservices notifications dialog box and expand the process details, only the process steps (type: process) appear in the list.

- New email notifications are disabled in the application, its application processes, and the process steps before you configure them.



- You configure notifications in the "Application notifications / edit" dialog box.

  Important: The first time that you set notifications in this dialog box, the Notifications toggle changes to **On**. After you enter notification settings and click **OK**, email notifications are enabled at that level.

By default, the application expects that the user creates an email configuration called "default." The email configuration defaults to the server property `/server/ec_deploy/ec_defaultEmailConfiguration`, which is set to "default."

If you want to use a different name for the email configuration, change the value of `/server/ec_deploy/ec_defaultEmailConfiguration` to the new email configuration name.

Starting from the Home page (https://*<ElectricFlow_server>*/flow/), to set email notifications:

1. Go to the Applications List.
2. Select an application.

   The Application Editor opens.
3. Click the **Menu** button.

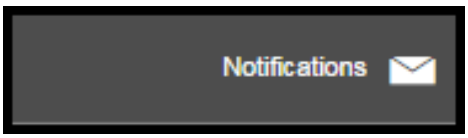
   ## Example:

   The Applications menu opens.

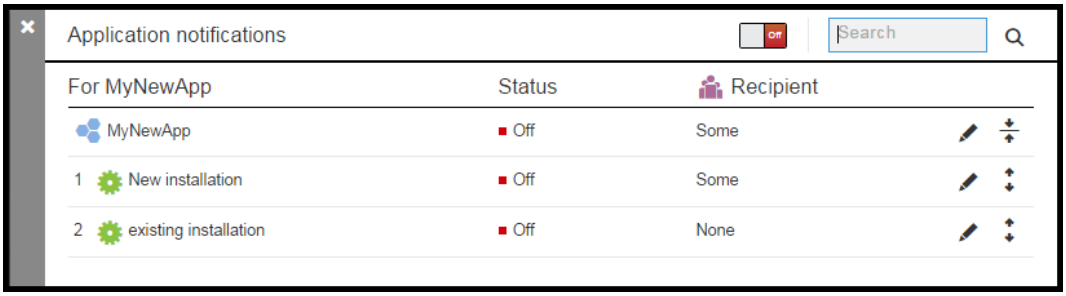
4. Click **Notifications** to add a new application.

Example:



The **Application notifications** dialog box opens.

Example:



5. Configure email notifications as follows:

   You can configure one or more notifications in an application process or an application process step (type: process).

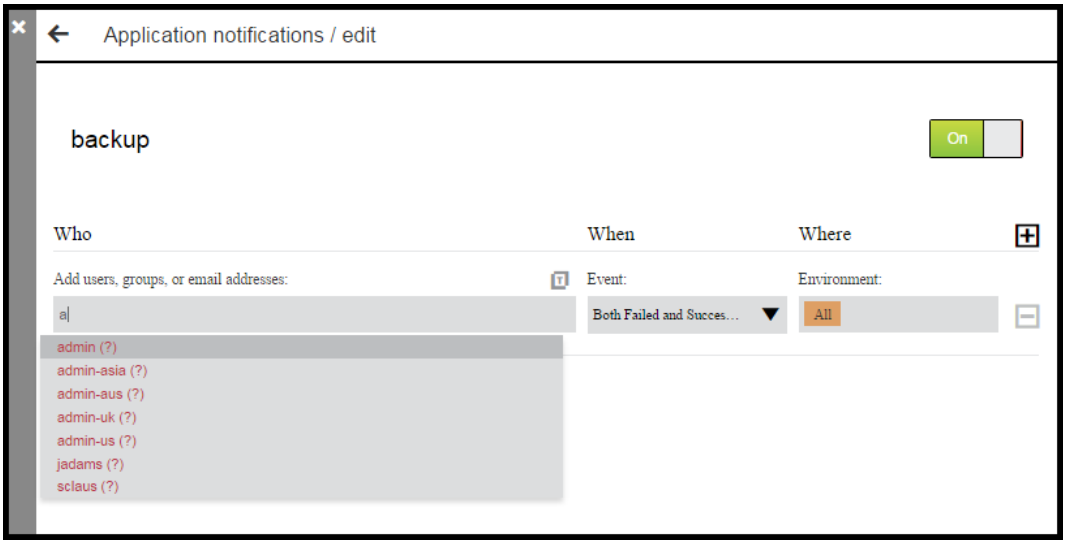
   ### Configuring recipients

   In the **Who** field, add users, groups, or email addresses.

   When you start typing a user name, group name, or email address, a list of names or email addresses appear that match what you are typing.

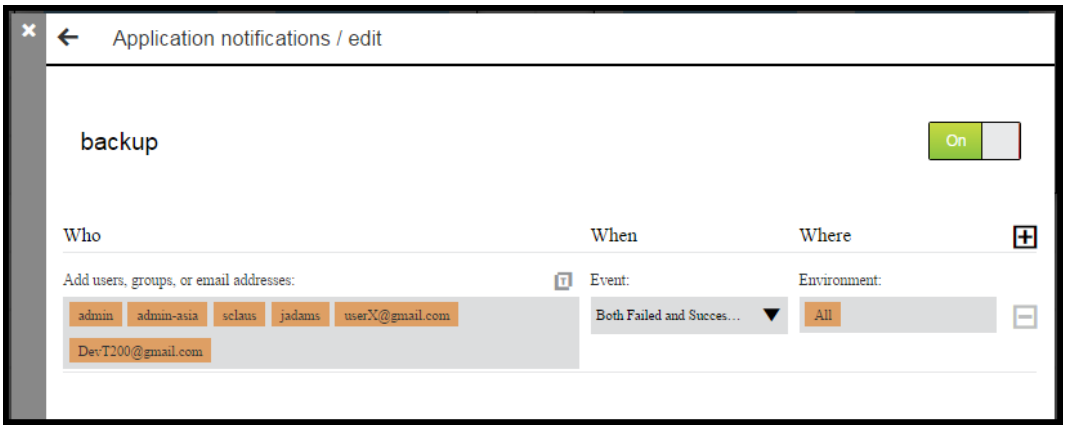
   > **Note:** The user and group settings are defined and managed at the platform level. See Users and Groups on page 1308for more information.

   ## Example:



   If one of the suggestions matches the name or email address, select it or continue typing. You can add more than one name or email address.

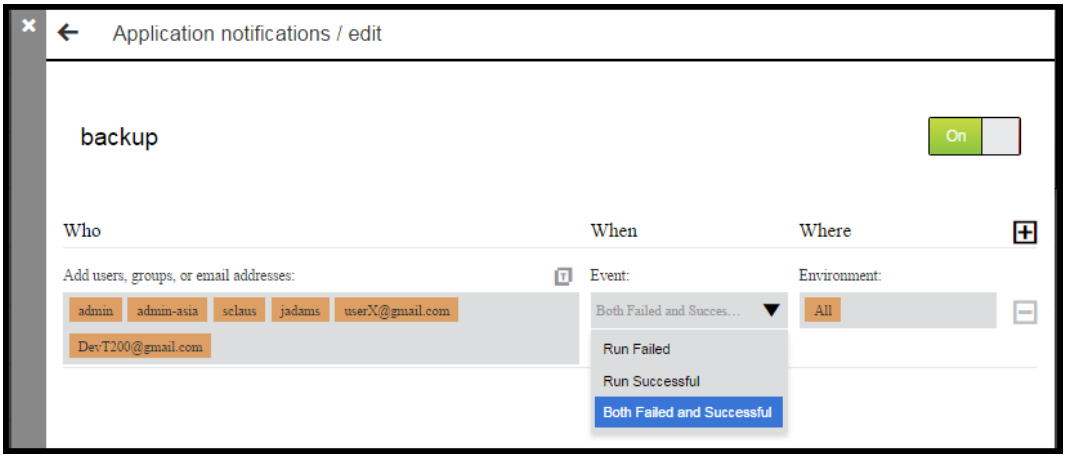
   ## Example:



Configuring the event that triggers the notification

In the **When** field, select the event that triggers a notification to be sent to the recipients in the **Who** field. The default is **Both Failed and Successful**. Clicking in the **When** field shows a list of event triggers.

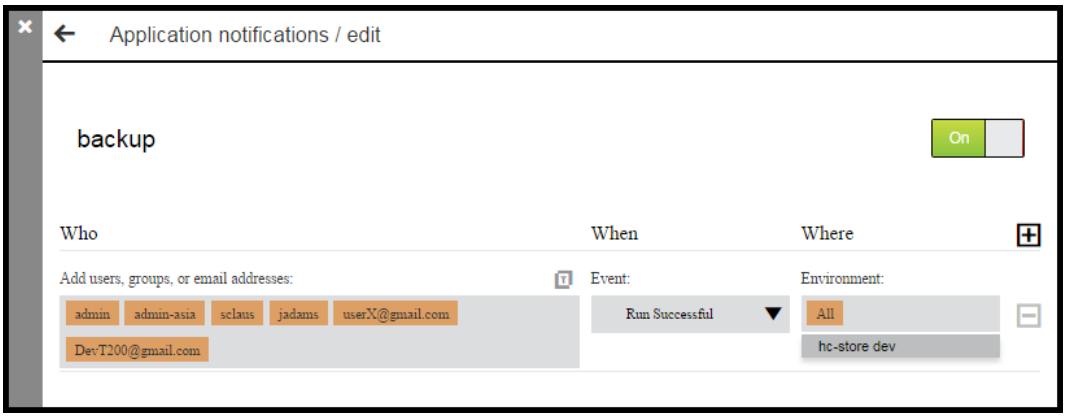
Example:



Configuring the environments where the notification applies

In the **Where** field, you select the environments to which the notifications apply. Clicking in the **Where** field shows the list of available environments to which the application is mapped in the tier map.

Example:



- Select and edit the email notification message.
- Add another notification for an application process or application process step.

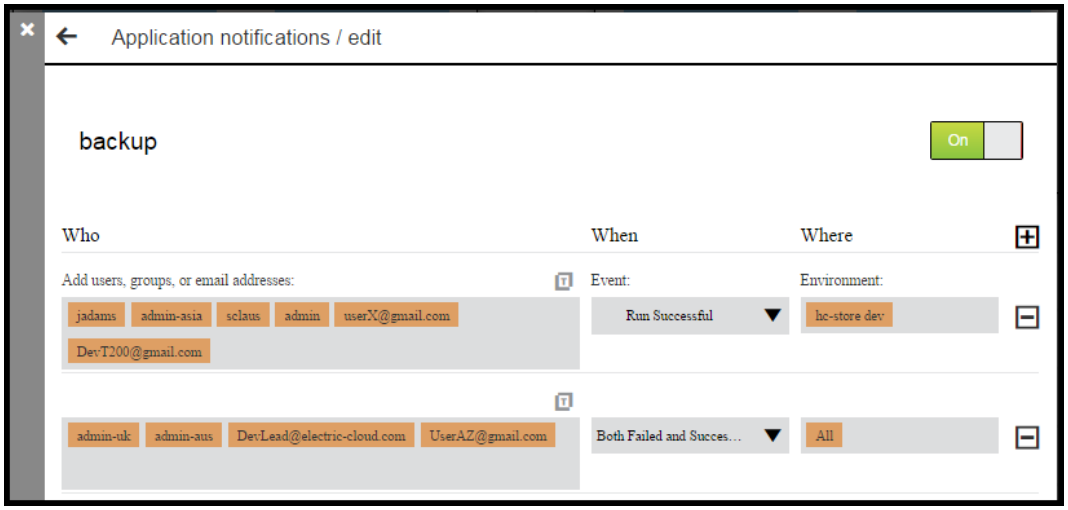
  Click the **Add Notifications** button to add a new notification.

  ## Example:

  After you have added your email notifications, click **OK** to save the settings and return to the **Application notifications / edit** dialog box.

  ## Example:



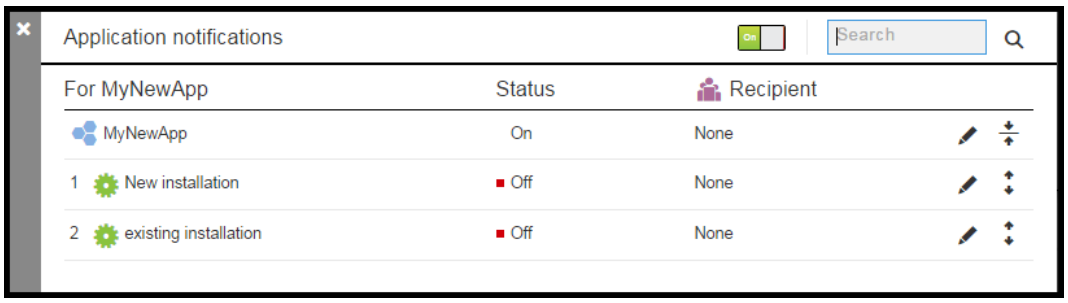
- (Optional) Enable email notifications for the application processes or application process steps that are not already enabled.

  To enable email notifications at the application level:

  - Click the Notifications toggle and change it to **On**.

    The status of the application changes to **On**.

    Example:



  - Click the **Edit** button to open the **Application notifications / edit** dialog box.

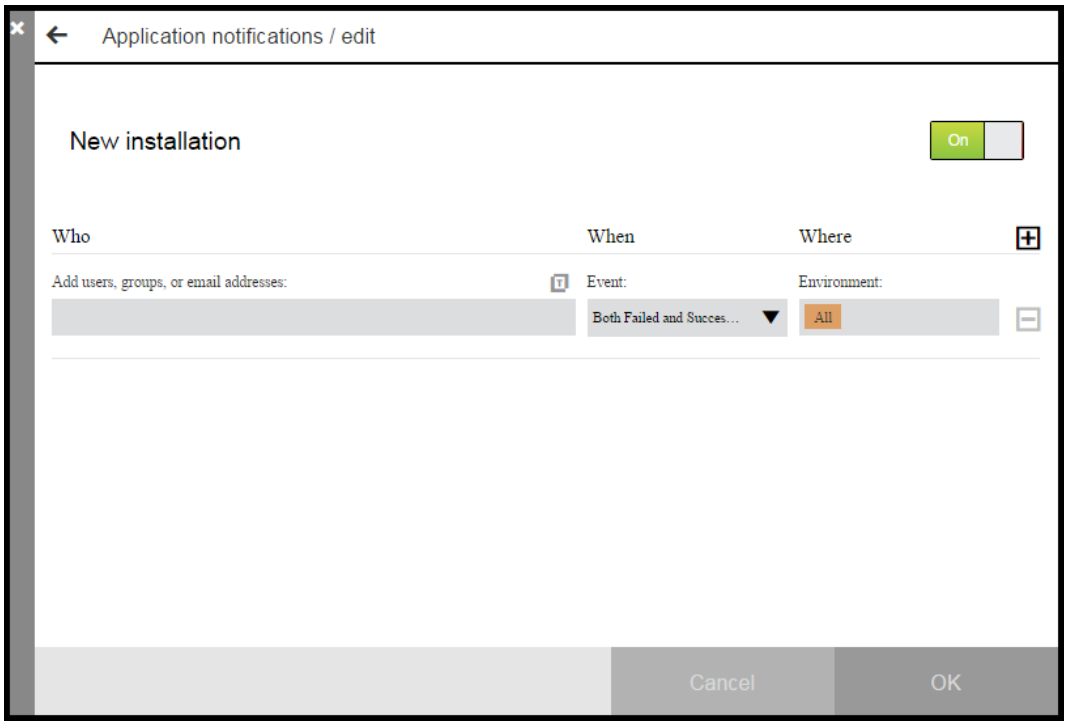
    The **Application notifications / edit** dialog box appears. The Notification toggle changes to **On**.

    Example:

  To enable notifications at the application process and process step levels, go to the **Application notifications / edit** dialog box for the specific process or process step.

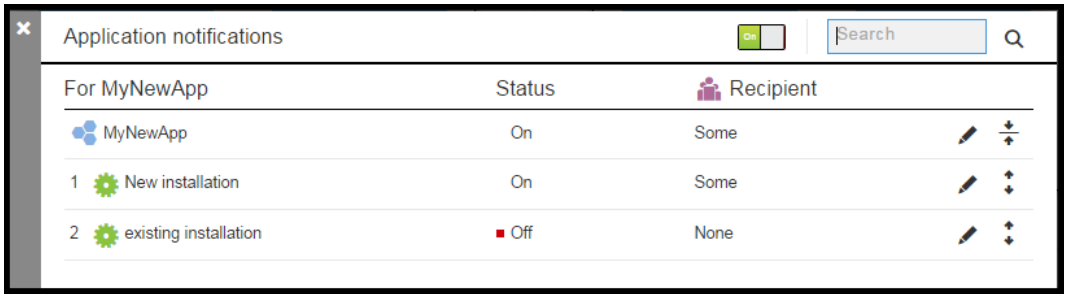
  The dialog box opens, and the Notifications toggle is now **On**.

  Example:



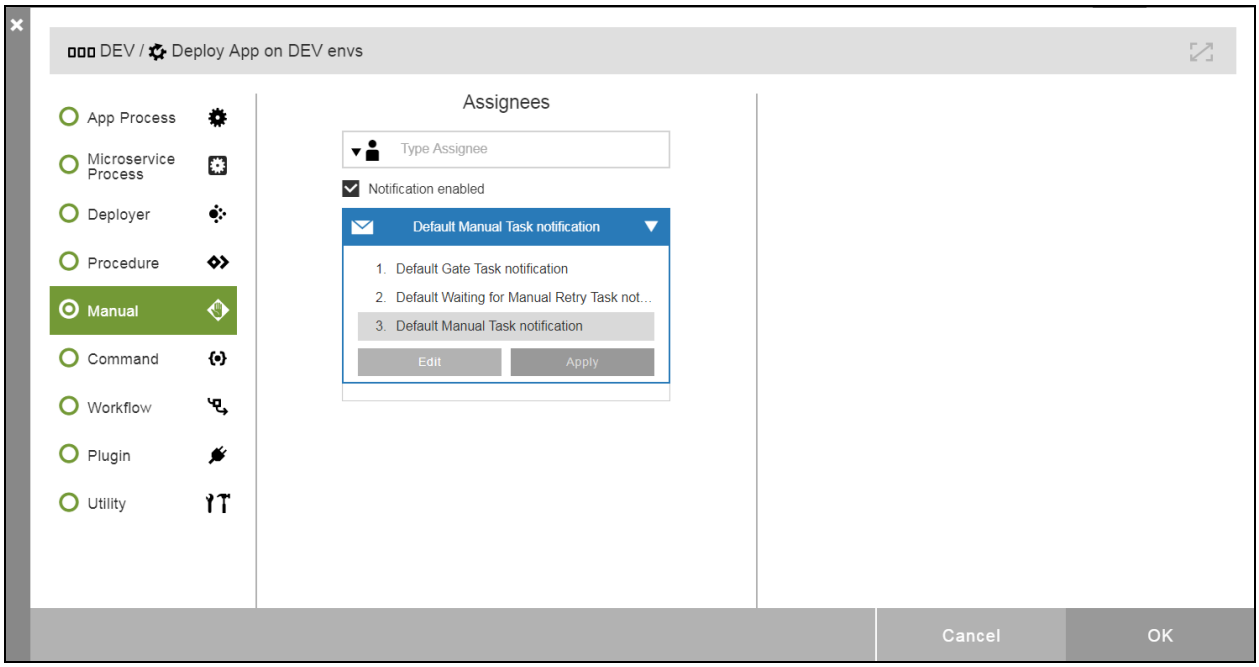
When you enter notification settings in the dialog box and click **OK**, the settings are saved. The **Application notifications** dialog box appears and now shows that the application process status is **On**.

Example:



Enabling or Disabling Email Notifications in Pipeline Gate Tasks or Manual Tasks

You can enable or disable email notifications for pipeline gate tasks or manual tasks by using the **Notification enabled** checkbox in the pipeline task editor:



Selecting and Editing Email Messages for Application Processes

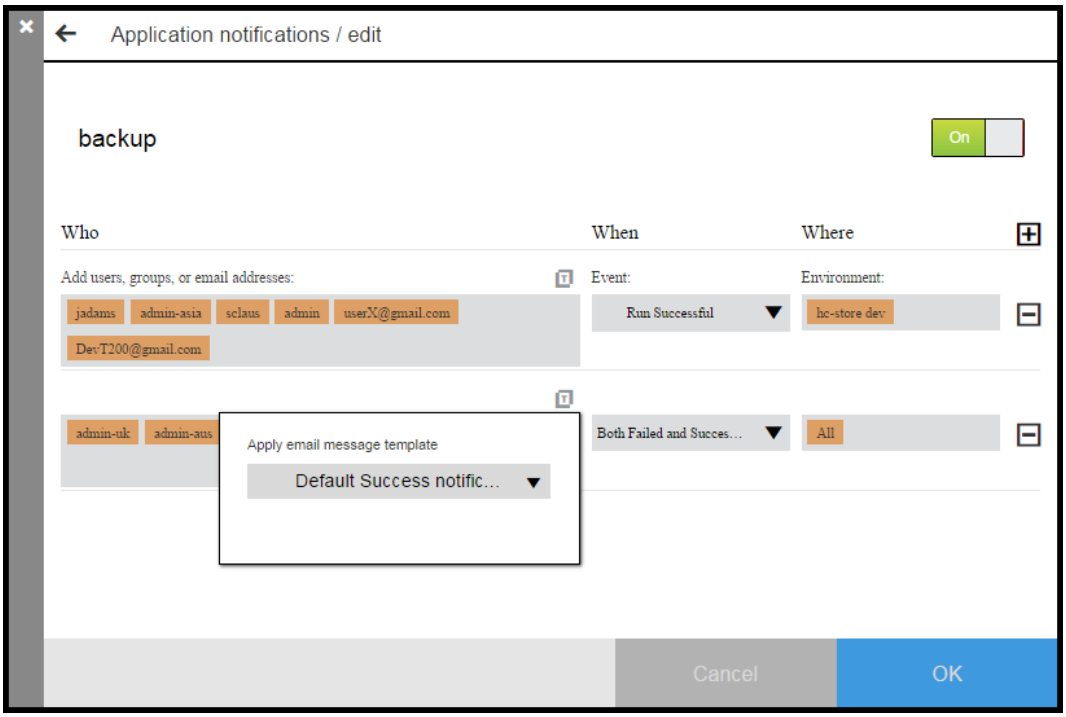
You can edit the email messages sent as notifications for application processes and application process steps (type: process).

Starting in the **Application notifications / edit** dialog box:

1. Click the **Template** button.

   A drop-down box opens.

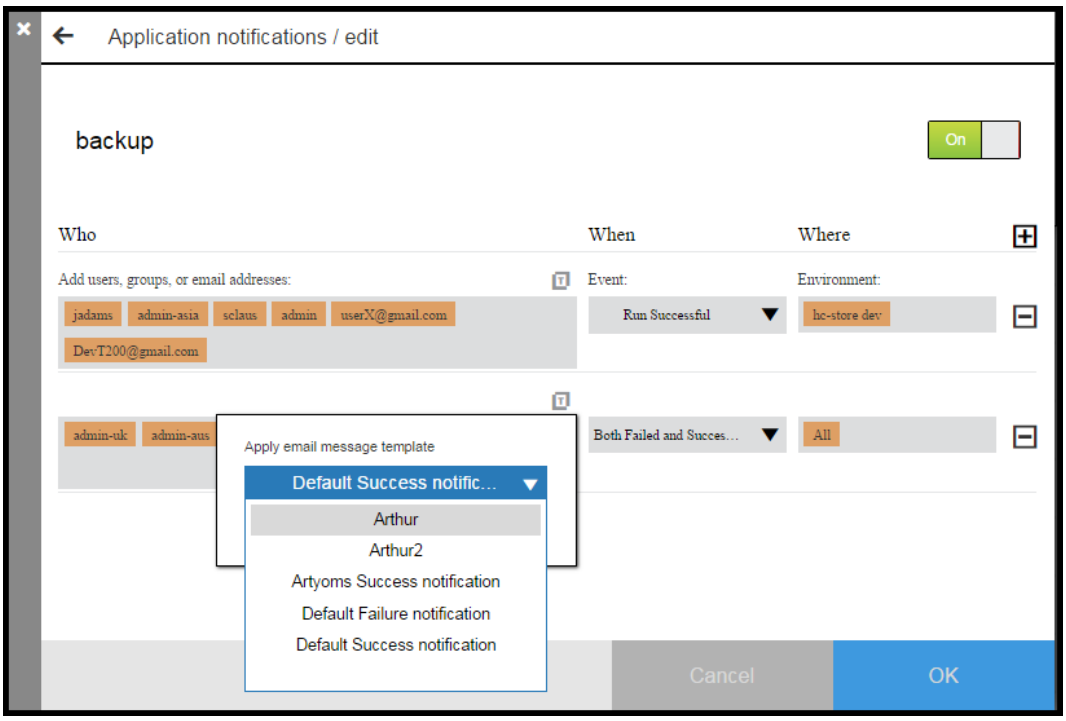
Example:



2. Click the down arrow to open the list of email message templates that can be applied to the application.

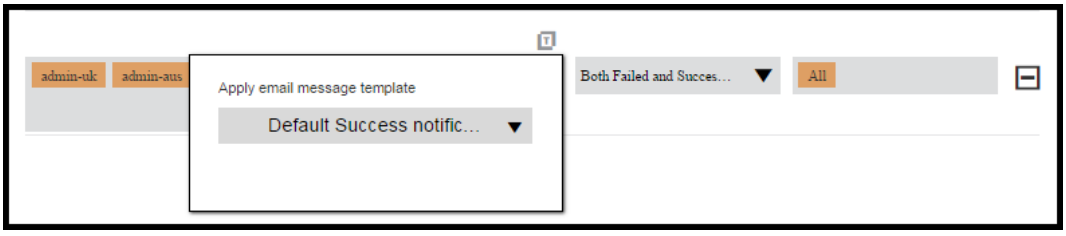
3. Select a template.

Example:



If the template is the current template applied to notification, the name of the template appears in the dialog box.

Example:



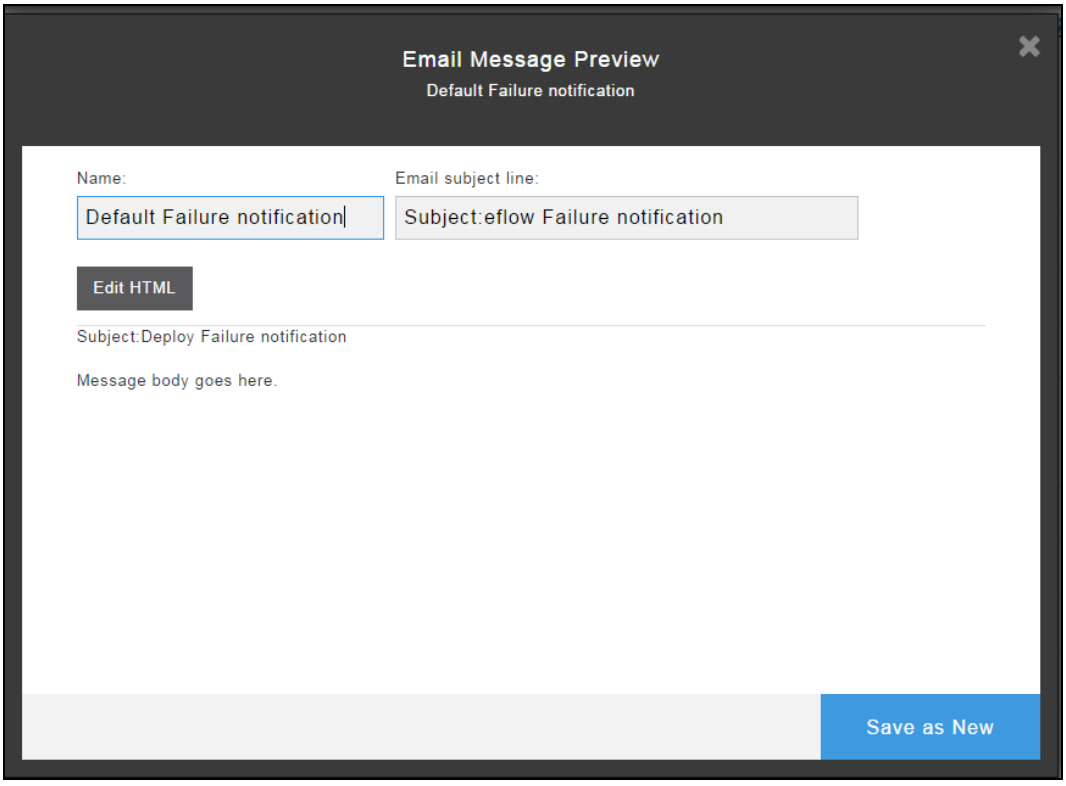
If the template is not the current template, the **Apply** and **Edit** buttons appear in the dialog box.

Example:

4. If you want to use the template that you selected instead of the current one and do not want to change the selected one, skip the remaining steps in this task.
5. If you want to apply a different template or edit the template that you selected, do the remaining steps in this task.
    - If you click **Apply** to use the template as-is, skip the remaining steps.
    - If you click **Edit** to modify the template, go to the next step.

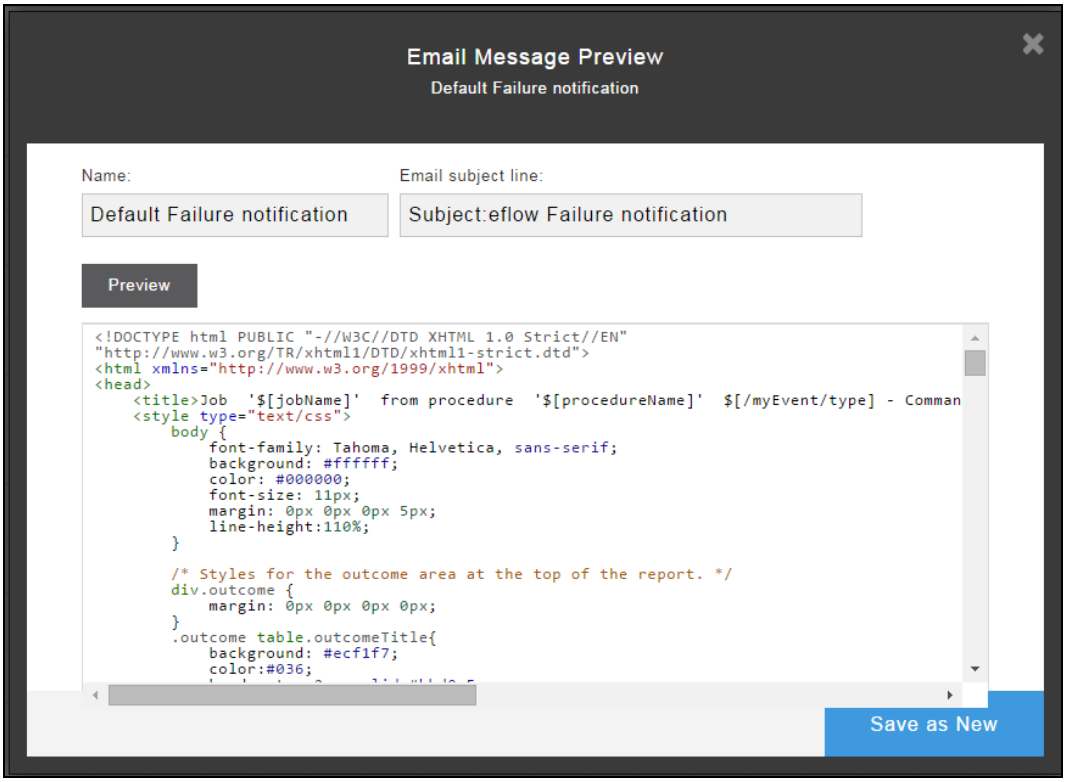
    The **Email Message Preview** dialog box appears.

Example:



6. To edit the template:
    1. Change the name of the template in the **Name** field.
    2. Change the subject of the email in the **Email subject line**.
    3. To modify the body of the email message, click **Edit HTML** and edit the HTML code.

    Example:



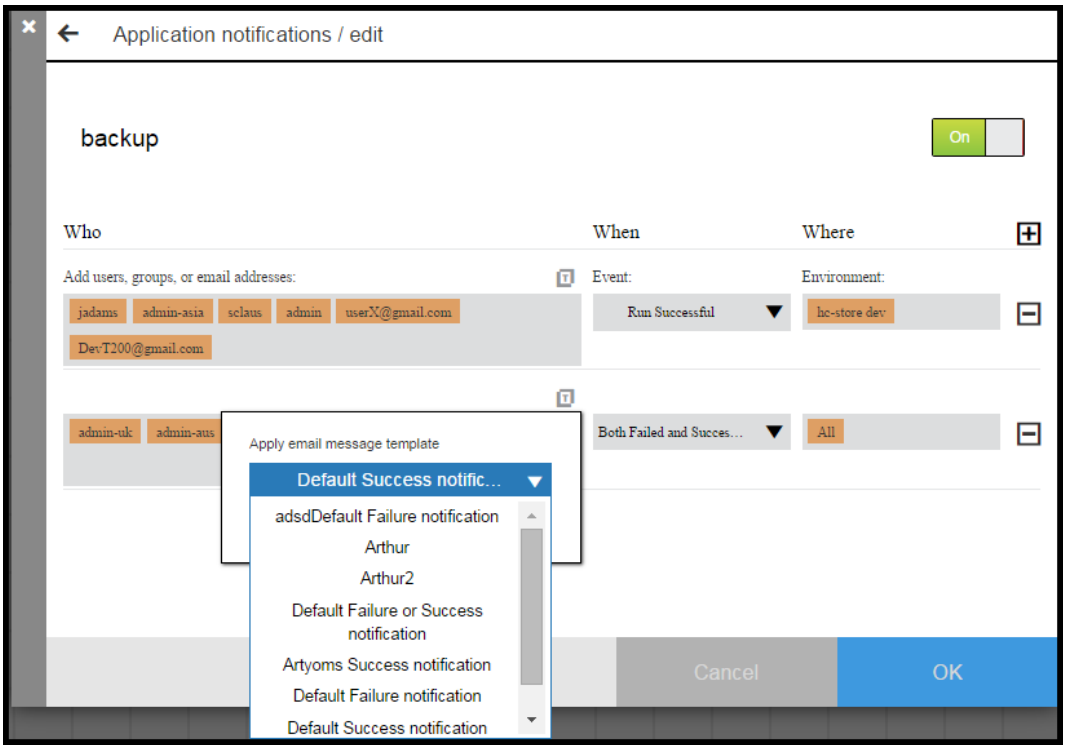
7. Click **Preview**.
8. To save your changes:
    - Click **Save Changes** to save the changes in the existing template
    - Click **Save as New** to save the template as a new template.

    The **Application notifications / edit** dialog box re-appears.

9. Click the down arrow to open the list of email message templates that be can applied to the application.

   The new email message template is in the list.

   Example:



10. Click **OK** to save the settings.

Chapter 9: Automation Platform

ElectricFlow is built on a powerful proven automation platform that natively integrates domain-specific capabilities for Enterprise-level continuous delivery. The automation platform gives distributed DevOps teams shared control and visibility into infrastructure, tool chains, and processes. It accelerates and automates the software delivery process and enables agility, availability, predictability, and security across many build-test, deployment, and release pipelines.

This section describes how to create, configure, and manage objects in the automation platform that make the automation of build-test processes, application or microservice deployments, and pipelines possible in ElectricFlow. Examples include:

- For build-test automation: Configuring resources and creating procedures to build and test your software.
- For deployment automation: Creating components based on artifacts, defining process steps with plugins, and assigning resources to environments.
- For pipelines: Defining tasks based on procedures, workflows, or plugins.
- For release management: Defining releases based on pipelines.

You use only the automation platform UI to configure ElectricFlow to perform these tasks. The objects and operations for build-test automation are the same as those for the automation platform.

You can also use Perl API commands through ec-perl and ectool, REST API commands, and DSL scripts. For information about using API commands and DSL scripts, see the *ElectricFlow API Guide* at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html.

Build-Test Automation

Before you use ElectricFlow for build-test automation, deployment automation, pipelines, or release management, you must create, configure, and manage these objects in the automation platform:

For build-test automation, you must create, configure, and manage these objects in the automation platform:

- Projects—A *project* is an object used in ElectricFlow to organize information. A project is a container object for procedures, steps, schedules, workflows, and properties. If you use ElectricFlow for different purposes, you can use a separate project for each purpose so different projects do not interfere with each other. When you work in one project, you do not normally see information in other projects. At the same time, a project can use information defined in other projects, which allows you to create shared library projects.
- Resources—A *resource* is defined as an agent machine where steps can execute. A resource has a logical name and a host name. In some situations, it is convenient to have multiple logical resources associated with the same host. A resource can also be associated with one or more pools. Each resource has a *step limit* that determines the maximum number of steps that can execute simultaneously on the resource. Resources can be grouped into *resource pools*. Multiple resources can be defined on the same machine.

- Procedures—*Procedures* and *steps* define tasks that you want ElectricFlow to execute. A procedure consists of one or more steps. A step includes a command or script executed on a single resource and is the smallest unit of work that ElectricFlow understands. Each step specifies a resource on which it should run (either the name of a specific machine or the name of a *resource pool* of equivalent machines, in which case ElectricFlow picks a machine from the pool). A step can be given a time limit, and if the step does not complete within the specified time, ElectricFlow automatically aborts it.

  Steps are ordered within a procedure and normally execute sequentially. However, it is possible to mark a consecutive range of steps for parallel execution, so all steps in that range run concurrently.

  You can define *parameters* for procedures. Parameter values are assigned when procedures are scheduled. Parameters can be required, optional, or have default values. Parameters are used for a variety of purposes such as specifying the branch to build or the set of platforms on which to run tests. Parameter values can be used in step commands and many other places.

  Procedures can be nested. A step in one procedure can invoke another procedure instead of running a command. The invoking step provides parameters needed by the nested procedure, also referred to as a subprocedure.

- Schedules—A schedule is used to execute procedures and determine when specific procedures run. A schedule can trigger at defined times, for example, every 2 hours from 10:00 pm to 6:00 am on Mondays, Wednesdays, and Fridays, or when modifications are checked into a particular branch of your source code control system. It is also possible to create a schedule that runs immediately and disappears after the job runs. When you create a schedule, you must provide the parameters required by the procedure that you want to invoke.

  The Continuous Integration Dashboard works with your source code management (SCM) system and provides visibility into running builds, the ability to add a project to continuous integration quickly, and easily accessed configuration pages to setup or modify a continuous integration schedule.

- Workflows—Managing a build-test-deploy product life cycle spanning multiple procedures and projects requires a significant amount of "meta-programming" and a heavy use of properties, and the *workflow* feature simplifies this process. Using the workflow object, you can create build-test-deploy life cycles by defining a set of states and transitions. Any ElectricFlow project can contain a workflow.

- When a procedure is executed or run, a *job* is created. A job is an object that is created each time a procedure begins to execute or run. The job keeps track of all data associated with the procedure's execution, such as the running time of each step and any errors that may occur during the step. ElectricFlow retains job information after the job completes so you can examine what occurred.

After setting resources, procedures, and schedules, ElectricFlow automatically runs the procedures that you created using these objects and facilities:

- Zones and Gateways—A zone (or top-level network) that you create is a way to partition a collection of agents to secure them from use by other groups. A gateway is a secured connection between two zones when you want to share or transfer information between the zones. For example, you might want a developers zone and a test zone. The ElectricFlow server is a member of the *default* zone, created during ElectricFlow installation.

- *Continuous Integration Builds* and other schedules—Run jobs according to *schedules* that you define. Scheduled jobs can run at specific times or when source code changes are checked in to your source control system. ElectricFlow integrates with major source control systems. The Continuous Integration Dashboard allows you to add more projects easily and create build configurations quickly so you can visually see running builds, build status, and so on.
- *Artifact Management* functionality—Using artifacts can improve performance across builds, provide better reusability of components, and improve cross-team collaboration with greater tractability. For example, instead of developers repeatedly downloading third-party packages from external sources, these components can be published and versioned as an artifact. Developers then simply retrieves a specific artifact version from a local repository, guaranteeing a consistent package from build to build.
- *Preflight build functionality*—Used by developers to build and test code changes in isolation on their local machines before those changes are committed to a production build.
- *Plugin* capability—ElectricFlow is built with an extensible UI, enabling easy development of plugins that include integrations with other tools, custom dashboards, and unique user experiences based on roles. "Bundled" plugins, installed during ElectricFlow installation, provide easy integration with your SCM systems, defect tracking applications, and so on.
- *Workflow* functionality—Use a workflow to design and manage processes at a higher level than individual jobs. You can use workflows to combine procedures into processes to create build-test-deploy life cycles (for example). A workflow contains states and transitions that you define to provide complete control over your workflow process. The ElectricFlow Workflow feature allows you to define an unlimited range of large or small life cycle combinations to meet your needs.
- *Resource* management—If a resource is overcommitted, ElectricFlow delays some jobs until others are finished with the resource. You can define pools of equivalent resources, and ElectricFlow spreads usage across the pool.
- Recording a variety of information about each job, such as the running time and the success or failure of each step. A set of reports is available to provide even more information.
- Powerful and flexible *reporting* facilities—Various statistics such as number of compiles or test errors are collected after each step and recorded in the ElectricFlow database. A variety of reports can be generated from this information.
- Allowing you to observe jobs as they run and to cancel jobs.
- Credentials—Use a credential, consisting of an user name and password, can be attached directly to a step or schedule at the platform level. You can also attach impersonation credentials to procedure steps, procedures, and projects before executing the job step. It allows ElectricFlow to use a specific account with special privileges on a per-job or per-step basis.
- *Workspace* for each job, which is a disk area a job uses for storage—ElectricFlow also provides a facility for reclaiming space occupied by workspaces.
- Powerful data model based on *properties*—Properties are used to store job input data such as the source code branch to use for the build, to collect data during a job (such as number of errors or warnings), and to annotate the job after it completes (for example, a build has passed QA).
- *Access control* for users logged into the system—ElectricFlow uses this information to control their activities and integrates with Active Directory and LDAP repositories.
- *Search, sort, and filter* functions to minimize viewing or "wading" through information that is of no interest to you, allowing you to access to the information you need quickly.

- *Email notifications* to get important information or data to individuals or groups immediately and on a regular basis for a particular job or a specific job aspect.
- All ElectricFlow operations and features are available from a command-line application tool (Perl API), *ectool*, the REST API, DSL methods, and a user interface (UI).

For more information, see Automation Platform Objects and Functionality on page 1307.

Getting Started

Overview

ElectricFlow is an application for automating and managing your development life cycle, the software build, test, and deployment process. It helps development teams make these tasks repeatable, visible, and efficient.

ElectricFlow creates an environment where IT and Development organizations can work together to connect physical and virtual environments, processes, and tools already in use to create a private development cloud, also known as a *smart development cloud*. ElectricFlow is a scalable solution, solving some of the biggest challenges of managing "back-end" software development tasks.

Automation Platform Terminology

To get an overview of how the automation platform works and understand the ElectricFlow concepts and processes, review the following basic terms:

- Project
- Procedure
- Job Step
- Plugins
- Parameter
- Schedule
- Continuous Integration
- Resource
- Job
- Workspace
- Zones
- Gateways

See the glossary for a list of ElectricFlow terminology.

Getting Started Scenarios Help Topic Series

The series of scenario Help topics are designed to help you learn to use the ElectricFlow automation platform quickly. Follow the step-by-step scenario format to learn the automation platform basics:

- Creating a simple procedure—Scenario 1
- Creating a procedure that uses a Source Code Management (SCM) system— Scenario 2
- Setting up email notifications, reporting, and scheduling—Scenario 3
- Creating a multi-agent build procedure—Scenario 4

Each scenario builds on what you learned in the previous scenario, and each scenario ends with one or more scenario extensions to introduce you to other related ElectricFlow functionality.

Navigating the Automation Platform User Interface

To quickly link to task pages, view running or completed jobs, configure resources and so on, the automation platform web interface displays tabs across the top of the page. These tabs remain available at the top of all ElectricFlow automation-platform web pages.

When you select some of the main tabs, subtabs are provided. For example (see the next screen example), when the **Home** tab is selected, notice that the Overview tab is in bold font, which means you are looking at the Overview page. This page can be customized to show the information that you want to see quickly. You can use this page for shortcuts, quick links to jobs, "thumbnail" report views, and so on.

Selecting the Continuous Integration subtab takes you to the Continuous Integration Dashboard, which is place to configure your projects for continuous integration builds.

In addition to the main set of tabs, the top bar includes:

- **Logged in as** clearly shows the name of the logged-in user.
- **Logout**—Use this link to log out between ElectricFlow sessions if necessary.
- **Help**—This link provides a Help topic for the current page you are viewing, but if you select the down-arrow adjacent to the Help link, you have access to the following:
  - **Tutorials**—Use this link to display the Table of Contents for all currently available tutorial examples you may find useful as you become familiar with ElectricFlow .
  - **Documentation**—Use this link to access the entire ElectricFlow online Help system, which is fully searchable in the event you do not quickly see what you are looking for in the left-pane Table of Contents.

    The Help Table of Contents contains feature overview/concept user-guide style Help topics, which are not linked to a particular web page. These topics are listed in the Table of Contents above the "Web Interface Help" section. The Web Interface Help section contains all of the page-specific Help topics.

    > **Note:** *To print a Help topic*— Right-click your mouse in the Help topic pane or go to your browser File menu.

  - **Visit the Community Site**—This link takes you to the Electric Cloud Knowledge Base, all product documentation, and so on.

- **Contact Support**—Use this link to access the Electric Cloud Technical support phone number and email address. At a later time, you may choose to reconfigure this link to redirect it to your internal support time. See Reconfiguring the "Contact Support" Link on page 1029 in Customizing the ElectricFlow UI.
- **About**-Use this link to display the current version of ElectricFlow you are using.

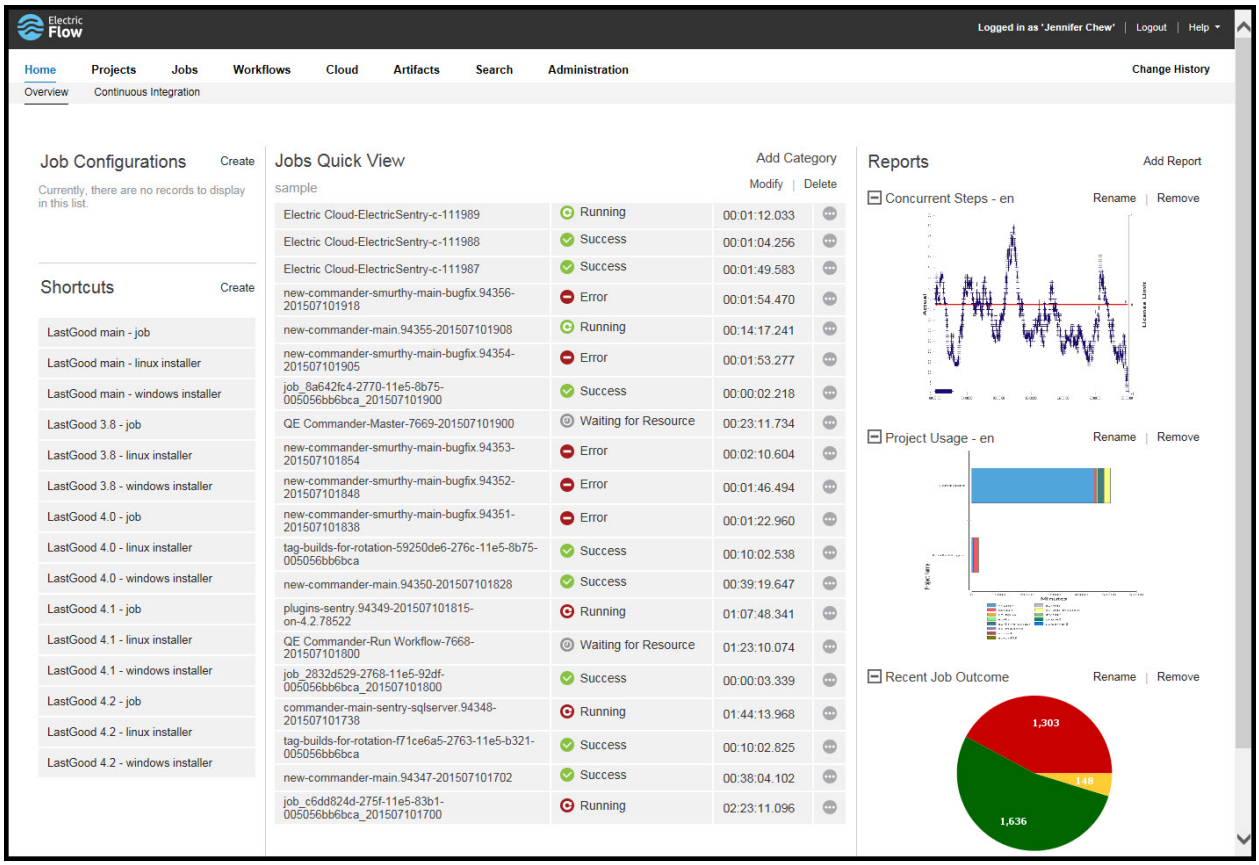
Home Tab

You can customize the Home page (https://<*ElectricFlow_server*>/commander/) to see the information you need at-a-glance or add Shortcuts to quickly return to the ElectricFlow pages you visit frequently. The Home page provides four categories:

- **Job Configurations**
  ElectricFlow procedures can contain complex sets of parameters, making it difficult to remember the correct parameters for a particular situation and tedious to re-enter those parameters every time the procedure is invoked. Job Configurations provide a one-click solution to this problem.
- **Shortcuts**
  Use shortcuts to save frequently visited ElectricFlow web pages, so those pages are immediately accessible. You can also create a shortcut to any page on the web.
- **Jobs Quick View**
  The Jobs Quick View allows you to define job categories that are interesting to you. You may be interested in a few jobs on this server. For example, you want only information about jobs that you launch manually and official builds for the products you work on. You do not want information about production builds for other products or personal jobs for other users.
- **Reports**
  You can configure reports you would like to see on a regular basis and display a thumbnail"report graphic in this section.

In the following Home page example, there are additional links. This page allows you to create, edit, and delete objects you see so that you can easily keep the page updated and current.

For more information about Home page functionality, see Automation Platform Home Page on page 1117.



Continuous Integration Subtab

The Continuous Integration Manager (CI Manager) provides a front-end user interface (the Continuous Integration Dashboard) for creating, managing, and monitoring continuous integration builds.

The Continuous Integration Dashboard provides:

- Visually seeing your running builds, build progress, build status, and accumulated build status.
- Easily accessed "Actions" to configure a continuous integration build.
- Quick configuration of your preferred SCM system.
- A project that contains any number of continuous integration builds, depending on the work you have already setup for your procedures and steps to perform.

The following is an example of the dashboard. Your initial view will be blank until you add configurations (using your preferred SCM) to run your build procedures continuously. When this tab is opened to view the dashboard, click the **Help** link in the upper-right corner of the page for instructions to begin using the dashboard.

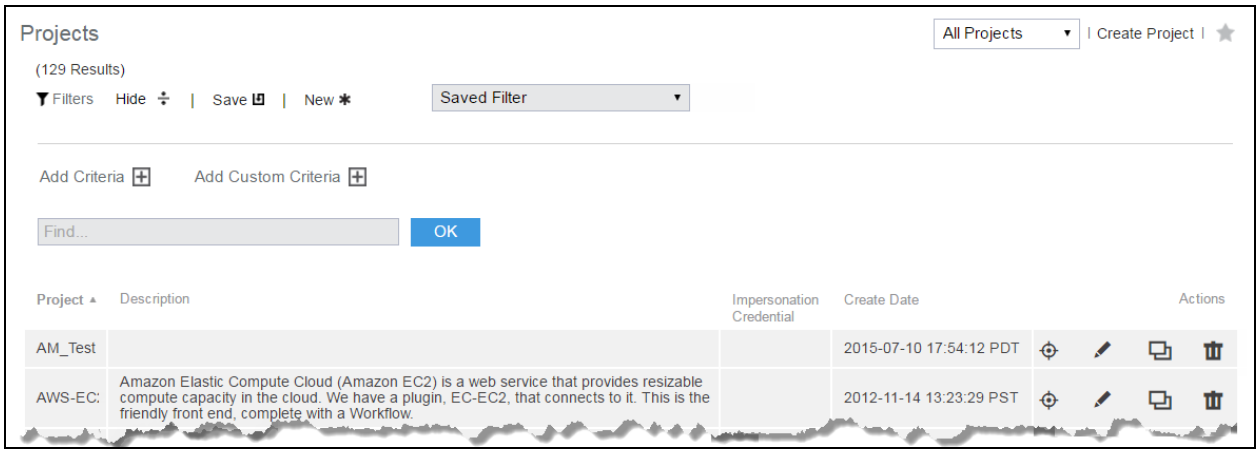
Projects, Jobs, and Workflow Tabs

Selecting the Projects, Jobs, or Workflows tab displays a table listing all previously configured projects, all completed jobs, or all previously configured workflows, respectively. Each table contains links in the table and links in the section above the table.

For example, selecting the Projects tab displays a table (see the next screen example), which has the projects you created and ElectricFlow default projects, which were added during ElectricFlow installation.

- Links at the top of the table include:
  - A "star" icon—Click this icon to add this page to your Home page.
  - A drop-down menu to choose which projects you want to see.
  - Create Project link to go to the New Project page to define and add a new project.
  - New Search link to find a project whose name you may not remember or projects of the same type.
- Links within the table include:
  - Click on a Project name (first column) to go to the Projects Details page for that project.
  - Actions column—This column usually contains links to Track Changes, Edit, Copy, or Delete the object, which is a project in this example.

Each table, Projects, Jobs, or Workflows, has similar links to help you get more information, modify existing information, or create a new object. For more information on any of these ElectricFlow pages, see these Help topics: Projects, Jobs, or Workflows.



Cloud Tab

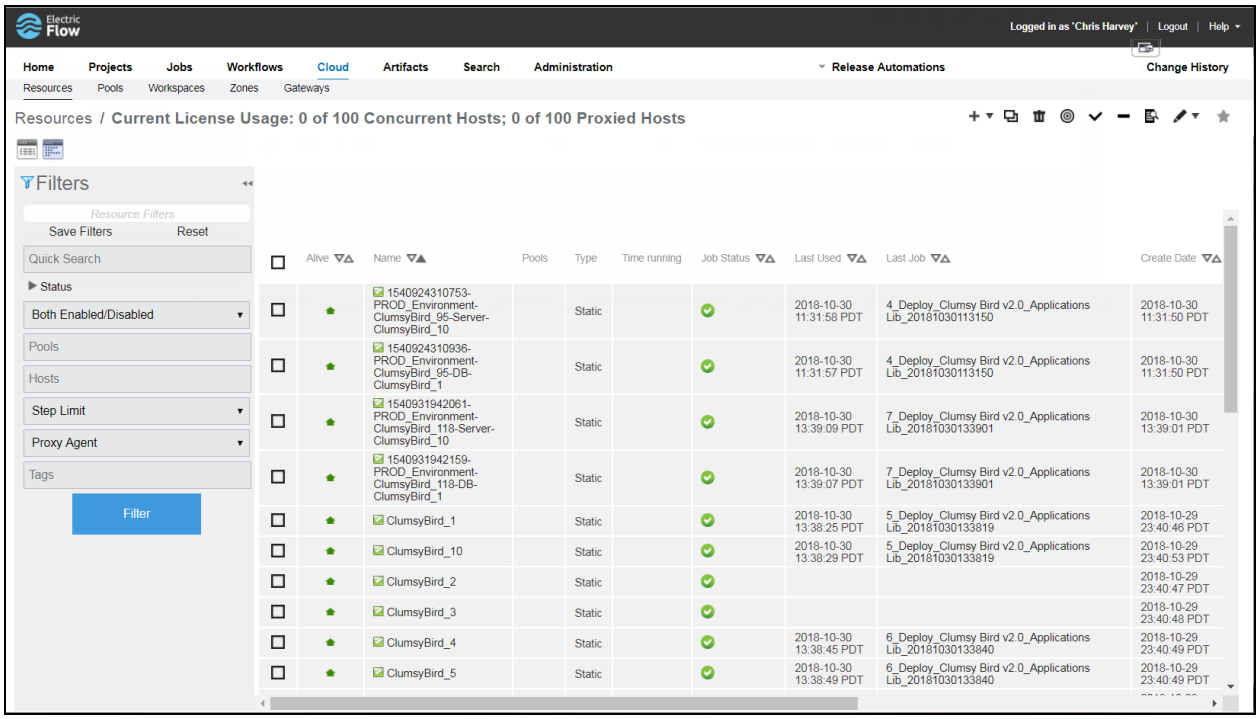
This tab opens to the Resources page and provides several other subtabs—Pools, Workspaces, Zones, Gateways, and the Cloud Manager plugin if it is installed.

Resources Page

The following example displays all resources that you configured ElectricFlow to use.

- In the information immediately after the **Resources** title at the top of the table, you can see how many licensed resources are in use.
- Use the row of buttons at the top of the page to perform resource management tasks.
- A resource name, in the **Name** column in the table, is a link to open the **Resource Details** panel for more information about the resource, and to access the **Edit Resource** panel to change or add access control privileges.

See the Resources Help topic for more information.



Pools Subtab

Click this subtab to display a list of all resource pools available to ElectricFlow. You may find it useful to group resources, creating one or more resource pools. For example, you can have resource pools for specific purposes, development teams, or other groups in your organization.

Similar to the Resources page, this page provides:

- A link to create a new resource pool.
- A Search link to find an existing resource pool.
- Summary information for each resource pool in the list.
- Clicking the resource Pool Name takes you to the Resource Pool Details page for more information about that pool.
- And the Action column contains Copy and Delete links.

For more information about pools, see the Resource Pools or Resource Pool—create new or edit existing pool Help topics.

Workspaces Subtab

Similar to the Resources and Pools pages, the Workspaces page displays a table all available workspaces and summary information for each workspace.

Available links are:

- **Create Workspace** for adding another workspace.
- **New Search** to find an existing workspace.

- Clicking the Workspace Name takes you to the Edit Workspace page to make modifications.
- And the Action column contains **Copy** and **Delete** links.

For more information, see the Workspaces and Disk Space Management, Workspaces, or Workspaces—create new or edit existing workspace Help topics.

Zones and Gateways Subtabs

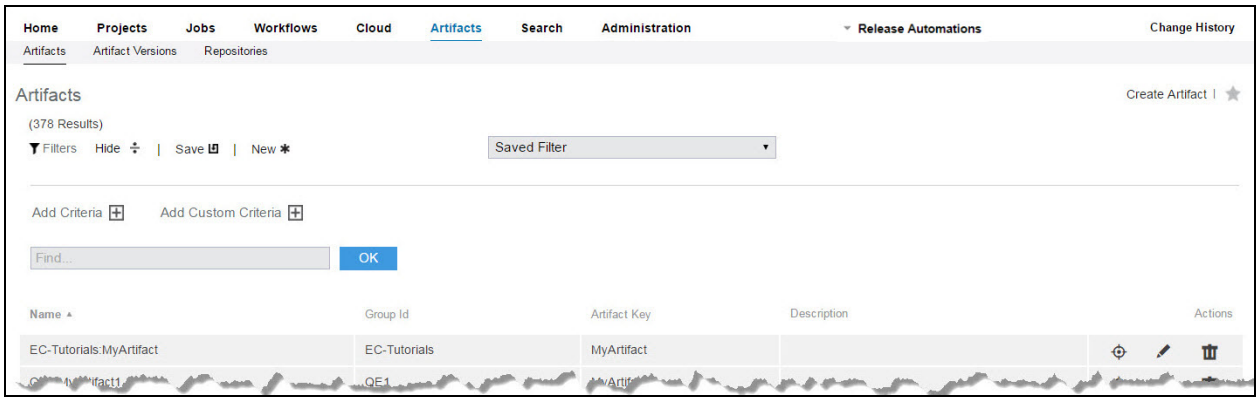
Similar to the other subtabs under the Cloud tab, the Zones and Gateways subtabs each have a table, listing all available zone or gateways, respectively.

- Selecting an object in the Name column takes you to the "Detail" page for more information.
- These pages are your management centers for zones or gateways, respectively.

For more information, see the Zones and Gateways Help topics.

Artifacts Tabs

This page displays all artifacts available on this ElectricFlow server. Clicking on an artifact Name (first column) takes you to the Artifact Details page. For more information, see the Artifact Management Help topic.



Artifact Versions Subtab

The Artifact Versions page displays all artifact versions available on the ElectricFlow server. Links in the table are the similar to other ElectricFlow pages. For more information, see the Artifact Versions Help topic.

Repositories Subtab

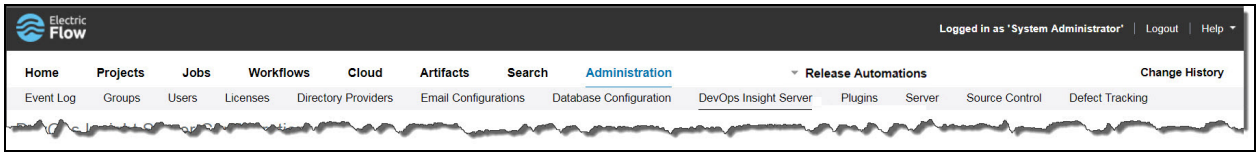
The Repositories page displays all artifact repositories available to the ElectricFlow server. For more information about repositories, see the Artifact Management and Repositories Help topics.

Search Tab

While many ElectricFlow web pages have a Search link, some do not. The Search page is always available. You can search any ElectricFlow supported object type on this page. For more information, see the Searching and Filtering Help topic.

Administration Tab

The next screen example shows all the current subtabs available when you select the Administration tab. Less-frequently used tasks are grouped for you as subtabs under the Administration tab. For example, after users and groups are set up, you do not need these pages unless a user or group configuration changes.



Administration subtab descriptions:

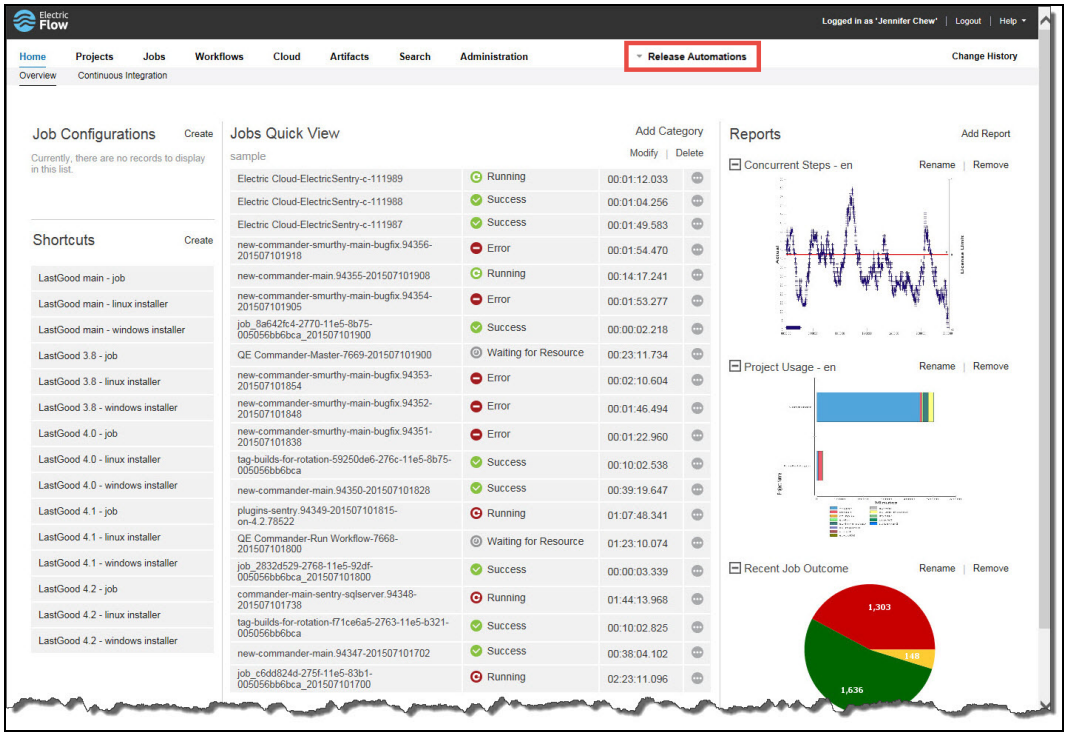
- Event Log—This page displays a log of events generated anywhere in the system, including jobs and workflows.
- Groups—Use this page to view a filtered list of ElectricFlow *local* users or groups.
- Users—Use this page to view a filtered list of ElectricFlow *local* users or groups.
- Licenses—This page displays all license information known to the ElectricFlow server. Typically, a single license is displayed, which describes the usage to which you are entitled. The license on the server may be based on concurrent resources, concurrent users, concurrent steps, registered hosts, or any combination of these licenses.
- Directory Providers—ElectricFlow uses account information from multiple sources. In most cases, the primary account information source is an external LDAP or Active Directory repository: both user and group information is retrieved from the repository. *Local* users and groups are defined in ElectricFlow.
- Email Configurations—This page displays all previously configured email configurations. You must create an email configuration so ElectricFlow can communicate with your mail server to send email notifications
- Database Configuration—If you do not use the ElectricFlow built-in (default) database, use this page to configure another database to communicate with ElectricFlow. See the *ElectricFlow Installation Guide* for a list of approved databases for use with ElectricFlow.
- DevOps Insight Server—Use this subtab to disable or enable DevOps Insight and configure connectivity and authentication between the DevOps Insight server and the ElectricFlow server.
- Plugins—This subtab displays the Plugin Manager page. A *plugin* is a collection of one or more features that can be added to ElectricFlow. Numerous plugins are bundled and installed with ElectricFlow, integrating third-party products seamlessly with ElectricFlow. For example, Source Code Management plugins are installed and so that you can configure your preferred SCM to communicate with ElectricFlow.

- Server—This page displays overall information about the ElectricFlow server. Use the Settings or Access Control links to make changes.
- Source Control—This page displays all SCM (source code management) configurations you have created to communicate with ElectricFlow.
- Defect Tracking—This page displays previously configured Defect Tracking systems known to the ElectricFlow server. Use plugins to integrate ElectricFlow with numerous defect tracking systems.

Go to Scenario 1.

Opening the Deploy UI from the Automation Platform UI

The top of the Automation Platform UI contains a **Release Automations** menu. This menu provides easy access to the main functionality of the Deploy UI: applications, microservices, environments, pipelines, and releases:



Getting Started—Scenario 1—Creating a Simple Procedure

To begin, select the Projects tab, then the **Create Project** link at the top of the table. You need to create a project to contain the procedures you create.

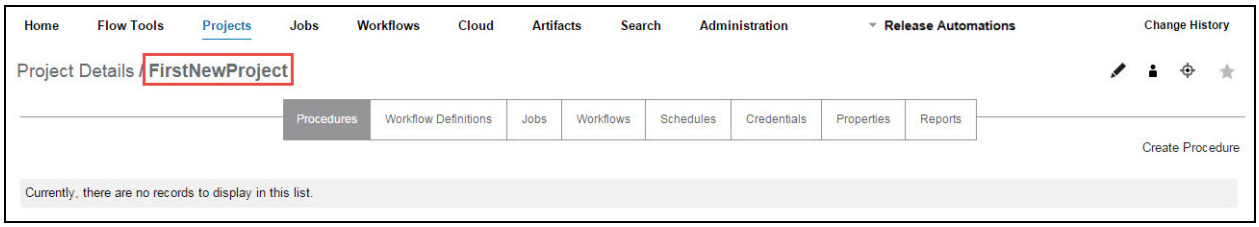
On the New Project page, enter information in the fields as follows:

Field Name	Description
Name	Type a unique project name. You may want your project names to reflect the work groups or teams that will be using them. For example, you might set project names based on the products they support. For our scenario examples, we will use "FirstNewProject" for our project name.
Description	Enter a text description for your reference if you choose. ElectricFlow does not use or interpret this information.
Default Resource	(optional) For the purpose of "getting started", leave this field blank to use the Default resource already created during the ElectricFlow installation.
Default Workspace	(optional) For the purpose of "getting started", leave this field blank to use the Default workspace already created during the ElectricFlow installation.

Click **OK** to save the information you entered.

After clicking **OK**, you see the Project Details page and the name of the new project adjacent to the page title.

- For our scenarios, the project name is FirstNewProject.
- Your new project name will appear on the Projects page also.



Scenario requirement: No additional requirements for this scenario. This scenario uses the *local* agent (resource) that was included on your local machine when ElectricFlow was installed.

Overview

This scenario establishes that ElectricFlow and its components were installed successfully and guides you through creating a simple procedure with a parameter that will echo Hello World. After you create this procedure, you will run the procedure and review the results.

At the end of this scenario, you will be familiar with the following ElectricFlow concepts and features:

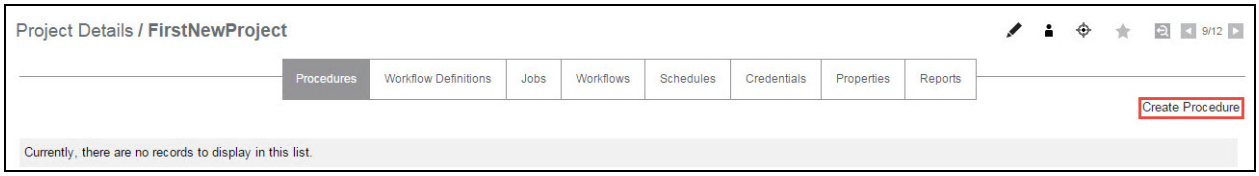
- Projects
- Procedures
- Steps
- Parameters
- Jobs
- Navigating the Job Details web page

Begin Scenario 1

Step1. Select the Projects tab

On the Projects page, you can see the default projects installed during the ElectricFlow installation and the new project you just created.

Select your new project name to go to its Project Details page, then click the **Create Procedure** link.



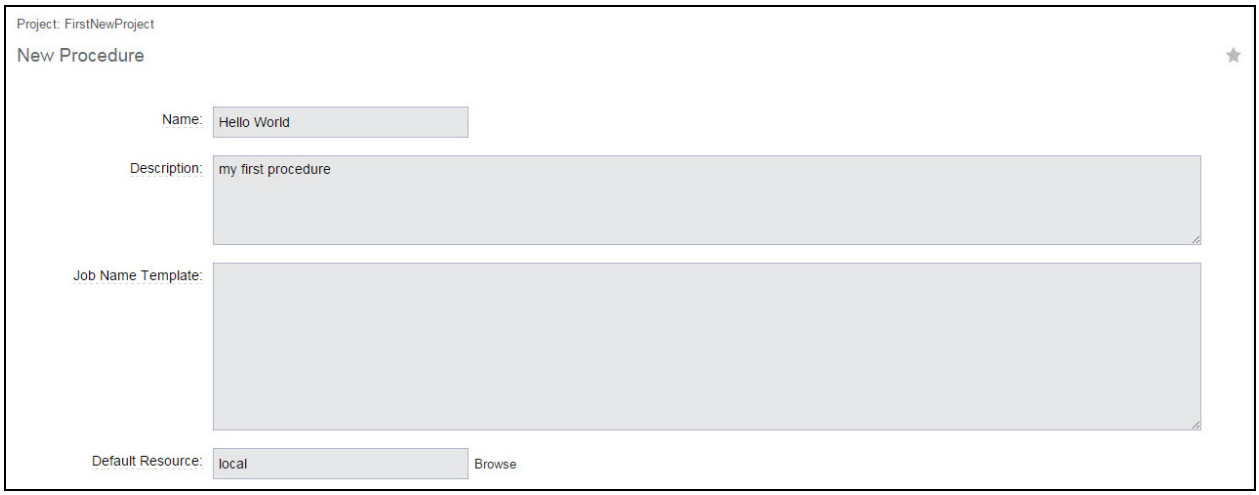
Step 2. Create a procedure

On the New Procedure page, you need to be concerned with the section of the page illustrated below only.

Enter the following information:

Field Name	Description
Name	For this scenario, `Hello World` is the procedure name. At a later time when you are creating your own procedures, you can choose any unique name that is most meaningful for procedures in your project.
Description	(optional) Enter a text description for your procedure if you choose to do so.
Default Resource	Use the Browse link and choose `local`. Using `local` accesses the agent installed during the ElectricFlow installation. At a later time after you have configured other resources, your Browse list could contain many choices.

Your New Procedure page will look similar to the following:



Before leaving this page, notice the "breadcrumb" in the top-left corner. Most ElectricFlow web pages contain a breadcrumb for easy return to the previous page.

Click **OK** to continue.

After clicking **OK**, ElectricFlow takes you to the Procedure Details page—this is the page you use to add important information to your procedure. When your procedure contains steps, parameters, or email notifiers, and so on, those objects will be displayed here for your reference.

Step 3. Create a parameter

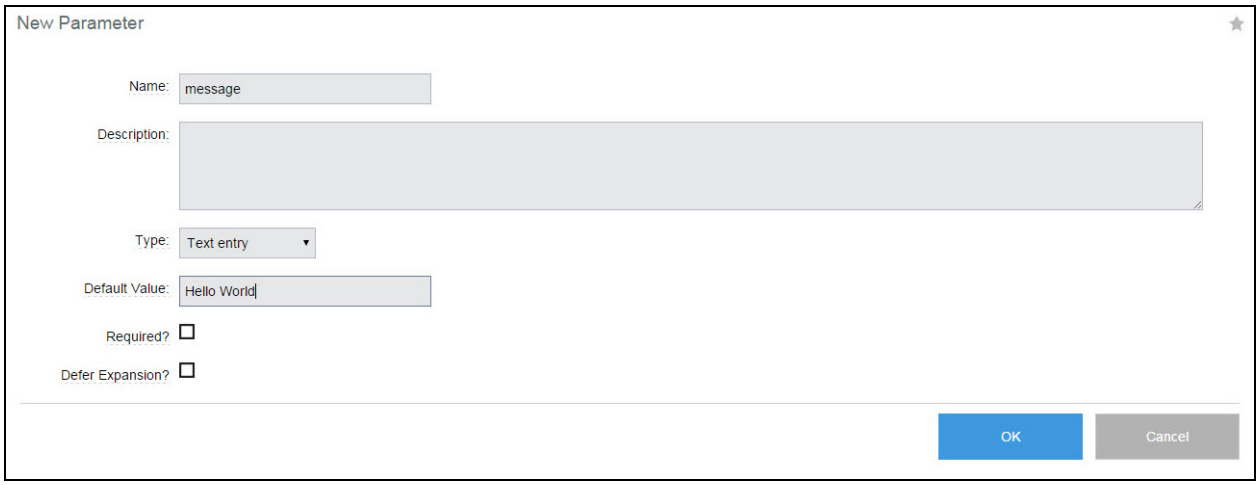
An ElectricFlow procedure can define one or more parameters. These parameters are similar to the parameters of a subroutine in a programming language. Parameters allow you to define a "reusable" procedure by using symbolic variable names in place of a single, fixed value. When you run the procedure, you can type in a value to use for that particular job. You can refer to the parameter in a step for a procedure, using the standard ElectricFlow "property reference syntax"—`$[...]`.
For example, see **Step 4**, specifically the text in the Command(s) text box.

On the Procedure Details page, click the **Create Parameter** link to go to the New Parameter page.

Enter the following information:

Field Name	Description
Name	Enter the parameter name `message` for this scenario.
Type	Use the default `Text entry` for the parameter type.
Default Value	Type-in `Hello World` for the default parameter value.

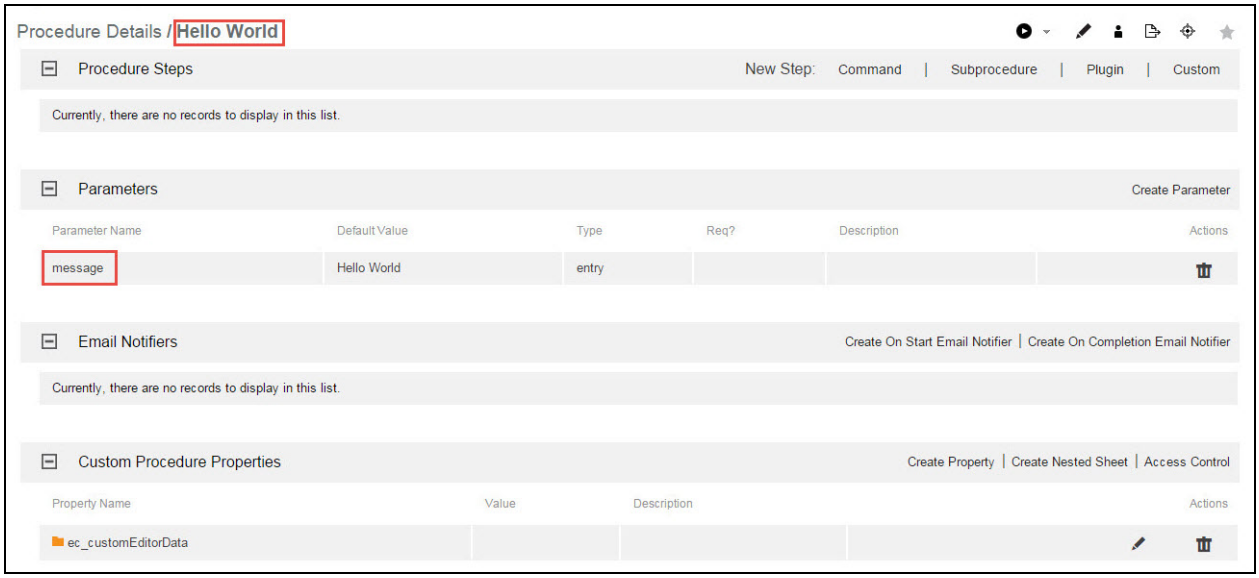
Your New Parameter entries will look like the following:



Click **OK** after filling in the fields and ElectricFlow returns you to the Procedure Details page.

On the Procedure Details page:

Notice the Parameters table is populated with your parameter entry.



Also notice the name of the procedure we created (Hello World) is shown adjacent to the page title for your reference.

You can use the DSL Export ( ) button to download the objects as a DSL file.

Step 4. Create a step

Now we can create a step on this procedure. On the Procedure Details page, to create a New Step for our current purpose, click the **Command** link to go to the New Step page.

On the New Step page:

The following screen displays the portion of the New Step page you need for this scenario.

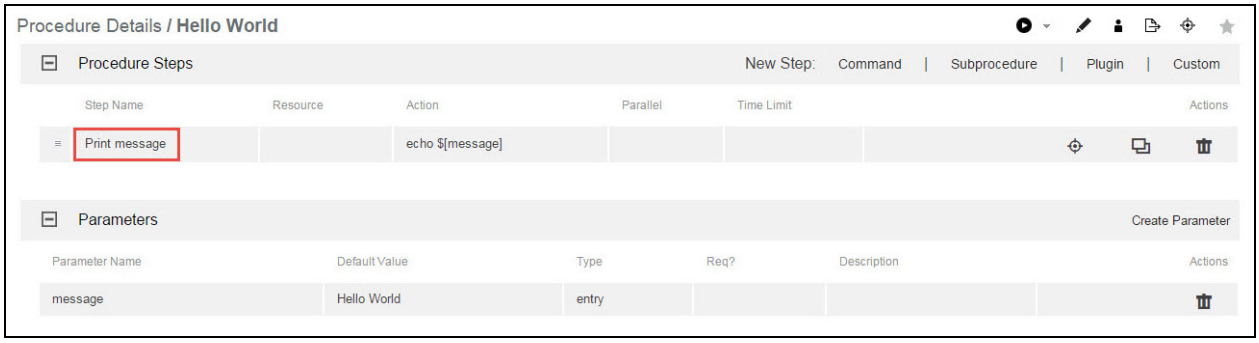
Notice this version of the New Step page contains a Command text box to enter the information you need. At a later time, when you choose to create a different step type, the New Step page will be populated with the appropriate fields required for that step type.

Enter the following information:

Field Name	Description
Step Name	Use `Print message` for the step name. For work outside this scenario, you can enter any unique name of your choice for the step name.
Description	(optional) Add a text description about this step if you need one.
Command(s) box	Type `echo $[message]` In this command, `message` refers to the parameter we created. **Note:** The ElectricFlow Server does not actually interpret (or even understand) the contents of this field. ElectricFlow simply expands all property references and then passes the resulting text block to the ElectricFlow Agent. The Agent copies the text block to a temporary file and then invokes the Agent's native shell command, passing the name of the temporary file as a parameter to the shell command. The overall result is the same as if you had created this block of text as a BAT file (on Windows) or a shell script (on Linux). The operating system of the Agent machine takes over from that point to actually run your commands.
Resource(s)	For this scenario, leave this field blank to use the default resource you set on the procedure. If multiple resources are configured, you have the option to change the resource for each step if you prefer.

Your New Step page will look like the following example:

Click **OK** to create the step and ElectricFlow returns you to the Procedure Details page to see its entry as displayed in the next screen example.

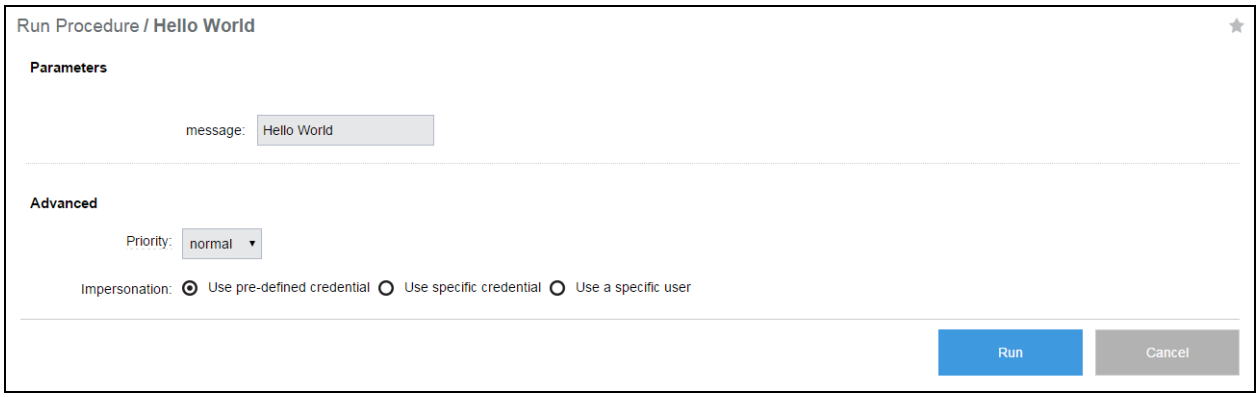


Step 5. Run the procedure

Notice the Run-arrow icon among the links at the top of the tables. If you click the **Run** link, the procedure will run as-is immediately.

For this scenario, hover your mouse over the adjacent down-arrow and select the **Run...** option.

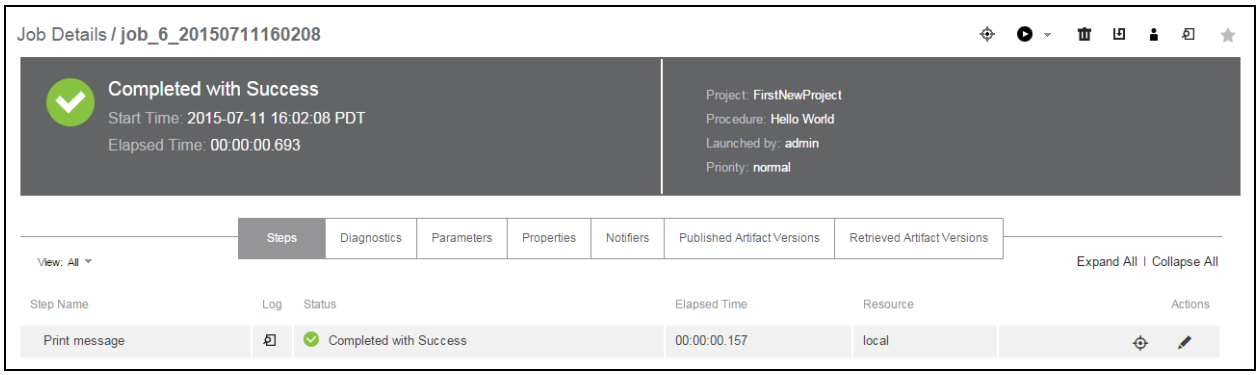
Selecting this option provides the Run Procedure page where you can add or modify the parameter values as necessary.



The Run Procedure page has the information we want for this scenario, so click the **Run** button.

When a procedure runs, it creates a "job".

After clicking **Run**, ElectricFlow takes you to the Job Details page to see the status of the "job" and watch its running progress.
This web page supplies a wealth of other information.



The Job Details page contains two sections. The section at the top of the page is the job's summary, displaying the job's status and other general information:

- The "green check" icon is displayed to show the job completed successfully.
- The "Completed with Success" section displays the result of the procedure that ran.
- The General Information section displays the project and procedure names and who launched the job.

The section below the summary area displays detailed job information, including tabs to expand information and links to access various other ElectricFlow web pages.

- Notice the "Success" entry in the Status column—this is your step result.
- In the Log column, click the icon in the Print message row to see the outcome/result of your Hello World Print message step. Your file will look similar to the following:

- Select the Parameters tab to see the parameter you created and its value.

For complete information about the Job Details page functions and available information, click the **Help** link in the top-right corner of this ElectricFlow web page.

Scenario Extension—Adding Another Step

Each procedure you create can contain as many steps as you need. We created only one step in this scenario. This example shows how to modify a procedure by adding additional steps. You can always add more steps or edit an existing step before you run a procedure again.

To add a step to an existing procedure:

1. On the Job Details page, select your procedure name (at the top of the page) to go to the Procedure Details page.
2. On the Procedure Details page, click the **Command** New Step link again.
3. On the New Step page, enter information in the fields as follows:
   - Name–`Print Step Info`
   - Command(s)–Enter the following text:
     ```
     print "This step was launched by $[launchedByUser] on the resource
     $[/myResource/resourceName].\n";
     ```
   - Shell–`ec-perl`
   - Click **OK** to save the step and return to the Procedure Details page.

     > **Note:** You can move this new step to the top of the procedure by hovering your mouse over the icon next to the step name, then drag-and-drop the step to the top of the list. This is a useful function particularly if you find you have created steps in a different order in which you need them.

Click **Run** (from the Procedure Details page).

Finding an ElectricFlow Automation Platform Web Page

If you "lose" the Job Details page or any other ElectricFlow web page, several navigation methods are available:

- Use your browser's Back button if you were recently on a page you want to revisit.
- Use ElectricFlow "breadcrumbs." On some pages you see breadcrumbs on the left-side of the page that note your current position/level within the project.
- Use the tabs at the top of the page to drill-down to the page you need.
  For example:

- **To find the Job Details page:**
  Select the Jobs tab and notice that your procedure name is now listed in the Job column, accompanied by a date, job number, and other related job information on the same row. When you run a procedure, it becomes a job. Select the job you want to see and you will go to the Job Details page for that job.
- **To find the Procedure Details page:**
  Select the Projects tab, then select your project to go to the Project Details page to see a list of procedures in that project. Click on a procedure to go to the Procedure Details page for that procedure.

> **Note:** A "project" is a container for all related procedures you create. You can create as many projects as you need, using the **Create Projects** link on the Projects page.

As you use ElectricFlow, you will quickly learn other navigation paths for your particular interests. In addition, most ElectricFlow web pages contain a "star" icon in the upper-right corner. Click this icon to add that page to your Home page for quick "one-click" access in the future.

Summary

This simple `Hello World` procedure demonstrated how to quickly create a procedure and how any command can be wrapped and parameterized using an ElectricFlow procedure. If you have an existing script you want to automate, you can call that script within a step command block. This scenario also introduced you to running a job, entering parameters for the job, and viewing results for the job you ran.

Go to Scenario 2

Getting Started—Scenario 2—Creating a Procedure That Uses an SCM

Scenario requirements: Before you begin this scenario, ElectricFlow needs to be able to access your Source Control Management (SCM) system.

To set up a Source Control system (SCM) to use with ElectricFlow, you need:

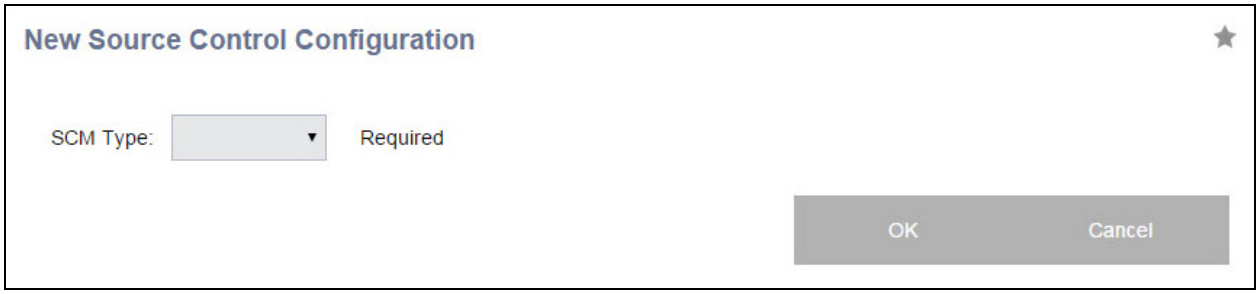
- a running SCM system
- an active source depot in your SCM system
- an ElectricFlow agent that can communicate with your SCM server

  If the machine where you installed ElectricFlow can communicate with your Source Control (SCM) server, this requirement is met. If not, return to the installation executable file and run the ElectricFlow Agent install program on a machine that has access to your SCM system.

  To install ElectricFlow agent software on another machine, copy the installation executable file to that machine, then double-click on the file to run the installation program. Select Express Agent when the install window opens.

When installation is complete, configure this new agent as an ElectricFlow Resource. See the beginning of Scenario 4 for more information.

- Configure ElectricFlow to communicate with your SCM system.
- Select the Administration tab, then the Source Control subtab.
- The Source Control Configurations page displays a table listing Source Control configurations after you create them. Click the **Create Configuration** link and the following web page is displayed.



- On the New Source Control Configuration web page, use the drop-down menu to choose your SCM type.
  For our example, we chose **Perforce**.
- Depending on which SCM you choose, the page expands to display appropriate fields for entering values to configure your SCM.

See the next screen example.

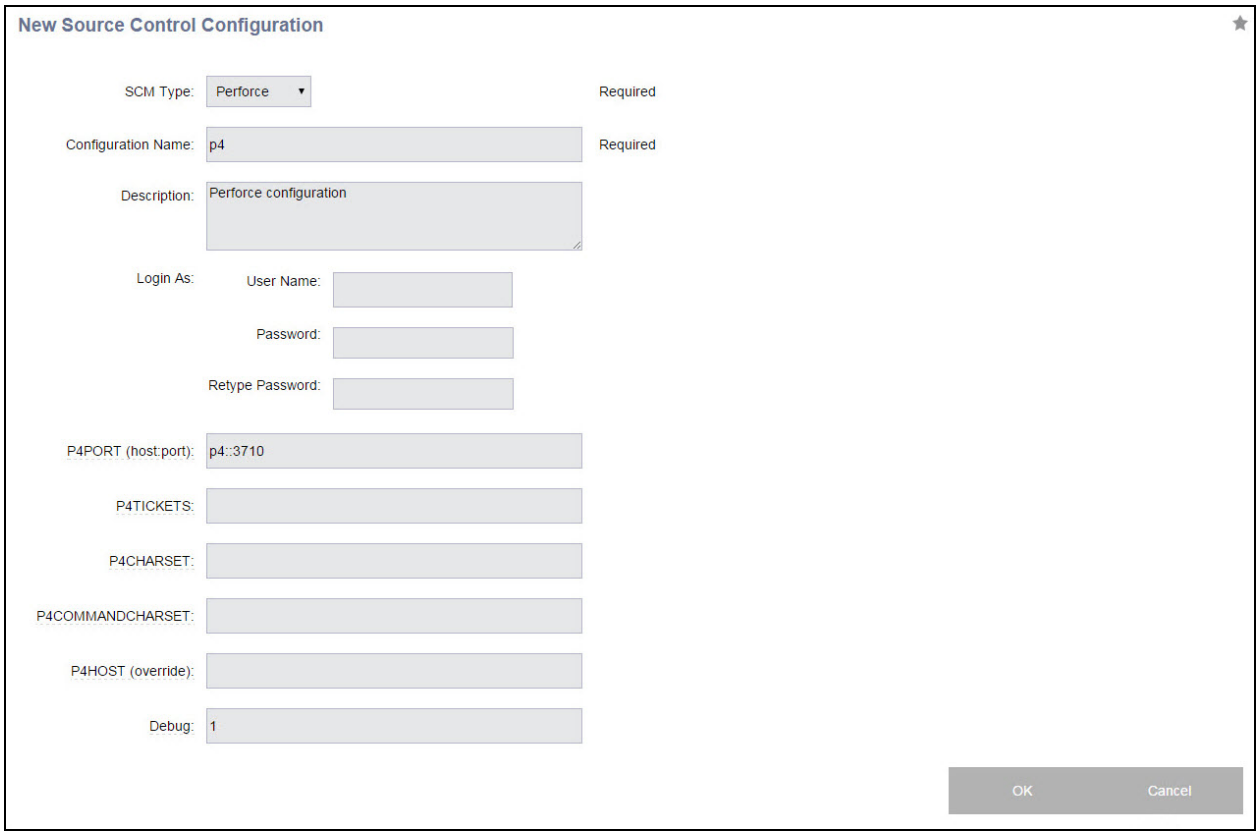
Enter information into the fields as follows:

- Configuration Name—For our example, we use the name, "p4". You can choose any name for your SCM Configuration.

  Note: You will need this name later, so remember the name you choose for your SCM configuration. A configuration name is important because you may want to create more than one source control configuration.

- P4PORT—For our example, we use p4:3710 to designate the P4PORT. Your specification will be different.

Important: Any blank fields will use the defaults, but you may need to specify some or all information in the remaining fields depending on how your internal systems are set up to access your SCM system.



Click **OK** after filling in the fields.

For additional help with this ElectricFlow web page, click the **Help** link in the top-right corner of the web page.

Overview

This scenario guides you through creating a common build process integrated with your SCM system and build utility. And you will learn how to navigate and modify the procedure definition using the ElectricFlow Procedure Details web page.

At the end of this scenario, you will be familiar with the following ElectricFlow concepts and features:

- Source control configuration
- Creating more complex steps
- Creating a build process
- Navigating the Procedure Details web page

Begin Scenario 2

Step 1. Select the Projects tab

On the Projects page, select the project name you created in Scenario 1 (in the first column) to go to the Project Details page, then select the **Create Procedure** link to begin.

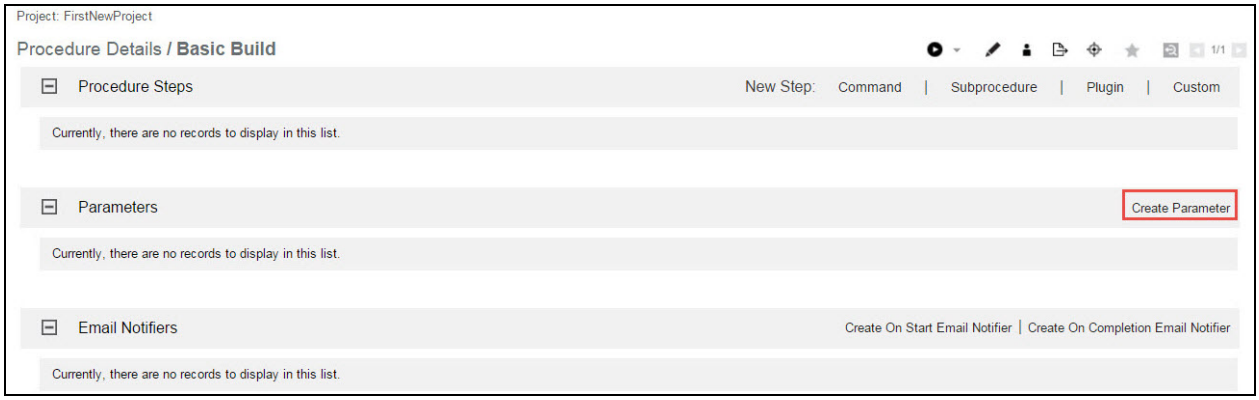
Step 2. Create a procedure

On the New Procedure page, enter a Procedure Name (required).

You can use any unique name you choose. Do not use the same procedure name you chose for Scenario 1. Procedure names within the same project must be unique.

- Name—For this scenario, we use `Basic Build` for the procedure name.
- Default Resource—Use `local` for the resource name.

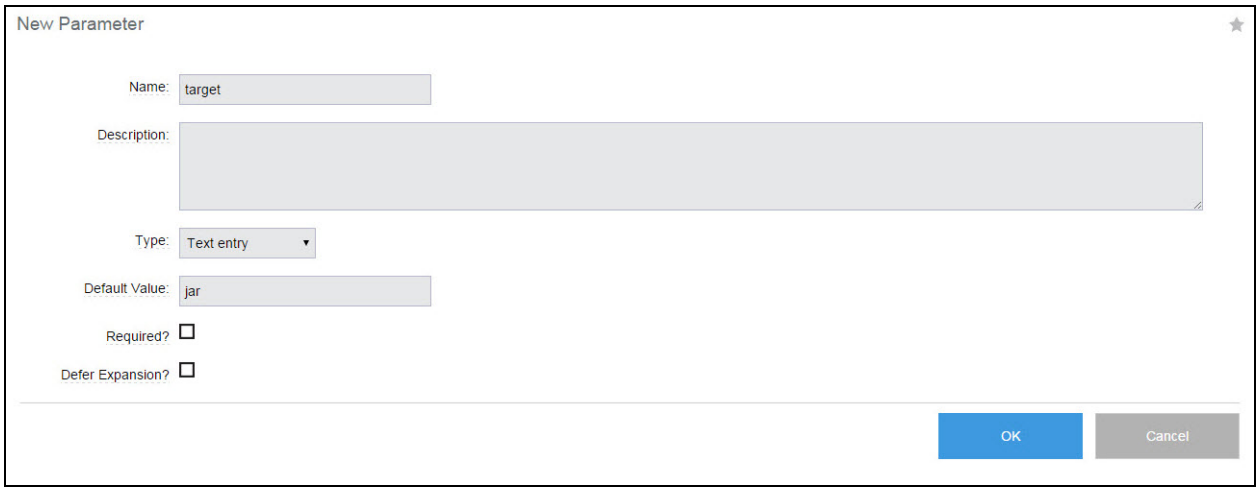
Click **OK** to go to the Procedure Details page.



You can use the DSL Export ( ) button to download the objects as a DSL file.

Step 3. Create a parameter

On the Procedure Details page (illustrated above), click the **Create Parameter** link to go to the New Parameter page.



- Name—For our example, the parameter name is `target`. You can choose any unique name of your choice for your parameter name—parameter names must be unique within a procedure.
- Default Value—We chose `jar` for this value.

Click **OK** to continue and to return to the Procedure Details page to see the parameter you just created.

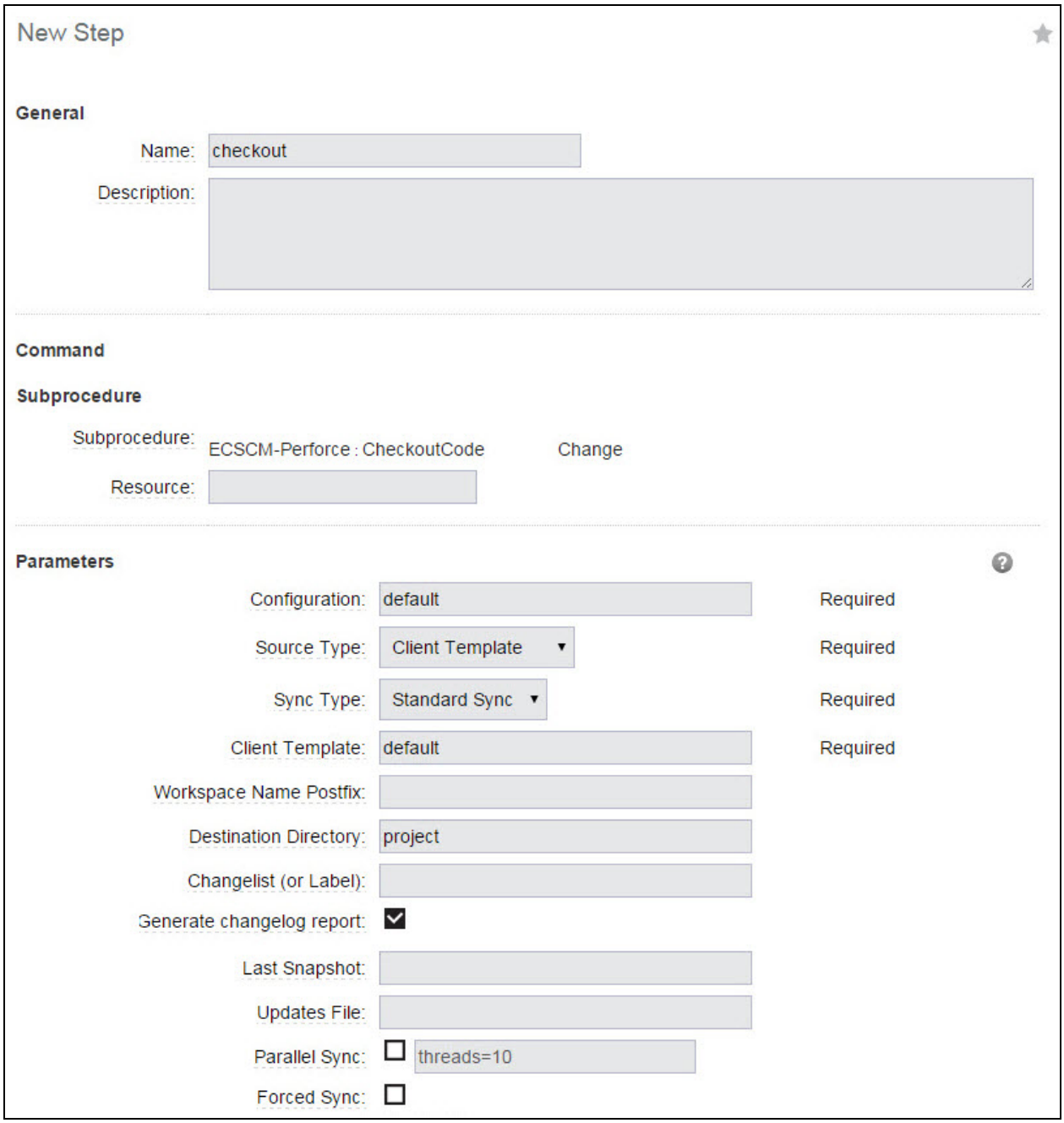
Step 4. Create a step to "checkout" source files from your SCM system

From the same Procedure Details page, click the Plugin link to create a New Step.

- In the Choose Step dialog, select Source Code Management from the left pane.
- Next, select ECSCM-Perforce from the list, then select "Perforce—Checkout" from the right pane.
- Click **Close** to go to the New Step page, which will be populated with the fields you need to enter for the Perforce SCM.

  Both the Subprocedure and the Parameters sections are populated with fields to enter information so ElectricFlow can communicate with your Perforce SCM system, using the ElectricFlow Perforce plugin.

Depending on the SCM you chose to work with, including Perforce, the New Step screen you see may be different than the following example. If you need additional help configuring your SCM, see the corresponding SCM plugin on the Plugin Manager page. Each plugin has its own Help topic.



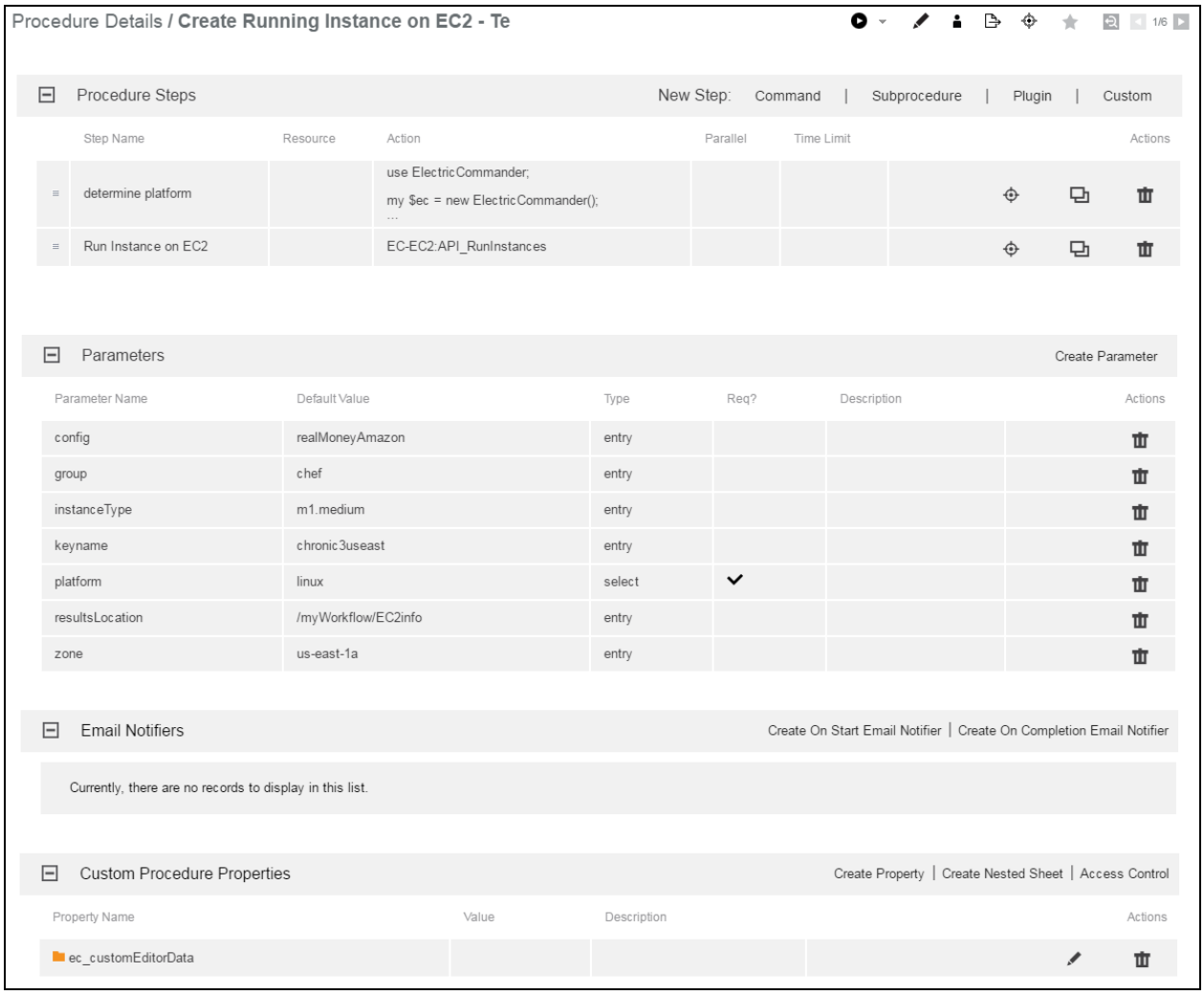
Enter information into the fields as follows:

Field Name	Description
Name	Enter a unique name for your subprocedure step. For this scenario, the step name we supplied is `checkout`.
Description	(Optional) Enter a text description for this step.

Field Name	Description
In the Subprocedure section: This section displays the subprocedure name or a plugin name because the plugin is being called as subprocedure. The plugin name defined: "EC" is for Electric Cloud, "SCM" identifies the plugin as belonging to the Source Control Management category, and "Perforce" identifies the name of the source control system. Our example illustrates the fields for the Perforce SCM plugin. Your fields will be different if you are using a different SCM. These parameter fields request some of the same values you already specified when you created the Source Control Configuration.	
ECSCM-Perforce	Clicking this link takes you to the Project Details page for the plugin. Generally, and for this scenario, you do not need to access this page to make changes—it is for your reference only.
CheckoutCode	Clicking this link takes you to the Procedure Details page for this procedure. Generally, and for this scenario, you do not need to access this page to make changes—it is for your reference only.
Change	Clicking this link displays a dialog that allows you to change the project, plugin, or subprocedure. If you make a change in the Change Subprocedure dialog, the Subprocedure and Parameters sections on the New Step page will update automatically with corresponding fields for your new choice. All SCM plugins that can checkout code contain the `checkoutCode` procedure.
Resource	The `local` resource was specified earlier. If this field remains blank, the `local` resource is used.
The following two parameter fields are common among all SCM plugins:	
Configuration	The name of your Source Control Configuration.
dest	The destination where you want to put your checked-out code—it is the directory within your job workspace. Each job adds a new directory to the workspace. At this point, you have not defined multiple workspaces, so the ElectricFlow default workspace is used.

For more information about workspaces, see Workspaces and Disk Space Management.

Click **OK** after entering values to continue to the **Procedure Details** page, which should now look similar to the following example:



The Procedure Details page now displays the name of our new procedure, Basic Build, and its new `checkout` step using the Perforce SCM, and the `target` parameter with the `jar` value.

Step 5. Create a step to run the build

This step can be another subprocedure step, a command step, or a custom step. Any procedure can contain multiple types of steps, any of which may be either simple or complex.

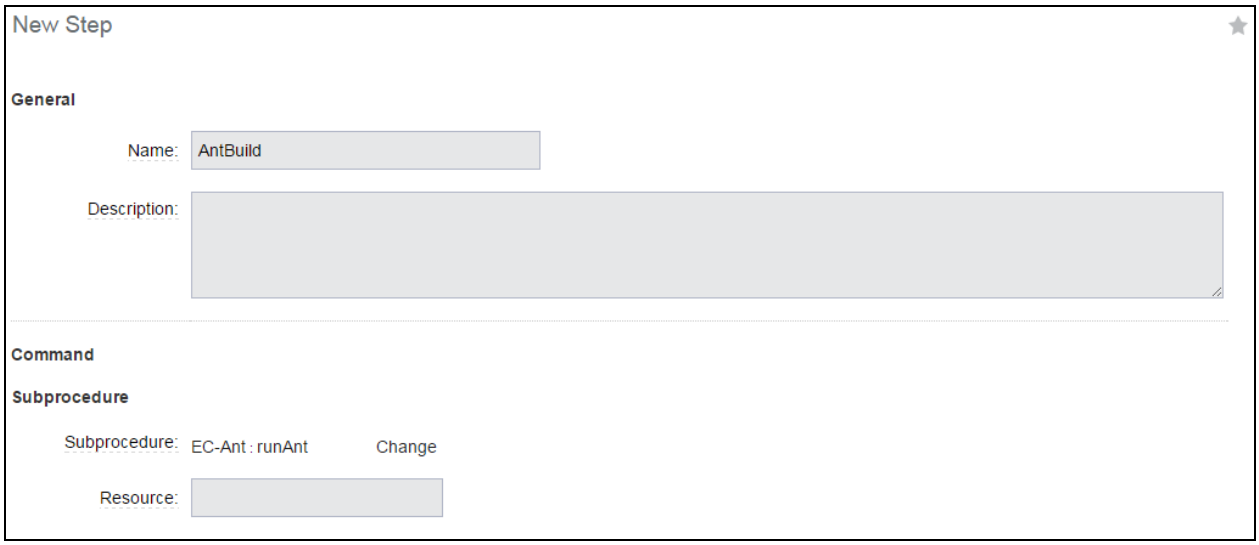
Steps can contain or use:

- parameters or properties
- many of your existing scripts, including shell scripts

For this step example (to run the build), we will create an `Ant` step. To invoke Ant, you could write a script that calls Ant directly, or you could call the ElectricFlow Ant plugin to invoke the `runAnt` procedure. For this example, we will use the ElectricFlow Ant plugin.

- From the Procedure Details page, click the **Plugin** link to see the Choose Step dialog and select the Build category.
- Select the EC-Ant plugin to see and select the "Ant—Run Ant" step.

- When the New Step page is displayed, notice the Subprocedure section is populated with the chosen EC-Ant plugin, and the Parameters section provides the appropriate fields for this Ant step.



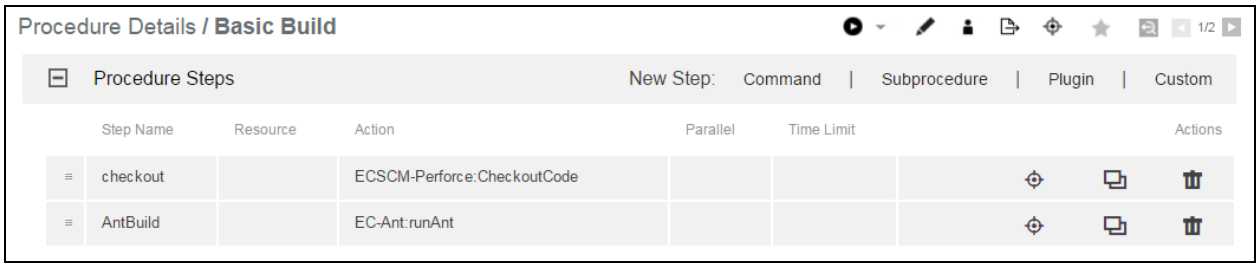
Enter information into the fields as follows:

Field Name	Description
Name	For this example, we specified `AntBuild` for the step name. You can use any unique name you choose for this step.
Subprocedure section: Note the EC-Ant plugin project name and the `runAnt` procedure name.	
Resource	Leave this field blank to continue using the local resource.
Parameters section:	
Build File	Use `build.xml`.
Target	For this example, we used `$[target]`.
Working Directory	For this example, we used `project`.

For this scenario, we leave the remaining fields blank, but you may need to enter additional information depending on your Ant invocation.

Click **OK** to continue and return to the Procedure Details page.

The following screen example illustrates the two steps we created for this scenario.

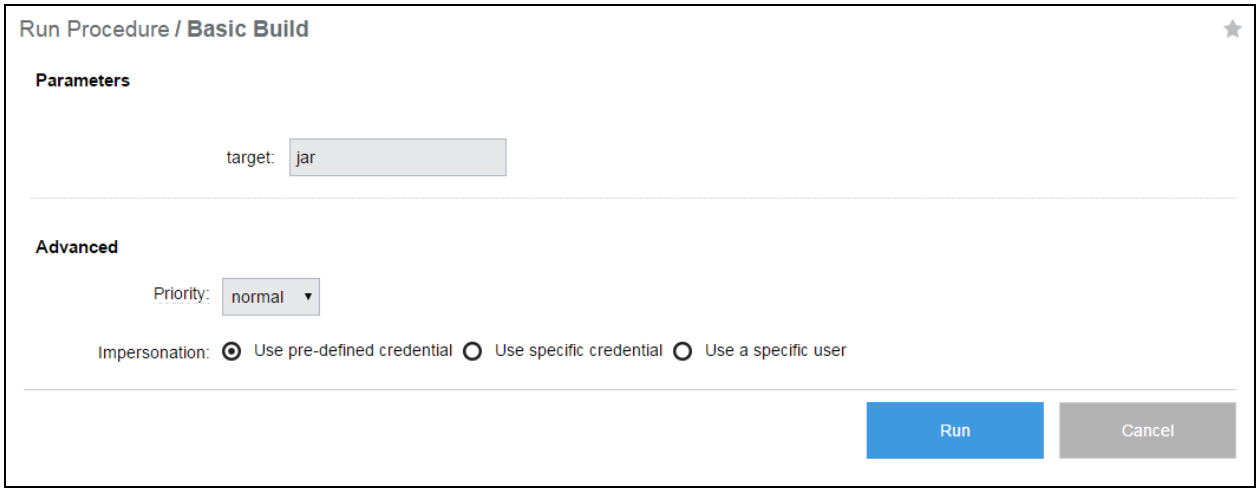


Step 6. Running the procedure

On the Procedure Details page, hover your mouse over the small down-arrow on the right-side of the **Run** link, choosing **Run...**

- **Run Immediately**—This option runs the procedure immediately as set.
- **Run...**—This option displays the Run Procedure web page where you may alter any existing parameters for this procedure, or set an existing credential.

  Note: This scenario does not introduce you to the Advanced section of the Run Procedure page. For more information on these topics, click the **Help** link in the top-right corner of the Run Procedure page.

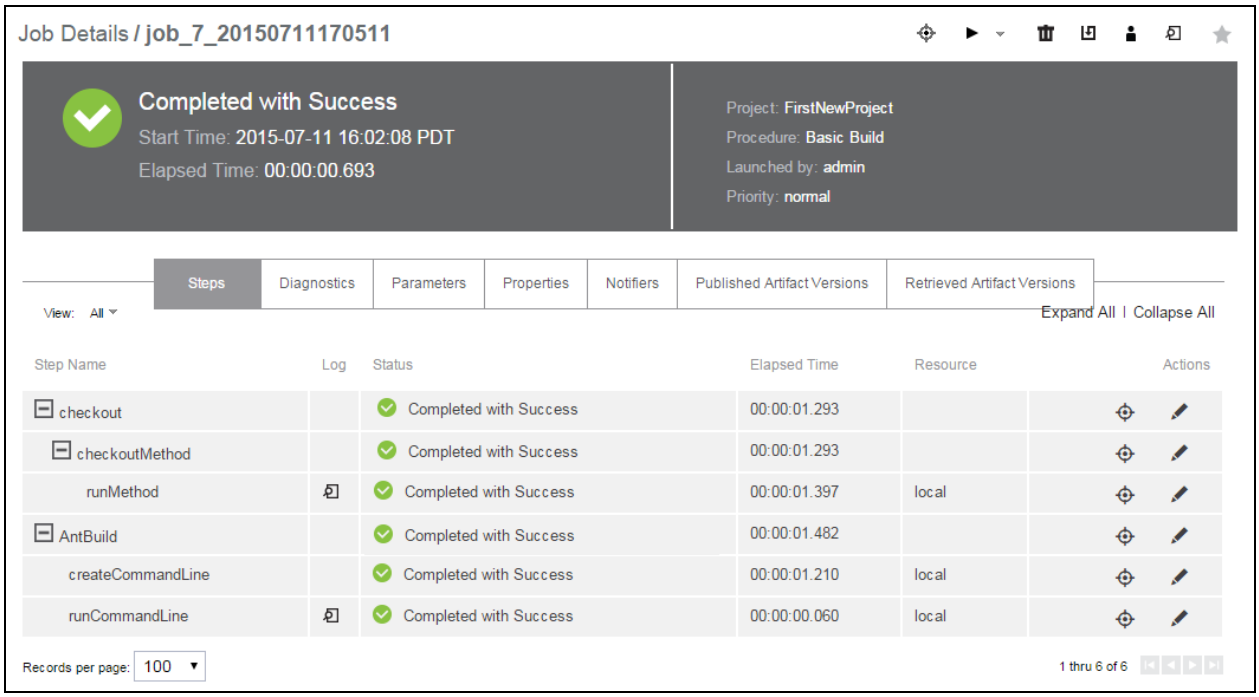


Click **Run** to run the procedure and continue to the Job Details page to see the status of your running job.

Step 7. On the Job Details page

The Job Details page now displays the status and results of running the Basic Build procedure.

Your Job Details page will look somewhat different if you supplied other values in this scenario.



More about the Job Details page

- The General Information section at the top of the table provides links back to the **Project** > **Procedure** that was launched to create this job.
- Click on a Step Name to go to the Job Step Details page.
- Click the Parameters tab to see parameters in this job.
- If there are errors, the Diagnostics tab has specific information.

More about the Procedure Details page

To go to the Procedure Details page at any time:

- Select the Projects tab.
- Select the Project name (that contains the procedures you want to see) to go to the Project Details page.
- Select a Procedure name to go to the Procedure Details page.

You may want to review to the Procedure Details page frequently because it displays the steps that make up your procedure, the parameters contained in the procedure, email notifiers, and so on. Full edit capabilities are available by selecting any object (step, parameter, and so on).

Also, you can run this procedure again "on demand" by clicking the **Run** link at the top of the page.

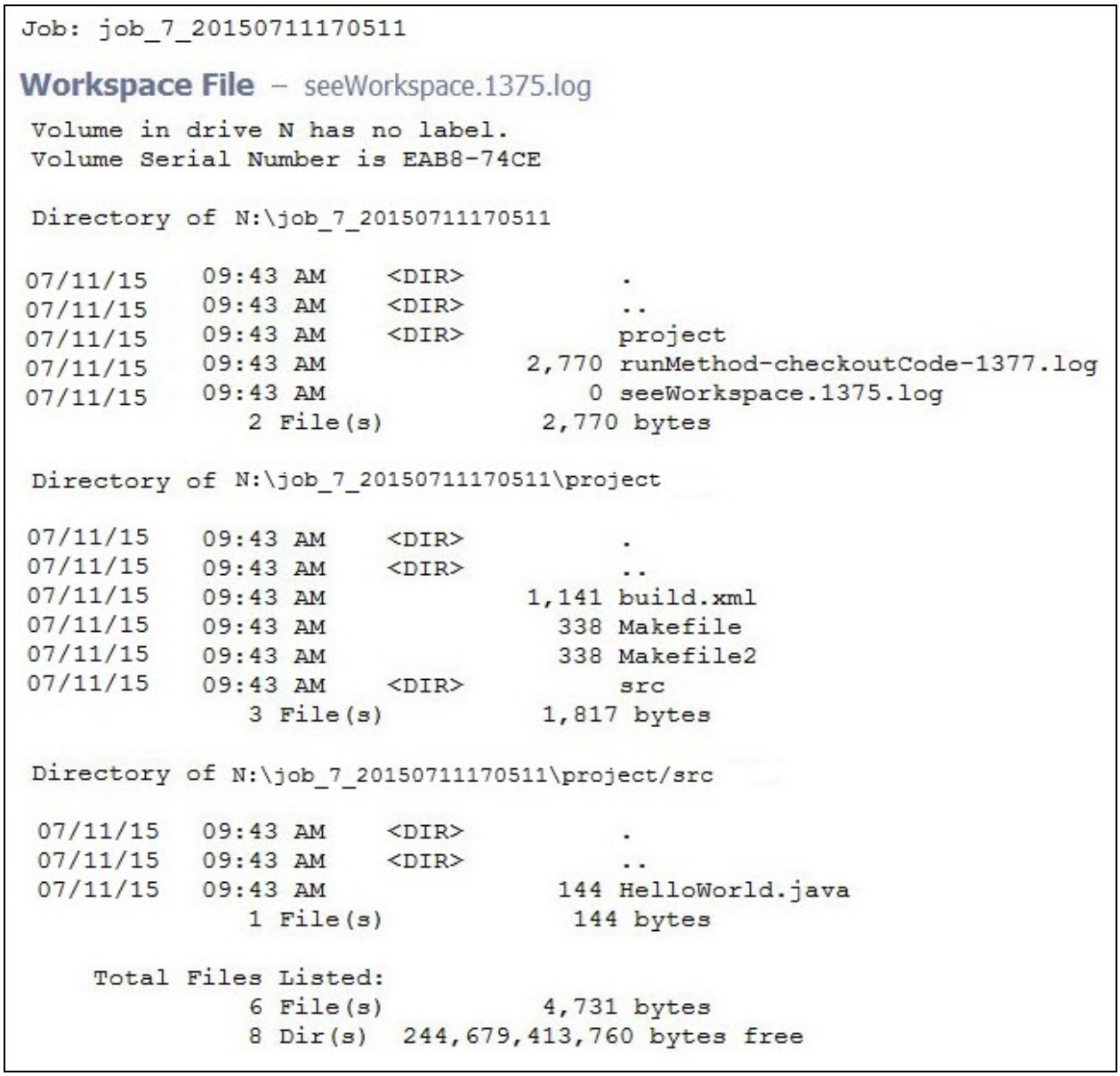
For more information about the Procedure Details page functions, click the **Help** link in the top-right corner of the Procedure Details page.

Scenario extension—Add a step to show workspace contents

If you are curious about what your current default workspace contains at this point, you can create a step to see the workspace.

- Select the Projects tab, select FirstNewProject (or your project name if you used something different), select the Basic Build procedure name.
- Click the **Command** link to create a New Step—a new *command* step.
- The New Step page is displayed with a Command text box.
  - For Name, use `seeWorkspace`.
  - In the Command(s) text box:
    - for Windows—type `dir /s`
    - for Linux—type `ls -r`
- Click **OK** to create the step and go to the Procedure Details page.
- Click **Run** to create the job and to go to the Job Details page.

When the job is completed, click the log icon in the Log column to see the contents of the current workspace for this procedure. Your Workspace File will be similar to the following example:



Job: job_7_20150711170511

Workspace File – seeWorkspace.1375.log

```
Volume in drive N has no label.
Volume Serial Number is EAB8-74CE

Directory of N:\job_7_20150711170511

07/11/15    09:43 AM    <DIR>          .
07/11/15    09:43 AM    <DIR>          ..
07/11/15    09:43 AM    <DIR>          project
07/11/15    09:43 AM             2,770 runMethod-checkoutCode-1377.log
07/11/15    09:43 AM                 0 seeWorkspace.1375.log
               2 File(s)          2,770 bytes

Directory of N:\job_7_20150711170511\project

07/11/15    09:43 AM    <DIR>          .
07/11/15    09:43 AM    <DIR>          ..
07/11/15    09:43 AM             1,141 build.xml
07/11/15    09:43 AM               338 Makefile
07/11/15    09:43 AM               338 Makefile2
07/11/15    09:43 AM    <DIR>          src
               3 File(s)          1,817 bytes

Directory of N:\job_7_20150711170511\project/src

07/11/15    09:43 AM    <DIR>          .
07/11/15    09:43 AM    <DIR>          ..
07/11/15    09:43 AM               144 HelloWorld.java
               1 File(s)            144 bytes

    Total Files Listed:
               6 File(s)          4,731 bytes
               8 Dir(s)  244,679,413,760 bytes free
```

Summary

This scenario demonstrated how easily ElectricFlow can integrate with your SCM system and build utilities.

Go to Scenario 3

Getting Started—Scenario 3—Notification, Scheduling, and Reporting

Scenario requirements: For this scenario, you must set up an Email Configuration to use the email notification feature.

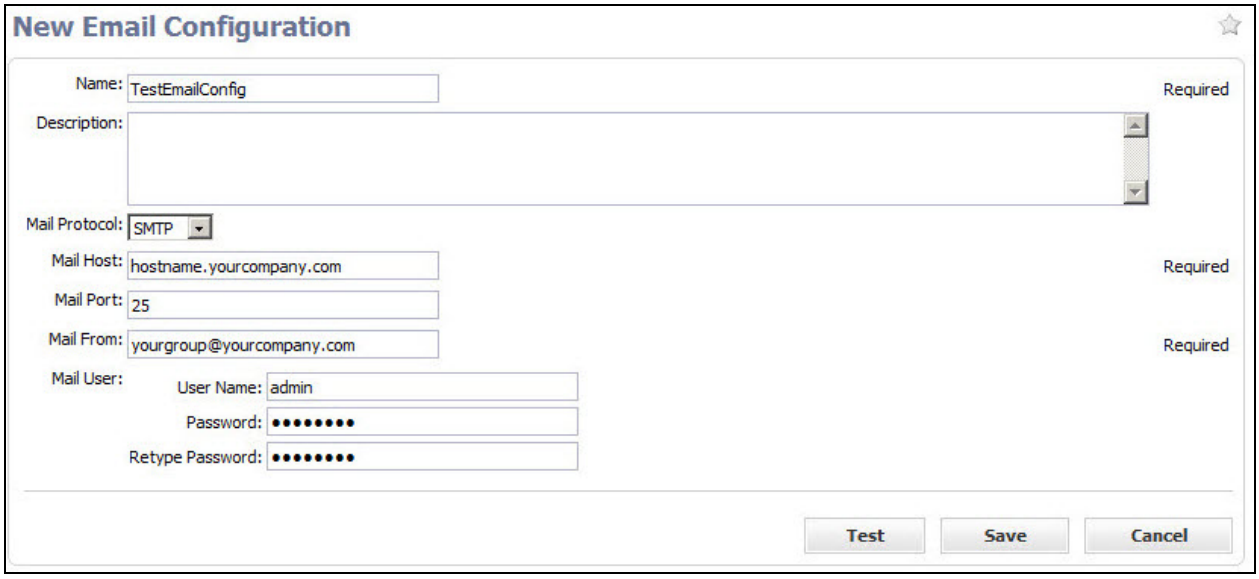
To set up an Email Configuration:

- Click **Administration** > **Email Configurations**.
- Click the **Add Configuration** link.

Enter information in the fields to define your mail system.

Name	New version	Status
Name	Enter a unique name for your email configuration.	
Description	Optional) Enter a text description for your reference.	
Mail Protocol	Use the drop-down arrow to make a selection.	
Mail Host	Enter the name of your mail host.	

Enter information in the remaining fields according to the specifications you need. Your New Email Configuration page should look similar to the following example:



Click **Save** after filling in the fields or click the **Test** button first if you want to make sure your email configuration works.
Enter an email address in the pop-up box and click **Send**.

> **Note:** For additional help with this ElectricFlow page, click the **Help** link in the topright corner of the page.

Overview

This scenario guides you through creating an email notification, invoking your build process with a timed (standard) schedule, invoking your build process using a continuous build integration trigger, and introduces ElectricFlow reporting features.

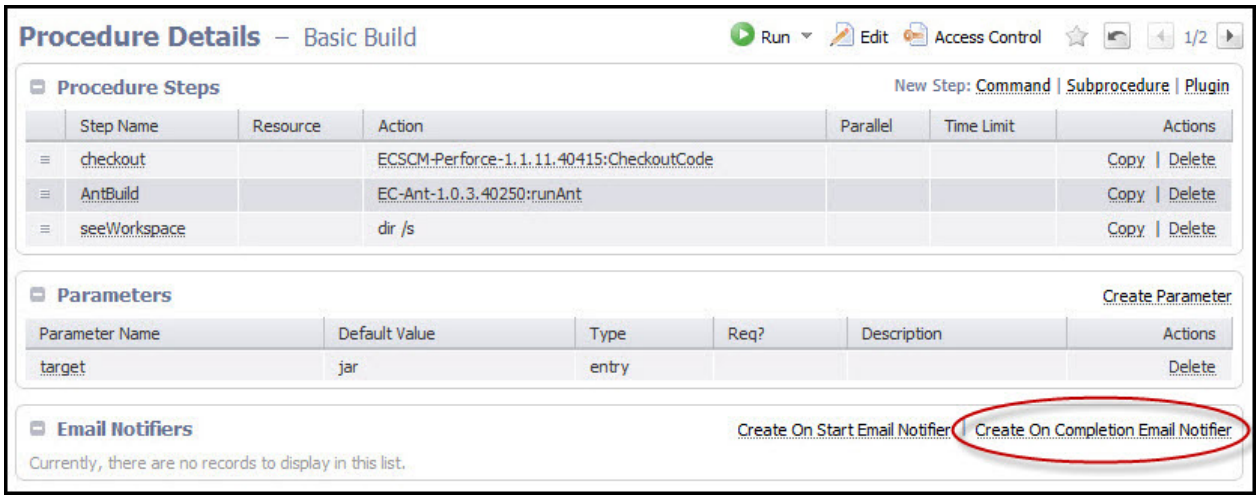
At the end of this scenario, you will be familiar with the following ElectricFlow concepts and features:

- Creating an email configuration
- Creating an email notifier
- Creating schedules
- Creating reports

Begin Scenario 3

Step 1. Creating an email notifier

On the Procedure Details page for the Basic Build procedure, click the **Create On Completion Email Notifier** link to begin. This link takes you to the New On Completion Email Notifier page.



Two email notifier types for jobs or job steps:

- On Start Notifier–Sends an email when a job or job step starts.
- On Completion Notifier–Sends an email when a job or job step completes.

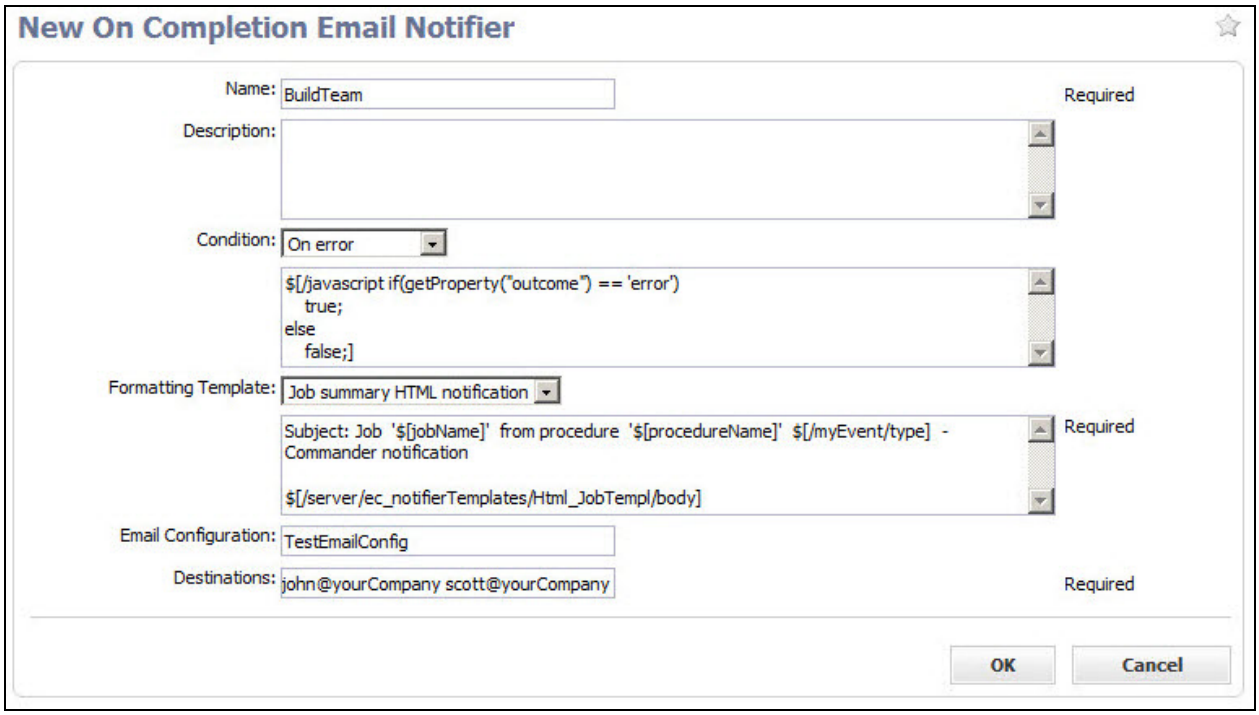
The fields for each of these notifiers are similar, but we chose the On Completion notifier because you may find you use this one more frequently.

On the New On Completion Email Notifier page:

Enter information into the fields as follows:

Field Name	Description
Name	For this example, we use `BuildTeam`. This name can be an arbitrary text string.
Description	(Optional) Enter a text description for your reference. ElectricFlow does not use or interpret this information.
Condition	Use the pull-down menu to select the type of condition you need for this email notifier. Edit the auto-supplied condition in the text box or add a completely new script for your purpose. The condition specifies whether the notifier should send a message depending on the result of a property expansion. If the result is empty, non-zero, or "true", the message is sent. If the result is "0" or "false", the message is not sent.
Formatting Template	For this example, choose the "Job summary HTML" notification template, or you can use the drop-down menu to select from a list of global, ready-to-use formatting templates. Depending on the type of email notifier you are creating, the available template choices in the drop-down menu will be different. To customize your template, edit the auto-supplied text in the Formatting Template text box, or you can add a completely new template for your purpose. **Note:** Any edits made in this text box will not be saved to the global template, and the template will appear as a Custom template when you return to this notifier definition. Additionally, make sure the content is formatted correctly, i.e., no illegal characters or spacing.
Email Configuration	Use `TestEmailConfig`, the email configuration we created.
Destinations	This is a space-separated list of valid email addresses, email aliases, or ElectricFlow user names, or a property reference that expands into such a list.

After filling in the fields, your page may look similar to the following example:

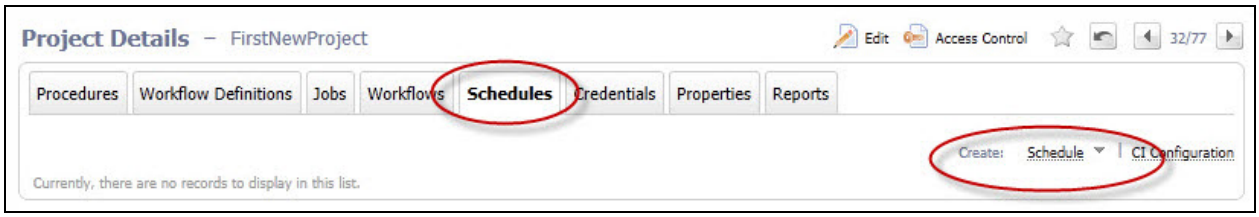


Click **OK** to create your email notifier and to return to the Procedure Details page, where you can now see the Email Notifier you created. For more information, see the Email Notifiers Help topic.

Step 2. Creating a standard schedule

To create a standard "timed" schedule:

- Select the Projects tab.
- Select your project name to go to the Project Details page.
- Select the Schedules subtab, then select the down-arrow to the right of the " Schedule" link, and choose "Standard Schedule" to go to the New Schedule page.

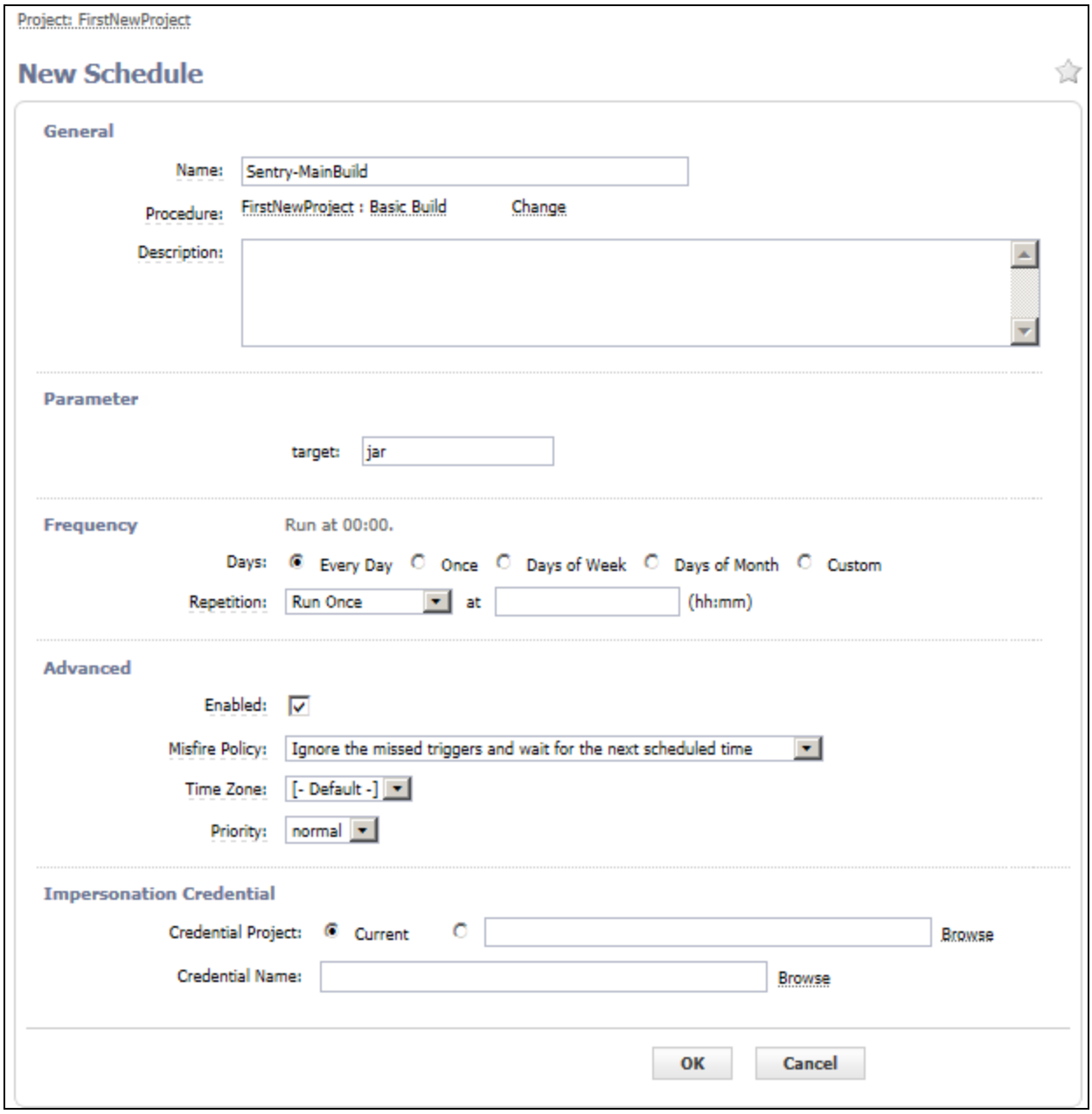


On the New Schedule page:

The following is an example of the New Schedule page.

- Notice the "breadcrumb" above the New Schedule page title—this entry displays the name of the project that will contain this schedule.

- After entering a name (unique within your project) for your schedule (we chose "Sentry-MainBuild"), choose the procedure your schedule will run.
  - Click the **Change** link adjacent to the Procedure field.
  - In the pop-up menu, leave the Current project selected.
  - Place your cursor in the Procedure field to see a list of procedures in your project.
  - Select the procedure you want this schedule to run. We chose "Basic Build".



Enter information in the remaining fields as follows:

Field	Description
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Parameters section. If the procedure has parameters, enter their values in this section. The procedure cannot execute unless all required parameters are provided.	
Frequency section. "Run at" is a summary section: As you make selections for the following Days and Repetition sections, a summary of what you selected appears here.	
Days	Select weekdays, month days, or create a custom frequency when you want the schedule to run. The "day" applies to the start time if the time range spans two days, that is, if the time range crosses the midnight boundary.
Repetition	If Run Once—the schedule will run once per selected day, at the time you specify. If Run Every—the procedure will execute repeatedly during the specified time range. For example, if you select a time range from 11:00-14:00 and an interval of 40 minutes, the procedure will execute 5 times on each of the selected days: at 11:00 am, 11:40 am, 12:20 pm, 1:00 pm, and 1:40 pm. If Run Continuously—as soon as one job ends, the scheduler will trigger immediately to run the job again.
Advanced section	
Enabled	If this box is "checked," the schedule will run. You can disable this schedule.

Note: For this scenario, you do not need to enter information for any remaining fields. Later, when creating an actual timed schedule for one of your procedures, refer to the Help topic for this page for more information.

Click **OK** to create this schedule.

At any time, you can return to the Project Details page for this project, select the Schedules subtab, and then select the Schedule Name from the table to go to the Edit Schedule page to modify this schedule.

Step 2a. Creating a continuous integration build trigger

At a later time when you have procedures and steps created to do the work you need, you may prefer implementing continuous integration build configurations for your software builds. Unlike a standard schedule that runs on days and times you specify, a continuous integration build can be triggered to run each time code is checked into your source code management (SCM) system.

Adding a project to the Continuous Integration Dashboard is quick and easy. Each project can have one or more CI configurations, depending on the number of source branches you want to monitor. After the simple configuration process, you can visually see status and other information about your continuous integration jobs as builds are triggered to run.

To find the Continuous Integration Dashboard, select the Home tab, then select the Continuous Integration subtab. The dashboard will be empty until you start adding projects. For help using the Continuous Integration Manager and Dashboard, click the **Help** link in the upper-right corner of the dashboard web page.

Step 3. Creating a report

ElectricFlow provides multiple reports and custom report capabilities to help you manage your build environment.
Summary of available report types:

- Real-time reports—filtered view of your ElectricFlow data in real-time
- Build reports—summary reports produced at the end of a build and attached to the job
- Batch reports—summaries of your build environment with trends over time, two types:
  - Default Batch reports—automatically installed during ElectricFlow installation and scheduled to run daily (Cross Project Summary, Variant Trend, Daily Summary, Resource Summary, Resource Detail)
  - Optional Batch reports—you can configure and schedule these reports to fit your requirements (Category, Procedure Usage, Count Over Time, or Multiple Series reports)
- Custom reports—your choice to create and add at any time

> **Note:** Batch and Custom reports must be run on the ElectricFlow server agent (local agent) only. These reports use the BIRT report engine, which is installed only on the ElectricFlow server.

Defining a "saved" search

Before you create a report, you need to define a "search" to focus the report on the information you want to see.

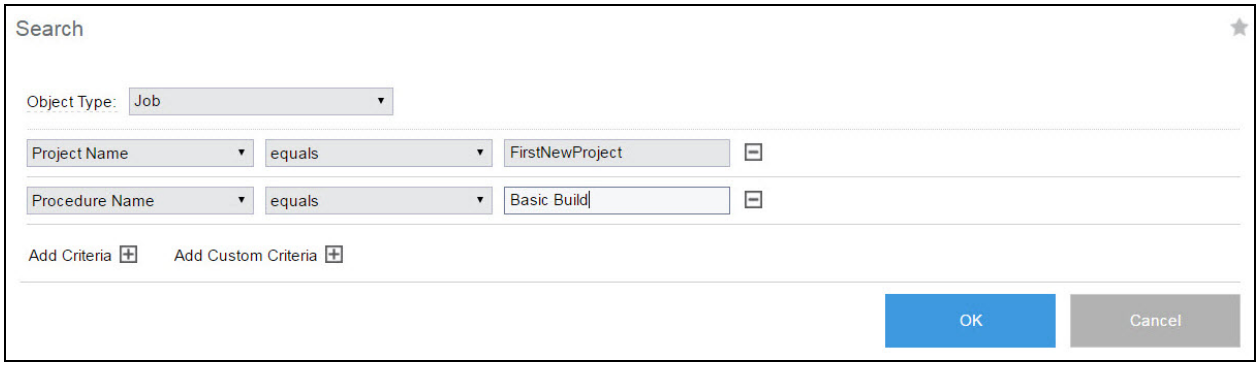
- Select the Search tab to go to the Searching and Filtering page.

Enter information into the fields as follows:

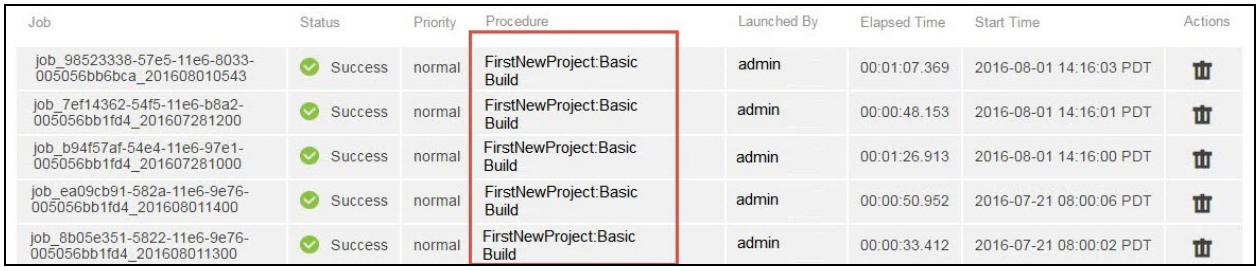
Field	Description
Object Type	Use the drop-down menu to select the object you wish to search.

Field	Description
Criteria	Click **Add Criteria** one or more times to further define the sort criteria for your search. • Use the drop-down arrow to select a criteria argument. These arguments are properties. For a description of any of these properties, see the Properties Help topic. • Use the drop-down arrow to select an operator for the search. • To narrow your search, enter an actual object name. For example, if you selected Project Name to begin your search, you might want to enter the actual project name in this field or the first few letters if you have multiple projects named in a similar pattern.
Custom Criteria	Not needed for this scenario.

Your Searching and Filtering page will look similar to the following example:

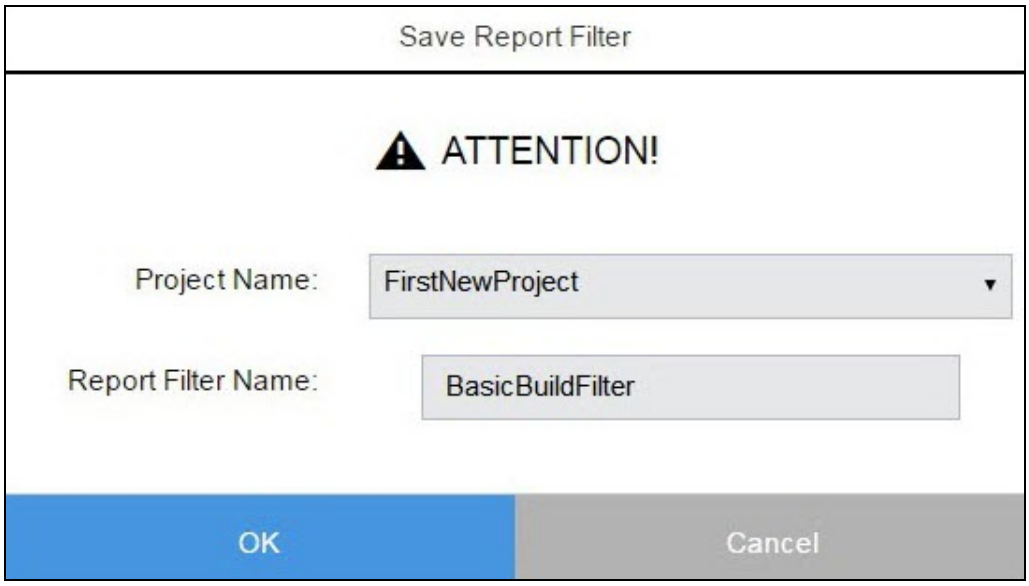


Click **OK** after filling in the fields and to go to the Search Results page.



Click **Save Report Filter** to see the Save Filter dialog box.

In this dialog box, use the drop-down arrow to select the project name for your report, then enter a name of your choice for the filter. We used `BasicBuildFilter` for the filter name to tie the filter name to the Basic Build procedure for which we are creating a filter.

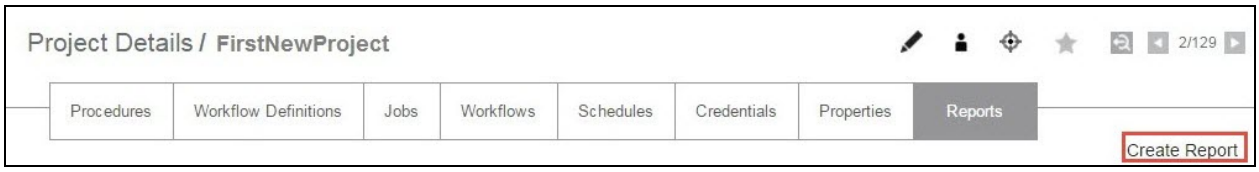


Click **OK** to return to the Job Search Results page—now we are ready to define the report we want to see.

Define the report

For this scenario, we will create an Optional Batch report. Select the Projects tab, select *your* project name to go to the Project Details page, then select the Reports subtab.

Click **Create Report**.



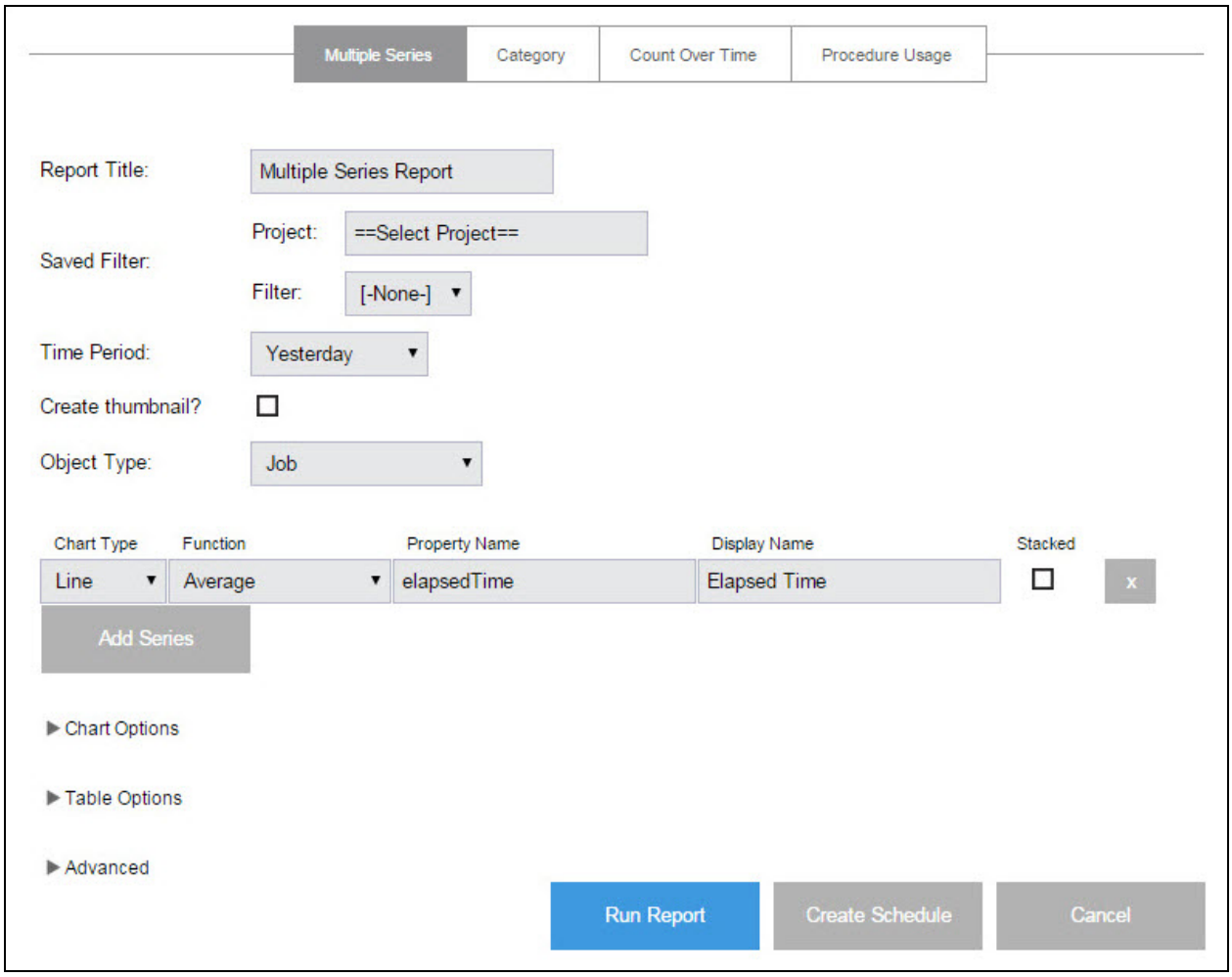
On the Optional Batch Reports page:

For this scenario, select the Multiple Series report tab.

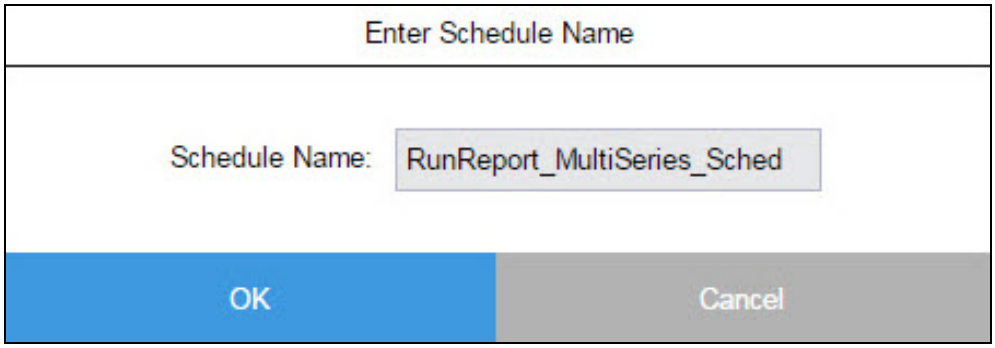
Enter information in the fields or select the appropriate values as follows:

Field	Description
Report Title	This is your report title. Type over the default report name, choosing any unique name for your report. We supplied `Basic Build Report` for our report name.
Saved Filter	Project—Click your mouse inside this field to see a list of projects from which to make a selection. We select our FirstNewProject project name. Filter—Use the BasicBuildFilter we created earlier.
Time Period	Use the drop-down menu to select the time period.
Create thumbnail?	Check this box so you can view this report on your Home page. (We will create a "thumbnail" report view on your Home page at the end of this scenario.)
Object Type	Use the drop-down menu to select Job for this example.
Table column choices	
Chart Type	Use the drop-down menu to choose the chart type.
Function	Use the drop-down menu to choose the function you need.
Property Name	Use the default property value or delete the text and click your mouse in the blank field to see a list of possible properties. We chose "outcome" for this example.
Display Name	You can choose a different unique Display Name if you prefer to do so.
Stacked	Select this check box to see your report results "stacked" versus overlaid.
The "X" icon	Click this icon to delete any row you no longer need.
Add Series button	Select this button if you would like to create additional table entries for additional report information.
Chart Options	Use the down-arrow to adjust the Time Grouping and see the defaults for the X and Y axis.

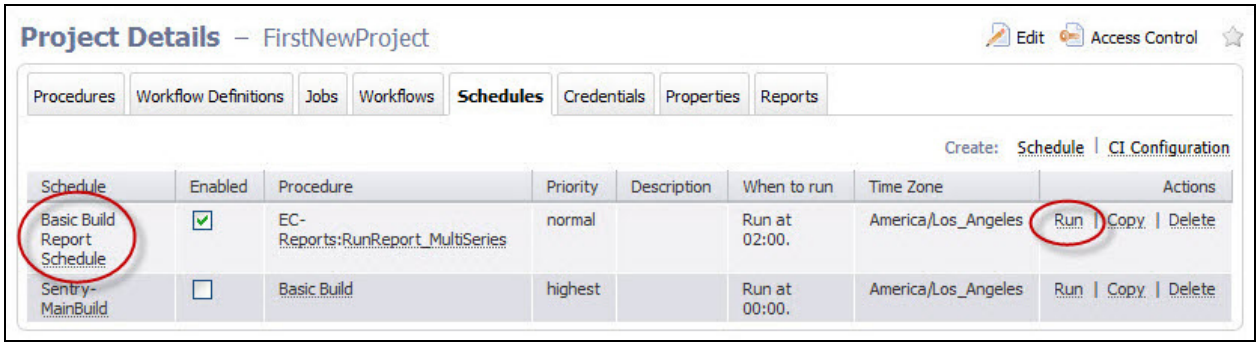
Your Multiple Series report page will be similar to the following example:



- Run Report button—(information only for this scenario) this button takes you to the Job Details page to run the report immediately—one time only.
- Click the **Create Schedule** button—this button displays a box with a default schedule name you can change.



- Enter a name for the schedule—again, we tied the schedule name to the procedure name for which we want the report.
- Click **OK** to go to the Project Details page.

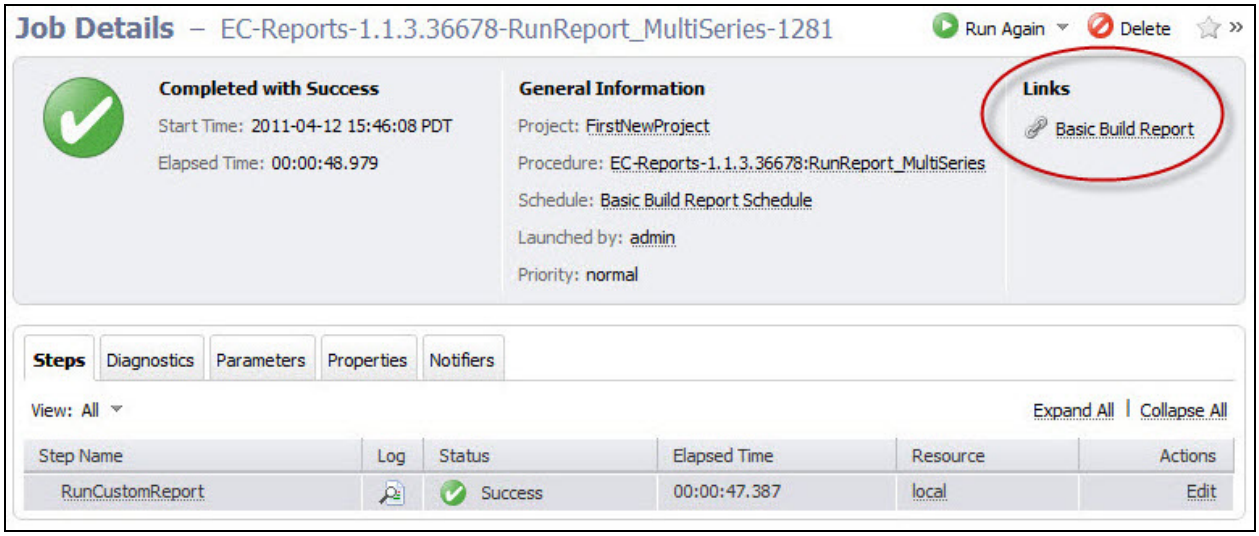


Now you can see the report you just created listed in the Schedules table.

Click the **Run** link in the Actions column to run this report and then go to the Job Details page.

On the Job Details page , you will see your report running.

When the report is generated, you will see a Links section in the summary section at the top of the Job Details page. This section contains a link to the report you generated. Click this link to see your report.



For more information about the Job Details page, click here. For more information about ElectricFlow reports, see the Reports, Creating Custom Reports, and other report Help topics.

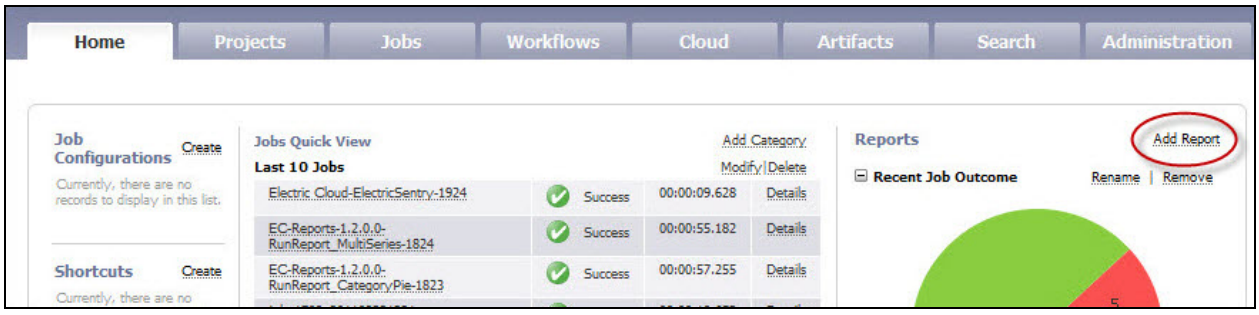
Scenario extension—Add a thumbnail report to your Home page

When we created a report, we "checked" the Create thumbnail? checkbox, so we are ready to activate the thumbnail report view on the Home page.

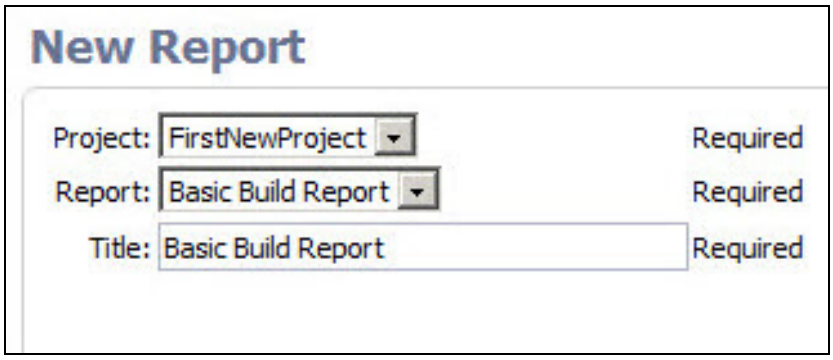
Note: Viewing an interesting report assumes there is ample data to report. While you can see a report after running one job, it is not very interesting unless you can see trends or comparisons after running numerous jobs.

To see a thumbnail report on the Home page:

- Select the Home tab.
- Click the **Add Report** link and a New Report page is displayed.



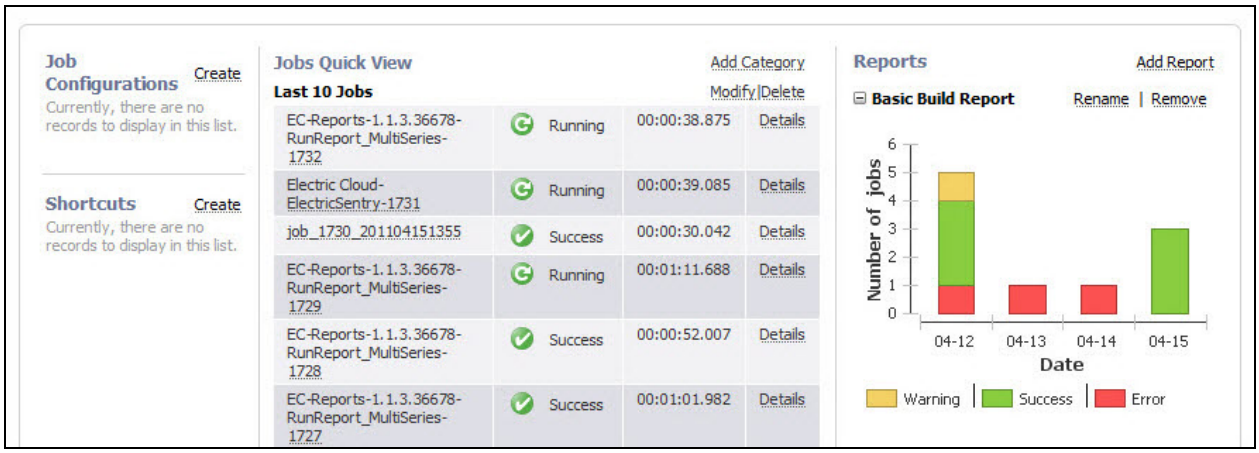
On the New Reports page:



- Project—Use the down-arrow to select the project containing the report you want to view—FirstNewProject.
- Report—This is the report name you entered in the Report Title field when you configured this report—Basic Build Report.
- Title—This is the report title you want to see on your Home page to identify this report. We used the same Basic Build Report name.

Click **OK** to generate the thumbnail report view and to go to the Home page.

The following screen example illustrates how your Home page may look with the thumbnail report.



For more information about how to use the Home page, see Automation Platform Home Page on page 1117 Help topic.

Summary

This scenario demonstrated how to create an Email Configuration so you could use an email notifier, how to create a continuous integration schedule, and how to set up a basic report with a "thumbnail" report view on your Home page.

Go to Scenario 4

Getting Started—Scenario 4—Multi-agent Build and Test

Scenario requirements: For this scenario, you can use the configurations you created for Scenario 2 and 3. However, before you begin this scenario, you need to configure an additional agent machine. If you have a firewall running on the ElectricFlow server machine, disable it now or allow access to ports 8000, 8443, 61613, 80, and 443.

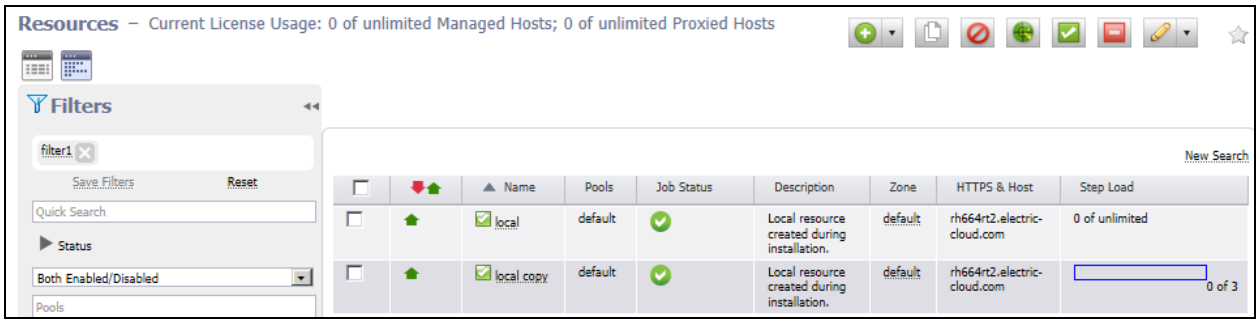
Installing ElectricFlow Agent software:

Install ElectricFlow Agent software on the machine you intend to use as an agent to run builds. For the simple purpose of getting started, install this agent on a different machine with the same operating system as the ElectricFlow server.

1. Copy the ElectricFlow installation executable file to the machine you intend to use as an ElectricFlow agent.
2. Double-click the executable file to begin the installation program.
3. When the installation program opens, select the Express Agent installation.

   > **Note:** During the agent installation, you can create a resource and provide a resource name. Enter any name you choose for your new resource. For this scenario, we will name our new resource, `Resource2`.

4. After agent installation, verify that your new resource information is included in the ElectricFlow resource table. Select the Cloud tab to go to the Resources page.



You can use the Resources page to install additional agents. See the Resources page Help topic, then specifically review the "Install or Upgrade Remotes Agents" section.

To configure a shared workspace:

For Linux

If your agents are installed on Linux, change the default workspace to point to a network location accessible to both machines:

1. Select the Cloud tab, then select the Workspaces subtab, to see a list of workspaces.
2. Select the workspace named "default" to edit the default workspace.
3. Change the value of the UNIX Path to a network location accessible to the ElectricFlow server and agent machines.
4. Make sure this location is readable and writable by the users chosen during the server and agent installations.

After fulfilling these scenario prerequisites, you are ready to proceed.

For Windows

If using a Windows environment, the shared workspace is created for you already.

Overview

This scenario guides you through creating a multi-agent build and test process integrated with your SCM system, build utility, and unit or system tests. You can invoke this build using a schedule, Continuous Integration (CI), or "on demand". At the end of this scenario you will be familiar with the following ElectricFlow concepts and features:

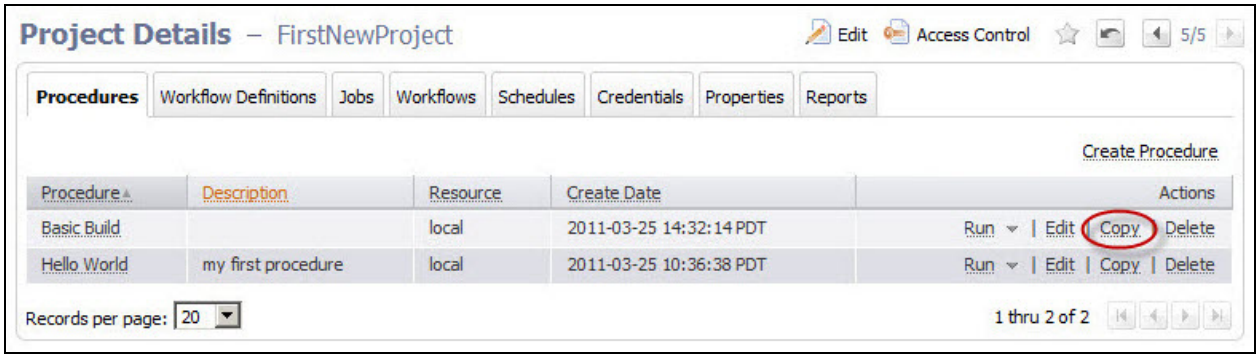
1. Agent-only installation
2. Copying a procedure to create a new procedure
3. Running steps in parallel
4. Pools
5. Post processor
6. Search

Begin Scenario 4

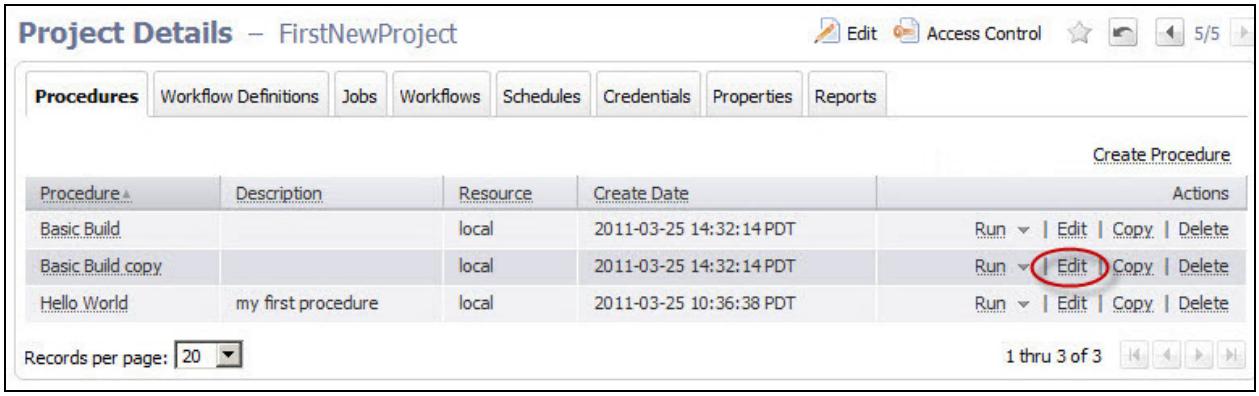
Step 1. Create a new procedure

This time, we are going to *copy* an existing procedure, rename it, and add another step.

1. Select the Projects tab, then select the FirstNewProject project name.
2. On the Project Details page, click the **Copy** link to copy the Basic Build procedure.



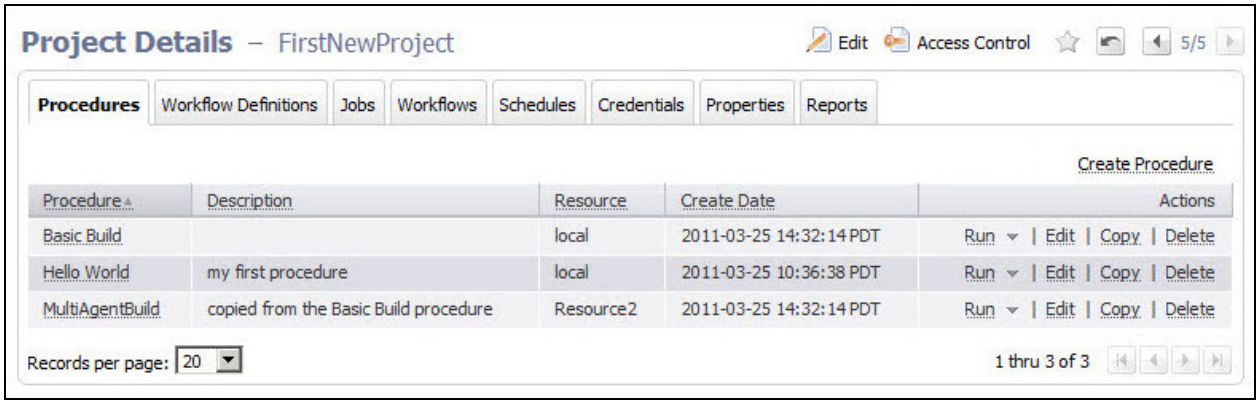
3. When the Basic Build copy procedure is displayed, you can click its name to go to the Procedure Details page to verify it contains the same contents of the Basic Build procedure.
4. Return to the Project Details page.



5. Select the **Edit** link for the Basic Build copy procedure.
6. On the Edit Procedure page:
   - Name–Enter a new procedure name, `MultiAgentBuild`.
   - Description–Add a description noting this procedure was copied from the Basic Build procedure.
   - Default Resource–Select your new resource, `Recource2`, for this procedure.

7. Click **OK** after entering your information and ElectricFlow will take you back to the Project Details page.

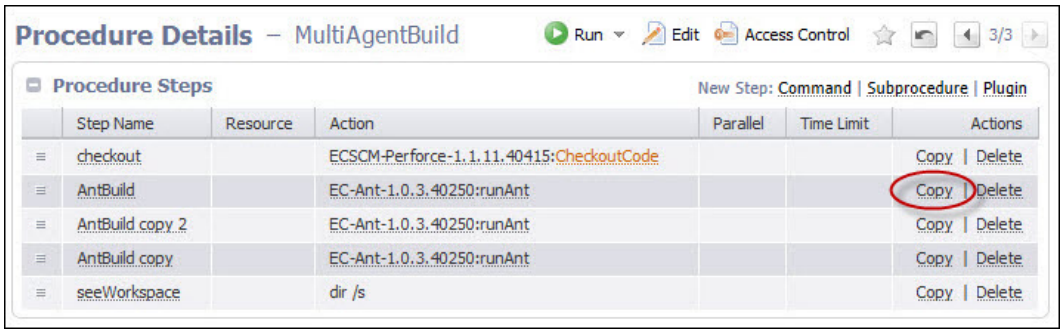
   Your Project Details page should now look similar to the following example:



This page now displays the new MultiAgentBuild procedure with a description, and a new resource is designated for this procedure to use.

