API Gateway

Version 7.6.2

14 July 2020

DevOps Deployment Guide

This documentation describes the following Axway software:

Axway API Gateway 7.6.2

Contents

Preface

This guide describes how to promote and deploy API Gateway configuration. This involves moving API Gateway configuration from one environment to another (for example, from development to testing to production), and configuring environment-specific values so that the configuration can be deployed in each environment.

It also describes how a team of policy developers can develop APIs and policies to be deployed as a single API Gateway configuration using a Source Code Management (SCM) system.

Note For a recommended DevOps approach to zero downtime upgrade, see the *API Gateway Upgrade Guide*.

Who should read this guide

The intended audience for this guide is policy developers and system integrators who are responsible for promoting and deploying API Gateway configuration from a development environment to a production environment.

Before deploying API Gateway configuration in a production environment you should consult the *API Gateway Administrator Guide* for information on planning and managing an API Gateway system in production. You should also understand exactly what message filters are, and how they are chained together to create a message policy. These concepts are documented in detail in the *API Gateway Policy Developer Guide*.

Others who might find parts of this guide useful include network or systems administrators and other technical or business users.

How to use this guide

This guide should be used in conjunction with the other guides in the API Gateway documentation set.

Before you begin promoting and deploying API Gateway configuration, review this guide thoroughly. The following is a brief description of the contents of each chapter:

Introduction to API Gateway deployment and promotion on page 12 – Describes promotion and deployment concepts, including API Gateway domains and configuration packages.

API Gateway deployment and promotion tasks on page 18 – Describes the tasks involved in promoting API Gateway configuration, such as environmentalizing configuration, and deploying policy and environment packages.

Configure package properties on page 24 – Describes how to configure package properties in Policy Studio and Configuration Studio.

Example: Promote from development to testing environment on page 27 – Provides a detailed example of the steps involved in promoting API Gateway configuration from a development environment to a production environment.

Sample promotion and deployment scripts on page 38 – Describes sample scripts that you can use to environmentalize configuration and promote configuration between environments.

Externalize API Gateway instance configuration on page 40 – Describes how to externalize API Gateway instance configuration using environment variables in the `envSettings.props` file.

Introduction to API Gateway team development on page 42 – Describes how policy developer teams can develop APIs and policies to be deployed as a single API Gateway configuration using a Source Code Management (SCM) system.

Team development best practices on page 49 – Describes the best practices for team development.

Manage API Gateway project dependencies on page 55 – Describes how to manage dependencies between team development projects.

Automate processes for continuous integration on page 58 – Describes how to use command-line tools to automate processes for continuous integration.

Manage X.509 certificates and keys on page 71 – Describes how to manage API Gateway certificates and keys.

Manage API Gateway users on page 84 – Describes how to manage API Gateway users.

Related documentation

The AMPLIFY API Management solution enables you to create, publish, promote, and manage Application Programming Interfaces (APIs) in a secure and scalable environment. For more information, see the *AMPLIFY API Management Getting Started Guide*.

The following reference documents are also available on the Axway Documentation portal at https://docs.axway.com:

- *Supported Platforms*

  Lists the different operating systems, databases, browsers, and thick client platforms supported by each Axway product.

- *Interoperability Matrix*

  Provides product version and interoperability information for Axway products.

Support services

The Axway Global Support team provides worldwide 24 x 7 support for customers with active support agreements.

Email support@axway.com or visit Axway Support at https://support.axway.com.

See "Get help with API Gateway" in the *API Gateway Administrator Guide* for the information that you should be prepared to provide when you contact Axway Support.

Training services

Axway offers training across the globe, including on-site instructor-led classes and self-paced online learning. For details, go to: http://www.axway.com/support-services/training

Accessibility

Axway strives to create accessible products and documentation for users.

This documentation provides the following accessibility features:

Screen reader support

- Alternative text is provided for images whenever necessary.
- The PDF documents are tagged to provide a logical reading order.

Support for high contrast and accessible use of colors

- The documentation can be used in high-contrast mode.
- There is sufficient contrast between the text and the background color.
- The graphics have the right level of contrast and take into account the way color-blind people perceive colors.

Updates and revisions

This guide includes the following documentation changes.

Changes in version 7.6.2

No changes.

Changes in version 7.6.1

No changes.

Changes in version 7.6.0

- Removed references to API Gateway server-side support for Windows.

Introduction to API Gateway deployment and promotion 1

This topic introduces the concepts in the deployment and promotion of API Gateway configuration. A typical enterprise-level customer will have several environments through which an API Gateway configuration will move from development to production. For example, this typically includes completely separate development, testing, and production domains. Promotion refers to the act of moving API Gateway configuration from one environment to another, and configuring environment-specific values so that the configuration can be deployed in each environment. For details on general API Gateway concepts, see the *API Gateway Concepts Guide*.

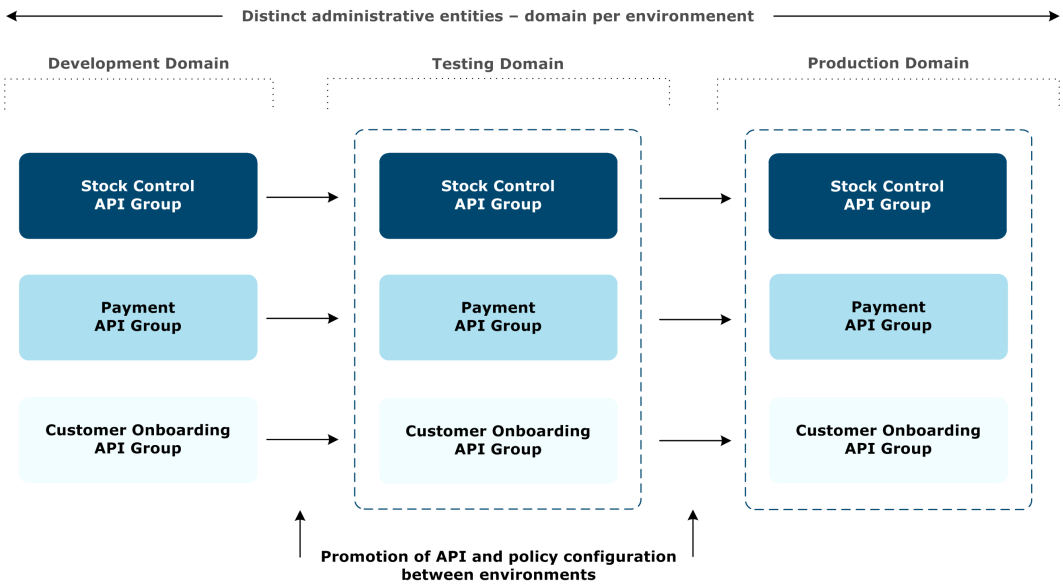
API Gateway supports a range of different operating systems. This means that API Gateway configuration can be promoted and deployed across environments running on different operating systems. For details on supported platform versions, see the *API Gateway Installation Guide*.

Environment topology

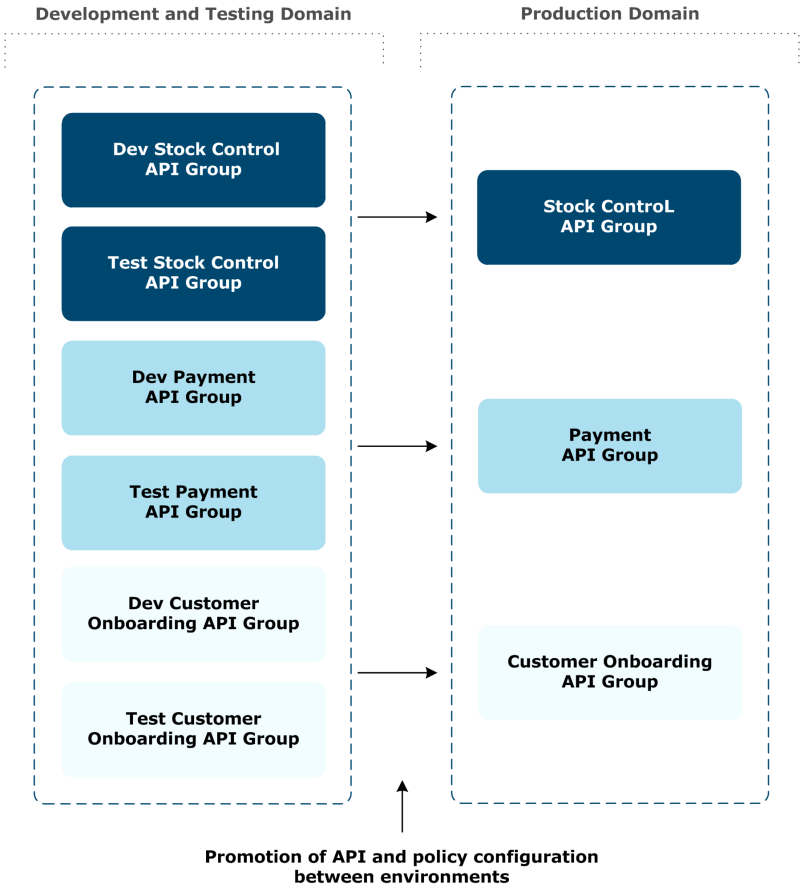
In a typical environment topology, each environment is implemented as a completely separate API Gateway domain. The exact mapping of environments to domains is determined by how each environment is administered, and which users have access rights.

Environments are distinct administrative entities in which only certain users have the privileges to perform operations. For example, only Production Operations staff have access to the *production* environment. In the API Gateway architecture, a domain is a distinct administrative entity for managing groups of API Gateways. For example, the production environment is implemented as a distinct production domain to which only Production Operations staff have access.

In the following diagram, each environment is implemented as a distinct API Gateway domain. Developers work in their own *development* environments, and then promote their API Gateway configurations to a central Testing team that performs testing in a single *testing* environment. When testing is complete, the Testing team promotes the API Gateway configurations to the Production Operations team for deployment in the production environment. Development, Testing, and Production Operations teams have access to their respective environments only. Therefore, each environment should be implemented as a distinct domain.



The API Gateway configuration is deployed to a group of API Gateways. Therefore, each domain consists of the required API Gateway groups to run the configurations. In the following diagram, the Development and Testing teams work in the same environment with common access to all API Gateway configurations. Therefore, in this case, there is a single domain for all the development and testing API Gateway groups.



Note The API Gateway does not mandate a specific environment-to-domain configuration, and is flexible enough to work with any architecture. However, you should manage your environments in an environment and domain topology. Implementing each environment as a distinct API Gateway domain is a good starting point.

Promotion and deployment

In a multienvironment topology, *promotion* refers to physically moving an API Gateway configuration between environments. For example, this might involve using FTP to transfer a configuration file, or loading and retrieving the file in a Configuration Management (CM) repository. In addition, promotion involves configuring environment-specific settings for the target environment (for example, users, certificates, and external connections to third-party systems). This is known as *environmentalization*.

Promotion typically involves two distinct tasks performed by different user types:

- In the downstream development environment, the policy developer prepares the configuration for promotion to upstream environments (for example, testing and production). This involves deciding what settings are environment-specific, and assumes expertise in policy development and configuration tools such as Policy Studio.
- The upstream user takes the configuration prepared by the policy developer, creates the environment-specific configuration, and deploys it. This is typically performed by an API Gateway administrator in upstream environments. The Configuration Studio tool used for this promotion step is designed for the skills of upstream administrators, and does not assume expertise in policy development and configuration.

Deployment refers to deploying configuration to an API Gateway group in a local domain. For example, you can deploy using the following tools:

- Policy Studio in a development environment
- API Gateway Manager in a testing environment
- Scripts in a production environment (for example, `managedomain` or a custom script)

For more details, see *API Gateway deployment and promotion tasks* on page 18.

API Gateway configuration

API Gateway configuration consists of the following types of information:

These component configuration types are described as follows:

Configuration type	Description	Environment specific
Policy	Policy rule definitions and configuration	Some might be environment specific
Listener	Configuration for listeners used by policies (for example, for HTTP, JMS, or TIBCO)	Some environment specific
External Connection	Settings for external configuration (for example, database connections or authentication repositories)	Some environment specific
Certificate	Security certificates and keys used by policies	All environment specific
User	Local user name and password store for API client authentication	All environment specific

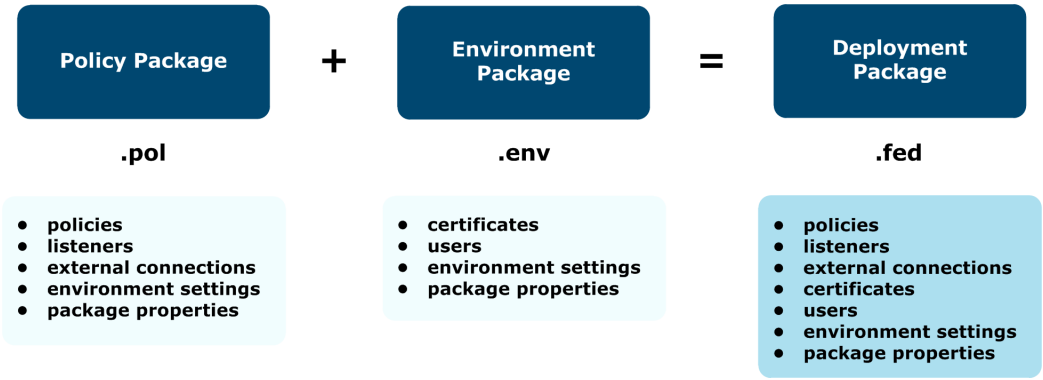
Configuration type	Description	Environment specific
Environment Setting	Environment-specific settings for environmentalized configuration in the policy, listener, and external connection configuration	All environment specific
Package Property	Name-value pairs used to describe the contents of the configuration package (for example, `Version`, `ID`, or `Timestamp`).	Some environment specific

API Gateway configuration packages

The API Gateway deployment and promotion tools bundle the API Gateway configuration into the following configuration package files:

Package	Description	Used when
Deployment package (`.fed`)	Contains all API Gateway component configuration (policy, listener, external connection, user, certificate, and environment setting). Implemented as a `.fed` file. This contains all of the data that would be contained in separate policy and environment packages combined.	Used by the policy developer in Policy Studio during the iterative development cycle to deploy all configuration. For more details, see *Deploy in a development environment* on page 18.
Policy package (`.pol`)	Contains the policy, listener, external connection, and environment setting configuration. Implemented as a `.pol` file. The environment settings in the `.pol` file contain a list of what has been environmentalized in the policy, listener, and external connection configuration. It does not contain the environment-specific values.	Used when promoting APIs and policy configuration to an upstream environment (for example, testing or production). Its contents remain unchanged in the upstream environment. For more details, see *Environmentalize configuration* on page 19.
Environment package (`.env`)	Contains the user, certificate, and environment setting configuration. Implemented as an `.env` file. The environment settings in the `.env` file contain a list of what has been environmentalized in the policy, listener, and external connection configuration, along with the environment-specific values.	Environment-specific settings used in upstream environments only (for example, testing or production). For more details, see *Promote upstream (first cycle)* on page 19.

The combined contents of the policy package and environment package are equivalent to the contents of the deployment package, which contains all API Gateway configuration.



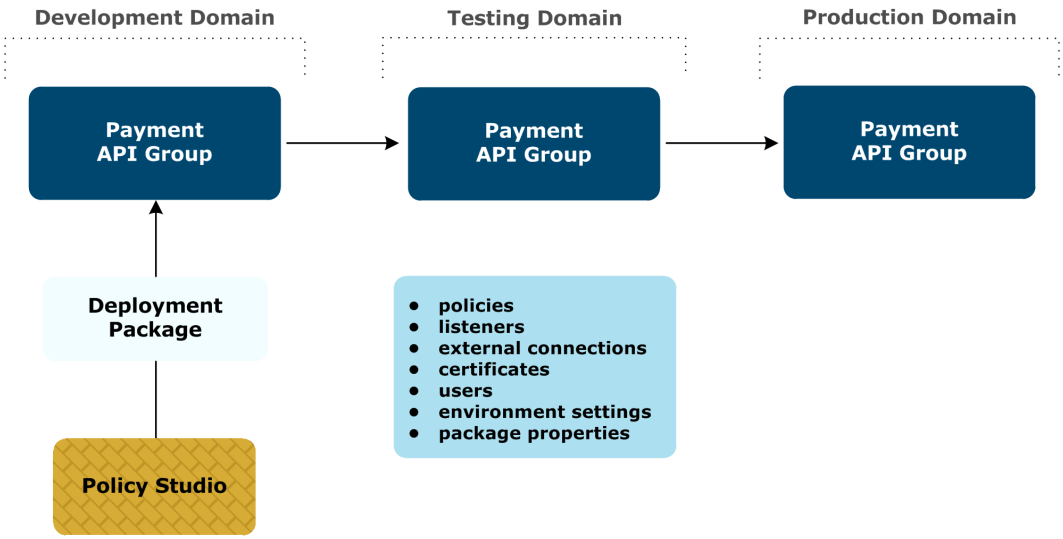
For more details on package properties, see *Configure package properties* on page 24.

API Gateway deployment and promotion tasks 2

This topic describes the tasks, tools, and architecture used in API Gateway deployment and promotion. It explains the breakdown of tasks performed by a policy developer in a development environment, and the tasks performed by an API Gateway administrator in an upstream environment (for example, testing or production).

Deploy in a development environment

In a development environment, the policy developer works in a continuous cycle of iterative development, deployment, and testing. In this environment, it makes sense to keep all API Gateway configuration in a single package. This enables the policy developer to deploy the API Gateway configuration directly from Policy Studio in a single *deployment package*. The following diagram shows an example environment topology.



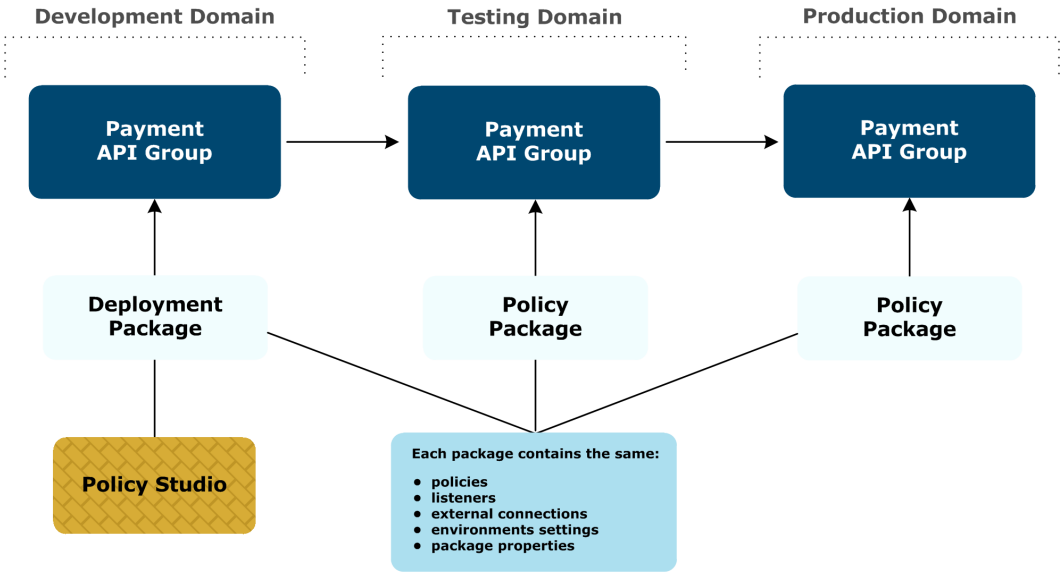
The deployment package contains the entire API Gateway configuration, and is implemented as a `.fed` file.

