webMethods API Gateway User's Guide

Version 10.15

October 2022

WEBMETHODS

This document applies to webMethods API Gateway 10.15 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: YAI-UG-1015-20240403

Table of Contents

About this Documentation

This documentation describes how you can use API Gateway and other API Gateway components to effectively manage APIs for services that you want to expose to applications, whether inside your organization or outside to partners and third parties.

To use this content effectively, you should have an understanding of the APIs that you want to expose to the developer community and the access privileges you want to impose on those APIs.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies service names and locations in the format *folder.subfolder.service*, APIs, Java classes, methods, properties.
Italic	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at https://documentation.softwareag.com.

In addition, you can also access the cloud product documentation via https://www.softwareag.cloud. Navigate to the desired product and then, depending on your solution, go to "Developer Center", "User Center" or "Documentation".

Product Training

You can find helpful product training material on our Learning Portal at https://learn.softwareag.com.

Tech Community

You can collaborate with Software AG experts on our Tech Community website at https://techcommunity.softwareag.com. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at https://github.com/softwareag and https://containers.softwareag.com/products and discover additional Software AG resources.

Product Support

Support for Software AG products is provided to licensed customers via our Empower Portal at https://empower.softwareag.com. Many services on this portal require that you have an account. If you do not yet have one, you can request it at https://empower.softwareag.com/register. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

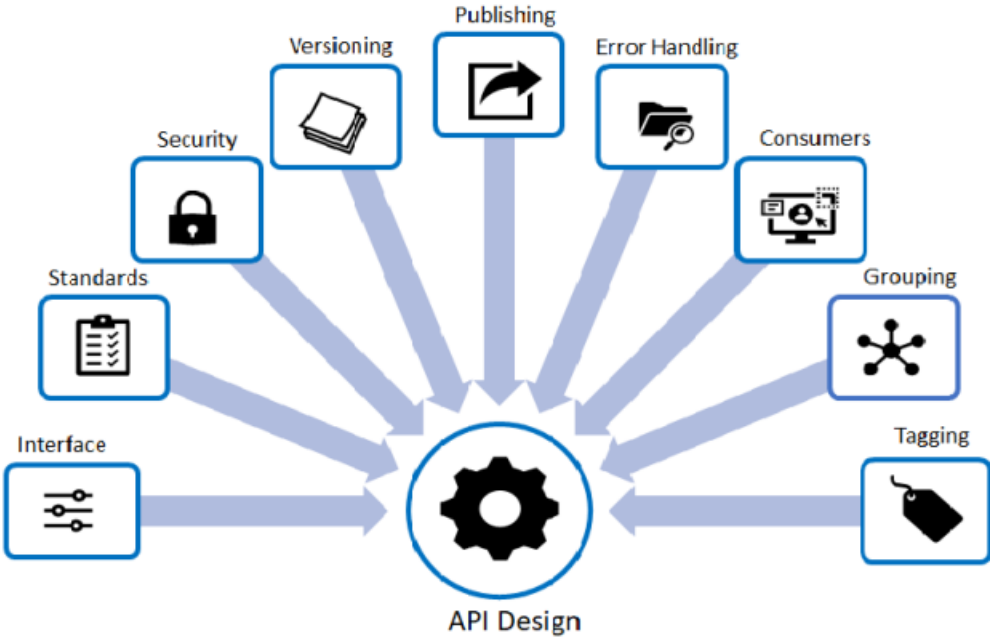
Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

1 Define and Manage APIs

Defining and Managing APIs

APIs are designed to expose application functionality and data for use by consumers and developers, as necessary. A basic API definition influences how efficiently consumers or developers are able to consume and use an API. Core API design considerations are fueled by business drivers, developer and consumer audiences, as well as the available resources. Therefore, a careful consideration of all influencing factors affects the planning and architectural decisions you make while designing and defining your API.

The following figure depicts the various factors that drive the designing and defining of an API.



API Gateway enables organizations to define and manage their APIs. Developer and partner users can use API Gateway's design and customization capabilities to build, develop, and then publish APIs to a portal for consumption by internal and external API consumers.

The following sections describe the types of APIs that API Gateway supports, the different ways in which you can create APIs, update APIs, create and maintain different versions of an API, and group and tag APIs.

API administrators and users with the appropriate functional privileges can use API Gateway's capabilities to create and manage APIs, and publish the APIs to Developer Portal or service registries from where they can be consumed.

API Gateway supports the following API types:

- **Representational State Transfer (REST)**

  Defines a set of architectural principles that allow accessing and manipulating resources by using capabilities already built into HTTP, including uniform and predefined set of stateless operations and resources. These operations and resources are accessible using URIs and are represented by media types. The RESTful framework provides REST APIs based on the REST

architecture. There are multiple specification formats for REST APIs. In API Gateway, you can create a REST API using RESTful API Modeling Language (RAML), Swagger 2.0, or OpenAPI 3.0 specifications.

- **Simple Object Access Protocol (SOAP)**

  Defines a communication method for XML-based message exchange over different transport protocols, such as Hypertext Transfer Protocol (HTTP) and Simple Mail Transfer Protocol (SMTP). The SOAP framework provides SOAP APIs based on Web Services Description Language (WSDL).

- **Open Data Protocol (OData)**

  Defines a set of best practices for the creation and consumption of RESTful APIs. It provides a uniform way to describe both data and the data model. The OData framework provides interoperable OData APIs with a RESTful interface based on OData standards.

- **WebSocket**

  Defines two-way (full duplex) communication between the client and the server, over a single Transmission Control Protocol (TCP) socket. The WebSocket protocol facilitates real-time data transfer from and to the server. The WebSocket framework provides WebSocket APIs with a RESTful interface based on W3C standards.

- **GraphQL**

  Defines a query language designed to build client applications by providing a flexible syntax and a comprehensive description of data within an API. API Gateway supports proxying an existing GraphQL endpoint and provides API management capabilities to clients like authentication, analytics, and so on. API Gateway supports GraphQL version 16.2.

Asynchronous APIs

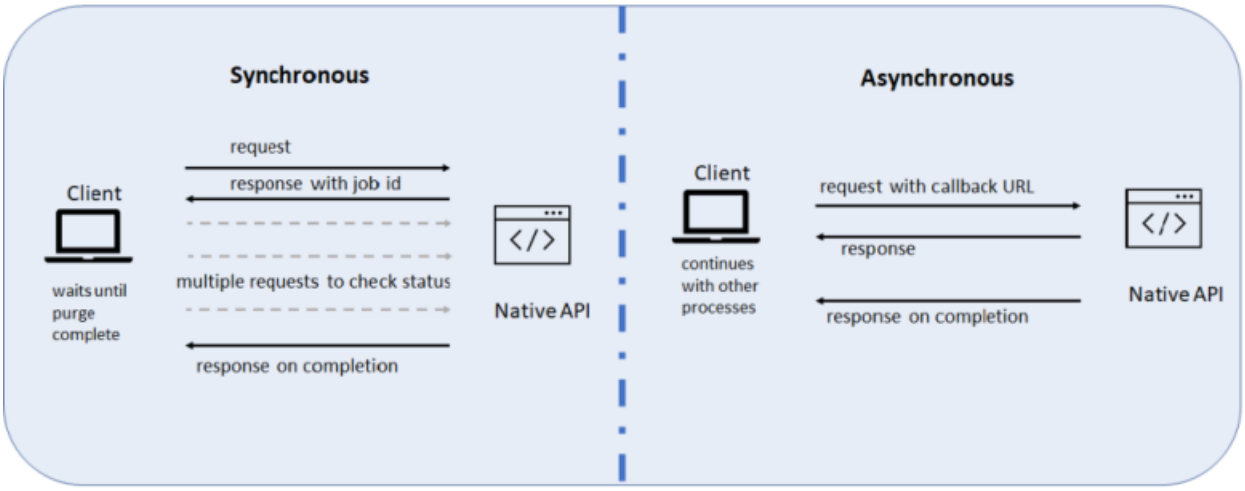
API Gateway provides an asynchronous form of API support for REST APIs. The synchronous and asynchronous nature of an API depends on the way how a request sent by an application is processed, and the mode and the time frame in which the data is returned to the client.

The following are the differences between the synchronous and asynchronous APIs.

Synchronous API	Asynchronous API
■ Synchronous API is used where data or service availability, resources and connectivity are high and requires low latency ■ The client application requests data and waits until it receives a response to proceed with other processes. The expectation is an immediate response.	■ Asynchronous API is used where data or service availability, resources and connectivity are low or over-saturated with demand ■ The client application requests data and continues with other processes without waiting for a response. There is no expectation of an immediate response.

Consider this example of synchronous and asynchronous APIs performing a log purge operation. In this scenario, because there is a wait period for the return of data, synchronous API invocations

might time out when the API processes take a significant period of time to complete. The following figure depicts the differences between how a synchronous and an asynchronous API perform the log purge operation



While creating a REST API, API Gateway provides the capability of defining the callback component with the supported method parameters. For details about creating an API with the callback definition, see "Creating a REST API from Scratch" on page 20.

Creating APIs

You can create and manage APIs from the Manage APIs page in API Gateway UI. The page lists all your APIs, their description, and version number. On this page, you can create an API, delete an API, view API details, activate or deactivate an API, publish or unpublish an API, view API analytics, group APIs, and add tags to an API.

You can create an API in one of the following ways:

- Create an API by importing a definition for an existing API (for example, in Swagger or RAML format) using an API importer.

  An API importer generates an API from a URL or an input file in one of the supported formats. For example, the RAML importer installed with API Gateway reads a RAML file and generates a REST API that the RAML definition describes. The importer also uploads the RAML file to the API Gateway repository and links the file to the REST API.

- Create an API from scratch and set its attributes manually.

The following sections explain in detail about different ways of creating APIs.

Note:
Do not provide values starting with a dot (.), in any of the fields in API Gateway UI as Elasticsearch does not support saving those values.

Creating an API by Importing an API from a File

You must have the API Gateway's manage APIs or activate/deactivate APIs functional privilege assigned to perform this task.

To create an API by importing an API from a file

1. Click **APIs** in the title navigation bar.

   A list of available APIs appears.

2. Click **Create API**.

3. Select **Import API from file**.

4. Click **Browse** to select a file.

5. Select the required file and click **Open**.

   The Swagger parser is a self-contained file with no external references and can be uploaded as is. If the RESTful API Modeling Language (RAML) file to be imported contains external references, the entire set of files must be uploaded as a ZIP file with a structure as referenced by the RAML file.

   Note:
   Importing an API fails for an invalid WSDL file.

6. Type a name for the API name in the **Name** field.

   - If you provide an API name, this overwrites the API name mentioned in the uploaded file and the API is displayed with the name provided.
   - If you do not provide an API name, the API name mentioned in the uploaded file is picked up and the API is displayed with that name.
   - If you do not provide an API name and the uploaded file does not have an API name mentioned, then the API is displayed as Untitled.

7. Select the required type.

   The available types are **OpenAPI**, **RAML**, **Swagger**, **WSDL**, and **GraphQL SDL**.

8. Provide a version for the API in the **Version** field.

9. Select the team to which the API must be assigned in the **Team** field.

   This field appears only when the Team feature is enabled. It displays only the teams that you are a part of. If you have the User management functional privilege, all teams are displayed.

You can select more than one team. To remove a team, click the  icon next to the team to be removed.

10. Click **Create**.

    An API is created with default policies.

Note:

- To avoid encountering errors while parsing large responses from the native service, you have to change the enablesoapValidation property by commenting out the `<parameter name="enableSoapValidation">true</parameter>` in *`SAG_Install_Directory`*`\IntegrationServer\instances\`*`default`*`\config\wss\axis2.xml` and restart the server for the change to take effect.
- Since the GraphQL API schema does not contain a native endpoint, you must manually update the **Native endpoint URL** in the **API details** section and the **Endpoint URI** in the routing policy after you create a GraphQL API.

Creating an API by Importing an API from a URL

You must have the API Gateway's manage APIs or activate/deactivate APIs functional privilege assigned to perform this task.

To create an API by importing an API from a URL

1. Click **APIs** in the title navigation bar.

   A list of available APIs appears.

2. Click **Create API**.

3. Select **Importing API from URL**.

4. Type the URL from where the API is to be imported.

5. Select **Protected** to make the API a protected API and provide the required credentials.

6. Type a name for the API name in the **Name** field.

   - If you provide an API name, this overwrites the API name mentioned in the uploaded file and the API is displayed with the name provided.
   - If you do not provide an API name, the API name mentioned in the uploaded file is picked up and the API is displayed with that name.
   - If you do not provide an API name and the uploaded file does not have an API name mentioned, then the API is displayed as Untitled.

7. Provide a description for the API in the **Description** field.

8. Select the required type.

   The available types are **OData**, **OpenAPI**, **RAML**, **Swagger**, **WSDL**, and **GraphQL SDL**.

9. Provide a version for the API in the **Version** field.

10. Select the team to which the API must be assigned in the **Team** field.

    This field appears only when the Team feature is enabled. It displays only the teams that you are a part of. If you have the User management functional privilege, all teams are displayed.

    You can select more than one team. To remove a team, click the  icon next to the team to be removed.

11. Click **Create**.

    An API is created with default policies.

Note:

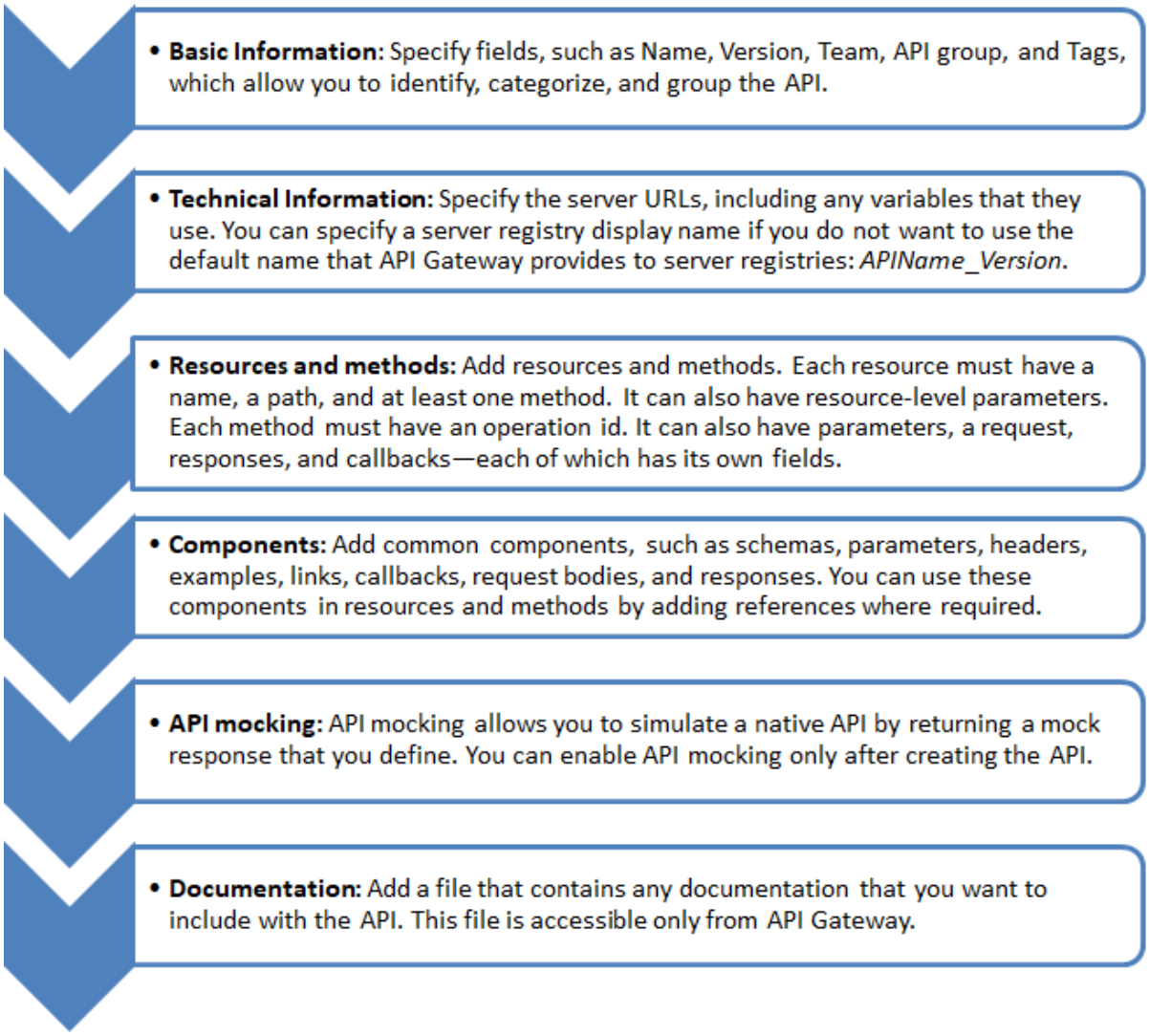
- Importing an API fails for an invalid WSDL file.
- Creating an API by importing swagger files from an HTTPS URL that is using self-signed certificates might fail. To workaround this, you can set the system environment variable as: `export TRUST_ALL=true`, so that the invalid certificates are ignored. Be aware that setting this variable makes the swagger-parser ignore all invalid certificates too. So this workaround has to be used with caution.
- To avoid encountering errors while parsing large responses from the native API, you have to change the enablesoapValidation property by commenting out the `<parameter name="enableSoapValidation">true</parameter>` in *`SAG_Install_Directory`*`\IntegrationServer\instances\`*`default`*`\config\wss\axis2.xml` and restart the server for the change to take effect.
- Since the GraphQL API schema does not contain a native endpoint, you must manually update the **Native endpoint URL** in the **API details** section and the **Endpoint URI** in the routing policy after you create a GraphQL API.

Creating an API from Scratch

You can create the following APIs from scratch, meaning that you create the asset and set its attributes manually:

- REST
- WebSocket

The **Create REST API** wizard breaks down the task of creating a REST API from scratch into logical steps. The following figure illustrates the different pages of the wizard.



Basic Information

The Basic Information page includes fields that allow you to identify, categorize, and group an API.

Technical Information

The Technical Information page includes fields that allow you to define one or more server URLs for the API. You can also define and include variables in the URLs.

You can also specify parameters for data that must be included in every request to the API. For example, if you want a specific query parameter to be included in every request, you can add a parameter of the type `Query` and specify the value that it must include.

Resources and methods

The Resources and methods page includes fields that allow you to define the API resources and methods, including callback methods. In this page, you can add all the resources and their methods that are exposed by the API.

At the resource level, you add a resource by defining the following properties: name, path, and supported methods. You can additionally add parameters for data that must be included in every request to that resource. For example, if the methods in a resource are invoked using URLs that have a query string; you can add a query string parameter that captures the queries sent by the clients.

At the method level, you identify a method by adding an operation id. In addition, you can add tags that help you to categorize and search for similar methods. You can also add parameters at the method level. Similar to the parameters at the API and resource levels, method parameters enable you to capture and process the data that is sent in a particular request. In the case of method parameters, the data in the request for that method is captured and processed.

Method Requests

In the request section of a method, you can define the schema for requests that contain a JSON or XML payload. As a method can support multiple content types, you need to add a content type and then define the schema supported by that content type.

You can enter a schema or select an existing schema or global schema that you have previously added on the Components page, Schemas section. You can also add a sample for the schema that you have added or selected. These samples can be used for API mocking. They can also be used by end users to get a better understanding of the API.

Method Responses

You can define responses for different HTTP status codes. API Gateway gives you the flexibility to define responses for a status codes series (such as the 2XX series or the 4XX series) or for specific response codes, such as 201 or 400.

Note:
If you have defined the response for a series and specific numbers in that series, the more specific one is used. Example: If you have added an entry for 2XX and 201, a response with the HTTP status code 200 will be the same as 2XX. However, a response with the HTTP status code 201 will pick the one that is defined for 201.

For each status code in a method response, you define the following:

- **Response body**: you define the response body using the following fields:
  - Content Type: You can select from any of the content types.
  - Schema: You can define a schema if the response contains JSON or XML data.
  - Sample: The samples are used for API mocking. They can also be used by end users to get a better understanding of the API.

- **Header parameter**: You can add a parameter to capture and process a header in the response sent by the native API.
- **Links**: Links allow the developer of the native API to define the relationship and traversal mechanism between a response and other operations. You can include links to other methods that are related to the response. This enables an API client to dynamically navigate the methods that are exposed by the API. For example, a method that returns the temperature in Fahrenheit for a given place may also include links to methods that return: a) the temperature in Centigrade; and b) the temperature of the place on a given day of the year.

Note:
You can define the complete response, or any part of it (response body schema, header parameter, or link), in the **Components** page; and reuse it wherever required by giving a reference.

Method Callbacks

A callback is an asynchronous API request that originates from the API server and is sent to the client in response to an earlier request sent by that client. APIs can use callbacks to signal an event of interest and share data related to that event. API clients that are interested in an event or data related to that event, include a callback URL in the request they send to the API. For more information about Asynchronous APIs, see " Defining and Managing APIs" on page 10.

To enable API Gateway to process callback messages, you must configure the Callback processor settings, as explained in *webMethods API Gateway Administration*.

If your API supports callbacks, you can use API Gateway to process the initial requests, the callback URLs sent by clients, and the response sent by the API—including the callback messages. Clients can provide the callback URL to API Gateway in any of the following ways:

- Request header
- Query parameter
- Request body (if the response body has JSON or XML content)

You must define the relevant parameter to capture the callback URL to process it. API Gateway can wrap the client callback URLs with its own URL to process these requests if the callback URL path defined in the following formats. Otherwise, API Gateway sends the requests received from client as it receives it.

Format	Description
{$request.query.*param-name*}	Where *param-name* is the name of the query parameter that contains the callback URL.
{$request.header.*header-name*}	Where *header-name* is the name of the header that contains the callback URL.
{$request.body#/*field-name*}	Where *field-name* is a field in the request body. If the field is an array, use the syntax {$request.body#/*field-name*/*arrayIndex*}, where

Format	Description
	arrayIndex is the index of the callback URL in the array.
${response.header.*header-name*} and ${response.headers.*header-name*}	Where *header-name* is any of the valid header.
${request.query.*param-name*}	Where *param-name* is the name of the query parameter that contains the callback URL.
${response.payload.jsonPath[*queryValue*]}	Where *queryValue* is a valid JSON path expression.
${response.payload.xpath[*queryValue*]}	Where *queryValue* is a valid XPath path expression.

If you have enabled API Gateway to process callback messages, API Gateway wraps the callback URL provided by the client and sends an API Gateway URL to the native API. When the native API invokes the same callback URL, API Gateway processes the response and applies the policies that you have defined.

API Gateway can apply the following policies on the callback messages:

- Invoke webMethods IS
- Response Transformation
- Validate API Specification
- Data Masking
- Log Invocation

Note:
These policies are applied to the immediate responses of an API request and to all its callback requests. These policies are enforced against callback request payloads.

API mocking

API mocking allows you to simulate a native API that is not available. The mock response that you define is returned to the client that invokes the API, if the native API is not available. API mocking is not available while you are creating an API. To use API mocking, you must edit the API after creating it and enable API mocking. For more information about API mocking, see “API Mocking” on page 74.

Components

The Components page allows you to add reusable elements that you can use in other pages of the wizard. You can reference these global elements using the `$ref` variable. You can add the following global elements:

- **Schemas**: The schema specified here can be reused in the resource and method specifications across multiple methods and resources.
- **Parameters**: You can define parameters that can be used as API, resource, and method parameters.
- **Headers**: You can define parameters that can be reused as header parameters at the API, method, and response levels.
- **Examples**: You can add examples that can be reused as samples across operations in the API.
- **Links**: You can define links that can be reused in responses. For more information about links, see *Links* within *Method Responses* section.
- **Callbacks**: You can define callback methods in this page and include them in the callback section of the methods that use it.
- **Request Bodies**: You can define request bodies in this page and reuse them in methods. A request body includes the content type, a schema, and a sample.
- **Responses**: You can define responses in this page and reuse them in methods. A response includes the content type, a schema, and a sample. It can also include header parameters and links.

Documentation

In the view mode, the Documentation page provides the following links:

- Links to the Swagger, RAML, and OpenAPI versions of the API on the Integration Server.

  Note:
  If Cross-Site Request Forgery (CSRF) token is enabled on the Integration Server, the links to three types of APIs will not work. You must configure Integration Server to allow these links to work.

- Links to download the API in the three different formats: Swagger, RAML, and OpenAPI.

In the edit mode, the **Documentation** page allows you to add a file that contains any documentation that you want to include with the API. This file is accessible only from API Gateway.

Creating a REST API from Scratch

You must have the API Gateway's manage APIs or activate/deactivate APIs functional privilege assigned to perform this task.

You can create a REST API from scratch by providing the basic information, technical information, and defining the resources and methods as required.

To create a REST API from scratch

1. Click **APIs** in the title navigation bar.

A list of all existing APIs appears.

2. Click **Create API**.
3. Select **Create from scratch**.
4. Select **REST**.
5. Click **Create**.

   The **Basic information** page of the **Create REST API** wizard appears.

6. Provide the following information in the Basic information section:

Field	Description
Name	Name of the API.
Version	Version of the API being created.
Team	Team to which the API must be assigned. This option is visible only if you have enabled the Teams feature. You can select more than one team. To remove a team, click the  icon next to the team to be removed.
Maturity state	Maturity state of the API. Available values are: **Beta**, **Deprecated**, **Experimental**, **Production**, **Test**. The available values depend on the Maturity states configured in the `apiMaturityStatePossibleValues` property under **Administration > Extended settings** section.
API grouping	Group under which the API would be categorized. Available values are: **Finance Banking and Insurance**, **Sales and Ordering**, **Search**, and **Transportation and Warehousing**. The available values depend on the groups configured in the `apiGroupingPossibleValues` property under **Administration > Extended settings** section.
Tags	Keywords for categorizing, identifying, and organizing APIs. You select from the list of existing tags or create new tags.
Description	Description of the API.

7. Click **Continue to provide technical information for this API >**.

Note:
Click **Save** to save the API at this stage and close the **Create REST API** wizard.

8. Provide the details of the servers that serve the API in the **Add server details** section.

   a. Click **Add server** and provide a **Server URL** and **Description**.

      You can include variables in the server URL by enclosing them in curly braces. These variables are added to the list of variables. However, you have to edit these variables to add a default value, and optionally one or more values and a description.

   b. Click **Add variables** and provide the following values:

      - Name
      - Description
      - Default
      - Value

      Note:
      Click **+** to add the value that you have entered.

   c. Click **Add** to add the variable.

9. Click **Add Parameter** and provide the following information to add the API-level parameters.

Field	Description
Name	Name of the parameter.
Reference	If you want to reuse a parameter defined on the **Components** page, select the parameter from the drop-down list.
Description	Description of the parameter.
Type	Specifies the parameter type. Available values: **Query-string**, **Header**, **Cookie**.
Data type	Specifies the data type. Available values: **String**, **Date**, **Date time**, **Integer**, **Double**, **Boolean**, **File**.
Required	Specifies the parameter is required if selected.
Repeat	Select if the input parameter is of type array.
Value	Specifies the possible values.

Note:
You need to define parameters only for data that you want API Gateway to process.

10. Type a **Service registry display name**.

    By default, the API is displayed in service registries with the name: *APIName_Version*. If you want the API to be displayed in the service registries with a different name, you can type the name here.

11. Click **Continue to provide Resource and methods for this API>**.

    Note:
    Click **Save** to save the API at this stage and close the **Create REST API** wizard.

12. Add resources to the API using the Resources and methods page:

    a. Click **Add Resources** and provide the following information:

Field	Description
Resource name	Name of the resource. This is the display name of the resource and resource path is used for execution.
Resource path	Specifies the path of the resource. The resource path should contain a "/".
Description	Description of the resource.
Supported methods	Select the methods that are supported by the API: GET, HEAD, POST, PUT, DELETE, PATCH.

    b. Click **Add**.

    The resource is added. You can multiple resources, if required.

    c. Add **Tags**.

    d. Click **Add Resource Parameter** and provide the following information:

Field	Description
Name	Name of the parameter.
Reference	If you want to reuse a parameter defined on the **Components** page, select the parameter from the drop-down list.

Field	Description
Description	Brief description of the parameter.
Type	Specifies the parameter type. Available values: **Path**, **Header**, **Query-string**, **Cookie**.
Data type	Specifies the data type. Available values: **String**, **Date**, **Date time**, **Integer**, **Double**, **Boolean**, **File**.
Required	Specifies the parameter is required if selected.
Repeat	Select if the input parameter is of type array.
Value	Specifies the possible values for the parameter.

e. Click **+ Add** to add the resource parameter.

13. For each supported method that you have added for a resource, provide the following information:

a. **Common information**:

Field	Description
Description	Type a description for the operation.
OperationId	Type an operation Id.
Tags	Type or select the keywords that you want to add to the operation.

b. **Method parameters**

Field	Description
Name	Name of the parameter.
Reference	If you want to reuse a global parameter defined on the **Components** page, select the parameter from the drop-down list.
Description	Brief description of the parameter.
Type	Specifies the parameter type. Available values: **Query-string**, **Header**, **Cookie**.
Data type	Specifies the data type.

Field	Description
	Available values: **String**, **Date**, **Date time**, **Integer**, **Double**, **Boolean**, **File**.
Required	Specifies the parameter is required, if selected.
Repeat	Select if the input parameter is of type array.
Value	Specifies the possible values for the parameter.

c. **Requests**.

You can select an existing global request defined on the **Components** page or specify a new request. To create a new request, select **New request**.

To add a new request that has to be processed, click **Add Request +** and provide the following information:

- **Content type**. Select one and click **Add**.
- **Schema**. Type a schema in the text box or select an existing schema from the **Select a Schema** list. You can also click **Add global schema** and create a new global schema on the **Components** page. After creating the global schema you can select it from the **Select a Schema** list.
- **Sample**. Type a sample for selected schema. This sample can be used for API mocking, if required.

To use an existing global request to process a request, select **Global request** and provide the following information:

- **Name**.
- **Reference**. Select one and click **Add**.

d. **Responses**.

First, add a status code using the **Status Code** drop-down list. Next, click on the status code to select it. For the selected status code, you can select an existing global response defined on the **Components** page or type a new response. To enter a new response, select **New response** and define the response by adding a schema and a sample for the response body, header parameters, and links.

Note:
You can also define the response for an HTTP status code series, such as 2** or 4**.

To define a new response for the selected status code, click **Add response +** and provide the following information:

- **Content type**. Select one and click **Add**.

- **Schema**. Type a schema in the text box or select an existing schema from the **Select a Schema** list. You can also click **Add global schema** and create a new global schema on the **Components** page. After creating the global schema you can select it from the **Select a Schema** list.
- **Sample**. Type a sample for selected schema. This sample can be used for API mocking, if required.

To use an existing global response, select **Global response** and provide the following information:

- **Name**. Name of the response.
- **Reference**. Select one and click **Add**.

To add a header parameter, click **+ Add method parameter** and provide the following information to add a method parameter:

Field	Description
Name	Name of the parameter.
Reference	If you want to reuse a global parameter defined on the **Components** page, select the parameter from the drop-down list.
Description	Brief description of the parameter.
Type	Specifies the parameter type. Available values: **Header**.
Data type	Specifies the data type. Available values: **String**, **Date**, **Date time**, **Integer**, **Double**, **Boolean**, **File**.
Required	Specifies the parameter is required if selected.
Repeat	Select if the input parameter is of type array.
Value	Specifies the possible values for the parameter.

Click **+** in the **Value** text box to add a value to the list, and click **Add** to add the header.

To add a link, click **+ Add links** and enter the following information to add a link:

- **Name**. Name of the link.
- **Description**. Description for the link.
- **Link**. You can add a new link or select an existing global link that is defined on the **Components** page.

  To add a new link, select **New link** and provide the following information:

- **Type**. Select **OperationId** for local operations only. **OperationRef** can be used for both local and external operations.
- **Value**. If **Type** is **OperationRef**, provide a reference to the target operation using the JSON Reference syntax (using by the **$ref** keyword); and if the **Type** is **OperationId** provide the **OperationId** of the target operation.
- **Parameters**. Specify the parameters of the target operation that are required to follow the link. Enter a **Name** and **Value**, and click **Add**.
- **Request body**. Type a request body only if the target operation has a body. Define the contents of the body of the target operation.

To include an existing global link, select **Global link** and then select an existing global link from the **Reference** drop-down list.

e. **Callbacks**. You can add the callbacks that are supported by the method. You can add new callbacks and select existing global callbacks.

> **Note:**
> For more information about using callbacks to develop asynchronous APIs, see *Asynchronous APIs* in " Defining and Managing APIs" on page 10. For more information on defining and using callbacks in API Gateway, see "Creating an API from Scratch" on page 15.

To specify a new callback, click **+ Add callbacks** and define the callback:

- **Name**. A name for the callback resource.
- Click **+ Add resources** and provide details of the API that serves as the callback API.

> **Note:**
> The user interface and procedure for defining a callback is similar to defining a resource and methods within the resource.

To include a global callback defined on the **Components** page, provide the following information:

- **Name**. Name of the callback resource.
- **Reference**. If you want to reuse a global callback defined on the **Components** page, select the callback from the drop-down list and click **Add**.

14. Click **Continue to provide Mocking information for this API>**.

> **Note:**
> Click **Save** to save the API at this stage and close the **Create REST API** wizard.

The API mocking page appears. API mocking is not enabled for a new API. You must edit the API and enable API mocking after creating the API.

15. Click **Continue to define API components for this API>**.

Alternatively, you can click **Components**.

Note:
Click **Save** to save the API at this stage and close the **Create REST API** wizard.

16. Define the reusable elements that you want to reuse in other pages of the Create REST API wizard.

    An API may have several elements that are common across resources and methods, such as schemas for response bodies. You can place such common elements in the **Components** section and reference them using the *$ref* alias.

    a. In the **Schemas** section, click **+ Add schema** and provide the following information:

Field	Description
Name	Name of the schema.
Value	Specifies the schema type. Available types: ■ **Inline schema**. Type the request and response values for the schema in the text box. ■ **Upload schema**. Click **Browse** and upload a schema file that you have from a saved location.
Action	Click  to add the schema.

    Click **+ Add** to add the schema component.

    b. In the **Parameters** section, click **+ Add parameter** and provide the following information:

Field	Description
Name	Name of the parameter.
Description	Description of the parameter.
Type	Specifies the parameter type. Available values: **Path**, **Query-string**, **Header**, and **Cookie**.
Data type	Specifies the data type. Available values: **String**, **Date**, **Date time**, **Integer**, **Double**, **Boolean**, and **File**.
Required	Specifies the parameter is required if selected.

Field	Description
Repeat	Select if the input parameter is of type array.
Value	Specifies the possible values for the parameter.

Click **+ Add** to add the parameter component.

c. In the **Headers** section, click **+ Add header** and provide the following information:

Field	Description
Name	Name of the header.
Description	Description of the header.
Type	Specifies the header type. This is fixed as **Header** for headers.
Data type	Specifies the data type. Available values: **String**, **Date**, **Date time**, **Integer**, **Double**, **Boolean**, and **File**.
Required	Specifies the header is required if selected.
Repeat	Select if the input parameter is of type array.
Value	Specifies the possible values for the header.

Click **+ Add** to add the header component.

d. In the **Examples** section, click **+ Add examples** and provide the following information:

Field	Description
Name	Name of the example.
Summary	Description of the example.
Value	The content of the example.

Click **+ Add** to add the example component.

e. In the **Links** section, click **+ Add links** and provide the following information:

Field	Description
Name	Name of the link.
Description	Description of the link.

Field	Description
Type	Specifies the link type: **OperationId** or **OperationRef**.
Value	Path to the target operation or a reference to the target operation.
Parameter name	Name of the parameter to pass as a parameter to the target operation.
Parameter value	Value for the parameter. Click **+ Add** to add the parameter. You can additional parameters if required.
Request body	Payload of the request sent to the target operation.

Click **Add** to add the link component.

f. In the **Callbacks** section, click **+ Add callback** and provide the following information:

a. Type a name for the callback.

b. Click **+ Add resources**.

c. Type the **Callback path**.

d. Select the supported methods.

e. Click **Add**.

f. For each method that you have just added, complete the next two steps.

g. Click **+ Add Resource Parameter** and add the required resource parameters. The procedure for adding resource parameters is given in Step 11d.

h. Define the selected methods. The procedure for defining methods is given in Step 12.

g. In the **Request Bodies** section, click **+ Add request** and provide the following information:

Field	Description
Name	Name of the request.
Content type	Select a content type from the list.
Schema	Select an existing schema from the list.
Sample	Type a sample of the schema.

Click **Add** to add the request component.

h. In the **Responses** section, click **+ Add Response** and provide the following information:

Field	Description
Name	Name of the response.
Content type	Click **Add**.
Schema	Select an existing schema from the list.
Sample	Type a sample of the schema.
Header Parameter	Click **+ Add Header Parameter** and provide the required information. Then, click **+ Add** to add the header parameter.
Links	Click **+ Add Links** and provide the required information. Then, click **Add** to add the link.

Click **Add** to add the response component.

17. Click **Continue to provide API documents for this API>**.

Note:
Click **Save** to save the API at this stage and close the **Create REST API** wizard.

The Documentation page appears.

18. Type a display name and click **Browse** to select a file.

19. Click **+ Add** to upload the file and add a new row.

20. Click **Save** to save your changes and create the API.

Creating a WebSocket API from Scratch

You must have the API Gateway's manage APIs or activate/deactivate APIs functional privilege assigned to perform this task.

You need the WebSocket port to access the WebSocket API. Assigning global and API-specific policies is similar to assigning policies to REST or SOAP APIs.

Note:
You can not apply global policies and policy templates to a WebSocket API.

To create a WebSocket API from scratch

1. Click **APIs** in the title navigation bar.

2. Click **Create API**.

3. Select **Create from scratch**.
4. Select **WebSocket**.
5. Click **Create**.
6. Provide the following information in the Basic information section:

Field	Description
Name	Name of the API.
Version	Version of the API.
Team	Team to which the API must be assigned. This option is visible only if you have enabled the Teams feature. You can select more than one team. To remove a team, click the icon next to the team to be removed.
Description	Description of the API.

7. Click **Continue to provide technical information for this API>**.

   Alternatively, you can click **Technical information** to go to the Technical information section.

   Click **Save** to save the API at this stage and provide the technical information for the API at a later time.

8. Provide the following information in the Technical information section:

   a. Type the WS URL in the **WS Url** field.

      The format used is `ws://`hostname:port`/path`.

   b. Click **+ Add parameter** and provide the following information:

Field	Description
Name	Name of the parameter.
Description	Description of the parameter.
Type	Specifies the parameter type. Available values are: **Query-string**, **Header**.

Field	Description
Data type	Specifies the data type. Available values are: **String**, **Date**, **Date time**, **Integer**, **Double**, **Boolean**.
Required	Select this to specify that the parameter is required.
Array	Select this to specify that the array is required.
Value	Type the required value and click **+** to add the value. Click  to include multiple values.

c. Click **+ Add message** and provide the following information.

Field	Description
Origin	Specifies the origin of the message. Available values are: **Server**, **Client**.
Type	Specifies the message type. Available types are: **Text**, **Binary**
Sample message payload	Provide the sample message payload.
Message description	Provide the message description.

Click  to include multiple messages.

9. Click **Save**.

Viewing API List and API Details

You can view the list of registered APIs, activate, delete, or view analytics of a specific API in the Manage APIs page. In addition, you can view API details, modify API details, activate and deactivate an API in the API details page.

Note:
If you encounter any problem viewing the API details with a message that says API loading has failed, this would be because the property `watt.server.http.jsonFormat` is set to a value that is not `parsed`(the default value), which API Gateway does not support.