Step 2. Create two new Ant steps

1. Beginning on the Project Details page, click the MultiAgentBuild procedure name to go to the Procedure Details page.
2. Click the AntBuild step **Copy** link, twice.



   Now you see two additional steps names: AntBuild copy and AntBuild copy 2.

3. Click the AntBuild copy step name to go to the Edit Step page.

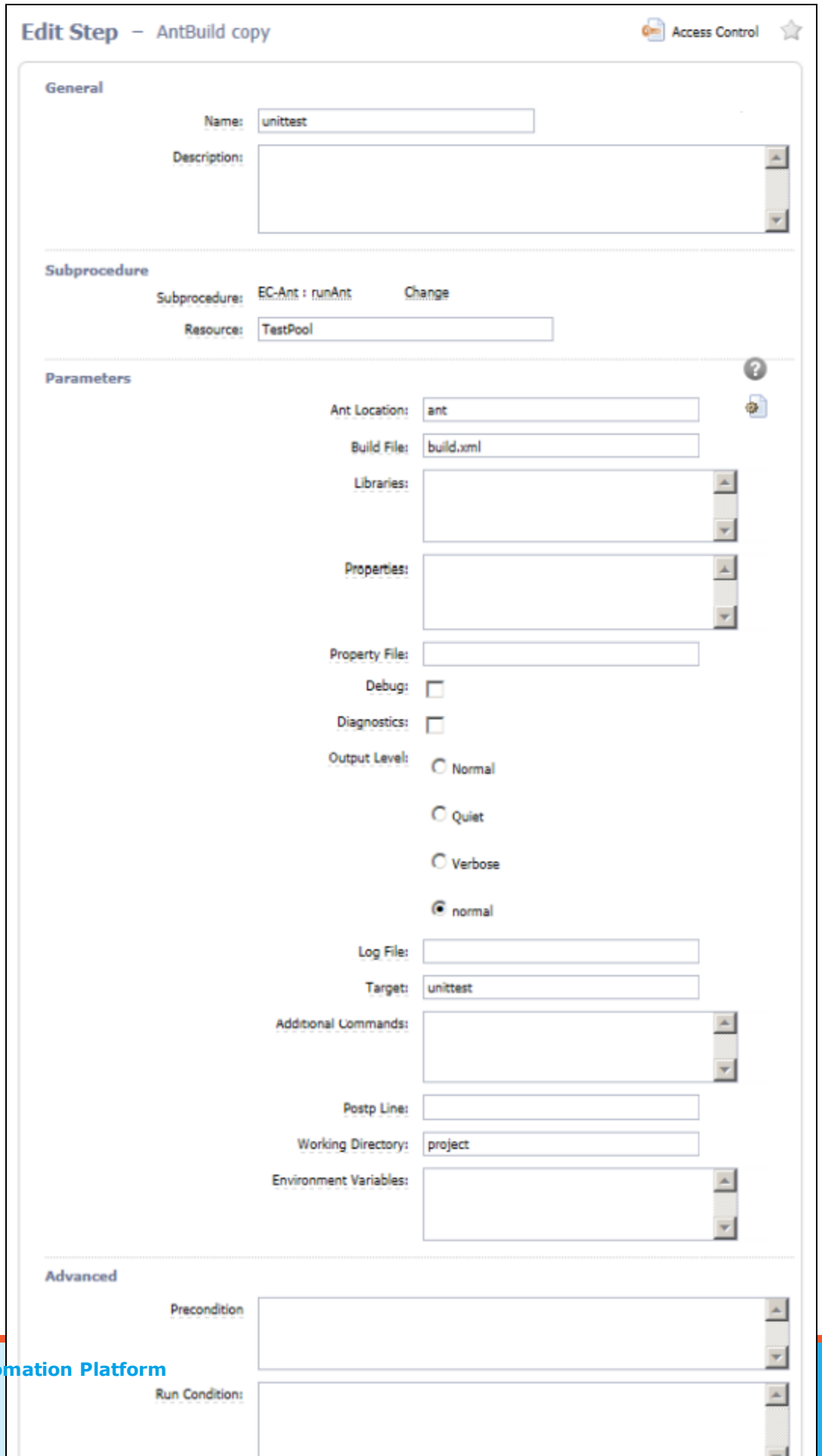
   Notice the AntBuild copy step name is adjacent to the page title because this is the step we are going to edit.

4. Modifications for creating the new Ant steps:

   Because this is an "Edit" page, some of the fields already contain the information we originally supplied to create the first Ant step. The following list contains the fields we need to edit or add new information:

   - Name–For this example, we chose `unittest` for the step name. Type over the existing text to edit the text.
   - Resource–Enter the name, `TestPool`, because we will be creating a resource pool for testing. At this point, you need to enter a pool name only (no spaces in the name).
   - Target–Edit this field to: `unittest`. In the Advanced section:
   - Run in Parallel–Click the check box to select this option. You are designating that you want this step to run in parallel with one or more other steps.

5. Click **OK** to create the first of two new Ant steps and ElectricFlow returns you to the Procedure Details page.


Edit Step – AntBuild copy
Access Control
General
Name: unittest
Description:
Subprocedure
Subprocedure: EC-Ant : runAnt Change
Resource: TestPool
Parameters
Ant Location: ant
Build File: build.xml
Libraries:
Properties:
Property File:
Debug:
Diagnostics:
Output Level:
Normal
Quiet
Verbose
normal
Log File:
Target: unittest
Additional Commands:
Postp Line:
Working Directory: project
Environment Variables:
Advanced
Precondition
Run Condition:

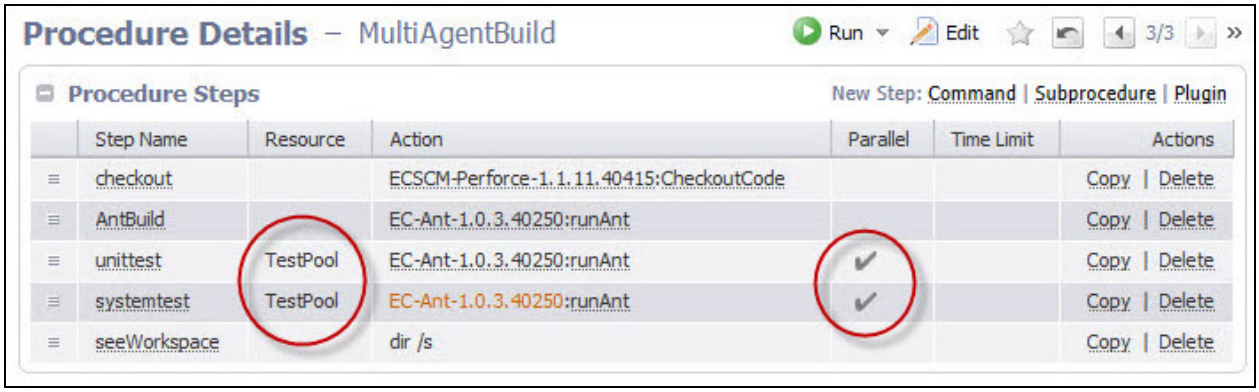
6. From the Procedure Details page, click the AntBuild copy two step name to go to the Edit Step page again.

   To create the next Ant step, make the same edits (above) as you did to create the `unittest` step, except:

   1. Name–Change the step name to `systemtest`.
   2. target–Change this field to: `systemtest`

7. Beginning on the Project Details page, click the MultiAgentBuild procedure name to go to the Procedure Details page.

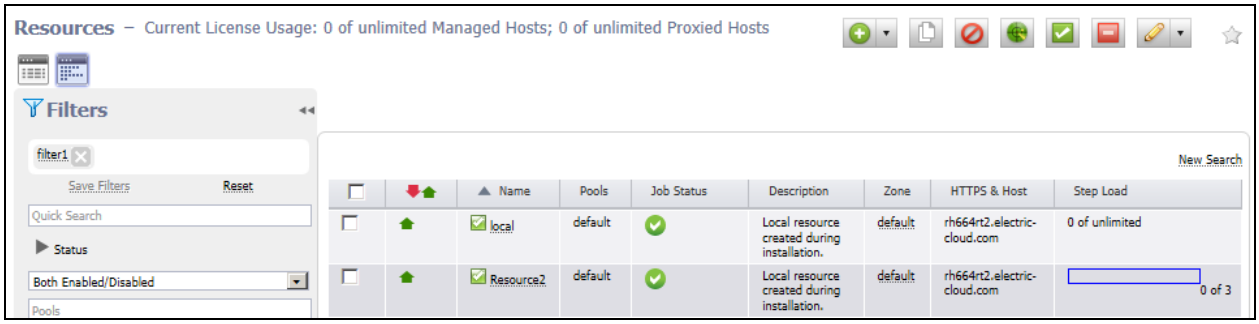
8. Click **OK** to create the second of two new Ant steps and to return to the Procedure Details page. You now have three Ant steps, including two new Ant steps, `unittest` and `systemtest`, with `TestPool` as the designated resource.



Also notice, the Parallel column now contains "checkmarks" for the 2 steps we want to run in parallel.

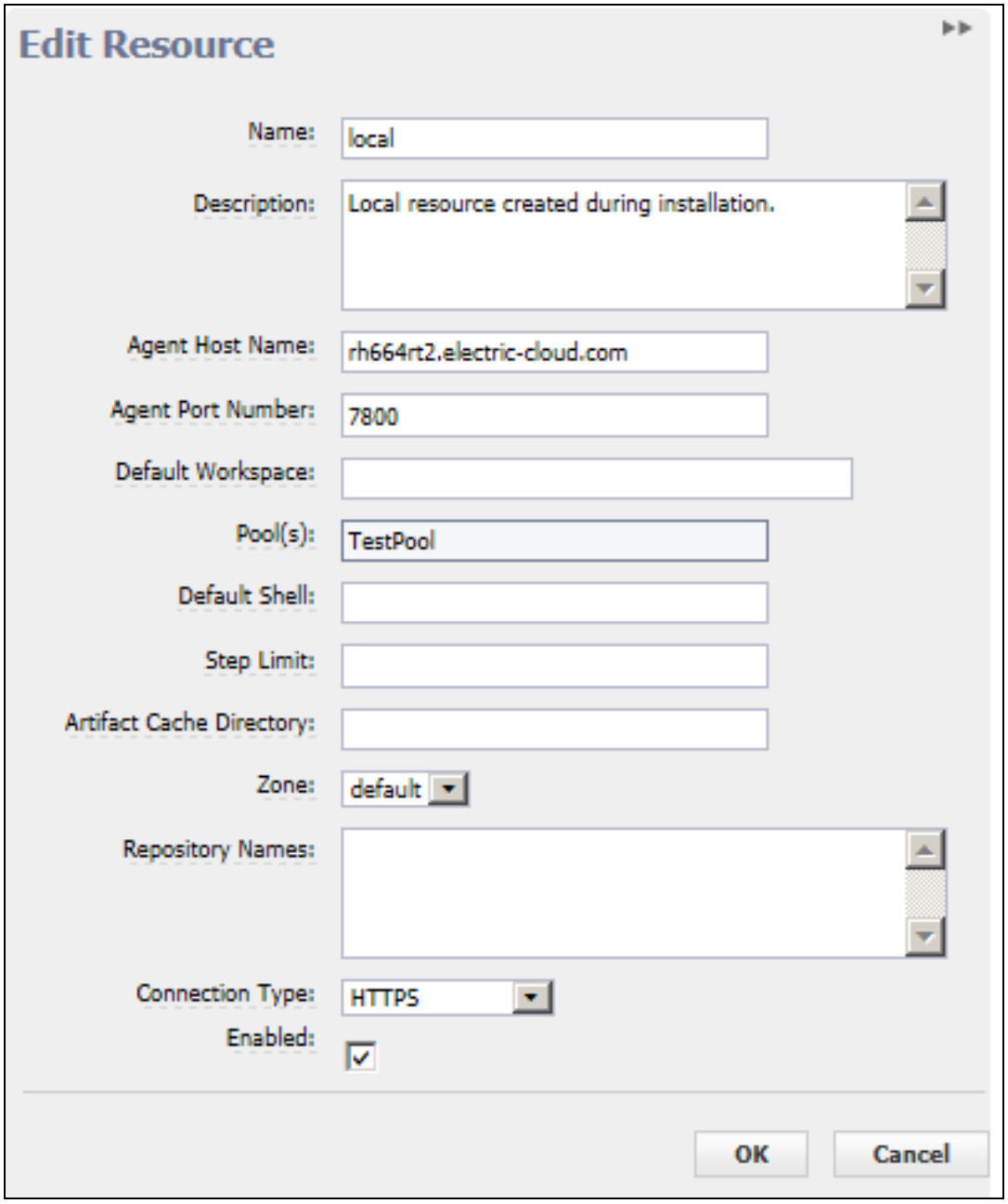
Step 3. Create a resource pool

1. Select the Cloud tab to see the following Resources page with a table listing the currently available resources.



To put these two resources in a resource pool that we previously "named" when we created the new Ant steps, we need to edit each resource.

2. **On the Edit Resource panel**, the only change we need to make is to add the `TestPool` pool name to the Pool(s) field.



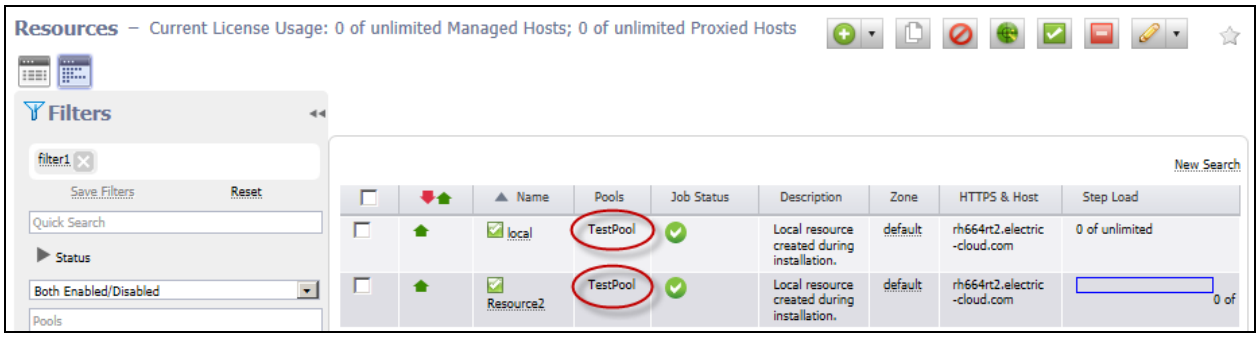
3. Click **OK** to save your change and return to the Resources page.

   Notice the Pool(s) column now has TestPool specified as a resource pool where the `local` resource is a member.

4. Repeat this process to put `Resource2` in the TestPool resource pool.

   A resource can belong to one or more pools. If any resource belongs to multiple pools, all pools would be listed in this column. For more information about the **Resources** page, see Resources on page 1208 or click the **Help** link in the top-right corner of the web page.

   The following screen example should be similar to your Resources page.



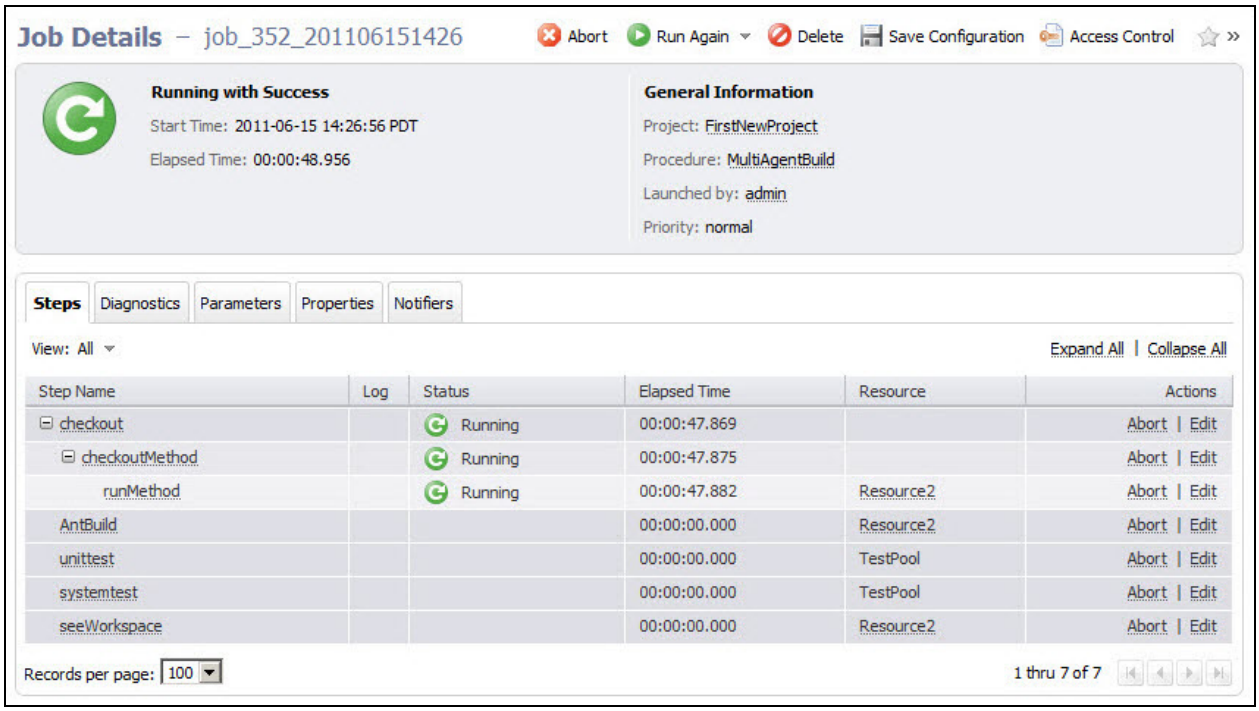
Step 4. Run the procedure

To run the new MultiAgentBuild procedure, select the Projects tab > FirstNewProject project name > MultiAgentBuild procedure.

On the Procedure Details page:

1. Click the **Run** link at the top of the page.
2. Click **Run** on the Run Procedure page.

   Your Job Details page should be similar to the following example.



Note: The screen example above illustrates a job that is still running—notice the green circular-arrow icon at the top of the page. This icon changes to a check mark if this job completes successfully.

Scenario extension—Using postp

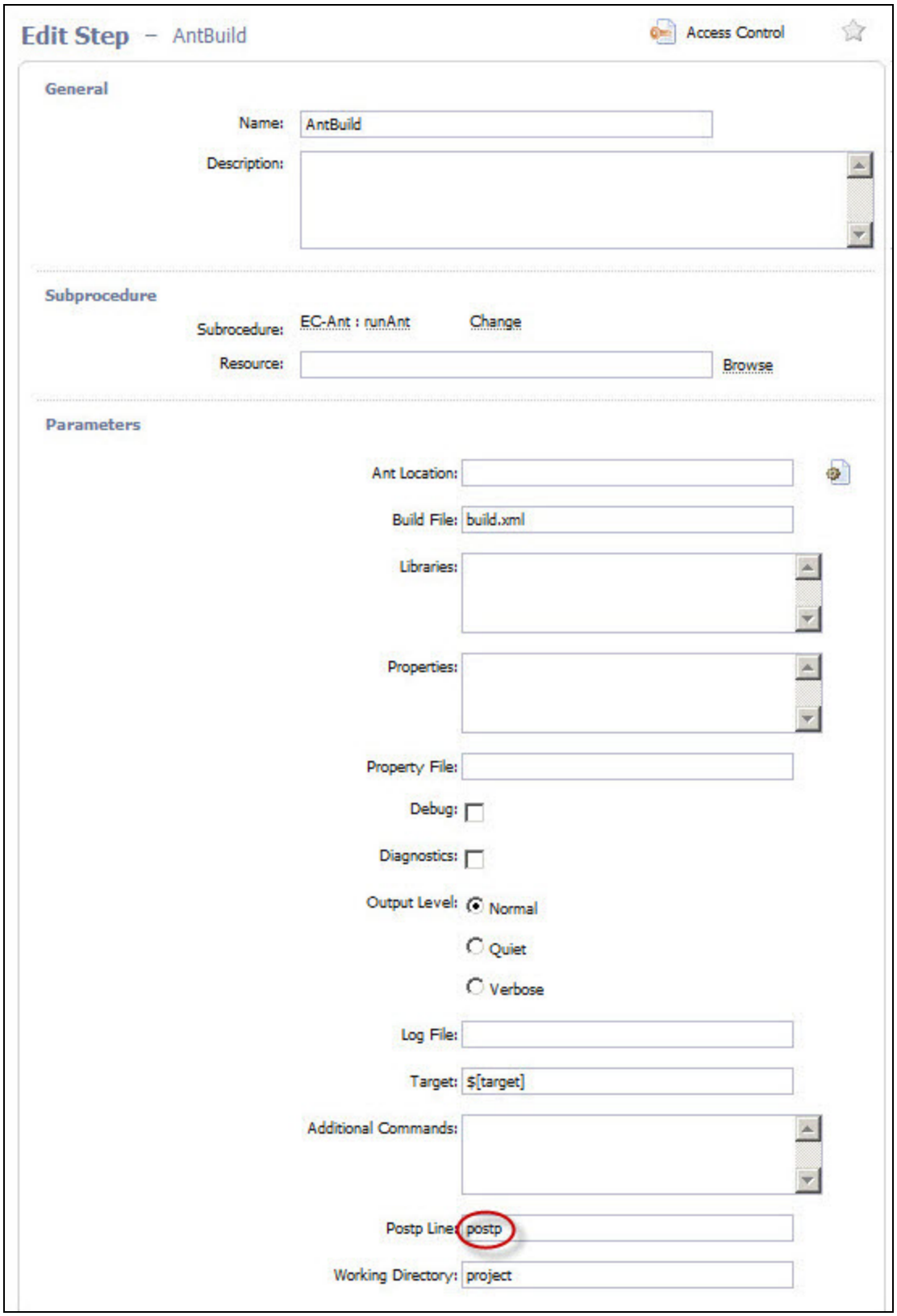
ElectricFlow implements data collection with a postprocessor. The postprocessor is a command associated with a particular procedure step. If the postprocessor is specified for a step, it executes concurrently with the main step command. The postprocessor runs on the same machine as the main command and in the same working directory, and it retrieves the log file from the step as its standard input.

The standard ElectricFlow postprocessor is called ***postp***. ***postp*** scans the step's log file looking for interesting output such as error messages and then sets properties on the job step to describe what it found. For example, postp might create a property named "errors" whose value is the number of error messages in the log file, or a property named "tests" that counts the number of tests the step executes. Also, postp can extract portions of the step log that contain useful diagnostic information and save this information for reporting.

When you create a step, you can specify `postp`. If you already have a step and want to add postp reporting, you just have to edit the step to include postp.

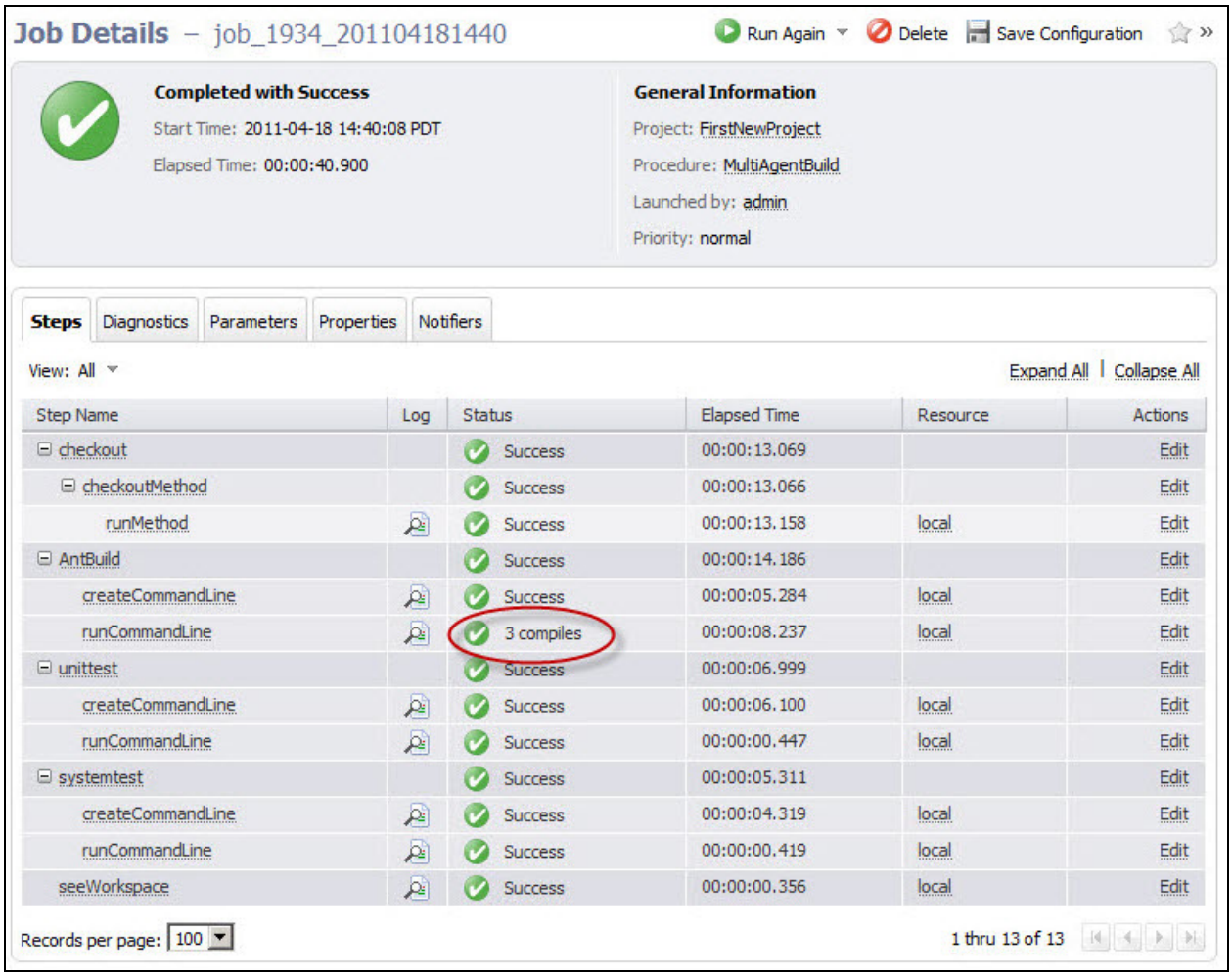
1. From the previous Job Details page screen example, click the **Edit** link for the AntBuild step.

2. On the Edit Step—AntBuild page, in the Parameters section, add `postp` to Postp Line field.


Edit Step – AntBuild
Access Control
General
Name: AntBuild
Description:
Subprocedure
Subrocedure: EC-Ant : runAnt Change
Resource: Browse
Parameters
Ant Location:
Build File: build.xml
Libraries:
Properties:
Property File:
Debug:
Diagnostics:
Output Level: Normal
Quiet
Verbose
Log File:
Target: $[target]
Additional Commands:
Postp Line: postp
Working Directory: project

3. Click **OK** to save your edit and return to the Job Details page.
4. Run this job again to see the postp result—click the **Run/Run Again** link.

   On the next screen example, notice the AntBuild step now shows "3 compiles" in the Status column. Using postp supplies additional information.



For more information about using postp, see Postprocessors.

Summary

This scenario demonstrated how ElectricFlow can drive a complete build and test process—including building steps to run in parallel and creating a resource pool. This scenario also provided a brief look at how you can use ElectricFlow to execute rapid root cause analysis by reviewing postp diagnostics.

Automation Platform Setup

The topics in this section describe how to set up the automation platform UI. After completing the tasks, you can create, configure and manage the objects needed to automate your build-test processes, application or microservice deployment, and pipelines.

Automation Platform Home Page

Overview

This page (https://<*ElectricFlow_server*>/commander/) provides a convenient console for running jobs and viewing results.

> **Note:** When using the EC-Homepage plugin to share your Home page sections, you will not see certain links that are normally available. For example, if a particular Jobs Quick View category is shared, you cannot delete or modify that category. You can perform these functions only if the category belongs to you alone.

For information on sharing your Home page configuration, see Home page configurations.

Job Configurations

Procedures in ElectricFlow can contain complex sets of parameters— making it difficult to remember the correct parameters for a particular situation and tedious to re-enter those parameters every time the procedure is invoked. Job Configurations provide a one-click solution to this problem. When you create a job configuration, you enter all the information needed to run a procedure, including parameters or a credential. All job configurations are displayed here on the Home page. Invoke a particular configuration by clicking its name in the Job Configurations section.

Create Job Configurations three ways

- On the Home page, create a job configuration from "scratch" by clicking the **Create** link in the Job Configurations section. In the Create Configuration pop-up menu, select the project and procedure you want to use for creating this configuration.
- From the Job Details page for a previously invoked job, click the **Save Configuration** link at the top of the page. Your saved job configuration is displayed on your Home page.
- From the Edit Schedule page, click the **Save Configuration** link at the top of the page. Your saved configuration is displayed on your Home page.

Shortcuts

Use shortcuts to save frequently visited ElectricFlow web pages, so those pages are immediately accessible. You can create a shortcut to any page on the web also.

Create Shortcuts two ways

- Mouse-over the "star" icon at the top of any automation platform page and click "Add current page" to add that page to the shortcut list. Mouse-over the star icon again and click "Remove current page" to remove the shortcut for that page. The star icon is yellow for pages saved as a

shortcut and gray for those pages not saved as a shortcut.

- Click the **Create** link in the Shortcut section and provide a name and URL to create a shortcut.

To modify or update a shortcut, click the **Edit** link adjacent to the shortcut you want to change.

Shortcuts can be accessed conveniently from any automation platform web page. Mousing-over the star icon displays a list of shortcuts saved by the current user. Click on a shortcut name to view the page.

> **Note:** The Shortcut section may contain static entries that cannot be deleted. And if you "share" your Home page, the Edit link will not be available for shared items.

Jobs Quick View

Perhaps only a few jobs on this server are of interest to you. For example, you may care about jobs you launch manually and official builds for the products you work on, but you may not care about production builds for other products or personal jobs for other users.

The Jobs Quick View allows you to define job categories that are interesting to you.

The Home page displays the most recent jobs in each category, and you can easily click-through to get more details about any of those jobs. For example, clicking on a job name takes you to that job's Job Details page.

Create a job category

- Click the **Add Category** link in the Jobs Quick View section.
  After creating a category, results are displayed on the Home page. Clicking the **Details** link displays a summary to the right of the category. In addition to job status and other diagnostic information, the summary displays running steps and failed steps (containing errors and warnings).
- Click the **Modify** link to edit the Jobs Quick View category or the **Delete** link to remove that job from the Jobs Quick View category.
- Click on the **Details** link to see job summary information—the summary remains visible, regardless of mouse location, until you click somewhere else on the page.

If you "share" your Home page, the Modify and Delete links will not be available for shared items.

Reports

You can configure reports you would like to see on a regular basis and display a "thumbnail" report graphic in this section.

- Click the **Add Report** link to go to the New Reports page.
  After filling in the information on the New Reports page, you will see a thumbnail view of your report (after it runs) on your Home page.
  Note: If the drop-down menu on this page is empty, click the **Help** link on the New Report page for more information.
- Click the Collapse/Expand (+/-) box to see the full thumbnail report or just the report title.
- Click the **Rename** link if you need to rename your report.
- Click the **Remove** link if you need to remove this report from your Home page.

Note: If you "share" your Home page, the Rename and Remove links are not available for shared items.

Customizing the ElectricFlow Platform UI

You can customize the ElectricFlow platform UI to get a more intuitive, task-specific user interface.

- Customizing parameters
- Customizing tab layouts
- Home page configurations

Customizing parameters

Two types of objects contain formal parameters – procedures and state definitions. For these objects, you can customize the way their formal parameters are presented. You can reorder the parameters and set the labels, form element types, default values, tooltip Help text, and whether or not the parameter is required. When defined, this customization shows up whenever the parameters are displayed—running a procedure, creating a step to call a procedure, creating a transition definition to a specific target state, and so on.

How do you customize parameters?

Create a property named `ec_parameterForm` on the object containing formal parameters (the procedure or state definition). The value of this property is an XML-style specification of form elements that map to the parameters. This property must be in sync with the formal parameters, which means all formal parameters must have a corresponding XML element, with no "extra" XML elements.

Custom parameter form contents

The value of the property in XML: Under a top-level element called `<editor>`, you need an element called `<formElement>` for each parameter. Under a `<formElement>`, you can specify the following tags:

- property*—the name of the formal parameter
- type*—`entry|textarea|select|radio|checkbox|project|savedSearch|credential`
- label*—displayed next to the element
- value—the initial value of the element (does not apply to credentials, which must be specified at runtime)
- required—if true, the user must enter a value for the element
- documentation—text displayed in the tooltip when the form element is "moused" over
- numRows—valid for `textarea` elements; represents the height of the element
- option—a single option for `select|radio` elements; at least one is required
  - name*—the text displayed in the option
  - value—the value of the option
- checkedValue—valid for `checkbox` elements; the value of the element when it is checked
- uncheckedValue—valid for `checkbox` elements; the value of the element when it is unchecked

- initiallyChecked—valid for checkbox elements; whether or not the element is "checked" by default

Note: An asterisk (*) in the list above indicates a *required* tag.

Example

The following example is a parameter form that rearranges parameters and sets the labels, descriptions, default values, and form elements for each parameter.

```
<editor>
    <formElement>
        <label>One:</label>
        <property>one</property>
        <documentation>The first parameter.</documentation>
        <type>entry</type>
        <value>Test value</value>
    </formElement>
    <formElement>
        <label>Two:</label>
        <property>two</property>
        <documentation>The second parameter.</documentation>
        <type>textarea</type>
        <required>1</required>
    </formElement>
    <formElement>
        <label>Three:</label>
        <property>three</property>
        <documentation>The third parameter.</documentation>
        <type>select</type>
        <option>
            <name>ABC</name>
            <value>abc</value>
        </option>
        <option>
            <name>XYZ</name>
            <value>xyz</value>
        </option>
        <value>xyz</value>
    </formElement>
    <formElement>
        <label>Four:</label>
        <property>four</property>
        <documentation>The fourth parameter.</documentation>
        <type>radio</type>
        <option>
            <name>First Option</name>
            <value>123</value>
        </option>
        <option>
            <name>Second Option</name>
            <value>456</value>
        </option>
        <value>123</value>
    </formElement>
    <formElement>
        <label>Five:</label>
```

```
            <property>five</property>
            <documentation>The fifth parameter.</documentation>
            <type>checkbox</type>
            <checkedValue>true</checkedValue>
            <uncheckedValue>false</uncheckedValue>
            <initiallyChecked>1</initiallyChecked>
            <value>true</value>
        </formElement>
        <formElement>
            <label>Six:</label>
            <property>six</property>
            <documentation>The sixth parameter.</documentation>
            <type>project</type>
            <value>myProject</value>
        </formElement>
        <formElement>
            <label>Seven:</label>
            <property>seven</property>
            <documentation>The seventh parameter.</documentation>
            <type>savedSearch</type>
            <value>/projects/myProject/ec_savedSearches/mySavedSearch</value>
        </formElement>
        <formElement>
            <label>Eight:</label>
            <property>eight</property>
            <documentation>The eighth parameter.</documentation>
            <type>credential</type>
        </formElement>
    </editor>
```

Preserving spaces in parameter values containing only spaces

You can restore the behavior in ElectricFlow versions 4.2.*x* in which spaces are not stripped from parameter values that contain *only* spaces. To do so, add the `COMMANDER_XML_READER_STRIP_WHITESPACE_TEXT` variable to the wrapper.conf file on the server and set it to `false`:

```
set.default.COMMANDER_XML_READER_STRIP_WHITESPACE_TEXT=false
```

When set to `true` (the default), these spaces are stripped. This variable does *not* strip spaces in parameter values that contain leading or trailing spaces in conjunction with other characters.

Customizing the tab layout

Overview

A "view" defines the layout of tabs in the ElectricFlow web UI. One or more tab views may be defined at the server, group, and user level. The default view the user sees when they first log in can be set at the server, group, and user level. The default defined on the user takes precedence over its groups and the groups take precedence over the server. Electric Cloud provides a system default view that will always show up in the user's list. Views can inherit from each other and add/modify/remove/reposition tabs and subtabs as needed.

If you define a custom view, you have to manually add tabs such as Continuous Integration so that they will appear in your custom view.

View definition syntax

A view is an XML property. The following is a list of elements that can be defined in a view:

- tab—a top level tab
  - label—the text to display for the tab
  - url—the target URL of the tab
  - accesskey—the keyboard shortcut to access this tab (a single letter)
  - position—the position of this tab relative to the other tabs (first tab is 1, second tab is 2, and so on)
  - show—when inheriting from another base view, if 0, the tab is not visible, if 1 it is visible (default 1)
  - tab—a subtab displayed within this tab
    - label—the text to display for the subtab
    - url—the target URL of the subtab (relative to ElectricFlow base)
    - position—the position of this subtab relative to the other subtabs (integer greater than 1)
    - show—when inheriting from another base view, if "0", the tab is not visible, if "1" it is visible (default 1)
- base—the name of a view from which to inherit tab and subtab definitions

The following is a basic definition of a two-tab view where one of the tabs has three subtabs.

```
<?xml version="1.0" encoding="utf-8"?>
  <view>
    <tab>
      <label>Home</label>
      <url>home.php</url>
      <accesskey>h</accesskey>
    </tab>
    <tab>
      <label>Administration</label>
      <url>workspaces.php</url>
      <accesskey>a</accesskey>
      <tab>
        <label>Workspaces</label>
        <url>workspaces.php</url>
      </tab>
      <tab>
        <label>Directory Providers</label>
        <url>directoryProviders.php</url>
      </tab>
      <tab>
        <label>Licenses</label>
```

```
        <url>licenses.php</url>
      </tab>
    </tab>
  </view>
```

The following is a view that inherits from the above view and:

1. Changes the URL of the Home tab
2. Adds a new tab called Projects and places it at the beginning of the list
3. Changes the accesskey of the Administration tab
4. Moves the Workspaces subtab to the end of the list
5. Hides the Licenses subtab

```
<?xml version="1.0" encoding="utf-8"?>
<view>
   <base>firstView</base>
   <tab>
      <label>Home</label>
      <url>customizedHome.php</url>
   </tab>
   <tab>
      <label>Projects</label>
      <url>projects.php</url>
      <accesskey>p</accesskey>
      <position>1</position>
   </tab>
   <tab>
      <label>Administration</label>
      <accesskey>z</accesskey>
      <tab>
        <label>Workspaces</label>
        <position>3</position>
      </tab>
      <tab>
         <label>Licenses</label>
         <show>0</show>
      </tab>
   </tab>
```

```
</view>
```

Storing views

Tab views are stored in the server, group, and user property sheets. The view definitions are stored in a property sheet called `ec_ui/availableViews`. The name of the view is set to the property's description if defined, otherwise it is set to the property's name. The value of the view property is the XML definition document described above.

Default views

The property `ec_ui/defaultView`, if set, determines the default view for users inheriting from that object.

- If this property is set on the server, it is the default for all users.
- If this property is set on a group, all users belonging to that group will see that view, overriding the server's default if it is set also.
- Finally, a user can explicitly set their view by defining the property on their own property sheet. If the user belongs to multiple groups that define defaults, then ElectricFlow chooses the first group alphabetically.

The user can set their view by clicking on their name in the navigation bar, then clicking on the **Edit Settings** link. The list of views shown here comprise all views defined in the `ec_ui/availableViews` property sheets for the server, groups to which the user belongs, and the user itself.

If views have the same name at different levels, then the user overrides the group which overrides the server. If a user selects a view and chooses to Save, then the `ec_ui/defaultView` property on their sheet is set accordingly.

Two special values are always available in this list: "EC Default" and "Inherit". The former is the default tab layout defined in the ElectricFlow web UI. The latter means to use the first view in the list. By default, all users inherit their tab view.

For more information on user settings, see the Edit User Settings Help topic.

Developing and troubleshooting

If you create an invalid view definition on a group or server property sheet, you will break the UI for all users who inherit that view. The best practice for developing tab views is to use a "test user". Store the view definition in the `ec_ui/availableViews` property sheet for that user, and set the `ec_ui/defaultView` for that user to the name of the new view. When the view is working properly and ready for other users, you can migrate the new view to a group or to the server.

If you continue to have difficulty and tab definitions are not working correctly and can no longer get to the user settings page, revert to the ElectricFlow default by running the following command:

```
ectool setProperty /users/$USERNAME/ec_ui/defaultView ec_default
```

When you log out and then log back in, you will have the default view again. Note that changes to views are not visible until the next time a user logs in.

Home page configuration

The Automation Platform Home page (https://<*ElectricFlow_server*>/commander/) is your configurable Dashboard. This page allows you to manage shared configurations and you can customize

this page for your work preferences also. Refer to The Home page Help topic for details on creating Job Configurations, Shortcuts, Jobs Quick View, and Reports.

Shared Home page configurations can be set globally or for specific groups of users only. The general model for creating shared configurations is:

1. Set up your personal configuration the way you want it to appear to others.
   To get to the configuration page, select the Administration tab > Plugins > and click the **Configure** link for EC-Homepage.
2. Click **Save** to make the configuration available to a specific group or globally to everyone on the server.

   If you published the configuration to a specific group, no further action is needed. If you published the configuration globally, continue to the next step.
3. Copy and paste the Tab XML string into a specific user's or group's view.

   This adds the public configuration to that user's or group's own home page and no further action is required for the user or group. If this step is not completed, additional action is required as described in Installing the Home page on page 1029.

You can alter an existing shared configuration by loading from a previously saved location, make changes, then save the changes back to the original or an alternate location.

User settings

Use the Backup Settings action to save and restore your personal Home page configuration.

- Select **Create** to back up and temporarily set aside your personal settings while you create or update shared settings.
- Select **Restore** to retrieve your personal settings after updating shared settings.

Location

Select Global to share the new Home page with everyone on the server, or select Group to share with specific users only.

Installing the Home page

If a globally available Tab XML string (from the Configure EC-Homepage page) has ***not*** been pasted to the user's or group's view, it must be added to an existing view definition to replace the standard Home page with the one provided by this plugin. To do this, click **Load**. This replaces the current home page with the one that's publicly available.

Reconfiguring the “Contact Support” Link

To redirect the Electric Cloud Support URL (available from the Help link on each ElectricFlow web page) to your own support center, you can create a file to add to the ElectricFlow installation directory at `<ElectricFlow Install Dir>/apache/htdocs/commander`. Create a file named `config_user.php` with the following contents:

```
<?php
$config["contactSupportUrl"] = "your URL";
?>
```

Replace `your URL` in the example above with the URL for your customer support site. If no value is supplied, a blank window appears.

Note: You must restart the ElectricFlow web server for your change to take effect.

Configuring ElectricFlow

Use this page for quick access to configuring resources, workspaces, email configurations, your source control system, and defect tracking to communicate with ElectricFlow.

Note: If you are viewing this Help topic from the "Configuring ElectricFlow" web page, use these instructions:

Click the **Add** link for any object you want to configure and that objects configuration page is displayed.

- Each item you configure will be listed in this table for your easy reference.
- Configure objects listed in this table are linked to their respective **"Edit..."** page in the event you need to modify any values previously supplied. Select a configured object to edit its values.

The table below provides a brief description of ElectricFlow configuration tasks.

Setting Up Resources	Before resources can be set up, you need to know which agent machines are available to allocate to jobs. The ElectricFlow Resource web page displays all resources currently available to the ElectricFlow server. This section also describes how to run the resource scheduler in multiple threads to parallelize scheduling, which potentially increases scheduler throughput (and thus overall ElectricFlow server throughput).
Setting Up Workspaces	ElectricFlow provides each job step with an area on the disk it can use for "working files" and results. This disk area is called a job *workspace*. A job step can create whatever files it needs within its workspace and ElectricFlow automatically places files, such as step logs, in the workspace. The location of the job step workspace is displayed on the Job Details web page.
Setting Up Email Configurations	If you do not create an Email Configuration, you will be unable to send email notifications to individuals or groups. If you have multiple users or groups in remote locations who use a different mail server, you will want to create additional email configurations to accommodate those locations if they need to receive notifications.
Setting Up a Source Control Configuration	ElectricFlow needs to communicate with your Source Control system (SCM) if you intend to use the CI Dashboard for continuous integration schedules to start a new build based on new/modified source control checkins, or if you plan to configure Preflight builds (to build and test code changes before those changes are "committed").

Setting Up Directory Providers	ElectricFlow uses account information from multiple sources. In most cases, the primary account information source is an external LDAP or Active Directory repository: both user and group information is retrieved from the repository. Local users and groups can be defined within ElectricFlow.
Setting Up Defect Tracking	ElectricFlow uses plugins to integrate with numerous defect tracking systems. If the plugin you need for your defect tracking system is not automatically installed with ElectricFlow, see the Plugin Catalog to find the integration you need.
Setting Up a Separate Web Server	Using the **Web Server Host** server setting ensures correct URLs in email notifications from the ElectricFlow server if the Apache web server name differs from the ElectricFlow server name. For example, if the servers are on separate machines.

Web Interface Online Help System

Use the automation platform online Help system for more information. Click the **Help** link in the top-right corner of any product web page to see a specific Help topic for that page.

When the Help system opens, we recommend reviewing the Help table of contents. All Help folders above the Web Interface Help folder are user-guide style Help topics that provide more detailed information on each of their topics. If you generally prefer to use a command line rather than the ElectricFlow web interface, you will find complete ectool (the ElectricFlow command-line tool) commands and arguments here too.

Setting Up Resources

Select **Cloud** > **Resources** to view the Resources page. Before resources can be configured, you need to know which agent machines are available to allocate to jobs. The Resources page displays all resources available to the ElectricFlow server. This page is your Resources management center where you create, modify, or delete resources and see the resource status.

To view the Resource Pools page where you can manage the resource pools in ElectricFlow, go to **Cloud** > **Pools**. To view the Zones page where you can manage the zones in ElectricFlow, go to **Cloud** > **Zones**.

What is a Resource?

A *resource* is an agent machine where steps can execute. In addition:

- Each resource has a logical name.
- Each resource refers to an agent machine by its host name.
- Multiple resources can be defined on the same physical host.
- Each resource can be assigned to one or more pools, or one zone.

- Each resource has a step limit that determines the maximum number of steps that can execute simultaneously on the resource.

  In Create/Edit Resource panels, you define the number of steps that can run on a resource. Remember that setting Step Limit to "0" or leaving the field blank defaults to no step limit.

  When ElectricFlow allocates resources for a job and steps are executed on the resource, it considers the step limit, not the CPU load from the steps . The step limit for a resource in your system depends on the agent machine performance. For example, using *unlimited* as the step limit for a heavy load can slow the agent and cause less consistent agent performance if this is not managed properly.

What is a Resource Pool?

A resource pool is a group of interchangeable resources. For example, a pool of Windows servers. Naming a pool in a procedure step lets ElectricFlow assign any resource from that pool to do the work for that step. By default, ElectricFlow tries to balance the requests against the set of resources in the pool, but you can also define rules for how resources are selected.

To define a resource for a procedure step, you can:

- Specify a resource or a pool name in a procedure step to execute that step on that resource/pool.
- Define custom properties for your resources or pools.
- Mark a resource as "exclusive" (the step acquires and retains its resource exclusively).

For example, if configuration information varies from resource to resource, you can use custom properties to hold this information and then reference it from procedure steps where it is needed. Suppose you have a pool of test machines where the test hardware is in a different location on each machine. You could define a property named "testLocation" on each resource, then pass this property to procedure steps using a reference such as `$[/myResource/testLocation]`. This approach lets you define procedure steps to run on any resource and automatically handle configuration differences.

> **Note:** When the ElectricFlow server and agents are installed on different hosts, make sure that the configuration for each agent specifies the Domain Name System (DNS) server.

Resource Categories

- Standard
- Proxy

Standard Resources

This resource category specifies a machine running an installed ElectricFlow agent on one of the supported agent platforms, as specified in the *ElectricFlow Installation Guide*, Chapter 2, "System Requirements and Supported Platforms."

Proxy Resources

This resource category requires SSH keys for authentication. You can create proxy resources (agents and targets) for ElectricFlow to use on numerous other remote platforms/hosts that exist in your environment. A proxy agent is an ElectricFlow agent, channeling to a proxy target.

- **Proxy agent**—This is an agent on a supported Linux or Windows platform, used to proxy commands to an otherwise unsupported platform. A proxy agent is an ElectricFlow agent, channeling to a proxy target.
- **Proxy target**—This is a machine on an unsupported platform that can run commands via an SSH server. Proxy targets have limitations, such as the inability to work with plugins or communicate with ectool commands.

For a step that will run on a proxy target, specify the **Working Directory** as the directory to which the step's commandFile is uploaded from the proxy agent to the proxy target via SSH.

When a step runs on a proxy resource, the proxy agent performs the following tasks:

- Uploads the `commandFile` to the Working Directory on the proxy target via SSH.
  - Working Directory—the step runs in this directory on the proxy target, which defaults to the UNIX path to the workspace
- Creates a wrapper `sh` shell script to:
  - CD to the Working Directory
  - Set COMMANDER_environment variables that exist in the proxy agent's environment
  - Run the commandFile that was uploaded earlier
- Uploads the wrapper script to the Working Directory on the proxy target
- Runs the wrapper script on the proxy target
- Cleans up script files on the agent and proxy target

Setting Up SSH Keys

For details about setting up SSH keys, see the *KBEC-00049—Setting up SSH Keys in preparation to use a Proxy Agent* KB article.

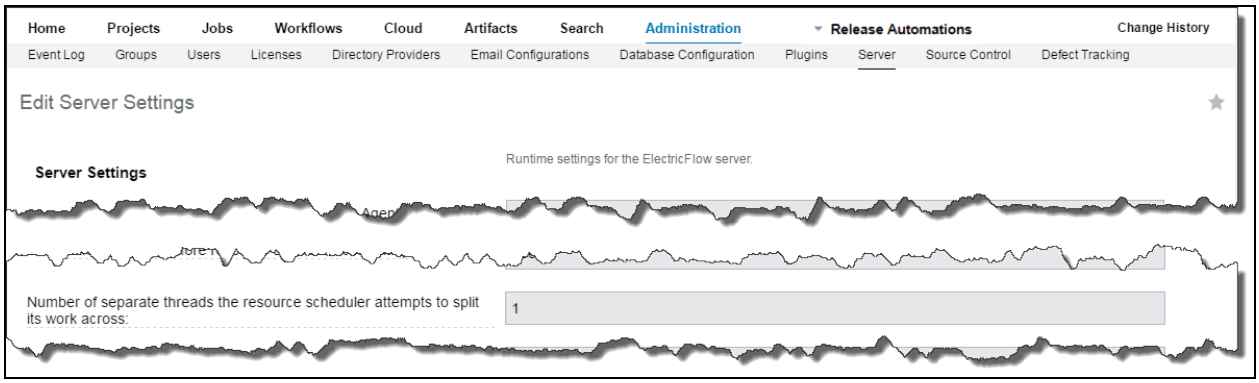
On UNIX, the ElectricFlow agent looks in the `~/.ssh` directory for the private and public key files to set up the SSH connection to the proxy target. Therefore, you should generate your key files in that directory.

On Windows, the ElectricFlow agent does not presume a default location. Therefore, in the proxy resource definition you must specify the key file location. To do so, enter the following line in the resource "Proxy Customizations" field:

```
setSSHKeyFiles('c:\ foo\priv.key');
```

Optimizing Resource Scheduler Performance

The resource scheduler is the part of the ElectricFlow server that schedules runnable job, process, and workflow steps to run on resources. The resource schedule runs as an ElectricFlow *singleton service*, so in a cluster, it runs on only one node. If your resource scheduler schedules thousands of steps per minute, you can optimize its performance by using the **Number of separate threads the resource scheduler attempts to split its work across** server setting. This setting lets you run the scheduler in multiple threads to parallelize resource scheduling; the multiple threads still run as a singleton service and thus are all on the same node in a cluster. The setting is in the **Administration** > **Server** > **Settings** screen in the Automation Platform:



The setting specifies the number of "buckets" of work that the resource scheduler attempts to divide resources into, and thus the number of threads that it runs to handle these buckets of work. The range is 1 through 8. The default is 1 (one bucket). Depending on server workload and other details, setting it to more than 1 might increase resource scheduler throughput and thus might also increase overall ElectricFlow server throughput.

For a value exceeding 1, the ElectricFlow server tries to divide the lists of enabled resources and pools into that number of buckets so that each resource is in the same bucket as all of its pools, and each bucket size (measured by the number of resources and pools in it) is about equal. The division reoccurs (as part of the same database transaction) whenever:

- A resource or pool is created, enabled, deleted, disabled, or renamed
- The list of resources in a pool is modified
- The server or a server node starts

Depending on system speed and the numbers of resources and pools, redivision could slightly delay these operations. During redivision, all the resource schedulers are paused, which delays the scheduling of steps.

> **Note:** After changing the value, you must restart the ElectricFlow server or cluster nodes so that the new value takes effect. You should restart all nodes as soon as possible after a value change to avoid problems.

Potential Performance Increase with Parallelization

Electric Cloud Test Results

In general, the potential performance increase is small for a single-server ElectricFlow installation and large for a cluster. Electric Cloud tested the sustained throughput (not the number of steps running concurrently) on a cluster with parallelization using the following scenario:

- Five schedulers on a five node cluster
- Up to 2400 agents
- Very short steps (such as `echo hello world`) to maximize the stress on the scheduler
- Several thousand steps scheduled per minute
- Peaks of several thousand steps waiting to be scheduled…
- Several hundred steps executing concurrently

The actual measured speedup was about 2.8 (limited by the speed of the database that processed the resulting steps).

Potential Speedup

A single step scheduler on a fast modern server can generally schedule 25-30 steps per second. This assumes a database round trip time of about ½ ms (which requires a good local connection to a fast database server). Five steps schedulers can at least triple that rate (and possibly more depending on database performance).

Situations Where Resource Scheduler Parallelization Might Not Be Advantageous

Parallelization might not be advantageous in the following situations.

Parallelization might not be effective if you:

- Both frequently create, enable, delete, disable, or rename resources or pools or modify the list of resources in any pool (which likely means that you have automated one or more of these operations)
- And have more than 1000 resources or pools, which causes the redivision to require about several seconds

You should not use parallelization if the resulting decrease in performance of these operations because of redivision is unacceptable. Also, any of the resource or resource pool changes as described above that cause a redivision will briefly pause parallelized resource schedulers (typically for several seconds). This briefly delays all jobs running on the server. Thus, if you modify resources so frequently that the resulting jobs delay decreases performance that exceeds the potential gain from parallelization, then you should not use parallelization.

If a Resource Pool or a Set of Overlapping Pools Contains More Than Its Share of All Resources

If you have one (enabled) pool or a set of overlapping (enabled) pools that contains more than 1/*N*th (in other words, more than its share) of all resources, then using a parallelization value greater than *N* will not divide resources into perfectly equal buckets of work. If the largest bucket exceeds about twice the size of the smallest, a warning appears in the ElectricFlow log at each redivision (such as at server startup).

The warning indicates the pool most responsible for the problem. If the warning appears, then disabling or subdividing the pool (if the pool's purpose allows it) might fix the problem. If not, you should avoid parallelization (or at least keep the value sufficiently low).

If Steps are Run Mostly on Resources in the Same Pool or Overlapping Set of Pools During Peak Workload

If during peak workload, the list of all runnable steps that can be scheduled usually contains mostly steps being run on resources in the same pool or an overlapping set of pools (most likely because they are usually broadcast steps in the same job), then parallelization is unlikely to increase performance significantly. This is because at any point time, most of the workload would still be on a single thread.

Determining the Best Resource Scheduler Setting

If ElectricFlow server performance is a concern, then unless an issue discussed above keeps you from using parallelization, you should experiment with using it. You should monitor performance (particularly the amount of time that steps spend in the "Runnable" state) and try various values to find the best one for your setup and usage pattern. Whether enabling this feature helps, and what the

optimum setting is, depend greatly on your usage pattern and how your resources and resource pools are arranged. It can even decrease performance.

For one ElectricFlow server, you should use a value of 1 or 2. For an ElectricFlow cluster, the best value is probably between 1 and a few over the number of cluster nodes. You should first try a value of 2 for one server or a value equal to the number of nodes in the cluster and then experiment from there. Any resulting performance increase depends on your usage pattern such as volume of jobs and the volume of resources.

Optimizing Oracle Database Performance When Using Parallelization

If changing the value speeds up your ElectricFlow server, it increases the load on your database server and can make the database server the performance bottleneck (particularly for a cluster). If you experiment with using a value greater than 1, you should also work with your database administrator (DBA) to optimize database performance (for example, to ensure that table/index fragmentation is under control).

Oracle database performance optimization is complex because of Oracle's numerous performance tuning tools and mechanisms. Their proper use is outside the scope of this manual. However, your Oracle DBA should know that:

- The resource scheduler interacts heavily with tables with names of the form EC_SM_*.
- The contents of these tables are short-term data that is deleted typically within a few minutes after the relevant job ends.

This means that the contents of these tables and their indexes turn over fast and are also normally empty if the ElectricFlow server is idle. Unless your ElectricFlow workload is stable and always consistent, they can thus interact poorly with the default table/index statistics-gathering mechanism used by the Oracle performance optimizer unless you "lock" the statistics while the ElectricFlow server is under typical high and extended job load. (In which case the performance optimizer will still rapidly claim that the "locked" statistics have become stale.) So on Oracle, unless your ElectricFlow workload is stable and always consistent, you should use a "locked" snapshot of the statistics for these tables and their indexes. Depending on workload and usage patterns (and especially your deletion pattern), this might also be true for several other tables in the Electric Cloud schema. Your Oracle DBA should work with you to identify those tables.

Creating a Resource

Click the Cloud tab in the ElectricFlow web interface to go to the Resources page.

At the top right-side of the table, click the down arrow next to the plus sign and select **Create Resource** or **Create Proxy Resource** link to see the New Resource or New Proxy Resource panel.

From either panel, New Resource or New Proxy Resource, click the **Help** link in the upper-right corner of the page to access the Resources Help topic in the Automation Platform to see descriptions and other information to assist with filling in the fields.

Gateways and Zones

ElectricFlow provides the capability of establishing security zones for agents and repository servers.

What is a Zone?

A zone consists of a collection of resources that can directly communicate with each other. In a large corporation, a zone could encapsulate a physical site, and communication between any two sites occurs through firewalls.

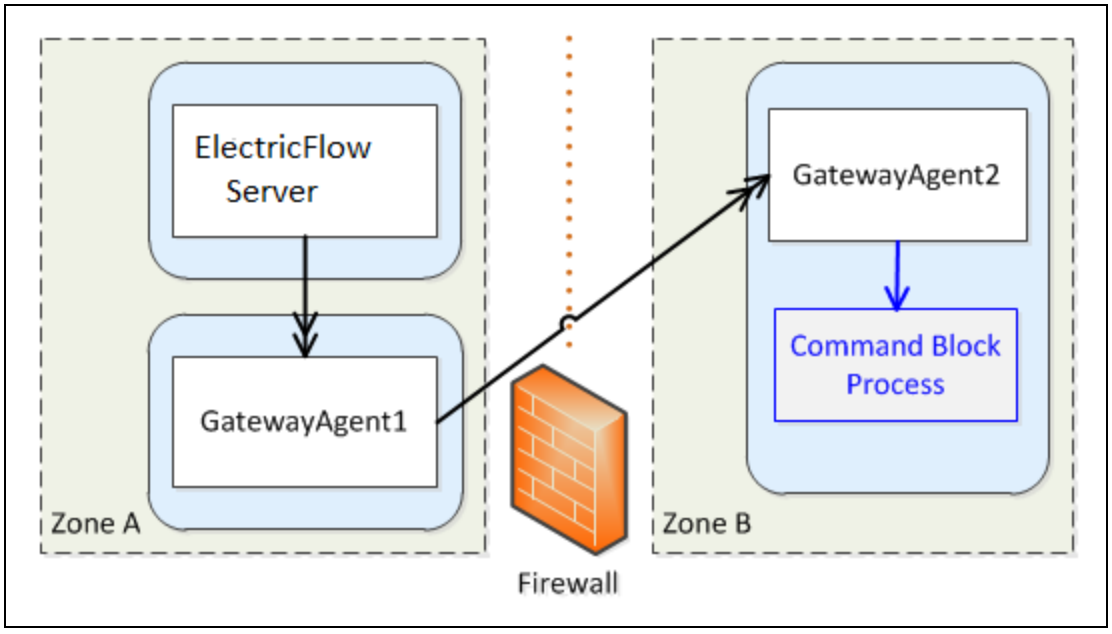
What is a Gateway?

The details of this connection information are recorded in a gateway object in ElectricFlow. Using the example in Cross-Zone Communication on page 1037 as a reference, a gateway object consists of the following information:

- Name of a gateway resource in one zone (GatewayAgent1)
- Name of a gateway resource in another zone (GatewayAgent2)
- Host/port GatewayAgent2 uses to communicate with GatewayAgent1 (for example, the firewall IP address in ZoneB)
- Host/port GatewayAgent1 uses to communicate with GatewayAgent2 (for example, the firewall IP address in ZoneA)

Cross-Zone Communication

The following figure shows an example gateway and zone configuration with two zones and three machines.



Cross-zone communication is managed by proxying requests through specially marked resources called "gateway resources". In the previous example, if the ElectricFlow server wants to run a command block process step on an agent in Zone B, it issues its request through GatewayAgent1. The request includes extra metadata that instructs GatewayAgent1 to forward the request to the target agent. The firewall is configured to allow connections from GatewayAgent1 to GatewayAgent2. It is also configured to allow connections from GatewayAgent2 to GatewayAgent1. This is necessary because API communication (CLI commands and the step completion message) is sent from GatewayAgent2 on its own outbound connection to the server (through GatewayAgent1).

Because the firewall separates two private networks, there's no guarantee that IP address ranges in one network do not overlap with those of the other. When either gateway agent wants to connect to the other, the gateway agent uses an IP address to the firewall that is valid for its side of the firewall.

Support for Gateways

The previous example shows two zones for simplicity, but ElectricFlow supports an unlimited number of zones, including chained zones. For example, if a third zone called Zone C is only accessible from Zone B via GatewayAgent3 (in Zone B) and GatewayAgent4 (in Zone C), the server could issue a request to GatewayAgent1, which forwards the request to GatewayAgent2, which forwards it to GatewayAgent3, which forwards it to GatewayAgent4.

Also, for resiliency, ElectricFlow supports having multiple gateway agents between two zones. If one gateway agent goes down, the system will detect the failure and route all requests through the other gateway agent.

Setting Up Workspaces

ElectricFlow provides each job step with an area on the disk it can use for "working files" and results. This disk area is called a job workspace.

A job step can create whatever files it needs within its workspace, and ElectricFlow automatically places files in the workspace, such as step logs. Normally, a single workspace is shared by all steps in a job, but it is possible for different steps within a job to use different workspaces. The location of the job step workspace is displayed on the Job Details page for the job under "Details" for the step.

To create a new workspace

ElectricFlow can handle any number of workspaces.

- To define a workspace from the web interface, click the Cloud > Workspaces tabs.
- On the Workspaces page, click the **Create Workspace** link at the top right-side of the table and the New Workspace web page is displayed.

Click the **Help** link in the top-right corner of the web page for assistance with filling in the fields to create a workspace. To see the Workspaces Help topic now, click Workspace—create new or edit existing workspace.

After you create a workspace specification, you will see it listed on the Workspaces page.

Setting Up Email Configurations

This topic describes how to set up an email configuration. It must be created and configured before you set up email notifications, also referred to as *email notifiers* at the platform level. Then you can send them to individuals or groups.

If you have multiple users or groups in remote locations who use a different mail server, you need to create additional email configurations to accommodate those locations if they receive ElectricFlow notifications.

Creating an Email Configuration

To create an email configuration in the platform GUI:

1. Click **Administration** > **Email Configurations** to go to the Email Configurations page.
2. Click **Add Configuration** to go to the New Email Configuration page.

3. Enter information in the fields on the page.

   There are two ways to get help in entering information in the fields:

   - Click Email Configuration—create new or edit existing email configuration to go to the Email Configuration help topic.
   - If you are in the platform GUI, click the **Help** link in the upper-right corner of the page to see descriptions and other information.

After the email configuration is created, you will see it listed in the table on the Email Configurations page.

Setting Up a Source Control Configuration

You must configure your source control system to communicate with the ElectricFlow server if you plan to takes advantage of the ElectricFlow Continuous Integration Dashboard and associated functionality and Preflight functionality—both of these features are designed to work with a number of source control (SCM) systems.

- Continuous Integration Dashboard—use this feature to create continuous integration schedules for your build environment. The Continuous Integration Manager is the front-end user interface for the ElectricSentry Continuous Integration engine.
- Preflight—use this feature to build and test developer code before it is *committed* to the code base for your product.

ElectricFlow installs (bundles) and supports numerous source control types. After creating a source control configuration, your entry will appear in the table on the Source Control Configurations web page—to see this web page, select the Administration > Source Control tabs.

If the SCM you prefer is not listed here, see the Plugin Manager page (Administration > Plugins) for a list of all currently available SCM plugin integrations available for ElectricFlow. To configure a different SCM, see the Help topic associated with that plugin.

Select one of the following links to go to the source control configuration page to configure your SCM system:

- AccuRev
- ClearCase
- File
- Git
- Perforce
- Property
- Subversion

The Continuous Integration Dashboard has its own Help topic. For more information on ElectricSentry or Preflights builds, see their respective Help topics: ElectricSentry , Preflight Builds.

Setting Up Directory Providers

ElectricFlow uses account information from multiple sources. In most cases, the primary account information source is an external LDAP or Active Directory repository: both user and group information is retrieved from the repository. Local users and groups can be defined within ElectricFlow.

To specify a directory provider

ElectricFlow includes a web page to facilitate adding your existing LDAP or Active Directory users and groups to ElectricFlow.

- To access the Directory Providers web page, select the Administration > Directory Providers tabs.
- Click the **Add Active Directory Provider** or the **Add LDAP Provider** link to add a new provider.

Depending on which **Add... Provider** link you choose, you will see either the New Active Directory Provider or the New LDAP Provider web page. From either page, click the **Help** link in the upper-right corner of the page. The Help topic describes each field on either web page so you can easily enter information in the forms to have ElectricFlow use your existing user or group account information.

To see the Directory Provider Help topic now, click Directory Providers—create new or edit existing directory providers. After adding a provider, you will see it listed in the table on the Directory Providers web page.

Enable/Disable Local ElectricFlow Users

Two server settings properties are supplied for local ElectricFlow users (both are "on" by default):

- `enableAdminUser`—if set to `'0'`, attempts to log in as the 'admin' user are denied as if the user did not exist
- `enableLocalUsers`—if set to `'0'`, attempts by a non-admin local user to log in are denied as if the user did not exist

These settings are stored in properties in the `'/server/settings'` property sheet and can be set from ectool by calling:

```
ectool setProperty /server/settings/enableAdminUser 0
```

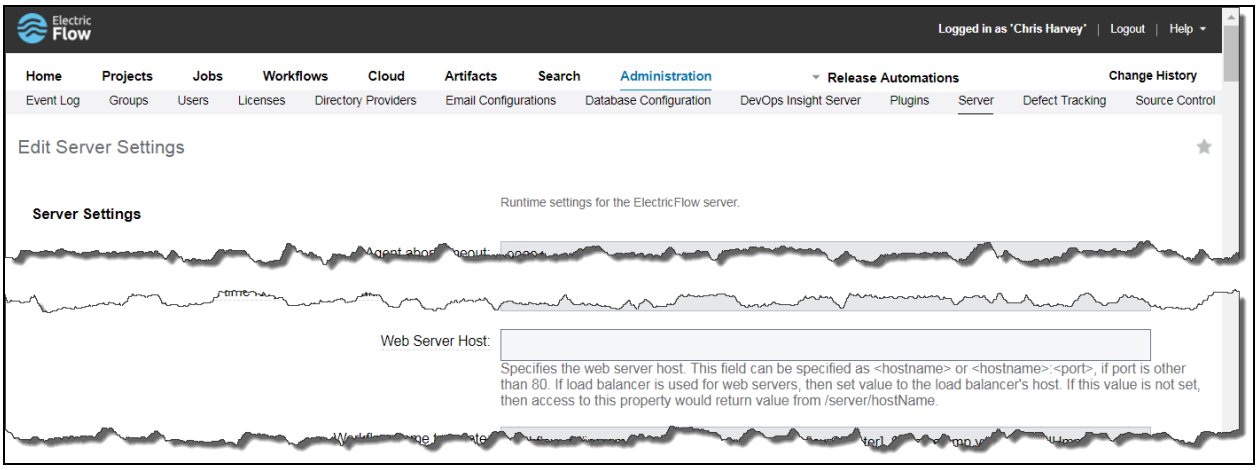
Or, these settings can be set from the automation platform web interface:

- Select the Administration > Server tabs.
- At the top of the page, select the **Settings** link.

> **Note:** Disabling these properties takes effect only for new sessions. Existing sessions continue to function "as-is" until the user logs out and attempts to log in again.

Setting Up a Separate Web Server

Using the **Web Server Host** server setting ensures correct URLs in email notifications from the ElectricFlow server if the Apache web server name differs from the ElectricFlow server name. For example, if the servers are on separate machines. This setting is on the **Administration** > **Server** > **Settings** page of the Automation Platform:



You can enter *<hostname>* or *<hostname>*:*<port>* if the port number is other than 80. If a load balancer is used for the web server, then enter the load balancer host. If this value is not set, then access to this property returns the value from `/server/hostName`.

wrapper.conf Properties

If you need to edit the wrapper.conf file, see http://wrapper.tanukisoftware.com/doc/english/properties.html for complete information about property formats.

Using ElectricFlow in Your Environment

This topic addresses some common issues you may encounter as you start using ElectricFlow and offers some tips about managing your ElectricFlow projects.

What's in a step?

One of the first questions you may ask when you start to implement your build and test processes on ElectricFlow is how to divide processes into individual steps: "Should I use only a few steps, each of which does a lot, or a large number of fine-grain steps?" Here are some factors to help you decide how many steps to use:

- **Reporting**—If you would like a separate report of success/failure for two activities, put those activities in separate steps. If you are happy to have a single report for both activities, a single step may make sense. For example, compilation and test phases should probably be in different steps because errors in the two phases will probably be handled differently. Unit tests for product

components managed by different groups may make sense in separate steps; each group can watch for errors in its step.

- **Parallelism**—If you want to use ElectricFlow to run two activities in parallel, put them in separate steps. If these activities are in the same step, ElectricFlow cannot run them in parallel.

  ElectricFlow parallelism works best at a coarse grain, such as running different sets of unit tests in parallel or compiling for different platforms. Electric Cloud does not recommend trying to do fine-grain parallelism with ElectricFlow, such as compiling every individual source file in a separate step— this is likely to be complicated and brittle, resulting in an enormous number of job steps, which will make it difficult to view results. If you would like to use fine-grain parallelism for compilation, we recommend using our ElectricAccelerator product. In this case, you would make your compilation steps large, with as much work in each makefile as possible because ElectricAccelerator automatically subdivides the work and runs as many sub-steps as possible in parallel. The more work in a makefile, the more efficient the process becomes.

- **Resources**—If two different activities need to run on different resources, they need to be in different steps. A single step runs entirely on a single resource.
- **Conditional steps**—If you want to skip portions of your process during some jobs, but execute them during others, it probably makes sense to put those activities in separate steps and use the ElectricFlow "Run condition" mechanism to decide whether they run during each particular job. However, if you are invoking programs like *make* that already allow you to choose which actions to perform, it may make more sense to have a single large step and handle conditional behavior with those programs, rather than with ElectricFlow. For example, when running *make,* you can specify the targets to be built.
- **Setup**—You might want to have a single step at the beginning of a procedure that processes the procedure's parameters and sets up the environment for the rest of the procedure. The setup step will create a snapshot of your source code or set up a ClearCase view also. After setup, the remaining steps should be easier to write because they just use information created during the setup step.

Where's the script for a step?

Typically, each step executes a script of some type, which is processed by a command language such as *cmd* on Windows, a *shell* on UNIX, or Perl.

Two ways to handle script execution

- The first approach—Enter the script directly into the command field for the step. You can specify the language interpreter as the shell for the step.
- The second approach—Store the script in a file that is part of the source code for your project, then specify a simple command for the step that invokes the command language to process the script file.

We recommend using the second approach in most cases. The main advantage of this approach is that it makes your ElectricFlow procedures more robust. For example, suppose the script needs to change as your product evolves. If you have kept the script outside ElectricFlow with the source code for the product, each product version can have its own copy of the script. When you extract the source code for the product at the beginning of a job, you also extract the scripts for its steps. You can change a step's script without worrying about its impact on other versions of the product.

However, if the script for a step is stored in the step, it is more difficult to evolve the script with new product versions. You probably still need to build older product versions, so you will worry whether the

new script for the step will work with older product versions. If the script does not work, perhaps you can modify the step's script to test the version being built and take different actions for each version—this process becomes increasingly more complex as the number of versions increases. Or, you can make a separate copy of the build procedure for each product version (more on this below), but this process also gets complicated as you acquire more and more procedure versions. We find it is easier to store scripts for steps with the product code, so script changes are handled in the same way as changes to the product.

Two cases where it makes sense to store the script for a step in the ElectricFlow step

- The first case is for the first step of a job. At this point you have not extracted the source code for the product, so you do not yet have access to any scripts stored with the source code.
- The second case is for steps with one or two commands only, or steps already specified primarily by information stored with your source code. For example, if a step is running a *make* or *ant* command, step behavior is already specified almost entirely by a makefile or a `build.xml` file for Ant. In this case, the step's script invokes a single, relatively simple command; no need to store this in a file. If you subsequently find that the script for a step is changing frequently, consider moving it out of ElectricFlow and into a file stored with your source code.

The object is to make your ElectricFlow procedures as reusable as possible, so no change is needed with every small change to your product. Even better, organize your ElectricFlow procedures so a single procedure can be used for multiple products. To do this, store product-or version-specific information with the product, not with ElectricFlow.

Environment variables

Your build and test scripts probably depend on certain environment variables having certain values. In your existing system, you probably set those values at the beginning of your script. However, in ElectricFlow you need to set those values in each step that depends on them.

The easiest way to set values is to create a short command file during the first step of the job and save it in the top-level directory of the job workspace. The command file should contain a sequence of commands to set all environment variables required by the job. Then, in each subsequent step, invoke that command file at the beginning of the step to set environment variables for that step.

This approach simplifies management of your environment variables: if you need to change a variable, you change only the setup step at the beginning of the job, and the value is reflected in all of the following steps automatically.

When do you use subprocedures?

In ElectricFlow, the action for a step can invoke another procedure, passing parameters. This is a powerful tool for structuring your processes. Subprocedures tend to be used in two ways: for *encapsulating reusable processes* and for *managing concurrency*.

- *encapsulating reusable processes*, is the preferred use. If you need to perform certain activities repeatedly in different places, you can use a subprocedure for them.
- For example, when we build and test the ElectricFlow product, we do it on multiple platforms, but the mechanism is virtually the same on all platforms. We implement the basic build and test mechanism for one platform in a subprocedure named BuildAndTest. In our main procedure for production builds, we invoke BuildAndTest multiple times, once for each of the platforms. If the mechanism changes, we only need to change it once in BuildAndTest to fix all six platforms.

- *managing concurrency*—Suppose you have three steps A1, A2, and A3, that must run sequentially, and two other steps B1 and B2 that must also run sequentially, but the two groups can run in parallel. The way to implement this is to put A1, A2, and A3 in one subprocedure named "A", and B1 and B2 in another subprocedure named "B".

  Now you can create a top-level procedure with two steps: one invoking A and the other invoking B, and mark those steps for parallel execution. As a result, A and B will run in parallel but the steps inside each subprocedure will run serially.
  If you placed all five steps inside the top-level procedure, there would be no way to achieve this effect.

How do you evolve procedures?

After you start using ElectricFlow for managing your build and test processes, it will not be long before you encounter the following situation:

You have a set of ElectricFlow procedures that work fine on all existing versions of the product, but you are about to change the product in a way that affects ElectricFlow procedures. For example, you might be adding a new component, or perhaps you are going to change the way the product is installed for testing, or perhaps you have restructured the product to allow more steps to run concurrently. As a result, you need to change ElectricFlow procedures for the current version. At the same time, you need older product versions to continue building as well.

In most cases, the easiest way to handle these situations is to leave the current ElectricFlow procedures alone and keep using them for your older product version. This ensures you will not "break" older versions. Make copies of the procedures that need to change, then modify the copies and use them for your new software version.

You will have one version of procedures for each significant version of the product. For example, at Electric Cloud, we incorporate the version number into the procedure names: "Master-1.0" is the version of the Master procedure used with the 1.0 release, "Master-1.1" is used for 1.1, and "Master" with no version number is used for our current development.

Another approach would be to make a copy of the entire project for each release. This method may be easier if other information in the project, such as property values, needs to evolve also.

The "copying" approach gets more and more complicated as you add more and more versions of the procedure. To minimize this complexity, try to structure your procedures so they do not have to change frequently. Some techniques you can use:

- Use parameters for things that do change frequently, such as the branch of software to build.
- Where possible, store scripts and other information [used by the procedure] as part of the source code for the project as described above, so this information is versioned automatically, along with your source code.

Porting existing scripts

You probably already have scripts you have been using to build and test software, before switching to ElectricFlow.

Over time, you will probably want to do quite a bit of restructuring to take full advantage of ElectricFlow. However, you probably do not need to do major restructuring to get started with ElectricFlow. Here are some steps you might take to "get up and running" with ElectricFlow as quickly as possible and gradually convert to make the best use of the system.

1. ***To begin,*** see if you can take your existing script and run it monolithically as a single step in a procedure under ElectricFlow. This will get you running and start providing some ElectricFlow benefits for resource management, error analysis, and reporting.
2. ***Next***, start dividing your previous script into separate steps for ElectricFlow. Begin with the steps easiest to separate from the rest of the script, such as the commands to compile your software. One of the challenges you face is how to pass data from one step to another. In a monolithic script, you can keep the data in variables that persist throughout the job. As you divide the script into steps, you need to figure out which variables are used only in a single step and which variables pass from step to step. For variables that pass between steps, use ElectricFlow properties to store their values. In many cases you can compute shared data values in a single step at the beginning of the job, store their values in properties on the job, then access those values read-only in later job steps.
3. ***After dividing the steps***, you can do finer-grain reporting. Start using the *postp* postprocessor to scan your log files and generate statistics and diagnostics. Over time you will probably discover you have a few error and warning messages peculiar to your site, and are not captured by postp's patterns. Learn how to extend postp with additional patterns to capture all the information that matters to you.
4. ***Now that reporting statistics are being recorded***, you can develop your own reports to summarize those statistics. You can easily use *ectool* to read statistics from the properties where they are stored.
5. ***Something else you can do after steps are divided:*** Start tuning the performance of your procedures. For example, you can start marking steps to run in parallel, and you can choose resources on a step-by-step basis to finish your jobs faster and share resources more effectively.

ElectricFlow project version control

We recommend exporting your ElectricFlow project data at regular intervals and saving it in the same configuration management system you use for your source code (use the "export" ectool command to generate an XML file representing the contents of a project or procedure). This process allows you to track project changes and also allows you to "look back" at older versions if that should become desirable. Also, you should snapshot a version of ElectricFlow project data at the time of each major software release, so you can revert to the exact ElectricFlow configuration used to generate that release if necessary.

ElectricFlow Installed Tools

Tool Name	Description
`eccert`	A command-line tool used to manage the ElectricFlow Certificate Authority (CA) and the certificates configured in ElectricFlow Server and ElectricFlow Agent installations.
`ecconfigure`	A command-line tool that can change configuration values for any locally installed ElectricFlow server, web, agent, or repository service. `ecconfigure` is a more user-friendly mechanism for configuring aspects of ElectricFlow that would otherwise require manual configuration file updates. `ecconfigure` actually manipulates relevant service configuration files on your behalf.

Tool Name	Description
ecdaemon	A "wrapper" program that can be used to start another program from an ElectricFlow job step—the "started" program will run as a daemon process. The ElectricFlow agent uses the facilities of the underlying operating system to make sure the process runs in a separate process group on a UNIX-based system, or outside of the normal "Windows Job" grouping in a Windows system. In either case, the ElectricFlow agent does not treat the process as one it should wait for or one it should try to "kill" if ElectricFlow needs to abort the step.
ecproxy	A driver script with built-in support for SSH. Every major operation can be overridden by defining a Perl function in the Proxy Customization field on the New Proxy Resource panel, available from the Resources page.
ecremotefilecopy	When ElectricFlow agents (on platforms other than Linux or Windows) run steps that create log files in a workspace the ElectricFlow web server cannot access (through Linux or Windows agents), use *ecremotefilecopy* to recreate job logs so they are visible on those ElectricFlow agents, which then enables the web server to retrieve and render those log files.
ZKConfigTool	A command-line tool that imports your ElectricFlow database configuration information into your ZooKeeper server.
ClusterInfoTool	A command-line tool that displays information on the running ElectricFlow server cluster from ZooKeeper.

eccert

A command-line tool used to manage the ElectricFlow Certificate Authority (CA) and the certificates configured in ElectricFlow Server and ElectricFlow Agent installations.

Do not use `eccert` as `sudo`, which would change the ownership of the configuration files (such as the keystore file) to the root user. These files must be owned by the user who starts the ElectricFlow services.

Usage

```
eccert [ options ] command [ arg ... ]
```

Commands

`addTrustedServer crt`	Add a server CA certificate to the agent's keystore.
`getCRL`	Retrieve the contents of the current certificate revocation list.

`initAgent [ --local | --remote ] [ options ]`	Initialize the agent keystore with a new public/private key pair. Generates the agent certificate signing request. If run on the server host, the certificate will automatically be signed by the server CA, and the CA certificate and the signed agent certificate are installed in the agent's keystore. If run on a non-server host, the signing request is left in the agent directory. If CA Cert is provided, the CA certificate is installed in the agent's keystore.
	`--local` Use the local server CA to sign the agent certificate.
	`--remote` Connect to a remote ElectricFlow server to sign the agent certificate.
	`--force` Replace any existing keystore.
	`--cname name` Use the specified name as the common name (CN) in the agent certificate subject. This is normally the fully qualified domain name used by clients to connect to the agent.
	`--altNames entries` Use the specified list of entries (comma or space separated) as the `subjectAlternateNames` list in the agent certificate. Simple names are interpreted as `dns` entries. Entries may begin with `"dns:"` or `"ip:"` to indicate the type (for example, `"ip:192.168.0.1"` or `"dns:myHost"`). If no entries are specified, then reverse DNS is used to look up the registered names of the host's IP addresses.
`initCA`	Initialize the server CA. Creates a new CA key and certificate.

`initServer [ options ]`	Initialize the server keystore. Creates and signs the server certificate. Installs the CA certificate and the signed server certificate into the server's keystore.
	`--force` Replace any existing keystore.
	`--cname name` Use the specified name as the common name (CN) in the server certificate subject. This is normally the fully qualified domain name used by clients to connect to the server.
	`--altNames entries` Use the specified list of entries (comma or space separated) as the `subjectAlternateNames` list in the server certificate. Simple names are interpreted as `dns` entries. Entries may begin with `"dns:"` or `"ip:"` to indicate the type (for example, `"ip:192.168.0.1"` or `"dns:myHost"`). If no entries are specified, then reverse DNS is used to look up the registered names of the host's IP addresses.
`list [ --agent | --server | --index [ --verbose ]`	Display certificate information for agent and/or server keystores or the CA certificate index. If no options are specified, both the agent and server keystores are listed.
	`--agent` List the contents of the agent keystore.
	`--server` List the contents of the server keystore.
	`--index` List the contents of the CA issued certificates index.
	`--verbose` Display additional details.
`refreshCRL`	Refresh the certificate revocation list from the ElectricFlow server.
`revoke index`	Revoke a previously issued certificate by index.
`signCertificate csr crt`	Sign the certificate signing request provided in file `csr` and write the signed result to the file `crt`. The request is rejected by the CA if there is a matching certificate already in the CA database.
`updateAgentCertificate crt`	Install a previously signed certificate crt into the agent's keystore.

Server Communication Options

`--server host`	Address of the ElectricFlow server. Defaults to the value of the `COMMANDER_SERVER` environment variable. If that does not exist,it defaults to `localhost`.
`--securePort port`	HTTPS listener port on the server. Defaults to 8443.

Global Options

`--help`	Print the Help message.
`--version`	Print the version message.

Examples

Example 1: Configure an agent to talk to any server (untrusted mode)

This example generates a new self-signed certificate for the agent and recreates the keystore with no trusted authorities.

```
$ eccert initAgent -force

Generating keys
Generating certificate request
    cname=<myAgent.company.com
    san=<dns:myAgent.company.com
```

Example 2: Configure an agent to accept connections only from a single remote ElectricFlow server

This example generates a new certificate for the agent that is signed by the remove server's certificate authority and installs the signed certificate and its associated trust chain in the agent's keystore. After this point, the agent will only accept requests from the specified server and will be used as a trusted resource by the server.

```
$ ectool --server myserver login admin pw
$ eccert --server myserver initAgent -remote

Generating certificate request
    cname=<myAgent.company.com
    san=<dns:myAgent.company.com
Asking server 'myserver' to sign certificate
Importing 'CA:myserver.company.com' certificate
Importing 'jetty' certificate
```

Example 3: Configure an ElectricFlow server with additional host names in the certificate

This example regenerates the ElectricFlow Server Certificate, the specified common name, and alternate subject names to allow trusted connections with multiple external `dns` names.

```
$ eccert initServer --force --cname "myServer.company.com" --altNames
"myServer,server2.company.com"

Generating keys
Generating certificate request
    cname=<myserver.company.com
    san=<dns:myserver,dns:server2.company.com
Signing server certificate
Importing 'CA:myserver.company.com' certificate
Importing 'jetty' certificate
```

ecconfigure

A command-line tool for changing values in configuration files for any locally-installed ElectricFlow server, web server, repository server, or agent. `ecconfigure` is an easier way to configure ElectricFlow settings than manually editing configuration files.

Usage

```
ecconfigure [<options>]
```

Agent Configuration Options

Option	Description
`--agentAcceptQueueSize=max`	The maximum number of pending connections the agent will queue up.
`--agentArtifactCache=path`	The directory containing cached artifactVersions.
`--agentCaFile=path`	A single file containing multiple CA certificates.
`--agentCaPath=path`	A directory containing a file for every CA, where each file's name is the CA subject name hash value.

Option	Description
--agentCertFile=path	Location of the certificate file used by the agent to support SSL connections from the server.
--agentCrlFile=relativepath	Relative path of the file containing the agent's certificate revocation list for SSL.
--agentDomainName=domain	The domain name that the agent uses for fully-qualified names.
--agentDuplicateDetectionListSize=size	The size of the list of recently seen requests used in duplicate request detection.
--agentEnableProxySettings=<1|0>	Enable (1) or disable (0) the proxy server configuration. If enabling for the first time, --agentProxyHost and --agentProxyPort must be specified.
--agentIdleConnectionTimeout=milliseconds	Idle connection timeout, in milliseconds.
--agentIdleOutboundConnectionTimeout=seconds	Idle time after which an outbound connection is closed, in seconds.
--agentIdlePostRunnerTimeout=seconds	Idle time after which a PostRunner thread is terminated, in seconds.
--agentIdleServerRequestWorkerTimeout=seconds	Idle time after which a ServerRequestWorker thread is terminated, in seconds
--agentIdleWorkerTimeout=seconds	Idle time after which a Worker thread is terminated, in seconds.
--agentInitMemory=percent	Initial java heap size as a percentage of the total system memory.
--agentInitMemoryMB=size	Initial java heap size in MB.
--agentKeyFile=path	Location of the key file used by the agent to support SSL connections from the server.

Option	Description
--agentKeystore=path	Location of the keystore file used by the agent to support SSL connections from the server.
--agentKeystorePassword=password	Password used to access the agent's keystore.
--agentLoadProfile=<true|yes|1|false|no|0>	Enable (1) or disable (0) loading the impersonated user's profile for impersonation steps if on Windows.
--agentLocalPort=port	Port used by the Commander agent for http communication on the localhost network interface.
--agentLogFile=path	Path where the C++ agent log file should be written.
--agentLogLevel=<TRACE|DEBUG|INFO|WARN|ERROR>	Logging level used by the C++ portion of the agent.
--agentLogMaxFiles=max	Maximum number of log files the C++ agent will accrue.
--agentLogMaxSize=max	Maximum size of each log file from the C++ agent. The value may be suffixed with a unit (MB, KB, B). Without a unit, the value is interpreted as bytes.
--agentMaxConnections=max	Maximum number of network connections for the agent.
--agentMaxConnectionsPerRoute=max	Maximum number of network connections per route for the agent.
--agentMaxHttpThreads=max	Maximum number of threads for handling inbound requests.
--agentMaxLoggedMessageLength=max	Maximum message length, used when logging requests/ responses to/from the ElectricFlow server.
--agentMaxMemory=percent	Maximum java heap size as a percentage of the total system memory.

Option	Description
--agentMaxMemoryMB=size	Maximum java heap size in MB.
--agentNoProxyHosts=hosts	Comma delimited list of hosts that should be reached directly, bypassing the proxy server.
--agentOutboundConnectTimeout=milliseconds	Timeout for the agent establishing outbound connections, in milliseconds.
--agentOutboundRequestInitialRetryInterval=seconds	Initial delay between retries for sending outbound requests to a server, in seconds.
--agentOutboundRequestMaxRetryInterval=seconds	Maximum delay between retries for sending outbound requests to a server, in seconds.
--agentOutboundRequestTimeout=hours	Timeout after which the agent gives up trying to send a request to a server, in hours.
--agentPluginsDirectory=path	The path used by the agent to get to the plugins directory of the ElectricFlow server where its resource definition lies.
--agentPort=port	Port used by the Commander agent for https communication on any network interface.
--agentProto=<http|https>	Protocol used internally by the agent.
--agentProtocol=<http|https>	Protocol used by the agent.
--agentProxyHost=host	The IP address of the proxy server.
--agentProxyPort=port	The port of the proxy server.
--agentServerConnectTimeout=seconds	Socket connection timeout for outbound requests to a server, in seconds.
--agentServerReadTimeout=seconds	Socket read timeout for responses from a server, in seconds.

Option	Description
`--agentServerSessionsFile=relativepath`	Relative path to the persisted server sessions file.
`--agentTLSEnabledProtocol <protocols>`	Comma-delimited list of SSL/TLS protocols that will be allowed for agent connections using HTTPS. The possible values are any combination of `TLSv1`, `TLSv1.1`, `TLSv1.2`, and `SSLv2Hello`. The default security configurations are as follows: • First-time ElectricFlow installations: TLSv1, TLSv1.1, and TLSv1.2 are enabled • Existing ElectricFlow installations: TLSv1, TLSv1.1, TLSv1.2, and SSLv2Hello are enabled We recommend removing the `SSL 2.0 Client Hello` or `SSLv2Hello` protocol from your security configurations for all components. When you do this, you would also need to upgrade older agents to the latest version to avoid security risks. You would need to upgrade agents if you are using the following agent versions: • Windows: 6.0.3 or older • Linux: 6.2 or older • Sun Solaris, HP UX, Mac OS: 8.4 or older

Option	Description
--agentUnixShellPattern=pattern	Windows-only: ordinarily, the agent creates script-files with CRLF line termination. But some shells on Windows require script files to be LF line-terminated, like Unix. This option sets a regular expression pattern for such shells. Defaults to a pattern that matches sh and bash, which in modern versions of Cygwin require LF-terminated script files.
--agentVerifyPeer=<true|yes|1|false|no|0>	Enable (1) or disable (0) verifying the certificate presented by the ElectricFlow server when it connects.
--agentWrapperConsoleFormat=format	Format of output for the agent wrapper console.
--agentWrapperEcwrapperWriteMaxAttempts=max	Workspace write maximum attempts count
--agentWrapperEcwrapperWriteRetryInterval=milliseconds	Workspace write interval between two attempts, in milliseconds.
--agentWrapperJavaAdditional=nnnnn=value	Set a custom line of the form wrapper.java.additional.nnnnn=value in the agent's wrapper.conf file. nnnnn must be an integer >= 10000.
--agentWrapperJavaClasspath=n=path	The Java classpath. n must be an integer >= 1.
--agentWrapperJavaLibraryPath=n=path	The Java Library Path (location of Wrapper.DLL or libwrapper.so). n must be an integer >= 1.
--agentWrapperLogfile=path	Log file to use for agent wrapper output logging.
--agentWrapperLogfileFormat=format	Format of output for the agent wrapper log file.
--agentWrapperLogfileLoglevel=loglevel	Log Level for agent wrapper log file output.

Option	Description
--agentWrapperLogfileMaxsize=size	Maximum size that the log file will be allowed to grow to before the log is rolled. Size is specified in bytes, kilobytes with a 'k' suffix, or megabytes with an 'm' suffix.
--agentWrapperLogfileMaxfiles=max	Maximum number of rolled log files which will be allowed before old files are deleted.
--agentWrapperNtserviceDependency=n=service	NT service dependencies. Add dependencies as needed starting from 1. n must be an integer >= 1.
--agentWrapperNtserviceStarttype=<AUTO_START|DEMAND_START>	Mode in which the NT service is installed.
--agentWrapperNtserviceInteractive=<true|false>	Whether to allow the NT service to interact with the desktop.
--agentWrapperPingInterval=seconds	Java virtual machine-wrapper heartbeat interval, in seconds.
--agentWrapperPingTimeout=seconds	Java virtual machine-wrapper heartbeat timeout, in seconds.
--agentWrapperRequestThreadDumpOnFailedJvmExit=<true|false>	Whether to try to get a thread dump if the Java virtual machine doesn't exit cleanly.
--agentWrapperShutdownTimeout=seconds	The wrapper's shutdown timeout, in seconds.
--agentWrapperStartupTimeout=seconds	The wrapper's startup timeout, in seconds.
--agentWrapperSuccessfulInvocationTime=seconds	The amount of time the agent has to be up before the wrapper considers it a 'successful' invocation. If there are 5 or more consecutive failed invocations, the wrapper will no longer start up the agent.
--agentWrapperSyslogLoglevel=loglevel	Log Level for sys/event agent wrapper log output.

Web Server Configuration Options

Option	Description
`--skipServiceRestart`	Disable the attempt to restart the web server. The ElectricFlow web server can only be started with `sudo`access. Therefore, you must use `sudo` to restart it manually afterward or use the `--skipServiceRestart` option to disable the restart attempt.
`--webDefaultUI=<flow|commander>`	The type of the Commander server user interface that will be used by default. This setting determines whether the Deploy UI or the Automation Platform UI appears when users browse to https://<*ElectricFlow_server*> without appending /flow or /commander respectively to the end of the URL.
`--webHostName=host`	The host name of the current machine in the form that users will typically use in their browser to access the web server.
`--webHttpPort=port`	The HTTP port of the web server.
`--webHttpsPort=port`	The HTTPS port of the web server.
`--webTargetHostName=host`	The host name of the ElectricFlow server to which the web server points. This argument modifies the ElectricFlow web server configuration and therefore also attempts to restart the ElectricFlow web server. If you used the `ecconfigure` command without `sudo` as recommended, the `commanderApache` service will not start and produces an error. Therefore, you must restart it manually afterward using `sudo`. You can also use the `--skipServiceRestart` argument to avoid the `ecconfigure` command's restart attempt and the error message.
`--webTargetHttpPort=port`	The HTTP port of the ElectricFlow server to which the web server points.
`--webTargetHttpsPort=port`	The HTTPS port of the ElectricFlow server to which the web server points.
`--webTimeZone=timezone`	The Olson TimeZone format (example: America/Los Angeles) for the php web server.
`--webPluginsDirectory=path`	The path used by the web server to get to the plugins directory of the ElectricFlow server to which it points. **Note:** You must have root privileges to use this option.

Option	Description
--webProxyUrl=url	The IP address and port of the proxy server in the following format. http://<IP_ADDRESS_PROXY>:<PROXY_PORT>
--webNoProxyHosts=hosts	Comma delimited list of hosts that should be reached directly, bypassing the proxy server.
--webEnableProxySettings=<1|0>	Enable (1) or disable (0) the proxy server configuration. If enabling for the first time, --webProxyUrl must be specified.
--webDLC=url	The URL to use for downloadable content requests.
--webDisableHttpsRedirection=<1|0>	Disable (1) or enable (0) HTTP -> HTTPS web redirection by the web server.
--webCsrfProtection=<true|false>	Enable (true) or disable (false) Cross-Site Request Forgery protection on the web server.

ElectricFlow Server Configuration Options

Option	Description
--serverAcceptQueueSize=max	The maximum number of pending connections the ElectricFlow server will queue up.
--serverBatchDbRequestsOverride=<1|0|auto>	Enable (1), disable (0) or let the server decide (auto) for using db request batching.
--serverBindIp=<ip or hostname>	The IP address or host name the ElectricFlow server listen to.
--serverCertFile=relativepath	Relative path of the certificate file used by the ElectricFlow server to support SSL connections.
--serverChangeTrackingHardMaxRecords=number	The maximum number of records for change tracking
--serverCommanderPort=port	The HTTP port of the server.
--serverCommanderHttpsPort=port	The HTTPS port of the server.

Option	Description
--serverCriticalServicesMonitoringEnabled=<true|false>	Enable (true) or disable (false) monitoring for the critical services.
--serverCriticalServicesMonitoringFrequency=seconds	The interval between the critical services checks and the disk space checks, seconds.
--serverCrlFile=relativepath	Relative path of the file containing the ElectricFlow server's certificate revocation list for SSL.
--serverDatabaseName=name	The name of the database the ElectricFlow server uses for its operation.
--serverDatabaseUsername=name	The user name for the database the ElectricFlow server uses for its operation.
--serverFileTransferPort=port	The file transfer port of the server.
--serverForceEnableAdmin=<1|0>	Set to '1' or '0' to override the current value of the 'enableAdminUser' ElectricFlow server setting.
--serverName=host	The name the ElectricFlow server, usually its fully-qualified domain name, or for a cluster the fully-qualified domain name of the load balancer.
--serverHttpPort=port	The HTTP port of the server, default value 8000.
--serverHttpsPort=port	The HTTPS port of the server, default value 8443.
--serverIgnoreServerMismatch=<1|0>	Enable (1) or disable (0) ignoring the ElectricFlow server host name mismatch.

Option	Description
--serverInitMemory=percent	Initial java heap size as a percentage of the total system memory.
--serverMaxMemory=percent	Maximum java heap size as a percentage of the total system memory.
--serverInitMemoryMB=size	Initial java heap size in MB.
--serverMaxMemoryMB=size	Maximum java heap size in MB.
--serverMaxThreadsApi=max	The size for the API thread pool. '0' means let the ElectricFlow server decide.
--serverKeyFile=relativepath	Relative path of the CA key file used by the ElectricFlow server to support SSL connections.
--serverKeystore=path	Location of the keystore file used by the ElectricFlow server to support SSL connections.
--serverKeystorePassword=password	Password used to access the ElectricFlow server's keystore.
--serverLogClusterConnectionProblems=<true|false>	Enable (true) or disable (false) additional logging for the connection problems in the cluster environment.
--serverMaxThreadsDispatch=max	The size for the dispatch thread pool. '0' means let the ElectricFlow server decide.
--serverMaxThreadsHttp=max	The size for the HTTP thread pool. '0' means let the ElectricFlow server decide.
--serverMaxThreadsQuartz=max	The size for the quartz thread pool. '0' means let the ElectricFlow server decide.

Option	Description
--serverMaxThreadsWorkflow=max	The size for the workflow thread pool. '0' means let the ElectricFlow server decide.
--serverMonitoringEnabledDataDirectory=<true|false>	Enable (true) or disable (false) disk space monitoring for the data directory.
--serverMonitoringEnabledLogDirectory=<true|false>	Enable (true) or disable (false) disk space monitoring for the log directory.
--serverMonitoringEnabledMqDirectory=<true|false>	Enable (true) or disable (false) disk space monitoring for the message broker data directory.
--serverMqDataDirectory=path	The directory the ElectricFlow server uses to store message broker data.
--serverMqDiskSpaceLimitHard=size	The limit of the free disk space when ElectricFlow server will be switched in bootstrap mode, in MB.
--serverMqDiskSpaceLimitSoft=size	The limit of the free disk space when ElectricFlow server will start to log warnings, in MB.
--serverMqDiskSpaceMonitoringEnabled=<true|false>	Enable (true) or disable (false) disk space monitoring for the message broker data storage.
--serverMqDiskSpaceMonitoringInClusterOnly=<true|false>	Enable (true) or disable (false) disk space monitoring for the message broker data storage only in cluster environment.
--serverNestedLdapGroupsMaxDepthLimit=max	Maximum allowed depth limit of nested LDAP groups
--serverPasskeyFile=path	Path to the server's passkey file.
--serverProxyHost=host	The IP address of the proxy server.

Option	Description
--serverProxyPort=port	The port of the proxy server.
--serverNoProxyHosts=hosts	Comma delimited list of hosts that should be reached directly, bypassing the proxy server.
--serverEnableProxySettings=<1|0>	Enable (1) or disable (0) the proxy server configuration. If enabling for the first time, --serverProxyHost and --serverProxyPort must be specified.
--serverPreserveSessions=<1|0>	Enable (1) or disable (0) preserving sessions even if there is the ElectricFlow server hostname mismatch.
--serverRestPort=port	The port for the REST documentation
--serverRestProtocol=<http|https>	The transfer protocol for the REST documentation.
--serverStatsdHost=host	The host of the statsd server the ElectricFlow server uses to send data.
--serverStatsdPort=port	The port of the statsd server the ElectricFlow server uses to send data.
--serverStatsdPrefix=string	The prefix the ElectricFlow server uses in the data to the statsd server.
--serverStatsdIncludeHostname=<true|false>	Enable (true) or disable (false) inclusion of the ElectricFlow server host to the prefix of the data sent to the statsd server.

Option	Description
`--serverTLSEnabledProtocol <protocols>`	Comma-delimited list of cryptographic protocols that will be allowed for ElectricFlow server connections using HTTPS. The possible values are any combination of `TLSv1`, `TLSv1.1`, `TLSv1.2`, and `SSLv2Hello`. The default security configurations are as follows: • First-time ElectricFlow installations: TLSv1, TLSv1.1, and TLSv1.2 are enabled • Existing ElectricFlow installations: TLSv1, TLSv1.1, TLSv1.2, and SSLv2Hello are enabled We recommend removing the `SSL 2.0 Client Hello` or `SSLv2Hello` protocol from your security configurations for all components. When you do this, you would also need to upgrade older agents to the latest version to avoid security risks. You would need to upgrade agents if you are using the following agent versions: • Windows: 6.0.3 or older • Linux: 6.2 or older • Sun Solaris, HP UX, Mac OS: 8.4 or older
`--serverZooKeeperConnection=host:port [,host:port,host:port[,host:port,host:port]]`	Comma-separated list of host IP/FQDN and ports for the ZooKeeper servers for a clustered configuration.

Option	Description
--serverExhibitorConnection=host[:port][,host,host[,host,host]]	Comma-separated list of host IP/FQDN for the Exhibitor servers for a clustered configuration, if you are using Exhibitor. To use a port number other than 80, add the port number after the first host IP/FQDN: the same port number will be used for all hosts.
--serverEnableClusteredMode=<1|0>	Enable (1) or disable (0) the clustered configuration. If enabling for the first time, --serverZooKeeperConnection must be specified.
--serverWrapperConsoleFormat=format	Format of output for the ElectricFlow server wrapper console.
--serverWrapperJavaClasspath=n=path	The Java classpath. n must be an integer >= 1.
--serverWrapperJavaLibraryPath=n=path	The Java Library Path (location of Wrapper.DLL or libwrapper.so). n must be an integer >= 1.
--serverWrapperLogfile=path	Log file to use for the ElectricFlow server wrapper output logging.
--serverWrapperLogfileFormat=format	Format of output for the ElectricFlow server wrapper log file.
--serverWrapperLogfileLoglevel=loglevel	Log level for the ElectricFlow server wrapper log file output.
--serverWrapperLogfileMaxfiles=max	Maximum number of rolled log files which will be allowed before old files are deleted.

Option	Description
`--serverWrapperLogfileMaxsize=size`	Maximum size that the log file will be allowed to grow to before the log is rolled. Size is specified in bytes, kilobytes with a 'k' suffix, or megabytes with an 'm' suffix.
`--serverWrapperPingInterval=seconds`	Java virtual machine-wrapper heartbeat interval, in seconds.
`--serverWrapperPingTimeout=seconds`	Java virtual machine-wrapper heartbeat timeout, in seconds.
`--serverWrapperRequestThreadDumpOnFailedJvmExit=<true|false>`	Whether to try to get a thread dump if the Java virtual machine doesn't exit cleanly.
`--serverWrapperStartupTimeout=seconds`	The wrapper's startup timeout, in seconds.
`--serverWrapperShutdownTimeout=seconds`	The wrapper's shutdown timeout, in seconds.
`--serverWrapperSuccessfulInvocationTime=seconds`	The amount of time the ElectricFlow server has to be up before the wrapper considers it a 'successful' invocation. If there are 5 or more consecutive failed invocations, the wrapper will no longer start up the ElectricFlow server.
`--serverWrapperSyslogLoglevel=loglevel`	Log level for sys/event ElectricFlow server wrapper log output.
`--serverXmlReaderStripWhitespaceText=<true|false>`	Enable (true) or disable (false) the clipping of the values of the UI-form parameters that contain only spaces.
`--wrapperJavaAdditional=nnnnn=value`	Set a line wrapper.java.additional=value in the server's wrapper.conf file. nnnnn must be an integer >= 10000.

Built-In (Default) Database Configuration Options

Option	Description
`--databaseEnableService=<1|0>`	Enable (1) or disable (0) the built-in database service.
`--databaseMemoryBufferSize=size`	Size of the database memory buffer. The value can be suffixed with a unit (K, M, or G). Without a unit, the value is interpreted as bytes. The default size is 256 MB.
`--databasePassword=password`	Password used to access the database. The default password is `changeme`.
`--databasePort=port`	Port used by the database. The default port is 8900.

Repository Server Configuration Options

Option	Description
`--repositoryAcceptQueueSize=max`	The maximum number of pending connections the repository server will queue up.
`--repositoryAgentUrl=url`	The agent URL to use for proxying server requests
`--repositoryPort`	The repository server port.
`--repositoryIdleConnectionTimeout=milliseconds`	The idle connection timeout, in milliseconds.
`--repositoryKeystore=path`	Location of the keystore file used by the repository server to support SSL connections from the ElectricFlow server.
`--repositoryKeystorePassword=password`	Password used to access the repository server's keystore.
`--repositoryMaxConnections=max`	The maximum number of total connections.
`--repositoryMaxConnectionsPerRoute=max`	The maximum number of connections to one machine.

Option	Description
`--repositoryInitMemory=percent`	Initial java heap size as a percentage of the total system memory.
`--repositoryMaxMemory=percent`	Maximum java heap size as a percentage of the total system memory.
`--repositoryInitMemoryMB=size`	Initial java heap size in MB.
`--repositoryMaxMemoryMB=size`	Maximum java heap size in MB.
`--repositoryMaxHttpThreads=max`	The maximum number of threads for handling inbound requests.
`--repositoryStorageDirectory=path`	Path to the repository backing store. The artifact repository will use this directory to store artifacts. **Note:** `path` cannot be a mapped drive path. For example, `c:/repository-data` is not allowed and will cause artifact publishing to fail. You must use a UNC path such as `//10.0.109.72/repo_share/repository-data`.
`--repositoryTargetHostName=host`	The host name of the ElectricFlow server to which the repository server points.

Option	Description
`--repositoryTargetHttpPort=port`	The HTTP port of the ElectricFlow server to which the repository server points.
`--repositoryTargetHttpsPort=port`	The HTTPS port of the ElectricFlow server to which the repository server points.
`--repositoryTargetProtocol=<http|https>`	The protocol that the repository server uses to talk to the ElectricFlow server.

Option	Description
`--repositoryTLSEnabledProtocol <protocols>`	Comma-delimited list of cryptographic protocols that will be allowed for ElectricFlow repository server connections using HTTPS. The possible values are any combination of `TLSv1`, `TLSv1.1`, `TLSv1.2`, and `SSLv2Hello`. The default security configurations are as follows: • First-time ElectricFlow installations: TLSv1, TLSv1.1, and TLSv1.2 are enabled • Existing ElectricFlow installations: TLSv1, TLSv1.1, TLSv1.2, and SSLv2Hello are enabled We recommend removing the `SSL 2.0 Client Hello` or `SSLv2Hello` protocol from your security configurations for all components. When you do this, you would also need to upgrade older agents to the latest version to avoid security risks. You would need to upgrade agents if you are using the following agent versions:

Option	Description
	• Windows: 6.0.3 or older • Linux: 6.2 or older • Sun Solaris, HP UX, Mac OS: 8.4 or older
`--repositoryProtocol`	The protocol that the repository server uses to talk to client applications.
`--repositoryProxyHost=host`	The IP address of the proxy server.
`--repositoryProxyPort=port`	The port of the proxy server.
`--repositoryNoProxyHosts=hosts`	Comma delimited list of hosts that should be reached directly, bypassing the proxy server.
`--repositoryEnableProxySettings=<1|0>`	Enable (1) or disable (0) the proxy server configuration. If enabling for the first time, --repositoryProxyHost and --repositoryProxyPort must be specified.
`--repositoryValidateFromDisk=<1|0>`	Enable (1) or disable (0) disk validation.
`--repositoryWrapperConsoleFormat=format`	Format of output for the repository server wrapper console.
`--repositoryWrapperJavaClasspath=n=path`	The Java classpath. n must be an integer >= 1.
`--repositoryWrapperJavaLibraryPath=n=path`	The Java Library Path (location of Wrapper.DLL or libwrapper.so). n must be an integer >= 1.

Option	Description
`--repositoryWrapperLogfile=path`	Log file to use for the repository server wrapper output logging.
`--repositoryWrapperLogfileFormat=format`	Format of output for the repository server wrapper log file.
`--repositoryWrapperLogfileLoglevel=loglevel`	Log level for the repository server wrapper log file output.
`--repositoryWrapperLogfileMaxfiles=max`	Maximum number of rolled log files which will be allowed before old files are deleted.
`--repositoryWrapperLogfileMaxsize=size`	Maximum size that the log file will be allowed to grow to before the log is rolled. Size is specified in bytes, kilobytes with a 'k' suffix, or megabytes with an 'm' suffix.
`--repositoryWrapperPingInterval=seconds`	Java virtual machine-wrapper heartbeat interval, in seconds.
`--repositoryWrapperPingTimeout=seconds`	Java virtual machine-wrapper heartbeat timeout, in seconds.
`--repositoryWrapperRequestThreadDumpOnFailedJvmExit=<true|false>`	Whether to try to get a thread dump if the Java virtual machine doesn't exit cleanly.
`--repositoryWrapperStartupTimeout=seconds`	The wrapper's startup timeout, in seconds.
`--repositoryWrapperShutdownTimeout=seconds`	The wrapper's shutdown timeout, in seconds.

Option	Description
--repositoryWrapperSuccessfulInvocationTime=seconds	The amount of time the repository server has to be up before the wrapper considers it a 'successful' invocation. If there are 5 or more consecutive failed invocations, the wrapper will no longer start up the repository server.
--repositoryWrapperSyslogLoglevel=loglevel	Log level for sys/event repository server wrapper log output.

General Options

Option	Description
-v,--version	Display version information.
-h,--help	Display this information.

Examples

Setting initial and maximum memory settings

For example, to set the ElectricFlow Server initial memory percentage to 21% and the maximum memory percentage to 31%, specify:

```
ecconfigure --serverInitMemory 21 --serverMaxMemory 31
```

Adjusting proxy settings for servers if your ElectricFlow server, web server, or repository server is deployed behind a proxy server that restricts Internet access

To use the following Perl scripts, remove the brackets ("< >"), and replace the bracketed example text with the values you need.

- To set ElectricFlow Server proxy settings:

```
ecconfigure --serverProxyHost <IP_ADDRESS_PROXY> --serverProxyPort <PORT>
 --serverNoProxyHosts "<HOST1,HOST2>"
```

- To set Repository Server proxy settings:

```
ecconfigure --repositoryProxyHost <IP_ADDRESS_PROXY> --repositoryProxyPort
<PORT>
 --repositoryNoProxyHosts "<HOST1,HOST2>"
```

- To set Web Server proxy settings:

```
ecconfigure --webProxyUrl <http://IP_ADDRESS:PORT> --webNoProxyHosts
<HOST1,HOST2,HOST3>
```

Changing the Apache web server port

Run `ecconfigure` with `--webHttpPort` and `--webHttpsPort`.

Your web server port setting will be changed appropriately in `httpd.conf`, `ssl.conf`, and `config.php`._____

Configuring the backing store location for the Artifact Repository

```
ecconfigure --repositoryStorageDirectory <new-path>
```

ecdaemon

`ecdaemon` is a "wrapper" program that can be used to start another program from an ElectricFlow job step—the "started" program will run as a daemon process. The ElectricFlow agent uses the facilities of the underlying operating system to make sure the process runs in a separate process group on a UNIX-based system, or outside of the normal "Windows Job" grouping in a Windows system. In either case, the ElectricFlow agent does not treat the process as one it should wait for or one it should try to "kill" if ElectricFlow needs to abort the step.

Use Cases

- `ecdaemon` is useful in the case where you are trying to deploy a "server-style" program in an ElectricFlow step. You do not want ElectricFlow to wait for that step to complete because it may run continuously, but you do want ElectricFlow to start the server program and then continue on to the next step.
- `ecdaemon` is useful if you want to "pre-load" some type of background process

`ecdaemon` launches the command and exits. Optionally, it sets a property in ElectricFlow with the `pid` of the program it spawned to make it possible for a later step to "kill" the daemonized program if desired.

For example:
`ecdaemon c:/install.exe a b c`

Using ecdaemon

```
$ ecdaemon
Usage: ecdaemon [options] cmd cmdArg1 cmdArg2 ...
       ecdaemon --version

Required:
    command                    The command to daemonize, followed by its arguments, if
any.
Options:
    --pidProperty=<prop-path    A property to set with the PID of the daemonized process.
    --version                  Print the version of this program and exit.
```

Command-line parsing

`ecdaemon` supports the standard UNIX-style `'--'` flag to indicate there are no more `ecdaemon`options and all subsequent options should be treated as simple arguments to the command. This is particularly important for commands that themselves take `'--'` arguments.

For example:

```
ecdaemon /usr/bin/myserver --config /etc/myserver.conf
```

will not run properly because ecdaemonwill attempt to parse the --config option instead of passing it to the myserver program. The correct way to invoke ecdaemon in this case is:

```
ecdaemon -- /usr/bin/myserver --config /etc/myserver.conf
```

If you want to store the daemonized process's pid in a property, do so as follows:

```
ecdaemon --pidProperty /myJob/serverPid -- /usr/bin/myserver --config
/etc/myserver.conf
```

As a daemon process, any output goes to /dev/null, therefore no output file is generated.

ec-perl considerations

Use the perl system () call to start ecdaemon. The system () returns an exit status, "backticks" capture and return output that waits for the daemonized command to complete on Windows, and exec never returns at all if it is successful.

ecproxy

A driver script with built-in support for SSH. Every major operation can be overridden by defining a Perl function in the Proxy Customization field on the New Proxy Resource panel, available from the Resources page (by specifying which operation this function re-implements. These operations must have certain "signatures" for the driver to invoke them properly—the operations are listed and described below. For more detail, see the SSH implementation in ecproxy.pl.

ecproxy Algorithm

ecproxy invokes the operations detailed below to perform the following actions:

1. Uploads the command-file to a "workingDirectory" on the proxy target, using the protocol specified in the proxy config. Currently, only SSH is supported.
2. Creates a wrapper sh shell script that CD's to workingDirectory, sets "COMMANDER_ environment variables" that exist in the proxy agent's environment, and runs the command-file previously uploaded.
3. Uploads the wrapper script to workingDirectory on the proxy target.
4. Runs the wrapper script on the proxy target and streams its output to the proxy agent's stdout.
5. Deletes the local wrapper shell script, the remote wrapper, and remote command-file.
6. Exits using the exit code of running the wrapper script.

ecproxy Operations

getDefaultWorkingDirectory

Description	Computes the default working directory where a command needs to run on the proxy target, if the step is not defined with a working directory.
Arguments	None
Returns	The path to a directory as it would be accessed on the proxy target.

SSH implementation function	Not SSH-specific, so the function is '`getDefaultWorkingDirectory`'
Existing implementation	Return `$ENV{COMMANDER_WORKSPACE_UNIX};`
Reason to override	If the "Working Directory" field is empty on a step that is going to run on a proxy target, the working directory for the step should be the workspace, just as it would be if the step were running on a non-proxy ElectricFlow agent. `ecproxy` is guaranteed to run in the workspace directory on the proxy agent, but it is not guaranteed that the proxy target has the same path to the workspace. For example, the workspace on a Windows proxy agent is not the same as the path a Unix proxy target uses to access the workspace—so the existing implementation of this operation simply returns the UNIX path to the workspace. However, if the proxy target has a different path for accessing the workspace, the existing implementation will give the wrong answer. Thus, a user can provide a different implementation that gives the right answer.

This operation is applicable only if "Working Directory" is empty, and is used as the working directory on the proxy target in that case for running the command.

getDefaultTargetPort

Description	Computes the default port where the proxy target is listening for this protocol.
Arguments	None
Returns	The default port.
SSH implementation function	`ssh_getDefaultTargetPort`

This operation is applicable only if the resource definition specifies no port value.

connect

Description	Opens a connection to the proxy target using the desired protocol.
Arguments	`host, port` (optional)
Returns	"connection-context hash-ref" on successful connection. This context can contain anything other functions can use to perform their tasks (for example, a connection handle).
Example 1	`my $context =< connect('myhost', 22)`
Example 2	`my $context =< connect('myhost')`
SSH implementation function	`ssh_connect`

Note: Because the port is optional, the implementation of `connect` can default to whatever is reasonable for the protocol. This means the 'Proxy Target Port' need not be specified in the ElectricFlow web UI for proxied agents reachable on the default port.

uploadFile

Description	Uploads the given `srcFile` to the proxy target as `tgtFile`.
Arguments	context, `srcFile` (typically simple file-name), `tgtFile` (typically workingDirectory/file-name)
Returns	Nothing; on failure it does a 'die' with an appropriate error message.
Example	`uploadFile($context, 'agent123.tmp', '/opt/work/joe/agent123.tmp')`
SSH implementation function	`ssh_uploadFile`

generateWrapperScript

Description	Generates the script body that will run the command-file on the proxy-target in the `workingDirectory`.
Arguments	`workingDirectory, cmd, cmdArg1, ..., cmdFileName` (just the base-name, no directory)
Returns	A string containing the script to execute on the proxy target.
Example	`generateWrapperScript('/opt/work/joe', 'perl', 'agent123.tmp'`
SSH implementation function	Not SSH-specific, so the function is `'generateWrapperScript'`
Existing implementation	cd workingDirectory; set COMMANDER_ environment variables; run command-file, properly quoting the command and args.
Reason to override	If the proxy target does not have `sh`, the wrapper script needs to be written in a language available on the target.

uploadWrapperScript

Description	Uploads the wrapper-script code to the proxy target.
Arguments	context `workingDirectory`. `wrapperScriptBody`
Returns	The path to the wrapper-script on the proxy target.

Example	`my $wrapperFile =< uploadWrapperScript($context, '/opt/work/joe', 'cd /opt/work/joe; perl agent123.tmp')`
SSH implementation function	3.1, 3.1.1: `ssh_uploadWrapperScript`
3.1.2 and later	Not SSH-specific anymore, so the function is `'uploadWrapperScript'`

Note:

- This function must generate a uniquely named file that will not conflict with other `ecproxy` invocations that might be occurring in parallel steps. The recommended approach is to generate a file name containing the job-step-id.
- Depending on the protocol and facilities available in the Perl implementation, you may or may not need to create a local `tmp` file to upload to the proxy target. If you do, record that fact in the context and clean up the local file in the `cleanup` operation. Setting the local wrapper file path in `$context->{wrapperFile}` is recommended because the default `cleanup` operation implementation looks for that string.

generateWrapperInvocationCommand

Description	Generates the command-line for running the wrapper script on the proxy target.
Arguments	`remoteWrapperFile` (path on proxy target)
Returns	A string containing the command-line for running the wrapper script file on the proxy target.
Example	`my $wrapperCmdLine =< generateWrapperInvocationCommand ($wrapperFile)`
SSH implementation function	Not SSH-specific, so the function is `'generateWrapperInvocationCommand'`
Existing implementation	Return `"sh $remoteWrapperFile";`
Reason to override	The default implementation of this function returns something like `'sh $wrapperFile'`. If the wrapper script is not an `sh` script, or if you want to pass different arguments to the shell, you must override this function.

runCommand

Description	Runs the given command-line on the proxy target.

Arguments:	`context`, `cmdLine`
Returns	exit-code from running the command on the proxy target, `undef` if the command could not be run for some reason.
Example	`runCommand($context, $wrapperCmdLine)`
SSH implementation function	`ssh_runCommand`

cleanup

Description	Performs any cleanup task after the command has completed on the proxy target. Typically, it deletes any locally created temp files and uploaded files on the proxy target.
Arguments	context, `cmdFile`, `wrapperFile` (both are of the form `workingDirectory/file-name`)
Returns	`1` on success, `undef` on failure.
Example	`cleanupTarget($context, '/opt/work/joe/agent123.tmp', '/opt/work/joe/cmdwrapper.123.tmp')`
SSH implementation function	3.1, 3.1.1: `ssh_cleanup`
3.1.2 and later	Not SSH-specific anymore, so the function is `cleanup`

Note: The default implementation deletes the locally created wrapper script file whose path is stored in `$context->{wrapperFile}`, if it exists. Thus, if the uploadWrapperScript operation is overridden, it is recommended the overriding function set this attribute—that way, `cleanup` need not be overridden.

ping

Description	A test to see if the proxy target is usable.
Arguments	`host`, `port` (optional)
Returns	`1` on success, `undef` on failure.
Example	`ping('myhost', 22)`
SSH implementation function	Not SSH-specific, so the function is `ping`.

Existing implementation	Opens a socket connection to the proxy target on the desired port.
Reason to override	The existing implementation may be deemed too simple for doing a ping; overriding ping to open a connection and do some protocol-specific handshaking might be more appropriate for some protocols / use cases.

Available Helper Functions

To make proxy customization easier, `ecproxy` provides the following helper functions.

mesg

Description	Debug logging function. Writes to the file referenced in the ECPROXY_DEBUGFILE environment variable (if it exists). No-op otherwise.
Arguments	`message`
Example	`mesg("myCleanup: about to delete $cmdFile on proxy target");`

This function automatically adds a newline to whatever it emits, so the caller does not have to incorporate a newline in message.

readFile

`Description`	Reads a file.
`Arguments`	`fileName`
`Returns`	Contents of the file. If there is an error, it returns an empty string.
`Example`	`my $data =< readFile("foo.txt");`

writeFile

Description	Creates a local file containing data.
Arguments	`fileName`, `data`
Returns	`1` on success, `undef` on failure.
Example	`writeFile("myWrapper.$ENV{COMMANDER_JOBSTEPID}.cmd", "perl foo.pl")`

initDispatcher

Description	Initialize the operation dispatcher map to point to functions for the given protocol. For each operation, `initDispatcher` checks if a function named `protocol_operation` exists, and if so, assigns that function as the implementation for that operation.
Arguments	`protocol`
Example	`initDispatcher("ssh") sets the "connect" operation to run "ssh_connect", "uploadFile" =<> "ssh_uploadFile", etc.`

setOperation

Description	Sets the implementation of an operation to be a particular function.
Arguments	`operation`, `function`. The 'function' argument may be the name of a function or a reference to a function.
Example	`setOperation("ping", "my_ping"); sets the "ping" operation to run the "my_ping" function`
Example	`setOperation("ping", \&my_ping);` same as above, but using a function `ref`

This function manipulates the `gDispatcher` hash, but provides a safe interface to it.

loadFile

Description	Load proxy customizations from a file.
Arguments	fileName
Example	`loadFile("custom.pl")`

setSSHKeyFiles

Description	Set the paths to the public and private key files that ssh will use to authenticate with the proxy target.
Arguments	`publicKeyFile`, `privateKeyFile`
Example	`setSSHKeyFiles('c:\foo\pub.key', 'c:\foo\priv.key')`

Tip: This is very useful on Windows proxies, where there is no reasonable default for ssh to use.

setSSHUser

Description	Set the name of the user to authenticate with the proxy target.
Arguments	`userName`
Example	`setSSHUser('user1')`

> **Note:** By default, the user name the agent is "running as" is used to log into the proxy target. If key-based authentication is configured on the target system such that '`agentUser`' can log into the '`user1`' account on the proxy target, this function leverages that configuration.

useMultipleSSHSessions

Description	Normally, ecproxy uses one ssh session with a number of "channels" to perform tasks like uploading files, running the command, and running a cleanup command on the proxy target. Some SSH servers don't allow this. This method configures ecproxy to use a separate SSH session for each operation; this requires authenticating with the SSH daemon on the proxy target several times, and thus it may perform worse than the single-session-multi-channel mode.
Arguments	None
Example	`useMultipleSSHSessions()`

Examples

Specify public/private key files for SSH

1. Set the `proxyCustomization` property on the resource like this: `?setSSHKeyFiles ('c:\foo\pub.key', 'c:\foo\priv.key');`
2. Set the ECPROXY_SSH_PRIVKEYFILE and ECPROXY_SSH_PUBKEYFILE environment variables on the proxy agent as system environment variables.

Override one of the operations (for example, to enable SSH connection with username/password)

Set the proxyCustomization property on the resource like this: `?sub myConnect($$) {...} setOperation("connect", \&myConnect);`

Load proxy customizations from a file rather than having all the logic in the `proxyCustomization` property on the resource

Set the proxyCustomization property on the resource like this: `?loadFile('c:\foo\custom.pl');`

Implement a whole new protocol

Specify protocol as `'myproto'` and have a proxy customization block like this:

```
sub myproto_getDefaultTargetPort() {
...
}
sub myproto_connect($;$) {
...
}
sub myproto_uploadFile($$$) {
...
}
sub myproto_uploadWrapperScript($$$) {
# Note: As of 3.1.2, the default implementation is likely good enough, so
it may not be necessary to define this override
....
} sub myproto_runCommand($$) {
...
} sub myproto_cleanup($$$) {
# Note: As of 3.1.2, the default implementation is likely good enough, so
it may not be necessary to define this override
....
}
# Initialize the dispatcher to run these functionsinitDispatcher("myproto");
```

Override ping to do a connect operation (which does a full protocol handshake, authentication, and so on)

Write a specialized ping function for the proxy customization like this:

```
sub heavy_ping($$) {
   my ($host, $port) =< @_;
   return ssh_connect($host, $port);
}
setOperation("ping", \&heavy_ping);
```

"Real World" Examples

ClusterExec

A basic integration for using `clusterupload` and `clusterexec` to reach a proxy target is here. It has been tested on a Windows target with a Cygwin installation. It will not work "out-of-the-box" because it makes the following assumptions:

- The proxy target has `sh` and other UNIX tools (for example, `rm`).
- The locations of the `clusterexec` and `clusterupload` binaries are hard-coded at the top of the proxy customization.

To make this proxy customization work on a Windows machine that does not have Cygwin, the `generateWrapperScript` operation would need to be overridden with a function that generates a `cmd` batch script, and the `generateWrapperInvocationCommand` operation would have to be overridden to generate a `"cmd /c ..."` command rather than `"sh ..."`.

MySQL

The idea here is that the proxy target need not be a host for running arbitrary commands. It could be a special entity like a `db`. This integration uses the `mysql clt` to run the step command (which should be SQL) on the `db` referenced by the proxy target host and port.

A bare-bones integration with MySQL:

```
# Set the path to the mysql binary; if the directory is in the proxy agent's
# PATH, this variable can simply contain the name of the executable.

my $gMySQL =< "c:/cygwin/usr/local/tools/i686_win32/bin/mysql.exe";

sub mysql_getDefaultTargetPort() {
    return 3306;
}
sub mysql_connect($;$) {
    # This "protocol" implementation is just going to use the mysql
    # command-line tool, so just save off host/port.

    my $host =< $_[0];
    my $port =< $_[1] || mysql_getDefaultTargetPort();

    return {host =<> $host,
            port =<> $port};
}
sub mysql_uploadFile($$$) {
    my ($context, $cmdFile, $rmtCmdFile) =< @_;

    # We do not need to upload the command-file to the proxy target.
    # We are going to run the mysql clt on the proxy agent to run
    # the query (contained in the local command-file),
    # so just save off the name of the command-file.

    $context->{cmdFile} =< $cmdFile;
}
sub mysql_uploadWrapperScript($$$) {
    my ($context, $workingDir, $wrapperScript) =< @_;

    # This has no meaning for this integration.  No-op.
}
sub mysql_runCommand($$) {
    my ($context, $cmdLine) =< @_;

    # cmdLine is a command-line for running the wrapper script, which
    # has no meaning for this integration. We just want to run
    # 'mysql' for the desired host/port with the command-file.

    system("$gMySQL -D commander -h $context->{host} -P $context->{port} " .
        "-u commander -pcommander -e \"source $context->{cmdFile}\"");
}
sub mysql_cleanup($$$) {
    # We didn't create any temp files.  No-op.
}
# Initialize the dispatcher to run these functions
initDispatcher("mysql");
```

Android

This example uses the `adb` tool to upload files to the device and run commands on it. Initial testing has been only against the android emulator, but it is implemented in such a way that it should work against a real android device attached using USB to the proxy agent, or a device on the network.

A first attempt at proxying to android devices:

```
# Set the path to the adb binary; if the directory is in the proxy agent's
# PATH, this variable can simply contain the name of the executable.

my $gADB =< "c:/android-sdk-windows-1.6_r1/tools/adb.exe";

sub android_getDefaultTargetPort() {
    # Not sure what a good meaningful value is here.
    return 0;
}
sub android_connect($;$) {
    # This "protocol" implementation uses the adb
    # command-line tool.  Depending on the value of
    # host, construct the appropriate adb command-line
    # argument.

    my $host =< $_[0];
    my $context =< {};
 #   if ($host eq "emulator") {
    if ($host eq "localhost") {
        # We want to talk to the emulator running on this host.
        $context->{targetArg} =< "-e";
    } elsif ($host eq "usb") {
        # We want to talk to the single android device connected
        # to the computer via a USB.
        $context->{targetArg} =< "-d";
    } else {
        # This must be the serial number of some device somewhere.
        $context->{targetArg} =< "-s $host";
    }
    return $context;
}
sub android_uploadFile($$$) {
    my ($context, $srcFile, $tgtFile) =< @_;
    my($filename, $directories) =< fileparse($tgtFile);

    my $result =< `$gADB $context->{targetArg} push $srcFile
    "/data/tmp/$filename" 2>&1`;
    if ($? !=< 0) {
        die ("android_uploadFile: Error uploading file $srcFile to
        /data/tmp/$filename: $result\n");
    }
}
sub android_runCommand($$) {
    my ($context, $cmdLine) =< @_;

    # cmdLine is a command-line for running the wrapper script, which
    # has no meaning for this integration.  We just want to run
    # 'adb' for the desired device with the command-file.

    system("$gADB $context->{targetArg} shell $cmdLine");
}
sub android_cleanup($$$) {
    my ($context, $remoteCmdFile, $remoteWrapperFile) =< @_;

    # This was copied from ssh_cleanup except that we do "rm",
    # not "rf -f".
```

```
    mesg("cleaning up");

    # Delete the locally generate wrapper file.
    unlink($context->{"wrapperFile"});

    # Delete the cmd-file and wrapper script file on the proxy target.
    $gDispatcher{"runCommand"}($context,
        "rm $remoteWrapperFile $remoteCmdFile");
}
sub android_ping($;$) {
    my ($host, $port) =< @_;
    $port =< $gDispatcher{"getDefaultTargetPort"}() unless isPortValid($port);

    my $socket =< IO::Socket::INET->new(PeerAddr =<> $host,
                                        PeerPort =<> $port,
                                        Proto    =<> "tcp",
                                        Type     =<> SOCK_STREAM)
                           or die "Couldn't connect to $host:$port : $@\n";
}

# Initialize the dispatcher to run these functions
initDispatcher("android");
1;
```

ecremotefilecopy

When ElectricFlow agents (on platforms other than Linux or Windows) run steps that create log files in a workspace the ElectricFlow web server cannot access (through Linux or Windows agents), use *ecremotefilecopy* to recreate job logs so they are visible on those ElectricFlow agents, which then enables the web server to retrieve and render those log files.

Using postp and *ecremotefilecopy*, the log file is populated and recreated in a workspace accessible to the ElectricFlow web server, allowing the Job Details page to display the log file. Although this functionality is supported, it is not a recommended method of operation. This method should be used only as a last resort when a shared file system (between alternate agents and primary platform agents [Linux and Windows]) is not possible.

The reasons `ecremotefilecopy` is not recommended are:

- You will not see logs in real time. Logs are not visible until the "recreate step" has completed running.
- There is a performance penalty, especially when running with large files.

Setting up the process

- Create a "Setup" step in your procedure
- Update the Postprocessor field for each step whose results you want to see on the server.
- Add a step (one or more times) to the procedure to recreate the ElectricFlow server log files.

Creating a Setup Step

In your procedure, create a step called "Setup". This step needs to be in your procedure *before* any step running on a remote workspace.

Note: This is your top-level procedure, not a subprocedure.

- Navigate to your procedure.
- To create a new step, click the Command step link.
- Set the fields as follows:

  Step Name: `Setup`

  Command(s): `ecremotefilecopy setup`

  Resource: `local`

  Workspace: you have two choices:

  - use `default`
  - use: `alternateWorkspaceForDisplay`
    There is a property on the Workspace called alternateWorkspaceForDisplay, which is a secondary location to look for workspace files. This secondary location is used when the workspace files are not accessible to the web server. If the Apache server cannot locate the file in the original workspace, it looks in the alternate one.

Update the Postprocessor field for steps in your procedure

This step defines a postprocessor that will run at the end of the steps you specify. Add the following information to every step running on a remote agent if you want to see its results in the web interface.

- Navigate to a procedure and a step.
- In the Postprocessor field, enter:
  `postp --check none --loadProperty /myJob/jobSteps[Setup]/postpExtensions`
- If you are using postp in this step to scan your step log for errors, warnings, and so on also, omit `"--check none"` from the invocation line.

Add a Final Step to your procedure

This step adds a new step at the end of your existing procedure. This step finds all properties created by the postprocessor, then reads the properties and creates local log files based on the properties, then deletes the properties.

- Navigate to your procedure.
- To create a new step, click the Command step link.
- Set the fields as follows:

  Step Name: `Recreate the Log Files`

  Always run step: (Check the box)

  Command(s): `ecremotefilecopy recreateFiles`

  Resource: `local`

  Workspace: `default`

After the final step runs, you should see links (icons) displayed in the Log column on the Jobs Details page.

Click the icon to display the log file.

Copying Other Files from the Workspace

By default, ecremotefilecopy copies only postp log and diag files, and step logs. You can also copy other files from the workspace using a function named postpEndHook2.

You must do the following in your step:

1. Make sure that the file you want to copy is in the step workspace. (It can be copied there, created there, etc.)
2. For your procedure, create a property (named postpEndHook2, for example).
3. Define a function named postpEndHook2 inside the property. For example:

```
sub postpEndHook2() {

    # Missing param does not cause an error
    $::gCommander->abortOnError(0);

    # Add filename to a "special" property such that it will be picked up by
ecremotefilecopy
    my $fileName =< 'paul.txt';
    copyFileToProperty ($fileName);

    # Restore default error handling
    $::gCommander->abortOnError(1);
}
```

4. Add the following line in the Postprocessor field of this step:

```
postp --check none --loadProperty /myJob/jobSteps[Setup]/postpExtensions --
loadProperty /myProcedure/postpEndHook2
```

ZKConfigTool

Before starting the ElectricFlow server cluster, you must populate your Apache ZooKeeper server with ElectricFlow database configuration information that all ElectricFlow server nodes will use in the cluster. You use `ZKConfigTool` to import this information into your ZooKeeper server. For information about using `ZKConfigTool`, see the "Uploading Configuration Files to ZooKeeper" section in the "Clustering" chapter of the *ElectricFlow Installation Guide* at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html.

ClusterInfoTool

Use `ClusterInfoTool` to get information on the running ElectricFlow server cluster from ZooKeeper.

Prerequisites

- The ElectricFlow server cluster must be installed and running on the network.
- Configuration files that all ElectricFlow server nodes will use in a clustered configuration must be uploaded to Apache ZooKeeper server using the ZKConfigTool.

- The ZooKeeper cluster must be running an odd number of Zookeeper nodes and there must be a leader node.
- The system must be running a version of Java supported by ElectricFlow. Java is automatically installed on a system with the ElectricFlow software as part of the Tools installation.

Locations

The ElectricFlow installer adds the `ClusterInfoTool` to the following default locations:

- Windows: `C:\Program Files\Electric Cloud\ElectricCommander\server\bin\cluster-info-tool-jar-with-dependencies.jar`
- Linux: `/opt/electriccloud/electriccommander/server/bin/cluster-info-tool-jar-with-dependencies.jar`

Command

The following command shows how to run `ClusterInfoTool` from the *<data_dir>*`\conf` directory:

```
../jre/bin/java -DCOMMANDER_ZK_CONNECTION=<ZooKeeper_Server1_IP>:2181,<ZooKeeper_
Server2_IP>:2181,<ZooKeeper_Server3_IP>:2181 -jar ../server/bin/cluster-info-tool-jar-
with-dependencies.jar
```

where `DCOMMANDER_ZK_CONNECTION` must point to the ZooKeeper system(s) to which the ElectricFlow server cluster is connected.

Output

This is sample output generated by `ClusterInfoTool`:

```
Checking /commander/jgroups/hornetq:

        582d3642-9736-e87f-4e19-c22d7776ccab ->

                WIN-M3Q09A2PNFP-21066   6c80e9a9-2fed-6352-e437-fe76b65aa80d
                10.0.175.78:5446    F
                WIN-M3Q09A2PNFP-28728   0afa39df-d072-b05a-034a-aed74b7a39ee
                10.0.238.179:5446   F
                WIN-M3Q09A2PNFP-28295   c69f0430-df04-bf24-1294-b471a4a3f151
                10.0.2.207:5446     F
                WIN-M3Q09A2PNFP-15869   582d3642-9736-e87f-4e19-c22d7776ccab
                10.0.2.206:5446     T


Checking /commander/jgroups/commander:

        0c32103a-d4e7-da91-f8c4-1fd8e125c156 ->

                WIN-M3Q09A2PNFP-19713   0c32103a-d4e7-da91-f8c4-1fd8e125c156
                10.0.2.206:5447     T
                WIN-M3Q09A2PNFP-14432   75afe3eb-7a04-d160-c8d0-d64f8ac3c796
                10.0.175.78:5447    F
                WIN-M3Q09A2PNFP-33530   1cf1795f-e658-3cae-c515-546082bffec9
                10.0.238.179:5447   F
                WIN-M3Q09A2PNFP-60743   cf8b4201-3fac-71c4-18da-0a885b3e1e61
                10.0.2.207:5447     F
```

How to interpret `ClusterInfoTool` output:

- The nodes `/commander/jgroups/hornetq` and `/commander/jgroups/commander` contain information on these JGroups clusters:
  - `commander` for the ElectricFlow server cluster
  - `hornetq` for the HornetQ cluster
- The children nodes under each of the JGroups nodes represent the participating ElectricFlow servers in the cluster. Each child node entry is in this form:

  ```
  <Logical Name>   <UUID>   <IP address>:<port>   T|F
  ```

- The number of entries in both JGroups nodes should be the same, with matching IP addresses but with different port numbers and distinct logical names and UUIDs. The coordinator node in each JGroups cluster is identified with a 'T' against its entry.

Automation Platform Tasks

All the topics in this section are about a particular feature or function in the ElectricFlow automation platform. From the automation platform UI, you can access the topics as follows:

- All ElectricFlow Help topics above this directory are expanded, user-guide style Help topics about a particular ElectricFlow feature or function. Direct access to these Help topics is from the Help system Table of Contents. Although these topics are not directly linked within the ElectricFlow product, many "page-specific Help topics contain links to one or more of these topics to provide more detailed information quickly.
- The Help topics listed below this directory are "page-specific" Help topics, which means each of these topics contains specific information for the automation platform web page you are viewing. To access these Help topics while using ElectricFlow, click the **Help** link at the top-right corner on any automation platform web page. (The topics in this section of the AllHelp directory are listed singularly or in groups as is appropriate.)

Access Control

Use this GUI page to view and modify access privileges for a particular ElectricFlow object. Depending on the object where you want to set permissions, you will see that object's name as part of the page title (above the tables).

For example, if you clicked the Access Control link from a Project Details page, you will see the project name as part of the Access Control page title.

Reading and Using This Page

- This page displays one or more access control lists. The top list contains entries for the object itself (specified in the page title) and also identifies the object.

  For example, if the heading for the top list reads "Privileges for Procedure: buildAndTestAll," you are viewing access privileges for a procedure named "buildAndTestAll". Click the object name to view the main page for that object.

- Typically, you will see more than one list on the page. Each list below the first one contains privileges for an object that "contains" the objects above it.

  For example, a project contains all of its procedures and a procedure contains all of its steps. Privileges for the top-level object are determined by all the privileges in all of the displayed lists. The lists form an "inheritance chain" where it each object "inherits" permissions from the objects below it on the page.

- When a user attempts a particular operation on the object, ElectricFlow examines the lists on this page from top to bottom. If there is an entry specifying "deny" for the user (or a group containing the user) in the top list, access is denied. Otherwise, if an entry specifies "allow" for the user (or a group containing the user) in the top list, access is allowed.

  If access is neither allowed nor denied by the top list, ElectricFlow proceeds to the next list and processes it in the same way.

  > **Note:** If access is neither allowed nor denied by any list, ElectricFlow denies access.

- The inheritance mechanism makes it easy to control access for a large number of objects in a single place. For example, project access control entries automatically apply to new objects created within the project. Each new object in the project has an empty access control list, but will inherit from the project.

Using the Links

Use these links to add or increase Access Control for an object.

- **Add User**—Use this link to add permissions for a specific user.
- **Add Group**—Use this link to add permissions for a specific group, which means all users in that group would have the permissions allowed to the group.
- **Add Project**—Use this link to set or redefine permissions for a project.
- **Break Inheritance**—If you use the "Break Inheritance" action for any list, no additional inheritance occurs below that list and you no longer see other lists on this page. This action is useful if you want privileges for an object to be totally different than its containing object. If an object has no entries in its access control list and you break inheritance for that object, you make the object *completely inaccessible*—you will not even have the "Change Permissions" privilege, so you cannot restore inheritance. If this occurs, see your system administrator to restore inheritance.

  > **Important:** Be very careful if you break inheritance!

- Actions:
  - **Edit**—Use this link to modify current permissions, but be careful if you modify permissions in an *inherited* access control list. Modifying inherited access control affects all other objects that inherit from the same list.
  - **Delete**—Deletes the current privileges granted for that user, group, or project.

Privilege Definitions

The following four privilege types (for each ElectricFlow object) can be assigned *allow*, *deny*, or *inherit* permission.

- Read—Allows object contents to be viewed.
- Modify—Allows object contents (but not its permissions) to be changed.
- Execute—If an object is a procedure or it contains procedures (for example, a project), this privilege allows object procedures to be invoked as part of a job. For resource objects, this privilege determines who can use this resource in job steps.
- Change Permissions—Allows object permissions to be modified.

For more information, see the main Access Control Help topic. This Help topic also contains two examples that illustrate how you might use Access Control to increase ElectricFlow security.

Access Control—defining entries

Use this page to define access control entries selected from a filtered list of ElectricFlow users, groups, or projects, or by providing an explicit name. The following instructions are generic. For example, if you chose Add User, the Filter field will be labeled "User Filter".

Enter information into the fields as follows:

Field Name	Description
Radio selector	Choose between selecting "Search for existing..." or select "Provide explicit ... name". Providing an explicit name bypasses the search and allows you to enter an arbitrary name.
Filter	Enter a partial name to retrieve a list of all users/groups/projects that include the partial name you supplied.
Include Inactive	[User/Group only]—Use this checkbox only if requesting external LDAP or Active Directory providers. • If "checked," all available users or groups are returned. • If not checked, only those users or groups known to the ElectricFlow server are returned, for example, users who have logged in or groups containing users who have logged in.

Click **OK** to retrieve your information. If you entered filter criteria, the page will refresh with a list of matches. Select any row in this table to create an access control entry for that principal.

Column descriptions

- **Name**—Name of the user, group, or project
- **Location**—Only viewable for users or groups from non-local directory providers. The name combined with the location is the unique identifier for this user or group, local to the ElectricFlow server

Privileges—create new or edit existing privileges

Use this page to create or modify an access control list entry. Each entry names a *principal*, which can be either an individual user or a group, and defines four privileges for the principal.

For each of the four privileges, **Read**, **Modify**, **Execute**, or **Change** Permissions:

- Creating a new entry or modifying an existing entry allows you to select either Inherit, Allow, or Deny access entries.
- On the server access control table, "Don't Care" replaces the "Inherit" option.

For more information about access control, see the Access Control Help topic.

Artifacts

This page displays all artifacts available on this ElectricFlow server. You can search for artifacts and save the artifact search filters for later use. See Context Searching and Filtering on page 1244 for details on using the search capabilities on this page.

Links and actions at the top of the table

- **Create Artifact**—Use this link to go to the New Artifact page to create a new artifact.
- The "star" icon allows you to save this page to your Home page for quick access.

Column descriptions

Column Name	Description / Actions
Name	The name of the artifact—a system-generated name created by combining the Group Id and Artifact Key components. Selecting an artifact name takes you to the Artifact Details page for that artifact.
Group Id	The user-generated group name for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.
Artifact Key	The "key" component of the artifact name.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Actions	**Edit—**Use this link to go to the Edit Artifact page to modify the artifact on this row. **Delete—**Use this link to delete the artifact on this row.

Artifact Details

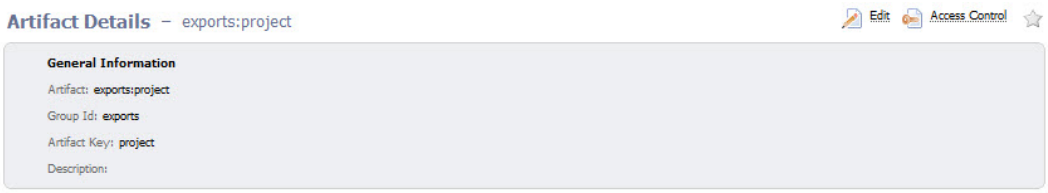
This page displays detailed information about an artifact.

Links and actions at the top of the table

- **Edit—**Use this link to go to the Edit Artifact page to make any necessary changes for this artifact's description or artifact version name template.
- **Access Control**—Use this link to set privileges for this artifact. *For more information*, see the Access Control Help topic.
- The "star" icon allows you to save this job information to your Home page.

General Information section

This section displays the artifact name (Artifact), Group Id, Artifact Key, and any Description previously supplied for this artifact.



The "tabbed" sections

The next section of tabs allows you to select the type of information you want to see.

Artifact Versions table

This table displays all artifact versions included in this artifact.

Column descriptions

Artifact Version—Click on an artifact version name to go to the Artifact Version Details page for that artifact version.

Description—The previously supplied description, if any, for this artifact version.

Actions

- Click the **Edit** link to go to the Edit Artifact Version page.
- Click the **Delete** link to delete this artifact version from the artifact.

Properties table

This table contains custom properties for this artifact.

- Click on a Property Name to edit that property.
- If a "folder icon" precedes a property name, it denotes a Nested Property Sheet.
- This section also provides **Create Property**, **Create Nested Sheet**, and **Access Control** links to add more custom properties or change or set privileges on this artifact.

Artifact—create new or edit existing artifact

To create a new artifact

Enter information into the fields as follows:

Field Name	Description
Group Id	Enter a group name of your choice for this artifact. A Group Id (groupId) acts as a namespace for grouping related artifacts. The idea is that artifact "sdk" for group "platform" can co-exist with artifact "sdk" in group "ui" without there being any name collisions. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.
Artifact Key	Enter an identifier of your choice for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Artifact Version Name Template	A template for the names of artifact versions published to this artifact. For example: `$[/myArtifactVersion/groupId]:$[/myArtifactVersion/artifactKey]:$[/myArtifactVersion/version]` Produces a name like: `platform:sdk:1.0.0-36` Only absolute property paths and `/myArtifactVersion`-based paths are supported. If not specified, the "Artifact Version name template"server setting is used.

Click **OK** after filling in the fields and to go to the Artifacts page to see your new artifact listed in the first column.

To edit an artifact

- Click the **Access Control** link at the top of the page to add permissions to this artifact.
- You can add a Description or modify your existing description.
- You can add an Artifact Version Name Template or modify your existing template.
- Click **OK** to save your changes and return to the Artifacts page.

Artifact Versions

This page displays all artifact versions available on this ElectricFlow server.

You can search for artifact versions and save the search filters for later use. See Context Searching and Filtering on page 1244 for details on using the search capabilities on this page.

Links and actions at the top of the table

The "star" icon allows you to save this page to your Home page for quick access.

Column descriptions

Column Name	Description / Actions
Name	The name of the artifact version. Select an artifact version name to go to the Artifact Version Details page for that artifact version.
Artifact	The name of the artifact. Select an artifact name to go to the Artifact Details page for that artifact.
Group Id	The name of the group where this artifact version is a member.
Artifact Key	The "key" identifier for this artifact version.
Version	The artifact version represented, by default, as `major.minor.patch-qualifier-buildNumber`. **Note:** You will not see all 5 artifact version components specified if only 3 components were defined.
State	The current state of this artifact version. Possible values are:`available|publishing|unavailable`.
Modify Time	The last time this artifact version was modified.
Actions	**Edit** —Use this link to go to the Edit Artifact Version page to modify the artifact version on this row. **Delete—**Use this link to delete the artifact version on this row.

Artifact Version Details

This page displays detailed information about an artifact version.

Links and actions at the top of the table

- **Edit—**Use this link to go to the Edit Artifact Version page.
- **Access Control**—Use this link to set privileges for this artifact version. *For more information*, see the Access Control Help topic.
- The "star" icon allows you to save this artifact version information to your Home page.

General Information section

This section lists current information about the artifact version. While most of the information is static, two items link to other pages:

- Artifact—Click the artifact name to go to the Artifact Details page for the "owning" artifact.
- Published By—Click this link to go to the Job Details page for the job that published this artifact version. If this artifact version was published by an end user outside of a job context, this field will display None.

The "tabbed" sections

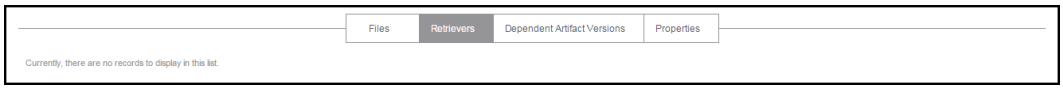
The next section of tabs allows you to select the type of information you want to see.

Files

This section shows the collection of files included in this artifact version. The left-pane displays the artifact version directory structure and the right-pane displays files and subdirectories for the currently selected folder in the left-pane. This structure is similar to Windows Explorer.

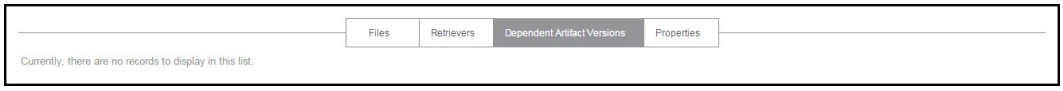
Retrievers table

This table lists the 50 most recent jobs that retrieved this artifact version and includes links to the Job Details page for those jobs.



Dependent Artifact Versions table

This section displays dependent artifact version queries registered with the artifact version during a publish operation. These queries are evaluated during retrieval and the most current artifact version that matches each query is returned along with the primary artifact version.



Properties table

This table contains custom properties for this artifact version.

- Click on a Property Name to edit that property.
- If a "folder icon" precedes a property name, it denotes a Nested Property Sheet.
- This section also provides **Create Property**, **Create Nested Sheet**, and **Access Control** links to allow you to add more custom properties.

Artifact Version—edit an existing artifact version

This page displays an existing artifact version you can modify.

Links and actions at the top of the table

- **Access Control**—Use this link to add or change permissions for this artifact version.
- The "star" icon allows you to save this page to your Home page for quick access.

To edit an artifact version

Field Name	Description / Action
Name	You can type-over the artifact version name in this field to change or modify the existing name.
Description	You can add a description or modify your existing description.

Field Name	Description / Action
Publisher	This is the job that completed the publish operation. Click this link to go to the Job Details page for this job. If this artifact version was published outside of a job context (for example, from ectool), the value for this field is "None".
State	An artifact version can be in one of three states: `available`, `unavailable`, and `publishing`. If the artifact version is: • in the Publishing state, the artifact version cannot be retrieved, but it will transition to the Available state when the publishing operation is complete. The web interface does not allow you to change the state if it is currently Publishing. • in the Available state, the artifact version can be retrieved. To confirm this state, the checkbox is "checked," and unchecking this box makes the artifact version unavailable. • in the Unavailable state, the artifact version cannot be retrieved. "Checking" the box changes the artifact version state to Available.

Repositories

This page displays all artifact repositories available to this ElectricFlow server.

Links and actions at the top of the table

- **Add Repository**—Use this link to go to the New Repository page to add another artifact repository.
- "Star" icon—Save this page to your Home page for quick access.

Column descriptions

Column Name	Description / Actions
Repository Name	The name of the artifact repository. Select a repository name to go to the Edit Repository page if you need to modify that repository. **Tip:** If you have multiple repositories and want to change the search order for retrieving artifacts, drag the icon (in the far left column) up or down to reposition an artifact to the order you need.
Zone	The name of the zone where this repository resides.

Column Name	Description / Actions
URL	The server URL is in the form `protocol://host:port/`. Typically, the repository server is configured to listen on port 8200 for `https` requests, so a typical URL looks like `https://host:8200/`
Enabled	Select this checkbox to enable or disable this artifact repository. By disabling this repository, users will not be able to retrieve any artifacts stored in this repository.
Description	A plain text or HTML description previously supplied for this repository.
Actions	**Delete**—Delete the artifact repository on this row.

Repository—create new or edit existing repository

To create a new repository

Enter information into the fields as follows:

Field Name	Description
Name	Enter any name of your choice for this repository.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
URL	The server URL is in the form `protocol://host:port/`. Typically, the repository server is configured to listen on port 8200 for `https` requests, so a typical URL looks like `https://host:8200/`.
Zone	Use the drop-down menu to select a zone for this repository. If no zone is selected, this repository will reside in the *default* zone.
Enabled	Select this checkbox to enable this repository. If this repository is disabled ("unchecked"), users will not be able to retrieve any artifacts stored in this repository.

Click **OK** after filling-in the fields and to return to the Repositories page to see your new repository listed in the first column.

To edit a repository

You can:

- Use the **Access Control** link at the top of the table to set permissions on this repository
- Change the repository name

- Add a description or modify the existing description
- Modify the URL
- Select a different zone.
- Enable or disable the repository

Click **OK** after making any modifications and to return to the Repositories page.

Database Configuration

If you choose not to use the ElectricFlow built-in (default) database, use this page to configure your alternate ElectricFlow-supported alternate database (such as MySQL, SQL Server, or Oracle) to communicate with ElectricFlow.

> **Important:** The ElectricFlow built-in database is not recommended for production use.

Fill in the fields as follows:

Field Name	Description
Database Type	Select your database type from the drop-down menu. If the **Built-in (MariaDB)** option is selected, no other information is required. **Note:** The **Built-in (MariaDB)** option is supported only for the built-in database that is installed by ElectricFlow. Any other MariaDB database is not supported; that is, you cannot install another MariaDB instance and use it with ElectricFlow. Also, changing the database configuration options (such as the database name, host name, and credentials) to use any other database such as MySQL when using the **Built-in (MariaDB)** option is not supported.
Database Name	Enter your database name.
Host Name	Enter the host name for your database server.
Port	Use the default port or enter a new port number.
Database Credentials	• **User Name**—Accept the default "commander" user name or enter the appropriate user name for your database. • **Password**—Enter the database password.

> **Note:** When you set the SQL server as the database, you can only use a non-named database.

Click **Save and Restart Server** after entering information in all fields. You may need to consult with your Database Administrator if you lack all of the information required on this web page. For more

information, see the *ElectricFlow Installation Guide* at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html.

Defect Tracking Configurations

This page displays Defect Tracking configurations known to the ElectricFlow server—ElectricFlow integrates, using plugins, with numerous defect tracking systems.

Link at the top of the table

Use the **Create Configuration** link to create a new Defect Tracking configuration.

Column descriptions

Column Name	Description
Configuration Name	The name you provided to the configuration you created.
Description	This column describes the Defect Tracking configuration type.
Plugin	The name of the plugin where the Defect Tracking integration resides.
Actions	• **Edit**—Click this link to modify an existing configuration. • **Delete**—Click this link to remove the configuration.

Defect Tracking—create new or edit existing configuration

Use this page to define a new defect tracking configuration or modify an existing defect tracking configuration. A configuration is a collection of properties that define how ElectricFlow communicates with a particular defect tracking system.

After creating a defect tracking configuration, your entry will appear in the table on the Defect Tracking Configurations web page— to see this web page, select the Administration > Defect Tracking tabs.

To create a defect tracking configuration

From the Defect Tracking Type drop-down menu, select your defect tracking system—ElectricFlow integrates with numerous defect tracking systems. Each integration was created using plugin technology.

Click the **Submit** button and a set of fields is displayed to enter information to create your configuration.

Enter information into the fields as follows:

Field Name	Description
Configuration Name	Enter a unique name for this defect tracking configuration—any name you choose.

Field Name	Description
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Login As	• **User Name**—This is the name ElectricFlow needs to use to communicate with your defect tracking system. For example, you may be using a special "read-only" user name similar to "Build" for your user name. • **Password**—This is the password for the specified User Name. • **Retype Password**—Type the password again. • **URL**—This is the URL to use to connect to a JIRA server. For example, you might use something similar to: `http://10.10.10.10:8080` or `http://yourJIRAserver`

Click **Submit** to save your information.

Using the defect tracking integration

1. Go to Projects > select a project > select a procedure.
2. To create a New Step for defect tracking, select the Plugin link.
3. In the Choose Step panel, select Defect Tracking from the left pane, then select the defect tracking system you configured.

   The right-pane now shows the types of steps available for your configuration.
4. Select the step you need and automatically go to the New Step page.

   On the New Step page, notice the Subprocedure section now contains the defect tracking integration you configured and the step you chose.
5. Enter the remaining information to create your defect tracking step.

See the Defect Tracking Help topic for more information.

To edit an existing defect tracking configuration

Modify any of your defect tracking configuration information by "typing-over" any previously entered information. Click **Submit** to save your modified defect tracking information.

Defect Tracking Reports

This is a report of URLs to JIRA defects associated with the ElectricFlow job. The report was compiled by searching job properties for JIRA defect IDs then querying the configured JIRA server for the current defect state.

If you would like to create a defect tracking configuration, go to Administration > Defect Tracking and click the **Create Configuration** link to see the New Defect Tracking Configuration web page. Access the Help link on that page for more information.

For more information about Defect Tracking, see the main Defect Tracking Help topic.

Email Configurations

This page displays all existing email configurations.

- Click a Configuration Name to go to the Edit Email Configuration web page to modify an existing email configuration.
- Click the Add Configuration link to create new email configuration.
- Action column—This column contains two links to Copy or Remove an Email Configuration.

This page also lists the email server name and the email notification sender.

Email Configuration—create new or edit existing email configuration

Use this page to specify an email server to ElectricFlow. If you do not create an email configuration, you cannot send email notifications to individuals or groups.

If you have multiple users or groups in remote locations that use a different mail server, create additional email configurations to accommodate those locations if they need to receive ElectricFlow notifications.

To create a new email configuration

Enter information into the fields as follows:

Field Name	Description
Name	Unique name to identify the email configuration
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Mail Protocol	Specify either `SSMTP` or `SMTP`
Mail Host	Mail server name
Mail Port	(Optional) Port number for the mail server. Protocol software determines the default value (`25` for SMTP and `465` for SSMTP). Specify a value for this argument when a nondefault port is used
Mail From	Email address used as the email sender address for ElectricFlow notifications. This must be a fully-qualified email address (meaning that it must be of the form `<name>@<domain>`)
Mail Username	Individual name or a generic name such as `ElectricFlow`
Mail Password	Password for the mail user name

Click **Save** after filling in all fields and then click the **Test** button to make sure your email configuration works. Then enter a destination email address into the **Email Configuration Test** popup that appears and click **Send**.

To edit an existing email configuration

Use the **Access Control** link at the top of the page to add or change permissions. You can modify any information or add new information. Click **OK** after completing your edits.

Email Notifier—create new or edit existing email notifier

Use this page to set up email notifiers. You must have already set up an email configuration so ElectricFlow knows which email server to use when sending any notifications. If you have not configured an email server to communicate with ElectricFlow, click the **Administration** > **Email Configurations** tabs to do this configuration now.

For jobs and job steps, two types of email notifiers:

- On Start Notifier – Sends an email when a job or job step starts.
- On Completion Notifier – Sends an email when a job or job step completes.

For workflow states, three types of email notifiers:

- On Enter Notifier – Sends an email when the state becomes the workflow's active state.
- On Start Notifier – Sends an email after the state's subjob or subworkflow starts. If no subjob or subworkflow is defined for that state, the notifiers will not be sent.
- On Completion Notifier – Sends an email after the state's subjob or subworkflow completes. If no subjob or subworkflow is defined for that state, the notifiers will not be sent.

Using email notifiers:

- Attaching an email notifier to a procedure results in a corresponding email notifier associated with the job that is created when the procedure is executed.
- Attaching an email notifier to a procedure step results in a corresponding email notifier associated with the job step created when the procedure referencing the procedure step is executed.
- Attaching an email notifier to a state definition results in a corresponding email notifier associated with the state that is created when the workflow is executed.

To create a new email notifier

Enter information into the fields as follows:

Field Name	Description
Name	This name can be an arbitrary text string.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.

Field Name	Description
Condition	Use the pull-down menu to select the type of condition you need for this email notifier. Edit the auto-supplied condition in the text box or add a completely new script for your purpose. The condition specifies whether the notifier should send a message depending on the result of a property expansion. If the result is empty or non-zero, the message is sent. If the result is "0", the message is not sent

Field Name	Description

Formatting Template	Use the pull-down menu to select from a list of global, ready-to-use formatting templates. Depending on the type of email notifier you are creating, the available template choices in the drop-down menu will be different. Make sure the content is formatted correctly, i.e., no illegal characters or spacing.
	
	To customize your template, edit the auto-supplied text in the Formatting Template text box or you can add a completely new script for your purpose. Any edits made in this text box will not be saved to the global template. See Email notifier templates on page 1107 for a list of available templates and instructions for making global modifications to these templates.
	
	To create a custom template, the basic structure is:
	• zero or more email header lines
	• blank line
	• message body
	
	The template undergoes property expansion in a `job`, `jobStep`, or `state` context. In addition to normal property paths, an additional event object is accessible via `/myEvent/`.
	
	The following example is a sample formatting template for a job:
	
	`Subject: Job '$[jobName]' from procedure '$[procedureName]' $[/myEvent/type]—ElectricFlow notification`
	`cc: owner@example.org`
	`bcc: admin@example.org`
	
	`ElectricFlow Notification—from email notifier '$[/myEvent/notifier]'`
	
	`Job '$[jobName]' $[/myEvent/type] at: $[/myEvent/time]`
	
	`Project: $[projectName]`
	`Procedure: $[procedureName]`
	`Started: $[start]`
	`Completed: $[finish]`
	`Directory Name: $[directoryName]`
	`Launched by User: $[launchedByUser]`
	`Outcome: $[outcome]`
	`Error Code: $[errorCode]`
	`Error Message: $[errorMessage]`
	
	This example demonstrates how you can send this notification using CC or BCC fields in addition to specified Destinations. Because headers are interpreted by SMTP, normal email rules apply.

Field Name	Description
Email Configuration	Click inside this field or start typing to bring up a list of possible email configuration names. An email notifier that does not specify an email configuration will use the configuration named 'default' if it exists.
Destinations	A space-separated list of valid email addresses, email aliases, or ElectricFlow user or group names, or a property reference that expands into such a list, or you can enter an LDAP DL name (group name).

Click **OK** to save your email notifier configuration.

To edit an existing email notifier

Modify or add information to any of the fields and click **OK** to save your changes.

Email notifier templates

The table below lists global email notifier templates for your use. Each template name is a link to an example of the template output and the script.

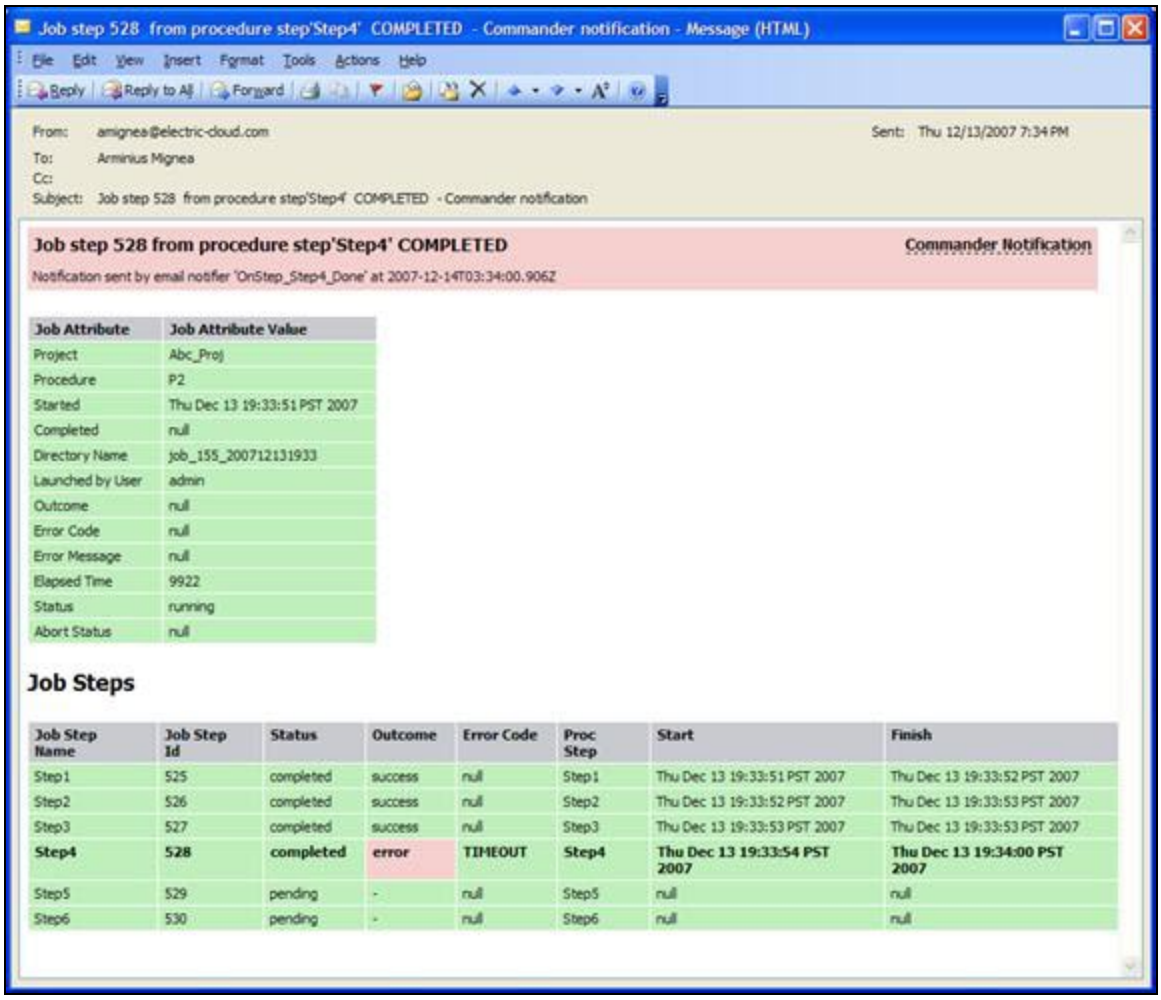
Global templates are stored as property sheets under the "`/server/ec_notifierTemplates`" property sheet. You can modify template properties or add new global templates if you have modify privileges on the server property sheet. An email notifier will be updated if changes are made to its global template.

Template Name	Format	Usage	Description
Job Summary text notification	Plain text	Procedure	Formatting template for Procedure starting/completion notifications using the long form of property names (for example, `$[/myJob/jobName]`)
Step summary text notification	Plain text	Procedure step	Formatting template for Procedure Step starting/completion notifications using the long form of property names (for example, `$[/myJobStep/jobStepId]`)
State summary text notification	Plain text	State	Formatting template for State notifications using the long form of property names (for example, `$[/myWorkflow/workflowName]`)
Job summary HTML notification	HTML	Procedure	HTML formatting template for Procedure starting/completion notifications

Template Name	Format	Usage	Description
Step summary HTML notification	HTML	Procedure step	HTML formatting template for Procedure Step starting/completion notifications displaying information about the Job and the Job Step
All steps HTML notification	HTML	Procedure step	HTML formatting template for Procedure Step starting/completion notifications displaying information about the Job and many Job Steps
Approval request HTML notification	HTML	State	HTML formatting template for State notifications displaying information about the workflow and all of its states
Workflow summary HTML notification	HTML	State	HTML formatting template for State notifications providing the recipient with links to view or take a manual transition

Email Notifier Template—Html_JobStepTempl_AllSteps.txt

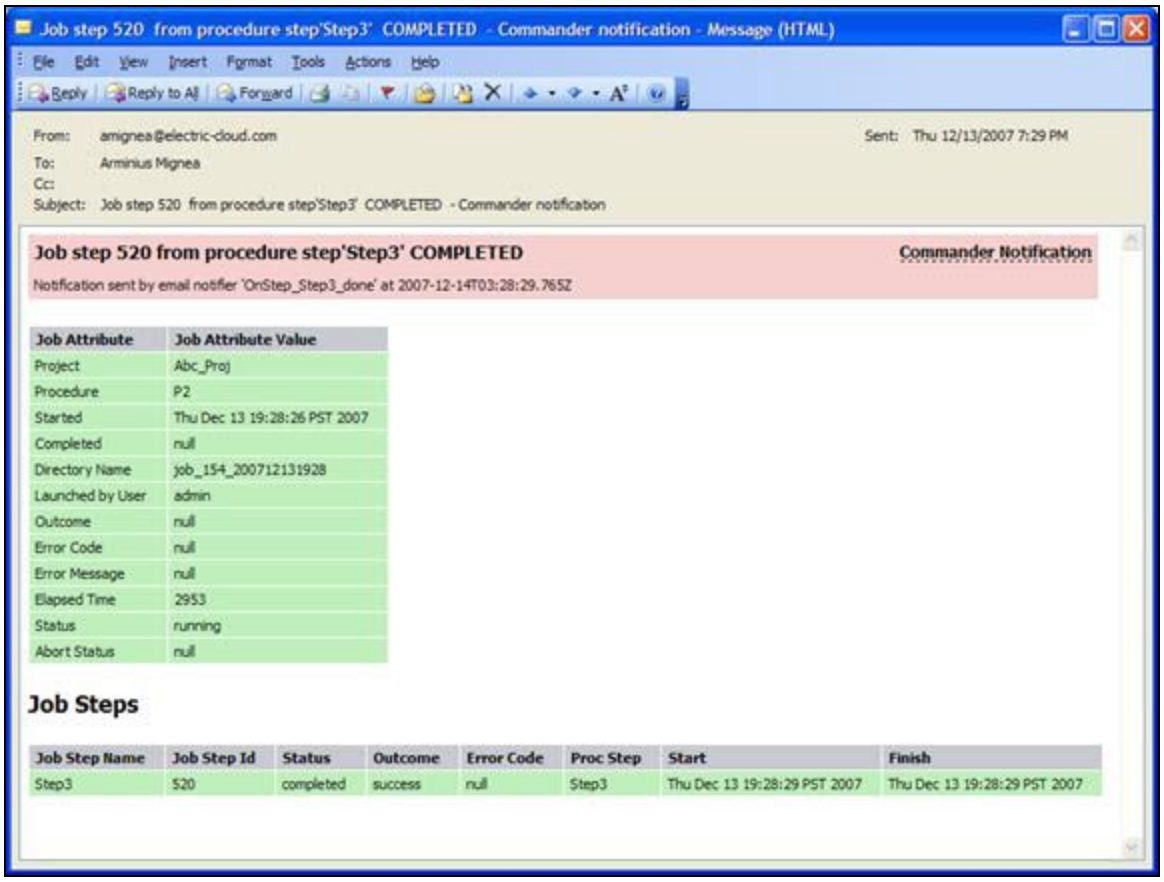
If you use this template, the following screen example is similar to how your email notifier will look.



1. Go to the corresponding template available in `<installDir>\src\samples\notifier`
2. Copy the script into a text editor to edit it for your purpose
3. "Copy and paste" your version into the Formatting Template field.

Email Notifier Template—Html_JobStepTempl_SingleStep.txt

If you use this template, the following screen example is similar to how your email notifier will look.

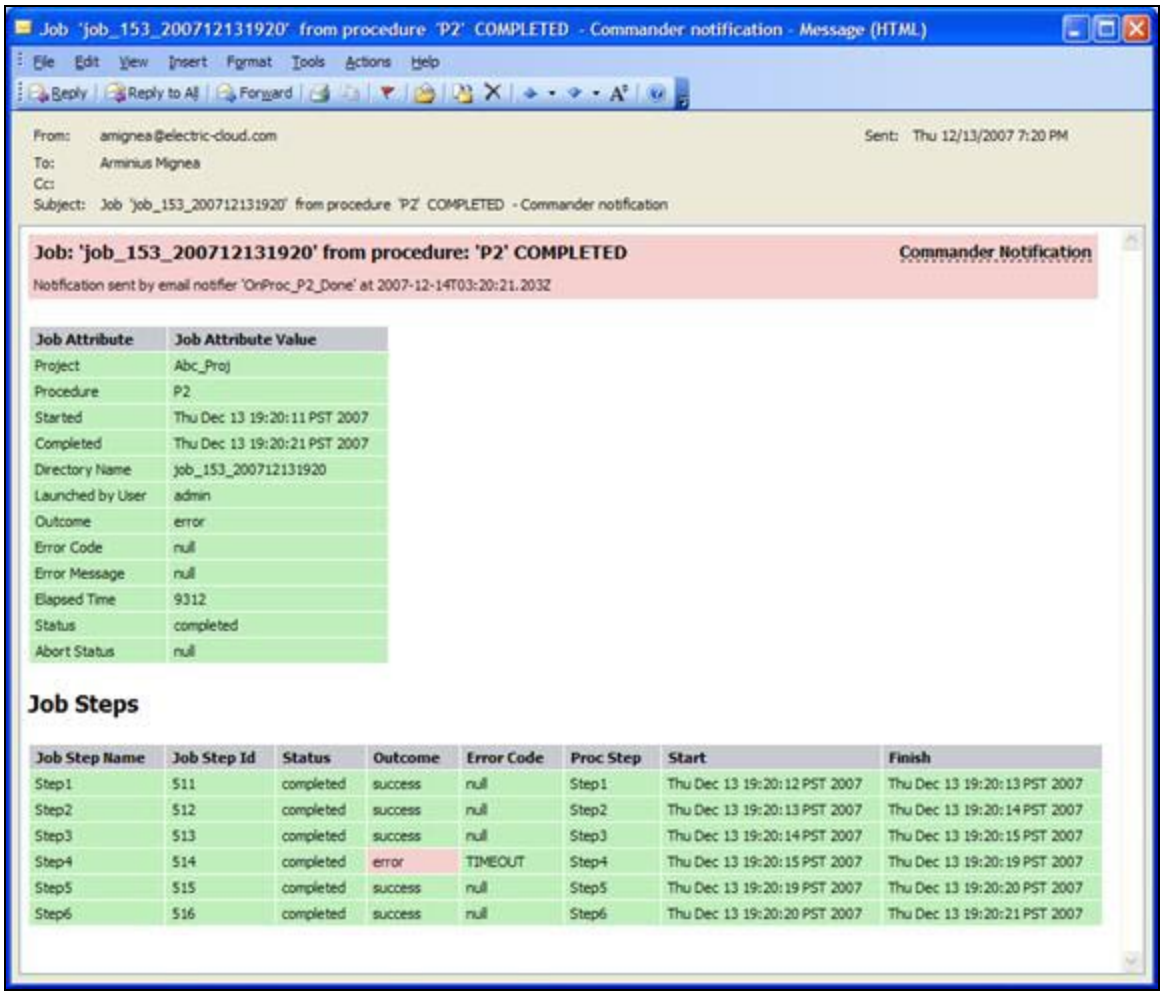


Note: In the following script, myJob.steps and myJob.jobSteps return an array of step names.