Environmentalize configuration

When development is complete, the policy developer must prepare the configuration for promotion to upstream environments. This involves *environmentalizing* the configuration that will require environment-specific settings in upstream environments. The policy developer performs the following tasks in Policy Studio:

- Selects the policy, listener, and external connection configuration settings that are environment specific.
- Enters values for these environment-specific settings to ensure that the configuration remains deployable in the Development environment. These environment-specific settings are contained in the environment settings in the deployment package. If you have already entered values for these settings, these are used so you do not have to manually re-enter them.
- Exports a *policy package* (`.pol` file) on disk for promotion. For example, this enables you to transfer the file using FTP to the upstream environments, or to load the file into a CM repository.

The following diagram shows an example environment topology.



The policy package that is exported for promotion is implemented as a `.pol` file. This file should remain unchanged when it is promoted to upstream environments.

Promote upstream (first cycle)

A *first cycle* promotion refers to promoting to an upstream environment in which no previous promotions have been performed. This means that the upstream environment is still running the default factory configuration that is installed with API Gateway. In this case, there is no existing upstream *environment package* (`.env`) to load into Configuration Studio at promotion time.

Create an environment package

In an upstream environment (for example, testing), the API Gateway administrator uses Configuration Studio to create an environment package that is specific to their environment. Because this is the first promotion cycle, the administrator opens the policy package (`.pol`) received from the development environment, and performs the following tasks:

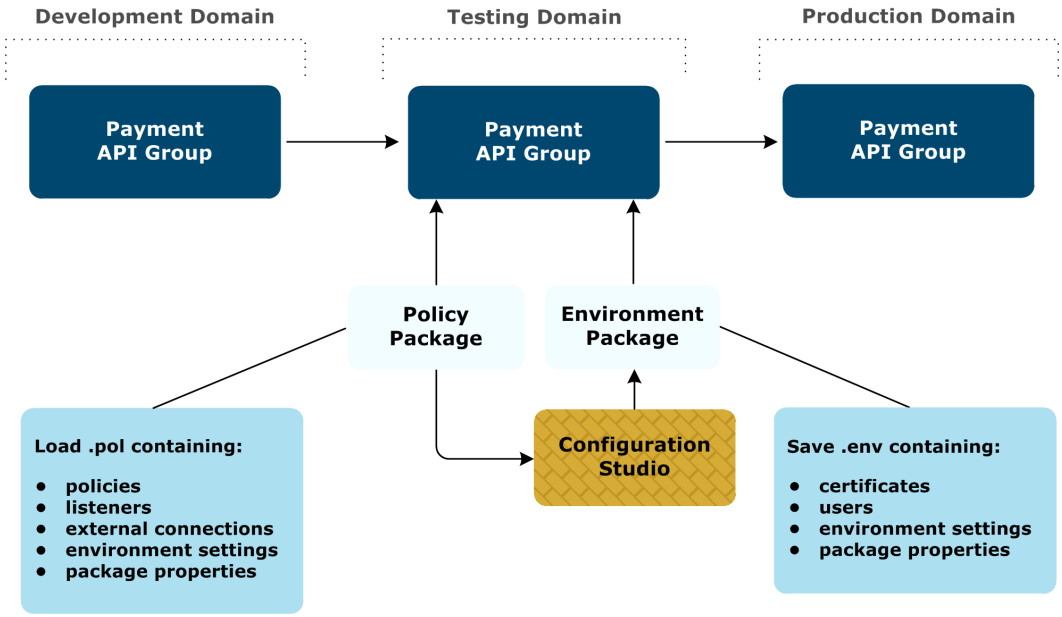
- Specifies values for the environment-specific settings selected in the development environment (for example, policy, listener, and external connections).
- Imports or creates certificates and keys.
- Defines users and user groups.
- Exports the environment package to a file on disk. The environment package is implemented as an `.env` file. For version history and rollback, you could also load the file into a CM repository.

Deploy the policy and environment packages

When the environment package has been created, the administrator can use API Gateway Manager or scripts to deploy both the policy package from the development environment and the newly created environment package. Each environment will have its own version of the `.env` file containing environment-specific settings, certificates, users, and so on. This constitutes a full deployable configuration when combined with the unmodified `.pol` file from the development environment.

Note Alternatively, the administrator can save a deployment package (`.fed`) from Configuration Studio, which merges the policy and environment package data. If you are not concerned with moving an unmodified policy package from the development environment to all upstream environments, you can save a single `.fed` file, and deploy this using API Gateway Manager or scripts (for example, if you want a single file for convenience).

The following diagram shows an example environment topology.



Note The environment settings in the environment package (`.env`) override the environment settings in the policy package (`.pol`). The environment settings in the policy package indicate settings for which you need to specify environment-specific values.

Promote upstream (subsequent cycles)

A *subsequent cycle* promotion refers to promoting to an upstream environment that has already had configuration promoted to it (any number of times). In this case, there is an existing version of the upstream environment package (`.env`) to load into Configuration Studio at promotion time.

Create the environment package

In the upstream environment, the API Gateway administrator uses the Configuration Studio to create an environment package specific to their environment that contains all the environment-specific settings, certificates, and so on. This is required for the new policy package received from the development environment. Because this is not the first promotion cycle, there is already an environment package deployed in this environment. The administrator must merge this with the new policy package from the development environment, which enables reuse of environment-specific settings already entered.

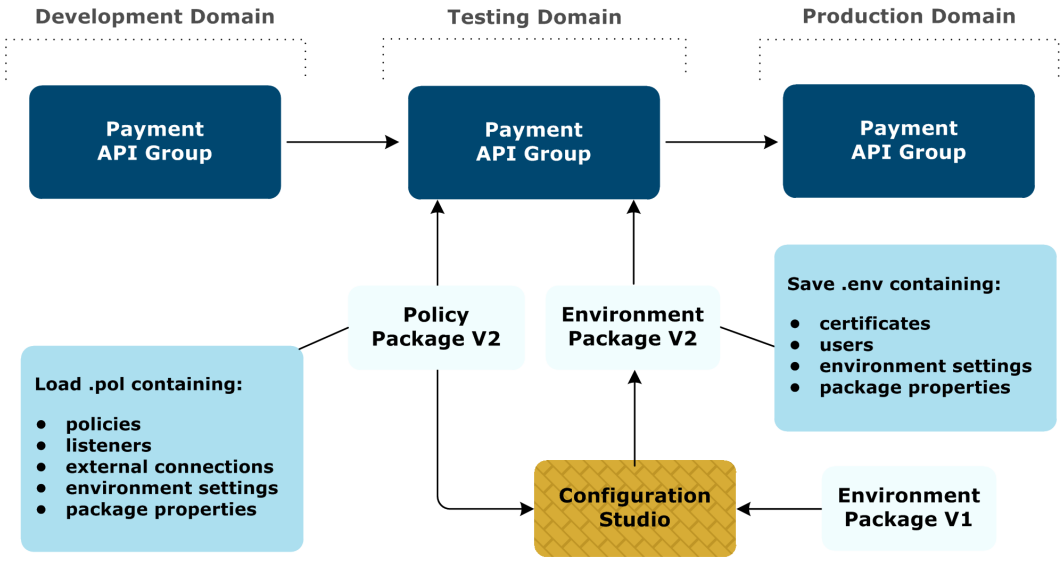
The administrator opens the new policy package from the development environment, and the environment package currently deployed in their environment. Opening these `.pol` and `.env` files displays a merged view of the environment settings. The administrator then performs the following tasks:

- Specifies values for new environment-specific settings required by the new policy package from the development environment.
- Updates values for environment-specific settings that previously existed (if necessary).
- Adds or removes certificates and keys.
- Adds or removes users and user groups.
- Exports the environment package to a file on disk. Alternatively, for version history and rollback, you could load the file into a CM repository.

Deploy the policy and environment packages

When the environment package has been created, the API Gateway administrator can then deploy both the policy package received from the development environment, and the new environment package using API Gateway Manager, or using scripts.

The following diagram shows an example environment topology.



Roll back configuration

You must ensure that you maintain a copy of previous configuration versions (policy and environment packages) in case you need to roll back and deploy an earlier configuration version. For example, you could use a Configuration Management (CM) repository to manage and roll back configuration package versions.

Multisite HA environments

Some environments might require different environment values for connections, certificates, and so on (for example, a remote High Availability (HA) site for a production environment in an active/passive configuration). In this scenario, the primary site is actively processing requests. The remote site is the backup passive configuration, deployed but not processing requests, and only becomes active if the primary site goes down. The same API Gateway configuration is deployed in both sites. Each site could be a separate domain, or one domain with different groups for each site. Specific environment values could be different for each site, for example, the remote site might connect to a different backup authentication server.

When the administrator receives the policy package (`.pol`) from the downstream environment, they can use Configuration Studio to create separate environment packages (`.env`) for the primary site and the remote site. The only difference between both environment packages is in the environment values required. In the primary site, the administrator deploys the policy package and the primary site environment package. In the remote HA site, the administrator deploys the same policy archive and the remote site environment package.

Passphrase-protected configuration

When you promote and deploy passphrase-protected configuration between environments (for example, from testing to production), the passphrase for the target configuration (production) can be different from the passphrase in the source configuration (testing).

If you are using a different passphrase in each environment, when the deployment takes place, you can specify the correct passphrase for the target configuration in Policy Studio. For details on setting encryption passphrases in Policy Studio, see the *API Gateway Administrator Guide*.

Configure package properties 3

The API Gateway configuration package files include property files that contain name-value pairs describing the package contents, and which are known as *package properties*. This topic describes these properties, and explains how to configure default and custom package properties using the Policy Studio and Configuration Studio tools.

The API Gateway bundles its configuration in the following package formats:

- Deployment package (`.fed`)
- Policy package (`.pol`)
- Environment package (`.env`)

For a description of each package, see *API Gateway configuration packages* on page 16.

Configure package properties

All three API Gateway configuration package formats (`.fed`, `.pol`, and `.env`) contain property name-value pairs, which you can use to describe the package contents. These package property values are stored in package property files (`.mf`). A deployment package (`.fed`) has two sets of package properties, one associated with the policy-related configuration, and one associated with the environment-related configuration. Policy packages (`.pol`) and environment packages (`.env`) have a single set of properties each.

Default properties

The default set of package properties that can be edited includes the following:

Property	Description
Name	Name associated with the configuration (for example, `Payment API Configuration`)
Description	Description associated with the configuration (for example, `API Gateway configuration settings for the Payment API`)
Version	Configuration version (for example, `v3`)

Property	Description
VersionComment	Comment relating to the configuration version (for example, `Added SSL`)

These fields are all free format text fields. You can set them to an empty value, or remove them completely, as required. The set of properties is completely customizable. You can add your own custom properties if required.

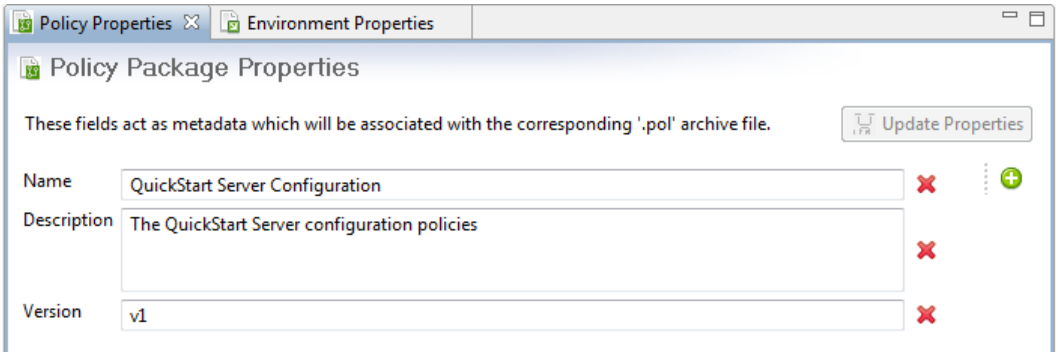
Read-only properties

The package also includes the following read-only, system-controlled package properties:

Property	Description
Id	A unique ID for the package
Timestamp	The time that the package was written

Configure properties in Policy Studio

When editing an API Gateway configuration in Policy Studio, you can add, edit, or remove the policy properties and environment properties in the **Environment Configuration** > **Package Properties** tree node. For example, the following window is displayed when you select **Policies**:



To add a new package property, click the add icon on the right of the window. Similarly, to delete a package property, click the delete icon to the right of the property.

Configure properties in Configuration Studio

You can edit environment properties in Configuration Studio using a similar window. You can only view policy properties because these are read-only.

Package property values are deployed to an API Gateway along with the entire configuration in the relevant configuration package structure.

4 Example: Promote from development to testing environment

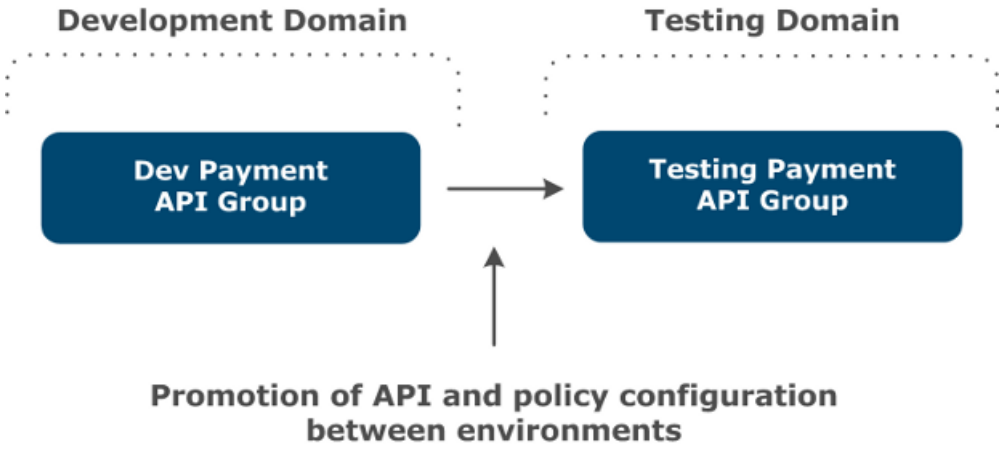
This topic describes a step-by-step example of promoting configuration from a development environment to a testing environment. If further promotions to more upstream environments are required, you can repeat Step 4 and Step 5 only.

Note Some environments (for example, testing and production) might be exact copies of each other, which enables you to deploy the same environment package to both environments. In these cases, repeat Step 5 only.

Example topology

This example assumes the following simple environment topology:

- A domain is configured in the development environment with a group of API Gateways named **Dev Payment API Group**.
- A domain is configured in the testing environment with a group of API Gateways named **Testing Payment API Group**. The configuration developed in the development environment must be promoted to the servers in this group.

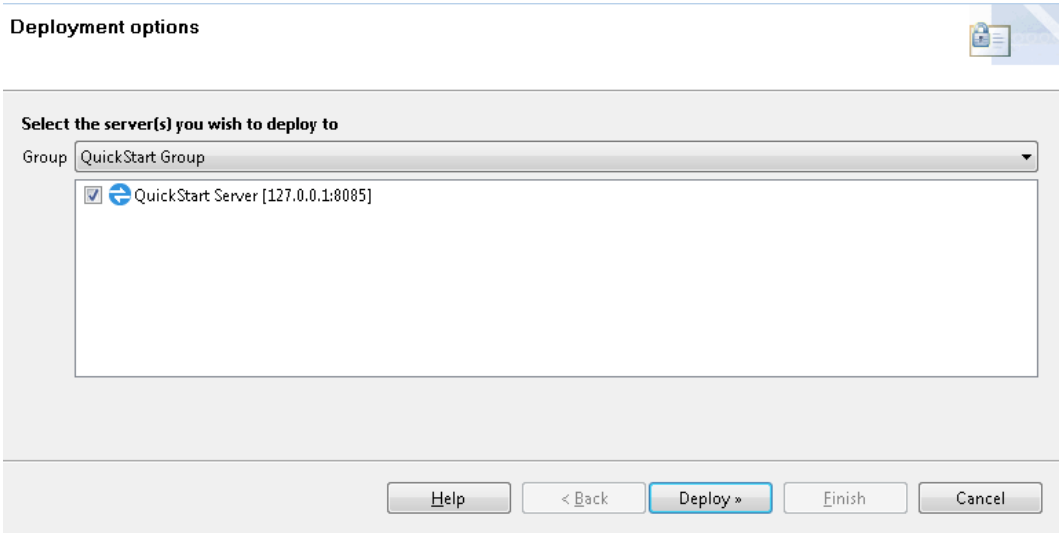


Step 1: Policy developer edits configuration and deploys in development environment

The policy developer in the development environment uses Policy Studio to create policies, users, certificates, listeners, and so on as required for the business solution they are developing. The policy developer will edit and deploy the configuration to the **Dev Payment API Group** repeatedly until they are finished with the configuration.

Deploy in Policy Studio

The policy developer deploys the configuration in Policy Studio by clicking the **Deploy** button in the toolbar when editing the configuration. This displays the following window:



Select the **Group** and API Gateway instances to which to deploy the configuration, and click **Deploy**. This uploads the configuration to the Admin Node Manager for the group, and then deploys it to the API Gateway instances on the hosts.

Note This simple example shows a group with a single API Gateway instance. Groups will typically have multiple API Gateway instances. If some Node Managers in the group are not running, do not select the API Gateways on those hosts, and you can still deploy to the other hosts in the group.

Step 2: Policy developer environmentalizes the environment-specific settings

When the policy developer is developing policies in an iterative manner as described in Step 1, they might choose not to consider what settings are environment-specific yet, or they might choose to environmentalize these settings as they go along. Either way, before promotion can occur, all settings that are environment-specific must be environmentalized to prepare the configuration for promotion to upstream environments.