To view API list and API details

1. Click **APIs** in the title navigation bar.

   A list of all registered APIs appears. The APIs are sorted based on their names. When there is more than one API with same name, they are sorted based on their system versions. The list displays the following details:

Column	Description
Name	Displays API name with an icon representing the API type. API type can be REST, SOAP, OData, and WebSocket.
Description	Displays brief description of the API.
Active endpoints	Indicates the active endpoints available for the API and shows how an API can be called. These are the active endpoint indicators: ■  specifies that the API Gateway endpoint is active. This implies that the API can be called on the API Gateway endpoint. ■  specifies that the API Gateway endpoint is inactive. The API is not exposed by API Gateway and API calls are rejected by API Gateway with HTTP 404 responses. ■   specifies that the API is deployed to one or more Microgateways and therefore has active Microgateway endpoints. The API does not have any active API Gateway endpoints. Any API calls against API Gateway are rejected by API Gateway with HTTP 404 responses. The available Microgateway endpoints can be looked up in the API details screen. The list of active Microgateway endpoints is updated whenever a new Microgateway is registered or a Microgateway is de-registered. If the last Microgateway becomes unavailable, the endpoint indicator no longer shows active Microgateway endpoints ■   specifies that API Gateway and Microgateway endpoints are active but there is no routing of API calls from API Gateway to Microgateway endpoints. This situation results from deploying an API with an active API Gateway endpoint to one or more Microgateways. The policy enforcement is done on the API Gateway and Microgateways independently. Deactivating and activating the API in API Gateway establishes the routing to the Microgateway endpoints.

Column	Description
	■  specifies that API Gateway and Microgateway endpoints are active. API calls against the API Gateway endpoint are routed to the Microgateway endpoints. If multiple Microgateway endpoints are available, API Gateway applies load balanced routing to the API calls. The load balancing follows the round-robin algorithm. If a Microgateway endpoint becomes unavailable the next endpoint is contacted. If no Microgateway endpoint replies, the API call in API Gateway fails. The list of Microgateways covered by the routing is updated dynamically. The policy definitions in API Gateway are enforced by Microgateways. To activate the routing in API Gateway to Microgateways, the APIs have to be deployed to Microgateway first before activating the API in API Gateway. If the last Microgateway becomes unavailable the routing is not removed implicitly. API calls against API Gateway fail as no Microgateway endpoint is available. ■  specifies that an API has an active AppMesh endpoint, but no active API Gateway endpoint. An AppMesh endpoint is established by performing an APIfy operation. The policy definitions in API Gateway are enforced within the AppMesh. The API is not exposed by API Gateway and API calls are rejected by API Gateway with HTTP 404 responses. ■  specifies that API Gateway and AppMesh endpoints are active. API calls against the API Gateway endpoint are routed to the AppMesh endpoint. The policy definitions in API Gateway are enforced within the AppMesh.
Version	Displays API version.
Modified Time	Displays the time when the API was last modified.

You can perform the following operations in the **Manage APIs** page.

- Filter APIs by **Type**, **Activation status**, **Team**, or **Active endpoints**. Select the required API type, status, team or active endpoints to view the APIs based on the provided filters.

  Note:
  The Team filter is applicable only if you have enabled the Teams feature.

- Activate an API by clicking  that denotes an inactive state.

  Once an API is activated, the Gateway endpoint is available which can be used by the consumers of this API.

- Deactivate an API by clicking  that denotes an active state.
- Export an API by clicking 
- Delete an API by clicking  in the respective row.
- View API analytics by clicking  in the respective row.
- Publish or Unpublish an API by clicking  and  respectively.

2. Click any API to view API details.

   The API details page displays the basic information, technical information, resources and methods, and specification for the selected API. This page allows you to edit some of the API details. Also, this page provides options to activate or deactivate an API. Click to export, enable or disable mocking, update, and create new version operations.

   Note:
   The link provided in the **Documentation** section of the **API details** tab can be accessed using API Gateway internal users credentials and cannot be accessed using SSO user credentials.

REST API Details

The REST framework enables you to model APIs conforming to the Resource Oriented Architecture (ROA) design. For example, you might model an API that serves to expose the web service data and functionality as a collection of resources. Each resource is accessible with unique Uniform Resource Identifiers (URIs). In your API, you expose a set of HTTP operations (methods) to perform on a specific resource and capture the request and response messages and status codes that are unique to the HTTP method and linked within the specific resource of the API.

The API details view for a REST API displays the details of the API such as basic and technical information, resources and methods, API mocking details, and specifications. You can also view the scopes associated, policies enforced, registered applications and the API-specific analytics.

The table lists the API details displayed for the API

Field	Description
Basic information	Displays the information about the API, such as Name, Version, Owner of the API, the teams that the API is assigned to, status of the API whether its is Active or Inactive, the maturity state of the API, the date on which the API was created and a brief description of the API.
Technical information	Displays the following endpoints of the API: - **Native endpoints**.

Field	Description
	■ **Gateway endpoints**. Displays these endpoints when the API is deployed to a gateway. ■ **Microgateway endpoints**. Displays these endpoints when the API is deployed to a Microgateway. ■ **AppMesh endpoints**. Displays these endpoints when the API is deployed through AppMesh. ■ **Service Registry display name**. Displays the name of the service registry where the API is deployed.
Resources and methods	Displays a list of resources or methods available in the API sorted by resource/pathname. The list of resources are displayed in sorted order of the path names. Click each resource to view the corresponding HTTP methods, along with a summary. Below each of these methods, details such as parameters and response codes are displayed.
API mocking	Details are visible only when API mocking is enabled. Displays a list of mocked responses for the operations in the API, custom IS service list and conditions along with its mocked response.
Components	Displays the schemas defined at the API level.
Documentation	Displays the definition of the API in different formats.

Various tabs displayed in the API details page display the following details:

- The **Scopes** tab lists the scopes available for the API.
- The **Policies** tab displays the policies enforced for the API.
- The **Mashups** tab displays the mashups defined in the API.
- The **Applications** tab displays all the applications registered with the API.
- The **Analytics** tab displays the API-specific analytics for the time interval selected.

You can perform the following operations from the API details page:

- You can enable API mocking by clicking the **Enable mocking** button. If API mocking is enabled, you can disable it by clicking the **Disable mocking** button. This option is available when the API is in the deactivated state.
- You can update an API by importing from file or from URL by clicking the **Update** button. This option is available when the API is in the deactivated state.
- You can create a new version of the API by clicking the **Create new version** button.

- You can modify details of an API by clicking the **Edit** button. This option is available when the API is in the deactivated state.
- You can activate an API by clicking the **Activate** button. If the API is already activated, you can deactivate it by clicking the **Deactivate** button.

Microgateway endpoints

Microgateway endpoints are exposed in this section when one or more Microgateways start connecting to API Gateway with a particular API. When you activate the API, the routing to the connected Microgateway endpoints comes in effect. This means that when you call the API Gateway endpoint in the usual way, the request is directly routed to the Microgateway. If there are multiple Microgateways, the routing is done in a round-robin order to each of the participating Microgateways. The called Microgateway processes the request with all the defined policies.

AppMesh endpoints

The AppMesh endpoint gets exposed only for APIs created by AppMesh's APIfy operation. In an AppMesh context, Microgateway and the corresponding micro services are behind a Kubernetes service or a loadbalancer. When you activate the API, the routing to this AppMesh endpoint (depending on your Kubernetes service loadbalancer setting) comes in effect. This means that when you call the API Gateway endpoint in the usual way the request is directly routed to that endpoint.

SOAP API Details

The API details view for a SOAP API displays the details of the API such as Basic and Technical information, Operations available, REST transformation details, API mocking details, and specifications. You can also view the scopes associated, policies enforced, registered applications and the API-specific analytics.

The table lists the API details displayed for the API

Field	Description
Basic information	Displays the information about the API, such as Name, Version, Owner of the API, the teams that the API is assigned to, status of the API whether its is Active or Inactive, the maturity state of the API, the date on which the API was created and a brief description of the API.
Technical information	Displays the following endpoints of the API: - **Native endpoints**. - **Gateway endpoints**. Displays these endpoints when the API is deployed to a gateway. - **Service Registry display name**. Displays the name of the service registry where the API is deployed.

Field	Description
Operations	Displays a list of operations available in the API and they are sorted alphabetically. Operations are displayed along with their type of binding (SOAP 11 , SOAP 12, and other HTTP methods). Click each method to view details such as input, output, and fault messages.
REST transformation	Displays a list of operations exposed as REST resources and they are sorted alphabetically. Operations are displayed along with their type of binding. Click each method to view details such as input, output, and fault messages.
API mocking	Details are visible only when API mocking is enabled. Displays a list of mocked responses for the operations in the API, custom IS service list and conditions along with its mocked response that contains the status code and mock payload details.
Documentation	Displays a list of specifications for the API.

Various tabs displayed in the API details page display the following details:

- The **Scopes** tab lists the scopes available for the API.
- The **Policies** tab displays the policies enforced for the API.
- The **Applications** tab displays all the applications registered with the API.
- The **Analytics** tab displays the API-specific analytics for the time interval selected.

You can perform the following operations from the API details page:

- You can enable API mocking by clicking the **Enable mocking** button. If API mocking is enabled, you can disable it by clicking the **Disable mocking** button. This option is available when the API is in the deactivated state.
- You can update an API by importing from file or from URL by clicking the **Update** button. This option is available when the API is in the deactivated state.
- You can create a new version of the API by clicking the **Create new version** button.
- You can modify details of an API by clicking the **Edit** button. This option is available when the API is in the deactivated state.
- You can activate an API by clicking the **Activate** button. If the API is already activated, you can deactivate it by clicking the **Deactivate** button.

Exposing a SOAP API to Applications

An active SOAP API exposes its WSDL with a couple of adaptions:

- The service name becomes the selected API Name.
- Custom endpoints as well as URL aliases appear with separate port elements.
- The HTTP and HTTPS endpoints are exposed if they meet the respective settings in the Transport policy.
- The SOAP and SOAP12 entries are exposed if they meet the respective settings in the Transport policy.
- Only enabled operations are exposed.
- The values from the Inbound Auth - Message policy are integrated into the WSDL as ws:Policy entries. Note that the original ws:Policy entries from the importing WSDL are not considered.

OData API Details

Open Data Protocol (OData) enables the creation of REST-based APIs, which allow resources to be exposed as endpoints and identified using the Uniform Resource Identifiers (URIs). In general, OData is represented by an abstract data model called Entity Data Model (EDM). This Entity Data Model allows web clients to publish and edit REST services and their resources using simple HTTP messages. OData leverages the principles of HTTP, REST and ATOM, and combines the simplicity of REST and SOAP metadata definitions to describe service interfaces, data models, and semantics.

API Gateway supports OData V4 and V2 services.

The API details view for an OData API displays the details of the API such as basic and technical information, OData entity sets, singletons, function imports, actions imports and specifications. You can also view the policies enforced, registered applications and the API-specific analytics.

The API Gateway UI exposes only OData navigation properties to visualize the resource path structure of OData APIs. Any other OData property is not displayed.

Note:
API Gateway does not support the querying of Derived Entity Types. This includes the following operations:

- Requesting a Derived Entity
- Requesting a Derived Entity Collection
- Filter on Derived Type

Operations on Derived Types are rejected by API Gateway.

The table lists the API details displayed for the API.

Field	Description
Basic information	Displays the information about the API, such as Name, Version, the teams that the API is assigned to, status of the API whether its is Active or Inactive, the date on which the API was created, and a brief description of the API.

Field	Description
Technical information	Displays base URL of the API and the OData version supported
OData entity sets	Displays a list of OData entity sets. An entity set element represents a single entity or a collection of entities of a specific entity type in the data model The list of entity sets is sorted alphabetically. Click each entity set to view the resource path, entity type, resource parameter details, and the corresponding HTTP methods. The entity types are structured records consisting of named and typed properties and key properties whose values uniquely identify one instance from another.
OData singletons	Displays a list of OData singletons. Singletons are single entities which are accessed as children of the entity container. The list of singletons is sorted alphabetically. Click each singleton to view the resource path, entity type, the corresponding HTTP methods, and the navigation properties that allow navigation from an entity to related entities. The OData navigation property has an impact on the resource structure. This property is represented as an OData Resource and denoted as **OData Navigation properties** inside the OData Resources profile. There is no restriction to the number of levels you can drill down.
OData function imports	Displays a list of OData function imports. The Function Import element represents a function in an entity model. The list of OData function imports is sorted alphabetically. Click each function import to view the resource path, entity type, and the corresponding HTTP methods.
OData action imports	Displays a list of OData action imports. The Action Import element represents an action in an entity model. The list of OData action imports is sorted alphabetically. Click each action import to view the resource path, entity type, and the corresponding HTTP methods.
Documentation	Displays a list of specifications for the API. The metadata document. The metadata document describes the Entity Data Model that is, the structure and organization of the OData service resources) exposed as HTTP endpoints by that particular service. This document describes the entity types, entity sets, functions, and actions.

Various tabs displayed in the API details page display the following details:

- The **Policies** tab displays the policies enforced for the API.
- The **Applications** tab displays all the applications registered with the API.
- The **Analytics** tab displays the API-specific analytics for the time interval selected.

You can perform the following operations from the API details page:

- You can update an API by importing from URL by clicking the **Update** button. This option is available when the API is in the deactivated state.
- You can create a new version of the API by clicking the **Create new version** button.
- You can modify details of an API by clicking the **Edit** button. This option is available when the API is in the deactivated state. Only the following properties of an OData API can be modified:
  - Name
  - Description
  - Version
  - API group
  - Maturity state

  For updating the OData entity sets, singletons, function imports and action imports a new import has to be performed.
- You can activate an API by clicking the **Activate** button. If the API is already activated, you can deactivate it by clicking the **Deactivate** button.
- To create an OData API accessing a secure endpoint, you must configure **Administration>Security>Keystore/Truststore** for Outbound connections.

GraphQL API Details

API Gateway supports proxying an existing GraphQL endpoint and provides API management capabilities to clients like authentication, analytics, and so on. The GraphQL APIs can be accessed using the HTTP GET and POST methods. You can create and deploy a GraphQL API using the API Gateway UI or REST endpoints.

The following table lists the features that API Gateway supports for GraphQL.

GraphQL Features	Supported
Basic GraphQL concepts: - Schema	Yes

GraphQL Features	Supported
■ Operations ■ Types	
Root operations for resources: ■ Query ■ Mutation	Yes
Root operations for resources: Subscription	No
Input and output data types: ■ Scalar type ■ Object type ■ Interface type ■ Union type ■ Enum type	Yes

The following table lists the API Gateway-specific features that are not supported for GraphQL API.

API Gateway Features for GraphQL	Supported
API tagging	No
API mocking	No
Policy scopes	No
Packages and plan	No
Adding or updating GraphQL schema types	No
Publishing GraphQL API to API Portal	No

The **API details** view for a GraphQL API displays the details of the API such as basic and technical information, operations available, and specifications. You can also view the policies enforced, registered applications, and the API-specific analytics.

The table lists the API details displayed for the API:

Field	Description
Basic information	Displays the information about the API, such as Name, Version, Owner of the API, the teams that the API is assigned to, Active or Inactive status of the API , the maturity state of the API, the date on which the API was created, and a brief description of the API.
Technical information	Displays the following endpoints of the API: ■ **Native endpoint(s)**. ■ **Gateway endpoint(s)**. Displays these endpoints when the API is deployed to a gateway. ■ **Service Registry display name**. Displays the name of the service registry where the API is deployed.
Operations	Displays a list of operations available in the API sorted alphabetically. Operations are displayed along with their type (Query and Mutation).
Documentation	Displays a list of specifications for the API.

Various tabs displayed in the API details page display the following details:

- The **Policies** tab displays the policies enforced for the API.
- The **Applications** tab displays all the registered applications with the API.
- The **Analytics** tab displays the API-specific analytics for the time interval selected.

You can perform the following operations from the API details page:

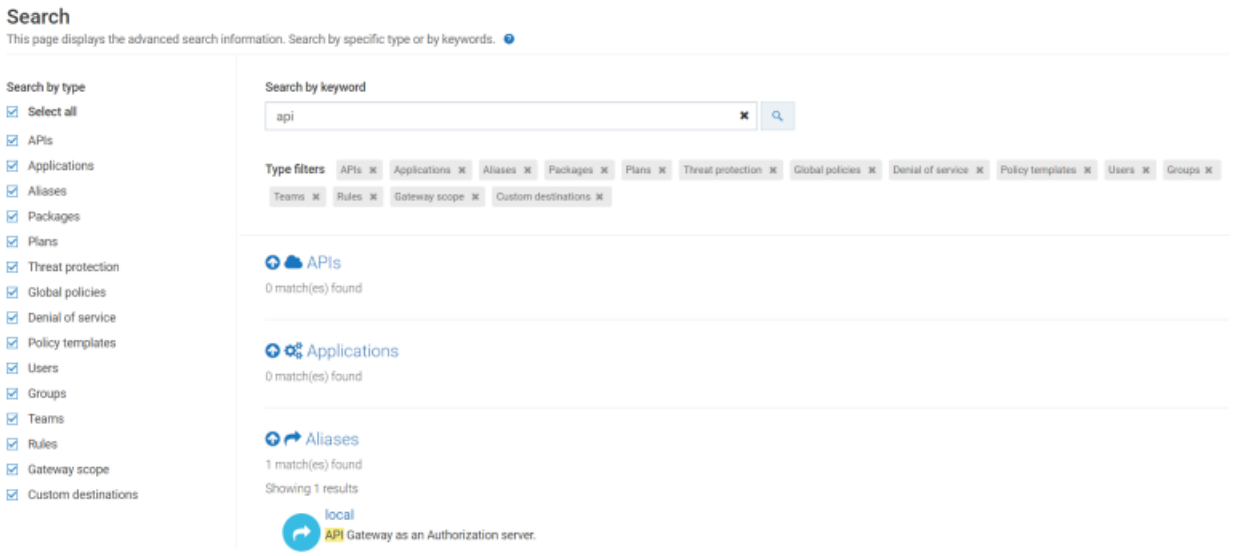
- You can export an API using the **Export** button.
- You can update an API by importing from file or from URL by clicking the **Update** button. This option is available when the API is in the deactivated state. For more information about updating APIs, see “Updating APIs” on page 48.
- You can create a new version of the API by clicking the **Create new version** button. For more information about versioning APIs, see “Versioning APIs” on page 58.
- You can modify details of an API by clicking the **Edit** button. This option is available when the API is in the deactivated state. For more information about modifying API details, see “Modifying API Details” on page 48.
- You can activate an API by clicking the **Activate** button. If the API is already activated, you can deactivate it by clicking the **Deactivate** button.

Searching Data in API Gateway

The search feature in API Gateway is a type-ahead search; a simple and easy to use search facility where you can type the text of interest to search. You can search for all items that contain one or

more specified keywords (that is, text strings) in the item's properties. Some of the properties are name, description, version, key, value, and so on in the API.

You can search for the following types of data as shown in the image.



To search for an item, type a string in the search box in the title navigation bar. A list of search result is displayed directly below the Search box. The number of matches found are displayed in sections depending on the type they fit in. For example, **APIs**, **Application**, **Alias**, **Packages**, and so on. A minimum of five search results are displayed in each category. If there are no results as per the search string typed, a message displays saying so.

If you find what you are searching for in the search result box, click on it to view the details. You are navigated to the specific page that displays more information. For example, if you are searching for an API and click the displayed result, you are navigated to the specified API details page. If you are searching for an application and click the displayed result, you are navigated to the specified Application details page.

If you want to see all the search results click **Show all results** in the search result box. The Advanced search page is displayed. This is a dedicated page that displays extensive search results. In the Advanced search information page, you can search or filter the results in the following ways: by type, or by keyword.

- **By type**: Select one or more types in the **Search by type** section to see search results pertaining to the selected types. For example, if you select the type **APIs**, all the APIs that have the specified string is displayed. By default, all filters are selected. To remove a filter, you can clear the check box next to a filter from the left pane or click ✖ next the filter you want to remove.
- **By keyword**: Type a keyword in the **Search by keyword** field, all the search results containing the specified keyword are displayed in the list. For example, if you type the keyword `petstore`, all search results containing the `petstore` would be filtered and displayed.

Note:

Search by keyword will not show any search results, if the field names have any special characters. The following special characters are not supported - ! ? & # $ * % : ; = ' " ( ) / \ < >

The fields that does not support special characters are as follows:

- Maturity state
- Scope name
- Scope description
- API Operations info name
- API Resource path
- API Tags
- Application identifiers named values
- User Login ID
- User First name
- User Last name
- OAuth scope name
- OAuth Scope description

For example, if an API has a tag name Test-001, and you search APIs with the tag name Test-001, you will not get any search results.

Note:
You cannot search for REST resources and methods in a REST API. The search function only works for the name and description of the REST API. For example, you can search for a REST API named `LibraryAPI`. But you cannot search for a REST resource named `book` or a REST method `POST` within the REST API. However, the search function works for name, description, and operations of SOAP APIs.

You cannot search for resources and methods of an OData API.

There are a few configurable properties available for search. These properties can be configured in the file, uiconfiguration.properties, located at *SAGInstallDir*\profiles\IS_*default*\apigateway\config\. Edit the file as required. After modifying the properties file, you have to restart Integration Server for the changes to take effect.

You must type in a minimum number of characters in the global search box, to search for data. This property can be configured.

The following property is used to configure the minimum number of characters to search. The default value is 3.

apigw.search.minimum.num.chars=3

Note:
The value provided must be a number greater than 0. If you provide an invalid value, it takes up the default value of 3.

The following property is used to configure the number of search results to load for each type in the advanced search page. The default value is 10.

apigw.num.results.search=10

Note:
The value provided must be a number greater than 0. If you provide an invalid value, it takes up the default value of 10.

Filtering APIs

You can filter APIs based on the API type, the activation status, team association or deployment type of the API.

To filter APIs

1. Click **APIs** in the title navigation bar.

   A list of all registered APIs appears.

2. You can filter APIs based on the following filter options. You can use a combination of these options to filter the APIs.

   - **Type**. Select **REST**, **SOAP**, **OData**, **WebSocket**, **GraphQL** or all to filter APIs by type.
   - **Activation status**. Select **Active** or **Inactive** to filter APIs by their activation status.
   - **Team**. This filter is applicable only if you have enabled the Teams feature. Select the teams listed to filter APIs by their association with the teams selected.
   - **Active endpoints**. Select **API Gateway**, **Microgateway**, or **AppMesh** to filter APIs by their active endpoints available.

3. Click **Apply filter**.

   The filtered list of APIs is displayed. You can click **Reset** to reset the values to the original values.

Configuring the Number of APIs listed on a Page

The default number of APIs that are listed in the Manage APIs and Manage applications page can be configured through the properties file located at *SAGInstallDir*\profiles\IS_*default*\apigateway\config\uiconfiguration.properties.

Edit the configuration file as required. You can configure the number of results to load for pagination. The default value is 20. The provided value should be a number greater than 0.

apigw.num.results.pagination=20

You have to restart Integration Server for the changes to take effect.

You can configure the number of APIs that get listed per page in the Manage APIs or the Manage applications page. In each of these pages, you can use the pagination bar at the bottom of the page to navigate from one page to another, the first page, or the last page when there are more than 20 APIs in the list. To change the number of APIs listed in a page, select the required number in the

Show # results per page field in the pagination bar at the bottom of the page. The API list now displays only those many APIs in one page as specified. For example, if you select **Show 10 results per page**, only 10 APIs are listed in one page.

This configuration that you change through the drop down is maintained as long as you are logged in to API Gateway. Once you log out, the value is reset to the default configured value in the uiconfiguration.properties file.

The value is set in the drop down is applicable for both APIs and applications listing. For example, if you change the show results to 10 in the Manage APIs drop down, then the number is retained for Manage applications page as well.

Modifying API Details

You must have the API Gateway's manage APIs or activate/deactivate APIs functional privilege assigned to perform this task.

You can modify API details, as required, from the API details page.

To modify API details

1. Click **APIs** in the title navigation bar.

   A list of all registered APIs appears.

2. Click the required API.

   The API details page appears.

3. Click **Edit**.

   Note:
   If the API is in the active state, you cannot modify the name and version of the API. The API mocking section is unavailable for any changes.

4. Modify the information as required.

5. Click **Save**.

   Note:

   - If the API is in the active state when you modify API details, the active API is replaced with the modified API.
   - The modified APIs do not become effective for ongoing requests.

Updating APIs

You can update the definition of an existing API by uploading WSDL, Swagger, or RAML file or URL. The uploaded file can also be in a ZIP format. When an API is updated, it retains the `Expose`

`to consumers` settings, the existing scope definitions, the configured policies, and the REST-enabled path configurations for SOAP API. You can also edit an API using the **Edit** option for minor edits, whereas the update feature helps you to overwrite the complete API definition using a file or a URL at the same time.

You can update an active API. You cannot modify the name and version of an API while updating an active API.

Note:
The active APIs are replaced with the updated API. The updated APIs do not become effective for ongoing requests. Updates to an activated API are propagated across a cluster and trigger a hot deploy on each cluster node separately.

You can update an existing API in the following ways:

- By importing an API definition from a file
- By importing an API definition from a URL

Updating an API by Importing an API from a File

You must have the API Gateway's manage APIs or Activate/deactivate APIs functional privilege assigned to perform this task. You can not update an API by importing an API from a file if the API is in the active state.

To modify API details

1. Click **APIs** in the title navigation bar.
2. Select the required API from the list of APIs.

   The API details page for the selected API appears.
3. Click and select **Update**.

   The Update API window appears.
4. Select **Update API by importing from file**.
5. Provide the following information:

Field	Description
Select file	Click **Browse** to browse to the location of file to be imported and select the required file or ZIP format file.

Field	Description
	The REST API can be updated using only the Swagger or RAML file type. The SOAP API can be updated using only the WSDL file type.
Root File Name	If you have selected a file in ZIP format, type the relative path of the main file within the ZIP file.
Name	Name for the API. Edit or delete the name of the existing API displayed. ■ If you provide an API name, this overwrites the API name mentioned in the uploaded file and the API is updated with the name provided. ■ If you do not provide an API name, the API name mentioned in the uploaded file is picked up and the API is updated with that name.
Type	Select the required type. The available types are **OpenAPI**, **RAML**, **Swagger**, **WSDL**, and **GraphQL SDL**. ■ For a REST API, the available options are RAML and Swagger. ■ For a SOAP API, the available option is WSDL. ■ For a GraphQL API, the available option is GraphQL SDL.
Version	Version number of the API. The existing version number of the API is automatically displayed. You can edit or delete the version number. If the version number is deleted and the imported file does not have a version number, then the system automatically assigns a version number during the update. This overwrites the version of the API.
Description	Description of the API. The existing description of the API is automatically displayed. You can edit or delete the description. If you delete the description then the description from the imported file is used.

6. Click **Update**.

The API definition is updated with the latest changes from the file and is displayed in the API details page.

Updating an API by Importing an API from a URL

You must have the API Gateway's manage APIs or activate/deactivate APIs functional privilege assigned to perform this task.

To modify API details

1. Click **APIs** in the title navigation bar.
2. Select the required API from the list of APIs.

   The API details page for the selected API appears.
3. Click and select **Update**.

   The Update API window appears.
4. Select **Update API by importing from URL**.
5. Provide the following information:

Field	Description
URL	Type the URL from which the API is being imported. **Note:** The REST API can be updated using only the Swagger or RAML type information that the URL is pointing to. The SOAP API can be updated using only the WSDL file type information that the URL is pointing to. The entity sets, singletons, function imports, and action imports of an OData API can only be updated by a re-import of the OData API definition through the URL.
Protected	Select this option if you want to import an API from a URL that is password protected. The user name and password fields are displayed using which you can access the provided URL.
Username	Type the user name required to access the password protected URL. If you have selected the **Protected** option, this field is displayed.
Password	Type the password associated with the username.

Field	Description
	If you have selected the **Protected** option, this field is displayed.
Name	Name for the API. The existing name of the API is automatically displayed. ■ If you provide an API name, this overwrites the API name mentioned in the file referred by URL and the API is updated with the name provided. ■ If you do not provide an API name, the API name mentioned in the file referred to by URL is picked up and the API is updated with that name.
Type	Select the required type. The available types are **OpenAPI**, **RAML**, **Swagger**, **WSDL**, **OData**, and **GraphQL SDL**. For a REST API, the available options are RAML and Swagger. For a SOAP API, the available option is WSDL. For a OData API, the available option is OData. For a GraphQL API, the available option is GraphQL SDL.
Version	Version number of the API. The existing version number of the API is automatically displayed. You can edit or delete the version number. If the version number is deleted and the file referred to by URL does not have a version number, then the system automatically assigns a version number during the update. This overwrites the version of the API.
Description	Description of the API. The existing description of the API is automatically displayed. You can edit or delete the description. If you delete the description then the description from the file referred to by URL is used.

6. Click **Update**.

   The API definition is updated with the latest changes from the URL and is displayed in the API details page.

Exporting Specifications

For a REST API, you can export specifications in Swagger and RAML formats to your local system. Similarly, for a SOAP API, you can export a specification in WSDL format to your local system.

The exported WSDL is in a ZIP format consisting of the WSDL file whereas for Swagger and RAML the respective files are directly exported. API Gateway supports the following versions:

- Swagger 2.0 for a Swagger file
- RAML 0.8 for a RAML file

You can export APIs that have been created from scratch or by importing their respective definitions. The Swagger or RAML definition provides the consumer view on a REST API deployed to the API Gateway. Similarly, the WSDL definition provides the consumer view on a SOAP API. Consumer view indicates that the Swagger, RAML, or WSDL definitions contain the API Gateway endpoint and information about those resources and operations, which are exposed to customers.

Note:
In the downloaded Swagger document, the valid JSON schemas attached to a response or a request does not always appear. Only the valid JSON schemas appear correctly. For any other schema information just the generic JSON schema such as `{"type":"object"}` appears.

To export the specification

1. Click **APIs** in the title navigation bar.
2. Select the required API from the list of available APIs .

   The API details page for the selected API appears.
3. Click **Documentation**.
4. Based on the type of specification that you have selected to export, select any of the following:
   - **Swagger data** link to export the Swagger specification.
   - **RAML data** link to export the RAML specification.
   - **OpenAPI data** link to export the OpenAPI specification.
   - **Artifacts** link to export the WSDL specification.
   - **OData meta document** link to a zip containing the OData API and metadata document. If the OData API is active, a link to the service document and a link to the metadata document are also displayed.
   - **Schema** link to download the GraphQL schema.
5. Select the appropriate option and click **OK**.

Attaching Documents to an API

Pre-requisites:

You must have the Manage APIs functional privilege assigned to perform this task.

You can associate an input document that includes the RAML, Swagger, or WSDL specification, and additional documents such as programming guides, sample code, script files, and project plan with an API. For example, SOAP APIs can contain external documents such as Functional Requirements, Error Messages, Release Notes, and so on.

When attaching a document to an API, keep the following points in mind:

- You cannot attach or modify a document to the API if it is in active state. You have to deactivate the API before attaching or modifying it.
- API Gateway relies on file extensions to determine a file's type. When you upload a file from your local machine to the API, be sure the name of the file on your local machine includes a file extension so that API Gateway can determine the file's type and attach it correctly to the API.
- You cannot upload types of files that are restricted for attaching as the input document to the API.

  API Gateway provides the ability to restrict certain kinds of files from being uploaded to the API, based on the file extension. The list of restricted files may vary depending on the file extensions configured in the `apiDocumentsRestrictedExtension` property under **Administration > Extended settings** section.

  When you try to upload a file type that is restricted, API Gateway prompts you with an error message.
- By default, several standard file extensions are blocked in API Gateway, including any file extensions that are treated as executable files by Windows Explorer. The file extensions blocked by default are - `.bat`, `.bin`, `.dll`, and `.exe`.
- You cannot upload files that exceed the maximum allowed size for the API.

  API Gateway provides the ability to limit the maximum file upload size to the API. The maximum file upload size is configured in the `apiDocumentsUploadSizeLimitInMB` property under **Administration > Extended settings** section.

  When you try to upload a file that exceeds the maximum file upload size, API Gateway prompts you with an error message.
- You can rename an uploaded document. When you rename a document, you can only modify the display name of the document and not the document itself. If you want to modify the document as well, you must delete the file attachment, and attach the latest file.

To attach a document

1. Click **APIs** in the title navigation bar.

   A list of all registered APIs appears.

2. Select the required API.

   The API details page appears.

3. Click **Edit**.
4. Click **Documentation**.
5. Click **Browse** to select a file and upload it.
6. Rename the document in the **Display name** field as required.

   This is the display name of the document in the API details page.
7. Click **Add**.

   The attached document is listed in a table. You can edit and delete the document by clicking the  and  icons.
8. Repeat steps 5 to 7 for each document that you want to attach to the API.
9. Click **Save**.

API Grouping

You can group APIs based on various categories. Categories help consumers locate APIs easily. For example, if you are offering APIs to help your consumers manage their sales and ordering better, classifying the APIs under Sales and Ordering helps them locate these APIs easily.

The default groups available under which you can group the APIs are **Finance Banking and Insurance**, **Sales and Ordering**, **Search**, and **Transportation and Warehousing**. If you want to include more groups you can update the property `apiGroupingPossibleValues` under **Administration > Extended settings** that enables API grouping. You can modify the existing list of groups by deleting or adding new group names as comma separated values in this field. Ensure that the group name does not contain a comma as part of the name.

API grouping can be applied in one of these ways:

- While creating an API from scratch
- While editing an API

You can select one or more groups in the **API grouping** field. When an API is published to API Portal, the published APIs in API Portal are grouped as per the group assigned.

API Tagging

Tags are words or phrases that act as keywords for categorizing, identifying, and organizing APIs.

In API Gateway, you can assign tags to APIs, and their resources, methods, or operations. Tags help to logically categorize APIs in different ways, for example, by usage, owner, consuming application, or other criteria. Tags are especially useful when there are multiple APIs of the same

type - it enables to quickly identify a specific API based on the tag assigned to it. For example, you can assign the tag `GET-Methods` to specific GET methods in different REST APIs, and use it to search for the list of REST APIs with the `GET-Methods` tag in API Gateway.

You can use tagging, for example, to do the following:

- Tag and untag REST APIs in API Gateway.
- Use tags to search for multiple resources and methods across the REST APIs that are available in API Gateway.
- Use tags to search for multiple operations across the SOAP APIs that are available in API Gateway.

You can assign one or more tags, remove a tag, and view the tags on the API details page. When a tagged API is published to API Portal, the published API in API Portal is tagged with the same tag defined in API Gateway.

Adding Tags to an API

You must have the Manage APIs functional privilege assigned to perform this task.

Tags are not automatically assigned to APIs, resources, methods, or operations. You can add one or more tags, and you can remove tags from an API, resource, method, or operation at any time.

You can define a set of consistent tags that meets your needs for each API, resource, method, or operation. Using a consistent set of tags makes it easier to manage the APIs, resources, methods, or operations. You can search the APIs, resources, methods, or operations based on the tags you add. To add an existing tag, you can use the typeahead search support that lists the existing tags, which match the character you type. You can restrict the number of existing tags that display, which match the typeahead character you provide, by configuring the extended setting `tagsTypeAheadSearchResultSize` in the **Administration** > **General** > **Extended settings** section. For details about configuring extended settings, see *webMethods API Gateway Administration*.

When tagging an API, keep the following points in mind:

- You can assign tags to the following API types and their components:
  - **SOAP API**. You can assign tags to the SOAP API and to its operations.
  - **REST API**. You can assign tags to the REST API, and to its resources and methods.
  - **REST-enabled SOAP API**. You can assign tags to the REST-enabled SOAP API. Also, you can assign tags to the REST resources and methods which correspond to the transformed SOAP operations.
  - **OData API**. You can assign tags to the OData API only.
  - **WebSocket API**. You can assign tags to the WebSocket API only.
- When you delete an API, resource, method, or operation in API Gateway, any tags that were assigned to that API, resource, method, or operation are not deleted.

To tag an API

1. Click **APIs** in the title navigation bar.

   A list of all registered APIs appears.

2. Select the required API.

   The API details page appears.

3. Click **Edit**.

4. To add tags to an API, in the Basic information section, do one of the following:

   - To add an existing tag, select an existing tag from the drop-down list and click  .

     Alternatively, you can search an existing tag by typing characters in the **Tags** field that displays a list of existing tags that contain the character, select the required tag, and click  .

   - To add a new tag, type the new tag and click  .

   The tag is listed below the **Tags** field. To delete a tag, click the **x** icon.

5. To add tags to resources or methods of a REST API, in the Resources and methods section, locate the required resource or method and do one of the following:

   - To add an existing tag, select an existing tag from the drop-down list and click  .

     Alternatively, you can search an existing tag by typing characters in the **Tags** field that displays a list of existing tags that contain the character, select the required tag, and click  .

   - To add a new tag, type the new tag and click  .

   The tag is listed below the **Tags** field. To delete a tag, click the **x** icon.

6. To add tags to an operation of a SOAP API, in the Operations section, locate the required operation and do one of the following:

   - To add an existing tag, select an existing tag from the drop-down list and click  .

Alternatively, you can search an existing tag by typing characters in the **Tags** field that displays a list of existing tags that contain the character, select the required tag, and click 

.

- To add a new tag, type the new tag and click 

.

The tag is listed below the **Tags** field. To delete a tag, click the **x** icon.

7. Click **Save**.

Versioning APIs

API Gateway supports the creation of new API versions from the existing versions. The new API has the same metadata but with an updated version. The version can either be a number or a string.

The API details page has a drop-down list that displays all the existing API versions. You can create a new version of an API and retain applications that are associated with older versions of the API. When an API is updated, it retains the `Expose to consumers` settings, the existing scope definitions, the configured policies, and the REST-enabled path configurations for SOAP API.

When you create a new version, the newer version is assigned to the teams of the older version by default. You can later change the teams, if required.

Creating New API Version

You must have the API Gateway's manage APIs functional privilege assigned to perform this task.

You can create a new version of an API from the latest version available for the API. For example, if the existing version is 1.1 for an API, you can create a version 1.2. If you want to create a version 1.3, you can only create it from the latest version 1.2 and not from 1.1. However, you can delete the intermediate versions. Additionally, even though the owner of the older API version is a different provider, when you create a new version of the API, you are the owner of the newly created version of the API. The new API version is in inactive state, irrespective of the state of the API from which it was versioned.

To create a new version

1. Click **APIs** in the title navigation bar.

2. Select the required API from the list of APIs.

   The API details page for the selected API appears.

3. Click and select **Create new version**.

4. In the **Version** field, type the new version for the API.

5. Clear the **Retain applications** checkbox if you do not want to retain applications that are associated with older versions of the API.

6. Click **Create**.

The **Version** drop-down lists the newly created API version in latest to older order in the API details page. The corresponding API details page is displayed when you select any particular version.

Note:
The version is appended to the **Gateway endpoint(s)** URL once the API is activated and this can be seen in the **Technical information** section of the API details page. When a client application invokes the API without the version in the endpoint, API Gateway invokes the latest version.

Deleting APIs

Deleting an API permanently removes the API from API Gateway.

When deleting an API, keep the following points in mind:

- You cannot delete an API if it is in active state. You have to deactivate the API before deleting it.
- You must have the Manage APIs functional privilege.

Deleting a Single API

You must have the Manage APIs functional privilege assigned to perform this task.

You delete an API to remove it from API Gateway permanently.

To delete an API

1. Click **APIs** in the title navigation bar.

   A list of all APIs appears.

2. Click the **Delete** icon for the API that you want to delete.

3. Select the **Force delete** option to delete an API forcefully.

   API Gateway ignores any failures even if the API is used by other applications, and clears all data from the API Gateway database.

4. Click **Yes** in the confirmation dialog.

The API is deleted forcefully.

Deleting Multiple APIs in a Single Operation

You must have the Manage APIs functional privilege assigned to perform this task.

You can bulk delete APIs in API Gateway.

To delete multiple APIs in a single operation

1. Click **APIs** in the title navigation bar.

   A list of all APIs appears.

2. Select the APIs that you want to delete.

3. In the **Menu** icon, click **Delete**.

4. Select the **Force delete** option to delete APIs forcefully.

   API Gateway ignores any failures even if the selected APIs are used by other applications, and clears all data from the API Gateway database.