1. Go to the corresponding template available in `<installDir>\src\samples\notifier`
2. Copy the script into a text editor to edit it for your purpose
3. "Copy and paste" your version into the Formatting Template field.

Email Notifier Template—Html_JobTempl.txt

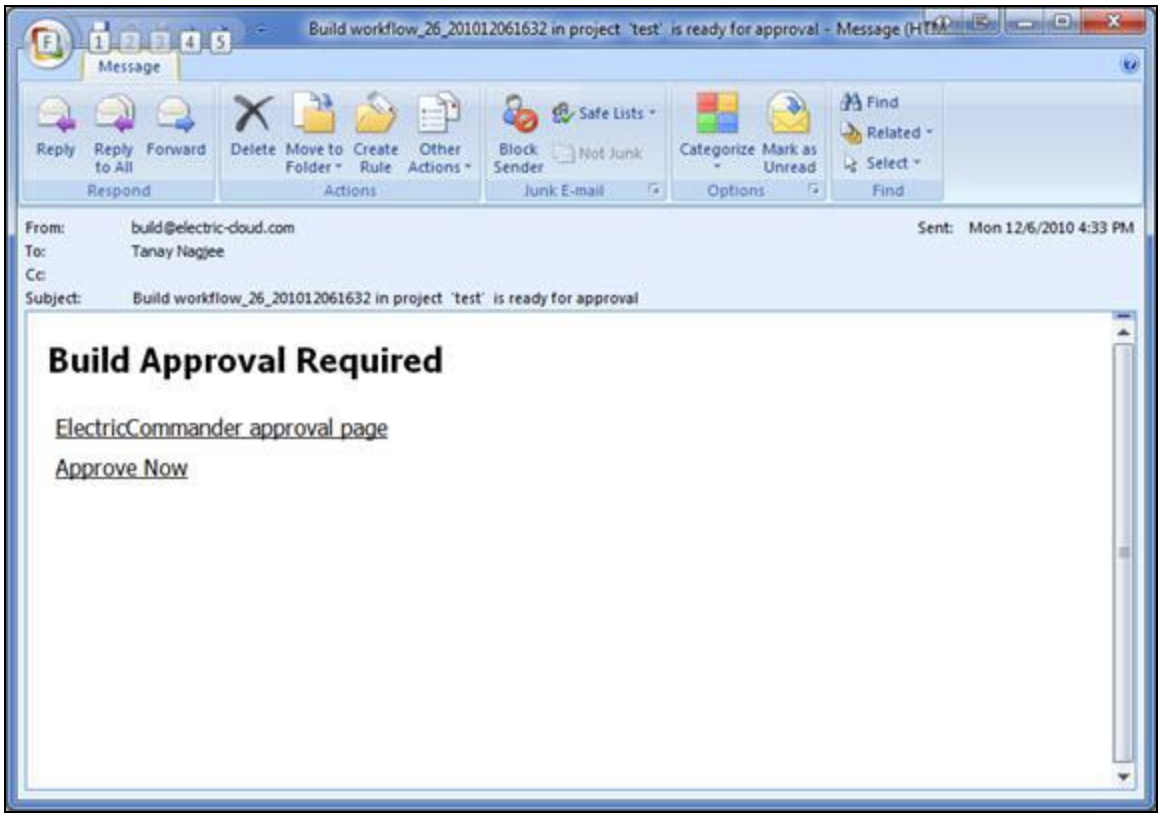
If you use this template, the following screen example is similar to how your email notifier will look.



1. Go to the corresponding template available in `<installDir>\src\samples\notifier`
2. Copy the script into a text editor to edit it for your purpose
3. "Copy and paste" your version into the Formatting Template field.

Email Notifier Template—Html_StateTemplate_ApproveWorkflow.txt

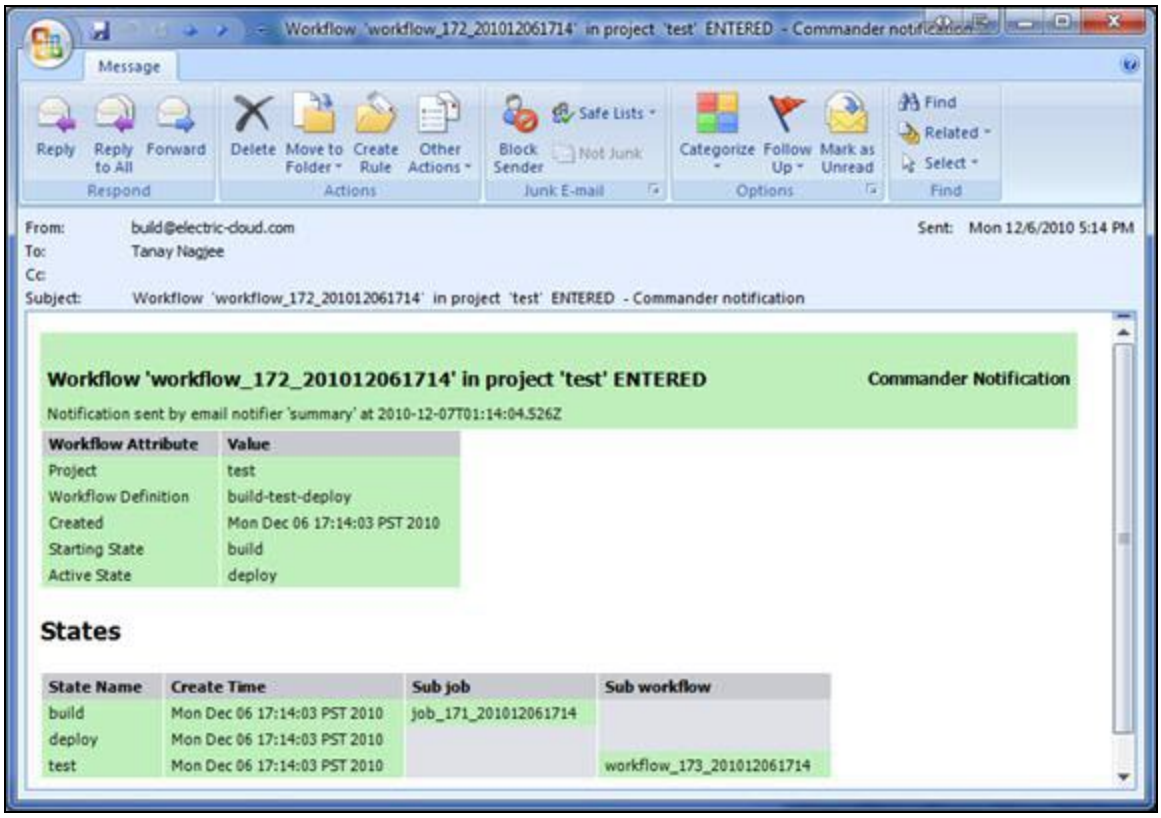
If you use this template, the following screen example is similar to how your email notifier will look.



1. Go to the corresponding template available in `<installDir>\src\samples\notifier`
2. Copy the script into a text editor to edit it for your purpose
3. "Copy and paste" your version into the Formatting Template field.

Email Notifier Template—Html_StateTemplate_FullWorkflow.txt

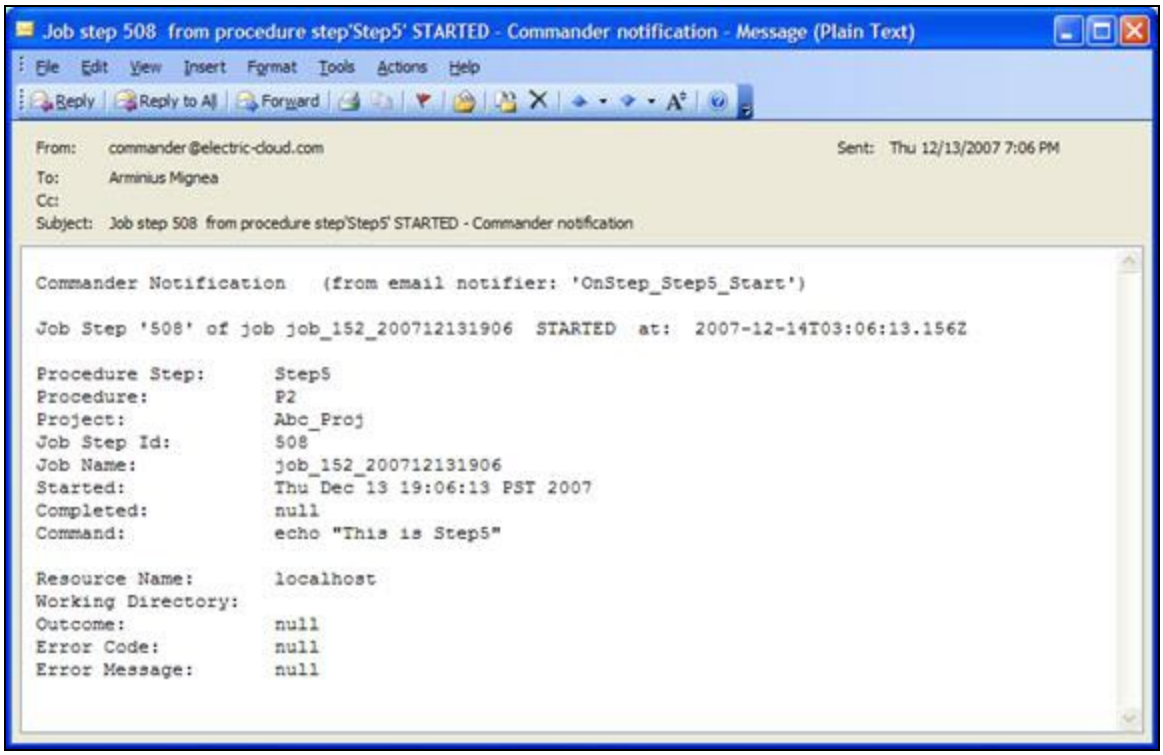
If you use this template, the following screen example is similar to how your email notifier will look.



1. Go to the corresponding template available in `<installDir>\src\samples\notifier`
2. Copy the script into a text editor to edit it for your purpose
3. "Copy and paste" your version into the Formatting Template field.

Email Notifier Template—JobStepTempl_FullProps.txt

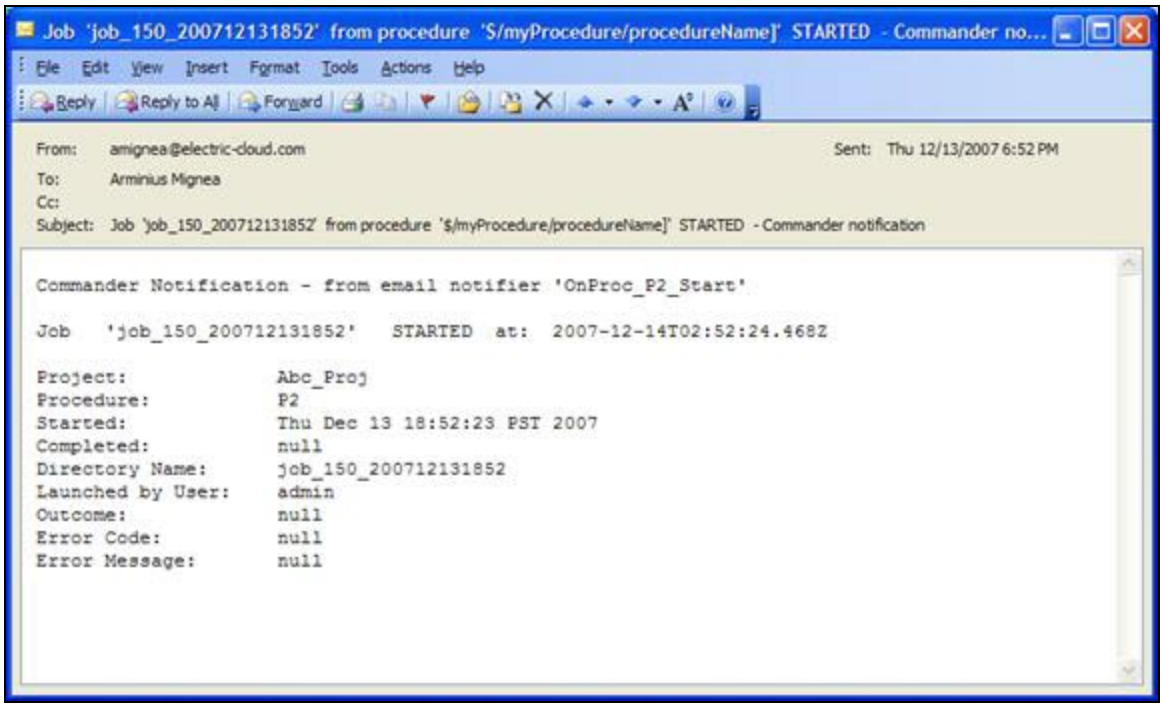
If you use this template, the following screen example is similar to how your email notifier will look.



1. Go to the corresponding template available in `<installDir>\src\samples\notifier`
2. Copy the script into a text editor to edit it for your purpose
3. "Copy and paste" your version into the Formatting Template field.

Email Notifier Template—JobTempl_FullProps.txt

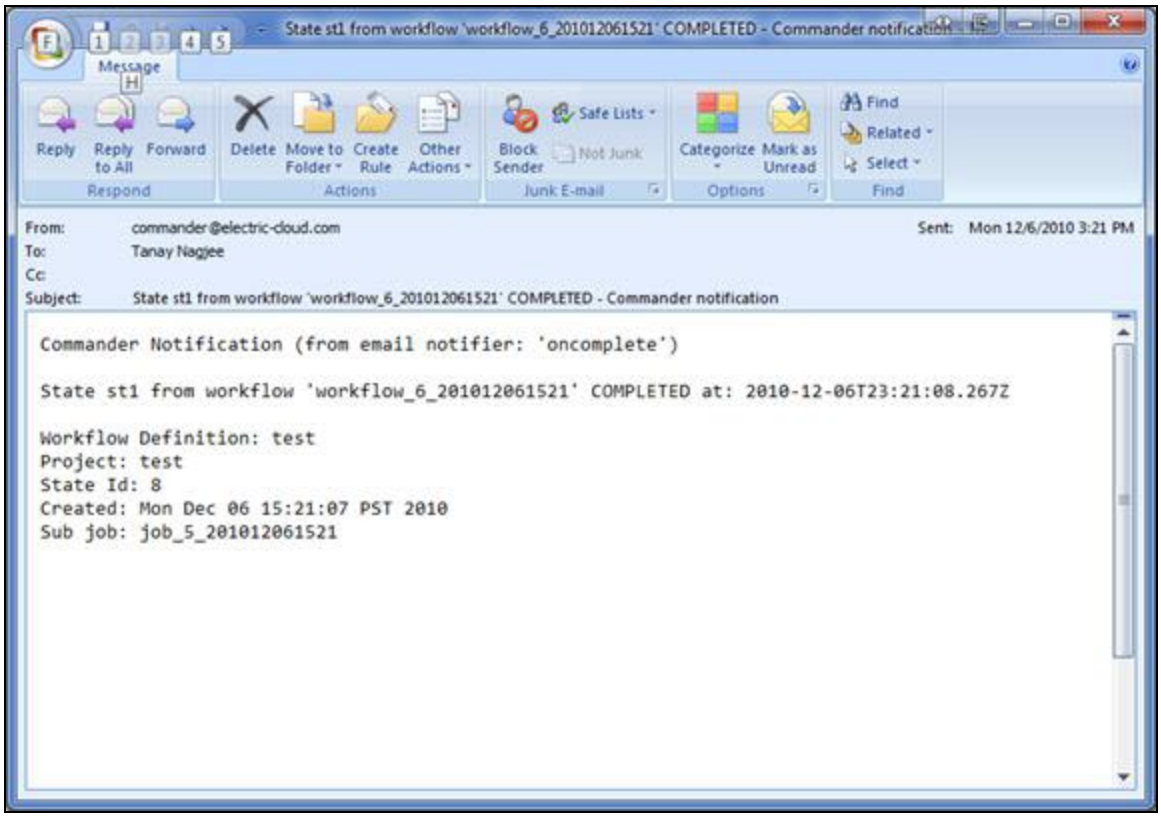
If you use this template, the following screen example is similar to how your email notifier will look.



1. Go to the corresponding template available in `<installDir>\src\samples\notifier`
2. Copy the script into a text editor to edit it for your purpose
3. "Copy and paste" your version into the Formatting Template field.

Email Notifier Template—StateTemplate_FullPropertyPaths.txt

If you use this template, the following screen example is similar to how your email notifier will look.



1. Go to the corresponding template available in `<installDir>\src\samples\notifier`
2. Copy the script into a text editor to edit it for your purpose
3. "Copy and paste" your version into the Formatting Template field.

Event Log

This page displays a log of events generated anywhere in the system, including jobs and workflows.

- To see only events for a single workflow, select the Workflows tab, then a workflow Name to go to the Workflow Details page and click the **View Log** link at the top of the page.
- To see only events for a single job, select the Jobs tab, then the Job name to go to the Job Details page and click the **View Log** link at the top of the page.

  > **Note:** When running a job in a cluster, the name of the server managing the job is recorded in the job-log. If the management of the job moves to a different node (if the current node crashes for example) the change is logged into the job log as well.

- To see only events for a specific object, select the Search tab to go to the Searching and Filtering page.

For example, You might want to select the Object Type, "Log Entry", then click the **Add Intrinsic Filter** link. Select the down-arrow where you see "Container" auto-populated and select "Container Type". Use the "equals" operator, then select the next down-arrow to choose an object. Click **OK** to start the search.

> **Note:**
> - From the Administration tab, the default view for this Event Log page is the warning (WARN) level.
> - For workflow and job event logs, the default view from their respective pages is the information (INFO) level.

Column descriptions

Time—This is the time the event was logged.

Severity—Can be either INFO, WARN, or ERROR. Use the pull-down menu at the top of the table to change between these views.

User—The user associated with the event—the user of the session at the time the event was generated. Clicking on a project link in this column takes you to the Project Details page for that project.

Container—Typically, this is the type and name of the workflow or job with a corresponding ID. Clicking on a link in this column takes you to the Job Details page for that job or workflow.

Subject—The object associated with the message. Clicking on a link in this column takes to you the source for this object.

Message—The message text will either be informational or display the warning or error message. The message may contain important information about a resource or workspace issue.

Configuring the Event Log

Because log entries can accumulate in large numbers, you can configure the Event Log for auto-deletion. By default, Event Log retention is 30 days. The ElectricFlow server automatically marks old log entries for deletion and the background deleter cleans out old log entries along with its other tasks.

To change the default settings, go to Administration > Server > Settings and modify the following two fields:

Event log retain time—Default is 30 days (in "minute" units).

> **Note:** Setting the "retain" period to "0" disables the automatic deletion mechanism, allowing you to use an external cleanup script to implement a custom cleanup policy.

Maximum background delete delay—Default is 3600 seconds (or one hour).

Click **OK** to save your changes.

Automation Platform Home Page

Job Configurations

Shortcuts

Jobs Quick View

Reports

Overview

This page (https://<*ElectricFlow_server*>/commander/) provides a convenient console for running jobs and viewing results.

> **Note:** When using the EC-Homepage plugin to share your Home page sections, you will not see certain links that are normally available. For example, if a particular Jobs Quick View category is shared, you cannot delete or modify that category. You can perform these functions only if the category belongs to you alone.

For information on sharing your Home page configuration, see Home page configurations.

Job Configurations

Procedures in ElectricFlow can contain complex sets of parameters— making it difficult to remember the correct parameters for a particular situation and tedious to re-enter those parameters every time the procedure is invoked. Job Configurations provide a one-click solution to this problem. When you create a job configuration, you enter all the information needed to run a procedure, including parameters or a credential. All job configurations are displayed here on the Home page. Invoke a particular configuration by clicking its name in the Job Configurations section.

Create Job Configurations three ways

- On the Home page, create a job configuration from "scratch" by clicking the **Create** link in the Job Configurations section. In the Create Configuration pop-up menu, select the project and procedure you want to use for creating this configuration.
- From the Job Details page for a previously invoked job, click the **Save Configuration** link at the top of the page. Your saved job configuration is displayed on your Home page.
- From the Edit Schedule page, click the **Save Configuration** link at the top of the page. Your saved configuration is displayed on your Home page.

Shortcuts

Use shortcuts to save frequently visited ElectricFlow web pages, so those pages are immediately accessible. You can create a shortcut to any page on the web also.

Create Shortcuts two ways

- Mouse-over the "star" icon at the top of any automation platform page and click "Add current page" to add that page to the shortcut list. Mouse-over the star icon again and click "Remove current page" to remove the shortcut for that page. The star icon is yellow for pages saved as a shortcut and gray for those pages not saved as a shortcut.
- Click the **Create** link in the Shortcut section and provide a name and URL to create a shortcut.

To modify or update a shortcut, click the **Edit** link adjacent to the shortcut you want to change.

Shortcuts can be accessed conveniently from any automation platform web page. Mousing-over the star icon displays a list of shortcuts saved by the current user. Click on a shortcut name to view the page.

Note: The Shortcut section may contain static entries that cannot be deleted. And if you "share" your Home page, the Edit link will not be available for shared items.

Jobs Quick View

Perhaps only a few jobs on this server are of interest to you. For example, you may care about jobs you launch manually and official builds for the products you work on, but you may not care about production builds for other products or personal jobs for other users.

The Jobs Quick View allows you to define job categories that are interesting to you.

The Home page displays the most recent jobs in each category, and you can easily click-through to get more details about any of those jobs. For example, clicking on a job name takes you to that job's Job Details page.

Create a job category

- Click the **Add Category** link in the Jobs Quick View section.
  After creating a category, results are displayed on the Home page. Clicking the **Details** link displays a summary to the right of the category. In addition to job status and other diagnostic information, the summary displays running steps and failed steps (containing errors and warnings).
- Click the **Modify** link to edit the Jobs Quick View category or the **Delete** link to remove that job from the Jobs Quick View category.
- Click on the **Details** link to see job summary information—the summary remains visible, regardless of mouse location, until you click somewhere else on the page.

If you "share" your Home page, the Modify and Delete links will not be available for shared items.

Reports

You can configure reports you would like to see on a regular basis and display a "thumbnail" report graphic in this section.

- Click the **Add Report** link to go to the New Reports page.
  After filling in the information on the New Reports page, you will see a thumbnail view of your report (after it runs) on your Home page.
  Note: If the drop-down menu on this page is empty, click the **Help** link on the New Report page for more information.
- Click the Collapse/Expand (+/-) box to see the full thumbnail report or just the report title.
- Click the **Rename** link if you need to rename your report.
- Click the **Remove** link if you need to remove this report from your Home page.

Note: If you "share" your Home page, the Rename and Remove links are not available for shared items.

Job Configuration

To create a new job configuration

Enter information into the fields as follows:

- **Name**—Type a name you choose for the job configuration.
- **Procedure**—Displays the current project and procedure. Click **Change** to use the pop-up menu to select a new project/procedure.

Parameters

Depending on the Project and Procedure you selected, a set of Parameters is displayed. Enter all parameter values or just the required values if differentiated.

Advanced

Priority:

Use the drop-down menu to select the priority for this run procedure. Available priorities are: low, normal (default), high, or highest.

- You can select the job's priority here, on Run Procedure web page, or when you schedule the procedure to run at a regular time or interval. When a job is launched, its priority cannot be changed.
- Priorities take effect when two or more job steps in different jobs are waiting for the same resource. When the resource is available, it will be used by the job step that belongs to the job with the highest priority. If the priority level is the same, the resource will be used by the job step that belongs to the job with the lowest job ID number. If the job steps are in the same job, the resource will be used first by the step with the lowest job step ID number.
- If using ectool, the priority can be set by passing `"--priority=<low|normal|high|highest>"` to a `runProcedure`, `createSchedule`, or `modifySchedule` operation.

> **Note:** To raise the job priority level above "normal," the user who launches the procedure must have execute permission on the system-level access control list for priorities.

Impersonation:

Click one of the three "radio" buttons to determine the default login user to use when the configuration is run.

- Use pre-defined credential—If you select this option, ElectricFlow looks for a credential first in the top-level procedure and then in its project. If it finds a credential, it uses the login from that credential as the default for the job. If there is no credential in either place, by default, the job's steps run as the user under which the agent for the step is running.
- Use specific credential—If you select this option, you can choose a particular credential from those defined for the project containing the procedure —the login user from that credential is used as the default for the job.
- Use a specific user—If you select this option, you can enter a login name. When the configuration is run, you will be asked to enter the password for that account. The specified login will be used as the default for the job.

For more information on credentials and user impersonation, see the Credentials and User Impersonation Help topic.

Click **OK** when all selections are complete. Return to the Home page to see your new job configuration displayed.

To edit an existing job configuration

Change or modify any of your previously specified information. Click **OK** when all selections are complete. Return to the Home page to see your edited job configuration displayed.

Shortcuts

Use this page to create a **New Shortcut** or **Edit** an existing Shortcut. Shortcuts appear on your Home page and provide quick access to other web pages, either in ElectricFlow or elsewhere.

- To create a new shortcut, type-in a shortcut name and the URL.
- To edit an existing shortcut, re-type the new shortcut name or URL (or both).

The easiest way to create a shortcut to an ElectricFlow automation platform page, such as the page for a particular project or procedure, is to navigate to that page and then click the "star" icon in the upper-right corner. This action automatically creates a shortcut and the star turns yellow to indicate a shortcut is in effect for the page.

If you click on the star icon again, the shortcut for that page is canceled.

Click **OK** to save your new or edited information and see your new/edited shortcut on the Home page.

Jobs Quick View

A quick view category provides a convenient way to select a group of jobs that interest you, such as "all production builds for product xyz version 3.6" or "all jobs I invoke manually." ElectricFlow displays the most recent jobs for each category on your Home page so you can review them easily.

To create a new jobs quick view category

Enter information into the fields as follows:

Name—Type a name for your new job category.

Number of Jobs—Type the maximum number of jobs you want to see in your quick view summary. The summary displays the most recent applicable jobs.

Include Last Success—Select this check box if you want to see the last successful job in this category, even if it was not one of the most recent jobs.

Two Filter categories:

- **Intrinsic filters**—These filters provide a convenient way to access certain well-defined fields for jobs.
- **Custom filters**—These filters allow you to access a much broader range of values, including custom properties. Any values accessible through an intrinsic filter can be checked using a custom filter also (though not as conveniently).

Each filter specifies three values: the name of a particular property to check, a value it should be compared with, and relational operators such as "equals" or "less than." For example, if you select "Outcome" in an intrinsic filter with operator "not equals" and value "Success," the filter selects jobs that did not complete successfully because they had errors or warnings.

For an intrinsic filter, select a property by choosing one of several predefined properties from a list. For a custom filter, enter a property name. This name can be either the name of an intrinsic property for the job (such as "status" or "procedureName"), or the name of a property defined in the global property sheet for that job, or the name of a parameter for that job.

For the relational operator, select one of the values in the list. The behavior of the relational operators is determined by the underlying database. In most cases, comparisons are done using string comparison; operators such as "less than" may not be useful on numbers because "12" is considered less than "9". For some built-in properties, the database treats values according to a specific type such as number or date, so comparisons are implemented in the way appropriate for that type.

The relational operator "like" invokes an SQL wildcard comparison where "%" is the wildcard character. For example %test% matches "first test" and "tests failed" but not "Test 44".

Note: Job Quick View filters allow property path references. However, property path references are not allowed in search parameters on the Search page.

Click **OK** after making your selections to save your new job category and see it on the Home page.

To edit an existing jobs quick view category

Keep or change any of the previously entered information. See the Filter information above to add or change existing custom or intrinsic filters.

Click **OK** after making your selections to save your modified job category and see it on the Home page.

New Report

Enter information in the following fields to define a new report to display on your Home page.

Field Name	Description
Project	The name of the project that contains the report you want to view.
Report	The report name you entered in the Report Title field when you configured this report. If you did not specify a "thumbnail" view when you defined your report, you will not see it here.
Title	This is the report title you want to see on your Home page to identify this report.

Click **OK** to continue. After the report runs, the report you specified will be displayed on your Home page and updated whenever the report is generated again.

To populate the drop-down menu

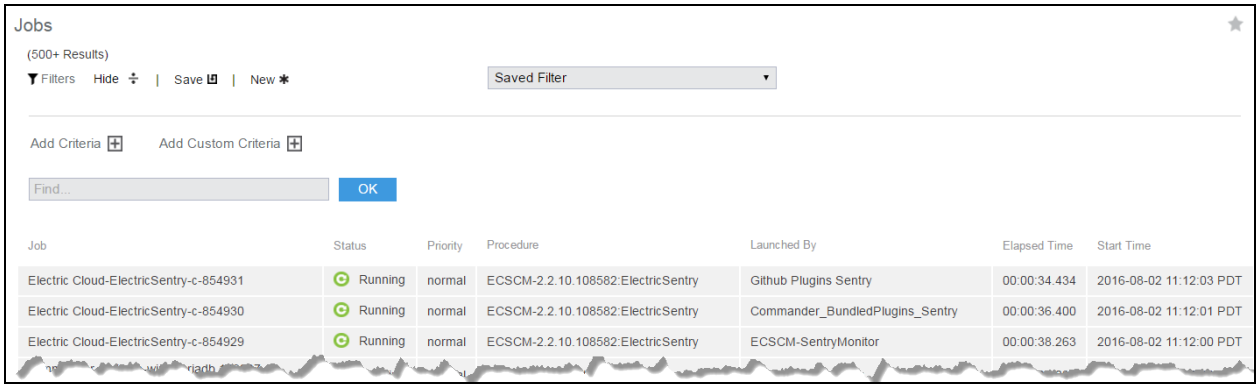
- Go to the Project Details page for the project containing the reports you want to view.
- Select the Report tab.
- Click the **Create Report** link.
- Check the **Create thumbnail?** check box, which enters this report's name in the New Reports drop-down menu.

All reports, ***except*** the Procedure Usage report, are available as "thumbnail" reports on your Home page.

Jobs

This page displays all jobs in the ElectricFlow system, both running and completed jobs. You can:

- Sort the Job, Status, Elapsed Time, and Start Time columns by clicking on the column name.
- Search for jobs and save the job search filters for later use. See Context Searching and Filtering on page 1244 for details about using the search capabilities on the Jobs page.



Column descriptions

- **Job**—Click Job to sort the column alphabetically or click on a job name to go to that job's Job Details page.
- **Status**—Click Status to sort this column by Running, Success, Warning, Error, or Aborted.
- **Priority**—Displays the job's priority set by a "run procedure" command or by the job's schedule.
- **Procedure**—In this column you can click on a project name to go to the Project Details page or click on the procedure name to go to the Procedure Details page for that procedure.
- **Launched By**—Click a name in this column to go to the page for the schedule (Edit Schedule page) or the user that ran the job.
- **Elapsed Time**—Click Elapsed Time to sort the time values from longest to shortest time, or the reverse.
- **Start Time**—Click Start Time to sort this column from the start time of the first job to the start time of the most recent job, or the reverse.

- **Actions**—Use this column to abort a running job or delete a completed job.
  - Aborting a job requires the execute privilege on the job—not just the modify privilege.

    When you click Abort, the following dialog box appears:

    Select either **OK** or **Force** to abort the job, or **Cancel** if you change your mind.
  - Deleting a job on this page causes removal of job information from the ElectricFlow database, but information in the job's on-disk workspace area is not affected. You must delete workspace information manually.

Tips

If you use RSS, an active RSS icon is provided in Windows Explorer on this page for your convenience. If you use Firefox, click **Bookmarks** > Subscribe to this page to display the feed. You can then add the feed URL to a viewer of your choice.

Job Step Details

This page displays detailed information about a job step.

Links and actions at the top of the table

- **Access Control**—Use this link to set privileges for this job step. For more information, see the Access Control Help topic.
- "Star" icon—Click this icon to add this page to your Home for quick one-click access in the future.

Summary section (at the top of the page)

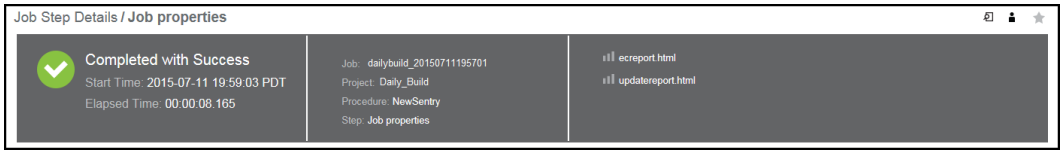
Notice in the screen example below that the name of the job step you are viewing is adjacent to the page title, Job Step Details.

The Summary section provides:

- The job's Start, Wait (waiting for a resource), and Elapsed times
- Wait times specified if your job steps were restricted by resource, workspace, a precondition, or license availability.
  For example, you will see the license wait time if a job step could not run because your license-allowed maximum concurrent step limit was met or exceeded, meaning no steps will run until the number of concurrent steps is reduced.
- A General Information section with links to get you to the job step's Job Details, Project Details, Procedure Details, or Edit Step pages.
- Any custom reports in the summary area to the right of General Information. Reports on this page are for command steps only.

Reports

To create a special-purpose HTML report for a job: In the top-level of any of the job's workspaces, add a file that contains the word "report" or "Report" and ends in the `.html` file extension. The link is automatically created when the Job Details page is displayed. The job's workspace must be accessible to the web server. The link will be in the Reports section, within the Summary section.



Links

This page can include a Links section also, which will appear on the far right-side of the Summary section.

To add a link to this section:

- **Create a link to any file:**
  To add a link, run a command in any job's job step using this format (this works for both local [disconnected] and non-local workspaces):

  ```
  ectool setProperty "/myJobStep/report-urls/<myReportName>"
  "/commander/jobSteps/<jobStepId>/<artifactDirectoryName>/<file-path>"
  ```

  Example command:

  ```
  ectool setProperty "/myJobStep/report-urls/Test Report" "/commander/jobSteps/
  $[/myJobStep/jobStepId]/testreport/index.html"
  ```

  > **Note:** If you are using this format (from ElectricCommander 4.2 and earlier):

  ```
  ectool setProperty "/myJob/report-urls/<myReportName>"
  "/commander/jobs/<jobId>/<workspaceName>/<artifactDirectoryName>]/<file-path>"
  ```

  Example command:

  ```
  ectool setProperty "/myJob/report-urls/Test Report"
  "/commander/jobs/$[/myJob/jobId]/
  $[/myJobStep/workspaceName]/testreport/index.html"
  ```

It will continue to work only for non-local workspaces. We recommend, however, that you change the command to use the newer format.

- **Create a link to any directory:**
  To add a link, run the following style command in any job's job step:

```
ectool setProperty "/myJobStep/report-urls/Main Job Workspace"
"file:////WinStor2/scratch/chron55build/$[/myJob/jobName]"
```

> **Note:** Links to a directory automatically work in Internet Explorer, but if using Firefox, local links are disabled unless the default security policy is modified or an extension is used. See http://kb.mozillazine.org/Links_to_local_pages_don%27t_work for details.

The "tabbed" Sections

The next section of tabs allows you to select the type of information you want to see. You may see six or fewer tabs, depending on the type of job step you are viewing.

Child Steps table

The Child Steps table is available for subprocedure and broadcast steps only.

- Click on a Step Name to go to that step's Job Step Details page.
- Click the Log "icon" to see that job step's log information.
- Status—This column provides the job's step status.
  - Aborted—The step or job was aborted.
  - Canceled—The step did not run because the job was aborted or a previous step was aborted.
  - Error—The command failed or `postp` detected an error in the output.
  - Running—The step is currently running or a step inside the subprocedure call is running.
  - Skipped—The step did not run because the run condition evaluated to "false".
  - Success—The step completed with no errors or warnings.
  - Waiting for Resource or Workspace—The step is waiting to run as soon as a resource or workspace is available.
  - Warning—`postp` detected a warning message.

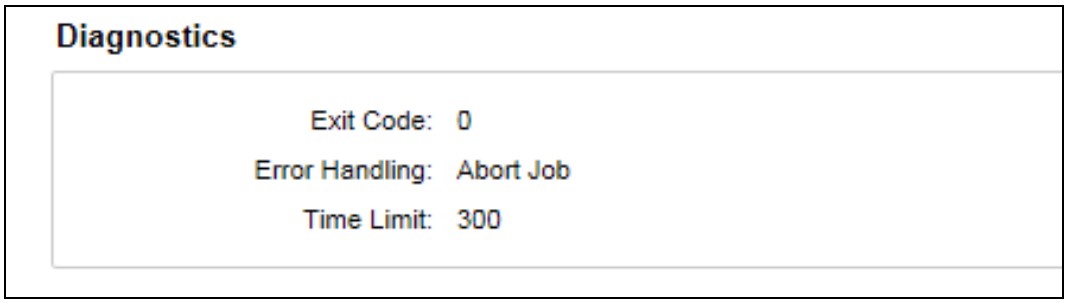
  The Status field can be over-ridden by `postp` to include more specific information. `postp` information may be linked directly to diagnostic information—also available by selecting the Diagnostics tab.
- Click on a Resource name to go to the Edit Resource page.
- Click the **Edit** link in the Action column to go to the Edit Step page.

General table

This section displays basic information including the step ID, step name, create and end times, and whether the step invoked a subprocedure or ran a command.

Diagnostics table

If the job step or child step (for subprocedure and broadcast steps) generated warning or error messages, you will see those messages here. This section contains one or more diagnostics, each of which is a portion of a step's log file that was identified as "interesting" by the step's postprocessor. Typically, each diagnostic relates to an error or warning detected by the postprocessor. All diagnostics for a particular job step appear together.

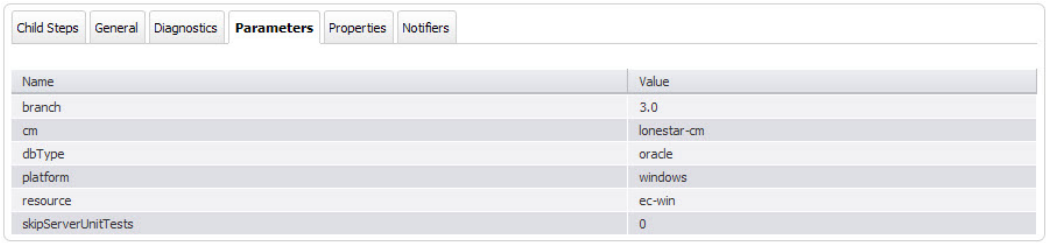


Parameters table

This section displays parameter values supplied to the top-level procedure when the job step was invoked and has these columns:

- Name
- Value

This table is available for steps with child steps only.



Properties table

This section displays information computed or used by the job step while it is running. After the job step completes, these properties are normally read-only.

- Click a Property Name to edit that property.
- If a "folder icon" precedes a property name, it denotes a Nested Property Sheet.
- This section also provides **Create Property**, **Create Nested Property**, and **Access Control** links to update or enter additional information for the next time this job step is run. For more information on properties, see the Properties Help topic.

Notifiers table

If you have designated certain persons or groups to be notified of details for specific job steps, this section displays who receives an email notification, the type of notification, any condition, and a description [if supplied]. If a notifier is available, you can click on its name to view details or make modifications.

Job Step Time and waitTime Properties Explained

- start—The time when this job step began executing.

  When the ElectricFlow server assigns a resource to run the step, it issues a `<runcommand>` request containing the expanded command body to that resource's agent. The "start" time is when that `<runcommand>` is sent to the agent.

- finish—The time when this job step completed.

  When a command finishes executing, the ElectricFlow agent issues a `<finishcommand>` request containing the exit code and other metadata back to the ElectricFlow server. The "finish" time is when the `<finishcommand>` is received by the server.

- elapsedTime—The number of milliseconds between the start and end times for the job.

  This is exactly "finish"—"start".

- totalWaitTime—The sum of resource, workspace, and license wait times for this job step.

  The sum of resourceWaitTime, licenseWaitTime, and workspaceWaitTime.

- waitTime—The number of milliseconds the step spent between runnable and running (for example, waiting for a resource).

  Covers known reasons why a step couldn't run as soon as it became runnable (same as totalWaitTime), plus any other possible reason (slow transaction commits, system paging, etc).

- resourceWaitTime—The length of time this job step stalled because it could not get a resource. This could indicate that eligible resources for the step have reached their step limit, are in-use but the step requires a resource exclusively, or resources are down.

  For subprocedure steps, this is the sum of all child steps.

- licenseWaitTime—The length of time this job step had to wait to process because the license limit was reached or exceeded.

  For subprocedure steps, this is the sum of all child steps.

- workspaceWaitTime—The time this job step had to wait because no workspace was found or available.

  For subprocedure steps, this is the sum of all child steps.

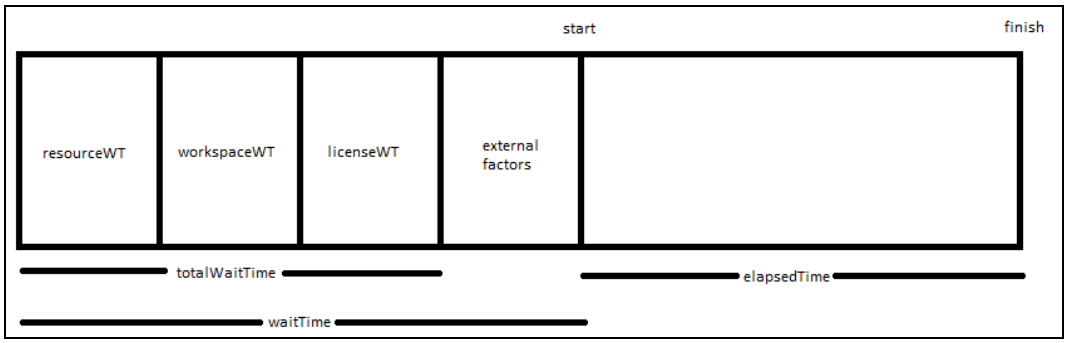
When it is time to schedule the step, ElectricFlow checks the license and starts counting time against licenseWaitTime until the license is satisfied (e.g., running this step would not exceed the maximum

number of licensed hosts). After the step is scheduled, ElectricFlow sends a runCommand message to the agent. If it fails because ElectricFlow cannot talk to the agent, ElectricFlow accrues time against resourceWaitTime. If ElectricFlow can talk to the agent, but it has trouble manipulating the workspace (for example, it cannot create the workspace dir, map the drive [on Windows], or create files), ElectricFlow accrues time against workspaceWaitTime.

It is possible for the following to occur:

1. A job step becomes stalls due to a license issue.
2. The license limit is no longer exceeded and ElectricFlow tries to run the step but hits a bad resource.
3. When the resource is good, the step stall because of a license again (adding more time to the licenseWaitTime),
4. When the license is satisfied, ElectricFlow reaches the resource, and everything runs.

This is why ElectricFlow continues cycling through these states until everything is satisfied.



The first 4 blocks (resourceWaitTime, workspaceWaitTime, licenseWaitTime, and external factors) can occur in any order. External factors could include things such as system paging, slow transaction commits, etc.

Licenses

This page displays all license information known to the ElectricFlow server. A single license is typically displayed. The license defines the usage to which your organization is entitled. For explanations of the various ElectricFlow license types, see License Types below.

License Types

Any combination of the following license types is available from Electric Cloud:

- Concurrent Resource License on page 1130
- Concurrent User License on page 1132
- Concurrent Step License on page 1133
- Registered Host License on page 1134
- Registered User License on page 1135
- Application License on page 1136
- Microservice License on page 1136

Concurrent Resource License

This license controls the number of unique host names on which steps can be scheduled at a time. Two properties, `maxHosts` and `maxProxiedHosts`, are defined in a license to control the maximum number of concurrent and proxied hosts that can be in use at a time. The `localhost` host name and the `127.0.0.1` IP address refer to the local agent installed on the ElectricFlow server machine, and these host names are not part of the total count for hosts in use.

If two or more steps are running concurrently on the same host, the host name still counts once only toward the license limit. Any number of resources can point to a host; from a licensing perspective, they all count as one host. When the host limit is met, any new job step that attempts to run on a host not already in use stays in the "runnable" state until its host is available.

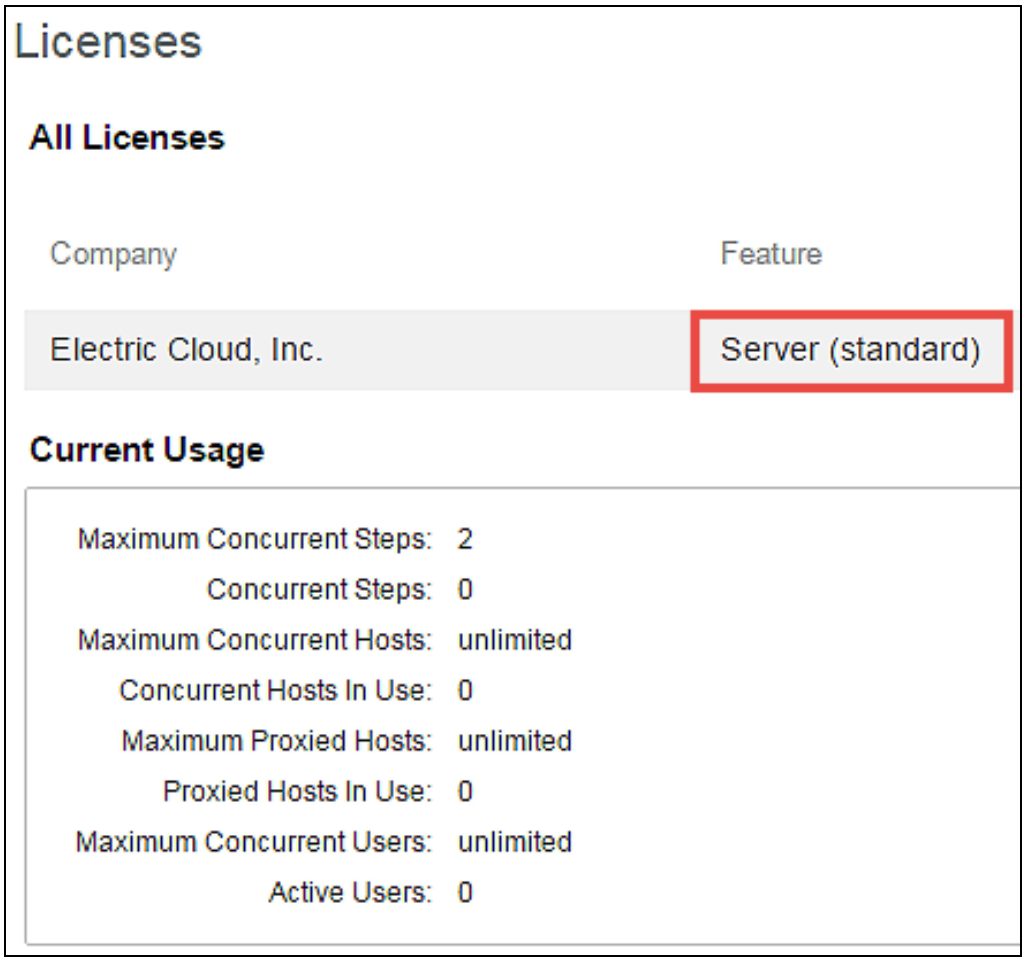
Current license usage statistics are at the top of the **Resources** page. If no license is available, jobs cannot be launched. When a license expires, running jobs are allowed to complete, but no new jobs can be launched.

A concurrent resource license can be a standard or enterprise license.

Standard License

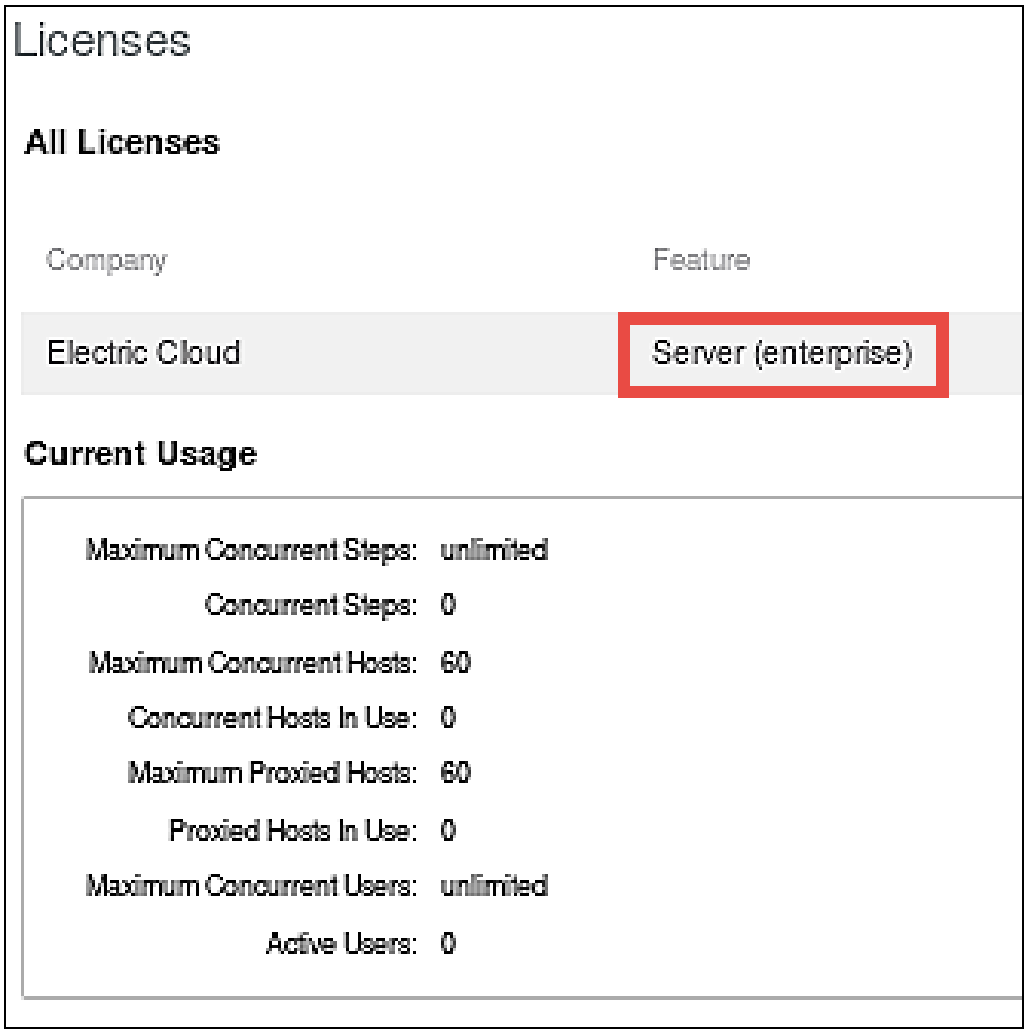
A standard license allows you to use only the built-in database. A standard license is packaged with the server and is automatically installed if no other license is present. A standard license restricts you to running only two concurrent steps.

This is an example of an ElectricFlow server with a standard license:



Enterprise License

An enterprise license allows you to use an external (alternate) database. You must apply an enterprise license key when you want to configure an alternate database. This is an example of an ElectricFlow server with an enterprise license:



Concurrent User License

If the `maxConcurrentUsers` property is set on the license, its value defines the maximum number of users who can actively use the ElectricFlow server. A unique license is defined by the combination of the user's name and the IP address from which the user is making requests. A user can have multiple active sessions (through the web UI or through ectool) from the same machine while only using a single license.

If a user makes a licensed API call without an active license, the server attempts to assign a license. If a license is available (meaning that the maximum is not yet met), the user is assigned a license automatically. If no license is available, the user sees a `NoAvailableLicenses` error and is redirected to the licenses page.

You can view currently-held licenses and their expiration dates by calling the `getLicenseUsage` API or by viewing the Active Users table on the Licenses web page. When a license becomes available, the first user to make a licensed API call is assigned that license. The only API calls that are not licensed are `login`, `getLicenseUsage`, `getAdminLicense`, `getServerStatus`, and `getServerInfo`.

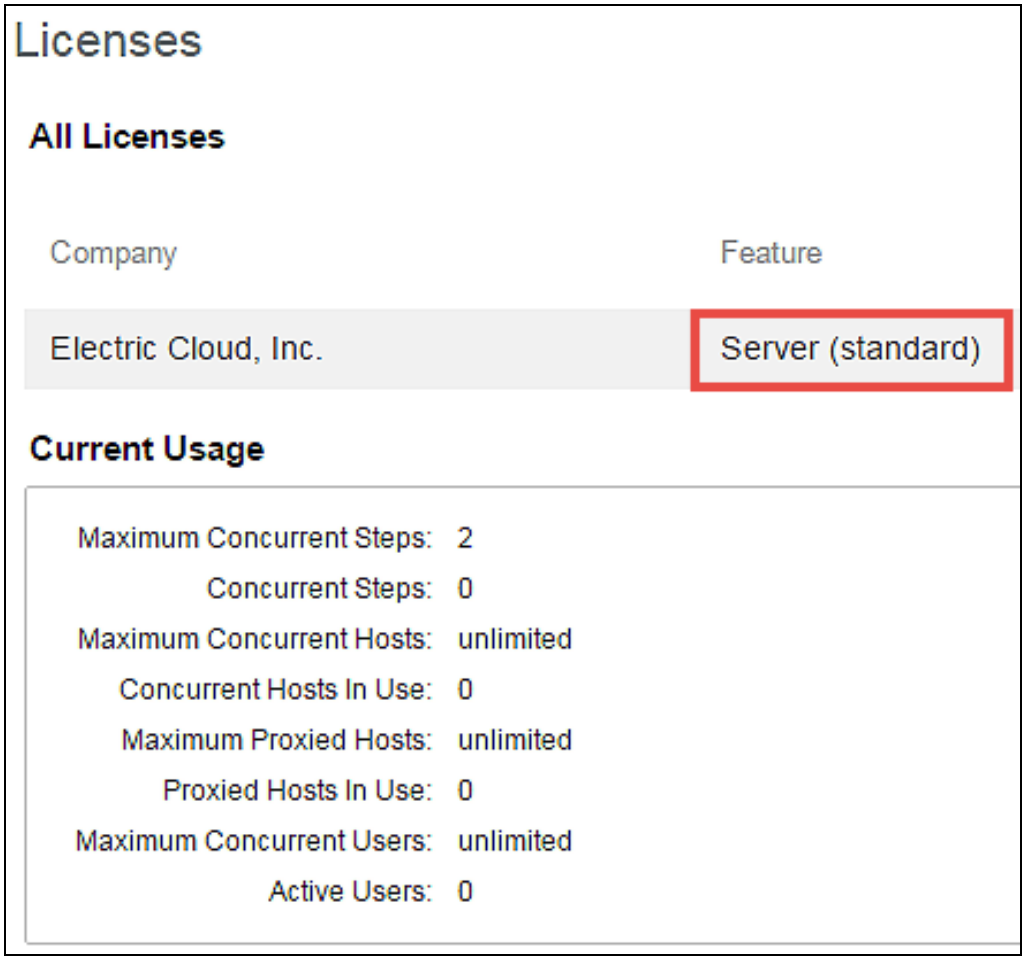
When a license is first acquired, it expires in one hour. After an hour, each time the user makes a licensed API call, the license is extended another 10 minutes (this is a configurable server setting).

When a user's license passes the expiration date, it is released. When the user makes another licensed API call, a license is assigned automatically if one is available. Otherwise, the `NoAvailableLicenses` error appears.

A single administrator license can be used in an emergency. All users with *execute* permission on the system-level licensing access control list can use the administrator license by calling the `getAdminLicense` API. This license has no expiration as long as all regular licenses are in use, but this license obeys the regular license expiration policy if regular licenses are available.

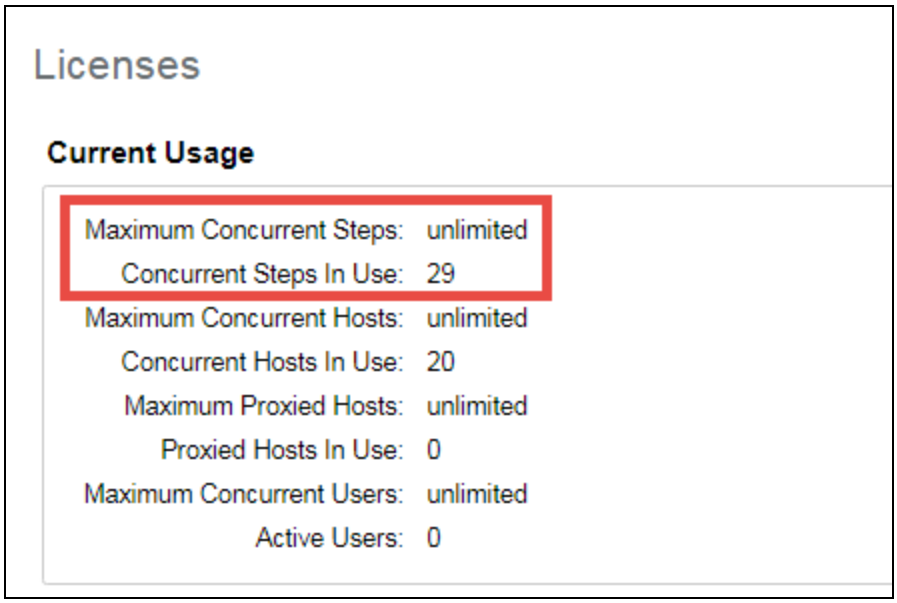
A concurrent user license can be a standard or enterprise license.

This is an example of an ElectricFlow server with a standard license:



Concurrent Step License

This license limits the number of unique steps that can run at a time in parallel. The `maxConcurrentSteps` license file property sets this limit.



Registered Host License

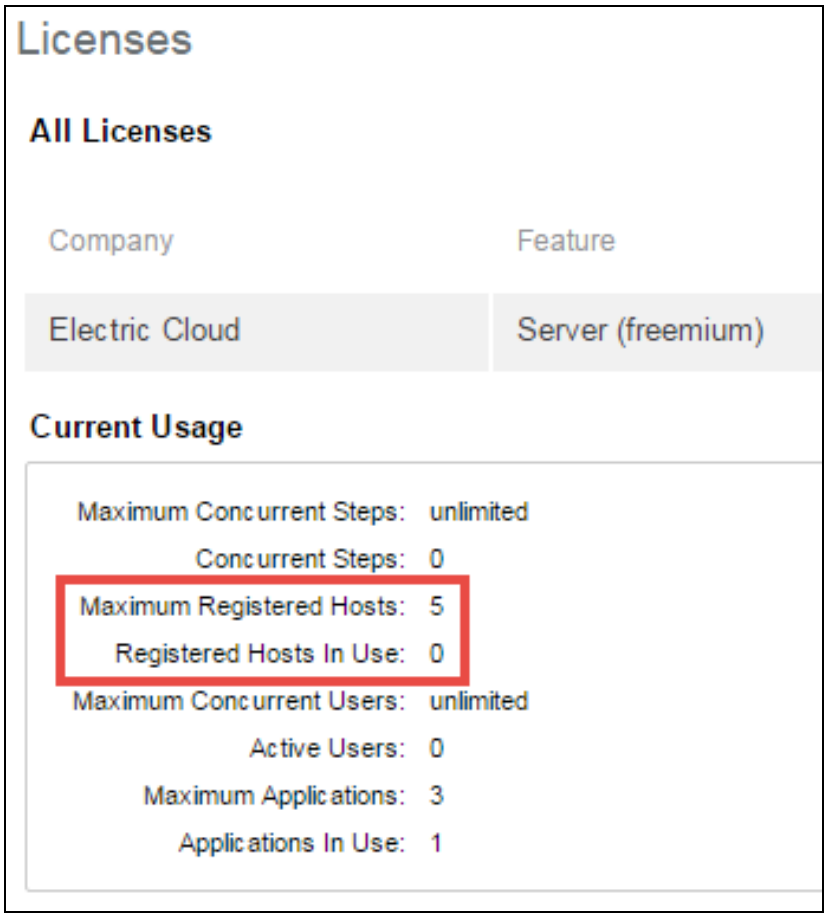
Registered hosts are dedicated hosts on which a procedure or step can always run. This license controls the number of unique registered host names. ElectricFlow can have a mixed license with concurrent and registered hosts. The `maxRegisteredHosts` license file property sets the maximum number of registered and proxied hosts that can be used at a time.

The `localhost` host name and the `127.0.0.1` IP address represent the local agent installed on the ElectricFlow server machine. These are part of the total count for hosts in use with a registered host license.

If multiple steps are running concurrently on the same host, the host name still counts once only toward the license limit. All resources pointing to a single host count as one host.

When a license expires, running jobs are allowed to complete, but no new jobs can be launched.

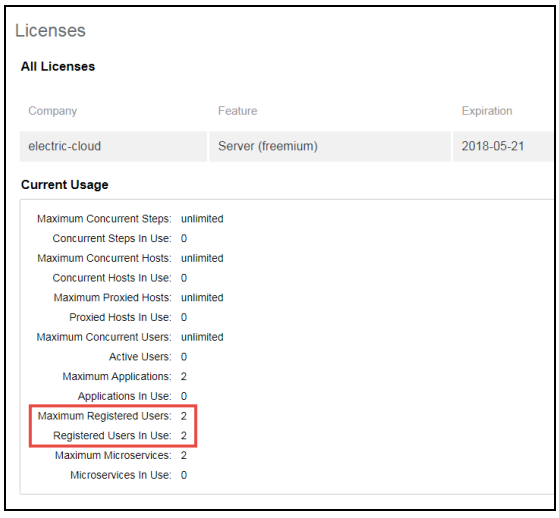
This is an example of an ElectricFlow server with a registered host license:



Registered User License

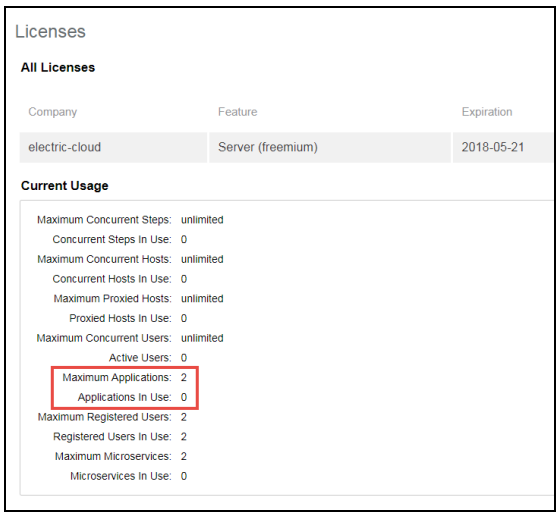
This license allows registered users to be created up to the license limit.

This is an example of an ElectricFlow server with a registered user license:



Application License

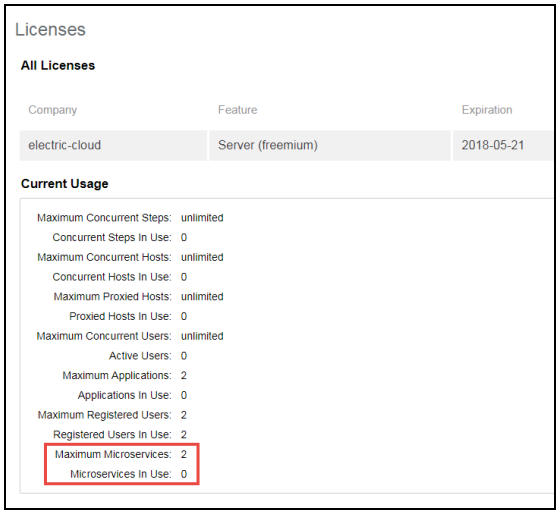
This license allows applications to be created up to the license limit. This is an example of an ElectricFlow server with an application license:



Microservice License

This license allows microservices to be created up to the license limit. This license applies to independent microservices as well as microservices contained in application.

This is an example of an ElectricFlow server with a microservice license:



All Licenses Section

The following information appears in the **All Licenses** section and applies to all types of licenses.

Column Descriptions

Column Name	Description
Company	Name of the company or organization to which the license was granted. If this is not your organization, contact Electric Cloud Customer Support.
Feature	Type of license. For example, `Server (enterprise)`.
Expiration	Last day when this license is valid. After your license expires (and after any grace period), running jobs will complete, but no new jobs can be launched.
Grace Period (Days)	Number of days that you may continue using ElectricFlow after the expiration date. (However, you are warned frequently during the grace period that your license has expired.)
Actions	• Click the  (View License) button to see additional details about this license. • Click the  (Delete) button to delete this license.

License Current Usage Section

Some or all of the following fields in the **Current Usage** section will appear depending on your license types.

Concurrent Resources, Users, and Steps

Concurrent resource licenses	
Field Name	**Description**
Maximum Concurrent Hosts	Maximum number of unique host names allowed to run ElectricFlow at a time.
Maximum Concurrent Hosts In Use	Maximum number of unique host names defined in concurrent (non-proxy) resources where steps can run at any time.

Concurrent resource licenses	
Field Name	**Description**
Concurrent Hosts in Use	Number of unique host names defined in concurrent resources where steps are running.
Maximum Proxied Hosts	Maximum number of unique host names defined in proxied resources.
Maximum Proxied Hosts In Use	Maximum number of unique host names defined in proxied resources where steps can run at any time.
Proxied Hosts in Use	Number of unique host names defined in proxied resources where steps are running.
Concurrent user licenses	
Field Name	**Description**
Maximum Concurrent Users	Number of users allowed to use ElectricFlow at a time.
Active Users	Number of users with an active license.
User IP	The <user>@<IP> combination that holds an active license.
Expiration	Date and time when the license will be released.
Last Use	Date and time when the license was last used.
Concurrent step licenses	
Field Name	**Description**
Maximum Concurrent Steps	Maximum number of allowed concurrent steps.
Concurrent Steps In Use	Number of running concurrent steps.

Registered Hosts and Users

Registered host licenses	
Field Name	**Description**

Maximum Registered Hosts	Number of registered hosts on which a procedure or step can always run. This appears when the license on the ElectricFlow server is based on a combination of concurrent and registered hosts or only registered hosts.
Registered Hosts In Use	Number of registered hosts where steps are running.
Registered user licenses	
Field Name	**Description**
Maximum Registered Users	Maximum number of registered users who can use ElectricFlow at a time.
Registered Users In Use	Number of registered users currently using ElectricFlow.

Applications and Microservices

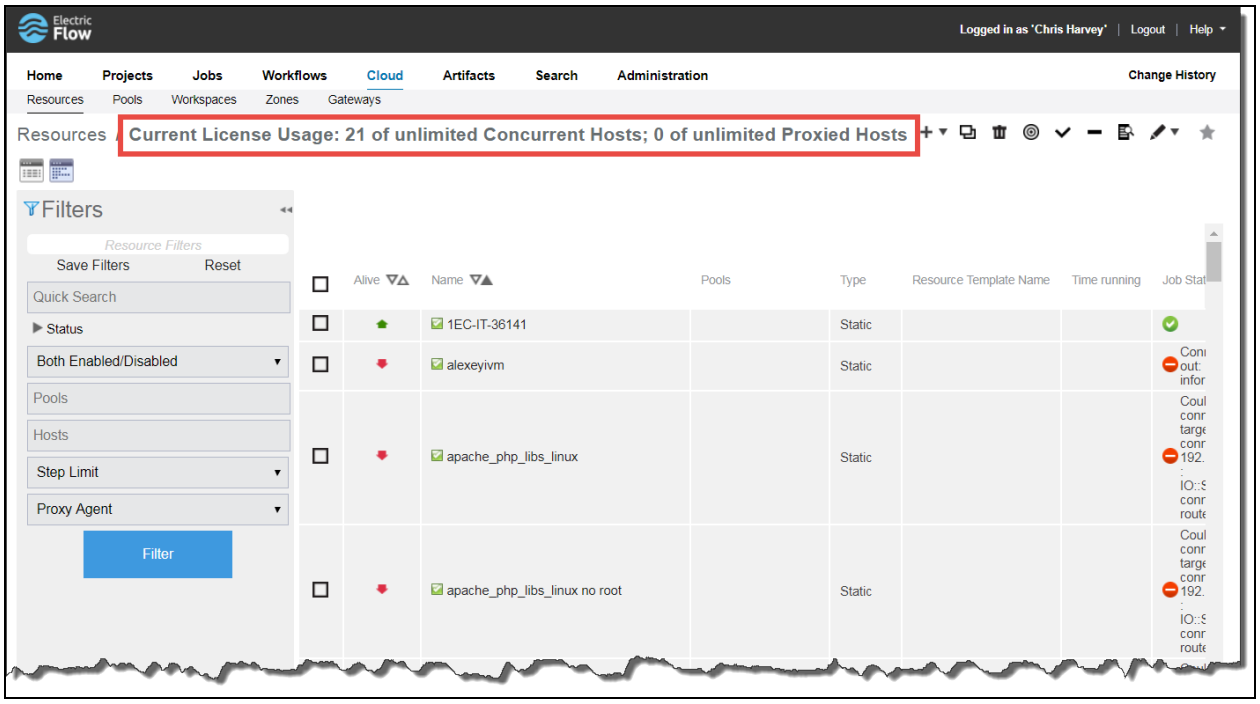
Application licenses	
Field Name	**Description**
Maximum Applications	Maximum number of applications that ElectricFlow can deploy at a time.
Applications In Use	Number of running applications.
Microservice licenses	
Field Name	**Description**
Maximum Microservices	Maximum number of microservices that ElectricFlow can deploy at a time.
Microservices In Use	Number of running microservices.

Importing a License

To import a new license file, click **Import License**.

Viewing License Usage Statistics

License usage statistics are at the top of the **Cloud** > **Resources** page in the Automation Platform web UI. For example:



View License

This page displays the details for a specific license.

License Details Section

Name	Description
Company Name	Name of the company or organization to which the license is granted. If this is not your company, contact Electric Cloud Customer Support.
Product Name	Product to which this license applies (if it is not "ElectricFlow," it does not apply here).
Feature Name	Product feature to which this license applies. This must be "Server" for ElectricFlow licenses.
Maximum Concurrent Hosts	Maximum number of unique host names allowed to use ElectricFlow at a time.
Maximum Proxied Hosts	Maximum number of unique host names defined in proxied resources.
Maximum Concurrent Users	Number of users allowed to use ElectricFlow at a time.
Maximum Concurrent Steps	Maximum number of concurrent steps allowed by this license.

Name	Description
Maximum Registered Hosts	Number of registered hosts on which a procedure or step can always run. This appears when the license on the ElectricFlow server is based on a combination of concurrent and registered hosts or registered hosts only.
Maximum Registered Users	Maximum number of registered users that can use ElectricFlow at a time.
Maximum Applications	Maximum number of applications that ElectricFlow can deploy at a time.
Maximum Microservices	Maximum number of microservices that ElectricFlow can deploy at a time.
Expiration Date	Date when this license becomes invalid. After this date (and after any grace period), running jobs will complete, but no new jobs can be launched.
Grace Period	Number of days that you may continue using ElectricFlow after the expiration date. However, you will be warned frequently during this period that your license has expired.

Deleting a License

To remove this license, click the (**Delete**) button.

Import License

Use this page to provide a new license for ElectricFlow. You should have received a license (as a block of text) from Electric Cloud. If you have not received a license, contact Electric Cloud Customer Support or your sales representative.

- Use a text editor to open the license file.
- Copy the text into the **Contents** field.
- Click **OK**.

You will see your license information on the **Licenses** page.

Plugin Manager

A plugin is a collection of one or more features that can be added to ElectricFlow. A plugin:

- Can provide one or more new pages for the UI and can also provide a configuration page so you can provide additional information needed to implement the plugin.
- Has an associated project that can contain procedures and properties required by the implementation.

- Is delivered as a JAR file containing the features implementation. When a plugin is installed, the ElectricFlow server extracts the JAR contents to disk into a configurable plugins directory.

The ElectricFlow installation includes numerous bundled plugins. These plugins are created and supported by Electric Cloud. You can also create your own plugins as described below.

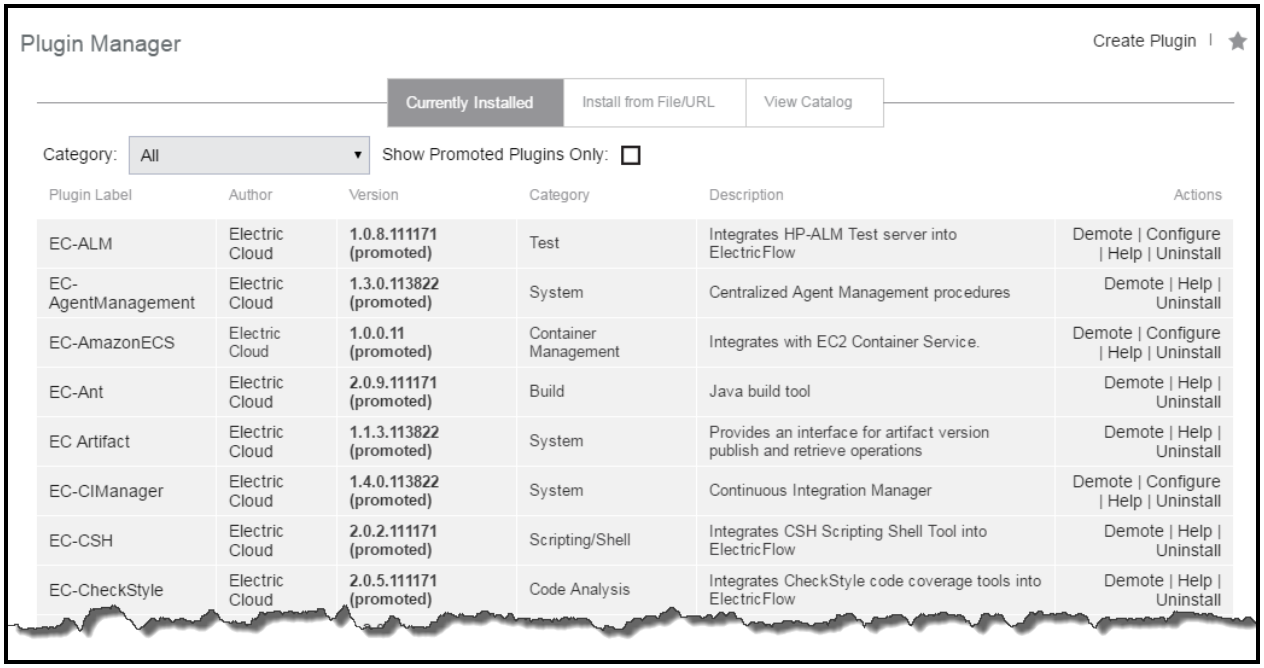
Navigating the Plugin Manager User Interface

You can use the Plugin Manager to:

- See a list of installed plugins and whether an installed plugin is promoted.
- Perform a variety of operations on plugins in the list, such as demoting, configuring, and uninstalling them.
- Install plugins from the Plugin Catalog.
- Create plugins, modify those plugins, and export those plugins.

Descriptions of Tabs and Filtering Options for the Installed Plugins List

Following is the Plugin Manager main page. It provides a list of installed plugins:



The **Plugin Manager** main page displays the following tabs and the following options for filtering the list of plugins.

Installation or upgrade task	Description
Currently Installed	Displays the installed plugins. Plugins that you create or install from other sources also appear on this page.

Installation or upgrade task	Description
Install from File/URL	Displays the screen for plugin installation options: • **File Install**—Upload and install a local JAR file from the machine running your web browser. Click **Upload** after specifying the path that you need. • **URL Install**—Install a plugin from a location specified by an URL. This location can either be an external web server (using `http://`) or a file on the ElectricFlow server host (using `file://`). You can also use ectool to install a plugin from the command line. ectool contains a full set of commands to perform plugin tasks. For more information on available ectool plugin commands, see the "API Commands—Plugin Management" topic in the *ElectricFlow API Guide* at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html. **Note:** The server has a configurable maximum file upload size that defaults to 50 MB. Use the server **Maximum upload size** setting to adjust the default.
View Catalog	Displays all plugins available for installation. The **View Catalog** tab is not present if the ElectricFlow server lacks internet access (such as if it is locked down or has a connectivity issue.) If so, an error message appears in the Plugin Manager (such as on the main page or the **New Plugin** page): `Catalog not found : Catalog URL is http://plugins.electric-cloud.com/catalog/catalog.xml`. **Catalog Install**—Click the **Install** action (next to a plugin listed in the catalog) to download and install the plugin.
Category	A drop-down menu to filter the plugin list by a specific category.

Descriptions of the Columns in the Installed Plugins List

Columns in the list of installed plugins provide a variety of details about each plugin.

Column name	Description
Plugin Label	Name of the plugin. Click the label of an installed plugin to see the **Project Details** page for that plugin.
Author	Person or organization who created the plugin.
Version	Plugin version. Promoted plugin versions also appear in this column. The promoted version number is labeled `(promoted)` immediately after the version number.

Column name	Description
Category	Plugin type. Plugins that provide similar types of functionality are grouped into categories.
Description	Short text string that describes the plugin purpose or function.
Actions	List of plugin tasks. Not all plugins have the same list of tasks. For example, some plugins might perform their own setup during installation and thus do not need more configuration information entered here. • **Configure**—Opens the configuration page for the plugin. • **Demote**—Makes the plugin inactive and removes any tabs associated with it. If you reinstall a previously demoted plugin, previous values from the demoted version are not copied to the new version that you install. • **Help**—Opens the plugin documentation if it exists. • **Install**—Downloads and installs a plugin from the plugins catalog site. If this option is not available, you might not have the access control permissions to install a plugin. • **Promote**—Upgrades the plugin to a new version. Old plugin values are copied to the promoted version. • **Re-install**—Downloads and re-installs a previously installed plugin. • **Uninstall**—Removes the plugin from your system. If the action that you need is not in the **Actions** column, you might not have sufficient modify permission in the plugins ACL for that action.

Creating, Modifying, or Exporting Plugins by Using the Plugin Manager

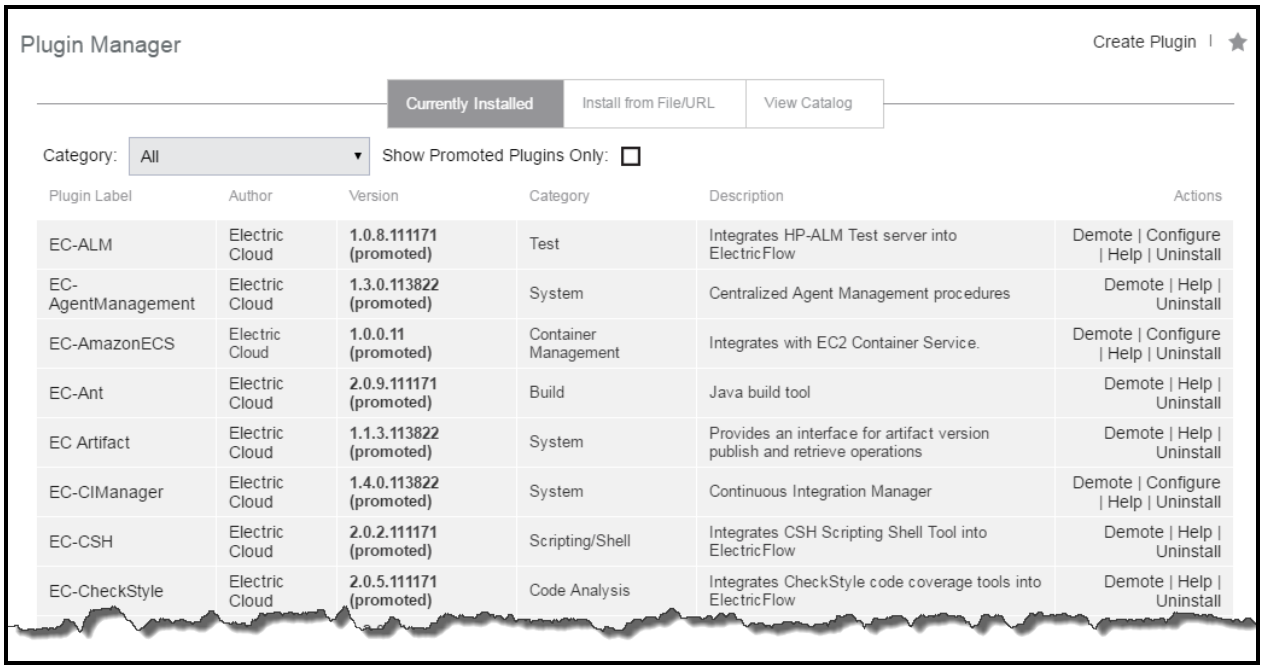
Creating a Plugin

To create a plugin, you set up the plugin configuration and then add procedures and properties.

Setting Up the Plugin Configuration

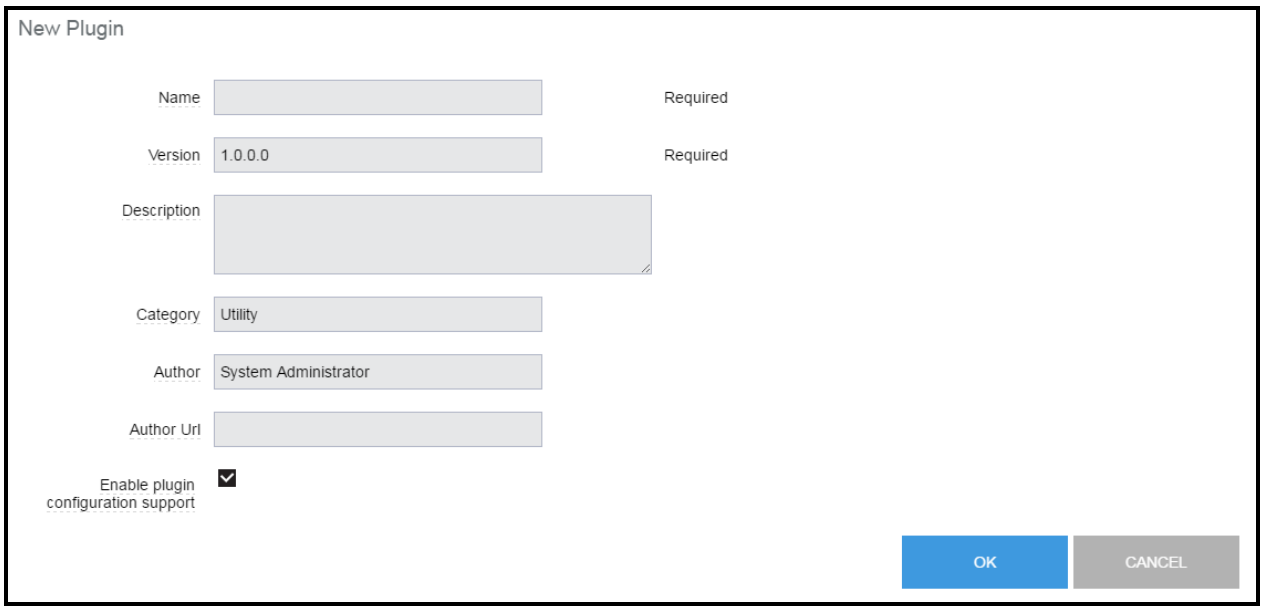
- In the Automation Platform, choose **Administration** > **Plugins**.

  The **Plugin Manager** page appears. For example:



- Click **Create Plugin**.

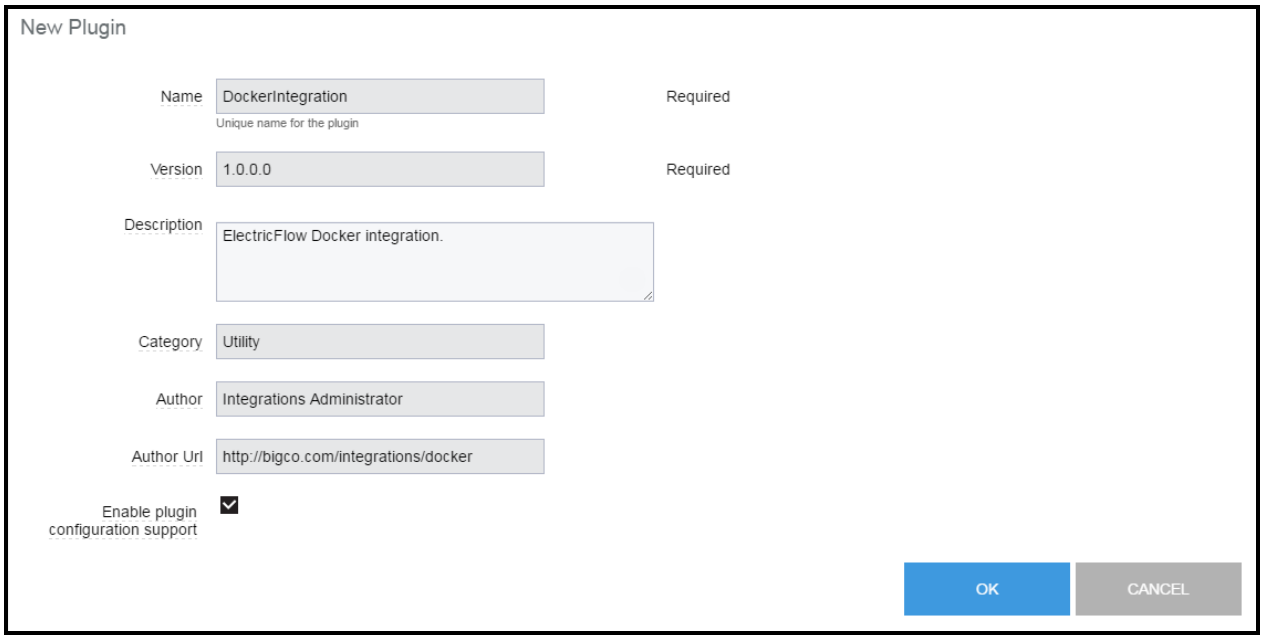
  The **New Plugin** page appears:



- Complete the page as follows:

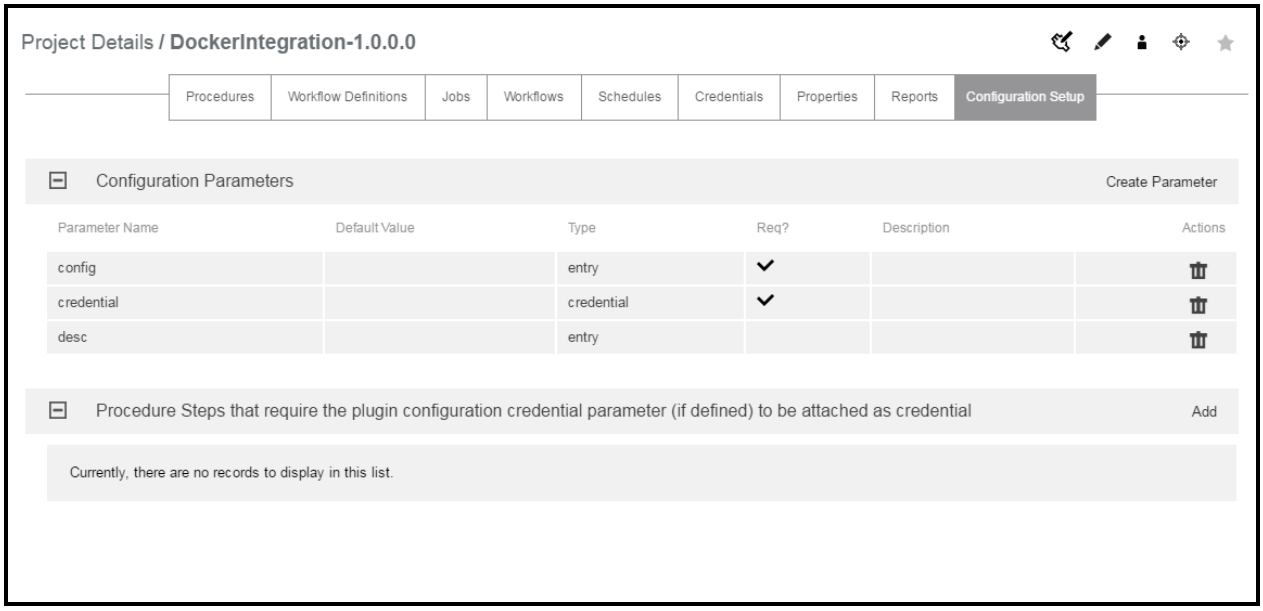
Field name	Description
Name	Plugin name. Enter a unique name.
Version	Plugin version. The default is `1.0.0.0`.
Description	(Optional) Plain text description of the plugin. ElectricFlow does not interpret this text.
Category	(Optional) Category for the plugin. Categories let you group plugins by functionality or purpose and provide a convenient way to look up a specific plugin if you know its functionality or purpose. For example, when you are configuring a process step and want to deploy an application on WebSphere, you first choose the Application Server category and then the WebSphere plugin. You can select an existing plugin category by using the autofill functionality or enter a new category. The default category is `Utility`.
Author	(Optional) Plugin author's name. For example, a user or organization name. The default is the current user name (such as `System Administrator`).
Author Url	(Optional) Plugin author's URL. For example, `http://bigco.com/integrations/docker`.
Enable plugin configuration support	(Optional) Determines whether to allow the addition of plugin procedures to support the creation of plugin configurations. Disabling this option displays a confirmation dialog box if the plugin already exists: `Any existing plugin configurations for this plugin will be deleted if the 'Enable plugin configuration support' is unchecked.` If you enable this feature, the **Configuration Setup** tab is added to the **Project Details** page for the plugin, and the CreateConfiguration and DeleteConfiguration procedures are added to the plugin (you can click the **Procedures** tab to see these procedures). This tab lets you modify information required to support plugin configurations such as the parameters required by the CreateConfiguration procedure. This option is enabled by default (meaning that the check box is checked).

For example:


New Plugin
Name
DockerIntegration
Unique name for the plugin
Required
Version
1.0.0.0
Required
Description
ElectricFlow Docker integration.
Category
Utility
Author
Integrations Administrator
Author Url
http://bigco.com/integrations/docker
Enable plugin configuration support
OK
CANCEL

- Click **OK**.

  If you selected **Enable plugin configuration support** in the **New Plugin** page above, the **Configuration Setup** tab in the **Project Details** page appears. (A plugin is a type of project.) For example:

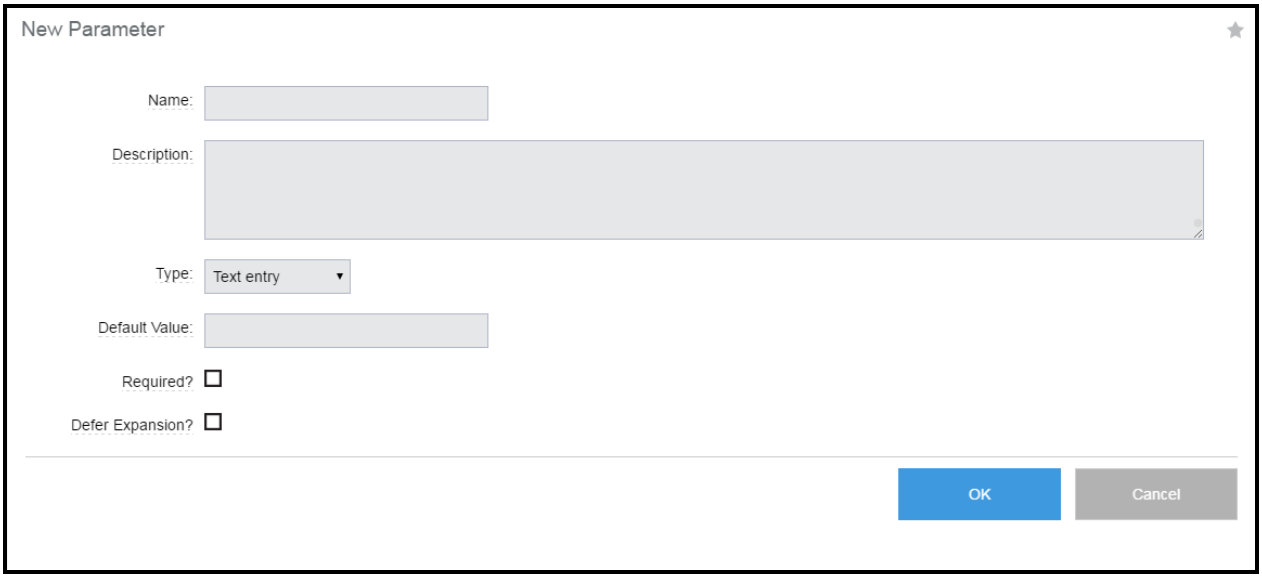


In the **Configuration Setup** tab, the plugin configuration parameters appear under the **Configuration Parameters** section. The CreateConfiguration procedure uses these parameters when creating the plugin configuration. By default, the following configurations parameters are added to a new plugin:

- config—Used to provide a unique name for the configuration
- desc—Description of the plugin configuration
- credential—User name and password used to communicate with the third-party system with which the plugin is integrated

- In the **Configuration Setup** tab, click **Create Parameter** to add configuration parameters that your plugin requires.

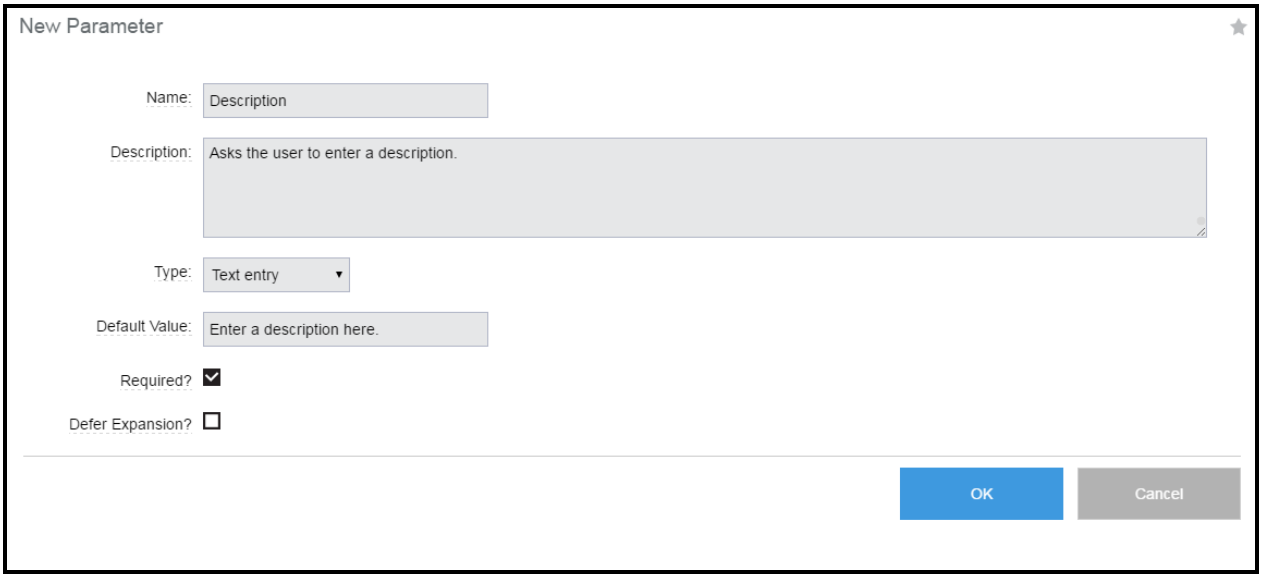
  The **New Parameter** page appears. For example:



- Complete the page as follows:

Field name	Description
Name	Name of the parameter for which values are requested when the CreateConfiguration procedure is invoked. Enter a unique name.
Description	(Optional) Plain text description of the parameter. ElectricFlow does not interpret this text.
Default Value	Default value of the parameter if no value is provided when the procedure is invoked.
Required?	Determines whether a value for the parameter is required. If this is enabled, the CreateConfiguration procedure ignores the default value and does not run unless a value is provided. After the procedure begins execution, the parameter value is available in the job property sheet with the same name as the parameter. The check box is unchecked by default (meaning that the parameter is optional).
Defer Expansion?	When the step executes (the check box is checked). The check box is unchecked by default (meaning that expansion is not deferred).

For example:



The configuration parameters that you specified are added as parameters to the **Create Configuration** procedure in the **Procedures** tab for the plugin.

- From the **Type** pull-down menu, choose a form element type and complete the page for that type as follows:

Type	Description
Text entry	Creates a field for users to enter a small amount of text.
Text area	Creates a field that expands for users to enter a large amount of text (such as a script).

Dropdown menu	Creates a drop-down menu for users to select a value when the parameter is presented. Choose one of the following:
	• **Enter options**—Click **+ Add Option** to add one or more new rows. Then enter the text and value for each option. The text is what will appear in the menu, and the value is the parameter value if that option is selected.
	• **Load options from list**—Enter a pipe-separated list of options (for example, `foo|bar|baz`). The text and value for the options will be the same.
	• **Load options from property sheet**—Enter the path to the property sheet that contains options for the parameter. The property sheet must be created in a specific format:
	  • An `optionCount` property must exist whose value is the number of options.
	  • For each option, create a nested property sheet named `option`*N*, where *N* is the option number, starting with `1`. For example, if `optionCount` is set to `3`, you must create three nested sheets: `option1`, `option2`, and `option3`.
	  • In each nested sheet, create two properties: `text` and `value`. The value of the `text` property will appear in the menu. The value of the `value` property is the parameter value if that option is selected.

Type	Description

Radio selector	Creates radio buttons for users to select a value when the parameter is presented.
	• **Enter options**—Click **+ Add Option** to add one or more new rows. Then enter the text and value for each option. The text is what will appear in the menu, and the value is the parameter value if that option is selected.
	• **Load options from list**—Enter a pipe-separated list of options (for example, `foo|bar|baz`). The text and value for the options will be the same.
	• **Load options from property sheet**—Enter the path to the property sheet that contains options for the parameter. The property sheet must be created in a specific format:
	  • An `optionCount` property must exist whose value is the number of options.
	  • For each option, create a nested property sheet named `option`*N*, where *N* is the option number, starting with `1`. For example, if `optionCount` is set to `3`, you must create three nested sheets: `option1`, `option2`, and `option3`.
	  • In each nested sheet, create two properties: `text` and `value`. The value of the `text` property will appear in the menu. The value of the `value` property is the parameter value if that option is selected.

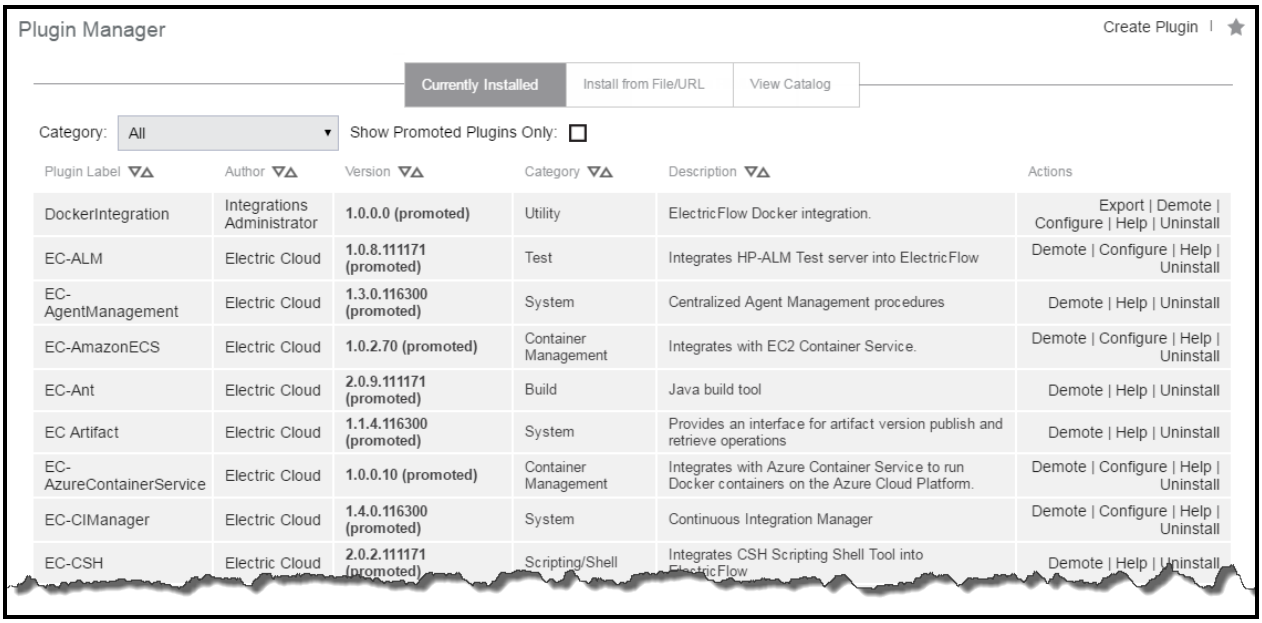
Checkbox	Creates a check box for users to select or deselect a value when the parameter is presented. • **Value when unchecked**—Value of the parameter when the check box is unchecked (`false` by default) • **Value when checked**—Value of the check box when the check box is checked (`true` by default) • **Initially checked?**- Whether the check box should be checked initially. If `true`, set **Default value** to match **Value when checked**. If `false` (the default) set **Default value** to match **Value when unchecked**.
Credential	Creates a text box and an input field for users to provide a user name and password respectively to use this parameter at runtime.

- Click **OK**.

  The new parameter appears in the **Project Details** page under the **Configuration Parameters** list. For example:

- Repeat the previous three steps to create more parameters as needed.

The new plugin appears on the **Plugin Manager** page under the **Currently Installed** tab along with the other installed plugins. For example:



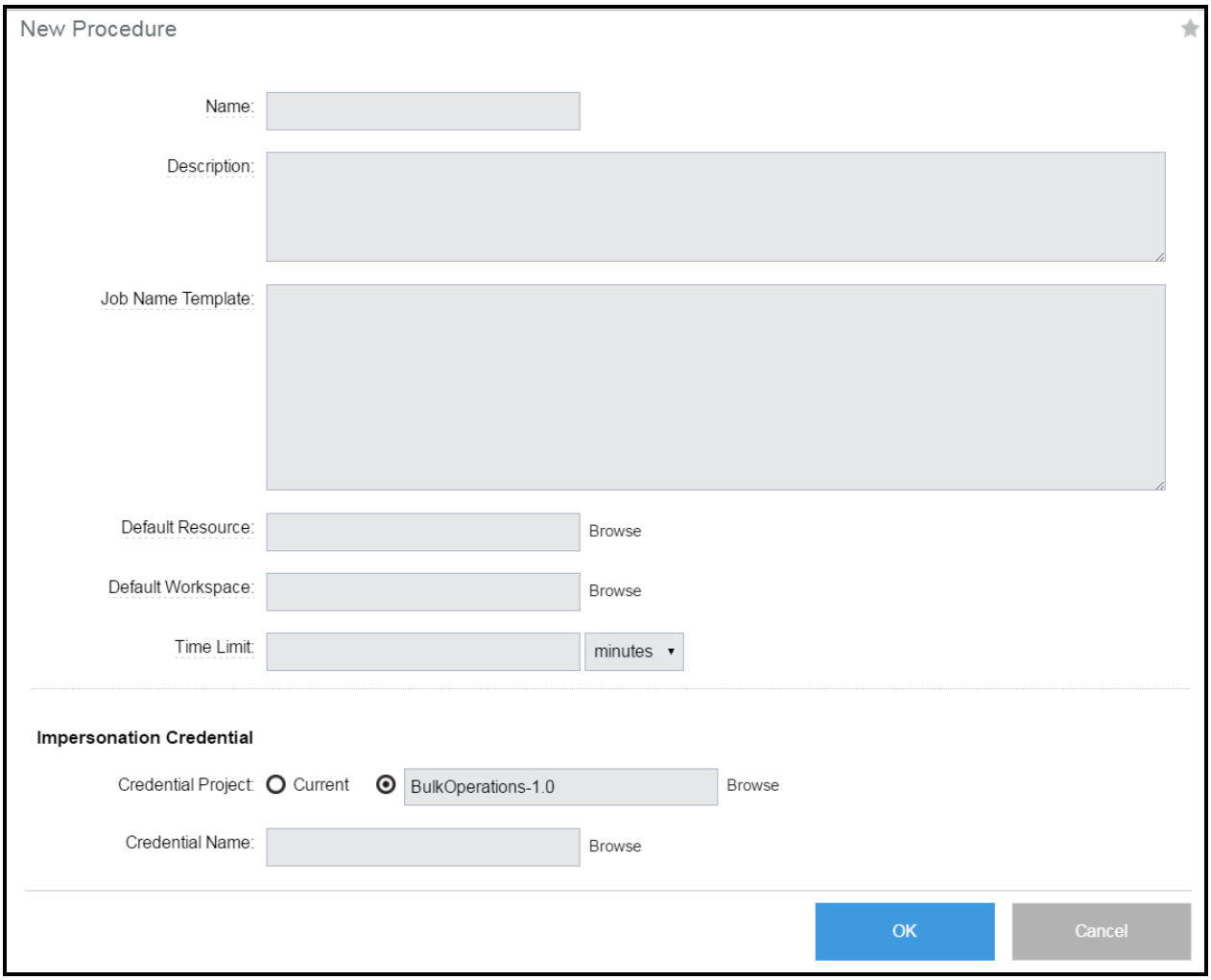
Adding Procedures

A procedure defines a process to automate one or more steps. A procedure is run to carry out a process.

- In the **Plugin Manager** page, click the plugin name.

  The **Project Details** page for the plugin appears.
- Click the **Procedures** tab and then click **Create Procedure**.

  The **New Procedure** page appears:


New Procedure
Name:
Description:
Job Name Template:
Default Resource:
Browse
Default Workspace:
Browse
Time Limit:
minutes
Impersonation Credential
Credential Project:
Current
BulkOperations-1.0
Browse
Credential Name:
Browse
OK
Cancel

- Complete the page as follows:

Field name	Description
Name	Procedure name. This name must be unique within the project.
Description	(Optional) Plain text description of the procedure. ElectricFlow does not interpret this text.

Job Name Template	Template used to determine the default name of a job launched from the procedure. For example:
	`$[projectName]_$[/increment /myproject/jobCounter]_ $[/timestamp]`
	produces a name such as:
	`projectFoo_1234_20140102130321`
	Enter any combination of elements to create meaningful procedure names. For example, you could include the build number.
	Note: You should not use the ElectricFlow-generated `jobId`, because it is not a human-readable integer. This means that it does not provide identifiable information and cannot be used as a counter.

Default Resource	Resource name. Enter a name or click **Browse** and choose a resource. This name must be unique among resource names.
Default Workspace	Name of the default workspace. Enter a name or click **Browse** and choose a workspace.
Time Limit	Maximum amount of time during which the step can execute. The step is aborted if it exceeds this time. If no time limit was specified on the calling step, time limits are copied to the calling step from the procedure. If the procedure is called from `runProcedure` (or a schedule), the time limit acts as a global job timeout. The timer for the procedure starts when the calling step or job becomes runnable (when all preconditions are satisfied). Specify a number and then use the pull-down menu to select the units (seconds, minutes, or hours).
Impersonation Credential	
Credential Project	Name of the project that contains this credential. Use the current project or click the other radio button and enter a project name (or click **Browse** and choose another project). The current project is used by default.

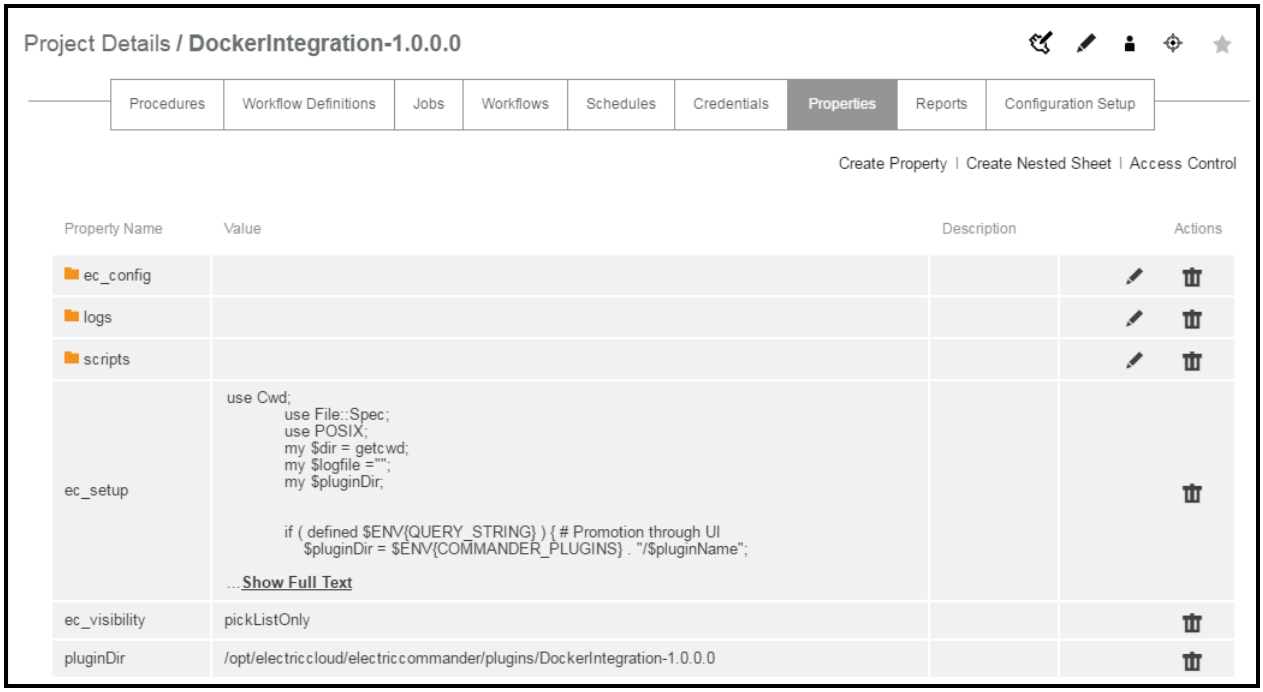
Field name	Description
Credential Name	Name of the credential for the project. Enter a name or click **Browse** and choose another credential.

- Click **OK**.
- Repeat the above steps to add more procedures if needed.

Adding Properties

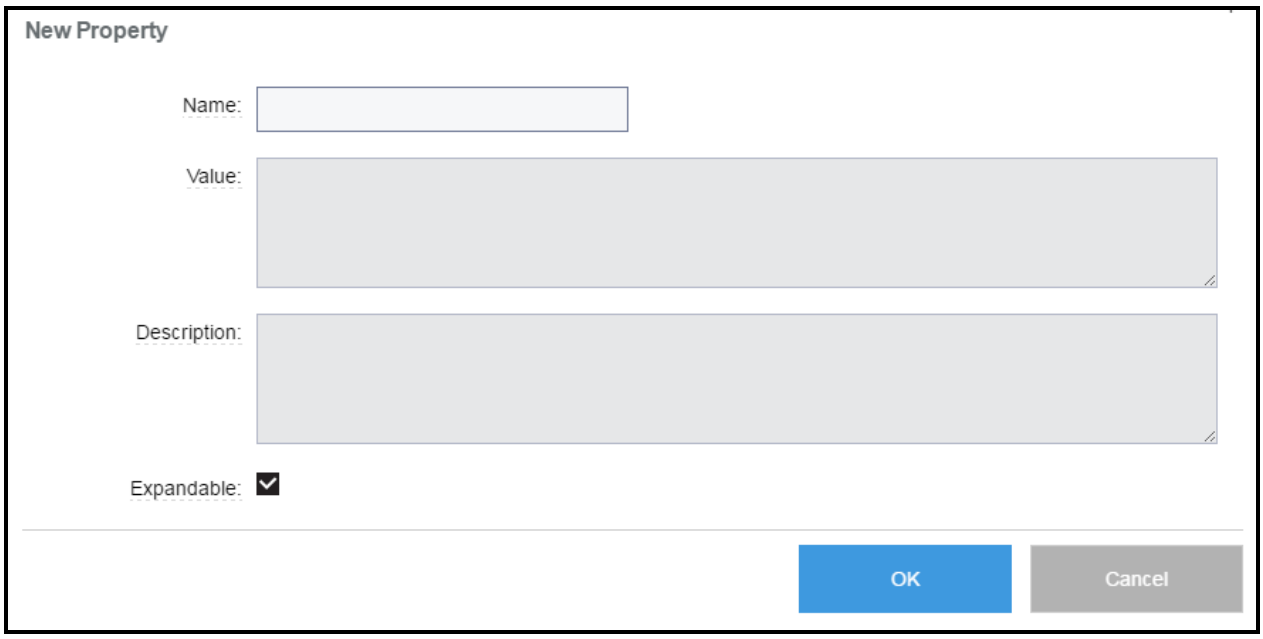
1. In the **Project Details** page for the plugin, click the **Properties** tab.

   The list of properties for the plugin appears. For example:



2. Click **Create Property**.

   The **New Property** popup appears:



3. Complete the popup as follows:

Column Name	Description / Actions
Name	Property name.
Value	Value to assign to the property.
Description	(Optional) Plain text description of this object. ElectricFlow does not interpret this text.
Expandable	Determines whether the property is marked for expansion. Enabled (checked) means that a property value such as `$[MYPROPERTY]` will be expanded or resolved to be the actual value of `$[MYPROPERTY]`. Disabled (unchecked) means that the value of the property is interpreted literally as `$[MYPROPERTY]`.

4. Click **OK**.
5. Repeat the above steps to add more properties if needed.

Editing a Plugin

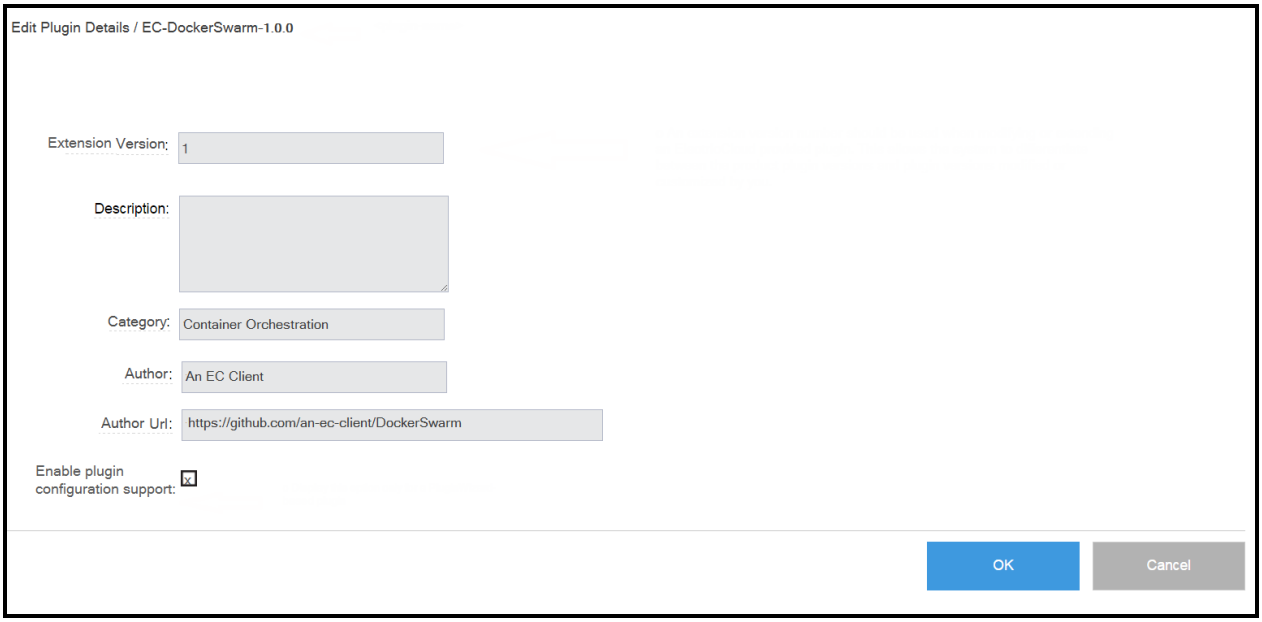
You can use the Plugin Manager to edit plugins that you have created in the Plugin Manager. (You cannot use it to edit plugins that are bundled with ElectricFlow or downloaded from the Plugins Catalog.)

- In the Automation Platform, choose **Administration** > **Plugins**.

  The **Plugin Manager** page appears.
- Click the name of the plugin that you want to edit.

  The **Project Details** page for the plugin appears.
- Create or update the procedures, properties, and so on as needed for the plugin project.
- Click the  (edit plugin details) button.

  The **Edit Plugin Details** page for the plugin appears. For example:



- Complete the page as follows:

Name	Description
Extension Version	(Optional) Number that ElectricFlow uses to differentiate between the Electric Cloud created plugins that you downloaded but did not modify or extend and Electric Cloud created plugins that you have modified or extended. This prevents an ElectricFlow upgrade that also upgrades the corresponding bundled plugin from overwriting the plugin that you have modified or extended. Use only positive integers with no other characters. For example, `12`. The default is `1`. For details about version extensions, see How Plugin and Extension Versions Determine Plugin Installation or Autopromotion on page 1162.
Description	(Optional) Plain text description of the plugin. ElectricFlow does not interpret this text.
Category	(Optional) Category for the plugin. Categories let you group plugins by functionality or purpose and provide a convenient way to look up a specific plugin if you know its functionality or purpose. For example, when you are configuring a process step and want to deploy an application on WebSphere, you first choose the Application Server category and then the WebSphere plugin. You can select an existing plugin category by using the autofill functionality, or you can enter a new category. The default category is `Utility`.
Author	(Optional) Plugin author's name. For example, a user or organization name.
Author Url	(Optional) Plugin author's URL. For example, `http://bigco.com/integrations/docker`.

Name	Description
Enable plugin configuration support	(Optional) Determines whether to allow the addition of plugin procedures to support the creation of plugin configurations. Disabling this option displays a confirmation dialog box: `Any existing plugin configurations for this plugin will be deleted if the 'Enable plugin configuration support' is unchecked.` If you enable this feature, the **Configuration Setup** tab is added to the **Project Details** page for the plugin, and the CreateConfiguration and DeleteConfiguration procedures are added to the plugin (you can click the **Procedures** tab to see these procedures). This tab lets you modify information required to support plugin configurations such as the parameters required by the CreateConfiguration procedure.

- Click **OK**.

Exporting a Plugin

You can export a plugin in order to import it to another ElectricFlow server. Only procedures and properties are exported. Workflows, schedules, credentials and so on are not exported.

- In the Automation Platform, choose **Administration** > **Plugins**.

  The **Plugin Manager** page appears.

- Click the **Export** link for the plugin that you want to export.

  The plugin .zip file is downloaded to your system as *<plugin-name>-<version>*.zip.

How Plugin and Extension Versions Determine Plugin Installation or Autopromotion

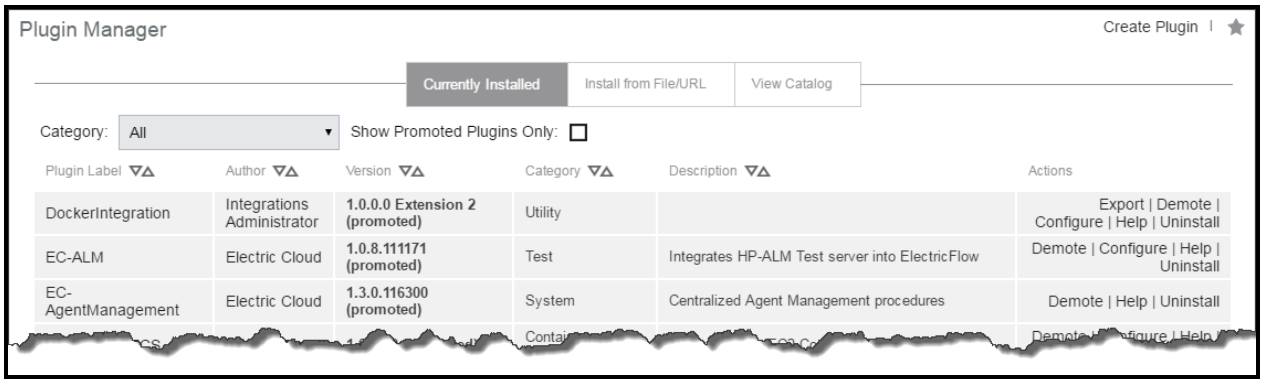
The Plugin Manager uses a combination of the plugin version and plugin extension version to determine whether to install or autopromote a new plugin version. The following table uses examples to demonstrate the rules for installation and autopromotion of a new plugin version:

Existing version	New version	Result	Rule used
1.2.3.300 (promoted)	1.2.8.1234	1.2.8.1234 (promoted)	For two plugins without extension versions, the newer plugin is installed and promoted regardless of the promotion status of the existing plugin.
1.2.3.300 (promoted)	1.3.4.1235	1.3.4.1235 installed. 1.2.3.300 remains promoted.	For two plugins without extension versions, the newer plugin is installed (but not promoted, because it is a minor-version upgrade and not just a patch).

Existing version	New version	Result	Rule used
1.2.3.300 (promoted)	1.2.2.124	1.2.2.124 not installed unless forced (via the `--force`) ectool argument or by checking the **Disable Plugin Version Check** check box in the **Install from File/URL** tab of the Plugin Manager.	For two plugins without extension versions, the newer plugin is not installed if the existing plugin is a newer version than the plugin being installed.
1.2.3.300 (promoted)	1.2.3.300 Extension 1	1.2.3.300 extension 1 (promoted)	With two plugins of the same version, a plugin with an extension version takes precedence over a plugin that is promoted and has no extension version (the plugin with the extension version is installed and autopromoted)
1.2.3.300 Extension 1 (promoted)	1.2.3.300 Extension 2	1.2.3.300 extension 2 (promoted)	With two plugins of the same version, a plugin with a higher extension version takes precedence over a plugin that is promoted and has a lower extension version (the plugin with the newer extension version is installed and autopromoted).
1.2.3.300 extension 2 (promoted)	1.2.3.300	1.2.3.300 is not installed unless forced (by using the `--force`) ectool argument or by checking the **Disable Plugin Version Check** check box in the **Install from File/URL** tab of the Plugin Manager.	A promoted plugin version with an extension version takes precedence over a plugin of the same version but without an extension version.

Existing version	New version	Result	Rule used
1.2.3.300 extension 2 (promoted)	1.2.8.1234	1.2.8.1234 is installed. 1.2.3.300 extension 2 remains promoted.	A newer plugin version is not autopromoted if the existing plugin has an extension version and is currently promoted.

An extension version for a plugin appears as `Extension <version>` on the **Plugin Manager** page in the **Version** column of the **Currently Installed** tab. For example, `Extension 2`:



Using a Plugin

Configuring ElectricFlow to Recognize Your SCM Plugin

For help using your SCM plugin with ElectricFlow Preflight, Default Tracking, and ElectricSentry, see Preflight Builds, and Defect Tracking.

Adding a Subprocedure Step Using the Edit Step or New Step Pages

1. Click the **Projects** tab and choose an existing project or click the **Create Project** link to create a new project.
2. On the **Project Details** page, select an existing procedure or click the **Create Procedure** link to create additional procedures.
3. On the **Procedure Details** page, select the **Plugins** link to go to the **Choose Step** pane to create a New Step using a plugin.
    - In the left pane, select a plugin category and then choose a plugin.
    - In the right pane, select a step type. This opens the **New Step** page so you can add a new subprocedure step (using your selected plugin).

    On the **New Step** page, note that the plugin that you selected already appears in the **Subprocedure** section.

4. Enter information into the **Parameters** section fields.

   These fields are plugin-specific and are frequently different for each plugin.

Configuring the Plugins Directory

For instructions on configuring the plugins directory for your ElectricFlow server, remote agents, or remote web servers, see the *ElectricFlow Installation Guide* at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html.

Procedure Details

This page displays procedure component tables, including a table for Procedure Steps, Parameters, Email Notifications, and Custom Procedure Properties. A procedure describes a process to execute that consists of one or more step. You use this page to create steps.

Procedure Steps

This table displays all steps in the procedure (named at the top of the page). The steps execute in order from top to bottom. Normally only one step executes at a time, but if a group of adjacent steps is marked as "parallel," all of those steps will execute simultaneously.

> **Tip:** If you want to change the order of the steps, click on the icon to the left of the step name and drag it up or down to the new step position.

This table displays only a few of the fields for each step. To view or modify the complete details for a step, click the step's name to go to the **Edit Step** page.

Links at the top of the page

- Use the **Run** link (hovering your mouse over the small down-arrow on the right-side of the link) to choose **Run Immediately** or **Run...**.
  - **Run Immediately**—Selecting this option runs the procedure immediately as set.
  - **Run...**—Selecting this option displays the **Run Procedure** page where you may alter any existing parameters for this procedure, or set an existing credential.
- Click the **Access Control** link to go to the **Access Control** page to add privileges for this procedure.
- Click the **Edit** link to go to the **Edit Procedure** page so you can make modifications if necessary.
- Use the **Copy** link in the **Actions** column to make an exact copy of this step.
- Choose a **New Step** link to create the type of step you need.

Creating a new step

The following list describes some of the more common steps you can create.

- **Command**—Takes you directly to the **New Step** page to write your own step. A **Command(s)** text box is available to add a script. This step invokes a `bat`, `cmd`, `shell`, Perl script, or similar.

- **Subprocedure**—The **Choose Subprocedure Step** dialog is displayed, allowing you to select a project or plugin from which you want to call an existing procedure to create a step. This step invokes another ElectricFlow procedure and will not complete until all subprocedure steps have finished.
- **Plugin**—Opens the Choose Step panel. The left pane displays plugin categories and the right pane displays available pre-configured steps for your left-pane selection. To quickly find your application or plugin, type your selection name in the **Search** box. Select the step you want to create and ElectricFlow takes you to the **New Steps** page.
  - On the **New Step** page, notice that the plugin step you selected is already displayed in the Subprocedure section.
  - Enter information in the **Parameters** section fields. These fields are specific to the plugin you chose. (These fields are different for each plugin.)

Enter information in any remaining fields to complete your step. See the New Step Help topic for more information.

A note about plugins

Plugins are the vehicle ElectricFlow uses to integrate source control systems (such as Subversion or Perforce), defect tracking applications (such as JIRA or MKS), reporting functionality, code analysis applications, build applications, and much more. To see the complete list of plugins bundled and installed with ElectricFlow, see the Plugin Manager page (go to **Administration** > **Plugins**).

> **Tip:** If you do not see the SCM system or defect tracking system you prefer, a Plugin Catalog is available from the Plugin Manager also. All plugins in the catalog are available for installation into ElectricFlow.

Parameters

Each procedure can define parameters, which are values provided to the procedure when it is invoked. Each parameter value is placed in a property associated with the job and the procedure can use the property to control its execution. For example, a parameter might specify a particular branch of a product to build; another parameter might indicate whether unit tests should be run during this build.

When you define a parameter, indicate whether the parameter is required (meaning the procedure will not run unless a value is provided for the parameter). Also, you can provide a default parameter value and a description to help callers understand how to use the parameter.

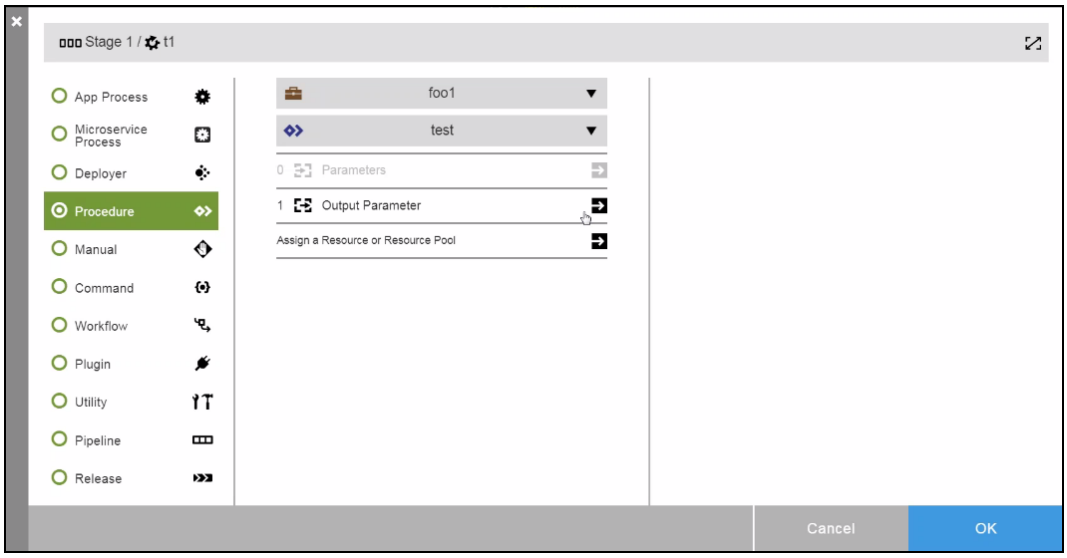
- Click the **Create Parameter** link to go to the **New Parameter** page.
- Or click on the name of the parameter to edit its contents.

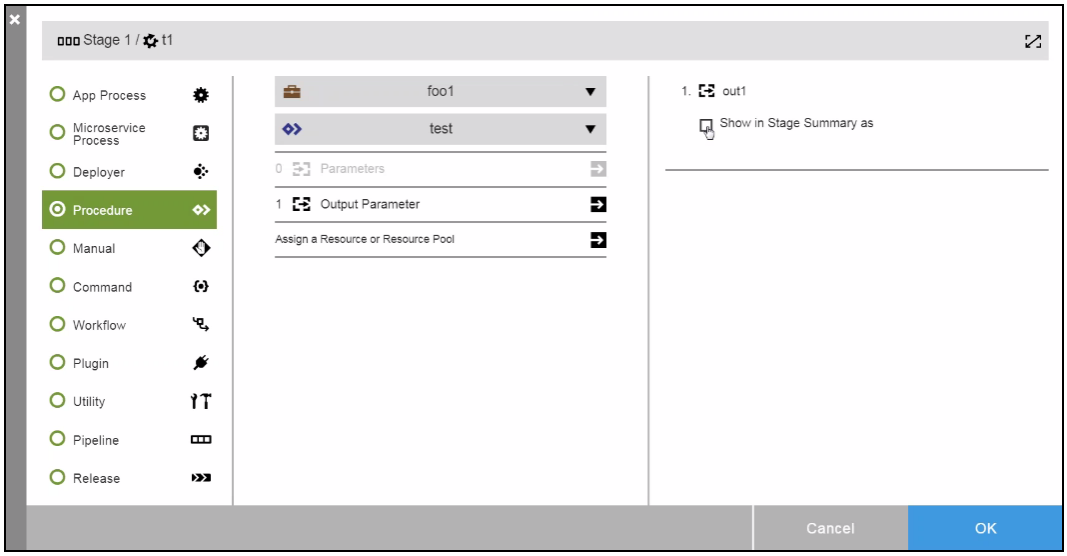
Output Parameters

Each procedure can output parameters, which are values provided by the procedure during runtime. Each formal parameter value is placed in a property associated with the job. Procedures (as well as plugins and processes) in ElectricFlow will generate output properties at runtime that can then be used in various ways in other places (such as for reporting, as input to another process, or used in conditions or gates).

This functionality provides an easy way to determine what properties are generated by a process or procedure in order to use. With output parameters defined at the procedure (or process) level, you do not need to look at the script under each step to determine what output properties will be created when

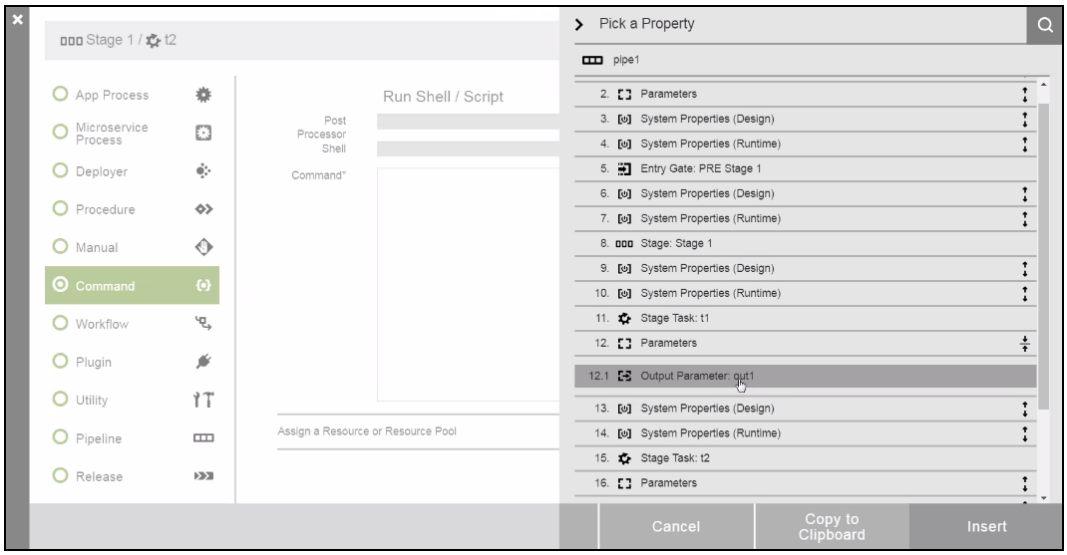
the procedure is run. Creating output parameters is easier than running the process and then inspecting the output properties and values, reading through the plugin code to find the output properties, or reading the plugin documentation.

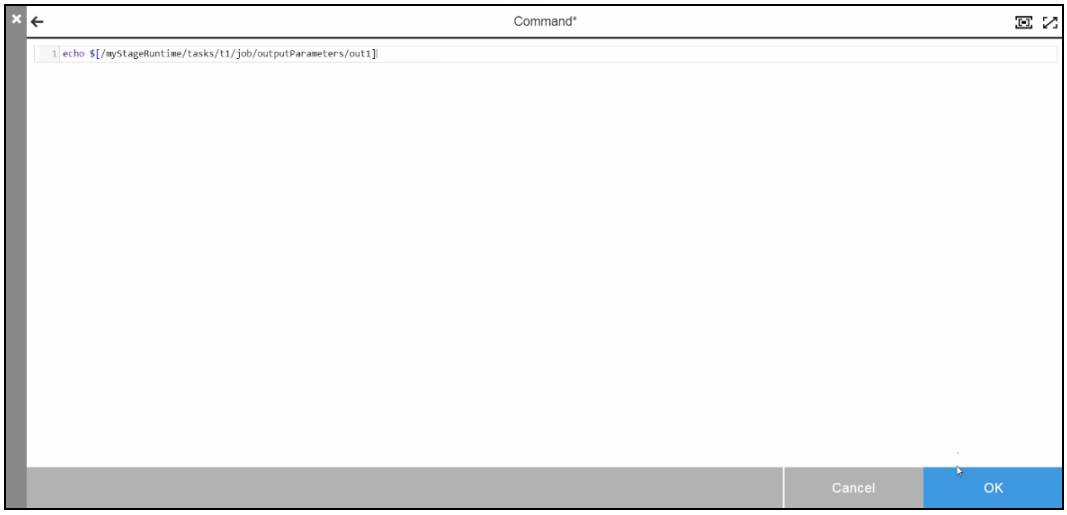




Output Parameters and the Property Picker

Output parameters are available through a property picker in the UI to simplify how you use the output of a job or pipeline run in a different job or task execution. You can use the Property Picker to browse and select properties and see the absolute path for that property to be inserted. Intrinsic properties are included, which is helpful for writing conditions (or any other places where properties are referenced).





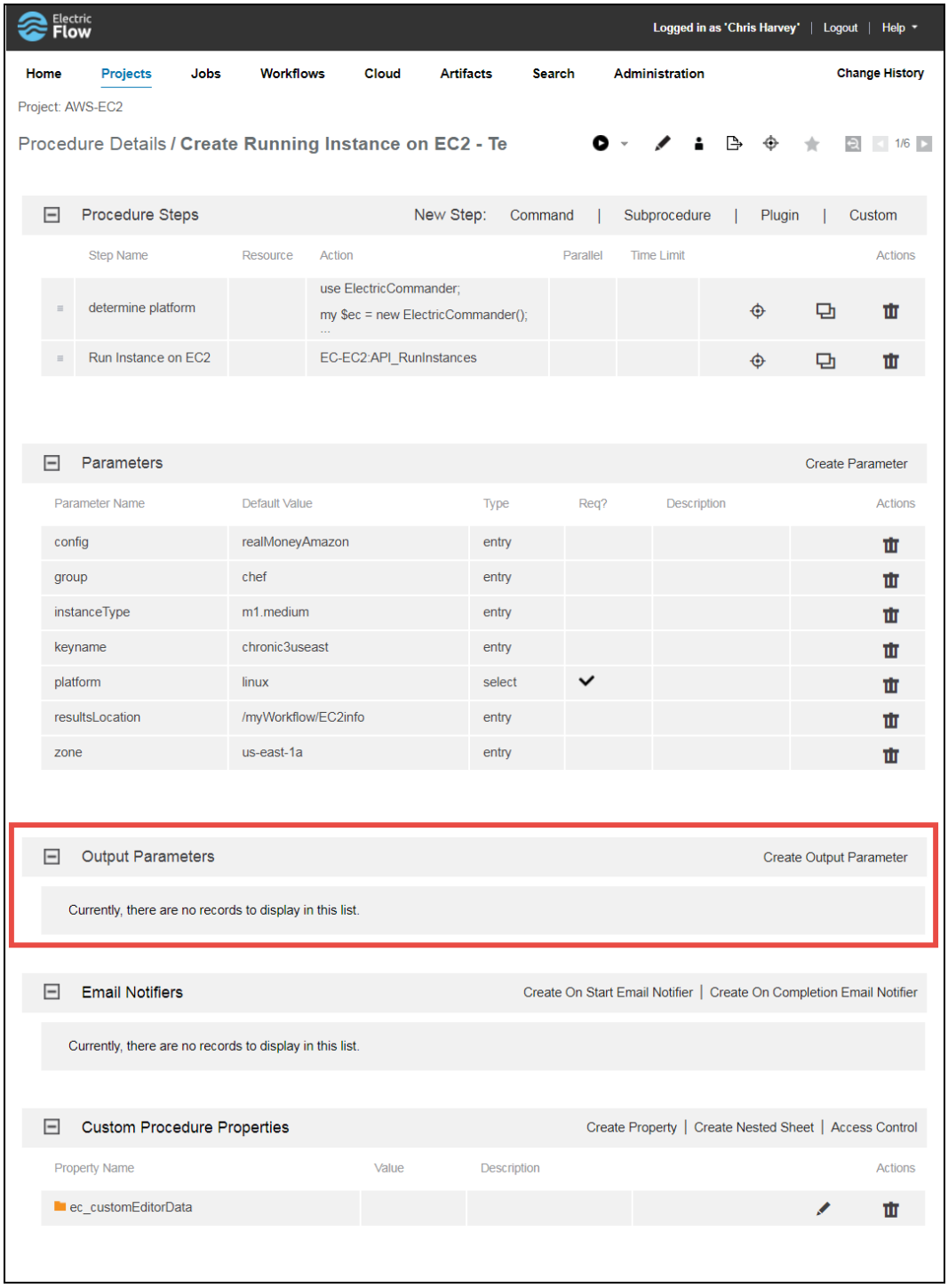
Usage Scenarios for Output Parameters

The following scenarios provide examples of how output parameters are useful.

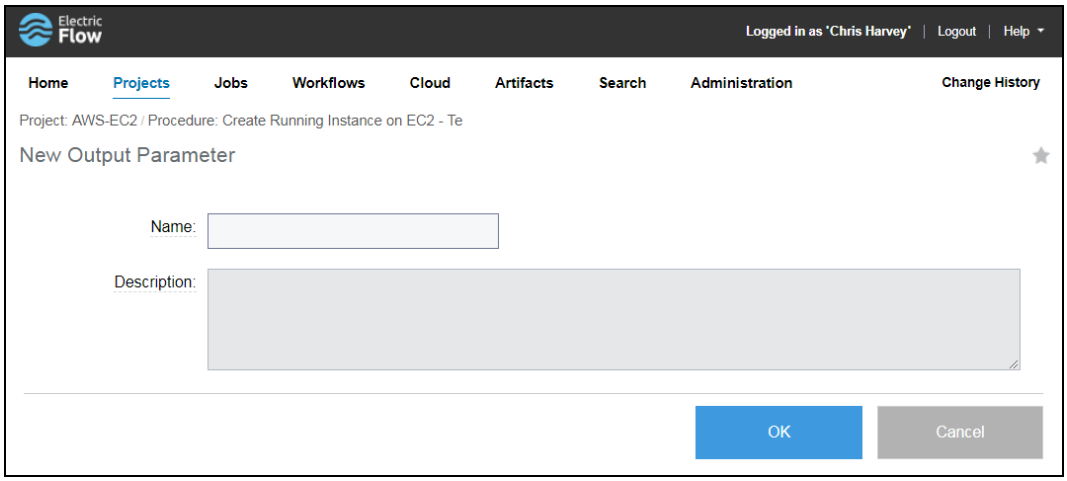
- I wrote a procedure that creates a number of properties in property sheets that will be used in a later job creating a report. When writing the report procedure, I do not remember the exact name of property sheets and hierarchy that I am creating. I have to go back to the procedure code to copy and paste the names.
- I am calling a plugin to run a set of tests from my pipeline task. In the automated exit gate condition, now I need to reference the result property generated by the plugin but do not know its name or how or where it is stored.
- I am calling a common utility procedure written by other team members and need to use the output values.

Creating or Editing an Output Parameter

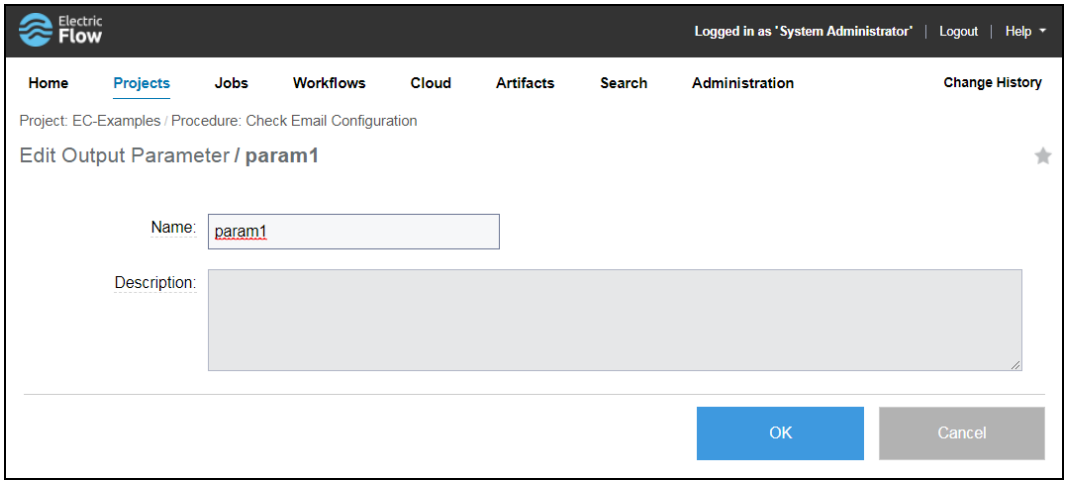
When you define a formal output parameter, you must indicate whether the parameter is required (meaning the procedure will not run unless a value is provided for the parameter). Also, you can provide a default parameter value and a description to help callers understand how to use the parameter.



Click the **Create Output Parameter** link to go to the **New Output Parameter** page.



Or click the name of the parameter to go to the **Edit Output Parameter** page to edit its contents.



You can set the value of an output parameter by using the `setOutputParameter` API command. The following DSL code provides an example:

```
project "Test",{
  procedure "Test",{
    formalOutputParameter "param1"
    step "Test", command: 'ectool setOutputParameter param1 "output value 1"'
  }
}
```

For details about using the `setOutputParameter` command and other commands related to output parameters, see the "API Commands—Parameter Management" chapter in the *ElectricFlow API Guide* at http://documentation.electric-cloud.com/html/eflow_doc/FlowIndex.html.

Email Notifiers

Use the **Create On Start Email Notifier** or **Create On Completion Email Notifier** links to go to one of the respective pages to create an email notifier for this procedure.

After creating an email notifier, click on the "bread crumb" link at the top of the page to return to the **Procedure Details** page.

Custom Procedure Properties

- To add new properties to this procedure, click the **Create Property**, **Created Nested Sheet**, or **Access Control** links.
- Click on the Property Name to edit its contents.
- Nested Property Sheets appear in this table preceded by a folder icon.

> **Tip:** To rename this procedure or change its description, click the **Edit** link at the top of the page.

Procedure—create new or edit existing procedure

Editing an Existing Procedure

Use this page to modify certain overall information about a procedure. Additional information, such as procedure steps and properties, can be viewed and modified on the **Procedure Details** page. To access this page, click the **Project Name** link on the location path (breadcrumb), then select a procedure name. To rename a procedure, enter a new procedure name in the **Name** field and click **OK**.

Step—create new or edit existing step

> **Note:** If you chose to create a step using the Plugin link, additional help is available in the Parameters section by clicking the "?" icon. The Parameter section is populated according to type you chose to create. Different step types require different information.

To create a new command or subprocedure step

Enter information into the fields as follows:

Field Name	Description
General section (for all step types)	
Name	Enter a name for the step that must be a unique name within the procedure.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Command section (for command steps only)	
Command(s)	A script to execute the step functions that is passed to the step's shell for execution.

Field Name	Description
Resource	Enter a unique resource or resource pool name or click Browse to select a resource. If you leave this field blank, the procedure or project provides a default resource. • The job step must have the execute privilege on the resource or resource pool. If it does not, the job step will be canceled with an access denied error. • The resource or resource pool must be enabled. If the resource or pool is not enabled, no error is generated, and the step remains in the runnable state until the pool is enabled.
Postprocessor	If this field remains blank, no postprocessor runs for the step. If this field is not blank, it specifies a command (passed to the step's shell for execution) that analyzes the log file for the step and collects diagnostic information for reporting. For more information, see the Postprocessors Help topic.
Subprocedure section (for subprocedure steps only)	
Procedure	Displays the current project and procedure. Click **Change** to use the pop-up menu to select a new project or procedure.
Resource	Enter a unique resource name or click Browse to select a resource. If you leave this field blank, the procedure or project provides a default resource.
Parameters section (for subprocedure steps only)	
This section expands automatically, entering the fields you need specifically for the Subprocedure you chose in the previous section. For additional help with field descriptions, click the "?" icon if available. To toggle between the standard view and property view for this section, click the  icon. This icon appears in the standard form when one or more of the parameter types are not **Text entry** or **Text area**. It appears in the custom form for all the supported parameter types when the form has at least one parameter.	
Advanced section (for both command and subprocedure steps)	

Field Name	Description
Precondition	By default, if this field is blank, the step has no precondition and will run when scheduled. Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a *precondition* is evaluated. A *precondition* is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a \"0\" or \"false\" is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE. You cannot use timestamps in preconditions on any object that supports preconditions. This includes stages, gates, and tasks as well as procedures and process steps. Precondition example: Assume we defined these 4 steps: 1. Build object files and executables 2. Build installer 3. Run unit tests 4. Install bits on test system Step 1 is an ordinary serial step. Steps 2 and 3 can run in parallel because they depend only on step 1's completion. Step 4 depends on step 2, but not step 3. You can achieve optimal step execution order with preconditions: • Make steps 2-4 run in parallel. • Step 2 needs a job property set at the end of its step to indicate step 2 is completing (`/myJob/buildInstallerCompleted=1`). • Set a precondition in step 4: `$[/myJob/buildInstallerCompleted]`
Run Condition	By default, if this field is blank the step will always run. This field accepts "fixed text" or text embedding property references that are evaluated into a logical TRUE or FALSE. A "0" or "false" is interpreted as FALSE. An empty string or any other result is interpreted as TRUE. The property reference can be a JavaScript expression, for example, this expression could test whether the name of a step is equal to the value of a property called "restartStep". `$[/javascript (myStep.stepName == myJob.restartStep)]`

Field Name	Description
Error handling	This field determines what happens to the procedure if the step fails. The first choice, "procedure continues, but overall status will be error" means an error in the step does not abort the procedure; subsequent steps will run as usual. However, an error will be reflected in the overall outcome of the procedure. If this is the top-level procedure in the job, the job will have an error outcome. If this is a subprocedure, the step invoking the subprocedure will have an error outcome, which still could cause the job to abort, depending on the error handling for that step. The second choice for this field is "abort procedure but allow running steps to continue." In this case, if an error occurs, no new steps are initiated in this procedure except those where "always run" was selected. Any steps currently running are allowed to complete, then the procedure will abort and its outcome will be set to error as described previously. Six error handling options are: • `failProcedure`—The current procedure continues, but the overall status is error (default). • `abortProcedure`—Aborts the current procedure, but allows already-running steps in the current procedure to complete. • `abortProcedureNow`—Aborts the current procedure and terminates running steps in the current procedure. • `abortJob`—Aborts the entire job, terminates running steps, but allows alwaysRun steps to run. • `abortJobNow`—Aborts the entire job and terminates all running steps, including alwaysRun steps. • `ignore`—Continues as if the step succeeded.
Time Limit	The maximum amount of time the step can execute. If the step exceeds this time, it will be aborted. Specify a number and then use the pull-down menu to select the "time units" of the number you specified.
Run in parallel	If this box is "checked," this step will run concurrently with any adjacent steps marked as parallel. If this box is not checked, the step will NOT run until all previous steps in the procedure have completed, and it will complete before any following steps execute.

Field Name	Description
Always run step	If a procedure is aborted (either because of an error in a step or because the job was aborted), under normal conditions no new steps will start in the procedure. However, if this box is "checked," the step will run even if the procedure is aborted. This facility is most commonly used for steps at the end of a job that generate reports or perform cleanup operations. **Note:** The Run Condition field is still evaluated and may cause the step to be skipped even if this box is checked.
Retain Exclusive	Select one of the following options: • None—the "default", which does not retain a resource. • Job—keeps the resource for the duration of the job. No other job can use this resource, regardless of its step limit, until this job completes or "Release Exclusive" is used in a step. Future steps for this job will use this resource in preference to other resources—if this resource meets the needs of the steps and its step limit is not exceeded. • Step—keeps the resource for the duration of the step. • Call—keeps the resource for the duration of the procedure that called this step, which is equivalent to 'job' for top level steps.
Workspace	Use this field to define a workspace for the step to use. You can either type-in a workspace name or click Browse to select a workspace. If you leave this field blank, the workspace is determined by the procedure or project.
These "Advanced" fields are used for command steps only	
Broadcast	"Checking" this box means replicate the step to execute (in parallel) on each of the specified resources (for example, if a resource pool is specified, run the step on each resource in the pool).

Field Name	Description
Release Exclusive	Select one of the following options: • `none`—The "default"—no action if the resource was not previously marked as "retain". • `release`—Releases the resource at the end of this step. If the resource for the step was previously acquired with "Retain exclusive" (either by this step or some preceding step), the resource exclusivity is canceled at the end of this step. The resource is released in the normal way so it may be acquired by other jobs. • `releasetojob`—Allows a step to promote a Step exclusive resource to a Job exclusive resource.

Field Name	Description

Shell	This shell will be used to execute the step's commands on a resource. For example, using `sh` or `cmd /c`, the agent saves the command block to a temporary file with a `.cmd` extension and runs it:
	`sh foo.cmd` or `cmd /c foo.cmd`
	
	If you do not specify a shell on a step, at step run-time the server looks at the resource shell. If a resource shell is not set, the shell line used by the agent is platform dependent:
	Windows: `cmd /q /c "{0}.cmd"`
	UNIX: `sh -e "{0}.cmd"`
	
	When you specify a shell (in the step or resource), and omit the `cmd-file` marker, the agent notices the omission and takes the correct action.
	For example: a user specifies `sh -x`.
	The agent converts this to `sh -x "{0}.cmd"`
	
	Two alternate forms of shell syntax where ElectricFlow uses a "marker," `{0}`, as a placeholder for the command file argument:
	• *<myShell>* `{0}` *<potential extra shell args>*
	In this example, the command file is not meant to be the last argument in the final command line.
	For example, `mysql -e "source {0}"`
	This shell example runs the `mysql` command against this step's command containing `sql`.
	• *<myShell>* `{0}` *<.file extension> <potential extra shell args>*
	In this example, the shell requires the command file to end in an extension other than `.cmd`.
	For example, `powershell "& '{0}.ps1'"`
	This shell example runs Microsoft PowerShell against this step's command containing PowerShell commands.
	
	When the agent parses the shell, it will parse the extension as everything after `{0}.` until it sees a space or non-alphanumeric character.
	
	Note: If your script uses International characters (non-ASCII), add the following block to the top of your `ec-perl` command block:
	`use utf8;`
	`ElectricCommander::initEncodings`

Working Directory	The directory where the commands for this step execute. A relative name is interpreted relative to the root directory for the job workspace. If you leave this field blank, the step's working directory will be the top-level directory in the job workspace. If this step will run on a proxy resource, this directory must exist on the proxy target. The step will run in this directory on the proxy target.

Field Name	Description
Log File	The step's log file name, specified relative to the root directory in the job workspace. If you leave this field blank, ElectricFlow picks a unique name for the log file based on the step name.
Impersonation section (for both command and subprocedure steps)	
Credential Project	By default, the Current project is used, or click Browse and select a different project.
Credential Name	If you select a credential in this field, the step runs under the account from that credential. This field is usually blank, which means a default credential is used. For more detailed information about credentials, see the Credentials and User Impersonation Help topic.

Click **OK** to create the step.

To edit an existing command or subprocedure step

You can modify any of your previously entered step information and add information to one or all of the four additional sections.

Attached Credentials

Click the **Attach Credential** link to retrieve an existing credential for this step. If you need to create a new attached credential, return to the Project Details page for this step.

Attached Parameter Credentials

Click the **Attach Parameter Credential** link to retrieve an existing parameter credential for this step.

Email Notifiers

Choose **Create On Start Email Notifier** or **Create On Completion Email Notifier** to go the respective page to create the email notifier you need.

Custom Step Properties

Choose from the **Create Property**, **Create Nested Property**, or **Access Control** links to add a custom property to this step. Each Property pop-up box has its own Help topic or go to the main Properties Help topic for more information. For more information about Access Control, go to the main Access Control Help topic.

Click **OK** to save your modified step information.

Publish Artifact Version Step

The following parameters are available to create this step:

Field Name	Description
Artifact	Artifact name, in the form `<groupId>:<artifactKey>`. If the artifact does not exist, it will be created if this procedure's launching user or this project principal has the required permissions.
Version	A full version string takes the form: `<major>.<minor>.<patch>-<qualifier>-<buildNumber>`. The version specification must be unique across all of this artifact's versions.
Repository	Name of the repository where this new artifact version will be published.
Enable Compression	Check this box to compress the artifact version before it is stored in the repository.
From Directory	Name of the directory in the job's workspace containing files that comprise the artifact version to be published. If not specified, the entire workspace is used.
Include Patterns	List file "include" patterns one pattern per line, to limit which files are published. If no patterns are specified, all files are included.
Exclude Patterns	List file "exclude" patterns one pattern per line, to limit which files are published. If no patterns are specified, no files are excluded.
Dependent Artifact Versions	List dependent artifact versions one per line, each in the form `<groupId>:<artifactKey>:<versionRange>`. All dependent artifact versions must exist for this artifact version to be retrievable. When this artifact version is successfully retrieved, its dependent artifact versions are retrieved also.

Retrieve Artifact Version Step

The following parameters are available to create this step:

Field Name	Description
Artifact	Artifact name, in the form `<groupId>:<artifactKey>`.
Version	Select the latest version (**Latest**), the exact version of the artifact (**Exact**), or a version range (**Range**). Version is in the form: `<major>.<minor>.<patch>-<qualifier>-<buildNumber>`
Retrieve to directory	If you want to retrieve this artifact version to a specific directory location, select the check box, then specify the full path to the directory where you want to retrieve this artifact version.

Field Name	Description
Overwrite	Use the drop-down menu to select one of the following options: • `true`—deletes previous content in the directory and replaces the content with your new version. • `false`—(existing behavior) if the directory does not exist, one will be created and filled with the artifact's content. If the directory exists, a new directory is created with a unique name and the artifact contents is supplied there. • `update`—this is similar to a merge operation—two artifact versions can be moved into the same directory, but individual files with the same name will be overwritten.

Retrieved Artifact Location Property	Name or property sheet path used by the step to create a property sheet. This property sheet stores information about the retrieved artifact versions as XML in the `artifactVersionXML` child property, and the file system location of the retrieved artifact version in the `cacheLocation` child property.
	
	The initial value of this field includes `$[assignedResourceName]` as one of the levels in the property path because different resources may have different cache directories, and you do not want a retrieval on one resource to clobber the data regarding a retrieval on a different resource.
	
	Typically, a step that needs to use files from an artifact retrieved by this step will specify it in a Perl script similar to this:
	`my $cmdr = new ElectricCommander();`
	`my $xpath = $cmdr->getProperty`
	`("/myJob/retrieveArtifactVersions/$[assignedResourceName]`
	`/MyGrp:MyKey/cacheLocation");`
	
	`my $retrieveDir = $xpath->findvalue("//value")->value();`
	
	`system("java", "-cp", "$retrieveDir/lib/bar.jar", "Bar");`
	
	This example shows how to retrieve the location of the retrieved `MyGrp:MyKey` artifact version on the resource, and add a file (`bar.jar`) in its `lib` directory to classpath, so the `Bar` class can be run.

Filters	Enter search filters, one per line, applicable to querying the ElectricFlow database for the artifact version to retrieve.

Artifact Retrieval Dependency Order Explained

The order of dependencies registered for an artifact version are significant.

Consider this scenario:

- A depends on B (any version) and C [1.0, 2.0)
- B depends on C (any version)
- C versions 1.0, 2.0, and 3.0 exist

When retrieving A, the dependency algorithm evaluates B first. The algorithm finds that the max version of B depends on any version of C, so the algorithm looks for max version C and finds C 3.0. Because this chain is satisfied, the algorithm returns to A and evaluates its next dependency "C [1.0, 2.0)". This results in matching C 1.0.

The returned artifacts are: A, B, C 1.0, and C 3.0.

Consider if the A dependency is changed to:

- A depends on C [1.0, 2.0), B (any version)

The algorithm will choose C 1.0 first. Then the algorithm evaluates B, determines that its "C (any version)" is satisfiable by the already-chosen C 1.0.

The returned artifacts are A, B, and C 1.0.

> **Note:** In the version range syntax [] indicates inclusive, and () indicates exclusive.

Send Email Step

The following parameters are available to create this step:

Parameter Name	Description
Email Configuration	The name of the email configuration to use. If no configuration is specified, the configuration named "default" is used.
To	List the "To" recipients for the email message, one per line. A recipient can be a user or group name or a complete email address.
CC	List the "CC" recipients for the email message, one per line. A recipient can be a user or group name or a complete email address.
BCC	List the "BCC" recipients for the email message, one per line. A recipient can be a user or group name or a complete email address.
Subject	The subject line for the email message.
Message	The body of the email message or a workspace file containing the body. If both a plain text and an HTML message are provided, both values are sent as alternates in a multipart message. If a raw message is provided, both values are sent as alternates in a multipart message. If a raw message is provided, the value should be a properly formatted RFC822 message.
Advanced options	
Multipart Mode	The multipart mode of the email message. Defaults to "none" unless there are multiple parts, in which case it defaults to "mixedRelated".
Header(s)	One or more RFC822 email header lines (for example: "`reply-to: user@host.com`").

Parameter Name	Description
Attachment(s)	One or more files from the job's workspace to send as attachments. The file name extension is examined to determine the content-type.
In-line Attachment (s)	One or more inline attachments specified as a `contentId` and a workspace file name. The file name extension is examined to determine the content-type.

New Extract Preflight Sources Step

Choose one of the preflight plugins to create a step to retrieve an artifact version. There are numerous source code management (SCM) plugins. For example, Accurev, Bazaar, CVS, ClearCase, Git, Mercurial, Perforce, SVN, StarTeam, Team Foundation Server, and Vault.

Enter the following information:

Field Name	Description
Step Name	Enter a unique name for your preflight step. You can use any name of your choice.
SCM Configuration	Use the pull-down menu to select a source control configuration. If a configuration for your source control type does not appear in the drop-down menu, select the Administration > Source Control tabs to configure your source control system.
Destination Directory	This field appears after specifying your SCM from the pull-down menu. If this field does not specify "Required", you do not need to enter this information. The Destination Directory is a path relative to the job's workspace, where the source tree will be created.

Click **Submit** after entering your information. After clicking **Submit**, the Edit Step page is displayed.

Use the Edit Step page to specify additional information for your preflight step. Also, you can attach a credential, create an email notifier, add custom properties, and so on to your step. For assistance using the Edit Step page, access the **Help** link in the upper right corner of the page.

For more information about preflight builds, see the Preflight Builds Help topic.

Parameter—create new or edit existing parameter

To create a new parameter

Enter information into the fields as follows:

Field Name	Description
Name	Enter a unique name to specify the parameter when a procedure or workflow is invoked.

Field Name	Description
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Type	Select the parameter type from the drop-down menu. The following "types" are available.
Text entry	Allows a short text entry in the Default Value field.
Text area	Expands the Default Value field to allow adding a longer script-type entry.

Dropdown menu	Creates a drop-down menu from which to select a value when the parameter is presented.
	• Enter options—Click **+ Add Option** to add a new row. Type-in the text and value for each option. The text is what will be displayed in the menu, and the value is the parameter value if that option is selected.
	• Load options from list—Enter a pipe-separated list of options (for example, `foo|bar|baz`). The text and value for the options will be the same.
	• Load options from property sheet—Enter the path to property sheet that contains options for the parameter. The property sheet must be created in a specific format:
	  • An `optionCount` property must exist whose value is the number of options.
	  • For each option, create a nested property sheet called `option`N, where N is the option number, starting with `1`.
	  • In each nested sheet, create two properties—`text` and `value`. The value of the property `text` will be displayed in the menu. The value of the property `value` is the parameter value if that option is selected.
	  • If `optionCount` is set to `3`, you must create three nested sheets—`option1`, `option2`, and `option3`.

Field Name	Description

Radio selector	Creates "radio" buttons to select an entry when the parameter is presented.
	• Enter options—Click **+ Add Option** to add a new row. Type-in the text and value for each option. The text is what will be displayed in the menu, and the value is the parameter value if that option is selected.
	• Load options from list—Enter a pipe-separated list of options (for example, `foo|bar|baz`). The text and value for the options will be the same.
	• Load options from property sheet—Enter the path to property sheet that contains options for the parameter. The property sheet must be created in a specific format:
	  • An `optionCount` property must exist whose value is the number of options.
	  • For each option, create a nested property sheet called `option`N, where N is the option number, starting with `1`.
	  • In each nested sheet, create two properties—`text` and `value`. The value of the property `text` will be displayed in the menu. The value of the property `value` is the parameter value if that option is selected.
	  • If `optionCount` is set to `3`, you must create three nested sheets—`option1`, `option2`, and `option3`.

Checkbox	Creates a checkbox for a value to select (or not) when the parameter is presented. • Value when unchecked—The value of the parameter when the checkbox is unchecked. • Value when checked—The value of the checkbox when the checkbox is selected. • Initially checked—Whether or not the checkbox should be checked initially. If true, set the "Default value" to match the "Value when checked". If false, set the "Default value" to match the "Value when unchecked".
Credential	Selecting this option requires the user to specify a user name and password to use this parameter at runtime. Credential parameters are not supported on state definitions.
Project	Creates a project selector field on the Run Procedure page to choose a different project name where the parameter needs to find additional information.

Field Name	Description
Saved Filter	Use this option to reference any saved filters, which is passing the filter into the procedure at runtime (primarily used for reports). Saved filters are stored as a property in your chosen project.
Default Value	Default value to assign to the parameter if no explicit value is provided.
Required?	Click the checkbox to select the parameter as "required." If the parameter is required, the procedure or workflow will not run without entering a value for that parameter. In this case, the default value is ignored. After the procedure or workflow begins execution, the parameter value is available in the job or workflow's property sheet with the same name as the parameter.
Defer Expansion?	Determines whether to expand the parameter value. A parameter value expansion occurs in one of two places: • When the procedure request is made (the default) (the check box is unchecked) • When the step executes (the check box is checked) The check box is unchecked by default (meaning that expansion is not deferred).

Click **OK** after completing the fields.

To edit an existing parameter

You may change any information or add new information as necessary. For example, to rename a parameter, enter a new parameter name in the **Name** field and then click **OK**.

For more information on property sheets, see the Properties Help topic.

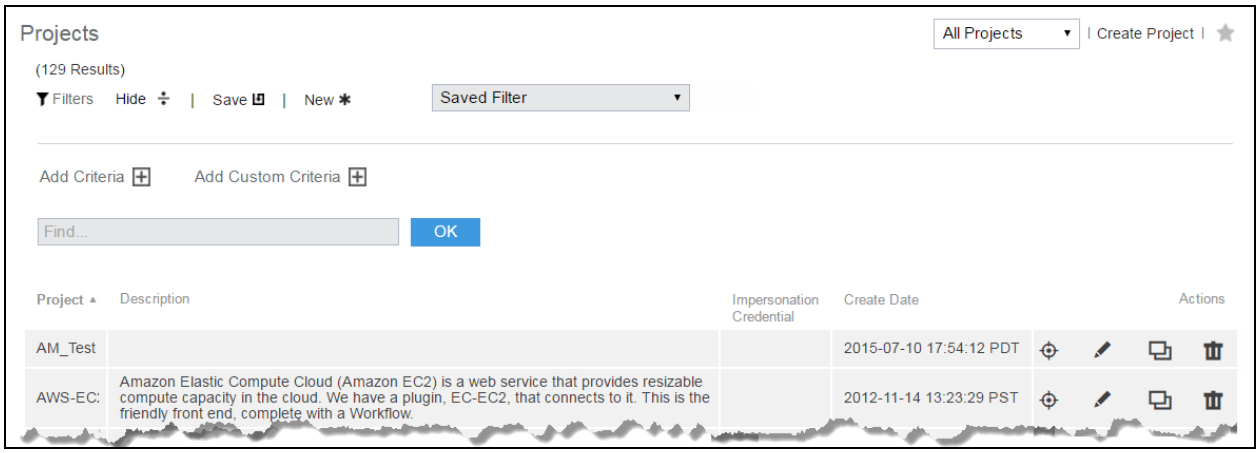
Projects

This page displays all projects available on this ElectricFlow server—these are the ElectricFlow-supplied default projects and the projects that you create. From this page, you can navigate to a project to view its procedures, steps, schedules, credentials, workflows, and other components.

A project is a top-level container within ElectricFlow. Most information about software production processes, such as procedures, schedules, jobs, and workflows, are contained within a project.

Searching for Projects

You can search for projects and save the search filters for later use. See Context Searching and Filtering on page 1244 for details.



Displaying Projects Marked with a Specific Tag

The **All Projects** drop-down menu (at the top of the table) displays all tags that you defined to mark, label, or group related or similar projects. When you select a tag, the project list changes to display only the projects with that tag.

Viewing Project Details

To see project details, click the project name in the **Project** column to go to the **Project Details** page.

Editing, Copying, or Deleting a Project

To **Edit**, **Copy**, or **Delete** an existing project, click a corresponding link in the **Actions** column.

> **Note:**
> If you delete a project, the background deleter marks the project for deletion, but the deletion process might not start until the server completes other tasks already queued or in progress. Deleting a large project (one containing hundreds of jobs, steps, or other objects) could take considerable time.

Adding a Tag to a Project

To add a tag to a project:

1. Click the **Edit** link for the project.

   The **Edit Project** page appears.

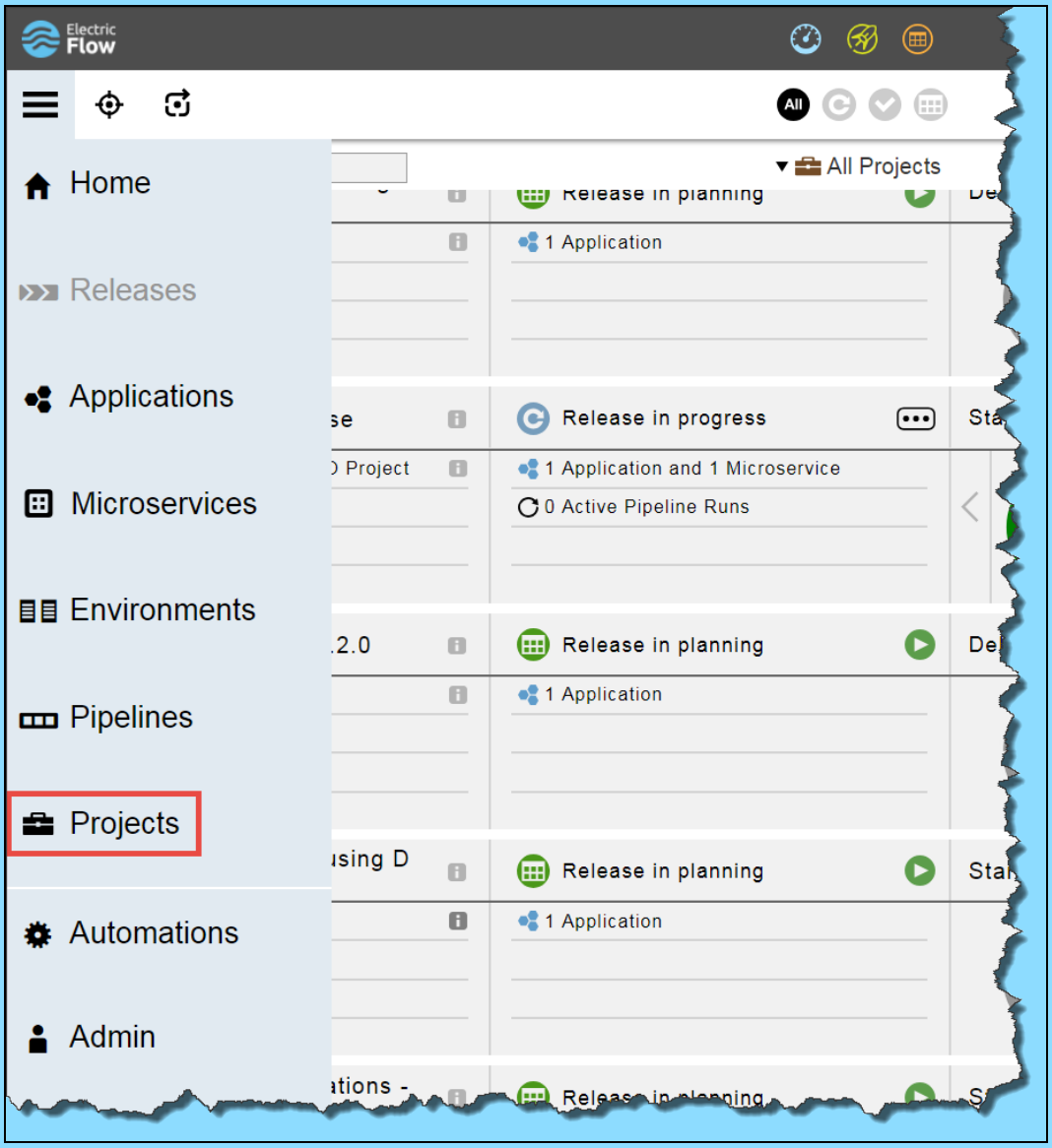
2. Enter a tag name in the **Tags** field, and then click **OK**.

   The tag name now appears in the drop-down menu on the **Projects** page.

Creating a Project

To create a project, click **Create Project**.

Tip:
You can also create, copy, edit, or delete a project via the **Projects** option from the launch pad in the Deploy web UI:



For details, see Creating or Editing a Project in the Deploy Web UI on page 56.

Project—Create or Edit a Project in the Automation Platform Web UI

Creating a Project

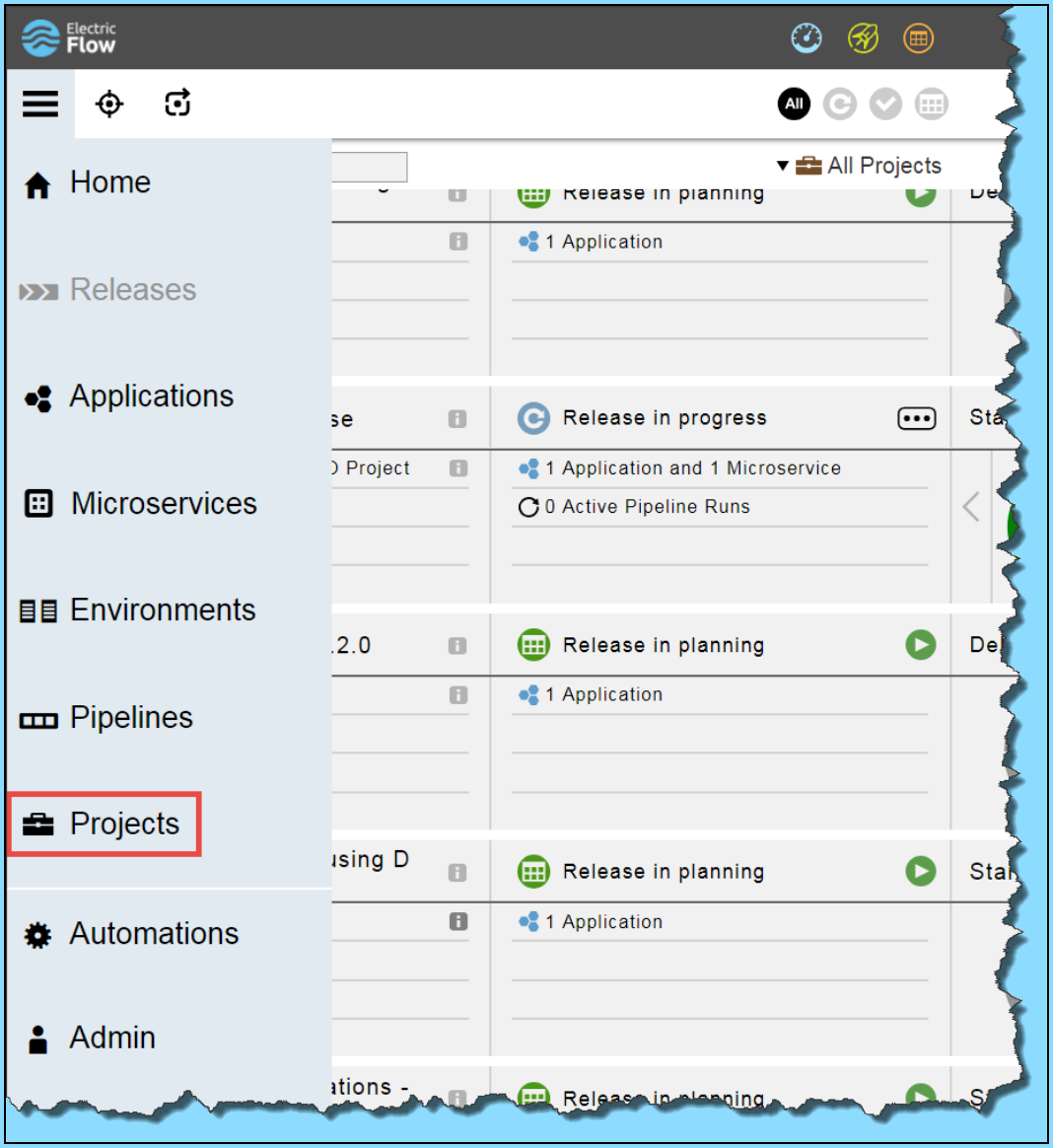
Enter information into the fields as follows:

Field Name	Description
Name	Enter a unique project name. Use a meaningful name such as a work group or product. **Note:** To avoid conflicts, do not use "Electric Cloud" or any other ElectricFlow-supplied project name. Also, the "EC-" prefix is reserved for ElectricFlow-supplied project names.
Enable Change Tracking	Disable or enable change tracking for this project. This feature records every change between every state of non-runtime objects and lets you revert to an object's prior state. Tracked objects include applications or microservices, procedures, workflows, workspaces, resources, and project-owned components such as library components. For more information, see Change Tracking on page 1374
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Credentials	(Optional) Credentials and impersonations available to this project. For new projects, you are prompted to create project credentials on a subsequent dialog after project details are entered. See Credentials and User Impersonation on page 876 for further information.
Default Resource	(Optional) Default resource for all jobs that run under the project. This is a convenient way to use a single resource for an entire project. You can specify resources in several other places such as in procedures and individual job steps. A workspace specified elsewhere takes precedence. If you have not set up resource names or locations yet, see Resources on page 1208 for instructions.
Default Workspace	(Optional) Default workspace for all jobs that run under the project. This is a convenient way to use a single workspace for an entire project. There are several other places where you can specify workspaces, such in as procedures and individual job steps. A workspace specified elsewhere takes precedence. If you have not set up workspace names or locations yet, see Workspaces and Disk Space Management on page 1587 for more information.