Display environmentalized configuration

You must first enable the display of configuration settings that are assigned for environmentalization in Policy Studio. Select **Window** > **Preferences** > **Environmentalization** in the main menu, and select **Allow environmentalization of fields**.

Environmentalize configuration settings

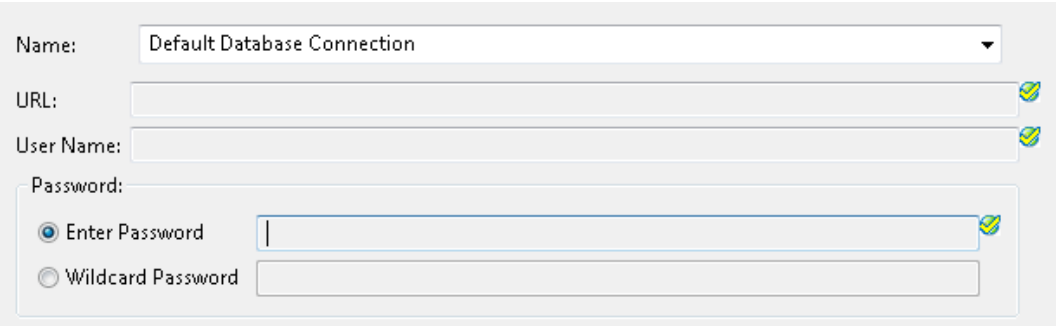
For example, the developer chooses to environmentalize the following settings in the configuration:

- **URL**, **User Name**, and **Password** fields in a **Default Database Connection**
- **URL** field in a **Connect to URL** filter in a policy named **GetProducts**
- **X.509 Certificate** field in an HTTPS interface named **OAuth 2.0 Interface**
- **URL**, **User Name**, **Password**, and **Signing Key** fields in a **Sample Active Directory Connection**

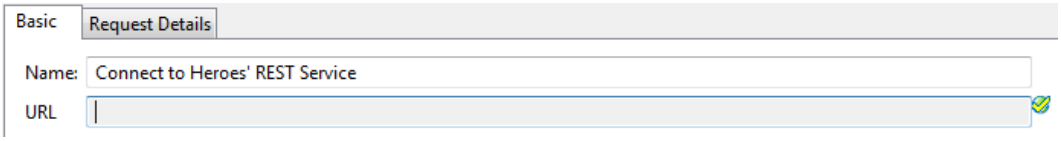
The policy developer edits the database connection, **Connect to URL** filter, HTTPS interface, and LDAP connection. You can click the **Environmentalize** icon (globe icon on the right of the fields) as shown in the following examples. Alternatively, press **Ctrl-E** to environmentalize a selected field.

Tip You must give the field focus before the **Environmentalize** icon is displayed.

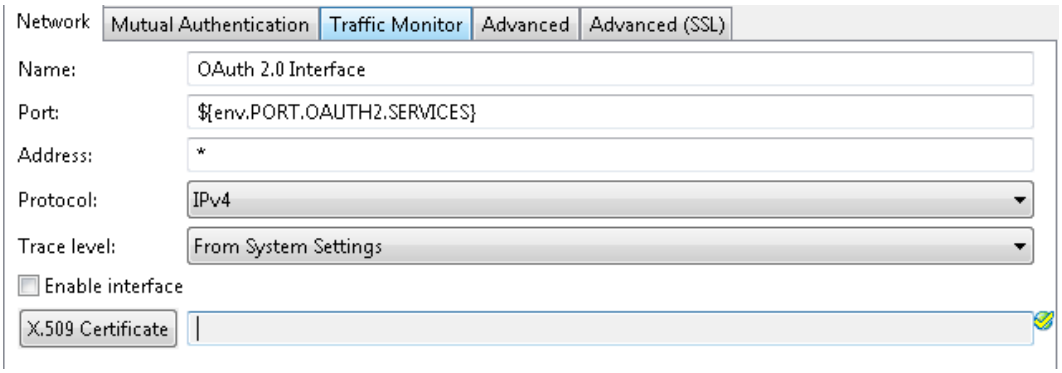
For example, to environmentalize the database connection, select **Environment Configuration** > **External Connections** > **Database Connections** > **Default Database Connection** > **Edit**, and click **Environmentalize** next to the appropriate fields:



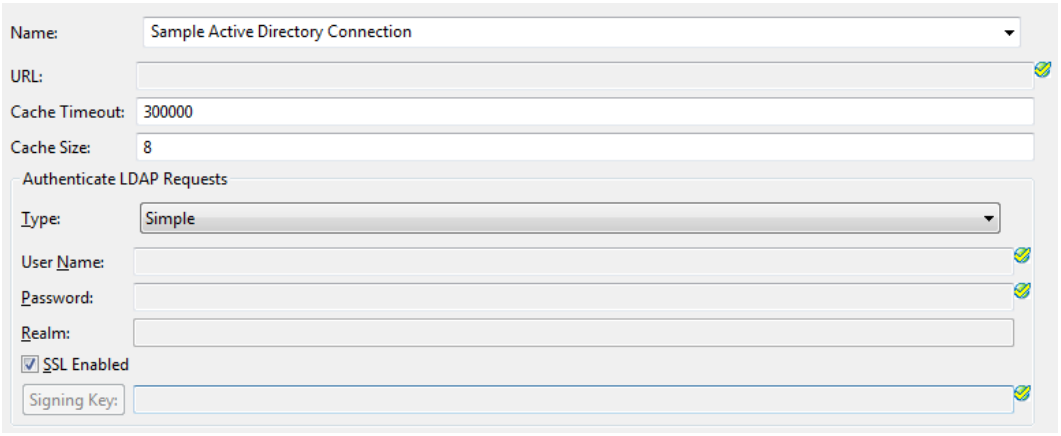
To environmentalize the **Connect to URL** filter, Select **Policies** > **QuickStart** > **Virtualized Services** > **REST** > **GetProducts** > **Connect to Heroes' REST Service**, and click **Environmentalize** next to the **URL** field:



To environmentalize the HTTPS interface, select **Environment Configuration** > **Listeners** > **API Gateway** > **OAuth 2.0 Services** > **Ports** > **OAuth 2.0 Interface**, and click **Environmentalize** next to the **X.509 Certificate** field:



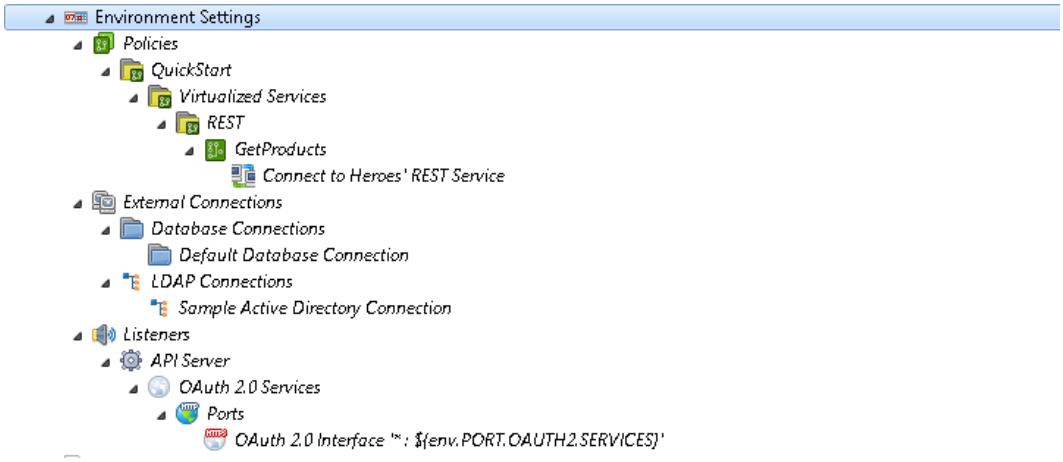
To environmentalize the LDAP connection, select **External Connections** > **LDAP Connections** > **Sample Active Directory Connection** > **Edit**, and click **Environmentalize** next to the appropriate fields:



When configuration settings have been environmentalized, the corresponding node in the Policy Studio tree is displayed with a globe icon and bold text.

View environment settings

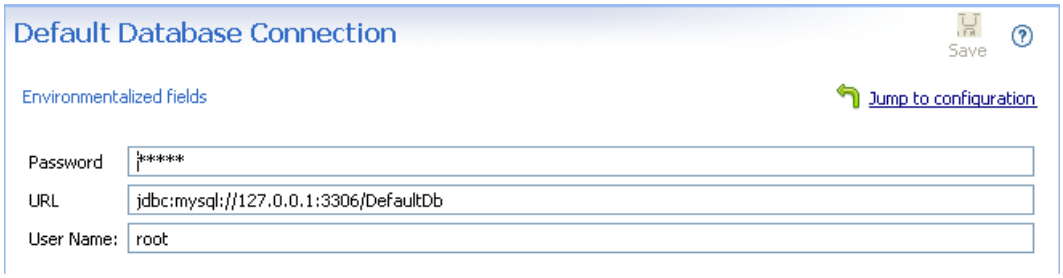
After environmentalizing the fields, the following nodes are available under the **Environment Configuration** > **Environment Settings** node in Policy Studio:



Update environment settings

Assuming the policy developer has already entered values for the fields that they have selected to be environmentalized, these values are automatically specified under the **Environment Configuration** >**Environment Settings** node. To update the setting values for the development environment, you must use the **Environment Settings** tree node.

For example, using the example environmentalized settings, the following window is displayed when you select **Environment Settings** > **External Connections** > **Database Connections** > **Default Database Connection**:



Deselect environment settings

To disable environmentalization for a setting, you can right-click its node in the **Environment Configuration** > **Environment Settings** tree, and select **Remove**. This also deselects the field in the window used to edit the configuration setting (for example, database connection). The value configured before environmentalization is displayed again.

Alternatively, you can click the **Jump to configuration** link, and return to the window used to edit the configuration setting, and deselect the **Environmentalize** icon on this field, or press **Ctrl-E**. This also removes the field as a setting to be configured under the **Environment Settings** tree. The value configured before environmentalization is displayed again.

Deploy the configuration

After all environment-specific fields have been selected, and appropriate values set for the development environment, the policy developer should deploy and test the updated configuration. For details on deploying to the group, see *Step 1: Policy developer edits configuration and deploys in development environment* on page 28. The deployment package (`.fed`) deployed to the **Dev Payment API Group** will contain the entries in the environment settings store, and the associated values suitable for the development environment.

Environmentalize reference fields

Configuration fields that point to other fields are known as *reference fields*. For example, in an HTTPS Interface or **XML Signature** filter, you environmentalize a reference to an X.509 certificate. You can also environmentalize references to complex types such as authentication repositories. If a reference to an authentication repository is environmentalized, you could set the repository to the local user store in the development environment, and to an LDAP repository in the testing environment.

The standard way to environmentalize a certificate at group level is to click **Environmentalize** on its configuration window. Environmentalizing a certificate, or any other reference field, is the same as all other fields. For example, when you environmentalize the signing certificate in an **XML Signature** filter, the **Environment Configuration** > **Environment Settings** tree where you enter environment-specific values displays a node for the **XML Signature** filter. The window on the right includes a **Signing Key** button to display a list of available certificates. You must select one of these certificates in Configuration Studio or Policy Studio. This field will most likely be prepopulated in Policy Studio if you already selected a certificate before clicking **Environmentalize**.

Alternatively, you can environmentalize a certificate using an alias. For example, in the development environment, the **XML Signature** filter could use a certificate named `MySigningCert`. The policy package (`.pol`) created from the development environment must be merged with an environment package (`.env`) that contains a certificate with the same alias.

Note You can also environmentalize certificates using an alias at the API Gateway instance level as described in *Externalize API Gateway instance configuration* on page 40. However, certificates are normally environmentalized at the API Gateway group level as described in this topic.

Step 3: Policy developer saves policy package in Policy Studio for promotion

The policy developer finishes editing and environmentalizing the configuration that they are running with, and deploys it to the API Gateway. They must then save the policy package in Policy Studio to enable promotion to the testing environment. To save the policy package, perform the following steps:

1. When the active configuration is loaded, select **File > Save > Policy Package**.

   Note Before creating the policy package, Policy Studio automatically detects any unenvironmentalized certificate references, and enables you to automatically environmentalize these settings before proceeding.

2. Browse to the directory in which to save the package, and enter its filename (for example, `c:\temp\payment.pol`).
3. Click **Save**.

A policy package (`.pol`) file is created on disk. The policy developer must transfer this file to the testing environment using some external mechanism (for example, FTP or email).

Note The steps described so far are the same for first and subsequent cycle promotions. For the first cycle, the policy developer will most likely use the default factory configuration as their starting point for editing the configuration. In subsequent cycles, the starting point will most likely be the existing configuration currently deployed to the **Dev Payment API Group**.

Step 4: API Gateway administrator creates testing environment package in Configuration Studio

This step depends on whether this is a first cycle promotion or a subsequent cycle promotion.

First cycle promotion: Open the policy package

If this is a first cycle promotion, the testing API Gateway administrator uses Configuration Studio to open the policy package created in the development environment by the policy developer in Step 3. The administrator does not need to open an environment package for a first cycle promotion. To open the policy package, perform the following steps:

1. Open a command prompt, and change to your Configuration Studio installation directory (for example, `INSTALL_DIR\configurationstudio`).
2. Enter `configurationstudio` to start Configuration Studio.

3. Select **File > Open File**.
4. Enter or browse to the location of the **Policy Package** (for example, `c:\temp\payment.pol`).
5. Click **OK**.

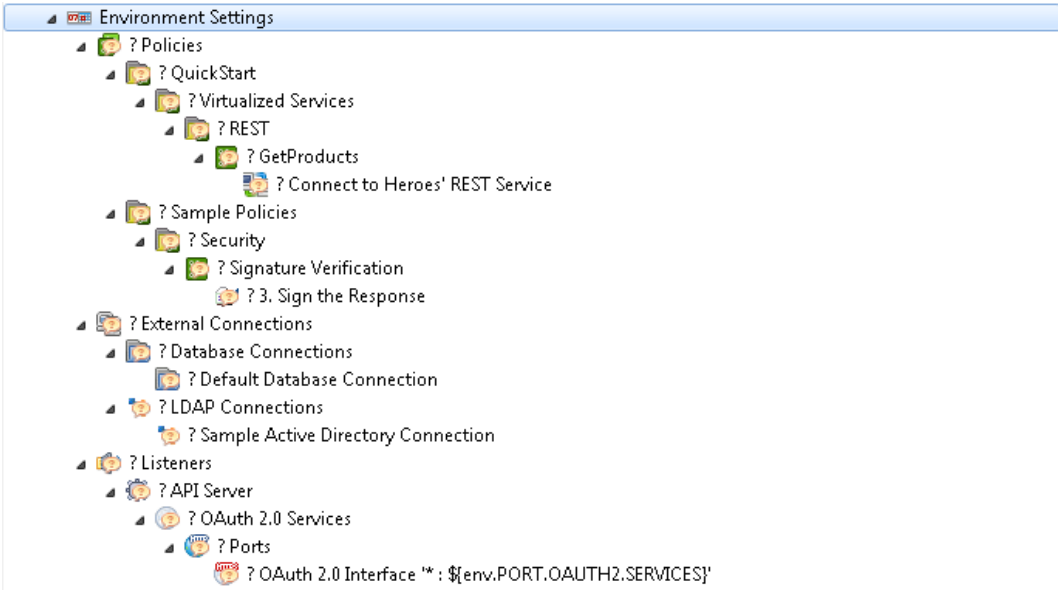
Note The Configuration Studio opens policy packages and environment packages by opening files available on disk. The administrator must ensure that the required files are available to the application.

Subsequent cycle promotion: Open the policy and environment packages

If this is a subsequent cycle promotion, the testing API Gateway administrator uses the Configuration Studio to open the policy package created in the development environment by the policy developer in Step 3. You must also open the currently deployed environment package for the testing environment. If you do not open the currently deployed environment package at this point, you might need to re-enter certificates and settings that you entered for the previous promotion.

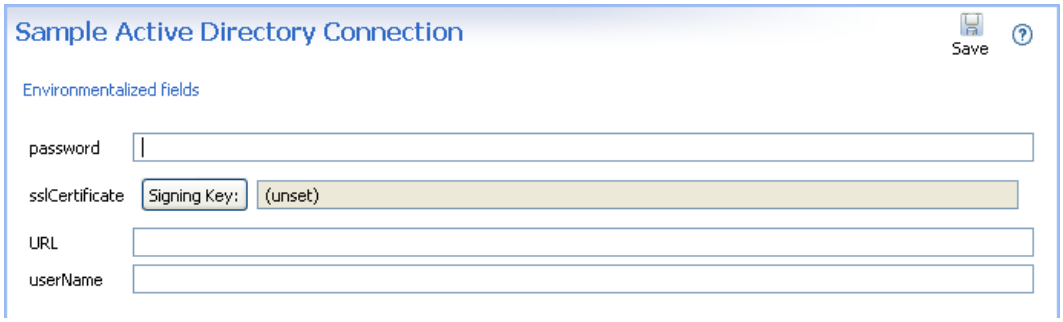
Specify environment settings

The administrator must navigate the **Environment Configuration** > **Environment Settings** tree node in Configuration Studio, and enter values that are specific to the testing environment. Values from the development environment are not displayed. The **Environment Settings** tree displays a question mark next to any settings without values. For a first cycle promotion, initially all environment setting values will be empty. Assuming a first cycle promotion with the sample data from Step 1, the **Environment Settings** tree is displayed in Configuration Studio as follows:



For subsequent cycle promotions, settings that are still required by the new policy package from the development environment, and that existed in previously promoted policy packages, will have values configured. Any certificates, keys, user, and user groups previously created will also be shown. Environment settings that existed in previously promoted configuration but are no longer required will be removed. New settings in the new policy package are listed with no value.

For example, assuming a first cycle promotion using the example environmentalized settings, the following window is displayed when you select **Environment Configuration** > **Environment Settings** > **External Connections** > **LDAP Connections** > **Sample Active Directory Connection**:



Note You cannot add or delete environment settings using Configuration Studio. These are predetermined by the policy developer in the development environment.