5. Click **Yes** in the confirmation dialog.

   The APIs are deleted forcefully from API Gateway.

6. Examine the **Delete APIs report** window and check for any errors that occurred during the deletion process.

   The **Delete APIs report** window displays the following information:

Parameter	Description
Name	The name of the deleted API.
Status	The status of the deletion process. The available values are: ■ Success ■ Failure
Description	A descriptive information if the deletion fails or if a warning occurs.

   API Gateway writes these results to the **Audit logs** dashboard, so you can view them later.

7. Click **Download the detailed report here** to download the detailed report as an HTML file.

Example: Managing an API

This section explains everything you would want to know about an API and how to manage it with an example API phonestore. You can model an API that serves to expose API data and functionality as a collection of resources. Each resource is accessible with unique Uniform Resource Identifiers (URIs). In your API, you expose a set of HTTP operations (methods) to perform on a specific resource and capture the request and response messages and status codes that are unique to the HTTP method and linked within the specific resource of the API.

The basic elements of an API are:

- The API itself (for example, phonestore)
- Its resource (*phones*), available on the unique base URL (*/phones*)
- The defined HTTP method (*GET*) for accessing the resource (*phones*)
- Parameters for request representations (*412456*)
- A request generated for this method (*Request 123*)
- A response with the status code received for this request (*Response ABCD*)

The example API phonestore considered here is defined to support an online phone store application. Assume, this sample phonestore API currently has a database that defines the various brands of phones, features in the individual phones, and the inventory of each phone. This API is used as a sample to illustrate how to model URL patterns for resources, resource methods, HTTP headers and response codes, content types, and parameters for request representations to resources.

Base URL

The base URL of an API is constructed by the domain, port, and context mappings of the API. For example, if the server name is www.phonestore.com, port is 8080, and the API context is api. The full Base URL is:

```
http://www.phonestore.com:8080/api
```

API Parameters

Parameters defined at the higher API level are inherited by all resources and methods included in the individual resources.

API Resources

Resources are the basic components of an API. Examples of resources from an online phonestore API include a phone, an order from a store, and a collection of customers. After you identify a service to expose as an API, you define the resources for the API.

For example, for the online phonestore API, there are a number of ways to represent the data in the phone store database as an API. The verbs in the HTTP request maps to the operations that the database supports, such as select, create, update, delete.

Each resource has to be addressed by a unique URI. Along with the URI you're going to expose for each resource, you also need to decide what can be done to each resource. The HTTP methods

passed as part of an HTTP request header directs the API on what has to be done with the addressed resource.

Resource URLs

An URL identifies the location of a specific resource.

For example, for the online phonestore API, the resources have the following URLs:

URL	Description
`http://www.phonestore.com/api/phones`	Specifies the collection of phones contained in the online store.
`http://www.phonestore.com/api/phones/412456`	Accesses a phone referenced by the product code `412456`.
`http://www.phonestore.com/api/phones/412456/reviews`	Specifies a set of reviews posted for a phone of code `412456`.
`http://www.phonestore.com/api/phones/412456/reviews/78`	Accesses a specific review referenced by the unique ID `78` contained in the reviews of the phone of code `412456`.

API Gateway supports the following patterns of resource URL: a collection of resources or a particular resource.

For example, in the online phonestore API, the patterns are as follows:

Collection URL: `http://phonestore.com/api/phones`

Unique URL: `http://phonestore.com/api/phones/412456/features` to retrieve a collection resource describing the key features of phone whose product code is `412456`.

Resource Parameters

Parameters defined at the higher resource level are inherited by all methods in the particular resource; it does not affect the API.

Resource Methods

Individual resources can define their capabilities using supported HTTP methods. To invoke an API, the client would call an HTTP operation on the URL associated with the API's resource. For example, to retrieve the key feature information for phone whose product code is `412456`, the client would make a service call HTTP GET on the following URL:

```
http://www.phonestore.com/phones/412456/features
```

Supported HTTP Methods

API Gateway supports the standard HTTP methods for modeling APIs: GET, POST, PUT, DELETE, and PATCH.

The following table describes the semantics of HTTP methods for the sample Phone Store API:

Resource URI	HTTP Method	Description
`/phones/orders`	GET	Asks for a representation of all of the orders.
`/phones/orders`	POST	Attempts to create a new order, returning the location (in the Location HTTP Header) of the newly created resource.
`/phones/orders/{order-id}`	GET	Asks for a representation of a specific Order resource.
`/phones/orders/{order-id}`	DELETE	Requests the deletion of a specified Order resource.
`/phones/orders/{order-id}/status`	GET	Asks for a representation of a specific Order's current status.
`/phones/orders/{order-id}/paymentdetails`	GET	Asks for a representation of a specific Order's payment details.
`/phones/orders/{order-id}/paymentdetails`	PUT	Updates a specific Order's payment details

Method Parameters

Parameters defined at the lower method level apply only to that particular method; it does not affect either the API or the resource.

API Parameters

Parameters specify additional information to a request. You use parameters as part of the URL or in the headers or as components of a message body.

Parameter Levels

A parameter can be set at different levels of an API. When you document a REST API in API Gateway, you define parameters at the API level, resource level, or method level to address the following scenarios:

- If you have the parameter applicable to all resources in the API, then you define this parameter at the API level. This indirectly implies that the parameter is propagated to all resources and methods under the particular API.
- If you have the parameter applicable to all methods in the API, then you define this parameter at the resource level. This indirectly implies that the parameter is propagated to all methods under the particular resource.
- If you have the parameter applicable only to a method in the API, then you define this parameter at the method level.

API-level Parameters

Setting parameters at the API level enables the automatic assignment of the parameters to all resources and methods included in the API. Any parameter value you specify at the higher API level overrides the parameter value you set at the lower resource level or the lower method level if the parameter names are the same.

For example, if you have a header parameter called API Key that is used for consuming an API.

```
x-Gateway-APIKey:a4b5d569-2450-11e3-b3fc-b5a70ab4288a
```

This parameter is specific to the entire API and to the individual components, that is resources and methods, directly below the API. Such a parameter can be defined as a parameter at the API level.

At an API level, API Gateway allows you to define the following types of parameters:

- Query-String parameter
- Header parameter

Resource-level Parameters

Setting parameters at the resource level enables the automatic assignment of the parameters to all methods within the resource. Any parameter value you specify at the higher resource level overrides the parameter value you set at the lower method level if the parameter names are the same. In contrast, the lower resource level parameters do not affect the higher API level parameters.

Consider the sample phonestore API maintains a database of reviews about different phones. Here is a request to display information about a particular user review, 78 of the phone whose product code is `412456`.

```
GET /phones/412456/user_reviews/78
```

In the example, `/user_reviews/78` parameter narrows the focus of a GET request to review `/78` within a particular resource `/412456`.

This parameter is specific to the particular resource phone whose product code is `412456` and to any individual methods that are directly below the particular resource. Such a parameter can be defined as a parameter at the resource level.

At a resource level, API Gateway allows you to define the following types of parameters:

- Query-String parameter
- Header parameter
- Path parameter

Method-level Parameters

If you do not set parameters at the API level or resource level, you can set them at a method level. Parameters you set at the method level are used for the HTTP method execution. They are useful to restrict the response data returned for a HTTP request. Any parameter value you specify at the lower method level is overridden by the value set at higher API-level parameter or the higher

resource-level parameter if the names are the same. In contrast, the lower method-level parameters do not affect the higher API-level or resource-level parameters.

For example, the phonestore API described might have a request to display information contributed by user `Allen` in `2013` about a phone whose product code is `412456`.

```
GET /phones/412456/user_reviews/78?year=2013&name=Allen
```

In this example, `year=2013` and `name=Allen` narrow the focus of the GET request to entries that user `Allen` added to user review `78` in `2013`.

At a method level, API Gateway allows you to define the following types of parameters:

- Query-String parameter
- Header parameter

Parameter Types

API Gateway supports three types of parameters in REST API: Query-String, Header, and Path.

The following example explains how you can use different parameter types for parameterizing the resources.

Query-String Parameters

Query-String parameters are appended to the URI after a `?` with name-value pairs. The name-value pairs sequence is separated either by a semicolon or an ampersand.

For instance, if the URL is `http://phonestore.com/api/phones?itemID=itemIDValue`, the query parameter name is `itemID` and value is the `itemIDValue`. Query parameters are often used when filtering or paging through HTTP GET requests.

Now, consider the online phonestore API. A customer, when trying to fetch a collection of phones, might wish to add options, such as, `android v4.3 OS` and `8MP` camera. The URI for this resource would look like:

```
/phones?features=androidosv4.3&cameraresolution=8MP
```

You can also use query string to invoke the required resource of an API by appending API Key to `?` like the example seen below:

```
http://pie-3HKYMH2:5555/gateway/PetstoreAPI/1.0.3/store/inventory?APIKey=faab7ac6-97a4-4228-908d-f1930faba470
```

Header Parameters

Header parameters are HTTP headers. Headers often contain metadata information for the client, or server.

```
x-Gateway-APIKey:a4b5d569-2450-11e3-b3fc-b5a70ab4288a
```

You can create custom headers, as required. As a best practice, Software AG recommends that you prefix the header name with `x-`.

HTTP/1.1 defines the headers that can appear in a HTTP response in three sections of RFC 2616: 4.5, 6.2, and 7.1. Examine these codes to determine which are appropriate for the API.

Path Parameters

Path parameters are defined as part of the resource URI. For example, the URI can include `phones/item`, where `/item` is a path parameter that identifies the item in the collection of resource `/phones`. Because path parameters are part of the URI, they are essential in identifying the request.

Now, consider the online phonestore API. A customer wishes to fetch details about a phone `{phone-id}` whose product code is `412456`. The URI for this resource would look like: `/phones/412456`

Important:
As a best practice, Software AG recommends that you adopt the following conventions when specifying a path parameter in the resource URI:

- Append a path parameter variable within curly `{}` brackets.
- Specify a path parameter variable such that it exactly matches the path parameter defined at the resource level.

Parameter Data Types

When you add a parameter to the API, you specify the parameter's data type. The data type determines what kind of information the parameter can hold.

API Gateway supports the following data types for parameters:

Data Type	Description
String	Specifies a string of text.
Date	Specifies a date stamp that represents a specific date. The date input parameters allow year, month, and day input. This data type only accepts date values in the format `yyyy-mm-dd`
Time	Specifies a timestamp that represents a specific time. The time input parameters allow hour and minute. This data type only accepts date values in the format `hh:mm:ss`
Date/Time	Specifies a timestamp that represents a specific date and/or time. The date/time input parameters allow year, month, and day input as well as hour and minute. Hour and minute default to 0. This data type only accepts date values in the format `yyyy-mm-dd`; and time values in the format `hh:mm:ss`
Integer	Specifies an integer value for the data type. This is generally used as the default data type for integral values.
Double	Specifies the double data type value.

Data Type	Description
	This is a double-precision 64-bit IEEE 754 floating point and is generally used as the default data type for decimal values.
Boolean	Specifies a `true` or `false` value.

Supported HTTP Status Codes

An API response returns a HTTP status code that indicates success or failure of the requested operation.

API Gateway allows you to specify HTTP codes for each method to help clients understand the response. While responses can contain an error code in XML or other format, clients can quickly and more easily understand an HTTP response status code. The HTTP specification defines several status codes that are typically understood by clients.

API Gateway includes a set of predefined content types that are classified in the following taxonomy categories:

Category	Description
1xx	Informational.
2xx	Success.
3xx	Redirection. Need further action.
4xx	Client error. Correct the request data and retry.
5xx	Server error.

HTTP/1.1 defines all the legal status codes. Examine these codes to determine which are appropriate for your API.

Now, consider the online phonestore API. The following table describes the HTTP status codes that each of the URIs and HTTP methods combinations will respond:

Resource URI	Supported HTTP Methods	Supported HTTP Status Codes
`/phones/orders`	GET	200 (OK, Success)
`/phones/orders`	POST	201 (Created) if the Order resource is successfully created, in addition to a Location header that contains the link to the newly created Order resource; 406 (Not Acceptable) if the format of the incoming data for the new resource is not valid

Resource URI	Supported HTTP Methods	Supported HTTP Status Codes
/phones/orders/{order-id}	GET	200 (OK); 404 (Not Found) if Order Resource not found
/phones/orders/{order-id}	DELETE	200 (OK); 404 (Not Found) if Order Resource not found
/phones/orders/{order-id}/status	GET	200 (OK); 404 (Not Found) if Order Resource not found
/phones/orders/{order-id}/paymentdetails	GET	200 (OK); 404 (Not Found) if Order Resource not found
/phones/orders/{order-id}/paymentdetails	PUT	201 (Created); 406 (Not Acceptable) if there is a problem with the format of the incoming data on the new payment details; 404 (Not Found) if Order Resource not found

Sample Requests and Responses

To illustrate the usage of an API, you provide a sample request and response messages. Consider the sample phonestore API that maintains a database of phones in different brands. The phonestore API might provide the following examples to illustrate its usage:

Sample 1 - Retrieve a list of phones

Client Request

```
GET /phones HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.api.phonestore.com
Accept-Language: en-us
Accept-Encoding: text/xml
Connection: Keep-Alive
```

Server Response

```
HTTP/1.1 200 OK
Date: Mon, 29 August 11:53:27 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Mon, 18 July 2016 09:18:16 GMT
Content-Length: 356
Content-Type: text/xml
<phones>
   <phone>
       <name>Asha</name>
       <brand>Nokia</brand>
       <price currency="irs">11499</price>
       <features>
           <camera>
               <back>3</back>
           </camera>
```

```
            <memory>
                <storage scale="gb">8</storage>
                <ram scale="gb">1</ram>
            </memory>
            <network>
                <gsm>850/900/1800/1900 MHz</gsm>
            </network>
        </features>
    </phone>
    <phone>
        <name>Nexus7</name>
        <brand>Google</brand>
        <price currency="irs">16499</price>
        <features>
            <camera>
                <front>1.3</front>
                <back>5</back>
            </camera>
            <memory>
                <storage scale="gb">16</storage>
                <ram scale="gb">2</ram>
            </memory>
            <network>
                <gsm>850/900/1800/1900 MHz</gsm>
                <HSPA>850/900/1900 MHz</HSPA>
            </network>
        </features>
    </phone>
</phones>
```

Client Request

```
GET /phones/phone-4156 HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.api.phonestore.com
Accept-Language: en-us
Accept-Encoding: text/xml
Connection: Keep-Alive
```

Server Response

```
POST /phones/phone HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.api.phonestore.com
Accept-Language: en-us
Accept-Encoding: text/xml
Content-Length: 156
Connection: Keep-Alive
<phones>
   <phone>
       <name>iPhone5</name>
       <brand>Apple</brand>
       <price currency="irs">24500</price>
       <features>
           <camera>
               <front>1.2</front>
               <back>8</back>
           </camera>
           <memory>
               <storage scale="gb">32</storage>
```

```
            <ram scale="gb">2</ram>
        </memory>
        <network>
            <gsm>850/900/1800/1900 MHz</gsm>
            <HSPA>850/900/1900 MHz</HSPA>
        </network>
    </features>
  <phone>
</phones>
```

Sample 3 - Create a phone

```
POST /phones/phone HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.api.phonestore.com
Accept-Language: en-us
Accept-Encoding: text/xml
Content-Length: 156
Connection: Keep-Alive
<phones>
   <phone>
       <name>iPhone5</name>
       <brand>Apple</brand>
       <price currency="irs">24500</price>
       <features>
           <camera>
               <front>1.2</front>
               <back>8</back>
           </camera>
           <memory>
               <storage scale="gb">32</storage>
               <ram scale="gb">2</ram>
           </memory>
           <network>
               <gsm>850/900/1800/1900 MHz</gsm>
               <HSPA>850/900/1900 MHz</HSPA>
           </network>
       </features>
   <phone>
</phones>
```

Server Response

```
HTTP/1.1 200 OK
Date: Mon, 29 August 11:53:27 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 18 June 2014 09:18:16 GMT
Content-Type: text/xml
Content-Length: 15
<id>2122</id>
```

CentraSite Provided APIs

When you want to perform governed API development with CentraSite and API Gateway, you can create an API in CentraSite defining the design-time aspects. The API can be deployed to the API Gateway. In such cases, you can also see that particular CentraSite destination is being configured in the API Gateway. The API details for the CentraSite provided APIs are set as read-only. However, you can edit the run-time aspects such as scope and policies.

Note:
When you remove the CentraSite destination from the API Gateway, this implies that the API is provided by the API Gateway and therefore the details of API are not read-only. You can edit them as required.

When you deploy

- A REST API from CentraSite, then you cannot modify the **Basic information**, **Technical information**, **Resource and methods**, and **Components** sections. The above mentioned sections are marked as read-only. However, you can modify the fields in the **API Mocking** and **Documentation** sections.
- A SOAP API from CentraSite, then you cannot modify the **Basic information**, **Technical information**, **Resource and methods**, and **Components** sections. The above mentioned sections are marked as read-only. However, you can modify the fields in the **REST transformation**,**API Mocking**, and **Documentation** sections.
- An OData API from CentraSite, then you cannot modify the **Basic information**and **Technical information** sections. The above mentioned sections are marked as read-only. However, you can modify the fields in the **Documentation** section.

For more information about Modifying API, see "Modifying API Details" on page 48.

2 Implement APIs

API Implementation

After you create an API, it's time to start defining the behavior of the API.

Some of the important considerations that you take into account when you define your API's behavior are:

- Enable API mocking
- Enforce policies
- Ensure proper caching, filtering and error handling mechanisms
- Enforce policies
- Apply rate limiting
- Enable API testing and debugging
- Define monitoring and debugging mechanisms

API Gateway UI provides a visual guided experience for designing, developing, and testing APIs. The API caching, filtering, and sorting capability helps in retrieving your APIs faster. You can use the pagination function to determine how much data must be displayed and at what frequency. These features ensure minimum processing and good response time.

API Gateway provides various policies that are designed to let you add management capabilities, to an API or to all APIs at a global level, easily. You can enforce these policies on an API to secure, limit, and route requests sent to APIs.

The API testing capability allows you to test your APIs before you publish them onto a portal for consumption. API Gateway also provides the API mocking capability that enables you to simulate the behavior of a real API for testing the API.

API Gateway provides logging and monitoring capabilities that help debug your APIs so that locating the root cause of the issue based on what is observed is easy. In addition, you can monitor API usage to understand API trends.

The following sections describe the various ways in which you can implement your APIs such as, API mocking, enforcing different types of policies as required, creating and managing applications and aliases, creating API mashup, and so on.

API Mocking

Using API Gateway, you can mock an API by simulating the native API. For example, if you have an API without a native implementation, you can mock that API. The mocked response is returned to the consumer when the API is invoked.

In API Gateway, when you enable mocking for an API, a default mock response is configured for each combination of resource, operation, status code, and content-type based on the example and schema specified in that API. You can add a condition to the operation in the resource.

Note:

- You cannot enable or disable mocking for an active API.
- API Gateway does not support API Mocking for GraphQL API.

As an API Provider, you can create or modify the default mock response. You can specify conditions and associate an IS service with the mocked API. When an IS service is associated with a mocked API, the associated IS service must adhere to the *apigateway.specifications:mockService* specification.

At runtime, when the mocked API is invoked, instead of calling the native API, API Gateway returns the mocked response to the consumer based on the following priorities:

1. If any of the conditions for the invoked operation satisfies, API Gateway returns the associated mocked response.
2. If any of the conditions for the invoked operation is not satisfied, and if an IS service is configured for the API, then API Gateway invokes the IS service and returns the IS service response.
3. If any of the conditions for the invoked operation is not satisfied, and if an IS service is not configured for the API, then API Gateway returns the default mocked response.

API mocking is supported only for SOAP and REST APIs.

Note:
You must have the API Gateway's manage APIs or activate/deactivate APIs functional privilege assigned to perform API mocking.

Enabling API Mocking

You can enable or disable API mocking through the API details page.

Note:
You cannot enable or disable API mocking for active APIs.

To enable API mocking

1. Click **APIs** in the title navigation bar.
2. Select the required API from the list of available APIs.

   The API details page for the selected API appears.
3. Click and select **Enable mocking**.

   This generates the default mock responses.

Modifying API Mocking Details

You must select **Enable mocking** from the API details page.

You can modify API mocking details, as required, from the API details page.

To modify API mocking details

1. Click **APIs** in the title navigation bar.

   A list of all registered APIs appears.

2. Click the required API.

   The API details page appears.

3. Click **Edit**.

4. Click **API mocking**.

5. Specify the following information in the ESB service section:

Field	Description
Invoke service	Specifies the webMethods Integration Server service to be invoked. **Note:** The webMethods Integration Server service must be running on the same Integration Server as API Gateway.
Run as user	Type the user name you want API Gateway to use to invoke the IS service.

6. Select the operation that you want to modify from the Mocked responses section.

7. Click **Add Response** if you want to add a response and select the status code from the drop-down.

8. Click 

.

   This adds the status code created to the existing status code list.

9. Select the status code you want to modify.

10. Click **+ Add Response Header** and provide the following information to add the required response headers:

Field	Description
Header key	Specify the HTTP header key that would be contained in the header of HTTP response.
Header value	Specify the HTTP header value that would be contained in the header of HTTP response.

You can add more response headers by clicking 

.

11. Click **+ Add Content-type** to add a content-type to the status code selected and provide the following information:

Field	Description
Content type	Select the content-type to be added to the selected status code from the drop-down list.
Mock payloads	Specify a mock response payload for the content-type selected.

You can add more content-types by clicking **Add** .

12. Click **+ Add Conditions** to add a condition to the operation in the resource:

Field	Description
Condition name	Specify the name for the condition.
Condition parameter	Select the type to which the condition is to be applied. The available options are: ■ **Body** ■ **Header** ■ **Query parameter** (Applicable only for REST APIs)
Key	The key can be a string for the header and query parameter and for body it can be a JSON path or an XML path. **Note:** The XML path must not contain namespace prefixes.
Value	The value of the condition. Additionally, you can type an * (asterisk) to ignore the value specified in this parameter and

Field	Description
	the condition is satisfied based on the value specified in the **Key** parameter.
Status code	Select the status code from the drop-down list. **Note:** ■ You must enable the property **Send native provider fault** in the **Administration > General > API fault** section in order to have correct mock response for status code 506. ■ This field is not applicable for APIs when they participate in API mashups. ■ While invoking an API remember to use the query parameter `expectedStatusCode` in order to have correct mock responses for status codes 100 and 202.
+ Add Response Header	Add a response header to the resource selected by providing the following information: ■ **Header key**. Specify the HTTP header key that would be contained in the header of HTTP response ■ **Header value**. Specify the HTTP header value that would be contained in the header of HTTP response.
+ Add Content-type	Add a content-type to the status code selected by providing the following information: ■ **Content type**. Select the content-type to be added to the selected status code from the drop-down list. ■ **Mock payload**. Specify a mock response payload for the content-type selected.

You can add more conditions by clicking **Add** .

13. Click **Save**.

Custom Replacer

API Gateway allows you to send a dynamic custom response instead of a static mocked response to the consumer when the mocked API is invoked. In the mocked response, you can specify multiple custom replacers. Custom replacer is used to replace the custom variables with the values defined in the request headers, query parameters, and request body. The custom replacer is available in the `${request.ConditionParameter.Key|JsonPath|XPath}` format. The custom replacers are:

- `${request.header.headerKey}`: To replace the value of the `headerKey` from the request headers.

- `${request.query.queryKey}`: To replace the value of the `queryKey` from the query parameters in the request URL.
- `${request.body.JsonPath|XPath}`: To replace the value of the `JsonPath|XPath` from the request body.

Consumer Applications

A consumer application defines the precise identifiers by which messages from a particular application is recognized at run time. The identifiers can be, for example, user name in HTTP headers, a range of IP addresses, such that API Gateway can identify or authenticate the applications that are requesting an API.

The ability of API Gateway to relate a message to a specific application enables it to:

- Control access to an API at run time (that is, allow only authorized applications to invoke an API).
- Monitor an API for violations of a Service-Level Agreement (SLA) for a specified application.
- Indicate the application to which a logged transaction event belongs.

An application has the following attributes for specifying the identifiers:

- IP address, which specifies one or more IP addresses that identify requests from a particular application. Example: `192.168.0.10`

  This attribute is queried when the Identify & Authorize policy is configured to identify applications using IP address.
- Claims set, which specifies one or more claims that identify requests from a particular application. The claims are a set of name-value pairs that provide sufficient information about the application. Example: `sub = Administrator`.

  This attribute is queried when the Identify & Authorize policy is configured to identify applications using a JWT token or an OpenID token.
- Client certificate, which specifies the X.509 certificates that identify requests from a particular application.

  This attribute is queried when the Identify & Authorize policy is configured to identify the applications by a client certificate.
- Identification token, which specifies the host names, user names or other distinguishing strings that identify requests from a particular application.

  This attribute is queried when the Identify & Authorize policy action is configured to identify applications by host name, token, HTTP user name, and WSS user name.

You can configure various authentication strategies to authenticate an incoming request to the application. You can create multiple strategies authorized by an API for an application. These strategies provide multiple authentication mechanisms or multiple authorization servers for a single authentication scheme. For example, in case of OAuth authentication scheme, you want the

application to support both OKTA and PINGFederate or OKTA with multiple tenants. This can be configured as OAuth strategy for the application.

If you have the **Manage Application** functional privilege assigned, you can create and manage applications, and register applications with the APIs.

These are the high level stages of managing and using an application:

1. API developer requests the API Gateway administrator to create an application for access as per the required identification criteria.
2. API Gateway provider or administrator validates the request and creates a new application, there by provisioning the application specific access tokens (API access key and OAuth credentials).
3. API developer, upon finding a suitable API, sends a request to API Gateway for consumption by providing the application details.
4. After validating the request, API Gateway provider or administrator associates the application with the API. Keys are generated for applications and not for every API that the application consumes.

   Note:
   The approval process, if any, is handled by the requesting application and not handled by API Gateway.
5. The API developer can then use the application with the proper identifier (such as the access key or identifier) to access the API.

API key expiration date

An API Gateway application has an optional expiration date for its API key. When the API access key expires, the application cannot be identified. The API Gateway Administrator can configure the **apiKeyExpirationPeriod** parameter from the **General > Extended settings** page. If the expiration date is not specified, then the API key never expires.

Suspended Applications

You can suspend applications so as to disable the identification of requests temporarily. If a suspended application is identified while processing a request the request is rejected with HTTP 403 (Forbidden) error. The response body has the following content:

```
Application has been identified but it is currently suspended. Please contact
the API Gateway administrator for further details.
```

You can resume the suspended applications to enable the identification again.

Creating an Application

You must have the API Gateway's manage applications functional privilege assigned to perform this task.

You can create an application from the Applications page.

To create an application

1. Click **Applications** in the title navigation bar.
2. Click **Create application**.
3. Provide the following information in the Basic information section:

Field	Description
Name	Type a name for the application.
Version	Version of the application. By default, it is 1.0 but can be modified to a required value.
Owner	Name of the team who owns the application. The application owner can view all details of the application including the API access key. If you specify a team as application owner, then all members of the team can view the API access key. **Note:** You cannot modify ownership details of the applications you create through Developer Portal.
Description	Type a description of the application.
Requestor comment	Specify your comments, if any. This field is visible only when the approval configuration for Create application is enabled in the **Administration** > **General** > **Approval Configuration** > **Create application** section.

The pending requests for an application can be approved by one of the following:

- List of users and user groups of the teams that the application is associated with. You must specify the required users and user groups in the **Approvers** section of the **Basic information** tab when creating or editing the corresponding team.
- List of users and user groups of the teams that the application is associated with. You must specify the required users and user groups in the **Team Administrators** section of the **Basic information** tab when creating or editing the corresponding. This is applicable only if the **Include team administrators as approvers** option is selected.

4. Click **Continue to Identifiers >**.

   Alternatively, you can click **Identifiers** in the left navigation panel.

You can save the application by clicking **Save** at this stage and add the Identifiers and APIs at a later time.

5. Provide the following information in the Identifiers section:

Field	Description
IP address range	Provide the IP address range or range of trusted IPv4 or IPv6 addresses that identify requests from a particular application. You can add more range options by clicking **+Add** and adding the required information.
Partner identifier	Specifies the third-party partner's identity. The specified partner can access the APIs if business-to-business communication between trading partners is enabled and where partners can invoke the exposed APIs to exchange information. For example, if you have enabled business-to-business communication between trading partners using APIs, partners can invoke the exposed APIs to exchange information. These APIs are available by associating Trading Networks with API Gateway. A partner can access the APIs that appear in the Partner Profiles and associated Partner Groups page. Once APIs are added as part of Partner, respective application is created in API Gateway with name ***partnerName* Application** and appropriate **Partner ID**. For more details on information on enabling business-to-business communication between trading partners and required configuration, see *webMethods Trading Networks Administrator's Guide* **Note:** No identification or enforcement of application happens in API Gateway using this identifier.
Client certificates	Click **Browse** and select the client certificate or certificate chain to be uploaded. The client certificate specifies the X.509 certificates that requests from a particular application. **Note:** API Gateway supports .cer and .pem certificates for identifying consumer applications. You can add multiple certificates by clicking **+Add**.
Claims	Provide a set of claims for the JWT and OpenID clients.

Field	Description
	A claim is a unique identifying information that identify requests from a particular consumer application. The claim set is identified by a unique **Name** and is defined as a name-value pair that consists of a **Claim name** and a **Claim value**. You can add more claims and claims sets by clicking **+Add** and adding the required information.
Header key	Specify the HTTP header key to identify the requests from an application.
Header value	Specify the HTTP header value to identify the requests from an application. You can add multiple header key and value by clicking **+Add**
Other identifiers	Select one of the options to identify requests from a particular application and provide the required value: ■ **Hostname.** The host name to identify requests from an application. ■ **Payload identifier.** The payload identifier to identify requests from an application. ■ **Team.** The team to identify requests from an application. A team can contain one or more groups or LDAP groups the application can be identified against a user belonging to any of these groups by the specified team. ■ **Token.** The token to identify requests from an application. ■ **Username.** The username credential to identify requests from an application. ■ **WS-Security username.** The WSS username to identify requests from an application.

6. Click **Continue to APIs >**

   Alternatively you can click **APIs** in the left navigation panel.

   You can save the application by clicking **Save** at this stage and add the APIs at a later time.

7. Type a keyword to find the required API and click **+** to add the API.

   Adding an API to the application enables the application to access the API. An API developer while invoking the API at runtime, has to provide the access token or identification token for API Gateway to identify the application.

8. Type the required Requestor comment.
9. Click **Continue to Advanced >**

   Alternatively you can click **Advanced** in the left navigation panel.

   You can save the application by clicking **Save** at this stage and add the APIs at a later time.
10. Specify the origin from which the responses originating are allowed during response processing for the application.

    Note:
    You cannot provide Regular expressions for allowed origins.
11. Click **+Add** to add the origin.

    You can add multiple origins using  .
12. Click **Continue to Authentication >**

    Alternatively you can click **Authentication** in the left navigation panel.

    You can save the application by clicking **Save** at this stage and add the Authentication strategy at a later time.
13. Click **Create strategy**.

    A strategy is a way to authenticate the incoming request and provides multiple authentication mechanisms or multiple authorization servers for a single authentication scheme. You can create multiple strategies authorized by an API for an application.
14. Select one of the **Authentication schemes**:

    - **OAUTH2**. Provide the following information:

Field	Description
Name	Provide the name for the strategy.
Description	Provide a description to describe the strategy.
Authentication server	Specify the authentication server. The available values are **local**, which is the default server or any other configured external authorization server.
Audience	Provide a value or URI, the intended recipient of the authorization server scope.

Field	Description
	The application that receives the token verifies that the audience value is correct and rejects any tokens intended for a different audience.

Generate Credentials	Enable the toggle button to generate the client dynamically in the authorization server and provide the following information:
	- **Type**. Select one of the client types:
	  - **Confidential**. A confidential client is an application that is capable of keeping a client password confidential to the world. This client password is assigned to the client app by the authorization server. This password is used to identify the client to the authorization server, to avoid fraud. An example of a confidential client could be a web app, where no one but the administrator can get access to the server, and see the client password.
	  - **Public**. A public client is an application that is not capable of keeping a client password confidential. For instance, a mobile phone application or a desktop application that has the client password embedded inside it. Such an application could get cracked, and this could reveal the password. The same is true for a JavaScript application running in the users browser. The user could use a JavaScript debugger to look into the application, and see the client password.
	- **Application type**. Specify the application type.
	  - **WEB**. A web application is an application running on a web server. In reality, a web application typically consists of both a browser part and a server part. The client password could be stored on the server. The password would thus be confidential.
	  - **USER_AGENT**. A user agent application is for instance a JavaScript application running in a browser. The browser is the user agent. A user agent application may be stored on a web server, but the application is only running in the user agent once downloaded.
	  - **NATIVE**. A native application is for instance a desktop application or a mobile phone application. Native applications are typically installed on the users computer or device (phone, tablet and so on). Thus, the client password will be stored on the users computer or device too.

Field	Description
	■ **Token lifetime**. Specify the token lifetime in seconds for which the token is active ■ **Token refresh limit**. Specify the number of times you can use the refresh token to get a new access token. ■ **Redirect URIs**. Specify the URIs that the authorization server can use to redirect the resource owner's browser during the grant process. You can add multiple URIs by clicking **+Add**. ■ **Grant type**. Specify the grant type to be used to generate the credentials. Available options can be **authorization_code**, **password**, **client_credentials**, **refresh_token**, and **implicit**, which are dynamically populated from the authorization server. For example, if the authorization server does not support client credentials, the option is not available in the options list. ■ **Scopes**. Select the scopes that are to mapped for the authentication strategy. **Note:** in API Gateway 10.2, the scopes are automatically created when you associate an API to an application. From API Gateway 10.3 onwards you have to select scopes from the authorization server that have to be associated with the strategy.
Client id	Specify the Client identifier for a client application available in the authorization server that identifies the client application in the authorization server to map the client to the API Gateway application. *This is required if you have a client application available in the authorization server and do not want to dynamically create a client*.

■ **JWT**. Provide the following information:

Field	Description
Name	Provide the name for the strategy.
Description	Provide a description to describe the strategy.
Authentication server	Specify the authentication server. The possible values are local, which is the default server or any other configured external authorization server.

Field	Description
Audience	Provide a value or URI, the intended recipient of the authorization server scope. The application that receives the token verifies that the audience value is correct and rejects any tokens intended for a different audience.
HMAC algorithm	Select if the authorization server is returning a JWT with HMAC algorithm and provide the shared secret value to validate the JWT.

- **OPENID**. Provide the following information:

Field	Description
Name	Provide the name for the strategy.
Description	Provide a description to describe the strategy.
Authentication server	Specify the authentication server. The available values are **local**, which is the default server or any other configured external authorization server.
Audience	Provide a value or URI, the intended recipient of the authorization server scope. The application that receives the token verifies that the audience value is correct and rejects any tokens intended for a different audience.

Generate Credentials	Enable the toggle button to generate the credentials required to identify the client application and provide the following information:
	
	- **Type**. Select the client type, Public or Confidential
	  - **Confidential**. A confidential client is an application that is capable of keeping a client password confidential to the world. This client password is assigned to the client app by the authorization server. This password is used to identify the client to the authorization server, to avoid fraud. An example of a confidential client could be a web app, where no one but the administrator can get access to the server, and see the client password.
	  - **Public**. A public client is an application that is not capable of keeping a client password confidential. For instance, a mobile phone application or a desktop

Field	Description

	application that has the client password embedded inside it. Such an application could get cracked, and this could reveal the password. The same is true for a JavaScript application running in the users browser. The user could use a JavaScript debugger to look into the application, and see the client password.
	■ **Application type**. Specify the application type.
	  ■ **WEB**. A web application is an application running on a web server. In reality, a web application typically consists of both a browser part and a server part. The client password could be stored on the server. The password would thus be confidential.
	  ■ **USER_AGENT**. A user agent application is for instance a JavaScript application running in a browser. The browser is the user agent. A user agent application may be stored on a web server, but the application is only running in the user agent once downloaded.
	  ■ **NATIVE**. A native application is for instance a desktop application or a mobile phone application. Native applications are typically installed on the users computer or device (phone, tablet etc.). Thus, the client password will be stored on the users computer or device too.
	■ **Token lifetime**. Specify the token lifetime in seconds for which the token is active
	■ **Token refresh limit**. Specify the time in seconds for which the token refresh is applicable
	■ **Redirect URIs**. Specify the URIs that the authorization server can use to redirect the resource owner's browser during the grant process. You can add multiple URIs by clicking +Add.
	■ **Grant type**. Specify the grant type to be used to generate the credentials. Available options are **Authorization code**, **Implicit**, **Resource owner**, **Client credentials**.
	■ **Scopes**. Select the scopes that are to be associated to the generated client.
	Note:
	In API Gateway 10.2, the scopes are automatically created when you associate an API to an application.

Field	Description
	From API Gateway 10.3 onwards you have to select scopes from the authorization server that have to be associated with the strategy.
Client id	Specify the Client identifier that identifies the client application in the authorization server to map the client to the API Gateway application. *This is required if you do not choose to generate credentials to identify the client application.*

15. Click **Add**.

    The strategy is configured and listed in the Strategies table.

    Note:
    API Gateway allows you to generate a new Client ID and Client Secret for an existing strategy. However, once the credentials are generated for a strategy, it can no longer be removed. The Generate credentials toggle is disabled in the UI when you update a strategy.

16. Click **Save**.

    The application creation request is sent for approval. If you are one of the approvers, then the application creation request is automatically approved, the application is created, and listed in the Manage applications page.

Viewing List of Applications and Subscriptions

You can view the list of applications and subscriptions in the Manage applications page from where you can create, delete, and select an application to view its details.

To view the application list and application and subscription details

1. Click **Applications** in the title navigation bar.

   A list of all registered applications and subscriptions appear.

   -  denotes application.
   -  denotes subscription.

2. Select an application.

   The application details page displays the following information: basic information that contains details such as name, description, owner, and creation time, identifiers, access tokens, APIs

registered for the application, advanced configurations, and authentication strategies configured for the application.

Application credentials, such as, API Keys or OAuth client secrets are visible only to the application owner. All other users can only see an encrypted value. Since API Portal and API Gateway do not support a central user management, API Gateway users cannot see the application credentials of the application requested through API Portal.

3. Select a subscription.

   You can view the applications and the associated package, plan, used quota, start time, end time, and the remaining period of the subscription.

   Note:
   You cannot create a subscription from this page. To create a subscription, use the subscription API. For details about creating subscriptions using a REST API, see

   . You can also create a subscription from the API Portal.

Regenerating API Access Key

You must have the API Gateway's manage applications functional privilege assigned to perform this task.

You can regenerate an API access key in the Application details page from where you can view application details.

Only the API owner can view the **API access key** field. This field is masked in the identification profile for all other users. An administrator can renew or revoke the API access key but cannot view it.

To regenerate an API key

1. Click **Applications** in the title navigation bar.

   A list of all registered applications is displayed.

2. Select an application.

   The application details page displays the basic information, identifiers, access tokens, API key, APIs registered and strategies configured for that application.

3. Click  .

   The API access key is regenerated and the new API access key appears in the **API access key** field.

Modifying Application Details

You can modify the details of an application as required from the application details page.

To modify application details

1. Click **Applications** in the title navigation bar.

   A list of registered applications is displayed.

2. Select an application.

3. Click **Edit** in the application details page.

4. Modify the required fields in the Basic information section.

   Note:
   You cannot modify ownership details of the applications you create through Developer Portal.

5. Click **Identifiers**.

6. Modify the required fields in the Identifiers section.

7. Click **APIs**.

8. Add or delete the APIs that are registered.

9. Modify the strategies or create a new strategy.

10. Modify the required values.

11. Click **Save**.

Registering an API with Consumer Applications from API Details Page

Consumer applications created in API Gateway can be associated with APIs from the API details page.