Click **OK** to save your changes.

Your new project name will appear on the **Projects** page.

Tip:
You can also create or edit a project via the **Projects** option on the launch pad in the Deploy web UI:



For details, see Creating or Editing a Project in the Deploy Web UI on page 56.

Editing a Project

Update the information in the fields as needed. See the table in Creating a Project on page 1187 above for field descriptions. If you are editing an existing project, you can make the following additional changes:

Field Name	Description
Tags	(Optional) One or more tags that categorize or mark this project to identify its relationship to one or more other projects or groups. For example, you can tag a group of projects as "production" or "workflow" or use your name so you can quickly filter the project list to see only those projects. These tags appear in the **All Projects** drop-down menu on the **Projects** page. When you select a tag from the drop-down menu, you see only the projects with that tag. Do not use spaces or symbols. For example, if you want to mark a project with a tag name that represents "Engineering Team A," enter "eng_team_a." If a project is related to several project categories, you can enter a space-separated list to add multiple tags at the same time. Assigning a tag creates an `ec_tag` property. You can see this property when you select the **Properties** subtab from the **Project Details** page.
Impersonation Credential	Credential (user name and password) to be used by default for all jobs that run under this project. If this field is blank, credential information from other sources is used. • **Credential Project**—Use the current project (the default), or click the other radio button and then click **Browse** to choose another project. • **Credential Name**—Name of the credential for the project. Enter the name, or click **Browse** to select one. For more information, see Credentials and User Impersonation.

Click **OK** to save your changes.

Project Details

This page displays project components, including procedures, workflow specifications, jobs, schedules, any credentials, properties, and reports.

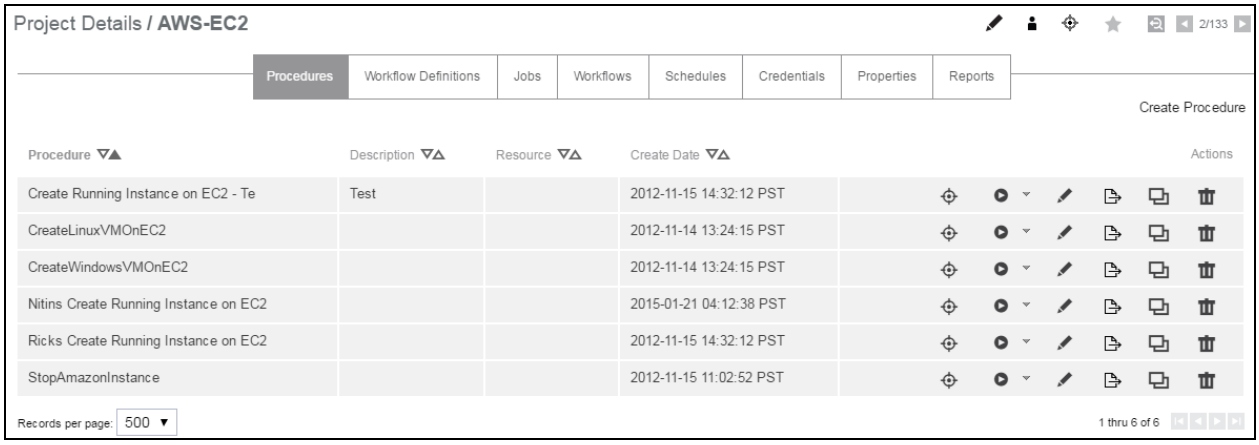
Links and actions at the top of the page

- **Edit**—Opens the Edit Project page.
- **Access Control**—Opens he Access Control page for this project.
- "star" icon—Saves this job information to your Home page.
- "curved arrow" icon—Returns you to the Projects page.
- Pagination—Use the "previous" and "next" arrow icons to view the previous or next project.

  The numbers between the arrow icons display the number of projects you can view and the first number indicates which project [in the list] you are viewing.

The "tabbed" sections

The tabs at the top of the table allow you to select the type of information you want to see. See the tabs illustrated in the following screen example:



The Project Details page opens with the Procedures tab highlighted, so the first table you see is the Procedures table.

You can use the DSL Export ( ) button to download the objects to a DSL file.

Procedures tab

The following links and actions are available in the Procedures table:

- **Create Procedure**(at the top of the table)—Opens the New Procedure page to create another procedure.
- Click on a procedure name (first column) to go to the Procedure Details page for that procedure.
- Action column—Click any of the links (**Run, Edit, Copy, or Delete**) to perform that function for the procedure listed in that row.

> **Note:** The Run link provides two choices when you hover the mouse over the down-arrow. Select **Run Immediately** or **Run… .**

  - **Run Immediately**—Runs the procedure "as-is" immediately.
  - **Run…**—Opens the Run Procedure web page where you can modify any existing parameters for this procedure or set an existing credential.

Workflow Definitions tab

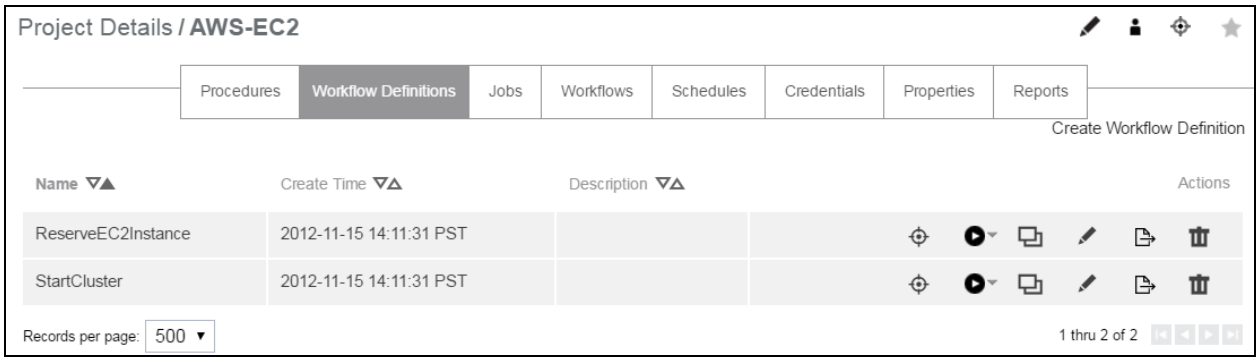
This tab provides a table listing all defined workflow definitions and includes the following functionality:

- Click the **Create Workflow Definition** link to go to the New Workflow Definition page to define additional definitions.

- Name column—Click a workflow definition name to go to the Workflow Definition Details page (for that workflow definition.
- Action column
  - **Run**—Use the down arrow to select **Run...** or **Run Immediately**

    Selecting **Run...** takes you to the Run Workflow page where you can set the starting state for the workflow. Selecting **Run Immediately** runs the workflow "as-is" immediately.
  - **Copy**—Make an exact copy of an existing workflow definition.
  - **Edit**—Edit an existing workflow definition or just to change the name of the copy you created.
  - **Delete**—Deletes an existing workflow definition on the same row.

The following screen example illustrates the Workflow Definitions table:

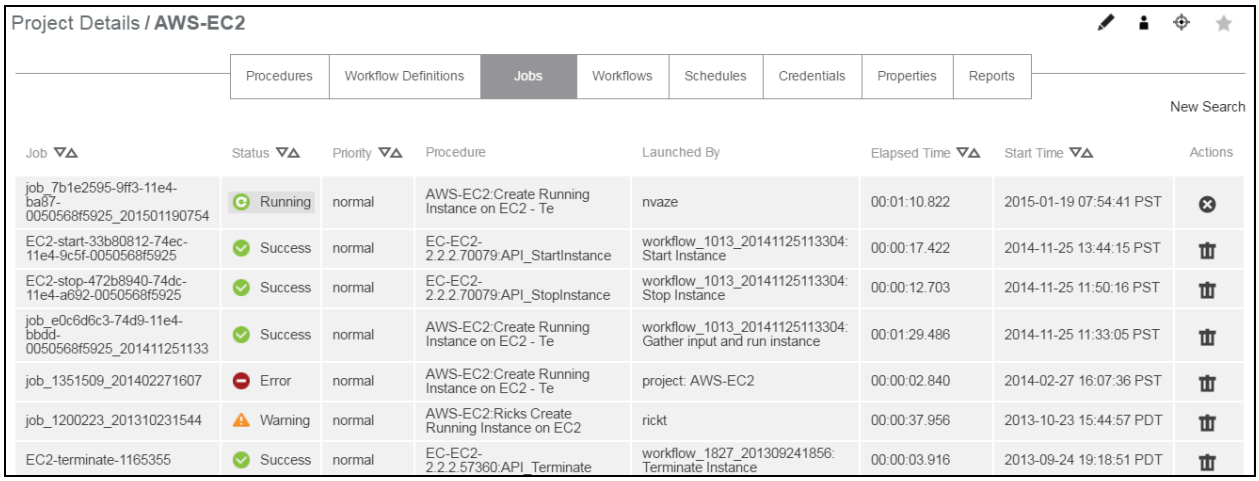


You can use the DSL Export ( ) button to download the objects as a DSL file.

Jobs tab

This tab provides a project-specific job table. Selecting the Jobs tab on the Project Details page displays only those jobs for the project you selected.

This table is a subset of the main Jobs tab that displays *all* jobs, regardless of the project of origin, and all Jobs page functionality is basically the same on either Jobs tab.



The following links and actions are available on this Jobs page:

- Job—Sort the column alphabetically or click on a job name to go to that job's Job Details page.
- Status—Sort this column by Running, Success, Warning, Error, or Aborted.
- Priority—Display the job's priority set by a "run procedure" command or by the job's schedule.
- Procedure—Click a Procedure name to go to the Procedure Details page for that procedure.
- Launched By—Click a name in this column to go to the page for the schedule (Edit Schedule page) or the user that ran the job.
- Elapsed Time—Sort the time values from longest to shortest time, or the reverse.
- Start Time—Sort this column from the start time of the first job to the start time of the most recent job, or the reverse.
- Actions—Use this column to Abort a running job or Delete a completed job.

  If your job has finished running, you will not see the Abort link. Aborting a job requires the execute privilege on the job—not just the modify privilege.

Select either **OK** or **Force** to abort the job, or **Cancel** if you change your mind.

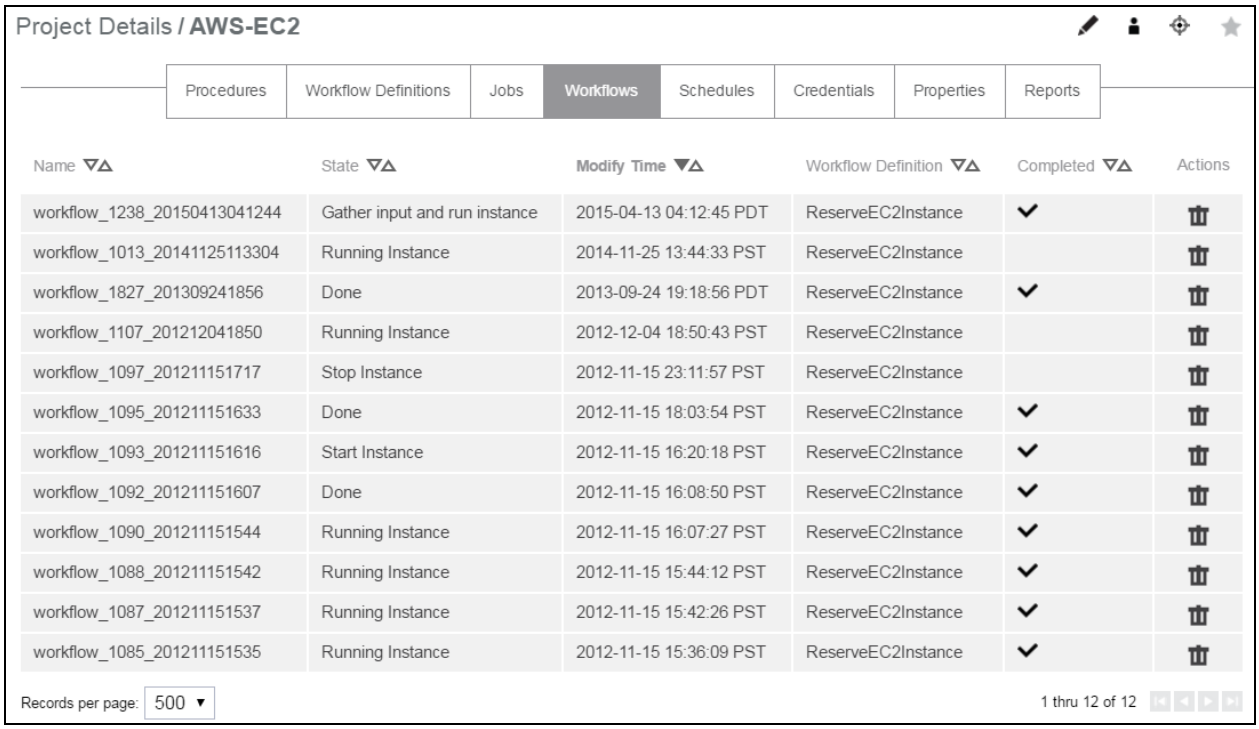
Note: Deleting a job on this page causes removal of job information from the ElectricFlow database, but information in the job's on-disk workspace area is not affected. You must delete workspace information manually.

Workflows tab

The Workflows tab provides the following functionality:

- Name column—Click on a workflow name to go to its Workflow Details page.
- State column—Click on a state name to go to its State Details page.
- Modify Time—This time value represents the most recent workflow activity.
- Workflow Definition column—Click on a workflow definition name to go to the Workflow Definition Details page.
- Completed column—A check mark in this column identifies this workflow as completed.
- Actions column—Click the Delete link to remove the workflow in that row.

The following screen example illustrates the Workflows table:



For more information about workflows, see the Workflow Help topic.

Schedules tab

The Schedules tab provides the following functionality:

- To create a new schedule for this project, click either the Schedule link to create a new standard schedule to run at specific and/or days or the CI Configuration link to setup a new continuous

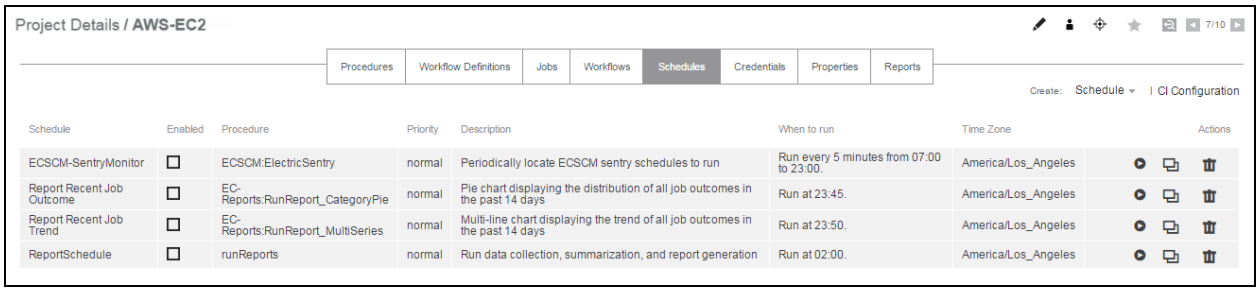
integration schedule.

- Choosing "Schedule" displays the New Schedule page.
- Choosing "CI Configuration" displays the New CI Configuration page.

Table column descriptions

- To edit an existing schedule, click on the schedule name (first column).
- Select the check box in the Enabled column to enable or disable the schedule in that row.
- The Priority column displays the job's priority you selected while creating the schedule for this procedure.
- Action column links
  - **Run**—Runs the schedule with the permissions of the logged in user.
    For example, if this schedule is timed to run every day at 11:00 pm, you might decide to run the schedule at 6:00 pm on a particular Thursday, without changing the regular schedule settings. However, if you do not have the appropriate permissions to run this schedule, it will fail.
  - **Copy**—Makes a copy of the schedule on that row.
  - **Delete**—Deletes the schedule in that row.

The following screen example illustrates the Schedules table:



For information on creating a standard schedule, see the Schedule - create new or edit existing schedule Help topic.

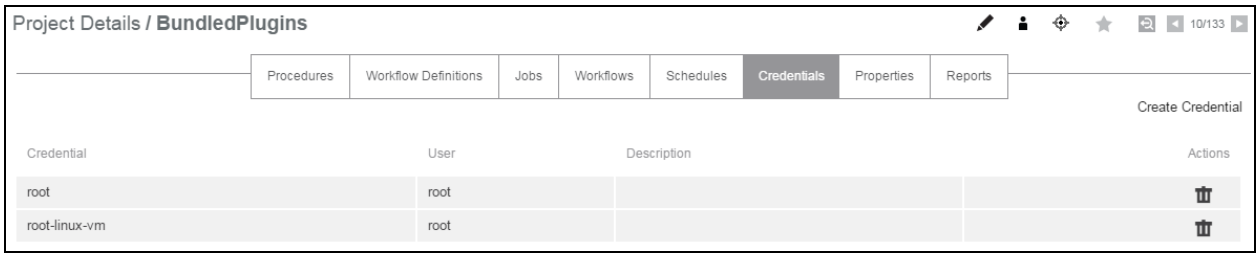
For information about CI Manager and adding a configuration, click the Home tab, the Continuous Integration subtab, then click the Help link in the upper-right corner of the page.

Credentials tab

The Credentials tab provides the following functionality:

- To create a new credential for this project, click the **Create Credential** link at the top of the table.
- To edit an existing credential, click on the credential name (first column).
- Action column—The Delete link deletes the credential on that row.

The following screen example illustrates the Credentials table:



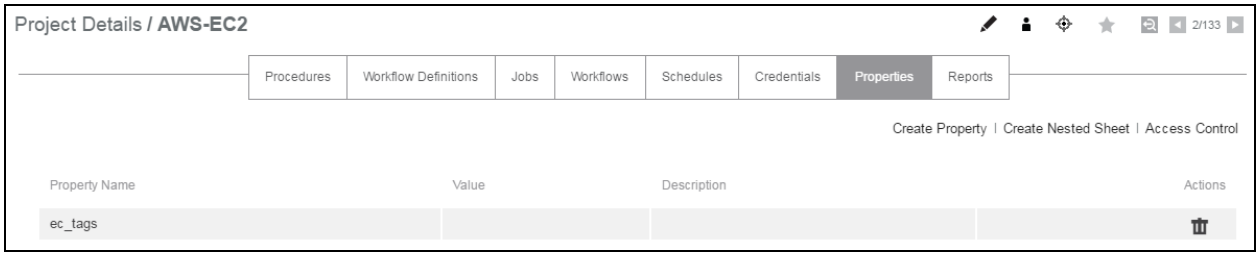
For information about credentials and access control, see the Credentials and User Impersonation or Access Control Help topics.

Properties tab

The Properties tab provides the following functionality:

- To create a new property for this project, click the **Create Property** or **Create Nested Sheet** link.
- To view or change privileges on the property sheet, click the **Access Control** link.
- To edit an existing property, click on the property name (first column).
- Action column—The Delete link deletes the property on that row.

The following screen example illustrates the Properties table:



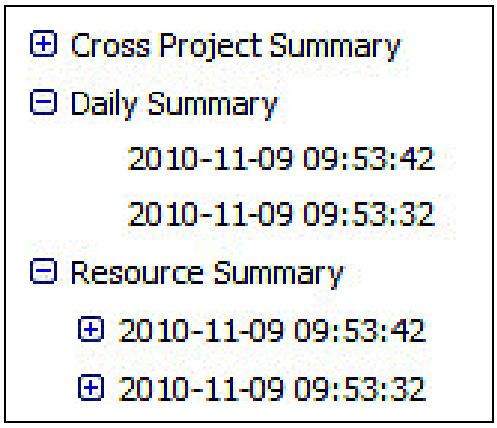
For more information about properties, see the Properties Help topic.

Reports tab

Selecting this subtab yields different results, depending on whether or not you have installed the additional ElectricFlow-supplied optional reports or have created custom reports for the specific information you need. Generally, if you have not yet configured additional reports, you will see the three installed, default ElectricFlow reports (Cross Project Summary, Daily Summary, and Resource Summary) as illustrated in the screen example below.

- When a report runs, the report "folder" is populated with the completed report. Reports are listed by date and time.
- Each individual report "date/time" entry is a link to see the full report.
- To create a new report, click the **Create Report** link to go to the web page to create a Multiple Series, Category, Count Over Time, or Procedure Usage report.

This is an example of the information on the Reports page :



For more information about reports, see the Reports Help topic series.

Tip:

- To rename this project, click the **Edit** link at the top of this page and enter a new name.
- To rename a procedure in this project, click the **Edit** link for the procedure (on this page) and enter a new name.

Run Procedure

Using the Run Procedure page

- You may see a message informing you that no parameters were defined for this procedure. If this is the case, click **Run** to run the procedure.
- If parameters were defined for this procedure, you will see specific fields based on those parameters. You may make changes to any field you choose if you want to run this procedure under different parameters at this time. However, any changes you make are specific to this run procedure only. Your changes on this web page will not affect previously defined parameters for this procedure.
  Click **Run** to run the procedure.
- You may see a "Required" field. To run the procedure, you must enter the required parameter previously defined on the Procedure Details page.

Click **Run** to run the procedure.

Advanced section

Priority:

Use the drop-down menu to select the priority for this run procedure. Available priorities are: low, normal (default), high, or highest.

- You can select the job's priority here on the Run Procedure web page or when you schedule the procedure to run at a regular time or interval. When a job is launched, its priority cannot be changed.

- Priorities take effect when two or more job steps in different jobs are waiting for the same resource. When the resource is available, it will be used by the job step that belongs to the job with the highest priority. If the priority level is the same, the resource will be used by the job step that belongs to the job with the lowest job ID number. If the job steps are in the same job, the resource will be used first by the step with the lowest job step ID number.
- If using ectool, the priority can be set by passing "`--priority=<low|normal|high|highest>`" to a `runProcedure`, `createSchedule`, or `modifySchedule` operation.

> **Note:** To raise the job priority level above "normal," the user who launches the procedure must have execute permission on the system-level access control list for priorities.

Impersonation:

Select one of the following:

- **Use pre-defined credential**—(Default option) Use this option if you have not defined a credential for this procedure.
  ElectricFlow looks for a credential first in the top-level procedure and then in its project. If it finds a credential, it uses the login from that credential as the default for the job. If there is no credential in either place, by default, the job's steps run as the user under which the agent for the step is running.
- **Use specific credential**—If you select this option, more fields are displayed for you to choose a particular credential and enter its name from those defined for the project containing the procedure. The login user from that credential will be used as the default for the job.
- **Use a specific user**—If you select this option, more fields are displayed to enter a User Name and Password. The specified login is used as the default for the job.

Click **Run** to run the procedure.

For more information on credentials and user impersonation, see the Credentials and User Impersonation Help topic.

Schedule—create new or edit existing schedule

Use this page to create or edit a standard schedule that specifies a set of times and days when a particular procedure should run. If you need to create a continuous build integration instead, see the Help information available from the Continuous Integration Dashboard web page.

For any schedule, click the "star" icon to add this schedule to the Shortcut section on your Home page.

To create a new standard schedule

Enter information into the fields as follows:

Field name	Description
General section	
Name	Enter a Schedule name. This must be a unique name among other schedule names in this project.

Field name	Description
Procedure	Displays the current project and procedure. Click **Change** to use the pop-up menu to select a new project or procedure.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Parameters section	
If the procedure has parameters, enter information in their values in this section. The procedure cannot execute unless all required parameters are provided. If you move the mouse over a field for a particular parameter, inline Help displays the description of that parameter if one was provided in the procedure's definition.	
	Click this icon to displays fields to add one or more parameters.
Frequency section	
Run at ???	This is a summary section: As you make selections for the following Days and Repetition sections, a summary of what you have selected appears here.
Days	Select weekdays, month days, or create a custom frequency when you want the schedule to run. The "day" applies to the start time if the time range spans two days, that is, if the time range crosses the midnight boundary.
Repetition	• **Run Once**—The schedule will run once per selected day at the time you specify. • **Run Every**—The procedure will execute repeatedly during the specified time range. For example, if you select a time range from 11:00-14:00 and an interval of 40 minutes, the procedure will execute 5 times on each of the selected days: at 11:00am, 11:40am, 12:20pm, 1:00pm, and 1:40pm. • **Run Continuously**—As soon as one job ends, the scheduler will trigger immediately to run the job again.
Advanced section	
Enabled	If this box is "checked," the schedule will run. You can disable this schedule whenever necessary.

Field name	Description
Misfire Policy	Choose between two options to manage how a schedule resumes in cases where the normal scheduled time is interrupted. The server may not be able to trigger a job at the scheduled time for one of several reasons: • A running job extends past the next scheduled runtime. • The server cannot find enough available resources, and a delay occurs. • The server is down or disconnected from the network for a period that overlaps one or more scheduled times. If any of these situations occur, the misfire policy defines two possible responses: • `skip`—All misfires are ignored and the job runs at the next scheduled time. • `run once`—After one or more misfires, the job runs at the soonest time that occurs within an active region. For example, at server startup a scheduled job will not trigger immediately on startup.
Time Zone	Default is the time zone of the ElectricFlow server, or select a country from the first pull-down menu, then select an area from the second pull-down menu. Another option: Select "other" from the first pull-down menu, then select GMT [or another selection] from the second pull-down list.
Priority	Use the pull-down menu to select one of four priorities for this procedure. Also, you can select the job's priority on the Run Procedure web page. When a job is launched, its priority cannot be changed. Priorities take effect when two or more job steps in different jobs are waiting for the same resource. When the resource is available, it will be used by the job step that belongs to the job with the highest priority. If the priority level is the same, the resource will be used by the job step that belongs to the job with the lowest job ID number. If the job steps are in the same job, the resource will be used first by the step with the lowest job step ID number. If using ectool, the priority can be set by passing `"--priority=<low|normal|high|highest>"` to a `runProcedure`, `createSchedule`, or `modifySchedule` operation. **Note:** Schedules are launched by the project principal user. To raise the priority above "normal," this user must have execute permission on the system-level access control list for priorities.

Field name	Description
Impersonation Credential section	If you select a credential for the schedule, it will be used as the default for jobs run from the schedule. For example, job steps will run under the login account specified in the credential. However, the schedule's credential is low in priority: it will be used only if there is no credential specified in the procedure or its project. Also, individual steps can override the credential with a specific credential for that step. For more information, see Credentials and User Impersonation. Impersonation credentials are used to set the top-level impersonation credential for a job. If specified, the impersonation credential (on the job) is used as the default impersonation credential for all steps in the job.
Credential Project	The name of the credential for the project. By default, the current project is selected. Enter a different project name or select **Browse** to find this name.
Credential Name	The name of the credential for the project. You might need to browse for this name.

Click **OK** to save your schedule information.

To create a new continuous integration "schedule"

If you selected the **CI Configuration** link, the New CI Configuration page is displayed. For information on populating these fields, see the "Continuous Integration Manager" plugin Help topic—click the **Help** link in the upper-right corner of the New CI Configuration page.

See the Standard Schedule information (above) to enter information in the fields common to both types of schedules.

To edit an existing schedule

Modify any of the information you previously specified. The following additional links are available at the top of the page:

- **Save Configuration**—The information in this schedule will be copied into a new Job Configuration and displayed on your Home page for easy access.
- **Access Control**—Set permissions for this schedule.

Attached Credentials

Click the **Attached Credential** link if you need to attach an existing credential to this schedule. If you need to create a new credential for this schedule, return to the Project Details page.

Custom Schedule Properties

Select the **Create Properties**, **Create Nested Sheet**, or **Access Control** links if you want to assign properties to this schedule. For more information on Properties or Access Control, see the following main Help topics: Properties, Access Control

Click **OK** to save your modifications for this schedule.

Credentials—create new or modify existing credential

To create a new credential

Enter information into the fields as follows:

Field Name	Description
Name	Enter a unique name of the credential object.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
User Name	Enter the user name you choose to use for login.
Password	Enter the password that identifies the user account.

Click **OK** to save your new credential information.

To edit a credential

Modify any previously entered credential information and click **OK** to save your changes.

Use the **Access Control** link to add or change existing permissions.

You can add Custom Credential Properties for this credential also. Click the **Create Property, Create Nested Sheet,** or **Access Control** link in this section if you need to specify any of these items.

For more information, see the following Help topics:

Credentials and User Impersonation

Properties

Access Control

Property—create new or edit existing property

Use this pop-up window to create or modify a custom property attached to an object.

Enter information into the fields as follows:

Field Name	Description
Name	The property name can be an arbitrary text string, but unless you are an experienced ElectricFlow user avoid using slashes and brackets as a part of the property name. **Note:** The names "properties" and "project" are not valid property names.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with <html> ... </html> tags. Allowable HTML tags are <a>, <b>, , , <dl>, <font>, <i>, , , <p>, <pre>, , <style>, <table>, <tc>, <td>, <th>, <tr>, and .
Value	This value can be an arbitrary text string.
Expandable	Do not select this check box if you do not want to allow property expansion.

Click **OK** to save your new or modified property information. For more information on creating and using properties, see the Properties Help topic.

Nested Property Sheet

Use this pop-up window to define or modify a nested property sheet.

1. Enter information in the fields as follows:

Field Name	Description
Name	The property name can be an arbitrary text string.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with <html> ... </html> tags. Allowable HTML tags are <a>, <b>, , , <dl>, <font>, <i>, , , <p>, <pre>, , <style>, <table>, <tc>, <td>, <th>, <tr>, and .

2. Click **OK** to save your new or modified nested property sheet information.

For more information on creating and using properties, see the Properties Help topic.

Reports

If you have upgraded ElectricFlow, this page displays historical reports (Cross Project Summary, Daily Summary, and Resource Summary) from your previous ElectricFlow version. These reports are no longer updated.

Click the Historic/web server local reports link to see existing custom reports.

- Cross Project Summary—30-day report summary
- Daily Summary—Daily report showing all the procedures that ran
- Resource Summary—Resource usage for a specific day
- Historic/web server local reports

For more information, see the Reports Help topic.

Create a new report

Use this page to create a new report. Select one of the report-type tabs to begin.

Multiple Series Report

Field and pull-down menu descriptions:

Field/Menu Name	Description
Report Title	This is your report title. Type over the default report name, choosing any unique name for your report. **Important:** Do not use special characters in the report title. Some examples of special characters are : / ? # [ ] @ !, $ & ' ( ) * + , ; =
Saved Filter	• Project—Click your mouse inside this field to see a list of projects from which to make a selection. • Filter—After selecting a project, this drop-down menu will contain any available filters for this project.
Time Period	Use the drop-down menu to select the time period.
Create thumbnail?	Check this box if you would like to make a thumbnail view of this report available on your Home page.
Object Type	Use the drop-down menu to select an object.

Field/Menu Name	Description
Table column choices	• Chart Type—Use the drop-down menu to choose the chart type. • Function—Use the drop-down menu to choose the function you need. • Property Name—Use the default property value or delete the text and click your mouse in the blank field to see a list of possible properties. • Display Name—You can choose a different unique Display Name. • Stacked—Select this checkbox to see your report results "stacked" versus overlaid. • The "X" icon—Click this icon to delete any row you no longer need.
Add Series button	Select the **Add Series** button if you would like to create additional table entries for additional report information.
Chart Options	Use the down-arrow to adjust the Time Grouping and see the defaults for the X and Y axis.
Table Options	Use the down-arrow to adjust the Time Grouping and see the default for the Time Column label.
Advanced	Use the down-arrow to select a Locale or add a Credential. Delete the Resource if you choose, then click your mouse in the blank field to see a list of other resources you might prefer to use.
Run Report button	Use this button to go to the Job Details page to run the report.
Create Schedule button	This button displays a box with a default schedule name that you can change if you choose. Click **OK** to go to the Job Details page.
Cancel button	Use this bottom to cancel the create report operation.

Category Report

Field and pull-down menu descriptions:

Field/Menu Names	Description
Report Title	This is your report title. Type over the default report name, choosing any unique name for your report. **Important:** Do not use special characters in the report title. Some examples of special characters are : / ? # [ ] @ !, $ & ' ( ) * + , ; =

Field/Menu Names	Description
Saved Filter	• Project—Click your mouse inside this field to see a list of projects from which to make a selection. • Filter—After selecting a project, this drop-down menu will contain any available filters for this project.
Time Period	Use the drop-down menu to select the time period.
Create thumbnail?	Check this box if you would like to make a thumbnail view of this report available on your Home page.
Object Type	Use the drop-down menu to select an object.
Object Property	You can delete the default value and click your mouse inside the blank field to see a list of properties from which to choose.
Advanced	Use the down-arrow to select a Locale or add a Credential. Delete the Resource if you choose, then click your mouse in the blank field to see a list of other resources you might prefer to use.
Run Report button	Use this button to go to the Job Details page to run the report.
Create Schedule button	This button displays box with a default schedule name that you can change if you choose. Click **OK** to go to the Job Details page.
Cancel button	Use this bottom to cancel the create report operation.

Count Over Time Report

Field and pull-down menu descriptions:

Field/Menu Name	Description
Report Title	This is your report title. Type over the default report name, choosing any unique name for your report. **Important:** Do not use special characters in the report title. Some examples of special characters are : / ? # [ ] @ !, $ & ' ( ) * + , ; =
Saved Filter	• Project—Click your mouse inside this field to see a list of projects from which to make a selection. • Filter—After selecting a project, this drop-down menu will contain any available filters for this project.

Field/Menu Name	Description
Time Period	Use the drop-down menu to select the time period.
Create thumbnail?	Check this box if you would like to make a thumbnail view of this report available on your Home page.
Object Type	Use the drop-down menu to select an object.
Chart Options	Use the down-arrow to adjust the Time Grouping and see the defaults for the X and Y axis.
Table Options	Use the down-arrow to adjust the Time Grouping and see the default for the Time Column label.
Advanced	Use the down-arrow to select a Locale or add a Credential. Delete the Resource if you choose, then click your mouse in the blank field to see a list of other resources you might prefer to use.
Run Report button	Use this button to go to the Job Details page to run the report.
Create Schedule button	This button displays box with a default schedule name that you can change if you choose. Click **OK** to go to the Job Details page.
Cancel button	Use this bottom to cancel the create report operation.

Procedure Usage Report

Field and pull-down menu descriptions:

Field/Menu Name	Description
Report Title	This is your report title. Type over the default report name, choosing any unique name for your report. **Important:** Do not use special characters in the report title. Some examples of special characters are : / ? # [ ] @ !, $ & ' ( ) * + , ; =

Field/Menu Name	Description
Project	Click your mouse inside this blank field to see a list of projects. This does not have to be the current project. You can generate reports that show when one project's procedures call procedures in another project. You can also use the '%' wildcard, for example, Project: % or Project: D%
Procedure pattern	Enter a SQL string match pattern for procedures to include in the report. The default % will use all procedures.
Advanced	Use the down-arrow to select a Locale or add a Credential. Delete the Resource if you choose, then click your mouse in the blank field to see a list of other resources you might prefer to use.
Run Report button	Use this button to go to the Job Details page to run the report.
Create Schedule button	This button displays box with a default schedule name that you can change if you choose. Click **OK** to go to the Job Details page.
Cancel button	Use this bottom to cancel the create report operation.

Resources

The **Cloud** > **Resources** page displays all resources on this ElectricFlow server and provides easy access to resource configuration and all other resource management functionality.

- Each resource has a logical name—a unique name to distinguish this resource from other resources.
- Each resource refers to an agent machine by its host name.
- Each resource can be assigned to one or more pools.

  A *pool* is a group of interchangeable resources. For example, a pool of Windows servers.

  If you name a pool in a procedure step, ElectricFlow can assign any resource in the pool to that step, which allows ElectricFlow to choose a lightly loaded resource. You can change the resources in a pool without modifying procedures that use the pool.

- Each resource can be assigned to a zone. A default zone is created during installation, and all resources are members of that zone until they are assigned to a different zone.

  A *zone* is a collection of agents. Every agent, and all resources defined on that agent, belong to only one zone. When ElectricFlow is installed, a *default* zone is created. If a zone is deleted, all agents in that zone are moved to the *default* zone. The ElectricFlow server resides in the default zone.

  > **Note:** The default zone cannot be deleted.

- Several resources can correspond to the same physical host or agent machine.
- When you specify a resource name in a procedure step, the step executes on that resource.

Supported Resource Categories

- **Standard**-This category specifies a machine running the ElectricFlow agent on one of the supported agent platforms, as specified in the *ElectricFlow 8.5Installation Guide* at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html.
- **Proxy**—This category requires SSH keys for authentication. You can create proxy resources (agents and targets) for ElectricFlow to use on other remote platforms or hosts that exist in your environment.
  - Proxy *agent*—This is an agent on a supported Linux or Windows platform, used to proxy commands to an otherwise unsupported platform. A proxy agent is an ElectricFlow agent, channeling to a proxy target.
  - Proxy *target* —This is a machine on an unsupported platform that can run commands through an SSH server. Proxy targets have limitations, such as the inability to work with plugins or communicate with ectool commands.

Help topic "Best Practice": You should review this entire Help to become familiar with resource management features and functionality. Later, when using the Resources page, use the following quick links to go to the Help section that you need to review.

- Icons, Links, and Buttons Above the Table
- Table View—Column Descriptions
- Grid View
- Filters Pane
- New Resource Panel
- New Proxy Resource Panel
- Edit Resource Panel
- Edit Proxy Resource Panel
- Resource Details Panel
- Switching a Non-Trusted Agent to Trusted

Resource Page Information and Functions

Immediately after the *Resources* title above the table, you can see how many licensed resources are in use.

Links, Icons, and Buttons Above the Table

Hover your mouse over an icon or button to see the available actions. You can select one of these actions on a single resource or on multiple resources (selected using the check box in the first column) to perform the action on one or more resources at the same time.

- Select one or more resources from the table, and then select the button to perform the task or action.

- **Exception**: When you click the **Action** icon and select **Create Resource** or **Create Proxy Resource**, a panel opens where you can create one of these new objects. You can create only one object at a time.

Icon or Link Name	Description
	Select one of these actions: • **Create Resource**—The **New Resource** panel opens. For information about the fields, see New Resource. • **Create Proxy Resource**—The **New Proxy Resource** panel opens. For information about the fields, see New Proxy Resource. • **Install Resource(s)**—A dialog box opens where you enter information about the agents to install. For more information, see the "Installing or Upgrading Remote Agents" section in the "Installing ElectricFlow" chapter of the *ElectricFlow 8.5 Installation Guide* at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html. The section describes how to use the ElectricFlow Centralized Agent Management (CAM) feature. If the **Install Resource(s)** menu option is not visible, you must log out and then log in as a user with the required permissions. For details about required permissions, see the "Permissions for Installing or Upgrading Remote Agents" section in the "Installing or Upgrading Remote Agents" chapter of the *ElectricFlow Installation Guide* at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html. **Note:** If you are upgrading one or more agents or resources, the label on this option is **Upgrade Resource(s)**. If the **Upgrade Resource(s)** menu option is not visible, you must log out and then log in as a user with the required permissions. For details about required permissions, see the "Permissions for Installing or Upgrading Remote Agents" section in the "Installing or Upgrading Remote Agents" chapter of the *ElectricFlow 8.5 Installation Guide* at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html. **Note:** If the EC-AgentManagement plugin is not installed, you will not see this option.
	**Copy Resource**—Makes a duplicate copy of one or more resources that you selected.

Icon or Link Name	Description
	**Delete Resource**—Deletes one or more resources that you selected. **Note:** If a resource is on a "gateway agent," the gateway will be deleted.
	**Ping Resource**—ElectricFlow pings one or more resources to check if they are available. When ElectricFlow pings a resource, it sends a message to the agent (for the resource) to make sure the agent is alive and running a version of software compatible with the ElectricFlow server. After the ping completes (or fails), the page refreshes to reflect the current resource state. A gateway must exist before you ping a resource in a remote zone.
	**Enable Resource**—Enables one or more resources that you selected in the check boxes in the first column..
	**Disable Resource**—Disables one or more resources that you selected in the check boxes in the first column.

	Select one of these actions:
	
	• When you click the button with no resources selected, the *Select some resources to edit* message appears.
	
	• When you select resources and click this button, these options appear:
	
	  • **Add to Pool**—Adds the selected resources to a pool.
	  • **Remove from Pool**—This is available only if all the selected resources have at least one pool in common. If not, this link becomes **Remove from Pool**<*name*>.
	  • **Move to Zone**—Moves the selected resources to a zone.
	  • **Set Trusted**—This is not available if the selected resources are already *trusted*. If not, all the selected resource will be changed to *trusted*.
	  • **Unset Trusted**—This is available only if all the selected resources are trusted. If not, the selected resources will be changed to *untrusted*.

Icon or Link Name	Description
 or	Select this button to add this Resources page to your Home page for one-click access to return to this page. If the icon is yellow, the page is already accessible from your Home page .
New Search link	Use this link to go to the Searching and Filtering page if you need to search for a particular object or set of objects. For example, job, resource, artifact, workflow, and so on. Use the "back" button to return to the Resources page.

Table View—Column Descriptions

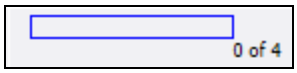
Not all resource configuration information appears within the table, but you can see all other information about the resource when you edit the resource or access other slide-out panels. For example, the action to see or add access control privileges to a resource is available on the Edit Resource panel or from the **Access Control** link on that panel.

Note: Most column headings can be sorted. Click on a column heading to re-sort the information in that column.

Column name / icon	Description
	Select this check box to enable the corresponding resource in that row for a single or batch action. For example, deleting or enabling multiple resources at the same time is a batch action. If you select this icon in the table header row, it will select or deselect all resources in the table.
	Status of the agent. If an agent machine is not available, any resources associated with that host will not be available. If an agent is down, the ElectricFlow server is not able to provide additional information about the cause.

Column name / icon	Description
Name	Name of the resource. The icon to the left of the resource name indicates whether this resource is *enabled* or *disabled*. These icons are: Enabled  or Disabled  . If a resource is disabled, no job steps are assigned to it. If a step requests a particular resource and that resource is disabled, the step waits until the resource becomes enabled again. If the resource is in a pool, ElectricFlow uses any enabled resource in the pool to satisfy the request for the resource. Other actions: • To change the enabled or disabled icon, select the check box (in first column) in the row for one or more resources, and then select the enabled or disabled icon above the table. Select multiple resources if this is a batch change. **Note:** You can change the resource enable/disable specification from the Edit Resource panel too, but this is an individual operation for one particular resource. • Click on a resource name to see the Resource Details panel—this panel displays how your resources is configured currently. To make changes, click the **Edit** link to go to the Edit Resource panel.
Pools	A list of pool names where this resource is a member.
Type	A resource can be *static* or *dynamic*. • A static resource is part of your system or network (not in the cloud), such as a server, database, or agent machine. • A dynamic resource is a cloud resource that can be provisioned and later spun up on-demand when an application or microservice is deployed.
Resource Template Name	The name of the resource template to which the resource is applied. Only dynamic resources can be applied to a resource template.
Time running	If the resource is a dynamic resource, how long it has been running.

Column name / icon	Description
Job Status	Indicates the current job status with the following icons:  *Running job*—A job is still running, which means the resource is currently being used for one or more jobs. The names of all jobs using the resource are displayed next to the icon.  *Success*—The job running on this resource completed successfully.  *Error*—The job running on this resource encountered an error and may or may not have completed. One or more error messages will be listed in this column.  *Warning*—The job running on this resource encountered a warning and may or may not have completed. One or more warning messages will be listed in this column.  *Busy*—ElectricFlow is attempting to refresh the job status, but the resource is not responding or the UI cannot parse the response. Changing the value of 'Agent Host Name' to the IP address can sometimes resolve this issue.
Last Used	Date and time when the resource was last used. The value of this field corresponds to when the job was scheduled to run on this resource.
Last Job	Last job that was scheduled to run on this resource. If the job is deleted from the system, then this field shows "Job deleted."
Create Date	Date and time when the resource was created.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Zone	The name of the zone where this resource is a member.
HTTPS & Host	The "Connection Type" you specified when this resource was created.

Column name / icon	Description
Step Load	Resource *load* is indicated by one of the following: •  —Indicates 1 step is using this resource and an unlimited number of steps can use this resource at the same time. If 5 steps are using this resource, you would see "5 of unlimited". •  —Indicates that no steps are running on this resource and the step limit is set to 4. The bar indicates graphically how close you are to the step limit. **Tip:** Use the Create/Edit Resource panels to define the number of steps that can run on a resource. Setting the Step Limit to "0" or leaving the field blank defaults to NO step limit.
Actions	Actions you can perform on the resource, including: • **Copy**—Create a new resource by copying this resource • **Delete**—Delete this resource • **Tear Down**—Remove (decommission) the dynamic resource from a dynamic environment  **Track Changes** --Open the Change History for the resource

Grid View

Two icons above the Filters pane:

- The first icon (on the left) is the default *Table view* icon.
- The second icon (on the right) displays the resource *Grid view*.
  Use the drop-down menus to filter the Grid view:

- Grouping — Select the resource "grouping" you want to see—choose Pool, Job, Operating System, Platform, Proxy Agent, Version, or Zone.

  For example, if you have multiple resource pools defined and chose Pool, you will see groups labeled with your various pool names and a group (No Pool) for resources not included in a pool.

- Coloring — Select what you want to view in each group—choose Platform, Step Count/Limit, Status, Held Exclusive, Version, or Operating System to sort your group.

  Colors are dynamically assigned per the view you select. Within a single view, the same color "square" implies similar information.

- Hovering your mouse over a grid "square" provides more detailed information about that resource.

Filters pane

Use the Filters pane to filter resources shown in either the Table or Grid view. The fields in the Filter pane correspond to columns in the Table view.

Enter the Filter criteria as follows:

Actions / Field	Description
Save Filters	Click this link to save your filter criteria. A dialog box appears so you can name your filter. For easy, repeat access, filters are saved in a "saved filters" box above the link. When you select a saved filter name, the Filter pane fields populate automatically.
Reset	Use this link to clear all entries in the Filter pane.
Quick Search	Enter any text for your search. For example, enter a host name. You do not need to enter a full name or text string—this field filters on a partial text entry, sometimes 2-3 characters are all you might need.
Status	The status for the agent machine.
Both Enabled/Disabled	Default is ***both*** enabled and disabled resources. Use the drop-down arrow to choose enabled or disabled only.
Pools	Name of a pool. The pool name must be entered exactly—a shortened version of the pool name will fail.
Hosts	List one or more full host names as shown in the Table view "HTTPS & Host" column (each name separated by a space). Entering a partial name will not yield a successful result.
Step Limit	Choose the step limit corresponding to your desired search.

Actions / Field	Description
Proxy Agent	Select No if not searching for a proxy agent or Yes if you are trying to locate a proxy agent.
Filter	Select this button after you complete the fields/selections to define your filter criteria.

New Resource Panel

To define a new standard resource, enter information in the following fields:

Field	Description
Name	Enter a unique name for this resource. ***Do not use*** "spaces" in a resource name. This is the name used to select the resource for a job step—It need not be the same as the host name for the machine.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Agent Host Name	Enter the domain name or IP address of the agent machine for this resource. This is the name all other machines in this agent/resource's zone will use to communicate with this agent host.
Agent Port Number	Enter the port number to use when connecting to the agent for the resource, Default port is 7800.
Default Workspace	Enter the workspace name or leave blank to use the local, default workspace. The workspace specification can be overridden at the project, procedure, or step level. If you specify a workspace name here, it will be used as the default for all job steps that run on this resource. See the Workspaces Help topic in the Automation Platform for more information.
Pool(s)	A comma-separated list of arbitrary names indicating the pools associated with this resource.

Field	Description
Host Type	Use the drop-down menu to select the host type: • **Concurrent** • **Registered** Depending on the license installed on the ElectricFlow server, this field may appear. • If the license on the server is a concurrent resource license, the host type defaults to **Concurrent** and this field does not appear. • If the license on the server is a registered host license, the host type defaults to **Registered** and this field does not appear. • If the license on the server is a mixed-mode license (concurrent resources and registered hosts), you must specify the host type when adding a resource.

Field	Description

Default Shell	The name of the shell program used to execute the step's commands on a resource. For example, using `sh` or `cmd /c`, the agent saves thecommand block to a temporary file with a `.cmd` extension and runs it: `sh foo.cmd` or `cmd /c foo.cmd`.
	
	If you do not specify a shell on a step, at step run-time the server looks at the resource shell. If a resource shell is not set, the shell line used by the agent is platform dependent:
	Windows: `cmd /q /c "{0}.cmd"`
	UNIX: `sh -e "{0}.cmd"`
	
	When you specify a shell (in the step or resource), and omit the `cmd-file` marker, the agent notices the omission and takes the correct action. For example: a user specifies `sh -x`. The agent converts this to `sh -x "{0}.cmd"`.
	
	Two alternate forms of shell syntax where ElectricFlow uses a "marker," `{0}`, as a placeholder for the command file argument:
	• *<myShell>* `{0}` *<potential extra shell args>*
	In this example, the command file is not meant to be the last argument in the final command line.
	For example, `mysql -e "source {0}"`
	This shell example runs the `mysql` command against this step's command containing `sql`.
	• *<myShell>*`{0}`*<.file extension> <potential extra shell args>*
	In this example, the shell requires the command file to end in an extension other than `.cmd`.
	For example, `powershell "& '{0}.ps1'"`
	This shell example runs Microsoft PowerShell against this step's command containing PowerShell commands.
	
	Note:
	• When the agent parses the shell, it will parse the extension as everything after `{0}.` until it sees a space or non-alphanumeric character.
	• If your script uses International characters (non-ASCII), add the following block to the top of your `ec-perl` command block:
	`use utf8;`
	`ElectricCommander::initEncodings`

Field	Description
Step Limit	The maximum number of steps that can execute simultaneously on this resource—a step limit applies to a particular resource only, not to its underlying host. For example, if you define two resources, resource1 with a step limit of 5 and resource2 with a step limit of 1, both specifying the same host machine, it is possible for a total of 6 steps to execute simultaneously on the underlying host. The Resource Details panel displays this information. **Note:** Setting the Step Limit to "0" or leaving the field blank defaults to *no* step limit.
Artifact Cache Directory	The directory on this resource's agent host from which artifact versions are retrieved and made available to job steps. Enter an absolute path to the resource containing this cache directory.
Zone	The name of the zone where this resource is or will be a member—a zone is a top-level network containing one or more mutually accessible resources.
Repository Names	A "new-line" delimited list of repository names for this resource, if needed. When a step attempts to retrieve artifact versions from an ElectricFlow repository, it queries repositories specified here. If no repositories are specified, it will "fallback" to the default repository.
Connection Type	Use the drop-down menu to select your Connection Type: • **HTTP**—Choose this option if you are not using SSL and the agent does not need to be "trusted". • **HTTPS**—Choose this option if you are using SSL, but this agent does not need to be "trusted". • **Trusted HTTPS**—Choose this option if you are using SSL and this agent must be "trusted." A trusted agent is "certificate verified"—The agent verifies the server or "upstream" agent's certificate. For information about the certificate used for trusted agents, see eccert. Agents can be either *trusted* or *untrusted*: • *trusted*—The ElectricFlow agent verifies the server or "upstream" agent's certificate. • *untrusted*—The ElectricFlow agent does not verify the server or "upstream" agent's certificate. This agent will accept communications from any ElectricFlow server. An untrusted agent is a potential security risk.

Field	Description
Enabled	Select this box to enable this resource. When this box is checked, the resource is enabled, which means job steps can be assigned to it. If a job step requests a pool containing this resource, the step executes on a resource in that pool. If disabled, and this is the only resource that satisfies a particular job step, the step's execution is delayed until the resource is re-enabled. If a resource is disabled while job steps are running on it, the running steps continue to completion but no new steps are assigned to that resource.

Click **OK** to save your settings and see your new resource listed in the table.

New Proxy Resource Panel

These are the definitions of a proxy agent and proxy target:

- Proxy *agent*—This is an agent on a supported Linux or Windows platform, used to proxy commands to an otherwise unsupported platform.
- Proxy *target* —This is a machine on an unsupported platform that can run commands via an SSH server.

To define a new proxy resource,enter information in the following fields:

Field	Description
General	
Name	Enter a unique name for this resource. ***Do not use*** spaces in a resource name. This is the name used to select the resource for a job step; it does not need to be the same as the host name for the machine.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Proxy Agent Host Name	Enter the domain name or IP address of the proxy agent machine corresponding to this resource.
Proxy Agent Port Number	Enter the port number to use when connecting to the agent for the resource.

Field	Description
Default Workspace	Enter the workspace name for leave blank to use the local, default workspace. If you specify a workspace here, it will be used as the default for all job steps that run on this resource. See the Workspaces topic in the Automation Platform online Help for more information.
Pool(s)	Enter a comma-separated list of arbitrary names, indicating the pools associated with this resource.
Host Type	Use the drop-down menu to select the host type: • **Concurrent** • **Registered** Depending on the license installed on the ElectricFlow server, this field may appear. • If the license on the server is a concurrent resource license, the host type defaults to **Concurrent** and this field does not appear. • If the license on the server is a registered host license, the host type defaults to **Registered** and this field does not appear. • If the license on the server is a mixed-mode license (concurrent resources and registered hosts), you must specify the host type when adding a resource.
Step Limit	Specify the maximum number of steps that can execute simultaneously on this resource—a step limit applies to a particular resource only, not to its underlying host. For example, if you define two resources, `resource1` with a step limit of 5 and `resource2` with a step limit of 1, both specifying the same host machine, it is possible for a total of 6 steps to execute simultaneously on the underlying host.
Zone	The name of the zone where this resource will be a member.

Field	Description
Connection Type	Use the drop-down menu to select your Connection Type: • **HTTP**—choose this option if you are not using SSL and the agent does not need to be "trusted". • **HTTPS**—choose this option if you are using SSL, but this agent does not need to be "trusted". • **Trusted HTTPS**—choose this option if you are using SSL and this agent must be "trusted". A trusted agent is "certificate verified"—the agent verifies the server or "upstream" agent's certificate. **Note:** For information about the certificate used for trusted agents, see eccert. Agents can be either *trusted* or *untrusted*: • *trusted* —The ElectricFlow server verifies the agent's identity using SSL certificate verification. • *untrusted* —The ElectricFlow server does not verify agent identity. Potentially, an untrusted agent is a security risk.
Enabled	Select this box to enable this resource. When this box is checked, the resource is enabled, which means job steps will be assigned to it. If a job step requests a pool containing this resource, the step executes on a resource in that pool. If disabled, and this is the only resource that satisfies a particular job step, the step's execution is delayed until the resource is re-enabled. If a resource is disabled while job steps are running on it, the running steps continue to completion but no new steps are assigned to that resource.
Proxy Target	
Proxy Target Host Name	Enter the domain name or IP address of the proxy target machine corresponding to this resource.
Proxy Target Port Number	Enter the port number to use when connecting to the proxy target host or leave blank to use the default. The default port is 22, the SSH server port.

Field	Description

Proxy Target Default Shell	The name of the shell program used to execute commands on this [the proxy target] resource—may be overridden at the step level.
	
	This shell will be used to execute the step's commands on a resource. For example, using `sh` or `cmd /c`, the agent saves the command block to a temporary file with a `.cmd` extension and runs it: `sh foo.cmd` or `cmd /c foo.cmd`.
	
	If you do not specify a shell on a step, at step run-time the server looks at the resource shell. If a resource shell is not set, the shell line used by the agent is platform dependent:
	Windows: `cmd /q /c "{0}.cmd"`
	UNIX: `sh -e "{0}.cmd"`
	
	When you specify a shell (in the step or resource), and omit the `cmd-file` marker, the agent notices the omission and takes the correct action. For example: a user specifies `sh -x`. The agent converts this to `sh -x "{0}.cmd"`.
	
	Two alternate forms of shell syntax where ElectricFlow uses a "marker," `{0}`, as a placeholder for the command file argument:
	• *<myShell>* `{0}` *<potential extra shell args>*
	In this example, the command file is not meant to be the last argument in the final command line.
	For example, `mysql -e "source {0}"`
	This shell example runs the `mysql` command against this step's command containing `sql`.
	• *<myShell>*`{0}`*<.file extension> <potential extra shell args>*
	In this example, the shell requires the command file to end in an extension other than `.cmd`.
	For example, `powershell "& '{0}.ps1'"`
	This shell example runs Microsoft PowerShell against this step's command containing PowerShell commands.
	
	Note:
	• When the agent parses the shell, it will parse the extension as everything after `{0}.` until it sees a space or non-alphanumeric character.
	• If your script uses International characters (non-ASCII), add the following block to the top of your `ec-perl` command block:
	`use utf8;`
	`ElectricCommander::initEncodings`

Proxy Customizations	This is proxy-specific customization data. Default is "none". For more information, see ecproxy on page 1074.

Click **OK** to save your settings.

Edit Resource Panel

All of the fields on this panel are the same as those on the **New Resource** panel:

- All information you previously provided to create this resource is filled-in for you.
- You can change any information already supplied.
- You can add information in any blank fields.

> **Note:** The **Gateway(s):** field that appears in the **Resource Details** panel is not editable (not even for a gateway resource) and therefore does not appear in the **Edit Resource** panel. This field is automatically filled and is only for gateway agents. For example, in the **Resource Details** panel for agentp1-gw-01 and agentp2-gw-01 resources as they form a gateway Default_to_zone1_gateway using the steps in the KBEC-00393 - Setting up an environment with two zones and a gateway KB article.

Click **OK** to save your modifications.

> **Note:** If you want to change the Connection Type, using the UI by itself is insufficient. You must also use the `ecconfigure` utility to change the protocol. Run: `ecconfigure --agentProtocol [http|https]`

Edit Proxy Resource Panel

All of the fields on this panel are the same as those on the New Proxy Resource panel:

- All information you previously provided to create this resource is filled-in for you.
- You can change any information already supplied.
- You can add information in any blank fields.

Click **OK** to save your modifications.

Resource Details Panel

This panel displays how this resource is configured with the ElectricFlow server. Some of the information supplied is linked to its source. For example, the zone name for this resource is a link to its Zone Details panel.

- The name of the resource is shown below the panel title.
- The resource description is shown under the resource name.
- Links at the top of the panel:
  - Use the **View Usage** link to go to the Job Step Search Results page.
  - Use the **Edit** link to modify the resource specifications.
  - Use the **Access Control** link [at the top of the panel] to specify or view access privileges for this resource.
- The tabs:

- General—By default, the panel opens to display the "general" resource configuration.
- Properties—Select this tab to add properties to this resource or see existing resource properties if already configured.

  Custom Resource Properties
  You can define custom properties for any ElectricFlow object. For example, if configuration information varies from resource to resource, use custom properties to store this information and then reference it from procedure steps where it is needed.
  For example, if you have a pool of test machines with test hardware in a different location on each machine, you could define a property named "testLocation" on each resource, then pass this property to procedure steps using a reference such as `$[/myResource/testLocation]`.

Note: The **Gateway(s):** field that appears in the **Resource Details** panel is not editable (not even for a gateway resource) and therefore does not appear in the **Edit Resource** panel. This field is automatically filled and is only for gateway agents. For example, in the **Resource Details** panel for agentp1-gw-01 and agentp2-gw-01 resources as they form a gateway Default_to_zone1_gateway using the steps in the KBEC-00393 - Setting up an environment with two zones and a gateway KB article.

Switching a Non-Trusted Agent to Trusted

If you have upgraded to ElectricCommander 4.2 or later, none of your existing agents were converted to "trusted" during the upgrade process. By default, the *local* agent is not trusted during a new or upgrade installation.

After an upgrade, if you want to use a trusted configuration, any existing agent or host machines must be manually switched to "trusted."

Perform the following steps on the agent machine to change the agent status to trusted in your environment and in the customer's site.

1. Enter `ectool --server <COMMANDER_SERVER_IP> login admin` to log into the server and save the session ID.
2. Enter `eccert --server <COMMANDER_SERVER_IP> initAgent --remote --force` to install the agent with a self-signed certificate that needs to be overwritten.
3. Restart the agent.

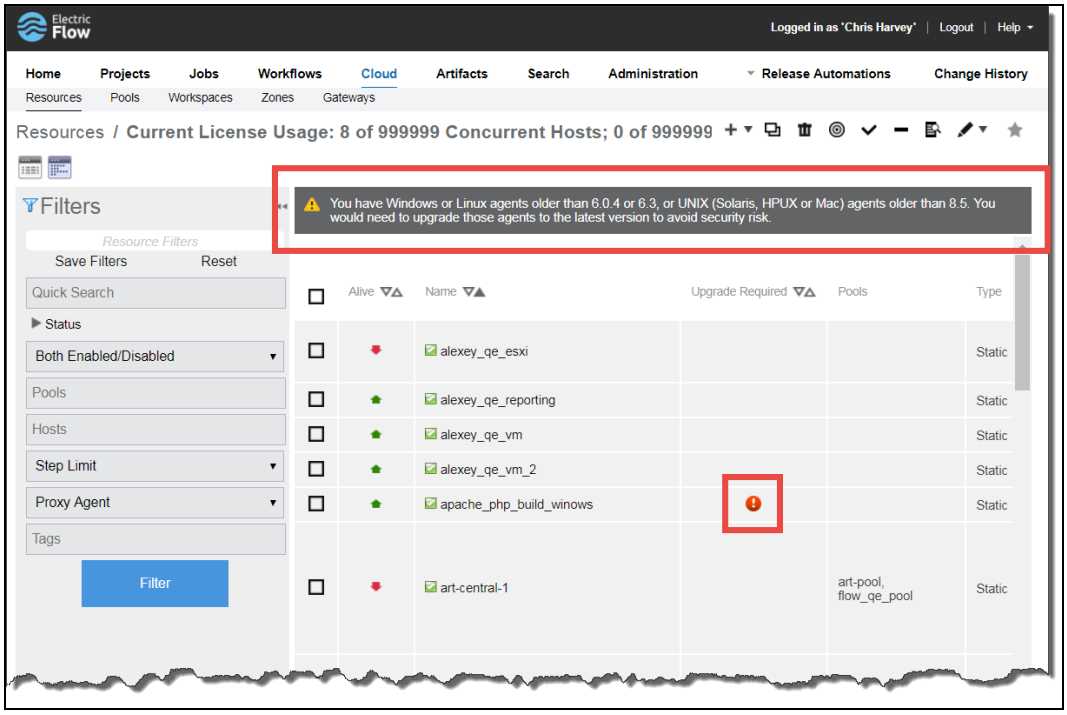
   If you do not restart the agent, it tries to use a previously-cached certificate if one exists. The old certificate is invalid because a new one was just created.
4. In the ElectricFlow UI, ping the agent.

For information about the ElectricFlow Certificate Authority (CA) and the certificates configured in ElectricFlow Server and ElectricFlow Agent installations, ElectricFlow Installed Tools on page 1045.

Upgrading Agents for Compatibility with Transport Layer Security (TLS)

If there are outdated agents, a warning appears on the **Resources** page that says `You have Windows or Linux agents older than 6.0.4 or 6.3, or UNIX (Solaris, HPUX or Mac) agents older than 8.5. You would need to upgrade those agents to the latest version to avoid security`

risk.. Also, those resources that require an upgrade are flagged with a icon in the **Upgrade Required** column:



The **Upgrade Required** column appears only if there are outdated agents.

The default security configurations are as follows:

- First-time ElectricFlow installations: TLSv1, TLSv1.1, and TLSv1.2 are enabled
- Existing ElectricFlow installations: TLSv1, TLSv1.1, TLSv1.2, and SSLv2Hello are enabled

We recommend removing the `SSL 2.0 Client Hello` or `SSLv2Hello` protocol from your security configurations for all components. When you do this, you would also need to upgrade older agents to the latest version to avoid security risks. You would need to upgrade agents if you are using the following agent versions:

- Windows: 6.0.3 or older
- Linux: 6.2 or older
- Sun Solaris, HP UX, Mac OS: 8.4 or older

For details about upgrading agents and enabling the TLS protocol for agents, see the *ElectricFlow Installation Guide* at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html.

Resource Pools

This page displays all resource pools currently available to ElectricFlow.

- To create a new resource pool, click the **Create Resource Pool** link.
- To find an existing resource pool, use the **New Search** link.

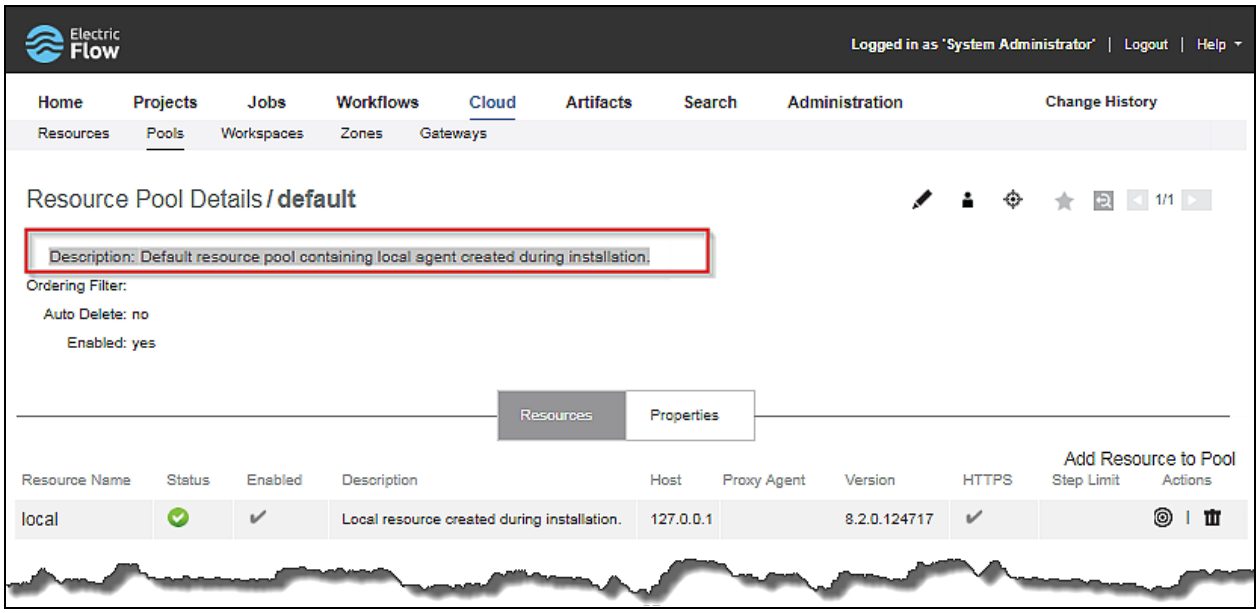
Column descriptions

Column Name	Description
Pool Name	This is the name you supplied for this resource pool. Click on a Pool Name to go to the Resource Pool Details page.
Enabled	If this box is "checked", this resource pool is enabled.
Auto Delete	If this column is selected with a check mark, the pool will be deleted when the last resource is deleted or removed.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Actions	Use the **Copy** link to make an exact duplicate of an existing resource pool, then select the copy to go to the Resource Pool Details page to change the resource pool name or add or remove resources from the pool. Use the **Delete** link to delete the resource pool on the same row. **Note:** Deleting a resource pool does not delete the resources referenced by the pool.

Resource Pool — Create a New Pool or Edit Existing Pool

Default resource pool

ElectricFlow includes a default resource pool, which is created during installation. It initially includes the "local" resource. The default resource pool's description is `Default resource pool containing local agent created during installation`:



Following is the order of precedence that determines how jobs select resources. The order of precedence includes the default resource pool as indicated below:

1. When a job step is run, it checks if a resource or pool was defined at the step level.
2. If the resource field at the step level is empty, it checks the resource field at the procedure level.
3. If the resource field at the procedure level is empty, it checks the resource field at the project level.
4. If the resource field at the project level is empty, a resource is chosen from the default resource pool.

> **Note:** If the default resource pool is disabled, the job step will hang until it is re-enabled. If the default resource pool is deleted, renamed, or empty, a `NonExistent_Resource` error message appears when a job is run, which means that no resource was chosen for the job.

Creating a new resource pool

Enter information in the following fields:

Field Name	Description
Pool Name	Enter a name for the resource pool. This name must be unique within a list of multiple resource pool names and cannot be the same as any resource name.
Enabled	Select this check box to enable the resource pool.

Field Name	Description
Type	A resource can be *static* or *dynamic*. • A static resource is part of your system or network (not in the cloud), such as a server, database, or agent machine. • A dynamic resource is a cloud resource that can be provisioned and later spun up on-demand when an application or microservice is deployed.
# Resources	Number of resources in the resource pool.
Resource Template Name	The name of the resource template to which the resource is applied. Only dynamic resources can be applied to a resource template.
Auto Delete	If you select this check box, the pool is deleted when the last resource is deleted or removed. • The `autoDelete` property flag is automatically set to "true" if the pool was created on the Create Resource page, as a side-effect of the resource API calls. • By default, resource pools created explicitly on the Create Resource Pool page, have the `autoDelete` flag set to false—the checkbox is "unchecked".
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Actions	Actions you can perform on the resource pool, including: • **Copy**—Creates a new resource by copying the this resource pool. • **Delete**—Deletes this resource pool. • **Tear Down**—Removes (decommissions) the dynamic resource pool from a dynamic environment. • **Track Changes**—Opens the Change History for the resource pool.

Click **OK** to save the information to create a new resource pool. You will see your new resource pool listed on the Resource Pools page.

> **Note:** Resource pools cannot be nested, which means if the name of a resource pool is added to a resource pool, it will be ignored.

A resource pool can override the default algorithm by providing a JavaScript fragment in the Ordering. The evaluation result is expected to be an array of resources, in the order they should be considered by

the scheduler. Before evaluation, the JavaScript fragment is wrapped as a function, to make it possible to use the `return` statement in the script.

It is possible to return any resources in the result, not just those grouped in a pool, so this feature enables true dynamic resource selection. Resources are still subject to the "usable" criteria. For details, see Unusable resources on page 1232.

Examples for the Ordering Filter field:

- Sort by resource name in descending order:

```
return resourceList.sort(function(a,b) {
return b.resourceName.localeCompare(a.resourceName);
});
```

- Ignore the default resources and return different resources:

```
var result = new Array();
result[0] = resources["resource1"];
result[1] = resources["resource2"];
return result;
```

Editing an existing resource pool

Use this page to modify any specifications for the resource pool. Click **OK** to save the information and update the resource pool.

Unusable resources

Resources, whether in a pool or not, are not considered usable if any of the following are true:

- The resource does not exist.
- The resource is disabled.
- The resource does not have a host name.
- The agent where the resource is associated is not reachable.
- The number of steps running on the resource is equal to the resource step limit.
- The resource is exclusive to a different job using the job `exclusiveMode`.
- The resource is exclusive to a step in a different job using the step or call `exclusiveMode`.
- The resource is exclusive to a different step in the same job using the call `exclusiveMode`. and the other step is not an ancestor of the step.
- The resource is exclusive to a different step in the same job using the step `exclusiveMode`.
- The job step does not have execute privileges on the resource.
- The resource assignment exceeds the license limit (if any).

Resource Pool Details

This page displays information for the resource pool you selected, including a summary of the specifications supplied to create the resource pool, current resources assigned to this pool, and the ability to add or remove resources.

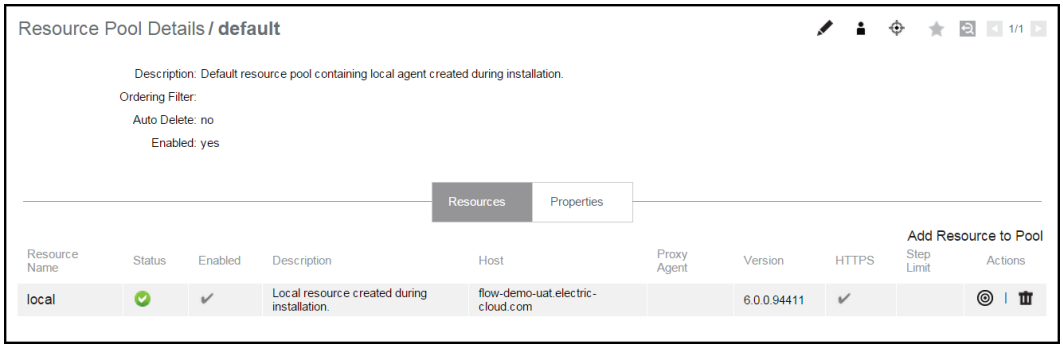
Links and actions at the top of the page

- **Edit**—use this link to go to the Edit Resource Pool page.
- **Access Control**—use this link to go the Access Control page for this resource pool.
- The "star" icon allows you to save this information to your Home page.
- The "curved arrow" icon returns you to the Resource Pools page.
- **Pagination**—Use the "previous" and "next" arrow icons to view the previous or next project. The numbers between the arrow icons display the number of projects you can view and the first number indicates which project [in the list] you are viewing.

The "tabbed" sections

The tabs at the top of the table allow you to select the type of information you want to see. The Resource Pool Details page opens with the Resources tab highlighted, so the first table you see is the Resources table. See the following screen example.

This table displays the resources currently included in this pool. The name of the pool ("custom" in our example) follows the Resource Pool Details page title.



The summary section at the top of the table contains information previously defined when the pool was created:

- **Description**—The text previously supplied for this object when it was created.
- **Ordering Filter**—The Javascript ordering filter (see the New Resource Pool Help topic for more information) or "empty" if no filter was applied.
- **Auto Delete**—When "yes" is specified, the pool will be deleted when the last remaining resource is removed or deleted from the pool.
- **Enabled**—"Yes" specifies this pool is enabled for use.

Links at the top of the table

Add Resources to Pool—Click this link to see a pop-up dialog to enter a resource name to add to this pool. You can type a name or select from a list of existing resource names. If other resources are already configured, you will see a list from which to select a resource. Click **OK** to add new resources to the pool.

Column descriptions

Column Name	Description or actions
Resource Name	The name of the resource. Click on a resource name to go to the Edit Resource page.
Status	Current status of the resource.
Enabled	Indicates whether the resource is enabled. If a resource is disabled, no job steps will be assigned to it. ElectricFlow will use other resources within the pool to satisfy requests for the pool.
Description	Text previously supplied for this object when it was created.
Host	Steps assigned to run on this resource will use this host.
Proxy Agent	Name or IP address of the proxy agent machine.
Version	Version of the ElectricFlow agent installed on this resource.
HTTPS	A check mark indicates HTTPS was selected when this resource was created.
Step Limit	Maximum number of steps that can execute simultaneously on this resource.
Actions	 • **Ping** —Use this link to check the status of the resource • **Remove**—Use this link to remove a resource from this pool.

Properties tab

This tab provides a table listing all properties for the resource pool and includes the following functionality.

Links at the top of the table

- To create a new property for this resource pool, click the **Create Property** or **Create Nested Sheet** link.
- To view or change privileges on the property sheet, click the **Access Control** link.

Column descriptions

Column Name	Description / Action
Property Name	The name of the property. Click on a property name to edit that property.

Column Name	Description / Action
Value	Indicates the value assigned to this property
Description	A text description previously supplied for your reference only.
Actions	**Delete**—Use this link to delete this property.

Zones

This page displays zones currently available to ElectricFlow and provides other zone operations—create,edit, delete, and so on.

- A zone is a way to partition a collection of agents to secure them from use by other groups. For example, you might choose to create a developers zone, a production zone, and a test zone—agents in one zone cannot directly communicate with agents in another zone.
- A *default* zone is created during the ElectricFlow installation. The server implicitly belongs to the default zone, which means all agents in this zone can communicate with the server directly (without the use of a gateway).
- Each zone can have one or more "gateway agents", which you define. Gateway agents are used for communication from one zone to another zone. For more information, see the Gateways Help topic in the Automation Platform online Help.
- Every agent, and all resources defined on that agent, can belong to one zone only.
- Within a zone, agents can be either *trusted* or *untrusted*.
  - Trusted—The ElectricFlow server verifies the agent's identity using SSL certificate verification.
  - Untrusted—The ElectricFlow server does not verify agent identity. Potentially, an untrusted agent is a security risk.

Links and actions at the top of the table

- To create a new zone, click the **Create Zone** link.
- The "star" icon allows you to save this zone information to your Home page.

Column descriptions

Column Name	Description / Actions
Name	The name supplied when this zone was created. Click on a Zone Name to see the Zone Details panel for that zone.
Descriptions	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.

Column Name	Description / Actions
Actions	**Delete**—Use this link to delete the zone on the same row. **Note:** Deleting a zone does not delete resources in the zone. All agents (and their respective resources) move to the default zone if their current zone is deleted. The default zone cannot be deleted.

Zone Details panel

This panel displays the zone name and description for the zone you selected.

Links and actions at the top of the table

- **Resources**—Displays a list of resources in the zone.
- **Edit**—Opens the Edit Zone panel.

  On this panel, you can change the zone name, or you can add or change the zone description. Click **OK** after making any changes.
- **Access Control**—Opens the Access Control page for this zone.
  - Current access privileges are displayed, if any.
  - Add the access control privileges you need for this zone.

Select the Properties tab to view any properties created for this zone, or use the "create" property links to create properties for this zone.

Creating a new zone

Enter information into the fields as follows:

Field Name	Description
Name	Enter a name of your choice for this zone. The name must be unique among other zone names.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.

Click **OK** to see your new zone displayed in the table. To add resources to this zone, go to the Resources page.

Access Control notes

A zone inherits privileges from the ZonesAndGateways ACL. See Access Control for more information.

Resource and Resource Pool inherit their privileges from Resources privileges. To create a resource, you must have modify privileges on Resources, and you must have modify privileges on the zone.

In addition, to move a resource from one zone to another, you must have modify privileges on both zones and the resource you want to move.

Gateways

To communicate with a resource, workspace, or artifact repository server in another zone, a "gateway" must be created. A gateway object contains two resource (agent) machines, for example, GatewayResource1 and GatewayResource2—each configured to communicate with the other. One gateway resource resides in the *source* zone and the other in the *target* zone. A gateway is bidirectional and informs the ElectricFlow server that each gateway machine is configured to communicate with its other gateway machine (in another zone).

If your company requires the added security of a firewall between zones, gateway agents can be configured to communicate with/through the firewall. Gateway agents can be trusted or untrusted (meaning that they just use HTTPS).

This page displays all gateways currently defined in ElectricFlow and provides other gateway operations—create, edit, delete, and so on.

- A firewall between zones— A gateway resource can be configured to communicate with an intermediary firewall in its path as a proxy to communicate with its peer on the other side of the gateway.

  > **Important:** If the actual gateway agents are behind a load balancer, do not register resources for them in ElectricFlow. The actual gateway agents should be pinged by ElectricFlow only via the load balancer.

- Each gateway records the host/port combination each gateway agent/resource must use to communication with its peer on the other side of the gateway.
- Multiple gateways can be defined for a zone if required.
  For example, you may have multiple resources in zoneA that need to communicate with each other, but some of those resources also need to communicate with zoneB, while others need to communicate with zoneC only. In this scenario, zoneA would require two gateways—one to zoneB and one to zoneC.
- One resource can participate in multiple gateways.
  For example, assume we have 3 zones, zone1, zone2, and zone3, each created to contain agent/resource machines for a different, specific purpose (production, testing), but we want to share or pass data from a resource in zone1 to another resource in zone2 or zone3:
  - We need two gateways:
    - Gateway1 connects ResourceA in zone1 to ResourceC in zone3
    - Gateway2 connects ResourceA in zone1 to ResourceB in zone2
  - With this gateway-resource configuration, ResourceA can communicate directly with zone2 or zone3.

Links and actions at the top of the table

- To create a new gateway, click the **Create Gateway** link.
- The "star" icon allows you to save this gateway information to your Home page.

Column descriptions

Column Name	Description / Action
Name	The gateway name specified when this gateway was created. Click a gateway name to see the **Gateway Details** panel for that gateway.
Enabled	A check mark in the box indicates that this gateway is enabled.
Resource 1	The first of two resources required to create a gateway. For actual gateway agents that are behind a load balancer, this is the resource for the inbound or outbound load balancer (not the actual agent).
Host 1	The external name that Resource 2 uses to communicate with Resource 1. This contains the host name or IP address of Resource 1. If this is blank, the Agent Host Name attribute in Resource 1's definition is used at runtime.
Resource 2	The second of two resources required to create a gateway. For actual gateway agents that are behind a load balancer, this is the resource for the inbound or outbound load balancer (not the actual agent).
Host 2	The external name that Resource 1 uses to communicate with Resource 2. This contains the host name or IP address of Resource 2. If this is blank, the Agent Host Name attribute in Resource 2's definition is used at runtime.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Actions	**Delete**—Use this link to delete the gateway on the same row.

Gateway Details Panel

This panel displays properties and access control privileges assigned to this gateway. Select the **Properties** tab to see any existing properties or to create properties for this gateway.

Links and actions at the top of the panel

- **Edit** —Click this link to display the **Edit Gateway** panel. On the **Edit Gateway** panel, you can change the gateway name or add or change the gateway description.

- **Access Control** —Click this link to go to the **Access Control** page to set access privileges for this gateway.

Create Gateway Panel

Enter the following information:

Field Name	Description / Action
Name	Name of your choice for this gateway. The name must be unique among other gateway names.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Enabled	Enables the gateway.
Resource 1	Name of your choice for the first of two required gateway resources. Do *not* include spaces in a resource name. For actual gateway agents that are behind a load balancer, specify the resource for the inbound or outbound load balancer (not the actual agent).
Host 1	Agent host name where Resource 1 resides. This external host name is used by Resource 2 to communicate with Resource 1. Specify only the host name or IP address of Resource 1. To use the host name from Resource 1's definition, leave this field blank.
Port 1	Port number used by Resource 1. The default is the port number used by the resource.
Resource 2	Name of your choice for the second of two required gateway resources. Do *not* include spaces in a resource name. For actual gateway agents that are behind a load balancer, specify the resource for the inbound or outbound load balancer (not the actual agent).
Host 2	Agent host name where Resource 2 resides. This external host name is used by Resource 1 to communicate with Resource 2. Specify only the host name or IP address of Resource 2. To use the host name from Resource 2's definition, leave this field blank.
Port 2	Port number used by Resource 2. The default is the port number used by the resource.

Click **OK** to see your new gateway displayed in the table.

Edit Gateway Panel

This panel is populated with previously supplied information to define the gateway. You can change any existing specifications or add new information. Click **OK** to save your changes.

Access Control Note

A gateway inherits privileges from the ZonesAndGateways ACL. See Access Control on page 1307 for more information.

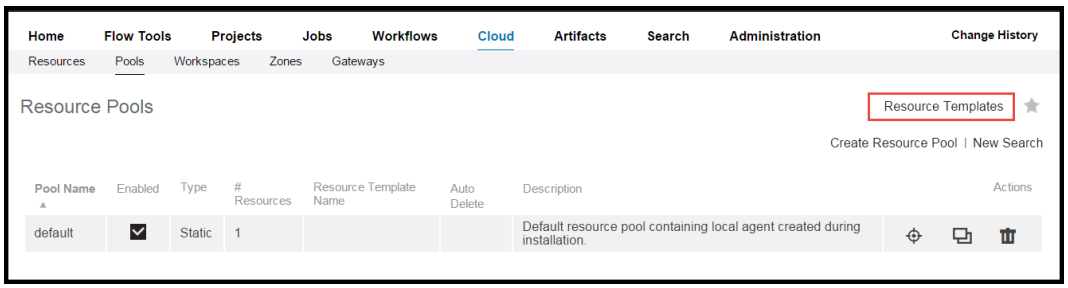
Accessing the Resource Templates in the Automation Platform

A resource template has the required information to provision and spin up cloud resources on an on-demand basis to model a dynamic environment. You set the cloud provider and configuration management details in the resource template. Then you define the environment tiers and add resource templates to these tiers in an environment template to complete the dynamic environment configuration.

You can generate new resource pools in ElectricFlow by accessing the resource templates from the automation platform UI.

- Go to **Cloud** > **Pools**.

  Example:



- Click **Resource Templates**.

  The **Resource Templates** list opens.

  Example:

- (Optional) To view details about a resource template, choose a template and click the **Menu** button

  ( ) for it.

  The Context menu opens.

- Click one of these options:
  - **Details**—The **Edit Resource Template Details** dialog box opens.

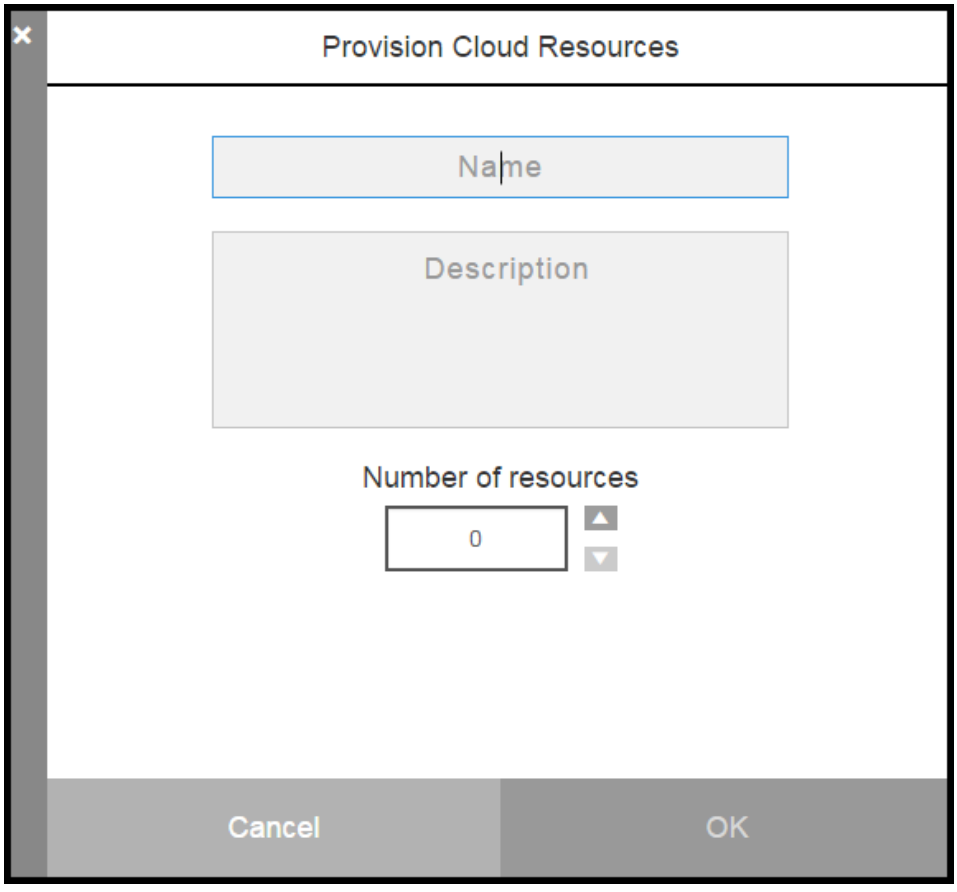
    The tabs that appear in the dialog box depend on the cloud provider specified in the resource template.
  - **Properties**—The **Properties** dialog box opens.

    You can view and edit the properties that apply to the resource template.
  - **Access Control**—The Access Control page opens.
  - **Track Changes**—The Change History for the resource template opens.

- To provision the resource template, choose a resource template, click the **Menu** button ( ), and select **Details** > **Provision**.

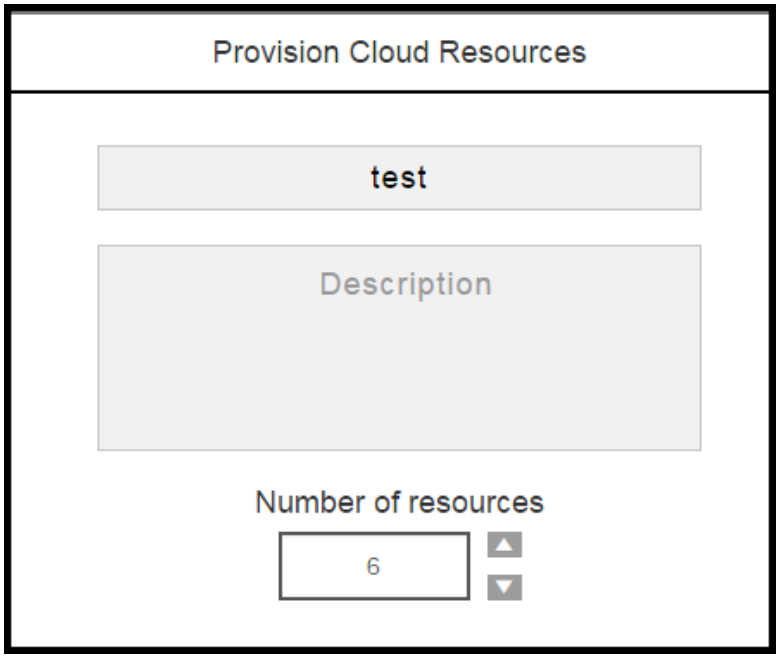
  The **Provision Cloud Resources** dialog box opens.

Example:



- Enter the resource pool name, an optional description, and the number of resources in the pool, and then click **OK**.

  Example:



You go to the Job Details page in the automation platform.

Searching and Filtering

The page allows you to search for any supported object type.

Enter information into the fields as follows:

Field Name	Description
Object Type	Use the drop-down menu to select the object you wish to search.

Field Name	Description
Criteria	Click **Add Criteria** one or more times to further define the criteria for your search. • Use the drop-down arrow to select a criteria argument. These arguments are properties. For a description of any of these properties, see the Properties Help topic. • Use the drop-down arrow to select an operator for the search. • Enter a value to search that is based on the selected operator and argument. For example, if you selected Project Name to begin your search and "contains" as the operator, you might want to enter the actual project name in this field or the first few letters if you have multiple projects named in a similar pattern.
Custom Criteria	Click **Add Custom Criteria** one or more times to enter custom criteria to further define your object search. Type the property name to search, and then select your search operator and the search value.

Click **OK** after filling in the fields and to go to the Search Results page.

Note:

- Only the first 450 characters of a property string are searchable. For example, if your property value happens to contain 1500 characters and the information you are trying to find is 800 characters into the property string, the search mechanism will not find it.
- When defining a search on this page, property path references are ***not*** allowed in search parameters. However, property path references are allowed in the Home page Quick View filters.
- The maximum number of search results that will be returned on the Search Results page is controlled by the server property `ec_ui/maxIds/default`. The out-of-the box default for this property is 1000. This default can be overridden for a specific object type. For example, the maximum number of jobs to display is controlled using the server property `ec_ui/maxIds/job` that is set to 500 out-of-the box.
- The default page size of 100 will be used to display the search results on the Search Results page, unless a user-specified page size is set (this user-specified setting will be used). You can save a user-specified page setting by changing the value in the Records Per Page drop-down field at the bottom of the Search Results page.
- The **Records Per Page** drop-down options are controlled through a server property `ec_ui/pageSizeOptions` with 10, 20, 50,100, 250, or 500 as the out-of-the-box value.
- Any changes to the properties within the server property sheet `ec_ui/maxIds` or the server property `ec_ui/pageSizeOptions` take effect after you log in again.

Context Searching and Filtering

Context searching allows you to search for objects on the current page without having to navigate to a separate Search page. It is available on these pages:

- Projects
- Jobs
- Artifacts
- Artifact versions
- Workflow

In-context search has these capabilities:

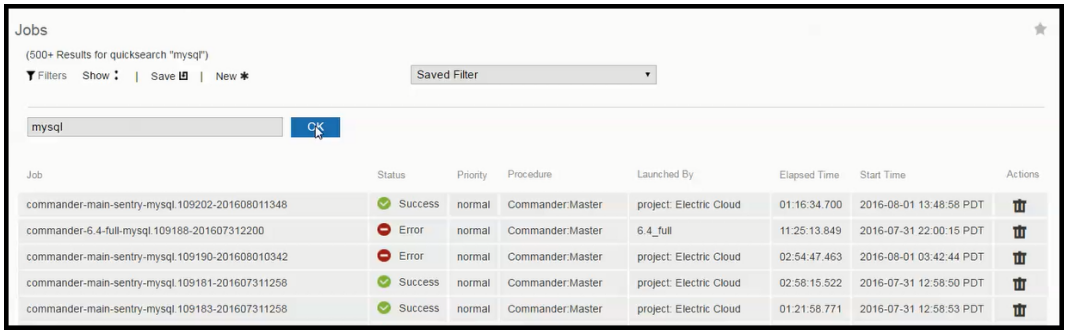
- Quick search: Type the search criteria in the text box and click **OK** to run a quick search.
- Search filters: Enter criteria to filter the results by specific attributes or properties on the selected object, and click **OK** to run the search.
- Saved searches: Save a search as a named saved search for later use.

1	Quick search. The **OK** button also applies to the search filters.
2	Search filters
3	Saved searches

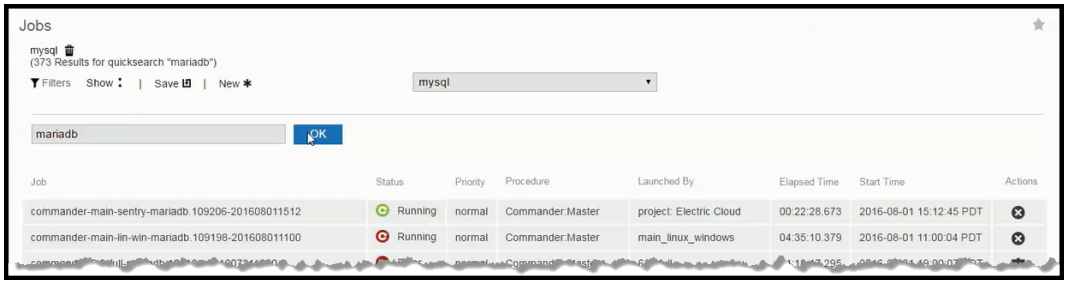
Example

Running Quick Searches

On the Jobs page, if you do a quick search for "mysql", you get these results:



If you do a quick search for "mariadb", you get these results:

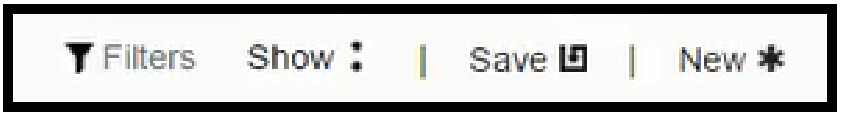


Saving Filters

To save the filter for jobs containing "mysql" in the job name:

1. Click **Save**.

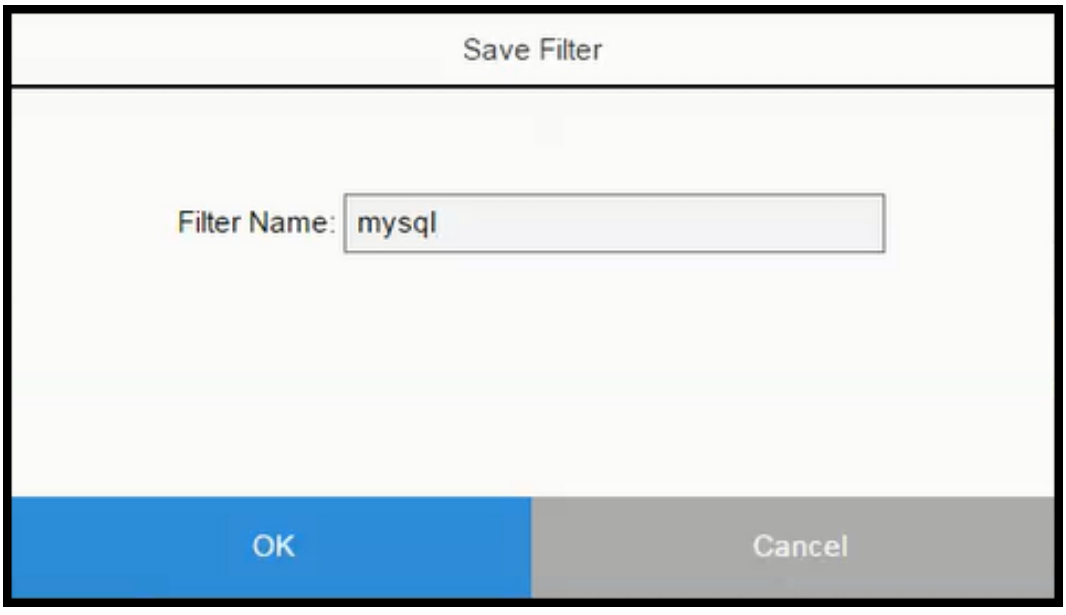
   Example:

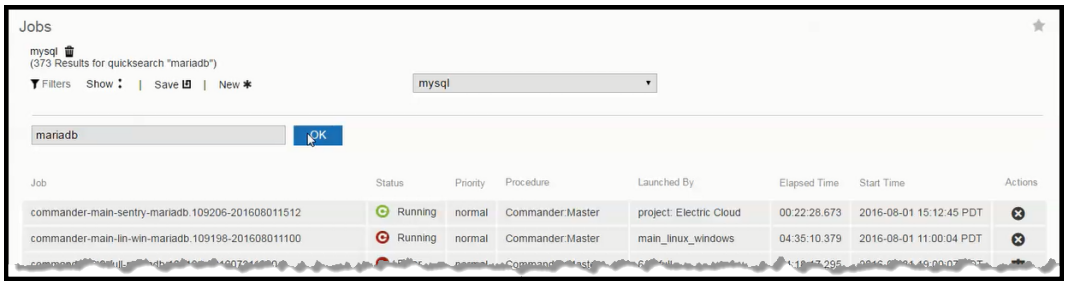
   The **Save Filter** dialog box opens.

2. Enter the filter name in the **Filter Name** field and click **OK**.

   Example:

The **Saved Filter** field now shows that there is a saved filter called mysql.



When you have more than one saved filter, click on the down arrow to see the list of filters you have saved. The saved filters are user-specific so you will only see your saved filters and not other users' filters.

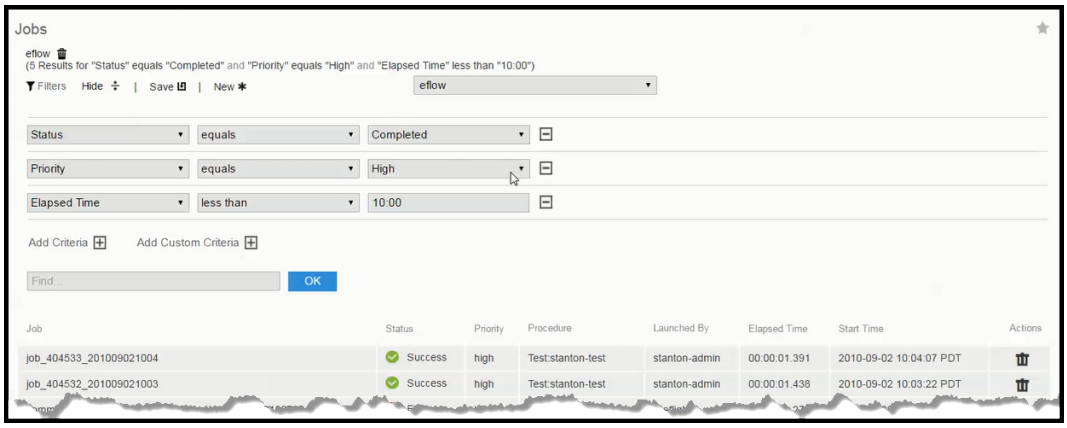
Setting Search Filters

To add filters for search:

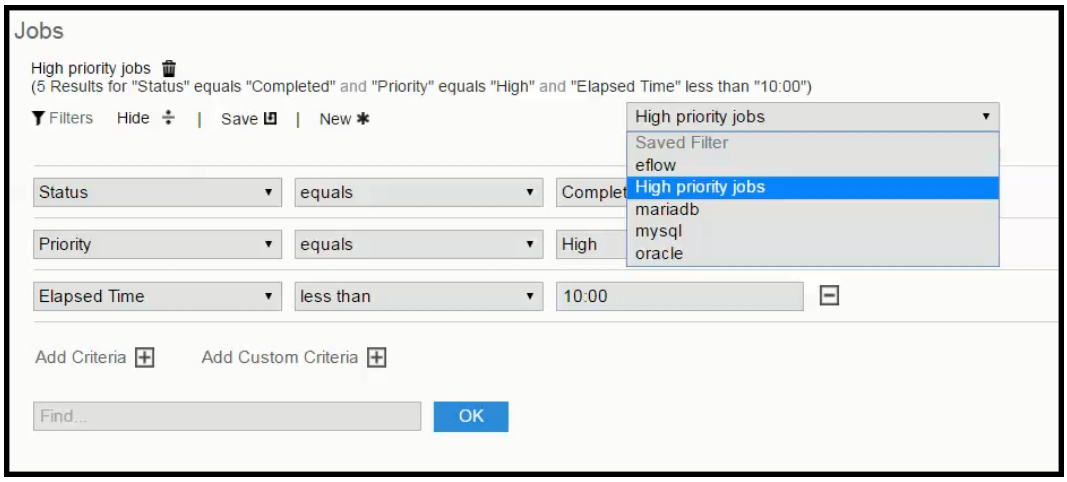
1. Click **Show** to show the filters.

   The **Add Criteria** and **Add Custom Criteria** options appear along with the filter fields.
2. Click **Add Criteria** to add criteria.
3. In the first field, select the search filter argument.
4. In the second field, select the operator for the filter.
5. In the third field, select the criteria to narrow down your search. The available options depend on the argument you selected.
6. To add another filter, enter a value in the Quick Search text box, click **Add Criteria**, or click **Custom Criteria**.
7. Repeat the previous step to add as many filters as you want.
8. To run the search, click **OK**.

This is an example of a search with three filters:



You can also save this filter as "High priority jobs":



Search Results

This page displays the results of the search you defined.

- If the results were not as expected, click the **Edit Search** link to modify your search criteria.
- Click the **New Search** link to define a new object you need to find.
- Click the **Save Report Filter** link if you want to save this filter for future use in a report. In the pop-up screen, enter a name for the filter and click **OK**. The Saved Filter will be stored as a property in your chosen project.
- To continually monitor the status of your found object, click the **Refresh Search** link.

Note:

- If you need to refer to this page on a regular basis, mouse-over the "star" icon to add this page to the Shortcut section on your Home page.

- If you use RSS, an active RSS icon is provided in Windows Explorer on this page for your convenience. If you use Firefox, click **Bookmarks** > Subscribe to this page to display the feed. You can then add the feed URL to a viewer of your choice.

Server

This page displays overall information about the ElectricFlow server. These links are at the top of the table:

- Click the **Settings** link to go to the Edit Server Settings page.

  Use this page to edit properties for the ElectricFlow server.

- Click the **Access Control** link to go to the Access Control page for the server.

  See the Access Control Help topic for more information.

The tables have this information:

- The System Access Control table displays top-level properties.Click on an item in the Category column to see, add, or modify privileges for that object.
- In the Custom Server Properties table, click on a Property Name to view the property content.

  If using *ectool*, these properties can be accessed using the property name starting with "`/server`". See the Properties Help topic for more information.

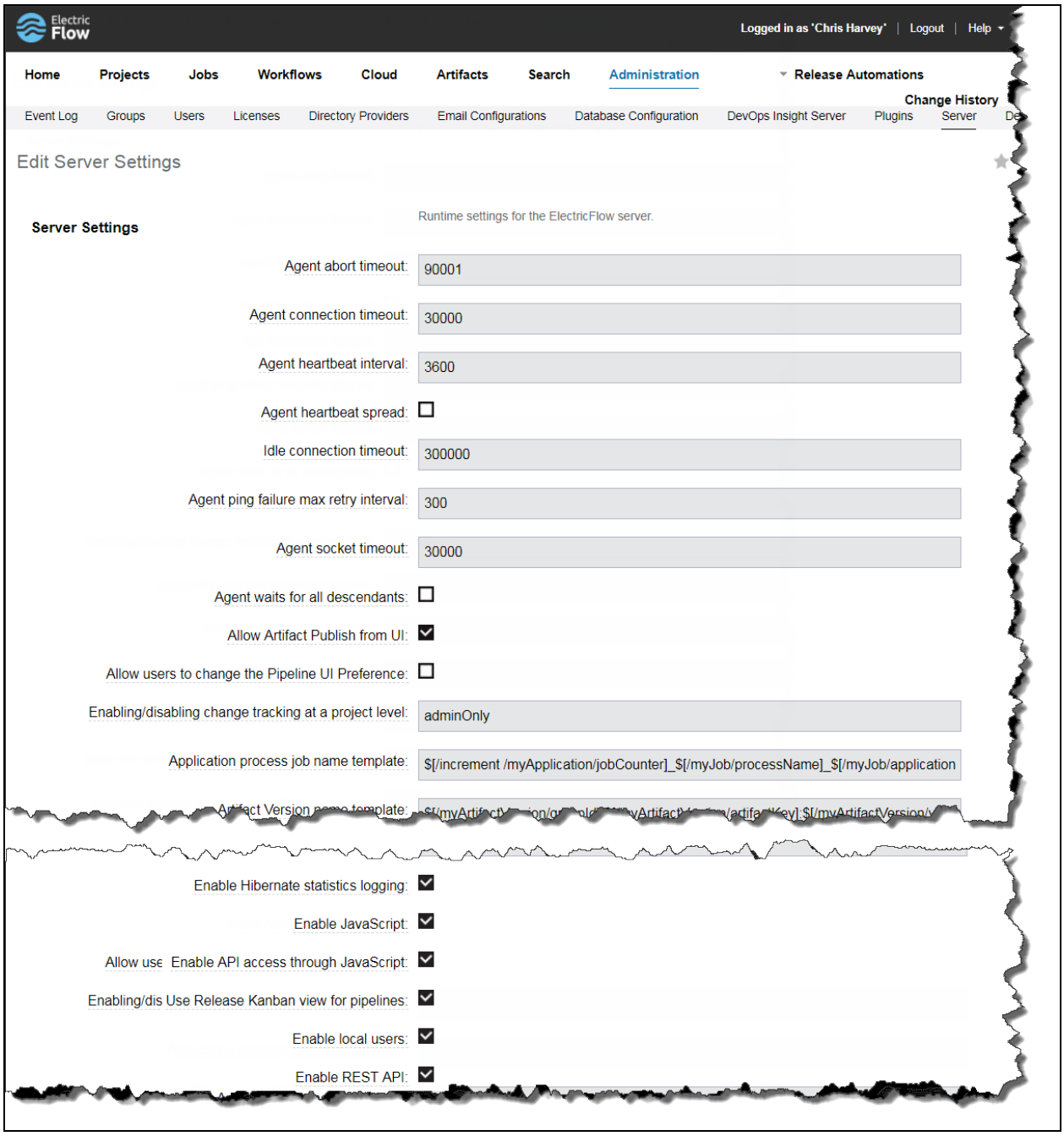
  These links are provided within the table:

  - **Create Property**—Use this link to create a new server property.
  - **Create Nested Property**—Use this link to create a new nested server property.
  - **Access Control**—This access control link allows you to create or edit privileges for the property sheet.
- The Actions column allows you to Edit or Delete the property in that row.

Settings—edit existing property settings

Use this page to edit property settings for the ElectricFlow server or any integrations that you use with ElectricFlow. If you access this page from the Server page, the page is named **Edit Server Settings**. This page contains fields for editing ElectricFlow server properties. From Perforce, the page is named **Edit Perforce Settings**, and the editable fields are specific to Perforce functionality.

The following example is from the **Edit Server Settings** page. For more information about a setting, click the name to see a tooltip that describes the setting.



Click **OK** to save the settings. If you must change these settings frequently, add this page to your Home page for quick access by clicking the star icon in the upper right corner.

Source Control Configurations

This web page displays all source control configurations you have created to communicate with ElectricFlow.

Link at the top of the table

- Use the **Create Configuration** link to create a new or additional source control configuration.

Table column definitions

Field or Option	Description
Configuration Name	Name you chose for this source control configuration.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Plugin	ElectricFlow plugin associated with this source control configuration.
Actions	• **Edit**—Modify information previously supplied to configure this source control system. • **Delete**—Remove this source control configuration.

Source Control Configurations—create new or edit existing configuration

Use this page to define a new source control configuration (SCM) or modify an existing source control configuration. A configuration is a collection of properties that define how to communicate with a particular source control system.

ElectricFlow bundles and supports numerous source control types. After creating a source control configuration, your entry will appear in the table on the Source Control Configurations web page— to see this web page, select the **Administration > Source Control** tab.

Other SCM systems are bundled with ElectricFlow and available in the Plugin Catalog. Each integration contains options specific to that SCM. The following SCM integrations are provided as examples:

- AccuRev
- ClearCase
- File
- Git
- Perforce
- Property
- Subversion

If you need a different SCM not specified in this Help topic, go to the Plugin Manager page to locate your SCM (**Administration > Plugins**). See the Help topic associated with your plugin—located on the Plugins page. At a minimum, you need to enter a name for your configuration, and you may want a minimal text description.

AccuRev

Enter information into the fields as follows:

- **Configuration Name**—Type a name for the configuration— any name you choose, but the name must be unique. For example, you might use "myAccuRevServer."
- **Description**—You can change or modify the default text description—ElectricFlow does not use this information.
- **Login As**
  - **User Name**—The name ElectricFlow needs to use to communicate with your AccuRev system. For example, you may be using a special "read-only" user name similar to "Build" for your user name.
  - **Password**—The password for the specified User Name.
  - **Retype Password**—Type the password again.

Click **Submit** to save your information.

ClearCase

Enter information into the fields as follows:

- **Configuration Name**—Type a name for the configuration— any name you choose, but the name must be unique. For example, you might use "myClearCaseServer."
- **Description**—You can change or modify the default text description—ElectricFlow does not use this information.

Click **Submit** to save your information.

File

You may want to use this configuration with ElectricSentry to "watch" for changes in any files.

Enter information into the fields as follows:

- **Configuration Name**—Type a name for the configuration—any name you choose, but the name must be unique.
- **Description**—You can change or modify the default text description—ElectricFlow does not use this information.

Click **Submit** to save your information.

Git

Enter information into the fields as follows:

Configuration Name—Type a name for the configuration— any name you choose, but the name must be unique.

Description—You can change or modify the default text description—ElectricFlow does not use this information.

Click **Submit** to save your information.

Perforce

Enter information into the fields as follows:

Configuration Name—Type a name for the configuration— any name you choose, but the name must be unique. For example, you might use "myPerforceServer."

Description—You can change or modify the default text description—ElectricFlow does not use this information.

Login As

User Name (P4USER)—The name ElectricFlow needs to use to communicate with your Perforce system. For example, you may be using a special "read-only" user name similar to "Build" for your user name.

Password (P4PASSWORD)—The password for the specified User Name.

Retype Password—Type the password again.

P4PORT—Sets the hostname and port number ElectricSentry uses to contact the Perforce server (`hostname:1234`).

P4HOST—The name of the host machine to impersonate.

P4TICKETS—The full path name of the ticket file used by the "p4 login." This value is set as an environment variable, not as a command-line option (`fullPathAndFileName`)

P4CHARSET—Sets the character set used for translation of Unicode files (`characterSet`)

P4COMMANDCHARSET—Used to support UTF-16 and UTF-32 character sets (`commandCharSet`).

Click **Submit** to save your information.

Property

You may want to use this configuration with ElectricSentry to "watch" for any property changes.

Enter information into the fields as follows:

Configuration Name—Type a name for the configuration—any name you choose, but the name must be unique.

Description—You can change or modify the default text description—ElectricFlow does not use this information.

Click **Submit** to save your information.

Subversion

Enter information into the fields as follows:

Configuration Name—Type a name for the configuration— any name you choose, but the name must be unique. For example, you might use "mySubversionServer."

Description—You can change or modify the default text description—ElectricFlow does not use this information.

Login As

User Name—This is the name ElectricFlow needs to use to communicate with your Subversion system. For example, you may be using a special "read-only" user name similar to "Build" for your user name.

Password—This is the password for the specified User Name.

Retype Password—Type the password again.

Click **Submit** to save your information.

Checking out code from a source control system

Go to Projects > select a project > select a procedure. To create a New Step for source code management, select the **Plugin** link.

- In the Choose Step panel, select Source Code Management from the left pane, then select the SCM you configured.
- The right-pane now shows the types of steps available for your configuration. Select the step you need and automatically go to the New Step page.
- On the New Step page, notice the Subprocedure section now contains the SCM integration you configured and the step you chose.
- Enter the remaining information to create your SCM step.

To edit an existing source control configuration

Modify any of your source control configuration information by re-entering any previously entered information. Click **Submit** to save your modified source control information.

Active Users

This page displays all users known to this ElectricFlow server, including users defined locally within the server and those defined in external repositories such as LDAP or ActiveDirectory.

Active Users are those persons who have logged into ElectricFlow, although if a another user views a selection of user accounts, those users [viewed] will appear on the Active Users page also.

Click the **Show Inactive Users** link to go to the All Users web page—this page lists all users: local, LDAP or Active Directory, active, or inactive. If you delete an LDAP or Active Directory user from this [Active Users] page, the user will not be deleted from the All Users page.

- To edit ***local*** user content, click on the user name you want to modify.
- To create a new local user, click the **Create Local User** link at the top, right-side of the table.
- To delete a local user, click the **Delete** link in the Action column for that user.
- Click the "star" icon at the top, right-side of the table if you want to add this page to your Shortcut section on the Home page.

While you can view *external* user content, you must go to the LDAP or ActiveDirectory repository to edit external user information. However, you may associate properties with external users and then use those properties in ElectricFlow.

In the **Filter** field, enter a string to be used to filter users. The filter will automatically apply a trailing '*' to find all users starting with the entered text. Use * for wildcards (for example, searching for *foo* will return all users that include the string foo).

To configure your existing LDAP and Active Directory account repositories to communicate with ElectricFlow, click the **Administration** > **Directory Providers** tabs.

Users and Groups

Use the **Users** page or the **Groups** page to view a filtered list of ElectricFlow users or groups.

Users and Groups Page Field Descriptions

Enter information into the fields as follows:

Field Name	Description
Filter	Name or partial name of users or groups that match that name.
Maximum Results	Maximum number of results to return. The default is 100. **Note:** ElectricFlow has no upward limit, but if you enter 1000 or higher, you might overload your browser, which causes performance degradation or worse problems.
Include Unregistered Users	(**Users** page only) • If this checkbox is checked, all directory provider users are returned, including those not registered in ElectricFlow. Use this to request external LDAP or Active Directory providers. • If not checked, only users registered in ElectricFlow are returned.
Include Inactive Groups	(**Groups** page only) • If this checkbox is checked, all users or groups are returned regardless of whether they are known to the ElectricFlow server. • If not checked, only users or groups known to the ElectricFlow server are returned. For example, users who have logged in or groups containing users who have logged in.

And then click **OK** to begin the search.

Search Results Column Descriptions

Column Name	Description
User Name or **Name**	Name of the user or group. The name combined with the location are the unique identifier for this user or group, local to the ElectricFlow server. Click the name for details.
Repository	Repository containing the user or group. This is usually LDAP or Active Directory. `Local` means local to the ElectricFlow server. Other names in this column refer to defined directory providers.
Location	"Path" to the group in a hierarchy such as Active Directory or another LDAP directory provider. This is viewable only for users or groups from nonlocal directory providers. This is useful if you use Active Directory and nested groups.
Real Name	(**Users** page only) Real, given name for the user.
Email	(**Users** page only) User's email address as known within the system.

Column Name	Description
Last Login Time	(**Users** page only) Date and time of the user's last login.
Actions	• **Edit** button—Edit the user or group definition. This is available only for local users and groups. • **Delete** button—Delete the user or group from the ElectricFlow server. This does not delete nonlocal users from an LDAP or Active Directory repository. • (**Users** page only) **Register** button—Register the LDAP or Active Directory user in the system.

Create Local User or Create Local Group Links

To create a new user or group for users or groups local to the ElectricFlow server, click **Create Local User** (on the **Users** page) or **Create Local Group** (on the **Groups** page).

User—create new or edit existing local user

Use this page to create or modify a *local* user only. ElectricFlow supports two kinds of user accounts: Those defined externally in an LDAP or Active Directory system, and local users defined in this ElectricFlow server. Local users are not visible outside ElectricFlow.

We recommend using external accounts whenever available, but you may need to create local users if you do not have a shared directory service or if you need special accounts to use for ElectricFlow only.

To create a new local user

Enter information into the fields as follows:

Field Name	Description
User Name	Name of the user account to be used for login
Real Name	User's real (given) name
Email	User's email address
Password	User's password

Click **OK** to save the new local user data.

> **Note:** If you do not assign a password when creating the user, the user cannot log into ElectricFlow. You can use this mechanism while you are setting up the user account. Once the user account is fully set up, you only need to assign a password.

To edit an existing local user

- You may highlight and re-enter any previously entered user information.
- If you leave the password fields blank and click **OK**, the existing password remains unchanged.
- To change the user's password, enter the new password for the user and also enter the current admin password.

Click **OK** to save the modified user data.

Use this page to modify user properties or assign Custom User Properties. Select the **Create Property**, **Create Nested Sheet** or **Access Control** links to set the properties you need.

To configure your existing LDAP and Active Directory account repositories to communicate with ElectricFlow click the **Administration** > **Directory Providers** tabs.

User Details

This page displays external user information from a repository such as LDAP or Active Directory. To edit this information, you must connect directly to the repository. However, you can associate properties with an external user and then use those properties in ElectricFlow.

User Details Column Descriptions

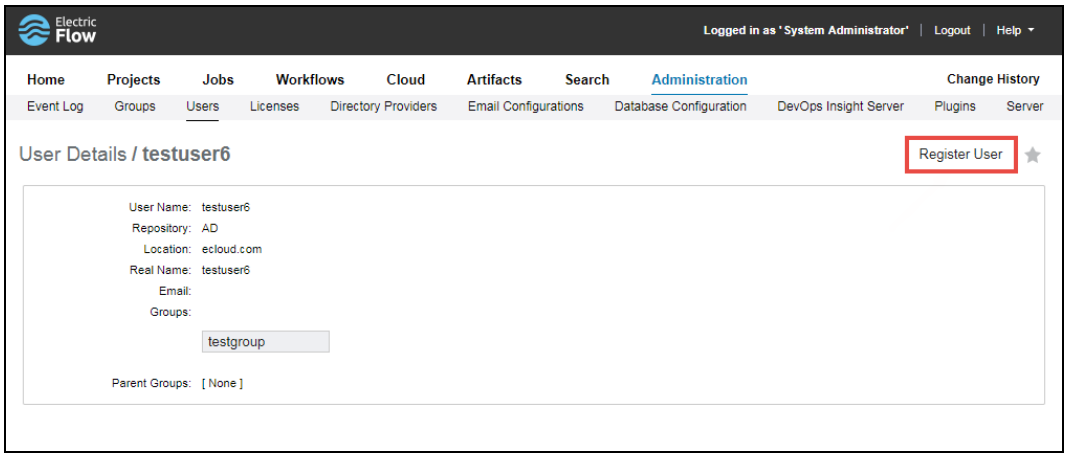
Column Name	Description
User Name	Name of the user. The name combined with the location are the unique identifier for this user local to the ElectricFlow server.
Repository	Repository containing the user. This is usually LDAP or Active Directory. `Local` means local to the ElectricFlow server. Other names in this column refer to defined directory providers.
Location	"Path" to the group in a hierarchy such as Active Directory or another LDAP directory provider. This is viewable only for users or groups from nonlocal directory providers. This is useful if you use Active Directory and nested groups.
Real Name	Real, given name for the user.
Email	User's email address as known within the system.
Last Login Time	Date and time of the user's last login.
Groups	Groups that include this user. Clicking a group name displays the **Group Details** page, which lists all members in that group.

Column Name	Description
Parent Groups	Parent groups for remote or external users if the repository (directory provider) to which the user belongs was configured with **Recursively Traverse Group Hierarchy** when the directory provider was defined. This field displays the groups in the LDAP or Active Directory server hierarchy in which the user's immediate groups are members. For example, if: • the user belongs to the "SFO" group, • and the "SFO" group belongs to the "California" group, • and the "California" group belongs to the "US" group, then the **Parent Groups** field for the user will display "California, US," and the **Groups** field will display "SFO."

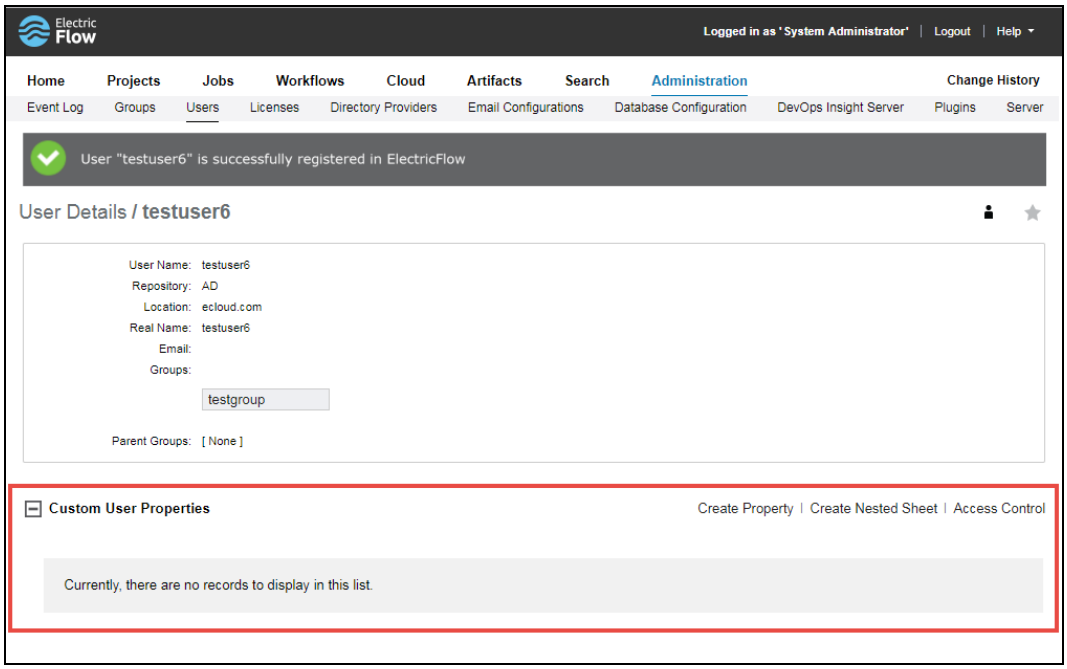
Tip: To configure your LDAP and Active Directory account repositories to communicate with ElectricFlow, click the **Administration** > **Directory Providers** tab.

Adding Registered Users

To add a registered user, click **Register User**:



After the user is registered, the **Custom User Properties** section and the **Access Control** button appear:



Modifying User Properties or Assigning Custom User Properties

For registered uses, you can modify user properties or assign custom user properties. To do so, click the **Create Property**, **Create Nested Sheet**, or **Access Control** links.

Edit User Settings

Use this page to modify your User Settings for the product tab view. Depending on which view you choose, you can have limited access to ElectricFlow features and functions.

In the Tab View, use the pull-down menu to make a new selection:

- **Default**—Selecting this view provides all regular/standard product tab views, including plugins, which by default are installed into this view.
- **Base**—This view removes the ability to use a plugin. You can see the plugin list (accessed by selecting the Plugins tab), but you are denied access to configure, install, uninstall, or promote a plugin.
- **Inherit**—Frequently this view is set for a group. The UI searches for the property, `ec_ui/defaultView`, first on the user, second in any groups the user belongs to, and last, on the server. The inherit setting clears the user's private defaultView setting and then picks up the first value from the rest of the search path.

Click **OK** after selecting a different tab view.

Groups

This page displays all groups known to this ElectricFlow server, including groups defined locally within the server and those groups defined in external repositories such as LDAP.

- To edit a ***local*** group, click on its name.
- To create a new local group, click the **Create Local Group** link.
- Click on an ***external*** group to view its contents, but you cannot edit the content through ElectricFlow.
  Instead, you must go to the group's repository directly. You can, however, associate properties with external groups, which can then be used in ElectricFlow.

In the **Filter** field, enter a string to be used to filter groups. The filter will automatically apply a trailing '*' to find all groups starting with the entered text. Use * for wildcards (for example, searching for *foo* will return all groups that include the string foo).

To configure your existing LDAP and Active Directory account repositories to communicate with ElectricFlow, click the **Administration**> **Directory Providers** tabs.

Group—create new or edit existing local group

To create a new local group

Enter information into the fields as follows:

- **Name**—Type a name for the group. Choose any name, but it must be unique among other local group names.
- **Users**—Type a list of users who need to belong to this group— type one user per line.

Click **OK** after populating the group.

To edit a local group

- You can highlight the existing group name and type a new name—or
- You can scroll to the bottom of the list of users and type-in additional members for this group, one name per line—or
- You can highlight a user name and press the delete key (on your keyboard) to delete that user from the group.
- Click **OK** after completing your changes.

Also, you can associate custom properties with a group. Select either the **Create Property, Create Nested Sheet**, or the **Access Control** links to set up the properties you need.

Group Details

This page displays external group information retrieved from a repository such as LDAP or ActiveDirectory.

- To edit external group information, you must connect directly to the repository to modify an external group.
  However, you can associate properties with an external group and then use those properties in ElectricFlow.
- Click the **Access Control** link (at the top of the table) to see the privileges assigned to this group, inherited privileges, add a user or group, and so on.
- You can associate also custom properties with a group. Select the **Create Property**, **Create Nested Sheet**, or the **Access Control** links to set up the properties you need.

Definitions for summary information at the top of page:

- **Name**: The name this user is known by in the system.
- **Repository**: Usually LDAP or Active Directory.
- **Users**: This field displays all users that are members of this group.
- **Nested Groups**: This field is displayed only for remote or external groups if the repository (directory provider) to which the user belongs is configured to **Recursively Traverse Group Hierarchy** (the **Recursively Traverse Group Hierarchy** option is selected when directory provider is defined). This field displays all the nested groups that are members of this group's hierarchy in the LDAP or Active Directory server.

  If the "SFO" a group is a member of the "California" group and the "California" group is a member of the "US" group, the **Nested Groups** field for the "US" group will display "California, SFO".

> **Tip:** To configure your existing LDAP and Active Directory account repositories to communicate with ElectricFlow, click the **Administration** > **Directory Providers** tabs.

Directory Providers

ElectricFlow uses account information from multiple sources. In most cases, the primary account information source is an external LDAP or Active Directory repository: both user and group information is retrieved from the repository. ***Local*** users and groups are defined within ElectricFlow.

Links at the top of the table

Click the **Add Active Directory Provider** or the **Add LDAP Provider** link to go to a new web page to configure your Directory Provider. After configuring your directory provider, that name will appear in the All Providers table, the Provider Name column.

Column descriptions

Column Name	Description / Actions
Provider Name	Click on a Provider Name to make changes to an existing directory provider.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Actions	• **Copy**—Use this link to make a copy of the Provider Name configured on this row. • **Remove**—Use this link to remove the Provider Name on this row.

Detailed Help for defining a directory provider in ElectricFlow is available from the New Active Directory Provider or the New LDAP Provider web page. Click the **Help** link from either web page. The Help text also contains examples you may find useful.

Directory Providers—create new or edit existing directory providers

ElectricFlow supports Active Directory and LDAP directory providers. This section contains instructions for creating an Active Directory or LDAP directory provider and editing those configurations. Use the following links to go the section you need:

To create a new Active Directory provider

To create a new LDAP directory provider

Examples for directory provider field descriptions

To edit an existing directory provider

To create a new Active Directory provider

Enter information in the fields as follows to specify your existing Active Directory users and groups to communicate with ElectricFlow. For examples of information you enter in the these fields, see the table after the following description sections.

Field Name	Description
General section	
Provider Name	This name identifies users and groups that come from this provider.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
URL Discovery	Select the method to retrieve the URL for the Active Directory server. Auto-discovery using DNS automatically discovers Active Directory servers on the given domain. Alternatively, you can specify a custom URL to an Active Directory server.
Domain Name	The domain where Active Directory servers are automatically discovered. For example, `my-company.com`
Use SSL	Select this box if you want to use SSL when the ElectricFlow server contacts your Active Directory server. **Note:** Transport Layer Security (TLS) has replaced Secure Sockets Layer version 3.0 (SSLv3) on the ElectricFlow web server and the ElectricFlow server.

Field Name	Description
URL	This is an explicit URL to the Active Directory server. The URL is in the form *protocol://host:port/basedn*. Protocol is either `ldap` or `ldaps` (for secure LDAP). The port is implied by the protocol, but you can override the port if you cannot use the default location (default is port 389 for `ldap` or 636 for `ldaps`). The `basedn` is the path to the top-level directory that contains users and groups at this site. Typically, this is the domain name where each part is listed with a `dc=` and separated by commas. **Note:** Spaces in the `basedn` must be URL encoded (use `%20` for spaces). If your site uses redundant directory servers for failover, you can enter multiple URLs separated by spaces.
Query User Name	The name of a user who has read-only access to user and group directories in Active Directory. This is the user name to use for fetching user and group information. When you provide a `domain name`, you can provide the simple name, for example, `myuser` instead of `electric-cloud\myuser`. When you provide an explicit URL, you need to provide a distinguished name, for example: `cn=myuser,ou=Users,dc=electric-cloud,dc=com`.
Query User Password	The password for the query user.
Membership Options section The membership options control whether the nested group hierarchy in the configured Active Directory server will be used by ElectricFlow. See LDAP Group Hierarchy on page 1366 for details on how the nested group hierarchy in Active Directory is used by ElectricFlow.	
Recursively Traverse Group Hierarchy	Select this to enable recursive traversal of the group hierarchy for nested group membership information. If **Recursively Traverse Group Hierarchy** is selected, select the *LDAP_MATCHING_RULE_IN_CHAIN* template for both the **Membership Filter** and **Group Member Filter** fields in the "Group Options" section to allow Active Directory to return the nested group membership information.
Membership Filter	Active Directory filter to use when searching for groups to which an Active Directory user or group belongs.
Include Nested Group Users in Notifications	Select this to include users in nested Active Directory groups when notifications for a parent Active Directory group are sent and **Recursively Traverse Group Hierarchy** is selected.

Field Name	Description
Include Nested Group Users as Approvers	Select this to allow users in nested Active Directory groups to complete or approve a manual task when a parent Active Directory group is assigned as an assignee or an approver for the task and **Recursively Traverse Group Hierarchy** is selected. **Note:** Because users who are authorized to approve the manual task will typically also be notified of the approval request, we recommend that **Include Nested Group Users in Notifications** be selected (enabled) when **Include Nested Group Users as Approvers** is selected.

User Options section

When creating an Active Directory provider, the ElectricFlow server automatically sets default values for any options (fields) that are empty. The default values match the most common Active Directory configurations. After the provider is created, you can view and modify defaults by modifying the provider.

User Base	This string is prepended to the `basedn` to construct the directory DN containing user records.
User Search Filter	This LDAP query is performed in the context of the user directory to search for a user by account name. The string `"{0}"` is replaced with the user's login ID. Typically, the query compares a user record attribute to the substituted user login ID.
User Name Attribute	This is the attribute in a user record that contains the user's account name.
Full User Name Attribute	(Optional) This is the attribute in a user record that contains the user's full name (first and last) for display in the UI. If this attribute is not specified or the resulting value is empty, the user's account name is used instead.
User Email Attribute	(Optional) This is an attribute in a user record that contains the user's email address. If this attribute is not specified, the account name and domain name are concatenated to form an email address.
Search User Subtree	Select this check box to search the specified directory by the user base and all directories below. If this check box is not selected, the search is limited to the specified directory only.

Field Name	Description
Group Options section **Note:** If you do not need group information at this time, do not enter information in the Group section, the group test fails without this information, but the test successfully authenticates users if both "user tests" are successful. When creating an Active Directory provider, the ElectricFlow server automatically sets default values for the options/fields that remain empty. These default values match the most common Active Directory configurations. After the provider is created, you can view and modify the defaults by modifying the provider.	
Enable Groups	Select this check box to enable external groups for this directory provider.
Group Base	(Optional) This string is prepended to the `basedn` to construct the directory DN containing group records.
Group Member Filter	(Optional) This LDAP query is performed in the groups directory context to identify groups that contain a specific user. Two common forms of group records in LDAP directories are: `POSIX` style groups where members are identified by account name, and `groupOfNames` or `uniqueGroupOfNames` records where members are identified by the full user DN. ElectricFlow supports both forms, so the query is passed two parameters: "`{0}`" is replaced with the full user record DN, and "`{1}`" is replaced with the user's account name.
Group Member Attributes	(Optional) This is a comma-separated attribute name list that identifies a group member. Most LDAP configurations only specify a single value, but if a mixture of POSIX and LDAP style groups exist in the directory, multiple attributes might be required.
Group Search Filter	(Optional) This LDAP query is performed in the context of the groups directory to enumerate group records. You can choose from common templates that include either security or distribution groups (or both). These templates are based on the most common Active Directory settings.
Unique Group Name Attribute	(Optional) This is the group record attribute that contains the group name. To prevent group name overlap between multiple directory providers (or within the same provider in a multi-forested Active Directory server), we recommend setting this attribute to the `distinguishedName`.
Common Group Name Attribute	The Unique Group Name Attribute may not be searchable if using the `distinguishedName`. In this case, the Common Group Name Attribute can be used to search for groups, depending on the filter used.

After filling in all fields, click the **Test** button. Three tests validate the information you supplied:

- user authentication
- user identified in Active Directory
- find all groups where the user is a member

If there is a test failure, correct the information you supplied and retest. Click **Save** after successful test results. New, defined directory providers will appear in the table on the Directory Provider web page.

To create a new LDAP directory provider

Enter information in the fields as follows to specify your existing LDAP users and groups to communicate with ElectricFlow. For examples of information you enter in the these fields, see the table after the following description sections.

Field Name	Description
General section	
Provider Name	This name identifies users and groups that come from this provider.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
URL	The LDAP server URL is in the form *protocol://host:port/basedn*. Protocol is either `ldap` or `ldaps` (for secure LDAP). The port is implied by the protocol, but you can override the port if you cannot use the default (default port is 389 for `ldap` and 636 for `ldaps`). The `basedn` is the path to the top-level directory that contains users and groups at this site. Typically, this is the domain name where each part is listed with a `dc=` and separated by commas. **Note:** Spaces in the `basedn` must be URL encoded (use `%20` for spaces). If your site uses redundant directory servers for failover, you can enter multiple URLs separated by spaces.
Realm	This is the realm of the LDAP directory provider, which is used to create unique user names when you have multiple providers. For example, if the realm is `my-company.com`, users are saved as `myuser@my-company.com`.
Query User Name	This is the name of a user who has read-only access to the user and group directories in LDAP. This is the user name to use for fetching user and group information. When providing a domain name, you can provide the simple name, for example, myuser. When providing an explicit URL, you need to provide a distinguished name, for example: `cn=myuser,ou=Users,dc=electric-cloud,dc=com`.

Field Name	Description
Query User Password	This is the password for the query user.
Membership Options section	The membership options control whether nested group hierarchy in the configured LDAP server will be used by ElectricFlow. See LDAP Group Hierarchy on page 1366 for details on how the nested group hierarchy in the LDAP server is used by ElectricFlow.
Recursively Traverse Group Hierarchy	Select this to enable recursive traversal of the group hierarchy for nested group membership information.
Membership Attribute	Attribute defined on an LDAP user or group entry that is used by the LDAP provider for specifying the group membership. `memberOf` is the membership attribute used by most LDAP servers.
Nested Groups Depth Limit	Maximum number of group hierarchy levels that will be traversed for retrieving nested group membership information. **Note:** Setting a high value for this option will impact the system performance when traversing the LDAP group hierarchy because a higher group hierarchy depth will result in a proportionally large number of network calls between the Electric Flow system and the LDAP server. The actual performance will depend on the system configurations of the ElectricFlow servers, the LDAP server, and the network latency.
Include Nested Group Users in Notifications	Select this to include users in nested LDAP groups when notifications for a parent LDAP group are sent and **Recursively Traverse Group Hierarchy** is selected.
Include Nested Group Users as Approvers	Select this to allow users in nested LDAP groups to complete or approve a manual task when a parent LDAP group is assigned as an assignee or an approver for the task and **Recursively Traverse Group Hierarchy** is selected. **Note:** Because users who are authorized to approve the manual task will typically also be notified of the approval request, we recommended that **Include Nested Group Users in Notifications** be selected (enabled) when **Include Nested Group Users as Approvers** is selected.
User Options section	

Field Name	Description
User Base	This string is prepended to the `basedn` to construct the directory DN containing user records.
User Search Filter	This LDAP query is performed in the context of the user directory to search for a user by account name. The string "`{0}`" is replaced with the user's login ID. Typically, the query compares a user record attribute to the substituted user login ID.
User Name Attribute	This is the attribute in a user record that contains the user's account name.
Full User Name Attribute	(Optional) This is the attribute in a user record that contains the user's full name (first and last) for display in the UI. If this attribute is not specified or the resulting value is empty, the user's account name is used.
User Email Attribute	(Optional) This is the attribute in a user record that contains the user's email address. If the attribute is not specified, the account name and domain name are concatenated to form an email address.
Search User Subtree	Select this check box to search the specified directory by the user base and all directories below. If this check box is not selected, the search is limited to the specified directory only.
Groups Options section **Note:** If you do not need group information at this time, do not enter information in the Group section, the group test fails without this information, but the test successfully authenticates users if both user tests are successful.	
Enable Groups	Select this checkbox to enable groups.
Group Base	(Optional) This string is prepended to the `basedn` to construct the directory DN containing group records.
Group Member Filter	(Optional) This LDAP query is performed in the groups directory context to identify groups containing a specific user. Two common forms of group records in LDAP directories are: `POSIX` style groups where members are identified by account name, and `groupOfNames` or `uniqueGroupOfNames` records where members are identified by the full user DN. ElectricFlow supports both forms, so the query is passed with two parameters: "`{0}`" is replaced with the full user record DN, and "`{1}`" is replaced with the user's account name.
Group Member Attributes	(Optional) This is a comma-separated attribute name list identifying a group member. Most LDAP configurations only specify a single value, but if you have a mixture of POSIX and LDAP style groups in the directory, multiple attributes might be required.

Field Name	Description
Group Search Filter	(Optional) This LDAP query is performed in the context of the groups directory to enumerate group records.
Unique Group Name Attribute	(Optional) This is the group record attribute containing the group name.
Common Group Name Attribute	The Unique Group Name Attribute may not be searchable if using `distinguishedName`. In this case, the Common Group Name Attribute is used if searching for groups, depending on the filter.

After fillingin all fields, click the **Test** button. Three tests validate the information you supplied:

- user authentication
- user identified in LDAP
- find all groups where the user is a member

If there is a test failure, correct the information you supplied and retest. Click **Save** after successful test results. New, defined directory providers will appear in the table on the Directory Provider web page.

Examples for directory provider field descriptions

The following table provides examples for filling in the fields described above:

Field Name	LDAP example	ActiveDirectory example
Provider Type	`LDAP`	`ActiveDirectory`
Domain Name	`N/A`	`my-company.com`
Realm	`my-company.com`	`N/A`
URL	`ldap://dir.company.com/dc=company,dc=com`	`ldaps://server/dc=company,dc=com`
Query User Name	`uid=JohnDoe,ou=People,dc=company,dc=com`	`cn=myuser,cn=Users,dc=company,dc=com`
Query User Password	`mypw`	`mypw`

Field Name	LDAP example	ActiveDirectory example
User Base	ou=People	cn=Users
User Search Filter	uid={0}	(&(|(userPrincipalName={0}) (sAMAccountName={0})) (objectClass=user))
User Name Attribute	uid	userPrincipalName
Full User Name Attribute	gecos	name
User Email Attribute	mail	mail
Group Base	ou=Group	cn=Groups
Group Member Filter	(|(member={0})(memberUid={1}))	member={0}
Group Member Attribute	member,memberUid	member
Group Search Filter	(|(objectClass=groupOfNames) (objectClass=posixGroup))	(objectClass=group)
Unique Group Name Attribute	distinguishedName	distinguishedName

Field Name	LDAP example	ActiveDirectory example
Common Group Name Attribute	cn	cn

To edit an existing directory provider

You may change any previously supplied information in the fields. After editing any fields, click the **Test** button. The same three tests validate the information you supplied:

- user authentication
- user identified in the directory provider
- find all groups where the user is a member

If there is a test failure, correct the information you supplied and retest. Click **Save** after successful test results. Edited, redefined directory providers will appear in the table on the Directory Provider web page.

Test Directory Provider

You must enter a valid User Name and Password on this page. Omitting or using an incorrect User Name and/or Password results in the test not completing or completing with an error. In either case, you will not obtain successful test results even if the Directory Provider information you supplied is valid and correctly entered into ElectricFlow.

Three tests validate the information you supplied:

- user authentication
- user identified in a directory provider
- find all groups where the user is a member

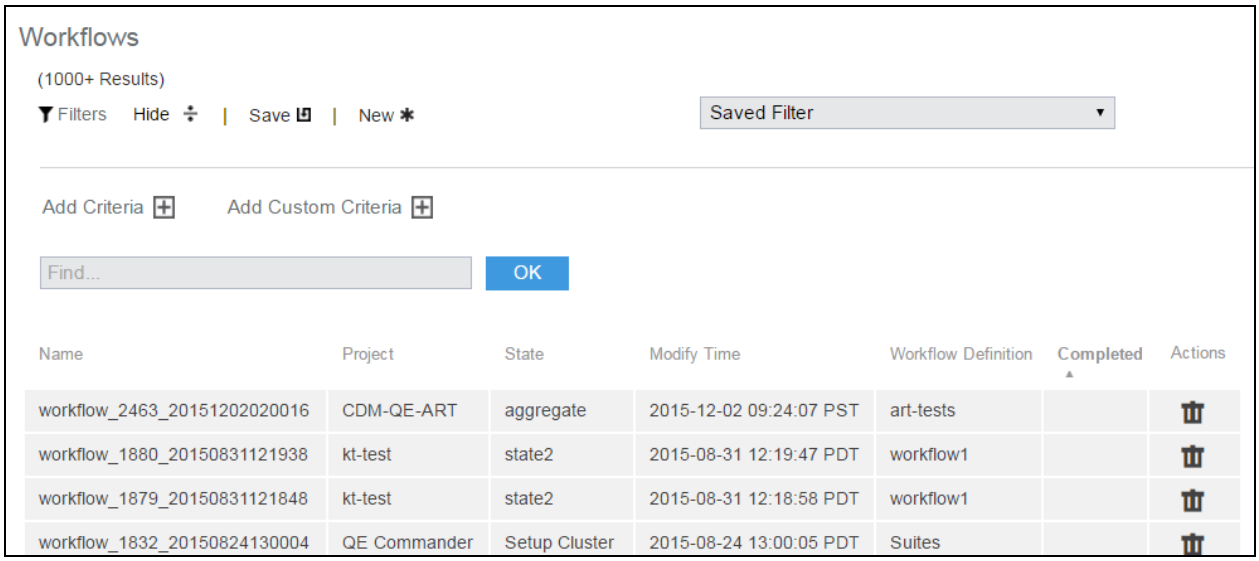
If there is a test failure, correct the information you supplied and retest by clicking the **Test** button.

If you do not need group information at this time, and did not enter information in the Group section, the group test fails without this information, but the test successfully authenticates users if both user tests are successful.

Workflows

This page displays all workflows in the ElectricFlow system, both running and completed workflows.

You can search for workflows and save the workflow search filters for later use. See Context Searching and Filtering on page 1244 for details on using the search capabilities on this page.



Links and actions at the top of the table

- Sort the Name, Project, State, Modify Time, Workflow Definition, and Completed columns by clicking on the column name.
  After sorting a column, the page changes to a Workflow Search Results page and more search options are available.
- The "star" icon allows you to save this web page to your Home page.

Column descriptions

Name—This is the ElectricFlow-generated workflow name, which is defined by the Workflow Name Template setting. Click on any workflow name to go to that workflow's Workflow Details page.

Project—The name of the project containing this workflow. Click on any project name to go to the Project Details page for that project.

State—The current state of the workflow.

Modify Time—The time this workflow was modified.

Workflow Definition—The name of the workflow definition that ran to create this workflow. Click on any workflow definition name to go to that workflow definition's Workflow Definition Details page.

Completed—A check mark in this column indicates the workflow has completed.

Actions—**Delete**—Use this link to delete the workflow on the same row.

Workflow Definition—create new or edit existing workflow definition

To create a new workflow definition

Enter information into the fields as follows:

Field Name	Description
Name	Enter a unique workflow definition name of your choice. You may want your workflow definition name to reflect the project where it belongs or its purpose. For example, you might set a name based on a product or team using this workflow.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.

Workflow Name Template	This is the template used to determine the default name of jobs launched from the workflow definition.
	For example:
	`$[projectName]_$[/increment /myproject/workflowCounter]_ $[/timestamp]`
	(substitute your values for the names above)
	Produces a workflow name like:
	`projectFoo_123_20140102130321`
	Enter any combination of elements to create workflow names more meaningful to you. For example, you could choose to include the build number.
	Note: Electric Cloud does not recommend using the ElectricFlow-generated `workflowId` because it is no longer a human-readable integer so it does not provide any identifiable information and cannot be used as a counter.

Click **OK** to save the information you entered.

Your new workflow definition will appear on the Project Details > Workflow Definitions page.

To edit an existing workflow definition

1. To rename the workflow definition, type a new name in the **Name** field and click **OK**.
2. Change or add to the description or modify the template.
3. Click **OK** when your edits are complete.

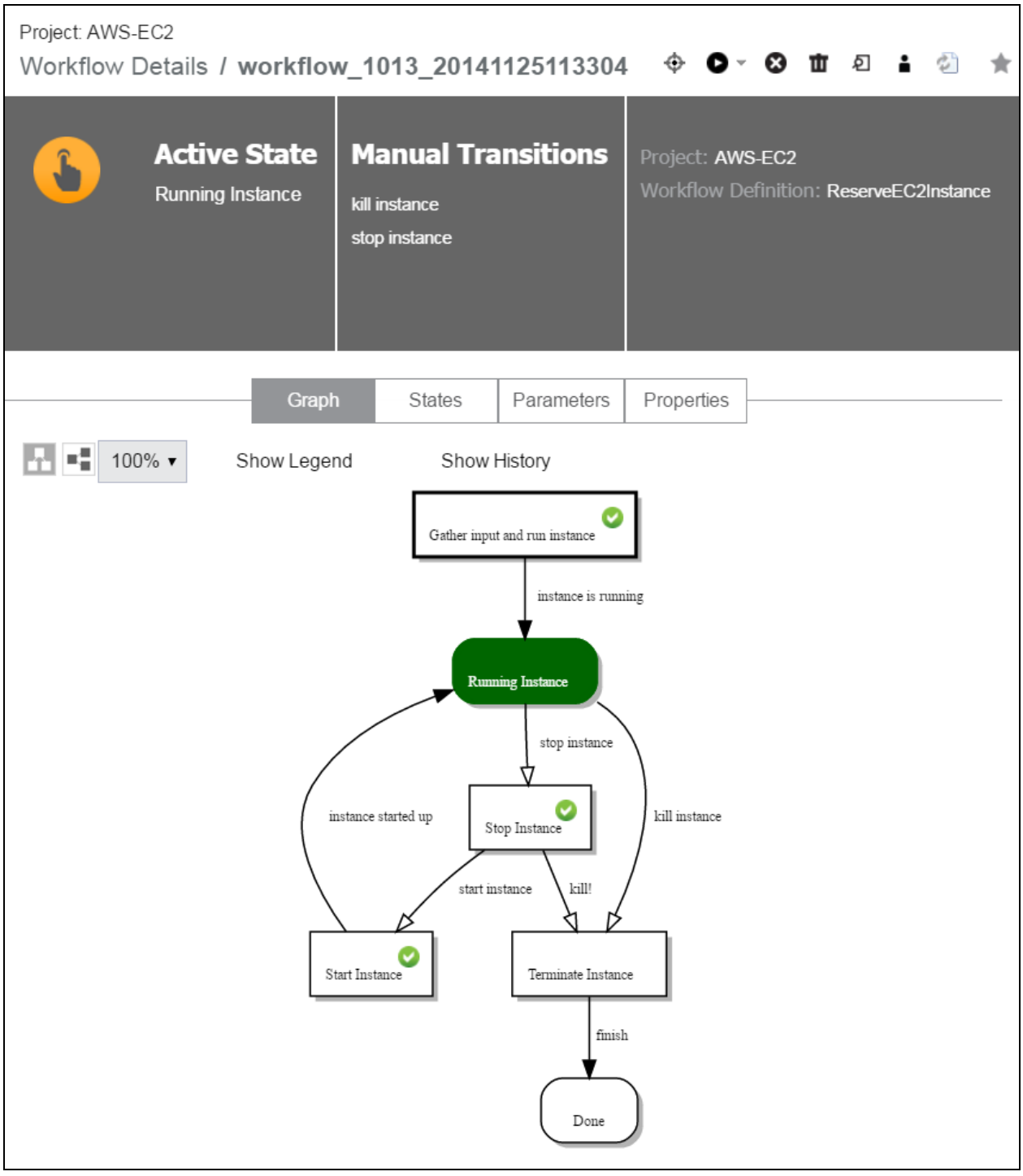
Workflow Definition Details

This Help topic provides a detailed overview for all views and functionality on this web page. You may want to read this topic in its entirety if this is your first experience with the ElectricFlow Workflow feature. If you are an experienced workflow user, the following quick links will take you to a specific topic section for quick review.

- Graph view
- List view
- Properties view

This page displays workflow definition components, including state and transition definition names, types, actions, descriptions, and various links to create additional transitions and state definitions, define access control, run the workflow, and so on. You can view any previously defined workflow objects and create new properties too.

When you open the Graph view, your first glance at this page will look similar to the following:



Links and actions at the top of the page:

- Above the Workflow Definition Details page title you see (in this example) "Project: Upgrade-End to End". This breadcrumb information tells you which project you are viewing or working with and provides a link back to the project, itself.

- Immediately after the page title, you see "workflow_9", which (for our example) is the name of this workflow.
- Track Changes ()–Click this to open the Change History page.
- **Run** ()- Use this link to run the workflow definition. Hover your mouse over the drop-down arrow to see these choices:
  - Selecting **Run...**—allows you to pick the "starting state" to run the workflow. After clicking **OK** from the Run Workflow pop-up dialog box, the Run Workflow page is displayed. If you previously created parameters for this workflow, they are displayed. You can accept the parameter values or change them before running the workflow.
  - Selecting **Run Immediately**—this option uses the first (default) "starting state" to run the workflow.
    Note: Clicking Run, without using the down-arrow to make a selection, is the same as selecting Run Immediately.
- **Access Control**—Use this link to set privileges for this workflow definition. For details, see the Access Control Help topic.
- The "star" icon allows you to save this workflow definition to your Home page.

Navigation and view summary

Each button has its own expanded description section following this summary.

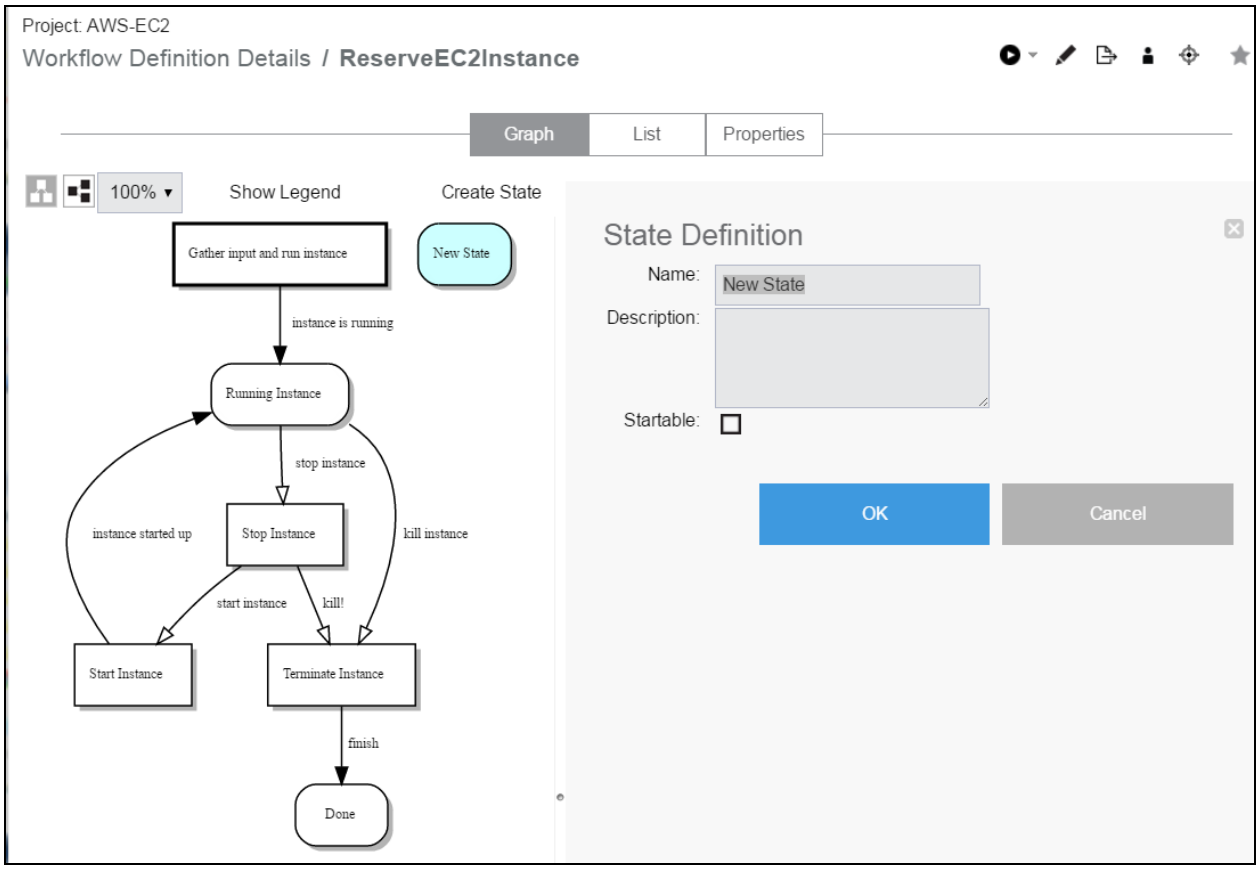
- Graph—the page opens in this view. The main portion of this view contains the work area for visually creating your workflow.
- List—this view provides a table displaying states, transitions, and so on in list form.
- Properties—this view provides a table displaying previously created properties and provides links to create additional properties.
- Show Legend—displays a legend for workflow objects.
- Create State—this button produces the State Definition panel where you can create new states as you build or modify your workflow.
- The multi-square directional icons allow you to change your workflow graph from vertical to horizontal, depending on how you choose to see the graph.
- The drop-down menu for percentages allows you to diminish or expand the workflow graph size.

Graph view

To create a workflow, minimally you need to create state definitions and transition definitions. Optimally, those objects will have parameters and properties, and you will want to create or enforce access control and perhaps create one or more email notifiers too. To begin, the following sections describing functionality in this view will familiarize you with how to create workflow objects and end with exploring the graph's quick-access features.

Click the **Create State** button to create a new state using the State Definition panel.

In the following screen example, you see a New State object added next to the Start state in an existing workflow graph.



You can use the DSL Export ( ) button to download the objects as a DSL file.

To create a new state

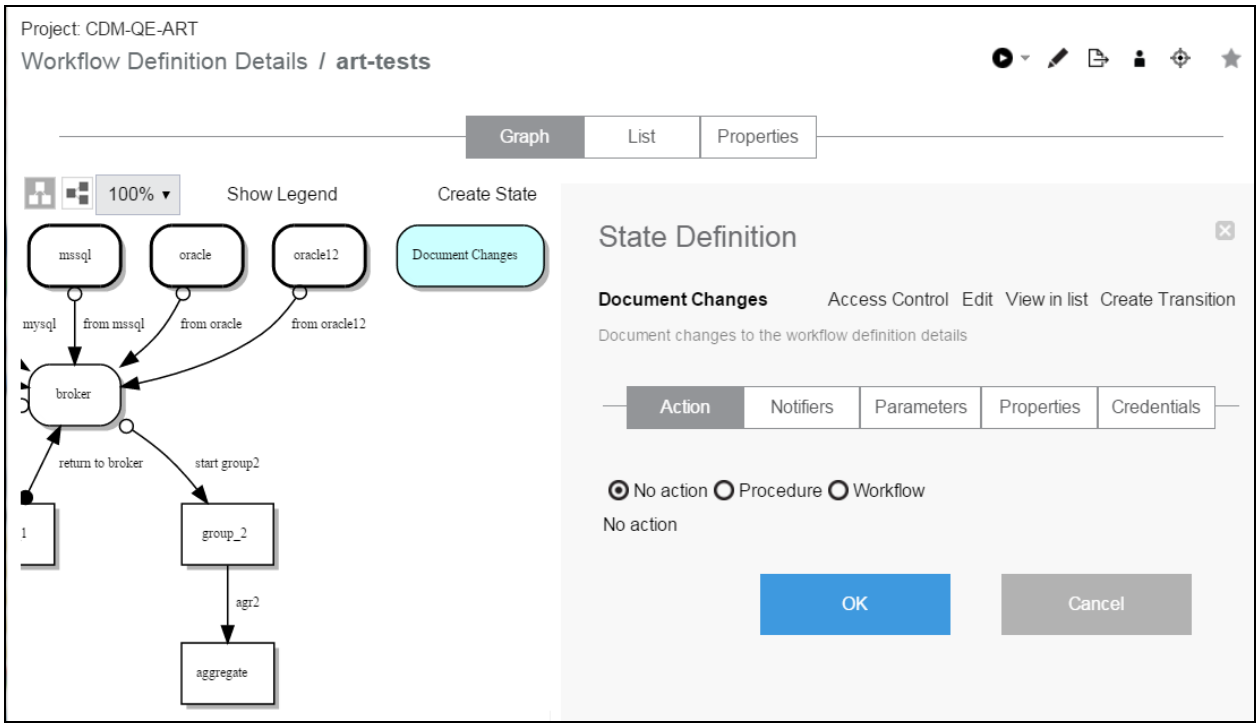
After clicking the Create State button, enter information in the State Definition panel as follows:

Field Name	Description
Name	(Required) Provide any name of your choice. The name must be unique to this workflow definition. If you do not provide a name, a system-generated name is created: New State, New State 2, and so on.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.

Field Name	Description
Startable	Select this checkbox if you want this state to be your workflow "starting" state. The first state in any workflow definition is saved as *Startable*, but you can change this later.

Click **OK** and the State Definition panel provides options to configure your new State Definition.

To configure the new state definition



In the State Definition Panel, the state name appears at the top. In this example in this topic, "New State" is the state name.

Configure the following settings.

Links at the top of the panel

- **Access Control**—use this link to add or modify access control privileges for this state (link not illustrated in the State Definition panel example above).
- **Edit**—use this link to modify this state's name, description, or whether or not the state is *Startable*.
- **View in list**—displays the List view to see the state definition in the Workflow Definition Details table. This state will be highlighted in the List view.
- **Create Transition**—displays the Transition Definition panel to create a transition for this state.

The "tabbed" or button sections

Action

As each state in a workflow becomes active, it performs an *action*. The state's action can be to create a job from a procedure or to start a workflow from a workflow definition. When the job or called workflow completes, On Completion transitions are evaluated to see if the workflow should change to a different state, possibly based on the outcome of the action.

Selecting *No action*, *Procedure*, or *Workflow* changes the fields that appear under these choices.

- No action—if selected, no other information is required. Click **OK**.
- Procedure—this selection displays the following fields:
  - Current—this selection refers to the project that contains the workflow definition.
  - Project—if selected to call a different project, start typing in the text box to display a list of (non-plugin) projects from which to select. Only the first 10 matches are displayed. Type more characters to refine the list.
  - Plugin—if selected, start typing in the text box to display a list of plugin projects from which to choose. Only the first 10 matches are displayed. Type more characters to refine the list.
  - Procedure—start typing in this field to see a list of available procedures from which to select, depending on your previous choice of Current, Project, or Plugin. Only the first 10 matches are displayed. Type more characters to refine the list.
- Workflow—this selection displays the following fields:
  - Similar to selecting Procedure (above), select Current, Project, or Plugin.
  - Workflow—if selected to call another workflow definition, start typing in this text box to display a list of workflow definitions from which to choose. Only the first 10 matches are displayed. Type more characters to refine the list.
  - Starting State—start typing in this field to see a list of available starting states definitions in the workflow definition you selected. Only the first 10 matches are displayed. Type more characters to refine the list.

After selecting the procedure or starting state, press the <Tab> key to leave the field and see a list of parameters for the selected action. After entering any procedure values, click **OK** to save the changes.

Notifiers

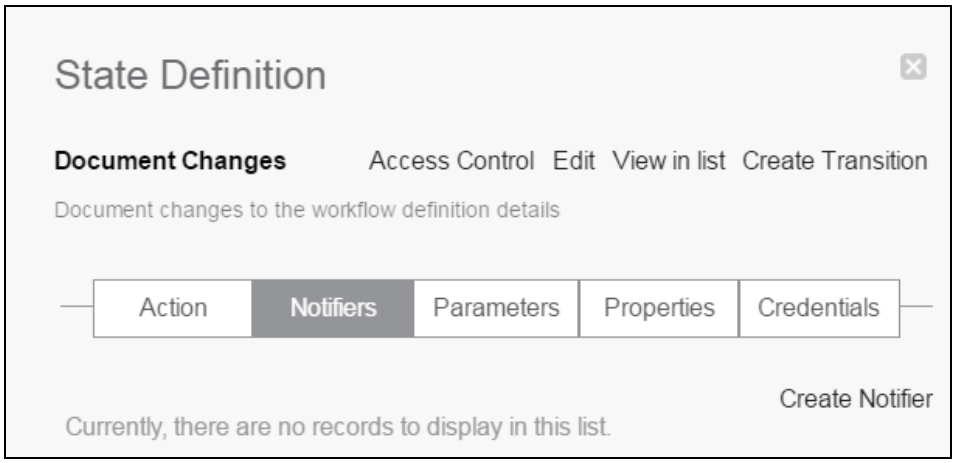
Select this tab to see a list of email notifiers previously created for this state definition or to create an email notifier. Notifiers are commonly used to inform interested parties about transitions to the state being defined.

> **Note: Note:** Before you can set up an email notifier, you need an email configuration. If you have not already defined an email configuration, you can do it now. Go to Administration > Email Configurations and select the **Add Configuration** link.

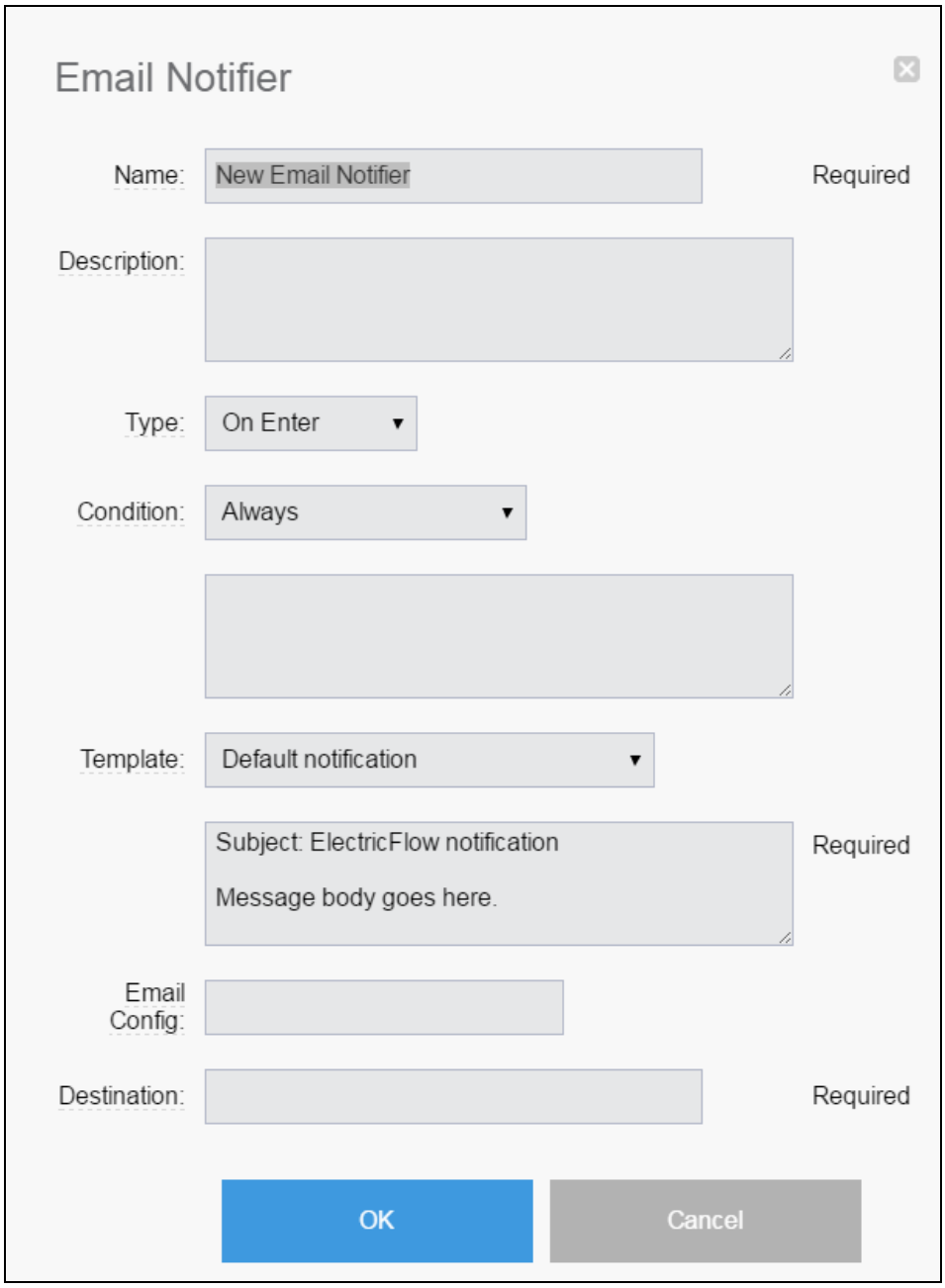
The Notifiers table contains a list of all previous created notifiers for this state definition.

Column descriptions

Column Name	Description / Actions
Name	The name of the email notifier. Select a name to go to the "edit" page for that notifier if you need to make changes.
Type	The type of email notifier specified during creation.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Actions	**Delete**—Use this link to delete the notifier on that row.



To create an email notifier, click the **Create Notifier** link to access the Email Notifier panel.



Enter information into the fields as follows:

Field Name	Description
Name	(Required) This name can be any text string you choose. The name must be unique among other notifier names in this project, on procedures or workflow definitions called from other projects.

Field Name	Description
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Type	Use the pull-down menu to choose a type: • On Completion Notifier—sends an email after the state's job or workflow completes. If no job or workflow is defined for that state, these notifiers will not be sent. • On Enter Notifier—sends an email when the state becomes the workflow's active state. • On Start Notifier—sends an email after the state's job or workflow starts. If no job or workflow is defined for that state, these notifiers will not be sent.
Condition	Use the pull-down menu to select the type of condition you need for this email notifier. Edit the auto-supplied condition in the text box or add a completely new script for your purpose. The condition specifies whether the notifier should send a message depending on the result of a property expansion. If the result is empty or non-zero, the message is sent. If the result is "0", the message is not sent.
Template	Use the pull-down menu to select from a list of global, ready-to-use formatting templates. Depending on the type of email notifier you are creating, the available template choices in the drop-down menu will be different. (Required) To customize your template, edit the auto-supplied text in the text box or you can add a completely new script for your purpose. Any edits made in this text box will not be saved to the global template. To create a custom template, the basic structure is: • zero or more email header lines • blank line • message body
Email Configuration	Click inside this field or start typing to bring up a list of possible email configuration names. An email notifier that does not specify an email configuration will use the configuration named 'default' if it exists.
Destinations	(Required) This is a space-separated list of valid email addresses, email aliases, or ElectricFlow user or group names, or a property reference that expands into such a list, or you can enter an LDAP DL name (group name).

Click **OK** to save your email notifier configuration. The next time you view the Notifier table, you will see this notifier in the list.

Parameters

Similar to Notifiers, a parameter table is displayed containing all formal parameters defined for this state definition. Parameters are presented when launching the workflow definition (if it is startable), or when taking a manual transition to the state. A transition may enter values for some or all of the formal parameters defined by the state definition, in which case only the unmapped parameters are presented to the user.

- **Create Parameter**—Use this link to go to the New Parameter page to create a parameter for this state definition.

  If you need assistance with creating a parameter, click the **Help** link in the upper-right corner of the New Parameter page.

Properties

Similar to the Notifiers and Parameters table, a properties table is displayed containing available properties for this state definition. If no properties were defined for this state definition, the table does not exist.

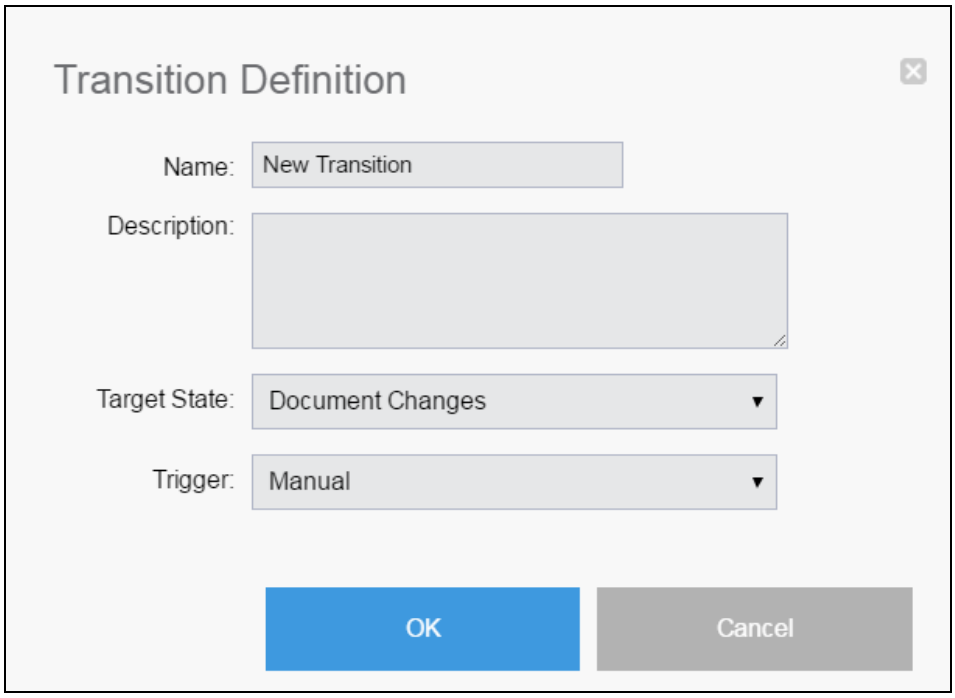
Click one of the available links to create one or more properties that will be displayed in the table when created:

- **Create Property**—Click this link to go to the New Property pop-up box to create a new property for this state definition.
- **Create Nested Sheet**—Click this link to go the New Nested Property Sheet pop-up box to create a nested property.
- **Access Control**—Click this link to go to the Access Control page for the property sheet.

The property pop-up boxes contain a Help link if you need assistance creating a property.

To create a transition definition

On the State Definitions panel, click the **Create Transition** link to see the Transition Definition panel:



Enter information into the fields as follows:

Field Name	Description
Name	(Required) Enter a unique name for the transition definition. It can be any name you choose, but the name must be unique within the state definition. If you do not provide a name, a system-generated name is created: *New Transition*, *New Transition 2*, and so on.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Target State	This field displays the "best guess" for the target state. If not correct, or you intended to create a transition for a different state, use the down-arrow to choose an available target state from the list.

Field Name	Description
Trigger	Use the drop-down menu to choose the transition type. The four types of transitions are: • On Completion—transition occurs when the action completes. These transitions are ignored if no action is specified for the source state. **Note:** On Completion transitions are taken only if the state is still active when the action completes, and are ignored if the workflow has transitioned to another state—this can occur if an On Start or Manual transition occurred before the action completed. • On Enter—transition occurs before sending notifiers or starting the action. • On Start—transition occurs immediately after starting the action. These transitions are ignored if no action is specified for the source state. • Manual—transition occurs when a user selects the transition in the UI and specifies parameters. The same action can occur using ectool or the Perl API by calling `transitionWorkflow`. Only users who have "execute" permission on the transition are allowed to use a Manual transition.
Condition	Use the drop-down menu to choose a condition. After selection, the text box is populated with a sample JavaScript string to edit for your purposes. **Note:** If you select Custom, no sample is provided. Enter a script in the text box. We recommend using JavaScript.

Click **OK** to save your transition definition.

Notice the new links and options available now:

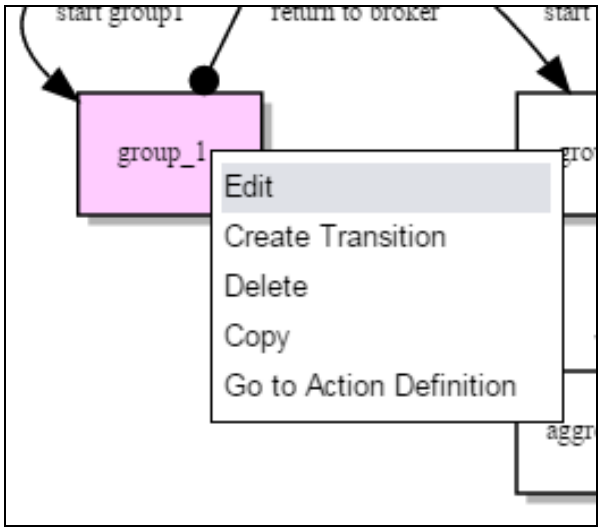
- **Access Control**–Use this link to add or modify access control privileges to this transition definition.
- **Edit** –Use this link to modify the transition definition.
- **View in list**–Use this link to see the new transition definition in the List view (table) on the Workflow Definition Details page. You can use the List view to move the transition to a different position within the state definition.
- Parameters–If parameters were defined on the State Definition Details page, you will not see them here. If parameters are defined on this page, you need to enter values. Manual transitions allow you to defer parameter assignment until transition time.
- Properties–Displays properties available for this object or you can create one or more properties at this time.

Using the graph's "quick-access" features

The following screen example is a portion of a workflow graph. In the graph, states and transitions are links:

- Click on a state to open the State Definition panel to modify that state.
- Click on a transition to open the Transition Definition panel to modify that transition. (Hover your mouse over a transition or transition name and click when it changes color.)

However, if you ***right-click*** on a state or transition, a context-sensitive menu with other options is available. In the following example, right-clicking on the "group_1" state provided the quick-link pop-up menu.