Update certificates and users

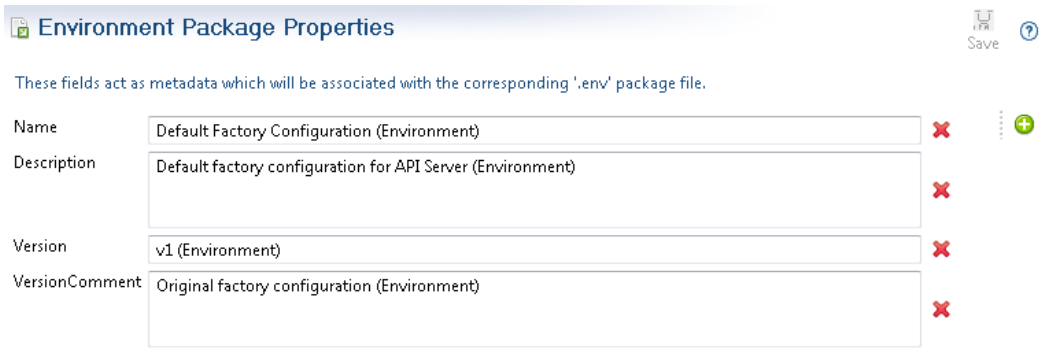
The administrator can add, edit, or remove new certificates, keys, users, and user groups in Configuration Studio in the same way as in Policy Studio. For more details, see the following topics:

- *Manage X.509 certificates and keys* on page 71
- *Manage API Gateway users* on page 84

Note If a certificate reference has been environmentalized, such as in the **Sample Active Directory Connection**, you must create or import a testing environment certificate in Configuration Studio. This makes the certificate available for selection when the environmentalized settings are edited in the **Environment Settings** tree in Configuration Studio.

Update package properties

At any time, the API Gateway administrator can edit the environment package properties by selecting the **Environment Configuration** > **Package Properties** > **Environment** tree node in Configuration Studio. For example:



If the API Gateway administrator selects the **Environment Configuration** > **Package Properties** > **Policy** node, this displays a read-only view of the policy package properties on a similar window. For example:

Note You cannot edit the contents of the policy package file in Configuration Studio.

Save the environment package

When you have entered all the environment-specific settings for the testing environment, select **File > Save > Environment Package** in Configuration Studio. An environment package (`.env`) file is saved to disk.

Step 5: API Gateway administrator deploys configuration to testing environment group

The testing API Gateway administrator takes the policy package unchanged from the development environment created in Step 3, and the environment package created using Configuration Studio for the testing environment created in Step 4, and deploys them to the **Testing Payment API Group** using API Gateway Manager or scripts.

For example, to deploy using API Gateway Manager, perform the following steps:

1. Enter the following URL in your browser to launch API Gateway Manager:

   ```
   https://127.0.0.1:8090/
   ```

2. On the **Dashboard** tab, select the API Gateway group in the **TOPOLOGY** section.
3. Click the **Edit** button on the right of the group, and select **Deploy Configuration**.
4. Select **I wish to deploy configuration contained in a Policy Package and Environment Package**, and browse to the `.pol` and `.env` files.
5. Click **Deploy**.

Step 6: Further configuration updates in testing environment

This section describes how to update environment-specific settings using Configuration Studio, and if necessary, using Policy Studio.

Update environment settings using Configuration Studio

If further updates are required to the environment-specific settings in the testing environment, the testing API Gateway administrator can open the policy package and environment package files in Configuration Studio at any time, and update the contents for the environment package file. The administrator can then deploy the policy package and updated environment package files to the **Testing Payment API Group** using API Gateway Manager or scripts.

Update environment settings using Policy Studio

Normally the policy package will be promoted through to upstream environments without any updates. However, in some cases, a single policy package for all environments will not be possible. For example, you might wish to use different authorization filters in development and testing environments. But the policy developer might not have sufficient knowledge to create the necessary configuration for all upstream environments in the policy package. In this case, the API Gateway administrator in the upstream environment must use Policy Studio to make the required changes.

The administrator will open a policy package from the development environment and the current testing environment package (if one exists) in Policy Studio, before making the testing environment-specific updates to the configuration. The administrator can save a policy package (`.pol`) and an environment package (`.env`) from Policy Studio. They can deploy them as usual to the **Testing Payment API Group** using API Gateway Manager or scripts. Alternatively, they can save a single deployment package (`.fed`), and deploy this package.

Sample promotion and deployment scripts 5

The API Gateway provides a collection of sample scripts to enable you to automate various common administration tasks. These scripts are based on the Jython Java scripting interpreter (see http://www.jython.org). You can extend these scripts to suit your needs by using the Jython language syntax. All Jython sample scripts are found in the following directory in your API Gateway installation:

```
INSTALL_DIR/samples/scripts
```

Note For details on packaging and deployment tools, see *Automate processes for continuous integration* on page 58.

Run the sample scripts

To run a sample script, call the `run` shell in the `/samples/scripts` directory, and specify the script to run. For example:

```
sh run.sh config/getEnvSettings.py
```

Scripts for environmentalizing configuration

You can use the following scripts in the `/sample/scripts/environmentalize` directory to environmentalize API Gateway configuration:

Script name	Description
`addEnvSettings.py`	Downloads a deployment package (`.fed`) from an API Gateway. Marks the **Traffic HTTP Interface** port field to be environmentalized. Creates an environment settings entry for the port, and sets it to `7878`.
`getEnvSettings.py`	Connects to an API Gateway and lists all the fields that have been marked for environmentalization. The associated values in environment settings are output.

Script name	Description
`mergeEnvSettings.py`	Offline script that does not connect to an API Gateway. Merges a policy package (`.pol`) from downstream with an environment package (`.env`) from upstream, and merges them to create a deployment package (`.fed`).
`removeEnvSettings.py`	Downloads a deployment package (`.fed`) from an API Gateway. Removes the **Traffic HTTP Interface** port field from being environmentalized (opposite of the `addEnvSettings.py` script).

Scripts for promoting configuration

You can use the following scripts in the `/sample/scripts/migrate` directory to promote API Gateway configuration:

Script name	Description
`archive.py`	Downloads the current API Gateway deployment package (`.fed`), policy package (`.pol`), and environment package (`.env`) files from the Node Manager.
`createDeploymentPackage.py`	Creates a deployment package (`.fed`) from policy (`.pol`) and environment (`.env`) packages. Where the policy and environment packages are obtained from is out of scope for this script. For example, they could have been obtained from a running server (see `archive.py`), a source code repository (CVS, Git, SVN, and so on), or an FTP or USB connection.
`envMigrate.py`	Demonstrates the promotion of configuration from a development environment to a staging environment.

Externalize API Gateway instance configuration 6

When API Gateway configuration is deployed to a group, the configuration package settings are applied to all API Gateway instances in the group. You can also specify API Gateway configuration values on a per-API Gateway instance basis using environment variables in the `envSettings.props` file. For example, you can specify different values for the port on which the API Gateway listens for HTTP traffic, depending on the environment in which the API Gateway is deployed.

The environment variable settings in the `envSettings.props` file are external to the API Gateway core configuration. The API Gateway runtime settings are determined by a combination of external environment variable settings and core configuration. This mechanism provides a simple and powerful approach to configuring specific API Gateway instances in the context of API Gateway group configuration defined in policy and environment packages.

Note Environment variables in the `envSettings.props` file apply to the API Gateway instance only. Configuration packages (`.fed`, `.pol`, and `.env` files) apply to the API Gateway group.

The `envSettings.props` file is located in the `conf` directory of your API Gateway installation, and is read each time the API Gateway starts up. Environment variable values specified in the `envSettings.props` file are displayed as environment variable selectors in the Policy Studio (for example, `${env.PORT.TRAFFIC}`). For more details on selectors, see the *API Gateway Policy Developer Guide*.

Configure environment variables

The `envSettings.props` file enables you to externalize configuration values and set them on a per-server environment basis. This section shows the configuration syntax used, and shows some example values in this file.

Environment variable syntax

If the API Gateway configuration contains a selector with a format of `${env.X}`, where *X* is any string (for example, `MyCustomSetting`), the `envSettings.props` file must contain an equivalent name-value pair with the following format:

```
env.MyCustomSetting=MyCustomValue
```

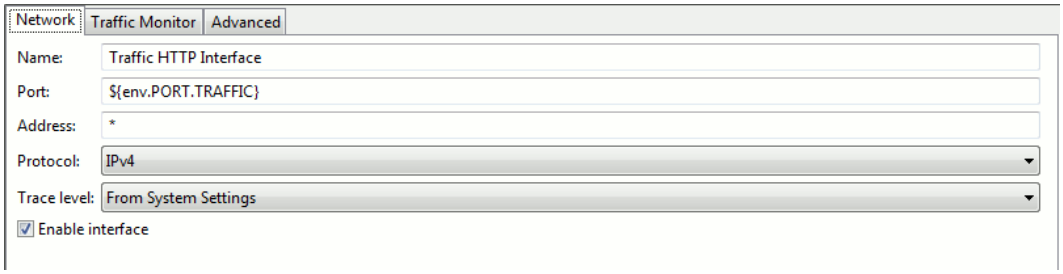
When the API Gateway starts up, every occurrence of the `${env.MyCustomSetting}` selector is expanded to the value of `MyCustomValue`. For example, by default, the HTTP port in the server configuration is set to `${env.PORT.TRAFFIC}`. Specifying a name-value pair of `env.PORT.TRAFFIC=8080` in the `envSettings.props` file results in the server opening up port 8080 at startup.

Example settings

The following simple example shows some environment variables set in the `envSettings.props` file:

```
# default port API Gateway listens on for HTTP traffic
env.PORT.TRAFFIC=8080
# default port API Gateway listens on for management & configuration HTTP traffic
env.PORT.MANAGEMENT=8090
# path to sample directory in API Gateway instance directory
env.SAMPLE.PATH=${environment.VINSTDIR}/sampleDir
```

The following example shows the corresponding `${env.PORT.TRAFFIC}` selector displayed in the **Configure HTTP Interface** dialog. At runtime, this is expanded to the value of the `env.PORT.TRAFFIC` environment variable specified in the `envSettings.props` file:



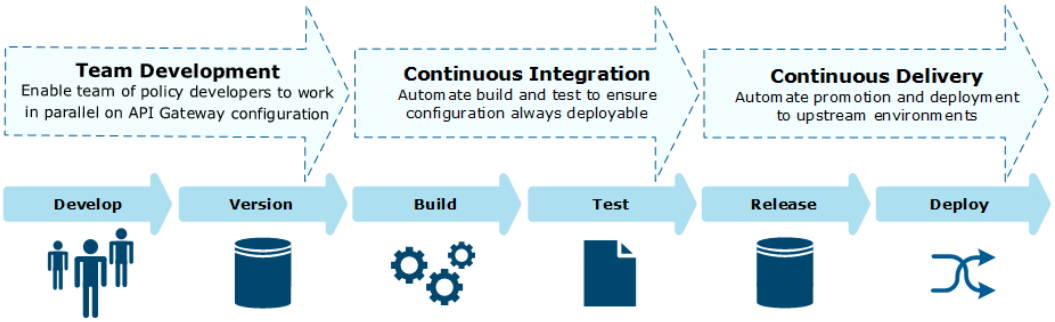
Note All entries in the `envSettings.props` file use the `env.` prefix, and the corresponding selectors specified in Policy Studio use the `${env.*}` syntax. If you update the `envSettings.props` file, you must restart or deploy the API Gateway for updates to be applied to the currently running API Gateway configuration.

Introduction to API Gateway team development 7

This topic introduces the concepts and main features of API Gateway team development. This enables a team of API Gateway policy developers to work in parallel developing APIs and policies to be deployed as a single API Gateway configuration using a Source Code Management (SCM) system.

In addition, team development enables continuous integration (CI) and continuous delivery (CD) practices to be used in an API Gateway system. This enables the API Gateway to use best practices for development, deployment, and promotion, and to support the increasing use of DevOps tooling.

The following diagram shows the different stages of delivery, from development through to deployment.



The following details the various tasks that you might perform at each stage.

Develop	Team of developers develop API and common projects in parallel on local development environments	Run unit tests on projects prior to SCM check in
Version	Manage and version projects in SCM	Diff projects to identify conflicts Audit who changed what when Control developer access to projects Rollback changes
Build	Build script (`projpack`) builds API Gateway configuration by merging API and common projects	Creates the policy package (`.pol`) containing all the projects
Test	Deployment script (`projdeploy`) deploys configuration to test environment to execute system tests	Deploys policy and environment packages for test environment Run full test suites on entire configuration to detect regressions

Release	Manage and version configuration in package management repository	Policy and environment packages by version and environment
Deploy	Deploy configuration from package management repository to target environments	Extract and deploy policy and environment packages for specific configuration version and environment

This topic contains the following sections:

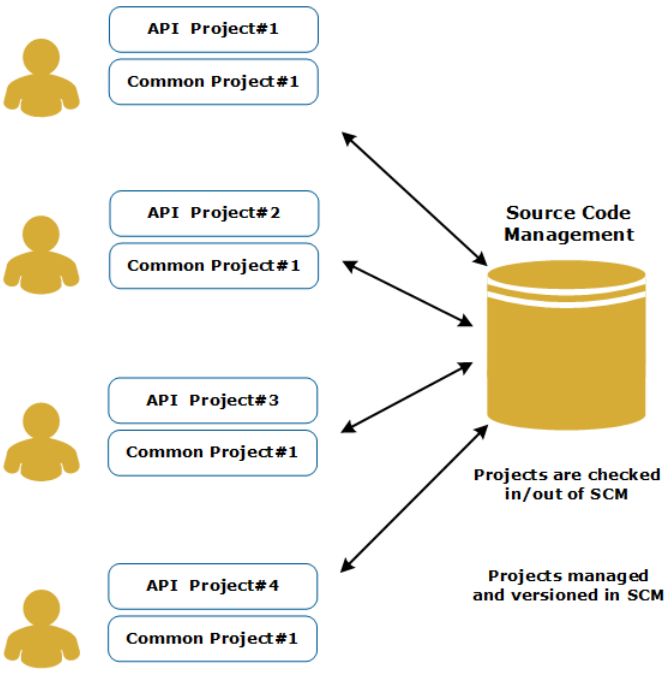
API Gateway projects

Note To enable team development project templates and project dependencies in API Gateway, you must turn on the team development option in Policy Studio preferences. In Policy Studio select **Window > Preferences > Team Development** and select **Enable Team Development**.

API Gateway provides a project-based approach to developing APIs, policies, and associated resources. API Gateway configuration is split into independent projects:

- API projects – Contain resources that are specific to an API and not shared with other APIs. Your configuration can contain multiple API projects.
- Common project – Contains shared resources used by multiple APIs. Your configuration can contain only one common project with server settings.

Policy developers develop and test individual API projects and a common project in a local development environment. These projects must be managed and versioned in a SCM system.



API project resources

An API developed on the API Gateway consists of a set of resources developed in Policy Studio:

- Listener configuration (for example, HTTP URL path, JMS queue, and so on), which is the entry point to the API
- Policies that implement the API and integrate with back-end systems and services
- Other resources (for example, XSLTs, scripts, and so on) that the policies use to implement the API

All these resources are specific to the API, and not shared with other APIs. Therefore each API can be packaged as a standalone API project with no dependencies on other API projects.

Common project resources

Some API Gateway deployments might have resources that are common to and used by multiple APIs. For example, a corporate security policy that must be applied to all APIs. These common resources can be packaged in a *dependent project* that can be used by multiple APIs.

For example, if an API Gateway deployment consists of 100 APIs, and a common set of corporate security policies, the configuration will be organized as 100 API projects, and a dependent project containing the common security policies.

For more details, see *Manage API Gateway project dependencies* on page 55.

Project versioning

Use of a SCM system is required as the system of record for managing and versioning projects. Projects are the granular unit of configuration that is managed and versioned in the SCM. This enables a version history of APIs to be maintained in the SCM, and allows policy developers to roll back to previous versions of an API.

The SCM also maintains an audit trail of changes to individual projects, allowing organizations to determine who changed what and when. In addition, the SCM can be used to control what level of access individual policy developers have to specific projects.

Policy developer workflow

Note To enable team development project templates and project dependencies in API Gateway, you must turn on the team development option in Policy Studio preferences. In Policy Studio select **Window > Preferences > Team Development** and select **Enable Team Development**.

When team development is enabled, the primary experience of using Policy Studio is project-centric and file-system oriented. Policy developers use Policy Studio to open projects from the file system, edit policies and other resources in the project, and save the project including their changes back to the file system.

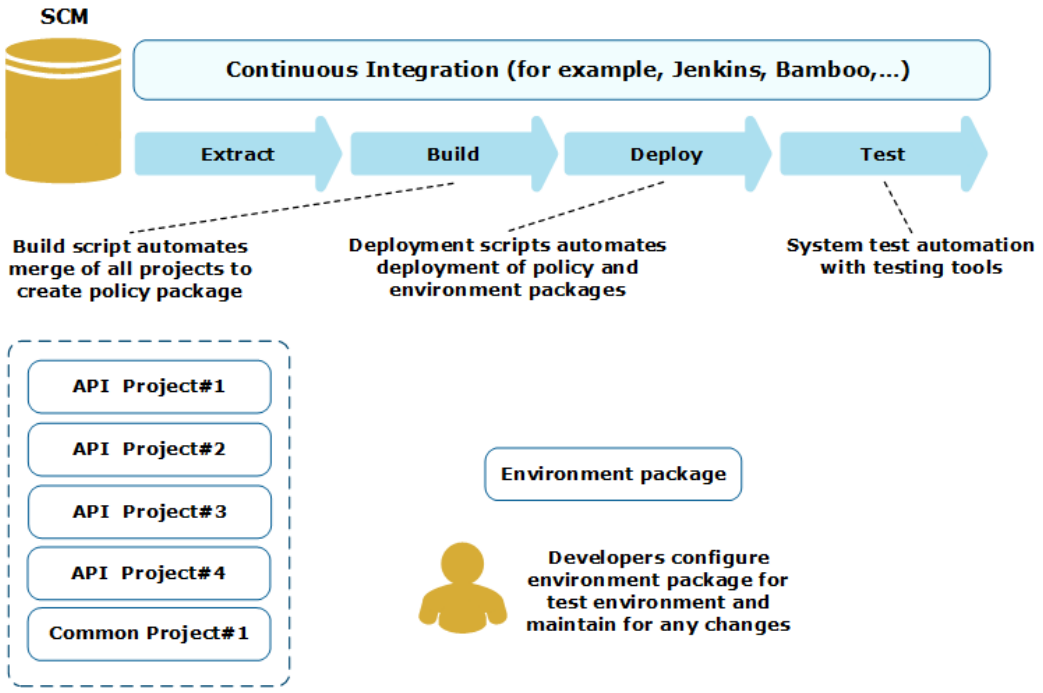
If the project being worked on has a dependent project, the dependent project can be opened read-only in Policy Studio. Then in Policy Studio, the policy developer can deploy the project being worked on and the dependent project to an API Gateway to test their changes.

The integration between Policy Studio and the SCM system is through the file system. Policy Studio reads and writes projects from and to the file system, and policy developers check projects in and out of the SCM system using the SCM tools. This approach provides the maximum flexibility and support for many SCM solutions.