To register APIs with consumer applications

1. Click **APIs** in the title navigation bar.

   A list of APIs is displayed.

2. Select an API.
3. Click **Edit** in the API details page.
4. Click **Application** tab in the API details page.
5. Type characters in the search field and click the **Search** icon.

   This field displays the only list of applications that are assigned to the teams that you are a part of.
6. Select the required applications and click **+**.

   You can add more applications in a similar way.
7. Click **Save**.

Suspending an Application

You must have the API Gateway's manage applications functional privilege assigned or you must be the owner of the application to perform this task.

You can suspend an application from the Applications details page.

To suspend an application

1. Click **Applications** in the title navigation bar.

   A list of all the available applications are displayed.
2. Click the toggle button  (Active state), in the action column for the respective application, to suspend the application.

   Alternatively, you can click **Suspend** in the application details page.
3. Click **Yes** in the confirmation dialog box.

   The application is suspended. The toggle button in the Applications page changes to  (suspended state) and the option in the application details page changes to **Suspend**.

Activating a Suspended Application

You must have the API Gateway's manage applications functional privilege assigned or you must be the owner of the application to perform this task.

You can activate a suspended application, from the Applications details page, which enables the identification again.

To activate a suspended application

1. Click **Applications** in the title navigation bar.

   A list of all the available applications are displayed.

2. Click the toggle button  (suspended state), in the action column for the respective application, to activate the application.

   Alternatively, you can click **Activate** in the application details page.

3. Click **Yes** in the confirmation dialog box.

   The application resumes. The toggle button in the Applications page changes to  (active state) and the option in the application details page changes to **Suspend**.

Policies

API Gateway provides a policy framework that enables you to program API behavior and implements specific limited management functions without writing any code. You can enforce a policy on an API to perform specific tasks, such as transport, security, logging, routing of requests to target services, and transformation of data from one format to another. You can also define a policy to evaluate and process the various API invocations at runtime. For example, a policy could instruct API Gateway to perform any of the following tasks and prevent malicious attacks:

- Verify that the requests submitted to an API come from applications that are authenticated and authorized using the specified set of identifiers in the HTTP header to access and use the particular API.
- Validate digital signatures in the security header of request and response messages.
- Monitor a user-specified set of run-time performance conditions and limit the number of invocations during a specified time interval for a particular API and for applications, and send alerts to a specified destination when these performance conditions are violated.
- Log the request and response messages, and the run-time performance measurements for APIs and applications.

You can enforce policies on an API at the following levels:

- **Global policy enforcement**: This enforcement applies globally to all APIs defined in API Gateway. For example, the threat protection policies can be enforced for all APIs to protect against malicious attacks. For more information about threat protection policies, see *webMethods API Gateway Administration*.
- **API-level policy enforcement**. This enforcement applies to all resources and its nested methods of a REST API, or all operations of a SOAP API. These policies are further categorized into stages such as Transport, Identify and Access, Request and Response Processing, Routing, Error Handling, depending on their usage. For example, the Identify and access category of

policies can be enforced on an API to specify the kind of identifiers that are used to identify the application and authorize it against all applications registered in API Gateway.

- **Scope-level policy enforcement**
  - **Resource-level policy enforcement**. Applicable only for REST APIs. This enforcement applies to one or more resources and its nested methods in the REST API.
  - **Method-level policy enforcement**. Applicable only for REST APIs. This enforcement applies to one or more methods nested within a resource in the REST API.

    -OR-

    Operation-level policy enforcement. Applicable only for SOAP APIs. This enforcement applies to one or more operations in the SOAP API.

How does the policy enforcement precedence work?

When you apply the policies both globally (through global policies) and directly (through API-level policies and scope-level policies) to an API, API Gateway determines the effective set of policies for that API by taking into account the precedence of policy enforcement at the API-level, the policy stages, the priority of policies, run-time constraints, and the status (activated or deactivated) of any applied global policy.

For example, consider an API is enforced with the identify and access policy at the following policy enforcement levels: global, API-level, and scope-level. The precedence of the policy enforcement that is effective for the API at runtime is as follows:

1. Global policy enforcement
2. Method-level policy enforcement or operation-level policy enforcement
3. Resource-level policy enforcement
4. API-level policy enforcement

If the API has the identify and access policy applied both globally and at the API level, API Gateway does not show conflict. The identify and access policy applied through the global policy takes precedence and is processed at runtime.

Similarly for a REST API, identify and access policy is applied through a scope-level policy at the resource level and at the API level, the identify and access policy applied through the scope-level policy takes precedence and is processed at run-time.

When you apply a transport policy at the global level, the transport policy applied at the API level is in the disabled state. When you try deleting the API-level transport policy that is in the disabled state, an error displays and you are not allowed to delete this policy as the API-level transport policy is required and gets enforced when you deactivate the global policy.

Variable framework

All types of variables such as request, response, custom, custom-context, and system context variables are handled through the common framework called variable framework. The variable framework in API Gateway provides an option to extract variable values that can be used across

stages. For example, you can use the extracted variable to transform request and response contents such as headers, query parameters, path parameters payload, and so on as per your requirement. With the variable framework, you can normalize the syntax and create a common template for accessing the various variable types. For details about the variable syntaxes to use, see "Variable Framework" on page 373.

Aliases

API Gateway provides the capability of using aliases. An alias holds stage-specific property values that can be be shared by multiple policy configurations. Aliases referenced by policy configurations are substituted during runtime. Changing an alias value affects all referencing policies. Aliases are referenced through a name therefore alias names have to be unique within an API Gateway. The corresponding alias value is substituted in place of an alias name during run-time. Thus the same alias can be referred to in multiple policies and the change in a particular alias would affect all the policy properties. For more details about aliases and how to use them, see " Aliases" on page 454.

Policy templates

API Gateway provides policy templates, which are a set of policies that can be associated directly with an individual API. Policy templates provide the flexibility to alter the policy's configurations to suit the individual API requirements. These policy templates apply at the API level, and can be customized to suit the needs of a particular API. For more details about policy templates and how to use them, see "Policy Templates" on page 502.

Policy validation and dependencies

When you enforce a policy to govern an API at run-time, API Gateway validates the policies to ensure that:

- Any policy (for example, Log Invocation) that can appear in an API multiple times is allowed to appear multiple times.
- For policies (for example, Enable HTTP / HTTPS) that can appear only once in an API, API Gateway issues an error message.
- For policies (for example, Monitor SLA) that are dependent and use another policy in conjunction (for example, Identify & Authorize) in an API, API Gateway prompts you with a warning message to include the dependent policy.

When you save an API, API Gateway combines the policies from all the global and direct policies that apply to the API and generates what is called the *effective policy* for the API. For example, if your REST API is within the scope of two policies: one policy that performs a logging task and another policy that performs a security task and when you save the REST API, API Gateway automatically combines the two policies into one effective policy. The effective policy, which contains both the logging task and the security task, is the policy that API Gateway actually uses to publish the REST API.

When API Gateway generates the effective policy, it validates the resulting policy to ensure that it contains no conflicting or incompatible policies.

If the policy contains conflicts or inconsistencies, API Gateway computes the effective API policy according to policy resolution rules. For example, an effective API policy can include only one Identify & Authorize policy. If the resulting policy list contains multiple Identify & Authorize policies, API Gateway shows the conflict by including a Conflict (⚠) icon next to the name of the conflicting policies in the effective policy. For details about policy validation and dependencies, see "Policy Validation and Dependencies" on page 368.

Transport

The policies in this stage specify the protocol to be used for an incoming request and the content type for a REST request during communication between API Gateway and an application. The policies included in this stage are:

- Enable bulkhead
- Enable HTTP/HTTPS
- Enable JMS/AMQP
- Set Media Type

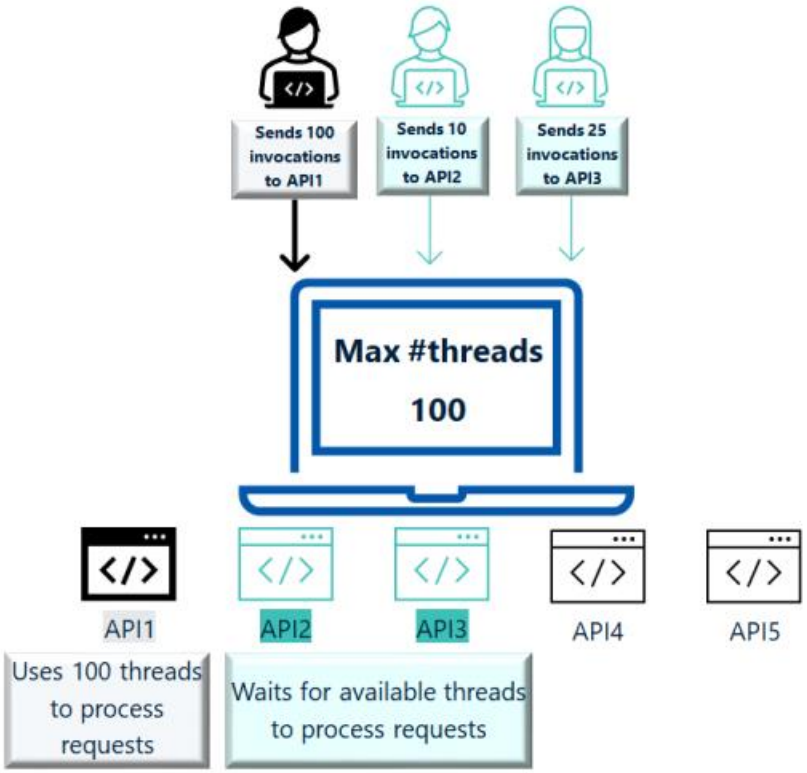
Enable Bulkhead

Bulkhead configuration allows you to specify the maximum number of concurrent requests processed by an API. You can configure this setting individually for an API or globally for all APIs.

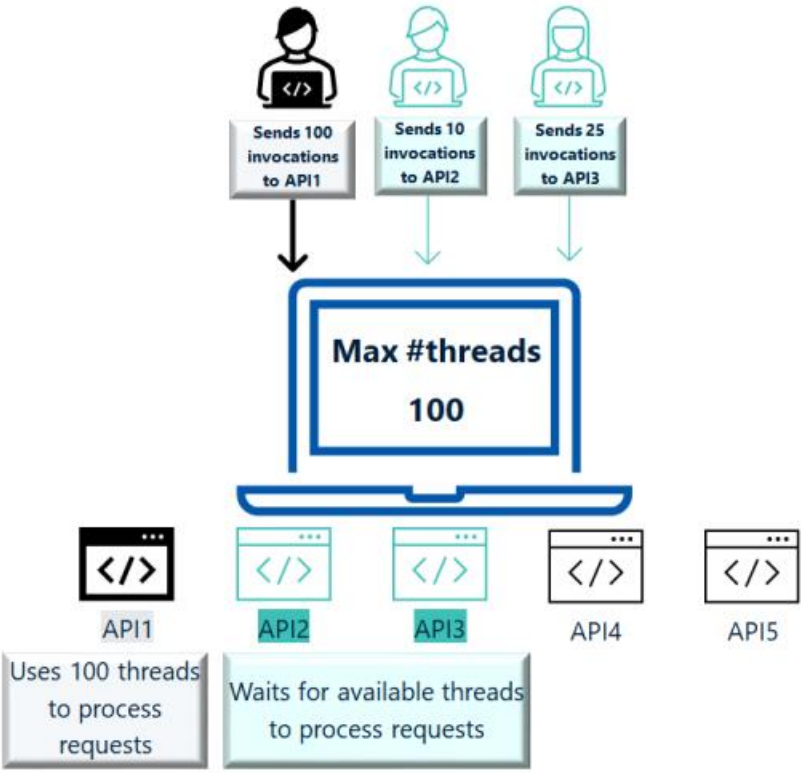
When the number of concurrent requests to an API exceeds the specified limit, the excess requests are rejected. In such scenarios, the policy violation events are generated to report the violations occurred for an API. If there are 100 violations, then 100 policy violation events are generated.

As per the order of policies, the Bulkhead limit policy is applied first. That is, if you have applied multiple policies including the Bulkhead limit policy, then the Bulkhead limit policy is applied first.

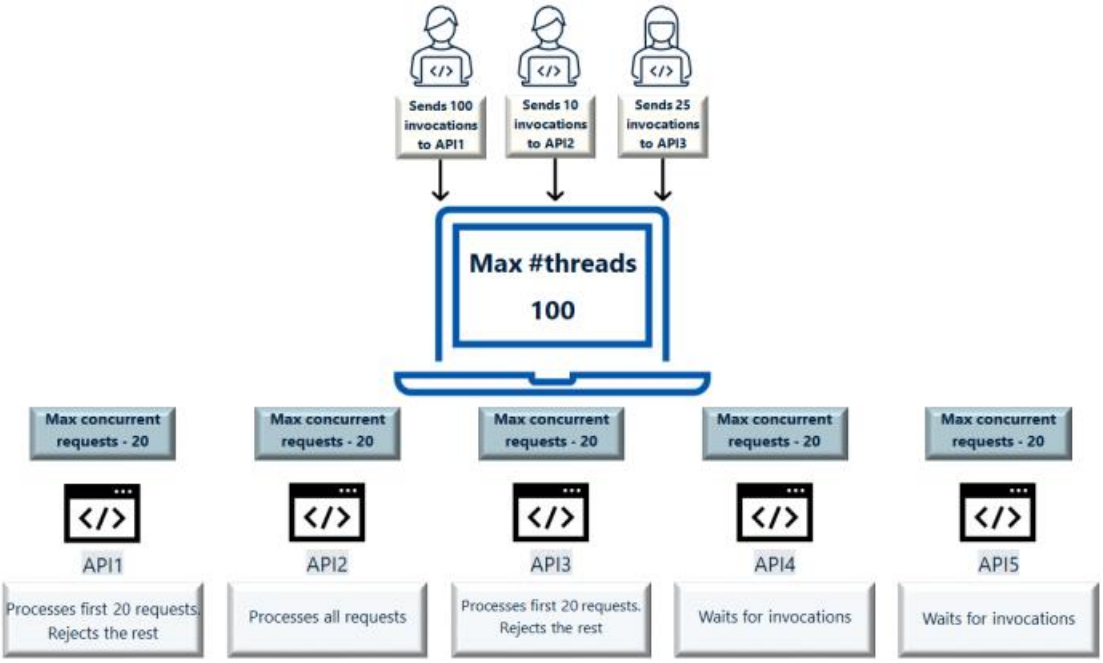
Why do we configure bulkhead limit for APIs?



In an environment where multiple APIs are running, there are chances of one API making maximum use of the system resources due to the number of invocations. This in turn suppresses the performance of other APIs. Consider the following example. The maximum thread pool size of your system is 100, and it is shared among five APIs. When there are 100 invocations to one API, then that API uses up all the available threads to process its requests. The others APIs must wait till there are available threads.



To prevent this, you can specify the maximum number of concurrent requests that an API can process. This number depends on the maximum thread pool size of your system. Consider the same scenario after specifying the bulkhead limit for the APIs. In our example, the thread pool size is 100 and if you configure 20 as the maximum concurrent limit for each API, then each of the APIs can process a maximum of 20 requests, exceeding which the requests are rejected.



This ensures optimal usage of the system resources.

As part of bulkhead configuration, you can specify

- **Bulkhead limit for an API at the API level**. Specifies the maximum concurrent request limit for an API, exceeding which the requests are rejected. This number does not include the callbacks that an API receives. So, you can separately specify the maximum concurrent callbacks that an API can handle. The `503 Service unavailable` error code is sent to the client service when the requests are rejected. You can configure the required error code and status phrase using the extended settings. You can customize the required status code and message using the extended settings **pg.bulkhead.statusCode** and **pg.bulkhead.statusMessage** respectively.
- **Bulkhead limit for all APIs (Global policy)**. Specifies the maximum concurrent request limit for all APIs. Similar to the API level configuration, you can provide the maximum concurrent callbacks and your choice to retry the rejected requests.

  When you have configured global-level bulkhead limit for APIs, and if you configure a different bulkhead limit at an API-level, then the limit configured at the global level takes precedence. To override this, you must exclude the required APIs from the global-policy using filters. You can apply the required API filters when creating or editing the bulkhead global policy. For information about creating global policies and applying API filters, see " Creating a Global Policy" on page 473.
- **Add Retry-After response header**. Specifies whether the **Retry-After** header must be included in the response sent to the client when requests are rejected. If you select to add, you must also specify the duration (in seconds) that the system must wait before sending the consecutive requests. You can configure this setting to keep the client informed on the waiting duration before processing the consecutive requests.

Enabling bulkhead for APIs at the API level

This use case explains how to configure the bulkhead limit for an API.

This use case starts when you have an API for which you have to enforce a bulkhead limit and ends when you have successfully applied the bulkhead limit to the API.

To enforce bulkhead limit to an API

1. Click **APIs** in the title navigation bar.
2. Select the required API.
3. Click the **Policies** tab.
4. From the **Policy catalog** section, expand **Transport**, and select **Enable bulkhead**.
5. Provide the following values in the **Policy properties** section:

Field	Description
Maximum concurrent calls	Specify the maximum number of concurrent requests that the API can process, exceeding which the requests are rejected.
Enable bulkhead for callbacks	Select this option to specify the maximum number of concurrent callbacks for the API. If you select this the **Maximum concurrent callbacks** field appears. **Note:** This setting is applicable only for REST APIs. Hence, this field is not displayed when you perform the bulkhead limit configuration for any other APIs.
Maximum concurrent callbacks	Specify the maximum number of concurrent callbacks that the API can process, exceeding which the callbacks are rejected. This field appears only when you have selected the **Enable bulkhead for callbacks** check box.
Add Retry-After response header	Select this option to include the **Retry-after** header in the response sent to the client when requests are rejected. This is to keep the client informed about the waiting duration before sending any consecutive requests. **Note:** This option is to provide an approximate waiting duration before sending any consecutive requests; and it does not guarantee that the requests sent after the specified time are processed. The maximum number of concurrent requests that are processed is purely dependant on the configured bulkhead limit.
Retry after (value, in seconds)	Specify the duration that the client must wait, after requests are rejected by API Gateway, to send consecutive requests. This field appears only when you have selected the **Retry after (value, in seconds)** check box.

6. Click **Save**.

   The bulkhead limit is applied to the API.

Bulkhead feature considerations

This section lists the impact of the bulkhead policy on other features that you must remember when using the policy:

Applicable API types

Bulkhead policy is applicable for REST, SOAP, OData, and GraphQL APIs.

However, you can specify the number of maximum concurrent callbacks only for the REST APIs.

Bulkhead limit configuration for APIs in a mashup

API-level bulkhead limit is not applied to the participating APIs. Instead, the bulkhead limit configured at the mashup API-level takes effect.

Bulkhead limit configured for APIs in cluster

In cluster environments, you can specify the bulkhead limit in each node. API Gateway checks the number of concurrent requests per instance and allows or rejects the requests based on the bulkhead limit configured for an API in the corresponding node.

Overall thread pool violation

When you enable the Bulkhead policy and if the overall thread pool is violated, then the requests exceeding the thread pool count are all rejected, and the **Connection timeout** message is displayed.

Conditional Routing policy

When the Bulkhead policy is enabled with the Conditional Routing policy, then bulkhead limit configured at API-level for the participating APIs is applicable.

Enable HTTP/HTTPS

This policy specifies the protocol to use for an incoming request to the API on API Gateway. If you have a native API that requires clients to communicate with the server using the HTTP and HTTPS protocols, you can use the Enable HTTP or HTTPS policy. This policy allows you to bridge the transport protocols between the client and API Gateway.

For example, you have a native API that is exposed over HTTPS and an API that receives requests over HTTP. If you want to expose the API to the consumers of API Gateway through HTTP, then you configure the incoming protocol as HTTP.

The table lists the properties that you can specify for this policy:

Property	Description
Protocol	Specifies the protocol (HTTP or HTTPS), SOAP format (for a SOAP-based API) to be used to accept and process the requests. Select one of the following: ■ **HTTP**. API Gateway accepts requests that are sent using the HTTP protocol. This is selected by default. ■ **HTTPS**. API Gateway accepts requests that are sent using the HTTPS protocol.

Property	Description
SOAP Version	For SOAP-based APIs. Specifies the SOAP version of the requests which the API Gateway accepts from the client. Select one of the following: ■ **SOAP 1.1**. This is selected by default. API Gateway accepts requests that are in the SOAP 1.1 format. ■ **SOAP 1.2**. API Gateway accepts requests that are in the SOAP 1.2 format.

Set Media Type

This policy specifies the content type for a REST request. If the content type header is missing in a client request sent to an API, API Gateway adds the content type specified here before sending the request to the native API.

The table lists the properties that you can specify for this policy:

Property	Description
Default Content-Type	Specifies the default content type for REST request received from a client .
Default Accept Header	Specifies the default accept header for REST request received from a client.

As both these properties support variable framework, you can use the available variables to specify the content type and accept header. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

Enable JMS/AMQP

Java Message Service (JMS) is a standard Java API for communicating with message oriented middleware and enables loosely coupled communication between two or more homogenous systems. It provides reliable and asynchronous form of communication.

Advanced Message Queuing Protocol(AMQP) is an open standard application layer protocol for delivering messages. AMQP can queue and route messages in a reliable and secured way. AMQP provides a standard messaging protocol that stands across all platforms and a description on how a message should be constructed. It doesn't provide an API on how the message should be sent. AMQP being language agnostic is useful in the message oriented middleware to achieve interoperability in asynchronous way among heterogenous systems.

When you want to expose a REST or SOAP API over JMS with broker native protocol or JMS with AMQP protocol add and configure the **Enable JMS/AMQP** policy in API Gateway, thereby allowing them to communicate through the messaging Queue or Topic.

For example, you can use this policy to expose your API over JMS/AMQP and hence enable your client to communicate through the messaging queue or topic.

- JMS with Message broker native protocol support

  For example, if your Message broker is using ActiveMQ and if you are relying on the default Active MQ TCP protocol, then essentially it is JMS on open wire protocol because open wire is the native protocol of ActiveMQ Message broker.

- JMS with AMQP protocol support

  For example, if you want to use JMS with AMQP with any message broker which supports AMQP 1.0 to achieve interoperability in asynchronous way among heterogenous systems. API Gateway supports AMQP 1.0 using Apache qpid JMS client.

Note:
The following are not supported if the **Enable JMS/AMQP** policy is added:

- **Threat protection** policies
- API Gateway SOAP to REST transformation feature

Use case 1: Expose a SOAP API over JMS with a message broker native protocol

This describes the high level workflow for the scenario where you want to expose a SOAP API over JMS with a message broker native protocol.

1. Create an alias to a JNDI Provider. For a detailed procedure, see *webMethods API Gateway Administration*.
2. Configure API Gateway to use a JMS connection alias to establish an active connection between API Gateway and the JMS provider. For a detailed procedure, see *webMethods API Gateway Administration*.
3. Create a WS (Web Service) JMS Provider endpoint alias and configure the Alias, Description, Type (Provider), Transport Type (JMS) fields and JMS Transport Properties. For a detailed procedure, see *webMethods API Gateway Administration*.
4. A WS (Web Service) endpoint trigger is created when you configure WS (Web Service) JMS Provider endpoint alias. This trigger consists of the input source details like Queue name or Topic name. You can update the WS (Web Service) endpoint trigger, as required. For detailed procedures, see *webMethods API Gateway Administration*.
5. Select the required API.
6. Click **Edit**.
7. In the API Details section click **Policies**.
8. Enforce the **Enable JMS/AMQP** policy with the following properties configured.

a. Specify the name of the JMS provider endpoint alias that specifies the trigger which listens to the source queue or topic for the input message.

b. Specify the SOAP version of the requests which the API Gateway accepts from the client.

For details on the **Enable JMS/AMQP** policy, see "Using Enable JMS/AMQP for a SOAP API" on page 108.

9. Click **Save**.

The enforced policy **Enable JMS/AMQP** with the required configuration now allows any java client to communicate with the API asynchronously.

Use case 2: Expose a SOAP API over JMS with AMQP protocol

This describes the high level workflow for the scenario where you want to expose a SOAP API over JMS with AMQP protocol.

1. Configure API Gateway to use JMS with AMQP protocol.

Configure JNDI settings and JMS settings as per the client you are using to achieve JMS over AMQP protocol support.

Note:
For a sample procedure on configuring API Gateway to use JMS with AMQP protocol using Apache qpid libraries, see "Configuring API Gateway for JMS with AMQP Protocol" on page 106.

2. Select the required API.
3. Click **Edit**.
4. In the API Details section click **Policies**.
5. Enforce the **Enable JMS/AMQP** policy with the following properties configured.

a. Specify the name of the JMS provider endpoint connection alias that specifies the trigger which listens to the source queue or topic for the input message.

b. Specify the SOAP version of the requests which the API Gateway accepts from the client.

For details on the **Enable JMS/AMQP** policy, see "Using Enable JMS/AMQP for a SOAP API" on page 108.

6. Click **Save**.

The enforced policy **Enable JMS/AMQP** with the required configuration now allows all the clients such as Python, Ruby, Java, and Dotnet to communicate with the API asynchronously.

Use case 3: Expose a REST API over JMS with a message broker native protocol

This describes the high level workflow for the scenario where you want to expose a REST API over JMS with a message broker native protocol.

1. Create an alias to a JNDI Provider. For a detailed procedure, see *webMethods API Gateway Administration*.
2. Configure API Gateway to use a JMS connection alias to establish an active connection between API Gateway and the JMS provider. For a detailed procedure, see *webMethods API Gateway Administration*.
3. Select the required API.
4. Click **Edit**.
5. In the API Details section click **Policies**.
6. Enforce the **Enable JMS/AMQP** policy with the following properties configured.
   a. Specify the name of the JMS provider endpoint alias that contains the configuration information needed to establish a connection to a specific JMS provider.
   b. Specify the input source name which API Gateway starts listening to when the API is activated.
   c. Specify the type of source type Queue or Topic, which the API Gateway listens for the request message.
   d. Specify the selector, a criteria for the API Gateway tp listen to a message containing the specified criteria

   For details on the **Enable JMS/AMQP** policy, see "Using Enable JMS/AMQP for a REST API" on page 109.
7. Click **Save**.

   The enforced policy **Enable JMS/AMQP** with the required configuration now allows any java client to communicate with the API asynchronously.

Use case 4: Expose a REST API over JMS with AMQP protocol

This describes the high level workflow for the scenario where you want to expose a REST API over JMS with AMQP protocol.

1. Configure API Gateway to use JMS with AMQP protocol.

   Configure JNDI settings and JMS settings as per the client you are using to achieve JMS over AMQP protocol support.

   Note:
   For a sample procedure on configuring API Gateway to use JMS with AMQP protocol using Apache qpid libraries, see "Configuring API Gateway for JMS with AMQP Protocol" on page 106.
2. Select the required API.
3. Click **Edit**.

4. In the API Details section click **Policies**.
5. Enforce the **Enable JMS/AMQP** policy with the following properties configured.
   a. Specify the name of the JMS provider endpoint alias that contains the configuration information needed to establish a connection to a specific JMS provider.
   b. Specify the input source name which API Gateway starts listening to when the API is activated.
   c. Specify the type of source type Queue or Topic, which the API Gateway listens for the request message.
   d. Specify the selector, a criteria for the API Gateway tp listen to a message containing the specified criteria.

   For details on the **Enable JMS/AMQP** policy, see "Using Enable JMS/AMQP for a REST API" on page 109.
6. Click **Save**.

   The enforced policy **Enable JMS/AMQP** with the required configuration now allows all the clients such as Python, Ruby, Java, and Dotnet to communicate with the API asynchronously.

Configuring API Gateway for JMS with AMQP Protocol

Before configuring AMQP in API Gateway, ensure your message broker supports AMQP 1.0

For using JMS with AMQP protocol in API Gateway you have to configure the appropriate settings for the provider URL and the connection factory lookup name required for API Gateway to communicate using JMS with AMQP protocol.

To configure API Gateway to use JMS with AMQP protocol

1. Create a properties file that contains the information for the JNDI lookup name and connectionfactory details.
2. Configure JNDI settings as per the client you are using to achieve JMS over AMQP protocol support.

   For a detailed procedure, see *webMethods API Gateway Administration*.
3. Configure JMS settings as per the client you are using to achieve JMS over AMQP protocol support.

   For a detailed procedure, see *webMethods API Gateway Administration*.

To configure API Gateway for JMS with AMQP protocol using Apache qpid libraries

1. Create a properties file that contains the information for the JNDI lookup name and connectionfactory details.

   A sample properties file, for example amqp.properties, would look like

```
# Set the InitialContextFactory class to use
java.naming.factory.initial = org.apache.qpid.jms.jndi.JmsInitialContextFactory
# Define the required ConnectionFactory instances
# connectionfactory.<JNDI-lookup-name> = <URI>
connectionfactory.qpidConnectionFactory = amqp://<hostname>:<port#>
```

2. Navigate to  > **Administration**.
3. Select **General > Messaging**.
4. Configure the JNDI provider alias as follows:
   a. Click **Add JNDI provider alias** in the JNDI provider alias definitions section.
   b. Provide the following information:
      - **JNDI Alias Name**. Provide a name that you want to assign to this JNDI provider.
      - **Description**. Provide a brief description for this JNDI alias.
      - **Predefined JNDI Templates**. Select the predefined JNDI template depending on the provider you may want to use.

        For example, if you want to use the JMS with AMQP protocol, select **Qpid AMQP (0-x)**.
      - **Initial Context Factory**. The JNDI provider uses the initial context as the starting point for resolving names for naming and directory operations. This value gets pre-populated depending on the predefined JNDI template selected. For example, if you have selected **Qpid AMQP (0-x)** as the predefined JNDI template the Initial context factory field would display `org.apache.qpid.jms.jndi.JmsInitialContextFactory`.
      - **Provider URL**. Provide the file path location of the properties file that contains the context factory details. For example, `C:\amqp.properties`.
   c. Click **Add**.

      The JNDI provider alias is created and listed in the JNDI Provider alias definitions table.
5. Configure the JMS settings as follows:
   a. Click **Add JMS connection alias** in the JMS connection alias definitions section.
   b. Provide the following information in the General Settings section:
      - **Connection Alias Name**. Provide a name for the connection alias. Each connection alias represents a connection factory to a specific JMS provider.
      - **Description**. Provide a brief description for the connection alias.
   c. Provide the following information in the Connection Protocol Settings section:

- **JNDI Provider Alias Name**. The alias to the JNDI provider that you want this JMS connection alias to use to look up administered objects. Select the JNDI Provider alias name created in the earlier step.
- **Connection Factory Lookup Name**. The lookup name for the connection factory that you want to use to create a connection to the JMS provider specified in this JMS connection alias. Provide the value `qpidConnectionFactory`.

d. Click **Add**.

The JMS Connection alias is created and listed in the JMS Connection Alias Definitions table.

e. Enable the JMS connection alias by clicking toggle button to enable it.

The JNDI provider alias and the JMS connection alias are now set up and API Gateway is configured to use JMS with AMQP protocol.

Using Enable JMS/AMQP for a SOAP API

This policy is used to expose a SOAP API over JMS/AMQP. A SOAP API can be exposed as HTTP/HTTPS or JMS/AMQP as the policies **Enable HTTP/HTTPS** and **Enable JMS/AMQP** are mutually exclusive.

If you are using JMS with Message Broker native protocol support ensure that following actions are performed before using the **Enable JMS/AMQP** policy:

- Create an alias to a JNDI Provider. For a detailed procedure, see *webMethods API Gateway Administration*.
- Configure API Gateway to use a JMS connection alias to establish an active connection between API Gateway and the JMS provider. For a detailed procedure, see *webMethods API Gateway Administration*.
- Create a WS (Web Service) JMS Provider endpoint alias and configure the Alias, Description, Type (Provider), Transport Type (JMS) fields and JMS Transport Properties. For a detailed procedure, see *webMethods API Gateway Administration*.
- Configure a WS (Web Service) endpoint trigger. For detailed procedures, see *webMethods API Gateway Administration*.

If you are using JMS with AMQP protocol support, ensure the following before using the Enable JMS/AMQP policy:

- You have configured API Gateway for JMS with AMQP. For details, see "Configuring API Gateway for JMS with AMQP Protocol" on page 106.

The table lists the properties that you can specify for this policy:

Property	Description
JMS Provider Endpoint Alias	Specifies the name of the JMS provider endpoint alias. The provider endpoint alias specifies the trigger which listens to the source queue or topic for the input message.
SOAP Version	Specifies the SOAP version of the requests which the API Gateway accepts from the client. Select one of the following: ■ **SOAP 1.1**. This is selected by default. API Gateway accepts requests that are in the SOAP 1.1 format. ■ **SOAP 1.2**. API Gateway accepts requests that are in the SOAP 1.2 format.

Using Enable JMS/AMQP for a REST API

This policy is used to expose a REST API over JMS/AMQP. A REST API can be exposed as both HTTP/HTTPS and JMS/AMQP at the same time.

If you are using JMS with Message broker native protocol support ensure that following actions are performed before using the Enable JMS/AMQP policy:

- Create an alias to a JNDI Provider. For a detailed procedure, see *webMethods API Gateway Administration*.
- Configure API Gateway to use a JMS connection alias to establish an active connection between API Gateway and the JMS provider. For a detailed procedure, see *webMethods API Gateway Administration*.

If you are using JMS with AMQP protocol support, ensure the following before using the Enable JMS/AMQP policy:

- You have configured API Gateway for JMS with AMQP. For details, see "Configuring API Gateway for JMS with AMQP Protocol" on page 106.

The table lists the properties that you can specify for this policy:

Property	Description
Connection Alias Name	Specifies the name of the connection alias. Each connection alias contains the configuration information needed to establish a connection to a specific JMS provider.
Add JMS/AMQP source details. Click to add the JMS/AMQP source details and provide the required information.	

Property	Description
Input Source Name	Specifies the input source name which API Gateway starts listening to when the API is activated.
Input Source Type	Specifies the type of source to which the API Gateway listens for the request message. Select one of the following source type: ■ **QUEUE**. Indicates that API Gateway listens to the specified queue for the request message. ■ **TOPIC**. Indicates that the API Gateway listens to the specified topic for the request message. **Note:** Provides support only for non-durable topic.
Selector	Specifies the criteria for the API Gateway to listen to a message containing the specified criteria. For example, operation = GET If you have multiple selectors it follows the OR condition. If there are no selectors the message that comes in is listened to without any condition. **Note:** Message selectors are only applicable for headers, properties and not for payload.
Resource	Specifies the resource of the API.
HTTP Method	Specifies the routing method used. Available routing methods: GET, POST, PUT, and DELETE.
Content Type	*Optional*. Specifies the content type of the JMS/AMQP message body. Examples for content types: `application/json`, `application/xml` **Note:** Alternatively, you can use the Set Media Type policy to set the default content type instead of setting it here.

Identify and Access

The policies in this stage provide different ways of identifying and authorizing the application, and provide the required access rights for the application. The policies included in this stage are:

- Inbound Auth - Message
- Authorize User
- Identify & Authorize
- Custom Extension

The Inbound authentication policies are used to authenticate the application by specifying user-based SPN or host-based SPN for a Kerberos token, using the basic credentials for the HTTP basic authentication or through various token assertions or through the XML security actions.

The Authorize User policy authorizes the application against a list of users and a list of groups registered in API Gateway.

The Identify & Authorize policy is used to identify the application, authenticate the request based on policy configured and authorizes it against all applications registered in API Gateway.

Custom Extension policies allow you to handle requirements that might not be provided by the out-of-the-box policies. You can add these custom extensions into API Gateway policy stages. To learn more about Custom Extension, see *webMethods API Gateway User's Guide*.

Note:
From API Gateway 10.3, the Identification and Authentication policies are merged into one and you would not be able to do identification alone for Basic Authentication. You must provide the right credentials for a successful invoke.

Identify & Authorize

This policy identifies and validates the authorization of the applications to access the APIs. The application are identified using a set of identification types such as API key, hostname address, and HTTP basic authentication and so on based on the configuration. API Gateway can identify and authorize the application based on the following **Application Lookup condition**:

- **Registered applications**. Identifies the application and validates the identified application against the registered applications. On successful validation, API Gateway allows access to the API. The application that are associated with the API are called as registered application.
- **Global applications**. Identifies the application and validates the identified application against the global applications. On successful validation, API Gateway allows access to the API. All the active applications that are available in API Gateway are called as global application.
- **Global applications and DefaultApplication**. Verifies the identity of the application against the global applications and on identification failure the API Gateway allows access to the API as default application.

Note:
If **Allow anonymous** is selected and even if the **Application Lookup condition** does not meet, API Gateway allows access to the API.

The table lists the properties that you can specify for this policy:

Property	Description
Condition	Specifies the condition operator for the identification and authentication types. Select any of the following condition operators: ■ **AND**. Applies all the identification and authentication types. ■ **OR**. Applies one of the selected identification and authentication types. **Note:** Even though this policy provides the option of choosing an **AND** or **OR** operation between the different identification and authentication types, the operation across the different policies in the IAM stage is always **AND**.
Allow anonymous	Specifies whether to allow all users to access the API without restriction. When you add a security policy and configure **Allow anonymous**, all requests are allowed to pass through to the native API, but the successfully identified requests are grouped under the respective identified application, and all unidentified requests are grouped under a common application named asDefaultApplication (sys:defaultApplication). While you allow all requests to pass through you can perform all application-specific actions, such as, viewing the runtime events for a particular application, monitor the service level agreement for a few applications and send an alert email based on some criteria like request count or availability, and throttle the requests from a particular application and not allow the request from that application if the number of requests reach the configured hard limit within configured period of time.

Identification Type. Specifies the identification type. You can select any of the following.

You can set the "trigger policy violation event" to true or false if authorization header is not provided for the following identification types:

1) HTTP Basic authentication

2) OAuth2 token

3) TokenId connect

For other identification types, the default value is true. That is, policy violation events are triggered for the requests without authorization headers.

Note:

Property	Description
When you add an API to a package for monetization, the **API key** authentication mechanism is automatically added to the IAM policy at API level. If the API already contains an IAM policy that has two authentication mechanisms with the **AND** condition, then the condition will be switched to **OR**. This ensures the monetization is supported when certain consumers access the API by just using the API key.	

API Key	Specifies using the API key to identify and validate the client's API key to verify the client's identity in the registered list of applications for the specified API.
	
	Select one of the **Application Lookup condition**:
	
	■ **Registered applications**. Identifies the client's API key against the API key of all the applications registered to the API. On successful identification, API Gateway allows access to the API.
	
	■ **Global applications**. Identifies the client's API key against the API key of all the applications available in API Gateway. On successful identification, API Gateway allows access to the API.
	
	■ **Global applications and DefaultApplication**. Identifies the client's API key against all the applications available in API Gateway. Even though, if no global application is identified, API Gateway allows access to the API as default application.
	