Available pop-up links include:

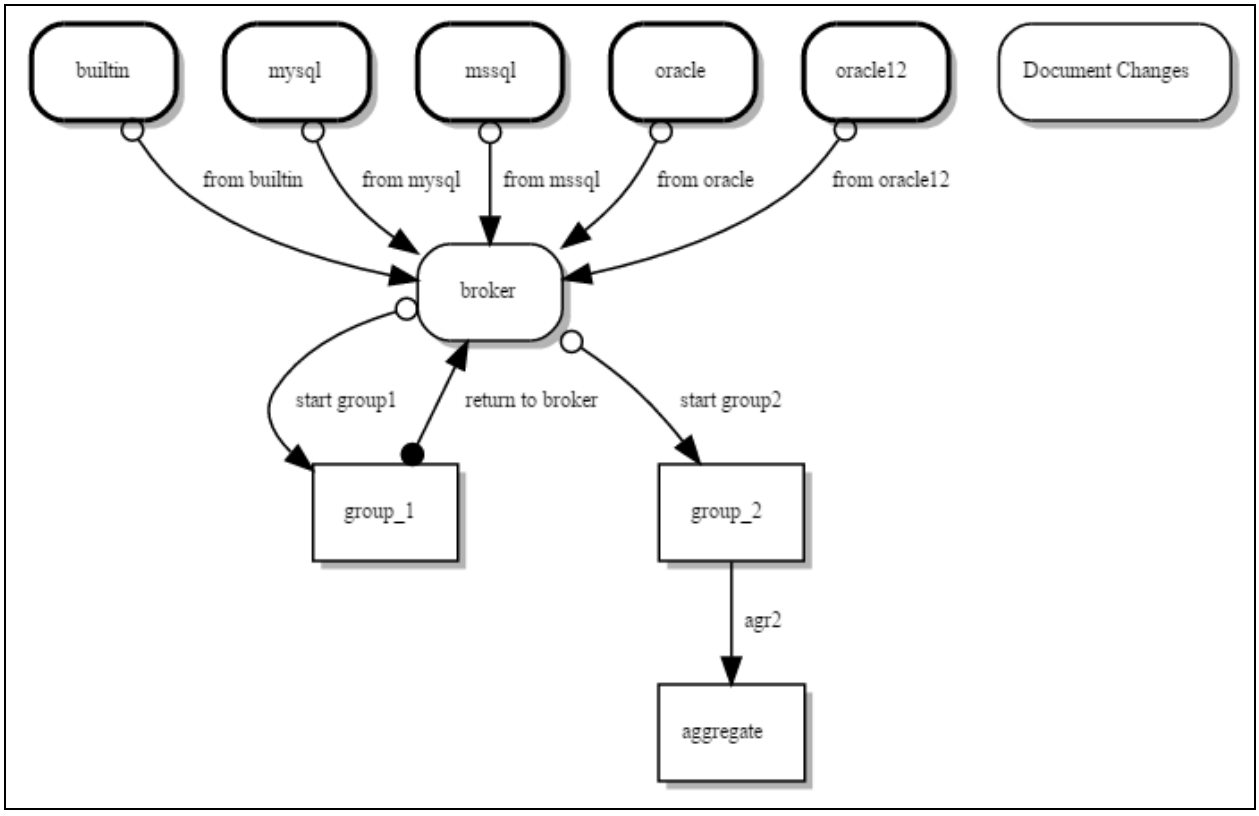
- For states calling a procedure or another workflow, menu choices can be:
  - Edit—opens the State Definition panel to make modifications. This is the same as selecting the Edit link on the State Definition panel.
  - Create Transition—opens the Transition Definition panel.
  - Delete—deletes the state.
  - Copy—makes a copy of the state with all associated transitions preserved.
  - Go to Action Definition—for states calling a procedure, the Procedure Details page is opened. For states calling a workflow, the Workflow Definition Details page for the "subworkflow" is opened.
- For states with "No action", the choices are the same, except the "Go to Action Definition" option is not available.
- For transitions, menu options include: Rename, Delete, and Copy.
- Right-clicking anywhere on the graph canvas provides two more choices:
  - Create State—allows you to quickly create a new state. This is the same as clicking the Create State button at the top of the graph.

- Rotate Graph—acts as a "toggle", rotating the graph from a vertical to horizontal view and back again.

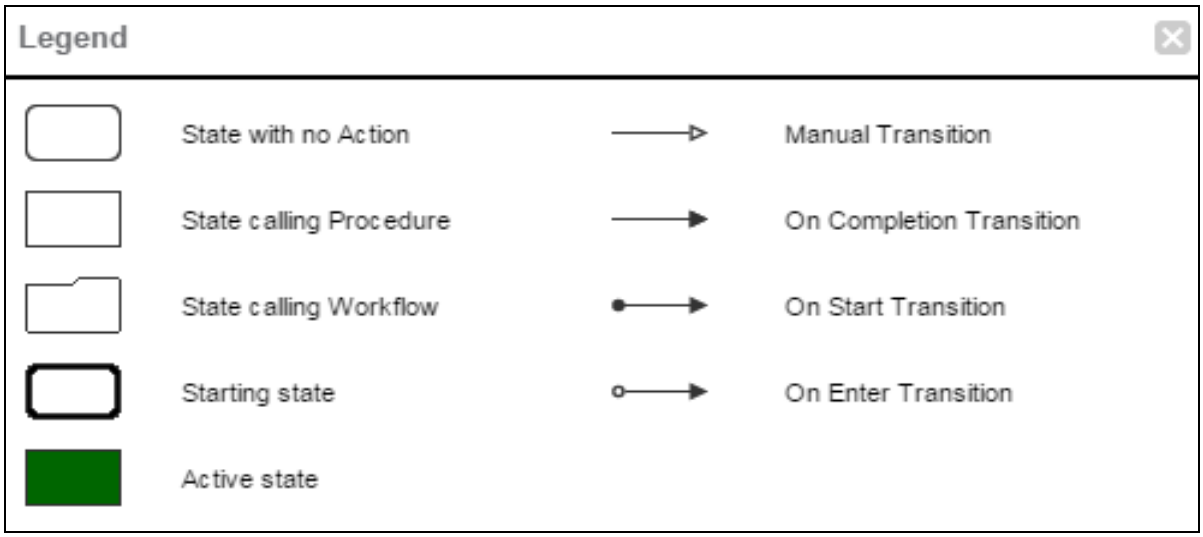
Show Legend

The following screen is another example of a workflow graph. Depending on your workflow, your graph could be very simple or much more complex than the example below.

Notice the transition line-endings and different shapes used for states.



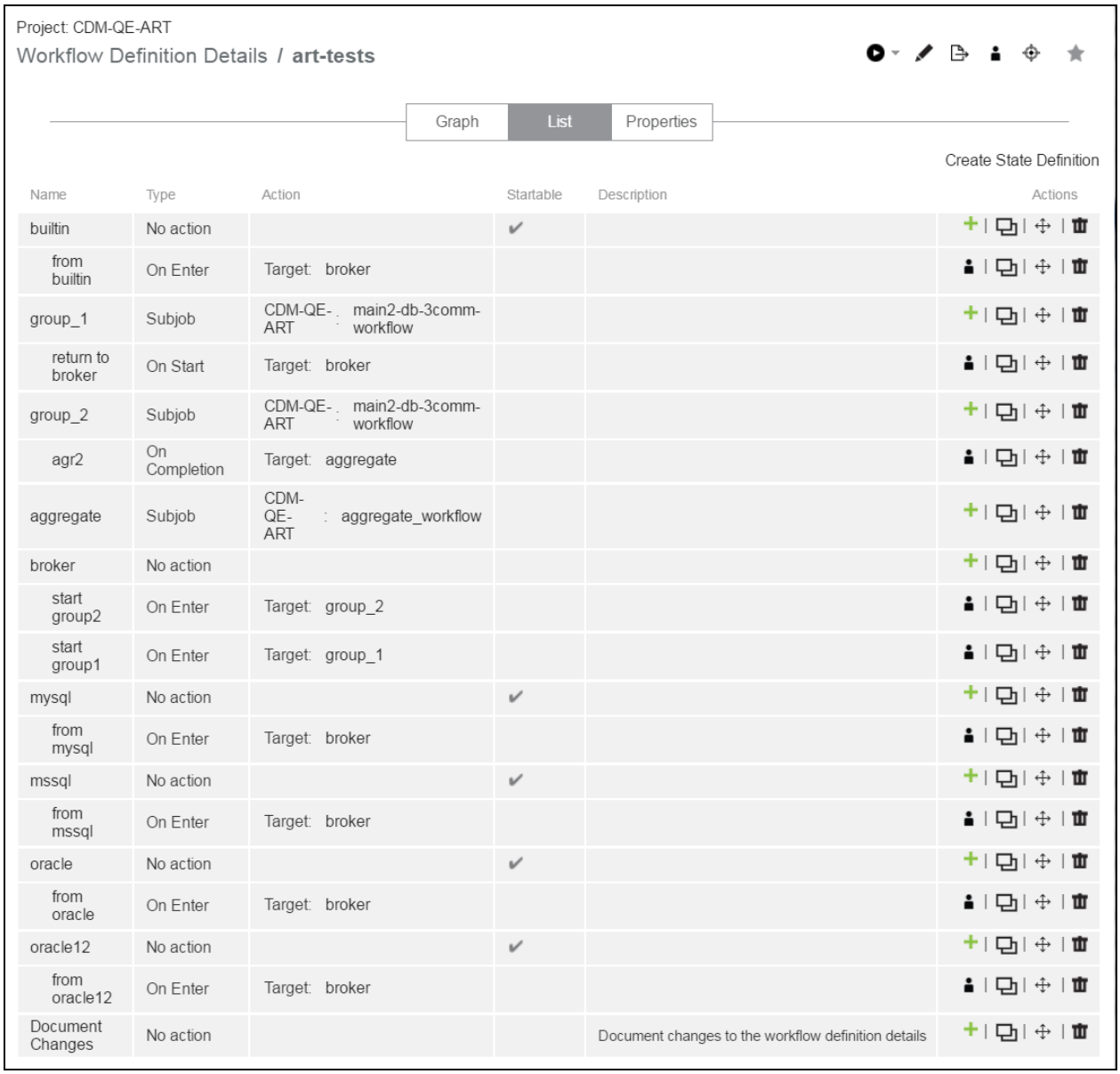
The legend is available for reference anytime to help you become familiar with state shapes and line-ending definitions.



List view

This view will be empty until you build a workflow in the Graph view. You can view this page at any time to see a list of states and transitions in your workflow definition.

The following screen is an example of a populated Workflow Definition Details page. A **Create State Definition** link is available above the Action column, which has the same function as clicking the Create State button in the Graph view.



Project: CDM-QE-ART

Workflow Definition Details / **art-tests**

Graph | List | Properties

Create State Definition

Name	Type	Action	Startable	Description	Actions
builtin	No action		✔		+ | | |
from builtin	On Enter	Target: broker			| | |
group_1	Subjob	CDM-QE-ART : main2-db-3comm-workflow			+ | | |
return to broker	On Start	Target: broker			| | |
group_2	Subjob	CDM-QE-ART : main2-db-3comm-workflow			+ | | |
agr2	On Completion	Target: aggregate			| | |
aggregate	Subjob	CDM-QE-ART : aggregate_workflow			+ | | |
broker	No action				+ | | |
start group2	On Enter	Target: group_2			| | |
start group1	On Enter	Target: group_1			| | |
mysql	No action		✔		+ | | |
from mysql	On Enter	Target: broker			| | |
mssql	No action		✔		+ | | |
from mssql	On Enter	Target: broker			| | |
oracle	No action		✔		+ | | |
from oracle	On Enter	Target: broker			| | |
oracle12	No action		✔		+ | | |
from oracle12	On Enter	Target: broker			| | |
Document Changes	No action			Document changes to the workflow definition details	+ | | |

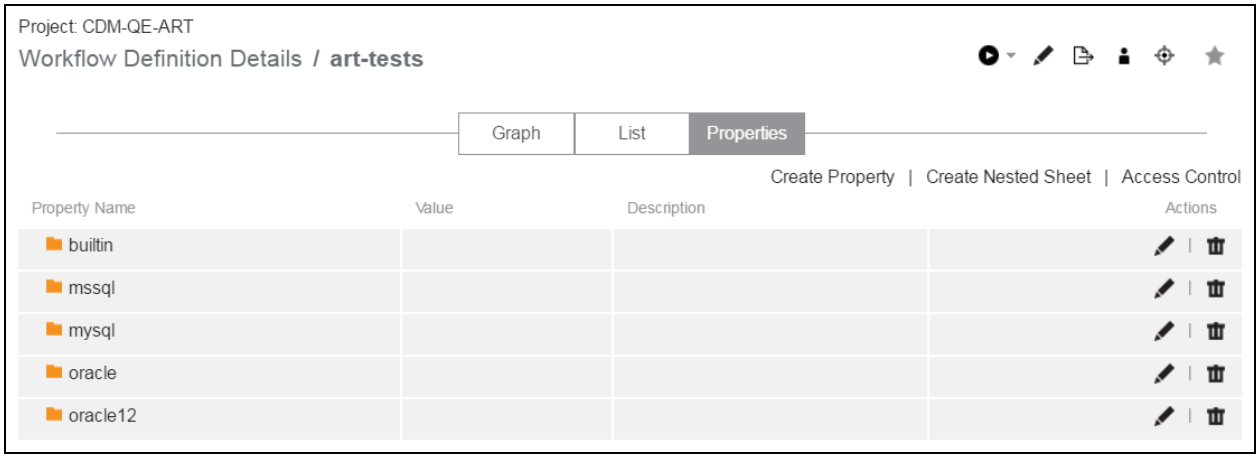
Column Name	Description / Actions
Name	This column displays state definition and transition definition names currently defined for this workflow definition. • Transition definition names are indented under a state definition name. • Clicking a state definition name displays the State Definition panel (in the Graph view) for that state to see how it was defined or to make modifications. • Clicking a transition definition name displays the Transition Definition panel (in the Graph view) for that transition to see how it was defined or to make modifications.
Type	This is the type of action or trigger currently assigned to the state definition or transition definition. • For states—the type may show calling a subjob, subworkflow, or no action. • For transitions—the trigger type is provided.
Action	This is the action previously defined for the state or transition. • For states—(where the Type is "subjob") the format for actions is `<projectName>:<procedureName>`. Clicking the first link displays the Project Details page for the workflow project. Clicking the second link displays the Procedure Details page for the procedure used by that state. • For states—(where the Type is subworkflow) the format for actions is `<projectName>:<workflowName>`. Clicking the first link displays the Project Details page for the workflow project. Clicking the second link displays the Workflow Definition Details page for the workflow used by that state. • For transitions—clicking the Target action displays the State Definitions panel in the Graph view.
Startable	A check mark in this column shows this state was defined as startable, which means this state is the beginning point for your workflow.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.

Column Name	Description / Actions
Actions	You can perform some or all of these actions: • **Create Transition**—Click this link to go to the Transition Definition panel to create a new transition definition for this state. • **Access Control**—Click this link to go to the Access Control page to define access privileges for this transition definition. • **Copy**—Use this link to make a copy of either the state definition or the transition definition. **Note:** Copying a state also copies its transitions, however, you must have modify privileges on the workflow project to make a copy. • **Move**—Use this link to move a state to another position in the workflow. You will be prompted to choose a state name to move this state to a location ***before*** that state. If moving a transition, note that transition can be moved only to another location for the same state—transitions cannot be moved from one state to another state. • **Delete**—Use this link to delete the state definition or transition definition on the same row.

Properties view

Select the Properties view button to see the Properties table. If no properties were defined for this workflow definition, no table appears.

The following screen is an example of the Properties table.



The following links and actions are available in the Properties table:

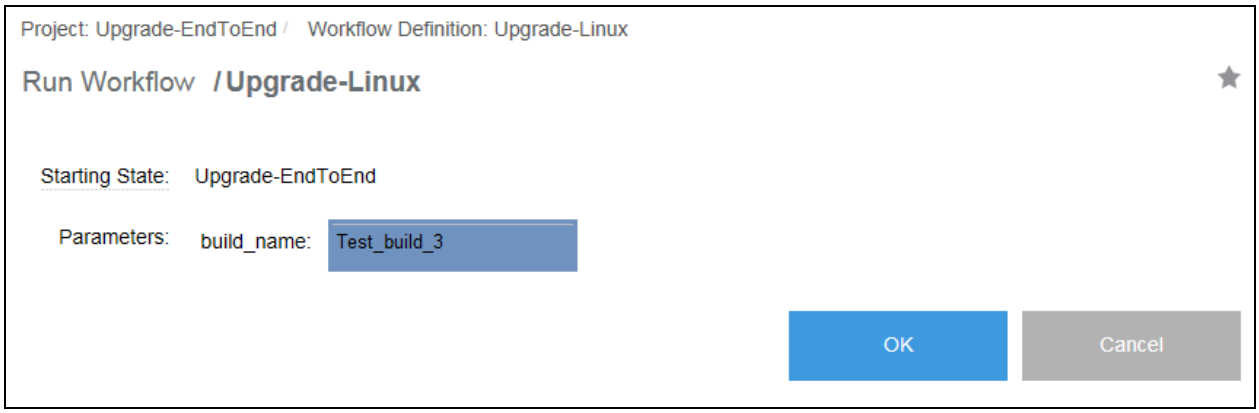
- Links and actions at the top of the table
    - **Create Property**—Click this link to go to the New Property pop-up box to create a new property for this workflow definition.
    - **Create Nested Sheet**—Click this link to go the New Nested Property Sheet popup to create a nested property.
    - **Access Control**—Click this link to change access control privileges on the property sheet.
- Column descriptions

Column name	Description / actions
Property Name	Select a property name to edit that property. You can change its name, value, or add/change its description. If the property name is preceded by a folder icon, this is a property sheet. Click on the property name to open the "folder".
Value	The value currently assigned to the property.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Actions	**Delete**—Click this link to delete the property on that row.

Run Workflow

Use this page to run a workflow.

- The "star" icon allows you to save this job information to your Home page.
- The "bread crumbs" Project: Upgrade-End to End / Workflow: Upgrade-Linux provide links to a previous web page.
- The name after the Run Workflow page title is the name of the workflow you intend to run.



Starting State—This is the name of the current starting state for this workflow. If this workflow has multiple starting states and this is not the one you want to use, return to the Run Workflow pop-up dialog and select a different starting state.

Parameters:

- Any parameters previously specified for this starting start are displayed. If no parameters were defined, this area will be blank.
- If values are supplied, these are the default values specified when the parameter was created.
  If necessary, you can "type-over" these values to change them before running this workflow.
- You must enter a value for any blank value field labeled "Required".

Click **OK** to run the workflow when your selections are complete.

Workflow Details

This page displays workflow details, including the workflow state, project name, workflow definition name, and any properties. The following links are provided for quick access to topic sections you may want to review again.

- Graph
- Show Legend
- Using the graph's "quick-access" features
- Show History
- States
- State Details panel
- Parameters
- Properties

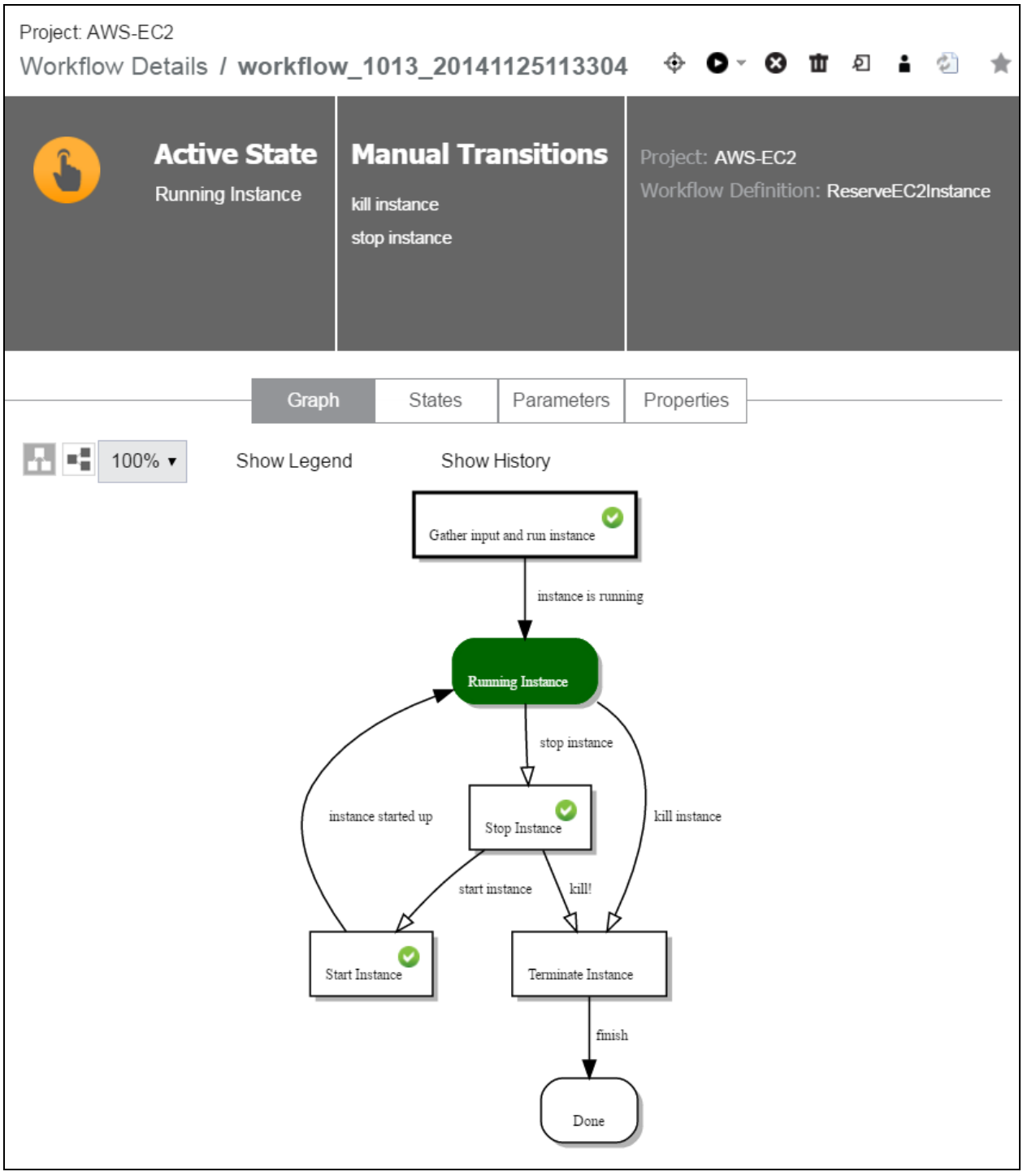
The name after the Workflow Details page title is the name of the workflow created after running a Workflow Definition. The number at the end of the workflow name is the workflow definition ID number. The ID number is auto-generated by ElectricFlow. Your workflow name will be different, depending on the information you supplied in the workflow definition.

Links and actions at the top of the table

Additional links are available if the workflow is running. When the workflow is complete, some links are no longer available.

- **Run Again**—Use this link to run the workflow definition again.
  When you select the **Run Again** button, you are requesting to re-run the workflow definition with the same parameter values as the original invocation.
  If you want to change the parameter values, you can select "**Run...**" from the dropdown menu (small down-arrow) next to the Run Again button.
- **Stop Workflow**—Select this option to stop the workflow. Any unfinished processes will complete, but no new transitions will occur.
- **Delete**—Deletes this workflow.
- **View Log**—Use this link to view the log created by this workflow.
- **Access Control**—Use this link to set privileges for this workflow. For more information, see Access Control.
- **Stop Refresh**—This link is available only if a workflow is currently running. Click this link if you do not want the page to refresh the page automatically while the workflow is running.
- The "star" icon allows you to save this workflow information to your Home page.

While the workflow is running, your Workflow Details page will be similar to the following example:



Summary section—at the top of a running workflow page

- Next to the icon, notice that the state name, "Approval needed", is a link. Click this link to open the State Details panel for this state.

- The **Manual Transitions** section allows you select a manual transition if any were defined. However, if the manual transition contains any parameters whose values were not supplied at definition time, selecting that transition takes you to the Transition Workflow page to enter values.
- The **General Information** section provides links from the Project name to the corresponding Project Details page and from the Workflow Definition name to its corresponding Workflow Definition Details page.

Links

This page can include a Links section also, which will appear on the far right-side of the Summary section. To add a link to this section:

- **Create a link to any file:**
  To add a link, run a command in any job's job step using this format (this works for both local [disconnected] and non-local workspaces):

  ```
  ectool setProperty "/myJobStep/report-urls/<myReportName>"
  "/commander/jobSteps/<jobStepId>/<artifactDirectoryName>/<file-path>"
  ```

  Example command:

  ```
  ectool setProperty "/myJobStep/report-urls/Test Report" "/commander/jobSteps/
  $[/myJobStep/jobStepId]/testreport/index.html"
  ```

  > **Note:** If you are using this format (from ElectricCommander 4.2 and earlier):

  ```
  ectool setProperty "/myJob/report-urls/<myReportName>"
  "/commander/jobs/<jobId>/<workspaceName>/<artifactDirectoryName>]/<file-path>"
  ```

  Example command:

  ```
  ectool setProperty "/myJob/report-urls/Test Report"
  "/commander/jobs/$[/myJob/jobId]/
  $[/myJobStep/workspaceName]/testreport/index.html"
  ```

  It will continue to work only for non-local workspaces. We recommend, however, that you change the command to use the newer format.

- **Create a link to any directory:**
  To add a link, run the following style command in any job's job step:

  ```
  ectool setProperty "/myJobStep/report-urls/Main Job Workspace"
  "file:////WinStor2/scratch/chron55build/$[/myJob/jobName]"
  ```

  > **Note:** Links to a directory automatically work in Internet Explorer, but if using Firefox, local links are disabled unless the default security policy is modified or an extension is used. See http://kb.mozillazine.org/Links_to_local_pages_don%27t_work for details.

When the workflow is complete…

- When the workflow is complete, fewer links are available at the top of the table and the Manual Transitions section is no longer available.

- The check mark icon indicates the workflow is complete.

  Click the state shown under Final State ("Success" in our example), to open that state's State Details panel.

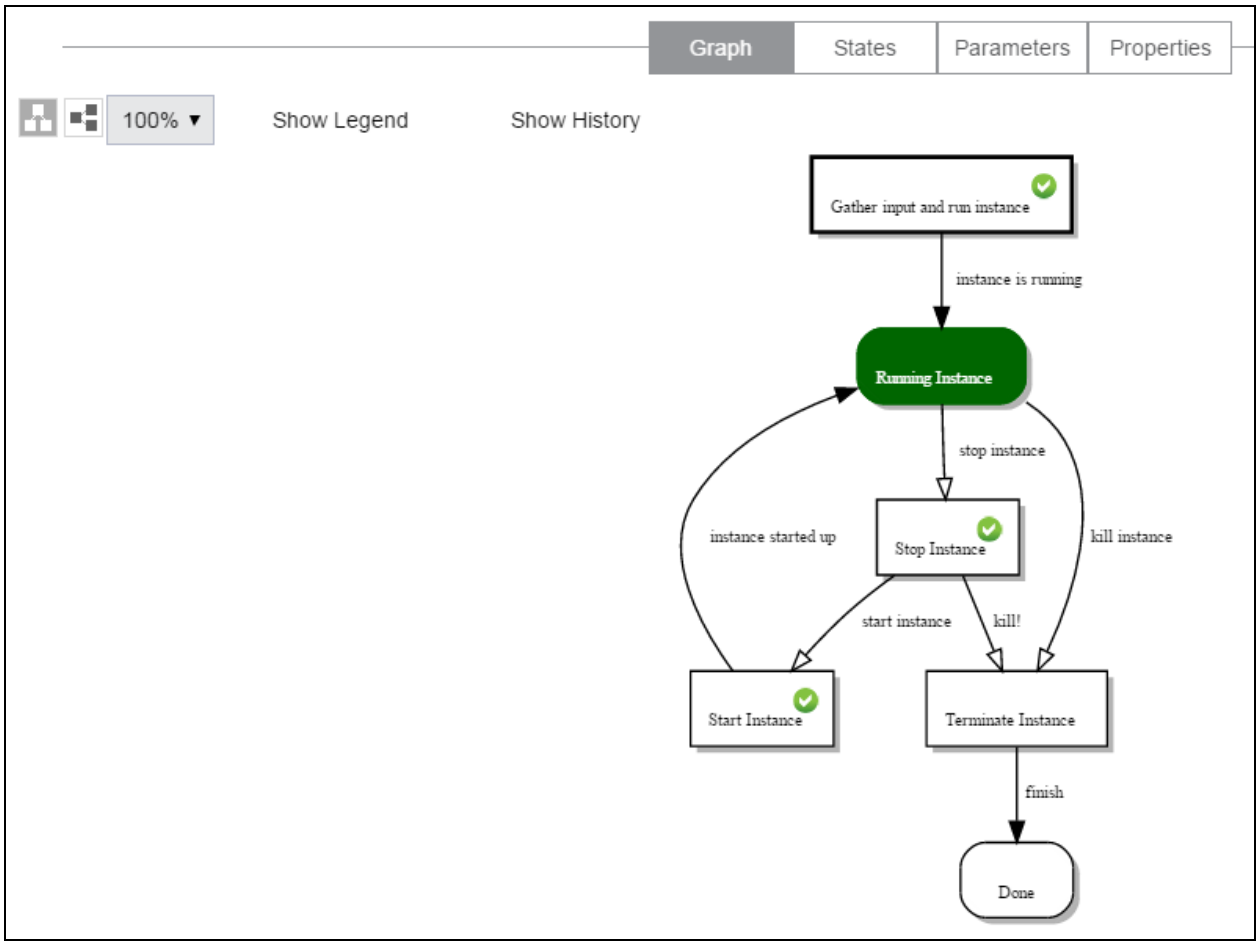
- The **General Information** section is the same as when the workflow was running.

Graph tab

The Graph tab displays your workflow in a visual format. This format is the default view. The following screen is an example of a workflow graph. Depending on your workflow, your graph could be very simple or much more complex than the example below.

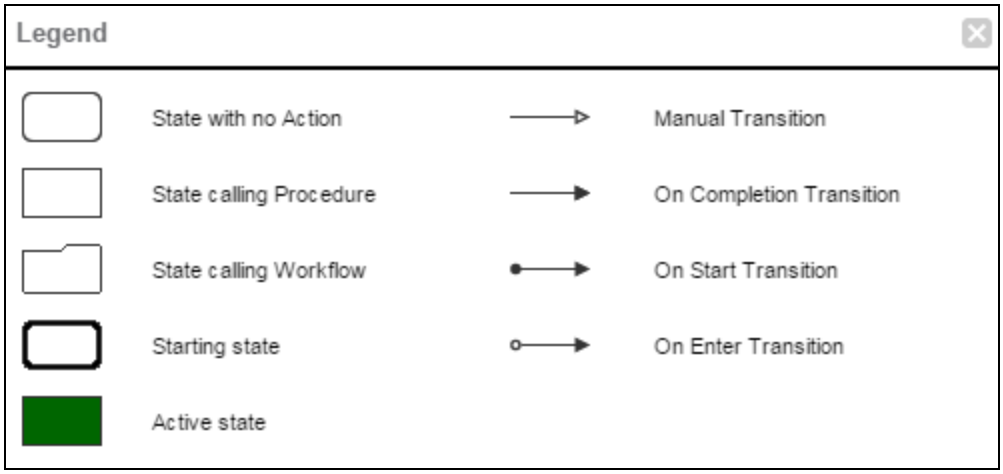
This graph updates in real time, which means you will see the "active" state highlighted with a color. An icon on the state represents an action.

Use the icons below the Graph tab to rotate the graph for better use of your screen viewing area, and you can re-size the graph using the "%" drop-down menu to choose another size.



Show Legend tab

The graph above illustrates all state and transition types. The state and transition names in this graph are labels for the type of state or transition represented. Use the Show Legend tab to see a quick reminder of state shape and transition line-ending definitions.



Using the graph's "quick-access" features

The following screen example is a portion of a workflow graph.

In the graph, states and transitions are links:

- Clicking on a state takes you to the State Details panel to that state's transitions, parameters, and properties.
- Clicking on a transition takes you to the Transition Details panel to see details specifically about that transition, including its condition information.
  (Hover your mouse over the transition or transition name and click when the color changes.)

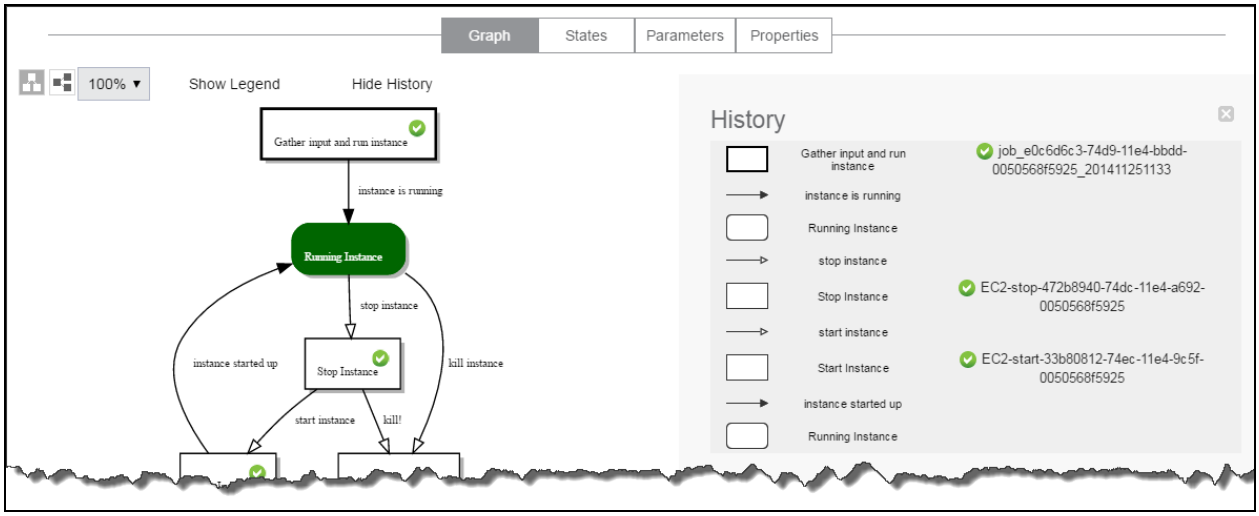
However, if you right-click on a state or transition, other link options are available. These options are different for a running or completed workflow versus the options on the Workflow Definition Details page that were available to you while you were defining your workflow.

Available pop-up context-sensitive links include:

- For states, depending on which actions were defined for the state, you may have one or more of the following options:
  - Go to Action—opens the Workflow Details or Job Details page for that state.
  - Go to Action Definition—for states calling a procedure, the Procedure Details page is opened. For states calling a workflow, the Workflow Definition Details page for the "subworkflow" is opened.
  - Go to State Definition—opens the State Definition panel for that state.
- For transitions, the only link available is: Go to Transition Definition, which opens the Transition Definition panel.

Show History

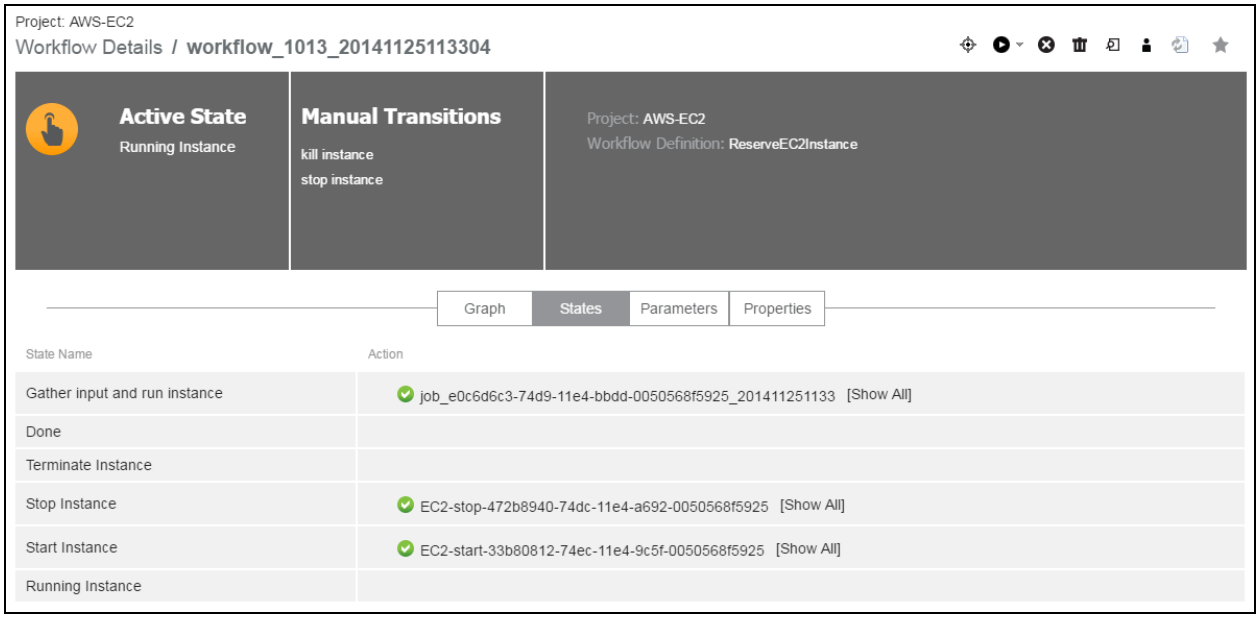
Selecting Show History allows you to see at-a-glance how your workflow progressed through its states and transitions, or for example, how many times the same state was visited and depending on outcome, which transitions were taken and then the outcome of those target states.



Notice the links displayed in the History panel. You can select links for completed jobs or workflows, which display the Job Details or Workflow Details page, respectively.

States tab

Selecting the States tab provides a table containing all state names and actions for this workflow.



The following links are available in the States table:

- State Name—Click any state name in this column to go to that state's State Details panel.
- Action—Actions are available in this column only if you are using a subjob or a subworkflow. If so, links in this column connect you to the Job Details or Workflow Details page for the subjob or subworkflow, respectively.
- [Show All] link—If the state executed multiple times during a workflow run, only the last subjob or subworkflow is displayed. However, clicking on Show All displays the results for all subjobs or subworkflows executed during a workflow run.

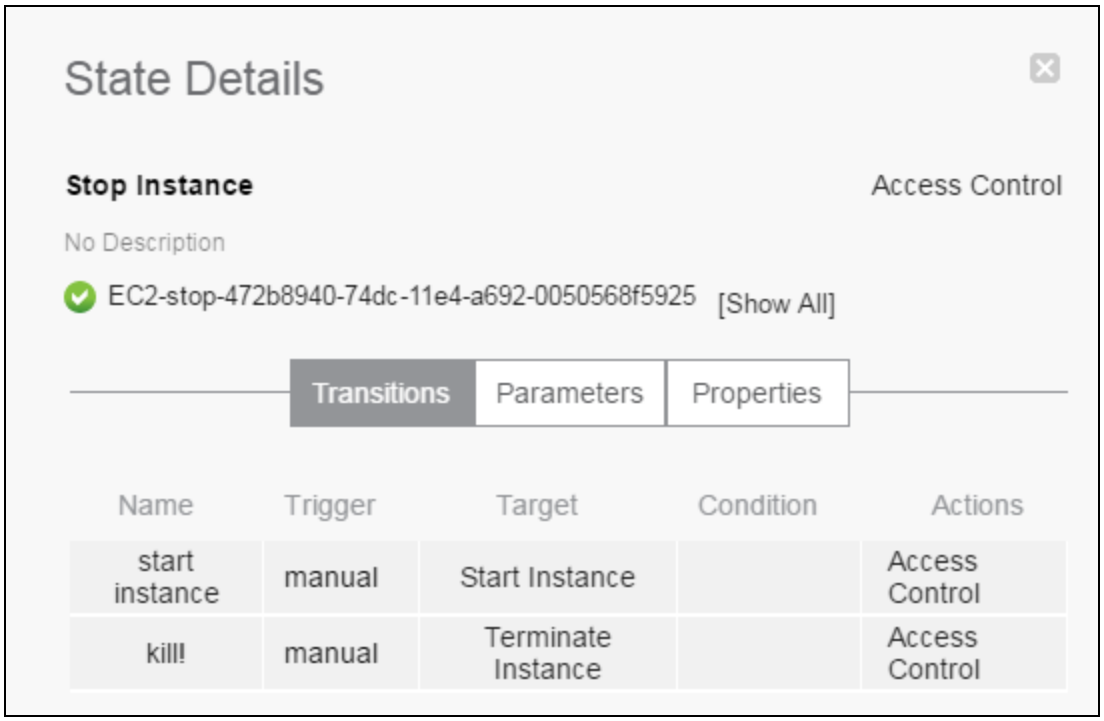
State Details panel

This panel displays state components, including transitions, parameters, and properties.

- Use the **Access Control** link to set privileges for this state. For details, see Access Control.
- The state name, "Build" (in our example), is shown at the top of the panel.
- Job or Workflow status links to the job or workflow created by the action.
- The Transitions tab is open to display the transition table listing all transitions for this state.

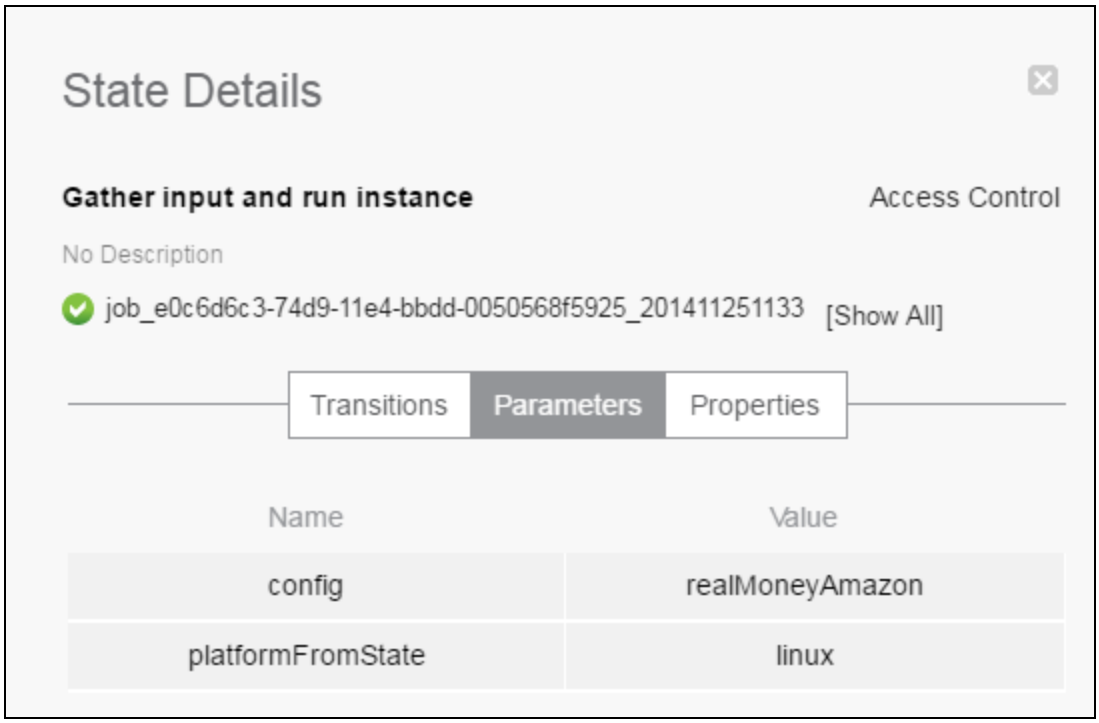
Column descriptions

Column Name	Description / Actions
Name	One or more names of transitions currently defined for this state. Click on a transition name to open the Transition Details panel.
Trigger	The type of trigger currently defined for this transition.
Target	The target state for this transition. Click on a target state name to open the State Details panel for that state.
Condition	This condition was specified when the transition was created. A condition specifies under which circumstances the transition is taken. You can edit this condition on the Transition Definition pane, but if you edit the condition during a running workflow, the change will not take effect until you run the workflow again.
Actions	**Access Control**—Use this link to set permissions on this transition.



Parameters tab

The **Parameters tab** provides a table listing all defined parameters for this state. You will see any parameters passed to this state the last time this state was entered, but only if parameters were created on the State Definitions Details panel. This table does not allow you to edit the parameter's value. These parameters are copied automatically to the top-level property sheet and you will see them in the properties table also.



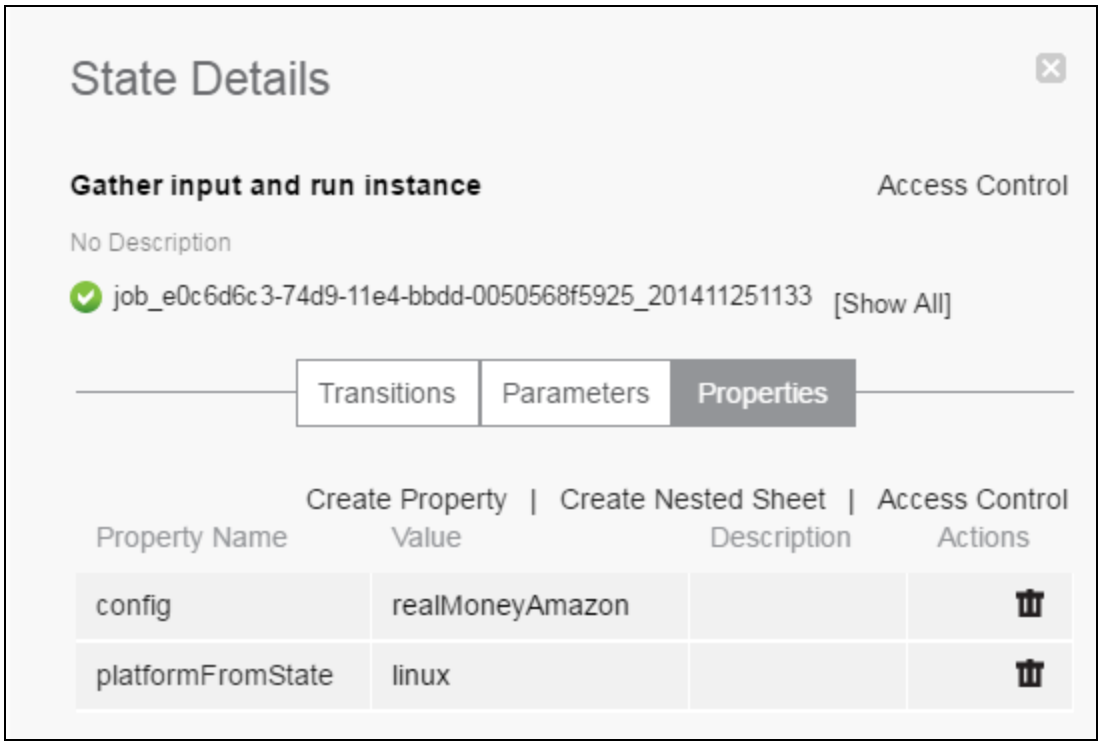
Column descriptions

- **Name**—The name of the parameter defined for this state.
- **Value**—The value currently defined for this parameter.

Properties tab

Select the Properties tab to see the Properties table. If no properties were defined for this workflow definition, this table will be blank.

The **Properties tab** provides a table containing all defined properties for this state. If no properties were defined for this state definition, this table will be blank. All defined parameters are automatically copied to the top-level property sheet. Any custom properties defined for the state will be included in this table also.



Links at the top of the table

- **Create Property**—Click this link to go to the New Property pop-up box to create a new property for this state.
- **Create Nested Sheet**—Click this link to go the New Nested Property Sheet pop-up box to create a nested property sheet.
- **Access Control**—Click this link to add or decrease access control on the property sheet.

Column descriptions

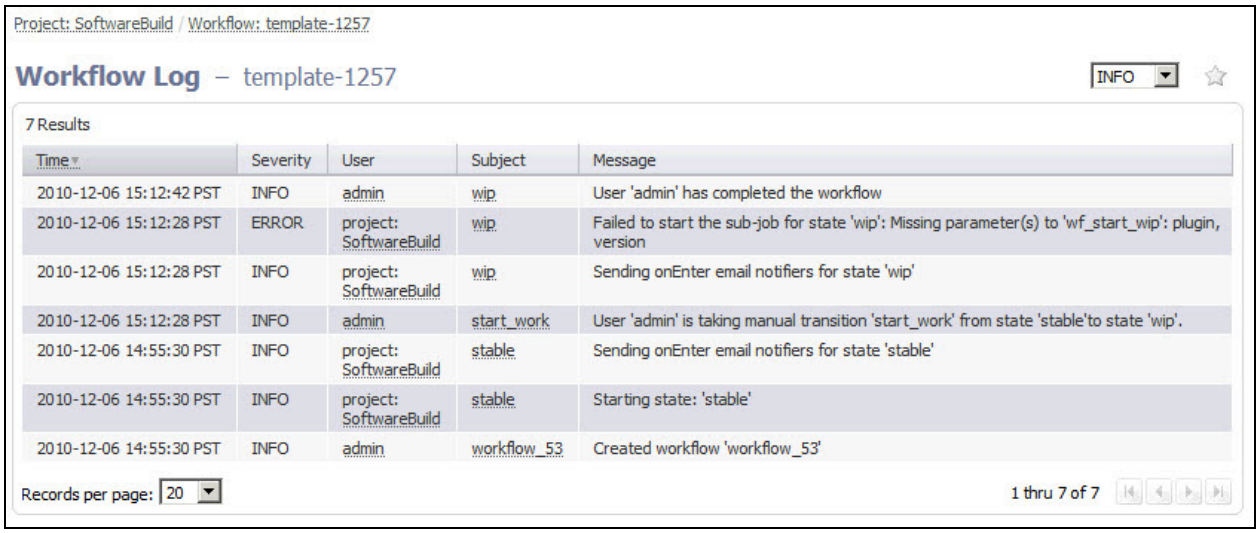
Column Name	Description / Actions
Property Name	Select a property name to edit that property. You can change its name, value, or add or change its description. If the property name is preceded by a folder icon, this is a property sheet. Click on the property name to open the "folder".
Value	The value currently assigned to the property.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Actions	**Delete**—Click this link to delete the property on that row.

Workflow Log

This page displays the workflow log. Notice the name after the page title, "template-1257" — this is the name of the workflow, which includes the workflow name plus the ElectricFlow auto-generated ID number, and it is the object of the workflow log in our example.

Links and actions above the table

- Drop-down menu—Use the down-arrow to select Error, Warn, or Info. The first time you choose a severity level, it will become your default level—the one you always see first when you view this page. Selecting another level changes your default view.
- The "star" icon allows you to save this workflow definition to your Home page.
- The "bread crumbs" *Project: SoftwareBuild / Workflow: template-1257* provide links to a previous web page.



Column descriptions

- **Time**—Displays the time when an event occurred that caused the server to generate a message.
- **Severity**—The three severity levels are:
  - ERROR—An unexpected failure was encountered while entering a state or launching a sub-action. Generally, this error indicates a critical problem with the workflow that requires fixing the workflow definition.
  - WARN—A non-critical issue was encountered while the workflow was running.
  - INFO—Provides workflow activity information including the state entered, transitions taken, and so on.
- **User**—The name of the user or project principal that explicitly launched the job. This property is blank when the job is launched by a schedule.
- **Subject**—Objects displayed in this column are the subject of the message. These objects are linked to either the Workflow Details or the State Details page.

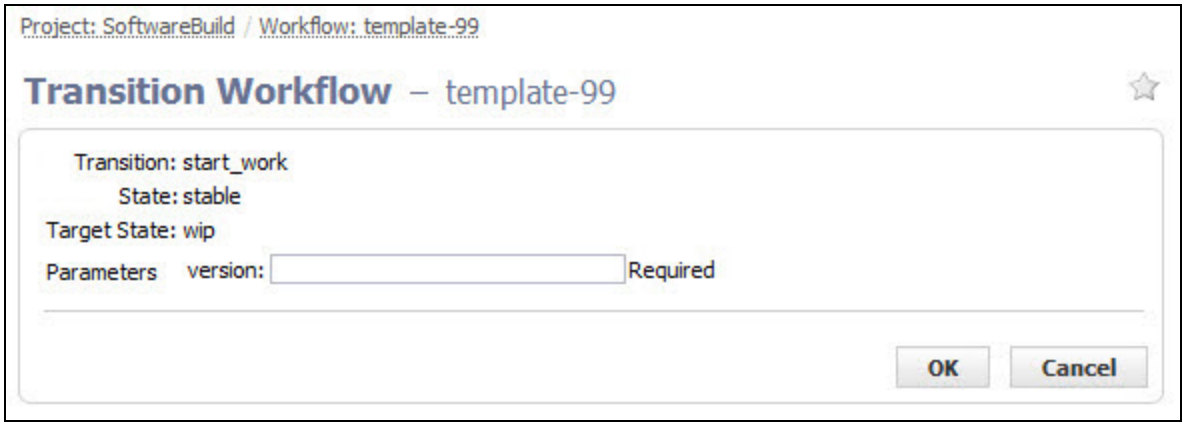
- **Message**—A text message generated by the ElectricFlow server while the workflow was running.

Transition Workflow

Use this page to transition your workflow to a different startable state. Note the name after the page title, "template-99"—this is the name of the workflow in our example.

Links and actions above the table

- The "star" icon allows you to save the Transition Workflow page to your Home page.
- The "bread crumbs" *Project: SoftwareBuild / Workflow: template-99* provide links to a previous web pages.



Field descriptions

Enter information into the fields as follows:

Field Name	Description
Transition	The name of this transition.
State	The name of the state that owns the transition— the active state of the workflow.
Target State	The name of the target state— the state where the workflow will transition.
Parameters	In the screen example above, "version" is the name of the parameter and you must enter a value for this parameter. This is a required field.

Click **OK** to transition the workflow to the target state.

Workspaces

This page displays all workspaces currently available to ElectricFlow. At a glance, you can see all paths to every workspace.

- To create a new workspace, click the **Create Workspace** link.
- To find a workspace, use the **New Search** link.

Column descriptions

Column Name	Description / Actions
Workspace Name	The name created for this workspace. Click on a Workspace Name in this column to edit that workspace.
Zone	The name of the zone where this workspace resides.
Enabled	This workspace is enabled if this box is "checked". When this box is checked, the workspace is enabled, which means it can be accessed. In the case where a job or job step cannot access the workspace, the job will "queue", waiting for the workspace to become available.
Local	If the box is "checked", the workspace is "local", which means files in the local workspace are accessible only from the resource that originally created the file.
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Drive Path / UNC Path / UNIX Path	The path to the workspace.
Actions	• **Copy**—Use this link to make an exact duplicate of an existing workspace, then select the copy to go to the Edit Workspace page to change the workspace name. • **Delete**—Use this link to delete the workspace on the same row.

Tip: Add access control privileges to a workspace on the Edit Workspace page.

Workspace—create new or modify existing workspace

To create a new workspace

Enter information into the fields as follows:

Field Name	Description
Name	Enter a unique name to distinguish this workspace from other workspace names.

Field Name	Description
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Enter three paths to use on resources to access the workspace root directory:	
UNC Path	A path in UNC notation (such as `//server/ec/a/b/c`) to use on Windows machines. The path can use either slashes or backslashes for separators.
Drive Path	A second path for Windows machines, based on drive notation such as `N:` or `N:/b/c`. If the drive path contains more than just a drive letter, additional directory names must match the trailing directories of the UNC path. For example, if the UNC path is `//server/ec/a/b/c`, then `N:/b/c` can be used as the drive path (agents automatically map `N:` to `//server/ec/a`), but `N:/x/c` is not allowed.
UNIX Path	The path to use for UNIX machines, typically based on an NFS mount. You must ensure appropriate mounts were created on all machines where the path will be used. *ElectricFlow does NOT automatically mount filesystems for UNIX.*
If the workspace will be used only on UNIX machines, you can omit the UNC and drive paths. Conversely, if the workspace will be used only on Windows machines, you can omit the UNIX path. For more information, see the Workspaces and Disk Space Management Help topic.	
Zone	Use the drop-down arrow to select a zone for this workspace.
Local	Select this checkbox if the workspace is "local", which means files in the local workspace are accessible only from the resource that originally created the file.
Enabled	"Check" this box to enable this workspace. When this box is checked, the workspace is enabled, which means it can be accessed. In the case where a job or job step cannot access the workspace, the job will "queue", waiting for the workspace to become available.
Credential Project and Credential Name	If you need to enter a user name and password to mount the workspace, you must enter the credential project name and then choose the appropriate credential name, which contains the required user name and password. Click the **Browse** link to select information for these fields.

Click **OK** to save your workspace specification and see it listed on the Workspaces page.

If you need to add properties or privileges to this workspace, click **OK** to create the workspace and return to the Workspaces page, then click on the workspace name to go to the Edit Workspace page.

To edit an existing workspace

All previously supplied workspace information is available on this page and you can modify existing information or add new properties or privileges.

1. Click the **Access Control** link [at the top of the page] to add access privileges to this workspace.
2. To assign a new workspace name, highlight the existing name and type-in a new name.
3. Re-assign paths if necessary.
4. Use the drop-down menu to select a different zone if necessary.
5. Add or modify the Credential Project name or the Credential Name if necessary.
6. Click **OK** to save your modified information.

If Custom Workspace Properties were already assigned, you may modify these properties or add new ones. If a property name is preceded by a "folder" icon, this denotes a Nested Sheet.

Click the "star" icon at the top of the page to add this page to your Shortcut section on the Home page. For more workspace information, see the Workspace and Disk Space Management Help topic.

Workspace File

This page displays a file from the workspace for a job, typically the log file for a job step. For log files, this file contains both standard output and standard error from the step.

If you view a file while the job step generating the file is still running, the page refreshes automatically as information is added to the file.

> **Tip:** If the log does not contain the output that you were expecting, check the job step commands. If the step redirects its output to a different location, this information will not appear in the log file.

Automation Platform Objects and Functionality

All the topics in this section are about a particular object or functionality in the ElectricFlow platform. They are the same as the topics in the Help system on the automation platform UI.

From the automation platform UI, you can access the topics as follows:

- Direct access to these Help topics is from the Help system Table of Contents. Although these topics are not directly linked within the ElectricFlow product, many "page-specific Help topics contain links to one or more of these topics to provide more detailed information quickly.
- To access these Help topics while using ElectricFlow, click the **Help** link at the top-right corner on any automation platform web page.

Access Control

Privileges

Users and Groups

Special users and groups

Overview

ElectricFlow provides a comprehensive mechanism to control how individuals use the system.

- Users must log in to view information or perform operations.
- After you log in, system access is limited based on:
  - username
  - the groups to which that user belongs
  - permissions specified for various ElectricFlow objects

Access Control is the ElectricFlow functionality that provides security for all system objects. After you are familiar with the following information describing the Access Control system, review the two examples, Basic ACL Setup and Team ACL Setup (at the end of this Help topic), which may provide guidance or insight for how you might setup enhanced ElectricFlow security at your site.

Privileges

ElectricFlow supports four privilege types for each object:

- **Read**—Allows object contents to be viewed.
- **Modify—**Allows object contents (but not its permissions) to be changed.
- **Execute**—If an object is a procedure or it contains procedures (for example, a project), this privilege allows object procedures to be invoked as part of a job. For resource objects, this privilege determines who can use this resource in job steps.
- **Change Permissions**—Allows object permissions to be modified.

Users and Groups

ElectricFlow uses account information from multiple sources. In most cases, the primary account information source is an external LDAP or Active Directory repository:

- Both user and group information is retrieved from the repository
- Local users and groups can be defined within ElectricFlow

> **Note:** To view user and group information, and to modify local user and group information, click the **Administration tab** and select either the **Users** or **Groups** subtab. External account information cannot be modified from ElectricFlow.

If the same user exists in multiple sources, only the highest priority name is used. A priority order is defined among external repositories, but local names have the highest priority. Thus, if you define a local user with the same name as an LDAP user, you will effectively mask the user in the LDAP account.

For local user accounts, only local groups are considered—group information from external repositories is not used. For accounts from a particular repository, groups from that repository are used along with local groups, but groups in other repositories are not considered.

In other words, you can define local groups in ElectricFlow to supplement groups defined in external repositories. When you view information about users in ElectricFlow, only relevant groups are shown. For example, when you view group information for a local user, only local groups are displayed.

> **Note:** Groups are managed by name only, without regard to source. If a particular group name exists in different repositories, there is no way to distinguish between these groups inside ElectricFlow. Access given to one group is the same for any other group with the same name.

Special Users and Groups

The *admin* local user has special significance. If you are logged in as "admin," you automatically have all privileges on all objects, regardless of any other system settings. This privilege set is a fallback mechanism in the event too many privileges get removed for an object, leaving it unusable.

The admin account cannot be deleted.

If the admin account is missing, ElectricFlow will recreate the account the next time it starts up with password "changeme."

The *Everyone* group is predefined by ElectricFlow and cannot be deleted. Every user is automatically a member of the Everyone group.

For each project, ElectricFlow uses the project name and automatically defines a user associated with that project, called the project principal. For example, the *project principal* for a project named "nightly builds" is "`project: nightly builds`" (notice that there are two spaces in this name)— this principal is used for jobs running under the project, as described in Access control and jobs.

Access Control Lists—allow and deny

Each ElectricFlow object, such as a project, procedure, job, property sheet, workspace, schedule, or resource contains an access control list (ACL). Individual properties do not have their own access control, but instead use access control lists from their parent containers.

To view the access control list for an object, go to the web page displaying the object and click the **Access Control** link. An access control list can contain any number of entries— each entry names a particular user or group and indicates *allow*, *deny*, or is blank for each of the four privileges.

To determine whether a user can perform an operation on a particular object, ElectricFlow determines which of the four privileges is required for that operation, then searches all access control entries that refer to that user or groups containing that user. To be allowed access, at least one of the matching entries must specify "allow" and none of the entries can be "deny." A "deny" entry overrides an "allow" entry in the same ACL.

Inheritance

If an object's access control list does not indicate what to do for the user who is trying to access it, ElectricFlow looks in the access control list for the object containing the target object, then its parent, and so on up to a top-level object covering the entire server. This mechanism is called *inheritance*.

For example, a nested property sheet on a procedure inherits access control information from its parent property sheet, the procedure, the project containing the procedure, and the server. When you view

the object access control list, you also see inherited access control lists so you can trace the object's inheritance chain.

When ElectricFlow performs an access check on an object, it begins with the access control list for that object.

- If a "deny" entry matches an entry in that list, access is denied.
- If there is no "deny" entry, but a matching "allow" entry is found, access is allowed.
- If there is no matching entry that specifies either "allow" or "deny," ElectricFlow moves to the next access control list in the inheritance chain and repeats the process.
- A matching entry in a lower-level access control list takes priority over entries in higher-level access control lists. If no matching entry in any list is found, access is denied.

Inheritance allows you to control access to a collection of objects because you can make changes in just one place. When new objects are created, their access control lists are empty, so all access control for the entire system is determined by the server-level list.

- Typically, when a new project is created, you define an access control list for the project and allow everything in the project to inherit from the project.
- If you do not define additional access control information in project components, all project access is determined by the project's access control list.

In some situations you may want tight control over object access, so changes in parent objects do not affect access to the object or its children. To achieve this, you can "break inheritance," which means a particular object does not inherit from its ancestors. The ElectricFlow web interface allows you to break and restore inheritance for any object on which you can change permissions. However, *if you break an object's inheritance, it affects all children of that object*.

Additional Information about "deny"

If a user matches a deny rule, however indirectly, the user is denied access. Even if a specific ACL entry granting (for example) read access by username exists, a less specific deny ACL entry overrides that specific rule. Therefore, avoid "deny" rules if at all possible (especially to groups), or you may experience some unexpected permissions issues.

You can, however, enforce an implicit "deny" by allowing an object to fall off the end of the inheritance chain. This practice allows denial behavior without the risks associated with an explicit ACL entry to deny access.

> **Important:** If you break inheritance on an object with an empty access control list, the object will become completely inaccessible. You will not be able to restore inheritance because you no longer have the right to change permissions on that object. If this happens, you must contact your system administrator who can login as admin and restore inheritance.

Consider ElectricFlow principals (actors) from two perspectives:

- A running job or workflow runs using the project as the identity, so if a job requires special permissions you must use the project as the principal in the ACL entry.
- Users and groups come into play when humans manipulate the GUI or execute ectool commands from the command line (or via a shell script) -- basically anywhere that a script or human needs to log in to ElectricFlow. Having logged in, a group and id exist, which can be used in ACL entries.

For example, consider a property that will store a build number. You can set an ACL that breaks inheritance (implicit deny), and then add an ACL entry granting permissions for the project to be able to increment the property value. This prevents humans from manipulating the build number but permits a running job to increment the build number at will.

Note:

- When it is practical, a job will run with multiple identities—the project and the user that launched the job. You cannot, however rely on this behavior for security because jobs run from a schedule (such as CI jobs) and jobs started by a workflow do not have a user principal, just the project principals.
- A job can have multiple project principals because it "accumulates" new project principals when it calls into another project's subprocedure (and then releases the other project's identity when the subprocedure returns). This makes is possible to create trusted libraries.

System Objects

A few special *system objects* contain access control lists related to overall ElectricFlow system administration. These access control lists are available from the Administration page. The system objects are:

- **Server—**an ElectricFlow system top-level object. Every other object in the system is contained in the server object and inherits access control information from the server object unless inheritance is broken.
- **Administration—***Read* permission allows access to the getStatus, `getDatabaseConfiguration[s]`, `getEmailConfig[s]`, and `export`(global) API functions.

  Modify permission allows access to the `shutdown`, `setDatabaseConfiguration`, `createEmailConfig`, `deleteEmailConfig`, `modifyEmailConfig`, and `import`(global) API functions.

  For Change Tracking, the *Read*, *Modify*, and *Execute* permissions allow you to revert changes to a tracked object and its tracked contents in the UI (see for more details) or access to the `revert` API function.
- **Artifacts**—*Read* permission allows access to the `getArtifact` API functions.
  Modify permissions allows access to `createArtifact` and `deleteArtifact` functions.
- **Directory**—*Read* permission allows access to the `getUsers`, `getGroups`, and `getDirectoryProviders` API functions.

  Modify permission allows access to the `createUser`, `createGroup`, `deleteUser`, `deleteGroup`, `createDirectoryProvider`, `modifyDirectoryProvider`, `deleteDirectoryProvider`, `testDirectoryProvider`, and `moveDirectoryProvider` API functions.
- **Email Configurations**—*Modify* permission allows access to the `createEmailConfig` and `deleteEmailConfig` API functions.
- **Force Abort**—*Execute* permission controls access to the `--force` flag on `abortJob`. By default, the ACL is created with Everyone: execute permission in addition to inheriting from the "Server". To force abort a job, the user must have execute permission on the job and execute permission on the `forceAbort` ACL.

- **Logging—***Modify* permission allows access to the `logMessage` API function.
- **Licensing—***Read* permission allows access to the `getLicense[s]` API functions. *Modify* permission allows access to the `importLicenseData` and `deleteLicense` API functions. *Execute* permission allows access to the `getAdminLicense` API function.
- **Plugins—***Modify* permission allows access to the `createPlugin`, `deletePlugin`, `installPlugin`, `uninstallPlugin`, `promotePlugin` API functions, and the `modifyPlugin` API function requires modify permission on the target plugin. For `getPlugin`, *Read* permission is required on the target plugin.
- **Priority—***Execute* permission allows the user who launches a procedure (using the `runProcedure` API function) to raise the priority of the job.
- **Projects—***Modify* permission allows access to the `createProject` and `deleteProject` API functions.
- **Repositories—***Read* permission allows access to the `getRepository` API function. Modify permission allows access to the `createRepository`, `deleteRepository`, `modifyRepository`, and `moveRepository` API functions.
- **Resources—***Modify* permission allows access to the `createReource` and `deleteResource` API functions.
- **Session—***Execute* permission allows access to the `login` API function.
- **Workspaces—***Modify* permission allows access to the `createWorkspace` and `deleteWorkspace` API functions.
- **ZoneAndGateways—***Modify* permission allows access to the `createZone` and `deleteZone` API functions. *Modify* permission also allows access to the `deleteResource` API function when the resource belongs to a Gateway.

Access Control and Jobs

When a job executes, it usually needs to access objects in ElectricFlow. For example, a job step command may refer to a parameter value, which is a property associated with the job object, or a step may invoke *ectool* to modify properties or any other ElectricFlow state. This process leads to questions:

- ***Under which username does the job execute?***

  Procedures always run under the project principal user ID for the project that contains the procedure. If a procedure invokes a subprocedure in another project, that subprocedure will run under its own project's project principal and the project principal of its calling procedure. When a procedure is running under multiple project principals, its steps will perform any operations for which any one of its project principals allow.

- ***How does ElectricFlow initialize job permissions when the job starts?***

  This question pertains to job object permissions. When a job starts, ElectricFlow sets full access control entries on the job for the project principal and the user who launched the job— assuming the job was launched by a user and not a schedule.

- ***What permissions are needed to abort a job?***

  Aborting a job requires *execute* permission on the job. If a job is launched by a user, that user is given all privileges on the job. If a job is launched by a schedule, the schedule's execute privileges are copied to the job.

The access control system determines whether jobs can be executed or not.

- For a user to run a job without creating a schedule, the user must have *execute* permission on the top-level procedure being executed.
- To create a schedule to run a procedure, a user must have *modify* permission for the project containing the schedule. After a schedule is created, no additional permissions are required to start jobs under the auspices of that schedule.

Examples for Increased Security

Initially, the *Everyone* group had Read and Execute permission on all ElectricFlow objects by default. The following examples illustrate how you might customize ElectricFlow default Access Control settings for your organization.

> **Important:** Make sure you are familiar with the Access Control system before making changes. Incorrectly setting ACLs can severely impact ElectricFlow behavior.

Abbreviations used in the examples

A = Allow

I = Inherit (some places referred to as "Don't Care")

D = Deny

Change Permission = Change

Example 1—Basic ACL Setup

This example illustrates how you might change Access Control defaults to set up a basic level of security. The easiest way to manage ACLs is by using different ElectricFlow groups that allow users to perform specific roles.

For our example, we will create and use the groups called EC-administrator and EC-designer, and we will modify the default Everyone group. (When setting up your groups, you would choose any group names that suit your organization, but note that the "EC-" prefix is reserved for Electric Cloud use only.) This is how we want to define our groups:

- EC-administrator group
  This group has most of the power and abilities of the 'admin' user login, but allows for more flexibility.
- EC-designer group
  This role is more significant. Users in this group can create and modify most ElectricFlow objects.
- Everyone group
  This predefined group is used for people who only need to run ElectricFlow procedures, and have no need to create or modify them.

As you begin to modify the Access Control system, note that the system has some project principal ACEs that should remain in place for correct system operation.

Server ACLs

The Server is the foundation for almost all ACL checks. Generally, every other object in the system inherits these privileges. For this basic ACL configuration we want to set Server ACLs this way:

Type	Name	Read	Modify	Execute	Change
group	EC-administrator	A	A	A	A
group	Everyone	I	I	I	I

Custom Server Properties ACLs

An often overlooked ACL setting is Server Property Sheet. Allow the Everyone group Read permission on this object.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

System Object ACLs

There are 14 categories of System objects with ACLs.

1—Administration – allow the EC-designer group Read privileges.

Type	Name	Read	Modify	Execute	Change
group	EC-designers	A	I	I	I

2—Artifacts – allow the EC-designer group Read privileges.

Type	Name	Read	Modify	Execute	Change
group	EC-designers	A	A	I	I
group	Everyone	A	I	I	I

3—Directory – allow the Everyone group Read privileges.

Type	Name	Read	Modify	Execute	Change
group	EC-designers	A	I	I	I

4—Email Configurations – allow the Everyone group Read and Execute privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

5—Force Abort—no changes needed for this category.

6—Logging—no changes needed for this category.

7—Licensing—no changes needed for this category.

8—Plugins—allow the Everyone group Read privilege.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

9—Priority—no changes needed for this category.

10—Projects—allow the Everyone group Read and Execute privileges, and the EC-designer group everything except Change privilege.

Type	Name	Read	Modify	Execute	Change
group	EC-designers	A	A	A	I
group	Everyone	A	I	A	I

11—Repositories—allow the Everyone group Read and Execute privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

12—Resources—allow the Everyone group Read and Execute privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

13—Session—allow the Everyone group Execute privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	I	I	A	I

14—Workspaces—allow the Everyone group to have Read and Execute privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

> **Note:** Sites that want a slightly more open system might consider granting the "EC-designers" group Modify access to Resources and Workspaces.

Plugin Project ACLs

Generally, you will want to allow the Everyone group Read privileges on every plugin you expect to use. (Each plugin has its own name, which is the project name also.)

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

Project ACLs

Next are special Electric Cloud projects, Default, EC-Utilities and EC-Examples.

Use caution when setting privileges on the EC-Utilities project. These are powerful procedures and some have their own ACL settings. Remember the 'admin' user will always be able to use these utilities. Allowing the EC-administrator group full privileges accomplishes the same goal for users in that group. We allow the Everyone group Read only privileges. This is a unique project because it does not inherit ACLs from anywhere else (for example, Server or Projects).

Type	Name	Read	Modify	Execute	Change
group	EC-administrator	A	A	A	A
group	Everyone	A	I	I	I

Example 2—Team ACL Setup

Some organizations may want to have two or more different and separate teams use ElectricFlow at the same time. In this example, people on one team have little knowledge about the other team who uses ElectricFlow and no direct access to any objects used by the other team. Our "Team" example describes how to modify default ElectricFlow Access Control settings to create a multi team security configuration using ACLs.

To divide your ElectricFlow system for use by separate teams, you will want to create one or more separate Projects for each team. You may decide to share some projects across teams.

Create your teams (groups). For our example, we will begin with two teams, T1 and T2. Each team has a designer group and a user group. Using groups allows the flexibility to have a designer from one group work as a user in another group.

- Create the EC-administrator group
  This group has most of the power and abilities of the 'admin' user login, but allows for more flexibility. Assigning people to the EC-administrator group allows better tracking of ElectricFlow administrative activity.
- Create two designer groups:T1-designer and T2-designer
  Members of the designer groups will have the ability to create and modify procedures and other ElectricFlow objects for their team. A member of the designer group from one team will not be able to view or use the objects of the other team.
- Create two user groups: T1-user and T2-user
  Members of the user groups will have the ability to run procedures that belong to their team. A member of the user group from one team will not be able to view or use objects that belong to the other team. who are not in any other group. This setup allows other users to access ElectricFlow without exposing the work of any of the teams. A person not in a group for either team will not be able to see objects of either of the teams.

As you begin to modify the Access Control system, note that the system has some project principal ACEs that should remain in place for correct system operation.

Server ACLs

The Server is the foundation for almost all ACL checks. Almost all objects in the system inherit these privileges. For this basic Team configuration, we will set the Server ACLs this way:

Type	Name	Read	Modify	Execute	Change
group	EC-administrator	A	A	A	A
group	Everyone	I	I	I	I

Custom Server Properties ACLs

An often overlooked ACL setting is Server Property Sheet. Allow the Everyone group Read permission on this object.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

System Object ACLs

There are 14 categories of System objects with ACLs.

1—Administration – allow both designer groups Read privileges.

Type	Name	Read	Modify	Execute	Change
group	T1-designer	A	I	I	I
group	T2-designer	A	I	I	I

2—Artifacts – allow the Everyone group Read privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

3—Directory – allow the Everyone group Read privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

4—Email Configurations – allow the Everyone group Read and Execute privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

5—Force Abort—no changes needed for this category.

6—Logging—no changes needed for this category.

7—Licensing—no changes needed for this category.

8—Plugins —allow the Everyone group Read privilege. Generally, privileges set here will be inherited by all plugins.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

9—Priority—no changes needed for this category.

10—Projects—no changes needed for this category. Privileges for projects are now handled on a case by case basis.

11—Repositories – allow the Everyone group Read privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

12—Resources—allow the Everyone group Read and Execute privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

13—Session—allow the Everyone group Execute privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	I	I	A	I

14—Workspaces—allow the Everyone group Read and Execute privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

Plugin Project ACLs

Generally, you will want to allow the Everyone group Read privileges on every plugin you expect to use. (Each plugin has its own name, which is also the project name.) Because you may want to use many plugins, you might want to use a script.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

ACLs for Existing Projects

Next are a few Electric Cloud projects, and two sample projects that can be seen and used by their respective team members only.

Project: EC-Examples

These are nice examples and we want to grant Everyone the ability to see and run them.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

Project: Default

Inherited settings will allow the EC-administrator group full privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

Project: EC-Utilities

Use caution when setting privileges on the EC-Utilities project. These are powerful procedures and some have their own ACL settings.

Remember the 'admin' user will always be able to use these utilities. Allowing the EC-administrator group full privileges accomplishes the same goal for users in that group. We allow the Everyone group Read only privileges. This is a unique project because it does not inherit ACLs from anywhere else (for example, Server or Projects).

Type	Name	Read	Modify	Execute	Change
group	EC-administrator	A	A	A	A
group	Everyone	A	I	I	I

Project: Electric Cloud

This project has one procedure and we want to grant everyone the ability to see it.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

ACLs for New Projects

Now we introduce the concept of ACL settings as they would be in a multiple team environment. For this example, we assume you have five projects being used by two teams in the following way.

- Project-A used by Team1
- Project-B used by Team1
- Project-C used by Team2
- Project-D used by Team2
- Project-E used by Team1 and Team2

Project-A

This project is visible and usable only by ElectricFlow users who are members of either the T1-user group or the T1-designer group. Notice that the designer group receives full permissions.

Type	Name	Read	Modify	Execute	Change
group	T1-designer	A	A	A	A

Type	Name	Read	Modify	Execute	Change
group	T1-user	A	I	A	I

Project-B

This project is the same as Project-A.

Type	Name	Read	Modify	Execute	Change
group	T1-designer	A	A	A	A
group	T1-user	A	I	A	I

Project-C

This project is visible and usable only by ElectricFlow users who are members of either the T2-user group or the T2-designer group. Notice that the designer group receives full permissions.

Type	Name	Read	Modify	Execute	Change
group	T2-designer	A	A	A	A
group	T2-user	A	I	A	I

Project-D

This project is the same as Project-C.

Type	Name	Read	Modify	Execute	Change
group	T2-designer	A	A	A	A
group	T2-user	A	I	A	I

Project-E

This project is a superset that includes both teams.

Type	Name	Read	Modify	Execute	Change
group	T1-designer	A	A	A	A
group	T1-user	A	I	A	I

Type	Name	Read	Modify	Execute	Change
group	T2-designer	A	A	A	A
group	T2-user	A	I	A	I

Enabling Same Team Subprocedures

To enable procedures in one project to use procedures in another project, one more step is necessary.

Add the project principals for both T1 projects to the T1-user group. In this case, these are called: "project: Project-A" and "project: Project-B".

Add the project principals for both T2 projects to the T2-user group. In this case, these are called: "project: Project-C" and "project: Project-D".

Optional: Restricting Resources and Workspaces by Team

Resource ACLs

Resource: local

Because each team will have dedicated resources, it is useful to keep one resource available for everyone to use.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

T1-resource

This resource is visible and usable only by ElectricFlow users who are members of either the T1-user group or the T1-designer group. This resource also needs Execute permission for the project itself to allow it to run from a schedule.

Type	Name	Read	Modify	Execute	Change
group	T1-designer	A	I	A	I
group	T1-user	A	I	A	I

T2-resource

This resource is visible and usable only by ElectricFlow users who are members of either the T2-user group or the T2-designer group. This resource also needs Execute permission for the project itself to allow it to run from a schedule.

Type	Name	Read	Modify	Execute	Change
group	EC-administrator	A	A	A	A
group	Everyone	I	I	I	I

Workspace ACLs

Workspace: default

Because there will be dedicated workspaces for each team, it is useful to keep one workspace available for Everyone to use.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

T1-workspace

This workspace is visible and usable only by ElectricFlow users who are members of either the T1-user group or the T1-designer group.

Type	Name	Read	Modify	Execute	Change
group	T1-designer	A	I	A	I
group	T1-user	A	I	A	I

T2-workspace

This workspace is visible and usable only by ElectricFlow users who are members of either the T2-user group or the T2-designer group.

Type	Name	Read	Modify	Execute	Change
group	T2-designer	A	I	A	I
group	T2-user	A	I	A	I

Artifact Management

About Artifact Objects

Configuring Access Control

Publishing Artifact Versions

Uploading and Publishing Artifacts from a Local File System Using the UI on page 1338

Overview

During the development process, project components or "outputs" are produced for consumption by other projects or the main project to create a whole entity. Applications are assembled by combining these various outputs (dependencies) and packaging them together into a deployable software application.

Outputs can be libraries, scripts, graphics, and so on. This output is an *artifact* that can be consumed by other projects.

Every stage in the application lifecycle produces and consumes artifacts. Artifacts are critical to the *build-test-deploy* process.

Using artifacts can:

- Improve performance across builds.
- Provide better reusability of components.
- Improve cross-team collaboration with greater traceability.

For example, instead of each developer repeatedly downloading third-party packages from external sources, these components can be published and versioned as an artifact. A developer then simply retrieves a specific artifact version from a local repository, which guarantees a consistent package from build to build.

About Artifact Objects

ElectricFlow has three types objects to support Artifact Management functionality: *artifact*, *artifact version*, and *repository*. Similar to other ElectricFlow objects, each of these objects supports custom properties and access control.

Artifact

An *artifact* is a top-level object containing artifact versions, a name template for *published* artifact versions, artifact specific properties, and access control entries to specify privileges. If an artifact is deleted, all published artifact versions within that artifact will be deleted also.

You can create artifacts in several ways:

- Use the New Artifact web page
- Use the `createArtifact` API command with the command-line tool (ectool) or an ec-perl script

- An artifact is created when you publish an artifact version if the artifact did not already exist, and the user issuing the publish command has permission to create artifacts. See Configuring Access Control on page 1336 and Publishing Artifact Versions on page 1349 for more information.

To create an artifact, you enter a Group Id and Artifact Key and together, these create an artifact name in the form `groupId:artifactKey`. After ElectricFlow creates the artifact name from your supplied Group Id and Artifact Key specifications, you cannot modify the name.

> **Note:** Plan your artifact Group IDs and Artifact Keys carefully to prevent the extra work of deleting a name and beginning your specifications again. The use of a `groupId` and `artifactKey` allow two "namespaces" for added flexibility in organizing artifacts within your company or organization.

Examples

- If your development team uses GWT for web development and JUnit for unit testing, you may want to create artifacts "`ThirdParty:GWT`" and "`ThirdParty:JUnit`".
- Your organization may have two different development teams (OS and Apps) who want to use artifacts. In building each product, both development groups want to publish and use an "SDK" with their product. But each SDK is different and unique. One solution would be to create one artifact named "`os:SDK`" and another artifact named "`apps:SDK`". By using different group Id's, each development team is free to name their artifacts however they choose without worrying about name collisions with artifacts produced by the other team.

To reference an artifact, you can use the `groupId` and `artifactKey` combination or simply use the `artifactName`. The property path `/artifacts/<artifactName>` allows you to access any properties of a given artifact.

Artifact Version

An *artifact version* is a collection of `0` to `N` files that were published to an artifact repository. Artifact version metadata is stored in an `artifactVersion` object in the ElectricFlow server.

Tips for working with artifact versions:

- To group a collection of files (typically from build output in an ElectricFlow workspace) into an artifact version, specify the "from" directory containing those files.
- If you want to include files from multiple directories in your artifact, you need to specify a common root directory to include these directories when publishing your artifact.
- To restrict which files and subdirectories in the "from directory" to include in the artifact version, you can specify "include" and "exclude" patterns.
- Artifact versions are stored on an artifact repository server in either uncompressed `tar` archives or compressed `tar-gzip` archives.

- Artifact versions can be created using the `publishArtifactVersion` API, which includes creating the object in the ElectricFlow server and publishing the artifact version to the artifact repository.

  > **Note:** Interactively creating an artifact version using the automation platform web UI is not supported.

- When you retrieve an artifact version, the original directory structure of the directory where files were published "from" is preserved.
- When you retrieve/publish artifacts, directories containing symlinks are always resolved; they are not preserved.

Artifact versions are referenced by *`Group Id`*, *`Artifact Key`* (or artifact name) and a version string.

- The artifact version's name is set based on the name template on the "owning" artifact. By default, the name template is in the form *`groupId:artifactKey:version`*.
- Each artifact version name must be unique within an ElectricFlow server installation.

The property path `/artifactVersions/`*`<artifactVersionName>`* allows access to any properties for a particular artifact version. Because you may not know the artifact version name (because of the name template), this path supports the form *`<groupId>:<artifactKey>:<version>`* as a substitute for the name of the artifact version.

All ElectricFlow API's that take an *`artifactVersionName`* argument also accept this alternate form as a substitute, similar to how other API's that accept a `jobId` argument accept the job name as a substitute.

Version Strings

A version string must be provided when publishing an artifact version.

For example:

`5.8.8-EN-55842 or 5.8.8.55842-EN` –In either form, the version string is interpreted as:

Major version number: 5

Minor version number: 8

Patch level number: 8

Qualifier: EN

Build number: 55842

You must provide separator punctuation. When interpreting (parsing) the first version string form, ElectricFlow interprets the text after the first hyphen as the build number if this string contains numeric characters only. Otherwise, the string is interpreted as the `qualifier`.

Version string examples and how ElectricFlow interprets them:

	Input (version string)	Major	Minor	Patch	Qualifier	Build Number
1	2.0.3-44834	2	0	3	<empty/null>	44834

	Input (version string)	Major	Minor	Patch	Qualifier	Build Number
2	3.8.4-RC1	3	8	4	RC1	0
3	3.8.4-RC1-36	3	8	4	RC1	36
4	3.8.4.36	3	8	4	<empty/null>	36
5	1-36	1	0	0	<empty/null>	36
6	3.8.4.36-RC1	3	8	4	RC1	36

In this table:

- Row 5 shows that you are not required to specify all three of `<major, minor, patch>` when publishing an artifact version if your organization's versioning conventions do not use all three fields. However, there is a caveat with this level of flexibility. You can create two artifact versions in the system with equivalent versions.
  Version string "1-36" is equivalent to "1.0-36" but they are not the same so you can publish two artifact versions with these version strings successfully. When retrieving an artifact version, in some cases it could be unclear as to exactly which one you will get. For this reason, we recommend that you adopt a convention and abide by it for a particular artifact.

The artifact version object exposes components of the version string by using these intrinsic properties:

- **`version`**— the version component is the combination of *`major.minor.patch-qualifier-buildNumber`*. A "version" component does not need to include all of the following intrinsic properties. See the table above.
- **`majorMinorPatch`**— this is the *`major.minor.patch`* version component as specified in the version string. For example, if you specified a version string of "1-36, the *`majorMinorPatch`* component would be "1", not "1.0.0", which makes it possible for you to meet your naming conventions by reconstituting the version string a different way in the artifact version name template.
- **`qualifier`**—the qualifier component of the version .
- **`buildNumber`**—the build number component of the version.

The `artifactVersionState` property

Artifact versions have an `artifactVersionState` property whose value can be one of the following:

`publishing`—The artifact version is currently being published and is not available for retrieval.

`available`—The artifact version is available for retrieval if needed.

`unavailable`—The artifact version is not available for retrieval while in this state.

You can manipulate the state between available and unavailable on the Edit Artifact Version web page. This action can be useful if an artifact version seems to be corrupt in some way, but you want to investigate before potentially deleting it from the system.

By making the artifact version unavailable, retrievers can acquire an older (potentially more stable) artifact version while you look into the problem. If the artifact version is good, you can simply make it available again.

Artifact Repository

The artifact repository is a machine where artifact versions are stored. The repository server is configured to store artifact versions in a directory referred to as the repository backing store.

By default, the backing store is the `<data_dir>/repository-data` directory in the repository installation. This default setting can be changed by running `ecconfigure --repositoryStorageDirectory` on the repository server. The repository server listens on port 8200 for HTTPS requests to publish and retrieve artifact versions.

> **Note:**
> You cannot provide a mapped drive path when defining a backing store. For example, the path in the following command is not allowed and will cause artifact publishing to fail:
>
> ```
> ecconfigure --repositoryStorageDirectory c:/repository-data
> ```
>
> You must enter a UNC path instead. For example, the path in the following command is allowed:
>
> ```
> ecconfigure --repositoryStorageDirectory //10.0.109.72/repo_share/repository-data
> ```

Connection information is stored in the repository object on the ElectricFlow server.

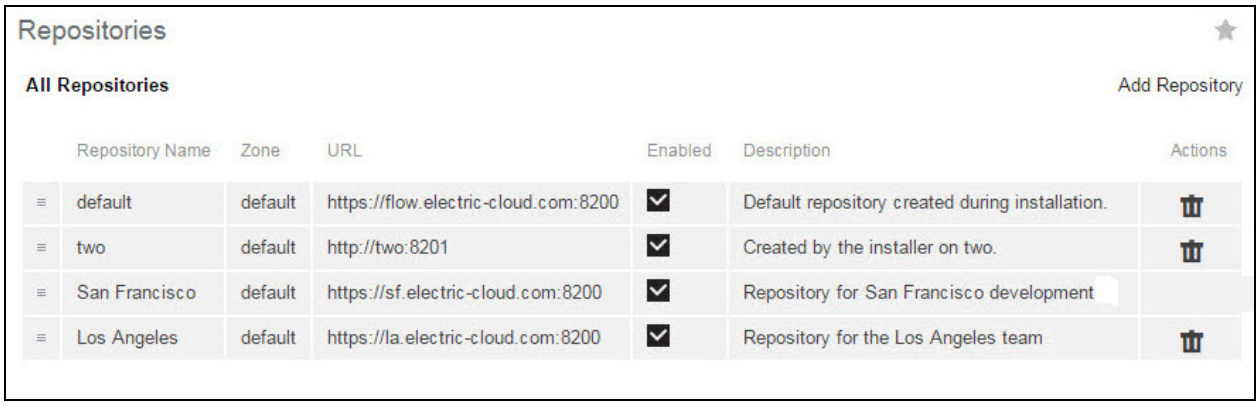
You can create a repository objects in several ways:

- Use the New Repository web page in the ElectricFlow UI
- Use `createRepository` API
- Repository objects can be created automatically during the ElectricFlow installation.

When you are installing a repository *server*, the installer creates a corresponding repository *object* on your ElectricFlow server. If you are installing a repository *server* on the same machine as your ElectricFlow server, a repository *object* named "default" is automatically created. If you are installing a repository *server* on a different machine than your ElectricFlow server, the installer will prompt you for information it uses to create the corresponding repository *object*.

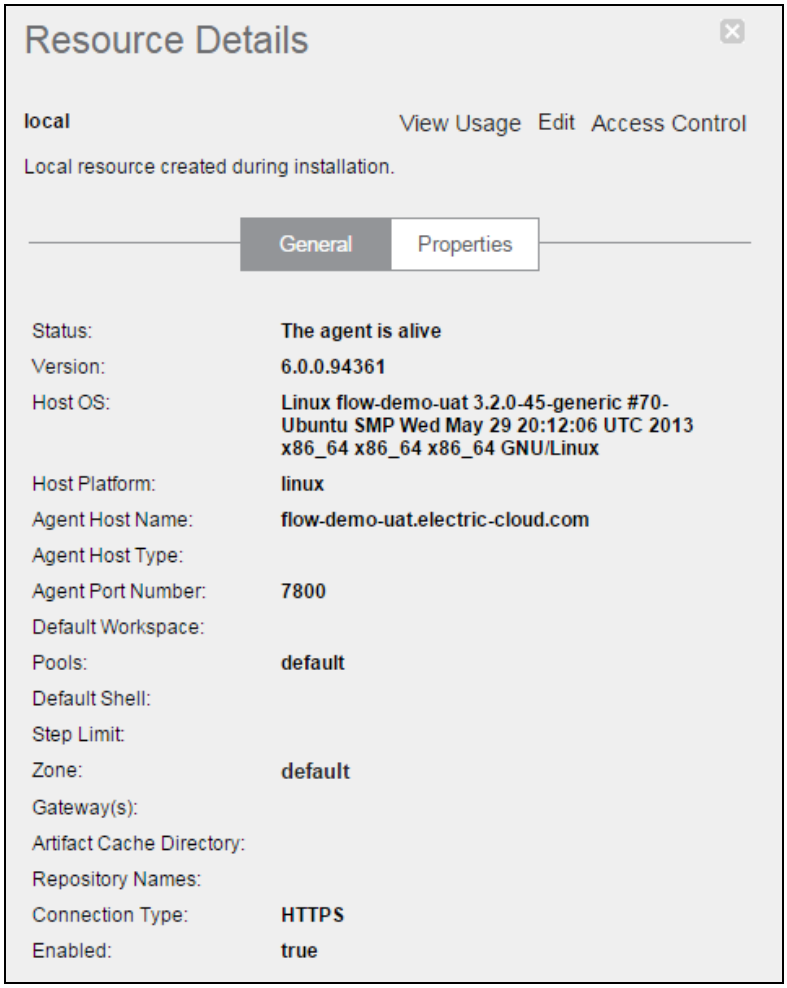
> **Tip:** When installing a repository server, you should install an agent also to ease maintenance tasks such as clearing out stale artifact versions from the repository backing store.

Click the Artifacts tab and then the Repositories subtab to see your list of created repositories.



You can re-order the repository search order (for retrieving artifact versions). To do so, use the mouse to grab and drag the icon in the left column to the row position you prefer, effectively re-ordering the repository list.

For distributed environments, where the preferred repository search order varies depending on which resource is performing the retrieval, you can specify a preference order on the Edit Resource panel. See the illustration below.

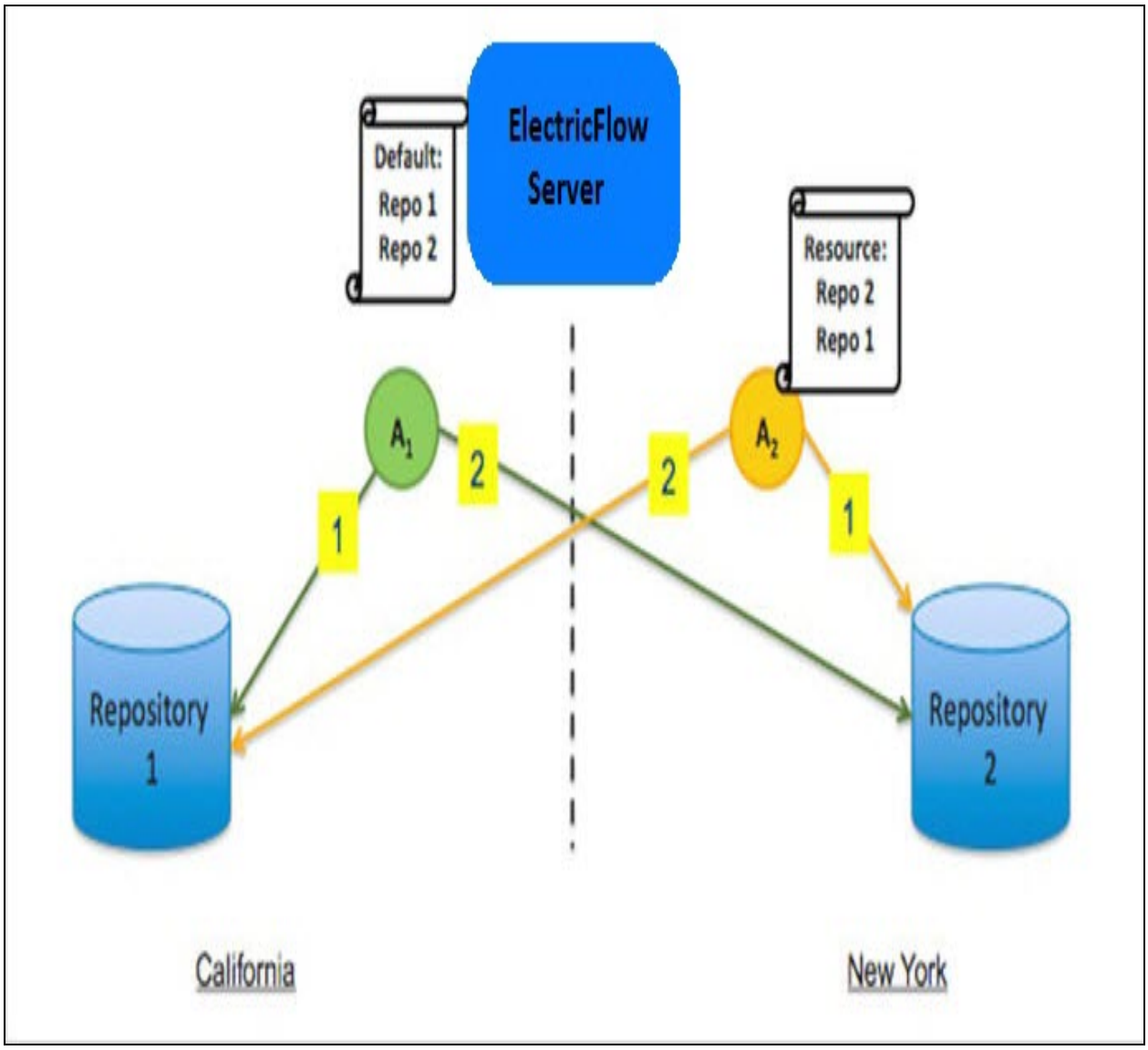


Note: The **Gateway(s):** field that appears in the **Resource Details** panel is not editable (not even for a gateway resource) and therefore does not appear in the **Edit Resource** panel. This field is automatically filled and is only for gateway agents. For example, in the **Resource Details** panel for agentp1-gw-01 and agentp2-gw-01 resources as they form a gateway Default_to_zone1_gateway using the steps in the KBEC-00393 - Setting up an environment with two zones and a gateway KB article.

To go to the Edit Resource panel: from the Repositories page, select the Cloud tab to open the Resources page, then select a resource name. Enter one repository name per line in the Repository Names field.

In the following example, a resource in California (A1) uses the "default" repository search order (that is, the order on the Repositories page). A1 searches for artifact versions by first requesting an artifact version from Repository 1. If the artifact version requested is not found, it will then request the artifact version from Repository 2.

The resource in New York (A2) has a specific search order specified where it searches for artifacts from Repository 2, first, then searches Repository 1.



Using Multiple Repositories

ElectricFlow supports multiple repositories, however you cannot share a repository between multiple ElectricFlow servers. Multiple repository servers may be required in your organization because of business or organizational reasons, government and compliance requirements, or for performance and data protection.

For example:

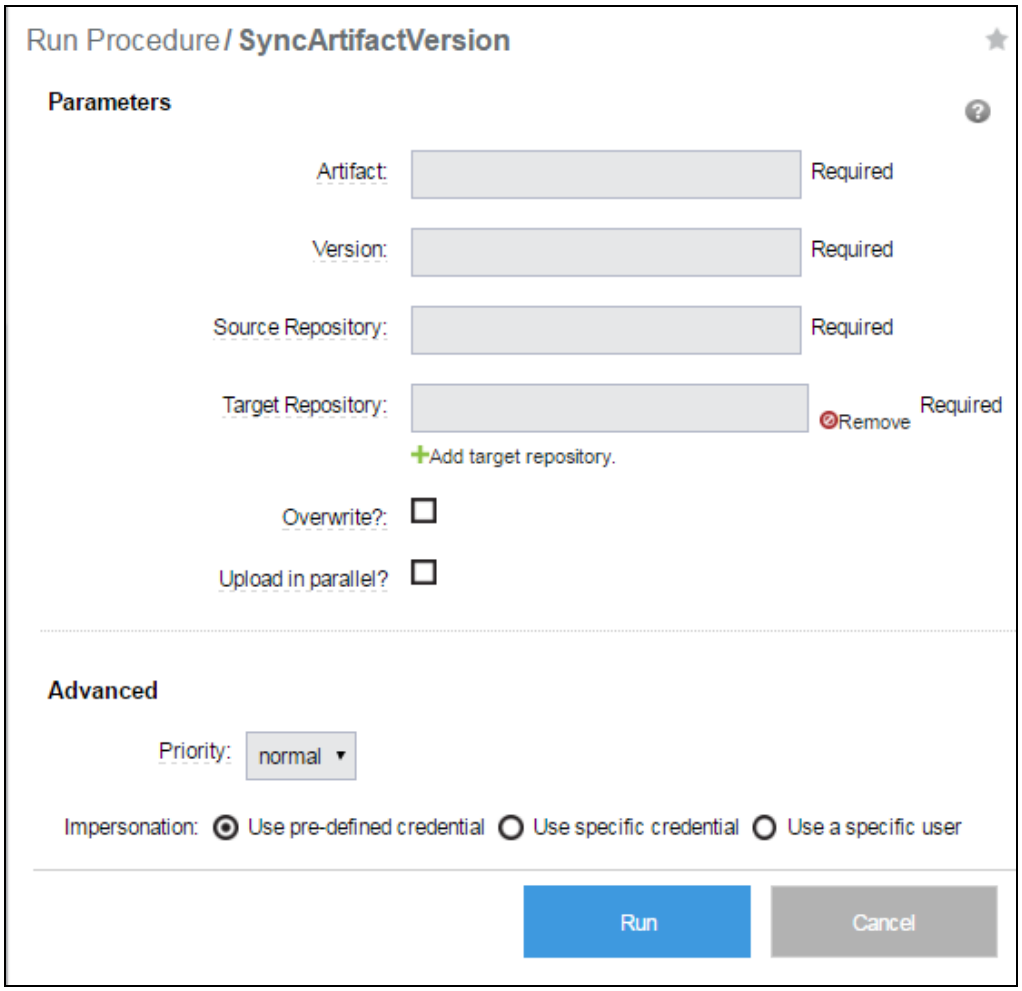
- Business or organizational structure— Within a company, two different development teams want to have artifact repositories containing artifacts and artifact versions specific to their product. A repository sever is created for each team. The two teams publish artifact version to their respective repositories.

- Performance—A company has multiple development sites. Instead of having a single repository server, which would create significant network performance issues from retrieving artifact versions during the build process, they create a repository at each development location and have artifact versions synchronized between the repository servers. In this way, when artifact versions are retrieved for use by a build process, they are retrieved from a local repository.
- Data protection and disaster recovery—A company has a single ElectricFlow server with two artifact repository server machines. The contents of the repository servers are synchronized and replicated between both machines. In the event repository 1 is unavailable during a retrieve, repository 2 is searched for the requested artifact retrieval.

Copying Artifacts to Multiple Repositories

You can synchronize (copy) artifacts from a source artifact repository to one or more target repositories. This helps to support distributed architectures with multiple ElectricFlow artifact repository servers. You must specify the exact artifact and its version as input. This feature uses the EC-Artifact plugin, which is bundled with ElectricFlow.

To synchronize artifacts to other repositories, choose **Administration** > **Plugins** > **EC-Artifact** and click  (Run) to run the SyncArtifactVersion procedure. The following dialog box appears:



Entering Parameters

Enter the following parameters as needed.

Parameter	Description
Artifact	Artifact name. The format is <*groupId*>:<*artifactKey*>.
Version	Full artifact version string. The format is <*major*>.<*minor*>>.<*patch*>-<*qualifier*>-<*buildNumber*>. For example, `5.8.8-EN-55842`. The version string must be unique across all of this artifact's versions. For more information, see Version_Strings.
Source Repository	Name of the repository from which this artifact version will be retrieved.
Target Repository	Name of the repository to which this artifact will be copied. You can add more repositories by clicking + (**Add target repository**).

Parameter	Description
Overwrite?	(Optional) Specifies whether to overwrite any existing artifact with the same name and version on the target repositories. This option is disabled by default.
Upload in parallel?	(Optional) Specifies whether to copy artifacts to multiple target repositories in parallel (for faster performance) instead of serially. You should choose whether to use this option based on how the number of target repositories and the artifact size could impact network traffic. This option is disabled by default.

Then click **Run**.

Understanding Overwrite Behavior

The following table provides an example that illustrates the overwrite behavior.

Repository	Result with Overwrite disabled	Result with Overwrite enabled
Repository 1 (source): Repository containing artifact1 to be copied	–	–
Repository 2 (target): Repository without artifact1	artifact1 is copied to Repo2	artifact1 is copied to Repository 2
Repository 3 (target): Repository containing artifact1	artifact1 on Repository 3 is not overwritten	artifact1 on Repository 3 is overwritten
Repository 4 (target): Repository that does not exist	Error	Error

Troubleshooting

The SyncArtifactVersion procedure finishes with an error in the following situations:

- Repository is unreachable: The target repository might be down.
- Insufficient access rights: Access rights for writing to the target repository might be missing.

Configuring the Artifact Repository Server for Amazon S3

You can configure the repository server to use Amazon S3 as the backing store and connect to Amazon S3 using IAM roles. This section describes the changes to make in the `/opt/electriccloud/electriccommander/conf/repository/server.properties` file.

Before starting this procedure, make sure the following have been created and are available on the AWS Management Console (https://aws.amazon.com/console/):

- An Amazon S3 bucket for the backing store.
- The IAM role with a policy that grants all the `s3:*` access to the Amazon S3 bucket you want to use with the repository server.

For example, the IAM role called `s3repositoryFullAccess` has a policy called `s3fullaccessAndAccessToInstallers` that grants all the `s3:*` access to the *backing store bucket* called `ec-test-repository-backingstore`.

Follow these steps to set up the artifact repository for Amazon S3:

1. In the `/opt/electriccloud/electriccommander/conf/repository/server.properties` file, set `REPOSITORY_BACKING_STORE` to use Amazon S3 by changing

   ```
   REPOSITORY_BACKING_STORE=repository-data
   ```

   to

   ```
   REPOSITORY_BACKING_STORE=s3://<name of the backing store bucket>
   ```

   You can get the `<name of the backing store bucket>` by going to **IAM** > **Roles** > `<name of the IAM role>` page on the AWS Management Console. Then click **Show Policy** to open the **Show Policy** window to view the name of the backing store bucket.

   In the following example, `<name of the backing store bucket>` is *ec-test-repository-backingstore:*

2. Restart the artifact repository server using a command like this:

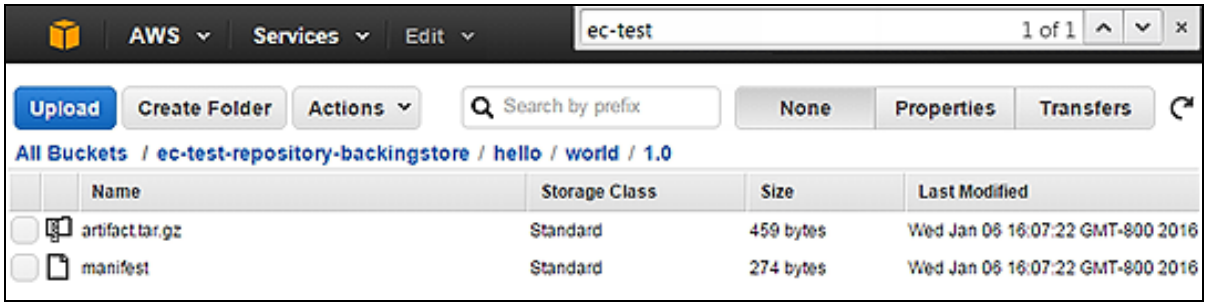
```
ubuntu@ip-10-101-214-28:~$ sudo /etc/init.d/commanderRepository restart
```

3. To test the `REPOSITORY_BACKING_STORE`, create two files in a `/tmp/<your username>` directory, publish the directory, and verify that it is published using commands like this:

```
ubuntu@ip-10-101-214-28:~$ cd /tmp

ubuntu@ip-10-101-214-28:/tmp$ mkdir <your username>

ubuntu@ip-10-101-214-28:/tmp$ cd <your username>

ubuntu@ip-10-101-214-28:/tmp/<your username>$ touch abc

ubuntu@ip-10-101-214-28:/tmp/<your username>$ ectool --server localhost login
admin changeme

ubuntu@ip-10-101-214-28:/tmp/<your username>$ ectool publishArtifactVersion --
version 1.0 --artifactName hello:world
```

4. Verify that the `/tmp/<your username>` is published to the Amazon S3 backing store.

   Example:



The `respositoryDisabled` Intrinsic Property

The repository object has a `repositoryDisabled` property that dictates whether or not artifacts can be published or retrieved from a particular repository. If you take a repository offline for maintenance, you can disable the repository in the ElectricFlow server rather than shutting down the repository machine or the service. Steps that attempt to publish to this repository will fail, and steps that attempt to retrieve artifact versions will skip this repository during the search for available repositories.

Configuring Access Control

You can apply access control permissions to artifacts and artifact versions to control who can create an artifact versus who can publish an artifact version versus who can read or retrieve an artifact version.

Action	Required permission	Object
Create an artifact	Modify	Artifacts system object

Action	Required permission	Object
Read artifact metadata	Read	The relevant artifact
Modify an artifact	Modify	The relevant artifact
Delete an artifact, which deletes all artifact versions within the artifact	Modify	Artifacts system object
Publish an artifact version	Modify	The artifact into which a new version is being published
Read artifact version metadata (required for retrieval)	Read	The relevant artifact version
Modify an artifact version	Modify	The relevant artifact version
Delete an artifact version	Modify	The artifact that owns the artifact version

Note: The Execute privilege does not take part in artifact or artifact version access control.

By default, an `<Everyone, modify>` access control entry is available that allows anyone to publish an artifact version to any artifact and allows anyone to create an artifact. You might consider removing this access control entry and setting up tighter control for individual artifacts. See the EC-Security plugin for help with this effort.

For more information about access control for artifact management, see the Access Control Help topic.

The following scenarios provide ideas for how you might set up access control for artifact management.

Control who can create new artifacts and publish artifact versions

ElectricFlow administrator, Adam, received communication from development manager, Max, requesting that only he should be able to create artifacts in ElectricFlow -- developers of ElectricFlow procedures under him should only be able to create steps to publish artifact versions under pre-defined artifacts.

His reasoning is that it is too easy for a developer to introduce a typo during publish (which does auto-create the artifact if the user has the appropriate privilege and the artifact doesn't exist).

Adam accommodates the request by giving Max the modify privilege on the artifacts system object and removing the `<Everyone, modify>` access control entry. Max then creates artifacts and assigns modify privileges to those artifacts for his development group, effectively allowing them to publish artifact versions.

At run-time, a step belonging to some project actually performs the publish, so Max adds project principal access control entries for the relevant projects as well, providing modify privileges.

Control which projects can retrieve artifact versions at run time

Acme Corporation's router product has a software stack consisting of OS, PlatformLibraries, and Apps. Each of these has a ElectricFlow project with several procedures that work together to build the relevant component. PlatformLibraries builds against some of the C header files and libraries produced by the OS build. Similarly, Apps builds against libraries produced by PlatformLibraries.

The lead developer for the OS component, creates a `Router:OSLib` artifact and gives the PlatformLibraries project principal read privileges so that project can retrieve any OSLib artifact version it needs to build PlatformLibraries. Similarly, Paul from the PlatformLibraries team creates a `Router:PlatformLib` artifact and gives the Apps project principal read privileges.

Group 1 publishes an artifact version and wants only Group 2 to be able to retrieve it

An organization wants to create artifacts for each of the third-party packages used by developers. Among the tools shared is the GWT package. The build manager chooses to create an artifact called `3rd-party-PKGS:GWT`. Because of licensing concerns, the build team must control or regulate who has access to the GWT and who can publish artifact versions used within the company's product.

In this case, the build team will publish a GWT artifact version and test it. During the test and approval cycles, only the build team has access to read and modify the artifact or publish new artifact versions. After a version has passed internal testing and approvals, the build team then grants read access to each of the developer groups.

Uploading and Publishing Artifacts from a Local File System Using the UI

You can use the Automation Platform UI to upload artifact files from a file system. This makes development and testing of new flows easier than other methods that use the EC-Artifact plugin and APIs. You can upload artifact files for publishing a new version of an existing artifact, or you create a new artifact and upload artifact files to publish the first version. You can upload a single file or a folder.

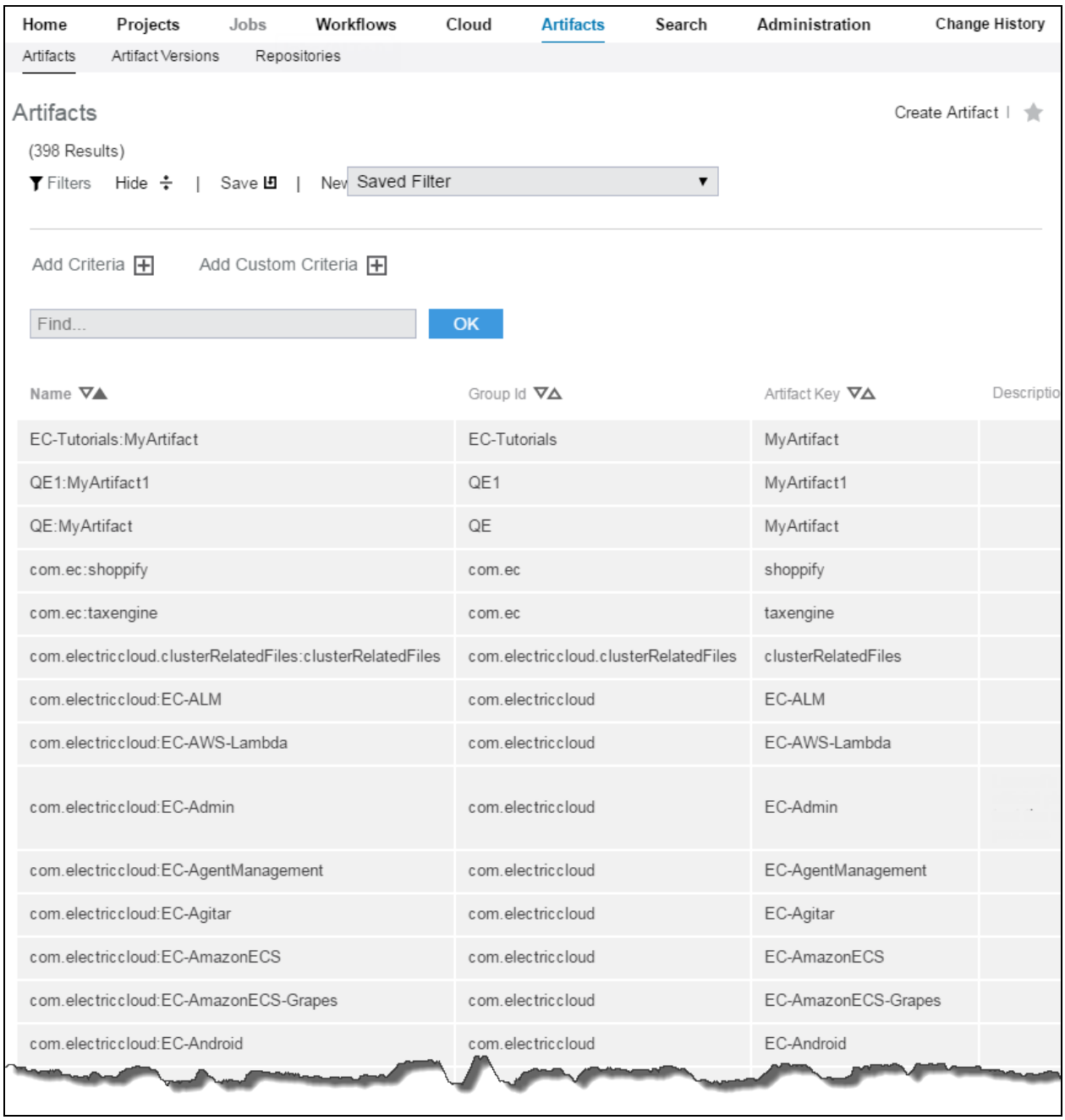
This functionality requires that the **Allow Artifact Publish from UI** check box in the **Edit Server Settings** page in the Automation Platform is checked. Note that this functionality is not supported in Internet Explorer or Safari.

Uploading Files to Publish a New Version of An Existing Artifact

To upload artifact files for publishing a new version of an artifact that already exists:

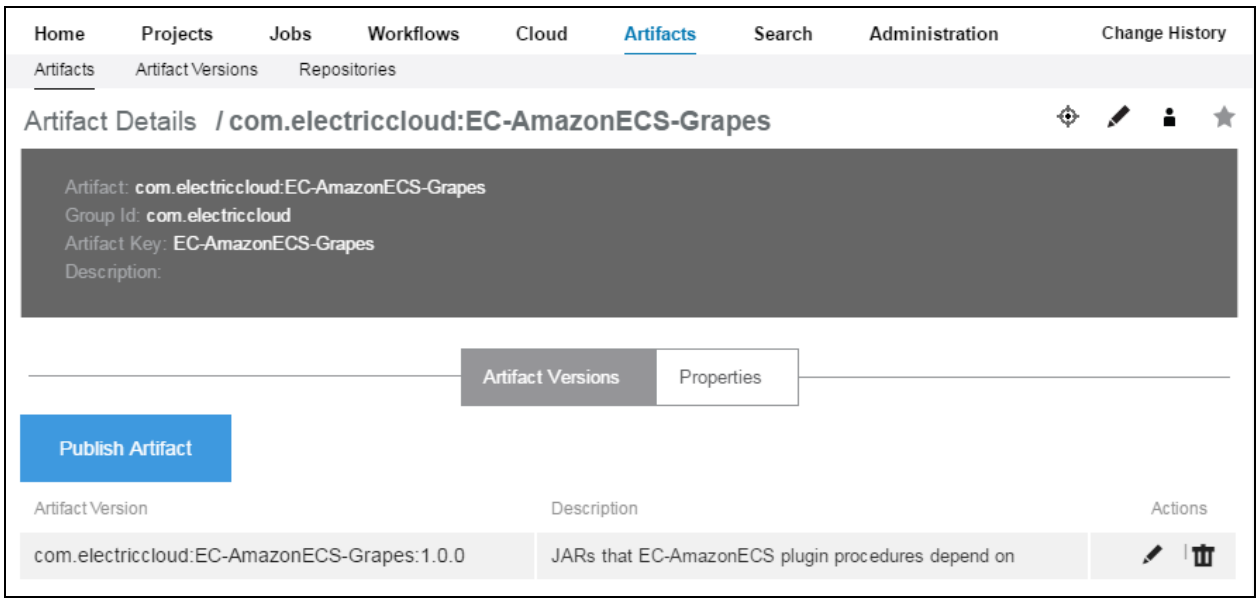
1. In the Automation Platform, choose **Artifacts**> **Artifacts**.

   The list of existing artifacts appears. For example:



2. Click the artifact for which you want to upload a new version.

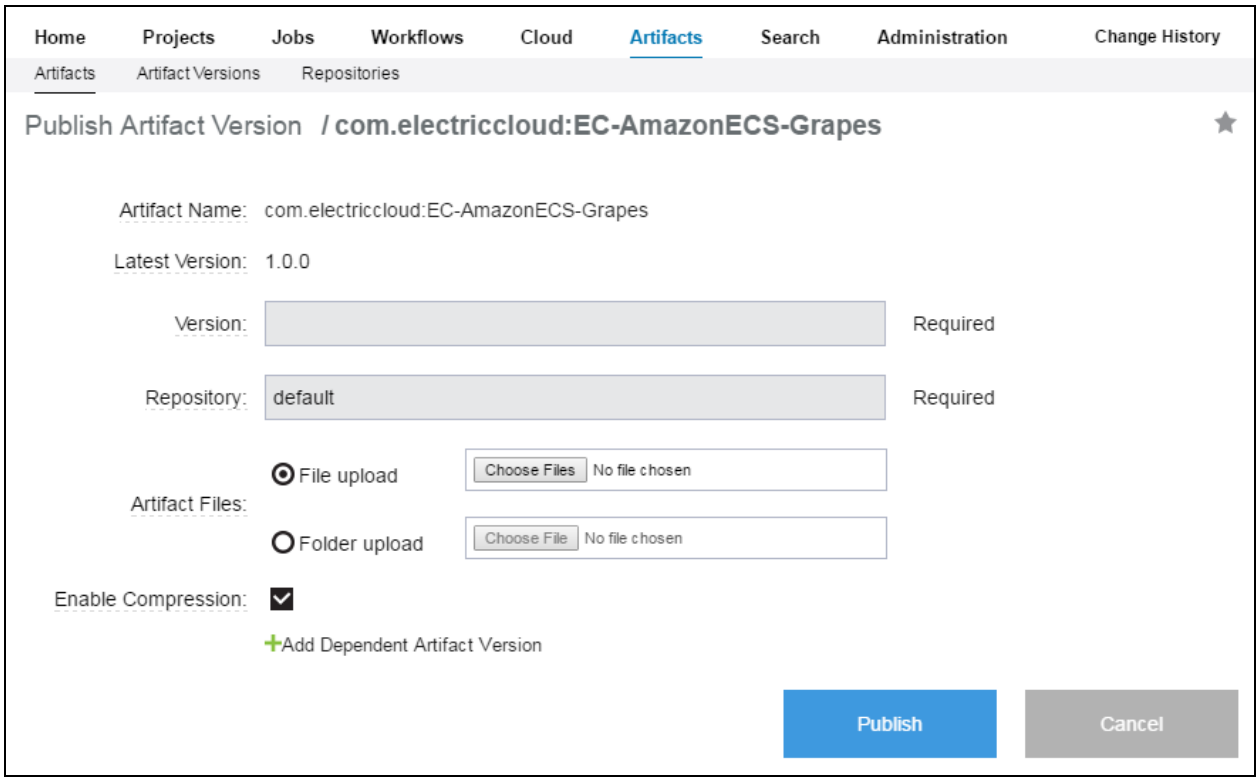
   The **Artifact Details** page appears. For example:



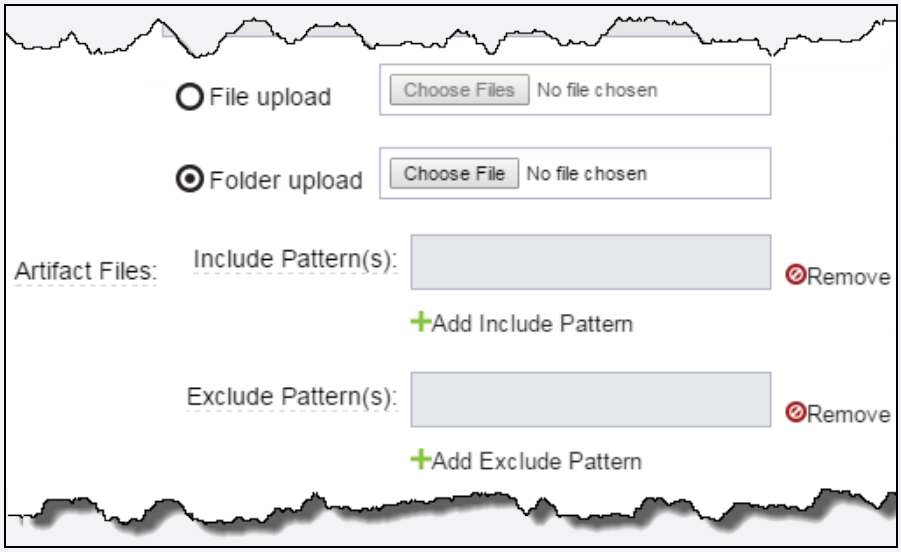
3. Click **Publish Artifact**.

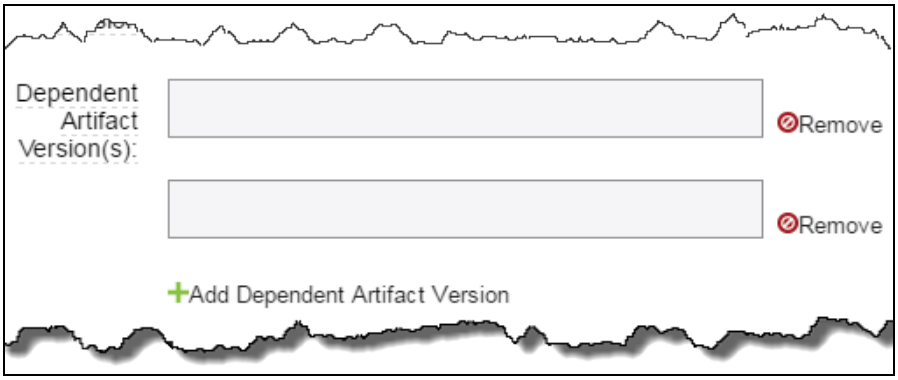
   If this button is not visible, you might not have the proper permissions to create an artifact version.

   The **Publish Artifact Version** page appears. For example:



4. Complete the page as follows:

Field name	Description
Version	Fully-qualified artifact version string. The version string must be of the form *<major>.<minor>.<patch>-<qualifier>-<buildNumber>*. The version string must be unique across all of this artifact's versions.
Repository	Name of the repository where the new artifact version will be published.
Enable Compression	Compress the artifact version before storing it in the repository. The default is enabled (the check box is checked).
Artifact Files	Browse files or folders for publishing to the repository: • **File Upload**—Lets you choose a single artifact. • **Folder Upload**—Lets you choose a folder that contains one or more artifacts. If you choose this option, you can add any combination of include or exclude patterns to filter files in or out:  The browser does not allow the upload if a linked file or folder with a broken link is specified.

Field name	Description
Add Dependent Artifact Version	(Optional) Dependent artifact version. You can enter one dependent artifact version per line, each in the form *<groupId>:<artifactKey>:<versionRange>*. All dependent artifact versions must exist for this artifact version to be retrieved. When this artifact version is successfully retrieved, its dependent artifact versions are retrieved also. 

5. Click **Publish**.

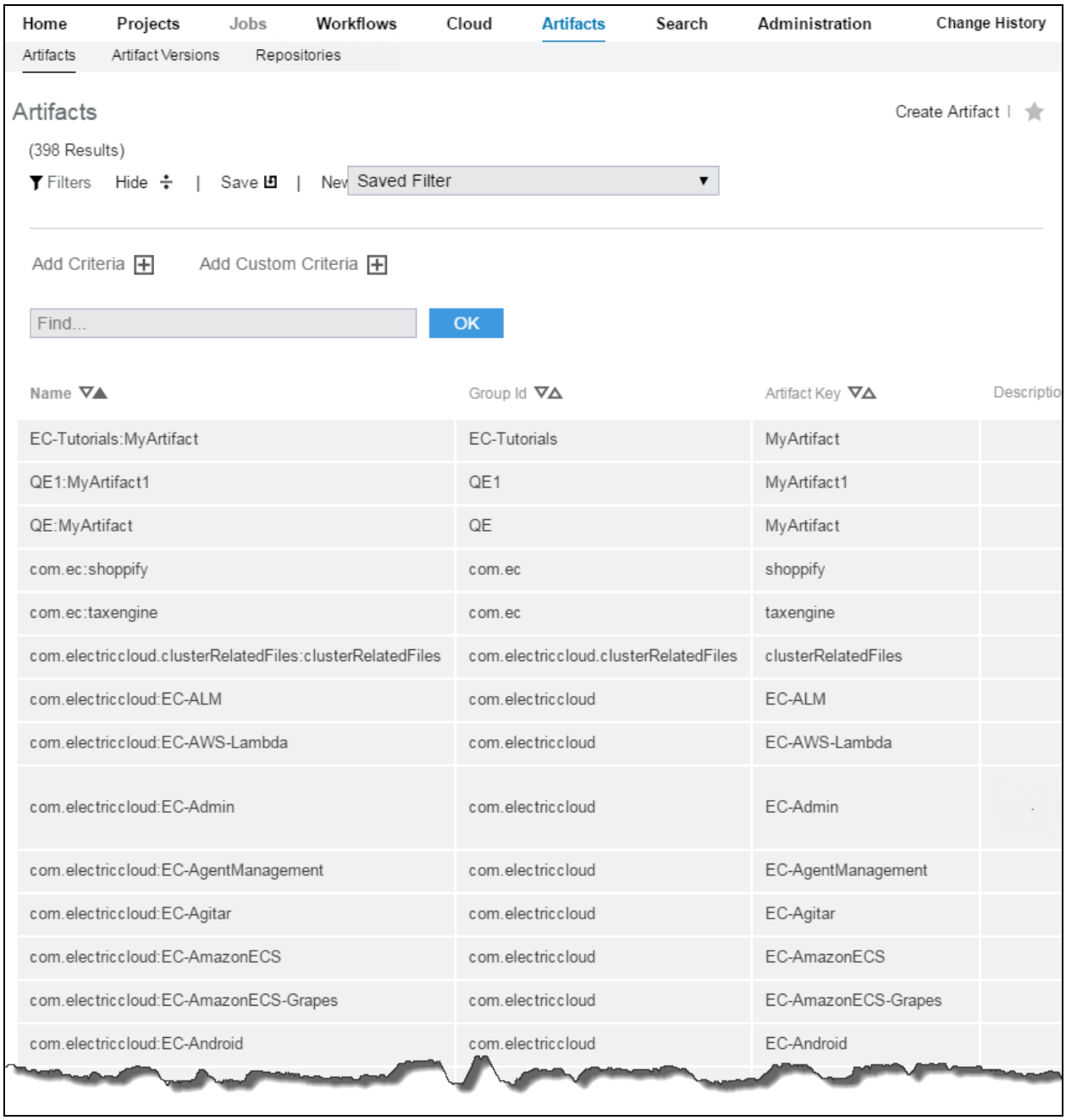
   The page that shows the list of artifacts with the updated artifact appears.

Creating a New Artifact and Uploading Files to Publish the First Version

To upload and publish the first version of an artifact that you create:

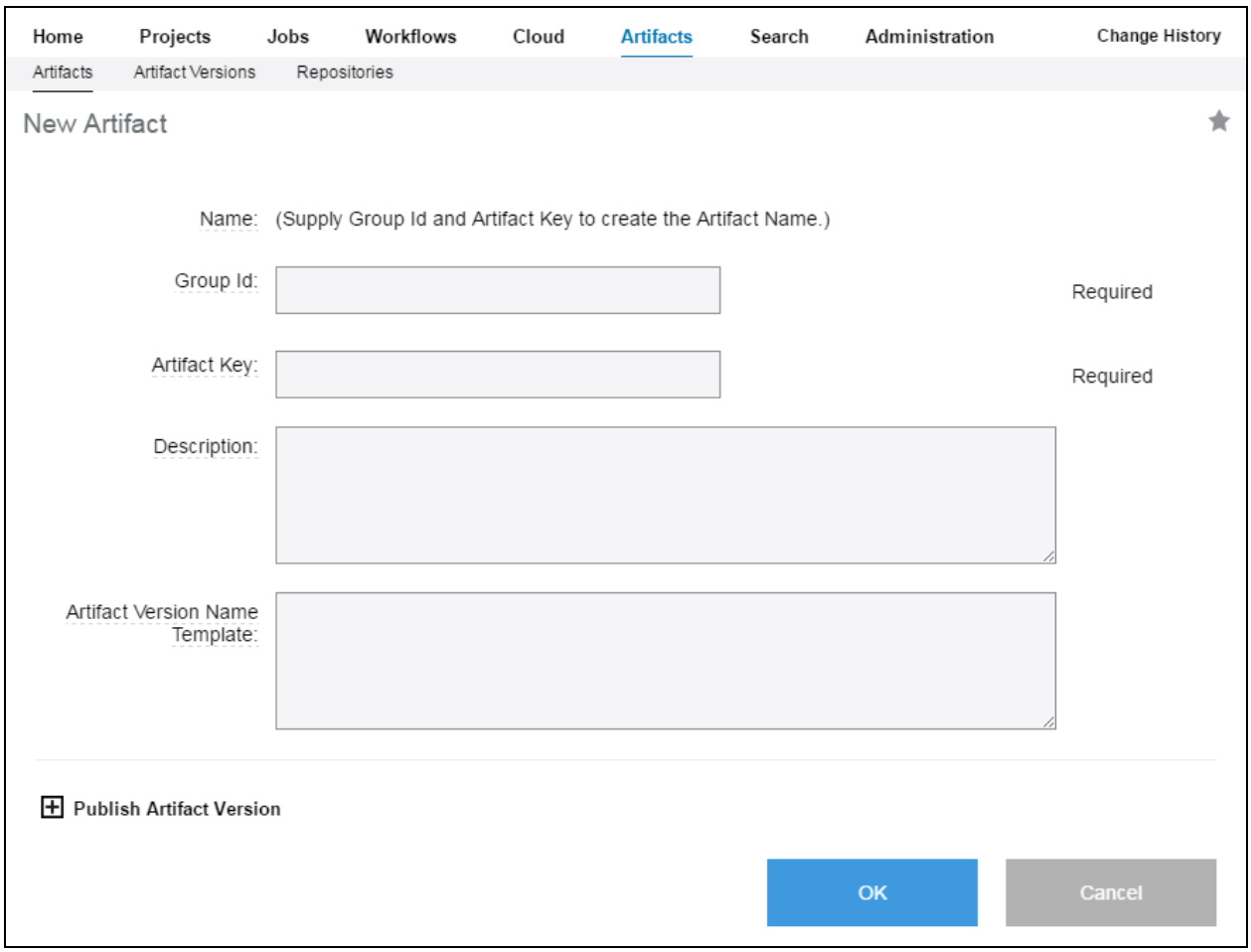
1. In the Automation Platform, choose **Artifacts**> **Artifacts**.

   The list of existing artifacts appears. For example:


Home
Projects
Jobs
Workflows
Cloud
Artifacts
Search
Administration
Change History
Artifacts
Artifact Versions
Repositories
Artifacts
Create Artifact
(398 Results)
Filters
Hide
Save
New
Saved Filter
Add Criteria
Add Custom Criteria
Find...
OK
Name
Group Id
Artifact Key
Descriptio
EC-Tutorials:MyArtifact
EC-Tutorials
MyArtifact
QE1:MyArtifact1
QE1
MyArtifact1
QE:MyArtifact
QE
MyArtifact
com.ec:shoppify
com.ec
shoppify
com.ec:taxengine
com.ec
taxengine
com.electriccloud.clusterRelatedFiles:clusterRelatedFiles
com.electriccloud.clusterRelatedFiles
clusterRelatedFiles
com.electriccloud:EC-ALM
com.electriccloud
EC-ALM
com.electriccloud:EC-AWS-Lambda
com.electriccloud
EC-AWS-Lambda
com.electriccloud:EC-Admin
com.electriccloud
EC-Admin
com.electriccloud:EC-AgentManagement
com.electriccloud
EC-AgentManagement
com.electriccloud:EC-Agitar
com.electriccloud
EC-Agitar
com.electriccloud:EC-AmazonECS
com.electriccloud
EC-AmazonECS
com.electriccloud:EC-AmazonECS-Grapes
com.electriccloud
EC-AmazonECS-Grapes
com.electriccloud:EC-Android
com.electriccloud
EC-Android

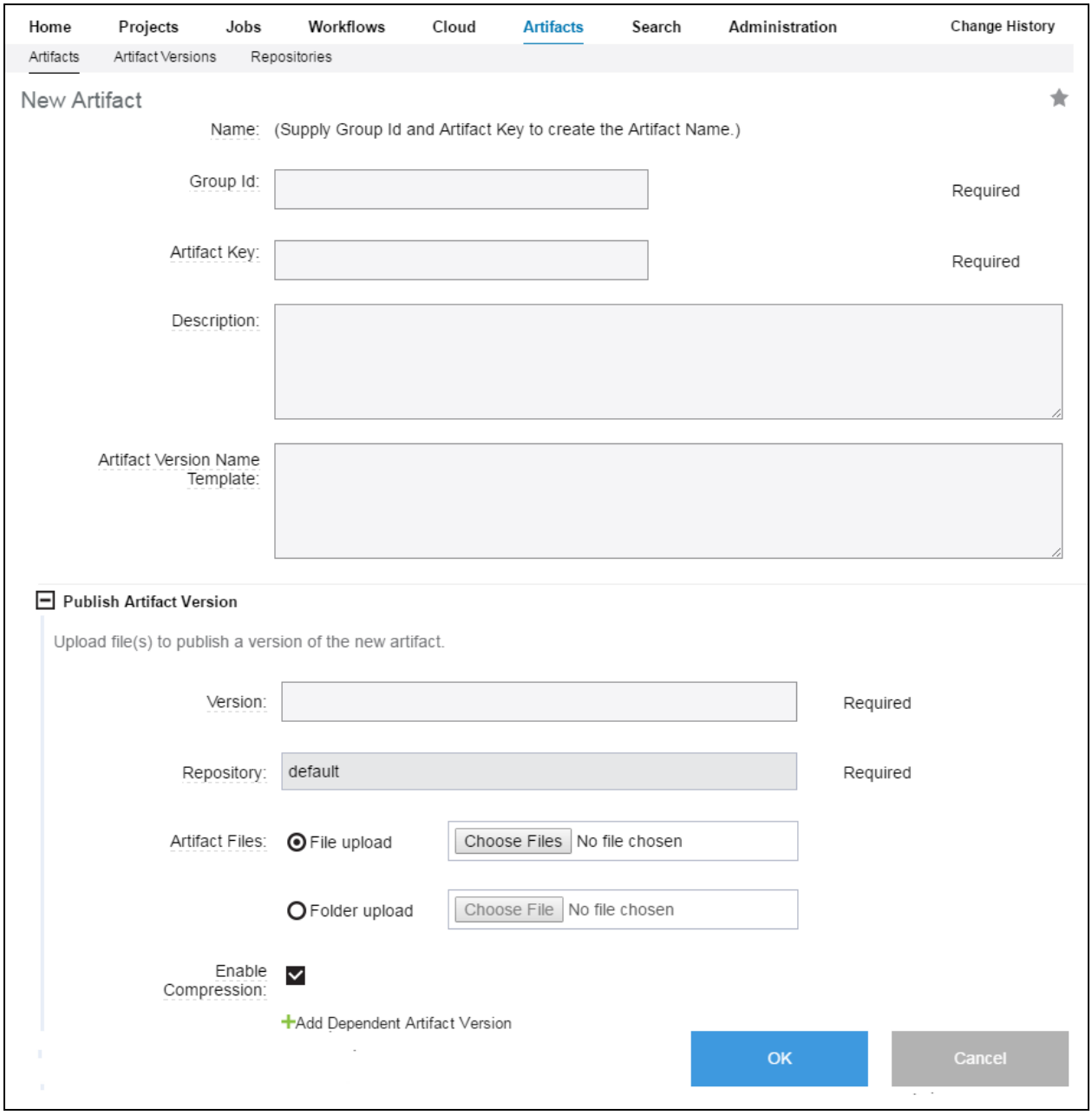
2. Click **Create Artifact**.

   The **New Artifact** page appears:



3. Click **Publish Artifact Version**.

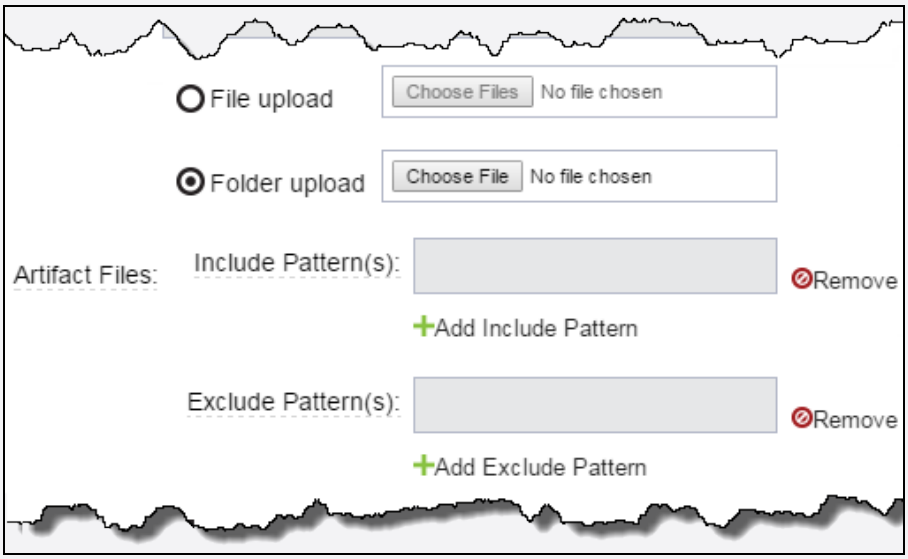
   The page expands to let you upload files from a file system to publish a version of the new artifact:

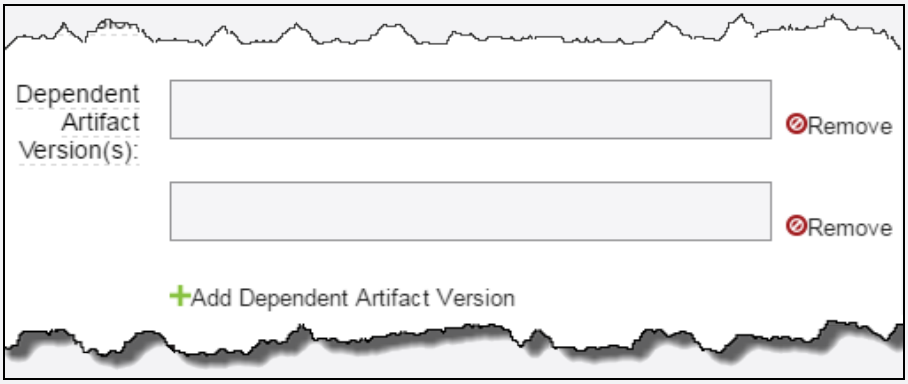


   Notice that the **Name** field is derived from the group ID and the key of the artifact version, which you will enter in the next step.

4. Complete the page as follows:

Field name	Description
Group Id	Group ID of the artifact version.
Artifact Key	Artifact ID of the artifact version.
Description	Text that describes this object. ElectricFlow does not interpret this text.
Artifact Version Name Template	Template used to determine the default names of artifact versions.
Publish Artifact Version	
Version	Fully-qualified artifact version string. A full version string must be of the form *<major>.<minor>.<patch>-<qualifier>-<buildNumber>*. The version string must be unique across all of this artifact's versions.
Repository	Name of the repository where the new artifact version will be published.

Field name	Description
Artifact Files	Browse files or folders for publishing to the repository: • **File Upload**—Lets you choose a single artifact. • **Folder Upload**—Lets you choose a folder that contains one or more artifacts. If you choose this option, you can add any combination of include or exclude patterns to filter files in or out:  The browser does not allow the upload if a linked file or folder with a broken link is specified.
Enable Compression	Compress the artifact version before storing it in the repository. The default is enabled (the check box is checked).

Field name	Description
Add Dependent Artifact Version	(Optional) Dependent artifact version. You can enter one dependent artifact version per line, each in the form *<groupId>:<artifactKey>:<versionRange>*. All dependent artifact versions must exist for this artifact version to be retrieved. When this artifact version is successfully retrieved, its dependent artifact versions are retrieved also. 

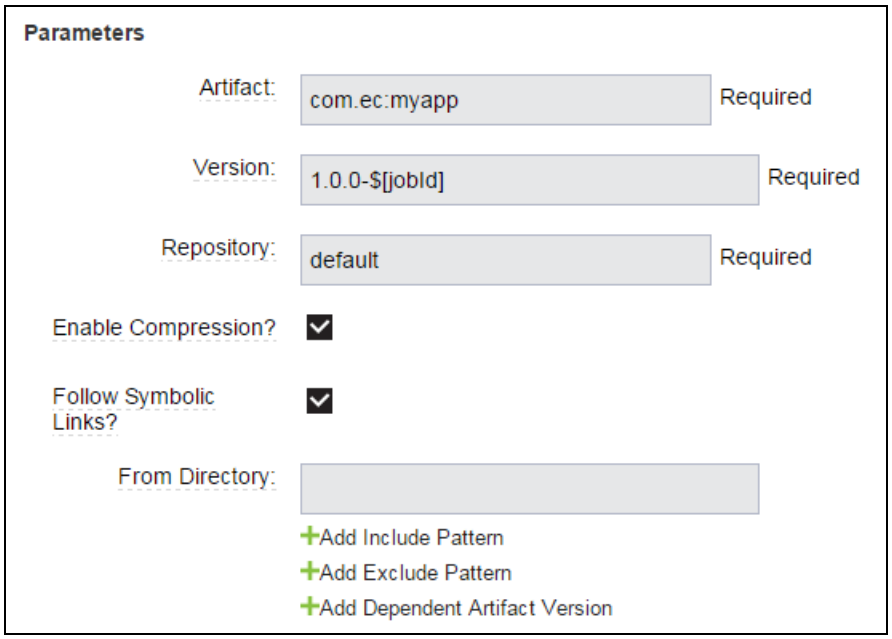
5. Click **OK**.

   The page that shows the list of artifacts with the new artifact appears.

Publishing Artifact Versions

An artifact version is published by using a procedure step or the *ectool / ec-perl* command-line interface. In most cases, using the Publish Artifact Version step is sufficient.

The following screen illustrates the parameters section for creating the Publish Artifact Version step:



Enabling Compression

When publishing an artifact, consider whether or not to enable compression. Using compression reduces transfer time during publish. However, compression also adds overhead when computing the compressed data. If files included in the artifact version are primarily text files or are another highly compressible file format, the benefit of reduced transfer time outweighs the cost of computing compressed data.

Artifact versions that contain installers, jars, audio, and video are almost certainly **not** compressible because these file types are already compressed, so the cost of compressing outweighs the (near zero) benefit of reducing transfer time. Another consideration: Artifact versions are stored in the same format as they are transferred, so highly compressible artifact versions use less space in the repository backing store.

Using Include and Exclude Patterns

When publishing an artifact, you can choose which files to include by using include and exclude patterns. File patterns are expressed as relative paths under the From Directory. Pattern syntax and behavior is the same as Ant and uses the following wildcard characters:

?—matches a single character

*—matches any number of characters, but only at a single directory level

**—matches any number of directory levels

Examples:

Use `*.jar` to match any `.jar` file in the top-level directory.

Use `*/*.jar` to match any `.jar` file in any child directory.

Use `**/*.jar` to match any `.jar` file at any level.

Retrieving Artifact Versions

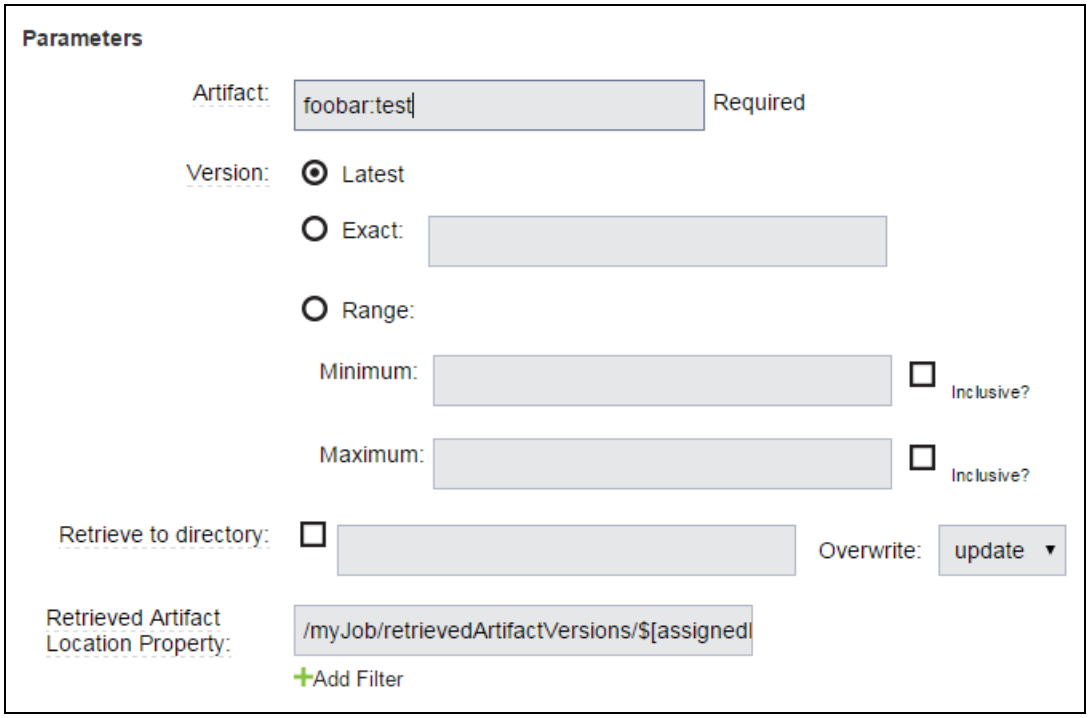
An artifact version is retrieved using a procedure step, or the `retrieveArtifactVersion` API call, used by the ectool/ec-perl command-line tools. This API call returns the most current artifact version that meets your criteria, per the algorithm described below.

Typically, you specify a version number range. Artifact version ranges are specified using interval notation. Brackets `[]` indicate versions to include and parentheses `()` indicate versions to exclude.

For example, the following artifact versions are published:

```
foobar:test:3.0.1-3645

foobar:test:1.2.3-2139

foobar:test:2.0.0-4000

foobar:test:2.0.0-DE-3395

foobar:test:2.0.0-DE-2445

foobar:test:2.0.0-DE

foobar:test:3.0.0-3539

foobar:test:4.0.0-5584
```

You want to retrieve the most current artifact version for the `foobar:test` artifact. To do this, you do not need to provide a version or version range, but instead enter retrieve parameters in the Retrieve Artifact Version step page as follows:



The equivalent command using ectool is:

```
ectool retrieveArtifactVersions --artifactName foobar:test
```

and with ec-perl, the syntax is:

```
my $cmdr = newElectricCommander();

$cmdr->retrieveArtifactVersions({artifactName => "foobar:test"});
```

Version numbers for two artifact versions (Artifact Version 1 and Artifact Version 2) are compared in the server as follows:

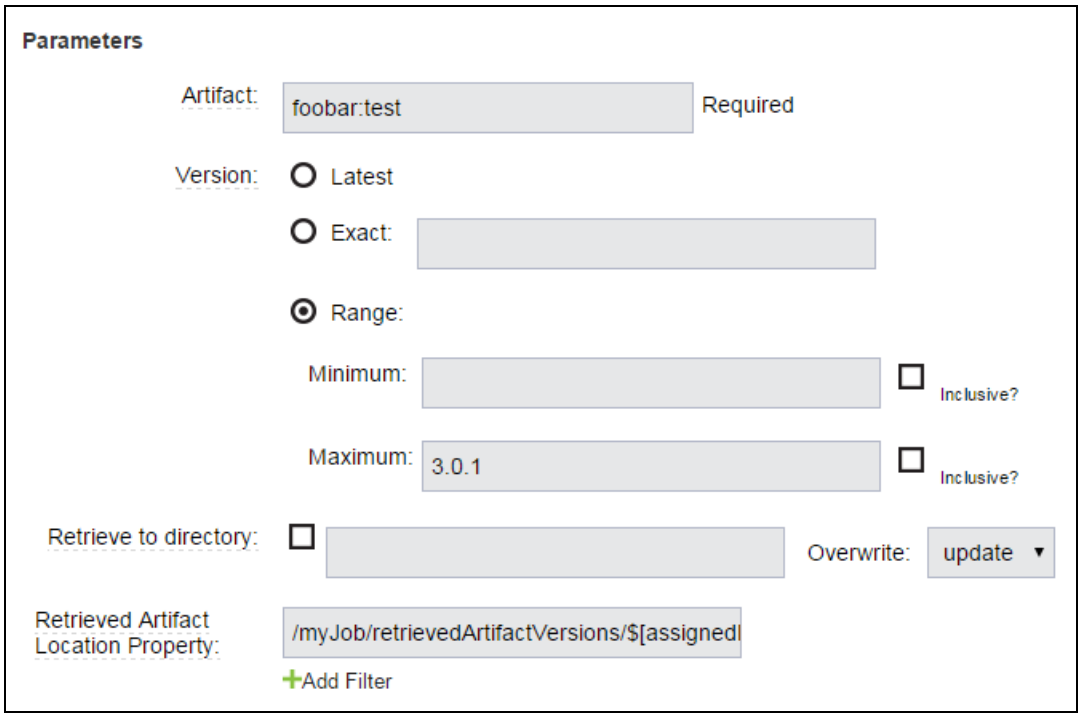
1. Compare `major.minor.patch`.
2. If they are equal, compare `qualifiers`.
3. If they are equal, compare `buildNumbers`.

> **Note:** This comparison algorithm does not take the time of publish into account. "Current" means the artifact version with the greatest version per the above algorithm.

So the above list of artifact versions would be sorted (most current to least) as follows:

```
foobar:test:4.0.0-5584

foobar:test:3.0.1-3645

foobar:test:3.0.0-3539

foobar:test:2.0.0-DE-3395

foobar:test:2.0.0-DE-2445

foobar:test:2.0.0-DE

foobar:test:2.0.0-4000

foobar:test:1.2.3-2139
```

If you want to retrieve the latest 3.0.0 artifact version, set maximum version to 3.0.1 "exclusive". The parameters entered in the retrieve step would look like this:



Setting this constraint ensures the entire "version string" is properly evaluated. In other words, you want to ensure all artifact versions with 3.0.0, including qualifiers and build numbers, are included in the evaluation.

The equivalent command using ectool is:

```
ectool retrieveArtifactVersions --artifactName foobar:test
    --versionRange "(,3.0.1)"
```

And with ec-perl, the syntax is:

```
my $cmdr = newElectricCommander();

$cmdr->retrieveArtifactVersions({artifactName => "foobar:test",
        versionRange => "(,3.0.1)"});
```

Note the use of parentheses to exclude version "3.0.1".

Generally, while in development, you will want to specify a minimum artifact version and take the latest version published, then when an artifact has reached a "released" state, indicate a specific version.

Parameters and Environment Variables for Artifact Retrieval Retry Attempts

The following parameters and their corresponding environment variables for artifact retrieval retry attempts are available in the Automation Platform server settings:

- **Number of retries for Retrieve Artifact** and `COMMANDER_NUMBER_OF_RETRIES_FOR_RETRIEVE_ARTIFACT`—Number of retries when doing artifact retrieval. Defaults to 3 retries.

- **Initial delay between retries for Retrieve Artifact** and `COMMANDER_INITIAL_DELAY_BETWEEN_RETRIES_FOR_RETRIEVE_ARTIFACT`—Initial delay (in milliseconds) between retries when doing artifact retrieval using backing off algorithm. Defaults to 100 ms.
- **Maximum delay between retries for Retrieve Artifact** and `COMMANDER_MAXIMUM_DELAY_BETWEEN_RETRIES_FOR_RETRIEVE_ARTIFACT`—Maximum delay (in milliseconds) between retries when doing artifact retrieval using backing off algorithm. Defaults to 30000 ms.

You set the variables on a client machine before doing a manual retrieval from the client via the `ectool retrieveArtifactVersions` command.

Entering Filters During a Retrieve Operation

At times, a company might want to distinguish between artifact versions not just by version range but also by some other property of the artifact version. Examples are:

- Specifying a qualifier in the version string when publishing.

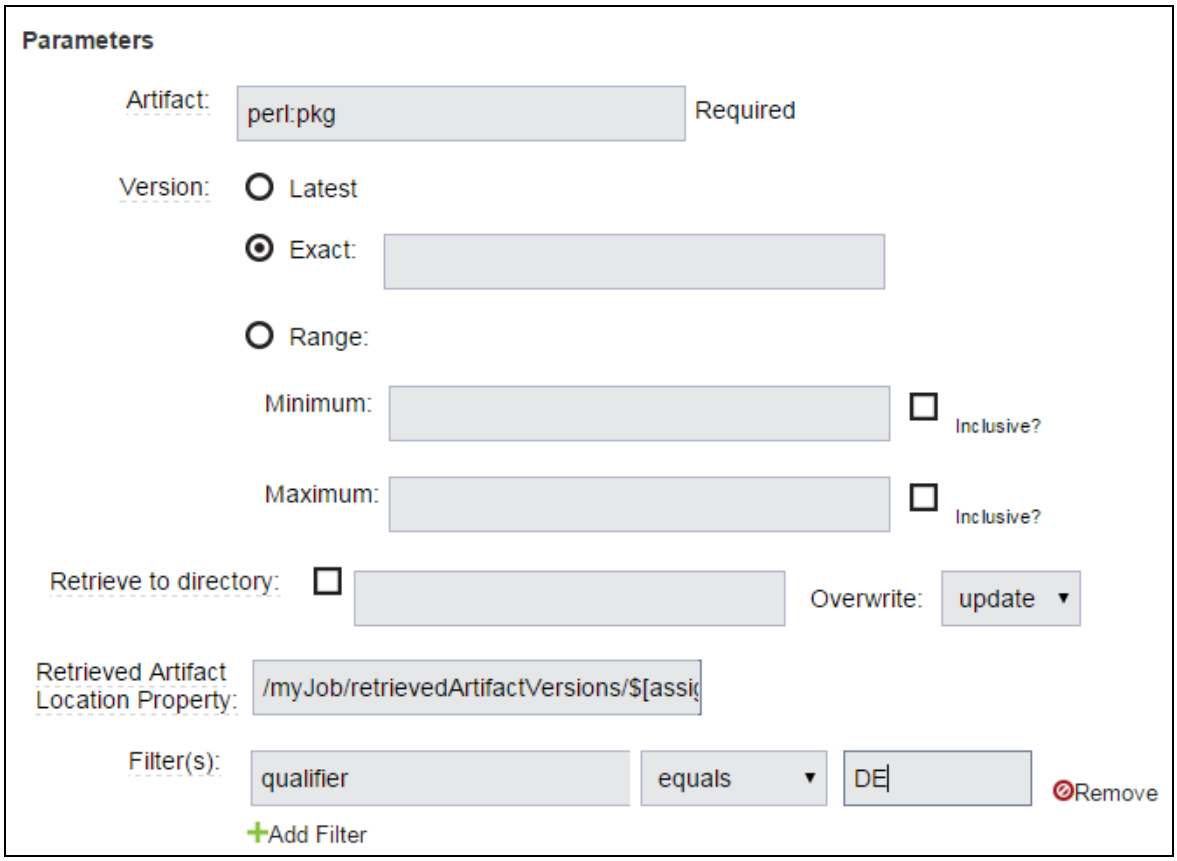
  For example, you might want to publish an artifact version for your product's English or German variants. You could publish the English version as `2.0.4-EN-55497` and the German version as `2.0.4-DE-55497`.

  When the qualifier string is used this way to classify artifact versions, it should be used as a filter to retrieve the latest version. This is described in the following example.

- Setting a custom property on the artifact version.

  For example, after your QA team has tested an artifact version, they could show approval by setting a `qaApproval` custom property, including the name of the QA engineer.

For either case, filters can be applied when retrieving an artifact version. To retrieve the latest German version for the artifact, the parameters entered on the retrieve artifact custom step page look like this:



Because `DE` was used as the qualifier, ElectricFlow will retrieve the latest artifact version for the `perl:pkg` artifact where the qualifier equals `DE`.

Only the following filter operators can be applied to the intrinsic property version:

`equals`

`greater than` (`greaterThan` ,for use in the API)

`less than` (`lessThan`, for use in the API)

`greater than or equals` (`greaterOrEqual`, for use in the API)

`less than or equals` (`lessOrEqual`, for use in the API)

"`equals`" is special because it does a string comparison on the version string of the artifact version, while the other operators compare the interpretation of the version string as described earlier.

All filter operators can be applied to version string components: `majorMinorPatch`, `qualifier`, or `buildNumber`. To find artifact versions with no qualifier, specify a filter on "qualifier" with operator "is not set" (or `isNull` in the API).

So for the QA approval case in a previous example, if a procedure is interested in an artifact version that was approved by QA, the retrieve step would specify a filter for "`qaApproval is set`".

ectool does not currently support the filters argument, but you can write a Perl script as follows:

```
my $cmdr = new ElectricCommander();

$cmdr->retrieveArtifactVersions({artifactName => "perl:pkg",
```

```
filters => {propertyName => "qaApprover",
operator => "isNotNull"}});
```

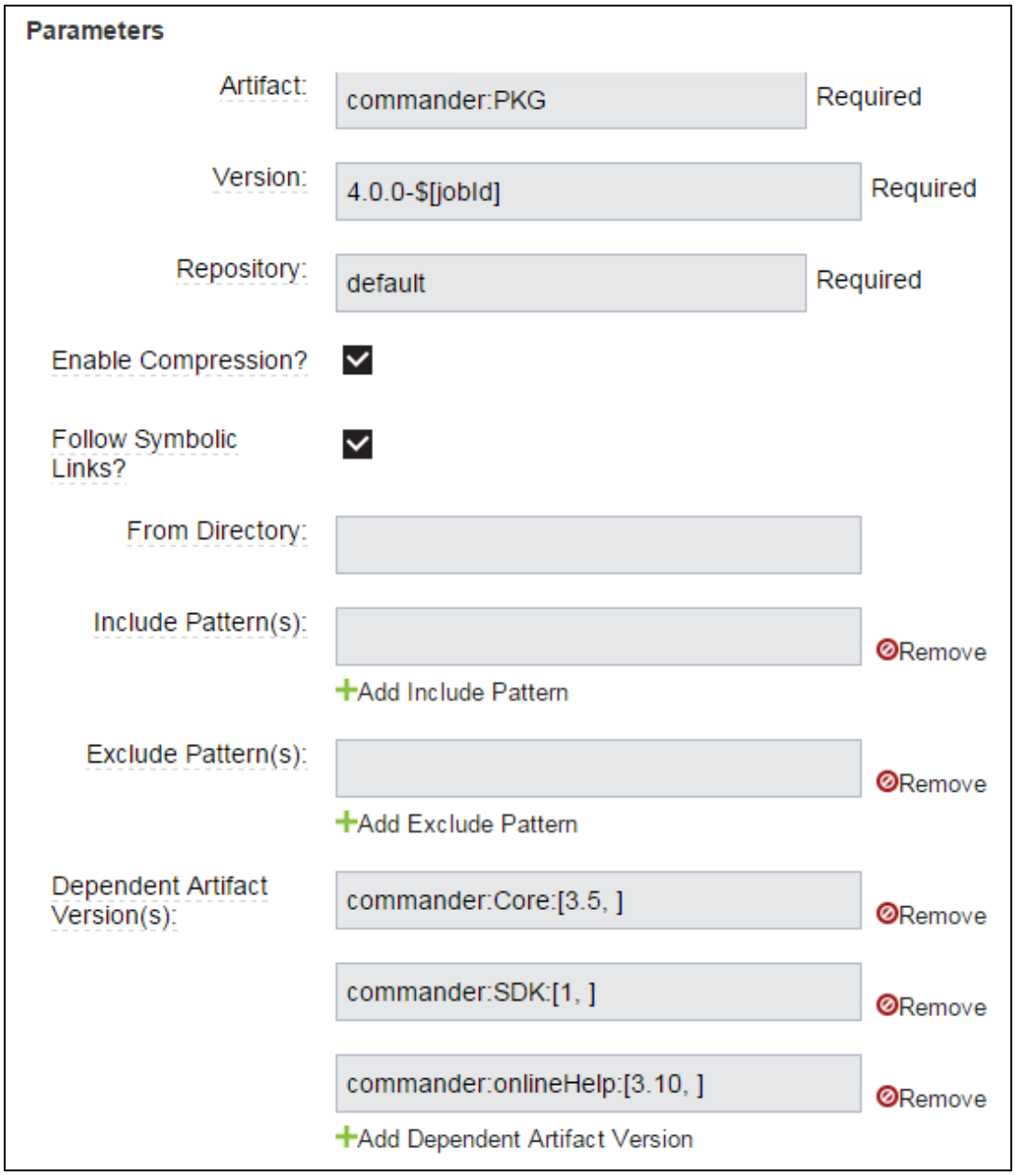
Entering Dependent Artifact Versions

A published artifact version can be dependent on a list of artifact versions. These dependent artifact versions are retrieved when the primary artifact version is retrieved and they are specified with a query syntax when publishing the primary artifact version.

For example, an artifact version for the ElectricFlow product could include dependent artifact versions for the core product, the SDK, and online Help:

- The latest `commander:Core` artifact version greater than or equal to version 3.5 (including 3.5)
- The most recent version of the `commander:SDK` artifacts
- The `commander:onlineHelp` artifact with a version greater than version 3.10 (excluding 3.10)

Because dependent artifact versions are evaluated during the retrieve process, specify them using the same syntax as you would for retrieving any artifact version, which is by including the group Id, the artifact key, and a specific artifact version or a version range. Parameters for the "publish" custom step page would be similar to the following:



A published artifact version could have zero files, that is, no files are published as part of the artifact version, but it still could have a list of dependent artifact versions. An empty artifact version might be useful as modular convenience container for retrieving a group of artifact versions in downstream processes.

Understanding Artifact Usage

ElectricFlow has an artifact repository that can store build outputs as artifacts and their versions. These artifacts are then used in component definitions for application models; they can be used in master component definitions as shared components. And when applications are deployed, these artifacts are deployed to various environments and their specific resources. You should understand various use cases in which a specific artifact version is used. Following are a few examples.

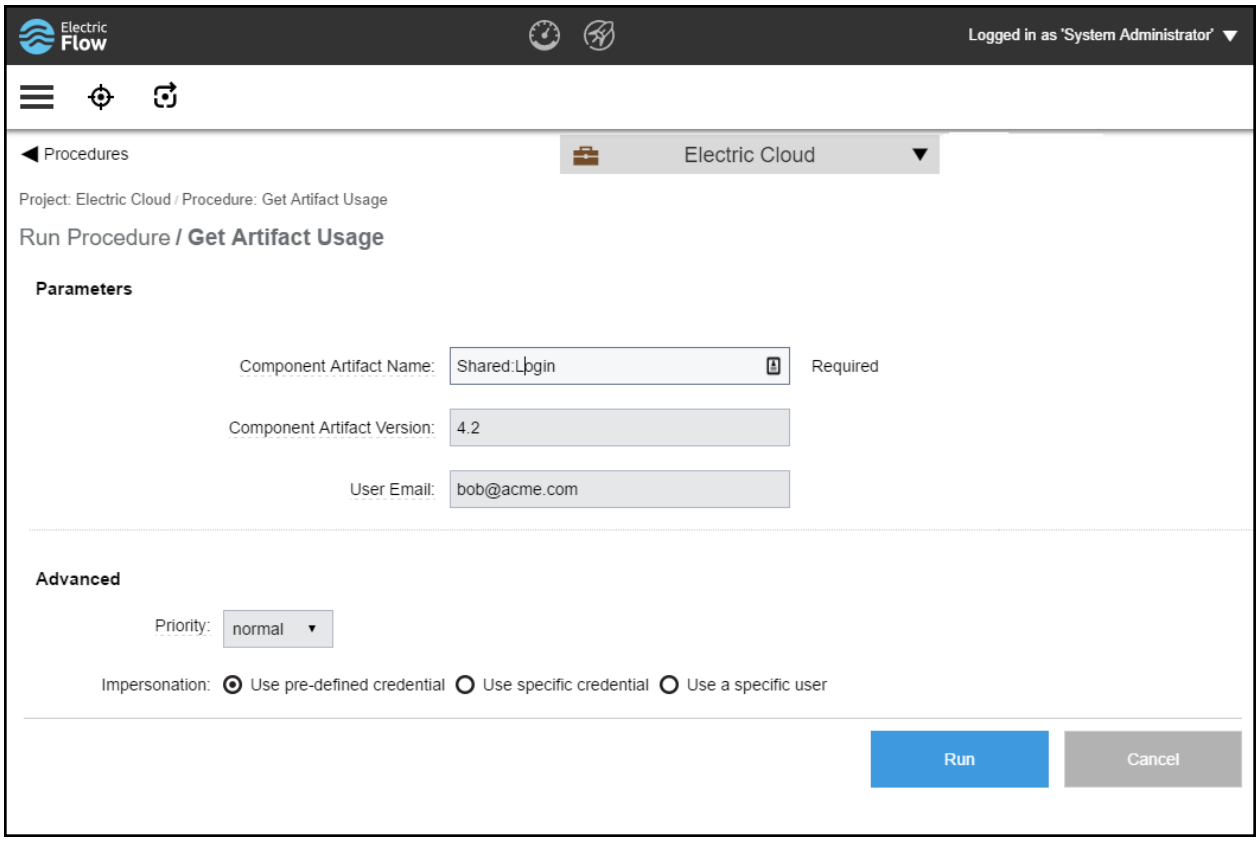
Example: Performing Vulnerability Analysis of an Artifact

Suppose that an artifact version is used in various application models and master components and that the artifact is deployed to various environments. Also suppose that it is discovered that the particular artifact version has a security vulnerability. In such cases, it might be important to quickly know the usage of that artifact version so that corrective action can be taken.

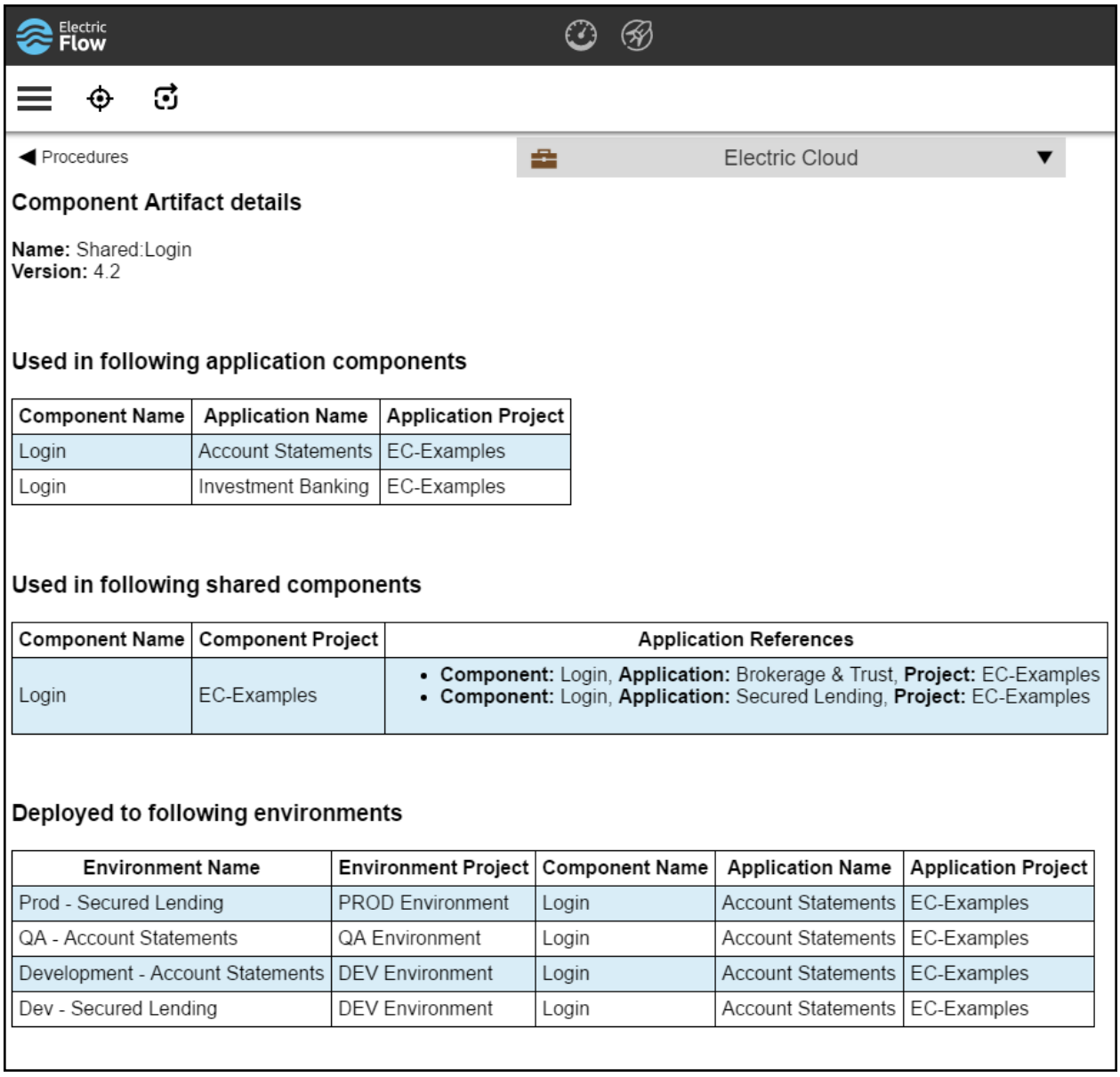
Example: Understanding Dependencies

Suppose that users have created application components using a copy of a master component. If the definition of that master component changes, you should understand the dependency and impact so that owners of applications that leverage the master component can take corrective action.

You can use the Get Artifact Usage procedure in the Electric Cloud project to understand how and where an artifact and component is used to perform vulnerability analysis or to understand dependencies or any other artifact usage use case. When this procedure is run, it asks for inputs such as a component artifact name and version as well as an email address for sending out a usage report.



For example, suppose that you want to generate a report for the usage of an artifact named Shared:Login with version 4.2 and then email the report to a user. You can just enter that information and run the procedure. This generates a report that shows clearly which master components, applications, and environments have links to the 4.2 version of the Shared:Login artifact.



Procedures

Electric Cloud

Component Artifact details

Name: Shared:Login
Version: 4.2

Used in following application components

Component Name	Application Name	Application Project
Login	Account Statements	EC-Examples
Login	Investment Banking	EC-Examples

Used in following shared components

Component Name	Component Project	Application References
Login	EC-Examples	• **Component:** Login, **Application:** Brokerage & Trust, **Project:** EC-Examples • **Component:** Login, **Application:** Secured Lending, **Project:** EC-Examples

Deployed to following environments

Environment Name	Environment Project	Component Name	Application Name	Application Project
Prod - Secured Lending	PROD Environment	Login	Account Statements	EC-Examples
QA - Account Statements	QA Environment	Login	Account Statements	EC-Examples
Development - Account Statements	DEV Environment	Login	Account Statements	EC-Examples
Dev - Secured Lending	DEV Environment	Login	Account Statements	EC-Examples

Users can easily understand the current usage of an artifact or a master component, and the system will email this report to the respective users. This procedure can be extended easily to take corrective actions if needed, such as redeploying a new version of the artifact to all the current environments to remove the security vulnerability.

Understanding the Artifact Cache

Artifact versions are retrieved to an artifact cache directory before being consumed by a build process. Properly configured, the artifact cache directory can significantly improve efficiency as you work with artifacts.

The artifact cache directory is set in one of these ways:

- Enter information in the Artifact Cache Directory field on the Edit Resource page.
- Use the `modifyResource` API to set the `artifactCacheDirectory` intrinsic property.
- Set the `artifactCache` field in the `agent.conf` file.

Thus, each resource can have its own artifact cache location (directory). Alternatively, multiple resources and an entire site can share an artifact cache if the cache directory resides on a shared file server.

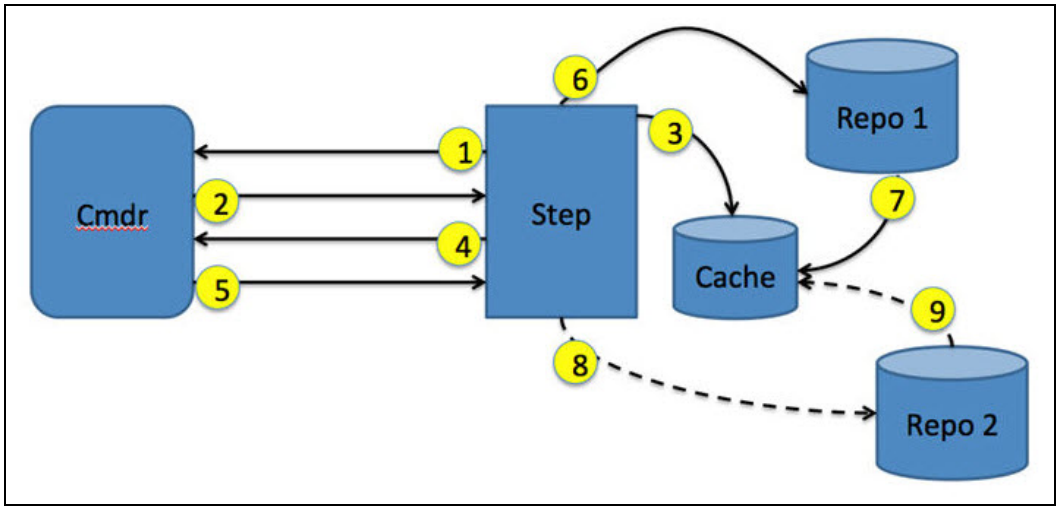
When an ElectricFlow step retrieves an artifact version, remember that often a version range or no version at all is specified. For example, to retrieve the latest ElectricFlow SDK artifact version, you could issue the following command:

```
ectool retrieveArtifactVersions --artifactName commander:SDK
```

No version is specified because we want the "latest" artifact version published.