To work on an API, the policy developer checks out the API project and any dependent project from the SCM, edits the resources in the API project, deploys the API and dependent project on the API Gateway to test, saves the API project to the file system, and checks the updated API project back into the SCM.

Continuous Integration: build and test

Continuous Integration (CI) techniques are used to automate the building and testing of API Gateway configuration from all projects in the SCM. The entire build and test process can be automated with CI tools and triggered by a developer checking in an updated project. This approach ensures that API Gateway configuration is always deployable.



You can use the API Gateway policy package (`.pol`) and environment package (`.env`) mechanisms to promote and deploy API Gateway configuration in upstream environments. For more details, see *Introduction to API Gateway deployment and promotion* on page 12.

API Gateway provides a build script (`projpack`) to build the unified API Gateway configuration by merging all the projects in the SCM to create the policy package. API Gateway also provides a deployment script (`projdeploy`) to deploy the policy and environment package to the API Gateway. For more details, see *Automate processes for continuous integration* on page 58.

The CI process can be automated using DevOps tools such as Jenkins. For example, when an updated API project is checked into the SCM, a job is kicked off that builds the policy package, deploys the policy and environment packages to the test server, and runs a suite of tests.

Policy developers can use the API Gateway environmentalization features to enable customization of deployment artifacts for specific environments. The customization of environment configuration is performed using Configuration Studio. If a policy developer has added new environmentalized properties as part of changes they have made, the environment package for the CI environment must be updated before deploying and testing. For more details, see *API Gateway deployment and promotion tasks* on page 18.

Developer collaboration and resolving conflicts

This section explains how to resolve conflicts between and within API Gateway projects.

Inter-project collaboration and conflicts

The team development approach enables multiple policy developers to collaborate on an API Gateway configuration that has multiple APIs. Each API is implemented as a different API Gateway project, and only one policy developer works on an API project at any one time. Multiple policy developers can work on different APIs in parallel without impacting each other.

Occasionally, there might be a conflict between projects due to a clash of resource naming, and this will be detected by the build script (`projpack`). If two policy developers working on different API projects create a resource of the same type with the same name in their respective projects, a conflict will occur.

For example, two policy developers, Joe and Fred, are developing two different API projects, and each creates a Database Connection called **TestDBConnection**. This creates a conflict when building the API Gateway configuration from the two projects. The `projpack` build script will attempt to merge the configuration from both projects, detect the conflict, and fail. If both DB Connections are intended to refer to different physical connections, you can resolve the conflict by keeping the DB Connections in both API projects, but adopting different names. If both DB Connections refer to the same physical connection, you can resolve the conflict by creating a single DB Connection in a common project referenced by both API projects.

Note The `projpack` script detects most conflicts between projects, however, it cannot detect conflicts between projects containing different REST APIs with the same name. For example, if `projectA` and `projectB` both have a REST API named `TestAPI`, and this REST API contains `method1` and `method2` in `projectA`, and in `projectB` it contains `method3` and `method4`, `projpack` succeeds in merging the projects with no conflicts, but the resulting `.fed` file is corrupt. You must ensure that you do not have two different REST APIs with the same name in different projects.

For more details on `projpack`, see *Generate configuration packages from API Gateway projects* on page 58.

Intra-project collaboration and conflicts

Occasionally, multiple developers might need to make changes to the same API Gateway project in parallel. This will most likely occur when different policy developers are working on different API projects, but need to make changes to a common or shared dependent project.

For example, our two policy developers, Joe and Fred, both check out dependent project v1 to make changes unaware that the other has done the same. Joe makes his changes first and checks the dependent project back in as v2. Fred subsequently makes his changes, and then tries to check the dependent project back in as v2. The SCM system will detect this as a conflict and prevent Fred from checking the dependent project back in as v2.

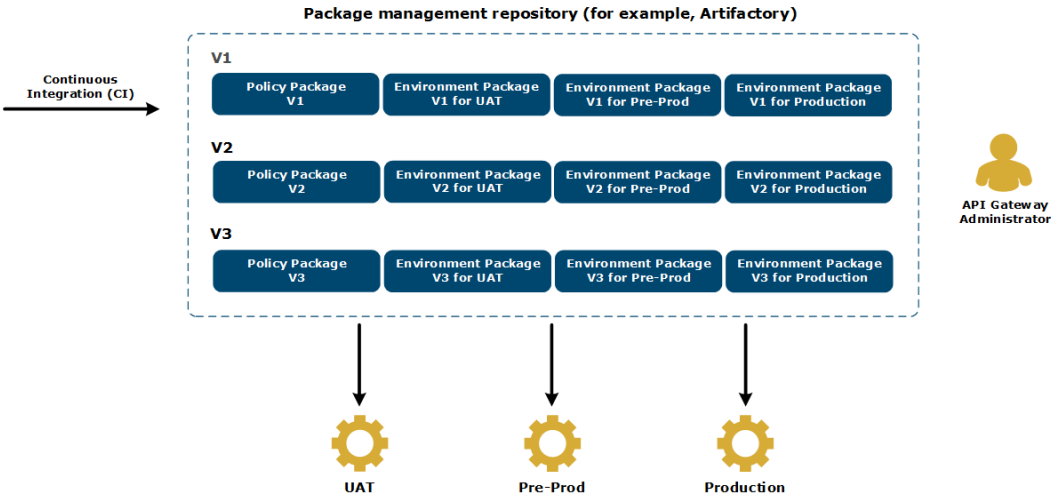
Fred must now resolve this conflict by merging his changes with Joe's, and checking the dependent project back in as v3. To perform this, Fred will check out Joe's v2 from the SCM, and then use Policy Studio to compare and merge the two project versions to create the v3. Policy Studio enables you to compare two configurations to identify the differences between them. For more details on compare and merge tools, see the *API Gateway Policy Developer Guide*.

Continuous Delivery: promote and deploy

Similar to how a Source Code Management (SCM) system is required for managing the projects in the development environment, a package management repository is recommended for managing the policy and environment packages in upstream environments.

When stable, this version of the policy package can be promoted and deployed to upstream environments. The API Gateway administrator can configure environment packages (`.env`) for each upstream environment for this version of the policy package (`.pol`). For example, if the upstream environments consist of test, pre-production, and production, for each version of the policy package, the equivalent environment package for the test, pre-production, and production environments should be created. For more details, see *Introduction to API Gateway deployment and promotion* on page 12.

The policy and environment packages are then stored in the package management repository. To deploy a specific version of the policy package to a specific environment, the correct versions of the policy package and environment package for that environment can be read from the package management repository and deployed (for example, deploy V2 to pre-production). Deployment of the policy and environment packages from the package management repository can be automated using the deployment script (`projdeploy`). For more details on `projdeploy`, see *Build and deploy API Gateway configurations* on page 62.



Team development best practices

8

This topic describes recommended best practices for creating API Gateway team development projects. This includes creating a common project and API projects, adding the projects to an SCM system, and setting up project dependencies.

For general details on creating projects, see the *API Gateway Policy Developer Guide*.

When to use team development

You should use team development if you have a large team of policy developers who need to work in parallel developing APIs and policies to be deployed as a single large API Gateway configuration.

When using team development you must:

- Use a Source Code Management (SCM) system
- Enable team development project templates and dependencies in Policy Studio
- Follow the workflow detailed in *Policy developer workflow* on page 45

The team development workflow requires you to split projects into separate pieces of configuration so that teams can work on them independently. If you do not have a requirement for multiple policy developers to work in parallel, you should not use a team development workflow. Instead you should use the previous workflows in Policy Studio where policy developers typically connect to a running API Gateway instance, download the configuration, make desired changes, and then deploy the configuration back to the API Gateway. This approach encourages use of API Gateway as a design-time repository for policies, in addition to as the runtime for executing policies. This workflow still requires you to create a project in Policy Studio, but there is no requirement to split the project into separate pieces of configuration, as in the team development workflow.

Increase Policy Studio memory

If you have a large API Gateway configuration, you might need to increase the memory allocation in Policy Studio. You can do this by increasing the value of the `Xmx` setting in the following file:

```
INSTALL_DIR/policystudio/policystudio.ini
```

The default value is `-Xmx768m`.

Create projects and dependencies for new or existing deployments

This section describes recommended best practices for creating API Gateway team development projects and project dependencies in new or existing API Gateway deployments.

Upgrade existing deployment to API Gateway version 7.6.2

If you have an existing deployment from an earlier version of API Gateway, you must upgrade your API Gateway installation to version 7.6.2. Additionally, after you upgrade your API Gateway installation, you must upgrade the configuration in your development environment to version 7.6.2. For more information, see the *API Gateway Upgrade Guide*.

Remove redundant configuration in existing deployment

If you have an existing deployment, you should reduce the size of the API Gateway configuration in Policy Studio by deleting redundant, duplicate, or older versions of the following API Gateway configuration:

- Web services
- Policies
- WSDL document bundles
- Listener paths

Enable team development in Policy Studio

To enable team development project templates and project dependencies in API Gateway, you must turn on the team development option in Policy Studio preferences. In Policy Studio select **Window > Preferences > Team Development** and select **Enable Team Development**.

Note An existing team development-enabled project is automatically detected by Policy Studio, and you do not need to enable team development in this case.

Break up existing configuration into a common project and API projects

If you have an existing deployment, you must break up your existing monolithic API Gateway configuration as follows:

- One common project (containing global configuration that applies across all other projects)
- Multiple projects per API

The following sections describe how you can do this.

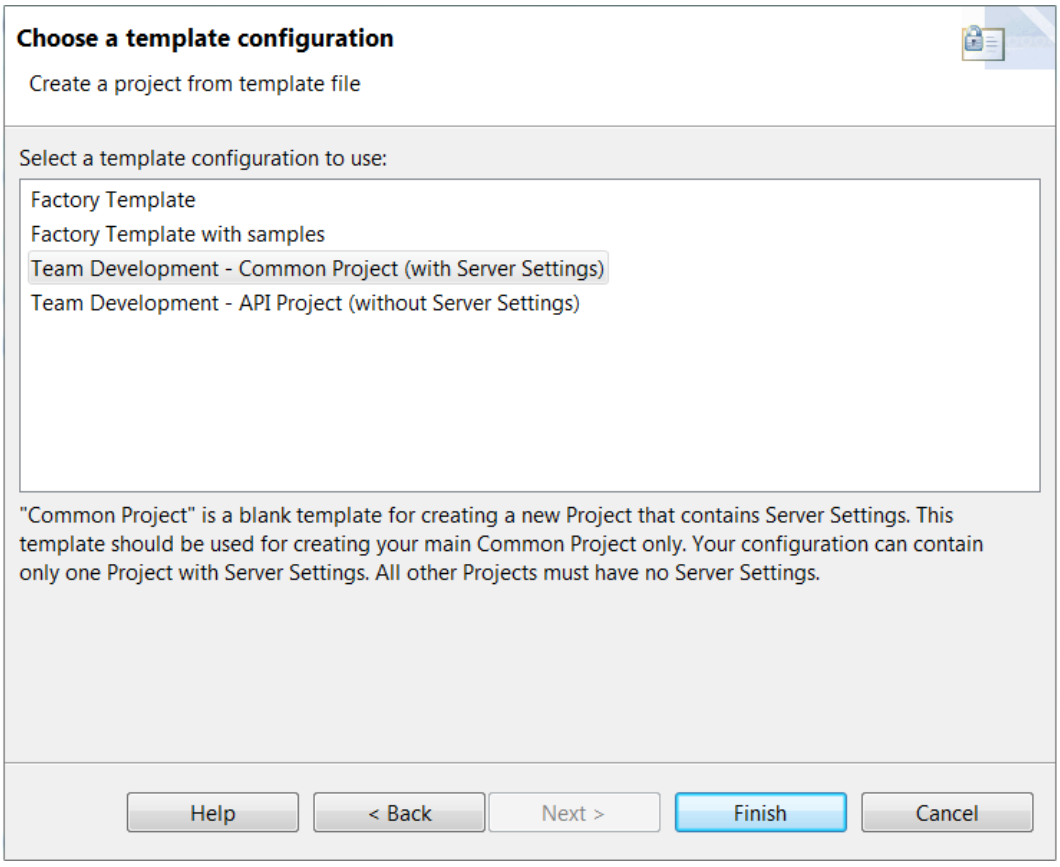
Create a common project

Create a common project that contains shared resources used by multiple APIs. For example, your common project might contain a corporate security policy that must be applied to all APIs. Your configuration can contain only one common project with server settings.

To create a common project, perform the following steps:

1. If you have an existing configuration, open the configuration in Policy Studio as a project, and use the export feature to create configuration fragments for a separate common project with common configuration. For example, right-click the relevant policy in the Policy Studio tree, and select **Export Policy**.

   For more details on exporting configuration, see the *API Gateway Policy Developer Guide*.
2. Create a new common project in Policy Studio. Select **New Project > From a template configuration > Team Development - Common Project (with Server Settings)**.

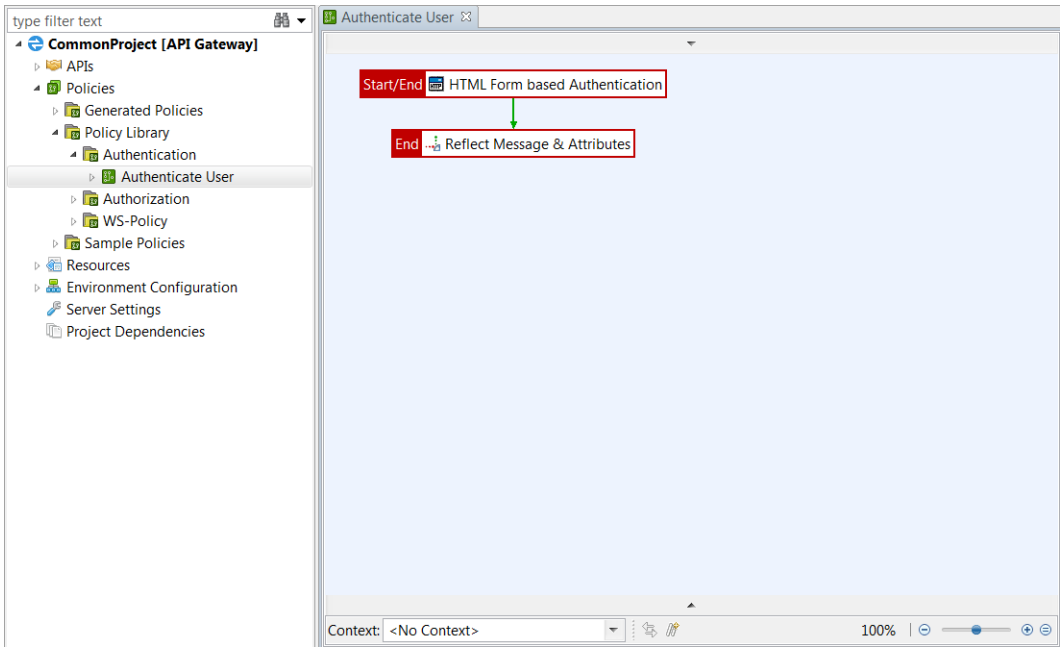


3. In this common project, create any configuration that you wish to reuse. For example:
   - All **Authentication** and **Authorization** policies should be in the common project and the other API projects should call out using **Policy Shortcut** filters to these policies.
   - Other common configuration includes global policies such as **Throttling**, XML/JSON schema validation, and XML to JSON transformation, or global settings such as **Alerts**.

   Note **Listeners** are not common configuration due to the link between **Paths** and **Ports**.

4. To import common configuration that you exported previously, select **Import Configuration Fragment**. You can put any configuration that you wish to reuse in the common project.

   The following example shows a common project containing Authentication and Authorization policies that were exported from an existing configuration and imported into the common project.

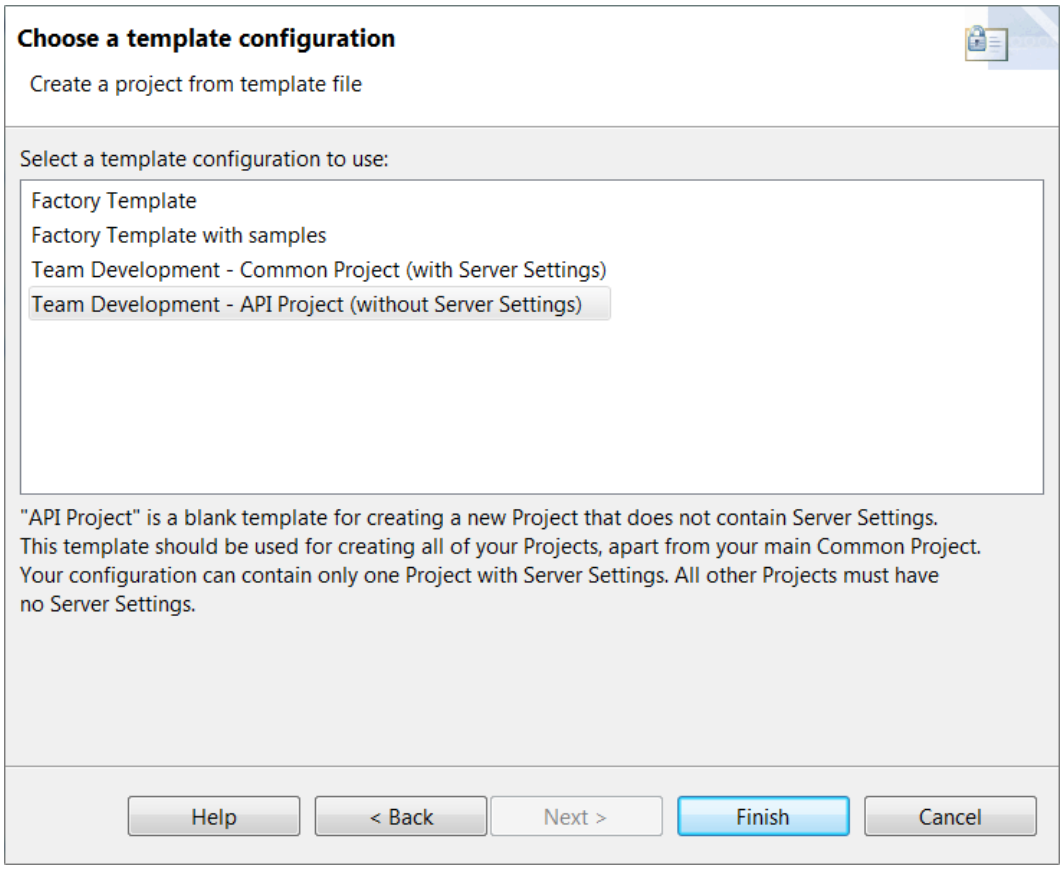


Create separate API projects

Create API projects that contain resources that are specific to an API and not shared with other APIs. Your configuration can contain multiple API projects.