	When this option is selected, you can use the API key as:
	
	■ Header parameter to consume an API. For example,
	`x-Gateway-APIKey:a4b5d569-2450-11e3-b3fc-b5a70ab4288a`
	
	■ Query parameter to invoke an API resource. For example,
	`http://pie-3HKYMH2:5555/gateway/PetstoreAPI/1.0.3/store/inventory?APIKey=faab7ac6-97a4-4228-908d-f1930faba470`

Hostname Address	Specifies using host name address to identify the client, extract the client's hostname from the HTTP request header and verify the client's identity in the specified list of applications in API Gateway. Select one of the **Application Lookup condition**: ■ **Registered applications**. Identifies the client's hostname against the *hostname* identifier of all the applications registered to the API. On successful identification, API Gateway allows access to the API. ■ **Global applications**. Identifies the client's hostname against the *hostname* identifier of all the applications available in API

Property	Description
	Gateway. On successful identification, API Gateway allows access to the API. ■ **Global applications and DefaultApplication**. Identifies the client's hostname against the *hostname* identifier of all the applications available in API Gateway. If no global application is identified, then API Gateway allows access to the API as default application. **Note:** If the client request has X-Forwarded-For header, then API Gateway resolves the hostname from the IP address present in the X-Forwarded-For header. Else, API Gateway resolves the hostname from the client's IP address.
HTTP Basic Authentication	Specifies using Authorization Header in the request to identify and authorize the client application against the list of applications with the identifier `username` in API Gateway. Provide the following information: ■ Select one of the **Application Lookup condition**: ■ **Registered applications**. Authenticates the user and identifies the user against *username* identifier of all the applications registered to the API. On successful authentication and identification, API Gateway allows access to the API. ■ **Global applications**. Authenticates the user and identifies the user against *username* identifier of all the applications available in the API Gateway. On successful authentication and identification, API Gateway allows access to the API. ■ **Global applications and DefaultApplication**. 1. Authenticates the user and identifies the user against *username* identifier of all the applications available in the API Gateway. 2. On successful authentication and if no global application is identified, then API Gateway allows access to the API as default application. 3. In case if the authentication fails, then API Gateway does not allow access to the API. ■ If **Global applications and DefaultApplication** and **Allow anonymous** are selected:

Property	Description
	1. Authenticates the user and identifies the user against *username* identifier of all the applications available in the API Gateway. 2. On successful authentication and if no global application is identified, then API Gateway allows access to the API as default application. 3. In case if the authentication fails, then API Gateway still allows access to the API. ■ **Trigger policy violation event on missing authorization header**. Creates a policy violation event for basic authentication if Authorization Headers are missing. Possible values: ■ `true`. Requests without authorization headers are logged as a policy violation event. `false`. Requests without authorization headers are not logged as a policy violation event.
IP Address Range	Specifies using the IP address range to identify the client, extract the client's IP address from the HTTP request header, and verify the client's identity against the specified list of applications in API Gateway. Select one of the **Application Lookup condition**: ■ **Registered applications**. Identifies the client's IP address against the *IP address range* identifier of all the applications registered to the API. On successful identification, API Gateway allows access to the API. ■ **Global applications**. Identifies the client's IP address against the *IP address range* identifier of all the applications available in API Gateway. On successful identification, API Gateway allows access to the API. ■ **Global applications and DefaultApplication**. Identifies the client's IP address against the *IP address range* identifier of all the applications available in API Gateway. If no global application is identified, then API Gateway allows access to the API as default application. **Note:** If the client request has X-Forwarded-For header, then API Gateway uses the IP address present in the X-Forwarded-For

Property	Description
	header. Else, API Gateway uses the client's IP address for identification.
JWT	Specifies using the JSON Web Token (JWT) to identify the client, extract the claims from the JWT and validate the client's claims, and verify the client's identity against the specified list of applications in API Gateway. Select one of the **Application Lookup condition**: ■ **Registered applications**. Identifies the JWT against the *claims* identifier of all the applications registered to the API. On successful identification, API Gateway allows access to the API. ■ **Global applications**. Identifies the JWT against the *claims* identifier of all the applications available in API Gateway. On successful identification, API Gateway allows access to the API. ■ **Global applications and DefaultApplication**. Identifies the JWT against the *claims* identifier of all the applications available in API Gateway. If no global application is identified, then API Gateway allows access to the API as default application. **Note:** You can use the claims in the JWT for further processing using request transformation policy.
Kerberos Token	Specifies using the Kerberos token to identify the client, extract the client's credentials from the Kerberos token, and verify the client's identity against the specified list of applications in API Gateway. **Note:** You have to enforce the Inbound Auth - Message policy with the property, Kerberos Token Authentication, configured, so when Identify & Authorize policy is executed, the user details fetched are used to match with application's data to identify the application. Select one of the **Application Lookup condition**: ■ **Registered applications**. Authenticates the incoming Kerberos token and identifies the user against the *username* identifier of all the applications registered to the API. On successful authentication and identification, API Gateway allows access to the API. ■ **Global applications**. Authenticates the incoming Kerberos token and identifies the user against the *username* identifier of all the applications available in API Gateway. On successful

Property	Description
	authentication and identification, API Gateway allows access to the API. ■ **Global applications and DefaultApplication**. 1. Authenticates the incoming Kerberos token and identifies the user against *username* identifier of all the applications available in the API Gateway. 2. On successful authentication and if no global application is identified, then API Gateway allows access to the API as default application. 3. In case if the authentication fails, then API Gateway does not allow access to the API. ■ If **Global applications and DefaultApplication** and **Allow anonymous** are selected: 1. Authenticates the incoming Kerberos token and identifies the user against *username* identifier of all the applications available in the API Gateway. 2. On successful authentication and if no global application is identified, then API Gateway allows access to the API as default application. 3. In case if the authentication fails, then API Gateway still allows access to the API. **Note:** You can use the username for further processing using the request transformation policy.
OAuth2 Token	Specifies using the OAuth2 token to identify the client, extract the access token from the HTTP request header, and verify the client's identity against the specified list of applications in API Gateway. By default, OAuth2 token is identified against the registered applications. **Note:** You can use the client id and other parameters for further processing using the request transformation policy.
OpenID Connect	Specifies using the OpenID (ID) token to identify the client, extract the client's credentials from the ID token, and verify the client's identity against the specified list of applications in API Gateway. Select one of the **Application Lookup condition**:

Property	Description
	■ **Registered applications**. Identifies the client's identity resolved as part of OpenID validation against all the applications registered to the API. On successful identification, API Gateway allows access to the API. ■ **Global applications**. Identifies the client's identity resolved as part of OpenID validation against all the applications available in API Gateway. On successful identification, API Gateway allows access to the API. ■ **Global applications and DefaultApplication**. Identifies the client's identity resolved as part of OpenID validation against all the applications available in API Gateway. If no global application is identified, then API Gateway allows access to the API as default application. **Note:** You can use the client id and other parameters for further processing using the request transformation policy.
SSL Certificate	Specifies using the SSL certificate to identify the client, extract the client's identity certificate, and verify the client's identity (certificate-based authentication) against the specified list of applications in API Gateway. The client certificate that is used to identify the client is supplied by the client to API Gateway during the SSL handshake over the transport layer or is added in the header of the request. The certificate included in the custom header can be in the following formats: ■ Base64 encoded PEM certificate with BEGIN CERTIFICATE and END CERTIFICATE delimiters ■ Non-Base64 encoded PEM certificate with BEGIN CERTIFICATE and END CERTIFICATE delimiters. ■ PEM certificate can be without BEGIN CERTIFICATE and END CERTIFICATE delimiters if a single certificate is added. ■ URL encoded PEM certificate with BEGIN CERTIFICATE and END CERTIFICATE delimiters. ■ URL encoded PEM certificate can be without the BEGIN CERTIFICATE and END CERTIFICATE delimiters if a single certificate is added. If the transport protocol is HTTP then API Gateway checks for the existence of a header and fetches the certificate from the certificate

Property	Description
	header. If the certificate is coming from the custom header, then API Gateway does not check the validity of the certificate. API Gateway identifies the application using the certificate. The certificate should be validated by some external entity before sending it to API Gateway in a custom header. If the transport protocol is HTTPS then API Gateway first tries to identify the application based on the certificate exposed by the client during the SSL handshake. If there is no client certificate or the identification based on the client certificate fails API Gateway tries to identify based on the certificate provided in the header. The header name is customizable and can be customized in the extended settings property, `customCertificateHeader`, the default value being `X-Client-Cert`. Select one of the **Application Lookup condition**: ■ **Registered applications**. Identifies the client's certificate against the *client certificate* identifier of all the applications registered to the API. On successful identification, API Gateway allows access to the API. ■ **Global applications**. Identifies the client's certificate against the *client certificate* identifier of all the applications available in API Gateway. On successful identification, API Gateway allows access to the API. ■ **Global applications and DefaultApplication**. Identifies the client's certificate against the *client certificate* identifier of all the applications available in API Gateway. If no global application is identified, then API Gateway allows access to the API as default application.
WS Security Username Token	This is applicable only for SOAP APIs. Specifies using the WS security username token to identify the application, extract the client's credentials (username token and password) from the WSSecurity SOAP message header, and verify the client's identity against the specified list of applications in API Gateway. **Note:** You have to enforce the Inbound Auth - Message policy with the property, Require WSS Username token, configured, so when Identify & Authorize policy is executed, the user details fetched are used to match with application's data to identify the application.

Property	Description
	Select one of the **Application Lookup condition**: ■ **Registered applications**. Authenticates the client's WSS username token and identifies the user against the *username* identifier of all the applications registered to the API. On successful authentication and identification, API Gateway allows access to the API. ■ **Global applications**. Authenticates the client's WSS username token and identifies the user against the *username* identifier of all the applications available in API Gateway. On successful authentication and identification, API Gateway allows access to the API. ■ **Global applications and DefaultApplication**. 1. Authenticates the client's WSS username token and identifies the user against the *username* identifier of all the applications available in the API Gateway. 2. On successful authentication and if no global application is identified, then API Gateway allows access to the API as default application. 3. In case if the authentication fails, then API Gateway does not allow access to the API. ■ If **Global applications and DefaultApplication** and **Allow anonymous** are selected: 1. Authenticates the client's WSS username token and identifies the user against the *username* identifier of all the applications available in the API Gateway. 2. On successful authentication and if no global application is identified, then API Gateway allows access to the API as default application. 3. In case if the authentication fails, then API Gateway still allows access to the API. **Note:** You can use the username for further processing using the request transformation policy.
WS Security X.509 Certificate	This is applicable only for SOAP APIs. Specifies using the WS security X.509 certificate to identify the client, extract the client identity certificate from the WS-Security SOAP

Property	Description
	message header, and verify the client's identity against the specified list of applications inAPI Gateway. **Note:** You have to enforce the Inbound Auth - Message policy with the property, Require X.509 Certificate, configured, so when Identify & Authorize policy is executed, the user details fetched are used to match with application's data to identify the application. Select one of the **Application Lookup condition**: ■ **Registered applications**. Identifies the client's X.509 certificate against the *client certificate* identifier of all the applications registered to the API. On successful identification, API Gateway allows access to the API. ■ **Global applications**. Identifies the client's X.509 certificate against the *client certificate* identifier of all the applications available in API Gateway. On successful identification, API Gateway allows access to the API. ■ **Global applications and DefaultApplication**. Identifies the client's X.509 certificate against the *client certificate* identifier of all the applications available in API Gateway. If no global application is identified, then API Gateway allows access to the API as default application.

Payload Element	Specifies using the payload identifier to identify the client, extract the custom authentication credentials supplied in the request represented using the payload identifier, and verify the client's identity against the specified list of applications in API Gateway.
	
	■ Select one of the **Application Lookup condition**:
	
	  ■ **Registered applications**. Identifies the client's payload against the *Payload Identifier* of all the applications registered to the API. On successful identification, API Gateway allows access to the API.
	
	  ■ **Global applications**. Identifies the client's payload against the *Payload Identifier* of all the applications available in API Gateway. On successful identification, API Gateway allows access to the API.
	
	  ■ **Global applications and DefaultApplication**. Identifies the client's payload against the *Payload Identifier* of all the applications available in API Gateway. If no global application is identified, then API Gateway allows access to the API as default application.

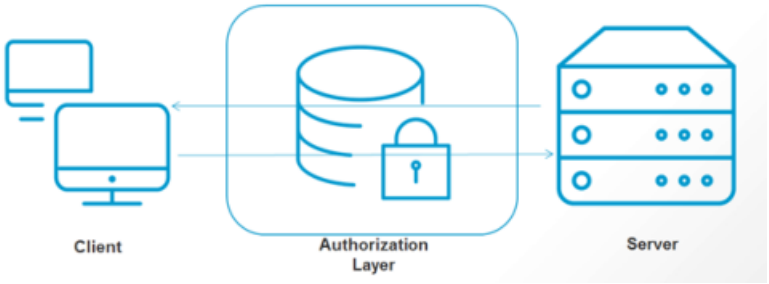
Property	Description
	In the Payload identifier section, click **Add payload identifier**, provide the following information, and click **Add**. ■ **Expression type**: Specifies the type of expression, which is used for identification. You can select one the following expression type: ■ **XPath**. *This is not applicable to a GraphQL API.* Provide the following information: ■ **Payload Expression**. Specifies the payload expression that the specified expression type in the request has to be converted to. For example: /name/id ■ **Namespace Prefix**. The namespace prefix of the payload expression to be validated. ■ **Namespace URI**. The namespace URI of the payload expression to be validated. **Note:** You can add multiple namespace prefix and URI by clicking +Add. ■ **JSONPath**. Provide the JSONPath for the payload identification. For example, $.name.id ■ **Text**. Provide the regular expression for the payload identification. For example, any valid regular expression. You can add multiple payload identifiers as required. **Note:** Only one payload identifier of each type is allowed. For example, you can add a maximum of three payload identifiers, each being of a different type.
HTTP Headers	Specifies using any header in the request to identify and authorize the client application against the list of applications with the identifier in API Gateway. Provide the following information: ■ Select one of the **Application Lookup condition**: ■ **Registered applications**. Identifies the client's header against the *Header Key - Value pair* identifier of all the applications registered to the API. On successful identification, API Gateway allows access to the API.

Property	Description
	■ **Global applications**. Identifies the client's header against the *Header Key - Value pair* identifier of all the applications available in API Gateway. On successful identification, API Gateway allows access to the API. ■ **Global applications and DefaultApplication**. Identifies the client's header against the *Header Key - Value pair* identifier of all the applications available in API Gateway. If no global application is identified, then API Gateway allows access to the API as default application.

OAuth Authentication in API Gateway

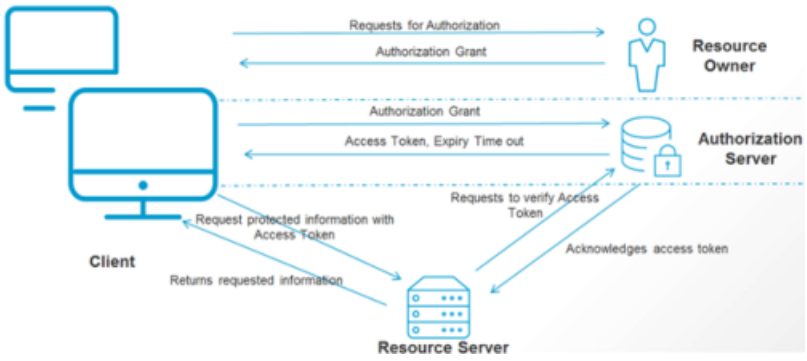
The Open Authorization is a flexible authorization framework for securing application access to protected resources of APIs.API Gateway can connect to the OAuth server of your choice to authorize client applications. API Gateway also includes an in-built authorization server that supports OAuth 2.0 for securing your APIs. This article explains how to use the OAuth 2.0 functionality in API Gateway.

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. For details, see https://tools.ietf.org/html/rfc6749.



OAuth adds a new layer called Authorization Layer for enabling access for protected information for the clients. Unlike the OAuth 1.0 protocol, OAuth 2.0 provides a rich authorization framework (not a protocol) with well-defined security properties. However, as a rich and highly extensible framework with many optional components, on its own, this specification is likely to produce a wide range of non-interoperable implementations. In addition, this specification leaves a few required components partially or fully undefined (for example, client registration, authorization server capabilities, endpoint discovery). Without these components, clients must be manually and specifically configured against a specific authorization server and resource server in order to interoperate.

The following diagram depicts the abstract flow of OAuth 2.0 Authorization framework. OAuth defines four roles: Resource Owner, Resource Server, Client, and Authorization Server.



Roles

- *Resource Owner* (or the end user). This is the holder of the protected resources that the client application accesses. The resource owner is typically a person (usually the end user), but could also be an application.
- *Resource Server* (or API Gateway). This is the server that stores the protected resources the application is trying to access and is capable of accepting and responding to protected resource requests using access tokens. API Gateway acts as a resource server.
- *Client Application* (or the Client). This is the application that is requesting access to protected resources on behalf of the resource owner and with its authorization.
- *Authorization Server*. This is the server that acts as an interface between the client application and end user, authenticates the end user, and issues access tokens to the clients after proper authorization. API Gateway can be configured to act as an OAuth 2.0 authorization server. You can configure API Gateway for use with a third-party OAuth 2.0 authorization server, such as OKTA and PingFederate.

Resource Server and Authorization Server interact with each other to verify the access tokens. There can be various levels of these interactions (3-legged OAuth, 4-legged OAuth, and so on). API Gateway can be used as an authorization server and as a resource server.

Clients can be of following types:

- *Confidential*. A confidential client is an application that is capable of keeping a client password confidential to the world. This client password is assigned to the client app by the authorization server. This password is used to identify the client to the authorization server, to avoid fraud. An example of a confidential client is a web app, where no one but the administrator can get access to the server, and see the client password.
- *Public*. A public client is an application that is not capable of keeping a client password confidential. For instance, a mobile phone application or a desktop application that has the client password embedded inside it. Such an application could get cracked, and this could reveal the password. The same is true for a JavaScript application running in the users browser. The user could use a JavaScript debugger to look into the application, and see the client password.

API Gateway as a Resource Server

When API Gateway acts as a resource server, it hosts the protected resources, accepts, and responds to the client applications' requests that include an access token. The client application sends the access token in the Authorization request header field using the Bearer authentication scheme. The resource server validates the access token locally or remotely if it cannot validate locally.

If the token is valid and the client application has privileges to access the protected resources, the resource server executes the request. If the access token is invalid, it rejects the request.

API Gateway as an Authorization Server

When API Gateway acts as an authorization server, it receives authorization requests from client applications. The authorization server handles the interactions between the client application, resource server, and resource owner for approval of the request.

As an authorization server API Gateway issues tokens to client applications on behalf of a resource owner for use in authenticating subsequent API calls to the resource server. The resource server hosts the protected resources, and can accept or respond to the protected resource requests using access tokens. If the client application is authorized to access the protected resources, the resource server executes the request. The authorization server retains the information about the access tokens it issues, including the user information. When a client presents an access token to the resource server, the resource server sends the token to the authorization server to ensure that the token is valid and that the requested service is within the scope for which the access token was issued. A *scope* is the definition of the resources that the client application can access on behalf of a resource owner. If the client application does not have privileges to access the resources, the resource server rejects the request.

Using API Gateway with an External Authorization Server

When API Gateway is the resource server, you must specify an authorization server. As an alternative to using API Gateway as the authorization server, you can use a third-party server as the authorization server. This allows API Gateway to validate access tokens issued by third-party servers and also allow to dynamically create clients in the third-party server.

Note:

- Before you configure API Gateway to use a third-party authorization server, make sure that the authorization server is compliant with the RFC 7662, OAuth 2.0 token introspection.
- From API Gateway release 10.3 onwards API Gateway supports multiple authorization servers.

To use an external authorization server, you must configure your third-party authorization server. This includes, but is not limited to, the following:

- To introspect the token, you should have a JWKS URI or you should create a client account that API Gateway uses to call the authorization server's introspection endpoint.

  Make a note of the client_id and client_secret values. You provide this information as part of defining the external authorization server alias for the API Gateway resource server.

Validation of JWT token of the external authorization server happens in the following ways:

Local Introspection	Remote Introspection
Validation of the JWT token happens within the gateway in the following methods: ■ **Using JWKS URI**. The external authorization server's signature is verified by using the public certificate in the JWKS URI. API Gateway's cache has a key as *kid* claim and its value is the certificate corresponding to the *kid* claim. The cache is populated on every restart of API Gateway by invoking the JWKS URI. In the runtime, while validating the token using the local introspection, the *kid* value from the incoming JWT is fetched and the corresponding certificate is retrieved from the cache and the signature validation happens. ■ **Using RSA**. The external authorization server's signature in the JWT is verified by the truststore defined in the local introspection configuration. ■ **Using HMAC**. If the authorization server uses HMAC algorithm, that means the signature validation of the JWT is performed using a shared key between the authorization server and API Gateway. You must specify the HMAC shared secret when creating the strategy of the application. The HMAC shared secret in the application is used to validate the authorization server's signature present in JWT.	Validation of the JWT token happens with the authorization server. Therefore, token caching is not possible in remote introspection. It has an introspection endpoint, which is used to validate the token. In addition, the client id and client secret are used to protect the endpoint, so that anonymous users cannot access the resource. To invoke an endpoint, you require a user; Gateway user is the one you can use to invoke the endpoint.

Make a note of the URL for the introspection endpoint. You provide this information as part of defining the external authorization server alias in the API Gateway resource server.

- Create the required scopes.
- Configure an alias to the authorization server.

Currently, API Gateway, by default, can be used with the following third-party authorization servers, but are not limited to, that are RFC 7662, OAuth 2.0 token introspection compliant:

- Okta
- PingFederate

You can also use other third-party authorization servers like Google, keycloak, and so on.

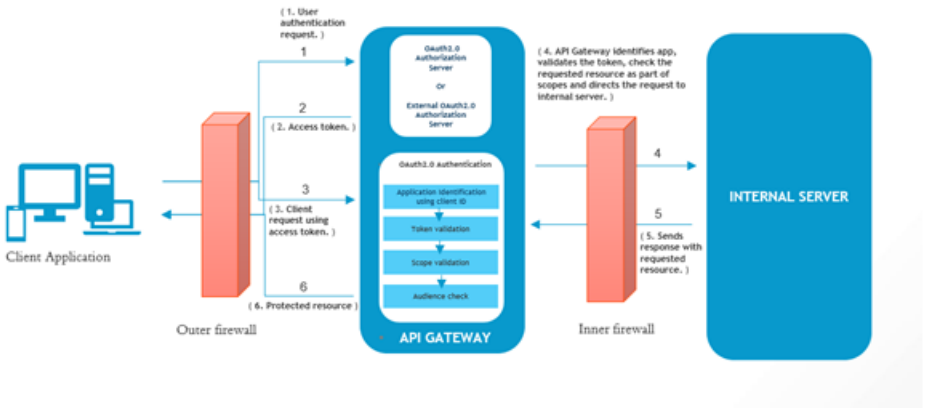
Authorizations for applications created from API Portal

When you create applications through API Portal, you must specify the required authorization server using the **watt.server.oauth.authServer.alias** settings in the Administration section of API Gateway.

If API Gateway is the authorization server, then provide `local` as the value of the **watt.server.oauth.authServer.alias** setting. Else, provide the name of the corresponding authorization server. For information on extended settings, see *webMethods API Gateway Administration*.

OAuth Authorization Workflow

The flow of authorization requests and responses between the end user, client application, authorization server, and resource server is as depicted in the following figure.



The OAuth authorization workflow is as follows:

1. The client sends user authentication request to the authorization server (local or external) to obtain an access token.
2. The authorization server validates the request, wuthenticates the client and generates an access token for the client.
3. The client uses this access token to send HTTP requests to API Gateway.

   API Gateway then performs the following:

a. Identifies the application using the clientId.

b. Validates the token locally or remotely if it is not possible locally.

c. Checks if the requested resource is part of the scopes in the token.

d. Checks the audience.

4. After validating the client, the request is directed to the Internal Server.

   If the access token is expired, authorization server returns a specific error response. The client application can then use Refresh Token to request a new access token. The Authorization Server returns a new access token that can be used to access the protected resource.

   Note:
   When a Policy violation event is logged in case of expired Oauth2 tokens, the application that is associated turn in to **Unknown**.

5. The Internal Server sends the response with requested resource to API Gateway.

6. API Gateway then sends the protected request resource to the Client.

Authorization Grant Types Supported

OAuth 2.0 provides several grant types, based on the work flow required by your application. The following grant types are supported by the API Gateway local Authorization Server. APIs can be enabled for more than one grant types. This table lists the grant types and the applications they are sutable for

Grant type	Suitable for
Authorization Code	Regular web applications executing on a server.
Implicit	Single page applications.
Resource Owner Password Credentials	Aplications that can be fully trusted with user credentials.
Refresh Token	To further secure applications that require the Authorization Code or Implicit grant types.
Client Credentials	Applications that involve only machine to machine interactions.

Authorization Code

The Authorization Code grant type enables API providers to open their APIs to unknown (but registered) third-party application developers. This grant enables a flow in which user credentials are never shared with the client application. Users are redirected to the authorization server to authenticate themselves. Only when users authenticate themselves and grant the required permissions to the client application can the client application access their resources.

When should you use the authorization code grant type?

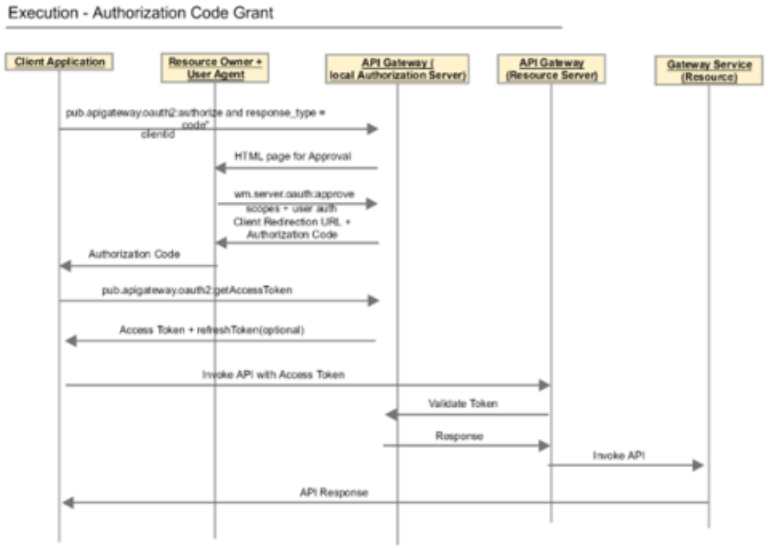
Use the authorization code grant type in scenarios where the API is being opened to unknown third-party application developers and exposing the user credentials is risky.

The user agent (web browser) is required component of this flow as many of the steps, such as user authentication, granting access permissions, passing an authorization code to the application and verifying the authenticity of the client application are performed by redirecting the web browser to different URLs. Therefore, it can be implemented in applications that either run in a web browser or can access a web browser.

It is suitable for:

- Web applications that execute on a server
- Mobile applications that can access a web browser

This diagram illustrates the flow for the Authorization Code grant type



For public clients, the *Authorization Code* grant type can be further secured by PKCE mechanism. For more details, see " Securing Access Token Calls with PKCE " on page 219.

Note:
API Gateway supports the confidential client authentication using Authorization headers. The confidential clients have to authenticate using their credentials, the client id and client secret combination. Few points to consider using the Authorization Code grant type:

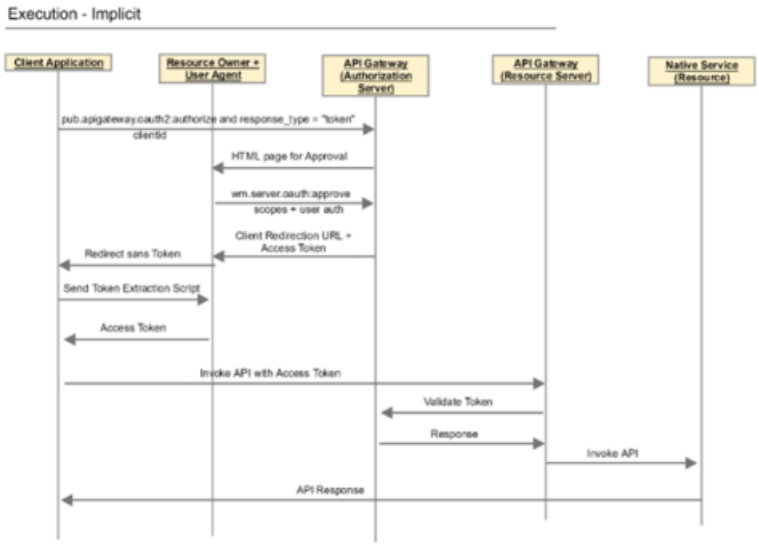
- If the property `watt.server.oauth.token.endpoint.auth=session` (the default value) and the confidential client already has a session when it comes to the token endpoint, the access to the endpoint is granted even if there are no credentials in the header.
- If the property `watt.server.oauth.token.endpoint.auth=credentials` or if the client does not already have a session, the confidential client must provide the client_secret in the Authorization header.
- API Gateway does not support the client secret in the body of the request for the Authorization code grant.

For details about the properties mentioned, see *webMethods Integration Server Administrator's Guide*.

Implicit

The implicit grant type is similar to the authentication code flow, but the access token is given to the user-agent to forward to the application. This exposes the token to the user and other applications on the user's device. Also, this grant type does not authenticate the identity of the application and relies on the redirect URI (that was registered with the service) to serve this purpose.

In addition, the implicit grant type does not support refresh tokens. This diagram illustrates the flow for the implicit grant type



The flow for the implicit grant type is similar to the flow for the authorization code grant type, except that the authorization server sends an access token to the user-agent instead of an authorization code. The user-agent then passes the access token to the client application.

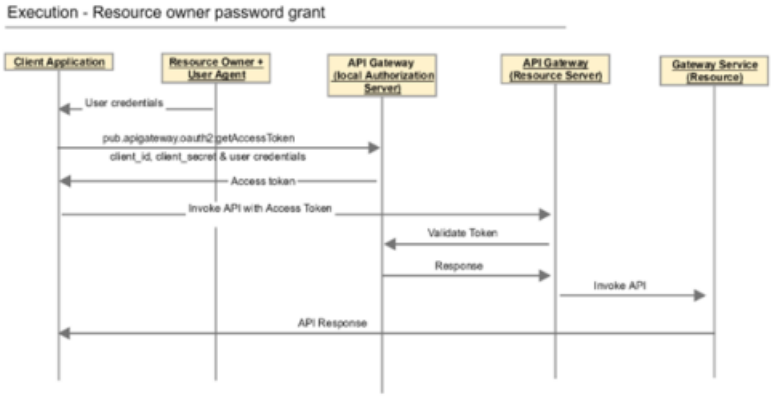
When should you use the implicit grant type? You should use the implicit grant type only with applications developed and published by trusted parties.

It is suitable for single-page applications that execute in the web browser as they cannot execute the flow required by the more secure and elaborate Authentication Code grant type

Resource Owner Password Credentials

In the flow for the resource owner password credentials grant type, users enter their username and password for a resource directly in the client application. The client application then passes the credentials to the authentication server to authenticate the user and obtain an access token.

This diagram illustrates the flow for the Resource Owner Password Credentials grant type.



Client application asks the user for her credentials and passes them to the Authorization Server. In response, the Authorization Server sends the access token directly to the client application.

When should you use the resource owner password credentials grant type? The resource owner password credentials grant type can be used only when users trust the client application with their credentials for a resource. This is usually the case when the client application is from the same party that hosts the resource or service that the client application is trying to access.

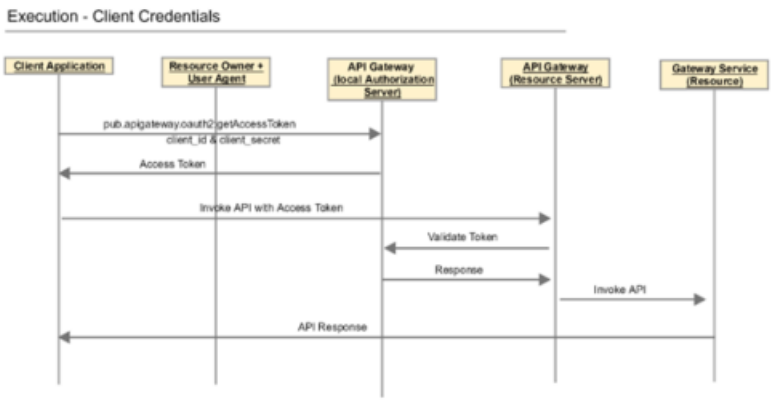
This grant type should only be used only if other grant types are not viable.

Client Credentials

This is the grant type used to obtain access tokens for client-only authentication.

The client credentials grant type is similar to the resource owner password credential grant type. The difference is that instead of user credentials, the client application provides its own client and secret. This grant type is intended for flows in which users are not involved.

This diagram illustrates the flow for the Client Credentials grant type.

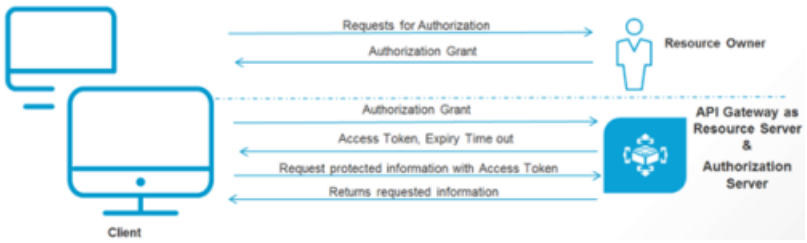


The steps are similar to the resource owner password credential flow. Clients use their client IDs and secrets to identify themselves. On success, the Authorization Server returns access tokens. Avoid using refresh tokens in this flow as it increases the risk of exposing the client credentials.

When should you use the client credentials grant type? In this grant type, users do not need to authenticate themselves or authorize access to a particular resource or service. This grant type is suitable for scenarios such as client applications accessing their own resources. For example: a client application retrieving data from its own account

Securing APIs using OAuth 2.0 in API Gateway with Local Authorization Server

This use case explains, with a simple example, how to secure an API with OAuth2 authentication with API Gateway configured as both Resource Server and Authorization Server. The diagram illustrates the work flow for this use case.



Note:
Using the Authorization Code Grant type in this example. To understand the grant types supported by API Gateway, see "Authorization Grant Types Supported" on page 128

Actors

- Developers with basic knowledge about webMethods API Gateway, Integration Server, OAuth2 architecture
- Customers with basic knowledge about webMethods API Gateway, Integration Server, OAuth2 architecture

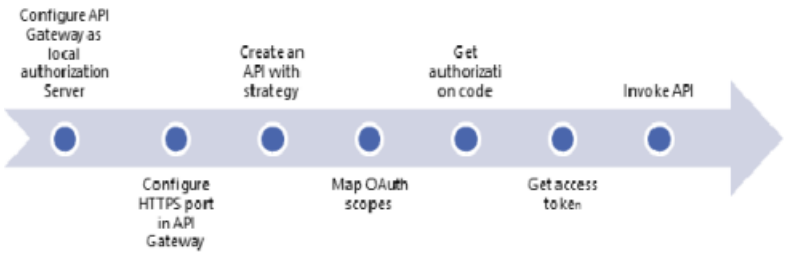
Before you begin

Ensure that you have:

- Installed Integration Server with API Gateway
- Knowledge about any REST Client
- API Gateway up and running

Basic Flow

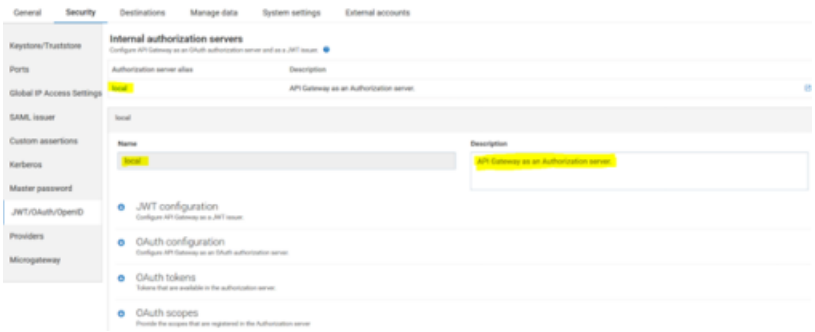
The following diagram depicts the high level steps of the basic flow for this use case.



Step 1: Configure API Gateway as local Authorization Server

In API Gateway, by default the internal authorization server for JWT, OAuth, or OpenID is set as `local`. It means that API Gateway acts as an authorization server.

1. Expand the menu options in the title bar and select **Administration**.
2. Select **Security** > **JWT/OAuth/OpenID**. Click **local**. All configurable settings for the local authorization server appear.
3. Expand **local**.
4. Expand **OAuth Configuration**, **OAuth tokens**, and **OAuth scope** sections.
5. In this use case, retain the default values of **Authorization code expiration interval (seconds)** and **Access token expiration interval (seconds)** in the **OAuth Configuration** section. You can modify these values as required.



6. Create OAuth Scopes,

   OAuth scopes allow you to limit the access level that is granted to an access token. In this use case, we define one OAuth Scope (OAuth2Scopes) to limit the resource usage.

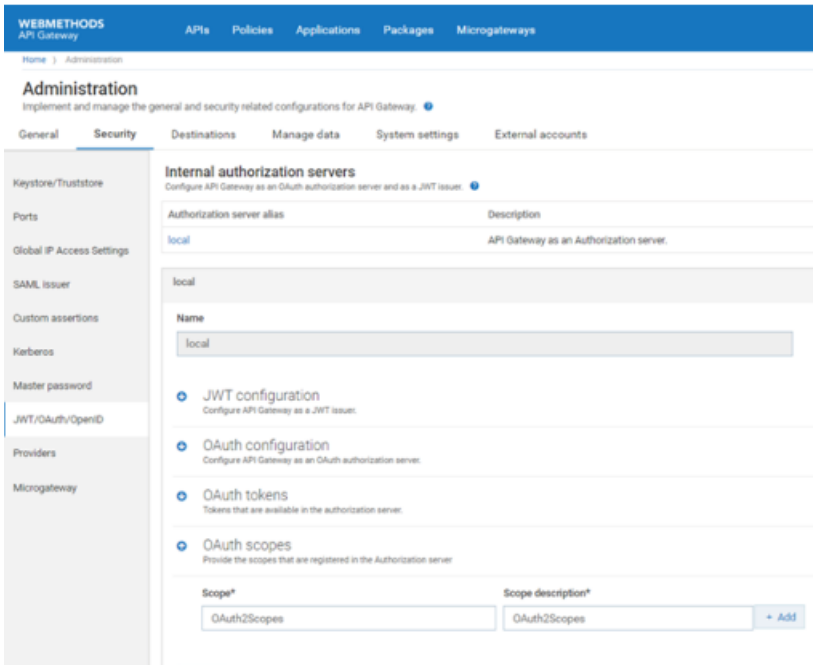
   a. Expand **OAuth scopes** to add scope.

   b. Provide values in the following fields

      - **Provide Scope**: *OAuth2Scopes*
      - **Scope description**: *OAuth2Scopes*

   c. Click **Add**.

d. Click **Update**. The OAuth scope is created.



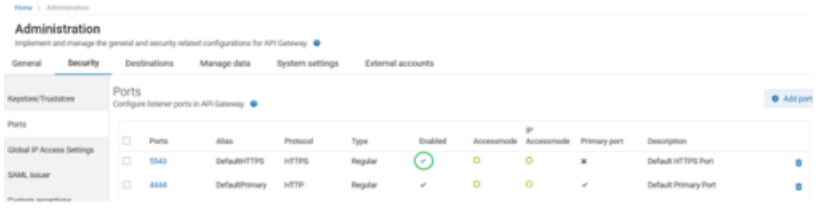
Step 2: Configure HTTPS port in API Gateway

In this example, authorization is through HTTPS. To enable API Gateway to accept request through https, define a new HTTPS port in API Gateway or use the default HTTPS port provided by API Gateway. The default HTTPS port is `5543` and it must be enabled from the Ports section of API Gateway.

1. Expand the menu options in the title bar and select **Administration**.
2. Select **Security** > **Ports**.
3. You can do one of the following:

   - Use the default HTTPS port

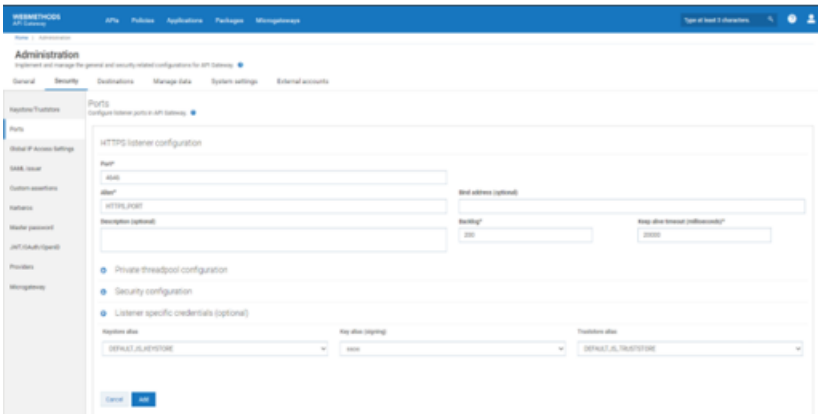
     Click the **Enable** field next to the 5543 port and enable it. The port is enabled.