In the following example, the step sends a request (1) to the ElectricFlow server:

1. The ElectricFlow server replies (2) with a fully qualified artifact version name. In this case, it returns `commander:SDK:1.2.0.43552`.
2. The step then looks in its artifact cache for the artifact version (3).
3. If the artifact is not found in the artifact cache, a request is issued (4) to get a list of repositories to search to find the artifact version.
4. The ElectricFlow server replies (5) with a list of repositories and the order in which to search them.
5. The step requests the artifact version from the first (6) repository in the list.
6. If the artifact version is found in the repository, the artifact version is retrieved to the cache (7).
7. If the artifact version is not found, the step requests the artifact version from the next repository in the search order (8).
8. If the artifact version is found in that repository, the artifact version is retrieved to the cache (9).
9. Otherwise, if the artifact version is not found in any repository, an error is displayed.



In this example, if the requested artifact version was found in the artifact cache (step 3), the resource would not have communicated with either of the repository servers. Also, any potential network latencies would have been avoided from retrieving the artifact version from the repository server.

If you have a cache shared by multiple resources and steps that run on those resources need to use the same artifact version, the first step requesting an artifact version retrieves it from the repository to the cache. Subsequent job steps using the artifact version will have immediate access to the artifact version in the cache, eliminating the need for each step to retrieve its own copy of an artifact version.

Cleaning Up Repositories and Caches

> **Note:** This method does not work for artifacts in an Amazon Simple Storage Service (Amazon S3) repository.

Deleting an artifact version in the ElectricFlow server does not delete it in the repository backing store, and it does not remove the artifact version from an artifact cache.

ectool and the ElectricFlow Perl module include a function for each of these tasks: `cleanupRepository` and `cleanupArtifactCache`. Both of these API calls are intended to be run on the machine containing the `cache/backingstore` directory that needs to be cleaned up.

The cleanup algorithm works as follows:

1. Reviews the directory tree looking for directories three-levels deep.
2. For each such directory, interprets the three path components as <`groupId`, `artifactKey`, `version`>.
3. Issues a `findObjects` request to the ElectricFlow server to see if an artifact version with that specification exists.
4. Deletes all directories identified by <`groupId`, `artifactKey`, `version`> and not recognized by the ElectricFlow server.

To operate correctly, using the `cleanupRepository` API *requires* read access to all artifact version objects stored in ElectricFlow and therefore must be run in an "admin" session. We also recommend running `cleanupArtifactCache` in an "admin" session—if not, artifact versions that are simply not readable by the user invoking the cleanup will be deleted from the cache.

In addition, both functions fail if the server does not recognize any of the chosen artifact versions. Thus, if the cleanup function is called on a directory that is not a `backingstore` directory or and artifact cache, it will not delete anything.

The recommended approach for removing stale artifact versions from a backing store is to use an ElectricFlow maintenance procedure that periodically runs on each repository server (which presumably has an agent install). Similarly, use a procedure with a broadcast step to run on each agent to clean up its cache. Note that the job step session does not necessarily have read privileges on all artifact versions, so it is important to login as a privileged user, then perform cleanup with those credentials.

Authenticating Users for LDAP and Active Directory

ElectricFlow uses account information from multiple sources. In most cases, the primary account information source is an external LDAP or Active Directory repository: both user and group information is retrieved from the repository. *Local* users and groups are defined within ElectricFlow.

Use the following LDAP information and examples to enter information in your LDAP template. If you are using the Active Directory template, you may still use the LDAP information as reference—the templates are similar with only a slight difference in the Active Directory template.

Configuring LDAP

A number of options need to be customized for any LDAP configuration. The following sample configuration shows how we have our own LDAP configuration set up. After the sample, see a list of properties with a description.

Sample LDAP Configuration

```
<bean id="LdapDirectoryProvider"
   class="com.electriccloud.security.ldap.
   LdapDirectoryProviderImpl">
   <property name="providerName" value="LDAP"/>
   <property name="url" value="ldap://dir.
      electric-cloud.com/dc=electric-cloud,dc=com"/>
   <property name="managerDn" value="uid=JohnDoe,ou=People,=
      electric-cloud,dc=com"/>
   <property name="managerPassword" value="****"/>
   <property name="userBase" value="ou=People"/>
   <property name="userSearchFilter" value="uid={0}"/>
   <property name="userNameAttribute" value="uid"/>
   <property name="fullUserNameAttribute" value="gecos"/>
   <property name="emailAttribute" value="mail"/>
   <property name="groupBase" value="ou=Group"/>
   <property name="groupMemberFilter" value="(|(member={0})
      (memberUid={1}))"/>
   <property name="groupMemberAttributes" value="member,
      memberUid"/>
   <property name="groupSearchFilter" value="(|(
      groupOfNames)(objectClass=posixGroup))"/>
   <property name="groupNameAttribute" value="cn"/>
</bean>
```

The following properties configure LDAP mapping:

`emailAttribute`—(optional) The attribute in a user record that contains the user's email address. If the attribute is not specified, the account name and domain name are concatenated to form an email address.

`fullUserNameAttribute`—(optional) The attribute in a user record that contains the user's full name (first and last) for display in the UI. If this attribute is not specified or the resulting value is empty, the user's account name is used instead.

`managerDn`—(optional) The DN of a user who has read-only access to LDAP user and group directories. If this property is not specified, the server attempts to connect as an unauthenticated user. Not all servers allow anonymous read-only access.

> **Tip:** This user does not need to be an admin user with modify privileges.

`managerPassword`—(optional) If the `managerDn` property is set, this password is used to authenticate the manager user.

`providerName`—This human readable name will be displayed in the user interface to identify users and groups that come from this provider.

`userBase`—This string is prepended to the `basedn` to construct the directory DN that contains user records.

`userNameAttribute`—The attribute in a user record that contains the user's account name.

`userSearchFilter`—This LDAP query is performed in the context of the user directory to search for a user by account name. The string `"r;{0}"` is replaced with the user's login ID. Typically, the query compares a user record attribute with the substituted user login ID.

`url`—The LDAP server URL is in the form `protocol://host:port/basedn`. Protocol is either `ldap` or `ldaps` for secure LDAP). The port is implied by the protocol, but can be overridden if it is not at the default location (389 for ldap, 636 for ldaps). The `basedn` is the path to the top level directory that contains users and groups at this site. This is typically the domain name where each part is listed with a `dc=` and separated by commas.

> **Note:** Spaces in the `basedn` must be URL encoded (`%20`).

In addition to user information, the LDAP server can be queried for group information. This query is optional because the local group mechanism can refer to LDAP users and *local* users. However, the following elements can be used to tell the server how to map groups in LDAP

`groupBase`—(optional) This string is prepended to the `basedn` to construct the directory DN that contains group records.

`groupMemberAttributes`—(optional) A comma separated attribute names list that identifies a group member. Most LDAP configurations only specify a single value, but if there is a mixture of POSIX and LDAP style groups in the directory, multiple attributes might be required.

`groupMemberFilter`—(optional) This LDAP query is performed in the groups directory context to identify groups that contain a specific user as a member. There are two common forms of group record in LDAP directories: POSIX style groups where members are identified by account name, and `groupOfNames` or `uniqueGroupOfNames` records where members are identified by the full user DN. Both forms are supported, so the query is passed two parameters: `"r;{0}"` is replaced with the full user record DN, and `"r;{1}"` is replaced with the user's account name.

`groupNameAttribute`—(optional) The group record attribute that contains the name of the group.

`groupSearchFilter`—(optional) This LDAP query is performed in the context of the groups directory to enumerate group records.

Determining LDAP Mapping

A typical POSIX user record in LDAP looks similar the example below. To set up a mapping for this record, it is necessary to identify various record components. First, identify the path in the directory that contains user records. In this example, the build user has a distinguished name (dn) of `uid=build,ou=People,dc=mycompany,dc=com`. This name uniquely identifies the build user account and this path splits into three parts:

base dn: `dc=mycompany,dc=com`

user base: `ou=People`

user element: `uid=build`

The `baseDn` is the parent of the directory that contains users. This value should be combined with the protocol and server to form the URL. In this case, the URL is `ldaps://dir/dc=mycompany,dc=com`.

Next, the `userBase` is the portion of the path that identifies the directory containing all user account records. This value is used directly as the `userBase` configuration element.

The remaining portion identifies the user without the People directory: `uid=build`. The user name is replaced in this value with the string `"r;{0}"` to form the `userSearchFilter: uid={0}`. This query allows the server to search for a user's account name by looking for a record with a matching `uid` attribute.

The final mapping step is to identify user record attributes that hold the account name, full user name, and (optionally) the user's email address. In this example, the account name is `uid` (identified earlier), the full user name attribute is `gecos`, and there is no email attribute.

At this point, the server is able to authenticate a user, look up a user by name, and determine the user's full name. For many installations this is sufficient.

Sample LDAP User Record

```
# build, People, electric-cloud.com
dn: uid=jdoe, ou=People, dc=mycompany,dc=com
loginShell: /bin/bash
uidNumber: 508
gidNumber: 508
objectClass: account
objectClass: posixAccount
objectClass: top
objectClass: shadowAccount
uid: jdoe
gecos: John Doe
 cn: John
homeDirectory: /net/filer/homes/build
```

Also, you can configure the server to look for *LDAP groups* that refer to user accounts. A typical group record is shown below. Like a user account, an LDAP group has a distinguished name with a `baseDn`, a group base, and a group element. In this case, the `basedn` is still `dc=mycompany,dc=com`. The `groupBase` configuration element is `ou=Group`, and the group name is `cn=build_users`.

The server needs to identify records in the directory that correspond to groups—it does this by applying the `groupMemberFilter` to the records in the `groupBase` directory. In this case, group records are members of the `posixAccount` object class, so the filter can be set to `objectClass=posixGroup`. To display a group by its name, the server needs to know which attribute represents the group name. In this case, set the `groupNameAttribute` to `cn`.

Finally, the server needs a filter to determine which accounts belong to the group and the attribute name that represents a single group member. Group membership can be identified in one of two ways. Either the members are listed by account name, or by their LDAP distinguished name. In this case, POSIX group membership is determined by account name, so set the `groupMemberAttributes` property to `memberUid`, and set the `groupMemberFilter` to `memberUid={1}`.

Sample LDAP Group Record

```
# build_users, Group, mycompany.com
dn: cn=build_users,ou=Groups,dc=mycompany,dc=com
objectClass: posixGroup
objectClass: top
gidNumber: 100
memberUid: jdoe
 memberUid: mary
cn: build_users
```

Sample Active Directory Configuration File

The following XML file defines parameters needed to connect to an Active Directory server and the query to use for looking up user information.

```
<beans xmlns="http://www.springframework.org/schema/beans"

   xmlns:

   xmlns:tx

   xmlns:aop

   xsi:schemaLocation

   http://www.springframework.org/schema/beans/spring-beans-2.0.xsd

   http://www.springframework.org/schema/tx

   http://www.springframework.org/schema/tx/spring-tx-2.0.xsd

   http://www.springframework.org/schema/aop

   http://www.springframework.org/schema/aop/spring-aop-2.0.xsd" default-lazy-
   init="true"

      <bean id="ADDirectoryProvider"

      class="com.electriccloud.security.ldap.ActiveDirectoryProviderImpl">

      <property name="providerName" value="ActiveDirectory" />

      <!-- START OF CUSTOMIZATIONS -->

      <!-- URL to AD server of the form: "r;ldap://host:port/base_dn" -->

      <property name="url" value="ldap://server:389/dc=company,dc=com" />

      <!-- Required credentials for an account that has read-only access to the server
      -->

      <property name="managerDn"

      value="cn=myuser,cn=Users,dc=company,dc=com" />

      <property name="managerPassword" value="mypw" />

      <property name="userBase" value="cn=Users" />
```

```
        <property name="userSearchFilter" value="(&(sAMAccountName={0})
        (objectClass=user))" />
        <property name="userNameAttribute" value="sAMAccountName" />
        <property name="fullUserNameAttribute" value="name" />
        <!-- Optional group configuration. This is only needed if you intend -->
        <!-- to manage groups externally via AD. -->
        <!-- <property name="groupBase" value="value=""/>-->
        <!-- <property name="groupMemberFilter" value="member={0}"/> -->
        <!-- <property name="groupMemberAttributes" value="member"/>-->
        <!-- <property name="groupSearchFilter" value="(objectClass=group)"/>-->
        <!-- <property name="groupNameAttribute" value="cn"/>-->
        <!-- Max number of results to get back in one query. This must be -->
        <!-- no greater than the AD server's setting, which is typically 1000. -->
        <property name="pageSize" value="500" />
        <!--   END OF CUSTOMIZATIONS -->
        </bean>
</beans>
```

LDAP Group Hierarchy

The LDAP or Active Directory provider can now be configured to recognize the nested group hierarchy in the LDAP or Active Directory server in the following use cases.

Access Control

When **Recursively Traverse Group Hierarchy** is selected (enabled) for the directory provider and access control is set up for a given LDAP group in ElectricFlow, the users belonging to any nested groups in LDAP will automatically inherit the access control rules defined for the parent groups. For more information about **Recursively Traverse Group Hierarchy**, see To create a new Active Directory provider on page 1261 and To create a new LDAP directory provider on page 1265 for the directory provider configuration details.

In this LDAP hierarchy:

- Group1
  - User11
  - Group11
    - User111
    - User112
    - Group111
  - Group12
    - User121
    - User122

These access control rules have been set up with **Recursively Traverse Group Hierarchy** enabled:

- Group1: `Project:Default Read->Allow, Modify->Allow, Execute->Deny`
- Group12: `Project:Default Read->Allow, Modify->Deny, Execute->Allow`

Because the users in Group111 and Group11 are also considered to be members of Group1, the access control rules defined for Group1 also apply to them:

```
Project:Default Read->Allow, Modify->Allow, Execute->Deny
```

Because the users in Group12 are also considered members of both Group1 and Group12, the access control rules defined in both groups may apply to the users in Group 12. However, ElectricFlow does not support the inherent group hierarchy. Thus, ElectricFlow does not apply special precedence logic based on the group hierarchy, and the same precedence logic that is currently applied for a user belonging to two or more groups will apply in this situation, giving `Deny` a higher precedence:

```
Project:Default Read->Allow, Modify->Deny, Execute->Deny
```

Manual Task Assignees and Approvers

If **Recursively Traverse Group Hierarchy** and **Include Nested Group Users as Approvers** are both enabled for the directory provider, users in nested LDAP groups are allowed to complete or approve a manual task when a parent LDAP group is assigned as an assignee or an approver for the task. For more information about these membership options, see To create a new Active Directory provider on page 1261 and To create a new LDAP directory provider on page 1265for the directory provider configuration details.

In the following LDAP hierarchy with both **Recursively Traverse Group Hierarchy** and **Include Nested Group Users as Approvers** enabled:

- Group1
  - User11
  - Group11
    - User111
  - Group12
    - User121

When Group1 is assigned as an approver for a manual task, User11, User111, and User 121 can approve the task.

Notifications

If **Recursively Traverse Group Hierarchy** and **Include Nested Group Users in Notifications** are both enabled for the directory provider, users in nested LDAP groups will be included when sending notifications to a parent LDAP group. For more information about these membership options,see To create a new Active Directory provider on page 1261 and To create a new LDAP directory provider on page 1265To create a new LDAP directory provider on page 1265for the directory provider configuration details.

In the following LDAP hierarchy with both **Recursively Traverse Group Hierarchy** and **Include Nested Group Users in Notifications** enabled:

- Group1
  - User11
  - Group11
    - User111

- Group12
    - User121

When notifications are set up to be sent to Group1, they are sent to User11, User111, and User 121.

Automated Environment Discovery

Automated Environment Discovery, also referred to as *auto-discovery*, makes it easier to create models using your existing resources and environments for deployment automation. You need to set up the resources and environments before creating the deployment model and authoring processes in ElectricFlow. When deployments have large numbers of resources and environments, automated environment discovery speeds up application and microservice authoring by:

- Providing resource discovery using middleware plugins, such as EC-WebSphere, on existing resources and environments.
- Allowing the use of discovered resource configurations to accelerate process authoring and deployment automation.

You can initiate the discovery operation on resources from the following pages in the UI:

- At the platform level, go to the **Cloud** > **Resources** page.
- From the Environment Editor for a specific environment.
- From an environment tier in the Environment Editor.

Resources Page

Starting from the **Cloud** > **Resources** page:

1. Select one or more resources.

Example:

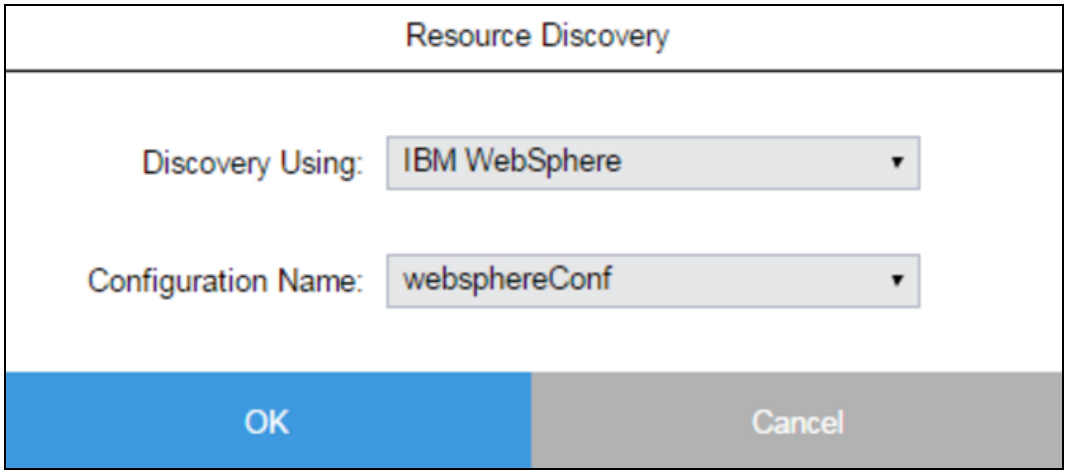
2. Click the **Resource Discovery** button ( ) in the upper right corner.

   The Resource Discovery dialog box opens.

3. In the **Discover Using** field, select the plugin to use to run the Discover procedure for the selected resources.

4. (Optional) If the plugin requires a configuration for the Discover procedure, select the configuration from the drop-down list in the **Configuration Name** field.

   Example:



5. Click **OK**.

If the discovery request was successfully submitted, a message appears in the dialog box:

If the discovery could not be run because the user does not have the appropriate permissions or because there were errors, an error message appears describing the error.

Environment Editor

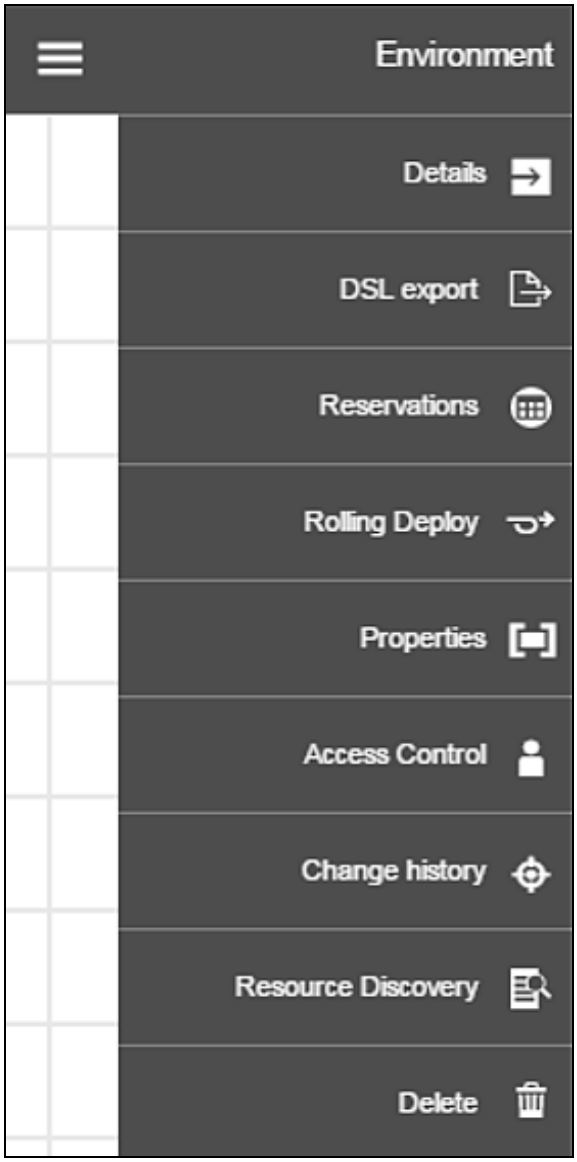
Starting from the Environments List:

1. Click on an environment name to select it.

   The Environment Editor opens.

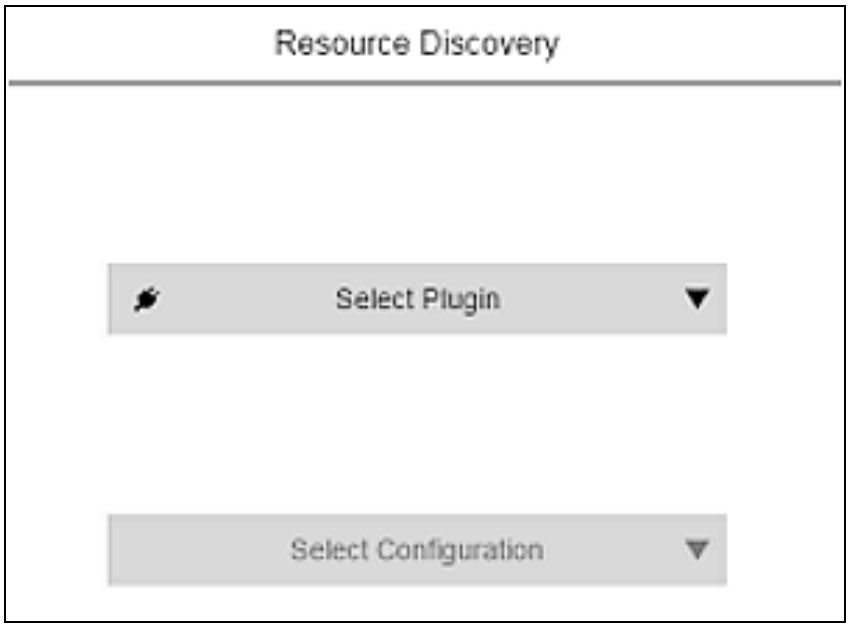
2. Click the button and select **Resources Discovery**.

Example:



The Resource Discovery dialog box opens.

Example:



3. In the **Discover Using** field, select the plugin to use to run the Discovery procedure for the selected resources.
4. (Optional) If the plugin requires a configuration for the Discovery procedure, select the configuration from the drop-down list in the **Configuration Name** field.

Example:

5. Click **OK**.

If the discovery request was successfully submitted, a message appears in the dialog box:

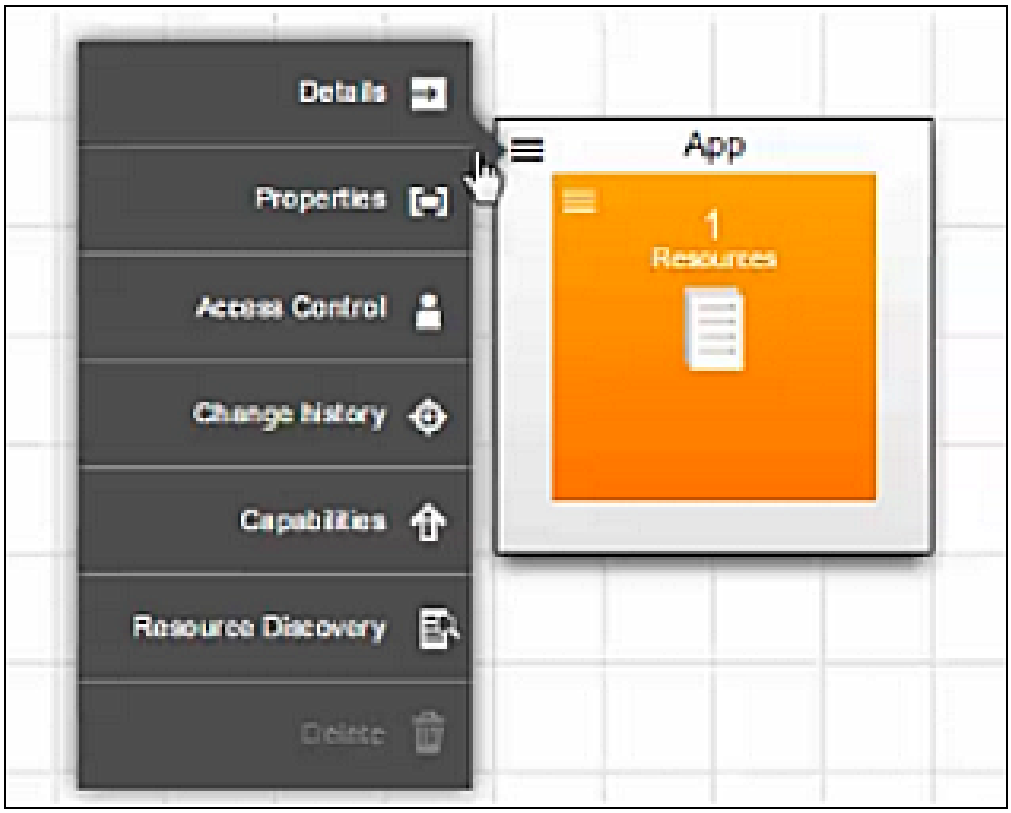
If the discovery could not be run because the user does not have the appropriate permissions or because there were errors, an error message appears describing the error.

Environment Tier

Starting from an Environment Tier:

- Select an environment tier in the Environment Editor.
- Click the button for the tier, and select **Resource Discovery**.

Example:

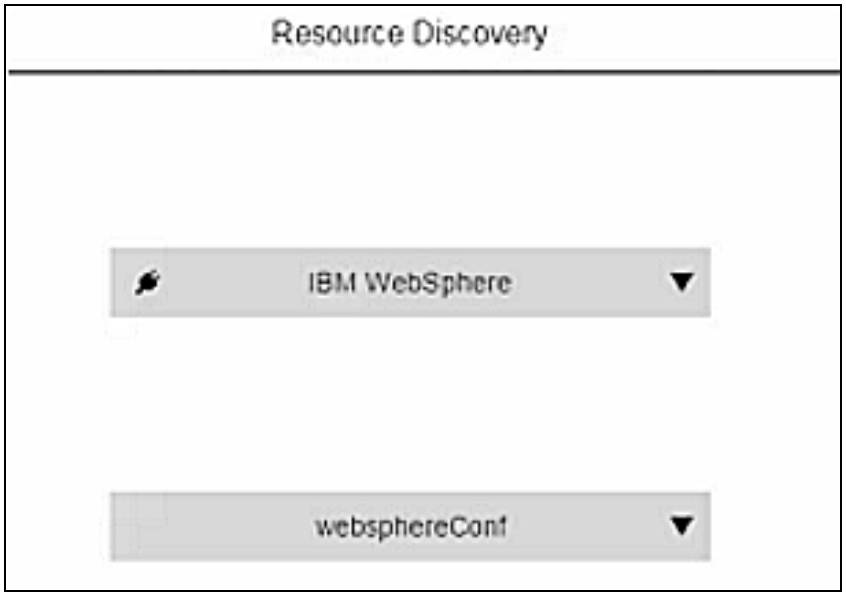


The Resource Discovery dialog box opens.

- In the **Discover Using** field, select the plugin to use to run the Discovery procedure for the selected resources

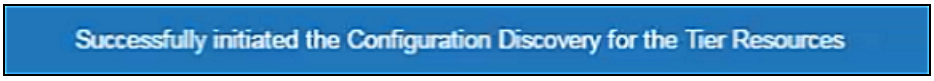
- (Optional) If the plugin requires a configuration for the Discovery procedure, select the configuration from the drop-down list in the **Configuration Name** field.

  Example:



- Click **OK**.

If the discovery request was successfully submitted, a message appears in the dialog box:



If the discovery could not run because the user does not have the appropriate permissions or because there were errors, an error message appears describing the error.

Change Tracking

Change Tracking is designed to track every change between every state of non-runtime ElectricFlow objects and to allow you to revert to any previous state of these objects. ElectricFlow tracks the changes to tracked objects including applications or microservices, procedures, workflows, workspaces, resources, and project-owned components such as library components and records a *Change History* of the historical states of the system and the changes between them. The change history has a copy of every state in which every non-runtime object has been, the object's current state, and indexing records for searching through the database. The tracked objects are not affected when ElectricFlow executes an object that is usually created or modified at execution time, such as a workflow or process.

Change tracking allows you to do the following:

- When debugging a failed job or looking for more information about a component in an ElectricFlow, see the Change History for the changes relevant to that object.

- When searching for specific change history records, filter the records by time frame, change type, entity type, or developer.
- Revert changes to an object or to an objects and its children.
- When you want to determine the differences between objects, export them at various levels in the object hierarchy.

In ElectricFlow, you can use Change Tracking in all aspects of the ElectricFlow end-to-end solution:

- In build-test automation, track the Change History of artifacts, resources, and ACLs.
- In deployment automation, track the Change History of components, environment tiers, and process steps (application, microservice, or component). You also use Change Tracking with snapshots to make it easier to deploy reliable and repeatable software for Continuous Delivery.
- In releases and Pipelines, track the Change History of stages and tasks.

Configuring Change Tracking

Change Tracking must be enabled when ElectricFlow starts for your system to track changes and record the Change History.

By default, Change Tracking is enabled for projects created in ElectricFlow 5.3 or later. It is disabled for projects created in ElectricFlow versions before ElectricFlow 5.3, but can be enabled manually.

Enabling Change Tracking Globally

When installing ElectricFlow:

1. Add this line to the `database.properties` file:

   ```
   COMMANDER_DB_AUDITING_ENABLED=true
   ```

2. Restart the ElectricFlow server.

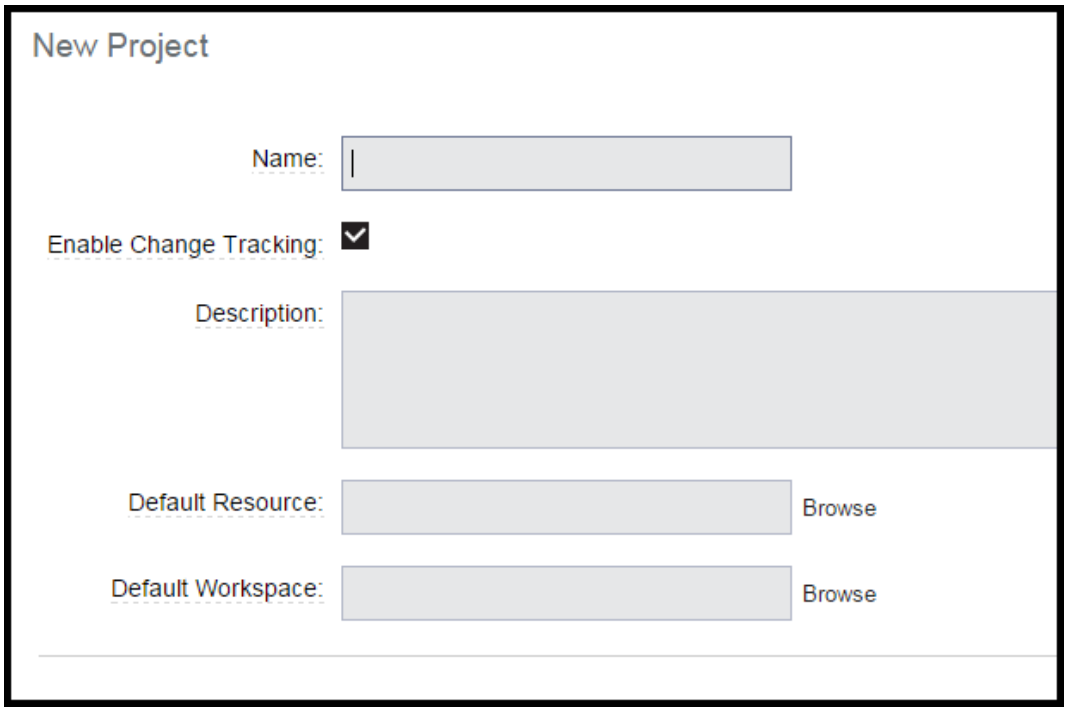
Enabling Change Tracking on a Per-Project Basis

> **Important:** Unless the `allowEnablingDisablingChangeTrackingProjectLevel` setting has been altered, only admin users are allowed to perform this task. The default setting is `adminOnly`. If the setting is changed to `anyoneWithAccess`, any user with access to the project can enable or disable Change Tracking for the project

You can enable Change Tracking one of the following ways:

In the ElectricFlow Platform UI

Change Tracking is enabled when the **Enable Change Tracking** check box is selected.



To disable Change Tracking, click the **Enable Change Tracking** check box to clear it. and click **OK**.

Using ectool

- Enter `ectool modifyProject <projectName> --tracked true` to enable Change Tracking.
- Enter `ectool modifyProject <projectName> --tracked false` to disable Change Tracking.

Using ec-perl:

- Enter `$cmdr->modifyProject(<projectName>, {tracked => true});` to enable Change Tracking.
- Enter `$cmdr->modifyProject(<projectName>, {tracked => false});` to disable Change Tracking.

For properties that are frequently updated always to be a numerical value, it is possible to suppress Change Tracking of numerical-value updates by using one of these commands:
ectool: `ectool modifyProperty <projectName> -- path <propertyPath> -- counter true`

or

ec-perl: `$cmdr-> modifyProperty (<projectName>, {path => <propertyPath>, counter => true});`

When the `counter` flag is set for a property, Change Tracking does not track changes to the property if the only change was a change in the property value from one numeric value to another. All other forms of changes to the properties that have this flag set are tracked normally.

Important: Reverting an object that owns a counter property to a previous state will revert the value of the counter property to its value at the previous time that a change to this property was

racked–typically this is when its value is initialized. This may not be the desired behavior, so you ay need to manually set a counter property if it is reverted.

Upgrading to ElectricFlow 6.x

Change Tracking is enabled when you upgrade to ElectricFlow 6.x. This can significantly increase the time it takes to complete the upgrade.

If you want to upgrade with Change Tracking disabled, add this line to the `database.properties` file before starting the upgrade:

```
COMMANDER_DB_AUDITING_ENABLED=false
```

Customizing the Change History Page

The performance of the Change Tracking feature is affected by number of records in the Change History as well as the number of entries being tracked. For example, a page showing 5000 entries may be slow to load and update and does not provide much useful information.

This ElectricFlow server uses the lowest of the following limits to determine the maximum amount of records to display in the Change History page:

- Maximum amount of records on the Change History page

  To change the maximum number of records in the Change History page:

  1. Set the `CHANGE_TRACKING_HARD_MAX_RECORDS` parameter in the `wrapper.conf` file to a new value.

     The default value is 1000.

  2. Restart the ElectricFlow server.

- Maximum number of records retreived

  Set the `TrackingMax Records` server setting to a new value not exceeding the `CHANGE_TRACKING_HARD_MAX_RECORDS` parameter in the `wrapper.conf` file.

  To set `TrackingMaxRecords`, do one of the following:

  Change the value in the UI. See the Server Settings page in the automation platform UI.

  Use ectool to change the value. For example, enter the following command to limit the number of records retrieved to 100:

  ```
  ectool setProperty /server/settings/changeTrackingMaxRecords --value 100
  ```

Viewing the Change History for Artifacts, Jobs, Projects, and Workflows

When troubleshooting why a job failed, you can view the Change History for artifacts, jobs, projects, and workflows in the automation platform.

- Artifacts on page 1378
- Jobs on page 1378
- Projects on page 1379
- Workflows on page 1380

Artifacts

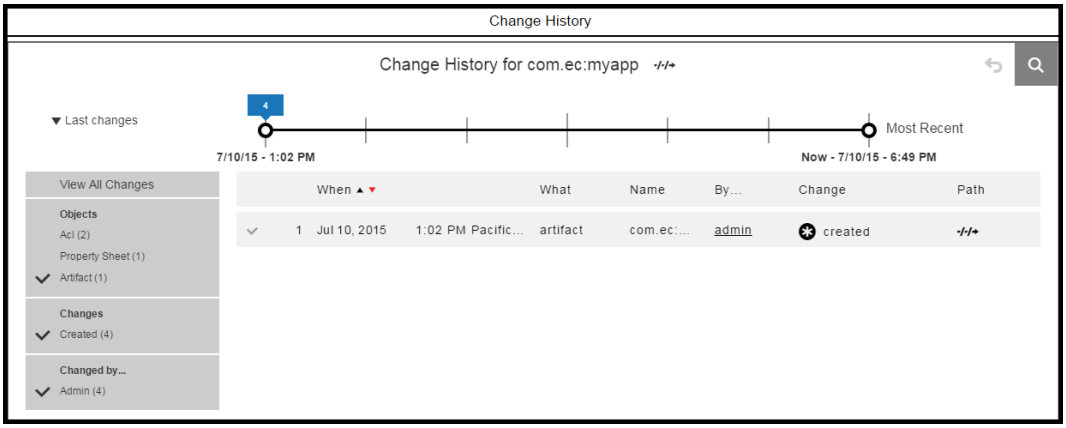
1. Open the home page of the Automation Platform web UI by browsing to https://<*ElectricFlow_server*>/commander/.
2. Go to the Artifacts tab.
3. Choose an artifact.
4. Click the **Track Changes**button.

Example:

The Change History for the selected artifact opens.

The default time increment is **Past 60 Minutes**.

Example:



Jobs

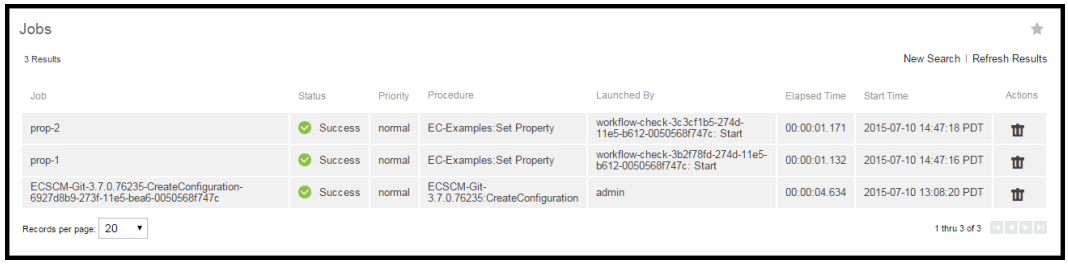
Starting from the Home page (https://<*ElectricFlow_server*>/commander/):

1. To go to the Job Details page, do one of the following:
   - Use the Jobs tab.
   - Use the Jobs Quick View list.

2. If you use the Jobs tab, follow these steps:
   - Click the Jobs tab.

     The Jobs page opens.

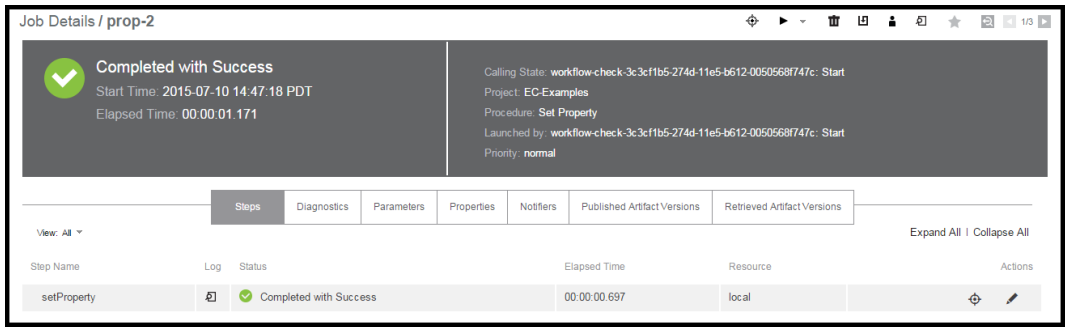
     Example:



   - Click a job name to select a job.

     The Job Details page opens.

     Example:



3. If you use the Jobs Quick View list, click a job name to select a job.

   The Job Details page opens.
4. Choose a job or job step.
5. Click **Track Changes** for the job or job step.

   The Change History for the job or job step opens.

   The default time increment is **Past 60 Minutes**.

Projects

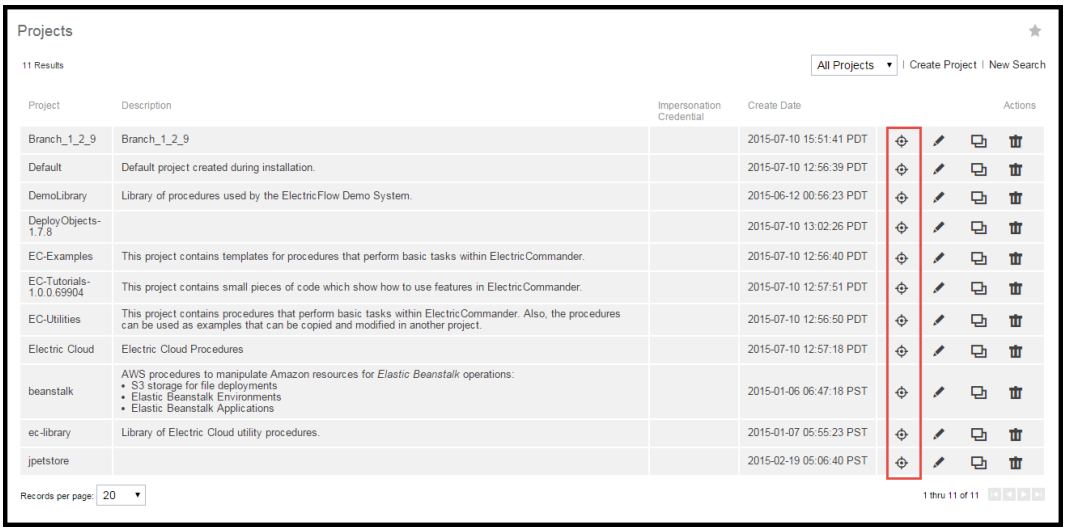
Starting from the Home page:

1. Go to the Projects tab.
2. Choose a project.

3. Click the **Track Changes** button.

   The Change History for the project opens.

   Example:



Workflows

Starting from the Home page:

1. Go to the Workflows tab.
2. Choose a workflow.
3. Click **Track Changes**.

   The Change History for the workflow opens.

Modifying What You See the Change History

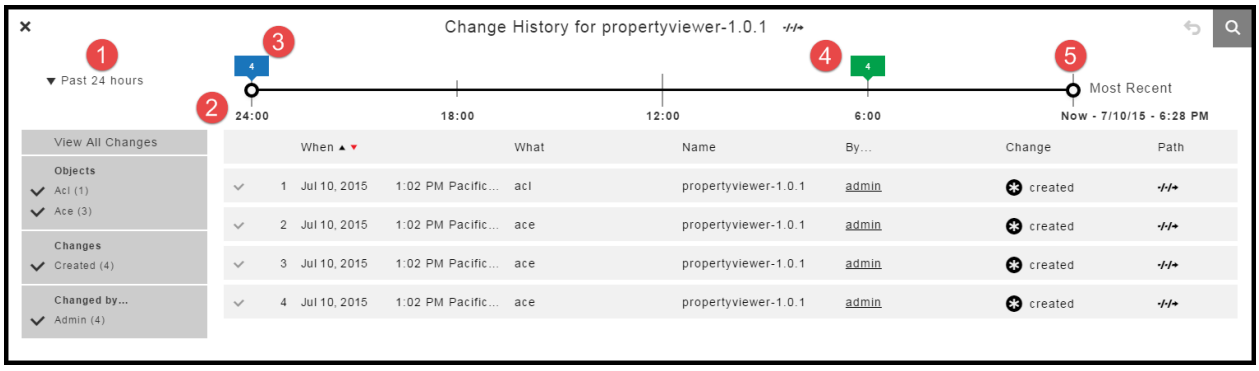
You can modify the information that appears in the Change History with these settings:

- Time line–See Change History Time Line on page 1380.
- Filters–See Change History Filters on page 1387.

Change History Time Line

The time line is at the top of the Change History page.

This example shows the Change History for a property called *propertyviewer-1.0.1*.



1	Time increment. The default is **Last Changes**. Click the down arrow to select another time increment.
2	The system automatically calculates the minutes, hours, and days since the last successful run. In the example, the last successful run occurred 24 hours ago. The time line is divided into four 6 hour subdivisions.
3	Total number of changes in the selected time increment.
4	The number of changes that occurred between 6 hours ago and now is 4. When you click the change number, the Change History is updated and shows only those changes.
5	Drag the start and end time markers to view specific changes.

Default Settings

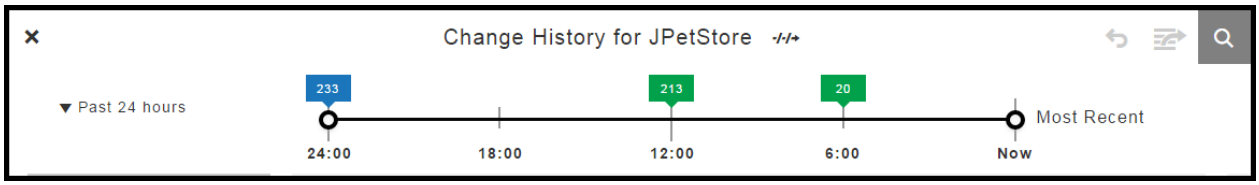
The default time increment is **Last Change**.

The entire time line is displayed, and all the changes are in the list below the time line.

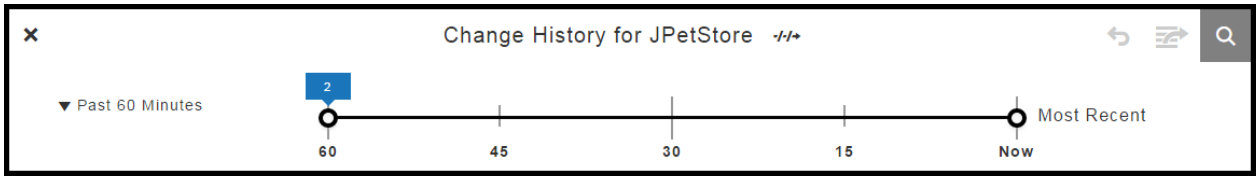
Number of Changes

The time line shows the number of changes throughout the time increment. In the following example:

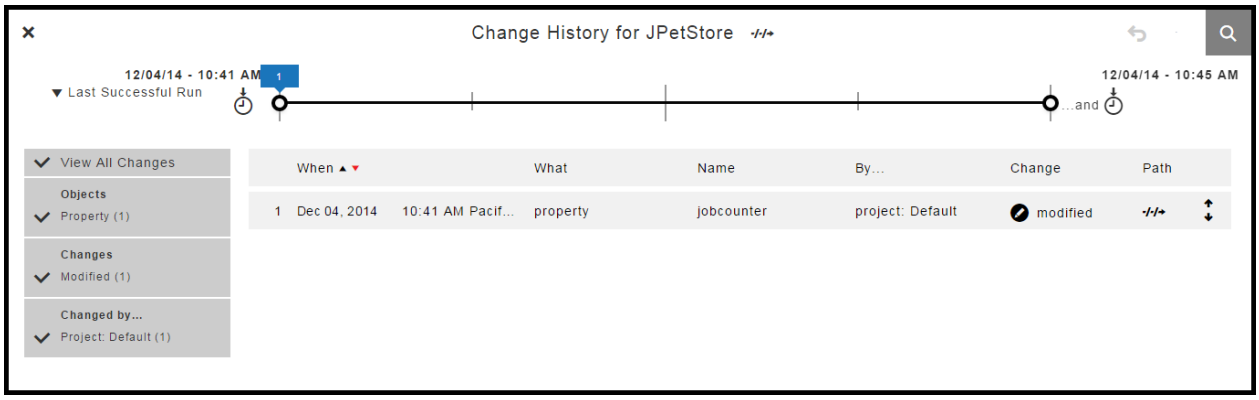
- There have been 233 changes in the last 24 hours.
- There have been 213 changes in the last 12 hours.
- There have been 20 changes in the last 6 hours.



When you change the time increment, there have been two changes in the previous 60 minutes.



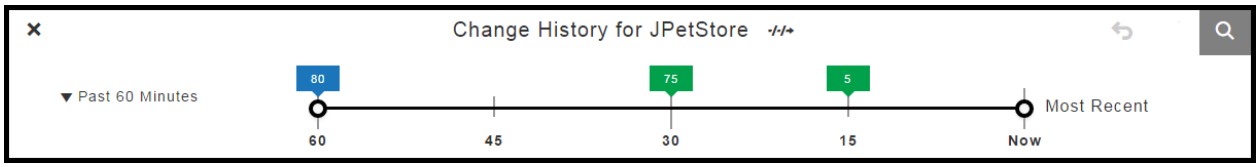
Time Increment



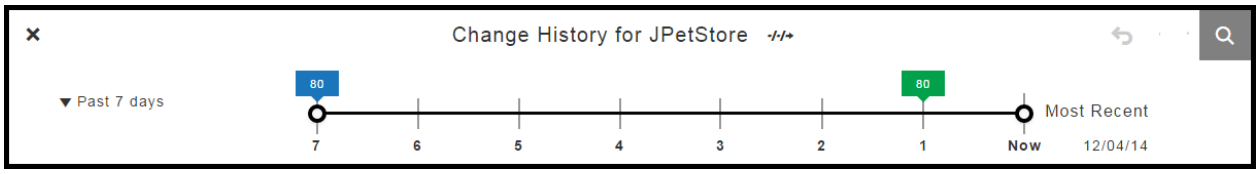
The system automatically determines how the time line is divided for the selected time increment.

When the range is changed to **Past 60 Minutes**, the time line changes:

- The start time is 60 minutes from **Now**.
- The end time is when the **Most Recent** change occurred (**Now**).
- The time line has four divisions.



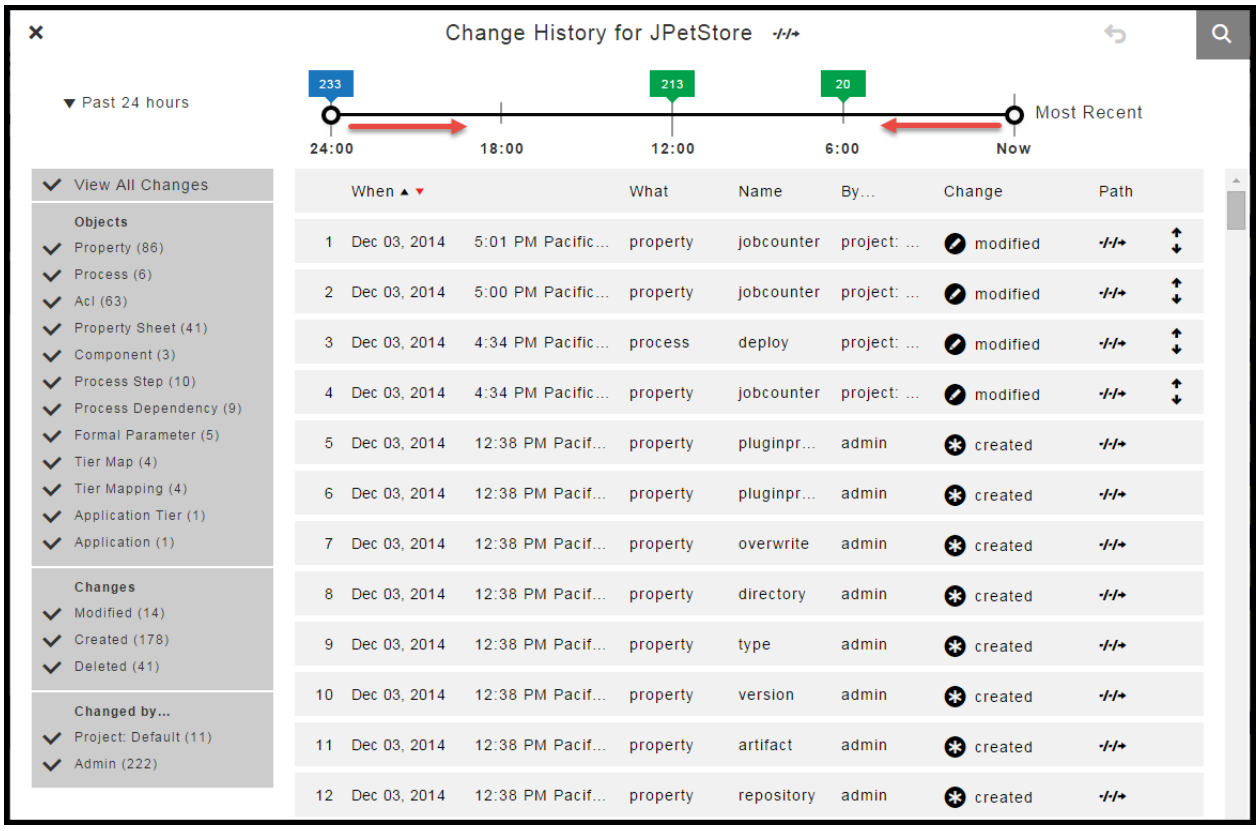
If the increment is **Past 7 Days**, the time line has seven one-day divisions.



Moving the Start and End Times Manually

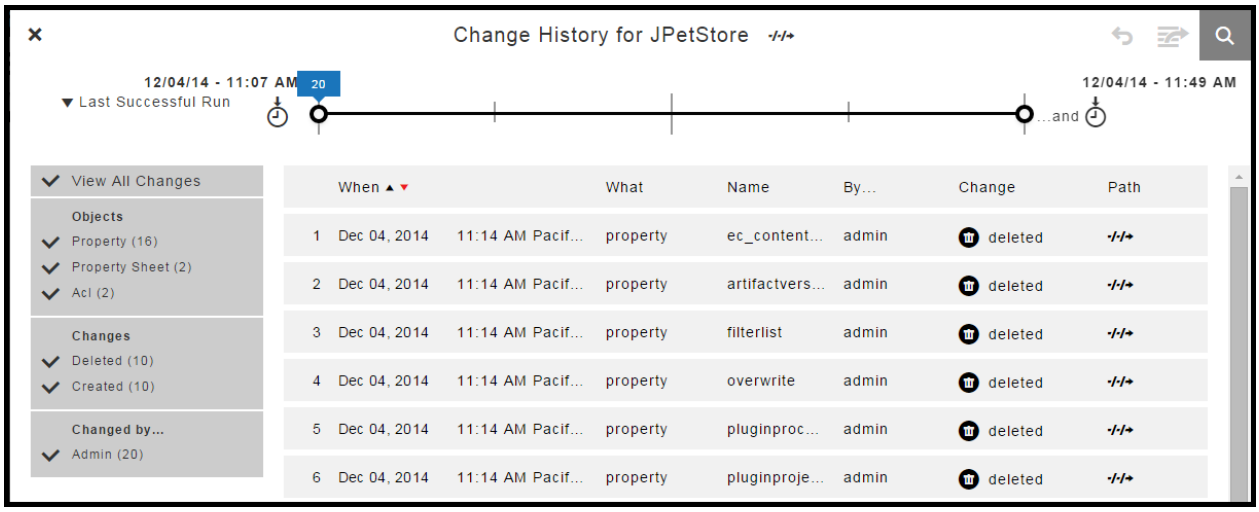
You can manually move the start and end times on the time line.

When you move the start time to 18:00 and the end time to 6:00, the list of objects in the change history changes.



Setting Custom Time Increments

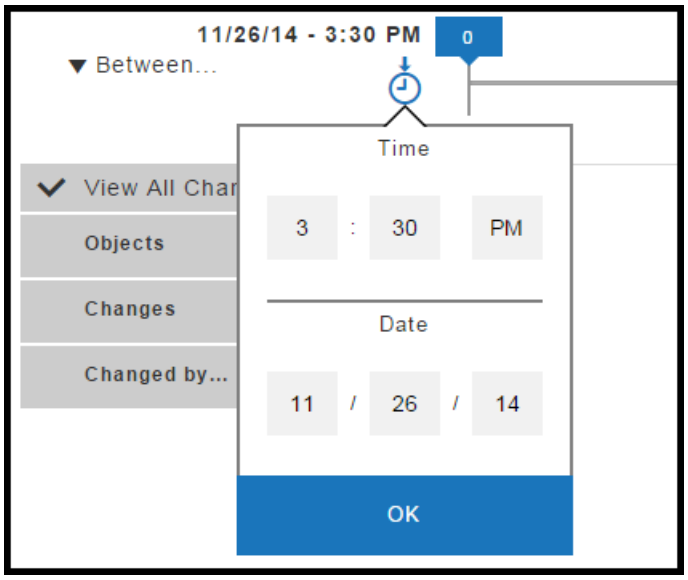
Example:



- Select **Between**.

  A drop down dialog box opens.

  Example:



- Select the time and date for the start of the time line.

  The default settings are **3:30 PM** and eight days before the current date.

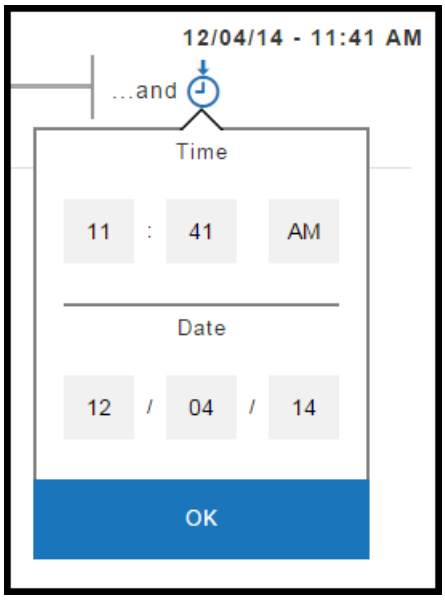
  Example:

  style="max-width: 6.5in;width: auto;height: auto;min-width: auto;min-height: auto;max-height: auto;" />

- Click **OK**.

  A drop down dialog box opens at the other end of the time line.

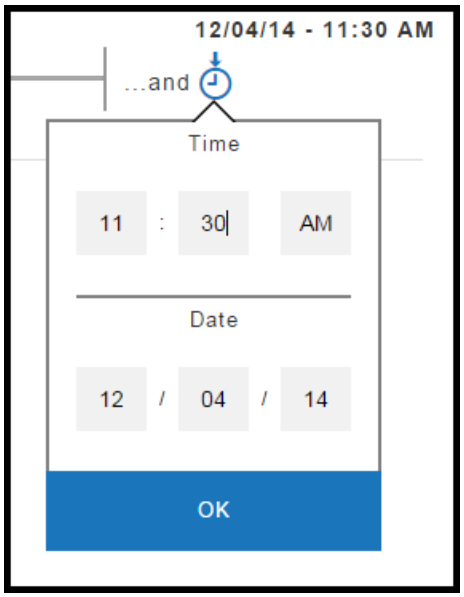
  ### Example:



- Select the time and date for the end of the time line.

  The defaults are **3:30 PM** and the current date.

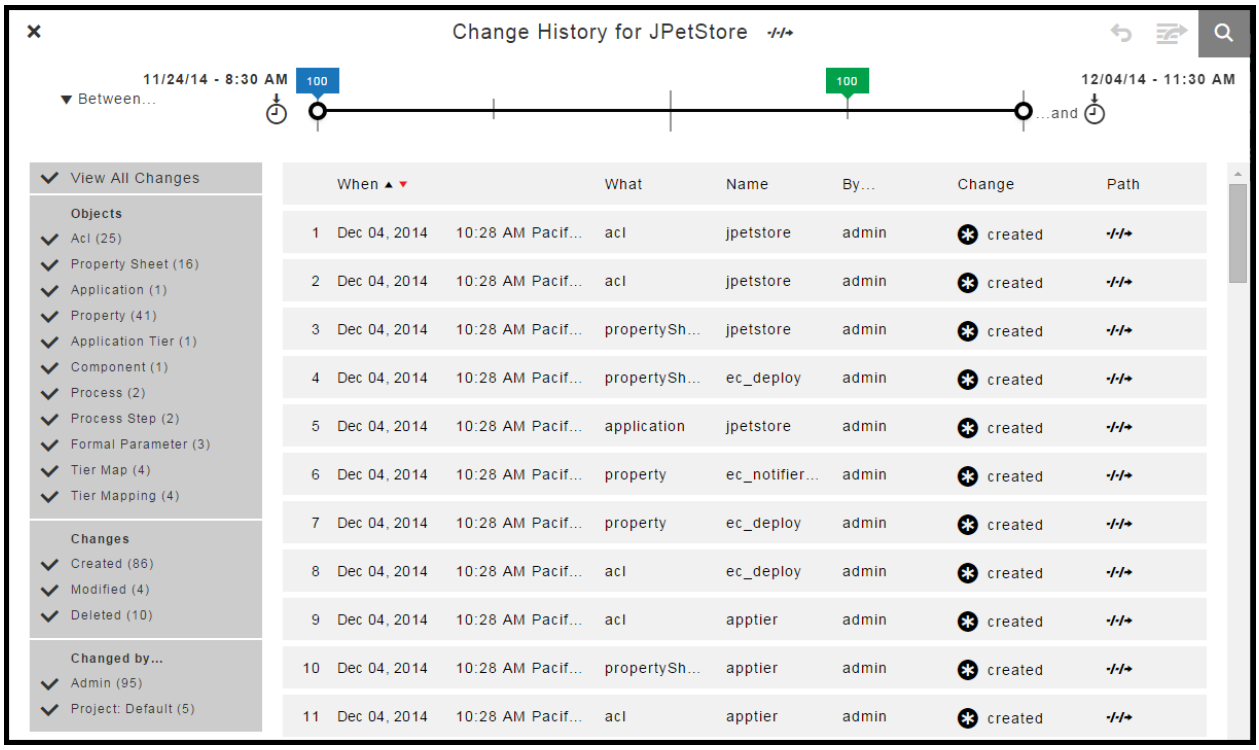
  ### Example:

- Click **OK**.

  The time line changes to show only the changes from the start and end times and dates that you selected.

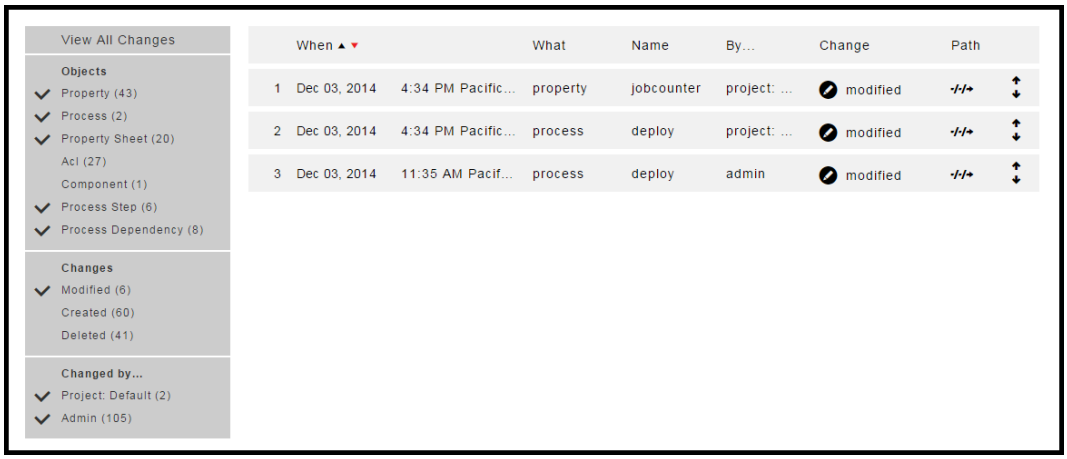
  ## Example:



Change History Filters

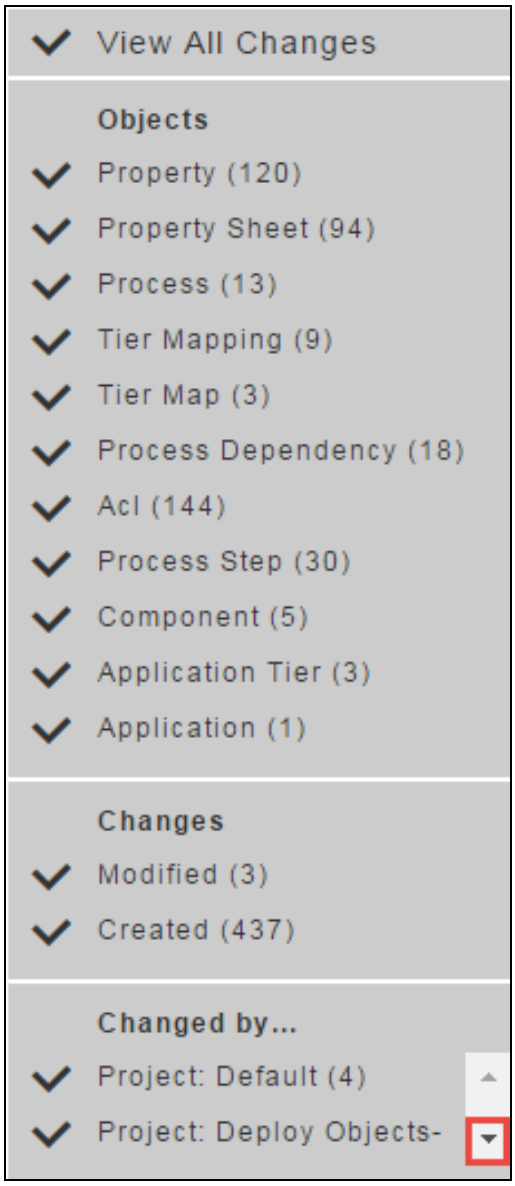
The Change History has records only for an object and its tracked entities, not all entities belonging to the object. You can use filters to view changes to specific objects, the types of changes, and the users how made those changes if they are are tracked.

In this example, instead of selecting **View All Changes**, you can select specific objects, such as only properties, processes, property sheets, process steps, and process dependencies that have been modified by the Project:Default and Admin users.

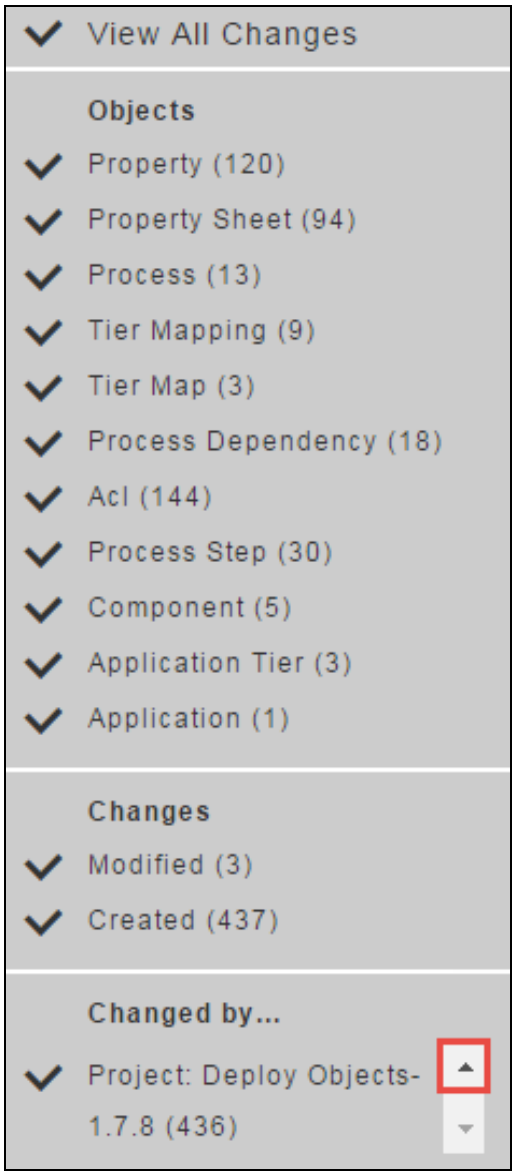


When the list of filter criteria is long, not all of the criteria may appear in the filter list. To see all of the criteria, use the up or down arrows to see all the options.

This list does not show all of the users. Use the up and down arrows to see all four of the users.



Click the down arrow to see the other users.



Searching the Change History

Follow these steps to start a search in the Change History.

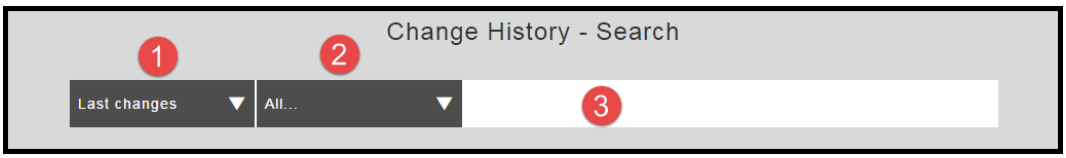
You can start a Change History search from most pages in the ElectricFlow UI.

- Click the **Search** button or click **Change History**.

Example:

The **Change History—Search** dialog box opens.

Example:

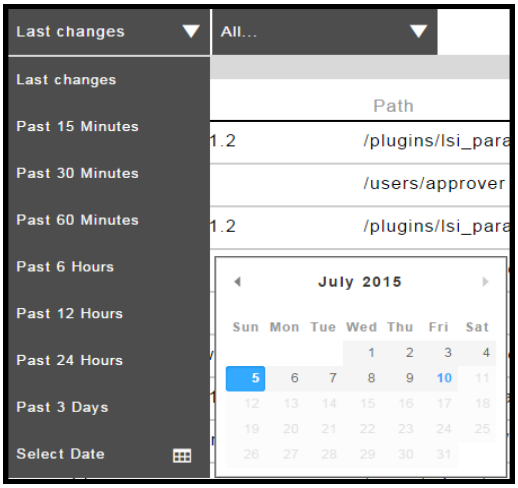


1	**Time range** field. Click the down arrow to open the drop-down list of start times. The end time is the current time.
2	**Objects** field. Click the down arrow to open the drop-down list of objects to include in the search. You can select **All** or specific objects. By default, seven of the most commonly tracked objects are selected.
3	Search criteria. After you type, the system starts searching for objects based on the time range and objects that you selected. The search results are in the Change History.

- To select a time range for the search:
  - Click the down arrow in the **Time range** field to open the drop-down list.
  - Select a time range.
  - If you want to use a time increment longer than three days, do the following:
    1. Click **Select Date**.

       The Date Picker opens.

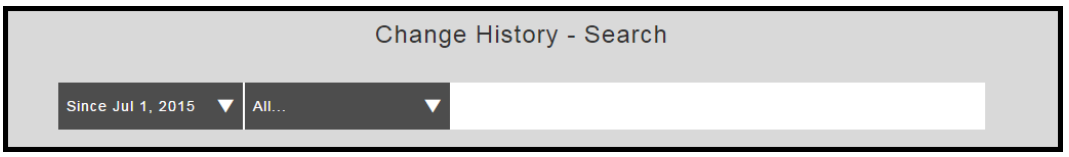
       Example:



    2. Select a date.

       The Date Picker closes and the date that you selected appears in the Time Increment field.

       Example:



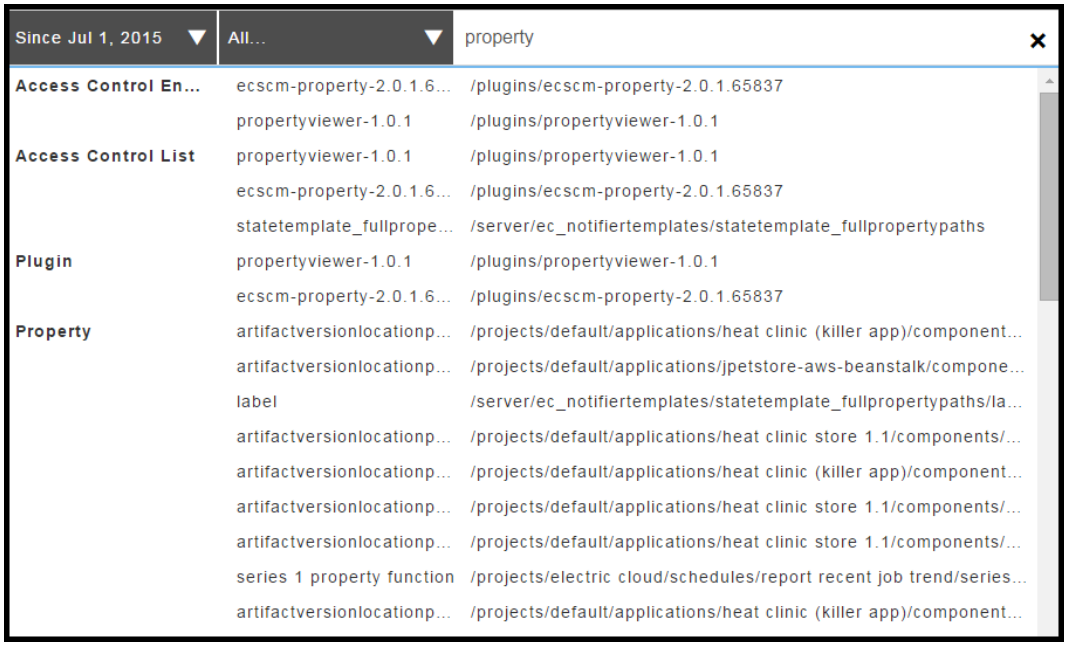
- To select an objects for the search:
  - Click the down arrow in the **Object** field to open the drop-down list.
  - Select the objects for the search.
- Enter the search criteria.

  As you type, the system starts to search for objects that match your search criteria.

  A list of objects matching your search criteria appears in the results section.

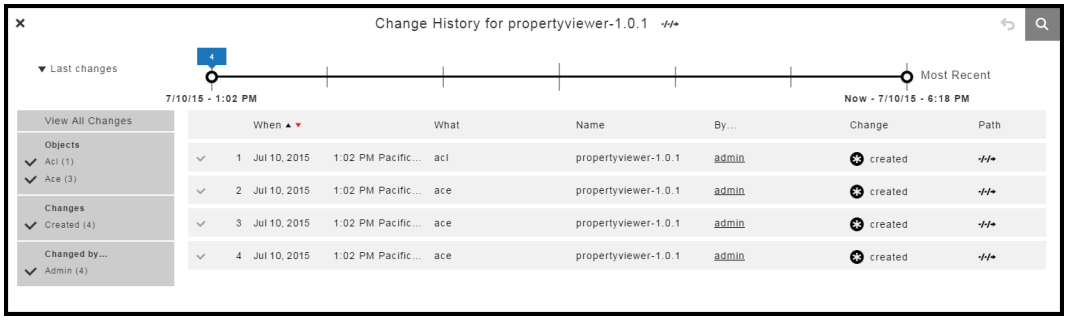
- Select an object in the list.

Example:



The change history for the object that you selected appears.

Example:



Reverting Changes to a Tracked Object and Its Tracked Contents

The Change History displays a list of changes in reverse chronological order (from latest to oldest) for tracked entities in objects such as applications, independent microservices, procedures, workflows, workspaces, resources, and project-owned components such as library components. Each row in the Change History or a tracked object is a record of a change made either to the tracked object or to a tracked object owned by the object (directly or indirectly). For example, the Change History of a project would show both changes to the project itself and changes to any tracked object in the project, such as its procedures, the steps for each procedure, or properties belonging to the procedure steps.

Some types of objects (mostly run-time objects such as jobs, because tracking those would be resource-intensive and seldom useful) are not tracked by the Change Tracking feature. For a few types

of tracked objects, some of their attributes are not tracked (for example, agents are tracked, but certain attributes of agents that vary frequently in real-time, such as their current status, are not).

Changes to untracked objects, or to untracked attributes of tracked objects, are not shown in the Change History and cannot be reverted. In general, objects that are normally created or changed manually are tracked, and objects that are normally created or modified during automated processing are not, However, depending on your specific usage patterns, this behavior may not be recorded in the Change History, such as when you frequently run jobs that use ectool or API commands to automate the creation of new procedures or steps.

> **Note:** You must have the *Read*, *Modify*, and *Execute* permissions to revert changes in the UI or to access to the `revert` API function.

Follow these steps to select a tracked object and its tracked entities that you want to revert:

- Go to the Change History for the object that you want to revert using one of these methods:

  - Click the **Track Changes** button ( )for the object.
  - From the Automation Platform Home page (https://<*ElectricFlow_server*>/commander/),

    click the **Change History**button ( ) to go to the "Change History—Search" page and search for the object.
- Locate the time period in which you are interested.

  A list of changes for the tracked object appears.
- If you decide to revert the changes for an object owned by the tracked object or for an object that owns the tracked object, use one of the methods to view the list of changes for this object:
  - Close the current Change History and navigate to the Change History for the object.

  - Click the **View Path** button ( ) on the top of the current Change History, copy the path, go to the "Change History—Search" page, and search for a suitable addition to or truncation of that path.

- Select a row (record) in the Change History corresponding to the earliest of the changes that you want to undo.

  You will be reverting the object and all its tracked contents to their state directly before the selected change was made.

  1. If the **Revert** button ( ) becomes enabled, ElectricFlow is capable of reverting the selected changes to the object. is tracking the changes to this object.

     Go to Step 5.

  2. If the **Revert** button is still disabled, ElectricFlow is not currently capable of reverting changes to the selected object. This could be due to tracking of this object being disabled and re-enabled, or the Change History feature being disabled and re-enabled. Also, not all types of tracked objects can currently be reverted (and due to mutual consistency requirements in the database format, certain types of objects cannot be reverted).

     If you are unable to revert the selected object, you may still be able to achieve what you need to do one of these ways:

     - Revert the parent object that owns the object you want to revert.
     - Separately revert the individual objects owned by the selected object or even nested partial-reversions-of-reversions, possibly combined with manually reversion (undoing) certain changes.

     Go to Step 3 to search for an for an owned or owning tracked object.

- Before clicking the **Revert** button, make certain that you know all the changes that will be reverted, and have confirmed that this is what you want to do. (While it is possible to re-revert the changes caused by a mistaken reversion operation, doing so leaves a lot of activity in the Change History.)

  Important: All of the changes for tracked entities will be reverted, even the ones that you do not see in the Change History because of the timeline and filter settings.

  1. You may need to modify the timeline and filter settings to display all records in the Change History before you verify that all of the records are changes you want to revert.
  2. Make sure that the end of the timeline is set to **Now** and that the filters are set to **View All Changes**.
  3. Look through the entire list of changes from the most recent to the one you have selected, expanding them to see the details if needed, and confirm that you really want to revert all of them.

  Note:
  If you want to revert some of the changes shown but not others, you may be able to achieve what you want to do with some combination of the following methods:

  - Reverting just certain objects owned by the object that you want to revert.
  - Using nested or overlapping sets of reverts of owned objects.
  - Undoing individual changes manually using the ElectricFlow UI.
  - Using nested partial-reverts-of-reverts.

  If you fully understand the XML format used for ElectricFlow exports, you can use the `export` API call to export the past and current states of a part of the object hierarchy, diff and merge them using a suitable tool for diffing and merging XML files, and then re-import the results. Caution is strongly suggested if you are using this approach to revert objects for which ElectricFlow does not currently support reversion, because:

  - There are some places in the ElectricFlow XML data hierarchy where peer objects refer to each other in ways that need to be kept consistent (such as by index of current position in a list).
  - For some types of objects, re-importing them can change their IDs and break references-by-id.

- Click the **Revert** button. Depending on the size of the object and every tracked object it owns, this operation could take some time (as long as exporting a past state of them and then re-importing it).

  If this operation is successful, the object and its tracked entities are reverted back to the state at the date and time of the record selected in Step 3. A message appears that the Revert operation was successful.

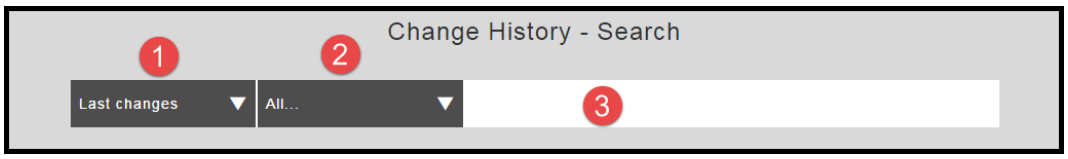
  If this operation is not successful, the object and its tracked entities are not reverted, and a message appears about the failed operation.

Change History Search Form

How to get here: From most pages, click the **Search** button to open the "Change History—Search" form.

Example:

The "Change History—Search" form has the following information:



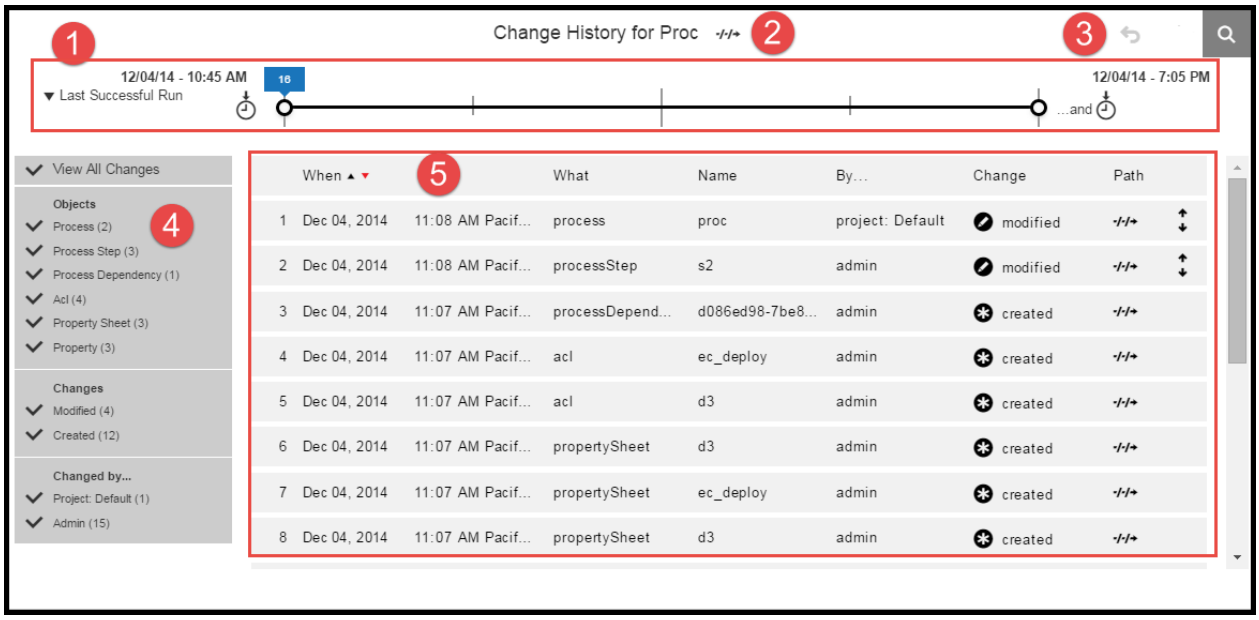
1	**Time range** field. Click the down arrow to open the drop-down list of start times. The end time is the current time.
2	**Objects** field. Click the down arrow to open the drop-down list of objects to include in the search. You can select **All** or specific objects. By default, seven of the most commonly tracked objects are selected.
3	Search criteria. After you type, the system starts searching for objects based on the time range and objects that you selected. The search results are in the Change History.

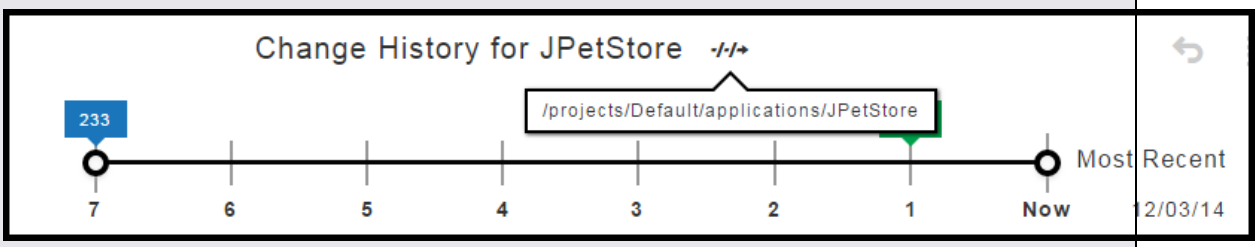
Change History Page

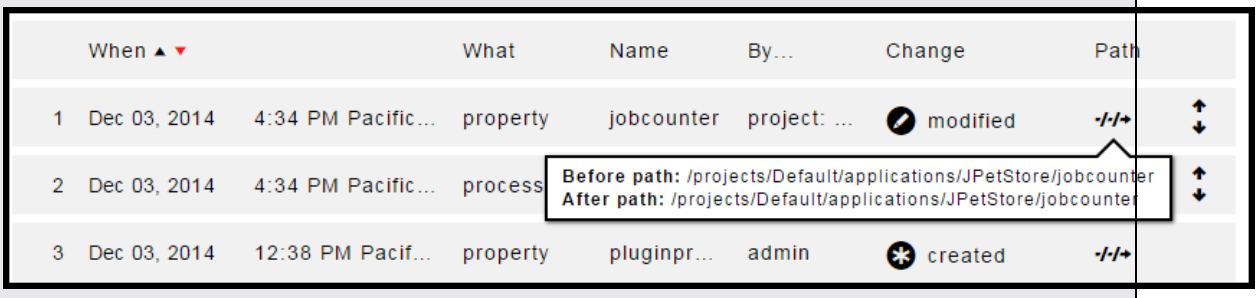
How to get here: Click the **Change History** button for a tracked object.

Example:

The Change History has this information about the object called *Proc*:

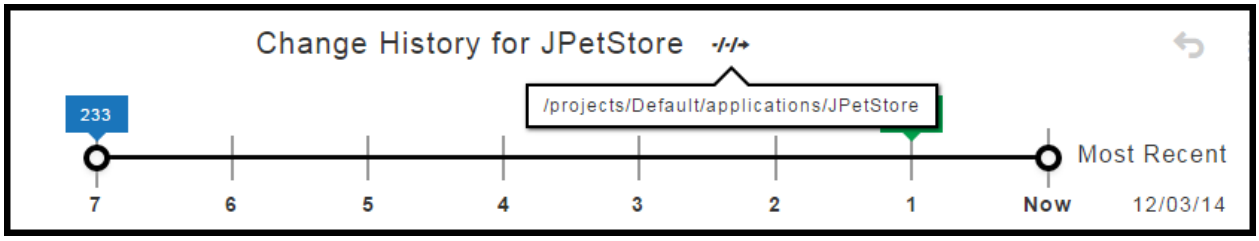


1	Time line You can modify the start and end times. Default: • The entire time line is selected. All changes appear in the Change History list. • The time increment is from the Last Successful Run to the most current change. • The start time is based when the last successful run occurred. • The end time is when the most recent changed occurred.
2	Path to the tracked object. Example: 
3	 Click to revert the selected changes.

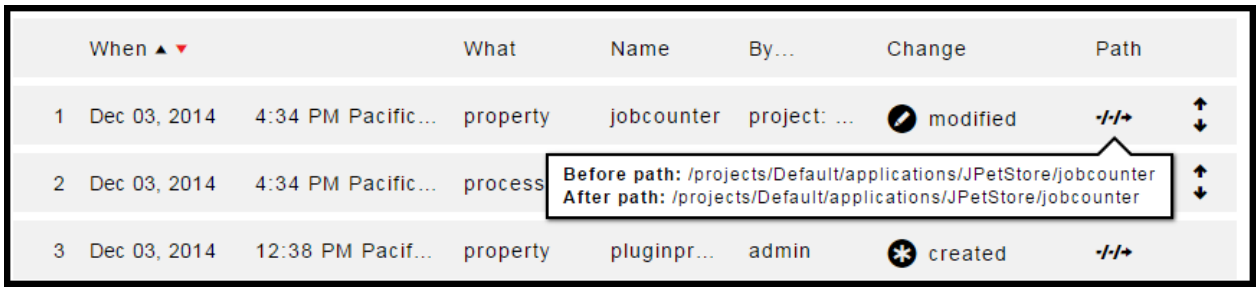
6	Time line. The start time is based on the time range that you selected. The end time is the current time. You can manually change the start and end times after you run the search and get the search results.
4	Filters for the change history. You can view all changes or view only selected changes. The objects in the list are the objects in the change history search results.
5	Change history for the selected object. • When—The date and time that the object changed. • What—The type of object. • Name—The name of the object. • By—The "user" that changed the object, which can be a project or a user. • Change—The type of change. • Path–Click the **View Path** button to see the path to the object.  Click the **View** button to view more information about a specific change in the change history.

Paths to Objects

Click the **View Path** button next to the "Change History for JPetStore" title to see the path to the application.

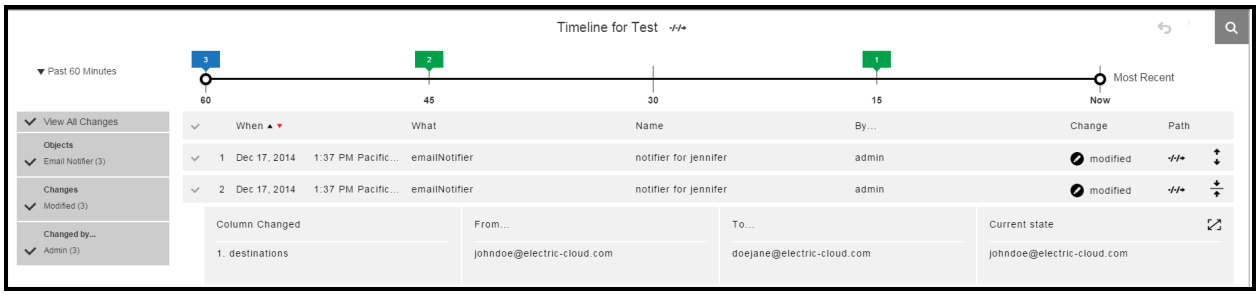


Click the **View Path** button to see the change in the path to the object before and after the change.



Detailed Object Changes

Click the **View** button to see the change in the property called emailNotifier.

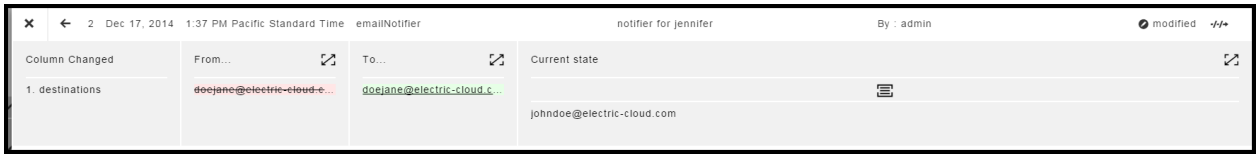


Click the **Expand** button to all the changes to the property,

Example:

When you click the **Expand** button in a cell, you can see more details in the current change in the cell.

If you click the **Select All** button, all the changes about the object appear.



To select an object to revert or import the changes to an XML file, select the row of the object in the expanded view.

Defect Tracking

The Defect Tracking plugin enables linking existing defects to an ElectricFlow job. "Existing defects" are those defects previously created in the defect tracking system and already associated with the ElectricFlow job in some way.

For example, a defect is associated with a ElectricFlow job if the fix for the defect is part of the source code snapshot being built and tested in the job.

ElectricFlow pre-installs numerous defect tracking plugins including Bugzilla, ClearQuest, Fortress, JIRA, Quality Center, Ration Team Concert, Rally, Team Foundation Server, TeamForge, TestTrack, and more, using plugin integrations.

The following examples and instructions are for a JIRA integration. The steps for other defect tracking systems are similar to those for JIRA.

Scenario Example

A developer fixes a defect in the "ABC" project and checks in the fix to the source control system, along with a comment that includes the fixed defect ID, for example:

```
ABC-123: fixed EFG bug
```

When the next ElectricFlow job is triggered for the ABC project, ElectricFlow checks out a source code snapshot from the source control system and queries the source control system for a log containing check-in details. This log, which should contain the above comment, will be stored in a property on the job.

The JIRA plugin will then do the following:

- Parse the property containing the source control system log to identify defect IDs.
- Query the configured JIRA server with identified defect IDs, including "ABC-123".
- Construct a descriptive URL to point to the JIRA defect on the JIRA server.
- Include the URL in the JIRA Report.
- Link to this JIRA Report from the Job Details page.

Enabling the JIRA Integration in Your Procedure

To ensure ElectricFlow links existing defects to a job, create a step to link the defects.

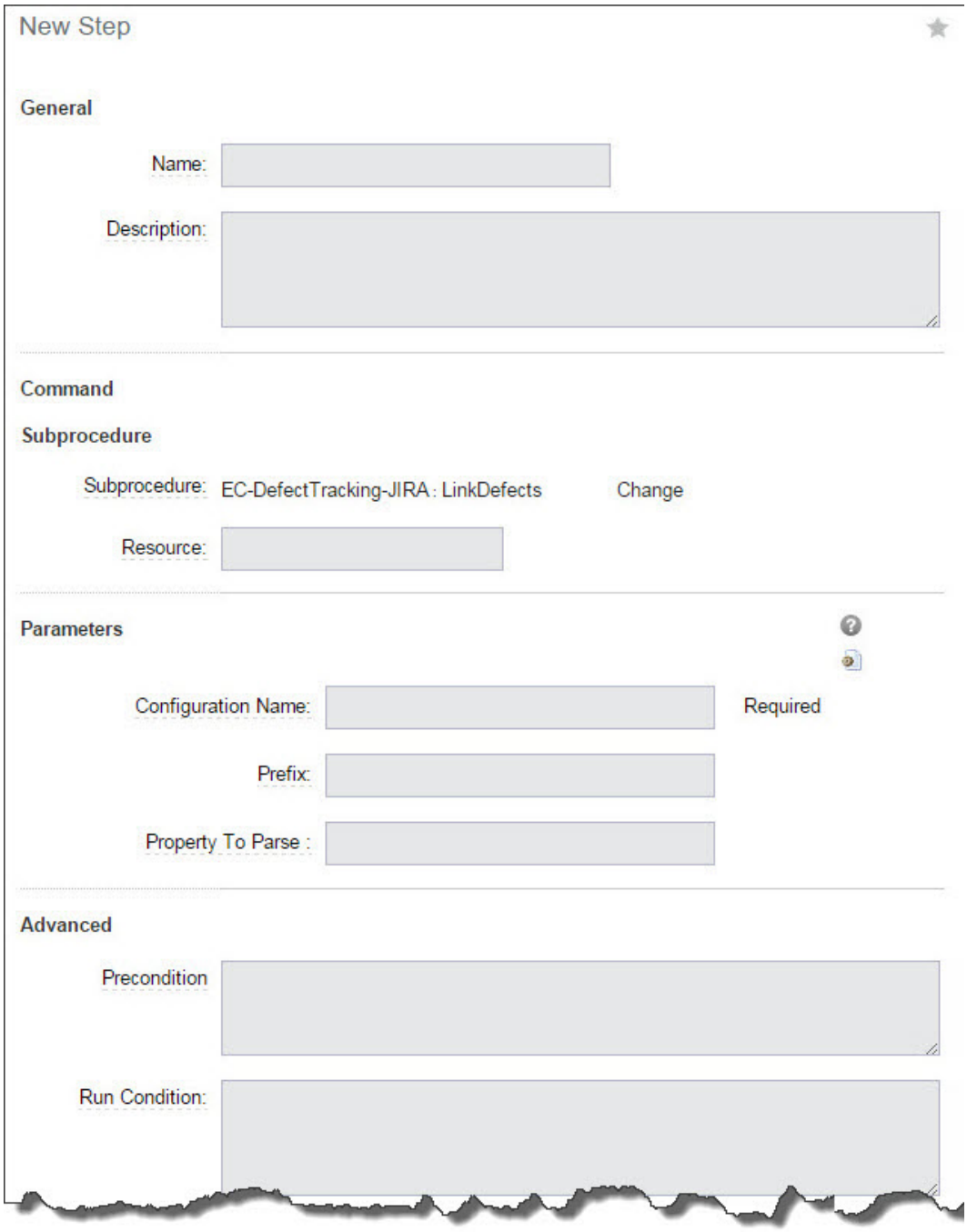
Go to Projects > select a Project > select a Procedure. To create a New Step, select the **Plugin** link.

- In the Choose Step panel, select Defect Tracking from the left pane, then select the defect tracking system you configured.
- The right pane now shows the types of steps available for your configuration. Select the step you need and automatically go to the New Step page.
- On the New Step page, notice the Subprocedure section now contains the defect tracking integration you configured and the step you chose.

On the New Step page, enter information in the following fields:

Field or Option	Description
Name	Unique name for your subprocedure step (any name of your choice).
Description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.
Subprocedure	(Optional) Name of a procedure to invoke during this step.
Resource	(Optional) Resource to run the procedure.
Configuration Name	Name of the Defect Tracking configuration you created on the New Defect Tracking Configuration page. **Note:** If you did not create a Defect Tracking configuration, you must do that now before you proceed. Click **Administration** > **Defect Tracking** > **Create Configuration** to see the New Defect Tracking Configuration page. You can click the **Help** link on that page for details.
Prefix	Key used by JIRA as the prefix for defects within a project. If this field is blank, a regular expression is used to match defect IDs.
Property To Parse	Property or property sheet to search for defect IDs. If the this field is blank, the default property `/myJob/ecscm_changeLogs` is used.
Precondition	(Optional) A fixed text or text embedding property references that is evaluated into a logical `TRUE` or `FALSE`. An empty string, a `\"0\"`, or a `\"false\"` is interpreted as `FALSE`. Any other result string is interpreted as `TRUE`. The step will block until the precondition is `TRUE`.
Run Condition	(Optional) A fixed text or text embedding property references that is evaluated into a logical `TRUE` or `FALSE`. An empty string, a `\"0\"`, or a `\"false\"` is interpreted as `FALSE`. Any other result string is interpreted as `TRUE`. If the run condition is false once the step meets its precondition, the step is skipped.

The following screen is an example of the New Step page where you will create your Defect Tracking subprocedure step.



ElectricSentry

ElectricSentry is the ElectricFlow engine for continuous integration, integrating with Source Control Management (SCM) systems. ElectricSentry is installed automatically with ElectricFlow and is contained in an ElectricFlow plugin named ECSCM. When ElectricSentry detects a new or modified source file check-in, it launches a user-defined procedure to build and test the latest version of source files. Using a few simple properties, the entire process is controlled by the user who determines when ElectricSentry is active and which projects and branches are monitored and built.

While you can interact with ElectricSentry directly, you may want to use the Continuous Integration Dashboard, *the front-end user interface* for ElectricSentry, which provides a visual display of your builds, and automates the configuration process for your branch, project, or procedure so you can start using continuous integration schedules quickly.

This Help topic describes using ElectricSentry directly, and *not* the Continuous Integration Dashboard, which has its own Help topic for dashboard use. To get started with the "dashboard", select the **Home > Continuous Integration** tab.

How ElectricSentry works

ElectricSentry finds procedures the user wants to monitor by scanning for a special type of ElectricFlow schedule. From the schedule, ElectricSentry retrieves the name of the procedure to run, along with values to use as actual parameters when the procedure runs.

ElectricSentry can run at a regular interval you determine. When it runs, ElectricSentry scans all ElectricFlow projects, looking for schedules configured for use with ElectricSentry. When it finds one, ElectricSentry checks to see if there is already a job running based on that schedule and will not start a second job for the same schedule until the previous job completes.

Next, ElectricSentry queries the SCM system to see if new check-ins have occurred since the last attempted build. If the most recent check-in is different from the last attempted build, ElectricSentry then checks a "quiet period" by comparing the current time to the check-in time. If the user-configured time interval has elapsed since the most recent check-in, ElectricSentry runs the procedure to start a new job.

Finally, ElectricSentry deletes previous runs of itself, so only the most recent run appears in the jobs list. If using the Continuous Integration Dashboard, this is slightly different—you will see the previous 10 builds that ran.

Configuring ElectricSentry

ElectricSentry does not require any specific configuration, but it does allow fine tuning for some key settings. To change any setting, you must be logged into ElectricFlow and then navigate to the Electric Cloud project.

Quiet time

When setting up a continuous integration system, it is common to require an inactivity period before starting a build. This time period allows developers to make multiple, coordinated check-ins to ensure a build does not start with only some of the changes—assuming all changes are checked-in within the specified inactivity time period. This time period also gives developers an opportunity to "back-out" a change if they realize it is not correct.

With ElectricSentry, the inactivity time period can be configured globally for all projects or individually for a single project. The global setup is stored in the ECSCM-SentryMonitor schedule in the Electric Cloud project, using a property named ElectricSentrySettings/QuietTimeMinutes. You can set this property to the number of minutes you prefer for the quiet time prior to starting a build. If you remove this property, ElectricSentry will use a five-minute default time.

Resource

The ECSCM plugin contains a procedure named ElectricSentry. This procedure has a parameter named `sentryResource` that determines where ElectricSentry will run. The parameter defaults to a resource named *local*, which is created by the ElectricFlow installer. The local resource is configured to run on the machine named *localhost*, which refers to the machine where the ElectricFlow server is running.

You may need to change the resource where ElectricSentry runs, if for example, the server machine is not configured to run your SCM tools. You can change this configuration by modifying the ECSCM-SentryMonitor schedule to specify a different resource in the `sentryResource` parameter.

To do this:

1. Go to the Projects page.
2. Select the Electric Cloud project name to go to the Project Details page.
3. Click the Schedules subtab.
4. Select the ECSCM-SentryMonitor schedule.
5. Change the sentryResource field to the name of the resource (or the resource pool name) where you want ElectricSentry to run.
6. Click **OK** to save your new information.

Polling frequency

ElectricSentry is set to look for new check-ins every five minutes. If you want ElectricSentry to check more or less often, open the ECSCM-SentryMonitor schedule in the Electric Cloud project. Change the number of minutes to speed up or slow down the ElectricSentry polling frequency entry.

Time-of-day and day-of-week

The initial setting allows ElectricSentry to run between 7 am and 11 pm every day of the week, but you can change this time period to monitor check-ins at different hours. The time period should cover the most likely hours developers would be checking-in changes, then frees the resources (after the last job started by ElectricSentry finishes) to run overnight builds. You can modify the ECSCM-SentryMonitor schedule to change the hours or days ElectricSentry monitors check-ins.

Configuring a build for continuous integration

You can control which projects and branches ElectricSentry monitors for continuous integration by creating a schedule that runs a procedure and by adding ElectricSentry settings. These tasks are made easier using the Continuous Integration Dashboard.

Source Control Management SCM) configurations

ElectricFlow bundles and supports a number of source control types. After creating a source control configuration, your entry will appear in the table on the Source Control Configurations web page—to see this web page, select the Administration > Source Control tabs.

Important: Click the **Create Configuration** link to configure your source control system.

Note: Electric Cloud has tested numerous SCM versions and where possible, all SCM integrations were created in a generic manner to avoid SCM release-specific differences. In most cases, Electric Cloud supports any version of Perforce, ClearCase, Subversion, AccuRev, CM Synergy, Borland's StarTeam, and many others.

Select the **Administration > Plugins** tab to go to the Plugin Manager page to the current list of available SCM plugins. If you do not see the SCM you need, it may be available in the Plugin Catalog—select the View Catalog tab.

Create a procedure

Navigate to your project and create a procedure that checks out sources from your SCM system and then runs your build process. we recommend having the procedure take a ”branch name” parameter, then use this single procedure to build multiple branches of the same project.

Create a schedule

In your project, create a schedule that calls the procedure and specifies parameters to call the procedure. On the Project Details web page, select the Schedules tab and click the **CI Configuration** link to go to the New CI Configuration page. Enter a name for the new schedule and choose the SCM Configuration you want to use. For more information on this page, click the **Help** link in the upper-right corner of the screen. If you set up your procedure with a parameter for a branch, create one schedule for each branch you want to monitor.

Optional —running ElectricSentry on multiple resources

ElectricSentry normally runs as a single procedure executing queries against all SCM systems configured by ElectricSentry "trigger schedules." Because all queries are executed in a single step, these queries are executed by the same ElectricFlow resource. This process works very well for many companies, but for some large enterprise companies this process is not adequate.

You can benefit from using ElectricSentry on multiple resources if any of the following items apply to your organization:

- Connectivity—No one single resource has access to all SCM systems in your organization.
- Security/Authentication —Specific resources may contain Authentication tickets required for access to certain parts of your SCM depots, and no single resource contains all of these tickets.
- Scalability—You may need too many trigger schedules to monitor in a single execution interval, which can cause unnecessarily long execution times for ElectricSentry jobs.
- Scheduling—ElectricSentry is most useful during the work day when developers are adding new code check-ins. At night, it is common to redeploy resources to do nightly builds. If ElectricSentry runs on a central resource that serves development teams in multiple time zones, there is no way to schedule independent Sentry operation for different time zones.

Overview

The solution to all of these issues [above] is to run multiple ElectricSentry instances concurrently.

- Each Sentry instance can choose its own resource and its own operation schedule.
- Each instance can be restricted to monitoring schedules for one or more ElectricFlow projects.
- Also, you can create a global instance to monitor all projects not monitored by a restricted instance.

Each ElectricSentry instance is implemented by reusing the ElectricSentry procedure in the ECSCM plugin. This procedure has two parameters, `sentryResource` and `projectList`, that allow you to specify the configuration for different instances.

Each instance will have its own copy of the SentryMonitor schedule—one for each resource where you want ElectricSentry to run.

The default configuration has no projects specified in the "projectList" parameter—this is called the *global* instance. The global instance monitors all projects not explicitly monitored by any other instance. If a global instance is your only instance, it will monitor all projects.

Configuring ElectricSentry to use multiple resources

When ElectricFlow is installed, it creates a project called Electric Cloud that contains ElectricSentry's Sentry Monitor schedules. By default, all steps will run on the resource named local and all projects will be monitored. To use multiple resources, you need to create new schedules in the Electric Cloud project, creating new ElectricSentry instances.

1. Navigate to the Electric Cloud project and select it.
2. Select the Schedules tab.
3. Disable the schedule named ECSCM-SentryMonitor.
4. Click the **Copy** link next to the ECSCM-SentryMonitor schedule—***repeat*** for each new ElectricSentry instance you want to create.
5. Re-enable the schedule named ECSCM-SentryMonitor.
6. Navigate to one of the new schedules you created and select it to edit it—change the name from ECSCM-SentryMonitor copy n to something more meaningful, for example, "ECSCM-SentryMonitor for ABC team."
7. In the Parameters section:
   projectList box—enter the project names you need to monitor—one project name per line.
   sentryResource box—enter the resource or pool name where you want this ElectricSentry instance to run.
8. In the Frequency section:
   Adjust the "time of day" and "days of the week" settings to schedule this ElectricSentry instance to run.
9. In the Advanced section, click the Enabled check box.
10. Click the **Save** button.
11. Repeat steps 6-10 for each new schedule that you create.
12. In the Advanced section, click the Enabled check box. Click the **Save** button.

> **Note:** If you are setting up ElectricSentry to run in a different time zone, make sure you set the Time Zone field for the location where ElectricSentry will run.

The Job Step Execution Environment

Terms and definitions

Various factors influence job step execution. Both a job step and its postprocessor (if any) run in the same environment, except shell. The shell used to run a command comes from what is specified in the step. If a shell is not specified on the step, the shell specified on the resource is used— this is also the same shell used for running the postprocessor.

Machine

The machine where a job step is executed is determined by the resource specified in the corresponding procedure step. If a pool is specified in the procedure step, ElectricFlow picks a specific resource from that pool. A job step can determine its actual resource by querying the property `/myResource/resourceName` or assigned `resourceName` property on the job step. The host where a step is expected is `/myResource/hostName`. If the resource is a proxy agent, this property contains the name of the proxy target. The agent host name is in `/myResource/proxyHostName`.

OS-level access control

For its resource, a job step executes under the same account as the ElectricFlow agent. You may want to contact Electric Cloud technical support for help configuring ElectricFlow agents. Basically, the agent needs to know what account it will run as, and Windows agents require additional setup if impersonation is used. If using impersonation, the job step runs under the credential effectively attached to the step.

Environment variables

For its resource, a job step inherits environment variables from the ElectricFlow agent. The agent's environment variables can be configured as part of the agent configuration. In particular, the PATH environment variable typically includes the ElectricFlow installation directory for easy access to applications such as *ectool* and *postp* (the ElectricFlow postprocessor).

Working directory

The default job step working directory is the top-level directory in the job's workspace. However, you can configure the working directory with a property on the procedure step. When you run on a proxy agent, the step actually runs on the proxy target, in the working directory specified in the step. If the working directory is not specified, the step runs in the UNIX path to the workspace.

Standard I/O

Standard job step output and errors are both redirected to the step log file.

ElectricFlow access

For easy access to ElectricFlow, the following environment variables are set automatically for a job step:

COMMANDER_HOME
A variable whose value is the base installation directory for an agent. On Windows, this directory is typically `C:\Program Files\Electric Cloud\ElectricCommander`. On UNIX platform, this directory is typically `/opt/electriccloud/electriccommander`.

COMMANDER_SERVER
IP address for the ElectricFlow server machine.

COMMANDER_PORT
Port number for normal communication with the ElectricFlow server.

COMMANDER_PLUGINS
(configurable) The directory where installed plugins live (for example, `C:\Documents and Settings\All Users\Application Data\Electric Cloud\ElectricCommander\Plugins`)

COMMANDER_HTTPS_PORT
Port number for secure communication with the ElectricFlow server.

COMMANDER_JOBID
Unique identifier for the job containing the current job step.

COMMANDER_JOBSTEPID
Unique identifier for this job step.

COMMANDER_SESSIONID
ElectricFlow session identifier for the current job, which allows ectool to access ElectricFlow with job associated privileges, without an additional login.

COMMANDER_WORKSPACE_UNIX
Absolute path to the workspace directory for this job, in a form suitable for use on UNIX machines.

COMMANDER_WORKSPACE_WINDRIVE
Absolute path to the workspace directory for this job, in a form suitable for use on Windows, and starting with a drive letter.

COMMANDER_WORKSPACE_WINUNC
Absolute path to the workspace directory for this job, in a form suitable for use on Windows,specified using UNC notation.

COMMANDER_WORKSPACE
Absolute path suitable for accessing the top-level workspace directory for this job on this machine. On a UNIX machine, this has the same value as COMMANDER_WORKSPACE_UNIX. For Windows, it is the same as COMMANDER_WORKSPACE_WINUNC.

COMMANDER_WORKSPACE_NAME
This is the name of the workspace on the ElectricFlow server.

These environment variables allow you to invoke ectool without specifying a `--server argument`. They also provide a context for accessing properties in ectool. For example, "`ectool getProperty foo`" looks for the property named "foo" on the current job step. Environment variables provide ectool with a session including all user privileges associated with the job.

Also, if the resource is a proxy agent, these environment variables are available in the step's environment on the proxy target.

Preflight Builds

Preflight Build Solution

Preflights with ElectricFlow

Samples

Why use Preflight Builds?

When developers make changes, they generally build and test their code locally and *commit* their changes if the code ran successfully. Unfortunately, this process is not sufficient.

Code changes may break the production build because of environment differences, platform-specific issues, or incomplete commits. These issues leave the product in a broken state until the changes are backed-out of the product or the developer commits a fix, which often affects the entire development team. It is a time-consuming process to find the problem when multiple developers commit their changes simultaneously and the build breaks.

Preflight Build Solution

A **Preflight Build** is used to build and test a developer's changes before those changes are committed. A "post-commit" source tree is simulated by creating a clean source snapshot and overlaying the developer's changes on top of it. These sources are then passed through the production build procedure to validate the changes work successfully.

Developers are allowed to commit their changes only if the preflight build is successful. Because developer changes are built and tested in isolation, many common reasons for broken production builds are eliminated.

Preflights with ElectricFlow

Workflow

After a preflight build is configured, the general flow of interaction between the server, agent, and client is:

- A developer invokes the client preflight program either through an IDE or via the command-line. The following arguments are specified in the XML configuration file or on the command-line:
  - Define which ElectricFlow server to use (hostname, ports, and so on),
  - Define the project, procedure, and parameters to use to start the preflight build, and
  - SCM-specific information to use (ports, changelists, and so on).
- Next, the client preflight program connects to the ElectricFlow server and launches the specified procedure.
  - The job is started and the developer's changes are uploaded.
  - If auto-commit was turned on, the client program waits for the job to complete. Otherwise, the client program exits immediately.
- The job progresses until it gets to the special step added to the procedure to extract sources and overlay changes.
  - The agent-side driver creates a clean source snapshot based on configuration information passed to it from the client-side.
  - The step overlays changes [uploaded by the client] on top of the snapshot to simulate the developer's check-in.
- The build continues as usual, using the modified sources.

- If the developer chose to auto-commit changes, the preflight program was waiting for the job to complete. If the job is successful, the changes are committed if:
  - No files were added or removed from the change sets.
  - No files were modified since the preflight build started.
  - All conditions set in the driver script were met.

Components

Three components are required for an ElectricFlow preflight build:

- **Server**—This is the main ElectricFlow server that will run the preflight builds.
- **Agent**—This is any ElectricFlow agent machine that is setup as a resource for the server. This machine must be able to communicate with the source control system because it is responsible for creating a clean source snapshot on which the developer's changes are overlayed.
- **Client**—This is the machine where a developer has active code changes to submit for a preflight build. Like the agent, this machine must be able to communicate with the source control system because it is responsible for determining which files were modified so it can transmit these files to the agent.

Installation

Preflight is automatically installed with SCM plugins. ElectricFlow pre-installs numerous source code management (SCM) plugins including Accurev, Bazaar, CVS, ClearCase, Git, Mercurial, Perforce, SVN, StarTeam, Team Foundation Server, and Vault. Before you can use your preferred SCM, you will need it to communicate with the ElectricFlow server.

Configuration

This section provides details for setting up and running Preflight builds in a typical build environment.

Server

Adding the preflight snapshot step

A typical production build procedure has a step that extracts a clean source code snapshot before the actual build starts. For preflight builds, a special step needs to run in place of this snapshot step. This special step is responsible for creating the base snapshot and then overlaying the developer's changes.

- On the Procedure Details page, to create a new step click the Plugin link.
- On the Choose Step panel, select Source Code Management.
- Choose the ECSCM plugin for your SCM or from the right-pane, select the "Preflight" step for your SCM.
- On the New Step page, notice that your SCM is displayed in the Subprocedure section.
  - Set the Resource name for the step to an agent that can communicate with the SCM system, so it can create the source snapshot.
- The Parameter section also displays appropriate fields to enter information for your SCM. The following two fields are common to most SCMs:
  - Configuration–Enter the name you created for your SCM configuration.

- Destination Directory–This is a path relative to the job's workspace, where the source tree will be created.
- Enter all other information required for your new step.
- Click **OK**.

Choosing between snapshot steps

Only one snapshot step should run depending on whether a production or preflight build is invoked. To enable the appropriate step, add a checkbox-style parameter to the build procedure to determine whether or not the build is a preflight. Now, use that parameter in the Run Condition field on the two snapshot steps. In the following example, the parameter is named "preflight", with an unchecked value of "false" and a checked value of "true".

- For the preflight snapshot step, set the run condition to `$[preflight]`
- For the production snapshot step, set the run condition to `$[/javascript getProperty("preflight") == "false"]`

Choosing how files are uploaded

By default, when developers run preflight builds, their changes are uploaded to the job workspace via the server. However, if the job workspace is on a network share, accessible to the developer's machine, a useful optimization is for files to be copied directly into the job workspace. To use the network share approach, set a flag on the top-level procedure that is invoked when running a preflight build. This property can be created on the Procedure Details web page as follows:

1. Create a nested property sheet on the procedure called `ec_preflight`.
2. Create a property in that procedure called `waitForStep` with a value of "1".
   This property can be created from the command-line by calling:

   ```
   ectool setProperty ec_preflight/waitForStep 1 --projectName <project> --
   procedureName <procedure>
   ```

Agent

The agent machine must be able to communicate with the SCM system to create the source snapshot.

Client

From a developer's machine, there are two ways to start preflight builds:

- From an IDE:
  - Eclipse
    Open a Run dialogue, select Launch Commander Procedure, then select an existing preflight configuration or create a new one.
    For more information, see the *ElectricCommander Eclipse Integration Tech Note* PDF file or the Eclipse plugin.
  - Visual Studio (See the *ElectricCommander Visual Studio Tech Note* PDF file or the Visual Studio plugin)

- From the command line:

  The executable program, `ecclientpreflight`, is used to run a preflight build from the command-line. This program is included in the ElectricFlow "Tools" installation. When you run ecclientpreflight, configuration options can be passed in an XML configuration file or on the command-line.

  If the same option is specified in both the configuration file and on the command-line, the value passed on the command-line takes precedence. The XML configuration file can be passed to the program via the `--config` option. If the XML configuration file is not passed in, current and ancestor directories are searched for a file named "`.preflight`", using the first one it finds as the configuration file.

  Generally, a "`.preflight`" file is stored at the root of a developer's workspace, containing configuration information specific to that workspace.

Samples

In the following samples, the SCM was set to Perforce. Some options may be specified using an abbreviated format (`-c`, with a single dash) or a more verbose format (`--config`, with two dashes).

General Options	
`-c,--config <file>`	Load configuration options from the specified XML file. If `<file>` is `'-'`, read standard input. If not supplied, the current and ancestor directories will be searched for a .preflight file. The first .preflight found will be used as the preflight configuration.
`--load <file>`	Evaluate `<file>` after option processing. May appear multiple times.
`-d,--debug`	Display debugging information.
`-v,--version`	Display version information.
`-h,--help`	Display this information.
Server Communication Options	
`--server <name>`	The host name (defaults to localhost).
`--port <port>`	The HTTP port (defaults to 8000).
`--securePort <port>`	The HTTPS port (defaults to 8443).
`--secure`	If set, only secure connections are used. Off, by default.
`--timeout <timeout>`	The response timeout in seconds (defaults to 180).
`--userName <name>`	The name of the ElectricFlow user to login as.

--password <*password*>	The password for the specified user. If blank and the user has no active session, a prompt to enter the password will be provided.
--driverLocation	Property path to a property sheet containing the driver scripts that will be loaded and run after a connection to the server is established. Defaults to /server/ec_preflight.
--mainDriver	The name of a property in the sheet specified by driverLocation. This is the main driver script that will be loaded and run after a connection to the server is established.
Logging Options	
-l,--log	If specified, debug information is logged. Off, by default.
--logDir	Where to store log and other files. Defaults to the user's home directory.
Procedure Invocation Options	
--projectName <*name*>	The name of the ElectricFlow project containing the procedure to invoke.
--procedureName <*name*>	The name of the ElectricFlow procedure to invoke.
-p,--param <*name*>=<*value*>	Enter additional parameters to the procedure. May appear multiple times. If any parameters were specified in the config file, those supplied on the command-line append to that list, overriding parameters with the same name.
--priority	The priority of the job. Possible values are low, normal, high, highest. If left unspecified, defaults to normal.
--jobTimeout <*timeout*>	The number of seconds to wait for the job to complete when auto-committing changes. Defaults to 3600 seconds (1 hour).
--waitForJob	Wait for the job to complete and report its outcome. This action does not occur by default unless a set of SCM charges are being committed automatically.
--runOnly	Run the procedure and exit immediately. The SCM driver is not downloaded in this case.
SCM Options	

--scmType	The name of the SCM, required, unless the procedure is invoked in 'run only' mode. The driver is downloaded from `$driverLocation/clientDrivers/$scmType`. Valid values: ECSCM-Accurev, ECSCM-Bazaar, ECSCM-ClearCase, ECSCM-CVS, ECSCM-Git, ECSCM-Mercurial, ECSCM-Perforce, ECSCM-Repo, ECSCM-StarTeam, ECSCM-SVN, ECSCM-TFS, ECSCM-Vault.
--autoCommit <*1|0*>	Whether or not the changes should be automatically committed if the job completes successfully. Off, by default.
--commitComment <*comment*>	A comment for the auto-commit.
Perforce Options	
--p4port <*port*>	The value of P4PORT. May also be set in the environment. This is a required value.
--p4user <*user*>	The value of P4USER. May also be set in the environment. This is a required value.
--p4passwd <*password*>	The value of P4PASSWD. May also be set in the environment.
--p4client <*client*>	The value of P4CLIENT. May also be set in the environment. This is a required value.
--p4template <*template*>	The name of a Perforce client used to create a base snapshot before overlaying local changes. Defaults to the value of `--p4client` if not specified.
--p4changelist <*change*>	The changelist number (or default) whose changes are being tested. May be specified multiple times. If no changelists are specified, all client changelists will be used.

The following is a sample XML configuration file with all options specified:

```
<?xml version="1.0" encoding="utf-8"?>
  <data>
    <server>
      <userName>myUser</userName>
      <password>myPass</password>
      <hostName>ElectricFlowServer</hostName>
      <port>1234</port>
      <securePort>2345</securePort>
      <stompPort>3456</stompPort>
```

```
    <secure>0</secure>
    <timeout>3456</timeout>
        <driverLocation>/server/ec_preflight/alternateDrivers</driverLocation>
        <mainDriver>myMainClientDriver</mainDriver>
    </server>
    <procedure>
      <projectName>myProject</projectName>
      <procedureName>myProcedure</procedureName>
      <parameter>
          <name>branch</name>
          <value>main</value>
      </parameter>
      <parameter>
          <name>preflight</name>
          <value>true</value>
      </parameter>
      <priority>high</priority>
      <waitForJob>1</waitForJob>
      <jobTimeout>3600</jobTimeout>
    </procedure>
    <scm>
      <type>perforce</type>
      <port>perf:1234</port>
      <user>myUser</user>
      <password>myPass</password>
      <client>myUser-main-client</client>
      <template>myUser-main-template</template>
      <changelist>default</changelist>
      <changelist>67382</changelist>
      <autoCommit>1</autoCommit>
      <commitComment>Fixing bug 38582.</commitComment>
    </scm>
  </data>
```

In most cases, a developer does not need to override all default values and will probably want to specify passwords and some SCM-specific options on the command-line. The following is a more typical XML configuration:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<data>
  <server>
    <userName>myUser</userName>
    <hostName>commanderServer</hostName>
  </server>
  <procedure>
    <projectName>myProject</projectName>
    <procedureName>myProcedure</procedureName>
    <parameter>
       <name>branch</name>
       <value>main</value>
    </parameter>
    <parameter>
       <name>preflight</name>
       <value>true</value>
    </parameter>
    <jobTimeout>3600</jobTimeout>
  </procedure>
  <scm>
       <type>perforce</type>
       <port>perf:1234</port>
       <user>myUser</user>
       <client>myUser-main-client</client>
  </scm>
</data>
```

If this file is stored in a `.preflight` file at the top-level of a source tree, the command to start the preflight would look something like this:

```
ecclientpreflight --p4changelist 56793 --autoCommit 1 --commitComment "Fixing bug 38582."
```

Default SCMs

ElectricFlow includes preflight support for some of the most common Source Control (SCM) systems. For an SCM plugin you installed, look for a Help link for that plugin (on the Plugins page). Command-line options are included in this Help topic for the following SCMs:

- AccuRev
- Bazaar
- ClearCase

- CVS
- Git
- Mercurial
- Perforce
- Repo
- StarTeam
- Subversion
- TFS
- Vault

Perforce

The command-line options for the Perforce driver for `ecclientpreflight` are:

Perforce Options / Descriptions	
`--p4port <port>`	The value of P4PORT may be set in the environment also.
`--p4user <user>`	The value of P4USER may be set in the environment also.
`--p4passwd <password>`	(Optional) The value of P4PASSWD may be set in the environment also.
`--p4client <client>`	The value of P4CLIENT may be set in the environment also.
`--p4template <template>`	(Optional) The name of a Perforce client used to create a base snapshot before overlaying local changes. Defaults to the value of `--p4client` if not specified.
`--p4changelist <change>`	(Optional) The changelist number (or default) whose changes are being tested and can be specified multiple times. If no changelists are specified, all client changelists will be used.
`--p4synctochange <change>`	(Optional) The changelist number the Preflight Job should use when sync'ing the source tree. Values are: *head*—The most recent changelist anywhere in the P4 depot (default). *have*—The changelist for the most recent file that was synced to 'p4client' changelist. This is a p4changelist number.

Subversion

The command-line options for the Subversion driver for `ecclientpreflight` are:

Subversion Options / Descriptions	
`--svnpath <path>`	The path to the locally accessible source directory in which changes were made.
`--svnupdatetohead`	Use this option to update the agent workspace created during preflight -- updated to HEAD. By default, the agent workspace is updated to the revision found in the client workspace.
`--svnignoreexternals`	Causes the preflight process to ignore svn externals.

AccuRev

The command-line options for the AccuRev driver for `ecclientpreflight` are:

AccuRev Options / Descriptions	
`--accurevuser <user>`	(Optional) The value of ACCUREVUSER may be set in the environment also. If not specified, the default is to the ElectricFlow user.
`--accurevpasswd <password>`	The value of ACCUREVPASSWD may be set in the environment also. This is a required value.
`--accurevpending <pending>`	(Optional) Use this option to scan the client workspace for all pending elements using the "stat -fn -p" command. By default, the workspace is scanned for all kept elements using the "stat -fn -k" command.
`--accurevpath <path>`	The value of the ACCUREVPATH. Also may be set in the environment.

ClearCase

Preflight support for ClearCase is available only for snapshot views. The command-line options for the ClearCase driver for `ecclientpreflight` are:

ClearCase Options / Descriptions	
`--ccpath <path>`	The path to the locally accessible source directory where changes were made.
`--ccUnixCSpecPath <unixCSpecPath>`	The path to the appropriate config spec on a UNIX platform for the current view.
`--ccWinCSpecPath <winCSpecPath>`	The path to the appropriate config spec on a Windows platform for the current view.

ClearCase Options / Descriptions	
`--ccUnixRelativePath <unixRelativePath>`	The relative path for the current view [on a UNIX platform] from the view root to the directory where changes were made.
`--ccWinRelativePath <winRelativePath>`	The relative path for the current view [on a Windows platform] from the view root to the directory where changes were made.

Bazaar

The command-line options for the Bazaar driver for `ecclientpreflight` are:

Bazaar Options / Descriptions	
`--workdir <path>`	The developer's source directory.
`--branch <name>`	Branch name for the preflight.
`--method=<local|remote>`	`local`—get tracked and untracked changes in the current workingdir `remote`—get changes between working tree and remote branch

Git

The command-line options for the Git driver for `ecclientpreflight` are:

Git Options / Descriptions	
`--gitdir=dir`	The Git directory to process.
`--method=<local_all | local_tracked | remote>`	• `local_all`—get tracked and untracked changes between working tree and local repo • `local_tracked`—get tracked changes between working tree and local repo • remote

CVS

The command-line options for the CVS driver for `ecclientpreflight` are:

CVS Options / Descriptions	
`--cvsroot <path>`	The path to the repository.
`--module <module>`	The module name for the preflight.

CVS Options / Descriptions	
`--workdir <path>`	The developer's source directory.

Mercurial

The command-line options for the Mercurial driver for `ecclientpreflight` are:

Mercurial Options / Descriptions	
`--hgpath <path>`	The path to the locally accessible source directory in which changes were made. Generally, this path is to the root of the workspace.

Repo

The command-line options for the Repo driver for `ecclientpreflight` are:

Repo Options / Descriptions	
`--repoworkdir <path>`	The developer's source directory.
`--agentworkdir <path>`	The path to the source directory, used by the agent.

StarTeam

The command-line options for the StarTeam driver for `ecclientpreflight` are:

StarTeam Options / Descriptions	
`--STProjectName <project>`	The StarTeam project name.
`--workingdir <path>`	Working `dir` for the updated files.

TFS

The command-line options for the Team Foundation Server driver for `ecclientpreflight` are:

TFS Options / Descriptions	
`--server <url>`	The equivalent value to Collection in VS2008 and under. If you use this option, it is assumed you are using TF version VS2008 or under and it will use the option /server in the preflight. This field is required if you have VS2008.

TFS Options / Descriptions	
`--collection <url>`	The URL that points to /collection. This value is used for VS2010. If you specify this value, the command uses the option /collection when it executes the TF query for preflight. This field is required if you have VS2010.
`--localfolder`	The path to the locally accessible source directory in which changes were made. Generally, this is the path to the root of the workspace.
`--workspace <workspace>`	The workspace containing the data for the preflight.

Vault

The command-line options for the Vault driver for `ecclientpreflight` are:

Vault Options / Descriptions	
`--workingdir <path>`	Working `dir` for the updated files.

Troubleshooting

Client

The `--debug` option to `ecclientpreflight` provides additional information about internal actions taken when starting the preflight build. By modifying the properties containing driver scripts, debug output can be added where needed.

If you want to capture debug information to a log file, use the `--log` option. By default, logs are stored in the user's home directory. To change this location, use the `--logDir` option.

Agent

The step log for the custom preflight step provides information about the snapshot being created and the files being overlayed. Debug output can be added where needed by modifying the properties containing the driver scripts

Properties

ElectricFlow provides a powerful data model based around the notion of a *property*.

- A property is a string value with a name.
- Properties can have arbitrary names and values.
- You can attach properties to any object in the ElectricFlow system, such as a project, procedure, or job.
- After a property is created, it is stored in the ElectricFlow database.
- You can retrieve and modify the property value later.
- You can delete properties that you no longer need.

Properties are used extensively throughout the ElectricFlow system and provide a flexible and powerful mechanism to manage data about your builds.

Property Use Case Examples

- When a job starts, it computes a unique identifier for that build and saves it in a property. Later build steps can retrieve the property value to embed the build number in binaries generated during the job.
- When a build executes, it can set a property on the procedure to identify the source code version used for the build (for example, a tag or timestamp from your source code control system). The next time the build executes, it can retrieve the property value from the procedure and use it to extract information from your source code control system about the files that changed since the last run.
- Suppose you have a collection of machines for testing, and some machines have older test hardware and some machines have newer hardware with slightly different characteristics. You can set a property on each test machine resource to indicate which version of test hardware is available on which machine. Later, when a test executes, the test can retrieve the property for the machine where it ran and configure tests appropriately for that hardware. If you upgrade test hardware on a machine, all you need to do is change the property on that resource to reflect the new version.
- You can set job properties to indicate the build status produced by that job. For example, if your QA team finds fatal flaws in a build, it can mark the job accordingly. Builds that need to be preserved (release candidates or builds undergoing beta testing at customer sites) can be marked with properties so those builds are not deleted.
- When a job executes, properties are set for its job steps that hold metrics such as how many files compiled during a build step, how many tests executed during a test step, or how many errors were detected in the step. These property values are included in reports and can be examined later to compute trends over time. You can define additional properties for metrics useful to you.

ElectricFlow provides Intrinsic properties and allows you to create Custom properties. This Help topic enumerates properties available within ElectricFlow (Intrinsic properties), distinguishes between relative and absolute property paths, and describes property hierarchies.

> **Note:** Intrinsic properties are case-sensitive. Custom properties, like all other object names in the ElectricFlow system, are "case-preserving," but *not case-sensitive*.

- **Intrinsic properties**
  These properties represent attributes that describe the object to which they are attached, and are provided automatically by ElectricFlow for each similar type object. For example, every project has a `Description` property that can be referenced with a non-local property path such as `/projects/Examples/description`.
- **Custom properties**
  Custom properties are identical to intrinsic properties and when placed on the same object, are referenced in the same manner, and behave in every way like an intrinsic object-level property with one exception: they are *not* created automatically when the object is created. Instead, custom properties can be added to objects already in the database before a job is started, or created dynamically by procedure steps during step execution.

Creating or modifying properties

You can set a property value by using one of the following three methods or applications: the automation platform web interface, the ectool command-line application, or the Perl API.

- Using the web interface, you will see a table labeled Custom Properties on the pages that display details for projects, procedures, and so on.
  Click **Create New Property** to create a new property for that object, or click on an existing property to change its value.
- Using the ectool application, set a property value with a command like:

```
ectool setProperty /myJob/installStatus complete
```

This command sets the property named `/myJob/installStatus` to the value `complete`, and creates the property if it did not already exist (the meaning of property names like `/myJob/installStatus` is explained below). You can use ectool from within one job step to set properties accessed by later steps in the same job.

- Using the Perl API, you can use an external script or a script in a step with *ec-perl* as the shell.
  A Perl code example:

```
use ElectricCommander ();
  my $ec = new ElectricCommander;
  $ec->setProperty ("/myJob/installStatus", "complete", {jobStepId =>
  $ENV{COMMANDER_JOBSTEPID}
```

> **Tip:** Using the Perl API may yield better performance if you are requesting multiple API calls in one step.

For more information, see the "Using the ElectricFlow API " topic in the *ElectricFlow API Guide* at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html.

Using property values

A job step can access a property value by two methods. The first method is *substitution*, using the `$[ ]` notation. Suppose you enter the following text as a command for a step:

```
make PLATFORM=$[platform]
```

Before running the step, ElectricFlow finds the property named `platform` and substitutes its value in the command string in place of `$[platform]`. For example, if the property contains the value `"windows"`, the actual command executed is:

```
make PLATFORM=windows
```

The substitution method can be used for a command in a step and for any step fields, such as the resource or working directory. This method allows you to compute configuration information in an early step of a job, then use that configuration information to control later steps in the job.

Another way for a step to access the property value is with the ectool `getProperty` command. For example, the following command returns the property value named `platform`:

```
ectool getProperty platform
```

Property sheets and intrinsic properties

A property value can be a simple string or a nested *property sheet* containing its own properties. Property sheets can be nested to any depth, which allows you to create hierarchical collections of information.

Most objects have an associated "property sheet" that contains "custom properties" created by user scripts. The property sheet is an intrinsic property of the containing object called "property sheet", so to reference a project's custom property "branchName", you could specify `"/projects/aProject/propertySheet/branchName"`.

As a convenience, ElectricFlow allows the "property sheet" path element to be omitted and the path written as `"/projects/aProject/branchName"`. If there is no intrinsic property with the same name, the path will find the property on the property sheet.

Custom properties in a property sheet can be one of two types: *string property* or a *property sheet* property. String properties hold simple text values. Property sheet properties hold *nested* properties. Nested properties are accessed via the property sheet property of their containing sheet.

For example:

```
/projects/aProject/propertySheet/topSheet/propertySheet/propB
```
- or -
```
/projects/aProject/topSheet/propB
```

All information managed by ElectricFlow exists in the form of properties and property sheets and your own custom-created properties. For example, each project, procedure, and step is represented internally as a property sheet—the command for a step is actually a property associated with the step, and so on. Every value in the ElectricFlow system can be accessed as a property, using the naming facilities described below. These properties are called *intrinsic* properties. *ElectricFlow enforces some restrictions on intrinsic property values, whereas custom properties can have any value you choose.*

To learn more about intrinsic properties defined for an object by ElectricFlow, see the "Using the ElectricFlow Perl API" chapter in the *ElectricFlow API Guide* at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html. For example, to learn about intrinsic properties associated with each project, find the documentation for the `getProject` ectool command. The result of running this command is an XML document whose field names and values represent the properties for the project.

For details about the intrinsic properties for each object in ElectricFlow Deploy, see the *Intrinsic Properties in ElectricFlow Deploy Objects* document at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html. For details about the intrinsic properties for each object in the Automation Platform, see Intrinsic properties listed by object type on page 1444.

Property names and paths

Properties are named using multi-level paths such as first/second/third, which refers to a property named "third" in a property sheet named "second" in a property sheet named "first." ElectricFlow also supports an equivalent notation using brackets instead of slashes. In this format, slashes are not considered separators when they appear between brackets.

For example, `first[second]/third` and `first[second][third]` both refer to the same property as first/second/third. The bracket `[ ]` notation is based on matching brackets. For example, `steps[build[1]]` refers to a property named "build[1]" inside a property sheet named "steps" (it does not treat "build" as a property sheet containing a property named "1").

Property names/paths have two forms: *absolute* and *relative*.

Absolute property paths

Absolute property paths are referenced by a fully-qualified path syntax that begins with a slash character ("/") followed by a top-level name. This path syntax is similar to a file system path specification. The first component after the `"/"` must be one of several reserved words that select a starting location to look up the property name. For example, consider the property name `/server/Electric Cloud/installDirectory`. `/server` means "lookup" starts in the topmost property sheet associated with the ElectricFlow server. This property name refers to the "installDirectory" property inside the server property sheet.

The system defines the following top-level names (absolute property names):

`/applications/...`

Start in the property sheet containing all applications. For example, `/applications[deploy]` refers to an application named "deploy" and `/applications[deploy]/processes[install]` refers to a process named "install" in that application.

`/artifacts/...`

Start in the property sheet containing all artifacts. For example, `/artifacts/myGrp:myKey/prop1` refers to the `prop1` property on the artifact whose name is `"myGrp:myKey"`.

`/artifactVersions/...`

Start in the property sheet containing all artifact versions. For example, `/artifactVersions/myGrp:myKey:1.0-36/prop1` refers to the `prop1` property on the artifact version whose name is `"myGrp:myKey:1.0-36"`.

> **Tip:** Throughout the API, you can substitute `"groupId:artifactKey:version"` wherever an `artifactVersionName` argument can be specified. This is the same if the `artifactVersionNameTemplate` specified in the artifact is in the form `"groupId:artifactKey:version"`. When the template is different, it is convenient to be able to specify this tuple if you do not know the `artifactVersionName`.

`/components/...`

Start in the property sheet containing all components. For example, `/components[warfile]` refers to a component named "warfile" and `/component[warfile]/processes[backup]` refers to a component process named "backup" in that component.

`/environments/...`

Start in the property sheet containing all components. For example, `/environments[qeserver]` refers to an environment named "qeserver".

`/groups/...`

Start in the property sheet containing all groups. For example, `/groups[dev]` refers to the group named "dev".

`/jobs/...`

Start in the property sheet containing all jobs. For example, `/jobs[ecloud.4096]` refers to the job named "ecloud.4096." You can name a job using either its name (as in the preceding example) or using the unique identifier assigned to it by ElectricFlow.

`/plugins/...`

Start in the property sheet containing all plugins. For example, `/plugins[EC-AgentManagement]` refers to the currently promoted plugin named "EC-AgentManagement" and `/plugins[EC-AgentManagement]/project/procedures[scpCopyFile]` refers to a procedure named "scpCopyFile" in that plugin.

> **Note:**
> There is a subtle difference between
>
> `ectool setProperty /plugins/EC-AgentManagement/project/foo 'bar'`
>
> and
>
> `ectool setProperty /plugins/EC-AgentManagement/foo 'bar'`
>
> The former places the property on the plugin's project and can be referenced by `$[/myProject/foo]` while the latter places the property on the plugin object and will not be found via `$[/myPlugin/foo]`.

`/projects/...`

Start in the property sheet containing all projects. For example, `/projects[nightly]` refers to the project named "nightly," and `/projects[nightly]/procedures[main]` refers to a procedure named "main" in that project.

`/repositories/...`

Start in the property sheet containing all artifact repositories. For example, `/repositories/repo1/prop1` refers to the `prop1` property on the repository whose name is `"repo1"`.

`/resourcePools/...`

Start in the property sheet containing all resource pools. For example, `/resourcePools[linuxA]` refers to the resource pool named "linuxA".

`/resources/...`

Start in the property sheet containing all resources. For example, `/resources[linux1]` refers to the resource named "linux1".

`/server/...`

Start in the top-level server property sheet. For example, `/server/a` refers to the server property named "a".

`/users/...`

Start in the property sheet containing all users. For example, `/users[Bob]` refers to the usernamed "Bob".

`/workspaces/...`

Start in the property sheet containing all workspaces. For example, `/workspaces[default]` refers to the workspace named "default."

Relative property paths

Property names that do not begin with `"/"` are *relative* and looked up starting in the *current context*. For example, when executing a job step, the current context includes properties defined for the job step, parameters for the current procedure, and global properties on the current job.

Relative property paths are distinguished from absolute property paths because they do not begin with one of the top-level names. To avoid having to construct full property paths, ElectricFlow supports the concept of a relative property path.

In use, the relative property path value is resolved by its context and a defined search order, which results in accessing the value of an absolute fully-specified property value. Contexts and search orders are as follows:

1. In a job step context, ElectricFlow searches for relative property paths in the following order:
   - a property on the job step object
   - a property of the parent job step object, which includes parameters of the procedure on which the step is defined

- a property on the job object

> **Note:**
>
> - The search can be enabled or disabled by using the `--extendedContextSearch` option on the ectool `getProperty` or `setProperty` commands. When searching for a property value, disable the search by setting the `--extendedContextSearch` switch to "false", requiring the property to exist on the job step object or return false. When writing a new value to a property, enable the search by setting the `--extendedContextSearch` switch to "true", allowing the search for a property within the search order before creating a new one. New properties are created on the job step object.
> - Parameters for subprocedure calls from job steps are searched for in a job step context.
> - For procedure parameters for nested subprocedures, properties referenced by parameters are looked up as described [above] for job steps.

2. When expanding schedule parameters, ElectricFlow searches the relative property path in the following order:
   - A property on the procedure being called
   - A property on the project on which the procedure being called is defined
   - A property on the server
3. In a job name template context, ElectricFlow searches for the relative property path as a property on the job.
4. In any other context, ElectricFlow searches for the relative property path as a property on the context object.

jobSteps Paths

You can also specify a property on a job step as `/jobSteps/parent/propertyName`where:

- `parent` is the root parent of the job step that is not visible on the ElectricFlow UI.
- `propertyName` is the name of the property.

Context-relative Shortcuts to Property Paths

A shortcut can be used to reference a property without knowing the exact name of the object that contains the property. You might think of a shortcut as another part of the property hierarchy. Shortcuts resolve to the correct property path even though its path elements may have changed because a project or procedure was renamed. Shortcuts are particularly useful if you do not know your exact location in the property hierarchical tree.

The table below lists all shortcuts and the context in which they are available. Click on a shortcut name for more information about it.

Shortcut Name	Available Context		
	Job, Step, Or Job Step	Pipeline, Task, Stage, Or Gate	Context Independent
/myApplication/... on page 1432	✓	N/A	N/A
/myApplicationTier/... on page 1432	✓	N/A	N/A
/myArtifactVersion/... on page 1432	N/A	N/A	`artifactVersionNameTemplate` field for the artifact
/myComponent/... on page 1432	✓	N/A	N/A
/myCredential/... on page 1432	✓	✓	N/A
/myDeployerTaskRuntime/... on page 1432	✓	N/A	N/A
/myEvent/... on page 1432	N/A	N/A	✓
/myEnvironment/... on page 1433	✓	N/A	N/A
/myEnvironmentTier/... on page 1433	✓	N/A	N/A
/myGate/... on page 1433	✓	✓	N/A
/myGateRuntime/... on page 1433	✓	✓	N/A
/myGroupTaskRuntime on page 1433	✓	✓	N/A
/myJob/... on page 1433	✓	N/A	N/A
/myJobStep/... on page 1433	✓	N/A	N/A
/myParent/... on page 1433	✓	N/A	N/A
/myPipeline/... on page 1434	✓	✓	
/myPipelineRuntime/... on page 1434	✓	✓	N/A
/myPipelineStageRuntime/... on page 1434	✓	✓	
/myProcedure/... on page 1434	✓	N/A	N/A

Shortcut Name	Available Context		
	Job, Step, Or Job Step	Pipeline, Task, Stage, Or Gate	Context Independent
/myProcess/... on page 1434	✓	N/A	N/A
/myProcessStep/... on page 1434	✓	N/A	N/A
/myProject/... on page 1435	✓	✓	N/A
/myResource/... on page 1435	✓	N/A	N/A
/myResourcePool/... on page 1435	✓	N/A	N/A
/myRetrievedArtifact/... on page 1435	✓	N/A	N/A
/myStage/... on page 1435	✓	✓	N/A
/myStageRuntime/... on page 1435	✓	✓	N/A
/myStep/... on page 1435	✓	N/A	N/A
/myState/... on page 1435	✓	N/A	N/A
/mySubjob/... on page 1436	✓	N/A	N/A
/mySubworkflow/... on page 1436	✓	N/A	N/A
/myTask/... on page 1436	✓	✓	N/A
/myTaskRuntime/... on page 1436	✓	✓	N/A
/myTransition/... on page 1436	✓	N/A	N/A
/myTriggeringPipelineRuntime/... on page 1436	✓	✓	N/A
/myUser/... on page 1437	N/A	N/A	✓
/myWorkflow/... on page 1437	✓	N/A	N/A
/myWorkflowDefinition/... on page 1437	✓	N/A	N/A
/myWorkspace/... on page 1437	✓	N/A	N/A

Shortcuts

`/myApplication/...`

Resolves to the `application` object associated with the current job or job step.

`/myApplicationTier/...`

Resolves to the `applicationTier` object associated with the current job step.

`/myArtifactVersion/...`

Resolves to the `artifactVersion` object associated with the current context. The only context where this property has a value is in the `artifactVersionNameTemplate` field for the artifact.

`/myComponent/...`

Resolves to the `component` object associated with the current context.

`/myCredential/...`

Resolves to the credential object associated with the current job step. This form produces an error if the current job step lacks a credential.

/`myDeployerTaskRuntime`/...

Resolves to the deployer runtime to retrieve information for the current deployer runtime instance.

Examples:

`/myDeployerTaskRuntime/outcome`—Retrieve the outcome of the deployer task in the current context.

`/myDeployerTaskRuntime/tasks/<subapplication>/job/jobId`—Access the specified deployer subapplication.

`/myDeployerRuntime/currentRunNumber`-Get the current run number.

`/myEvent/...`

This is a special property used only within an "email notifier." `/myEvent/` allows you to refer to fields associated with the notifier itself. You can include these properties, for example, in the text of the email you send out for notification:

`/myEvent/notifier`—Contains the name of the notifier that created the notifier event. You created this name when you created the notifier.

`/myEvent/entity`—Contains the object where the notifier is attached. Two possible values are "job" or "jobStep," depending on where the notifier is attached.

`/myEvent/source`—Contains the name of either the job or the jobStep where the notifier is attached.

`/myEvent/type`—Defines whether the notifier is an "On Start" or an "On Completion" notifier. Two possible values are "STARTED" or "COMPLETED".

`/myEvent/time`—Contains the time when the notifier occurred. The time is always specified in GMT and uses a formatted string, such as
`2009-06-11T21:00:56.502Z`

`/myEvent/timeMillis`—Contains the "timestamp" in milliseconds when the notifier occurred. This value is the number of milliseconds since January 1, 1970 GMT. The `timeMillis` corresponding to the time in the previous example is: `1244754056502`

`/myEnvironment/...`

Resolves to the `environment` object associated with the current job or job step.

`/myEnvironmentTier/...`

Resolves to the `environmentTier` object associated with the current job step.

`/myGate/...`

`/myGate/` is limited to `myGate.gateType` (`PRE` or `POST`) or `myGate.parentStage`. It is useful to include information about the active gate and stage in email notifications templates.

`/myGateRuntime/...`

Resolves to the current gate runtime object.

Examples:

`/myGateRuntime/tasks/<taskName>/propertyName`—Access a property on another task in an entry gate.

`/myGateRuntime/currentRunNumber`—Get current run number.

`/myGateRuntime/runNumbers/1/subFlowRuntime/prop1`—Access a property under the first gate run, previously set with `/myGateRuntime/prop1`.

`/myGroupTaskRuntime`

Resolves to the current `myGroupTaskRuntime` object.

Examples:

`/myGroupTaskRuntime/tasks/<taskName>/propertyName`—Access the subtask runtime within the current group task context.

`/myGroupTaskRuntime/currentRunNumber`-Get the current run number.

`/myJob/...`

Resolves to the current job. `/myJob/` points directly to the job object so you can reference built-in job properties (`jobName`, `createTime`, `outcome`, and so on) and custom properties. This shortcut can also be used to hold working data that needs to be passed from one step to another within the job.

`/myJobStep/...`

Resolves to the current job step.

`/myParent/...`

Resolves to the current parent job step or job if this is a top-level step. `/myParent/` points directly to the job or job step object, so you can reference intrinsic or custom properties. For example, you can reference the parameters that were passed to this procedure.

In addition, you can reference a sibling step by calling `/myParent/jobSteps/<sibling step name>`.
Or, you can create additional properties on `/myParent/` to pass information from one step in the procedure to another:
step1 calls `setProperty /myParent/results 100`

step2 can call `getProperty /myParent/results /myParent/`.

`/myPipeline/...`

Resolves to the current `pipeline` definition object. This can be used by a plugin, procedure, or process step executed from a stage task to retrieve property information defined in the pipeline.

`/myPipelineRuntime/...`

Resolves to the current `pipelineRuntime` runtime object. This can be used by a plugin, procedure, or process step executed from a stage task to retrieve property information stored on the pipeline runtime

Examples:

`/myPipelineRuntime/stages/<stageName>/outcome`—Retrieve the outcome of a previous stage within the context of the current stage.

`/myPipelineRuntime/stages/<stageName>/tasks/<taskName>/outcome`—Retrieve the outcome of a task within the context of the current stage.

`/myPipelineRuntime/stages/<stageName>/gates/PRE/tasks/<taskName<propertyName>`—A gate task accesses a property on another task's entry gate within the current pipeline runtime context.

`/myPipelineRuntime/stages/<stageName>/gates/POST/tasks/<taskName>/<propertyName>`—A gate task accesses a property on another task's exit gate within the current pipeline runtime context.

`/myPipelineRuntime/stages/<stageName>/tasks/<taskName>/<propertyName>`—A stage task acccesses a property on itself.

`/myPipelineRuntime/stages/<stageName>/gates/PRE/tasks/<taskName>/<propertyName>`—A stage task accesses a property on an entry gate

`/myPipelineRuntime/stages/<stageName>/tasks/<taskName>/job/<propertyName>`—A task accesses a property on a job created by a task.

`/myPipelineRuntime/stages/<stageName>/tasks/<taskName>/workflow/<propertyName>`—A task accesses a property on a workflow created by a task.

`/myPipelineStageRuntime/...`

Resolves to the `pipelineStageRuntime` with the current pipeline runtime. This can be used by a procedure step executed from a stage task to retrieve information for the summary property of the stage runtime.

`/myProcedure/...`

Resolves to the procedure in which the current job step was defined. If the current job step is executing as part of a nested procedure, `/myProcedure/` refers to the innermost nested procedure.

`/myProcess/...`

Resolves to the application or component process in which the current job or job step was defined.

`/myProcessStep/...`

Resolves to the application or component process step in which the current job step was defined.

`/myProject/...`

Resolves to the project in which the current job step was defined. If the job step has nested procedure invocations, this is the project associated with the innermost nested procedure; for example, the project associated with `/myProcedure/`.

`/myResource/...`

Resolves to the resource object assigned to the current job step.

`/myResourcePool/...`

Resolves to the resource pool object that provided the resource for the current job step. Returns null if the step did not specify a resource pool.

`/myRetrievedArtifact/...`

Resolves to the current `RetrievedArtifact` object available from a component process step.

`/myRetrievedArtifact/artifactVersion`—Version of the retrieved artifact.

`/myRetrievedArtifact/artifactName`—Name of the retrieved artifact.

`/myRetrievedArtifact/componentName`—Name of the component containing this retrieved artifact.

`/myStage/...`

Resolves to the current pipeline Stage object.

`/myStageRuntime/...`

Resolves to the stage runtime object for the current stage runtime instance. This can be used to retrieve information for the summary property of the stage runtime.

Examples:

`myStageRuntime/tasks/test/outcome`—Test the runtime outcome of the previous task in the current stage.

`/myStageRuntime/runnumbers/1/output1`—Get the runtime outcome for specific runs in the current stage.

`/myStageRuntime/runNumbers/1/subFlowRuntime/prop1`—Access the property under the first stage run, previously set with `/myStageRuntime/prop1`.

`/myStageRuntime/currentrunNumber`—Get the current run number.

`/myStageRuntime/tasks/<taskName>/propertyName`—Accesses a property on another task in a stage.

`/myStageRuntime/tasks/groupTask/tasks/<subTaskName>/job/<jobid>`—Access an underlying group task from withing the current stage runtime context.

`/myStep/...`

Resolves to the current `Step` object, where `Step` refers to the static definition of a step that is part of a procedure. This is in contrast to a job step, which represents a step when it executes dynamically in a job. Use `/myJobStep/` to access the runtime job step.

`/myState/...`

Resolves to the current `State` object to reference built-in and custom state properties or find parameter values passed to that state. When accessed from a state, `/myState/` refers to that state. When accessed from a transition, `/myState` refers to the transition's owning state. When accessed from a job or job step, `/myState/` refers to the state that launched that job as a subjob.

`/mySubjob/...`

Resolves to the current `Subjob` object used to reference built-in and custom job properties, parameters passed to the job, and properties on steps within that job. When accessed from a transition, `/mySubjob/` refers to the subjob started by the state that owns that transition. This property path is particularly useful in conditions for On Completion transitions because the outcome or other information for the subjob can influence which state the workflow transitions to next.

`/mySubworkflow/...`

Resolves to the current `Subworkflow` object used to reference built-in and custom workflow properties. When accessed from a transition, `/mySubworkflow/` refers to the subworkflow started by the state that owns that transition. This property path is particularly useful in conditions for On Completion transitions because the active state and other information for the subworkflow can influence which state the workflow transitions to next. You can access information about states and transitions belonging to the workflow by using the path `/mySubworkflow/states/someState` or `/mySubworkflow/states/someState/transitions/someTransition`.

`/myTask/...`

Resolves to the current pipeline `Task` .

`/myTaskRuntime/...`

Resolves to the current `Task` runtime. This can be used to retrieve information for the summary property of the task runtime.

Examples:

`/myTaskRuntime/runnumbers/2/output1`—Get the runtime outcome for specific runs in the current task.

`/myTaskRuntime/runNumbers/1/subFlowRuntime/prop1`—Access the property under the first task run, previously set with `/myTaskRuntime/prop1`.

`/myTaskRuntime/propertyName`—Task that accesses the property on itself.

`/myTaskRuntime/job/propertyName`—Task that accesses the property on a job created by a task.

`/myTaskRuntime/workflow/propertyName`—Task that accesses the property on a workflow created by a task.

`/myTaskRuntime/currentRunNumber`-Get the current run number.

`/myTransition/...`

Resolves to the current `Transition` object used to reference intrinsic and custom transition properties. `/myTransition/` is accessible only from a transition.

`/myTriggeringPipelineRuntime/...`

In a situation where a pipeline triggers a sub-pipeline, this property accesses the runtime context of the parent pipeline from within the sub-pipeline runtime context.

Examples:

`/myTriggeringPipelineRuntime/stages/<stageName>/outcome`—Retrieve the outcome of a previous stage within the context of the current stage of the triggering pipeline runtime context.

`/myTriggeringPipelineRuntime/stages/<stageName>/tasks/<taskName>/outcome`—Retrieve the outcome of a task within the context of the current stage of the triggering pipeline.

`/myUser/...`

This property is used only if the current session is associated with the predefined `admin` user, a user defined as `local`, or a user defined by a Directory Provider (LDAP or ActiveDirectory). This property cannot be used if the user is a project principal, which is normally the case when running inside an ElectricFlow step.

For example, with an interactive login you can use:
`ectool getProperty /myUser/userName` (to get the username of the logged in user, or...)
`ectool getProperty /myUser/email` (to get the email address)

`/myWorkflow/...`

Resolves to the current workflow object used to reference built-in and custom workflow properties. When accessed from a state or transition, `/myWorkflow/` refers to the "owning" workflow. When accessed from a job or job step, `/myWorkflow/` refers to the workflow whose state launched that job as a subjob. You can access information about states and transitions belonging to the workflow by using the path `/myWorkflow/states/someState` or `/myWorkflow/states/someState/transitions/someTransition`.

`/myWorkflowDefinition/...`

Resolves to the current `WorkflowDefinition`object used to reference built-in and custom workflow properties.

`/myWorkspace/...`

Resolves to the current `Workspace` object associated with the current job step.

Property path shortcuts in ElectricFlow 5.0 and later

The following shortcuts to property paths are available in ElectricFlow 5.0 and later. For more information, see Context-relative Shortcuts to Property Paths on page 1429.

Shortcuts	Descriptions	For jobs	For job steps
`/myApplication/...`	Starts in the property sheet for the application associated with the current job or job step.	Yes	Yes
`/myApplicationTier/...`	Starts in the property sheet for the application tier associated with the current job step.	No	Yes
`/myComponent/...`	Starts with the property sheet for the component associated with the current job step.	No	Yes

Shortcuts	Descriptions	For jobs	For job steps
/myCredential/...	Starts with the property sheet for the credential associated with the current job step.	No	Yes
/myEnvironment/...	Starts in the property sheet for the environment associated with the current job or job step.	Yes	Yes
/myEnvironmentTier/...	Starts in the property sheet for the environment tier associated with the current job step.	No	Yes
/myGate/...	/myGate/ is currently limited to myGate.gateType (PRE or POST) or myGate.parentStage. It is useful to include information about the active gate and stage in email notifications templates.	Yes	Yes
/myInventoryItem/...	Starts in the property sheet for the environment inventory.	Yes	Yes
/myJob/...	Starts in the property sheet for the job associated with the current job or job step.	Yes	Yes
/myJobStep/...	Starts in the property sheet for the job step associated with the current job step.	No	Yes
/myParent/...	Starts in the global property sheet for the parent job step.	No	Yes
/myPipeline/...	Start in the property sheet for the pipeline associated with the current job or job step.	Yes	Yes
/myPipelineRuntime/...	Start in the property sheet for the pipelineRuntime associated with the current job or job step.	Yes	Yes
/myPipelineStageRuntime/...	Start in the property sheet for the pipelineStageRuntime for a pipeline stage where the current job or job step was defined. This can be used by a procedure step executed from a stage task to retrieve information for the summary property of the stage runtime.	Yes	Yes
/myPlugin/...	Starts in the property sheet for the plugin associated with the current job.	Yes	No
/myProcedure/...	Starts in the property sheet for the procedure in which the current job or job step was defined.	Yes	Yes

Shortcuts	Descriptions	For jobs	For job steps
/myProcess/...	Starts in the property sheet for the process in which the current job or job step was defined.	Yes	Yes
/myProcessStep/...	Starts in the property sheet for the process step in which the current job step was defined.	No	Yes
/myProject/...	Starts in the property sheet for the project in which the current job was defined.	Yes	No
/myResource/...	Starts in the property sheet for the resource associated with the current job step.	No	Yes
/myResourcePool/...	Starts in the property sheet for the resource pool associated with the current job step.	No	Yes
/myResourcePool/...	Starts in the property sheet for the resource pool associated with the current job step.	No	Yes
/myStage/...	Start in the property sheet for the pipeline stage in which the current job or job step was defined.	Yes	Yes
/myState/...	Starts in the property sheet for the state object so you can reference built-in and custom state properties or find parameter values passed to that state.	Yes	Yes
/myStep/...	Starts in the property sheet for the step associated with the current job step.	No	Yes
/myTask/...	Start in the property sheet for the pipeline task in which the current job or job step was defined.	Yes	Yes
/myWorkflow/...	Starts in the property sheet for the workflow object so you can reference built-in and custom workflow properties.	Yes	Yes
/myWorkflowDefinition/...	Starts in the property sheet for the workflow definition object so you can reference built-in and custom workflow properties.	Yes	Yes
/myWorkspace/...	Starts in the property sheet for the workspace associated with the current job step.	No	Yes

Property name substitutions

Property names can contain references to other properties, which are then substituted into the property name before looking it up. For example, consider the following property name:

```
/myStep/$[/myProcedure/name]
```

If the value of `/myProcedure/name` is "xyz", the property above is equivalent to `/myStep/xyz`.

Expandable properties

Property values can contain property references using the `"$[]"` notation.

For example:

1. Create a property named "foo" with a value of `hello $[bar]`.
2. Create a property named "bar" with a value of `world`.
3. Reference "foo" (either using `$[foo]` or `ectool getProperty foo`). The value "hello world" is returned.

If you want just the literal value of "foo" (useful in the UI, for example), you can use the `expand` option in ectool:

`ectool getProperty foo --expand false`. The value `"hello $[bar]"` will be returned.

Properties are expanded by default when you use `getProperty` or `getProperties`.

If the value of a property contains `"$[]"` but you do not want it to be interpreted as a property reference, you can use the expandable option:

```
ectool setProperty symbols '$[!@]#' --expandable false.
```

This option can be toggled in the web UI as well. Properties are expandable by default.

Because you cannot control where your expandable property might be referenced (and therefore which context is used during expansion), we recommend using absolute paths when referencing a property from the value of another property.

In the example above, if you define both "foo" and "bar" as properties on a project "proj1", you might assume there is no problem with the value of "foo". However, if you later reference "foo" from a job under "proj1" (for example, `$[/myProject/foo]`), foo will be referenced with the job step as its context. Therefore, when the value of "foo" is expanded, you will get a PROPERTY_REFERENCE_ERROR because "bar" is not defined in the context of the job step.

Custom property names and values

The following properties are used by the standard ElectricFlow UI, so you should use these property names whenever possible, and avoid using these names in ways that conflict with the definitions below.

- **`compiles`**—the number of files compiled during the job step
- **`diagFile`**—the filename in the top-level directory of the job's workspace, containing diagnostics extracted from the step's log file
- **`errors`**—the number of errors (compilation failures, test failures, application crashes, and so on) that occurred during the job step. When property errors are set by postp, the step outcome is set to error also.
- **`tests`**—the number of tests executed by the job step, including successes and failures
- **`testsSkipped`**—the number of tests skipped during the job step
- **`warnings`**—the number of warnings that occurred during the job step. When property warnings is set by postp, the step outcome is set to warning also.

- **`preSummary`**—if this property exists, its value is displayed in the "Status" field (on the Job Details page) for this step. This property appears *before* whatever would normally be displayed for status. If the property contains multiple lines separated by newline characters, each line is displayed on a separate line in the status field.
- **`postSummary`**—if this property exists, its value is displayed in the "Status" field (on the Job Details page) for this step. This property appears *after* whatever would normally be displayed for status. If the property contains multiple lines separated by newline characters, each line is displayed on a separate line in the status field.
- **`summary`**—if this property exists, its value is displayed in the "Status" field (in the job reports) for this step, replacing whatever would normally be displayed for status. If the property contains multiple lines separated by newline characters, each line is displayed on a separate line in the status field.

The property hierarchy

All ElectricFlow properties fall into a hierarchical structure, and you can reference any property using an absolute path from the root of the hierarchy. This includes properties you define and intrinsic properties defined by ElectricFlow.

For example, each step contains a property "resource" that provides the resource name to use for that step. All steps of a procedure exist as property sheets underneath the procedure. All procedures in a project exist as property sheets underneath the project, and so on.

The examples below illustrate some nesting relationships between objects.

For example, the notation "*project*/procedures[*procedureName*]" means each project object contains a property sheet named "procedures" holding all procedures defined within that project. Within the procedures property sheet, there is a nested property sheet for each procedure, named after the procedure. Thus, the name "/projects[a]/procedures[b]" refers to a procedure named "b" contained in a project named "a."

- Each project contains procedures, schedules, credential definitions, workflow definitions, and workflows:
  project/procedures[*procedureName*]
  project/schedules[*scheduleName*]
  project/credentials[*credentialName*]
  project/workflowDefinitions[*workflowName*]
  project/workflows[*workflowName*]
- Each procedure contains steps and parameters. The parameters are "formal parameters," meaning they specify parameter names the procedure will accept, whether each parameter is required, and so on:
  procedure/steps[*stepName*]
  procedure/formalParameters[*paramenterName*]
- Steps that invoke subprocedures contain parameter values for the subprocedure. These are called "actual parameters" because they provide actual values that will be passed into the subprocedure:
  step/actualParameters[*parameterName*]
- Each job contains a collection of job step objects for steps in the outermost procedure, along with parameter values passed into the job when it was invoked:
  job/jobSteps[*stepName*]
  job/actualParameters[*parameterName*]

- If a job step invokes a nested subprocedure, its property sheet contains parameter values that were passed into the nested subprocedure, plus all job steps corresponding to that procedure:
  jobStep/actualParameters[*parameterName*]
  jobStep/jobSteps[*stepName*]
- Schedules contain parameter values for the procedures they invoke. These are called "actual parameters" because they provide actual values passed into the procedure:
  schedule/actualParameters[*parameterName*]
- Workflow definitions contain state definitions:
  workflowDefinition/stateDefinitions[*stateDefinitionName*]
- Transition definitions contain actual parameters:
  transitionDefinition/actualParameters[*parameterName*]
- State definitions contain transition definitions, actual parameters, and formal parameters:
  stateDefinition/actualParameters[*parameterName*]
  stateDefinition/formalParameters[*parameterName*]
  stateDefinition/transitionDefinitions[*transitionDefinitionName*]
- Workflows contains states:
  workflow/states[*stateName*]
- Transitions contain actual parameters:
  transition/actualParameters[*parameterName*]
- States contain transitions, actual parameters, and formal parameters:
  state/actualParameters[*parameterName*]
  state/formalParameters[*parameterName*]
  state/transitions[*transitionName*]

Special property references

ElectricFlow also supports several special property reference forms that are described in the following subsections.

increment

Use this form to increment the value of an integer property before returning its value. For example, suppose property xyz has the value 43. The property reference `$[/increment xyz]` first increments the value of property xyz to 44, then returns 44.

timestamp

Use this form to generate a formatted timestamp value. For example, the property reference `$[/timestamp yyyy-MMM-d hh:mm]` returns the current time in a form such as "2007-Jun-19 04:36". The pattern following `/timestamp` specifies how to format the time and defaults to "yyyyMMddHHmm". The pattern follows conventions for the Java class SimpleDateFormat, where various letters are substituted with various current time elements. For more information about the SimpleDateFormat class, see http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html.
Here are some of the supported substitutions:

y—Year, such as "2007" or "07"

M—Month, such as "April", Apr", or "04"

w—Week in year, such as "27"

W—Week in month, such as "3"

D—Day in year, such as "194"

d—Day in month, such as "18"

E—Day in the week, such as "Tuesday" or "Tue"

a—am/pm marker, such as "PM"

H—Hour in day (0-23), such as "7"

h—Hour in am/pm (1-12), such as "11"

m—Minute in hour, such as "30"

s—Second in minute, such as "15"

S—Millisecond, such as "908"

z—Time zone such as "Pacific Standard Time", "PST", or "GMT-08:00"

Z—Time zone such as "-0800"

Repeated letters cause longer forms to be substituted. For example, "yy" substitutes the last 2 digits of the current year, whereas "yyyy" substitutes the 4-digit year number. Single quotes can be used to substitute text directly without the above interpretations.

javascript

This form executes a Javascript code fragment inside the ElectricFlow server and returns the result computed by that code.
For example, `$[/javascript 4*2]` returns "8.0". Javascript code can be arbitrarily long and include multiple statements. The value of the last statement is returned by the property reference.

Javascript code executes in an interpreter that provides access to ElectricFlow properties:

1. The normal way to access property values is through Javascript objects. Global Javascript objects named "server," "projects," and so on, exist and correspond to all top-level objects in an absolute property path.
   For example, there is a Javascript object named `server` that corresponds to the path `/server`, and a Javascript object named `myJob` that corresponds to the path `/myJob`. You can use either `"."` or `"[]"` notation to access properties within the object.
   For a more complete list of top-level objects, see Absolute property paths.

   Examples:

   ```
   $[/javascript myJob.mightExist]
   ```
   this is a safe way to refer to an optional parameter

   ```
   $[/javascript (myJobStep.errors > 0) ? "step failed" : "no errors"]
   ```
   test a value and return another string based on the result

   ```
   $[/javascript server.settings.ipAddress]
   ```
   refer to a property in a nested property sheet

   ```
   $[/javascript server["Electric Cloud"].installDirectory]
   ```
   if the property name has a space, use `[""]` notation

   ```
   $[/javascript server["Electric Cloud"]["installDirectory"]]
   ```

do not use the "." in front of the "["

2. You can also call the function, "getProperty". It takes a property name as argument and returns the value of that property. The case where this is most useful is when you want to access another special property reference.

   Examples:

   ```
   $[/javascript getProperty ("/timestamp yyyy-MMM-d hh:mm")]
   ```
   returns the current time

   ```
   $[/javascript getProperty ("/increment /myJob/jsCount")]
   ```
   this is the only way to change a property via javascript

   For example, consider the following property reference:

   ```
   $[/javascript (getProperty("/myJobStep/errors") > 0) ? "step failed" : "no errors"]
   ```
   This example returns "step failed" if the property "errors" on the current job step had a value greater than zero, and it returns "no errors" otherwise.

3. If calling the "setProperty" function, similar to getProperty, there are two variations on the function:

   - A *global* function variation that uses the current context object.
     The global function takes 2 or 3 arguments. The 3-argument version takes a context object, a path, and a value. The 2-argument version omits the context object and uses the current context object (for example, *job step*).

     Example:

     ```
     setProperty (myProject, "foo", "bar")
     ```

     would set the value of the "foo" property on the current project to "bar".

   - An *object* function variation that can be called on objects, which uses that object as a context object.
     The object function takes two arguments: a "path" and a "value".

     Example:

     ```
     server.setproperty ("foo", "bar")
     ```
     would set the "foo" property on the server object to the value "var".

Intrinsic properties listed by object type

ElectricFlow Deploy intrinsic properties

For details about the intrinsic properties for each object in ElectricFlow Deploy, see the *Intrinsic Properties in ElectricFlow Deploy Objects* document at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html.

Automation Platform intrinsic properties

This section describes the intrinsic properties for some (but not all) of the objects in the Automation Platform. For each object type in the following list, there is a link to a table that includes a list of intrinsic properties applicable to that object and provides the property type and description.

acl	repository
artifact	resource
artifactVersion	resourcePool
credential	resourceUsage
directoryProvider	schedule
emailConfig	server
emailNotifier	state
formalParameter	stateDefinition
gateways	step
group	transition
job	transitionDefinition
jobStep	user
logEntry	workflow
procedure	workflowDefinition
project	workspace
property	zones
propertySheet	

Property Type definitions in the Automation Platform

This table provides Property Type definitions for each property listed in the following tables. (A Property Type precedes each property description in the Description column.)

Type	Definition
boolean	One of two possible values—either *true* or *false.* These values are more frequently represented by the numbers "0" and "1", where "0" equals false and "1" equals true.
date	A millisecond precision UTC date in ISO 8601 form: `[YYYY]- [MM]-[DD]T[hh]:[mm]Z` For example, `2007-06-19T04:36:22.000Z`

Type	Definition
id	Each time an object is created, ElectricFlow generates a unique ID number for that object.
name	This is a unicode string value with a maximum of 255 characters.
number	This is a simple integer numeric value.
reference	This property refers to another object.
string	This is a unicode string value with a maximum CLOB size of the database, but only the first 450 characters are indexed, which means a defined search will not "see" beyond the first 450 characters.

In the following tables, the Description column displays the property type preceding the property description.

Object Type: acl

Description: An *acl* is an Access Control List.

Property Name	Description
aclId	id: The unique identifier for this acl object. Other objects can refer to this acl by its ID.
inheriting	boolean: If true, the ACL inherits ACEs from the ACL's parent.
ownerType	This is any type of object the ACL controls and can be any of the objects listed below. acl admin artifact artifactVersion event formalParameter group job jobStep license notifier procedure procedureStep project property propertySheet repository resource resourcePool server schedule systemObject user userSettings workspace plugin
parentId	id: The parent ACL.

Object Type: artifact

Description: An *artifact* is an object that contains zero or more artifact versions. An artifact has two purposes:

1. **To group artifact versions and provide a template for naming the versions**
2. **To restrict who can publish artifact versions, based on** `groupId:artifactKey`

Property Name	Description
`acl`	`reference:acl`
`artifactId`	`id`: The artifact's ID number.
`artifactKey`	`string`: User-specified identifier for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.
`artifactName`	`name`: The name of this artifact.
`artifactVersionNameTemplate`	`name`: The template for artifact version names published to this artifact.
`createTime`	`date`: The time when this object was created.
`description`	`string`: A user-specified text description of the object.
`groupId`	`id`: A user-generated group name for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.
`lastModifiedBy`	`name`: This shows who (generally, a username) last modified this object.
`modifyTime`	`date`: The time when this object was last modified.
`owner`	`name`: The person (username) who created the object.
`propertySheet`	`reference`: `propertySheet`

Object Type: artifactVersion

Description: An artifact version is an object that represents a user-defined unit of related files typically produced by one job and consumed by one or more other jobs.

Property Name	Description
acl	reference: acl
artifactKey	string: User-specified identifier for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.
artifactName	name: The name of the artifact.
artifactVersionId	id: The ElectricFlow-generated ID number for this artifact version.
artifactVersionName	name: The name of the artifact version.
artifactVersionState	string: Possible values are: available|publishing|unavailable
buildNumber	number: User-defined build number component of the version attribute for the artifact version.
createTime	date: The time when this object was created.
description	string: A user-specified text description of the object.
groupId	id: A user-generated group name for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
majorMinorPatch	string: major.minor.patch component of the version attribute for the artifact.
modifyTime	date: The time when this object was last modified.
owner	name: The person (username) who created the object.
propertySheet	reference: propertySheet

Object Type: artifactVersion

Description: An artifact version is an object that represents a user-defined unit of related files typically produced by one job and consumed by one or more other jobs.

Property Name	Description
publisherJobId	id: The ElectricFlow-generated ID number for the job that published the artifact version.
publisherJobName	string: The name of the job that published the artifact version.
publisherJobStepId	id: The ElectricFlow-generated ID number for the job step that published the artifact version.
qualifier	string: User-defined qualifier component of the version attribute for the artifact.
repositoryName	name: The name of the artifact repository.
version	string: An artifact version specification uses the following form: *major.minor.patch.qualifier.buildNumber*

Object Type: credential

Description: In ElectricFlow, a *credential* is an object that stores a username and password for later use.

Property Name	Description
acl	reference: acl
createTime	date: The time when this object was created.
credentialId	id: The credential's ID number.
credentialName	name: The name of this credential.
description	string: A user-specified text description of the object.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.

Object Type: credential

Description: In ElectricFlow, a *credential* is an object that stores a username and password for later use.

Property Name	Description
modifyTime	date: The time when this object was last modified.
owner	name: The person (username) who created the object.
password	string: The password corresponding to the username and this credential.
projectName	name: The name of the project that contains this credential.
propertySheet	reference: propertySheet
userName	name: A saved string that represents the name portion of a credential, typically a user account name.

Object Type: directoryProvider

Description: A directoryProvider is an object containing information about an external directory service (LDAP or ActiveDirectory).

Property Name	Description
acl	reference: acl
commonGroupNameAttribute	string: The attribute in a group record that contains the common group name. If specified, this name is used only when searching for groups from an external provider.
createTime	date: The time when this object was created.
description	string: A user-specified text description of the object.
directoryProviderId	id: The ID of the directory provider.
domainName	string: The domain name from which Active Directory servers are automatically discovered.

Object Type: directoryProvider

Description: A directoryProvider is an object containing information about an external directory service (LDAP or ActiveDirectory).

Property Name	Description
emailAttribute	string: The attribute in a user record that contains the user's email address. If the attribute was not specified, the account name and domain name are concatenated to form an email address.
enableGroups	boolean: Determines whether or not external groups are enabled for the directory provider. Defaults to "true".
fullUserNameAttribute	name: The attribute in a user record that contains the user's full name (first and last) for display in the UI.
groupBase	string: This string is prepended to the basedn to construct the directory DN that contains group records.
groupMemberAttributes	string: A comma-separated attribute name list that identifies a group member.
groupMemberFilter	string: Two common forms of group record in LDAP directories: POSIX style groups where members are identified by account name, and groupOfNames or uniqueGroupOfNames records where members are identified by the full user DN. Both forms are supported, so the query is passed to parameters: "{0}" is replaced with the full user record DN, and "{1}" is replaced with the user's account name.
groupNameAttribute	name: The group record that contains the name of the group.
groupSearchFilter	string: This LDAP query is performed in the context of the groups directory to enumerate group records.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
managerDn	name: The DN of a user who has read-only access to LDAP user and group directories.
modifyTime	date: The time when this object was last modified.

Object Type: directoryProvider

Description: A directoryProvider is an object containing information about an external directory service (LDAP or ActiveDirectory).

Property Name	Description
owner	name: The person (username) who created the object.
propertySheet	reference: propertySheet
providerIndex	number: The index that specifies the search order across multiple directory providers. For example: 2 LDAP providers, one with index "0" and one with index "1" means the providers will be searched in that numerical order.
providerName	name: This human-readable name will be displayed in the user interface to identify users and groups that come from this provider.
providerType	string: <ldap|activedirectory>
realm	string: An identifier (string)used for LDAP directory providers so users and groups (within LDAP) can be uniquely identified in "same name" collisions across multiple directory providers. The realm is appended to the user or group name when stored in the ElectricFlow server. For example, <user>@dir (where the realm is set to "dir").
url	string: The server URL is in the formprotocol://host:port/basedn. Protocol is either ldap or ldaps (for secure LDAP).
userBase	string: This string is prepended to the basedn to construct the directory DN that contains user records.
userNameAttribute	name: The attribute in a user record that contains the user's account name.
userSearchFilter	string: This LDAP query is performed in the context of the user directory to search for a user by account name. The string "{0}" is replaced with the user's login ID. Typically, the query compares a user record attribute with the substituted user login ID.

Object Type: directoryProvider

Description: A directoryProvider is an object containing information about an external directory service (LDAP or ActiveDirectory).

Property Name	Description
userSearchSubtree	boolean: If true, the subtree below the user base is searched recursively.
useSSL	boolean: This flag is used to specify SSL to communicate with your Active Directory servers. **Note:** Transport Layer Security (TLS) has replaced Secure Sockets Layer version 3.0 (SSLv3) on the ElectricFlow web server and the ElectricFlow server.

Object Type: emailConfig

Description: An *emailConfig* is an object that stores information created and used to communicate with the email server.

Property Name	Description
acl	reference: acl
configName	string: The name of the email configuration.
createTime	date: The time when this object was created.
description	string: A user-specified text description of the object.
emailConfigId	id: The ElectricFlow-generated ID for the email configuration.
emailConfigName	string: The name of the email configuration.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
mailFrom	string: The email address used as the email sender address for notifications.

Object Type: emailConfig

Description: An *emailConfig* is an object that stores information created and used to communicate with the email server.

Property Name	Description
mailHost	string: The name of the email server host.
mailPort	number: The port number for the mail server. The protocol software determines the default value (25 for SMTP and 465 for SSMTP).
mailProtocol	string: This is either SSMTP or SMTP (not case sensitive). Default is SMTP.
mailUser	name: An individual or a generic name like "ElectricFlow" -- the name of the email user on whose behalf ElectricFlow sends email notifications.
modifyTime	date: The time when this object was last modified.
owner	name: The person (username) who created the object.
propertySheet	reference: propertySheet

Object Type: emailNotifier

Description: An *emailNotifier* is an object that stores information created and used to notify users about various types information, including status.

Property Name	Description
acl	reference: acl
condition	string: Only send mail if the condition evaluates to "true". The condition is a string subject to property expansion. The notification will NOT be sent if the expanded string is "false" or "0". If no condition is specified, the notification is ALWAYS sent.
configName	string: The name of the email configuration.

Object Type: emailNotifier

Description: An *emailNotifier* is an object that stores information created and used to notify users about various types information, including status.

Property Name	Description
container	id: The object ID to which the email notifier is attached (for example: procedure-123).
containerType	string: The type of object that the notifier is attached to (procedure, step, job, jobstep).
createTime	date: The time when this object was created.
description	string: A user-specified text description of the object.
destinations	string: A space-separated list of valid email addresses, email aliases, or ElectricFlow usernames, or a string subject to property expansion that expands into such a list.
emailNotifierId	id: The ElectricFlow-generated ID for the email notifier.
eventType	string: <onEnter|onStart|onCompletion> onStart triggers an email notification when the job or job step begins. onCompletion , default, triggers an email notification when the job finishes, no matter how it finishes.
formattingTemplate	string: The email address used as the email sender address for notifications.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
modifyTime	date: The time when this object was last modified.
notifierName	name: The name of the email notifier.
owner	name: The person (username) who created the object.
propertySheet	reference: propertySheet

Object Type:formalParameter

Description: A *formalParameter* is a parameter expected by a procedure, including its name, a default value, and an indication of whether the parameter is required. Formal parameters are different from "actual parameters"--- formal parameters define the type of parameters a procedure is expecting, and actual parameters provide values to use at run-time.

Property Name	Description
container	string: An object ID for a "container" that contains formal parameters.
containerType	string: The type of object containing the formal parameter.
createTime	date: The time when this object was created.
defaultValue	string: The formalParameter's default value.
description	string: A user-specified text description of the object.
expansionDeferred	boolean: Determines whether or not the property expansion is deferred.
formalParameterId	id: This is this formal parameter's ID.
formalParameterName	name: This is this formal parameter's name.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
modifyTime	date: The time when this object was last modified.
owner	name: The person (username) who created the object.
required	boolean: If true, a value for this parameter must be supplied when the procedure is called.
type	name: The custom type of formalParameter.

Object Type:gateway

Description: To communicate with a resource, workspace, or artifact repository server in another zone, a "gateway" must be created. A gateway object contains two resource (agent) machines, for example, GatewayResource1 and GatewayResource2—each configured to communicate with the other. One gateway resource resides in the *source* zone and the other in the *target* zone. A gateway is bidirectional and informs the ElectricFlow server that each gateway machine is configured to communicate with its other gateway machine (in another zone).

Property Name	Description
acl	reference: acl
createTime	date: The time when this object was created.
description	string: A user-specified text description of the object.
gatewayDisabled	boolean: If set to 1 (true), the gateway is disabled.
gatewayId	id: The unique ElectricFlow-generated ID for this gateway.
gatewayName	string: The name of the gateway.
hostName1	string: The agent host name where *Resource1* resides. This host name is used by *Resource2* to communicate with *Resource1*. Do not specify this option is you want to use the host name from *Resource1's* definition.
hostName2	string: The agent host name where *Resource2* resides. This host name is used by *Resource1* to communicate with *Resource2*. Do not specify this option is you want to use the host name from *Resource2's* definition.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
modifyTime	date: The time when this object was last modified.
owner	name: The person (username) who created the object.
port1	number: The port number used by *Resource1*—defaults to the port number used by the resource.
port2	number: The port number used by *Resource2*—defaults to the port number used by the resource.

Object Type:gateway

Description: To communicate with a resource, workspace, or artifact repository server in another zone, a "gateway" must be created. A gateway object contains two resource (agent) machines, for example, GatewayResource1 and GatewayResource2—each configured to communicate with the other. One gateway resource resides in the *source* zone and the other in the *target* zone. A gateway is bidirectional and informs the ElectricFlow server that each gateway machine is configured to communicate with its other gateway machine (in another zone).

Property Name	Description
propertySheet	reference: propertySheet
resourceName1	string: The name of your choice for the first of two required gateway resources. Do **not** include "spaces" in a resource name.
resourceName2	string: The name of your choice for the second of two required gateway resources. Do **not** include "spaces" in a resource name.

Object Type:group

Description: This is any *group* of users known to the ElectricFlow server, including groups defined locally within the server and those groups defined in external repositories such as LDAP or ActiveDirectory.

Property Name	Description
acl	reference: acl
createTime	date: The time when this object was created.
groupId	id: The unique ElectricFlow-generated group ID.
groupName	name: This is this group's name.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
modifyTime	date: The time when this object was last modified.

Object Type: group

Description: This is any *group* of users known to the ElectricFlow server, including groups defined locally within the server and those groups defined in external repositories such as LDAP or ActiveDirectory.

Property Name	Description
mutable	boolean: If true, the member list of this group is editable within ElectricFlow via the web UI or the modifyGroup API.
owner	name: The person (username) who created the object.
propertySheet	reference: propertySheet
providerName	name: The name of the directory provider that controls this group.

Object Type: job

Description: A *job* is an ElectricFlow structure associated with invoking a procedure. A new job is created each time a procedure begins to execute. The job keeps track of all data associated with the procedure's execution, such as the running time of each step and any errors that may occur during the step. ElectricFlow retains job information after the job completes so you can examine what occurred.

Property Name	Description
abortStatus	string: If set, indicates the step was aborted. Values will be either ABORT or FORCE_ABORT—indicating how the step was aborted.
abortedBy	name: This is the user who issued the "abort."
acl	reference: acl

Object Type: job

Description: A *job* is an ElectricFlow structure associated with invoking a procedure. A new job is created each time a procedure begins to execute. The job keeps track of all data associated with the procedure's execution, such as the running time of each step and any errors that may occur during the step. ElectricFlow retains job information after the job completes so you can examine what occurred.

Property Name	Description
actualParameter	reference: propertySheet An actualParameter is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
callingState	string: The full property path to the "calling state", which can appear on subjobs and subworkflows of a workflow.
callingStateId	id: The ElectricFlow-generated ID number for the calling state.
combinedStatus	Provides more inclusive step status output—the resulting query output may contain up to three sub-elements: status|message|properties
createTime	date: The time when this object was created.
credentialName	name: The name of the credential being used for impersonation when the job runs commands on a resource.
deleted	The object was marked for background deletion. Possible values are "0" or "1". Default is "0" (not set).
directoryName	name: The name of this job's directory within each workspace for this job.
elapsedTime	number: The number of milliseconds between the start and end times for the job.
errorCode	errorCode: When the outcome is error, this property displays the error code, identifying which error occurred.

Object Type: job

Description: A *job* is an ElectricFlow structure associated with invoking a procedure. A new job is created each time a procedure begins to execute. The job keeps track of all data associated with the procedure's execution, such as the running time of each step and any errors that may occur during the step. ElectricFlow retains job information after the job completes so you can examine what occurred.

Property Name	Description
errorMessage	string: When the outcome is error, this property displays the error description.
external	boolean: If 'true", the job is external.
finish	date: The time this job completed.
jobId	id: This is this job's ID number, which is a UUID.
jobName	name: This is this job's name.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
launchedByUser	name: The name of the user or project principal that explicitly launched the job. This property is blank when the job is launched by a schedule.
licenseWaitTime	The sum of time all job steps had to wait for a license.
liveProcedure	name: The current procedure name from which this job was created – if the procedure was renamed since the job launched, this will be the procedure's new name, and if the procedure was deleted, this will be null.
liveSchedule	name: The current schedule name that launched this job – if the schedule was renamed since the job was launched, this will be the schedule's new name, and if the schedule was deleted, this will be null.
modifyTime	date: The time when this object was last modified.
outcome	The overall result of the job: success,warning, error, or skipped.

Object Type: job

Description: A *job* is an ElectricFlow structure associated with invoking a procedure. A new job is created each time a procedure begins to execute. The job keeps track of all data associated with the procedure's execution, such as the running time of each step and any errors that may occur during the step. ElectricFlow retains job information after the job completes so you can examine what occurred.

Property Name	Description
owner	name: The person (username) who created the object.
priority	Values can be low, normal (default), high, or highest.
procedureName	name: The name of the procedure that defines the job's steps.
projectName	name: The name of the project that contains this job.
propertySheetId	reference: propertySheet
resourceWaitTime	The sum of time all job steps had to wait for a resource. This could indicate that eligible resources for the step have reached their step limit, are in-use but the step requires a resource exclusively, or resources are down.
runAsUser	name: The name of the user being impersonated in this job.
scheduleName	name: The schedule name that launched this job – this field differs from liveSchedule in that it is written at job creation time only, and not changed, even if the schedule is renamed or deleted.
start	date: The time when this job began executing.
status	Possible values are: pending—the job was created, but it is waiting for prerequisite steps to complete runnable—the job step is waiting for a resource scheduled—the job was assigned a resource, but the command has not started running running—the job/job step is running a command on the assigned resource completed—the job/job step has completed

Object Type: job

Description: A *job* is an ElectricFlow structure associated with invoking a procedure. A new job is created each time a procedure begins to execute. The job keeps track of all data associated with the procedure's execution, such as the running time of each step and any errors that may occur during the step. ElectricFlow retains job information after the job completes so you can examine what occurred.

Property Name	Description
totalWaitTime	The total sum of license, resource, a precondition, and workspace wait times job steps were restricted and had to wait to process.
workspaceWaitTime	The sum of time all job steps had to wait for a workspace.

Object Type: jobStep

Description: A *jobStep* is an instance of a step that ran—a single invocation of a command on a resource or a single call to a subprocedure within the context of a job.

Property Name	Description
abortStatus	string: If set, indicates the step was aborted. Values will be either ABORT or FORCE_ABORT—indicating how the step was aborted.
abortedBy	name: The user who issued the "abort."
acl	reference: acl
actualParameter	reference: propertySheet An actualParameter is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
alwaysRun	boolean: If true, this step runs even if the job is aborted before the step completes. Note that a "force abort" will abort an alwaysRun step.

Object Type: jobStep

Description: A *jobStep* is an instance of a step that ran—a single invocation of a command on a resource or a single call to a subprocedure within the context of a job.

Property Name	Description
assignedResourceName	name: The name of the resource assigned to this step by the resource scheduler.
broadcast	boolean: If true, this step runs on all resources in a pool.
combinedStatus	Provides more inclusive step status output—the resulting query output may contain up to three sub-elements: status|message|properties
command	string: This property specifies the command this step runs.
condition	string: If this element is not present, the event is ALWAYS triggered. If specified, the event is triggered only if the value of the condition argument is TRUE; if not true, a boolean value of "false" or "0" was used. Condition arguments can be a literal, a fixed value string, or a string subject to property expansion.
conditionExpanded	boolean: The result of the expansion on the step condition.
createTime	date: The time when this object was created.
delayUntil	For a step that was rescheduled due to a resource or workspace problem, this is the next time the step will be eligible to run.
elapsedTime	number: The number of milliseconds between the start and end times for the jobStep.
errorCode	errorCode: This property appears when the outcome is error and displays the error code, identifying which error occurred.

Object Type: jobStep

Description: A *jobStep* is an instance of a step that ran—a single invocation of a command on a resource or a single call to a subprocedure within the context of a job.

Property Name	Description
errorHandling	This is the error handling policy copied from the procedure step and indicates how the step responds to errors: failProcedure —The current procedure continues, but the overall status is error (default). abortProcedure —Aborts the current procedure, but allows already-running steps in the current procedure to complete. abortProcedureNow —Aborts the current procedure and terminates running steps in the current procedure. abortJob —Aborts the entire job, terminates running steps, but allows alwaysRun steps to run. abortJobNow —Aborts the entire job and terminates all running steps, including alwaysRun steps. failProcedure — The current procedure continues, but the overall status is error (default). ignore —Continues as if the step succeeded.
errorMessage	string: When the outcome is error, this property displays an error message description.
exclusive	boolean: If true, this step acquires and retains its resource exclusively.
exclusiveMode	string: Possible values are: none, job, step, call
exitCode	number: This is this step's exit code.
external	boolean: If "true", the job is external.
finish	date: The time when this job step completed.
hostName	name: The name of the host where this step was invoked (copied from the resource)

Object Type: jobStep

Description: A *jobStep* is an instance of a step that ran—a single invocation of a command on a resource or a single call to a subprocedure within the context of a job.

Property Name	Description
job	id: This is the ID number of the job that owns the job step. The string representation of the job is its name, so $[job] evaluates to the job name, but $[job/foo] is also legal and refers to the foo property of the job.
jobId	id: This is this job's ID number, which is a UUID.
jobName	name: This is the name of this step's job.
jobStepId	id: This is this step's ID number.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
licenseWaitTime	The length of time this job step had to wait to process because the license limit was reached or exceeded.
liveProcedure	string: The current procedure name for the procedure step from which the job or job step was created – if the procedure step was renamed since the job or job step was launched, this is the procedure step's new name, and if the procedure step was deleted, this will be null.
liveProcedureStep	name: This property shows the current procedure name for the procedure step from which this job step was created – if the procedure step was renamed since the job was launched, this will be the procedure step's new name, and if the procedure step was deleted, this will be null.
logFileName	name: The name of the log file produced by this step, relative to the job's workspace directory.
modifyTime	date: The time when this object was last modified.
outcome	string: The overall result of the job—possible values are: success, warning, error, or skipped

Object Type: jobStep

Description: A *jobStep* is an instance of a step that ran—a single invocation of a command on a resource or a single call to a subprocedure within the context of a job.

Property Name	Description
owner	name: The person (username) who created the object.
parallel	boolean: If true,this step runs in parallel with other adjacent steps also marked to run in parallel.
postExitCode	number: This is this step's post processor exit code.
postLogFileName	name: The log file name produced by this step's post processor.
postProcessor	string: The post processor name used to gather information about this step.
precondition	string: By default, if a step has no precondition, it will run when scheduled. Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a *precondition* is evaluated. A *precondition* is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a \"0\" or \"false\" is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.
procedureName	name: The name of the procedure that contains this jobStep.
projectName	name: The name of the project that contains this jobStep.
propertySheet	reference: propertySheet
releaseExclusive	boolean: A Boolean value indicating whether this step should release its resource upon completion.
releaseMode	string: Possible values are: none, release, releaseToJob
resourceName	name: The name of the resource or pool this step should use to run on.

Object Type: jobStep

Description: A *jobStep* is an instance of a step that ran—a single invocation of a command on a resource or a single call to a subprocedure within the context of a job.

Property Name	Description
resourceSource	This property indicates whether the resource for the job step is explicitly specified by the step or was defaulted from the procedure: procedure or procedureStep.
resourceWaitTime	The length of time this job step stalled because it could not get a resource. This could indicate that eligible resources for the step have reached their step limit, are in-use but the step requires a resource exclusively, or resources are down.
retries	number: This is a number—the number of retries before the request times out.
runAsUser	name: The name of the user being impersonated in this job step.
runnable	date: The time when the step became runnable.
runTime	number: The number of milliseconds the step command spent running on a resource.
shell	name: The shell used to execute the step's commands on a resource. The script name is inserted into the command at the position of a "{0}" marker in the command, or appended as a final argument if no marker is present. A filename suffix adjacent to the marker will be appended to the script name. For more information on shells, see the Shell definition in the Step Help topic.
start	date: The time when this job step began executing.

Object Type: jobStep

Description: A *jobStep* is an instance of a step that ran—a single invocation of a command on a resource or a single call to a subprocedure within the context of a job.

Property Name	Description
status	Possible values are: pending —the job step was created, but it is waiting for prerequisite steps to complete runnable—the job step is waiting for a resource scheduled—the job step was assigned a resource, but the command has not started running running—the job/job step is running a command on the assigned resource completed—the job/job step has completed
stepName	name: This is this step's name.
subprocedure	name: The nested procedure name to call when this step executes.
subproject	name: This property appears if the subprocedure element is present and specifies the project to which the subprocedure belongs (by default, the current project is used.)
timeLimit	number: The maximum length of time this step is allowed to run.
timeout	date: The time when this job step will be automatically aborted if it has not yet completed.
totalWaitTime	The sum of resource, workspace, and license wait times for this job step.
waitTime	number: The number of milliseconds the step spent between runnable and running (for example, waiting for a resource).
workingDirectory	name: The name of this step's working directory.
workspaceName	name: The workspace name used by the jobStep object.
workspaceWaitTime	The time this job step had to wait because no workspace was found or available.

Object Type: logEntry

Description: A *logEntry* is an object containing various types of information available in a log file generated from anywhere in the ElectricFlow system.

The *Event Log* can be configured for auto-deletion. By default, Event Log retention is 30 days. The ElectricFlow server automatically marks old log entries for deletion and the background deleter cleans out old log entries.

To change the default settings, go to Administration > Server > Settings and modify the Event log retain time and the Maximum background delete delay fields

Note: Setting the "retain" period to "0" disables the automatic deletion mechanism, allowing you to use an external cleanup script to implement a custom cleanup policy.

Property Name	Description
`category`	(currently not used)
`container`	`string`: Typically, this is the type and name of the workflow or job with a corresponding ID.
`containerName`	`name`: The name of the container.
`containerType`	`string`: The type of object the log entry pertains to (for example, procedure, job step, step).
`deleted`	`byte`: The object was marked for background deletion. Possible values are `"0"` or `"1"`. Default is `"0"` (which means "deleted" is not set).
`logEntryId`	`id`: The log entry ElectricFlow-generated ID.
`message`	`string`: The message text will either be informational or display the warning or error message. The message may contain important information about a resource or workspace issue.
`principal`	`string`: The user or project principal from the session that was active when the event occurred.
`severity`	`string`: Severity can be either `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`
`subject`	`string`: The object associated with the message.

Object Type: logEntry

Description: A *logEntry* is an object containing various types of information available in a log file generated from anywhere in the ElectricFlow system.

The *Event Log* can be configured for auto-deletion. By default, Event Log retention is 30 days. The ElectricFlow server automatically marks old log entries for deletion and the background deleter cleans out old log entries.

To change the default settings, go to Administration > Server > Settings and modify the Event log retain time and the Maximum background delete delay fields

Note: Setting the "retain" period to "0" disables the automatic deletion mechanism, allowing you to use an external cleanup script to implement a custom cleanup policy.

Property Name	Description
subjectName	name: The name of the object associated with the message.
subjectType	string: (similar to container) Refers to the object the event concerns. This may be the same as the container, or it may be a different object that is related to the event in some manner.
time	string: The time the event was logged.

Object Type:procedure

Description: A *procedure* describes a series of actions to be performed on one or more resources. A procedure contains steps and properties and can define formal parameters.

Property Name	Description
acl	reference: acl
createTime	date: The time when this object was created.
credentialName	name: The name of the credential assigned to this job step.
description	string: A user-specified text description of the object.
jobNameTemplate	name: The template used to name jobs created from this procedure.

Object Type:procedure

Description: A *procedure* describes a series of actions to be performed on one or more resources. A procedure contains steps and properties and can define formal parameters.

Property Name	Description
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
modifyTime	date: The time when this object was last modified.
owner	name: The person (username) who created the object.
procedureId	id: This is this procedure's ID number.
procedureName	name: This is this procedure's name.
projectName	name: The name of the project that contains this procedure.
propertySheet	reference: propertySheet
resourceName	name: The name of the resource or pool this procedure should use to run on.
workspaceName	name: The workspace name used by the procedure object.

Object Type:project

Description: A *project* is an object used in ElectricFlow to organize information. A project contains procedures, schedules, credentials, and properties.

Property Name	Description
acl	reference: acl
createTime	date: The time when this object was created.
credentialName	name: The name of the credential assigned to this project.
deleted	The object was marked for background deletion. Possible values are "0" or "1". Default is "0" (not set).

Object Type: project

Description: A *project* is an object used in ElectricFlow to organize information. A project contains procedures, schedules, credentials, and properties.

Property Name	Description
description	string: A user-specified text description of the object.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
modifyTime	date: The time when this object was last modified.
owner	name: The person (username) who created the object.
pluginName	name: The name of the plugin associated with this project.
projectId	id: This is this project's ID.
projectName	name: The name of the project.
propertySheet	reference: propertySheet
resourceName	name: The name of the resource.
workspaceName	name: This is the workspace name used by the project.

Object Type: property

Description: A *property* is a string value with a name. Properties can have arbitrary names and values. You can attach properties to any object in the ElectricFlow system, such as a project, procedure, or job. After a property is created, it is stored in the ElectricFlow database. You can retrieve and/or modify the value later, and you can delete properties you no longer need. Properties provide a flexible and powerful mechanism to manage data about your builds.

> **Note: The names "properties" and "project" are not valid property names.**

Property Name	Description
createTime	date: The time when this object was created.
description	string: A user-specified text description of the object.

Object Type:property

Description: A *property* is a string value with a name. Properties can have arbitrary names and values. You can attach properties to any object in the ElectricFlow system, such as a project, procedure, or job. After a property is created, it is stored in the ElectricFlow database. You can retrieve and/or modify the value later, and you can delete properties you no longer need. Properties provide a flexible and powerful mechanism to manage data about your builds.

> **Note: The names "properties" and "project" are not valid property names.**

Property Name	Description
expandable	boolean: If set to true, the property value will undergo string expansion when it is retrieved.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
modifyTime	date: The time when this object was last modified.
owner	name: The person (username) who created the object.
canonicalPath	string: The path that specifies the property object.
propertyId	id: This property's ID number.
propertyName	name: This property's name.
propertySheetId	reference: propertySheet—If the property is a nested property sheet, the propertySheet refers to the nested sheet.
value	string: The property or actual parameter's value—if this property is a string property.

Object Type: propertySheet

Description: A property value can be a simple string or a nested *propertySheet* containing its own properties. Property sheets can be nested to any depth, which allows you to create hierarchical collections of information.
Most objects have an associated property sheet that contains "custom properties" created by user scripts.

Property Name	Description
acl	reference: acl
createTime	date: The time when this object was created.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
modifyTime	date: The time when this object was last modified.
owner	name: The person (username) who created the object.
propertySheetId	id: This property sheet's ID.

Object Type: repository

Description: A *repository* is an object that stores artifact versions. This object primarily contains information on how to connect to a particular artifact repository. Similar to steps in a procedure, repository objects are in a user-specified order. When retrieving artifact versions, repositories are queried in this order until one containing the desired artifact version is found.

Property Name	Description
acl	reference: acl
createTime	date: The time when this object was created.
description	string: A user-specified text description for this object.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
modifyTime	date: The time when this object was last modified.

Object Type:repository

Description: A *repository* is an object that stores artifact versions. This object primarily contains information on how to connect to a particular artifact repository. Similar to steps in a procedure, repository objects are in a user-specified order. When retrieving artifact versions, repositories are queried in this order until one containing the desired artifact version is found.

Property Name	Description
owner	name: The person (username) who created the object.
propertySheet	reference: propertySheet
repositoryDisabled	boolean: Determines whether the repository is disabled. Default is "false".
repositoryId	id: The ElectricFlow-generated ID number of the repository.
repositoryIndex	number: The order of the repository, within a list of repositories.
repositoryName	name: The name of the repository.
url	string: The server URL is in the form protocol://host:port/. Typically, the repository server is configured to listen on port 8200 for https requests, so a typical URL looks like https://host:8200/.
zoneName	name: The name of the zone where this repository is or will reside.

Object Type:resource

Description: A *resource* is associated with a server machine available to ElectricFlow for running steps. A resource has a logical name in addition to a host name. A resource can be associated with one or more pools. The resource may be a "proxy target" machine that communicates with the ElectricFlow server though an ElectricFlow agent proxy machine.

Property Name	Description
acl	reference: acl

Object Type:resource

Description: A *resource* is associated with a server machine available to ElectricFlow for running steps. A resource has a logical name in addition to a host name. A resource can be associated with one or more pools. The resource may be a "proxy target" machine that communicates with the ElectricFlow server though an ElectricFlow agent proxy machine.

Property Name	Description
agentState	string: Specific information about an agent, including the state of the agent. Possible values are: unknown|alive|down
artifactCacheDirectory	string: The directory on the agent host where retrieved artifacts are stored.
createTime	date: The time when this object was created.
description	string: A user-specified text description of the object.
exclusiveJobId	id: The ID number of the job that owns this resource, which occurs when one of the job's steps requests exclusive use of the resource for the duration of the job.
exclusiveJobName	name: The name of the job that owns this resource, which occurs when one of the job's steps requests exclusive use of the resource for the duration of the job.
exclusiveJobStepId	id: The ID number of the job step that owns this resource, which occurs when one of the steps request exclusive use of the resource for the duration of the job.
exclusiveJobStepName	name: The name of the job step that owns this resource, which occurs when one of the steps request exclusive use of the resource for the duration of the job.
gateways	The gateway names associated with this resource.
hostName	name: The computer name or IP address for the machine containing the ElectricFlow agent for this resource.
hostOS	string: The full name of the host operating system, plus its version. However, if this host is a proxy, "proxied" appears as the host description/name.

Object Type:resource

Description: A *resource* is associated with a server machine available to ElectricFlow for running steps. A resource has a logical name in addition to a host name. A resource can be associated with one or more pools. The resource may be a "proxy target" machine that communicates with the ElectricFlow server though an ElectricFlow agent proxy machine.

Property Name	Description
hostPlatform	string: Examples for "platform" are: Windows, Linux, HPUX, and so on. However, if this host is a proxy, "proxied" appears as the hostPlatform name.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
lastRunTime	date: The most recent time a job step ran on the resource.
modifyTime	date: The time when this object was last modified.
owner	name: The person (username) who created the object.
pools	string: A space-separated list of one or more pool names where this resource is a member. Steps defined to run on a resource pool will run on any available member (resource) in the pool.
port	number: The port number for this resource.
propertySheet	reference: propertySheet
proxyCustomization	string: This property displays the customization of the ElectricFlow proxy agent.
proxyHostName	name: The name of the ElectricFlow agent being used to proxy to another agent, the proxy target.
proxyPort	number: The port number of the ElectricFlow agent being used to proxy to another agent, the proxy target.
proxyProtocol	name: The name of the proxy agent protocol used for communication from the ElectricFlow proxy agent to the proxy target—default is SSH.
repositoryNames	A "new line" separated list of repository names.

Object Type:resource

Description: A *resource* is associated with a server machine available to ElectricFlow for running steps. A resource has a logical name in addition to a host name. A resource can be associated with one or more pools. The resource may be a "proxy target" machine that communicates with the ElectricFlow server though an ElectricFlow agent proxy machine.

Property Name	Description
resourceDisabled	boolean: A Boolean value indicating whether this resource was disabled.
resourceId	id: This is this resource's ID.
resourceName	name: The name of the resource or pool.
shell	name: The shell used to execute the step's commands on a resource. If no shell was defined on a step, the shell defined on the resource is used, or if no shell was defined on the resource, a default shell is used. For shells: The script name is inserted into the command at the position of a "{0}" marker in the command, or appended as a final argument if no marker is present. A filename suffix adjacent to the marker will be appended to the script name. For more information on shells, see the Shell definition in the Step Help topic.
stepCount	number: The current number of executing steps on this resource.
stepLimit	number: This property specifies the maximum number of steps that can run on this resource at one time.
trusted	boolean: If "true", this agent is trusted.
useSSL	boolean: A Boolean value indicating whether this resource uses SSL for communication. Transport Layer Security (TLS) has replaced Secure Sockets Layer version 3.0 (SSLv3) on the ElectricFlow web server and the ElectricFlow server.
workspaceName	name: The workspace name used by this resource.
zoneName	sting: The name of the zone where this resource resides.

Object Type:resourcePool

Description: A *resource pool* is a container for a group of resources.

Property Name	Description
acl	reference: acl
autoDelete	boolean: If "true", the resource pool is deleted when the last resource is removed or deleted.
createTime	date: The time when this object was created.
description	string: A user-specified text description of the object.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
lastResourceUsed	name: The name of the most recently used resource from the pool.
modifyTime	date: The time when this object was last modified.
orderingFilter	string: A Javascript block invoked when scheduling resources for a pool. A Javascript block is not required unless you need to override the default resource ordering behavior.
owner	name: The person (username) who created the object.
propertySheet	reference: propertySheet
resourcePoolDisabled	boolean: A Boolean value indicating whether this resource pool was disabled.
resourcePoolId	id: This resource pool's ID number.
resourcePoolName	name: The name of the resource pool.

Object Type:resourceUsage

Description: Provides usage information for all resources in the system. For any job step running on a resource, there is a resource usage record that contains the ID and name of the job, job step, and resource.

Property Name	Description
jobId	id: This is this job's ID number, which is a UUID.
jobName	name: The name of the job.
jobStepId	id: The ID number of the job step.
jobStepName	name: The name of the job step.
licenseWaitTime	The time this job step had to wait because no license was found or available.
resourceId	id: The ElectricFlow-generated resource ID number.
resourceName	name: The name (or names) of the resource.
resourcePoolId	id: The ElectricFlow-generated resource pool's ID number.
resourcePoolName	name: The name of the resource pool.
resourceUsageId	id: The unique ID of the resource usage record.
resourceWaitTime	The time this job step had to wait because no resource was found or available. This could indicate that eligible resources for the step have reached their step limit, are in-use but the step requires a resource exclusively, or resources are down.
waitReason	Possible values are: license, resource, or workspace.
workspaceWaitTime	The time this job step had to wait because no workspace was found or available.

Object Type:schedule

Description:*Schedules* are used to execute procedures and determine when specific procedures run.

Property Name	Description
acl	reference: acl
actualParameter	reference: propertySheet An *actualParameter* is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
beginDate	string: The first day on which the schedule is active, in the form YYYY-MM-dd. For example, "2009-06-25"
createTime	date: The time when this object was created.
credentialName	name: The name of the credential being used for impersonation when the schedule launches a job.
description	string: A user-specified text description of the object.
endDate	string: The end of the range of dates the schedule is active, in the form: YYYY-MM-dd. The actual end date is not included in the range. For example, "2009-06-25"
interval	string: A floating point number that represents the time to wait between invocations of the schedule.
intervalUnits	string: These are the units to use when interpreting the *interval* value. The *units* can be hours, minutes, seconds, or continuous.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
lastRunTime	date: The last time a job was launched by a schedule.

Object Type:schedule

Description:*Schedules* are used to execute procedures and determine when specific procedures run.

Property Name	Description
misfirePolicy	string: The policy the schedule uses when it is unable to launch a job at the scheduled time: ignore—wait until the next scheduled time to run runOnce—run immediately, then resume normal scheduling
modifyTime	date: The time when this object was last modified.
monthDays	string: This property specifies which days of the month this schedule should run—shown as a space-separated list of numbers (1-31).
owner	name: The person (username) who created the object.
priority	string: Values can be low, normal (default), high, or highest
procedureName	name: The name of the procedure that contains this schedule.
projectName	name: The name of the project that contains this schedule.
propertySheet	reference: propertySheet
scheduleDisabled	boolean: A Boolean value indicating whether this schedule was disabled.
scheduleId	id: This schedule's ID number.
scheduleName	name: This property displays this schedule's name and differs from the liveSchedule property found under a job in that the liveSchedule property is written at job creation time only, and not changed, even if the schedule is renamed or deleted.
startTime	string: The time of day when this schedule should start running, in the form: HH:mm:ss
stopTime	string: The time of day when this schedule should stop running, in the form: HH:mm:ss
timeZone	string: A Java-compatible time zone string.

Object Type:schedule

Description:*Schedules* are used to execute procedures and determine when specific procedures run.

Property Name	Description
weekDays	string: This property specifies which days of the week this schedule should run. This is a space-separated list of MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY SATURDAY SUNDAY. (Note: These are not localized strings.)

Object Type:server

Description: This is the ElectricFlow server. There is only one such object in your database.

Read-only server properties
Two read-only server properties are available for use in a step, a script, or an email notifier, which can be used to identify the ElectricFlow server location.

`/server/hostName` —returns the ElectricFlow server name

`/server/hostIP` —returns the ElectricFlow server IP address

These properties are especially useful for building a URL link to an ElectricFlow web page. For example, a link in an email notifier that links back to a Job Details page.
A sample link: `https://$[/server/hostName]/commander/jobDetails.php?jobId=$[jobId]`

Property Name	Description
acl	reference: acl
createTime	date: The time when this object was created.
hostIP	string: The server's IP address.
hostName	name: Generally refers to the computer name or IP address for the machine containing the ElectricFlow server.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
modifyTime	date: The time when this object was last modified.

Object Type:server

Description: This is the ElectricFlow server. There is only one such object in your database.

Read-only server properties
Two read-only server properties are available for use in a step, a script, or an email notifier, which can be used to identify the ElectricFlow server location.

`/server/hostName` —returns the ElectricFlow server name

`/server/hostIP` —returns the ElectricFlow server IP address

These properties are especially useful for building a URL link to an ElectricFlow web page. For example, a link in an email notifier that links back to a Job Details page.
A sample link: `https://$[/server/hostName]/commander/jobDetails.php?jobId=$[jobId]`

Property Name	Description
owner	name: The person (username) who created the object.
propertySheet	reference: propertySheet
serverId	id: The server's ID number.
serverName	string: The ElectricFlow server's name—defaults to "server".

Object Type:state

Description: A *state* object belongs to a workflow and corresponds to the state definition, including a pointer to the workflow, formal parameters (cloned from definition), expanded actual parameters (form the last time a transition was taken to this state), and the "process" which includes the procedure or workflow (cloned from definition), unexpanded actual parameters (cloned from definition, expanded and passed to the process on entry), and the "last run" entity reference.

Property Name	Description
acl	reference: acl
active	boolean: If "true", the state of the workflow is active.

Object Type:state

Description: A *state* object belongs to a workflow and corresponds to the state definition, including a pointer to the workflow, formal parameters (cloned from definition), expanded actual parameters (form the last time a transition was taken to this state), and the "process" which includes the procedure or workflow (cloned from definition), unexpanded actual parameters (cloned from definition, expanded and passed to the process on entry), and the "last run" entity reference.

Property Name	Description
actualParameter	reference: propertySheet An actualParameter is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
createTime	date: The time when this object was created.
description	string: A user-specified text description of the object.
errorMessage	string: When the outcome is error, this property displays an error message description.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
modifyTime	date: The time when this object was last modified.
owner	name: The person (username) who created the object.
projectName	name: The name of the project that contains this state.
propertySheet	reference: propertySheet
stateId	id: The ElectricFlow-generated ID number for the state.
stateName	name: The name of the state.
subjob	name: The name of the subjob.
subparameters	reference: propertySheet

Object Type:state

Description: A *state* object belongs to a workflow and corresponds to the state definition, including a pointer to the workflow, formal parameters (cloned from definition), expanded actual parameters (form the last time a transition was taken to this state), and the "process" which includes the procedure or workflow (cloned from definition), unexpanded actual parameters (cloned from definition, expanded and passed to the process on entry), and the "last run" entity reference.

Property Name	Description
subprocedure	name: The name of the nested procedure called when a step runs.
subproject	string: If a subprocedure argument was used, this is the name of the project where that subprocedure is found. By default, the current project is used.
substartingState	name: Name of the starting state for the workflow launched when the state is entered.
subworkflow	name: The name of the subworkflow—a workflow called by another workflow.
subworkflowDefinition	name: The name of the subworkflow definition.
workflow	name: The name of the workflow.

Object Type:stateDefinition

Description: A *stateDefinition* is a named object that belongs to a workflow definition, which includes specifications for whether or not the state is "startable", formal parameters, notifications, and the process. A process includes a procedure or workflow, the starting state (for workflows only), and actual parameters.

Property Name	Description
acl	reference: acl

Object Type:stateDefinition

Description: A *stateDefinition* is a named object that belongs to a workflow definition, which includes specifications for whether or not the state is "startable", formal parameters, notifications, and the process. A process includes a procedure or workflow, the starting state (for workflows only), and actual parameters.

Property Name	Description
actualParameter	reference: propertySheet An *actualParameter* is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
createTime	date: The time when this object was created.
description	string: A user-specified text description of the object.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
modifyTime	date: The time when this object was last modified.
owner	name: The person (username) who created the object.
projectName	name: The name of the project that contains this state definition.
propertySheet	reference: propertySheet
startable	boolean: "True" means this state definition can be the initial state of an instantiated workflow.
stateDefinitionId	id: The ElectricFlow-generated ID number for the state definition.
stateDefinitionName	name: The name of the state definition.
subprocedure	name: The name of the nested procedure called when a step runs.

Object Type:stateDefinition

Description: A *stateDefinition* is a named object that belongs to a workflow definition, which includes specifications for whether or not the state is "startable", formal parameters, notifications, and the process. A process includes a procedure or workflow, the starting state (for workflows only), and actual parameters.

Property Name	Description
subproject	string: If a subprocedure argument was used, this is the name of the project where that subprocedure is found. By default, the current project is used.
substartingState	name: Name of the starting state for the workflow launched when the state is entered.
subworkflowDefinition	name: The name of the subworkflow definition.
workflowDefinitionName	name: The name of the workflow definition.

Object Type:step

Description: A *step* includes a command or script executed on a single resource and is the smallest unit of work ElectricFlow understands. Each step specifies a resource on which it should run (either the name of a specific machine or the name of a *pool* of equivalent machines, in which case ElectricFlow picks a machine from the pool). A step can be given a time limit and if the step does not complete within the specified time, ElectricFlow automatically aborts it.

Steps are ordered within a procedure and normally execute sequentially. However, it is possible to mark a consecutive range of steps for parallel execution, so all steps in that range run concurrently.

A step in one procedure can invoke another procedure instead of running a command. The invoking step provides parameters needed by the nested procedure.

Property Name	Description
acl	reference: acl

Object Type:step

Description: A *step* includes a command or script executed on a single resource and is the smallest unit of work ElectricFlow understands. Each step specifies a resource on which it should run (either the name of a specific machine or the name of a *pool* of equivalent machines, in which case ElectricFlow picks a machine from the pool). A step can be given a time limit and if the step does not complete within the specified time, ElectricFlow automatically aborts it.

Steps are ordered within a procedure and normally execute sequentially. However, it is possible to mark a consecutive range of steps for parallel execution, so all steps in that range run concurrently.

A step in one procedure can invoke another procedure instead of running a command. The invoking step provides parameters needed by the nested procedure.

Property Name	Description
actualParameter	reference: propertySheet An actualParameter is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
alwaysRun	boolean: If true, this step runs even if the job is aborted before the step completes. Note that a force abort will abort an alwaysRun step.
broadcast	boolean: If true, this step will run on all resources in a pool.
command	string: This property specifies the command this step runs.
condition	string: If this element is not present, the event is ALWAYS triggered. If specified, the event is triggered only if the value of the condition argument is TRUE; if not true, a boolean value of "false" or "0" was used. Condition arguments can be a literal, a fixed value string, or a string subject to property expansion.
createTime	date: The time when this object was created.
credentialName	name: The credential name assigned to this step.
description	string: A user-specified text description of the object.

Object Type:step

Description: A *step* includes a command or script executed on a single resource and is the smallest unit of work ElectricFlow understands. Each step specifies a resource on which it should run (either the name of a specific machine or the name of a *pool* of equivalent machines, in which case ElectricFlow picks a machine from the pool). A step can be given a time limit and if the step does not complete within the specified time, ElectricFlow automatically aborts it.

Steps are ordered within a procedure and normally execute sequentially. However, it is possible to mark a consecutive range of steps for parallel execution, so all steps in that range run concurrently.

A step in one procedure can invoke another procedure instead of running a command. The invoking step provides parameters needed by the nested procedure.

Property Name	Description
errorHandling	This is the error handling policy copied from the procedure step and indicates how the step responds to errors: failProcedure —The current procedure continues, but the overall status is error (default). abortProcedure —Aborts the current procedure, but allows already-running steps in the current procedure to complete. abortProcedureNow —Aborts the current procedure and terminates running steps in the current procedure. abortJob —Aborts the entire job, terminates running steps, but allows alwaysRun steps to run. abortJobNow —Aborts the entire job and terminates all running steps, including alwaysRun steps. ignore —Continues as if the step succeeded.
exclusive	boolean: If true, this step acquires and retains its resource exclusively.
exclusiveMode	string: Possible values are: none, job, step, call
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
logFileName	name: The name of the log file produced by this step, relative to the job's workspace directory.
modifyTime	date: The time when this object was last modified.

Object Type:step

Description: A *step* includes a command or script executed on a single resource and is the smallest unit of work ElectricFlow understands. Each step specifies a resource on which it should run (either the name of a specific machine or the name of a *pool* of equivalent machines, in which case ElectricFlow picks a machine from the pool). A step can be given a time limit and if the step does not complete within the specified time, ElectricFlow automatically aborts it.

Steps are ordered within a procedure and normally execute sequentially. However, it is possible to mark a consecutive range of steps for parallel execution, so all steps in that range run concurrently.

A step in one procedure can invoke another procedure instead of running a command. The invoking step provides parameters needed by the nested procedure.

Property Name	Description
owner	name: The person (username) who created the object.
parallel	boolean: If true, this step runs in parallel with other adjacent steps also marked to run in parallel.
postLogFileName	name: This property displays the log file name produced by this step's post processor.
postProcessor	string: This property displays the post processor name that is used to gather information about this step. Typically, this is either postp or postp <options>, or an appropriate command or path including options to a postprocessor available on the ElectricFlow server.
precondition	string: By default, if a step has no precondition, it will run when scheduled. Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a *precondition* is evaluated. A *precondition* is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a \"0\" or \"false\" is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.
procedureName	name: The name of the procedure that contains this step.
projectName	name: The name of the project that contains this step.

Object Type:step

Description: A *step* includes a command or script executed on a single resource and is the smallest unit of work ElectricFlow understands. Each step specifies a resource on which it should run (either the name of a specific machine or the name of a *pool* of equivalent machines, in which case ElectricFlow picks a machine from the pool). A step can be given a time limit and if the step does not complete within the specified time, ElectricFlow automatically aborts it.

Steps are ordered within a procedure and normally execute sequentially. However, it is possible to mark a consecutive range of steps for parallel execution, so all steps in that range run concurrently.

A step in one procedure can invoke another procedure instead of running a command. The invoking step provides parameters needed by the nested procedure.

Property Name	Description
propertySheet	reference: propertySheet
releaseExclusive	boolean: A Boolean value indicating whether this step should release its resource upon completion.
releaseMode	string: Possible values are: none, release, releaseToJob
resourceName	name: The resource's name this step should use to run on.
shell	name: The shell used to execute the step's commands on a resource. The script name is inserted into the command at the position of a "{0}" marker in the command, or appended as a final argument if no marker is present. A filename suffix adjacent to the marker will be appended to the script name. For more information on shells, see the Shell definition in the Step Help topic.
stepId	id: The step's ID number.
stepName	name: The step's name.
subprocedure	name: The nested procedure name to call when this step executes.
subproject	name: This property is displayed if the subprocedure element is present and specifies the project to which the subprocedure belongs (by default, the current project is used.)

Object Type:step

Description: A *step* includes a command or script executed on a single resource and is the smallest unit of work ElectricFlow understands. Each step specifies a resource on which it should run (either the name of a specific machine or the name of a *pool* of equivalent machines, in which case ElectricFlow picks a machine from the pool). A step can be given a time limit and if the step does not complete within the specified time, ElectricFlow automatically aborts it.

Steps are ordered within a procedure and normally execute sequentially. However, it is possible to mark a consecutive range of steps for parallel execution, so all steps in that range run concurrently.

A step in one procedure can invoke another procedure instead of running a command. The invoking step provides parameters needed by the nested procedure.

Property Name	Description
timeLimit	number: A floating point number that specifies the maximum length of time this step is allowed to run.
timeLimitUnits	The units to use when interpreting the time limit. Units can be hours, minutes, or seconds.
workingDirectory	name: The name of the step's working directory.
workspaceName	name: The workspace name used by this step.

Object Type: transition

Description: A *transition* object belongs to a state and corresponds to the transition definition, and includes the target state (unexpanded actual parameters cloned from definition, expanded and passed to the target state on entry), the Javascript condition, and the trigger (onEnter\|onStart\|onCompletion\|manual).

Property Name	Description
acl	reference: acl
actualParameter	reference: propertySheet An *actualParameter* is an object that provides the value for a parameter that is passed to the target state when the transition is taken.
condition	string: If empty or non-zero, the transition is allowed to trigger. If set to "0", the transition is ignored.

Object Type: transition

Description: A *transition* object belongs to a state and corresponds to the transition definition, and includes the target state (unexpanded actual parameters cloned from definition, expanded and passed to the target state on entry), the Javascript condition, and the trigger (`onEnter|onStart|onCompletion|manual`).

Property Name	Description
createTime	date: The time when this object was created.
description	string: A user-specified text description of the object.
index	The numeric index of a transition that indicates the transition order in the state definition's transition list.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
modifyTime	date: The time when this object was last modified.
owner	name: The person (username) who created the object.
projectName	name: The name of the project that contains this transition.
propertySheet	reference: propertySheet
stateName	name: The name of the state.
targetState	string: The target state for the transition definition.
transitionId	id: The ElectricFlow-generated ID for the transition.
transitionName	name: The name of the transition.
trigger	string: Possible values are: onEnter –before any actions onStart –after a subjob or workflow is created onCompletion –after a subjob or workflow completes manual –when a user manually requests a transition
workflowName	name: The name of the workflow.

Object Type: **transitionDefinition**

Description: **A *transitionDefinition* is a named object that belongs to a state definition, which includes the target state definition (with actual parameters to the target state, the Javascript condition, and the trigger (`onEnter|onStart|onCompletion|manual`).**

Property Name	Description
acl	`reference`: `acl`
actualParameter	`reference`: `propertySheet` An *actualParameter* is an object that provides the value for a parameter passed to the target state.
condition	`string`: If empty or non-zero, the transition is allowed to trigger. If set to "0", the transition is ignored.
createTime	`date`: The time when this object was created.
description	`string`: A user-specified text description of the object.
index	The numeric index of a transition that indicates the transition order in the state definition's transition list.
lastModifiedBy	`name`: This shows who (generally, a username) last modified this object.
modifyTime	`date`: The time when this object was last modified.
owner	`name`: The person (username) who created the object.
projectName	`name`: The name of the project that contains this transition definition.
propertySheet	`reference`: `propertySheet`
stateDefinitionName	`name`: The name of the state definition.
targetState	`string`: The target state for the transition definition.
transitionDefinitionId	`id`: The ElectricFlow-generated ID for the transition definition.
transitionDefinitionName	`name`:The name of the transition definition.

Object Type: transitionDefinition

Description: A *transitionDefinition* is a named object that belongs to a state definition, which includes the target state definition (with actual parameters to the target state, the Javascript condition, and the trigger (`onEnter|onStart|onCompletion|manual`).

Property Name	Description
`trigger`	`string`: Possible values are: `onEnter` –before any actions `onStart` –after a subjob or workflow is created `onCompletion` –after a subjob or workflow completes `manual` –when a user manually requests a transition
`workflowDefinitionName`	`name`: The name of the workflow definition.

Object Type: user

Description: *User* refers to any user currently logged into the ElectricFlow system or any user who may use ElectricFlow, but may not be currently logged in.

Property Name	Description
`acl`	`reference`: `acl`
`createTime`	`date`: The time when this object was created.
`email`	`name`: Displays this user's email address.
`fullUserName`	`name`: The user's real name.
`lastModifiedBy`	`name`: This shows who (generally, a username) last modified this object.
`modifyTime`	`date`: The time when this object was last modified.
`mutable`	`boolean`: If true, the user is editable within ElectricFlow via the web UI or the `modifyUser` API.
`owner`	`name`: The person (username) who created the object.
`propertySheet`	`reference`: `propertySheet`

Object Type:user

Description:*User* refers to any user currently logged into the ElectricFlow system or any user who may use ElectricFlow, but may not be currently logged in.

Property Name	Description
providerName	name: The name of the directory provider that controls this user.
userId	id: A unique ID number for a user object.
userName	name: Displays this user's name and also appears in a group object and displays the name of a user who belongs to this group.

Object Type:workflow

Description: A *workflow* object includes a pointer to the workflow definition, sets of states, the active state, the starting stare, parameters to the initial state, "complete"—a boolean value determining whether or not the workflow is complete, and a log containing the transactions taken and user events.

Property Name	Description
acl	reference: acl
activeState	string: The name of the active state on the workflow object.
actualParameter	reference: propertySheet An *actualParameter* is an object that provides the value for a parameter that is passed to the target state when the transition is taken.
callingState	string: The full property path to the state that created this workflow.
callingStateId	id: The ID number for the full property path to the state that created this workflow.
completed	boolean: If "true", the workflow is completed and no additional transactions will be evaluated.
createTime	date: The time when this object was created.

Object Type:workflow

Description: A *workflow* object includes a pointer to the workflow definition, sets of states, the active state, the starting stare, parameters to the initial state, "complete"—a boolean value determining whether or not the workflow is complete, and a log containing the transactions taken and user events.

Property Name	Description
deleted	boolean: The object was marked for background deletion. Possible values are "0" or "1". Default is "0" (not set).
elapsedTime	The time this object ran from start to finish.
finish	date: The date/time when this object finished.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
launchedByUser	string: The name of the user or project principal that explicitly launched the workflow.
liveWorkflowDefinition	name: The current workflow definition name for the workflow definition from which the workflow was created.
modifyTime	date: The time when this object was last modified.
owner	name: The person (username) who created the object.
projectName	name: The name of the project that contains this workflow.
propertySheet	reference: propertySheet
start	date: The date/time when this workflow began to run.
startingState	string: The initial state of the workflow.
workflowDefinitionName	name: The name of the workflow definition.
workflowId	id: The ElectricFlow-generated ID for the workflow.
workflowName	name: The name of the workflow.

Object Type:workflowDefinition

Description: A *workflowDefinition* object contains state and transition definitions, including the workflow name template (analogous to a job name template on a procedure), and ordered sets of state and transition definitions.

Property Name	Description
acl	reference: acl
createTime	date: The time when this object was created.
description	string: A user-specified text description of the object.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
modifyTime	date: The time when this object was last modified.
owner	name: The person (username) who created the object.
projectName	name: The name of the project that contains this workflow definition.
propertySheet	reference: propertySheet
workflowDefinitionId	id: The ElectricFlow-generated ID for the workflow definition.
workflowDefinitionName	name: The name of the workflow definition.
workflowNameTemplate	string: Template used to determine the default names for workflows launched from a workflow definition.

Object Type: workspace

Description: ElectricFlow provides each job step with an area on the disk it can use for "working files" and results. This disk area is called a job *workspace*. (The job workspace defaults to a directory under the workspace directory.) A job step can create whatever files it needs within its workspace, and ElectricFlow automatically places files such as step logs in the workspace.

Normally, a single workspace is shared by all steps in a job, but different steps within a job could use different workspaces. The location of the job step workspace is displayed on the Job Details page for the job under "Details" for the step.

Property Name	Description
acl	reference: acl
agentDrivePath	name: The drive-letter based mount point for this workspace on Windows agents.
agentUncPath	name: The UNC path for this workspace on Windows agents.
agentUnixPath	name: The UNIX path for this workspace on UNIX agents.
createTime	date: The time when this object was created.
credentialName	name: The name of the credential being used when a resource connects to a network share while setting up the workspace.
description	string: A user-specified text description of the object.
lastModifiedBy	name: This shows who (generally, a username) last modified this object.
local	boolean: If "true", this workspace is local.
modifyTime	date: The time when this object was last modified.
owner	name: The person (username) who created the object.
propertySheet	reference: propertySheet
workspaceDisabled	boolean: A Boolean value that indicates whether this workspace was disabled.
workspaceId	id: The workspace's ID number.

Object Type: workspace

Description: ElectricFlow provides each job step with an area on the disk it can use for "working files" and results. This disk area is called a job *workspace*. (The job workspace defaults to a directory under the workspace directory.) A job step can create whatever files it needs within its workspace, and ElectricFlow automatically places files such as step logs in the workspace.

Normally, a single workspace is shared by all steps in a job, but different steps within a job could use different workspaces. The location of the job step workspace is displayed on the Job Details page for the job under "Details" for the step.

Property Name	Description
workspaceName	name: The workspace's name.
zoneName	name: The name of the zone where this workspace resides.

Object Type: zone

Description: ElectricFlow provides the ability to create zones—often used to enhance security.

- **A zone is a way to partition a collection of agents to secure them from use by other groups.**
  For example, you might choose to create a developers zone, a production zone, and a test zone—agents in one zone cannot directly communicate with agents in another zone.
- **A *default* zone is created during ElectricFlow installation.**
  The server implicitly belongs to the default zone, which means all agents in this zone can communicate with the server directly (without the use of a gateway).
- **Each zone can have one or more "gateway agents", which you define.**
  Gateway agents are used for communication from one zone to another zone.
  For more information, see the Gateways Help topic.

Property Name	Description
acl	reference: acl
createTime	date: The time when this object was created.
description	string: A user-specified text description for the object.

Object Type:zone

Description: ElectricFlow provides the ability to create zones—often used to enhance security.

- **A zone is a way to partition a collection of agents to secure them from use by other groups.**
  For example, you might choose to create a developers zone, a production zone, and a test zone—agents in one zone cannot directly communicate with agents in another zone.
- **A *default* zone is created during ElectricFlow installation.**
  The server implicitly belongs to the default zone, which means all agents in this zone can communicate with the server directly (without the use of a gateway).
- **Each zone can have one or more "gateway agents", which you define. Gateway agents are used for communication from one zone to another zone. For more information, see the Gateways Help topic.**

Property Name	Description
lastModifiedBy	name: This person (generally, a username) last modified this object.
modifyTime	date: The time when this object was last modified.
owner	name: The person (username) who created the object.
propertySheet	reference: propertySheet
resources	string: A space-separated list of resources contained in this zone.
zoneId	id: The ElectricFlow-generated ID for this zone.
zoneName	string: The name of the zone.

Property error codes

The following list provides error codes that may appear as a value within an errorCode property.

ABORTED	AGENT_UNKNOWN_COMMAND
ACCESS_DENIED	AGENT_WRONG_FILE_TYPE
AGENT_BAD_WORKINGDIR	BAD_AGENT_RESPONSE

AGENT_FAILED_CONNECT_ BROKER	BAD_LOGFILE_PATH
AGENT_FAILED_CREATE_FILE	CANCELED
AGENT_FAILED_CREATE_ WORKSPACE	CIRCULAR_PROCEDURE_REFERENCE
AGENT_FAILED_IMPERSONATION	CORRUPT_CREDENTIAL
AGENT_FAILED_MAP_DRIVE	DUPLICATE_JOB_NAME
AGENT_FAILED_PROXYTARGET_ PING	EMPTY_SUBPROCEDURE
AGENT_INCOMPATIBLE_VERSION	FAILED_JOB_RENAME
AGENT_INTERNAL_ERROR	FORMAL_PARAMETER_ERROR
AGENT_INVALID_CWD	INTERNAL_ERROR
AGENT_INVALID_MESSAGE	INVALID_EMAIL_HEADER
AGENT_INVALID_PING_TOKEN	MY_EVENT_EXPANSION_ERROR
AGENT_INVALID_WORKSPACE	NO_EMAIL_HEADERS_SEPARATOR
AGENT_IO_CONNECTION_ REFUSED	NONEXISTENT_EMAIL_CONFIG
AGENT_IO_CONNECTION_RESET	NONEXISTENT_PROCEDURE
AGENT_IO_ERROR	NONEXISTENT_RESOURCE
AGENT_IO_NO_ROUTE_TO_HOST	NONEXISTENT_WORKSPACE
AGENT_IO_PORT_UNREACHABLE	NOTIFIER_EXPANSION_ERROR
AGENT_MALFORMED_XML	POST_PROCESSOR_ERROR
AGENT_NONABSOLUTE_PATH	POST_PROCESSOR_OUTPUT_FAILURE
AGENT_NONEXISTENT_DIR	PROPERTY_REFERENCE_ERROR
AGENT_NONEXISTENT_FILE	RESOURCE_WITHOUT_HOSTNAME
AGENT_PING_FAILED	SERVER_SHUTDOWN
AGENT_STREAM_STOPPED	TIMEOUT
AGENT_TIMEOUT	UNKNOWN_HOST

AGENT_UNKNOWN_CMDID	AGENT_UNKNOWN_COMMAND
	AGENT_WRONG_FILE_TYPE
AGENT_UNKNOWN_COMMAND	BAD_AGENT_RESPONSE
AGENT_WRONG_FILE_TYPE	BAD_LOGFILE_PATH
BAD_AGENT_RESPONSE	CANCELED
BAD_LOGFILE_PATH	CIRCULAR_PROCEDURE_REFERENCE
CANCELED	CORRUPT_CREDENTIAL
CIRCULAR_PROCEDURE_ REFERENCE	DUPLICATE_JOB_NAME
CORRUPT_CREDENTIAL	EMPTY_SUBPROCEDURE
DUPLICATE_JOB_NAME	FAILED_JOB_RENAME
EMPTY_SUBPROCEDURE	FORMAL_PARAMETER_ERROR
FAILED_JOB_RENAME	INTERNAL_ERROR
FORMAL_PARAMETER_ERROR	INVALID_EMAIL_HEADER
INTERNAL_ERROR	MY_EVENT_EXPANSION_ERROR
INVALID_EMAIL_HEADER	NO_EMAIL_HEADERS_SEPARATOR
MY_EVENT_EXPANSION_ERROR	NONEXISTENT_EMAIL_CONFIG
NO_EMAIL_HEADERS_SEPARATOR	NONEXISTENT_PROCEDURE
NONEXISTENT_EMAIL_CONFIG	NONEXISTENT_RESOURCE
NONEXISTENT_PROCEDURE	NONEXISTENT_WORKSPACE
NONEXISTENT_RESOURCE	NOTIFIER_EXPANSION_ERROR
NONEXISTENT_WORKSPACE	POST_PROCESSOR_ERROR
NOTIFIER_EXPANSION_ERROR	POST_PROCESSOR_OUTPUT_FAILURE
POST_PROCESSOR_ERROR	PROPERTY_REFERENCE_ERROR
POST_PROCESSOR_OUTPUT_ FAILURE	RESOURCE_WITHOUT_HOSTNAME

PROPERTY_REFERENCE_ERROR	SERVER_SHUTDOWN
RESOURCE_WITHOUT_HOSTNAME	TIMEOUT
SERVER_SHUTDOWN	UNKNOWN_HOST
TIMEOUT	AGENT_UNKNOWN_COMMAND
UNKNOWN_HOST	

Postprocessors: Collecting Data for Reports

Overview

In ElectricFlow, reporting is divided into two phases.

- The first phase is data collection: interesting information is extracted from job step logs and saved in the ElectricFlow database.
- The second phase is report generation: data collected previously is retrieved and organized into reports.

The report phases are separated so data is gathered once but then available to use in a variety of different reports either immediately or later. For example, one report might summarize errors within a particular job, and another report might display error trends from all jobs over the past month.

ElectricFlow implements data collection with a *postprocessor*.

- A postprocessor is a command associated with a particular procedure step. If a postprocessor is specified for a step, it executes concurrently with the main step command.
- The postprocessor runs on the same machine as the main command and in the same working directory, and it retrieves the log file from the step as its standard input.

ElectricFlow includes a standard postprocessor called *postp* that you can use and extend. *postp* scans the step's log file looking for interesting output such as error messages and then sets properties on the job step to describe what it found. For example, *postp* might create a property named "errors" whose value is a count of the number of error messages in the log file, or a property named "tests" that counts the number of tests executed by the step. Also, *postp* can extract portions of the step log that contain useful diagnostic information and save this information for reporting.

Standard ElectricFlow reports, such as those on the Job Details page, display information collected by *postp* such as properties named "errors" and diagnostic log extracts. This information is available immediately, even before the step completes execution, so you can view it via ElectricFlow's web interface to monitor step execution. Also, you can create additional report generators of your own, which can use the same information displayed by ElectricFlow and/or any other additional information of your choice.

Postp

ElectricFlow's general-purpose postprocessor, *postp*, uses regular expression patterns to detect interesting lines in a step log.

- *postp* is already configured with patterns to handle many common cases such as error messages and warnings from *gcc*, *gmake*, *cl*, *junit*, and *cppunit*, or any error message containing the string "error." You can use the empty matcher group named "none" if just want to run a postpEndHook.
- *postp* is easy to use by simply setting the postprocessor for a step to "`postp`."
- *postp* also supports several useful command-line options.

To explore these options, invoke "`postp --help`" from your command-line.

Extending *postp*: matchers

If you find useful patterns in your log files undetected by postp, you can extend postp with additional patterns. This feature is easily implemented, but the extension interface currently involves writing simple Perl scripts and you need to look at the Perl source file for postp as you do this. The postp source code is installed during ElectricFlow installation and located in the `src/postp.pl` file in the distribution directory.

Postp is driven by a collection of *matchers*, which are regular expression patterns that select certain lines from step logs, and by a collection of Perl functions the matchers invoke to handle lines of interest. A matcher is a Perl hash with three values similar to the following example:

```
{
   id      => "error",
   pattern => q{ERROR:|[Ee]rror:},
   action  => q{
      incValue("errors"); diagnostic("", "error", -4)
   },
}
```

Explanation of the values in the Perl hash example above:

- **`id`**—A unique name for the matcher—used to identify the matcher in command-line arguments and a few other places.
- **`pattern`**—A regular expression tested against each line of the step log file. This particular pattern matches lines containing any of the strings "ERROR:", "Error:" or "error:"
- **`action`**—A Perl script that executes whenever a log line matches the pattern for this matcher. The script in this example increments a variable named "errors" that is copied automatically to a job step property with the same name. The script also saves a portion of the step log beginning 4 lines prior and extending through the current line and associates those lines with this error so it can be displayed in the web interface. See below for more information on the `incValue` and `diagnostic` functions.

To extend postp with patterns of your own, write a Perl script to add new patterns to the `@::gMatchers` array. Here is a simple example:

```
my @newMatchers = (
    {
      id      => "coreDump",
      pattern => "core dumped",
      action  => q{
```

```
                    incValue("coreDumps")
                }
    },
    {
      id      => "segFault",
      pattern => "segmentation fault",
      action  => q{
                    incValue("segFaults")
                }
    },
);
push @::gMatchers, @newMatchers;
```

These matchers detect lines containing the strings "core dumped" or "segmentation fault" and increment separate variables for each line type.

After writing extension code, you must ask postp to execute the code when it starts up. You can execute extension code in one of two ways:

- Place the code into a file and invoke postp with the `--load` option.
  For example, `postp --load fileName`
- Copy the code into a property in ElectricFlow and use the `--loadProperty` option to postp.
  For example, `postp --loadProperty /myProject/extraMatchers`

Postp extensions can contain arbitrary Perl code, which means you can use this mechanism to define additional functions to invoke in matcher actions if existing functions do not provide what you need.

> **Tip:** Additional postp matchers are available for your convenience. The matcher sample directory was installed during ElectricFlow installation. Go to `src/samples/postp`.

Postp functions

Postp contains several built-in functions to invoke in your matchers. The most useful functions are summarized below. Also, you can scan postp code for additional functions.

debugLog(format, arg, arg, ...)

Outputs information to the debug log, if the `--debugLog` command-line switch is set. Format provides a format string similar to `printf`, and each argument provides a value to substitute into the format string.

backTo(pattern, start)

This function searches backward to find the first line in the step log matching "pattern" (a regular expression) and returns the offset of that line relative to the current line. The result is normally used as an argument to the "diagnostic" function. "Start" is optional; it specifies the first line to check and is specified as an offset relative to the current line (it defaults to -1).

backWhile(pattern, start)

This function searches backward to find the first line in the step log that does not match "pattern" (a regular expression) and returns the offset relative to the current line of the line just after the first non-matching line. The result is normally used as an argument to the "diagnostic" function. "Start" is optional; it specifies the first line to check and is specified as an offset relative to the current line (it defaults to -1).

currentModule()

Returns the name of the current module (as determined by previous calls to pushModule and popModule), or an empty string if there is no current module.

diagnostic(name, type, first, last)

This function extracts a group of contiguous lines from the log file and saves them along with additional information for reports such as the log extracts, which appear at the bottom of the Job Details web page. The range of lines to extract is indicated by "first" and "last," each of which is an offset relative to the current line. For example, if "first" is -3 and "last" is 2, then 6 lines will be recorded: 3 lines before the current line, the current line, and 2 lines after the current line. In many cases, the values for "first" and "last" are computed by calling functions such as "forwardTo" or "backWhile." "Name" provides an identifier for this particular diagnostic, such as the name of a test that failed or a file that did not compile. "Type" specifies which kind of information this is, and must be "error," "warning," or "info." In addition to log lines, this function records "name," "type," the current module, if any, and the name of the current matcher.

forwardTo(pattern, start)

This function searches forward to find the first line in the step log matching "pattern" (a regular expression) and returns the offset of that line relative to the current line. The result is normally used as an argument to the "diagnostic" function. "Start" is optional; it specifies the first line to check and is specified as an offset relative to the current line (it defaults to 1).

forwardWhile(pattern, start)

This function searches forward to find the first line in the step log that does not match "pattern" (a regular expression) and returns the offset relative to the current line of the line just before the first non-matching line. The result is normally used as an argument to the "diagnostic" function. "Start" is optional; it specifies the first line to check and is specified as an offset relative to the current line (it defaults to 1).

incValue(name, increment)

Adds "increment" to a value named "name" and arranges for that value to be written eventually to a property by the same name on the current job step. If this is the first call for "name," its value is initialized to 0. "Increment" is optional and defaults to 1.

> **Note:** postp does not check the job step for a pre-existing property with the same name; it simply overwrites it.

logLine(lineNumber)

Returns the line from the step log given by lineNumber. 1 corresponds to the first line in the step log and the Perl variable `$::gCurrentLine` holds the number of the current line. This function caches a sliding window of lines in the file, allowing you to go back to retrieve lines preceding the current line (as long as they do not precede it by too many lines). If the requested line is off the end of the file then `undef` is returned. If the requested line is before the beginning window of cached lines, an empty string is returned.

popModule()

Cancels the effect of the most recent call to pushModule, resetting the current module name to whatever it was before the corresponding call to pushModule.

postpEndHook()

If you define a function with this name, it invokes after postp has finished processing the log file, but before it makes its final properties update on the job step. Use this function to perform your own operations such as generating an error if the log file did not contain a particular line you were expecting.

pushModule(name)

In some situations it is possible to divide the log file into parts corresponding to different modules. For example, with recursive make invocations, there are typically notifications in the log output before and after each recursive make. This function is invoked to indicate a new module is being entered, where "name" is the name of the module. After this function is called, the "diagnostic" function will include "name" with error or warning messages to provide additional information in job reports. The previous module name, if any, is saved; you can return to it by calling popModule.

setProperty

This function sets a property in the ElectricFlow server. If the named parameter is a relative path, like `moduleCount`, the property is set or created on the current job step. You can use an absolute path, like `/myJob/fileLocation` also. Calling `setProperty` does not result in an immediate call to the ElectricFlow server. The property is added to an update list for updating at the next "update interval", typically every 30 seconds.

Integration with the ElectricFlow user interface

`postp` interacts with the ElectricFlowUI using two methods. The first method: Create custom properties with special names that are recognized by the UI itself. The second method: Create a file that contains "diagnostics", which are used to display errors or warnings, and link them to specific sections in the step's log file.

Custom property names and values

`postp` can be used to create properties in ElectricFlow, on the job step or anywhere else in ElectricFlow. However, the following properties are used by the standard ElectricFlow UI, so you should use these property names whenever possible, and avoid using these names in ways that conflict with the definitions below.

- **`compiles`**—the number of files compiled during the job step
- **`diagFile`**—the filename in the top-level directory of the job's workspace, containing diagnostics extracted from the step's log file
- **`errors`**—the number of errors (compilation failures, test failures, application crashes, and so on) that occurred during the job step. When property errors are set by postp, the step outcome is set to error also.
- **`tests`**—the number of tests executed by the job step, including successes and failures
- **`testsSkipped`**—the number of tests skipped during the job step
- **`warnings`**—the number of warnings that occurred during the job step. When property warnings is set by postp, the step outcome is set to warning also.
- **`preSummary`**—if this property exists, its value is displayed in the "Status" field (on the Job Details page) for this step. This property appears *before* whatever would normally be displayed for status. If the property contains multiple lines separated by newline characters, each line is displayed on a separate line in the status field.

- **postSummary**—if this property exists, its value is displayed in the "Status" field (on the Job Details page) for this step. This property appears *after* whatever would normally be displayed for status. If the property contains multiple lines separated by newline characters, each line is displayed on a separate line in the status field.
- **summary**—if this property exists, its value is displayed in the "Status" field (in the job reports) for this step, replacing whatever would normally be displayed for status. If the property contains multiple lines separated by newline characters, each line is displayed on a separate line in the status field.

Diagnostic information

The second form of postprocessor generated output contains diagnostic extracts from the step's log file, typically providing additional information about problems. The postprocessor stores diagnostic information in an XML file in the top-level directory of the job workspace, then sets the step's **diagFile** property with the name of the file. Diagnostic files must have a format like the following example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<diagnostics>
    <diagnostic>
        <matcher>compileError</matcher>
        <name>testa.c</name>
        <type>error</type>
        <module>util/timeLib</module>
        <firstLine>2</firstLine>
        <numLines>7</numLines>
        <message>testa.c: In function 'proc1':
           testa.c:12: error: parse error before ';' token
           testa.c:13: error: 'for' loop initial declaration used outside C99 mode
           testa.c:14: error: parse error before ';' token
           testa.c:16: error: too few arguments to function 'exit'
           testa.c:18:2: warning: no newline at end of file
           testa.c:18: error: parse error at end of input
        </message>
    </diagnostic>
    <diagnostic>
        <name>testa.o</name>
        <type>error</type>
        <module>util/timeLib</module>
        <firstLine>9</firstLine>
        <numLines>1</numLines>
        <message>make: *** [testa.o] Error 1 </message>
    </diagnostic>
</diagnostics>
```

XML elements in the diagnostic file are as follows:

- `diagnostics`—Overall container; its children consist of all of the diagnostic elements.
- `diagnostic`—Describes one diagnostic extract; the elements described below are all children of this element.
- `matcher`—(optional) Identifier for the matcher that triggered this diagnostic; used primarily for debugging.
- `name`—(optional) Identifier that indicates the problem or situation that resulted in this diagnostic, such as the name of a failed test or the name of a file that did not compile.

- `type`—Type of message: must be "error," "warning," or "info."
- `module`—(optional) Name of the module in which the issue occurred, such as the name of a source code module being compiled at the time of a compile error.
- `firstLine`—Line number in the log file for the first line of the diagnostic extract (1 means the first line of the file). This element is used to provide a link from the diagnostic extract to the full log file.
- `numLines`—Total number of lines included in the diagnostic extract.
- `message`—The actual lines from the log file.

Postp integration with Java Tools

ElectricFlow integrates with three standard Java tools through postp matchers. These Java tools are:

- EMMA—an open source toolkit for measuring and reporting Java code coverage (Emma v2.0)
- JUnit—a framework for writing and running automated tests (tested with Ant v1.7)
- Clover—Atlassian's Java code coverage (generated by system, functional, or unit tests) analysis tool (Clover v2.4)

> **Note:** If one of these Java tools is invoked in an ElectricFlow job step, ElectricFlow automatically detects the invocation (if you are using Postp) and adds any reports generated by these tools to the list of links at the top of the Job Details page.

The postp Process

Postp parses output from the invoked JAVA tool to match paths to reports it has already created. Then, postp copies all files that comprise the report to a unique location in the "Artifacts" Directory. Next, postp generates a link to the location in the Artifacts Directory to make the report available in the Links section on the Job Details page.

The following example of generated output from Emma illustrates this process:

```
emma:

init:
[mkdir] Created dir: C:\Documents and Settings\ptharani\Desktop\emma\emma-
2.0.5312\examples\out

compile:
[javac] Compiling 4 source files to C:\Documents and
Settings\ptharani\Desktop\emma\emma-2.0.5312\examples\out

run:
[emmajava] EMMA: processing classpath ...
[emmajava] EMMA: [3 class(es) processed in 47 ms]
[emmajava] main(): running doSearch()...
[emmajava] main(): done
[emmajava] EMMA: writing [txt] report to [C:\Documents and
Settings\ptharani\Desktop\emma\emma-2.0.5312\examples\coverage\coverage.txt] ...
[emmajava] EMMA: writing [html] report to [C:\Documents and
Settings\ptharani\Desktop\emma\emma-2.0.5312\examples\coverage\coverage.html] ...

all:
```

```
BUILD SUCCESSFUL
Total time: 1 second
```

From this output

The actual link may reside here:

```
/home/commanderWorkspace/job_112_
200901081732/artifacts/javaTools/238/emmaCoverage/2/coverage.html
```

While the link name may be:

```
Step Id 238 ant-on-the-fly—emma report# 2
```

And the value of the link may be:

```
jobSteps/238//javaTools/238/emmaCoverage/2/coverage.html
```

Java Tool matcher examples

Two examples of postp Emma matchers:

```
 {
      id      => "emmaReport1",
      pattern => q{EMMA: writing},
      action  => q{emmaExtractReport()},

},
 {
       id     => "emmaReport2",
      pattern => q{\[report\] writing},
       action => q{emmaValidateOutput()},
},
```

An example of postp JUnit matchers:

```
{
      id      => "junitReportCapture",
      pattern => q{\[junitreport\] Processing},
      action  => q{junitExtractReport ()},
 },
```

These matchers correspond to the following output:

```
junit.report:
[junitreport] Processing
/net/WinStor2home/ptharani/junitDemo1/sample/testreport/TESTS-TestSuites.xml to
/tmp/null1214791178
[junitreport] Loading stylesheet jar:[file:/usr/local/tools/common/apache-ant-
1.7.0/lib/ant-junit.jar\!/org/apache/tools/ant/taskdefs/optional/junit/xsl/junit-
frames.xsl
[junitreport] [file:/usr/local/tools/common/apache-ant-1.7.0/lib/ant-
junit.jar!/org/apache/tools/ant/taskdefs/optional/junit/xsl/junit-frames.xsl
[junitreport]] Transform time: 622ms
[junitreport] Deleting: /tmp/null1214791178
```

An example of postp Clover matchers:

```
{
     id      => "cloverHtmlReportAntTask",
     pattern => q{\[clover-html-report\] Writing HTML report to},
     action  => q{cloverExtractReport()},
 },
```

These matchers correspond to the following output:

```
clover.report:
[clover-html-report] Clover Version 2.4.0, built on November 05 2008 (build-747)
[clover-html-report] Loaded from: /home/cloverDemo/clover-ant-2.4.0/lib/clover.jar
[clover-html-report] Clover: Evaluation License registered to electric cloud.
[clover-html-report] You have 27 day(s) before your license expires.
[clover-html-report] Loading coverage database from: '/home/cloverDemo/clover-ant-
2.4.0/tutorial/.clover/clover2_4_0.db'
[clover-html-report] Writing HTML report to '/home/cloverDemo/clover-ant-
2.4.0/tutorial/clover_html'
[clover-html-report] Done. Processed 1 packages in 2559ms (2559ms per package).
```

Artifacts directory

The value of the artifacts directory determines the scope of what is visible in the ElectricFlow UI from a job's workspace. By default, the Artifacts Directory is set to "artifacts", so the artifacts directory would have the form:

```
<path to job workspace>/artifacts
```

The value of the artifacts directory can be set in any of the following properties:

```
/myJob/artifactsDirectory
```

```
/myProject/artifactsDirectory
```

```
/server/settings/artifactsDirectory
```

Postp queries these properties [in the order listed] to determine the location of the Artifacts Directory. If no value is found (because the property was never set), the default "artifacts" is used.

Postp has a feature where it recognizes that data belongs to some standard tool (such as Junit), copies the logs produced by that tool to the artifacts directory, and then creates the report-url properties. These actions are done with the junitReportCapture matcher.

If you want to disable this matcher in your postp invocation in the step that runs junit tests, do the following:

```
postp --dontCheck junitReportCapture
```

It is possible that the artifacts directory will still be created even if nothing is put in it. If that occurs, set the artifactsDirectory property on your job (or the owning project) to empty-string.

Reports

ElectricFlow provides multiple reports and custom report capabilities to help you manage your build environment.

- Real-time reports—filtered view of your workload in real-time
- Build reports—summary reports produced at the end of a build and attached to the job
- Batch reports—summaries of your build environment with trends over time, two types:

- Default Batch reports—automatically installed during ElectricFlow installation and scheduled to run daily
  (Cross Project Summary, Variant Trend, Daily Summary, Resource Summary, Resource Detail)
- Optional Batch reports—you can configure, rename, and schedule these reports to fit your requirements
  (Category Report, Procedure Usage Report, Count Over Time Report, Multiple Series Reports)

- Custom reports—your choice to create and add at any time

You can use also StatsD and Graphite to generate custom reports. For more information, see System Health Monitoring on page 1570.

Batch and Custom reports must be run on the ElectricFlow server's agent (local agent) only. These reports need the BIRT report engine, which is installed on the ElectricFlow server.

Run reports on a non-local resource

If you want to configure reports to run on a non-local resource, do the following:

1. In the web interface, go to Projects > Electric Cloud > runReports > runEcrptdata.
2. Change the Parameter Resource from the default value "local" to the name of a different resource. the resource must have access to the plugins directory.

Real-time reports

Home page

The Home page is a user-customizable dashboard that can be configured to show you just the ElectricFlow jobs you care about.

The Jobs Quick View section (on the right side of the Home page) shows jobs you define using filters on job properties. You can create multiple categories filtered on specific project names, procedure names, users, or any other job property. Summary details for the job "pop-up" if you hover your mouse over one of the jobs.

Jobs

Unlike the Home page, which allows you to see specific subsets of work, the Jobs page shows all jobs that are running and all jobs that have completed. This is a good report to view when you need to see a list of all jobs run by any user or schedule. System administrators and managers can use this page to monitor overall system throughput.

Job details / Step details

After a job starts, you can get execution details from the Job Details page. To view job details, click on the job from the Home page or the Jobs tab. From within the Job Details page you can drill down to see details for an individual step. This is the finest grain of reporting, showing the detailed results of a specific job.

Resources

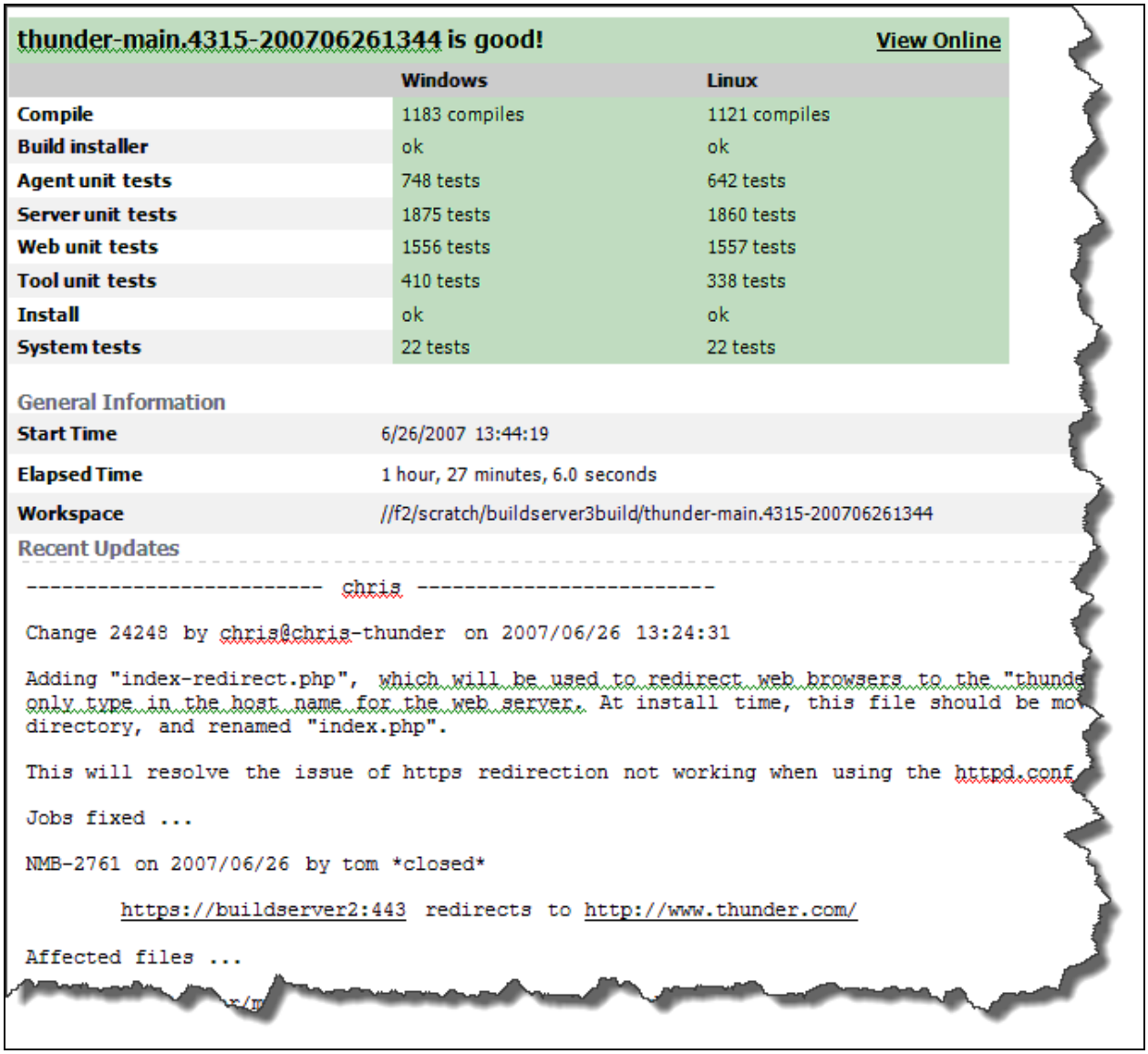
The Resources page reports the current state of your defined resources. This page also shows the current state of the agent (connected or not) and any steps currently running on that resource. Use this report to monitor your resources in near real time.

Build reports using *ecreport*

ElectricFlow includes a utility, *ecreport*, that summarizes build results and sends that summary through email. This customizable utility allows you to create a simple matrix based on results stored in properties and log files. Details such as SCM change logs and errors can be included.

In addition, you can attach your own reports to any job and have them show up as attachments to the Job Details Report. To attach custom reports, put them in the top-level workspace directory in a file that ends with "`.html`" and contains the string "report" or "Report." Names of all matching files are displayed in the Summary section in the upper right corner of the page. Click on a report name to view the report.

The following is an example of `ecreport` output:



thunder-main.4315-200706261344 is good! View Online

	Windows	Linux
Compile	1183 compiles	1121 compiles
Build installer	ok	ok
Agent unit tests	748 tests	642 tests
Server unit tests	1875 tests	1860 tests
Web unit tests	1556 tests	1557 tests
Tool unit tests	410 tests	338 tests
Install	ok	ok
System tests	22 tests	22 tests

General Information

Start Time	6/26/2007 13:44:19
Elapsed Time	1 hour, 27 minutes, 6.0 seconds
Workspace	//f2/scratch/buildserver3build/thunder-main.4315-200706261344

Recent Updates

```
------------------------  chris  -------------------------

Change 24248 by chris@chris-thunder  on 2007/06/26 13:24:31

Adding "index-redirect.php",  which will be used to redirect web browsers to the "thunde
only type in the host name for the web server.  At install time,  this file should be mo
directory,  and renamed "index.php".

This will resolve the issue of https redirection not working when using the httpd.conf

Jobs fixed ...

NMB-2761 on 2007/06/26 by tom *closed*

        https://buildserver2:443  redirects to http://www.thunder.com/

Affected files ...
```

---

ecreport extracts information about a particular job from the ElectricFlow server and writes an HTML report to standard output The report is organized as a 2-dimensional table summarizing the results of steps in the job. The table layout can be customized by defining its format using the ectool (or ec-perl) `--load` option. Invoke the program in the top-level directory of the job's workspace so it can access log file extracts in that workspace (or, use the `--dir` option).

The following options are available for use with the *ecreport* utility:

Option	Description
`--culprit`	If specified and the job failed, the report will be emailed to each user mentioned in the updates file. The value of this option provides the subject line for the message (%s will be replaced with the job name).
`--data`	Name of file containing job data for the report. If specified, this data is used in place of querying the ElectricFlow server. Used primarily for testing.
`--dir`	Change to this working directory before doing anything else.
`--help`	Print this message and exit without doing anything.
`--jobId`	Identifier for the ElectricFlow job to report on (required unless `--data` is supplied). This is a UUID.
`--load`	Name of a file containing Perl code to evaluate after option processing. Used primarily for debugging. There can be multiple `--load` options.
`--emailConfig`	The name of the Email Configuration to use when emailing the report. Defaults to "default".
`--mailto`	Email the report to this addresses.
`--replyto`	Value for the "Reply-To" field in emailed reports.
`--server`	Location of the ElectricFlow server (hostname or hostname:port), to retrieve job data. Defaults to the COMMANDER_SERVER environment variable.
`--subject`	Value for the "Subject" field in emailed reports. If the value contains a `%s`, a string summarizing the job replaces the `%s`.
`--updates`	Name of a file containing information about recent updates reflected in the job. If specified, the contents of this file are included in the report.
`--version`	Print ecreport version number.
`--webRoot`	Base URL for the ElectricFlow Web server where the job was run, such as `http://myServer`. URLs in the report will refer to ElectricFlow Web pages relative to this URL.

Example 1

```
ecreport --jobId $[/myjob/jobId] --load ./rptformat.pl --updates update.log --
webRoot
    http://myServer/commander > ecreport.html
```

This example writes a report to the file `ecreport.html`. Because this file has an `.html` extension, it will be displayed in the Reports section at the top of the Job Details page. The format of the table for this report is specified in `rptformat.pl`. An example of the contents from this file would be:

```
@::gTableRows = (
    ["",
       "Windows",
            "Linux",
            "Solaris"],
    ["Extract",
            "span",
            "span",
            "update.log"],
    ["Compile",
            "windows-compile.log",
            "linux-compile.log",
            "solaris-compile.log"],
    ["Unit test",
            "windows-unittest.log",
            "linux-unittest.log",
            "solaris-unittest.log"],
    ["System test",
            "windows-systemtest.log",
            "linux-systemtest.log",
            "solaris-systemtest.log"]
);
```

The "Recent Updates" report section is specified in `update.log`. The `--webRoot` argument defines the root for any URL that appears in the report.
Example 2

```
ecreport --jobId $[/myJob/jobName] --load ./ecreportConfig.pl --mailto
          user1@company.com --replyto user2@company.com
           --subject 'Build Report' --updates updates.log
           --webRoot http://myServer/commander
```

This report will be sent as the body of an email message to `user1@company.com` with subject "Build Report".

Default Batch Reports (Run Reports)

These reports are installed during the ElectricFlow installation and automatically scheduled to run daily.

To access these reports: Select the Projects tab, then select the Electric Cloud project, then click the Reports subtab. This report collection graphs trends over time, including success/failure, build times, and resource consumption. These reports allow you to customize two levels of grouping into "variants." These variants may represent products, branches, locations, architectures, or any other descriptors you would like to use to group your workload.

After a Default Batch Report runs, a link for each report appears on the Job Details page in the Links section. When you select a report from the Links section, it will be displayed in full-screen view.

Report types

Cross Project Summary—Shows the status of multiple variants over the past 30 days. This high-level summary of each of the 30 days shows green, yellow, or red depending on the result of the best build of the day. The summary also shows average times, total invocations, and other data.

Variant Trend—Shows more details about a particular variant such as build outcome over time, build quantity over time, elapsed build times over time, and a list of actual jobs that ran during the period.

Daily Summary—Shows the most recent results of each variant, including the last successful build.

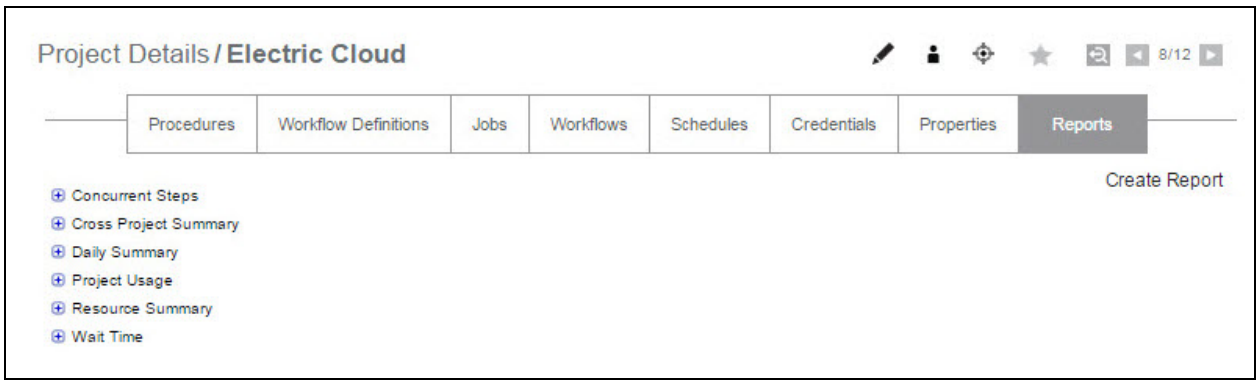
Resource Summary—Shows resource utilization over the last 24 hours. You can see which resources are idle and which resources are busy. ElectricFlow has advanced features that allow you to select resources for your job at runtime, using late binding of property names. This feature provides unparalleled flexibility and control over your environment, but could make it difficult to find out which resources were used. This report provides the visibility you need into resource loading.

Resource Detail—Shows the utilization of a particular resource over the course of one day.

Wait Time—Shows cumulative wait time due to license, resource, and workspace waits. It also shows the number of steps that were delayed for any of those reasons.

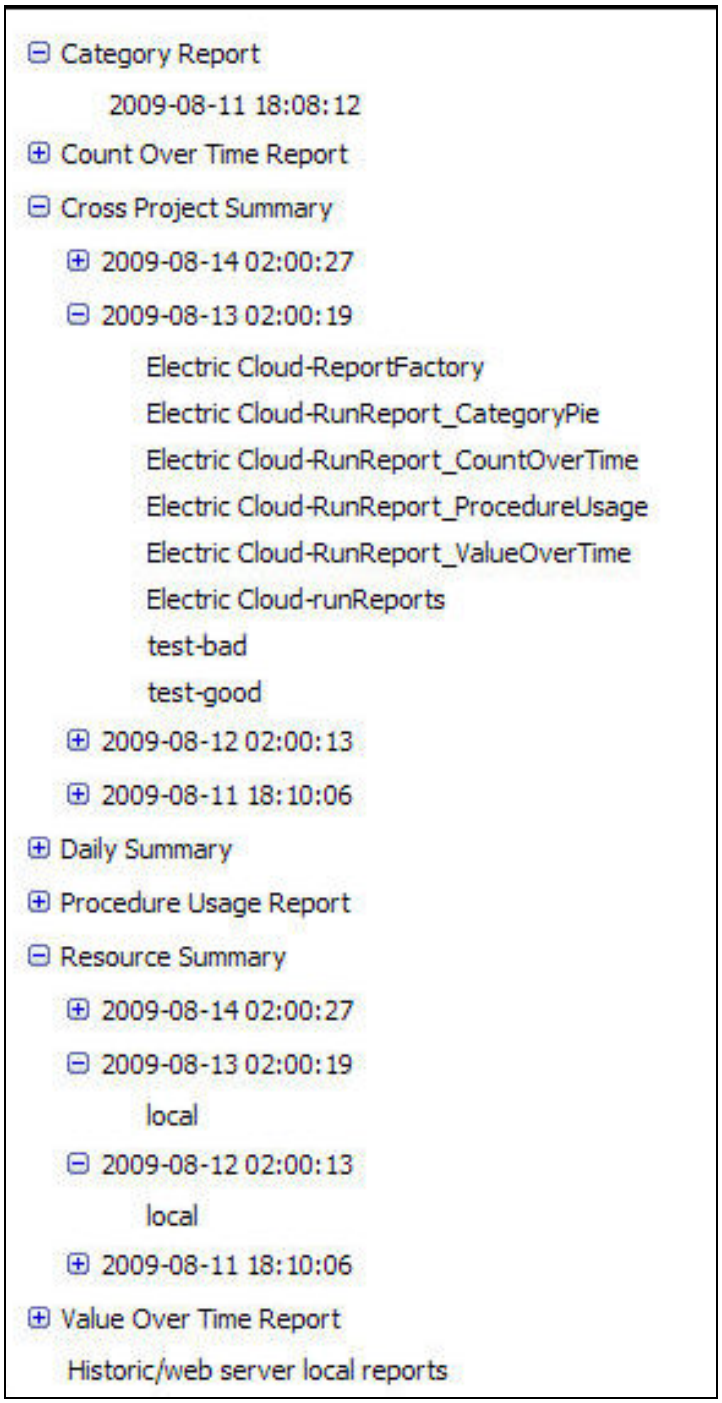
Default Batch Reports

- Cross Project Summary
- Variant Trend
- Daily Summary
- Resource Summary
- Resource Detail



The navigation tree on the left-side of the screen allows you choose the report and date of interest. These reports, which show long term trends over all projects and resources, are run from the `runReports` procedure in the Electric Cloud project.

The following example shows the Default Batch Reports, the historical reports link, and Optional Batch Reports (described below). Notice that you can expand a selected report to three levels. The third level represents drill-down links available within that report. In addition, if you click on a report name, the most current report is displayed.



You may change the number of reports displayed under each report type. Locate the `propertySheet` named `ec_reportConfiguration` on the project or server (project overrides server) and find the "groupings" section. Next, you can create nested property sheets under `ec_reportConfiguration` with the properties `groupingName`, `sortDirection`, and `limit`. If you set this information using ectool, you will also see it on ElectricFlow's web interface .

If the `ec_reportConfiguration` propertySheet does not exist, you will need to create it, then copy the example below into the propertySheet.

Example:

```
ectool setProperty ec_reportConfiguration/reportDaysLimit --value 90 --projectName foo

ectool setProperty ec_reportConfiguration/groupings/1/groupingName --value reportType
--projectName foo
ectool setProperty ec_reportConfiguration/groupings/1/sortDirection --value ascending
--projectName foo
ectool setProperty ec_reportConfiguration/groupings/1/limit --value 100 --projectName
foo

ectool setProperty ec_reportConfiguration/groupings/2/groupingName --value reportDate
--projectName foo
ectool setProperty ec_reportConfiguration/groupings/2/sortDirection --value descending
--projectName foo
ectool setProperty ec_reportConfiguration/groupings/2/limit --value 5 --projectName
foo

ectool setProperty ec_reportConfiguration/groupings/3/groupingName --value
reportResource --projectName foo
ectool setProperty ec_reportConfiguration/groupings/3/sortDirection --value ascending
--projectName foo
ectool setProperty ec_reportConfiguration/groupings/3/limit --value 100 --projectName
foo
```

This example explicitly defines the default behavior for project foo's reports.

The 1, 2 and 3 `propertySheets` define the order of the groupings. The actual names 1, 2, 3, could be anything as long as they are listed alphanumerically in the order you need.

Additionally, any groupings that do not exist will be skipped during output. For example, if you added a `reportRunByUser` grouping #4 and added it to some reports that define `reportType="User Builds"`, `reportDate`, and `reportRunByUser` properties (but not `reportResource`), you would get something similar to a resource usage report structure with users rather than resources at the deepest level.

> **Note:** Displaying 500 or more reports at a time might degrade performance.

Advanced reporting information—ecrptdata

How are default batch reports generated?

Reports are scheduled as part of a special procedure, runReports, in the Electric Cloud project. This procedure has one step, `runEcrptdata`, that runs the `ecrptdata` program. This program reads data from ElectricFlow, summarizes it for reports, and generates HTML reports using the BIRT reporting engine. The program takes input parameters. During installation, reporting is configured as follows:

```
ecrptdata --end yesterday --grain 300 --debug 2 --reports all
```

By default, all reports are run every day at 2am. The Cross Project Summary and Variant Trend shows data from the previous 30 days. The Daily Summary, Resource Summary, and Resource Detail reports show data from the last full day (yesterday). You can adjust the schedule to run more often or at different times by altering the "Report Schedule" schedule in the Electric Cloud project.

Changing the report time period

If you would like to alter the report time period, you must change the input parameters to the `ecrptdata` program within the procedure step `"Electric Cloud:runReports:runEcrptdata"`. Select

the Ecrptdata step name to go to the Edit Step page. Make the changes you need in the Parameters section.

If the `--end` option is omitted, the end of the reporting period is the time when the report runs (now). If the keyword "yesterday" is provided, reports will include full data for yesterday. Otherwise, a specific date and time can be given in the format "YYYY-MM-DDTHH:MM:SS".

Changing the report grouping

You can tell ElectricFlow how to group the Cross Project Summary and Variant Trend reports. By default, the outer grouping is "Project Name" and the inner grouping is "Procedure Name." To change these groupings, use the following ecrptdata options:

`--group1Name xxxx`	Set the column header for the first column of the Cross Project Summary report to "xxxx".
`--group1Property yyyy`	Use property "yyyy" as the name to group on in the first column of the Cross Project Summary report.

All lookups are done with respect to an individual job and will be converted to absolute property paths. The values `projectName`, `procedureName`, and `jobId` are set in context for the job being examined.

Examples:

`/myProject/foo` will be converted into the API call `"ectool getProperty /projects/projectName/foo"`

`/myProcedure/foo` will be converted into the API call `"ectool getProperty /projects/projectName/procedures/procedureName/foo"`

`/myJob/foo` will be converted into the API call `>"ectool getProperty /jobs/jobId/foo"`

`/myParameter/foo` will be converted into the API call `"ectool getActualParameter foo –jobId jobId"`

To show the value of the procedure name as the outer grouping and the value of the parameter `"branch"` as the inner grouping,
use the following ecrptdata options:

```
--group1Name Procedure --group1Property "/myProcedure/procedureName"
  --group2Name Branch --group2Property /myParameter/branch
```

> **Note:** The `/myParameter` "shortcut" to the property path was created specifically for `ecrptdata` only—it cannot be used as other `"/my"` property shortcuts are used. For more information on property shortcuts, see the Properties Help topic. In addition, when working with `ecrptdata`, any `/my` value is case-insensitive. For example, you can use `/myParameter` or `/myparameter`.

Filtering

`ecrptdata` supports `"findObjects"` type filters when searching for jobs and job steps.

This feature is implemented with two new command line switches: `"--jobFilter <property, operator, operand>"` and `"--stepFilter <property, operator, operand>"`. The `jobFilter` option filters the CrossProjectSummay, DailySummary, and VariantTrend reports. The `stepFilter` option filters the ResourceSummary and ResourceDetail reports.

Examples:

```
--jobFilter "branch,isNotNull,1"

--jobFilter "jobName,like,my-procedure%"

--stepFilter "outcome,notEquals,aborted"
```

Adding additional data columns to the report

You can add custom data columns to each report. These columns are specified by providing a comma separated list of property references in the following ecrptdata options:

`--crossProjectProperties`	sets extra columns for the Cross Project Summary Report
`--variantTrendProperties`	sets extra columns for the Variant Trend Report
`--dailySummaryProperties`	sets extra columns for the Daily Summary Report
`--resourceSummaryProperties`	sets extra columns for the Resource Summary Report
`--resourceDetailProperties`	sets extra columns for the Resource Detail Report

You must choose properties appropriate for the type of report and grouping you have chosen. For example, if you choose Project Name and Procedure Name for your groupings in the Cross Project Summary report, each line of the report will be in the context of a project or procedure.

You can add extra columns to show properties on the project or procedure, but it would not make sense to show properties on a resource, job, or job step. For resource reports, the properties must be in the context of a resource.

The column header is the name of the property. Examples:

```
--crossProjectProperties "/myproject/QA_State,/myprocedure/PartNumber"

--resourceSummaryProperties "/myresource/OS,/myresource/Architecture"
```

Secure login

`ecrptdata` provides options for secure login to the ElectricFlow server. There are three ways to login:

1. Use both the "`--user <user>`" and "`--pass <password>`" options.

   This method is not secure... the password will be in plain text and visible to others.

2. If you omit the "`--pass <password>`" option, the password will be read from `stdin`.

   You can then read the password from a secure file or from a stored credential. For example:

   ```
   cat mySafePasswordFile.txt | ecrptdata --user <user> ...
   ectool getFullCredential myCred --value password | ecrptdata --user <user> ...
   ```

3. Another option, "`--credential <credentialName>`", allows you to specify a credential name.

   If you use this option, "`--user <user>`" and "`--pass <password>`" are ignored. The credential must be attached to the runReports job step. Both the user and password are retrieved from the credential.

   Logic:

   1. If `--credential` is used, attempt to lookup credential and extract user and password
   2. If user is not set, return `error`
   3. If password is not set, read from `stdin`
   4. Attempt to login to server specified in `--server <server>` option using user and password

Optional Batch Reports

Additional reports are available, but these reports are not scheduled to run by default. You can schedule these reports to suit your purpose.

> **Note:** When you run an Optional Batch Report, a link for each report appears on the Job Details page in the Links section. When you select a report from the Links section, it is displayed in full-screen view. The following optional reports are available:

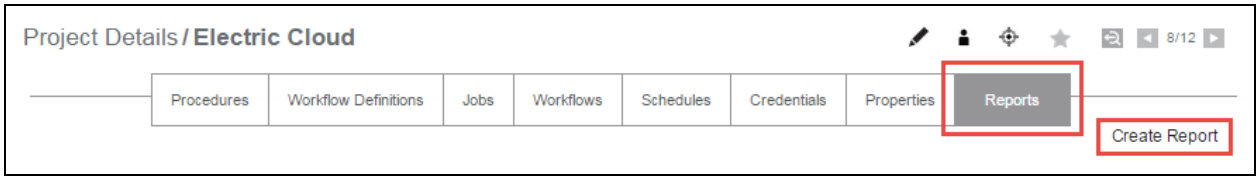
- Category Report
- Procedure Usage Report
- Count Over Time Report
- Multiple Series Report
  (formerly known as the "Value Over Time" report)

These reports are available from the Reports tab on the Project Details page for your Project. For the Category, Count Over Time, and Multiple Series reports, a **View Search Results** link is available at the bottom of the report if a Saved Filter was used to generate the report.

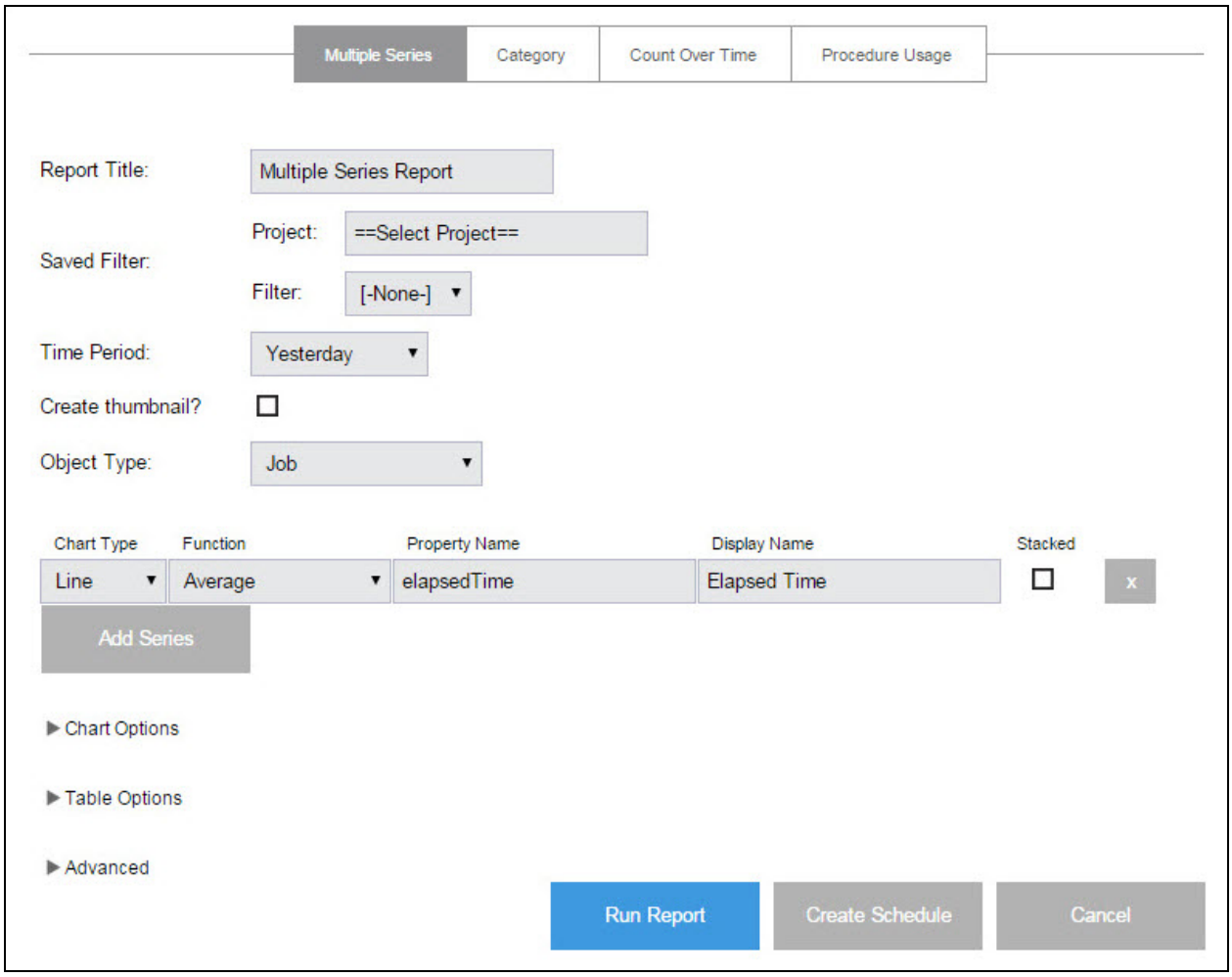
Creating a Report Job

ElectricFlow runs reports as a job, so reports can be managed like any other job. To create a report for your project:

1. Go to the Project Details page for your selected project.
2. Select the Reports subtab.
3. Click the **Create Report** link.



After clicking **Create Report**, the next screen is displayed with the default values for the selected tab (Multiple Series) in this example.



Enter information in the fields or select the appropriate values as follows:

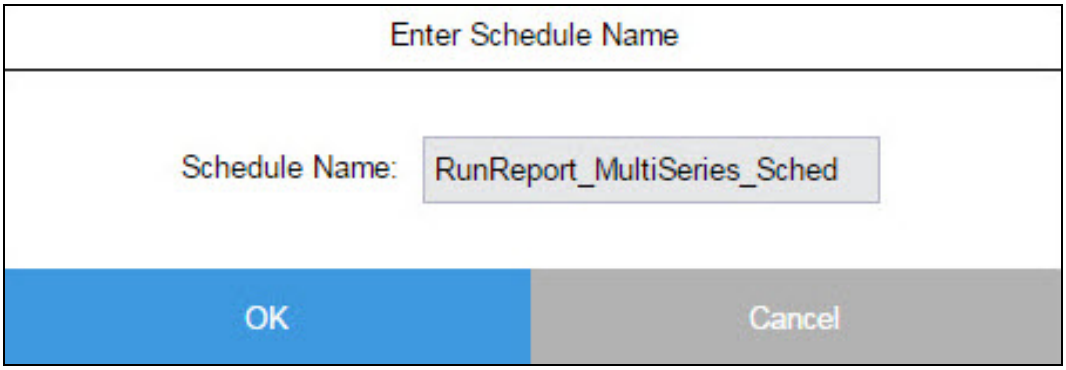
- Report Title—Your report title. Type over the default report name, choosing any unique name for your report. We supplied Basic Build Report for our report name.
- Saved Filter:
  - Project—Click your mouse inside this field to see a list of projects from which to make a selection. We select our FirstNewProject project name.
  - Filter—Use the BasicBuildFilter we created earlier.
- Time Period—Use the drop-down menu to select the time period.
- Create thumbnail?—Check this box so you can view this report on your Home page. (We will create a "thumbnail" report view on your Home page at the end of this scenario.)
- Object Type—Use the drop-down menu to select Job for this example.
- Table column choices:

- Chart Type—Use the drop-down menu to choose the chart type.
- Function—Use the drop-down menu to choose the function you need.
- Property Name—Use the default property value or delete the text and click your mouse in the blank field to see a list of possible properties. We chose "outcome" for this example.
- Display Name—You can choose a different unique Display Name if you prefer to do so.
- Stacked—Select this check box to see your report results "stacked" versus overlaid.
- The "X" icon—Click this icon to delete any row you no longer need.

- Select the **Add Series** button if you would like to create additional table entries for additional report information.
- Chart Options—Use the down-arrow to adjust the Time Grouping and see the defaults for the X and Y axis.

Buttons at the bottom of the page:

- Run Report button—this button takes you to the Job Details page to run the report immediately—one time only.
- Click the **Create Schedule** button—this button displays a box with a default schedule name you can change.



- Enter a name for the schedule, **Basic Build Report Schedule**—again, we tied the schedule name to the procedure name for which we want the report.

Viewing the report

After the report job runs, view the report by selecting the Reports subtab within the project. **Note:** All reports are viewed from within a project context. If you have reports that span multiple projects, run them from the Electric Cloud project or create a project to store summary reports.

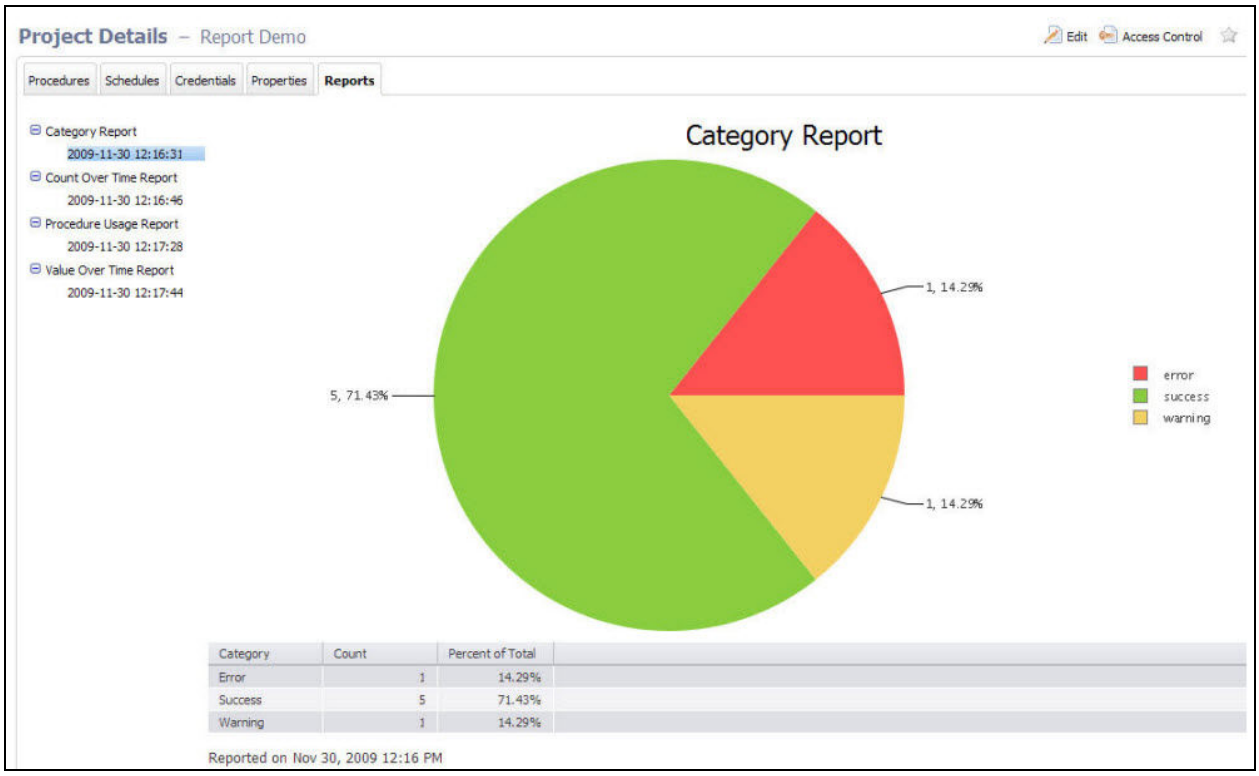
After selecting the Reports subtab, see the navigation tree on the left-side of the page. This tree allows you to select the report you want to view. The report tree contains three levels.

- Report Title—Click the "+" icon next to the report type to see available report versions.
- Report Date—Click this link to see report details.

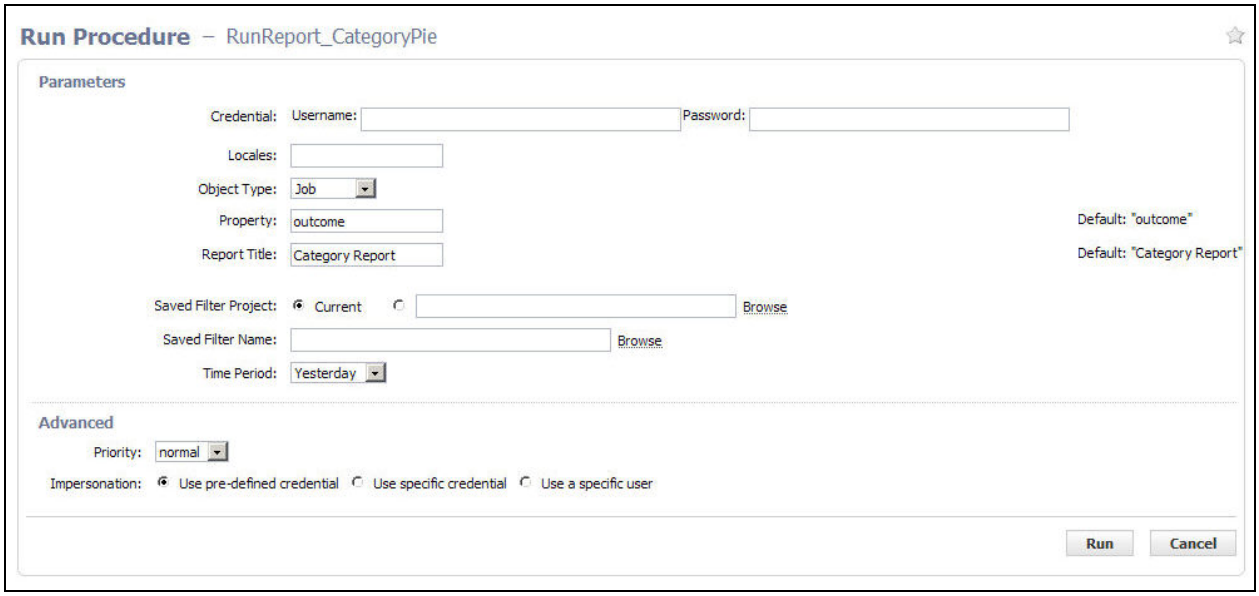
> **Note:** Click the report type to display the most recent report version.

Optional Batch report examples

Category sample report



Creating this report on the Run Procedure page:



Field descriptions:

- Credential—these are credentials to be used when collecting data. If no credentials are specified (Username/Password remain blank), the report runs with the privileges of the project from where it is run. If credentials are supplied, the data is collected using the ElectricFlow user security access in the credential. Use credentials when you want to access data outside of your project.
- Locales -a comma separated list of locales where you would like to render the report. Locale is specified as one or more of the following: `en`, `zh` , `ja`, or `ko`, which translate respectively to English, Chinese, Japanese, or Korean. If no locale is specified, the default is to use the locale of the ElectricFlow server. For example, a server already set for Chinese will see reports in Chinese with no other changes required. Note: Currently, ElectricFlow supports four locales only.
- Object Type—the type of object to search (default job)
- Property—the property of interest (default outcome) and must be a valid property for the object type selected
- Report Title—your report title (default Category Report)
- Saved Filter Project—select "Current" or Browse to find the project where you want to save this filter.
- Saved Filter Name—the name of a saved filter to use when filtering collected data.

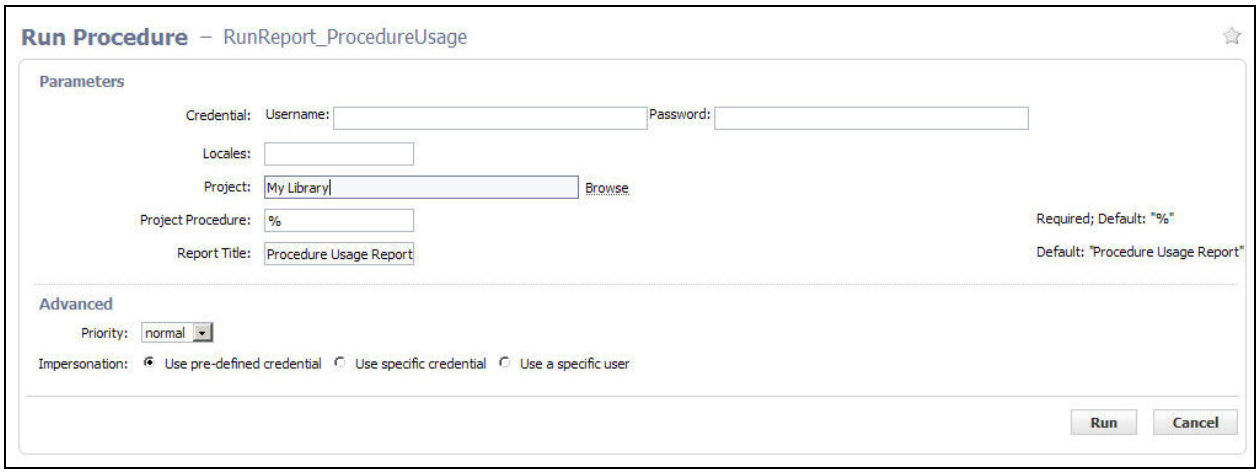
  Use the Search tab to create the filter. After entering your search criteria, click **OK**.
  If the preview displays the data you want, click **Save Filter** and provide a Project Name and Filter Name to specify where the filter is saved.
  On the Run Report parameters page, select the filter. If you leave the Filter Name blank, no filtering is performed.

  "Saved filters" can filter on start and end times. Because start and end times are specified in the Time Period parameter, there is a possibility you could select a combination of filter date and time periods that exclude all data. For example, you could select a time period of "Yesterday" and a filter of Start=less than Jan 1, 1900. To avoid this problem, we recommend you do NOT include dates or times in your filter.
- Time Period—these are keywords to define the report start and end times. The range is used to filter collected data. All values are in the local time zone.
  - Today—midnight until now
  - Yesterday—previous full day
  - This week—from the start of the previous Sunday until now
  - Last week—from Sunday to Sunday of the last complete week
  - This month—from the start of the first of this month until now
  - Last month—from the last complete calendar month, which means on January 3rd the last full month is all of December
  - This year—from January 1st until now
  - Last year—all of last year (Jan 1 to December 31)
  - All—all dates until now

Procedure Usage Sample Report

For a specific procedure, this report shows procedures called by the procedure and which procedures it called.

Creating this report on the Run Procedure page:



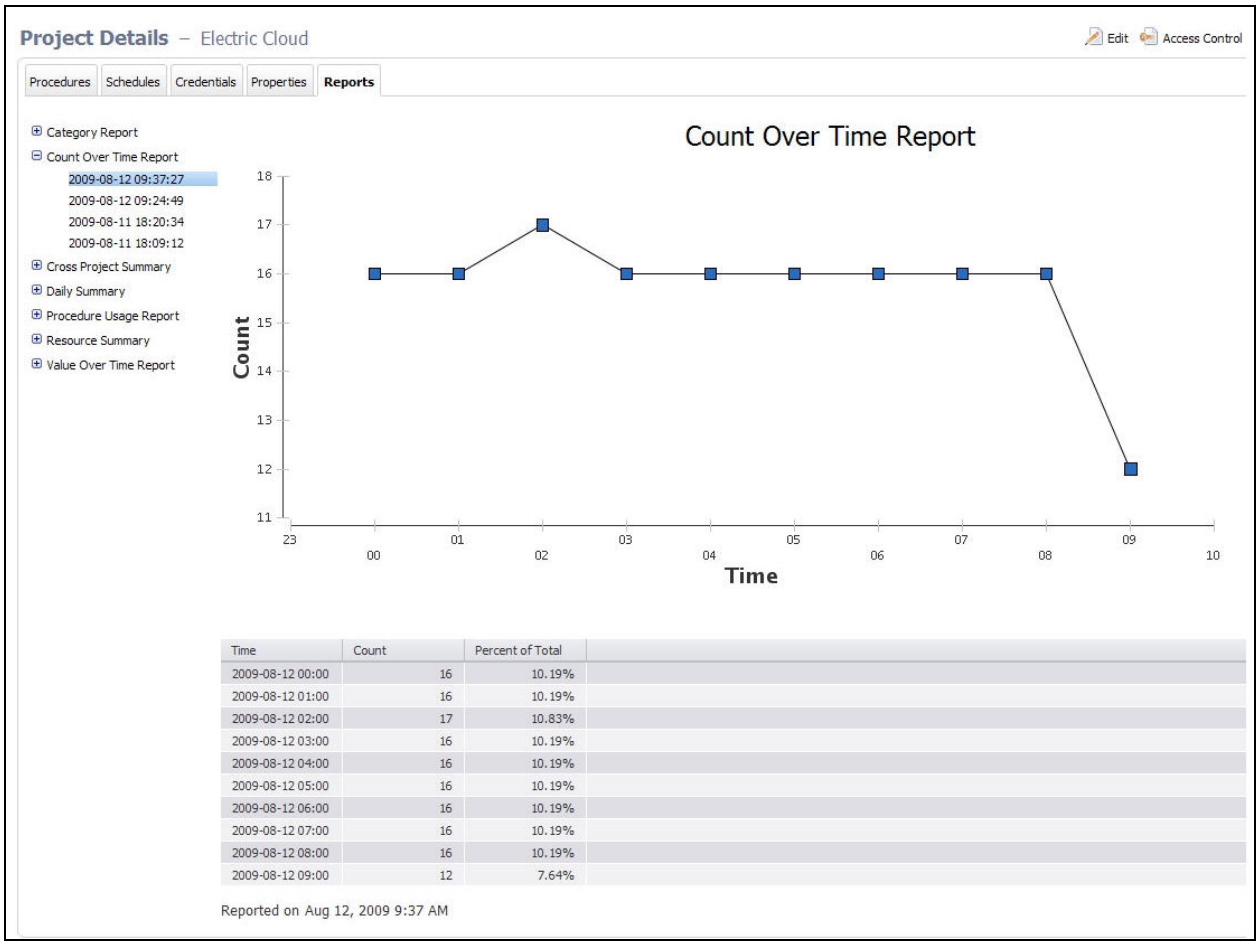
Field descriptions:

- Credential—these are the credentials to use for collecting data. If no credentials are specified (Username/Password remain blank), the report runs with the privileges of the project from where it is run. If credentials are supplied, data is collected using the ElectricFlow user security access in the credential. Use credentials when you want to access data outside of your project.
- Locales—a comma separated list of locales where you would like to render the report. Locale is specified as one or more of the following: `en` , `zh`, `ja` , or `ko`, which translate respectively to English, Chinese, Japanese, or Korean. If no locale is specified, the default is to use the locale of the ElectricFlow server. For example, a server already set for Chinese will see reports in Chinese with no other changes required. ElectricFlow supports four locales.
- Project—this is the name of the project where you want to install the report.
- Project Procedure—enter a project procedure. "called" and "called by" data is shown for that project/procedure combination. You can use "wildcards." For example, to see data for ALL projects and procedures, enter the `%` character in each field.
- Report Title—this is the title of the report

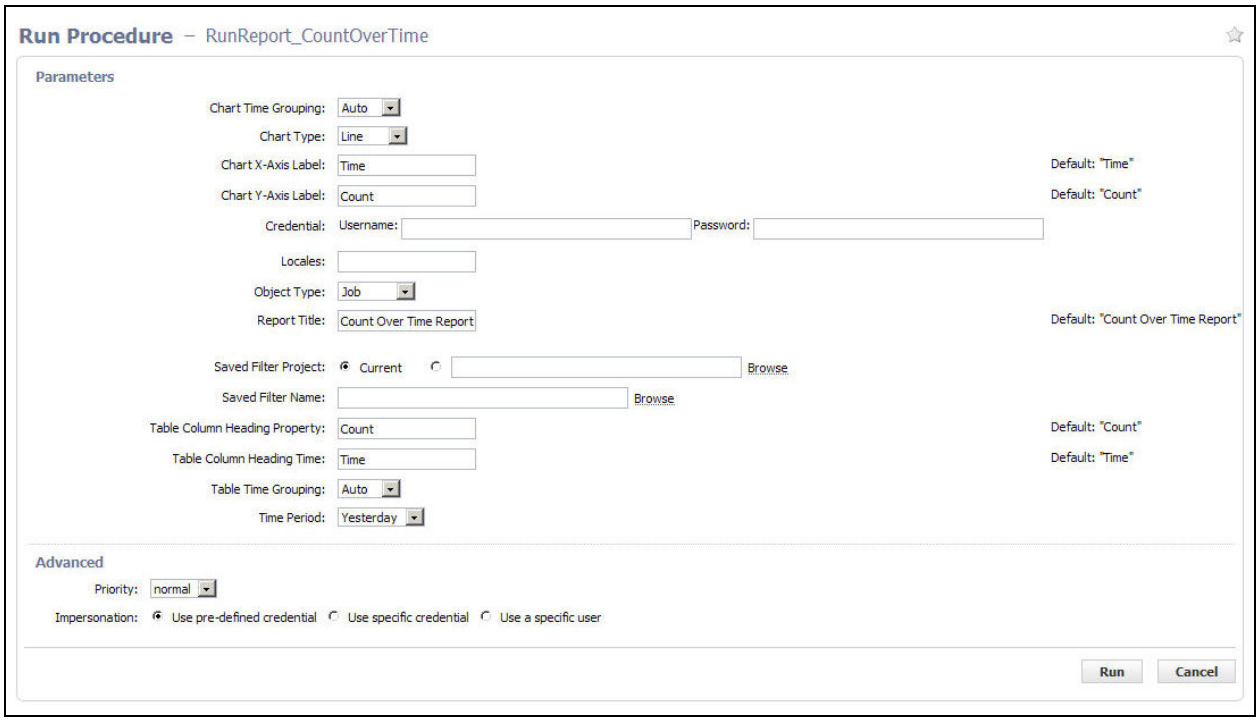
Count Over Time Sample Report

This report shows search results from counting multiple job, job step, or resource objects within the selected time range. For example, this report counts the number of occurrences for the specified object-type over the selected time range--if a filter exists, it will be applied. This report does not aggregate the value (use ValueOverTime if you want aggregation), each occurrence counts as one.

> **Note:** Generating large numbers of datapoints can exceed BIRT's charting capability.



Creating this report on the Run Procedure page.



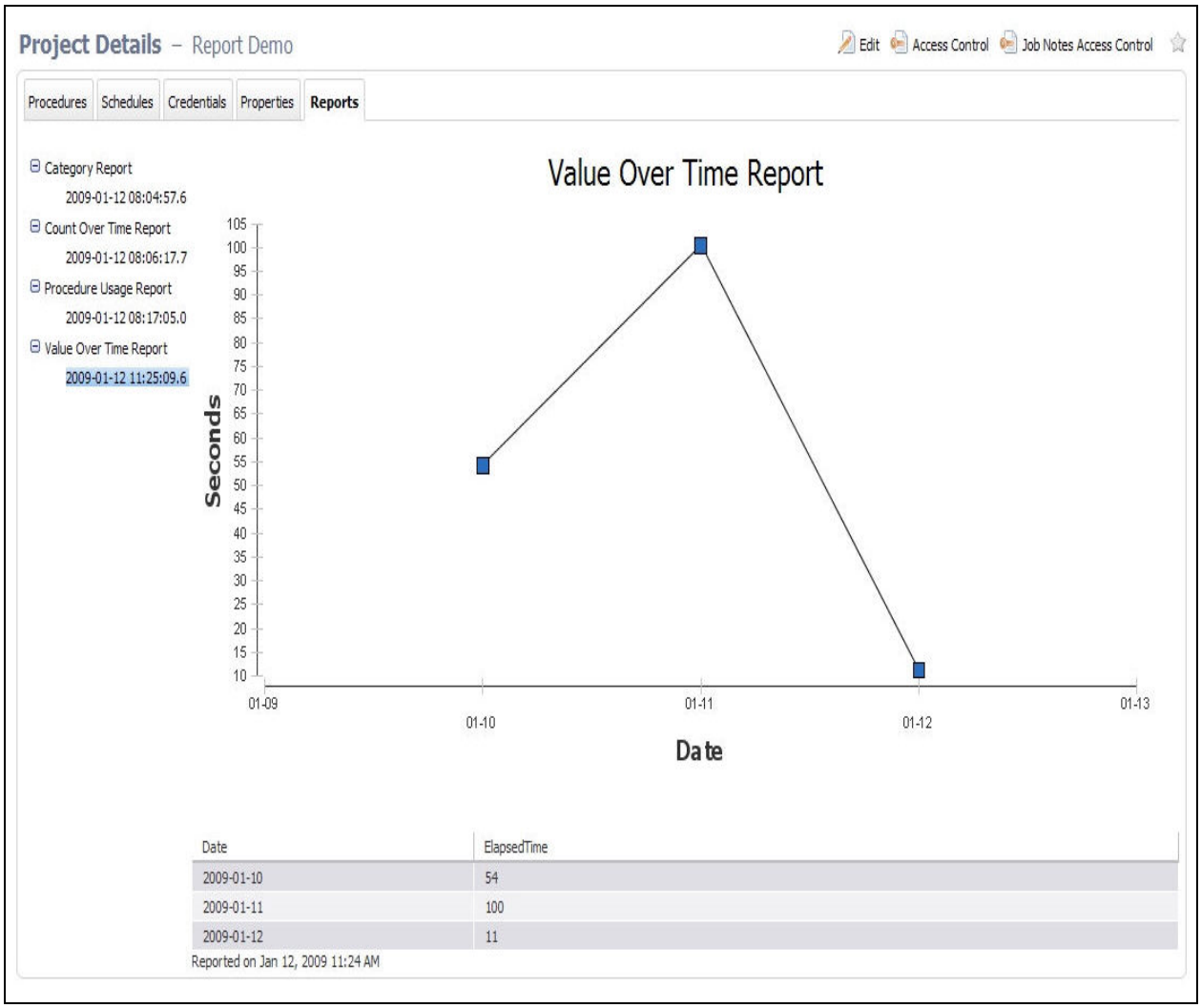
Field descriptions:

- Chart Time Grouping—these are: Auto (default), Minute, Hour, Day, Week, Month, or Year
- Chart Type—these are: Line (default), Bar, Cone, Tube, Triangle, or Area
- Chart X-Axis Label—the text to display on chart x-axis (default Date)
- Chart Y-Axis Label—the text to display on chart y-axis (default Count)
- Credential—credentials to be used when collecting data. If no credentials are specified (Username/Password remain blank), the report runs with the privileges of the project from where it is run. If credentials are supplied, the data is collected using the ElectricFlow user security access in the credential. Use credentials when you want to access data outside of your project.
- Locales—a comma separated list of locales where you would like to render the report. Locale is specified as one or more of the following: `en`, `zh`, `ja`, or `ko`, which translate respectively to English, Chinese, Japanese, or Korean. If no locale is specified, the default is to use the locale of the ElectricFlow server. For example, a server already set for Chinese will see reports in Chinese with no other changes required. Currently, ElectricFlow supports four locales only.
- Object Type—the type of object to search (default job)
- Report Title—your report title (default Count Over Time Report)
- Saved Filter Project—select "Current" or Browse to find the project where you want to save this filter.

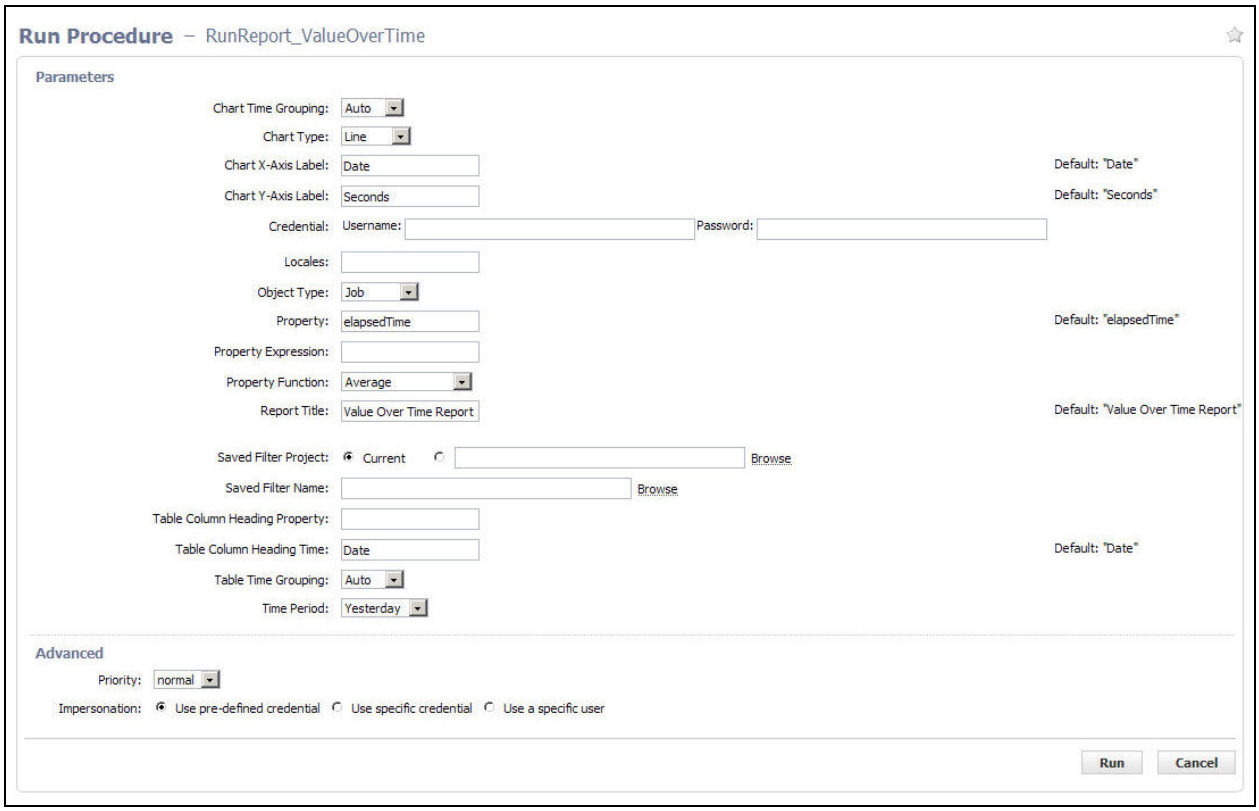
- Saved Filter Name—the name of a saved filter to use when filtering collected data. Use the Search tab to create the filter. After entering your search criteria, click **OK**. If the preview displays the data you want, click **Save Filter** and provide a Project Name and Filter Name to specify where the filter is saved. On the Run Report parameters page, select the filter. If you leave the Filter Name blank, no filtering is performed.
- "Saved filters" can filter on start and end times. Because start and end times are specified in the Time Period parameter, there is a possibility you could select a combination of filter date and time periods that exclude all data. For example, you could select Time Period of "Yesterday" and a filter of Start=less than Jan 1, 1900. To avoid this problem, Electric Cloud recommends you do NOT include dates or times in your filter.
- Table Column Heading Property—the table heading for the right column (default Count)
- Table Column Heading Time—the table heading for the left column (default Date)
- Table Time Grouping—these are: Auto (default), Minute, Hour, Day, Week, Month, Year
- Time Period—these are keywords to define the report start and end times. The range is used to filter collected data. All values are in the local time zone.
    - Today—midnight until now
    - Yesterday—previous full day
    - This week—from the start of the previous Sunday until now
    - Last week—from Sunday to Sunday of the last complete week
    - This month—from the start of the first of this month until now
    - Last month—from the last complete calendar month, which means on January 3rd the last full month is all of December
    - This year—from January 1st until now
    - Last year—all of last year (Jan 1 to December 31)
    - All—all dates until now

Multiple Series Report

This reports shows the cumulative value of a property over time. This report should be run against properties that contain numbers. (Use the Count Over Time report if your property contains string values.) You can specify the function to use for aggregating values (sum, count, average). For example: Assume a property that contains the number of failed unit tests is stored on each job. Using the Multiple Series report, you can see the failure trend over time.



Creating this report on the Run Procedure page:



Field descriptions:

- Chart Time Grouping—these are: Auto (default), Minute, Hour, Day, Week, Month, or Year
- Chart Type—these are: Line (default), Bar, Cone, Tube, Triangle, or Area
- Chart X-Axis Label—the text to display on chart x-axis (default Date)
- Chart Y-Axis Label—the text to display on chart y-axis (default Count)
- Credential—these are credentials to be used when collecting data. If no credentials are specified (Username/Password remain blank), the report runs with the privileges of the project from where it is run. If credentials are supplied, the data is collected using the ElectricFlow user security access in the credential. Use credentials when you want to access data outside of your project.
- Locales—a comma separated list of locales where you would like to render the report. Locale is specified as one or more of the following: `en`, `zh`, `ja`, or `ko`, which translate respectively to English, Chinese, Japanese, or Korean. If no locale is specified, the default is to use the locale of the ElectricFlow server. For example, a server already set for Chinese will see reports in Chinese with no other changes required. ElectricFlow supports four locales.
- Object Type—the type of object to search (default job)
- Property—the property of interest (default outcome) and must be a valid property for the object type selected
- Property Expression—an optional mathematical function to apply to your data. For example, elapsed time is recorded in milliseconds. To make a useful report with elapsed times, you may want to show seconds, so this field would be entered as "/1000".

- Property Function—the following list contains the aggregation functions to use for data in the chart and table.

Count	Median
Count Distinct	Mode
First	Std Deviation
Last	Sum
Min	

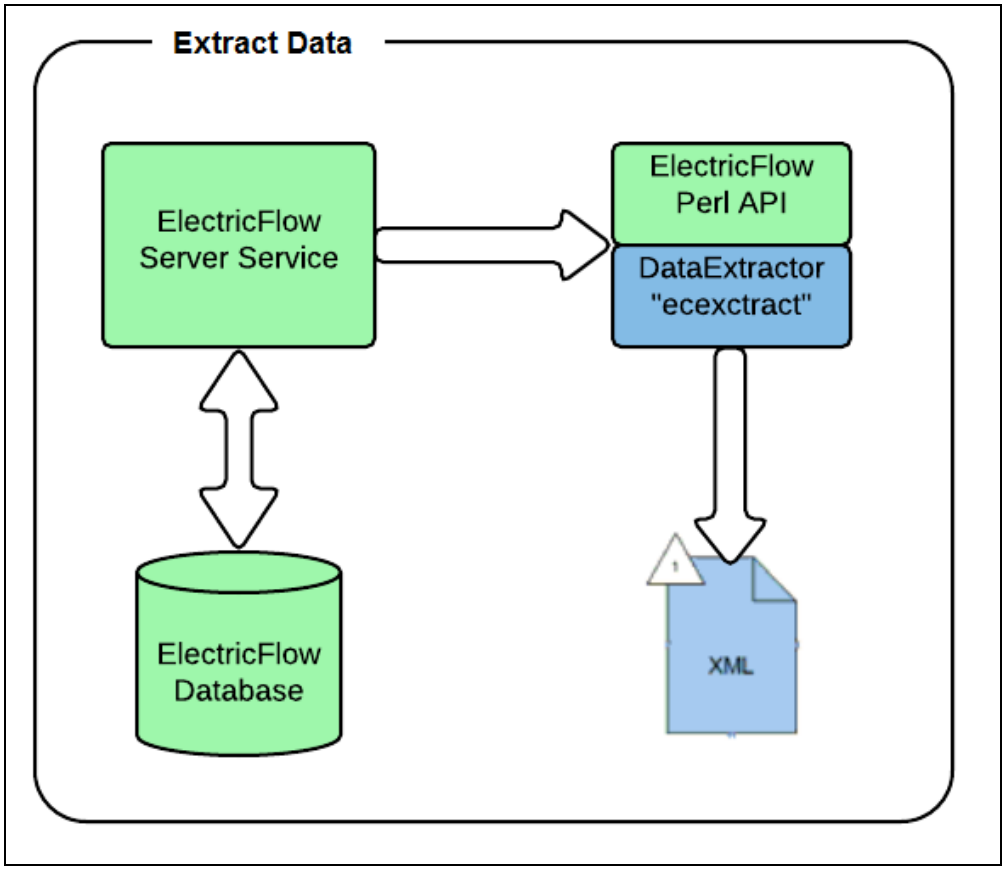
- Report Title—your report title (default Value Over Time Report)
- Saved Filter Project—select "Current" or Browse to find the project where you want to save this filter.
- Saved Filter Name—the name of a saved filter to use when filtering collected data.

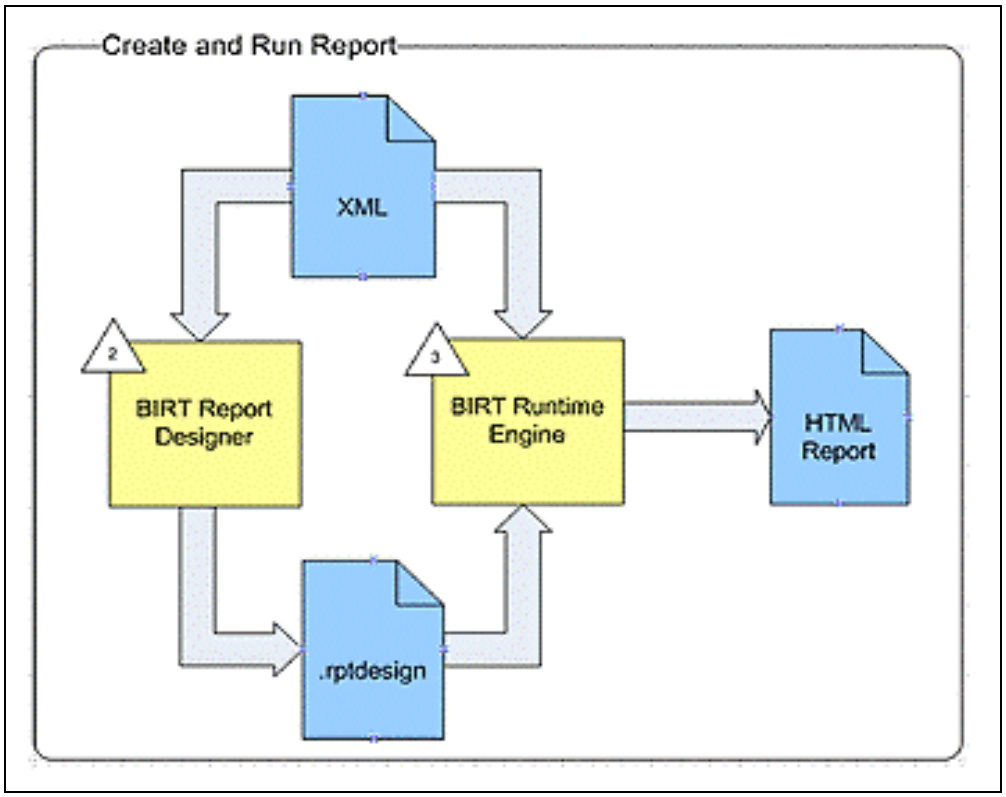
  Use the Search tab to create the filter. After entering your search criteria, click **OK**.
  If the preview displays the data you want, click **Save Filter** and provide a Project Name and Filter Name to specify where the filter is saved.
  On the Run Report parameters page, select the filter. If you leave the Filter Name no filtering is performed.

  "Saved filters" can filter on start and end times. Because start and end times are specified in the Time Period parameter, there is a possibility you could select a combination of filter date and time periods that exclude all data. For example, you could select Time Period of "Yesterday" and a filter of Start=less than Jan 1, 1900. To avoid this problem, Electric Cloud recommends you do NOT include dates or times in your filter.
- Table Column Heading Property—the table heading for the right column (default Count)
- Table Column Heading Time—the table heading for the left column (default Date)
- Table Time Grouping—these are: Auto (default), Minute, Hour, Day, Week, Month, Year
- Time Period—these are keywords to define the report start and end times. The range is used to filter collected data. All values are in the local time zone.
  - Today—midnight until now
  - Yesterday—previous full day
  - This week—from the start of the previous Sunday until now
  - Last week—from Sunday to Sunday of the last complete week
  - This month—from the start of the first of this month until now
  - Last month—from the last complete calendar month, which means on January 3rd the last full month is all of December
  - This year—from January 1st until now
  - Last year—all of last year (Jan 1 to December 31)
  - All—all dates until now

Creating Custom reports

The ElectricFlow batch mode reports which, by default, run once each night and are accessed through the Reports tab [in the Electric Cloud project] in the web UI. These reports are built using a combination of the ElectricFlow API and the BIRT Report Designer tool. BIRT is an open source Eclipse-based reporting system you can use to create your own custom reports, or modify existing ElectricFlow reports. This section describes how to write your own custom report and integrate it into ElectricFlow.





There are several steps to using BIRT to create reports:

- Create an XML file with data using the ElectricFlow Perl API
- Design a report using BIRT RCP Report Designer
- Run the report with the BIRT Report Engine
- View the report from within ElectricFlow

Data extraction

To begin creating a custom report, you must first extract report data from ElectricFlow and move it into a standalone XML file. The recommended way to create the XML file is to use the ElectricFlow `ecextract` utility program, which is derived from the Perl API. The benefits of using `ecextract` to create XML files are:

- Security—`ecextract` enforces all ElectricFlow security and ACLs. (Direct DB access does not honor ACLs.)
- Maintainability—`ecextract` is supported by Electric Cloud, including backward compatibility and insulates you from database schema changes.
- Simplicity—`ecextract` provides easy-to-use search and filtering options. Because of the complex nature of our schema, particularly for properties, it may be difficult to create custom SQL statements.
- Reuse—You may find that XML files created by built-in reports already contain all the data you need. If you simply want to present data differently or "slice" the data in a different way, you can reuse the XML from the existing built-in reports.

> **Note:** If you already use ectool or Perl for data extraction, and have already created the scripts you need for custom reports, you may continue to use those scripts. However, we recommend using `ecextract` if you are new to creating custom reports.

Using `ecextract.pl` for data extraction

A Perl program, `ecextract.pl`, is provided so you do not have to write Perl code—or you can use it as a starter for your own Perl API calls. This script allows you to specify the most common query types on a command line and generates an XML file. The following table lists currently available `ecextract` arguments:

Usage: ecextract [options]

Data Extract Options:	
`--type job|jobStep|resource`	The object type to query.
`--property [label=]property`	Add a property to the output. If label is specified, it is used as the XML tag name instead of the property name.
`--filter property, operator, operand1 [,operand2]`	Add a property filter to the query.
`--savedfilter filterName`	Use a saved filter (ignores `--filter`)
`--period period`	The time period. One of: today, yesterday, thisweek, lastweek, thismonth, lastmonth, thisyear, lastyear, all, window, <number of days>, or range,<start-time>,<end-time> Range start, end time is date in ISO 8601 format: YYYY-MM-DDTHH:MM:SS[Z] If the date ends with 'Z', it is considered to be GMT time, otherwise the local timezone is used.
`--sort property, order`	Add a sort to the query. Order is either 'ascending' or 'descending'.

`--outputPath`	Path and name of output file. Default is `out.xml`.
`--max`	Maximum number of objects to return.
Server Communication Options:	
`--server server`	Address for the ElectricFlow server. Defaults to the value of the COMMANDER_ SERVER environment variable. If it does not exist, defaults to localhost.
`--port port`	HTTP listener port on the server. Defaults to 8000.
`--securePort port`	HTTPS listener port on the server. Defaults to 8443.
`--secure 0|1`	If set to true, HTTPS is used to communicate with the server. Defaults to true.
`--timeout seconds`	Set the number of seconds for the timeout. Defaults to 280.
Login Options:	
`--user user`	ElectricFlow login user.
`--pass password`	Password for login user.
`--credentialName`	Credential name (valid only when running inside a job step).
Global Options:	
`--help`	Print this message and exit without doing anything.
`--version`	Debug level. 0—errors only 1—0 + System progress 2—1 + Property Warnings 3 and higher give detailed diagnostics

Plugin File Options:	
--load loadFile	Plugin perl program to execute
--	Stop processing ecextract options. Allows plugins to process their own options following this option.

ecextract.pl is part of the EC-Reports plugin—this script is located in the "agent/bin" directory of the EC-Reports plugin.

--property

Properties can be specified as high-level values (--property propertyName), properties in a sheet (--property sheet1/sheet2/propertyName), or as relative paths (--property /myProject/propertyName).

--filter

Each entry consists of a property name to test and an operator to use for comparison. The property can be an intrinsic property defined by ElectricFlow or a custom property added by the user. Each operator takes zero or one operand to compare against the desired property.

The operators are:

contains, equals, greaterOrEqual, greaterThan, in, lessOrEqual, lessThan, like, notEqual, notLike, isNotNull, isNull

Example: --filter jobName,equals,Hello

Examples

The following 3 examples contain the script input and the output.

Example 1: Find properties on job steps for one day in April

```
ec-perl ecextract.pl ^
  --user report ^
  --pass report ^
  --server localhost ^
  --outputPath jobStepOut.xml ^
  --type jobStep ^
  --max 1000 ^
  --property jobId ^
  --property finish ^
  --property jobStepId ^
  --property stepName ^
  --property status ^
  --property outcome ^
  --property runTime ^
  --property waitTime ^
  --filter jobName,like,commander-2.2%% ^
  --filter createTime,greaterOrEqual,2008-04-01T00:00:00.000Z ^
  --filter createTime,lessOrEqual,2008-04-02T00:00:00.000Z ^
  --sort jobId,descending
```

Output from example 1

```
<?xml version="1.0" encoding="utf-8"?>
<response>
    <object objectType="jobStep" objectId="jobStep-3511785">
        <jobId name="jobId">4fa765dd-73f1-11e3-b67e-b0a420524153</jobId>
        <finish name="finish">2009-04-01T23:54:45.424Z</finish>
        <jobStepId name="jobStepId">4fa765dd-73f1-11e3-b67e-b0a420524153</jobStepId>
        <stepName name="stepName">Setup</stepName>
        <status name="status">completed</status>
        <outcome name="outcome">success</outcome>
        <runTime name="runTime">0</runTime>
        <waitTime name="waitTime">922</waitTime>
    </object>
    <object objectType="jobStep" objectId="jobStep-3511915">
        <jobId name="jobId">4fa765dd-73f1-11e3-b67e-b0a420524153</jobId>
        <finish name="finish">2009-04-01T23:59:26.844Z</finish>
        <jobStepId name="jobStepId">5da765dd-73f1-11e3-b67e-b0a420524153</jobStepId>
        <stepName name="stepName">Drop</stepName>
        <status name="status">completed</status>
        <outcome name="outcome">success</outcome>
        <runTime name="runTime">0</runTime>
        <waitTime name="waitTime">266</waitTime>
    </object>
.....
</response>
```

Example 2: Find all jobs that were created in April 2008

```
ec-perl ecextract.pl ^
 --user report ^
 --pass report ^
 --server localhost ^
 --outputPath jobOut.xml ^
 --type job ^
 --max 100 ^
 --property jobName ^
 --property jobId ^
```

```
 --property projectName ^
 --property procedureName ^
 --property branch ^
 --filter createTime,greaterOrEqual,2009-04-01T00:00:00.000Z ^
 --filter createTime,lessOrEqual,2009-04-30T00:00:00.000Z ^
 --sort projectName,ascending ^
 --sort procedureName,ascending ^
 --sort createTime,descending
```

Output from example 2

```
<?xml version="1.0" encoding="utf-8"?>
<response>
    <object objectType="job" objectId="job-158461">
        <jobName name="jobName">commander-3.2.23916-200904011650</jobName>
        <jobId name="jobId">4fa765dd-73f1-11e3-b67e-b0a420524153</jobId>
        <projectName name="projectName">Commander</projectName>
        <procedureName name="procedureName">Master</procedureName>
        <branch name="branch">3.2</branch>
    </object>
    <object objectType="job" objectId="job-158455">
        <jobName name="jobName">commander-anders-main-ad.23915-200904011627</jobName>
        <jobId name="jobId">4fa765dd-73f1-11e3-b67e-b0a420524153</jobId>
        <projectName name="projectName">Commander</projectName>
        <procedureName name="procedureName">Master</procedureName>
        <branch name="branch">main</branch>
    </object>
……
</response>
```

Example 3: Find all resources

```
ec-perl ecextract.pl ^
 --user report ^
 --pass report ^
 --server localhost^
 --outputPath resourceOut.xml ^
 --type resource ^
 --max 100 ^
```

```
 --property resourceName ^
 --property stepCount ^
 --property stepLimit ^
 --property hostName ^
 --filter stepCount,greaterThan,0 ^
 --sort resourceName,ascending
```

Output from example 3

```
<?xml version="1.0" encoding="utf-8"?>
<response>
    <object objectType="resource" objectId="resource-106">
        <resourceName name="resourceName">ecbuild-lin1</resourceName>
        <stepCount name="stepCount">2</stepCount>
        <stepLimit name="stepLimit">0</stepLimit>
        <hostName name="hostName">ecbuild-lin1</hostName>
    </object>
    <object objectType="resource" objectId="resource-108">
        <resourceName name="resourceName">ecbuild-win1</resourceName>
        <stepCount name="stepCount">1</stepCount>
        <stepLimit name="stepLimit">0</stepLimit>
        <hostName name="hostName">ecbuild-win1</hostName>
    </object>
.....
</response>
```

BIRT Report Designer

Now that you have a dataset, you can create a BIRT report. You need an XML file to use while designing the report. If you did not create an XML data file yet, now is the time to do so. After you have sample data, you are ready to start designing a report.

Report definitions are created with the BIRT Report Designer, currently version 2.5.2. These definitions are then stored in `.rptdesign` files. You have several options for installing the designer tool:

- Use a BIRT Report Designer bundled and already installed with Eclipse, or
- download BIRT Report Designer to use with your existing Eclipse installation, or
- because Eclipse is not required, you can download only the BIRT Report Designer (this is the easiest and preferred method).

We recommend downloading the BIRT RCP Report Designer only, which is the standalone design tool. The url for downloading the Report Designer is

http://www.eclipse.org/downloads/download.php?file=/birt/downloads/drops/R-R1-2_5_2-201002221500/birt-rcp-report-designer-2_5_2.zip

The general download page for BIRT 2.5.2 with the other options discussed is http://download.eclipse.org/birt/downloads/build.php?build=R-R1-2_5_2-201002221500

Using the BIRT Report Designer is beyond the scope of this Help topic. Documentation and tutorials can be found on the BIRT website: http://www.eclipse.org/birt. You may find it helpful to view the report designs for built-in ElectricFlow. These designs can be found in the `agent/reports/<report name>` directory in the EC-Reports plugin.

Understanding additional report components

In addition to the XML dataset and report design, there are optional report components you may wish to include in your report.

- External style sheet
  BIRT supports external style sheets for reports. If you use these style sheets, they need to be located with your report design. For example, built-in ElectricFlow reports use an external style sheet file `"styles_birt.css"`.
- Externalized string properties
  String literals (used in your report design) can be externalized into property files to support localization. These property files need to be located with your report design also. The properties file for the default locale will have the same name as your report and a `".properties"` file extension, for example, `CategoryPie.properties`. Text for additional locales are stored in files named `<report name>_<locale>.properties`, for example, `CategoryPie_ja.properties` contains text localized for the Japanese language.

You may review examples of these additional components for built-in ElectricFlow reports. These files are located in the EC-Reports plugin, in the `agent/reports/<report name>` directory.

Packaging reports for ElectricFlow

After you have an XML dataset (using `ecextract.pl`), a BIRT report design (using BIRT Report Designer), and any additional optional report components completed, you are ready to integrate your report into ElectricFlow. To do this:

- Store all the report components in a location accessible to your ElectricFlow agent—you can use the ElectricFlow plugins directory. For example, for your report files, you could create a directory called "MyReport" in the ElectricFlow plugins directory.
- Create an ElectricFlow procedure to perform the necessary steps to generate a report, including: extracting data, running the report design, and registering report output to appear on the Report tab on the Project Details page. You may want to copy one of the procedures from the built-in ElectricFlow reports to use as a template. The following command creates a copy of the RunReport_CategoryPie procedure called RunReport_MyReport in the project of your choice. Substitute your project name for `<my-project-name>` in the following command.

```
ectool clone --cloneName/projects/<my-project-name>/procedures/RunReport_
MyReport
    --projectName/plugins/EC-Reports/project --procedureName RunReport_
CategoryPie
```

Example: customizing the command body

The following is an example of customizing the procedure contents for the RunReport_MyReport procedure, cloned from the RunReport_CategoryPie procedure above.

One customization is required and two are optional. The remainder of the command body does not need to be changed.

- Customizing the location of the BIRT report design file
  Required: update the location of the BIRT report design file.
  - Original value: `our $reportDesignFile = "$pluginDir/agent/reports/CategoryPie/CategoryPie.rptdesign";`
  - New value: `our $reportDesignFile = "$ENV{'COMMANDER_PLUGINS'}/MyReport/MyReport.rptdesign";`
- Customizing data extraction
  To customize data extraction using `ecextract.pl`, modify the following section of the command body:

```
#-------------------------------------------------------------------------
# extractReportData
#
#       Extract report data from Commander using ecextract utility
#-------------------------------------------------------------------------
sub extractReportData()
{
                # Time property varies based on object type
                my $timeprop = "modifyTime";
                if (
                          $timeprop = "finish";
                }
                 my $command = "ec-perl \"$pluginDir/agent/bin/ecextract.pl\" "
                        . " --debug 3"
                        . " --outputPath \"$ ecextractXmlFile\""
                        . " --credential \"$[Credential]\""
                        . " --type \"$[Object Type]\""
                        . " --property time=$timeprop"
                        . " --property series1=\"$[Property]\"";
                if ("$[Saved Filter]" ne "") {
                           command = $command . " --savedFilter \"$[Saved Filter]
\""
                }
                $command = $command . " --period \"$[Time Period]\"";
```

```
                print "data extract [$command]\n";
                my $result = `$command`;
                print "extract result:\n$result";
                if ($?) {
                     exit (1);
                }
}
```

- Customizing report parameters
  To customize the parameters passed to your report design, modify the following section of the command body:

```
#--------------------------------------------------------------------------
# getReportArgs
#
#       Get report arguments
#--------------------------------------------------------------------------
  sub getReportArgs() {
            # args for creating report
             my $categoryHeading = ucfirst($[Property]);
             my @args = (
                  "-f", "$reportFormat",
                  "-o", "$reportName",
                  "-i", "images/",
                  "-p", " xmlfile=../$ecextractXmlFile",
                  "-p", "ReportTitle=$reportTitle",
                  "-p", "TableColHeadingTime=$categoryHeading",
                  "-p", "TableColHeadingSeries1=Count" ,
                  "-p", " SearchURL=" . getSearchUrl(),
                  "$reportDesignFile",
             );

             # add locale args
             foreach my $locale (@locales) {
                       if ($locale ne "") {
                             unshift @args, "-l $locale";
                       }
```

```
        }

        return @args;

}
```

Helper functions provided in ElectricCommander::ReportUtils.pm

ecgenReport—runs the BIRT report, using the BIRT Report Designer

- **installationDirectory**—this is the directory where ElectricFlow is installed and where you will find JAVA and BIRT
- **artifactDirectory**—this is the directory where reports, images, and other artifacts should be stored. To display using the Reports subtab, these items must be in a directory named "artifacts" at the top of the job's workspace
- **args**—an array of arguments to the BIRT run-time engine, which specifies parameters, design templates, and so on.

registerReport—marks the report so it will be displayed when you select the Reports subtab

- **commander**—an `ElectricCommander()object` used to make ElectricFlow calls. If you need to use a specific user context, call the login method before calling `registerReport`
- **jobId**—this is the report job ID, which is used to set properties on the job
- **url**—this is the URL to register and it takes the form workspace name/html file name
- **title**—this is the text string for the report navigation tree
- **treeGroupings**—a hash grouped `name/values` set as properties to define the report tree groupings—report tree groupings are defined on a propertySheet named `ec_reportConfiguration` on the project or on the server
  - **hash key**—this is a property name that will be created on job's `ec_reportData/$title/` property sheet and used for report tree grouping
  - **hash value**—property value

registerArtifactsDirectory—creates the property required to set the artifacts directory

- **commander**—an `ElectricCommander() object` used to make ElectricFlow calls
- **jobId**—this is the report job ID, which is used to set properties on the job
- **artifactsDir**—this is a directory path relative to the workspace where job artifacts will be stored

extractFile—this function pulls content from the report definition (stored in properties) and creates files from them

- **commander**—an `ElectricCommander()object` used to make ElectricFlow calls. If you need to use a specific user context, make sure to call the `login` method before calling `registerReport`
- **jobId**—this is the report job ID, which is used to set properties on the job
- **reportType**—this is used to lookup report attributes
- **files**—an array of hashes that specify which properties should be loaded into which files
  - **prop**—the property name from the installed report type

- **dest**—the destination file name (relative to the current working directory)
- expand—this is "1" if you want ElectricFlow to expand property references in the property, "0" if not

getReportTimestamp—return hires timestamp formatted in a specified time zone

- **TimeZone**—(optional) a time zone specifier, default is "local"
- **FormatString**—(optional) a format specifier for the timestamp string

Custom Report Examples

This Help topic contains two examples illustrating how you might use the BIRT Report Designer to modify a built-in ElectricFlow report to create a custom report more meaningful for your purposes.

Example 1: modifying an existing report—adding a "banner" heading

The easiest way to customize a report is to take an existing report and modify it. ElectricFlow provides a set of built-in reports you can modify.

Following is a process overview for modifying an ElectricFlow built-in report:

- Make a copy of an existing report
- Make changes to the report
- Test your new custom report

To copy an existing ElectricFlow report

For this example, we will make a copy of the built-in Value Over Time report. The process to copy a report is:

- Make a copy of the file system report components of the Value Over Time report located in the ElectricFlow plugins directory.

  Rename the copied directory to MyReport.

  The default location of the ElectricFlow plugins directory is:

  On Linux: `/opt/electriccloud/electriccommander/plugins`

  On Windows: `C:\Documents and Settings\All Users\Application Data\Electric Cloud\ElectricCommander\plugins`

  Linux example—The following commands will make a copy of the Value Over Time report file components and then rename the files:

```
$ cd /opt/electriccloud/electriccommander/plugins

$ cp -r EC-Reports-1.0.0.*/agent/reports/ValueOverTime/ MyReport

$ cd MyReport

$ rename ValueOverTime MyReport ValueOverTime*
```

  The contents of the MyReport directory should be:

```
$ ls -1

MyReport_en.properties

MyReport_ja.properties
```

```
MyReport_ko.properties

MyReport.properties

MyReport.rptdesign

MyReport_zh.properties

styles_birt.css
```

- Make a copy of the RunReport_ValueOverTime procedure from the EC-Reports plugin that performs the steps to generate a report, including extracting data, running the report design, and registering the report output to appear on the Report tab of the Project Details page. In this example, we will create a Project called "MyReportProject" and the procedure we copy will be called "RunReport_MyReport".

```
$ ectool createProject MyReportProject

$ ectool clone --cloneName /projects/MyReportProject /procedures/RunReport_
MyReport

      --projectName /plugins/EC-Reports/project --procedureName RunReport_
ValueOverTime

response requestId="1"

   <cloneName>RunReport_MyReport</cloneName>

</response>
```

- Edit the "RunReport_MyReport" procedure in "MyReportProject" to reference the report design file copied earlier.
- Navigate to project MyReportProject, procedure RunReport_MyReport and select the RunCustomReport step to edit the Command block:

Find the following line:

```
our $reportDesignFile =
"$pluginDir/agent/reports/ValueOverTime/ValueOverTime.rptdesign";
```

and change this line to:

```
our $reportDesignFile = "$ENV{'COMMANDER_PLUGINS'}/ MyReport/MyReport.rptdesign";
```

Click **OK** to save the change.

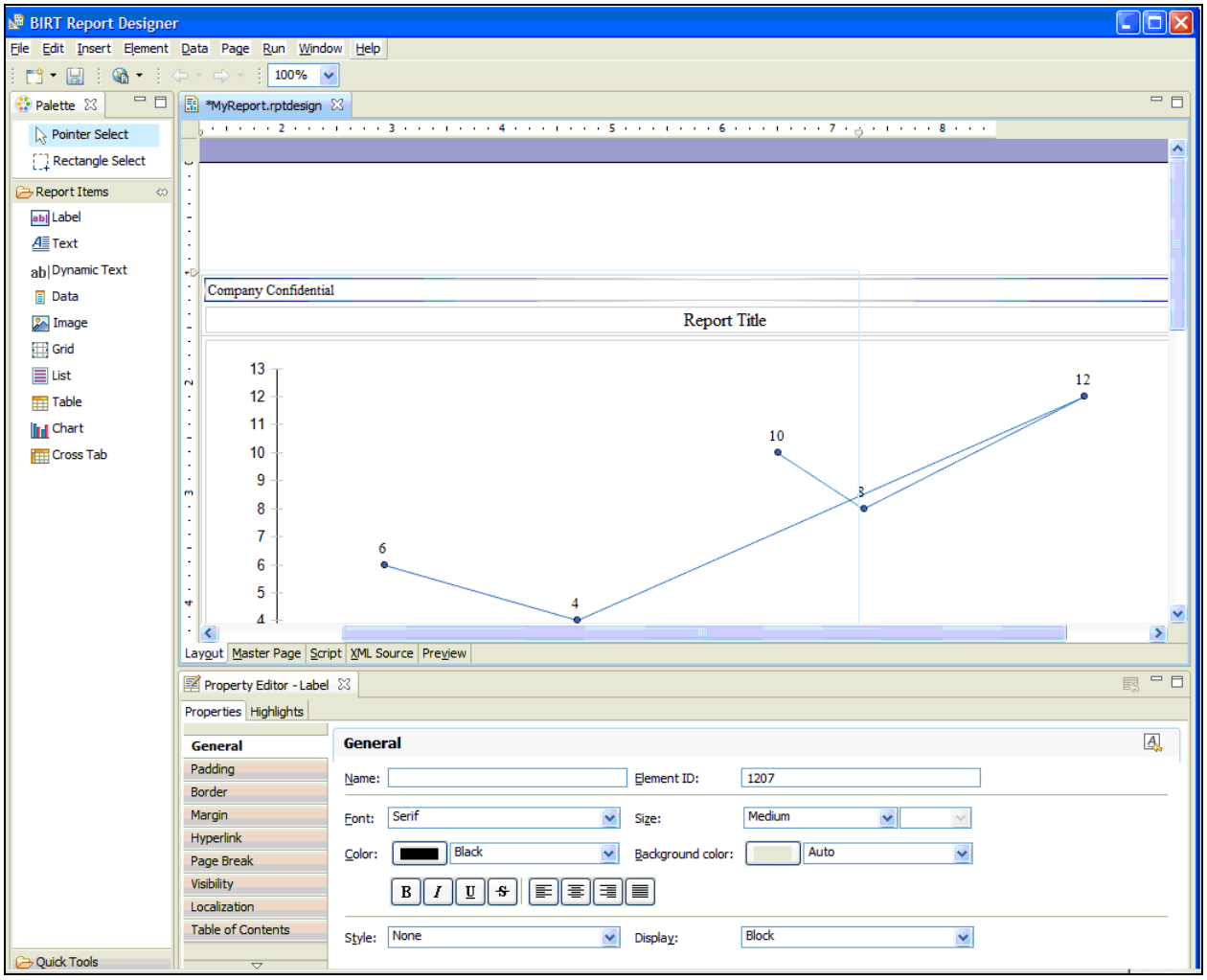
To modify the copied report design

Start the BIRT Report Designer and open the report design file (`MyReport.rptdesign`) located in the MyReport directory previously created in the ElectricFlow plugins directory. You are now ready to begin editing an existing report to create a custom report.

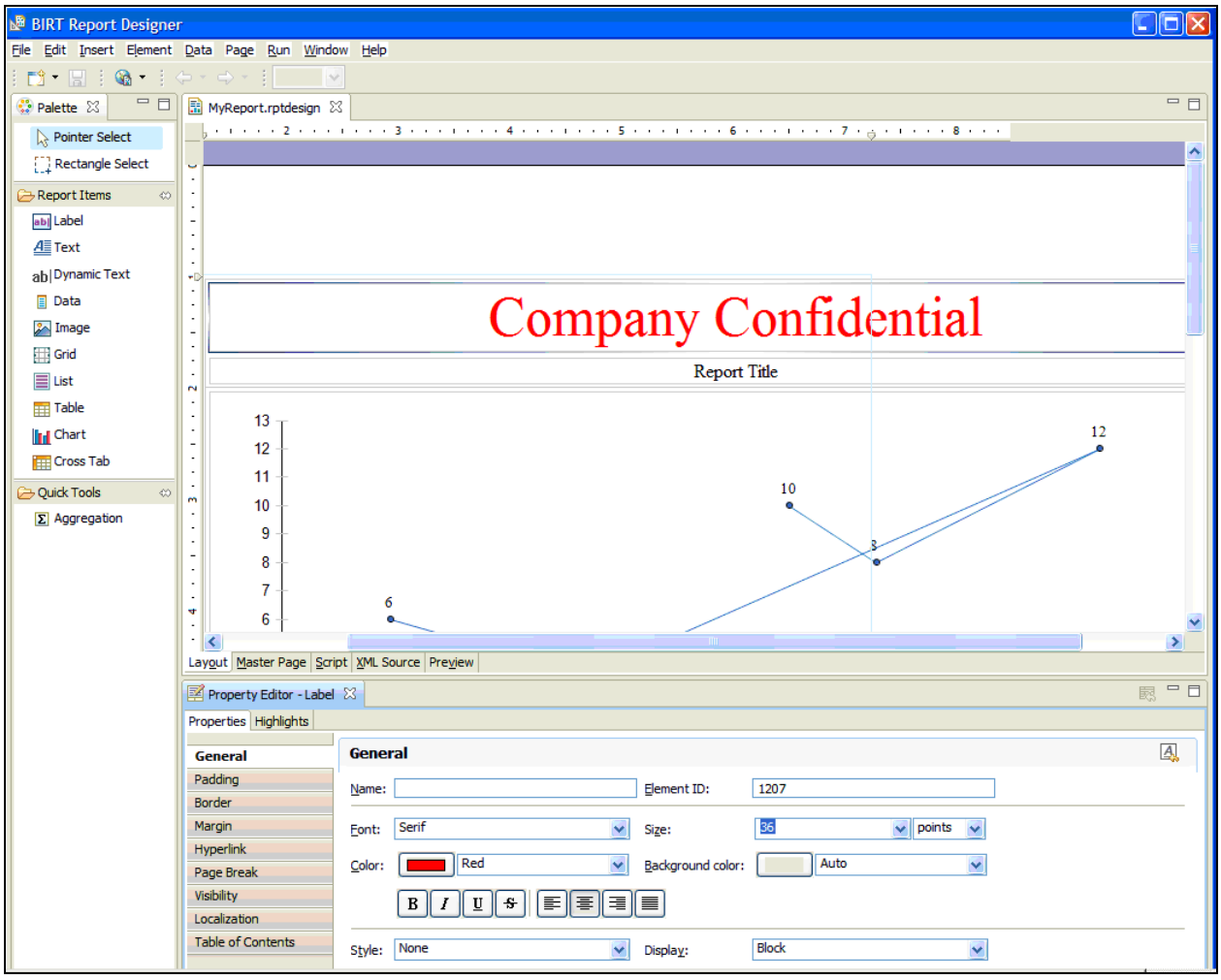
For our example, we are adding a new heading, "Company Confidential," (in large red letters) to the top of the existing report.

1. On the BIRT screen, select the Layout tab.
2. Drag a "Label" from the Palette window into the top of the report and type "Company Confidential" (or whatever text you would like to use for your test report) into the Label box.

   See the next BIRT screen example.

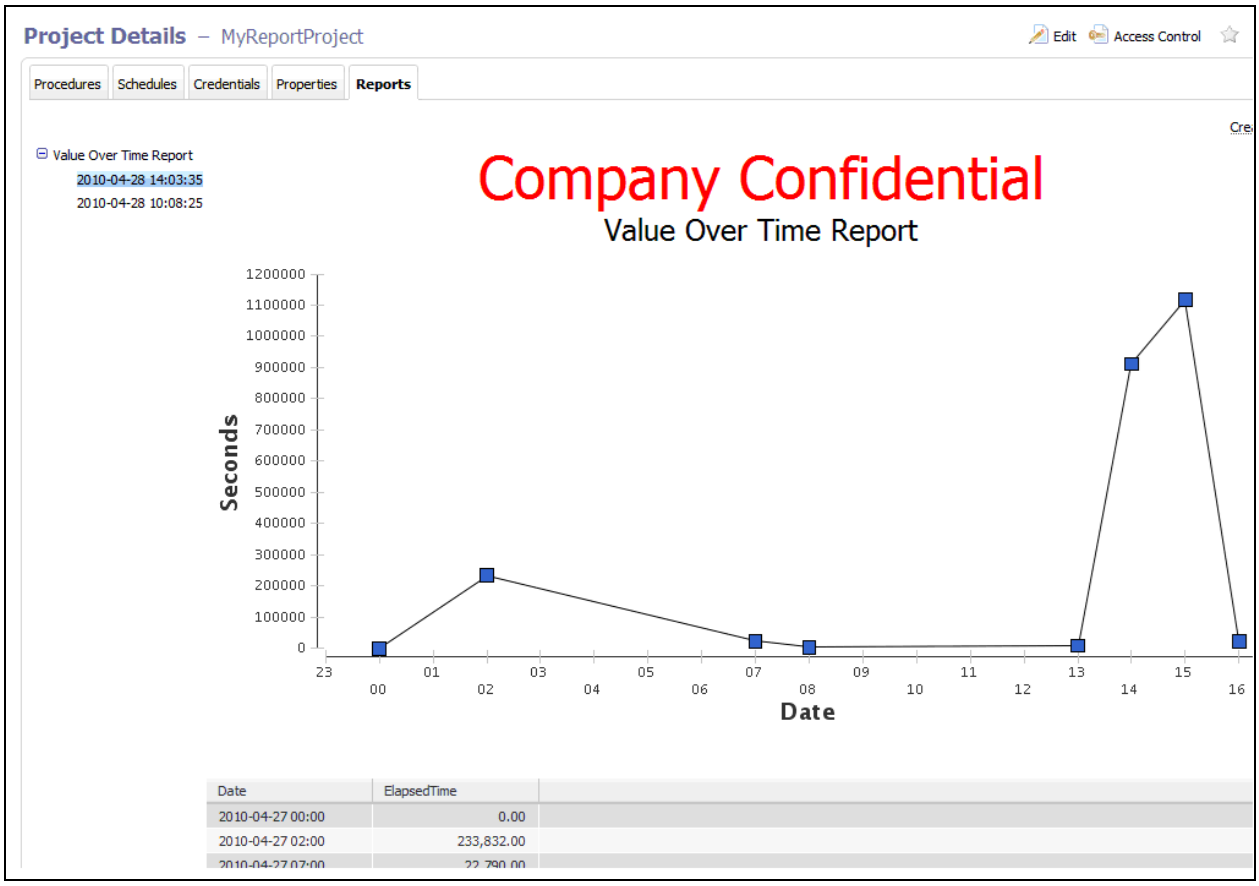


3. Now you can change the font size and color to create a banner appearance. Select the Property Editor window to customize your text—see the next screen example.
4. After customizing your text, save your report.