To create API projects, perform the following steps:

1. If you have an existing configuration, use the export feature in Policy Studio to create configuration fragments for the separate API projects with API configuration only. For example, right-click the relevant policy in the Policy Studio tree, and select **Export Policy**.
2. To create an API project for each separate API or group of APIs that need to be managed together, select **File** > **New Project** > **From a template configuration** > **Team Development - API Project (without Server Settings)**.



3. In this API project, create the configuration for each separate API or group of APIs that need to be managed together.
4. To import API configuration that you exported previously, select **Import Configuration Fragment**.

Add projects to SCM

You now have one common project, and separate API projects for each API, all stored locally. You must add your API Gateway project files to your SCM system before you start adding project dependencies in Policy Studio.

For example, using Git, an administrator creates a master repository for API projects. Policy developers can clone this master repository to their own local machine. They then create projects and commit to their local repository. When appropriate, they can push to the master repository. They must always perform a pull before a push request to ensure they get the latest from master and resolve any conflicts.

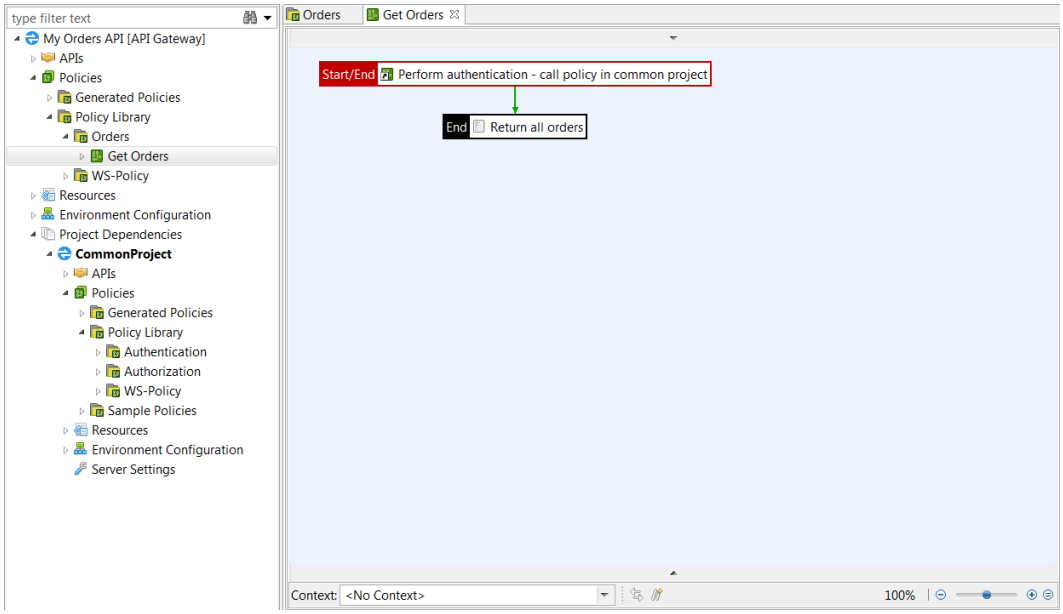
Create project dependencies

To create project dependencies between a common project and an API project, perform the following steps:

1. Open one of the API projects in Policy Studio.
2. To create a project dependency between this API project and the common project, right-click **Project Dependencies** in the Policy Studio tree, select **Manage Project Dependencies**, and click **Add**. For more details, see *Manage API Gateway project dependencies* on page 55.

   Note The common project configuration displayed in the **Project Dependencies** section of the API project are read-only.

3. Use a common configuration item (for example, a policy) from the common project in the API project (for example, using a policy shortcut).



4. Push the changes made to the API project to your SCM. For example, you are using Git and have changed an API project named Orders API in `/home/jdoe/apiprojects/ordersapi/`. If you changed the message in a **Set Message** filter from `OK` to `Not OK`, typically, the `PrimaryStore.xml` shows as modified when you run the `git status` command.

Manage API Gateway project dependencies 9

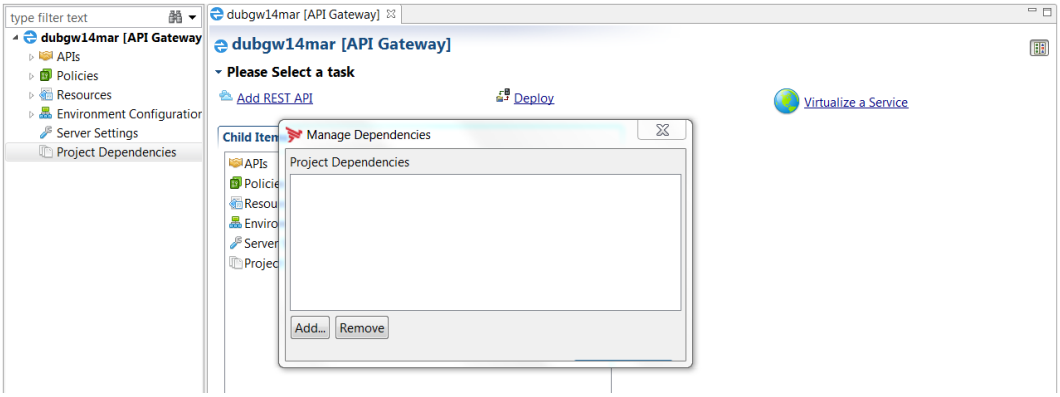
API Gateway deployments can have common resources used by multiple APIs. For example, this includes corporate security policies to be applied to all APIs. You can package such common resources in a *dependent project* used by multiple APIs. This topic describes how to manage dependencies between API Gateway projects.

For details on how to create API Gateway projects, see the *API Gateway Policy Developer Guide*.

Note To enable team development project templates and project dependencies in API Gateway, you must turn on the team development option in Policy Studio preferences. In Policy Studio select **Window > Preferences > Team Development** and select **Enable Team Development**.

Manage project dependencies

To manage dependencies between API Gateway projects in Policy Studio, select **File** > **Manage Dependencies** from the main menu. Alternatively, right click **Project Dependencies** in the navigation tree on the left. This opens the **Manage Dependencies** dialog where you can add and remove project dependencies:



Add project dependencies

In the **Manage Dependencies** dialog, click **Add** to add a new project as a dependent project of the current project. This enables you to use the common resources (policies, resources, external connections, and so on) of the dependent project in your current project. A dialog enables you to

browse to the directory of the project to add as a project dependency (for example, `C:\Users\jbloggs\apiprojects\MyTestAPIGateway`). If there are no conflicts between the projects, the dependent project is merged with the current project.

Note The encryption passphrase of the dependent project must be the same as the passphrase of the current project. If it is not, an error message is displayed and the dependency is not added.

Resolve project conflicts

If Policy Studio finds conflicts between the projects, this displays the **Project Dependencies Conflicts** dialog showing details of the conflicts. Policy Studio will not allow you to add the project as a dependency until the conflicts are resolved.

To resolve conflicts, open the project that you want to add as a dependency, and remove or resolve the conflicts.

The newly added project dependency is added to the navigation tree on the left. You can navigate the tree but its nodes are read-only and are only for informational purposes. There are no right-click options available and when you click on a node it does not open up an associated right hand pane (with the exception of **Show All References**).

Remove project dependencies

In the **Manage Dependencies** dialog, click **Remove** to remove a project dependency. If there are references to the project dependency in your current project, you cannot remove the project dependency until all references are removed. In this case, a warning message displays. When you click **OK**, the **Show All References** panel displays, and the entities that reference the dependent project are shown.

Identify project dependencies

When developing configuration in the current project, you can refer to and reuse entities from dependent projects. For example, a policy shortcut defined in a policy in the current project can refer to a common policy from a dependent project.

To help to distinguish entities from the current project and from a dependent project, reusable entities are displayed in an *italic* font. The name of the project to which a reusable entity belongs to is added as a suffix in square brackets (`[]`) to the entity name.

Show broken references

When you open a project or import a configuration fragment, Policy Studio checks for broken references in the configuration. This is especially important when dealing with project dependencies when there are references in one project to another project. If there are any, Policy Studio displays the list of broken references on the right. Each item in the list displays a name and a link to the broken reference. You can click the link and go to the policy with the broken reference to fix it. You can also access this feature by selecting **Window > Show View > Broken References View** in the main menu.

Automate processes for continuous integration 10

The API Gateway package and deploy tools enable you to automate processes for continuous integration. You can use the tools to:

- *Generate configuration packages from API Gateway projects* on page 58
- *Build and deploy API Gateway configurations* on page 62
- *Change the passphrase of an API Gateway project* on page 65
- *Upgrade an API Gateway project* on page 67

For details on installing the package and deploy tools, see the *API Gateway Installation Guide*. For details on using API Gateway projects, see the *API Gateway Policy Developer Guide*.

Run the package and deploy tools

You can run the package and deploy tools from the following directory:

```
INSTALL_DIR/apigateway/posix/bin
```

For example, to run the `projpack --help` command:

```
> cd /opt/Axway-7.6.2/apigateway/posix/bin
> projpack --help
```

Generate configuration packages from API Gateway projects

The `projpack` tool enables you to use automatic processes to generate API Gateway configuration packages from multiple API Gateway projects. For example, you can automatically generate deployment packages (`.fed`), policy packages (`.pol`), and environment packages (`.env`), which you can then use to promote APIs and policy configuration to upstream environments.

For more details on configuration packages and upstream environments, see *Introduction to API Gateway deployment and promotion* on page 12.

projpack command options

The `projpack` command options are described as follows:

Option	Description
`--help, -h`	Display help message and exit.
`--create, -c`	Create a new configuration package (`.pol`, `.env`, or `.fed`) from an API Gateway project.
`--import, -i`	Import project configuration into an existing configuration package (`.pol`, `.env`, or `.fed`).
`--name, -n`	Name of the configuration package. Possible values are the file name, and the directory path and file name for `--import`. When used with `--create` this is the name of the configuration package to be created (for example, `myarchive`). When used with `--import` this is the name of the configuration package to import into (for example, `/temp/dev.pol`). This option is required.

`--add, -a`	List of projects to merge into the configuration package. You must specify a passphrase for each list of projects with the `--projpass` option (or use `--projpass-none`). Use a space to separate multiple projects.
	For example, to specify two projects with the same passphrase and one project with a different passphrase:
	`--add /home/user1/apiprojects/proj1 /home/user1/apiprojects/proj2 --projpass=testa`
	`--add /home/user1/apiprojects/proj3 --projpass=testb`
	For example, to specify two projects with the same passphrase and two projects with no passphrase:
	`--add /home/user1/apiprojects/proj1 /home/user1/apiprojects/proj2 --projpass=testa`
	`--add /home/user1/apiprojects/proj3 /home/user1/apiprojects/proj4 --projpass-none`
	Note If you are using a passphrase file to specify the passphrases you must specify each project individually to the `--add` option.
	For example, to specify four projects with all passphrases in a passphrase file:
	`--add /home/user1/apiprojects/proj1 -add /home/user1/apiprojects/proj2 --add /home/user1/apiprojects/proj3`
	`--add /home/user1/apiprojects/proj4 --passfile=/home/user1/passfile.txt`

`--replace, -r`	Replace conflicts when `--add` is specified. You must specify `--replace` before `--add`. If there is a conflict, and `--replace` is not specified, `projpack` exits.

Option	Description
`--printDiffs`	Print a tree with the changes for every project as it is added to the package.
`--passfile, -f`	File that contains the project and target configuration package passphrases. You can use this option instead of the `--passphrase` (or `--passphrase-none`) and `--projpass` (or `--projpass-none`) options. For an example, see *Example projpack use cases* on page 60. For more details on the file format, see *Read passphrases from a file* on page 62.
`--projpass`	Passphrase used to encrypt projects specified by `--add` option. This can be any text. This option is required for projects specified by `--add`, or you can use `--projpass-none` to specify a zero-length passphrase.
`--passphrase`	Passphrase used to encrypt the generated target configuration package. This can be any text. This option is required, or you can use `--passphrase-none` to set a zero-length passphrase.
`--dir, -d`	Full output directory path to the generated target configuration package. Defaults to the current directory. You can also use this option with `--import` to specify the full directory path of the configuration package to import into.
`--type, -t`	The generated configuration package type (`pol`, `env`, or `fed`). A `pol` and `env` file are generated by default. You can also use this option with `--import` to specify the type of the configuration package to import into.
`--polprop`	Policy metadata added to the policy manifest for the generated target configuration package (for example, `--polprop=createdBy:joebloggs`). For more details, see *Configure package properties* on page 24.
`--envprop`	Environment metadata added to the environment manifest for the generated target configuration package (for example, `--envprop="envCreationTime:\$(date +'%c')"`). For more details, see *Configure package properties* on page 24.
`--tracelevel`	Trace level for the generated trace file. Supported trace levels are `FATAL`, `ALWAYS`, `ERROR`, `INFO`, `MIN`, `DEBUG`, `VERBOSE`. The default is `INFO`.
`--tracedir`	Directory to write trace files to. The default is none.

Example projpack use cases

The following examples describe how you might use `projpack`.

Create encrypted policy and environment packages from a single project

The following example shows how to create an encrypted policy package (`.pol`) and environment package (`.env`) from a specified API Gateway project:

```
projpack --create --passphrase=my_text --name=my_package --add
/home/user1/apiprojects/proj1 --projpass=my_text
```

The `--passphrase` and `--projpass` options are required, or you can use `--passphrase-none` and `--projpass-none`.

Create an encrypted deployment package from multiple projects

The following example shows how to create a deployment package (`.fed`) from multiple API Gateway projects:

```
projpack --create --dir=/home/jbloggs/testfeds/ --passphrase=my_text --name=dev.fed
--type=fed
--add /home/jbloggs/apiprojects/proj1 /home/jbloggs/apiprojects/proj2 --projpass-
none
```

Create an encrypted deployment package from multiple projects using a passphrase file

The following example shows how to create a deployment package (`.fed`) from multiple API Gateway projects, where the passphrases are supplied in a passphrase file:

```
projpack --create --dir=/home/jbloggs/testfeds/ --name=dev.fed --type=fed
--add /home/jbloggs/apiprojects/proj1 -add /home/jbloggs/apiprojects/proj2 --
passfile=/home/jbloggs/passfile.txt
```

For more details on the passphrase file format, see *Read passphrases from a file* on page 62.

Import project configuration into an existing deployment archive

The following example shows how to import configuration from specified API Gateway projects into a specified deployment package:

```
projpack --import --replace --dir=/home/user1/testfeds/ --passphrase=my_text --
```

```
name=my_package --type=fed
--add /home/user1/apiprojects/proj1 --projpass=my_text1 --add
/home/user1/apiprojects/proj2 --projpass=my_text2
```

Note You can use the `--replace` option before `--add` to override conflicts, otherwise if conflicts are found the command exits.

For more examples, see `projpack --help`.

Read passphrases from a file

The `projpack` command provides the`--passfile` or `-f` option to specify the location of a passphrases file. This file contains a passphrase for the generated target configuration package and passphrases for the projects to be packaged.

The section names in the passphrase file are predefined. The names of the keys in a section are arbitrary, but must be unique in that section. For example:

```
[target]
pp=my_text

[projects]
p1=my_text1
p2=my_text2
```

Note For security reasons, this file should be protected with appropriate permissions. For example, the following command changes the file ownership:

```
sudo chown root: <passphrases_file>
```

The following command specifies file permissions:

```
sudo chmod 400 <passphrases_file>
```

Build and deploy API Gateway configurations

Environment settings are subject to change during the development, test, and production phases. The `projdeploy` tool enables you to use automated processes to build API Gateway configurations, to apply or modify the environment settings of a specified configuration package (`.fed` or `.pol`), and to deploy the package to specified API Gateway group instances.

projdeploy command options

The `projdeploy` command options are described as follows:

Option	Description
`--help, -h`	Display help message and exit.
`--dir, -d`	Directory where the configuration package is located and modified. Defaults to the current directory.
`--name, -n`	Name of configuration package to deploy (for example, for `dev.fed` or `dev.pol` file, specify `dev`). This option is required.
`--type, -t`	Type of configuration package to deploy (for example, `pol` or `fed`). Defaults to `pol`.
`--apply-env, -e`	Name of environment package (`.env`) to apply to the configuration package being deployed (for example, `/temp/dev.env`).
`--passphrase`	Encryption passphrase for the source configuration. This can be any text. This option is required, or you can use `--passphrase-none` to specify a zero-length passphrase.
`--change-pass-to`	Encryption passphrase for the target configuration. This can be any text. You can also use `--change-pass-to-none` to set a zero-length passphrase.
`--polprop`	Policy metadata added to the policy manifest for the configuration package (for example, `--polprop=createdBy:joebloggs`). For more details, see *Configure package properties* on page 24.
`--envprop`	Environment metadata added to the environment manifest for the configuration package (for example, `--envprop="envCreationTime:\$(date +'%c')"`). For more details, see *Configure package properties* on page 24.
`--deploy-to`	Deploys a specified configuration package to a specified Admin Node Manager server.
`--host-name`	Use with `--deploy-to` option to specify Admin Node Manager host name to deploy configuration to.
`--port`	Use with `--deploy-to` option to specify Admin Node Manager port number to connect to.
`--user-name`	Use with `--deploy-to` option to specify user name for authentication.

Option	Description
`--password`	Use with `--deploy-to` option to specify user password for authentication.
`--truststore-file`	File name for the SSL certificate truststore that holds all trusted SSL certificates. Defaults to trusting all.
`--truststore-password`	Truststore file password that holds all trusted SSL certificates.
`--group, -g`	Use with `--deploy-to` option to specify API Gateway group name to deploy configuration to.
`--includes, -i`	Use with `--deploy-to` option to specify a list of names of API Gateway instances to deploy to. Alternatively, you can specify a list of API Gateways to exclude with the `--excludes` option.
`--excludes, -x`	Use with `--deploy-to` option to specify a list of names of API Gateway instances not to deploy to. Mutually exclusive with the `--includes` option.
`--tracelevel`	Trace level for the generated trace file. Supported trace levels are `FATAL`, `ALWAYS`, `ERROR`, `INFO`, `MIN`, `DEBUG`, `VERBOSE`. The default is `INFO`.
`--tracedir`	Directory to write trace files to. The default is none.

Example projdeploy use cases

The following examples describe how you might use `projdeploy`.