   - Add a new HTTPS port

     1. Click **Add ports** on the Ports page.
     2. Select `HTTPS` from the drop-down, and click **Add**.

3. Provide `4646` in the **Ports** field.
4. Provide `HTTP_Port` in the **Alias** field
5. *Optional*. Expand **Listener specific credentials**.
6. Select `DEFAULT_IS_KEYSTORE` from the **Keystore** drop-down list. The Key alias value appears automatically.
7. Select `DEFAULT_IS_TRUSTSTORE` from the **Truststore** drop-down list.
8. Click **Add**. The port appears on the Ports screen.

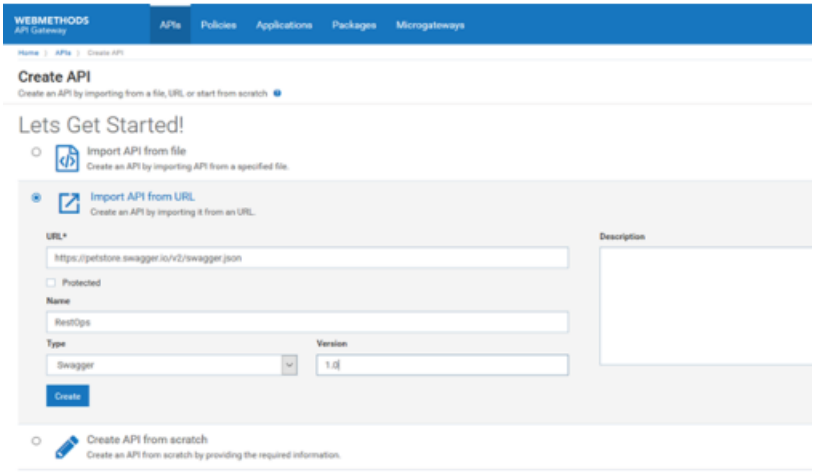


9. Click the **Enable** field next to the 4646 port to set the port to default.

Step 3: Create an API with strategy

In this example you create an API by importing from the URL https://petstore.swagger.io/v2/swagger.json.

1. In the **APIs** tab click **Create API**.
2. Select **Import API from URL**.
3. Provide the following details.
   - **Name**: `RestOps`
   - **Type**: `Swagger`
   - **Version**: `1.0`
4. Click **Create**. The API is created and the API details page for the API appears.



5. Enforce OAUTH2 policy on the API.

   You enforce OAuth2 policy on the RestOps API. This policy ascertains that an OAuth token is required to access this API.

   a. Click **API** in the title navigation panel.

   b. Click **RestOps**.

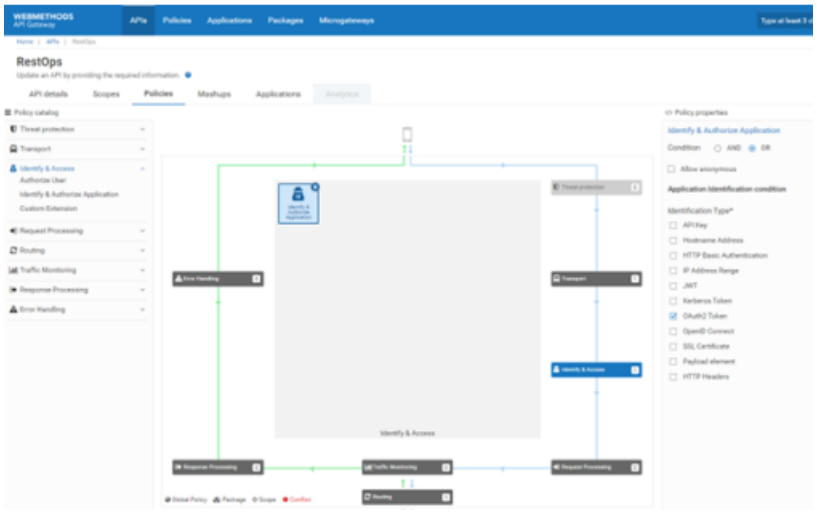
   c. Click the **Policies** tab.

   d. Click **Edit**.

   e. Click **Identify & Access** in the Policy catalog section.

   f. Select the **OAuth2 Token** check box in the **Identification Type** field of the Application Identification section and save the changes.

   g. Click **Activate** to activate the API on the API details page of the API.



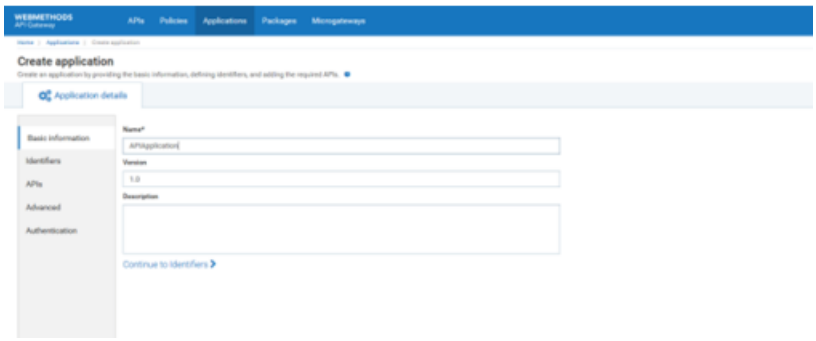
6. Create an application with strategy and register it to an API.

a. Click **Applications** in the title navigation bar.

b. Click **Create application**.

c. Provide the **Name**: `APIApplication`

d. Click **Continue to Identifiers**.



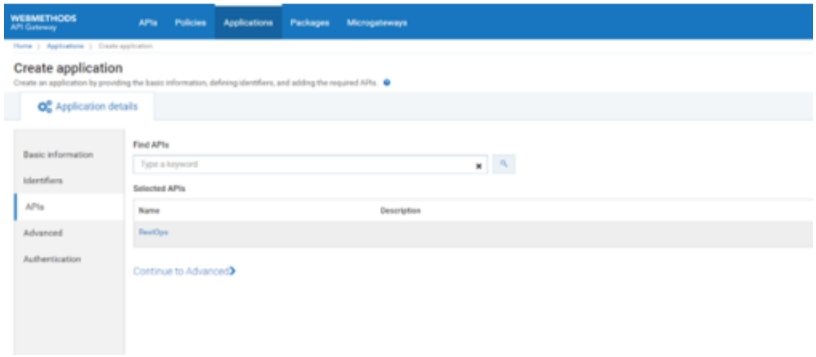
e. Click **Continue to APIs**.

f. Search the API RestOps, by typing `RestOps` in the **Find APIs** text box.

The RestOps API appears in the **Selected APIs** section.

g. Select the RestOpsAPI.

h. Click **Continue to Advanced**



i. Click **Continue to Authentication**.

j. Click **Create strategy**.

A strategy is a way to authenticate the incoming request and provides multiple authentication mechanisms or multiple authorization servers for a single authentication scheme. You can create multiple strategies authorized by an API for an application.

k. Provide the **Name** as `AppStrategy`.

l. Enable the toggle button **Generate credentials** to generate the credentials dynamically in the authorization server. The client-id and client-secret get created automatically.

m. Select `Confidential` from the **Application type** drop-down list.

n. Specify the redirect URIs that the authorization server can use to redirect the resource owner's browser during the grant process. In this example, provide `www.example.com`, which is not a valid URL.

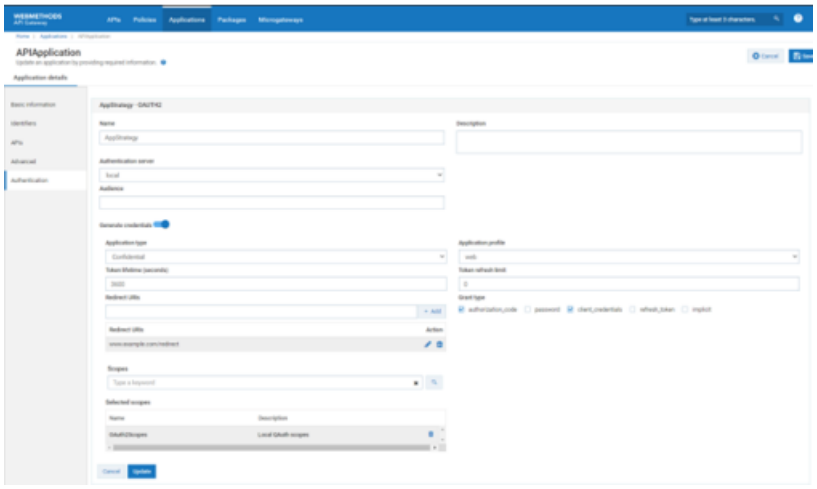
o. Select the required Grant types. In this example, the selected grant types are `authorization_code` and `client_credentials`.

p. Provide `OAuth2Scopes` in the **Scope** text box and click the search icon. The matching OAuth2Scopes appear.

q. Click the **+** icon next to the required scope to add the scope to the strategy.

r. Click **Add** to add the strategy.

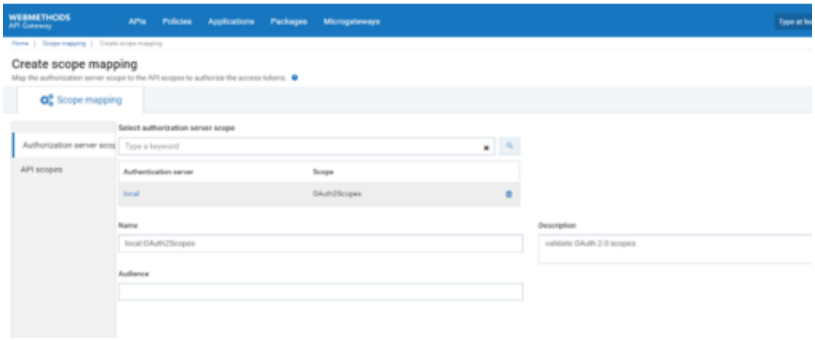
s. Click **Save**. The application is registered.



Step 4: Map OAuth scopes

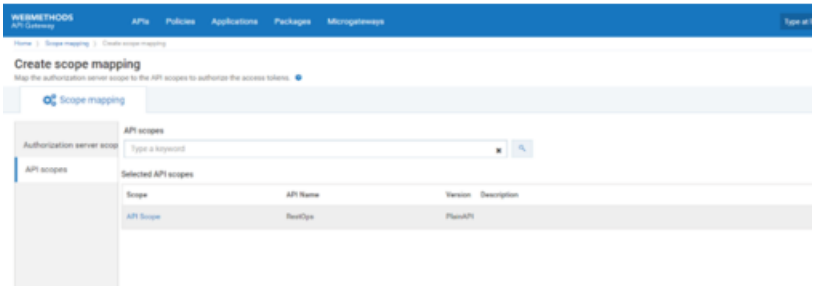
After registering an application, you must map the scope defined in the Authorization server with the APIs in API Gateway to authorize the access tokens used to access the protected resources. You can map either a complete API or parts (resources or methods) of an API to the scope or can add the scope details and modify the scope details as required from the OAuth/OpenID scopes page. In this example you select the OAuth2Scopes scope.

1. Expand the menu options in the title bar and select **OAuth/OpenID scopes**.
2. Click **Map scope**.
3. Type `OAuth2Scopes` in **Select authorization server scope** and select the Authorization server scope from the search list.



4. Select **API scopes** in the left pane.
5. Type `RestOps` or API Scope, which is to be linked to the authorization server, in the **API scopes** text box.
6. Click **Save**.

   The authorization server scope is mapped to the selected API scopes and the authorization scope appears in the in the scopes list.

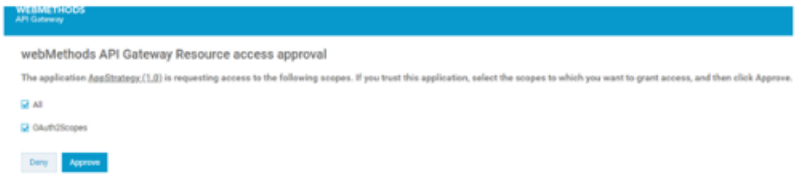


Step 5: Get authorization code

1. In the title bar click **Application**.
2. Click the application `APIApplication`.
3. Click **AppStrategy**. Note the **Client id** and **Client secret** values. You require these to get access token.

4. Access the Authorization code using the URL
   `https://<machinename>:555/invoke/pub.oauth/authorize?client_id=edb424fde142580a5b572e152&redirect_uri=http://example.com/redirect&response_type=code&state=121`

Approve the approval page and provide Integration server credentials if it prompts with a login page.



5. Receive the Authorization code through URL.

   You might get an error, but the authorization code is present in the URL as you are providing dummy redirect URI.

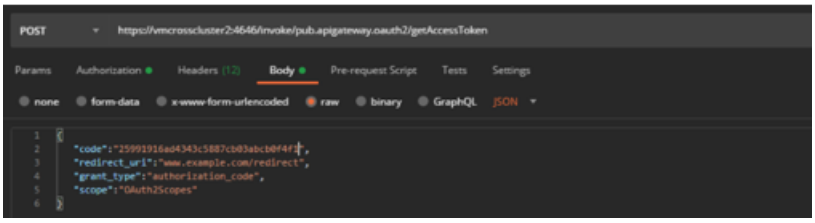
   The output is as follows:

```
...ww.example.com/redirect?code=25991916ad4343c5887cb03abcb04f1&grant_type=authorization_code&redirect_uri=http%253A%252F%252Fexample.com%252Fredirect&state=12&scope=View
```

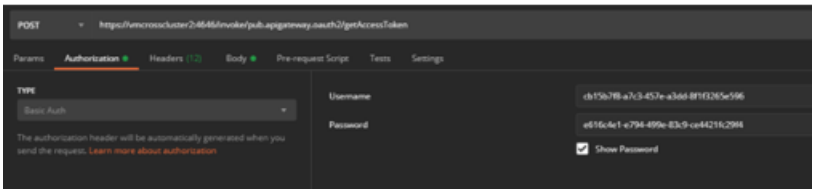
Step 6: Get access token

1. Open Postman or any other REST Client
2. Copy the authorization code received.
3. Make a POST call to the following URL, with the hostname of the system where API Gateway is installed in place of localhost and use external port instead of default port
   `https://<machinename>:4646 /invoke/pub.apigateway.oauth2/getAccessToken`
4. In the Body provide the following request payload

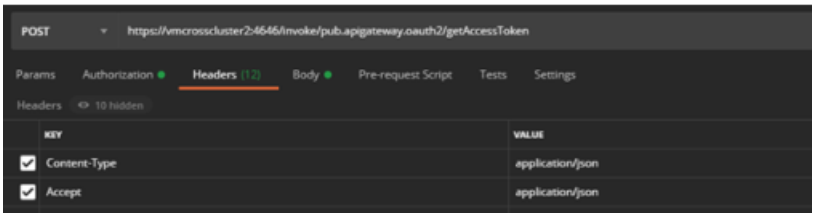
```
{
"code":"25991916ad4343c5887cb03abcb04f1",
"redirect_uri":"http://example.com/redirect",
"grant_type":"authorization_code",
"scope":"OAuth2Scopes"
}
```



5. Provide the client id and client secret in Basic Auth.



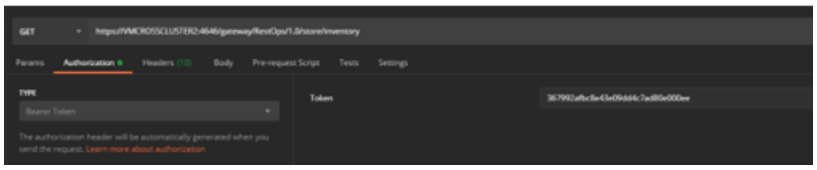
6. In **Headers** select `Content-Type` as `application/json` and `Accept` as `application/json`



7. The output gives the access token

Step 7: Invoke API

Invoke the API with the access token



Alternative Steps

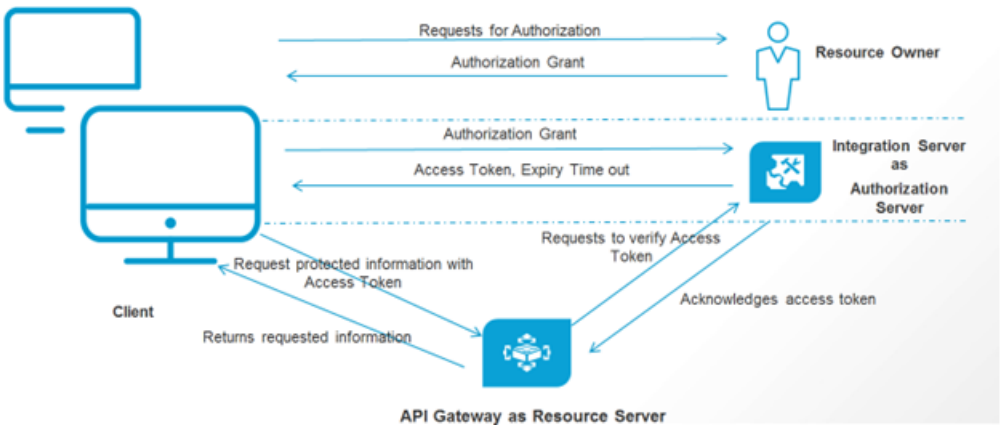
You can perform the same use case with Client Credentials as well. Acquire the access token through client credentials by providing the client_id and client_secret.

1. Open Postman or any other Rest Client.
2. Make a POST call to the following URL, with the hostname of the system where API Gateway is installed in place of localhost and use external port instead of default port.
   `https://<machinename>:4646 /invoke/pub.apigateway.oauth2/getAccessToken`
3. In Headers provide Content-Type as application/json (if not specified)
4. In the Body provide the following Request payload

```
{
   “client_id”:”xxxxx”,
   “client_secret”:”xxxxx”,
   “ grant_type”:”client_credentials”
}
```

Securing APIs using OAuth 2.0 with API Gateway as Resource Server and Remote Integration Server as Authorization Server

This use case explains, with a simple example, how to secure an API with OAuth2 authentication with API Gateway configured as Resource Server and remote Integration Server configured as the Authorization Server. The diagram illustrates the work flow for this use case.



Actors

- Developers with basic knowledge on webMethods API Gateway , Integration Server, OAuth2 architecture
- Customers with basic knowledge about webMethods API Gateway, Integration Server, OAuth2 architecture

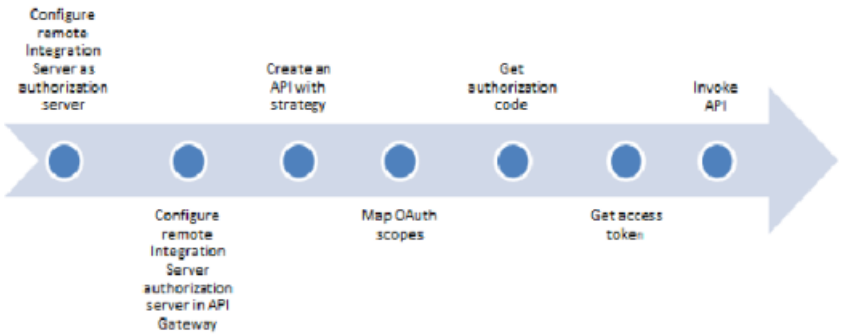
Before you begin

Ensure that you have:

- Installed Integration Server with API Gateway
- Knowledge on any Rest Client
- API Gateway up and running

Basic Flow

The following diagram depicts the high level steps of the basic flow for this use case.



Step 1: Configure remote Integration Server as authorization server

1. Disable HTTPS in OAuth in Integration Server.

   In this example you disable HTTPS in the remote Integration Server OAuth as authorization is through HTTP.

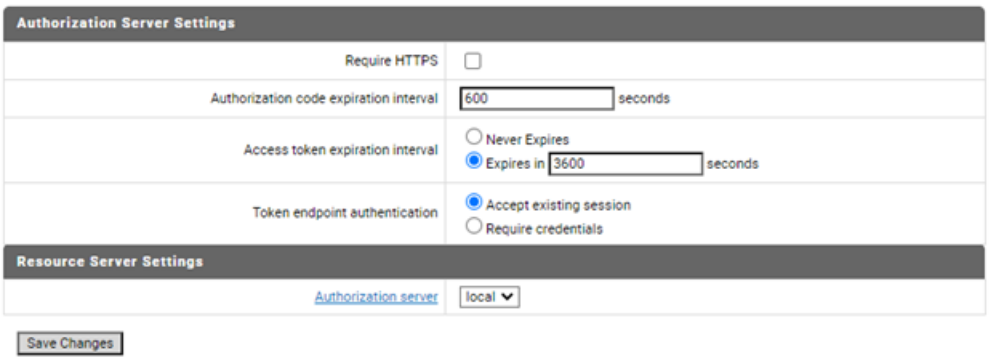
   a. Login to Integration Server with the URL. `https://mahinename:5555`

   b. Navigate to **Security** > **OAuth**.

   c. Click Edit **OAuth Global Settings**.

   d. Uncheck `Require HTTPS`.

   e. Click **Save Changes**.



2. Configure the following extended settings.

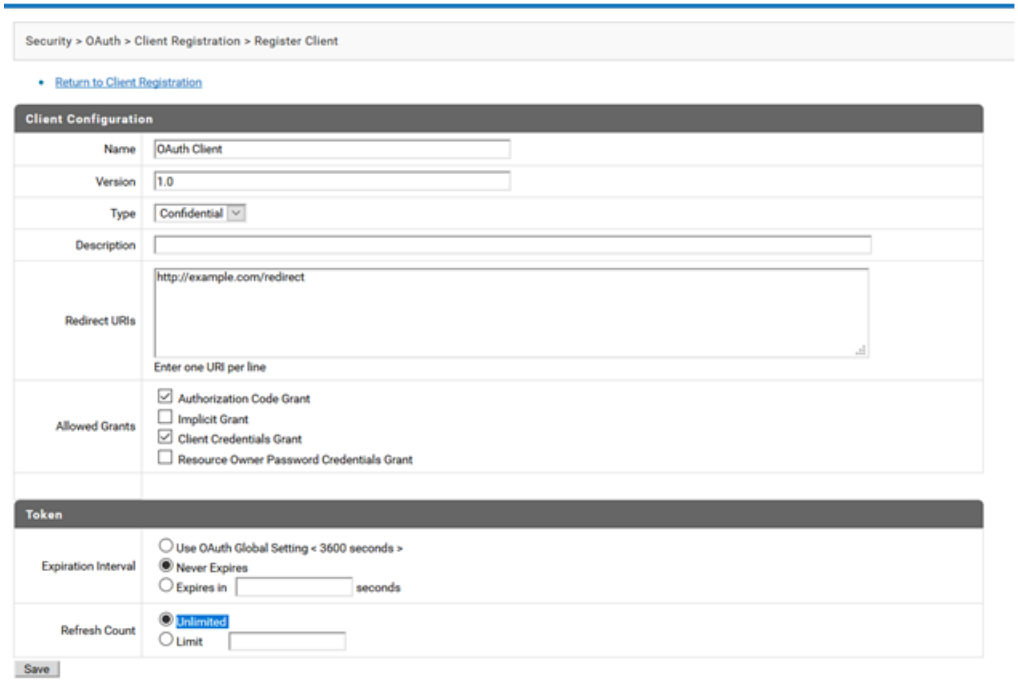
   a. Navigate to **Settings** > **Extended**.

   b. Click **Show** and **Hide** Keys.

   c. Select **watt.server.oauth.requireHTTPs** and `watt.server.oauth.requirePost .`

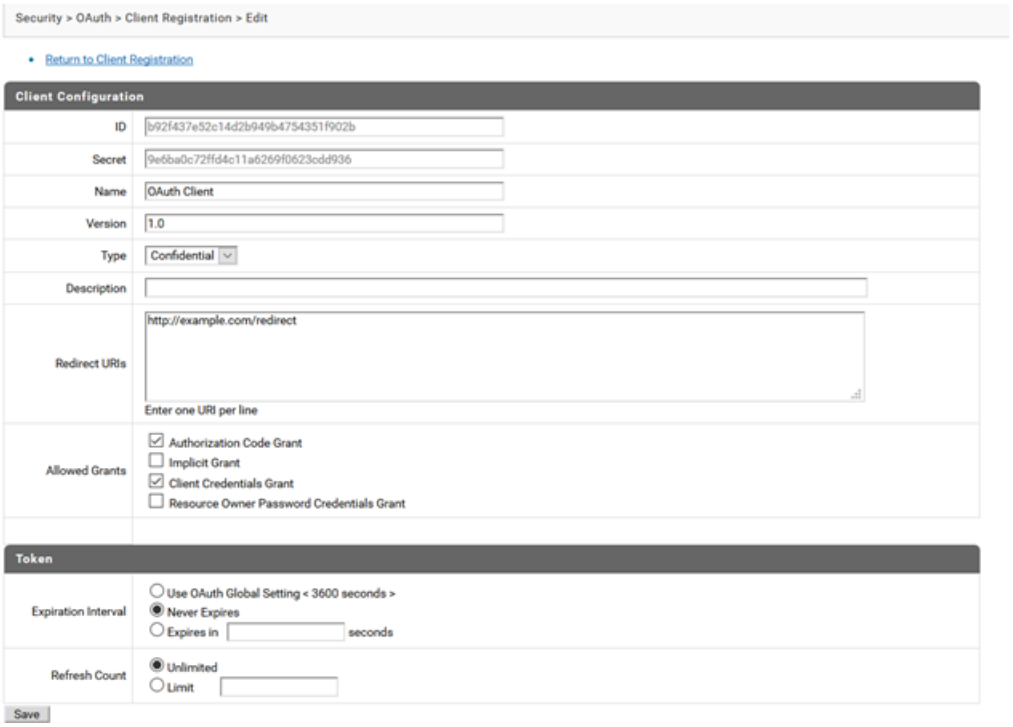
   d. Set both the value as `false`.

3. Create Client.
   a. Navigate to **Security** > **OAuth**.
   b. Click **Client Registration**.
   c. Click **Register Client**.
   d. Provide the following information:
      - **Name**: `OAuth Client`
      - **Version**: `1.0`
      - **Type**: `Confidential`
      - **Redirect URIs**: `https://example.com/redirect`
      - **Allowed Grants**: `Authorization Code Grant`, `Client Credentials Grant`
      - **Expiration Interval**: `Never Expires`
      - **Refresh Count**: `Unlimited`
   e. Click **Save** .



Onsuccessfully saving the following message displays: Successfully registered client OAuth Client, version 1.0. To use the client credentials grant, this client must be given ACL permissions to access the resources in the client's scopes.

f. Click **OAuth Client (1.0)**. Note down the ID and Secret that are generated.



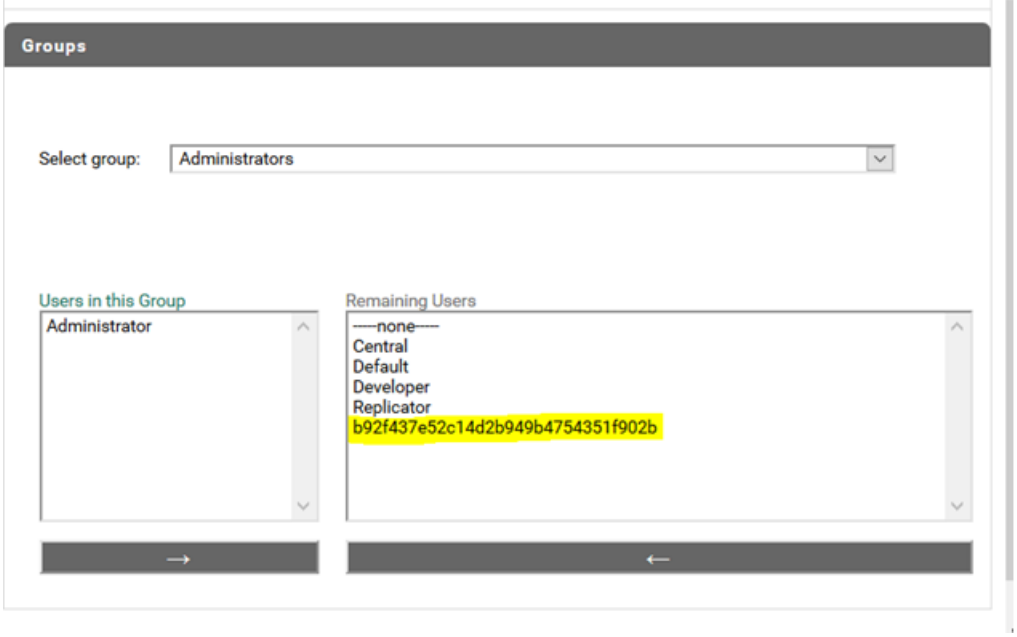
4. ACL Permission to Client

Provide Permission to client application by adding it to the Administrators group.

a. Navigate to **Security** > **User Management**

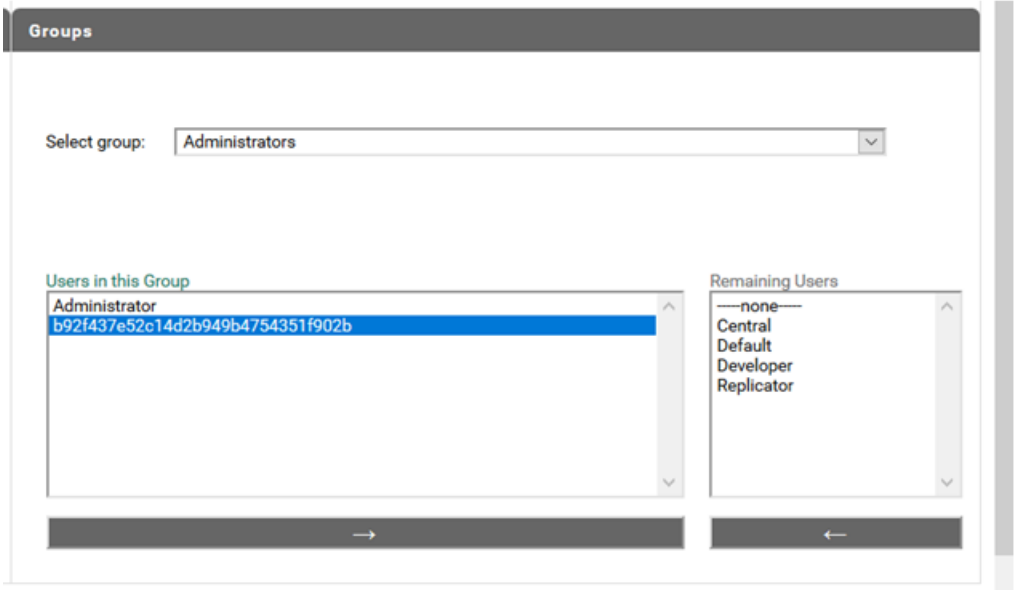
b. Under Groups, select the group **Administrator** (by default Administrators is selected)

c. The client id is present in the **Remaining Users** section,



d. Move the client ID under **Users in this Group**.

e. Click **Save Changes**.



5. Create Scope and associate with client

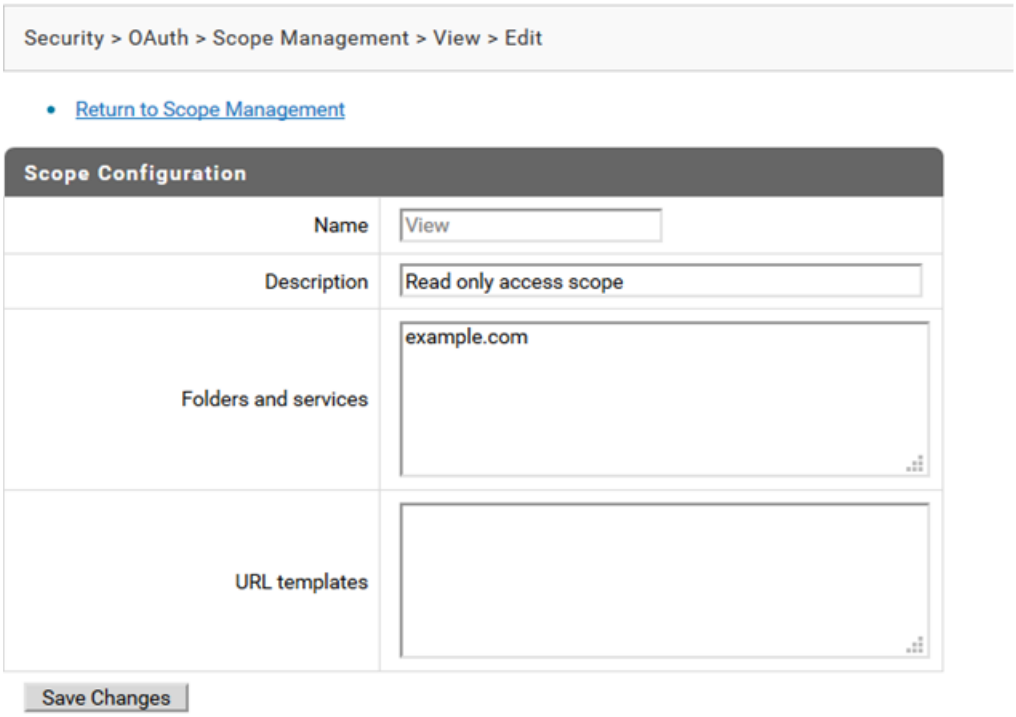
a. Navigate to **Security** >**OAuth**.

b. Click **Scope Management**.

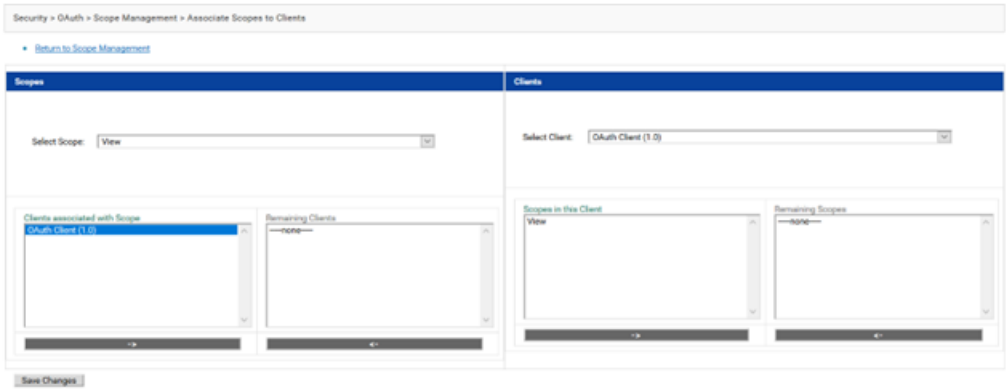
c. Click **Add Scope**.

d. Provide the following information:

- **Name**: `View`
- **Description**: `Read only access scope`
- **Folders and services**: `example.com`



- Click **Associate Scopes to Client**.
- Select `OAuth Client(1.0)` in the **Remaining Clients** section and move it to the **Clients associated with Scope** section.
- Click **Save Changes**.



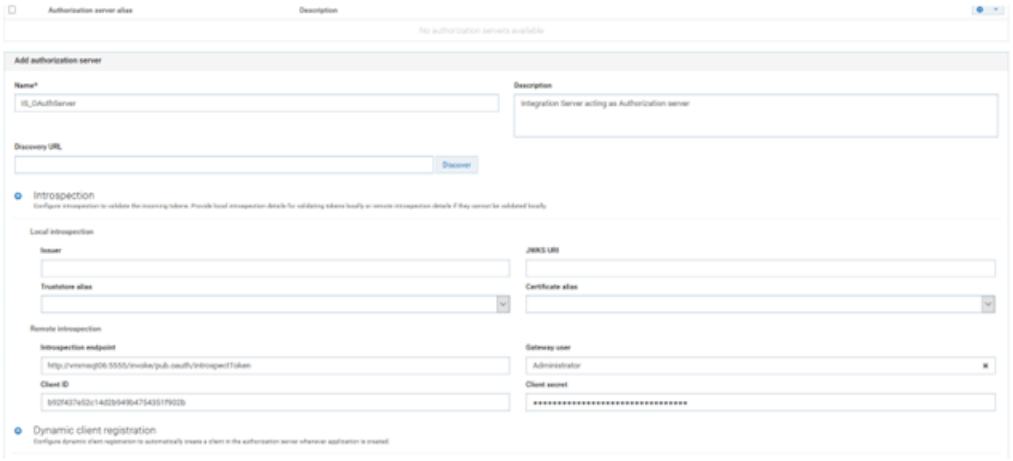
Step2: Configure remote Integration Server authorization server in API Gateway

Configure the remote Integration Server as an external authorization server in API Gateway.

1. Navigate to **Menu** >**Administration**
2. Click **Security** > **JWT/OAuth/OpenID**
3. Click **External authorization servers** > **Add authorization server**.
4. Provide the following:
   - **Name**: `IS_OAuthServer`
   - **Description (optional)**: `Integration Server acting as Authorization server`
5. Click **Introspection**.

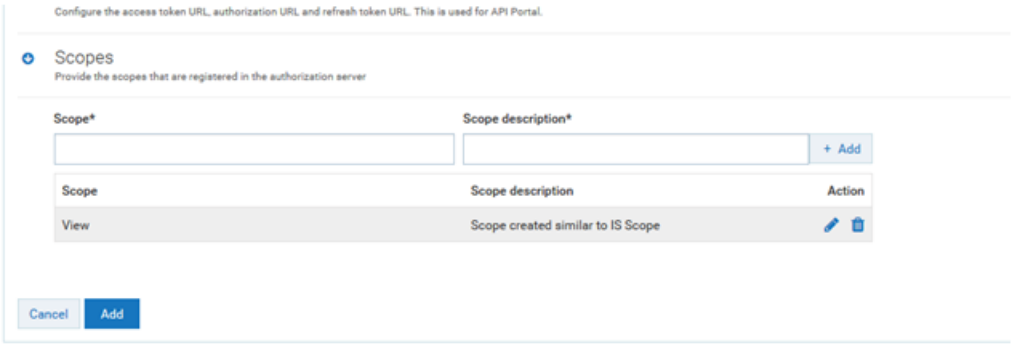
   You use Introspection to authenticate the token. It happens ollows

   - **Local introspection**. Validating the token within the gateway. This is done using the JWKS URI or the public certificate of the issuer. In this case the token should necessarily be JWT.
   - **Remote introspection**. Validating the token with the authorization server. It has introspection endpoint, which is used to validate the token. In addition, the client id and client secret are used to protect the endpoint, so that anonymous users cannot access to the resource. To invoke an endpoint you require a user; Gateway user is the one you cam use to invoke the endpoint. There is no support for token caching in remote introspection.
6. Provide the following in the Remote introspection section
   - **Introspection endpoint**: `http://{7BIntegrationserver_machinename}:5555/invoke/pub.oauth/introspectToken`
   - **Gateway user**: `Administrator`
   - **Client ID** and **Secret**: Use ID and secret noted down earlier.