Test your new report

Run the report by running the RunReport_MyReport procedure in the MyReportProject project. When the job completes, your new report will be available in the Links section on the Job Details page and from the Reports tab if the MyReportProject project. You will see the new header in the report output.



Example 2: complete end-to-end example

Using BIRT and some scripting expertise, you can create almost any kind of report you can imagine. This example creates a new report that shows the run-time trend of all job steps. Tasks for creating this report can be grouped into the following steps:

- Plan your report
- Create a new BIRT rptdesign file
- Deploy your custom report
- Test your new report

Planning this report example

We decided our report will be a single-series type report that will show job steps runtime. The report will include a line chart at the top of the page whose X and Y axis represent the number of seconds and the job step ID respectively. We also want a table with two columns—column names will be "Job Step Name" and "Run Time".

Next, we need to gather all materials:

Generate sample data to test in the report.

You can use data from a previous run of a Value Over Time report—this file is located in the workspace of a previous reporting job and has the name ecextract.xml.

Also from the copy of an ElectricFlow Value Over Time report, we will "copy and paste" a few values from it to save some typing.

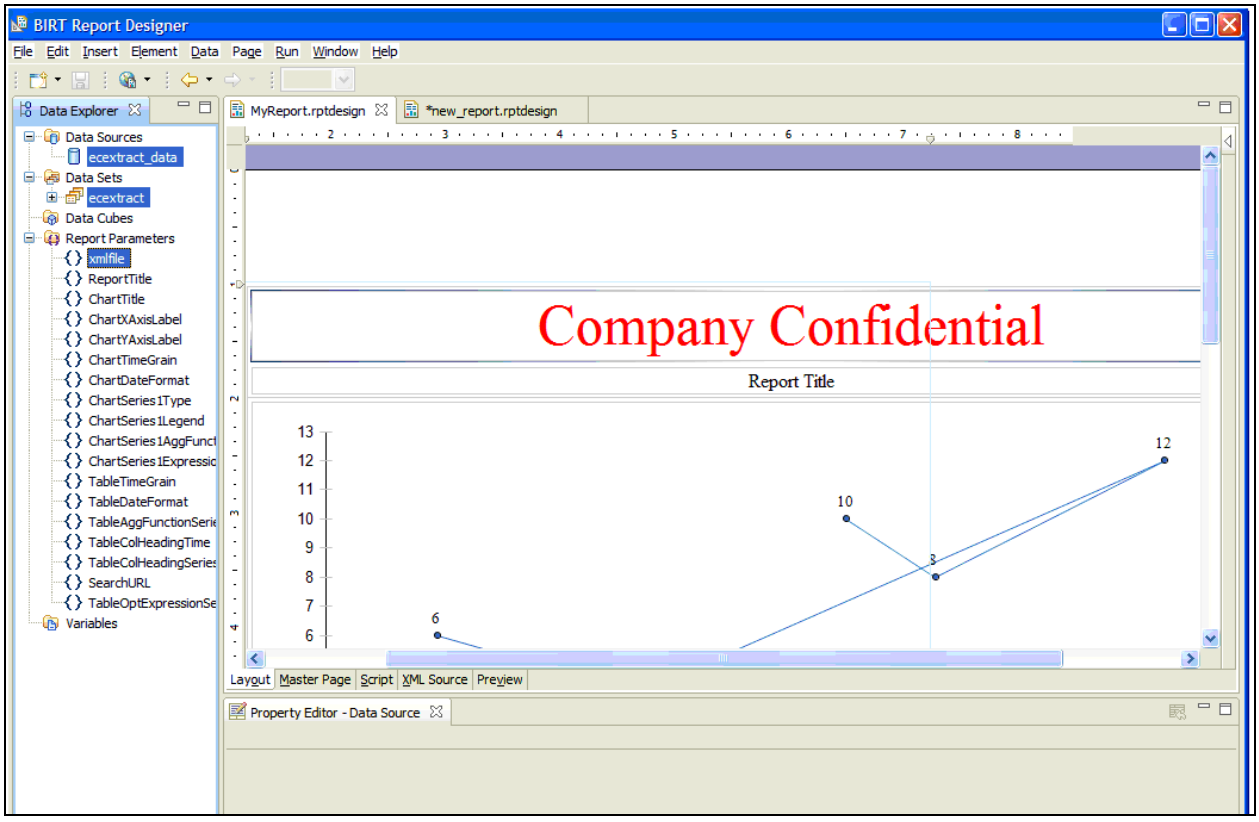
Creating a new BIRT rptdesign file

In BIRT Report Designer, create a new rptdesign file. Open the report file you created in "Example 1: modifying an existing report—adding a banner heading" so you can use the previous example report to copy some items into your new report.

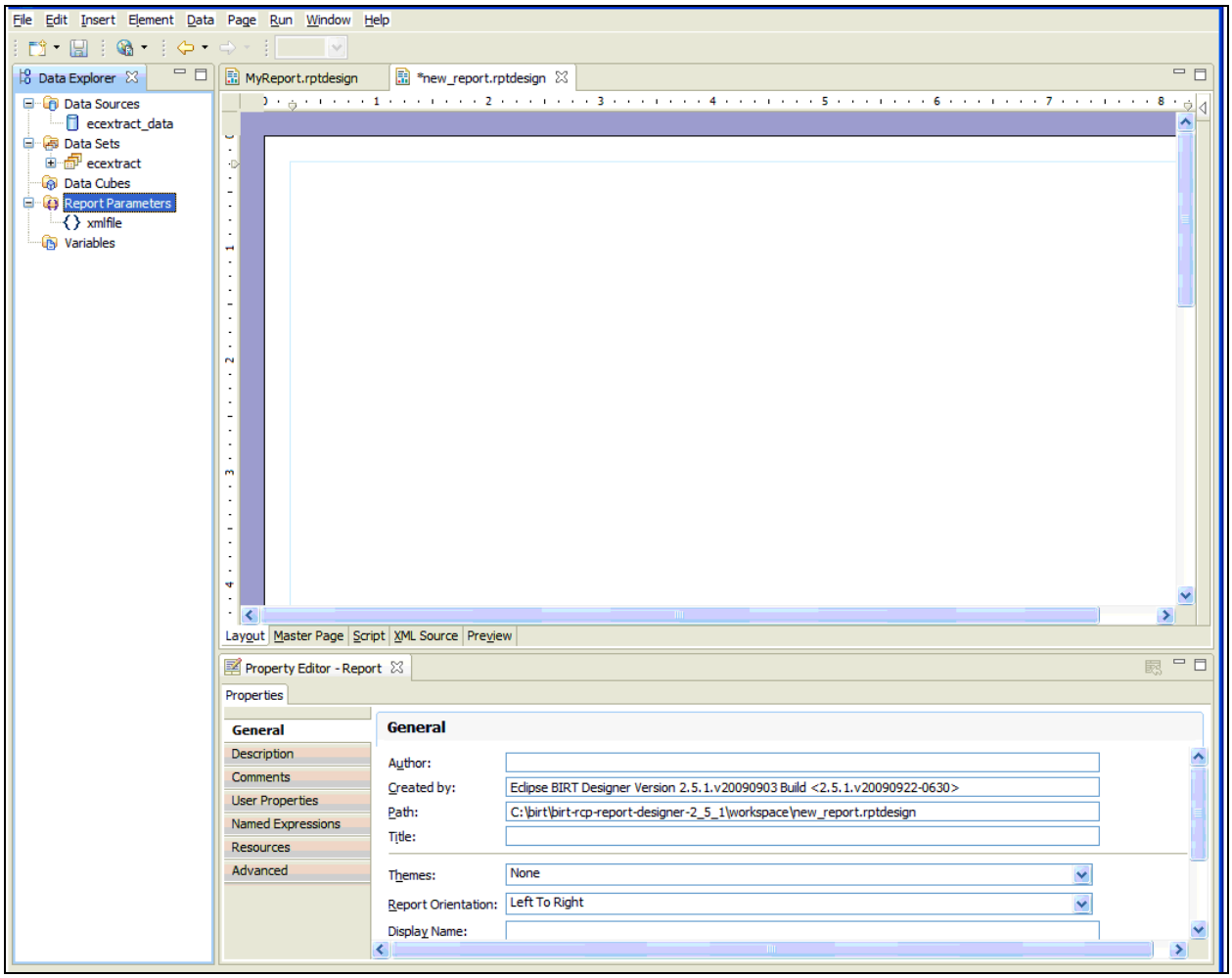
Items to copy into the new report:

- From Data Sources, copy ecextract_data
- From Data Sets, copy ecextract
- From Report Parameters, copy xmlfile

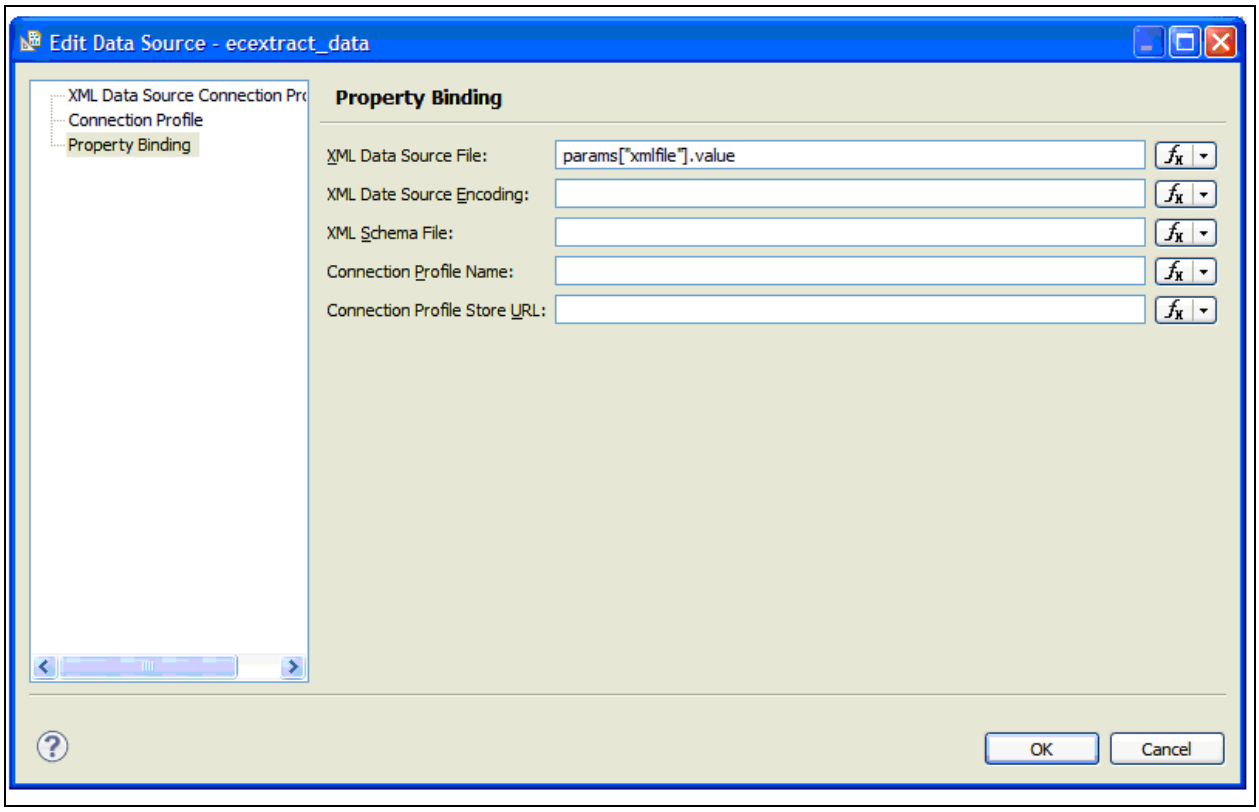
See the next screen example to help find these items.



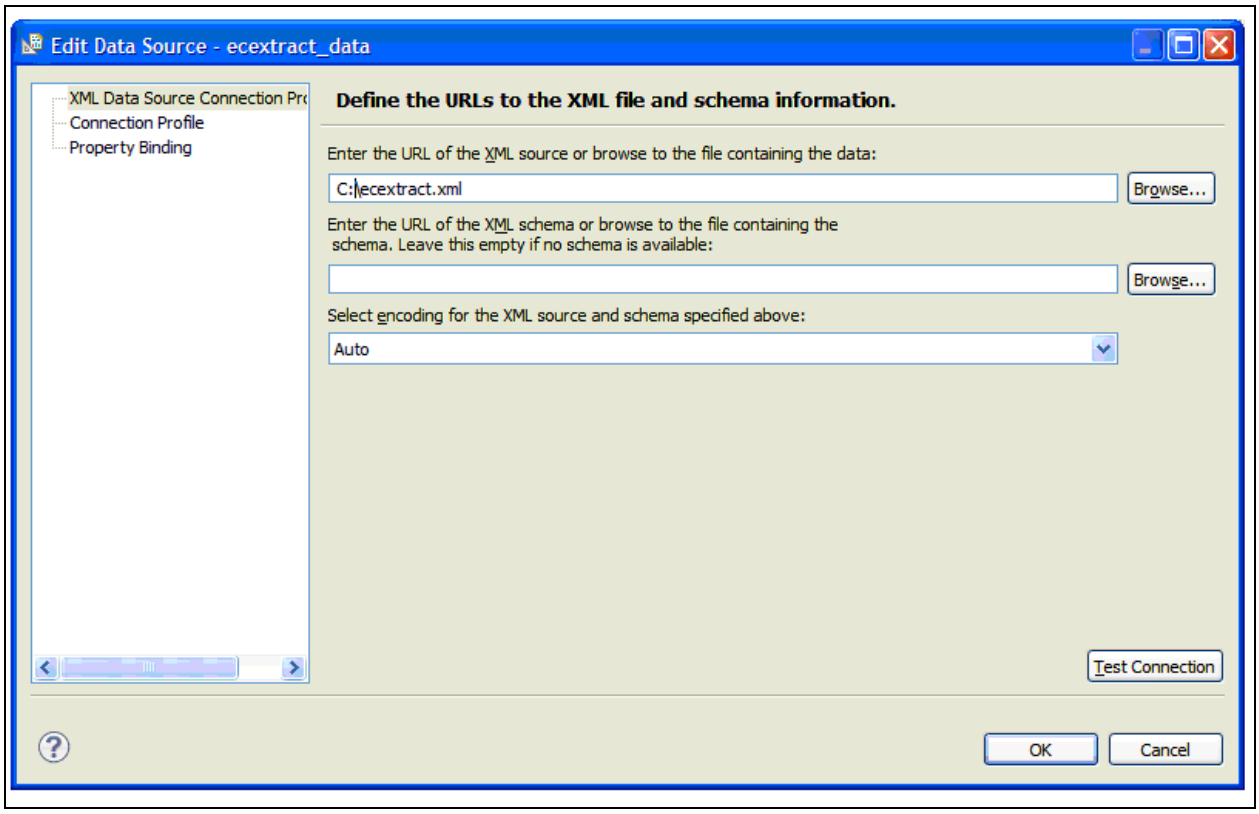
After copying these items, your new report design will look like the following example:



Double-click the Data Sources > `ecextract_data` file to ensure the XML file is referenced as a parameter. If it is not, type in the value as shown below.

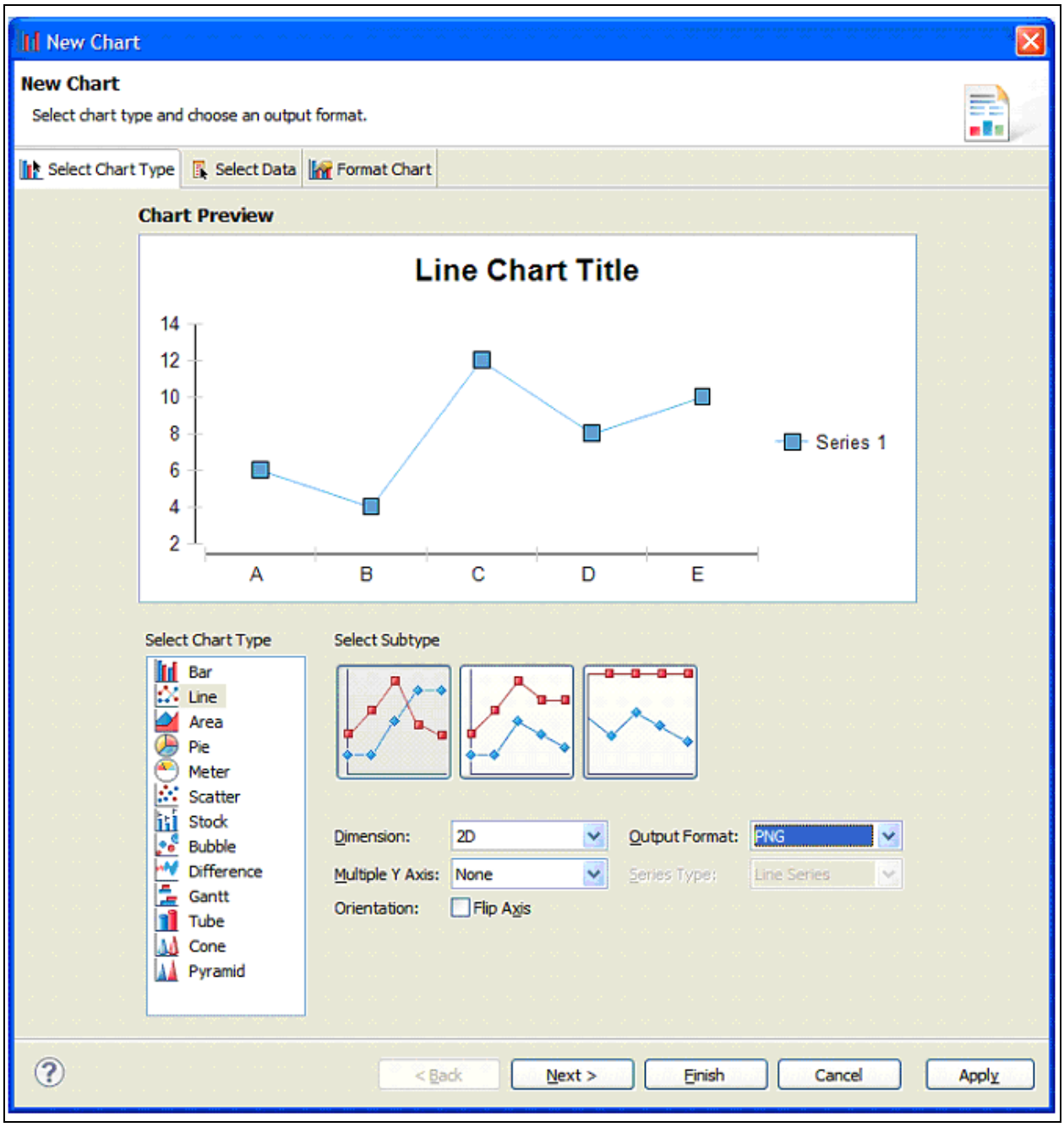


Double-click Data Sources > `ecextract_data` to ensure the XML Data Source Connection refers to the data file you downloaded earlier. In this example the file is `C:\ecextract.xml`.



From the Palette window, drag a Chart object into your report. Your view will be similar to the following:

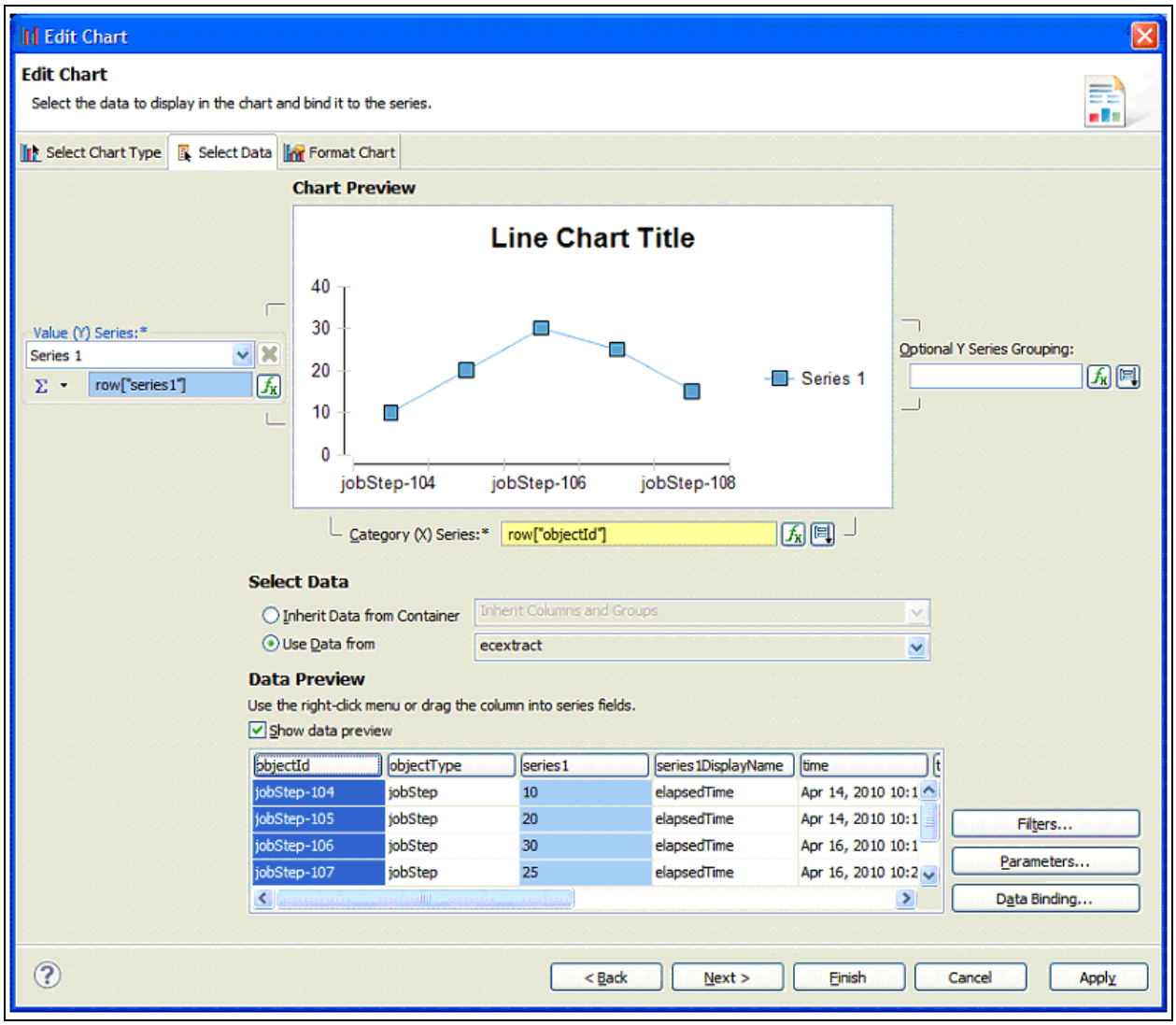
In the New Chart dialog, select Line under Select Chart Type and browse to select PNG for the Output Format.



Click **Next**.

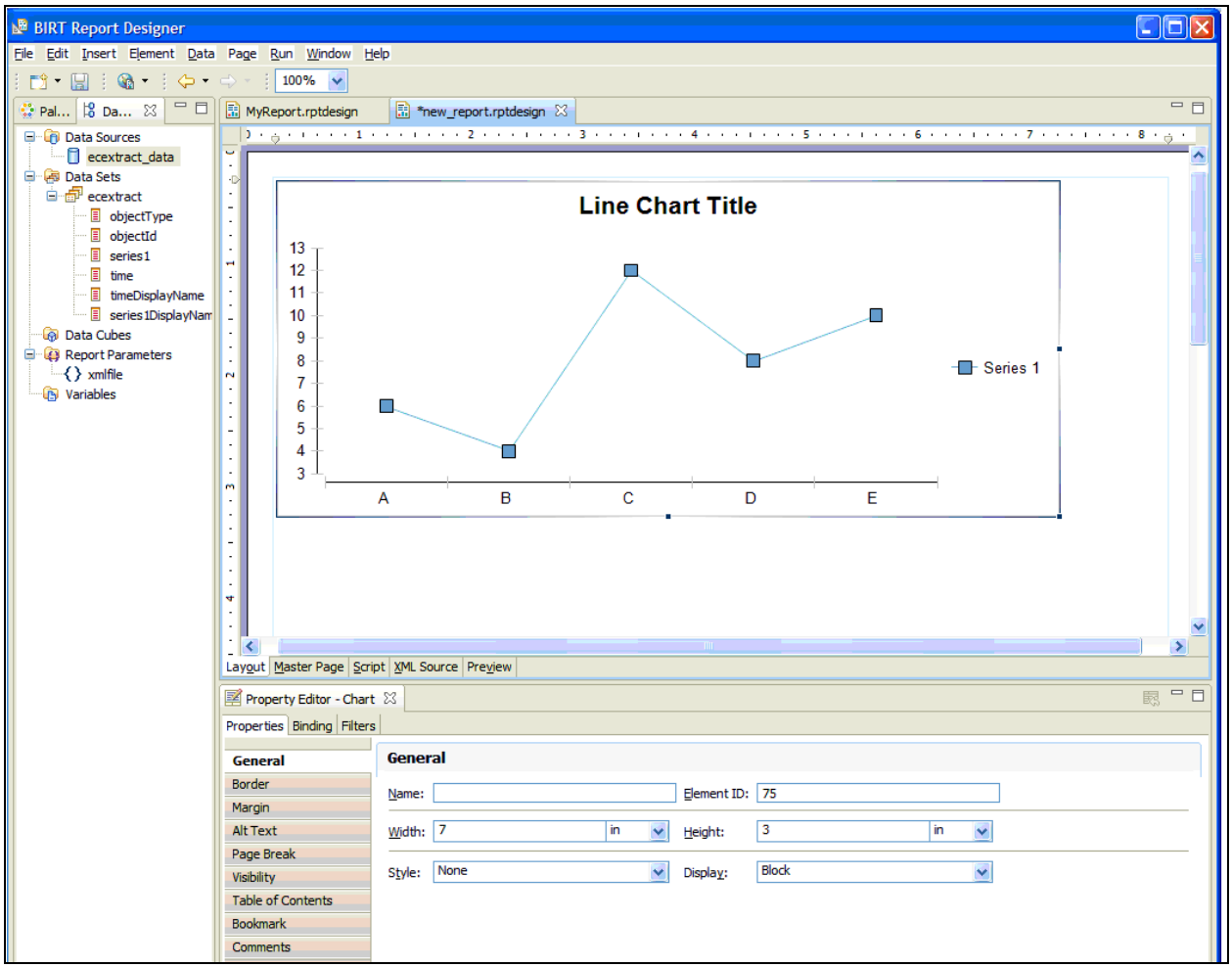
On the next screen:

- Under Select Data, browse to select ecextract.
- Make sure the Use Data From radio button is selected.
- Drag the "series1" header to the Value (Y) Series field.
- Drag the "objectId" header to the Category (X) Series field.



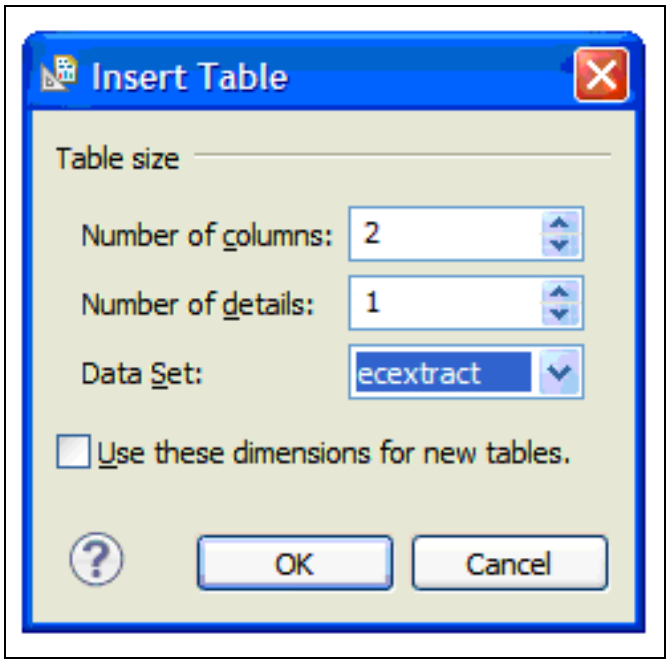
Click **Next** to apply any other formatting changes you choose, then click **Finish**.

On the next screen, after clicking Finish, you have to option to resize the chart as you wish.

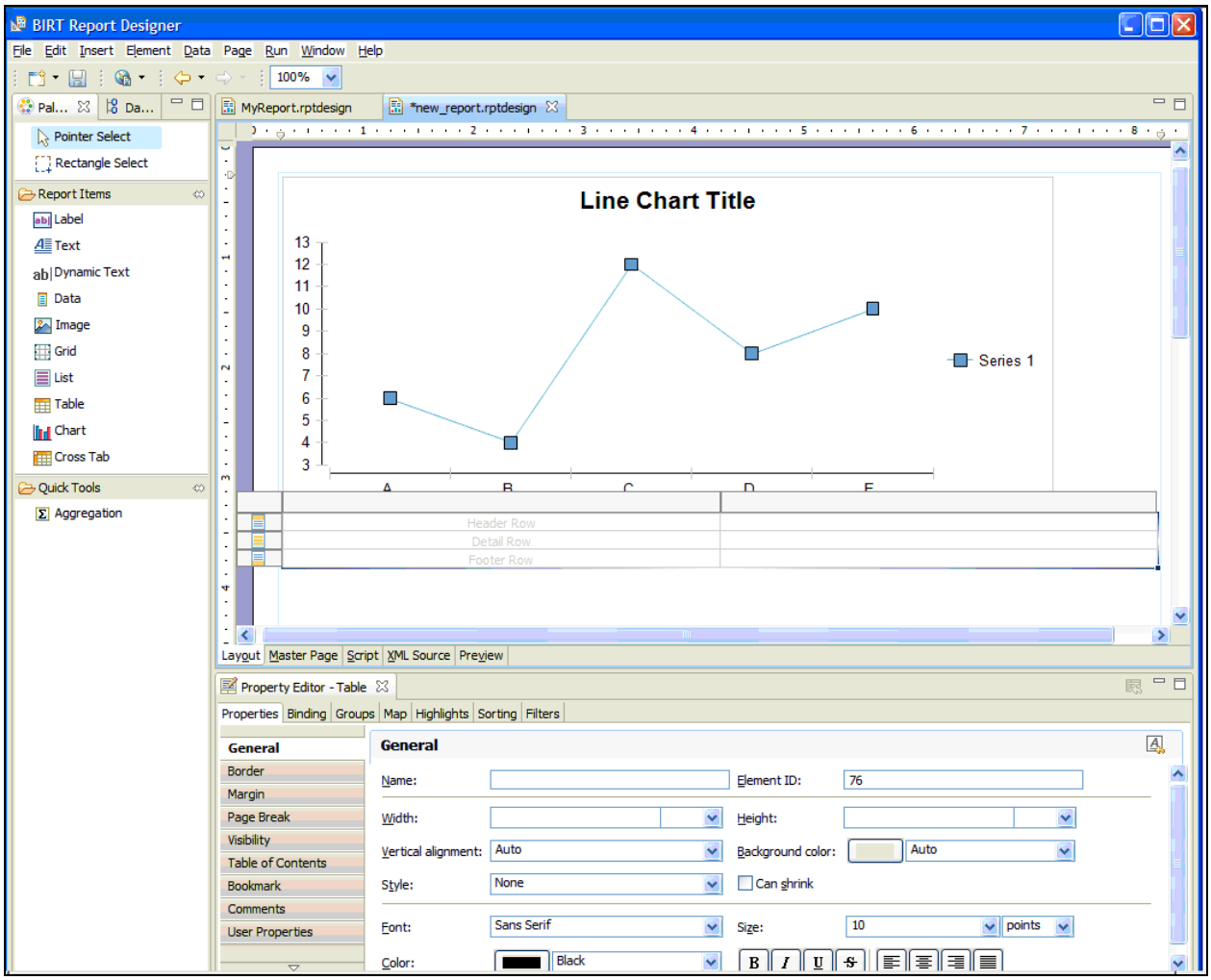


You are now ready to drag a Table item from the Palette to the report layout below the chart.

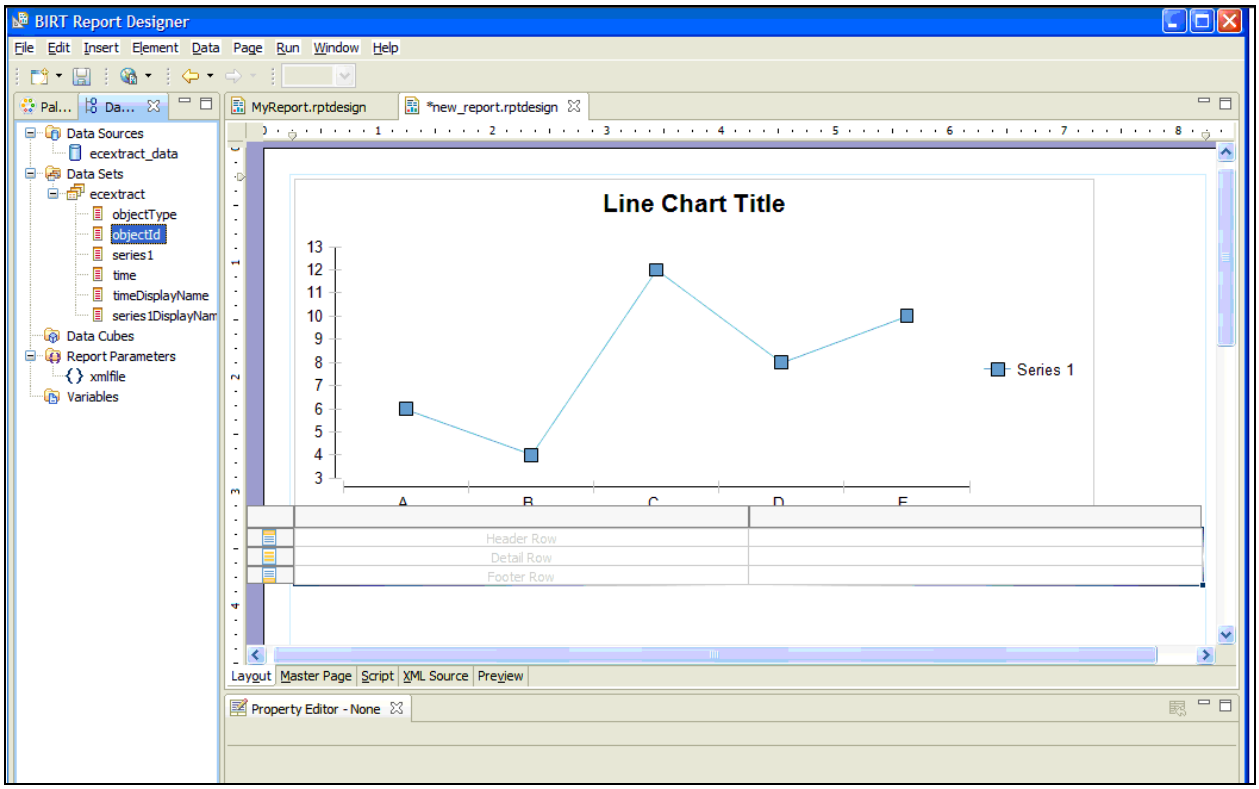
Select "2" columns and the ecextract dataset.


Insert Table
Table size
Number of columns:
2
Number of details:
1
Data Set:
ecextract
Use these dimensions for new tables.
OK
Cancel

Click **OK**.



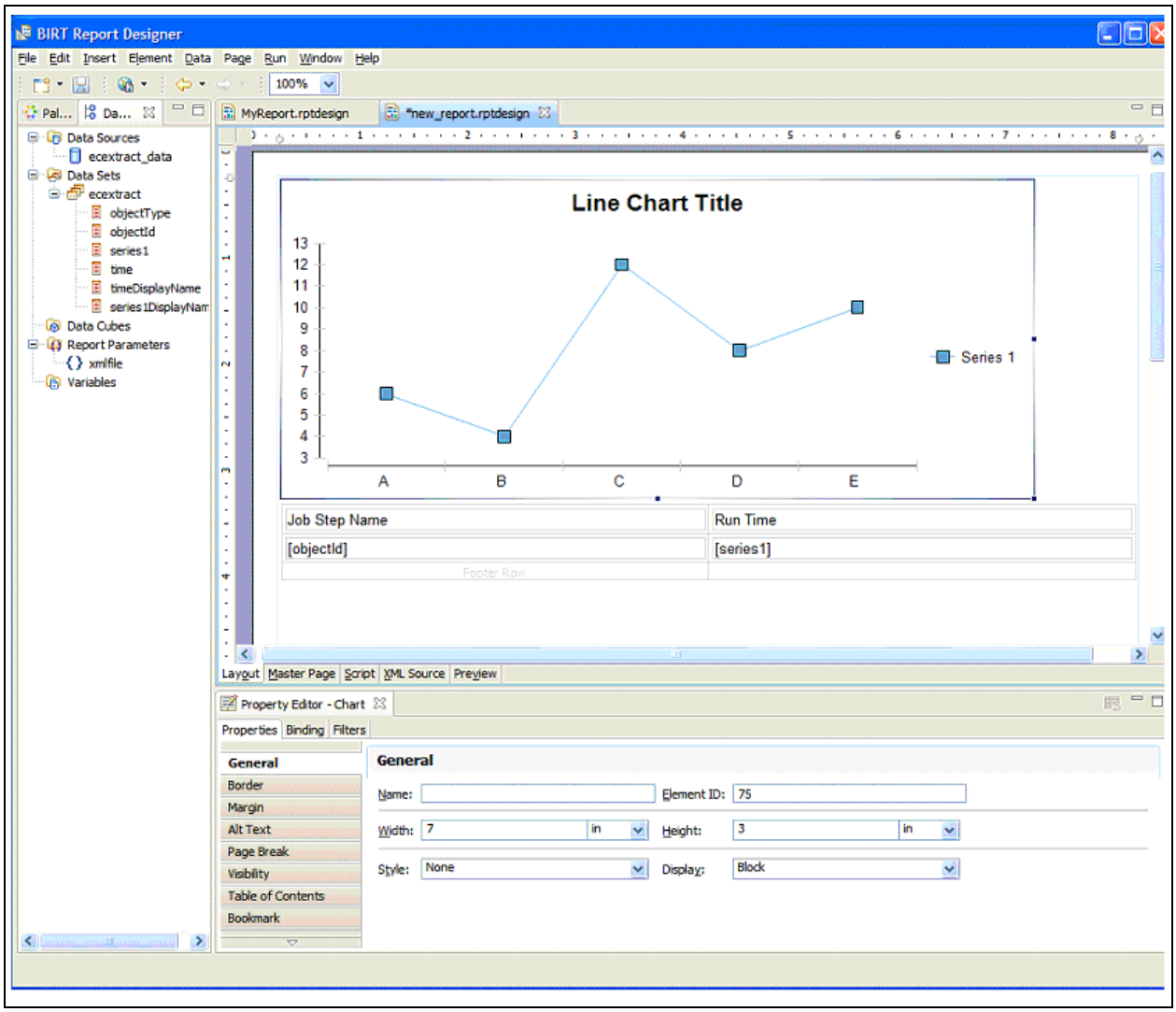
Select the Data Explorer tab and expand Data Sets > `ecextract`. See the next screen.



Now you are ready to:

- Drag "objectId" to the left Detail Row.
- Drag "series1" to the right Detail Row.
- Replace the Header Row for "objectId" and "series1" with Job Step Name and Run Time, respectively.

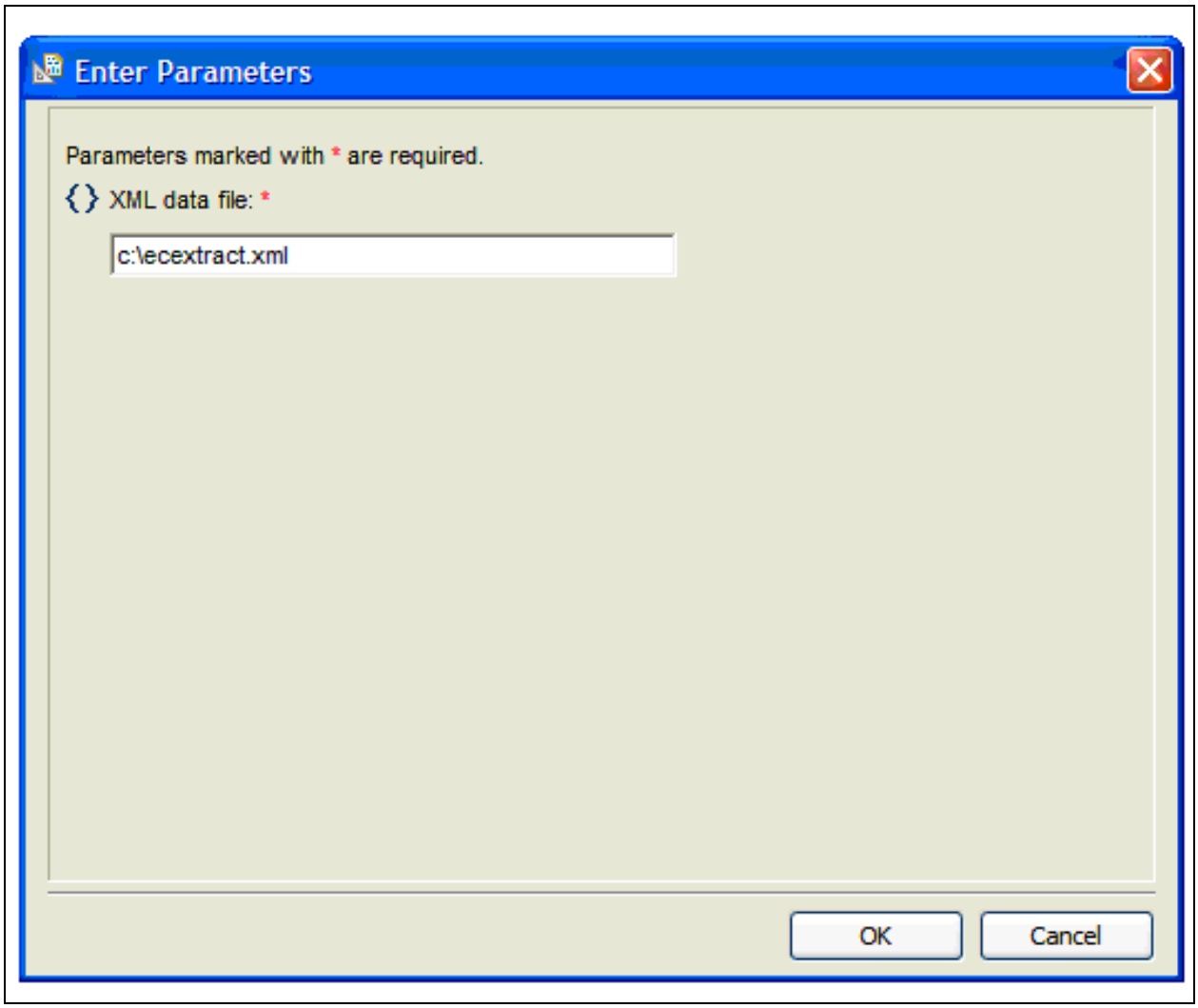
See the next screen example.

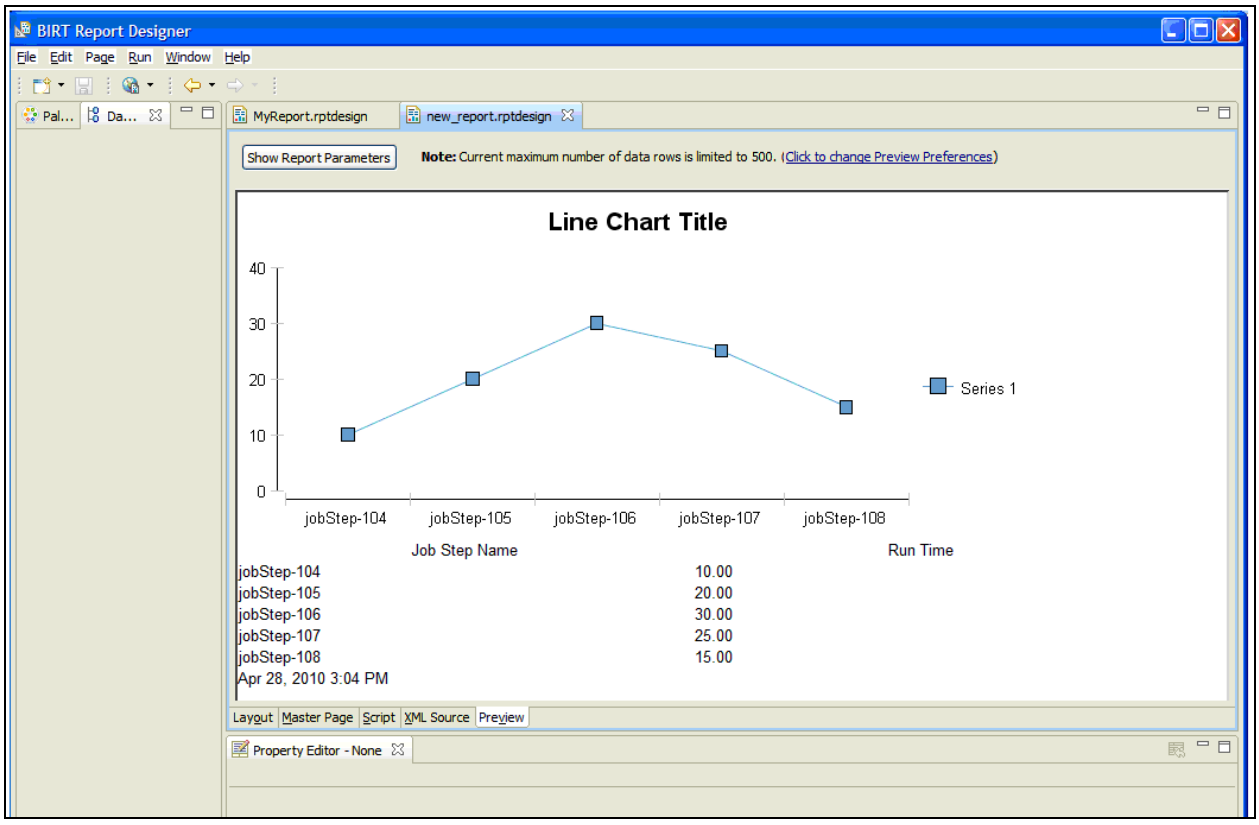


Select Preview to see your report.

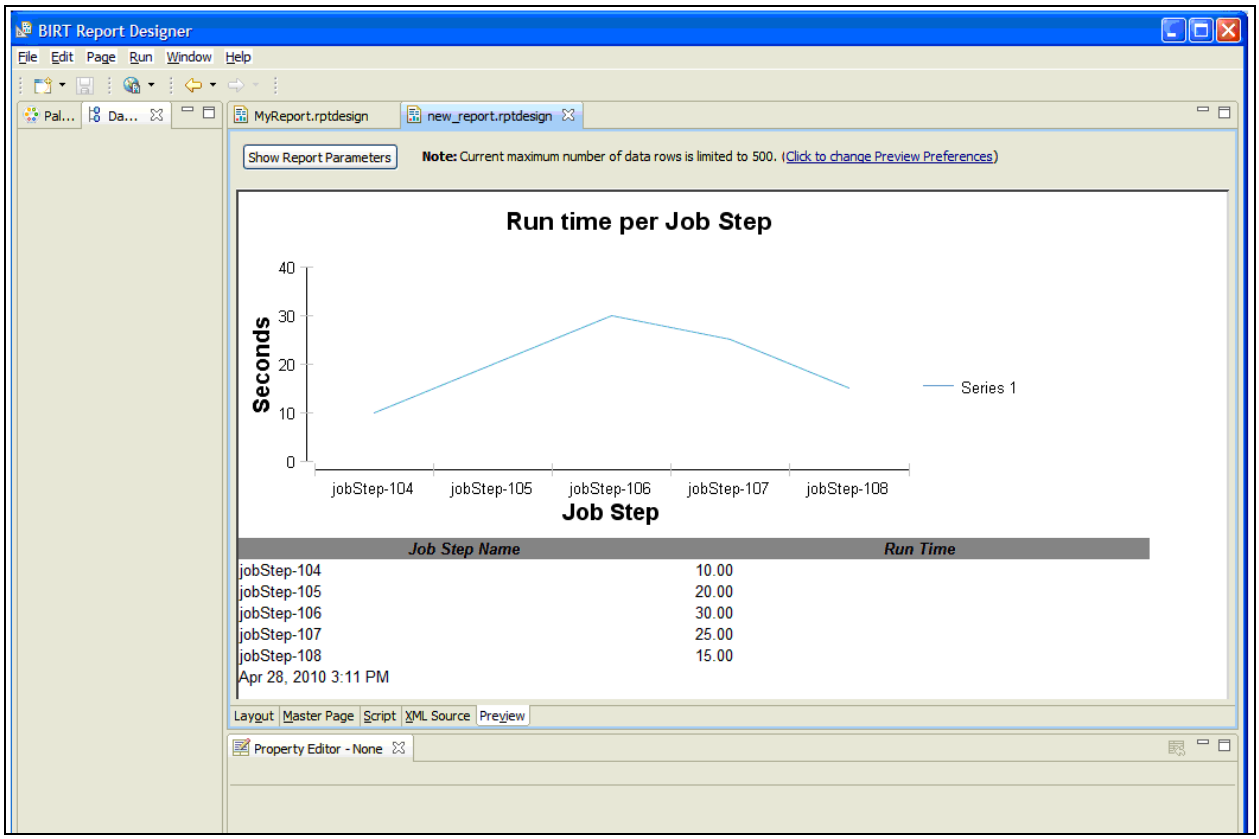
In the "Enter Parameters" dialog, type the location of your ecextract.xml data file.

Click **OK**.


Enter Parameters
Parameters marked with * are required.
XML data file: *
c:\ecextract.xml
OK
Cancel



At this point, you can continue to make formatting changes to make your custom report look exactly how you want it to look. Our example report, when finished, looked like the following example::



Deploy your custom report

If you have not already done so, save your new report design now. We have named our report design "`JobStepRunTime.rptdesign`".

The new report design needs to be stored in a location accessible to the ElectricFlow Agent that will run the report. As in the previous example, we will use the ElectricFlow plugins directory for this purpose.

- Save the JobStepRunTime.rptdesign file to a new JobStepRunTimeReport directory in the ElectricFlow plugins directory.

  The default location of the ElectricFlow plugins directory is:

  On Linux: `/opt/electriccloud/electriccommander/plugins`

  On Windows: `C:\Documents and Settings\All Users\Application Data\Electric Cloud\ElectricCommander\plugins`

  Linux example—The following commands will make a copy of the Value Over Time report file components and then rename the files:

  ```
  $ cd /opt/electriccloud/electriccommander/plugins

  $ mkdir JobStepRunTimeReport
  ```

- Now copy the previously saved JobStepRunTime.rptdesign file to the JobStepRunTimeReport directory.

- Make a copy of the RunReport_ValueOverTime procedure from the EC-Reports plugin that performs the steps to generate a report, including extracting data, running the report design, and registering the report output to appear on the Report tab of the Project Details page. In this example, we will create a Project called "JobStepRunTime" and the procedure we copy will be called "RunReport_JobStepRunTime".

```
$ ectool createProject JobStepRunTime

$ ectool clone --cloneName /projects/JobStepRunTime/procedures/RunReport_
JobStepRunTime

      --projectName /plugins/EC-Reports/project --procedureName RunReport_
ValueOverTime

<response requestId="1"

      <cloneName>RunReport_JobStepRunTime</cloneName>

</response>
```

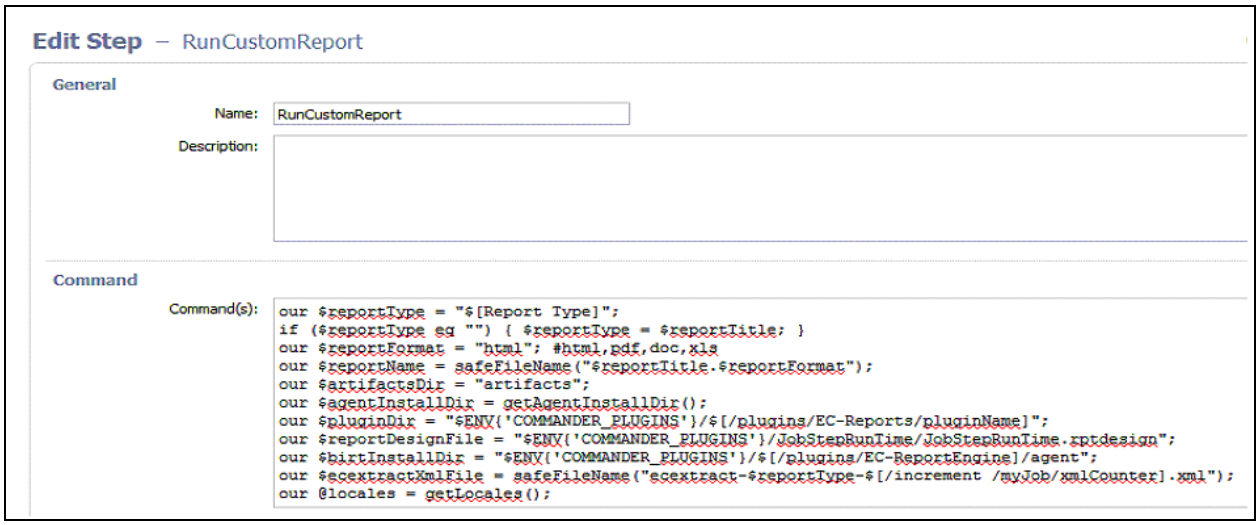
- Edit the "RunReport_JobStepRunTime" procedure in project "JobStepRunTime" to reference the report design file copied earlier.
- Navigate to project JobStepRunTime, procedure RunReport_JobStepRunTime and select the RunCustomReport step to edit the Command block:

Find the following line:

```
our $reportDesignFile =
"$pluginDir/agent/reports/ValueOverTime/ValueOverTime.rptdesign";
```

and change it to:

```
our $reportDesignFile = "$ENV{'COMMANDER_PLUGINS'}/
JobStepRunTime/JobStepRunTime.rptdesign";
```

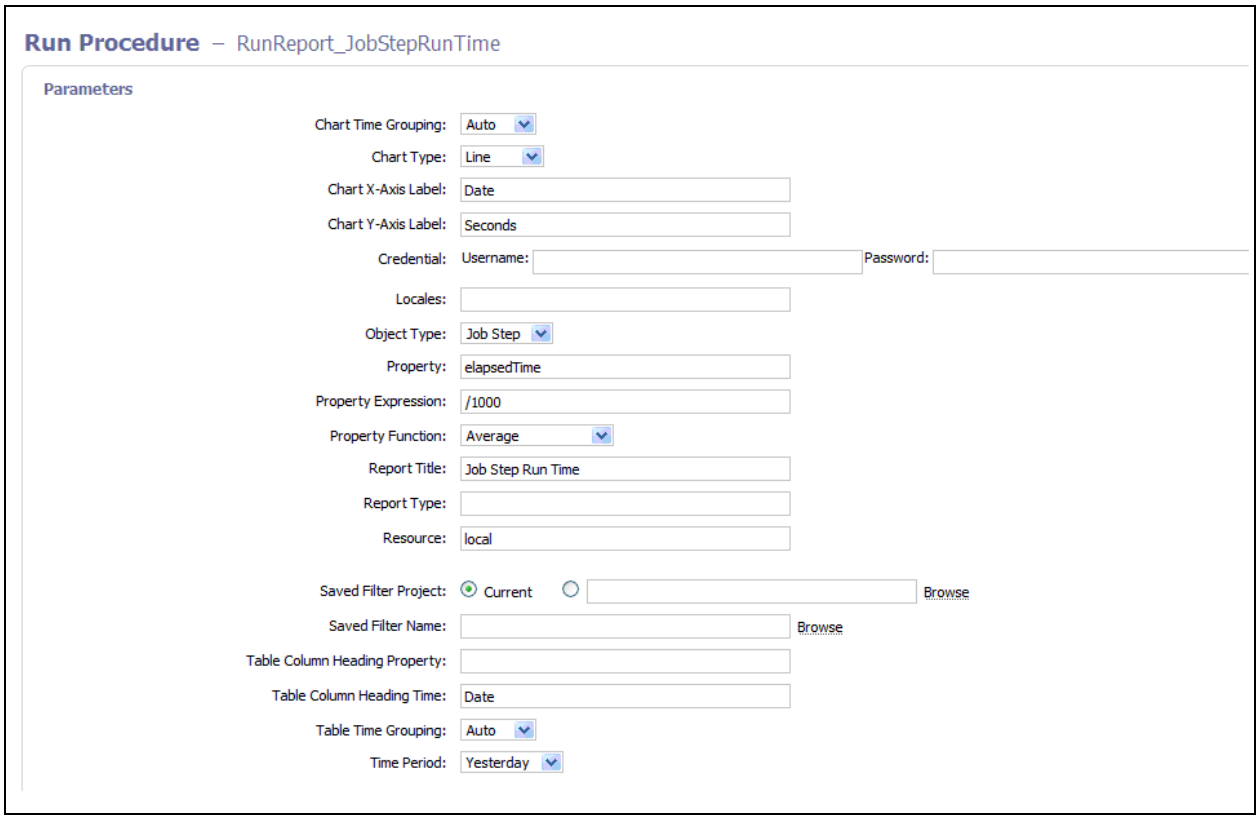


Click **OK** to save the change.

Test your new report

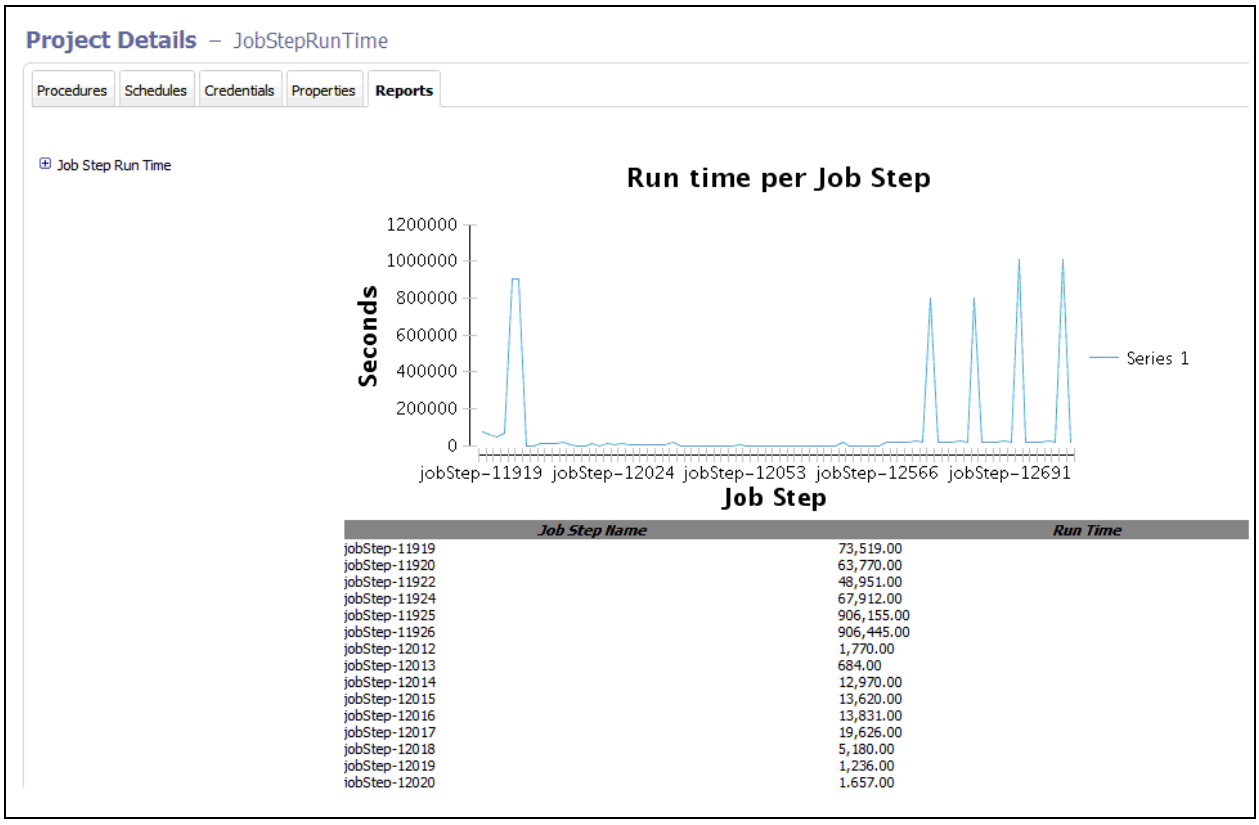
Run the report by running the RunReport_JobStepRunTime procedure in the JobStepRunTime project. On the Run Procedure page, change the values of the following parameters:

- Object Type => Job Step
- Property Expression => /1000
- Report Title => Job Step Run Time



Click **Run**.

When the job completes, your new report will be available in the Links section of the Job Details page and from the Reports tab in the JobStepRunTime project.



System Health Monitoring

You can also use StatsD and Graphite to generate custom reports for system health monitoring in ElectricFlow. After StatsD and Graphite are enabled, reports about a predefined list of items are generated.

This Help topic does not include instructions to set up StatsD and Graphite. Go to the following sites for documentation and tutorials:

- https://github.com/etsy/statsd/
- http://graphite.readthedocs.org/en/latest/index.html#
- https://www.digitalocean.com/community/tutorials/an-introduction-to-tracking-statistics-with-graphite-statsd-and-collectd
- https://www.digitalocean.com/community/tutorials/installing-and-configuring-graphite-and-statsd-on-an-ubuntu-12-04-vps

Workflow Overview

Workflow objects

Access Control

Property Search Paths

Parameters

Sending Notifications

Workflow logs

A Workflow Tutorial

Use a Workflow to design and manage processes at a higher level than individual jobs. Workflows allow you to combine procedures into processes to create build-test-deploy lifecycles (for example). A workflow contains *states* and *transitions* you define to provide complete control over your workflow process. The ElectricFlow Workflow feature allows you to define an unlimited range of large or small lifecycle combinations to meet your needs.

Some benefits of defining a workflow include

- **Manual intervention**—Workflows make it easy to introduce manual decision points into your process. With workflows, you can enable specific users or groups to choose how the workflow should proceed. Using an email notification, you can notify one or more users about the workflow status. If the workflow is waiting for someone to manually intervene, an email (including a screen link) can provide immediate attention and access to choices for how the workflow should proceed.
- **Aggregation**—Workflows allow you to collect information from a set of related jobs into a single location. A workflow can act as a central place to monitor the full lifecycle of your build, test, and deploy process.
- **Looping**—Workflows provide the native ability to run a job more than once. You have full control over the loop condition—you can re-run the job any number of times, only re-run the job if it did not succeed, or make this decision based on a custom job property.
- **Branching**—Workflows allow you to run multiple jobs in parallel and wait for them to complete before moving to the next state in the workflow. If you combine branching with looping, you can re-run these jobs based on a condition you defined (for example, only re-run jobs that failed).
- **Multipathed**—Design a workflow to take multiple paths automatically, depending on the outcome of each state's action, and the pre- and post-triggers you defined for each state.

Basic workflow concepts

Workflows contain states—One or more states can be designated as "starting" states to provide multiple entry points into the workflow. When a workflow is launched, a starting state is specified. You may prefer to design a complex or multi-purpose workflow so you can start the workflow at a different state to complete only a portion of the workflow for a particular outcome. For example, you might choose to run only the test suite workflow section.

As each state in a workflow becomes active, it performs an *action*. The state's action can be to create a job from a procedure or to start a workflow from a workflow definition. When the job or called workflow completes, "On Completion" transitions are evaluated to see if the workflow should change to a different state, possibly based on the outcome of the action.

You can create as many state definitions as you need. For example, you might have a small workflow definition with only 5 states, calling 5 procedures, or you might create 50-100 or more states to process a like-number of procedures from multiple projects.

A state can send email notifications before or after an action to notify interested users about workflow progress or to request manual user intervention to move the workflow to the next state.

Transitions are used to move workflow progress from one state to another state. Four types of transitions are available to move a workflow to the next state:

- **On Enter** —Evaluates before sending email notifiers or starting the action
- **On Start** —Evaluates immediately after starting the action. These transitions are ignored if no action is specified for the source state.
- **On Completion** —Evaluates when the action completes. These transitions are ignored if no action is specified for the source state.
- On Completion transitions are taken only if the state is still active when the action completes. These transitions are ignored if the workflow has transitioned to another state.
- **Manual** —Evaluates when a user selects the transition in the UI and specifies parameters. The same action can occur using ectool or the Perl API by calling `transitionWorkflow`. Only users who have "execute" permission on the transition are allowed to use the manual transition.

Similar to other objects in the ElectricFlow system, workflow objects can contain access control privileges, properties, and parameters.

Workflow objects

Workflow objects are split into two types: ***Definition*** objects and ***Instance*** objects. Definition objects provide the template for a running workflow instance. You create a new workflow by defining a Workflow Definition along with its State Definition and Transition Definition objects.

When you run the workflow definition, the system creates a new Workflow object with an equivalent set of State and Transition objects that represent the run-time instances of the workflow definition.

> **Note:** We omit the "Instance" qualifier for brevity in the API and the UI.

Workflow Definition

This is the top-level workflow object, which is a container for states, and transitions, and other information defining your workflow. In the same way a procedure defines the behavior of a job, the workflow definition defines the behavior of a workflow.

Create a project to get started:

1. Select the Projects tab, then click the **Create Project** link. After creating a project, its name will appear in the table on the Projects page.
2. Select your project name to go to its Project Details page, select the Workflow Definitions subtab, then select the **Create Workflow Definition** link.
3. On the New Workflow Definition page, name your workflow definition. Access the **Help** link on that page if you need help creating the Workflow Name Template (similar to the Job Name Template). Click **Save**.
4. Now, on the Workflow Definition Details page (Graph view), you are ready to create the first state for your workflow.

For an example of how this page might look as you add states and transitions to your workflow definition, see Workflow Definition Details for details about the Workflow Definition Details page. Notice also that you can add properties to the workflow definition at this point.

State Definition

Each workflow can contain one or more states. Defining states for a workflow is analogous to defining steps for a procedure.

You might choose to name your states for their intended purpose (for example: waiting for user, building, testing, or test1, test2, and so on) just as you might name steps for your procedure. You can define multiple states as "startable" so you can choose different starting points when you begin using the workflow. By default, the first defined state is always "startable".

As you create states, they will be displayed in the graph on the Workflow Definition Details page. Selecting a state in the Graph view opens the State Definition panel to define:

- Action to perform
- Parameters required by the action
- Transitions
- Parameters
- Email notifiers
- Properties that you want to assign

Transition Definition

Each state can contain one or more transitions. The transition definition requires a name for the transition. This transition name will appear on the Workflow Definition Details graph and the Workflow Definition Details List view.

You can define one or more transitions for each state. When defining a transition, you can:

- Add a text description for your reference
- Select the "target" state
- Select the type of transition trigger you want to use (On Enter, On Start, On Completion, or Manual)
  See the Workflow Definition Details Help topic for trigger type definitions and other information for defining a transition
- Select (or enter) a Condition to specify when the transition is allowed to trigger
- Assign parameters to a transition. These parameters are passed to the target state

Workflow

While we generically refer to or think of a "workflow" as an automated process to perform multiple tasks, it is actually the end result of running a Workflow Definition. Running a Workflow Definition produces a Workflow, which is analogous to running a Procedure, which produces a Job.

After you start running Workflow Definitions, select the main Workflow tab to see all workflows that ran (or are still running) listed in a table, each linked to its own Workflow Details page. The Workflow Details page main view is a graphical representation of the workflow, and also provides links to stop the workflow process, run the workflow again, see the workflow log, and so on.

A Workflow contains a pointer to:

- Its workflow definition
- Sets of states and their transitions
- The current and initial state
- Parameters to the initial state

- A "complete" value to specify whether or not the workflow is complete
- A log containing transitions taken and user events

State

The run-time instance of a State Definition is a State. The state is part of a workflow and contains information about actual parameter values specified by the user when the workflow was run or from transitions taken to enter the state.

If the state has an *action*, then the state contains information about the most recently created job or workflow. Because a state may be entered multiple times, the state contains a copy of the state definition properties needed to launch the action again.

After the state is created, it no longer depends on the state definition, so any changes made to the definition will not affect workflows already running. Changes do not take effect until the next time the workflow runs.

Transition

The run-time instance of a Transition Definition is a Transition. The transition is part of a state and contains a copy of information from the transition definition. Each transition object corresponds to a transition definition, but once defined, each transition is recognized by its unique name. A transition contains unexpanded actual parameters cloned from definition, expanded and passed to the target state on entry, the Javascript condition, and the trigger type instructing the transition when to occur.

After the transition is created, it no longer depends on the transition definition, so any changes made to the definition will not affect workflows already running. Changes do not take effect until the next time the workflow runs.

A transition moves the workflow from its state to a *target* state.

Access Control

Setting access control privileges for those persons who can or cannot run a workflow or execute a manual transition may relate to security policies in your organization.

Specific workflow access control notes:

- All workflow, state, and transition ACLs are copied from the definition object to their corresponding instance when the workflow is run initially.
- To run a workflow, you must have execute permission on the state definition you are using to start the workflow.
- To take a manual transition, you need execute permission on that transition.
- To strictly control who can take a manual transition:
  Grant read and modify privileges to anyone who can view or edit the workflow, but only give execute permission to those users or groups who can take the manual transition.
- You can control who can start each of the startable states by setting ACLs on state definitions accordingly.

For more information about access control, including examples you might use, see the Access Control Help topic.

Property Search Paths

A workflow serves as a hub of information that can be shared between states, transitions, jobs, and other workflows. Depending on the context, an implicit search path is used to reference a property by name. Using a simple property name like `$[someCustomProperty]` "walks" down the following paths until it finds a property by that name. If the search fails to find a property, an error is produced.

The search path followed to resolve a simple property name depends on where the reference occurs.

Transition context

1. Transition property sheet
2. State property sheet—Includes parameter values passed to that state and any custom properties.
   Because a state can be entered multiple times, parameter values are available for the most recent entry only.
3. Starting state property sheet—Used to get to parameter values for the initial entry into the workflow.
4. Workflow property sheet—Used to access information shared across the workflow.

State context

1. State property sheet—Includes parameter values passed to that state and any custom properties.
   Because a state can be entered multiple times, parameter values are available only for the most recent entry.
2. Starting state property sheet—Used to get to parameter values for the initial entry into the workflow.
3. Workflow property sheet—Used to access information shared across the workflow.

For information on context-relative property path shortcuts that allow passing information from one object to another or accessing a property from different parts of the workflow, review the Properties Help topic. The Properties Help topic also includes all intrinsic workflow-related properties.

Parameters

States may use formal parameters to determine what information to provide when the workflow enters that state. Parameter values can come from the initial run workflow call or from a subsequent transition into the state.

An automatic transition must specify all values required by the target state. A manual transition may omit some values to be passed to its target state. If a manual transition does not specify all values needed by the target state, then any missing values are collected from the user when the transition is taken.

Parameters to the starting state for the workflow are collected when the workflow is launched. These parameters are accessible throughout the workflow and act like global workflow parameters. See Property Search Paths.

It is important to distinguish between formal parameters on a state and formal parameters on the action the state calls (procedure or workflow starting state). The procedure (or workflow) may have its own parameters, so the state is responsible for entering required actual parameter values. In some

cases, values may be static values, in other cases the values may come from the state's parameters or those of the starting state (for example, global workflow parameters).

Parameters on the calling state do not need to be the same as the parameters on the action:

- A state parameter value can be passed as the value for an action's parameter by specifying a property reference in the value field on the Action tab for the state definition.
- All parameters on workflow current and starting states are accessible with a simple property reference (for example, `'$[param1]'`)
- Parameters on a workflow starting state act like global workflow parameters because they are available everywhere within the workflow.

Sending Notifications

A state can be defined to send email notifications when it is entered, when it starts its action, or when the action completes. Notifiers are created by visiting the Notifications tab on the State Definition panel. See Email Notifier for details on defining email notifiers. Sample notifier templates are provided also.

Workflow Logs

Each workflow keeps a log of events that relate to the workflow. The log contains information about the evaluation of transition conditions, transitions taken, actions started, and other information that might prove useful when debugging a workflow. Access the log by clicking the **View Log** link on the Workflow Details page. For more information, click here to go to the Workflow Log Help topic, which contains a screen example of the Workflow Log page.

Visualizing a Workflow

Two Workflow web pages, Workflow Definition Details and Workflow Details, open to the Graph view.

- Workflow Definition Details page—As you build your workflow, you can see a visual representation of state and transition definitions you define.
- Workflow Details page—Displays a graph, updated in real-time, for your running or completed workflow.

Your workflow Graph (on either page) contains context-sensitive links to various options. Options are different depending on whether or not you are building or running your workflow. Workflow Definition Details and Workflow Details Help topics include information about the context-sensitive link options provided within the workflow graph.

The next section, Building a Workflow—a Tutorial on page 1576, begins with a "sketched" representation of the workflow the tutorial will create. At the end of the tutorial, see the actual workflow graph created by the ElectricFlow workflow process.

Building a Workflow—a Tutorial

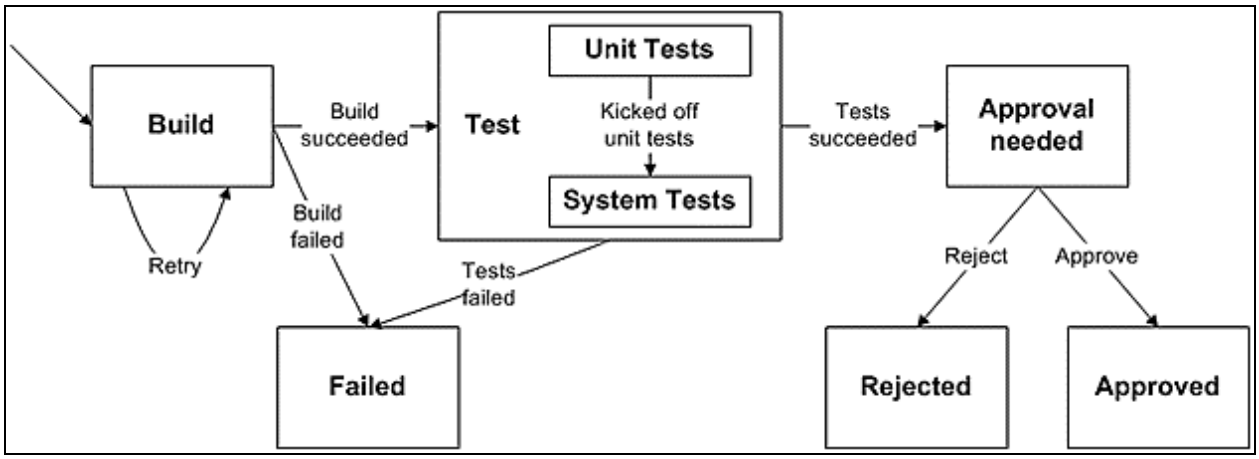
This overview tutorial is an example of how to create a "build-test-approve" workflow, building each state and transition we want the workflow to include, and building the workflow in the order we need to use to achieve our purpose. The diagram below outlines our workflow process.

You can read through the tutorial or actually recreate this example workflow on your machine. If you want to use this tutorial as a real, working example, you must do some initial setup first:

1. Create a project named "Sample"
2. Create a procedure name "BuildProduct"
3. On the "BuildProduct" procedure, add a parameter named "branch"

The following diagram shows the workflow that you will build:

- The starting state is *Build*, which could transition to the *Test* state or the *Failed* state
- Two test states—*Unit Tests* and *System Tests,* which could transition to a *Failed* state or an *Approval needed* state
- The *Approval needed* state could transition to one of two states—*Rejected* or *Approved*
- All transitions are represented by the lines between the states, which include "arrows" to show the possible workflow directions (depending on the outcome of each state's action



Best Practice tip

Designing your workflow on paper or whiteboard, creating your own state and transition diagram, is a great way to organize your thoughts and map your workflow process.

A summary of the component sections to build our workflow:

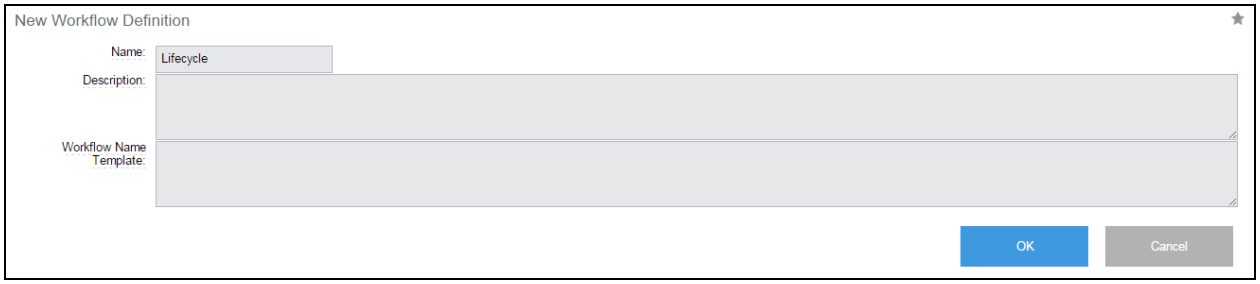
- Calling a job from within a workflow
- Collecting a parameter when a workflow is launched and passing its value to a job
- Retrying a state
- Automatically transitioning to different states based on the outcome of a job
- Invoking another subworkflow
- Running jobs in parallel
- Automatically transitioning to different states based on the outcome of jobs in a workflow
- Waiting for manual intervention
- Restricting who can take a manual transition
- Sending email notifiers

- Adding a global parameter to use later in the workflow
- Setting the name of your workflow
- Workflow list

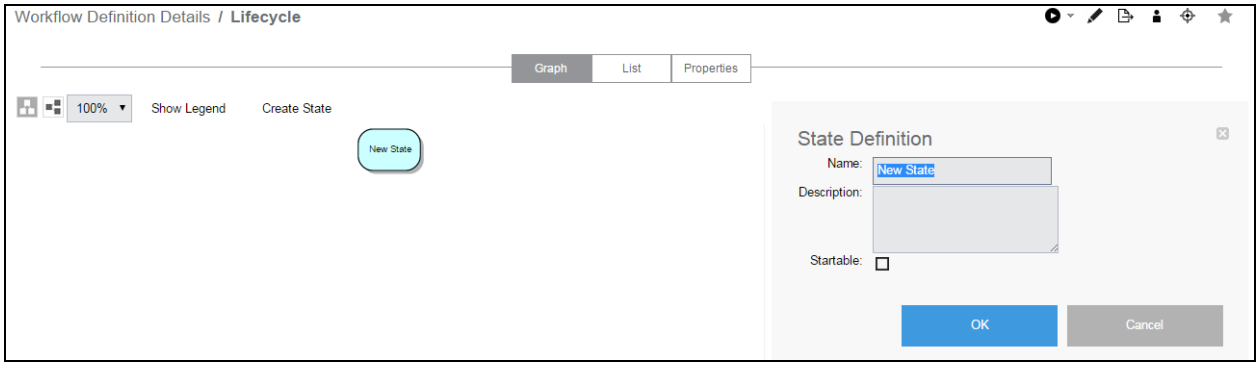
To begin calling a job from within a workflow

Or, running a job by calling its procedure from a workflow state.

1. Select the Projects tab, select the Sample project to go to the Project Details page, then select the Workflow Definitions subtab.
2. Click the **Create Workflow Definition** link to go to the New Workflow Definition page.
3. Create a new Workflow Definition named *Lifecycle.*



Because the workflow definition, *Lifecycle*, is a new definition and does not have any state or transition definitions, the State Definition panel opens, ready for you to create a new state definition for your workflow.



You can use the DSL Export ( ) button to download the objects as a DSL file.

4. Change the system-generated state Name to *Build* and click **OK**.

   The State Definition panel is displayed.

5. Specify the Action for this state, which is Procedure.
   In this example, the procedure *BuildProduct* already exists with a single parameter named *branch*.
6. Set the value of this parameter to "`3.0`".

   The *Build* state now has an Action associated with it.
7. Click **OK**.
   Note the shape of this state in the graph. The state shape will be different depending on the state Action selected.
8. Click **Run** to launch the workflow.
   - Running the workflow shows a Job was launched by the *Build* state. The workflow job completed successfully, indicated by the "check mark" on the state.
   - You can get more details about the job by clicking on the *Build* state, which opens the State Details panel, and clicking on the Job link displays the Job Details page.
   - Alternately, clicking on the Show History button opens the History panel to show what ran and the workflow outcome.

Collecting a parameter when a workflow is launched and passing its value to a job

Or, "prompting" for a parameter value when a workflow is launched and passing its value to a procedure.

1. Return to the Workflow Definition page for the **Lifecycle** workflow.
2. Click the *Build* state (from the graph) to open the State Definition panel, then select the Parameters tab.
3. Click the **Create Parameter** link to display the New Parameter page.
4. Create a parameter called "branch".
5. Add a description and make sure the Required? box is "checked" and click **OK**.

6. On the Workflow Definition Details page again, change the value passed to the job from 3.0 to `$[branch]`.
7. Now, when the workflow is launched, you will be prompted for the value of *branch* and you can specify a different value if necessary.

Retrying a state

If a State fails (technically, the job started by the workflow state), you may choose to retry it a few times before giving up.

1. Right-click on the *Build* state and select the **Create Transition** link to create a new transition from the *Build* state back to itself.
2. Create a new On Completion transition named "retry" from the *Build* state that returns to the same state when the job fails.
3. Add a limit of 3 retries by using the following JavaScript condition:
   ```
   $[/javascript mySubjob.outcome == "error" && myState.getProperty('/increment numberOfRetries')< 3; ]
   ```
   The new transition is displayed on the Workflow Definition Details page.
4. When we run the workflow, we see only the last job that was run.
5. Clicking the States tab and then the **Show All** link displays all jobs that were invoked by the *Build* state.

   Also, you can see the number of times a state was visited and which transitions were taken by clicking the **Show History** button.

Automatically transitioning to different states based on the outcome of a job

If the *Build* job fails, we might want to end the workflow. If the job succeeds, we might want to continue to the next stages of the workflow. In this section, you add new states to transition to, depending on the outcome of the job.

1. Create the *Test* state where the workflow will transition to when the build state succeeds.
2. Create a *Failed* state.
3. Create an On Completion transition from *Build* to *Test* when the job succeeds.

   Notice that the Condition dropdown menu provides a template condition called *Job success* that can be used here.
4. Create an On Completion transition from *Build* to *Failed* when the job fails.

   Notice that the Condition field remains blank. Because transitions are evaluated in order, if the workflow gets to this transition, we will know that the "success" case will be false. Leaving the Condition field "Always" allows us to handle all "non-successful" outcomes (error or warning).

   When you run the workflow, if the build succeeds the workflow will end up in the *Test* state.

   The workflow will end up in the *Failed* state if the build fails after trying three times.

Invoking another workflow

Just as a state can invoke a job as its "Action", it can invoke another workflow, which then becomes its subworkflow.

1. From the Workflow Definition Details page for *Lifecycle*, click on the *Test* state in the graph.
2. Specify the "Action" for the Test state, which is a Workflow.

   The *Test* state now has an Action associated with it. Because the *Test* state Action is Workflow, the object shape in the graph has changed.

   Running the workflow shows another workflow was launched by the *Test* state.
3. Click on the workflow link in the History panel to see more details about that workflow.

Running jobs in parallel

Run multiple jobs at the same time.

1. Look at the *AutomaticTests* workflow that was invoked (in the previous example) by the *Test* state in our main *Lifecycle* workflow.

   The first state in this workflow runs unit tests, and when the tests complete, the state automatically transitions to the second state that runs system tests.

   Now we will change the behavior of the transition to move from the *Unit tests* state to the *System tests* state as soon as the first job is launched.
2. Edit the existing transition by renaming it to *Kicked off unit tests* and changing it to an On Start trigger type.

   Now, when the *AutomaticTests* workflow is launched, both jobs will run in parallel. The workflow completes automatically when both jobs complete.

Automatically transitioning to different states based on the outcome of jobs in another workflow

If the jobs in the *Test* workflow fail, we will want the workflow to end. If the jobs succeed, we will want to continue to the next stages of a calling workflow.

1. Click the **Create State** link.
2. Create the *Approval needed* state that the workflow will transition to when tests succeed.

3. Create a transition named "Tests succeeded" from *Test* to *Approval needed* when all jobs in the workflow succeed.

   (Right-click on the *Test* state to create the transition.) The following JavaScript condition checks the outcome of all subjobs for a subworkflow and evaluates to "true"
   only if they all succeeded:

   ```
   $[/javascript

        var success = "1";

        for (var stateName in mySubworkflow.states) {

          var subjob = mySubworkflow.states[stateName].subjob;

          if (subjob !=null && subjob.outcome != "success") {

            success = "0";

            break;

          }

        }

        success;

      ]
   ```

   Note that we created the *Approval needed* state first, then we created the *Tests succeeded* transition to connect the two states.

4. Create a transition from the *Test* to the *Failed* state when either of the jobs in the workflow fails.

   Notice the Condition field remains blank. Because transitions are evaluated in order, if the workflow gets to this transition we know the success case was "false".

   When you run the workflow, if the test jobs succeed, the workflow will end up in the *Approval needed* state.

Waiting for manual intervention

Allow users to choose how to progress the workflow.

1. From the Workflow Definition Details page for *Lifecycle*, create an *Approved* state and a *Rejected* state for the workflow to transition to when a user approves or rejects the build.
2. For the *Approval needed* state, we create a manual transition called *Approve* that the user can select to progress the workflow to the *Approved* state.

3. Next, we create a manual transition called *Reject* that the user can select to progress the workflow to the *Rejected* state.

   When you run the workflow and the build and test phases succeed, the new manual transitions are made available to users on the Workflow Details page.

   > **Note:** If a user does not have execute permission for manual transitions, the Manual Transitions section will not be included in the summary section at the top of the Workflow Details page.

   If you choose the *Approve* transition, the workflow will end up in the *Approved* state. If you choose the *Reject* transition, the workflow will end up in the *Rejected* state.

Restricting who can take a manual transition

Only allow certain users to take manual transitions. From the Workflow Definition Details page for *Lifecycle*:

1. When the workflow is created, access control entries from the Transition Definition are copied to the corresponding transition.
   These are the access control entries for the *Approve* transition before we edit them (see below).
2. To add access control to a transition, select the transition or transition name.
3. When the Transition Definition panel opens, click the **Access Control** link.
4. When the Access Control page is displayed, click the **Add Group** link to add privileges for a group that can take this transition.

   > **Note: Note:** If you already have groups created, you can use them here or create two groups now: *Lifecycle Approvers* and *Lifecycle Modifiers*.

5. Allow read and execute access for a group named *Lifecycle Approvers*.

   These users can see and take a manual transition in a workflow.

6. Allow read, modify, and change permission access for a group named *Lifecycle Modifiers*.

   These users can edit the Transition Definition, but will not have access to take a manual transition in a workflow.

7. "Break the inheritance" so only specified groups have access to this transition.
8. Duplicate this access control configuration for the *Reject* transition.

   When a workflow is launched and reaches the *Approval needed* state, a user who is not a member of the *Lifecycle Approvers* group will not be allowed to execute any manual transitions. A user who is a member of the Lifecycle Approvers group can execute manual transitions.

   The Workflow Details page updates in real-time, which means you will see changes at the top of the page pertaining to the current state in the running workflow, and current changes in the graph also. For example, if the previous state had a manual transition, but the current state now processing does not have a manual transition, the Manual Transition section will not be displayed.

Sending email notifiers

Send email notifications so a user or group knows they need to take an action on a workflow.

1. From the Notifications tab for the *Approval needed* state definition, click the Notifiers tab.
2. Click the **Create Notifier** link.

   We want to create an On Enter email notifier. When a workflow enters the *Approval needed* state, this email notifier will be sent out.

   In this example, we want to send an email notification to the "approvers". The email will contain a link to the Workflow Details page and links to all available transitions.

3. Use the following formatting template for the email notifier.

   It includes a link to the ElectricFlow workflow, embeds a block of JavaScript that cycles through all the manual transitions from the active state, and has a link to go directly to those transitions. To ensure that the script works properly, use the script as is and make sure there are no white spaces before `<!DOCTYPE...>`, `<html>`, and `</html>`.

```
Subject: Workflow $[workflowName] is ready for approval

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"><html>
<body>
      <h2>Workflow Approval Required</h2>
      <a href="http://$[/server/hostName]
/commander/link/workflowDetails/projects/$[/myProject]
/workflows/$[/myWorkflow/workflowName]/s=Projects">View Workflow $[workflowName]
</a>
      <br /><br />
      $[/javascript
          var links = new String("");
          for (var tName in myWorkflow.activeState.transitions) {
          var transition = myWorkflow.activeState.transitions[tName];
          if (transition.trigger == "manual") {
             links += "<a href=\"http://$[/server/hostName]
/commander/link/transitionWorkflow/projects/$[/myProject]
/workflows/$[/myWorkflow/workflowName]/transitions/" + tName + "\">" + tName +
"</a><br />";
          }
        }
        links;
      ]
</body></html>
```

   This script is a sample of the type of script you can insert into the Formatting Template field illustrated in the following screen example. When a workflow is run and it reaches the *Approval needed* state, an email similar to the following example can be sent to specified destinations. Only users who have execute permission on these transitions will be able to approve them by clicking the link.

Adding a global parameter to use later in the workflow

Prompt for a parameter when the workflow is launched, and use its value later in the workflow.

1. Currently, there is one parameter for the starting state *Build*.
2. Click on the *Build* state in the graph and select the Parameters tab.
3. Click the **Create Parameter** link to create a second parameter.
4. Add a checkbox-type parameter named *skipTests* which, if selected, should skip the test suite after the build is completed.
5. Create an On Enter transition from *Test* to *Approval needed* if the *skipTests* parameter evaluates to "true".
   Because this is an On Enter transition, it will be taken before the automated tests workflow is started.
6. When the workflow is launched, you will see the *skipTests* checkbox.
7. If the parameter is checked, and after the *Build* state is completed, the workflow will immediately transition from *Test* to *Approval needed*, without starting the subworkflow.

   ooking at the workflow History, we see the *Test* state was entered, but the subworkflow (and subjobs) was not executed.

Setting the name of your workflow

Choose appropriate names for workflows so they are easy to identify.

1. If the Workflow Name Template was not set, workflows are created with a default name (`workflow <jobId> <timestamp>` or `workflow_239_201012091208`, for example).

   However, because you may have many different workflow definitions, you might want to create a distinguishing name for workflows launched from each definition.
2. From the Workflow Definition page for *Lifecycle*, click the **Edit** link at the top of the page.
3. Setting the Workflow Name Template to `lifecycle-$[branch]-workflow-$[/increment /myProject/lifecycle-counter]` achieves this goal.
   - `$[branch]` refers to the value of the branch parameter as specified when the workflow is launched.
   - `$[/increment /myProject/lifecycle-counter]` creates a property called `lifecycle-counter` on the project and increments the property every time a new workflow is launched so the names are unique.
4. Now, when the workflow is launched, its name is much more descriptive than the default workflow name.

Workflow List

Most of the time, you will want to interact with your workflow from the default Graph view. However, if you want to see your workflow in a list or table view, click the List button. See the Workflow Definition Details Help topic for detailed information about this page.

The transition order is important because transitions are evaluated in order and the first one with a condition that evaluates to 'true" is taken.

Another workflow example

An advanced sample Workflow Definition is pre-defined for you in the EC-Examples project. This is a "working" example, also modeling the workflow process, with which you can interact. To navigate to this example: Select the Projects tab > EC-Examples > Workflow Definitions subtab > then select Build Workflow With Approvals.

Workspaces and Disk Space Management

Overview

ElectricFlow provides each job step with an area on the disk it can use for "working files" and results. This disk area is called a **job workspace**. (The job workspace defaults to a directory under the workspace directory.)

- A job step can create whatever files it needs within its workspace, and ElectricFlow automatically places files in the workspace, such as step logs. Normally, a single workspace is shared by all steps in a job, but it is possible for different steps within a job to use different workspaces.
- The location of the job step workspace is displayed on the Job Details page for the job under "Details" for the step.
- Workspaces are grouped together, with workspaces for different jobs allocated as children of the same *workspace root*. When you define procedures, you can specify which workspace roots to use, then ElectricFlow creates a child directory within that workspace root for the job's steps to use.
- The term "workspace" is used at different times to refer to either a job workspace or a workspace root. Depending on the context, it should be easy to discern which "workspace" is being discussed or referenced.

Defining Workspaces

ElectricFlow can handle any number of workspace roots, which are defined by selecting the Cloud > Workspaces tabs on the web interface. To create a workspace root, provide a workspace name and three file paths for jobs to access the workspace:

- Workspace name—The workspace name must not have trailing spaces. When two workspaces have similar names, such as "Server1" and "Server1 " (with an extra space at the end), the database records them as two entries, one without trailing spaces and one with trailing spaces.

  If you run a job against one of these workspaces, it may hang and not proceed because ElectricFlow is waiting for a resource, an available workspace, or a step to complete.

- UNC path—A UNC path that Windows machines can use to access the workspace via the network, such as `//server/share/x/y/z`. You must ensure this path is accessible on any resource where the workspace is used.

- Drive path— Windows path, using a drive letter that refers to the same directory as the UNC path. Before running a job step using the workspace, the ElectricFlow agent creates a drive mapping for this path to match the UNC path. For example, if the UNC path is `//server/share/x/y/z` and the drive path is `N:`, the agent will map `N:` to `//server/share/x/y/z`. If the drive path is `N:/y/z` then the agent will map `N:` to `//server/share/x`.

  The directories in the drive path after the drive letter (`y/z` in the preceding example) must match the last directories in the UNC path.

- UNIX path—A path used on UNIX machines to access the workspace via the network, typically using an NFS mount. You must ensure appropriate mounts exist on all resources where the workspace is used.

  If your environment contains Windows machines only, you can omit the UNIX path, and if your environment contains UNIX machines only, you can omit the UNC and Drive paths.

During the ElectricFlow installation, you had the option of defining a workspace. If you defined a workspace, it was named "default" and will be the default workspace for all jobs, as described below.

By default, remote workspaces are accessed with the agent user's credentials. For Windows resources, you can override that user's credential by attaching a credential to the workspace. Using a credential attached to a workspace is useful if the agent machine is not a member of the domain, but the workspace is located on a machine in the domain. An attached credential for a domain user allows the agent to manipulate the workspace as that domain user.

Using Workspaces

The simplest way to use workspaces is to define a single workspace named "default" that is readable and writable on all resources, and never specify any other workspace information. With this approach the default workspace will be used for all steps in all jobs.

However, you can also arrange for different jobs to use different workspaces, or even for different steps within a single job to use different workspaces.

You can specify a workspace name in any of the following ElectricFlow objects:

- If you specify a workspace in the definition of a procedure step, it applies to that step.
- If you specify a workspace in the definition of a procedure, it applies to all steps in the procedure.
- If you specify a workspace in the definition of a project, it applies to all steps in all procedures defined in that project.
- If you specify a workspace in the definition of a resource, it applies to all steps assigned to that resource.

If workspaces are defined in several of these places, the workspace for a particular step is chosen in priority order corresponding to the list above: a workspace specified in a step takes priority over all others, followed by a workspace in the procedure, project, and finally resource. If no workspace is specified in any of these locations, the workspace named "default" is used.

Workspace Directory Names

A job workspace directory name, within its workspace root, is derived from the name of the job. For example, if a job named "job_63" uses a workspace whose root directory is `//ec/workspaces`, the job workspace directory will be `//ec/workspaces/job_63`. Changing the name of a job has no impact on workspace names: the initial name of the job is always used for workspace directory names.

A job is allocated a single job workspace only within a particular workspace root, which will be shared by all steps that specify that root. For example, suppose a job has 5 steps: 3 steps specify workspace A and 2 steps specify workspace B. The 3 steps that specify A will all share a single directory under A, and the other 2 steps will share a single directory under B.

Working Directories

By default, each job step begins execution with its current working directory set to the top-level directory in the job's workspace. A job step can override this location by specifying an explicit working directory. If a job step specifies a relative path for its working directory, the path is relative to the top-level workspace directory.

Workspace Accessibility

The simplest way to set up ElectricFlow is to make every workspace root accessible on every resource. This accessibility mode provides the most flexibility and simplifies ElectricFlow system management

Unfortunately, some environments cannot allow such universal access. For example, some sites do not provide file sharing between UNIX and Windows machines. In large enterprise environments where ElectricFlow is shared among multiple groups, you may want to partition resources among the groups and limit access by each group to the other groups' resources. In the most extreme case, you may not have network file sharing at all. ElectricFlow can handle all of these cases.

If a workspace root is not accessible on a particular resource, you should not use that workspace in any step that runs on the resource. If you do, the resource will not be able to create the job workspace directory and the step will stall until the issue is resolved. A workspace root (on a resource) must be both readable and writable to be used for job steps.

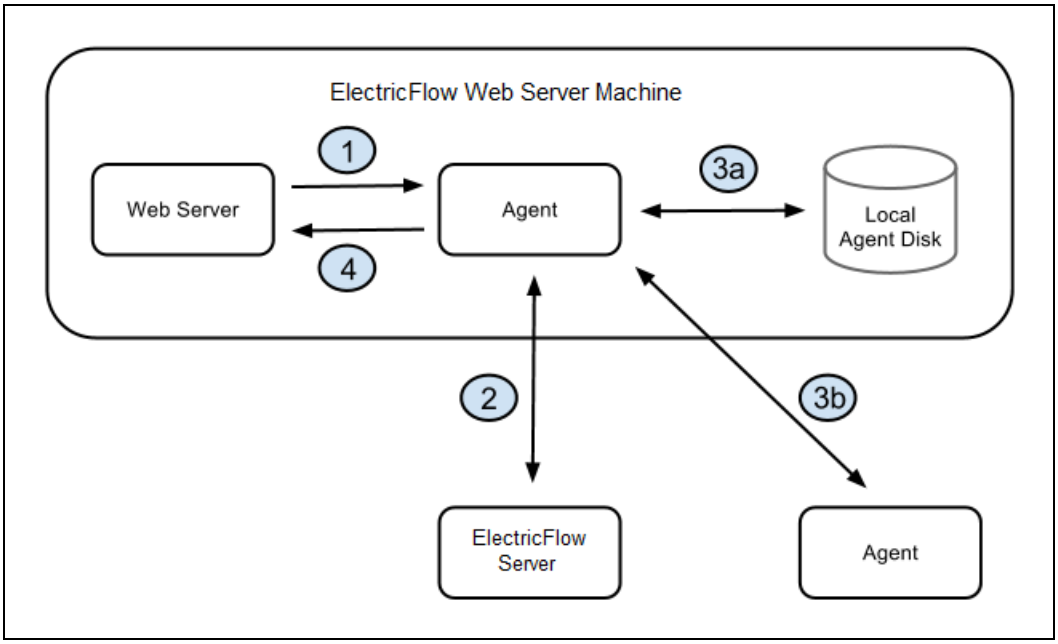
Local Workspaces (Disconnected Workspaces)

Normally, workspaces are shared, which means if the same workspace is used on different resources, they see the same set of files. However, it is possible to define a *local* workspace (also referred to as a disconnected workspace), which means it will refer to a different local directory on each resource where it is used.

- To define a local workspace on Windows, specify a local path such as `C:/Windows/Temp/ecworkspace` for the drive path.
- For UNIX, specify a local UNIX path such as `/usr/tmp/ecworkspace`. Local workspaces are advantageous because they do not require remote network access and access may be faster in some cases.

However, you cannot share workspace information among job steps running on different resources.

Note: Web servers "ask" the relevant agent for access to log files in workspaces. The agent streams the file back to the web server, which then streams the log file to your browser. However, this mechanism works only for Windows and Linux 4.2 (or later) agents. Solaris, MacOS, and HPUX agents must transfer log files to a workspace proactively so a Linux or Windows agent can access them for the web server—this task can be made easier, using the `ecremotefilecopy`tool. See ecremotefilecopy for details.



The diagram illustrates the architecture when the web server requests a log file and you have defined a local workspace.

1. The web server asks the local agent to retrieve the file from a workspace.
2. The agent (local to the web server) asks the ElectricFlow server how to find the workspace, and the ElectricFlow server replies with a route to an agent and validates the session ID.
3. The route to the agent that has the file could be: (a) the local agent itself or (b) a different agent that ran the step that produced the file.

   a. The agent (local to the web server) gets the file from its local disk.

   or

   b. The agent gets the file from some other agent.
4. The agent (local to the web server) sends the file to the web server.

Access control

You can restrict access to workspaces using the ElectricFlow access control mechanism. Before a step can use a workspace, it must have the "execute" privilege on the workspace, where "it" means the ElectricFlow user ID under which the step executes (the project principal). You can set permissions on a workspace to enable or disable access as you choose.

Impersonation and workspaces

Before a job step can use a particular workspace, three access limitations must be satisfied:

- The job step must have execute access to the workspace object. If not, ElectricFlow aborts the step with an access control violation. See the Access Control Help topic for details.
- The workspace must be accessible on the resource where the step runs. If not, the ElectricFlow agent is unable to create the step's log file and the step's execution will fail. See Workspace Accessibility (above) for more information.
- The Windows or UNIX agent account and/or job step must have read/write access to workspace files. If not, the step's execution will fail in one of several ways. This issue is the topic of this section.

Potentially, two system accounts can impact each job step execution:

- The first account is the one used by the ElectricFlow agent—you selected this account when ElectricFlow was installed. The agent account must have the ability to write the workspace root directory (to create the job workspace directory), and it must have the ability to write the job workspace directory to create log files.
- The second account is the one under which the step executes. If the step does not use user impersonation (in other words, no credential has been specified for it), the step runs under the same account as the agent so there are no additional issues. However, if the step is running with credentials, the credentialed account must also have read/write access to the workspace area.

Because of these requirements, you may end up with a configuration where any job step running in a workspace can read or write any file under that workspace root, including files from other jobs that share the same workspace root. In many environments this solution works, but in some situations you may want to prevent one job from accessing files from a different job.

To create job privacy, run all of the job's steps with credentials for a separate account. In addition, run a job step at the beginning of the job that creates a subdirectory within the job workspace and change the protection on that directory to permit access to the credentialed account only. Next, place all private files for the build in the protected subdirectory.

These files will not be accessible to other jobs or the agent. The top-level directory in the job workspace still needs to be accessible to the agent so it can read and write log files, but everything in the lower-level directory will be private.

ElectricFlow managed files

ElectricFlow automatically places certain files in the top-level job step workspace directory:

- Step logs—When a step's command executes, its standard output is redirected to a log file unique to that step. The log file name is derived from the step name and its unique identifier within ElectricFlow. For example, a step named "step1" will have a log file with a name like "step1.2770.log", where 2770 is a unique identifier assigned by ElectricFlow. A unique identifier is needed in situations where the same step name occurs multiple times in a job such as multiple invocations of a single nested procedure.
- Postprocessor logs—If a step specifies a postprocessor, standard postprocessor output is redirected to a file in the workspace. The postprocessor output file will have a name similar to the step log. For example, "step1.2770-postp.log".

- Diagnostic extracts—If a postprocessor extracts diagnostic information from a step's log file, it usually places that information in an XML file in the top-level job workspace directory, and then it sets a property that contains the file name. The ElectricFlow *postp* postprocessor uses file names like diag-2770.xml, where 2770 is the unique identifier for the step. Other postprocessors may use different file names.

Disk space management

The current ElectricFlow version does not provide a facility for automatically deleting old workspaces. You are responsible for deleting obsolete job workspaces yourself. If you delete a job, ElectricFlow does not automatically delete its workspace. (A future ElectricFlow version may provide automated facilities for identifying and deleting obsolete workspaces—until then, you may wish to create a procedure that runs on a regular schedule and deletes old job workspaces.)

ecremotefilecopy

When ElectricFlow agents (on platforms other than Linux or Windows) run steps that create log files in a workspace the ElectricFlow web server cannot access (through Linux or Windows agents), use the `ecremotefilecopy` tool to recreate job logs so they are visible on those ElectricFlow agents, which then enables the web server to retrieve and render those log files.

`ecremotefilecopy`is installed with ElectricFlow. For details, see ecremotefilecopy.

Tutorials

The ElectricFlow Help topics in this directory help you learn more about ElectricFlow and specific ElectricFlow tasks. You can directly access these Help topics from the Help system Table of Contents.

You can also access these Help topics while using the automation platform. Click **Help** at the top-right corner on any web page in the automation platform.

Adding a Link to a Job

The Workflow Details page and several other ElectricFlow web pages also allow adding links. This tutorial shows how to add links to a job and have them immediately available on the Job Details page.

To view an example procedure for this tutorial , go to the automation platform UI > **Projects** > **EC-Tutorials-**<*version*>, and click the procedure name.

An explanation of what you see:

- The Procedure Details page for the "Adding a link to a job" procedure contains two steps: Create File and Create Job Link
- The Action column displays a code fragment of the script used to create the `readme.txt` file and a reference for the Create Job Link step. For this example, we want to link to a "readme" file. (You can click on the **Create File** step name to go to the Edit Step page to see the full ec-perl script shown partially in the Action column.)

Click **Run** to run this sample procedure and see the resulting job status on the Job Details page, where the new link to the `readme.txt` file is displayed under the Links heading in the summary section at the top of the page.

Implementation

Use the following instructions if you would like to practice creating a link on the Job Details page now.

> **Important:** Any changes you make within this tutorial will not be saved when you upgrade ElectricFlow. These instructions only coach you through the process you would use to create links on your Job Details pages.

1. In the automation platform UI, click **Projects** > **EC-Tutorials-**<*version*> > **Adding a link to a job**.

   The Procedure Details page opens.

2. In the Procedure Steps section, click **Subprocedure** to create a new step based on a subprocedure.

   The Choose Subprocedure Step dialog box opens.

3. To select a project:

   1. Select **Project**. In the text box, start typing **EC-Utilities**.

      As you type, possible matches appear in a drop-down list.

   2. When you see **EC-Utilities**, select it.

4. To select a procedure:

   1. Select **Project**. In the text box, start typing **Create Job Link**.

      As you type, possible matches appear in a drop-down list.

   2. When you see **Create Job Link**, select it.

5. Click **OK**.

   The New Step page opens.

6. In the General section, `Google search` in the **Name** field.

7. In the Parameters section:

   1. In the **Link Label** field, enter `Google Search`.
      (This is any name you choose for the name of your link.)
   2. In the **Link Location** field, enter `http://www.google.com`
      (This is the URL to your link location, or it can be a relative path to where you want to create a link.)
   3. Click **OK** to save your entries and return to the Procedure Details page.

8. On the Procedure Details page, click **Run**.
9. On the Job Details page, as soon as the links are processed you will see the new "Google Search" link in addition to the "readme.txt" link.

Clicking on the new Google Search link will take to you to the Google website.

Related Information

Job Details—See this Help topic for more detailed information about creating links and complete information on other aspects for using the Job Details page.

Calling a Subprocedure

This tutorial shows how to use the step configuration process to call an existing procedure to use as a subprocedure.

To view an example procedure for this tutorial , go to the automation platform UI > **Projects** > **EC-Tutorials-**<*version*>, and click the procedure name.

An explanation of what you see:

- The Procedure Details page for the Calling a subprocedure procedure contains a step named Execute a procedure from this project.
  (Clicking on this step takes you to the Edit Step page.)
- The Action column displays a plugin (or project) name, EC-Tutorials, and a procedure name, Working with properties in a stored procedure.
  (Clicking on the plugin/project name takes you to the Project Details page for that project, and clicking on the procedure name takes you to the Procedure Details page for that procedure.)

Click **Run** to run this sample procedure and see the resulting job status on the Job Details page.

Implementation

Use the following instructions if you would like to practice creating another subprocedure for this "Calling a subprocedure" procedure.

> **Important:** Any changes you make within this tutorial will not be saved when you upgrade ElectricFlow and cannot be transferred to your procedures. These instructions only provide a clearer explanation for how you can call subprocedures in your projects.

To create a step that executes a subprocedure:

1. On the Procedure Details page, click the **Create Step** link to see the Choose Step Type dialog.
   - Select the Subprocedure tab.
   - Select Plugin and place your cursor in the text field and start typing EC-Tutorials.
   - For Procedure, click inside the text box and select Reserving a Resource for job step duration procedure or whichever procedure you would like to choose.
   - Click **Create** to go to the New Step page to create the step to call an existing procedure to use as a subprocedure.
2. On the New Step page, enter any new step name of your choice.
   Notice that no parameters are required for this step.
3. Click **OK** to return to the Procedure Details page to see your new step in the table, then click **Run**.
4. When the Job Details page is displayed, you can see your new step and the procedures it is calling.

Related Information

Procedure—create new or edit existing procedure—Help topic
Step—create new or edit existing step—Help topic

Checking the Outcome of Preceding Steps

ElectricFlow steps can have a *success*, *warning*, *error*, or *skipped* outcome. Sometimes it is useful to execute or skip subsequent steps based on the outcome of a previous step. Used with error handling behavior available at the step level, sophisticated flow control can be achieved easily.

This tutorial demonstrates how to see the outcome of a preceding step and use run conditions to execute steps based on that outcome.

To view an example procedure for this tutorial , go to the automation platform UI > **Projects** > **EC-Tutorials-**<*version*>, and click the procedure name.

An explanation of what you see:

The Procedure Details page for the procedure, "Checking outcome of preceding step", contains 3 steps:

- `step 1—force an error`—This step forces an error by attempting to run on a resource that does not exist.
  This action was achieved by hard-coding the step named "`step 1 - force an error`" to run on a resource called "`this-does-not-exist`".
- `step 2—execute if step 1 failed`—This step uses JavaScript to check the outcome of the first step and executes if the first step failed.
  You can view the JavaScript for the run condition by clicking on the step name to take you to the Edit Step page—see the Run Condition field in the Advanced section.
  This statement checks the result of "step 1" for an error. If there is an error, "step 2" will execute.
- `step 3—execute if step 1 succeeded`—This step also uses embedded JavaScript to check the outcome of the first step and executes if the first step succeeded.
  This is a slight variation of "step 2". Instead of looking at "step 1" for an error, this step checks for success and executes only if "step 1" succeeds.
  You can view the JavaScript for the run condition by clicking on the step name to take you to the Edit Step page—see the Run Condition field in the Advanced section.

Click **Run** to run this sample procedure and see the resulting job status on the Job Details page.

Implementation

Use the following instructions if you would like to practice creating steps to check the outcome of a preceding step.

> **Important:** Any changes you make within this tutorial will not be saved when you upgrade ElectricFlow and you cannot transfer any steps from this tutorial to your projects. These instruction are provided only to give you a clearer definition for how to recreate this process in your own projects.

- Click the **Create Step** link.
  - In the Choose Step Type dialog, choose **Command** step to go to the New Step page.

- Enter a Name for your step.
- In the Command(s) text box, you can use the information in the above section. "Copy and paste" the provided JavaScript, changing the step name (between the brackets) to the new step name you supplied.

- Repeat this process to create 3 steps, clicking **OK** to save each step.
- When your 3 new steps are displayed on the Procedure Details page, click **Run** to see your steps running on the Job Details page.

Related information

Step —create new or edit existing step—Help topic

Conditional Execution

Process automation frequently contains actions to be executed only if certain conditions are met. Conditional execution can be applied directly to your procedures rather than having to maintain larger scripts to handle these actions. For example, you may need to control whether or not to build on different operating system types.

This tutorial shows you how to implement simple conditional execution in a step.

To view an example procedure for this tutorial , go to the automation platform UI > **Projects** > **EC-Tutorials-**<*version*>, and click the procedure name.

An explanation of what you see:

This procedure defines one parameter and then uses a run condition to check the parameter value to decide whether or not to execute a step.

- The Procedure Details page for the Conditional execution procedure contains a step named step 1.
  (Clicking on this step takes you to the Edit Step page.)
- The Action column displays the command to be executed.
- The Parameters section displays a parameter with values.
  The parameter named "`Execute step 1`" was created as a checkbox.

  If checked, the parameter equates to "true".
  In the run condition for "step 1", the parameter is read using the following property reference notation:

  ```
  $[Execute step 1].
  ```

  (Clicking on the Parameter Name takes you to the Edit Parameter page to make modifications you may need.)

Click **Run** (at the top of the page) to run this sample procedure, which then takes you to the Run Procedure page where you could cause `step 1` to execute by selecting the Execute step 1 checkbox.

- Click **Run** again to go to the Job Details page to see the job created by running this procedure.

Implementation

Review the following instructions if you would like to see how to reproduce a conditional execution step.

Important: Any changes you make within this tutorial are not saved when you upgrade ElectricFlow. These instructions are provided only for more detailed information so you can easily see how to create your own "conditional execution" solutions in your procedures.

- Click the **Create Parameter** link to go to the New Parameter page.
  - Enter a new parameter name.
  - Click the Type down-arrow and select Checkbox.
  - Select Initially checked?
  - For Default Value, type "`true`".
  - Select Required?
  - Click **OK** to save your parameter and return to the Procedure Details page.
- Click the step name to go to the Edit Step page.
  - Change the Run Condition text to check the new parameter name you supplied above.
  - Click **OK** to return to the Procedure Details page.
  - Click **Run** to go to the Run Procedure page to see 2 parameters where you can now decide if both are required or not.
  - Click **Run** to go to the Job Details page.
    - If only 1 parameter was "checked" as required, you will see the "skipped" notation in the Status column.

Related information

More complex run conditions can be implemented using inline JavaScript. See the tutorial, "Checking outcome of preceding step" for a more complex run condition example.

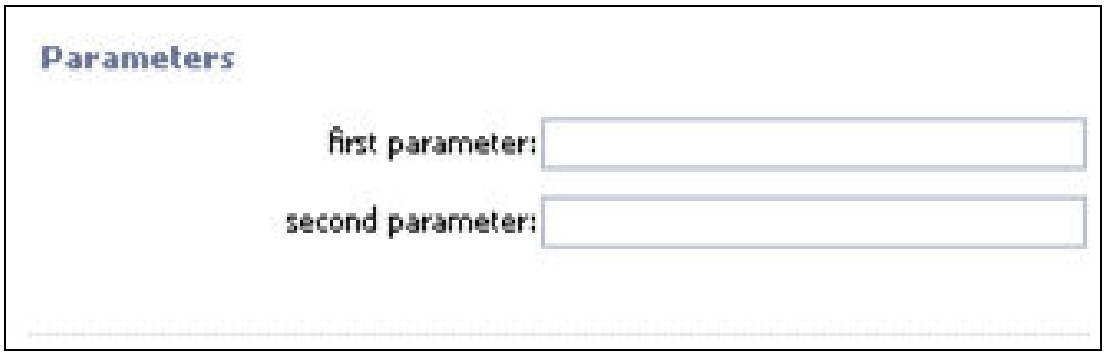
Custom parameter layouts

By default, parameters for procedures are displayed in alphabetical order based on the parameter name. Using the `ec_parameterForm` property allows parameters to be displayed in a specified order, different from the default, alphabetized order by name. This tutorial shows how to create a custom parameter layout for procedures.

To view an example procedure for this tutorial , go to the automation platform UI > **Projects** > **EC-Tutorials-**<*version*>, and click the procedure name.

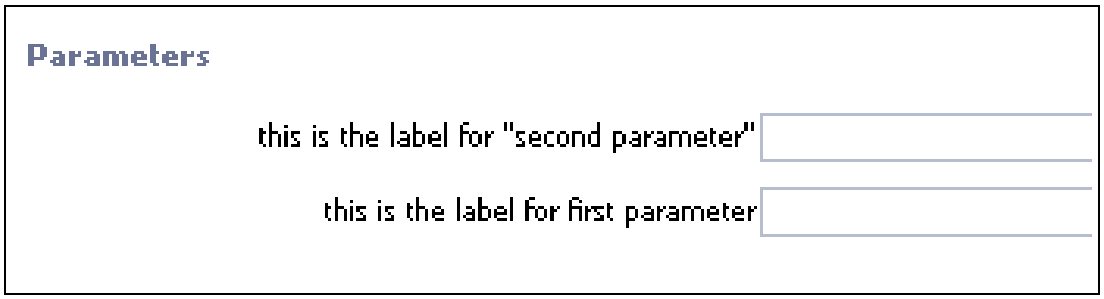
An explanation of what you see:

- This procedure implements two parameters.
- The Action column contains the command to execute. In this case, the action shows the `ec_parameterForm` property which was attached to the procedure.
- Notice the check marks in the Req? column, which indicates both parameters are "required" each time this procedure runs.
- In the Custom Procedure Properties table, notice the "`formElement`" values supplied to determine the new parameter order.
  - If the procedure is run ***without*** the `ec_parameterForm` property, the parameter input form displays parameters in default, alphabetical order:



Click **Run**.

On the Run Procedure page, notice you are presented with a Parameter section that shows the parameters in reverse order because values supplied for the `ec_parameterForm` property were called to create this new order.



The following XML creates the new order for these parameters:

```
<editor>
  <formElement>
    <property>second parameter</property>
    <label>this is the label for "second parameter"</label>
    <type>entry</type>
    <required>1</required>
  </formElement>
  <formElement>
    <property>first parameter</property>
    <label>this is the label for first parameter</label>
    <type>entry</type>
    <required>1</required>
  </formElement>
</editor>
```

Click **Run** to run this sample procedure and see the resulting job status on the Job Details page.

Implementation

> **Important:** This tutorial is intended for viewing, but not modifications within its procedure, and any changes you might make will not be saved when you upgrade ElectricFlow. However, in your own projects, you can use the sample XML above to change the parameter order in any of your procedures.

If you copy and paste the code samples we provided, using the `ec_parameterForm` property in one or more of your procedures, make sure you replace our example values with your parameter names, labels, and so on. If you have more than two parameters to reorder, create any number of additional `<formElement>` sections you need.

Related information

Customizing the ElectricFlow UI /Customizing Parameters—Help topic
Step—create new or edit existing step—Help topic
Parameter —create new or edit existing parameter—Help topic

Email Notifications

Use email notifiers to send information to people who need or want ElectricFlow notifications, whether or not they actually use ElectricFlow. For example, you might want to set up an email notifier to send log file error excerpts matched by *postp* to a team or individual responsible for investigating the error.

This tutorial illustrates how email notifiers can be attached to steps to automatically send an email when jobs, steps, or workflows execute.

> **Note:** If you do not have an Email Configuration, you must create one before you can set up an email notification. An Email Configuration tells the ElectricFlow server which host to use for mail delivery. Remember the email configuration name you choose because you will need it later.

To view an example procedure for this tutorial , go to the automation platform UI > **Projects** > **EC-Tutorials-**<*version*>, and click the procedure name.

An explanation of what you see:

- A step named "do something" that puts a short message in the log file
- A parameter that defines the email address for where you want to send the notification. This parameter is used as the destination value for the notifier named "start notifier". This notifier sends an email when the procedure runs.
- An email notifier named "start notifier" with its Type specified as "onStart".
  - Click on the notifier name to view the notifier details.
  - Make sure the value in the Email Configuration field matches the email configuration name from the "Prerequisite" section above.
  - Click **OK** to save any changes and return to the Procedure Details page.

Click **Run**.

On the Run Procedure page, in the Parameters section, enter an email address, an address list, or email group name.

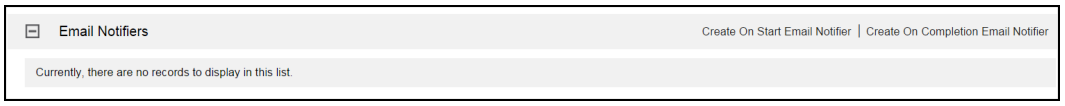
Click **Run** again to run this sample procedure and see the resulting job status on the Job Details page.

Implementation

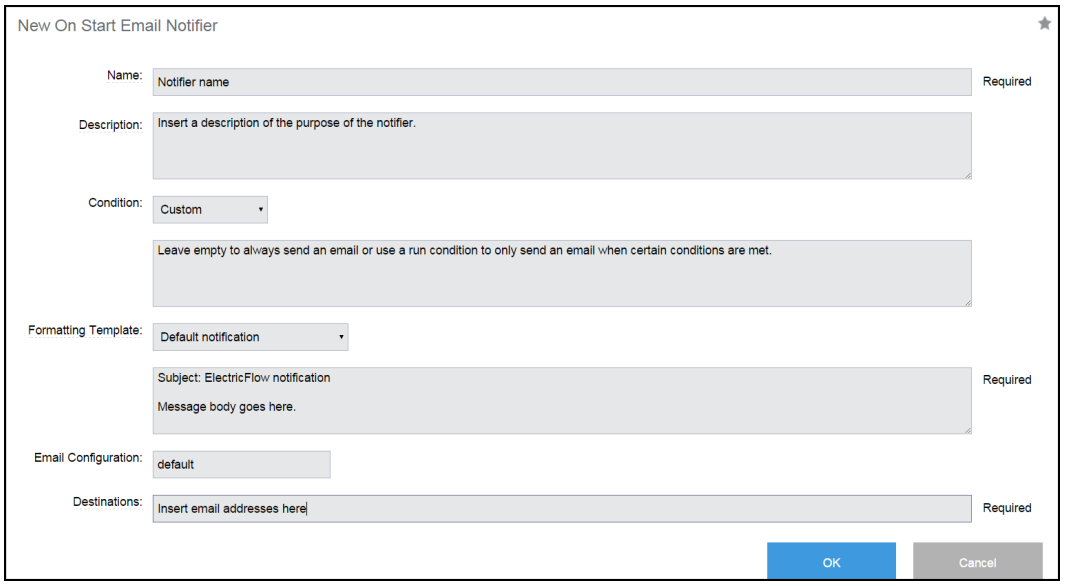
> **Note:** Use the following information to create an email notifier in your own project, assuming you have already setup an email configuration. Any changes you might make within this tutorial will not be saved when you upgrade ElectricFlow.

Using the ElectricFlow automation platform UI to add an email notifier:

1. On the Procedure Details page, go to the Email Notifiers section, and click **Create On Start Email Notifier**.



2. Enter information in the fields on the New On Start Email Notifier page. See the example below.



> **Note:** If the Email Configuration field is left blank, the system uses the email configuration with the name "default".

Click **OK** to save the notifier. When the procedure is run, an email notification is sent to the email addresses specified in the Destinations field. In this tutorial example, the Destinations field refers to the parameter value passed in to the procedure when it is run.

The value is referenced using the following notation:

```
$[/myJob/Send notification email to]
```

Related information

Email Configuration—create new or edit existing email configuration—Help topic
Email Notifier—create new or edit existing email notifier—Help topic

Executing Tasks on All Resources in a Pool

Creating resource pools (grouping resources) means you can reference many resources using one name, the pool name. When configuring a step, you can "broadcast" the step to all resources in a pool. For example, you might want to start an application or microservice deployment at the same time on all resources in a pool. This tutorial shows how to group resources into a resource "pool".

To view an example procedure for this tutorial , go to the automation platform UI > **Projects** > **EC-Tutorials-**<*version*>, and click the procedure name.

An explanation of what you see:

This procedure implements three steps:

1. The first step, `Setup environment`, creates a pool for resources.

   Clicking on the step name takes you to the Edit Step page to see the full command text supplied to create this step—a partial section is displayed in the Action column.

   > **Note:** Functionality to create this step is beyond the scope of this tutorial. For more information, see the link to "Using ectool and the ElectricFlow API" below.

   The following resources were created and added to a pool named EC-Tutorial-pool:
   EC-Tutorials-resource1
   EC-Tutorials-resource2
   EC-Tutorials-resource3

2. The second step, `Execute command on all resources in pool`, executes an echo statement on the resource EC-Tutorial-pool and has the "broadcast" option selected, which results in the step executing on all resources in the pool simultaneously.
3. The third step, `Cleanup`, deletes the resources created in the `Setup environment` step.

Click **Run** to run this sample procedure and see the resulting job status on the Job Details page.

Implementation

> **Important:** This tutorial is intended for viewing only and/or clicking on the available links to see more information. Any modifications you might make within this tutorial cannot be transferred to your projects and will not be saved when ElectricFlow is upgraded.

Before you can create the steps illustrated in this tutorial procedure for your own purpose, you must have already defined your resources and created resource pools.

See the following list of related links for more help with creating and using resource pools.

Related information

Resources—Help topic

Resource Pool—create new or edit existing pool—Help topic

Resource Pool Details—Help topic

Step —create new or edit existing step—Help topic

Factory Procedures

Factory procedures allow dynamic behavior in process automation. To implement a factory procedure, you must first create a procedure that other procedures can use as a template. A procedure uses the template to construct new procedures at runtime. Instead of creating steps or complex scripts to account for all possible scenarios, procedures can be created "on the fly" based on user input, resulting in a very dynamic system.

This tutorial shows how to create factory (dynamic) procedures, which are procedures that modify themselves at runtime based on parameters passed into them.

To view an example procedure for this tutorial , go to the automation platform UI > **Projects** > **EC-Tutorials-**<*version*>, and click the procedure name.

An explanation for what you see and what is actually happening:

This tutorial contains 2 procedures and creates a third procedure "on the fly":

- "Factory procedures"—this procedure uses the procedure template (Utility Procedure—Factory Template).
  This procedure has a parameter called "`number of steps`". This parameter takes an integer and is used to specify how many new steps should be generated.
  Note: For this tutorial example, only one procedure is constructed to use the template procedure, but you may have multiple procedures that use the factory template procedure.
- "Utility Procedure—Factory Template"—you do not see this procedure on this Procedure Details page, but it is included in the EC-Tutorials project. This is the factory procedure template used when generating new steps in the main procedure. This procedure implements a single step that writes some text to the log file.
  This procedure is designed to do a repetitive task, repeating the task a variable number of times. For example, you might need to run a test framework "*n*" number of times.
  Note: You can view this procedure in the EC-Tutorials project.

The main procedure, "Factory procedures", implements these steps:

- The first step: "create dynamic procedure" is a Perl script that uses the Perl API to create a new procedure.
  The following script creates a new procedure containing the number of steps specified by the parameter:

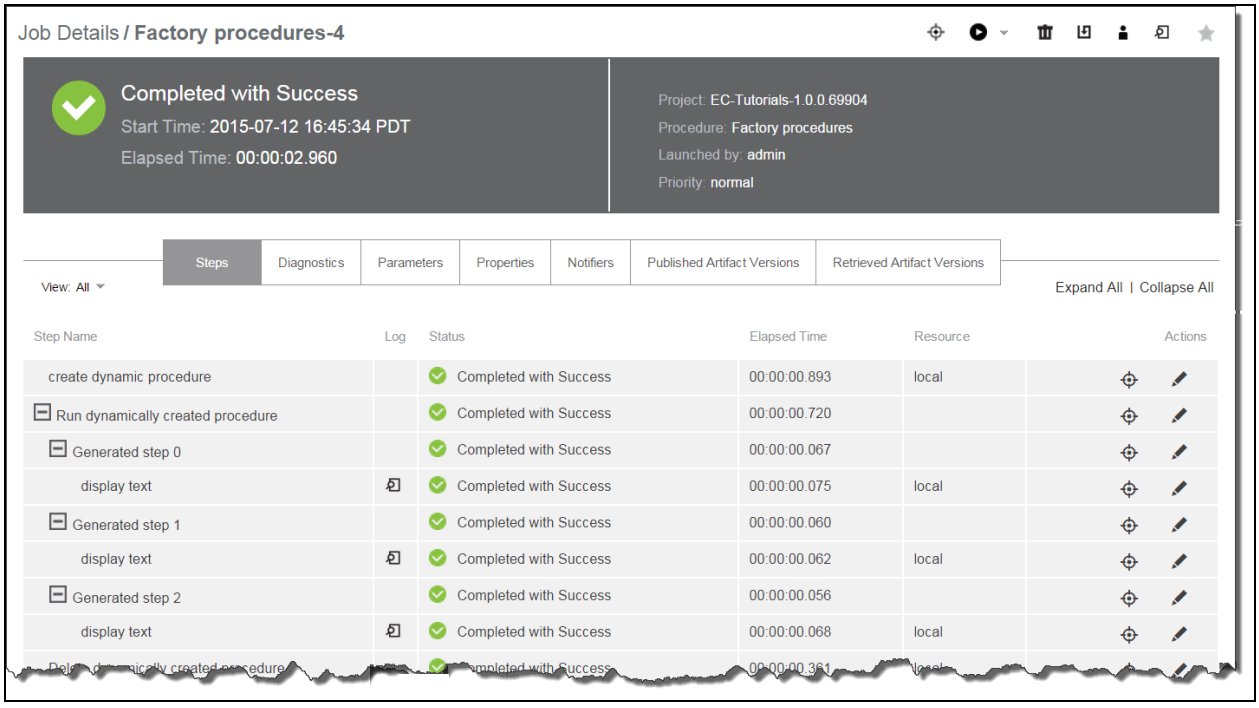
```
my $procedureName = "Dynamically Created Procedure $[jobId]";

$ec->createProcedure("$[/myProject/projectName]",$procedureName);

for(my $i=0;$i<$[number of steps];$i++)

{

$ec->createStep("$[/myProject/projectName]", $procedureName, "Generated step
$i", {subprocedure => "Utility Procedure—Factory Template",

 actualParameter => {actualParameterName => 'text', value => "Executing
generated step $i"}});

}

$ec->setProperty("/myJob/dynamicProcedure",$procedureName);
```

1. First, the procedure generates a unique name for the new procedure to be created. The name is created by appending the value of the `$[jobId]` property string.
2. A new procedure is created in the current project.
3. Based on the number of steps to be created that were passed in as a parameter, a number of subprocedure steps are created in the procedure. The subprocedure steps call the utility procedure associated with this tutorial.
4. The name of the newly created procedure is stored in `/myJob/dynamicProcedure`

- The second step: "Run dynamically created procedure" calls the procedure that was created in Step 1. Step 2 appears to call nothing, but in fact it is calling a procedure referred by `$[/myJob/dynamicProcedure]`.

Click **Run** to run this sample procedure and go to the Job Details page.

Running the procedure and specifying the number of steps to be generated as 3, results in the following job being executed:



Notice the steps that execute in parallel in the dynamically created procedure: Generated step 0, Generated step 1, and Generated step 2.

Implementation

To create a "factory procedure" process in your project:

1. Create a utility procedure—this procedure performs a task.
   - Create the steps you need for your task.
2. Create a second procedure—the "factory procedure," providing a name of your choice for this procedure.

- Create a minimum of 3 steps in this procedure.
  - step 1—create a new procedure that contains steps, each calling the utility template procedure.
  - step 2—this step executes the procedure created in step 1.
  - step 3—this step deletes the procedure created in step 1.

Each time you run the "factory procedure," it creates a temporary procedure to run the utility procedure.

Important: Any changes you make within this tutorial will not be saved when you upgrade ElectricFlow.

Postp extension

postp is a powerful ElectricFlow feature (postprocessor) you can use to monitor step output in real-time and take action. Frequently, the *action* is to extract information from step output, store it in the ElectricFlow database, and set properties for reporting. In addition, postp can be used to execute functionality depending on whether or not a pattern is found. Postp, an essential part of the ElectricFlow toolbox, provides fast feedback to users about job step status.

Postp matchers are written in Perl. Postp extensions are either loaded from a file located on the ElectricFlow server or from a stored property.

This tutorial demonstrates how to construct a simple custom *postp* matcher to extend postp capabilities, and loads the postp extension from a property.

To view an example procedure for this tutorial , go to the automation platform UI > **Projects** > **EC-Tutorials-**<*version*>, and click the procedure name.

An explanation of what you see:

- This procedure implements a single step, "get directory listing", that retrieves a directory listing using a simple Perl script. Perl is used for the example to ensure the same output no matter which operating system is used for this tutorial. Understanding the Perl code used to generate the tutorial listing is not required for this tutorial.
- A property was created. This procedure has a postp extension attached in a property named `matcher` and contains the following Perl code:

```
use ElectricCommander;
push (@::gMatchers,
{
id => "fileCountMatcher",
pattern => q{(\d+ File\(s\))},
action => q{
setProperty("summary", "Matcher $matcher->{id} found the following
output\n\n$1");
}
});
```

The postp matcher has an identifier ("id"), a regular expression ("pattern") that matches text and an "action" to be performed when text matching the regular expression is found. In this case, the regular expression "id" is `fileCountMatcher`, and an example of the output being

matched is "`2 File(s)`". When the pattern is matched, the action to perform is to set the summary property to the text matched by the regular expression.

The custom postp matcher is loaded using the following command in the Postprocessor field when creating the step:

```
postp -loadProperty /myProcedure/matcher
```

Click on the step name to go to the Edit Step page to see how this step was created—note the text in the Command box and the Postprocessor field. Click **Run** to run this sample procedure and see the resulting job on the Job Details page.

See the Status column. The number of files in the file system root where the workspace is located is displayed.

Implementation

To try using postp functionality in your own project:

- Edit an existing procedure or create a new one with a "matcher" as a property.
  For your practice example, you can "copy and paste" the matcher example provided in this tutorial (above).
- Create a new Command step for this procedure—filling in the Postprocessor field on the New Steps page.
  Again, for practice, you can use the postp string provide above in this tutorial.

When you run your new procedure, you will see results similar to those you saw after running the tutorial example procedure.

> **Important:** Any changes you make within this tutorial will not be saved when you upgrade ElectricFlow.

Related information

Postprocessors—Help topic

Step—create new or edit existing step—Help topic

Publishing and Retrieving an Artifact

To ensure traceability in processes, output needs to be versioned and tracked as it is handed from one process stage to another. With ElectricFlow Artifact Management, implementing your processes is auditable and secure.

This tutorial shows how to produce and consume artifacts as part of an ElectricFlow process.

To view an example procedure for this tutorial , go to the automation platform UI > **Projects** > **EC-Tutorials-**<*version*>, and click the procedure name.

An explanation of what you see—the following concepts are illustrated in this procedure:

- The first five steps are simple ec-perl scripts that create files and directories as use as an artifact.

  The first step, "Create myartifact.txt file", creates a text file in a workspace that can be added as an artifact. Click this step to go to the Edit Step page to see the full "dummy" script used to set up the artifact in the Command(s) field. This script creates a file named `myartifact.txt`.

  > **Note:** There is no code specific to ElectricFlow in this step, so understanding the Perl code used to generate the text file is not necessary.

  The next four steps create a directory, a subdirectory and the second text file in it.

- The "Publish artifact" step calls the EC-Artifact:Publish procedure. This step is configured to place the file `myartifact.txt`, created in the Setup step, into an artifact.

  To do this, the EC-Artifact: Publish procedure requires the following information:

  - The artifact where you want to publish. This is in the format `Artifact Group:Artifact Key`.
    In this case, the name of the artifact (where you want to publish) is EC-Tutorials:MyArtifact.
  - The artifact version being published. The version is in the form of `major.minor.patch-qualifier-buildnumber` or `major.minor.patch.buildnumber-qualifier`. For this tutorial, the version is set to `1.0.$[jobId]-tutorial`.
    Using the `$[jobId]` property expansion results in a new artifact being created every time the procedure is run.
  - The repository server from which you need to retrieve the artifact. This tutorial uses a repository named `default` that was created when ElectricFlow was installed.

  To see how this information is specified in this step, click the "Publish artifact" step name to go to the Edit Step page.

- The "Retrieve artifact" step calls the EC-Artifact: Retrieve procedure. This step takes two required parameters and several optional parameters (see the Edit Step page/Parameter section for this step). The required parameters are:
    - **Artifact** – EC-Tutorials:MyArtifact (this is the name of the artifact to retrieve).
    - **Version** – `1.0.$[jobId]-tutorial` (this is the artifact version to be retrieved—either an exact version or a version range). This tutorial uses an exact version.

  The optional parameters are:
    - **Retrieve to directory** – the location where the artifact files are downloaded when they are retrieved. This is where the retrieved artifact versions are stored.
    - **Overwrite** – the default is **update** where the artifact files are updated when they are retrieved. The other values are **true** (the files for the retrieved artifact versions are overwritten) or **false** (the files for the the retrieved artifact versions are not overwritten).

  The **Retrieved Artifact Location Property** parameter is the name or property sheet path that the step uses to create a property sheet. The default value is `/myJob/retrievedArtifactVersions/$[assignedResourceName]`.

  This property sheet has information about the retrieved artifact versions, including their location in the file system. It displays the location from where the artifact is to be retrieved and retrieves the property value in the `/myJob/retrievedArtifactVersions/$[assignedResourceName]`.

  For the **Filter(s)** parameter, enter search filters, one per line, applicable to querying the ElectricFlow database for the artifact version to retrieve.

  For more information about these parameters, see Retrieve Artifact Version Step.
- The "Output location that artifact was retrieved to" step prints the message "Artifact extracted to: <directory where the retrieved artifact will be located>".

- Click **Run** to run this sample procedure and see the resulting job on the Job Details page.

Implementation

To publish and retrieve artifacts, apply these concepts to your artifact projects, creating the steps as required to your procedure.

> **Important:** Any changes you make within this tutorial will not be saved when you upgrade ElectricFlow.

Related information

Artifact Management—Help topic

Artifact Details—Help topic

Job Details—Help topic

Publish Artifact Version step—Help topic

Retrieve Artifact Version step—Help topic

Reserving a Resource for Job Step Duration

Sometimes you may want a resource to perform work for only one job at a time. Because a ElectricFlow agent is capable of executing steps for many jobs on a resource, this can produce undesirable performance results. For example, you might want a dedicated resource to stress test an application or microservice after you have ElectricFlow initiate a load testing task.

This tutorial shows how to reserve a resource for one job step only while that job step is running.

To view an example procedure for this tutorial , go to the automation platform UI > **Projects** > **EC-Tutorials-**<*version*>, and click the procedure name.

An explanation of what you see:

- This procedure locks the resource for the step and executes the tasks.
- To demonstrate this functionality, two steps are implemented.
  - Notice that both steps are set to run in parallel on the local resource.
  - Both steps request a lock of the resource for the duration of the step execution.

The functionality implemented in each step:

- A simple Perl sleep statement to force a lock over a period of time when the procedure is run
- Select "Step" in the Retain Exclusive field in the Advanced section on the New/Edit Step page

Click a name in the Step Name column (to go to the Edit Step page for that step). You will see the Perl sleep statement added to the Command text box, and "Step" selected on the Retain Exclusive field.

Instead of running the steps in parallel as you may have previously requested, the steps will now execute serially. The second step must wait for the resource to become available because it was reserved by the first step.

Click **Run** to run this sample procedure and see the resulting job on the Job Details page.

Implementation

Create or edit Command Steps for your procedures if you need this functionality.

- Go to the Procedure Details page for the procedure where you want a step to reserve a resource.
- Add a Command Step to your procedure (use the step creation link at the top of the table), or edit an existing step.
- In the Command text box, enter a script if you want to specify any time constraints.
- On the New/edit Steps page, in the Advanced section, choose "Step" in the Retain Exclusive field.

The next time you run this procedure, your job step will have exclusive use of the resource you specified.

> **Important:** Any changes you make within this tutorial will not be saved when you upgrade ElectricFlow.

Related information

Step—create new or edit existing step—Help topic

Running Steps in Parallel

The main advantage to running steps in parallel is increased performance. You can set a step to run serially as a synchronization point and then execute more steps in parallel. For example, running steps to execute tests on different platforms concurrently would mean numerous tests could complete in the same time period. This tutorial demonstrates how to run steps in parallel.

To view an example procedure for this tutorial , go to the automation platform UI > **Projects** > **EC-Tutorials-**<*version*>, and click the procedure name.

An explanation of what you see:

This procedure contains 5 steps:

- Step 1 and 2 run in parallel.
- The third step (sync) is a sync point where the procedure waits until steps 1 and 2 finish executing.
- Steps 4 and 5 execute in parallel after the sync point.
- Steps 1, 2, 4, and 5 execute simple Perl scripts that print a message and "sleep" for a period of time.

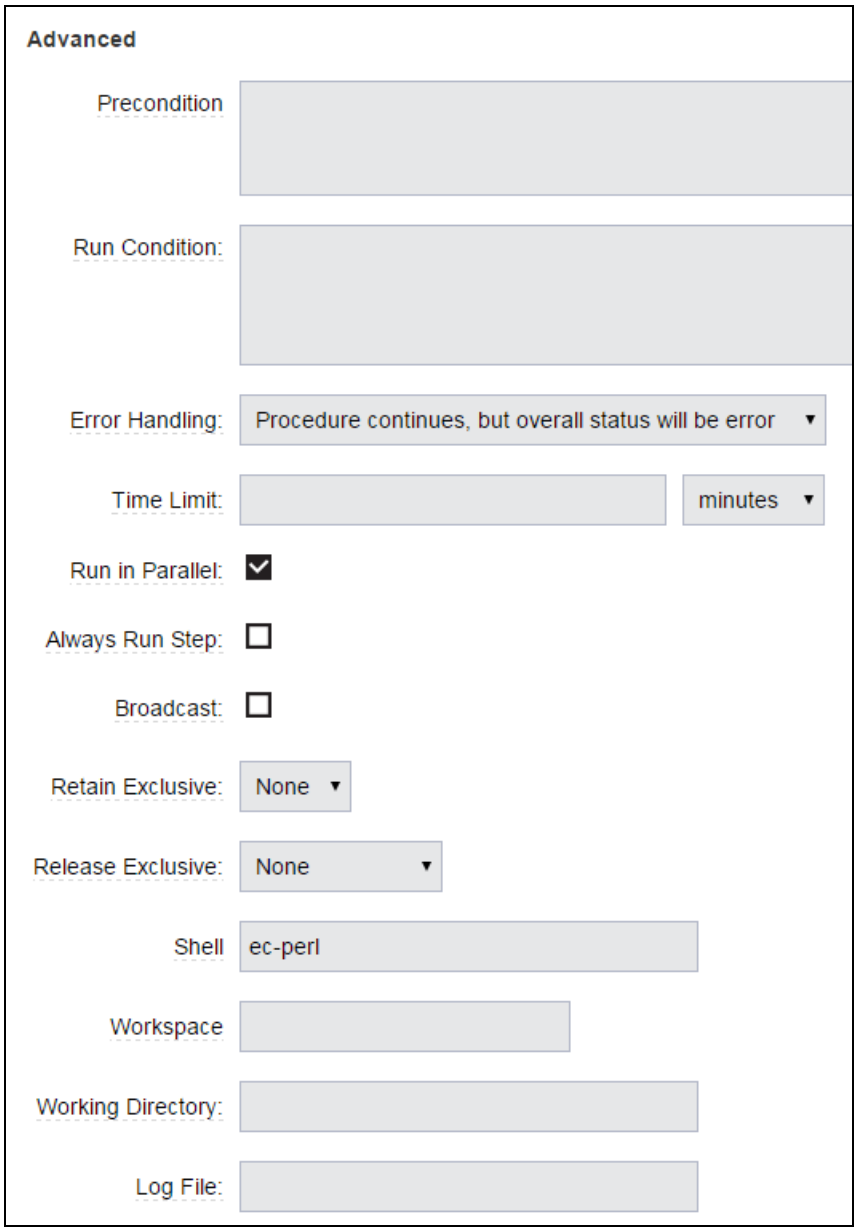
> **Note:** Using "sleep" is not required, but used here to make sure each parallel step runs long enough to be visible while running this tutorial.

Click **Run** to run this sample procedure and see the resulting job status on the Job Details page.

Implementation

> **Important:** Use the following information to create steps to run in parallel in your procedures. Any changes you make within this tutorial will not be saved when you upgrade ElectricFlow.

Steps will run in parallel if you select the Run in Parallel option in the Advanced section on the Step—create new or edit existing step page.



Related Information

Step—create new or edit existing step—Help topic

Step Timeouts and Steps that Always Run

Complex processes can be brittle so ElectricFlow provides a way to account for different errors in your procedures. You can specify timeouts, error handling methods, and steps that always run no matter what errors have occurred in preceding steps in the procedure. These facilities can make sure your build does not "hang" in a situation where you would prefer the build to continue.

This tutorial shows how to construct a procedure so processes are terminated after a set time period and how to implement a step that always runs in the event of an error (to perform cleanup).

To view an example procedure for this tutorial , go to the automation platform UI > **Projects** > **EC-Tutorials-**<*version*>, and click the procedure name.

An explanation of what you see:

This procedure implements 2 steps:
(Clicking on the step name takes you to the Edit Step page to see how the step was created.)

- The "force timeout" step has a 10-second timeout and is set to abort the procedure and terminate running steps in the event of a timeout or error.

  This step executes a sleep command using ec-perl, which makes the step execute for 15 seconds. Step error handling is set to "Abort procedure and terminate running steps". This step will always fail because the 15-second sleep time is longer than the 10-second timeout.

- The "always run" step set to run no matter what the error handling behavior is for any preceding steps.

  This step uses an echo statement to output some text. The step property, `Always Run Step`, is set to "true" so this step runs regardless of the result for the preceding step.

Click **Run** to run this sample procedure and see the resulting job on the Job Details page.

On the Job Details page: Notice that the "always run" step completed with Success while the "force timeout" step completed with a TIMEOUT error.

Implementation

> **Important:** This tutorial is intended for viewing only. Any changes you might make within this tutorial cannot be transferred to your projects will not be saved when you upgrade ElectricFlow.

As you create steps for your projects, you can duplicate these tutorial concepts. On the New Steps page, review the Advanced section. Notice the Error Handling and Always Run Steps options.

- For Error Handling, choose the option that best suits your purpose.
- For the Always Run Step option, select the checkbox to ensure this step will always run.

Storing and Retrieving Properties in a Job

This tutorial shows you how to create, set, and retrieve properties stored on a job.

To view an example procedure for this tutorial , go to the automation platform UI > **Projects** > **EC-Tutorials-**<*version*>, and click the procedure name.

An explanation of what you see:

This procedure implements two steps.

- Write data to a property attached to the running job:
  "Step 1—Create a property attached to the job" calls ectool to create a property name `property1`.
  - The path to the property being set is "`/jobs/$[/myJob]/property1`.
  - `$[/myJob]` is a shortcut to the running job name and can be used while the job is running. If the job number was 1000, the path "`/jobs/$[/myJob]/property1`" might expand to "`/jobs/job-1000/property1`".

- Read data from the property created in Step 1:
  "Step 2—Read the property attached to the job by Step 1"—this step could read step values stored on a job, running as part of the job, or could be executed using a simple property expansion.
  - In this case, `property1` was set in Step 1 and is referenced in Step 2 using the `$[property1]` notation.
  - Two other ways to reference a property (shown by example in the Action column):
    - Using `$[/myJob/property1]`
    - Using inline JavaScript with the notation `$[/javascript myJob.property1]`

Click **Run** to run this sample procedure and see the resulting job on the Job Details page.

- After the job completes, click the icon in the Log column for Step 2 to view three lines each showing the data (the property value) that was stored in the property that was retrieved.

Implementation

If you need to retrieve properties, adapt these concepts to your procedures.

- For help getting started, click on a step name in the tutorial procedure to go to the corresponding Edit Step page.
- You can reuse the text supplied in the Command(s) box, changing the "property1" name and other values to those more meaningful to your project/procedure.

> **Important:** Any changes you make within this tutorial will not be saved when you upgrade ElectricFlow.

Related information

Procedure—create new or edit existing procedure—Help topic

Step—create new or edit existing step—Help topic

Working with Properties Stored in a Procedure

This tutorial shows common ways to read and write properties on a procedure. For this tutorial, familiarity with creating procedures and steps is assumed. See Related Information on page 1613 if you would like to review these topics.

To view an example procedure for this tutorial , go to the automation platform UI > **Projects** > **EC-Tutorials-**<*version*>, and click the procedure name.

An explanation of what you see—this procedure implements the following functionality:

- This procedure implements five steps. Each step name is a description of its action.
  - Click on a step name to go to the corresponding Edit Step page to see how the step was created.
    This page also contains text in the Description text box to describe what you see in the Action column on the Procedure Details page. The full text of what appears in the Action column is provided in the Command text box.

- In the Custom Procedure Properties table, notice two properties were created: `property1` and `property2`

- There are numerous ways to work with stored properties. Each of the steps described below illustrates a different method.
  - Read data from property1 using the *property substitution* notation (syntax):
    The property value is read in this step using the property substitution reference `$[/myProcedure/property1]`.
  - Read data from property1 using inline JavaScript:
    The property name `property1` is read from the procedure level using this JavaScript notation:
    `$[/javascript myProcedure.property1]`.
  - Perform a calculation using data from property 2 and inline JavaScript:
    In this example, the value of `property2` is multiplied by 5 using this JavaScript notation:
    `$[/javascript myProcedure.property2 * 5]`.
  - Read data from a property1 using the ElectricFlow Perl API:
    Perl code is used to call the ElectricFlow API to retrieve the value of `property1`.
    The `getProperty` API returns an object so the property value must be retrieved after the `getProperty` call, using the `findnodes` method for the returned object.

    ```
    #read the property
    my $property1node = $ec->getProperty("/myProcedure/property1");
    #retrieve the property value from the object returned to $property1node
    my $property1 = $property1node->findnodes("//value")->string_value();
    ```

  - Read data from property1 using ectool:
    ectool code is used to retrieve the value of `property1`:

    ```
    ectool getProperty "/projects/$[/myProject/projectName]/procedures/Working
    with properties stored in a procedure/property1"
    ```

Click **Run** to run this sample procedure and see the results on the Job Details page. On the Job Details page, click the icon in the Log column to see the value read from the property.

Implementation

If you need to retrieve properties stored in a procedure, adapt any one of these step concepts to your procedures. You can reuse the text supplied in the Command(s) box (from any Edit Step page in this tutorial), changing names and other values to those more meaningful to your project/procedure.

> **Important:** Any changes you make within this tutorial will not be saved when you upgrade ElectricFlow.

Related Information

Procedure—create new or edit existing procedure—Help topic

Step—create new or edit existing step—Help topic

Properties—Help topic

Glossary

This glossary is a reference topic containing short descriptions for ElectricFlow objects, terms, and concepts.

Term	Description
access control	An access control list (ACL) determines if a particular user can perform a particular operation on a specified object. The list contains *access control entries* (ACEs), each of which specifies a user or group and indicates whether certain operations are allowed or denied for that user or group. Using access control provides security for ElectricFlow system use.
ACE (Access Control Entry)	Determines if a particular user can perform a particular operation on a specified object. The list contains ACEs.
ACL (Access Control List)	A list of ACEs.
actual parameter	An actual parameter is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters": formal parameters define parameters a procedure is expecting, and actual parameters provide values to use at run-time.
admin	"admin" is a special built-in user that has universal ElectricFlow access. If you log in as admin, you can perform any operation in the system, regardless of access control limitations.
agent	An agent is an ElectricFlow component that runs on each machine where job steps can execute. The agent works under the ElectricFlow server's control to execute job steps, monitor their progress, and record information about their completion. A single agent process can manage multiple job steps executing concurrently on a single machine. See the **Web Interface Help** > **Resources** topic for more information.
artifact	An artifact is a top-level object containing artifact versions, a name template for published artifact versions, artifact specific properties, and access control entries to specify privileges.
artifact key	An artifact key is an identifier for an artifact and the "key" component of the artifact name.
artifact repository	See the **Artifact Management** > **Artifact objects** > **Repository** topic for more information.
artifact version	An artifact version is a collection of 0 to N files that were published to an artifact repository.

Term	Description
backing store	The backing store is the directory on the repository server where artifact versions are stored. By default, the backing store is the <datadir>/repository-data directory in the repository installation—this default setting can be changed.
child step	A nested step that appears as a substep in the Job Details page when a step is either of the following: • A subprocedure step • A step that dynamically creates subordinate ("child") steps using the `createJobStep` API command For details about the createJobStep API command, see the ElectricFlow API Guide at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html).
compression	Compression reduces transfer time when publishing an artifact. However, compression also adds overhead when computing the compressed data. If files included in the artifact version are primarily text files or are another highly compressible file format, the benefit of reduced transfer time outweighs the cost of computing compressed data.
continuous integration	Using continuous integration means a build is launched every time code changes are checked into a Source Control Management (SCM) system. The ElectricFlow ElectricSentry component is the engine for continuous integration, while the CI Continuous Integration Dashboard is the front-end user interface for ElectricSentry.
credential	A credential is an object that stores a user name and password for later use. You can use credentials for user impersonation and saving passwords for use inside steps. Two credential types are available: *stored* or *dynamic*.
custom property	Custom properties are identical to intrinsic properties and when placed on the same object, are referenced in the same manner and behave in every way like an intrinsic object-level property with one exception: they are not created automatically when the object is created. Instead, custom properties can be added to objects already in the database before a job is started, or created dynamically by procedure steps during step execution. Custom properties in a property sheet can be one of two types: *string* property or a *property sheet* property. String properties hold simple text values. Property sheet properties hold nested properties. Nested properties are accessed by way of the property sheet property of their containing sheet.
description	(Optional) Plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. Allowable HTML tags are `<a>`, `<b>`, ` `, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, and `<ul>`.

Term	Description
diagnostic extract	A diagnostic extract is a log file portion from a job step, typically describing an error or interesting condition, extracted by a postprocessor and saved for reporting. The postprocessor usually places this information in an XML file in the top-level job workspace directory, and then sets a property that contains the file name. The ElectricFlow *postp* postprocessor uses file names like `diag-2770.xml`, where "2770" is the unique identifier for the step. Other postprocessors you may use can have a different file name configuration.
dynamic credential	Dynamic credentials are captured when a job is created. Dynamic credentials are stored on the server temporarily until the job completes and then discarded.
ec-perl	*ec-perl* is a small wrapper program installed as part of ElectricFlow. When the ec-perl wrapper runs, it sets up the environment, finds, and calls ElectricFlow's copy of Perl, passing all of its parameters to Perl.
ectool	*ectool* is the ElectricFlow command-line application that provides control over the ElectricFlow system if you prefer using a command-line interface rather than the ElectricFlow web interface. Most functions that can be invoked through the ElectricFlow web interface can be invoked using ectool.
ElectricAccelerator	ElectricAccelerator is a software build accelerator that dramatically reduces software build times by distributing the build over a large cluster of inexpensive servers. Using a patented dependency management system, ElectricAccelerator identifies and fixes problems in real time that would break traditional parallel builds. ElectricAccelerator plugs into existing Make-based infrastructures seamlessly and includes web-based management and reporting tools.
ElectricSentry	ElectricSentry is the ElectricFlow engine for continuous integration, integrating with numerous Source Control Management (SCM) systems. ElectricSentry is installed automatically with ElectricFlow and is contained in a plugin named ECSCM and in the Electric Cloud project. **Note:** The CI Continuous Integration Dashboard is the front-end user interface for ElectricSentry.
email configuration	Before you can send an email notifier, you must set up and email configuration, which establishes communication between the ElectricFlow server and your mail server.
email notifier	After setting up the ElectricFlow server and your mail server to communicate, you can send email notifications (notifiers). You can attach email notifiers to procedures, procedure steps, and state definitions.

Term	Description
Event log	See log.
Everyone	A special intrinsic access control on page 1614 group that includes all users.
filter	Two filter categories: • Intrinsic filters—these filters provide a convenient way to access certain well-defined fields for jobs. • Custom filters—these filters allow you to access a much broader range of values, including custom properties. Any values accessible through an intrinsic filter can be checked using a custom filter also (though not as conveniently).
formal parameter	A formal parameter is an object that defines a parameter expected by a procedure, including its name, a default value, and an indication of whether the parameter is required. Formal parameters are different from "actual parameters": formal parameters define the kinds of parameters a procedure is expecting, and actual parameters provide values to use at run-time.
gateway	To communicate with a resource, workspace, or artifact repository server in another zone, a "gateway" must be created. A gateway object contains two resource (agent) machines. For example, GatewayResource1 and GatewayResource2 are each configured to communicate with the other. One gateway resource resides in the *source* zone and the other in the *target* zone. A gateway is bidirectional and informs the ElectricFlow server that each gateway machine is configured to communicate with its other gateway machine (in another zone).
group	A group defines a collection of users for access control purposes. A group can be defined externally in an LDAP or Active Directory repository, or *locally* in the ElectricFlow server. See the **ElectricFlow Help** > **Web Interface Help** > **Users and Groups** > **Groups** topic for more information.
impersonation	Impersonation is a mechanism that allows a job step to execute under a particular login account (the ElectricFlow agent "impersonates" a particular user during the execution of that step). Impersonation is implemented using credentials.
inheritance	A feature of the ElectricFlow access control mechanism where access to a particular object is determined by the access control list for that object, and also by the access control lists of the object's parent and other ancestors. Each object can be configured to enable or disable inheritance from its ancestors.

Term	Description
intrinsic property	Intrinsic properties represent attributes that describe the object to which they are attached. ElectricFlow automatically provides intrinsic properties for each similar type object within ElectricFlow. For example: Every project has a `description` property that can be referenced with a non-local property path such as `/projects/Examples/description`.
job	A job is the output associated with invoking a ElectricFlow procedure. A new job is created each time you **run** (execute) a procedure.
job configuration	A job configuration is an object containing all parameter and credential information needed to run a procedure. A Job Configuration section is provided as part of the ElectricFlow Home page to make it easy for you to invoke your favorite configurations with a single mouse click. You can create job configurations in three ways: • From the Job Details page for a previously invoked job, click the **Save Configuration** link at the top of the page. Your saved job configuration will be displayed on your Home page. • Create a job configuration from "scratch" by clicking the **Create** link in the Job Configurations section (on the Home page). In the Create Configuration pop-up menu, select the project and procedure you want to use for creating this configuration. • On the page for editing a schedule, click the **Save Configuration** link at the top of the page. Your saved configuration will be displayed on your Home page.

job name template	This is the template used to determine the default name for jobs launched from the procedure. You can create a Job Name Template when you create a procedure.
	For example:
	In the Job Name Template field, you might enter:
	`$[projectName]_$[/increment /myproject/jobCounter]_ $[timestamp]`
	which produces a name like:
	`projectFoo_1234_20140102130321`
	You can enter any combination of elements to create procedure names more meaningful to you. For example, you could choose to include the build number and procedure name.

jobs quick view	A Jobs Quick View section is one of the facilities provided on the ElectricFlow Home page. This section allows you to define a category of jobs interesting to you (such as all running jobs or all jobs for a particular product version). Your Home page can display the last several jobs in each category you define.

Term	Description
job step	After a procedure is executed, the resulting job contains one job step for each step in the original procedure. The job step records information about the procedure step execution, such as the command executed, the resource where it executed, execution time, and error information.
job workspace	A directory (containing all files and subdirectories) allocated by ElectricFlow for a particular job. Each job workspace is allocated as the child of a workspace root directory. See Workspace on page 1627.
local group	A group defined *inside* ElectricFlow, as opposed to a group defined in an external repository. A local group can refer to both local and remote users, whereas a group in an external repository refers to users in that repository only. See group.
local user	A user defined *inside* ElectricFlow, as opposed to a user defined in an external repository. If a user defined in an external repository has the same name as a local user, the external user is not accessible. Local users are not visible outside ElectricFlow. We recommend using external accounts whenever available, but you may need to create local users if you do not have a shared directory service or if you need special accounts to use for ElectricFlow only. See user.
log	ElectricFlow provides a log for events generated anywhere in the system, including jobs and workflows. • To see *only* events for a single workflow, select the Workflows tab, then a workflow Name to go to the Workflow Details page and click the View Log link at the top of the page. • To see *only* events for a single job, select the Jobs tab, then the Job name to go to the Job Details page and click the View Log link at the top of the page. • To see *only* events for a specific object, select the Search tab to go to the Define Search page. For example: You can select the Object Type, "Log Entry", then click the **Add Intrinsic Filter** link. Select the down-arrow where you see "Container" auto-populated and select "Container Type. Use the "equals" operator, then select the next down-arrow to choose an object. Click **OK** to start the search. See the Event Log topic for more information. From the Administration tab, the default view for the Event Log page is the warning (WARN) level. For workflow and job event logs, the default view from their respective pages is the information (INFO) level.
matcher	A *matcher* controls the postp postprocessor. Use matchers to extend postp with additional patterns if you find useful patterns in your log files undetected by postp. A matcher contains a pattern that matches lines in a step's log and actions to carry out if/when the pattern matches.

Term	Description
Microservice	A variant of the service-oriented architecture (SOA) architectural style, which structures an application as a collection of loosely-coupled services. In a microservices architecture, services should be fine-grained, and the protocols should be lightweight. Decomposing an application into smaller services improves modularity and makes the application easier to understand, develop, and test. It also parallelizes development by letting small autonomous teams develop, deploy, and scale their respective services independently. It also lets the architecture of an individual service emerge through continuous refactoring. Microservices-based architectures enable continuous delivery and deployment.
misfire policy	A misfire policy allows you to manage how a schedule resumes in cases where the normal scheduled time is interrupted. Available options are `skip` (all misfires are ignored and the job runs at the next scheduled time) and `run once` (after one or more misfires, the job runs at the soonest time that occurs within an active region). See Schedule on page 1624 .
parameter	A property value passed into a procedure when it is invoked (at *run* time), and used by the procedure to change its behavior. Two types of parameters: actual and formal.
plugin	A plugin is a collection of one or more features, or a third-party integration or tool that can be added to ElectricFlow. Plugins are delivered as a JAR file containing the functional implementation. When a plugin is installed, the ElectricFlow server extracts the JAR contents to disk into a configurable plugins directory. A plugin has an associated project that can contain procedures and properties required by the implementation. A plugin can provide one or more new pages for the web interface and may also provide a configuration page so you can provide additional information that may be necessary to implement the plugin.
polling frequency	The *polling frequency* is how often the ElectricSentry continuous integration engine is set to look for new code check-ins. The default is set to every 5 minutes, but this number can be adjusted.
pool	Also known as "resource pool". A pool is a collection of resources. If a step specifies a pool name as its resource, ElectricFlow can choose any available resource within that pool.

Term	Description
postp	*postp* is a postprocessor included with ElectricFlow. postp uses regular expression patterns to detect interesting lines in a step log. *postp* is already configured with patterns to handle many common cases such as error messages and warnings from *gcc*, *gmake*, *cl*, *junit*, and *cppunit*, or any error message containing the string "error." *postp* also supports several useful command-line options, and it can be extended using "matchers" to handle environment-specific errors. See matcher.
postprocessor	A postprocessor is a command associated with a particular procedure step. After a step executes, the postprocessor runs to analyze its results. Typically, a postprocessor scans the step log file to check for errors and warnings. Also, it records useful metrics such as the number of errors in properties on the job step, and extracts step log portions that provide useful information for reporting. ElectricFlow includes a standard postprocessor called *postp* for your use and you can "extend" postp. See matcher.
preflight build	A preflight build provides a way to build and test a developer's changes before those changes are committed. A "post-commit" source tree is simulated by creating a clean source snapshot and overlaying the developer's changes on top of it. These sources are then passed through the production build procedure to validate the changes work successfully. Developers are allowed to commit their changes only if the preflight build is successful. Because developer changes are built and tested in isolation, many common reasons for broken production builds are eliminated.
privileges	ElectricFlow supports four privilege types (for access control and security) for each object: • Read—Allows object contents to be viewed. • Modify—Allows object contents (but not its permissions) to be changed. • Execute—If an object is a procedure or it contains procedures (for example, a *project*), this privilege allows object procedures to be invoked as part of a job. For resource objects, this privilege determines who can use this resource in job steps. • Change Permissions—Allows object permissions to be modified.
procedure	A procedure defines a process to automate one or more steps. A procedure is the ElectricFlow unit you execute (*run*) to carry out a process. A step in one procedure can call another procedure (in the same or different project), and this procedure then becomes known as a "subprocedure" (also known as a "nested" procedure). The step can pass arguments to the subprocedure.

Term	Description
project	A project is a top-level container for related procedures, workflows, schedules, jobs, and properties, which is used to isolate different user groups or functions, and also encapsulate shared facilities. Projects have two purposes: • Projects let you create separate work areas for different purposes or groups of people so they do not interfere with each other. For example, different projects can reuse the same names internally without conflict, and each project has its own access control that determines who can use and modify the project. In a small organization, you might choose to keep all work in a single project, but in a large organization, you might want to use projects to organize information and simplify management. • Projects simplify sharing. You can create library projects containing shared procedures and invoke these procedures from other projects. After creating a library project, you can copy it easily to other ElectricFlow servers to create uniform processes across your organization.
project principal	*Project principal* is a special user ID associated with each project. If a project name is "xyz," the project principal for that project is "`project: xyz`" (with an embedded space). This principal is used when procedures within the project are run, so you can create access control entries for this principal to control runtime behavior.

Term	Description
property	A property is a `name-value` pair associated with ElectricFlow objects to provide additional information beyond what is already built into the system. Built-in data is accessible through the property mechanism also. Two types of properties: *intrinsic* and *custom*. ElectricFlow provides Intrinsic properties and allows you to create Custom properties. Intrinsic properties are case-sensitive. Custom properties, like all other object names in the ElectricFlow system, are case-preserving, but not case-sensitive. • **Intrinsic properties** These properties represent attributes that describe the object to which they are attached, and are automatically added by ElectricFlow for each similar type object. For example, every project has a `Description` property that can be referenced with a non-local property path such as `/projects/Examples/description`. • **Custom properties** Custom properties are identical to intrinsic properties and when placed on the same object, are referenced in the same manner, and behave in every way like an intrinsic object-level property with one exception: they are not created automatically when the object is created. Instead, custom properties can be added to objects already in the database before a job is started, or created dynamically by procedure steps during step execution.
property sheet	A property sheet is a collection of properties that can be nested to any depth. The property value can be a string or a nested property sheet. Most objects have an associated "property sheet" that contains custom properties created by user scripts.
proxy agent	A proxy agent is an agent on a supported Linux or Windows platform, used to proxy commands to an otherwise unsupported agent platform. Proxy agents have limitations, such as the inability to work with plugins or communicate with ectool commands.
publisher	A publisher is the job that completes the operation for an artifact version.
quiet time	An inactivity period before starting a build within a continuous integration system. This time period allows developers to make multiple, coordinated check-ins to ensure a build does not start with some of the changes only—assuming all changes are checked-in within the specified inactivity time period. This time period also gives developers an opportunity to "back-out" a change if they realize it is not correct. Using ElectricSentry, the inactivity time period can be configured globally for all projects or individually for a single project.

Term	Description
resource	A resource specifies an agent machine where job steps can be executed. Resources can be grouped into a "pool", also know as a "resource pool." ElectricFlow supports two types of resources: • **Standard—**specifies a machine running the ElectricFlow agent on one of the supported agent platforms • **Proxy—**requires SSH keys for authentication. You can create proxy resources (agents and targets) for ElectricFlow to use on numerous other remote platforms/hosts that exist in your environment.
Schedule	An object used to execute procedures automatically in response to system events. For example, a schedule can specify executing a procedure at specific times on specific days. Three types of schedules are available: Standard, Continuous Integration, and Custom (custom schedules are typically continuous integration schedules that do not use the ECSCM plugin).
Sentry schedule	A continuous integration schedule created using the ElectricSentry engine for continuous integration or the CI Continuous Integration Dashboard, which is an easy- to-use front-end user interface for the ElectricSentry engine.
Service	See Microservice on page 1620.
shortcut	One type of shortcut is part of the ElectricFlow Home page facility and records the location of a page you visit frequently (either inside or outside of ElectricFlow), so you can return to that page with a single click from the Home page. Another type of shortcut is a context-relative shortcut to property paths. This shortcut can be used to reference a property without knowing the exact name of the object that contains the property. You might think of a shortcut as another part of the property hierarchy. These shortcuts resolve to the correct property path even though its path elements may have changed because a project or procedure was renamed. Shortcuts are particularly useful if you do not know your exact location in the property hierarchical tree.
state	Workflows always have a single active *state*. Each state in a workflow, when it becomes active, can perform an action. A state can run a procedure to create a subjob or run a workflow definition to create a subworkflow—in the same way that procedures can call other procedures. One or more states can be designated as "starting" states to provide multiple entry points into the workflow. See state definition.

Term	Description
state definition	Workflow objects are split into two types: *Definition* objects and *Instance* objects. Definition objects provide the template for a running workflow instance. You create a new workflow by defining a Workflow Definition along with its State Definition and Transition Definition objects. When you run the workflow definition, the system creates a new Workflow object with an equivalent set of State and Transition objects that represent the run-time instances of the workflow definition. **Note:** We omit the "Instance" qualifier for brevity in the API and the UI. Each workflow can contain one or more state objects. Defining states for a workflow is analogous to defining steps for a procedure.
stored credential	*Stored credentials* are given a name and stored in encrypted form in the database. Each project contains a list of stored credentials it owns. These credentials are managed from the Project Details page.
subprocedure	Creating subprocedures is a way of "nesting" procedures. A step (from any procedure) can call a procedure from another project or the same project. The procedure called by the step then becomes a subprocedure.
system object	This is a special object whose access control lists are used to control access to some ElectricFlow internals. System objects are: `admin`, `artifactVersions`, `directory`, `emailConfigs`, `forceAbort`, `licensing`, , `plugins`, `priority`, `projects`, `repositories`, `resources`, `server`, `session`, and `workspaces`.
tag	A way to categorize a project to identify its relationship to one or more other projects or groups. You can edit a project to add a tag. Enter a tag if you want to categorize or "mark" a project to identify its relationship to one or more other projects or groups. For example, you might want to tag a group of projects as "production" or "workflow", or you might want to use your name so you can quickly sort the project list to see only those projects that are useful to you.

Term	Description
transition definition	Workflow objects are split into two types: *Definition* objects and *Instance* objects. Definition objects provide the template for a running workflow instance. You create a new workflow by defining a Workflow Definition along with its State Definition and Transition Definition objects. When you run the workflow definition, the system creates a new Workflow object with an equivalent set of State and Transition objects that represent the run-time instances of the workflow definition. **Note:** We omit the "Instance" qualifier for brevity in the API and the UI. Each state can contain one or more transition objects. The transition definition object requires a name for the transition. This transition name will appear on the Workflow Definition Details page for quick reference and also on the State Definition Details page when you select the Transition Definitions tab. You can define one or more transitions for each state, depending on which transition options you want to apply to a particular state.
user	A user defines an account used to log into the system and control access to ElectricFlow objects. A user can be defined externally in an LDAP or Active Directory repository, or locally in ElectricFlow. See local user.
workflow	You can use a workflow to design and manage processes at a higher level than individual jobs. For example, workflows allow you to combine procedures into processes to create build-test-deploy lifecycles. A workflow contains and *transitions* you define to provide complete control over your workflow process. The ElectricFlow Workflow feature allows you to define an unlimited range of large or small lifecycle combinations to meet your needs. See workflow definition.
workflow definition	Workflow objects are split into two types: *Definition* objects and *Instance* objects. Definition objects provide the template for a running workflow instance. You create a new workflow by defining a Workflow Definition along with its State Definition and Transition Definition objects. When you run the workflow definition, the system creates a new Workflow object with an equivalent set of State and Transition objects that represent the run-time instances of the workflow definition. **Note:** We omit the "Instance" qualifier for brevity in the API and the UI.

Term	Description

workflow name template	This is the template used to determine the default name of jobs launched from the workflow definition. For example:
	`$[projectName]_$[/increment /myproject/workflowCounter]_ $[timestamp]`
	(substitute your values for the names above)
	Produces a workflow name such as:
	`projectName_123_20140102130321`

Workspace	A subtree of files and directories where job file data is stored. The term "workspace" typically refers to the top-level directory in this subtree.
workspace root	A directory in which ElectricFlow allocates job workspace directories. Each workspace root has a logical name used to refer to it in steps and procedures.
zone	A zone is a way to partition a collection of agents to secure them from use by other groups—similar to creating multiple top-level networks. For example, you might choose to create a developers zone, a production zone, and a test zone—agents in one zone cannot directly communicate with agents in another zone. A *default* zone is created during ElectricFlow installation. The server implicitly belongs to the default zone, which means all agents in this zone can communicate with the server directly (without the use of a gateway).

Appendix A: Plugins That Are Bundled with ElectricFlow

The following plugins are bundled in this version of ElectricFlow.

Name	Version	Name	Version
EC-ALM	1.1.2	EC-Kubernetes	1.1.2
EC-AmazonECS	1.0.5	EC-Maven	2.4.3
EC-Ansible	1.0.7	EC-MSBuild	2.0.5
EC-Ant	2.0.9	EC-MYSQL	2.0.8
EC-Artifactory	1.2.1	EC-OpenShift	1.4.1
EC-Azure	1.1.6	EC-Oracle	2.0.5
EC-AzureContainerService	1.1.0	EC-Powershell	2.1.0
EC-AzureContainerService-Docker	1.0.2	EC-Puppet	1.1.3
EC-Chef	1.2.1	EC-ReportEngine	1.0.0
EC-CloudFoundry	1.4.0	EC-Reports	2.1.0
EC-Core	1.2.5	EC-Security	1.2.1
EC-DefectTracking	1.1.7	EC-Selenium	2.0.6
EC-DefectTracking-JIRA	2.1.3	EC-SendEmail	1.0.1
EC-DefectTracking-TFS	2.0.8	EC-ServiceNow	2.2.0
EC-Docker	1.4.0	EC-Sonar	2.0.4
EC-Dynatrace	1.0.0	EC-SonarQube	1.1.1

Name	Version	Name	Version
EC-EC2	2.5.2	EC-SQLServer	2.0.7
EC-EMake	1.0.6	EC-Tomcat	2.3.0
EC-ESX	2.3.2	EC-WebLogic	3.4.0
EC-FileOps	2.0.5	EC-WebSphere	2.5.0
EC-FileSysRepo	1.0.0	ECSCM	2.2.12
EC-FlowLogCollector	1.0.0	ECSCM-File	2.0.3
EC-GoogleContainerEngine	1.1.0	ECSCM-Git	3.8.1
EC-HPQualityCenter	3.0.4	ECSCM-Perforce	2.8.6
EC-IIS	3.1.2	ECSCM-Property	2.0.1
EC-JBoss	2.6.0	ECSCM-Repo	2.2.0
EC-Jenkins	1.11.0	ECSCM-SVN	3.3.3
EC-Jetty	1.0.3	ECSCM-TFS	2.4.0
EC-JIRA	1.2.0	EF-Utilities	1.1.8
EC-Klocwork	2.0.3		

Appendix B: Using Special Characters in ElectricFlow Object Names

You should avoid using the following special characters when naming objects in ElectricFlow. These characters have special purposes or meanings within ElectricFlow or in the scripting language being used.

However, if any of these characters are used in object names, then your implementation must correctly escape them at runtime (meaning in context of the scripting language, such as JavaScript, Groovy, or Perl) to avoid errors or unexpected behavior.

Character	Description
`/`	Used for absolute property paths in ElectricFlow. The path syntax is similar to a file system path specification.
`[]` or `.`	Used in JavaScript.
`:`	Might cause problems when used in job names. Some operating systems do not allow directories containing this character.
`\`	Used for escaping.
(space) or `'`	Should be enclosed in quotes when used in ectool or JavaScript. Otherwise, it should be escaped properly. Also, spaces might cause issues when used in resource pool names and should be avoided.
`,` or `;`	Might cause issues when used in user names (such as in a list of approvers).