Deploy a deployment package on a local host

The following simple example shows how to deploy a specified deployment package (`.fed`) to a locally running API Gateway instance:

```
projdeploy --passphrase-none --name=test.fed
```

Deploy a deployment package with new environment settings and passphrase

The following example shows how to deploy a specified deployment package and apply a new passphrase and environment settings on a local host:

```
projdeploy --dir=/tests --passphrase=pass --name=my_package --type=fed
--change-pass-to=newpass --apply-env=/tests/prod/prod.env
```

Deploy a policy package to a specified host and group

The following example shows how to deploy a specified policy package and apply new settings on a specified Admin Node Manager host and port, and API Gateway group:

```
projdeploy --dir=/tests --passphrase=pass --name=mypackage --type=pol
--change-pass-to-none --apply-env=/tests/prod/prod.env
--deploy-to --host-name=myhost --port=myport --user-name=myname --password==mypass -
-group-name=mygroup
```

Deploy a deployment package to specified instances

The following example shows how to deploy a specified deployment package and apply new settings on a specified Admin Node Manager host, port, and set of API Gateway instances:

```
projdeploy --dir=/tests --passphrase=pass --name=mypackage --type=fed
--deploy-to --host-name=myhost --port=myport --user-name=myname --password==mypass -
-group-name=mygroup
--includes mygateway1 mygateway2 mygateway3
```

Change the passphrase of an API Gateway project

The `projchangepass` tool enables you to change the passphrase of an API Gateway project.

projchangepass command options

The `projchangepass` command options are described as follows:

Option	Description
`--help, -h`	Display help message and exit.
`--proj`	Directory where the project is located.
`--oldpass`	The old project passphrase.

Option	Description
`--newpass`	The new project passphrase.
`--confirmpass`	Confirm the new project passphrase.
`--passfile`	File that contains the project passphrases. For a sample file, see *Example projchangepass use cases* on page 66.
`--tracelevel`	Trace level for the generated trace file. Supported trace levels are `FATAL`, `ALWAYS`, `ERROR`, `INFO`, `MIN`, `DEBUG`, `VERBOSE`. The default is `INFO`.
`--tracedir`	Directory to write trace files to. The default is none.

Example projchangepass use cases

The following examples describe how you might use `projchangepass`.

Change the passphrase of a project

This example shows how to change the project passphrase on proj1 from `changeme` to `newpassPhrase`:

```
projchangepass --proj=/home/user1/apiprojects/proj1 --oldpass=changeme --
newpass=newpassPhrase --confirmpass=newpassPhrase
```

Change the passphrase of a project using a passphrase file

This example shows how to change the project passphrase on proj1 using the contents in `passfile.txt`:

```
projchangepass --proj=/home/user1/apiprojects/proj1 --
passfile=/home/user1/passfile.txt
```

Example `passfile.txt`:

```
[currentProj]
oldPP="changeme"
newPP=newpassPhrase
confirmPP=newpassPhrase
```

Upgrade an API Gateway project

The `projupgrade` tool enables you to upgrade one or more API Gateway projects from earlier versions (7.5.1 or later) to version 7.6.2. For example, after using `sysupgrade` to upgrade your API Gateway installation to version 7.6.2, you must upgrade the configuration projects in your development environment, or you cannot deploy them to your upgraded API Gateways.

You can use the `projupgrade` tool to upgrade a number of configuration projects at once. These projects might be independent of one another, or could include shared projects and their dependencies.

Note `projupgrade` upgrades all of the projects in a particular directory. If you keep projects in different source directories, remember to run `projupgrade` on all of them.

After using `projupgrade` to upgrade your projects, you can:

- Open these projects in Policy Studio 7.6.2 and continue working as before.
- Deploy them to an API Gateway 7.6.2 instance.
- Merge the projects to create configuration packages (`.fed` or `.pol`/`.env`) using `projpack`. For more information, see *Generate configuration packages from API Gateway projects* on page 58.

For more details on upgrading your API Gateway installation, see the *API Gateway Upgrade Guide*.

projupgrade command options

The `projupgrade` command options are described as follows:

Option	Description
`--help, -h`	Display help message and exit.
`--projdir`	Directory containing the projects to be upgraded (for example, `/home/user1/apiprojects`). This option is required.
`--backupdir`	Directory into which projects are copied before upgrade begins. This directory is created if it does not exist. If this option is not specified a backup is not created.
`--projpass`	Encryption passphrase for the projects to be upgraded. You can use the `--passfile` option to override this passphrase for individual projects. If passphrases are not specified using this option or `--passfile`, the command uses a zero-length string as the passphrase.
`--passfile`	File that contains the project passphrases. For a sample file, see *Example projupgrade use cases* on page 68.

Option	Description
`--tracelevel`	Trace level for the generated trace file. Supported trace levels are `FATAL`, `ALWAYS`, `ERROR`, `INFO`, `MIN`, `DEBUG`, `VERBOSE`. The default is `INFO`.
`--tracedir`	Directory to write trace files to. This directory is created if it does not exist. The default is the current directory.

Example projupgrade use cases

The following examples describe how you might use `projupgrade`.

Back up and upgrade projects

This example shows how to upgrade the projects contained in a specified directory:

```
projupgrade --projdir=/home/user1/apiprojects --backupdir=/home/user1/backup
```

Before the upgrade the projects are backed up to the specified backup directory. As no passphrases are specified the command uses a zero-length string as the passphrase for all projects.

Upgrade projects with different passphrases using a passphrase file

This example shows how to upgrade the projects contained in a specified directory, and specify the passphrases with the `--projpass` and `--passfile` options:

```
projupgrade --projdir=/home/user1/apiprojects --projpass=mypass --
passfile=/tmp/passfile.txt
```

Example `passfile.txt`:

```
[projects]
Project 1=
Project 2=My passphrase!
```

The passphrase `mypass` is used for any projects not listed in `passfile.txt`, while the passphrases specified in the file are used for the projects listed in `passfile.txt`.

Note `projupgrade` accepts the same passphrase file format as `projpack`, but only the section shown in the example above is actually used.

projupgrade output

The `projupgrade` command produces the following outputs:

- `projupgrade.log` – A log of any errors or warnings generated by the `projupgrade` command.
- `projupgrade.report` – A summary report of the projects upgraded.

For example, the following command is run on a project directory containing two projects:

```
projupgrade --projdir=/home/user1/apiprojects --backupdir=/tmp --tracedir=/tmp
```

The following is an example of the `projupgrade.report`:

```
# ProductName=projupgrade 7.5.3-2017-01-17 rev. 4712b77 (Linux.x86_64)
# CurrentDate=Mon, 23 Jan 2017 15:59:49 +0000
# CurrentDateUTC=1485187189
# TZ=GMT

REPORT 23/Jan/2017:15:59:49.876 ... Tracing to directory: /tmp/projupgrade_logs_
2017-01-23_15-59-49 filename: projupgrade.report
REPORT 23/Jan/2017:15:59:49.876 ... ====================
REPORT 23/Jan/2017:15:59:49.876 ... Running projupgrade
REPORT 23/Jan/2017:15:59:49.876 ... ====================
REPORT 23/Jan/2017:15:59:49.876 ... Projects directory : /home/user1/apiprojects
REPORT 23/Jan/2017:15:59:49.876 ... Backup directory : /tmp/projupgrade_backup_2017-
01-23_15-59-49
REPORT 23/Jan/2017:15:59:49.876 ... Logging directory : /tmp/projupgrade_logs_2017-
01-23_15-59-49
REPORT 23/Jan/2017:15:59:49.876 ...
REPORT 23/Jan/2017:15:59:49.876 ... Skipping non-project directory:
/home/user1/apiprojects
REPORT 23/Jan/2017:15:59:49.877 ...
REPORT 23/Jan/2017:15:59:49.882 ... Upgrading project 'Project1':
/home/user1/apiprojects/Project1 ...
REPORT 23/Jan/2017:15:59:49.885 ... Upgrading
'federated:file:////home/user1/apiprojects/Project1/configs.xml'
and putting output into '/home/user1/dev/temp/tmpB3F1P5'...
REPORT 23/Jan/2017:15:59:50.726 ... Writing upgrade to
'federated:file:/home/user1/dev/temp/tmpB3F1P5/configs.xml', please wait...
REPORT 23/Jan/2017:15:59:53.650 ... Migrating TivoliSettings from version 0 to
version 1
REPORT 23/Jan/2017:15:59:53.651 ... Downgrade the cardinality on version field so we
can remove it later
REPORT 23/Jan/2017:15:59:59.641 ... ... Upgrade successful.
REPORT 23/Jan/2017:15:59:59.863 ... OK
REPORT 23/Jan/2017:15:59:59.864 ...
REPORT 23/Jan/2017:15:59:59.864 ... Upgrading project 'Project2':
```

```
/home/user1/apiprojects/Project2 ...
REPORT 23/Jan/2017:15:59:59.866 ... Upgrading
'federated:file:////home/user1/apiprojects/Project2/configs.xml'
and putting output into '/home/user1/dev/temp/tmp2Pd4Dz'...
REPORT 23/Jan/2017:16:00:00.005 ... Problem connecting to store:
REPORT 23/Jan/2017:16:00:00.005 ... Invalid group passphrase
ERROR 23/Jan/2017:16:00:00.005 ... FAILED: 1
REPORT 23/Jan/2017:16:00:00.005 ...
REPORT 23/Jan/2017:16:00:00.005 ... projupgrade summary
REPORT 23/Jan/2017:16:00:00.005 ... ==================
REPORT 23/Jan/2017:16:00:00.005 ... Number of projects successfully upgraded: 1
REPORT 23/Jan/2017:16:00:00.005 ... Number of projects where upgrade failed: 1
REPORT 23/Jan/2017:16:00:00.005 ...
REPORT 23/Jan/2017:16:00:00.005 ... Projects where upgrade failed:
REPORT 23/Jan/2017:16:00:00.005 ... /home/user1/apiprojects/Project2
REPORT 23/Jan/2017:16:00:00.005 ...
REPORT 23/Jan/2017:16:00:00.005 ... The contents of these projects have not been
changed.
REPORT 23/Jan/2017:16:00:00.005 ... Fix the problems identified in the logs and
rerun projupgrade.
REPORT 23/Jan/2017:16:00:00.005 ... Logging directory: /tmp/projupgrade_logs_2017-
01-23_15-59-49
```

The output shows that Project1 was upgraded successfully, but Project2 failed to upgrade because of an "`Invalid group passphrase`". To fix this, you could write the Project2 passphrase to a file and rerun `projupgrade` with the `--passfile` option.

Manage X.509 certificates and keys 11

The **Environment Configuration > Certificates and Keys** node in the Configuration Studio tree enables you to manage the X.509 certificates and keys trusted by API Gateway. These settings are environment-specific, and typically need to be configured during promotion to an upstream environment.

For API Gateway to trust X.509 certificates issued by a specific Certificate Authority (CA), you must import that CA's certificate into the API Gateway's trusted certificate store. For example, if API Gateway is to trust secure communications (SSL connections or XML Signature) from an external SAML Policy Decision Point (PDP), you must import the PDP certificate, or the issuing CA certificate into the API Gateway certificate store.

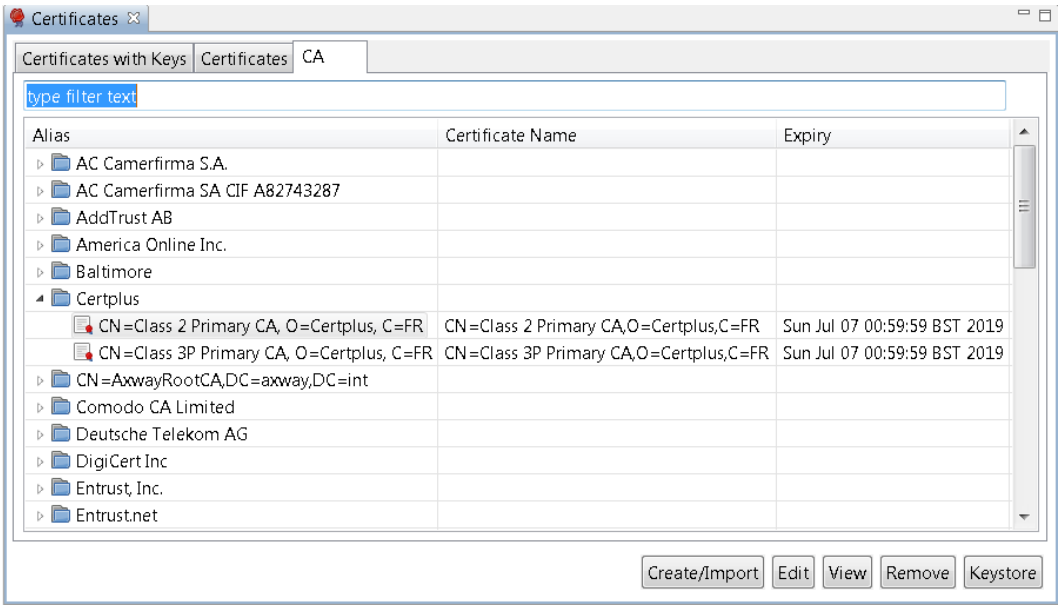
In addition to importing CA certificates, you can import and create server certificates and private keys in the certificate store. These can be stored locally or on an external Hardware Security Module (HSM). You can also import and create public-private key pairs. For example, these can be used with the Secure Shell (SSH) File Transfer Protocol (SFTP) or with Pretty Good Privacy (PGP).

View certificates and keys

To view the certificates and keys stored in the certificate store, select **Environment Configuration > Certificates and Keys > >Certificates** in the Configuration Studio tree. Certificates and keys are listed on the following tabs in the **Certificates** window:

- **Certificates with Keys**: Server certificates with associated private keys
- **Certificates**: Server certificates without any associated private keys
- **CA**: Certificate Authority certificates with associated public keys

You can search for a specific certificate or key by entering a search string in the text box at the top of each tab, which automatically filters the tree.



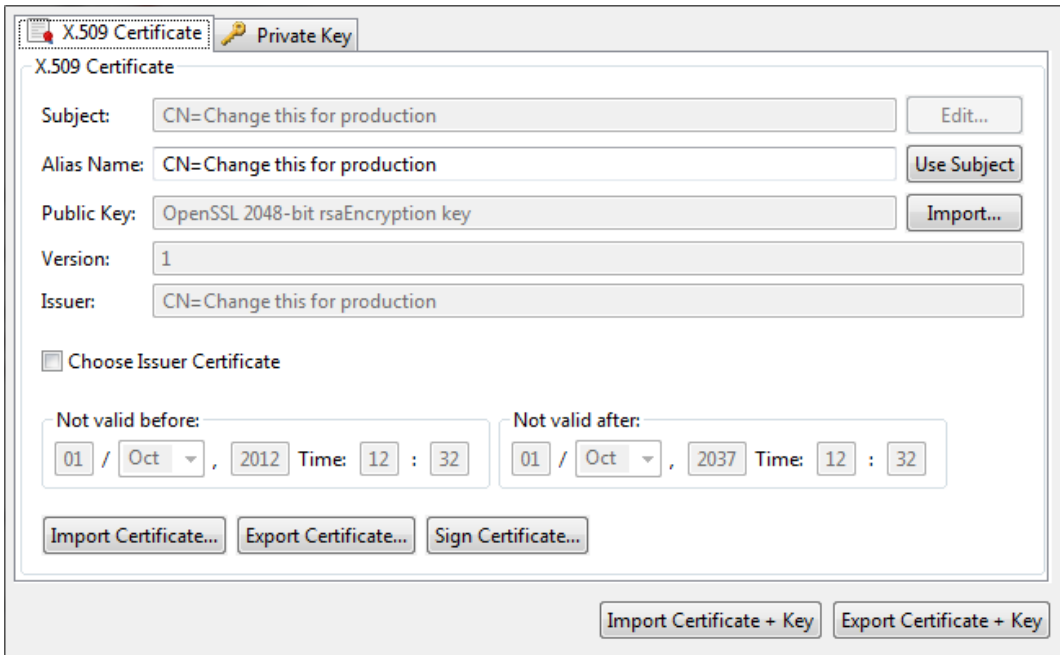
Certificate management options

The following options are available at the bottom right of the window:

- **Create/Import**: Click to create or import a new certificate and private key. For details, see *Configure an X.509 certificate* on page 72.
- **Edit**: Select a certificate, and click to edit its existing settings.
- **View**: Select a certificate, and click to view more detailed information.
- **Remove**: Select a certificate, and click to remove the certificate from the certificate store.
- **Keystore**: Click this to export or import certificates to or from a Java keystore. For details, see *Manage certificates in Java keystores* on page 83.

Configure an X.509 certificate

To create a certificate and private key, click **Create/Import**. The **Configure Certificate and Private Key** dialog is displayed. This section explains how to use the **X.509 Certificate** tab on this dialog.



Create a certificate

Configure the following settings to create a certificate:

- **Subject**:
  Click **Edit** to configure the *Distinguished Name* (DName) of the subject.
- **Alias Name**:
  This mandatory field enables you to specify a friendly name (or alias) for the certificate. Alternatively, you can click **Use Subject** to add the DName of the certificate in the text box instead of a certificate alias.
- **Public Key**:
  Click **Import** to import the subject's public key (usually from a PEM or DER-encoded file).
- **Version**:
  This read-only field displays the X.509 version of the certificate.
- **Issuer**:
  This read-only field displays the distinguished name of the CA that issued the certificate.
- **Choose Issuer Certificate**:
  Select to explicitly specify an issuer certificate for this certificate (for example, to avoid a potential clash or expiry issue with another certificate using the same intermediary certificate). You can then click the browse button on the right to select an issuer certificate. This setting is not selected by default.
- **Not valid before**:
  Select a date to define the start of the validity period of the certificate.
- **Not valid after**:
  Select a date to define the end of the validity period of the certificate.

- **Sign Certificate**:
  You must click this button to sign the certificate. The certificate can be self-signed, or signed by the private key belonging to a trusted CA whose key pair is stored in the certificate store.