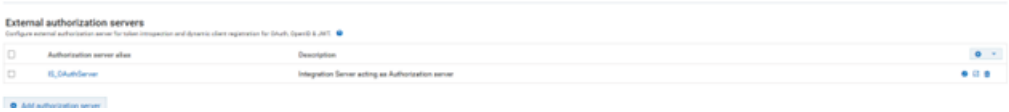


7. Create Scopes in External Authorization servers page

   - Click **Scopes** under **Add authorization server** Create Scope similar to the one in Integration Server.
   - Provide the Scope: **View**
   - Provide Description: **Scope created similar to IS Scope**
   - Click **Add**



   - Integration Server should be listed in External authorization servers

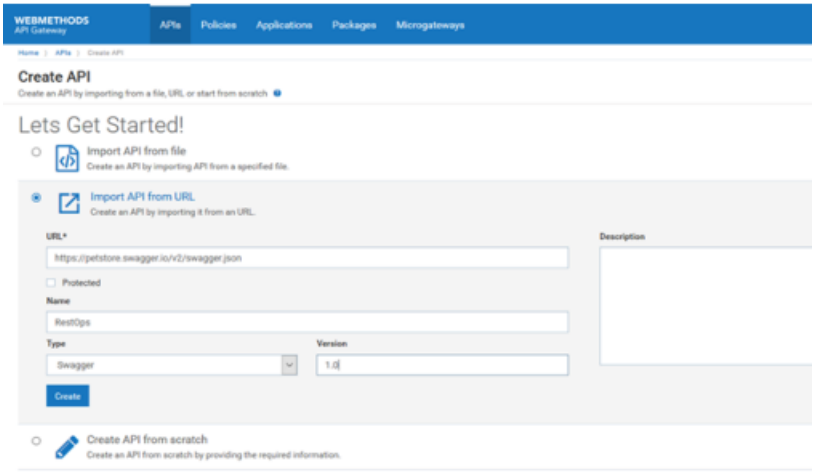


Step 3: Create an API with strategy

In this example you create an API by importing from the URL `https://petstore.swagger.io/v2/swagger.json`, enforce it with OAuth2 policy, and create an applicatin with strategy and associate it with the API. .

1. In the **APIs** tab click **Create API**.

2. Select **Import API from URL**.
3. Provide the following information.
   - **URL**: `https://petstore.swagger.io/v2/swagger.json`
   - **Name**: `RestOps`
   - **Type**: `Swagger`
   - **Version**: `1.0`
4. Click **Create**. The API is created and the API details page for the API appears.



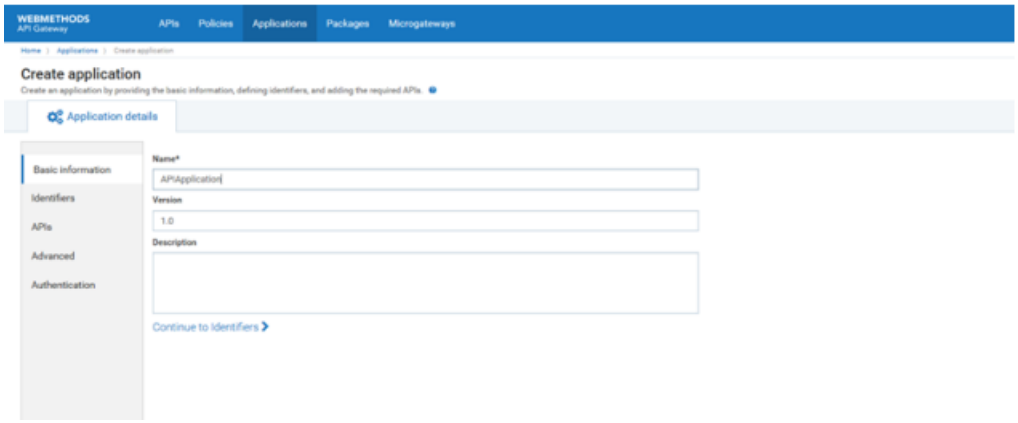
5. Create an Application with Strategy

   For creating an Application, follow the below steps.

   a. Click **Applications** in the title navigation bar.

   b. Click **Create application** provide the **Name**: `API Application`

   c. Click **Continue to Identifiers**.



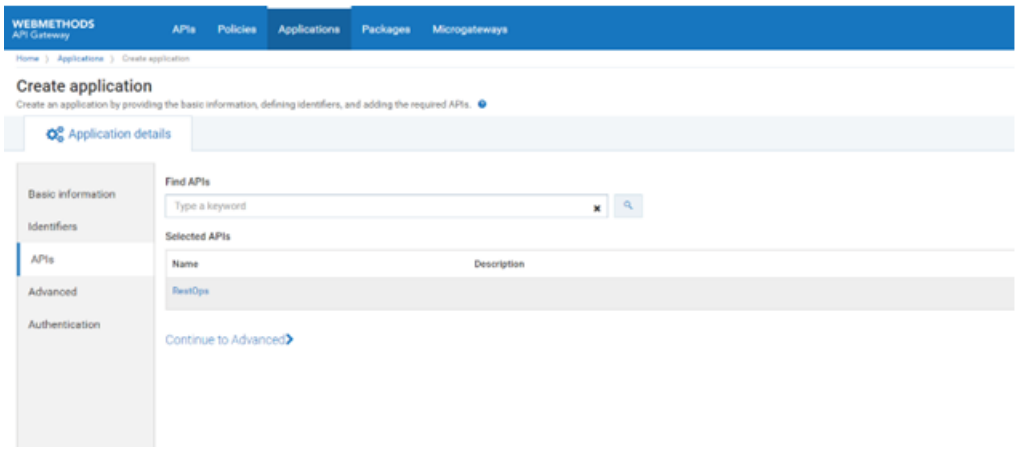
   In this use case, no details are provided for the Identifier page.

d. Search the API RestOps, by typing `RestOps` in the **Find APIs** text box.

The RestOps API appears in the **Selected APIs** section.

e. Select the RestOpsAPI.

f. Click **Continue to Advanced**.



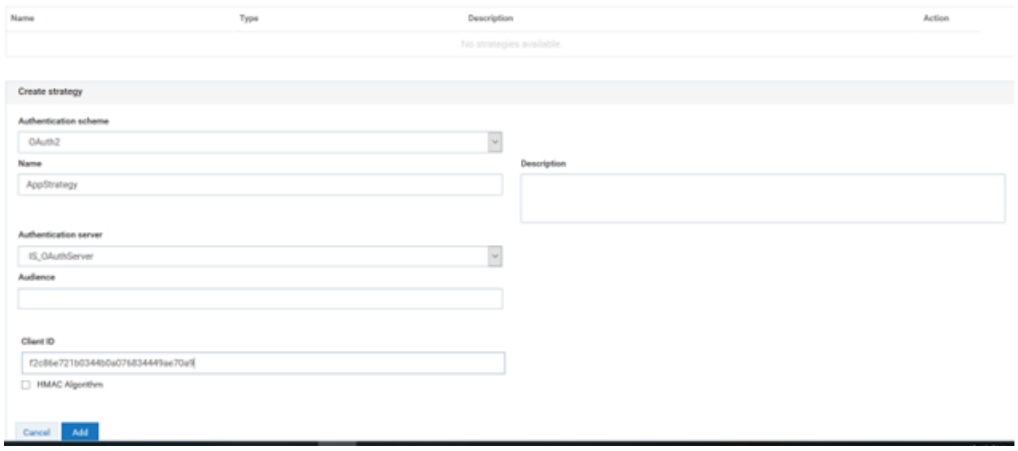
g. Click **Continue to Authentication** .

h. Click **Create Strategy** . A strategy is a way to authenticate the incoming request and provides multiple authentication mechanisms or multiple authorization servers for a single authentication scheme. One can create multiple strategies authorized by an API for an application.

i. Provide the **Name**: `AppStrategy`.

j. Click**Add**

k. Click **Save**



6. Enforcing OAUTH2 policy on the API

Enforce OAuth2 policy to the created API . This policy ascertains that an OAuth token is required to access this API..

a. Click **API** in the title Navigation Panel.

b. Click **RestOps**.

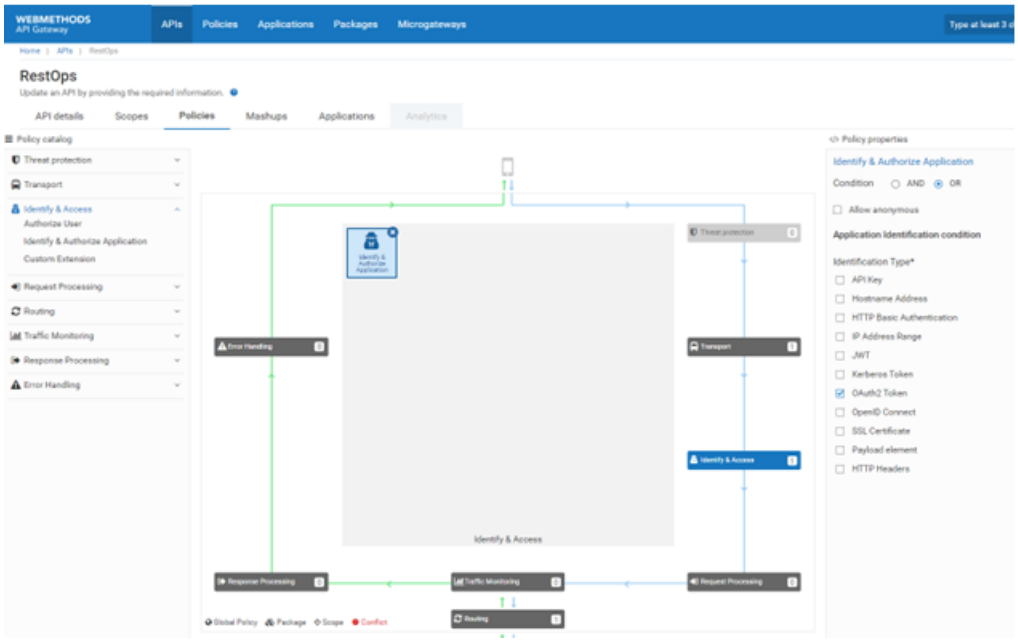
c. Click **Policies** tab.

d. Click **Edit**

e. Click**Identify & Access** from the Policy catalog.

f. Select the identification type as **OAuth2 Token** and **Save**.

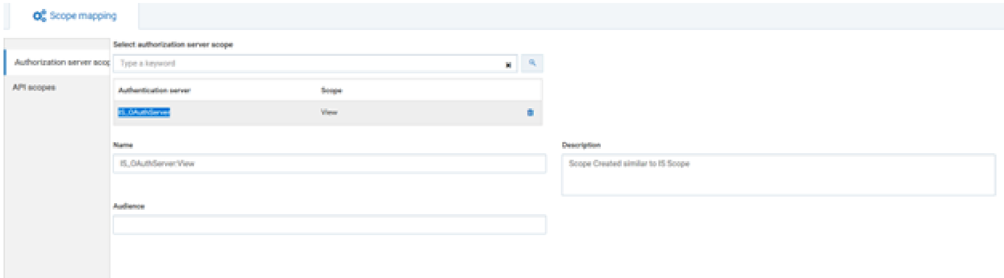
g. The API can be Activated, or can navigate to the APIs in the title navigation bar and **Activate**.



Step 4: Mapping OAuth scopes

Now map the scope that was defined in the Authorization server with the APIs in API Gateway to authorize the access tokens to be used to access the protected resources. One can map either a complete API or parts (resources or methods) of an API to the scope or can add the scope details and modify the scope details as required from the OAuth/OpenID scopes page.

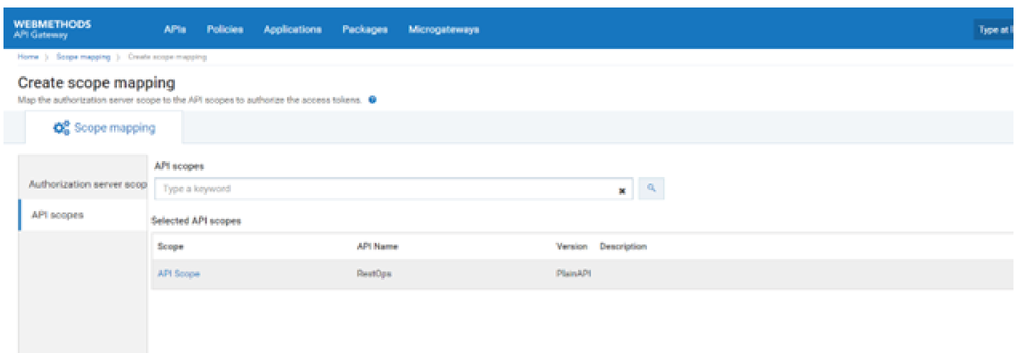
1. Expand the menu options in the title bar and Select **OAuth/OpenID scopes**.
2. Click **Map scope**.
3. Type**IS_OAuthServer** in Select authorization server scope and select the listed Authorization server scope from the search list populated.



4. Click **API** scopes.

5. Type `RestOps API Scope`, which is to be linked to the authorization server, in API scopes search text box.

6. Save the changes. This maps the authorization server scope to the selected API scopes and lists the authorization scope in the scopes list.



Step 5: Get authorization code

1. To get the Authorization code copy the client ID from the Integration server as seen in above step 1.

2. Access the Authorization code using below URL.
`https://<machinename_remoteintegrationserver>:5555/invoke/pub.oauth/authorize?client_id=f2c86e721b0344b0a076834449ae70a9&redirect_uri=http://example.com/redirect&response_type=code&state=121`

3. Click **Approve**

4. Copy the authorization code from the URL.
`http://example.com/redirect?code=4a69ffc12634a3e84123 7c0428bf0f9&grant_type=authorization_code&redirect_uri=http%253A%252F%252Fexample.com%252Fredirect&state=121&scope=View`Copy the authorization code from the URL.

You might get an error, but the authorization code is present in the URL as you are providing dummy redirect URI.

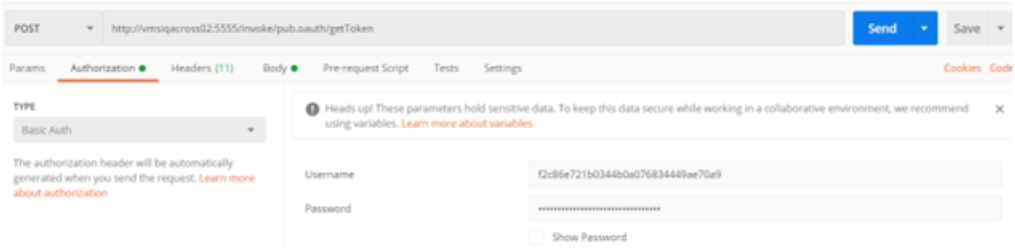
Step 6: Getting the access token

To get the access token, follow the below steps.

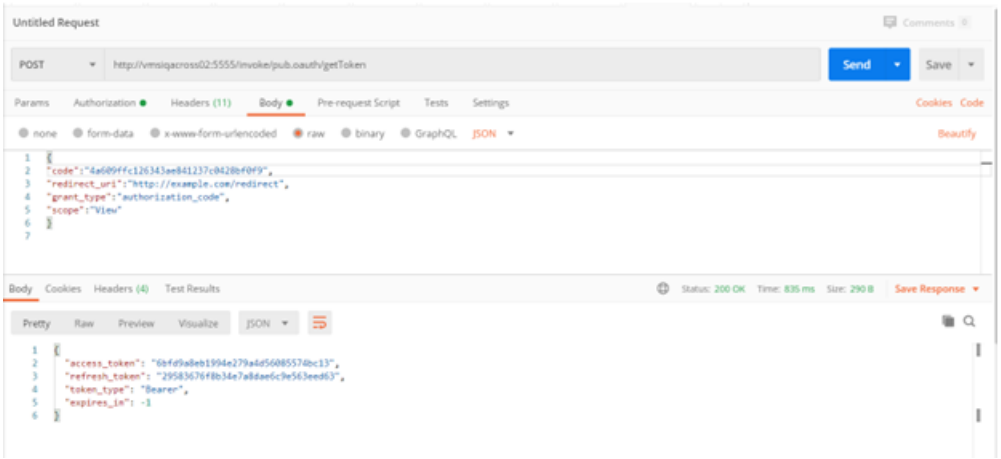
1. Open **Postman** or any other **Rest Client**
2. Copy the code from the above step
3. Make a POST call to the following URL, with the hostname of the system, where API Gateway is installed in place of localhost and use external port instead of default port like below.
   `https://<machinename_integrationserver>:5555 /invoke/pub.oauth/getToken`
4. In the Body provide the following Request payload:

```
{
"code":"4a609ffc126343ae841237c0428bf0f9",
"redirect_uri":"http://example.com/redirect",
"grant_type":"authorization_code",
"scope":"View"
}
```

5. In the Authorization select **Basic Auth** and provide the client_id in the Username and client_secret in the password.



6. The output is the access_token, refresh_token and token_type

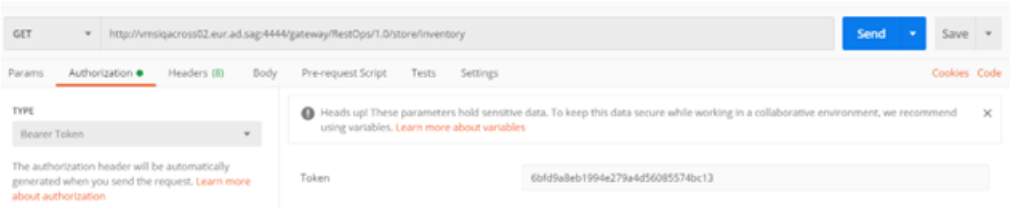


Step 7: Invoke API

The access_token can be used to invoke the API

1. Navigate to APIs in the title menu
2. Click **RestOps**

3. Under Technical information copy the Gateway endpoints
4. Have a look into the Resources and methods. You can use invoke any of these.
5. In PostMan invoke the API using the Gateway endpoints with the required resource or method.
6. In Authorization provide select **Bearer token** and provide the access_token



Alternative/Exception Flows

The same use case can be done with Client Credentials as well.

You can get the access token using client credentials. You require the client_id and client_secret.

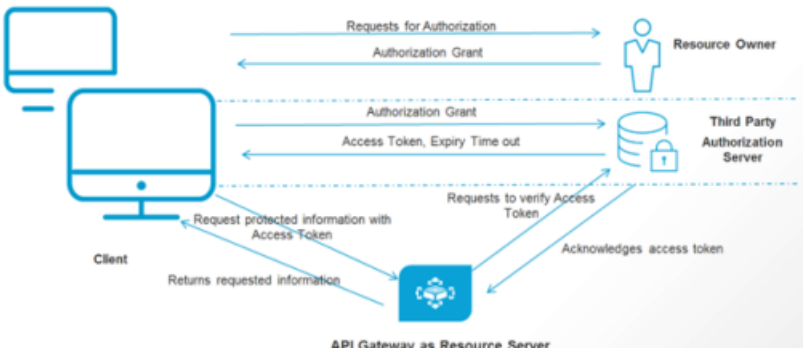
1. Open Postman or any other Rest Client
2. Make a POST call to the following URL, with the hostname of the system where API Gateway is installed in place of localhost and use external port instead of default port as follows:
   `https://<machinename_remoteintegrationserver>:5555 /invoke/pub.oauth/getToken`
3. In the Headers provide **Content-Type** as**application/json** (if not specified).
4. In the Body provide the following Request payload:

```
{
   “client_id”:”xxxxx”,
   “client_secret”:”xxxxx”,
   “ grant_type”:”client_credentials”
}
```

5. The output gives you the access token.
6. Use the access token to invoke the API.

Securing APIs using OAuth 2.0 in API Gateway using Third Party Authorization Server

This use case explains with a simple example on how to secure an API using OAuth2 authorization with a third-party OAuth2 identity provider and authorization server (OKTA). Here, the third-party OAuth2 provider acts as the Authorization server and API Gateway acts as a Resource Server



Actors

- Developers with basic knowledge about API Gateway, Integration Server, OAuth2 architecture
- Customers with basic knowledge about API Gateway, Integration Server, OAuth2 architecture

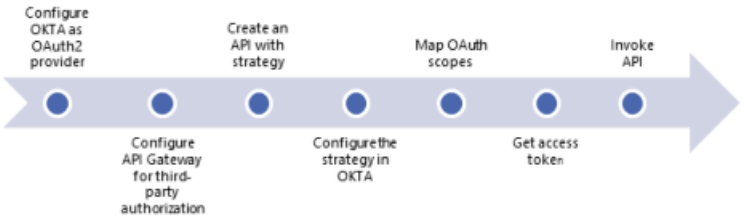
Before you begin

Ensure that you have:

- Created a tenant account in OKTA identity provider management portal
- Installed Integration Server with API Gateway
- Knowledge about any REST Client
- API Gateway up and running

Basic Flow

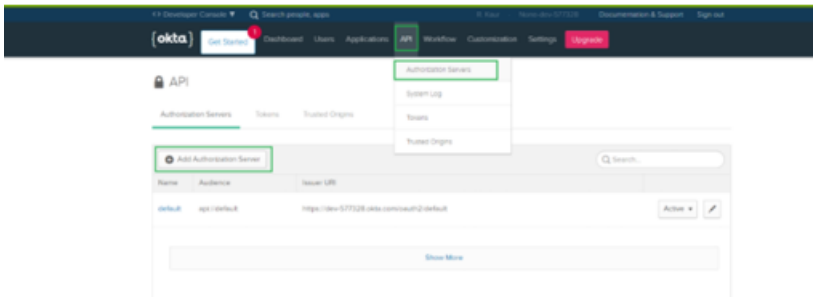
The following diagram depicts the high level steps of the basic flow for this use case.



Step 1: Configure OKTA as OAuth2 provider

1. Create Authorization server in OKTA as follows:
    a. Ensure you have a tenant account created in OKTA
    b. Navigate to **API** > **Authorization Servers**

c. Click **Add Authorization Server**.



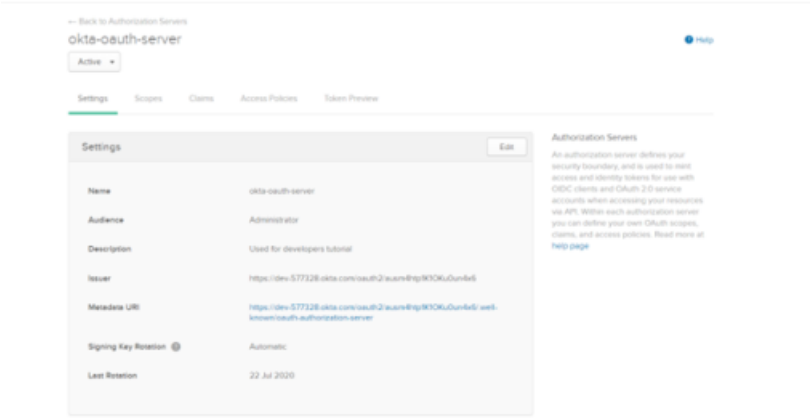
d. Provide the following details:

- **Name**: `okta-oauth-server`
- **Audience**: `Administrator`
- **Description**: `Used for developers tutorial`

e. Click `Save`.

2. Get the Metadata URI.

After you configure OKTA as OAuth2 provider the metadata URL informaton is available as shown.



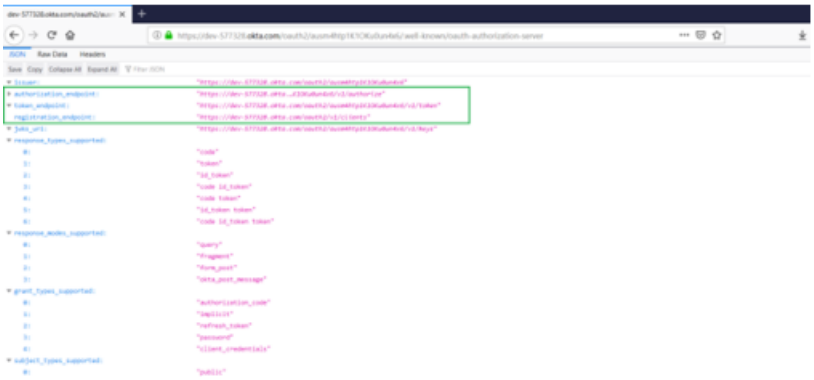
Copy the Metadata URI (also known as discovery URL) used to configure External Authorization server in API Gateway.

Note:

The same information is available if you click the Edit button for okta-oauth-server.

3. Get the endpoints.

   Invoke the Metadata URI from the browser and retrieve the endpoints as shown.

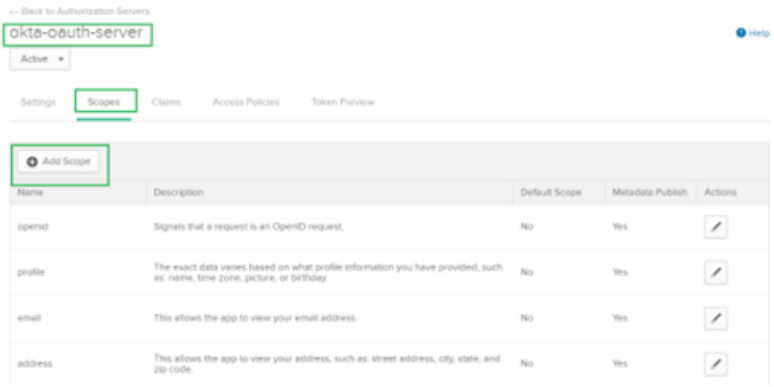


4. Create Scopes.

   a. Navigate to **API** > **Authorization Servers**

   b. Click **Edit** for okta-oauth-server.

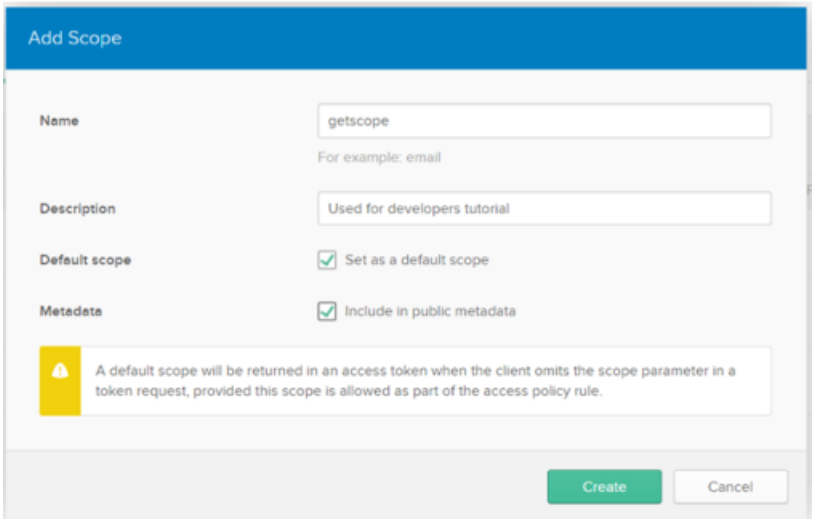
   c. Click **Add Scope**.



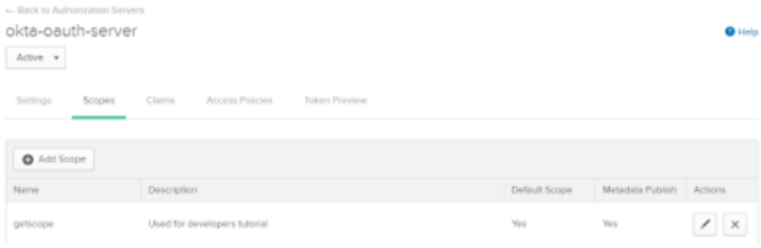
   d. Provide the following information

   - **Name**: `getscope`
   - **Description**: `Used for developers tutorial`
   - **Default Scope**: Select `Set as a default scope`
   - **Metadata**: Select **Include in public metadata**

   e. Click **Create**.



The scope is now listed under Scopes.

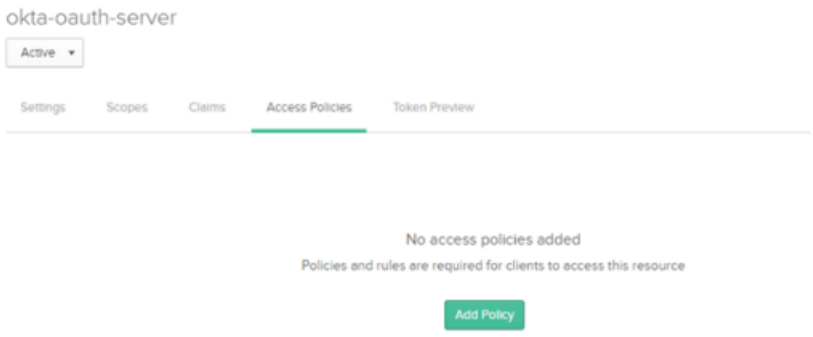


5. Create Access policies

   Create access policies and rule in OKTA to avoid accessibility issues while using access token during API invocation.

   a. Navigate to **Access Policies**.

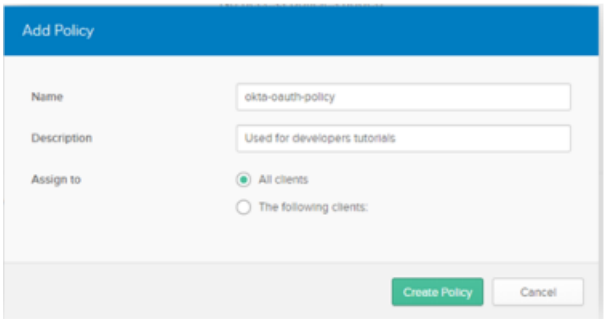
   b. Click **Add Policy**.



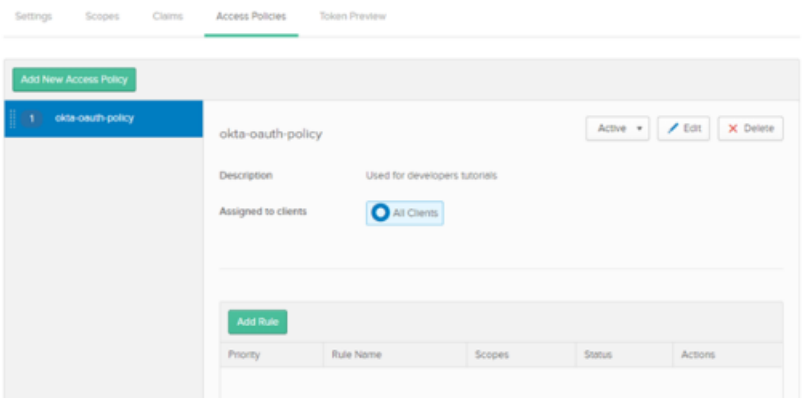
   c. Provide the following information:

      - **Name**: `okta-oauth-policy`

- **Description**: `Used for developers tutorials`
- **Assgn to**: Select **All clients**

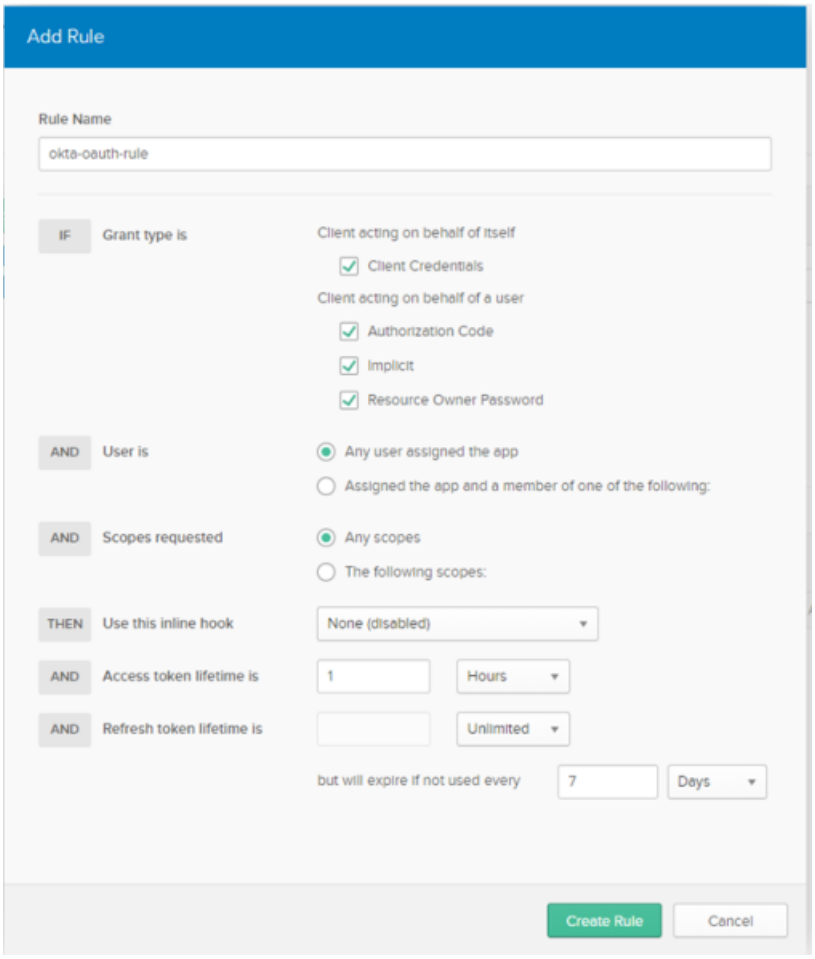


d. Click **Add Rule**.

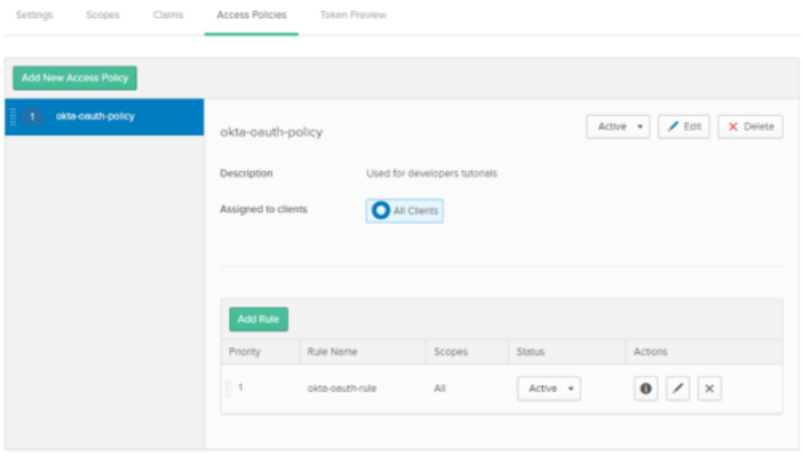


e. Provide the following information:

- **Rule Name**: `okta-oauth-policy`
- Do not change any default settings.

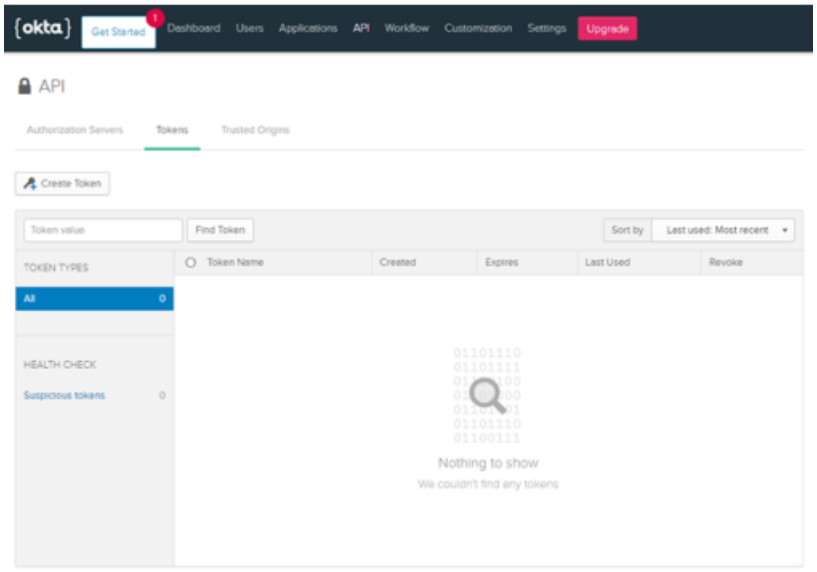


f. Click **Create Rule**.



6. Create **Token**.

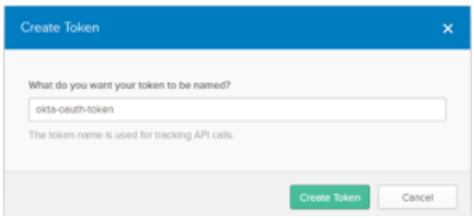
   a. Navigate to **API** > **Tokens**.



b. Click **Create Token**.

c. Provide the following information.

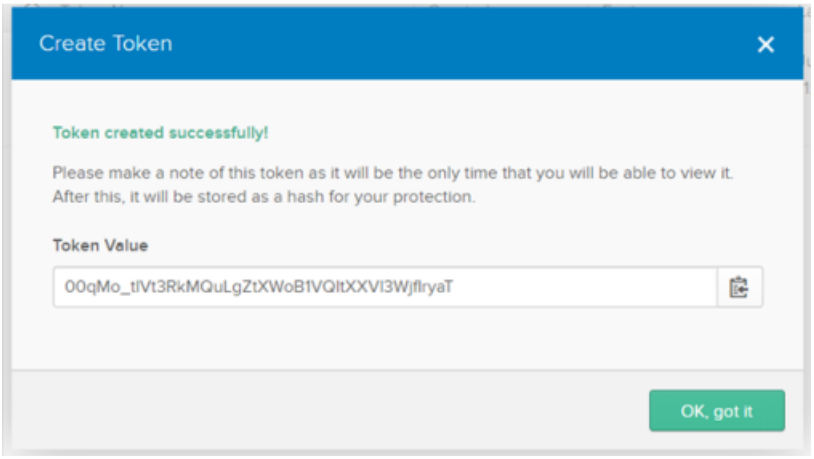
- **What do you want your token to be named?** : `okta-oauth-token`



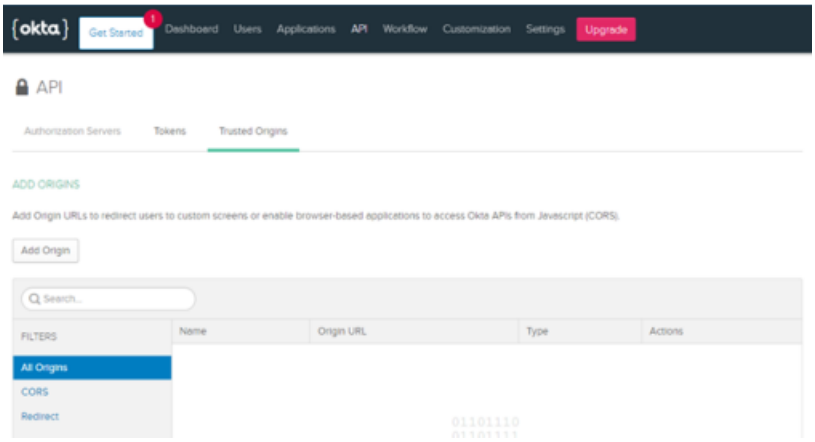
d. Copy the **Token Value**.

> **Note:**
> This token value is displayed only once.

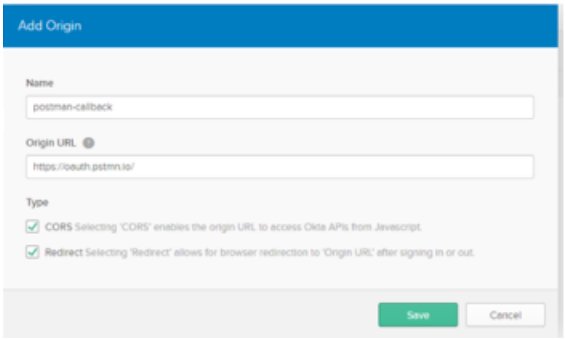
e. Click **OK, got it**.

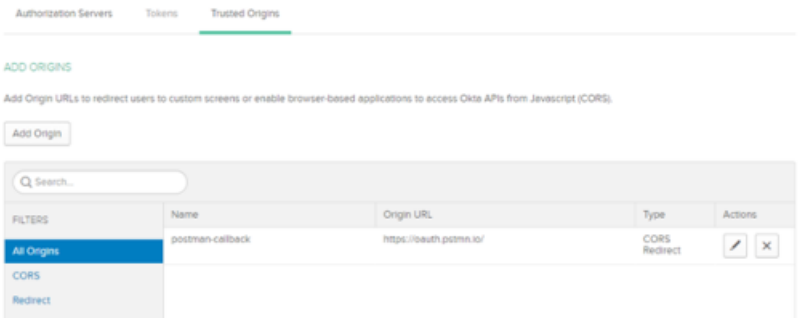


7. Add the postman callback url to Trusted Origins.
   a. Navigate to **API** > **Trusted Origins**



   b. Click **Add Origin**.
   c. Provide the following information.
      - **Name**: postman-callback
      - **Origin URL**: `https://oauth.pstmn.io`
      - **Type** : Select **CORS** and **Redirect**
      - Click **Save**.