Import certificates

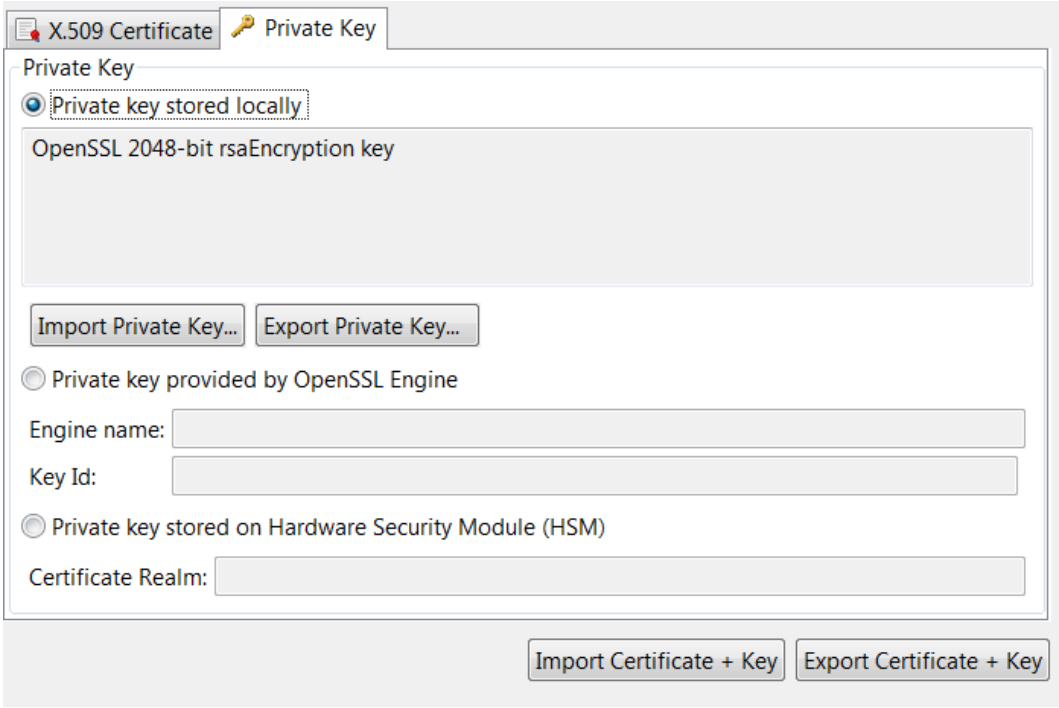
You can use the following buttons to import or export certificates into the certificate store:

- **Import Certificate**:
  Click to import a certificate (for example, from a `.pem` or `.der` file).
- **Export Certificate**:
  Click to export the certificate (for example, to a `.pem` or `.der` file).

Configure a private key

Use the **Private Key**tab to configure details of the private key. By default, private keys are stored locally (for example, in the API Gateway certificate store). They can also be provided by an OpenSSL engine, or stored on a Hardware Security Module (HSM) if required.

API Gateway supports PKCS#11-compatible HSM devices. For example, this includes Thales nShield Solo, SafeNet Luna SA, and so on.



Private key stored locally

If the private key is stored in the API Gateway certificate store, select **Private key stored locally**. The following options are available for keys stored locally:

- **Private key stored locally**:
  This read-only field displays details of the private key.
- **Import Private Key**:
  Click to import the subject's private key (usually from a PEM or DER-encoded file).
- **Export Private Key**:
  Click to export the subject's private key to a PEM or DER-encoded file.

Private key provided by OpenSSL engine

If the private key that corresponds to the public key in the certificate is provided by an OpenSSL engine, select **Private key provided by OpenSSL Engine**.

Configure the following fields to associate a key provided by the OpenSSL engine with the current certificate:

- **Engine name**:
  Enter the name of the OpenSSL engine to use to interface to an HSM. All vendor implementations of the OpenSSL Engine API are identified by a unique name. See your vendor's OpenSSL engine implementation or HSM documentation to find out the name of the engine.
- **Key Id**:
  Enter the key ID used to uniquely identify a specific private key from all others stored on an HSM. When you complete this dialog, the private key is associated with the certificate that you are currently editing. Private keys are identified by their key ID by default.

Private key stored on external HSM

If the private key that corresponds to the public key stored in the certificate resides on an external HSM, select **Private key stored on Hardware Security Module (HSM)**, and enter the name of the **Certificate Realm**.

Note To use the API Gateway's PKCS#11 engine to access objects in an external HSM, the corresponding HSM provider and certificate realms must also be configured. For more details, see *Configure HSMs and certificate realms* on page 75.

Configure HSMs and certificate realms

Certificate realms are abstractions of private keys and public key certificates, which mean that policy developers do not need to enter HSM-specific configuration such as slots and key labels. Instead, if a private key exists on an HSM, the developer can configure the certificate to show that its private

key uses a specific certificate realm, which is simply an alias for a private key (for example, `JMS Keys`).

For example, on the host machine, an administrator could configure the `JMS Keys` certificate realm, and create a *keystore*for the realm, which requires specific knowledge about the HSM (for example, PIN, slot, and private key label). The certificate realm is the alias name, while the keystore is the actual private keystore for the realm.

Manage HSMs with keystoreadmin

The `keystoreadmin` script enables you to perform the following tasks:

- Register an HSM provider
- List registered HSM providers
- Create a certificate realm
- List certificate realms

For example, if a policy developer is using JMS, and wants to indicate that private keys exist on an HSM, they could indicate that the certificate is using the `JMS Keys` certificate realm. On each instance using the configuration, it is the responsibility of the administrator to create the `JMS Keys` certificate realm.

For more details, enter `keystoreadmin` in the following directory, and perform the instructions at the command prompt:

```
INSTALL_DIR/apigateway/posix/bin
```

Use keystoreadmin in interactive mode

When you enter `keystoreadmin` without arguments, this displays an interactive menu with the following options:

Option	Description	When to use
1	`Change group or instance`	When registering HSMs or configuring certificate realms, you must choose the local group and instance to configure.
2	`List registered HSM providers`	Display the HSMs that are currently registered.

Option	Description	When to use
3	`Register an HSM provider`	Before creating certificate realms, you must first register the HSM. This option guides you through the steps. The HSM must be installed, configured, and active, and you must know the full path to the HSM device driver (PKCS#11). You give the HSM an alias (for example, LunaSA), which you use later when registering certificate realms.
4	`List Certificate Realms`	List configured certificate realms and associated keystores.
5	`Create a Certificate Realm`	Create a keystore and assign it to a certificate realm.

Step 1–Register an HSM provider

You must first register an HSM provider as follows:

1. Open a command prompt in the API Gateway `bin` directory (for example, `INSTALL_DIR/apigateway/posix/bin`).
2. Enter the `keystoreadmin` command.
3. Select option `3) Register an HSM provider`.
4. If prompted, select the appropriate API Gateway group or instance.
5. You are prompted for a provider alias name. The alias is local only. For example, if registering a LunaSA HSM, you might enter the `LunaSA` alias.
6. For convenience, `keystoreadmin` searches for supported HSM drivers. If found, it shows the list of supported drivers. If none are found, this does not mean the driver does not exist. You must see your HSM documentation for the location of the drivers. For example:

```
Choose from one of the following:1) /LunaSA/cryptoki.dll o)
Other q) Quit
```

7. If successful, `keystoreadmin` loads the driver and displays its details. For example:

```
Registering HSM provider...
Initializing HSM...
Crypto Version:2.20
Manufacter Id:SafeNet, Inc.
Library Description:Chrystoki
Library Version:5.1 Device registered.
```

Step 2–Create a certificate realm and associated keystore

To create a certificate realm and associated keystore, perform the following steps:

1. Open a command prompt in the API Gateway `bin` directory (for example, `INSTALL_DIR/apigateway/posix/bin`).
2. Enter the `keystoreadmin` command.
3. Select option `5) Create a Certificate Realm`.
4. You are prompted to enter a certificate realm name. This certificate realm name is used in Configuration Studio when configuring the private key of the corresponding X.509 certificate. The realm name is case sensitive (for example, `JMS Keys`).
5. The registered HSMs are listed. For example, select option `1) HSM`.
6. The command connects to the selected HSM, and a list of available slots is displayed. Select the slot containing the private key to use for the certificate realm (for example, select slot `1`).
7. You are prompted to input the PIN passphrase for the slot. The passphrase will not echo any output.
8. When you enter the correct PIN passphrase for the slot, this displays a list of private keys. Choose the key to use for the certificate realm. For example:

```
Choose from one of the following:
  1) server1_priv
  2) jms_priv
  q) Quit

Select option:2
```

9. You are prompted for a file name for the keystore. For example:

```
Certificate realm filename [jms keys.ks]:Successfully created the
certificate realm:JMS KeysPress any key to continue...
```

10. The keystore is output to the API Gateway instance directory. For example:

```
apigateway/groups/group-2/instance-1/conf/certrealms/jms keys.ks
```

Note Each API Gateway instance must have its certificate realm configured. When finished creating certificate realms, you must restart the API Gateway instance for the changes to take effect.

Step 3–Start API Gateway when using an HSM

When API Gateway is configured to use certificate realms, these realms are initialized on startup, and a connection to the corresponding HSM is established. This requires the PIN passphrase for the specific HSM slots. At startup, you can manually enter the required HSM slot PIN passphrase, or you can automate this instead.

Start API Gateway with manually entered PIN passphrase

When API Gateway is configured to use an HSM, API Gateway stops all processing, prompts for the HSM slot PIN passphrase, and waits indefinitely for input. For example:

```
INFO    07/Jan/2015:16:31:54 Initializing certificate realm 'JMS Keys'...
Enter passphrase for Certificate Realm, "JMS Keys":
```

API Gateway does not reprompt if the PIN passphrase is incorrect. It logs the error and continues, while any services that use the certificate realm cannot use the HSM.

Start API Gateway with automatic PIN passphrase

You can configure API Gateway to start and initialize the HSM by invoking a command script on the operating system to obtain the HSM slot PIN passphrase. This enables API Gateway for automatic startup without manually entering the PIN passphrase.

To configure an automatic PIN passphrase, perform the following steps:

1. Edit the API Gateway instance's `vpkcs11.xml` configuration file. For example:

   ```
   apigateway/groups/group-2/instance-1/conf/vpkcs11.xml
   ```

2. Add a `PASSPHRASE_EXEC` command that contains the full path to the script that executes and obtains the passphrase. The script should write the passphrase to stdout, and should have the necessary operating system file and execute protection settings to prevent unauthorized access to the PIN passphrase. The following example shows a `vpkcs11.xml` file that invokes the `hsmpin.sh` to echo the passphrase:

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <ConfigurationFragment provider="cryptov">

       <Engine name="vpkcs11" defaultFor="">
           <EngineCommand when="preInit" name="REALMS_DIR"
                value="$VINSTDIR/conf/certrealms" />
           <EngineCommand when="preInit" name="PASSPHRASE_EXEC"
                value=""$VDISTDIR/hsmpin.sh"" />
       </Engine>
   ```

```
</ConfigurationFragment>
```

3. API Gateway provides the certificate realm as an argument to the script, so you can use the same script to initialize multiple realms. The following examples show scripts that write a PIN of `1234` to stdout when initializing the `JMS Keys` certificate realm:

```
#!/bin/shcase $1 in"JMS Keys")echo 1234;;esac
```

Configure SSH key pairs

To configure public-private key pairs in the certificate store, select **Environment Configuration > Certificates and Keys > Key Pairs**. The **Key Pairs** window enables you to add, edit, or delete OpenSSH public-private key pairs, which are required for the Secure Shell (SSH) File Transfer Protocol (SFTP).

Add a key pair

To add a public-private key pair, click **Add** on the right, and configure the following settings in the dialog:

- **Alias**:
  Enter a unique name for the key pair.
- **Algorithm**:
  Enter the algorithm used to generate the key pair. Defaults to `RSA`.
- **Load**:
  Click to select the public key or private key files to use. The **Fingerprint** field is auto-populated when you load a public key.

Note The keys must be OpenSSH keys. RSA keys are supported, but DSA keys are not supported. The keys must not be passphrase protected.

Edit a key pair

To edit a public-private key pair, select a key pair alias in the table, and click **Edit** on the right. For example, you can load a different public key and private key. Alternatively, double-click a key pair alias in the table to edit it.

You can delete a selected key pair from the certificate store by clicking **Remove** on the right. Alternatively, click **Remove All**.

Manage OpenSSH keys

You can use the `ssh-keygen` command provided on Linux to manage OpenSSH keys. For example:

- The following command creates an OpenSSH key:

```
ssh-keygen -t rsa
```

- The following command converts an `ssh.com` key to an OpenSSH key:

```
ssh-keygen -i -f ssh.com.key > open.ssh.key
```

- The following command removes a passphrase (enter the old passphrase, and enter nothing for the new passphrase):

```
ssh-keygen -p
```

- The following command outputs the key fingerprint:

```
ssh-keygen -lf ssh_host_rsa_key.pub
```

Configure PGP key pairs

To configure Pretty Good Privacy (PGP) key pairs in the certificate store, select **Environment Configuration > Certificates and Keys > PGP Key Pairs**. The **PGP Key Pairs** window enables you to add, edit, or delete PGP public-private key pairs.

Add a PGP key pair

To add a PGP public-private key pair, click the **Add** on the right, and configure the following settings in the dialog:

- **Alias**:
  Enter a unique name for the PGP key pair.
- **Load**:
  Click **Load** to select the public key and private key files to use.

Note The PGP keys added must not be passphrase protected.

Edit a PGP key pair

To edit a PGP key pair, select a key pair alias in the table, and click **Edit** on the right. For example, you can load a different public key and private key. Alternatively, double-click a key pair alias in the table to edit it.

You can delete a selected PGP key pair from the certificate store by clicking **Remove** on the right. Alternatively, click **Remove All**.

Manage PGP keys

You can use the freely available GNU Privacy Guard (GnuPG) tool to manage PGP key files (available from http://www.gnupg.org/). For example:

- The following command creates a PGP key:

  ```
  gpg --gen-key
  ```

  For more details, see http://www.seas.upenn.edu/cets/answers/pgp_keys.html
- The following command enables you to view the PGP key:

  ```
  gpg -a --export
  ```
- The following command exports a public key to a file:

  ```
  gpg --export -u 'UserName ' -a -o public.key
  ```
- The following command exports a private key to a file:

  ```
  gpg --export-secret-keys -u 'UserName ' -a -o
  private.key
  ```
- The following command lists the private keys:

  ```
  gpg --list-secret-keys
  ```

Global import and export options

This section describes global import and export options available when managing certificates and keys.

Import and export certificates and keys

The following global configuration options apply to both the **X.509 Certificate** and **Private Key** tabs:

- **Import Certificate + Key**:
  Use this option to import a certificate and a key (for example, from a `.p12` file).
- **Export Certificate + Key**:
  Use this option to export a certificate and a key (for example, to a `.p12` file).

Click **OK** when you have finished configuring the certificate and private key.

Manage certificates in Java keystores

You can also export a certificate to a Java keystore. You can do this by clicking **Keystore** on the main **Certificates** window. Click the browse button at beside the **Keystore** field at the top right to open an existing keystore, or click **New Keystore** to create a new keystore. Choose the name and location of the keystore file, and enter a passphrase for this keystore when prompted. Click **Export to Keystore**, and select a certificate to export.

Similarly, you can import certificates and keys from a Java keystore into the certificate store. To do this, click **Keystore** on the main **Certificates** window. On the **Keystore** window, browse to the location of the keystore by clicking the browse button beside the **Keystore** field. The certificates/keys in the keystore are listed in the table.

To import any of these keys to the certificate store, select the box next to the certificate or key to import, and click **Import to Trusted certificate store**. If the key is protected by a password, you are prompted for this password.

You can also use the **Keystore** window to view and remove existing entries in the keystore. You can also add keys to the keystore and to create a new keystore. Use the appropriate button to perform any of these tasks.

Further information

For more details on supported security features, see the *API Management Security Guide*.

Manage API Gateway users 12

The **Users and Groups** node in the Configuration Studio tree enables you to manage API Gateway users and groups, which are stored in the API Gateway user store. These settings are environment-specific, and typically need to be configured during promotion to an upstream environment.

By default, the API Gateway user store contains the configuration data for managing API Gateway user information. The API Gateway user store is typically used in a development environment, and is useful for demonstration purposes.

In a production environment, user information may be stored in existing user Identity Management repositories such as Microsoft Active Directory, Oracle Access Manager, CA SiteMinder, and so on. For more details, see the relevant *API Gateway Integration Guide*.

Note API Gateway users provide access to the messages and services protected by API Gateway. However, *admin users* provide access to the API Gateway configuration management features available in Policy Studio, Configuration Studio, and API Gateway Manager. For more details, see the *API Gateway Administrator Guide*.

API Gateway users

API Gateway users specify the user identity in the API Gateway user store. This includes details such as the user name, password, and X.509 certificate. API Gateway users must be a member of at least one user group. In addition, users can specify optional attributes, and inherit attributes at the group level.

To view all existing users, select the **Environment Configuration > Users and Groups > Users** node in the tree. The users are listed in the table on the main panel. You can find a specific user by entering a search string in the **Filter** field.

Add API Gateway users

You can create API Gateway users on the **Users** page. Click the **Add** button on the right.

To specify the new user details, complete the following fields on the **General** tab:

- **User Name**:
  Enter a name for the new user.
- **Password**:
  Enter a password for the new user.
- **Confirm Password**:
  Re-enter the user's password to confirm.

- **Signing Key**:
  Click to load the user certificate from the **Certificate Store**. For details on how to create and import certificates, see *Manage X.509 certificates and keys* on page 71.

You can also specify optional user attributes on the **Attributes** tab, which is explained in the next section.

API Gateway user attributes

You can specify attributes at the user level and at the group level on the **Attributes** tab. Attributes specify user configuration data (for example, attributes used to generate SAML attribute assertions).

The **Attributes** tab enables you to configure user attributes as simple name-value pairs. The following are examples of user attributes:

- `role=admin`
- `email=steve@axway.com`
- `dept=eng`
- `company=axway`

You can add user attributes by clicking the **Add** button. Enter the attribute name, type, and value in the fields provided. The `Encrypted` type refers to a string value that is encrypted using a well-known encryption algorithm or cipher.

API Gateway user groups

API Gateway user groups are containers that encapsulate one or more users. You can specify attributes at the group level, which are inherited by all group members. If a user is a member of more than one group, that user inherits attributes from all groups (the superset of attributes across the groups of which the user is a member).

To view all existing groups, select the **Environment Configuration > Users and Groups > Groups** node in the tree. The user groups are listed in the table on the main panel. You can find a specific group by entering a search string the **Filter** field.

Add API Gateway user groups

You can create user groups on the **Groups** page. Click the **Add** button on the right to view the **Add Group** dialog.

To specify the new group details, complete the following fields on the **General** tab:

- **Group Name**:
  Enter a name for the new group.

- **Members**:
  Click the **Add** button to display the **Add Group Member** dialog, and select the members to add to the group.

You can also specify optional attributes at the group level on the **Attributes** tab. For more details, see *API Gateway user attributes* on page 85.

Update API Gateway users or groups

To edit details for a specific user or group, select it in the list, and click the **Edit** button on the right. Enter the updated details in the **Edit User** or **Edit Group** dialog.

To delete a specific user or group, select it in the list, and click the **Remove** button on the right. Alternatively, to delete all users or Groups, click the **Remove All** button. You are prompted to confirm all deletions.