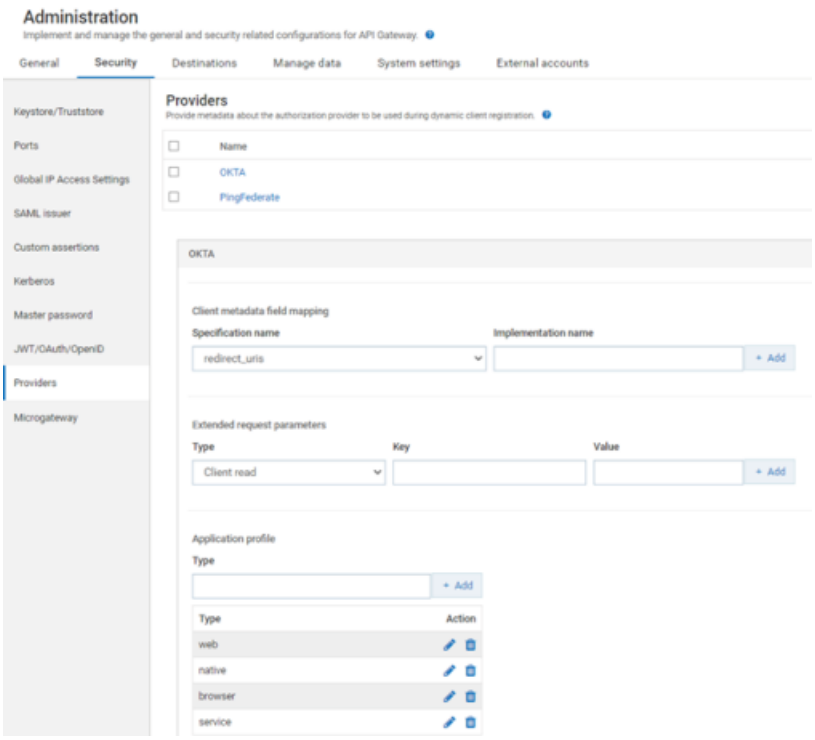


Step 2: Configure API Gateway for third-party authorization

This example uses the existing OKTA provider configured in API Gateway. API Gateway has providers OKTA and PingFederate predefined and configured as the third-party OAuth2 configuration servers.

Note:
You can also create your own identity providers by navigating to **Administration** > **Security** > **JWT/OAuth/OpenID** > **Providers**. You can then click **Add provider** and provide the required details. This provider configuration is used in the Dynamic client registration.

1. Click **Menu** > **Administrator**.
2. Click **Security**.
3. Click **OKTA**.



4. Click **JWT/OAuth/OpenID**.
5. Click **Add authorization server**.
6. Provide the following information:
   - **Name**: `okta-oauth-server`
   - **Discovery URL**: `https://dev-577328.okta.com/oauth2/ausm4htp1K1OKu0un4x6/.well-known/oauth-authorization-server`

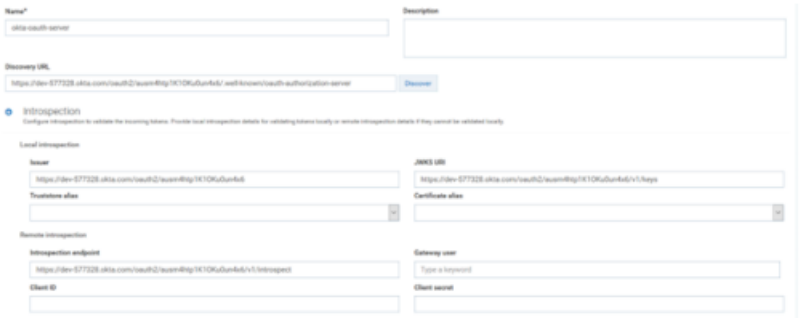
7. Click **Introspection**.

   You use Introspection to authenticate the token. It happens as follows

   - **Local introspection**. Validating the token within the gateway. This is done using the JWKS URI or the public certificate of the issuer. In this case the token should necessarily be JWT.

- **Remote introspection**. Validating the token with the authorization server. It has introspection endpoint, which is used to validate the token. In addition, the client id and client secret are used to protect the endpoint, so that anonymous users cannot access to the resource. To invoke an endpoint you require a user; Gateway user is the one you can use to invoke the endpoint. There is no support for token caching in remote introspection.

8. Remove the **Introspection endpoint** from **Remote Introspection**.

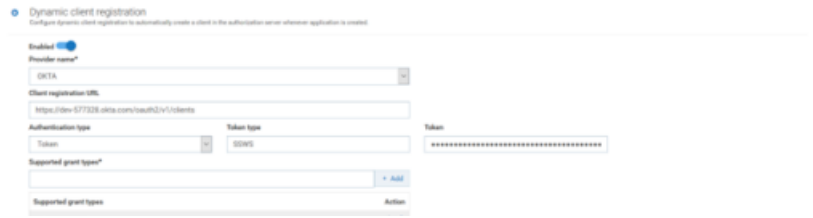


9. In **Dynamic client registration**, click **Enable** toggle button.

   Use Dynamic client registration when you want to create the client automatically in the OAuth2 authorization server when an application is created in API Gateway.

10. Provide the following information:

    - **Authentication type**: `Token`
    - **Token type**: `SSWS`
    - **Token**: Copy the saved token value.

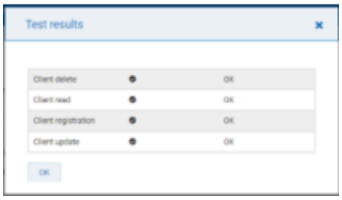


11. Click **Metadata**. Verify that the Metadata is auto populated with the right values.

12. Click **Scopes**. Verify getscope is present in the list. Delete the other scopes as as they are not required.

13. Click **Add**.

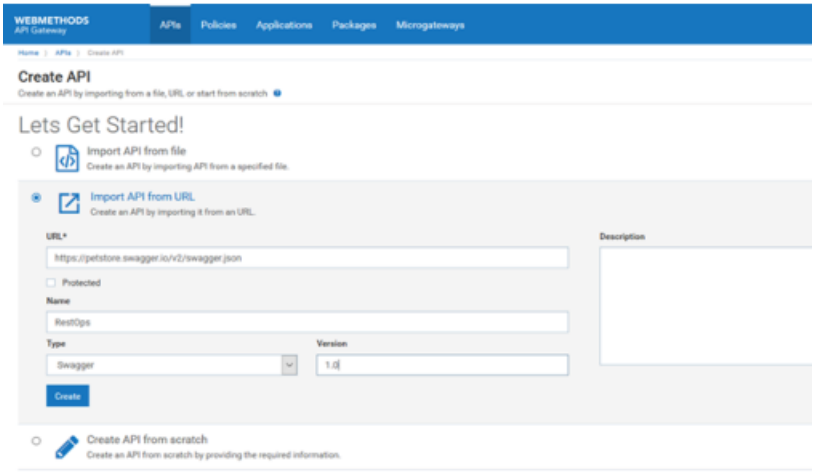
14. Click the added external authorization server to test it.



Step 3: Create an API with strategy

In this example we will create an API by importing from the URL `https://petstore.swagger.io/v2/swagger.json`, enforce it with OAuth2 policy, and create an applicatin with strategy and associate it with the API.

1. In the **APIs** tab click **Create API**.
2. Select **Import API from URL**.
3. Provide the following information.
   - **URL**: `https://petstore.swagger.io/v2/swagger.json`
   - **Name**: `RestOps`
   - **Type**: `Swagger`
   - **Version**: `1.0`
4. Click **Create**. The API is created and the API details page for the API appears.



5. Enforce OAUTH2 policy on the API.

   You enforce OAuth2 policy on the RestOps API. This policy ascertains that an OAuth token is required to access this API.

   a. Click **APIs** in the title navigation panel.

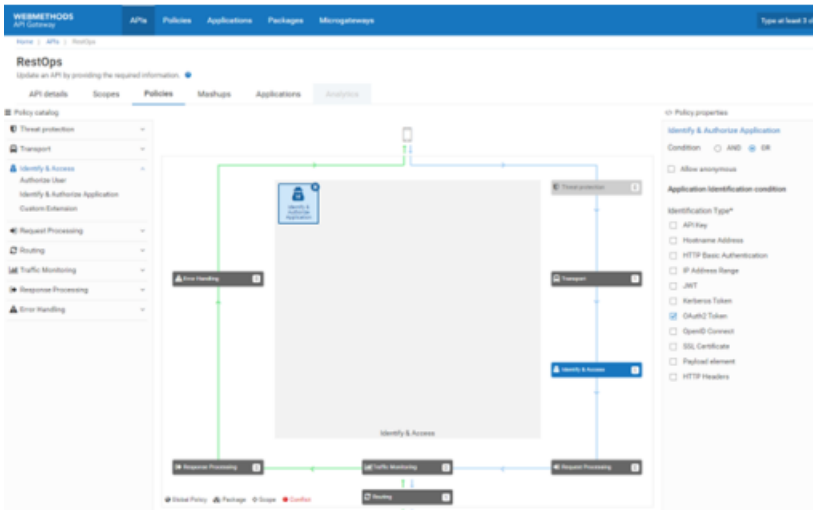
   b. Click **RestOps**.

   c. Click the **Policies** tab.

   d. Click **Edit**.

   e. Click **Identify & Access** from the policy catalog section.

   f. Select the **Application Identification Type** as **OAuth2 Token** and save the API.



   g. Click **Activate** on the **API details** page of the API to activate the API.

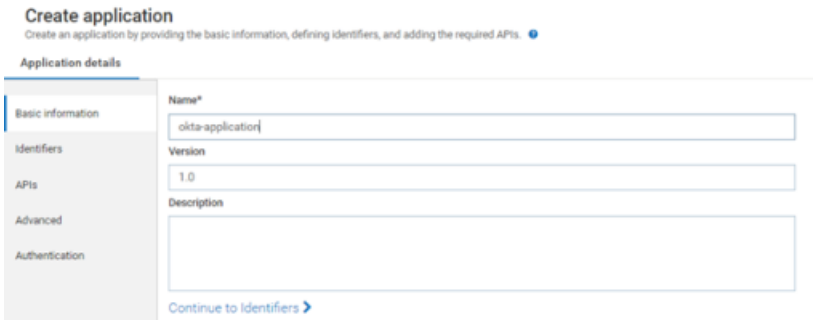
6. Create an application in API Gateway with a strategy and register it to an API.

   a. Click **Applications** in the title navigation bar.

b. Click **Create application**.

c. Provide the **Name** as `okta-application`.

d. Click **Continue to Identifiers**.

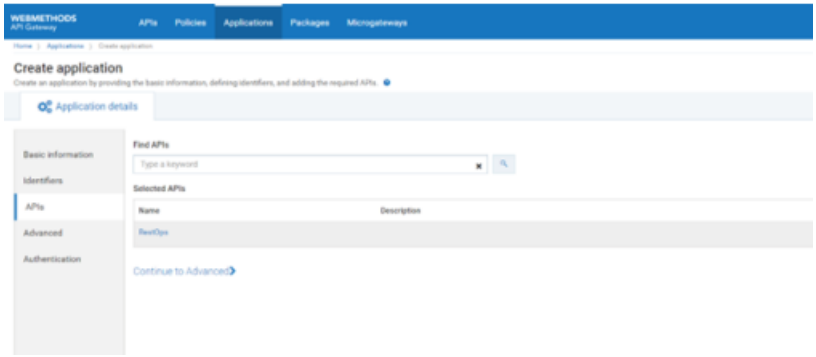


e. Click **Continue to APIs**.

f. Type `RestOps` in the **Find APIs** text box. The RestOps API appears in the **Selected APIs** section.

g. Select the RestOps API.

h. Click **Continue to Advanced**.



i. Click **Continue to Authentication**.

j. Click **Create Strategy**.

A strategy authenticates the incoming request and provides multiple authentication mechanisms or multiple authorization servers for a single authentication scheme. You can create multiple strategies authorized by an API for an application.

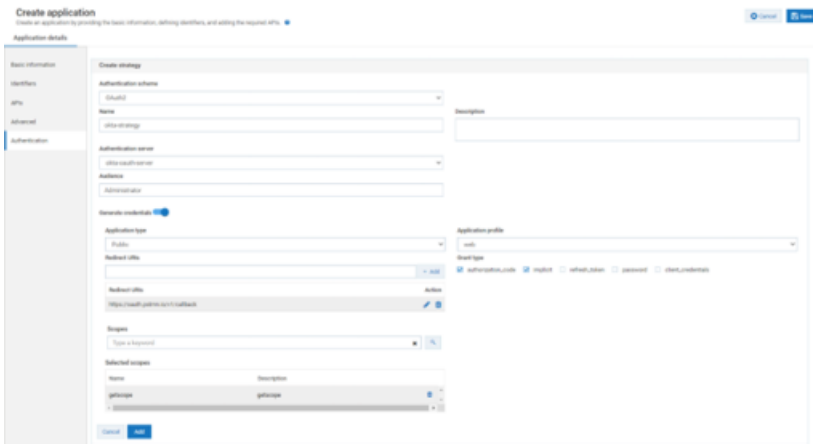
k. Provide the following information:

- **Name**: `okta-strategy`
- **Authentication Server** : Select `okta-oauth-server`
- **Audience** : `Administrator`

- Enable **Generate Credentials**.
- **Redirect URIs**: `https://oauth.pstmn.io/vl/callback`
- Click **Add** to add the Redirect URI.
- **Grant type**: Select `authorization_code` and `implicit`.
- **Scopes**: Search for `getscope`, click the + sign to add the scope.

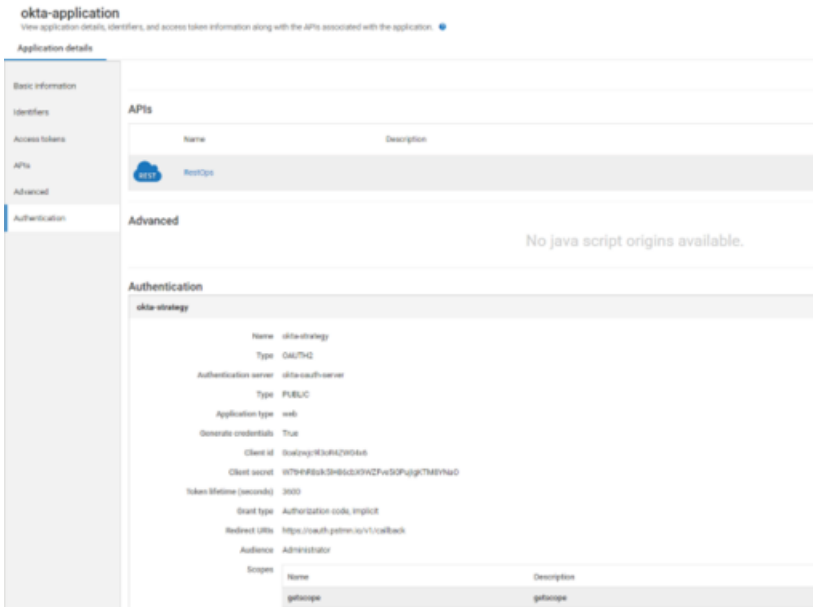
l. Click **Add** to add the strategy.

m. Click **Save**.



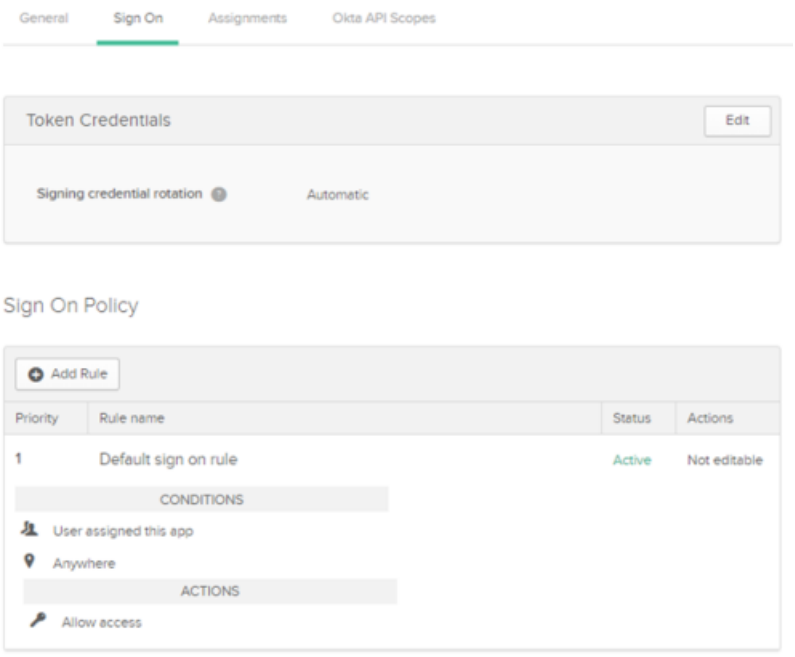
The strategy is created in OKTA as well.

n. Make a note of client_id and client_secret displayed in **Applications** > **okta-application** > **Authentication** > **okta-strategy** section.

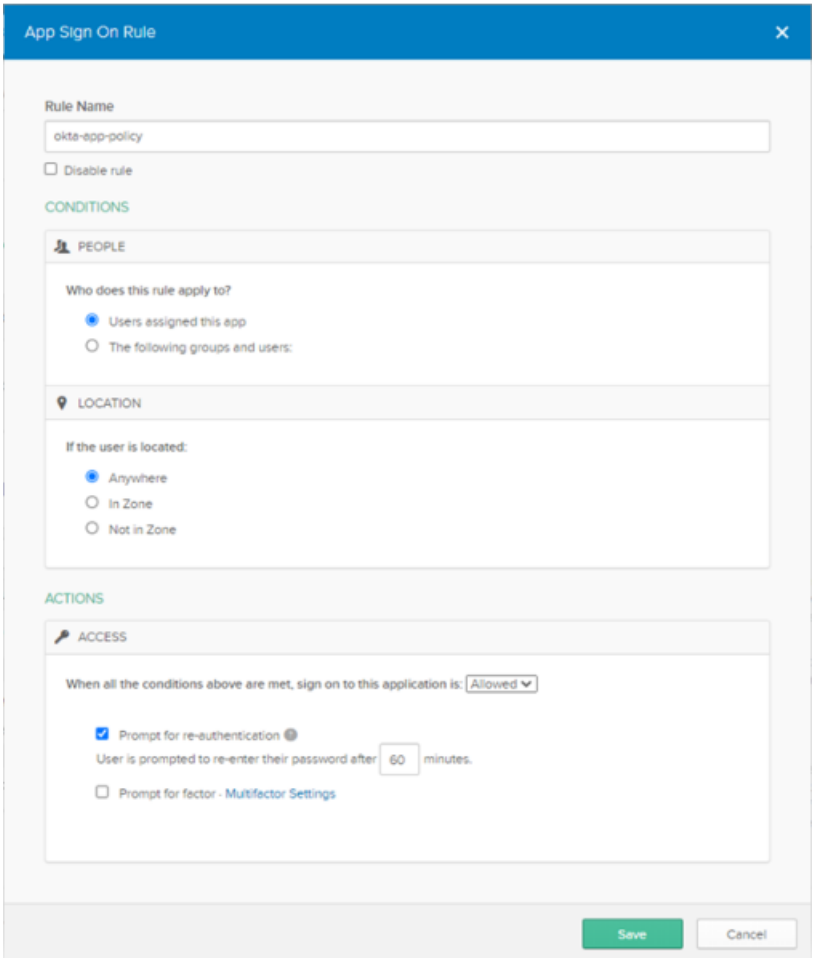


Step 4: Configure the Strategy in OKTA

1. Login to OKTA.
2. Navigate to **Application**. The strategy okta-strategy should be present.
3. Click the strategy okta-strategy.
4. Click **General** and verify that the details are same as in API Gateway.
5. Click **SignOn**.
6. Perform the following steps in the **SignOn** page:
   a. Click **Edit** in Token Credentials.
   b. **Signing credential rotation**: Select `Automatic`.
   c. Click **Save**.
   d. Under **Sign On Policy** click **Add Rule**.

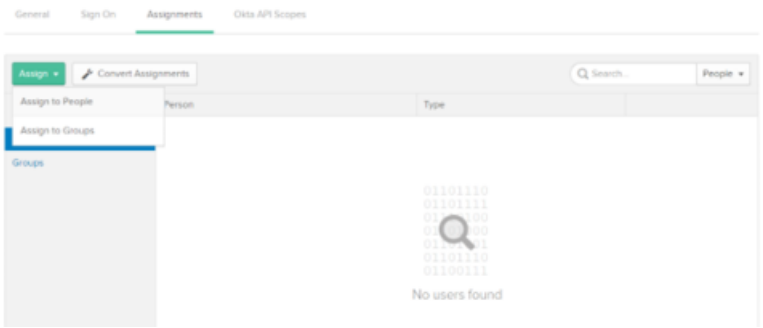


   e. Provide **Rule Name** as `okta-app-rule`.
   f. In **Access** select `Prompt for reauthentication`.
   g. Click **Save**.



h. Click **Assignments** to assign Users and groups to the applicaton to avoid ACL permission issues.

i. Click **Assign** and then **Assign** to `People`.

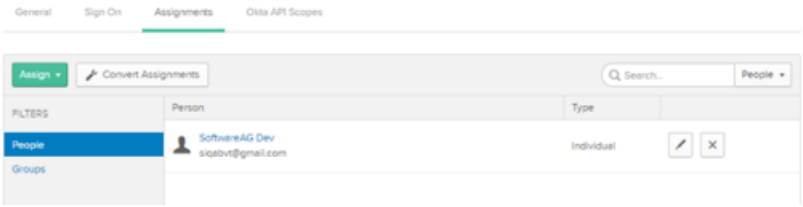


j. The login user should be listed else search the user in the search box.

k. Click **Assign**.

l. Click **Save and Go Back**.

m. Click **Done**.



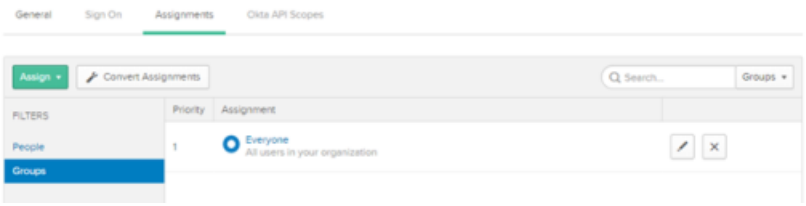
n. Click **Assign** and then **Assign** to `Groups`.

o. The group **Everyone** should be listed, else search the group in the search box.

p. Click **Assign**.

q. Click **Save and Go Back**.

r. Click **Done**.



Step 5: Map OAuth scopes

You must map the scope defined in OKTA Authorization server with the APIs in API Gateway to authorize the access tokens to be used to access the protected resources. One can map either a complete API or parts (resources or methods) of an API to the scope or can add the scope details and modify the scope details as required from the OAuth/OpenID scopes page.

1. Expand the menu options in the title bar and select **OAuth/OpenID scopes**.
2. Click **Map scope**.
3. Type `getscope` in **Select authorization server scope** and select the listed Authorization server scope from the search list populated.
4. Provide **Audience**as `Administrator`.

5. Click **API scopes**
6. Type `RestOps` or `API Scope`, which is to be linked to the authorization server, in API scopes search text box.
7. Save the changes. This maps the authorization server scope to the selected API scopes and lists the authorization scope in the scopes list.

Step 6: Get the access token

1. Open Postman, click Authorization and select OAuth2.
2. Click **Get New Access Token**.
3. Provide the following information:
   - **Token Name**: `oktaToken`
   - **Grant Type**: `Authorization Code`
   - **Callback URL**: `https://oauth.pstmn.io/v1/callback`
   - Select Authorize using browser
   - **Auth URL**: `http://dev-577328.okta.com/oauth2/ausm4htp1K1OKu0un4x6/v1/authorize`
   - **Access Token URL**: `https://dev-577328.okta.com/oauth2/ausm4htp1K1OKu0un4x6/v1/token`

   Note:
   Get the Auth URL and Access token URL from **Administration** > **Security** > **JWT/OAuth/OpenID** > **External authorization server** > **okta-oauth-server** > **Meta data** section.

- **Client ID**: *0oam51fdfAJcDyDkT4x6*
- **Client Secret**: *_MOgtr2gUHB8lkhZzM4uPL3wL53may1FgqHjHN2o*

  Note:
  The Client_ID and Client_Secret can be obtained from the **Application** > **okta-application** > **okta-strategy**.

- **Scope**: *getScope*
- **State**: 121
- **Client Authentication** : Send Client credentials in body

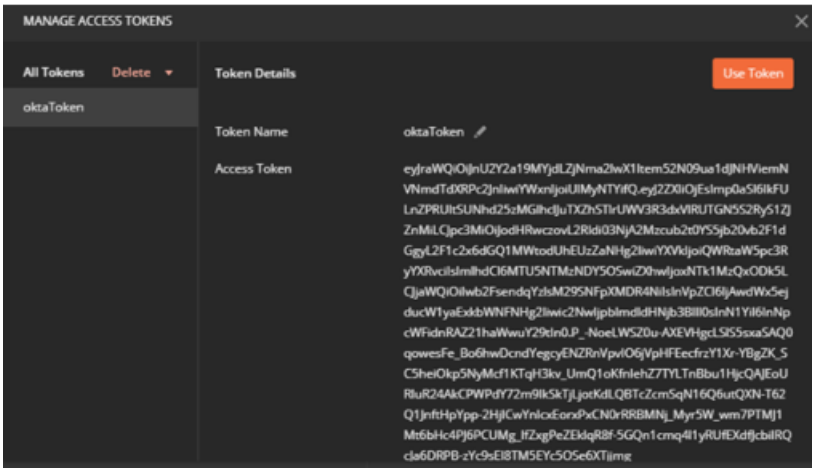
4. Click **Request Token**.

   You are redirected to OKTA login page.

5. Login into OKTA Page. A `Authentication completed` message displays and you are redirected to call back URL.

6. In Postman you will get the Access token from Postman.

   Click **Use Token** or copy the token which you can use to invoke the API.



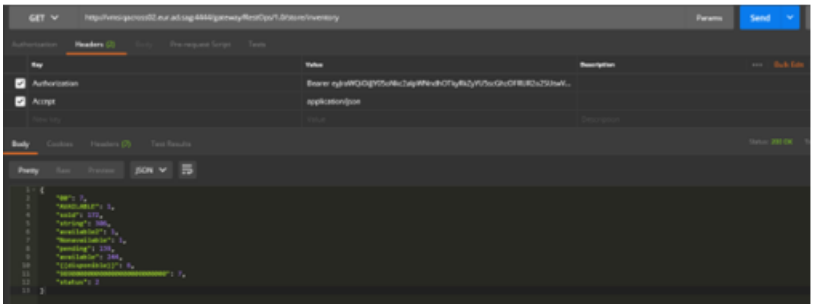
Step 7: Invoke API

Ensure you have the following to invoke the API

- Gateway endpoints. You will find the gateway endpoints in API Gateway UI in the APIs > **RestOps** > **Technical information** section. In this example the endpoint is `http://vmsiqacross02.eur.ad.sag:4444/gateway/RestOps/1.0/store/inventory`.
- The method or resource that must be invoked. The list operations are present in the **APIs** > **RestOps** > **Technical information** section under Resources and methods. Select anyone. In this example select `/store/inventory` which is a GET method.
- Access token.

1. Open Postman.
2. Select Get and `http://vmsiqacross02.eur.ad.sag:4444/gateway/RestOps/1.0/store/inventory`.
3. In **Headers** add `Authorization` as Key and provide value as `Bearer access token`
4. In **Headers** add `Accept` as Key and provide value as `application/json`.
5. Click **Send**.

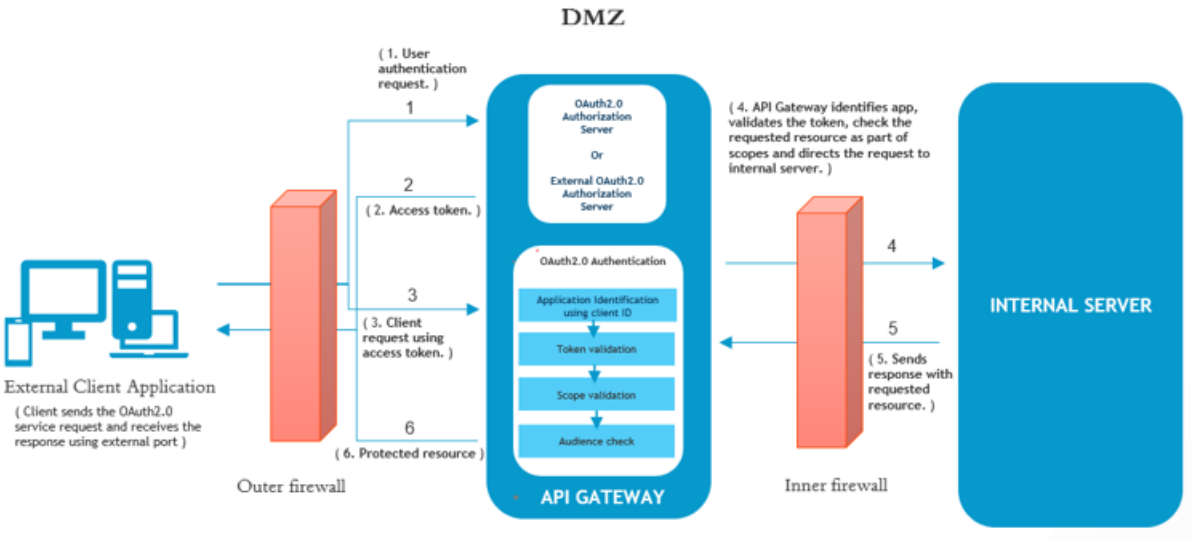
   The API is invoked and you see the expected response with a 200 OK status.



Securing APIs Using OAUTH 2.0 In API Gateway Using External Ports

This use case defines OAuth2.0 authentication protocol that identifies and authorizes a client application using external ports. In API Gateway OAuth2.0, you can configure services using both primary and external ports. When you do not want to expose the primary port to the outside world, you can configure an external port. This external port is exposed to the outside world for allowing users to consume the APIs.

This use case explains, with a simple example, how to secure an API with OAuth2 authentication using external ports. This example uses Client Credentials as the grant type and a single instance of API Gateway server. The diagram illustrates the work flow for this use case.



Actors

- Developers with basic knowledge on webMethodsAPI Gateway, Integration Server, OAuth2 architecture.
- Customers with basic knowledge on webMethods API Gateway, Integration Server, OAuth2 architecture.

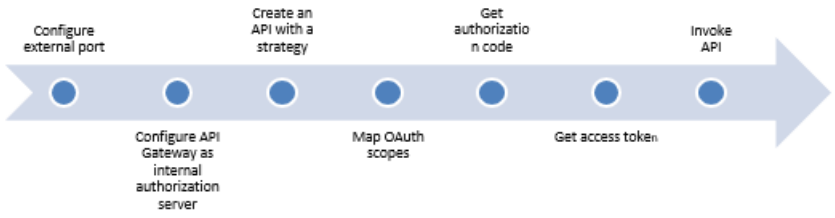
Before you begin

Ensure that you have:

- Installed Integration Server with API Gateway.
- Knowledge about any Rest Client.
- API Gateway up and running.

Basic Flow

The following diagram depicts the high level steps of the basic flow for this use case.



Step 1: Configure external port

1. Enable OAuth 2 authorization through HTTP in API Gateway.

   To enable API Gateway to accept request through HTTP, you must set the value of **pg_oauth2_isHTTPS** to `false` as follows.

   a. Expand the menu options in the title bar and select **Administration**.

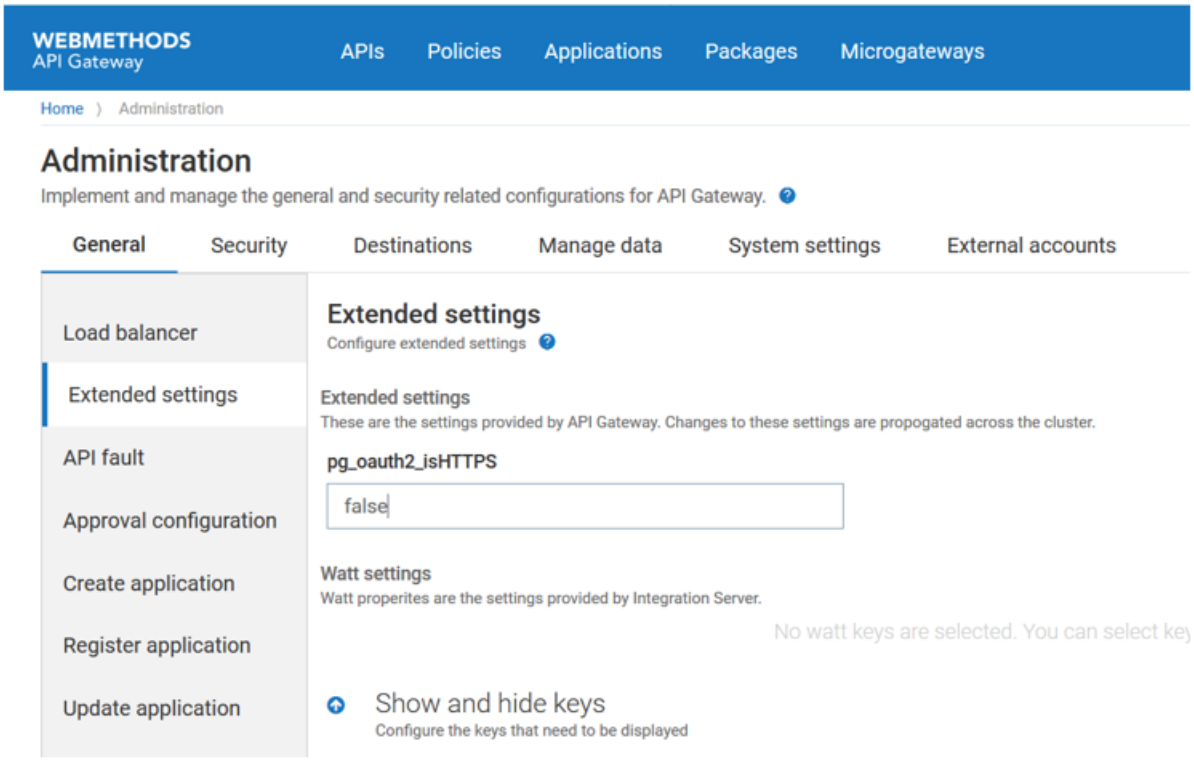
   b. Select **General** > **Extended settings**.

   c. Click **Show and hide** keys.

      This displays all the configurable parameters.

   d. Set the value of **pg_oauth2_isHTTPS** to `false`.

   e. Click **Save**.



2. Enable OAuth 2 service using the external port.

   For the OAuth service to be available through external ports, set the value of **watt.server.revInvoke.proxyMapUserCerts** property to `true` as follows. The default value of this property is `false`.

   a. Expand the menu options in the title bar and select **Administration**.

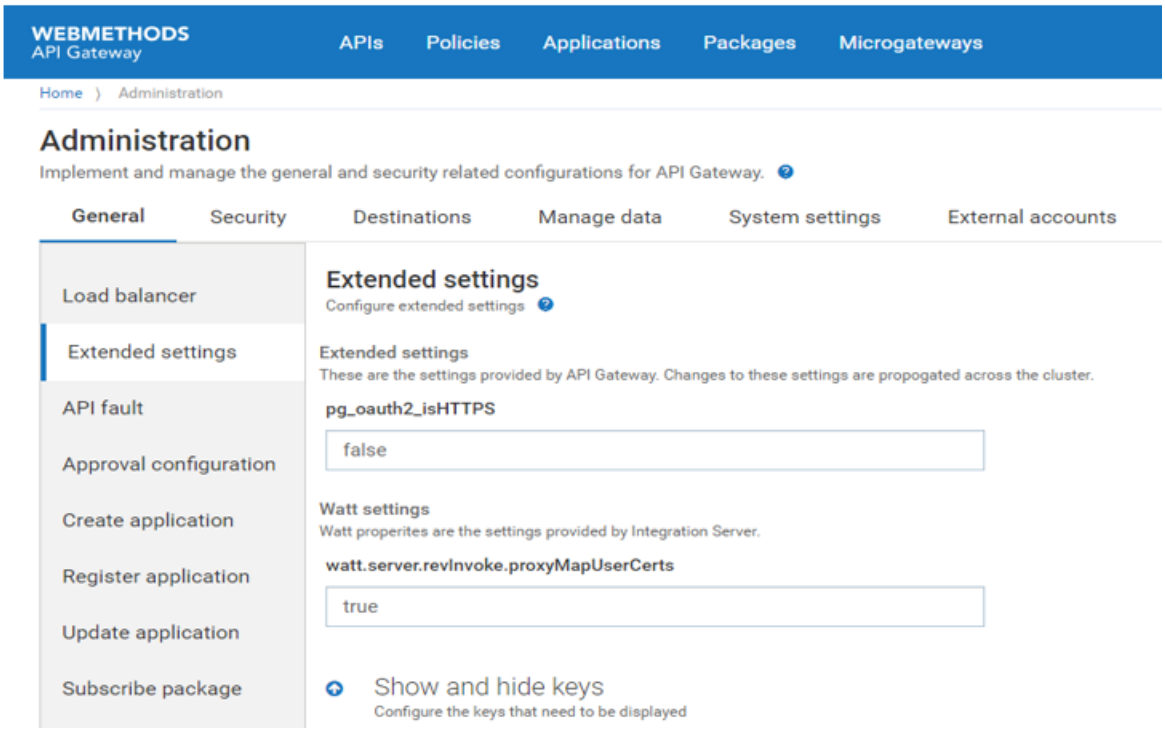
   b. Select **General** > **Extended Settings**.

   c. Click **Show and hide** keys.

      This displays all the configurable parameters.

   d. Set the value of **watt.server.revInvoke.proxyMapUserCerts** property to `true`.

   e. Click **Save**.

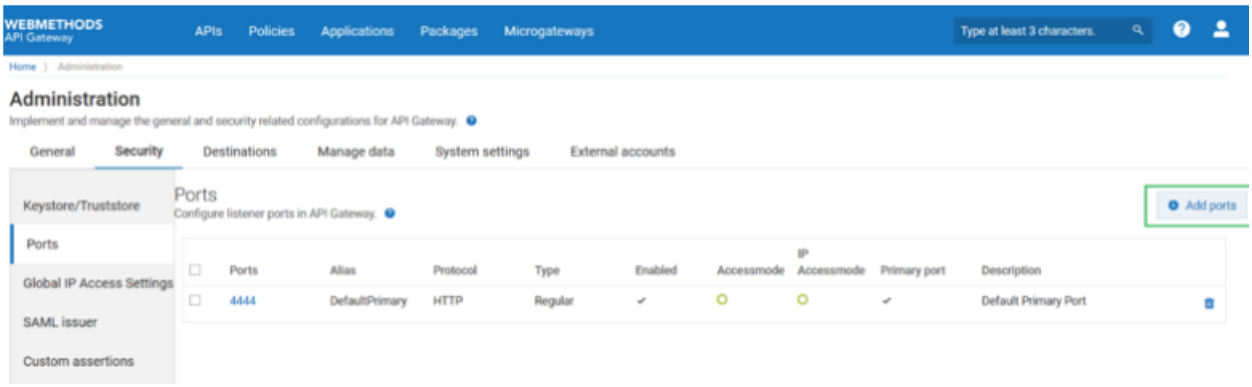


3. Create external and registration ports in API Gateway.

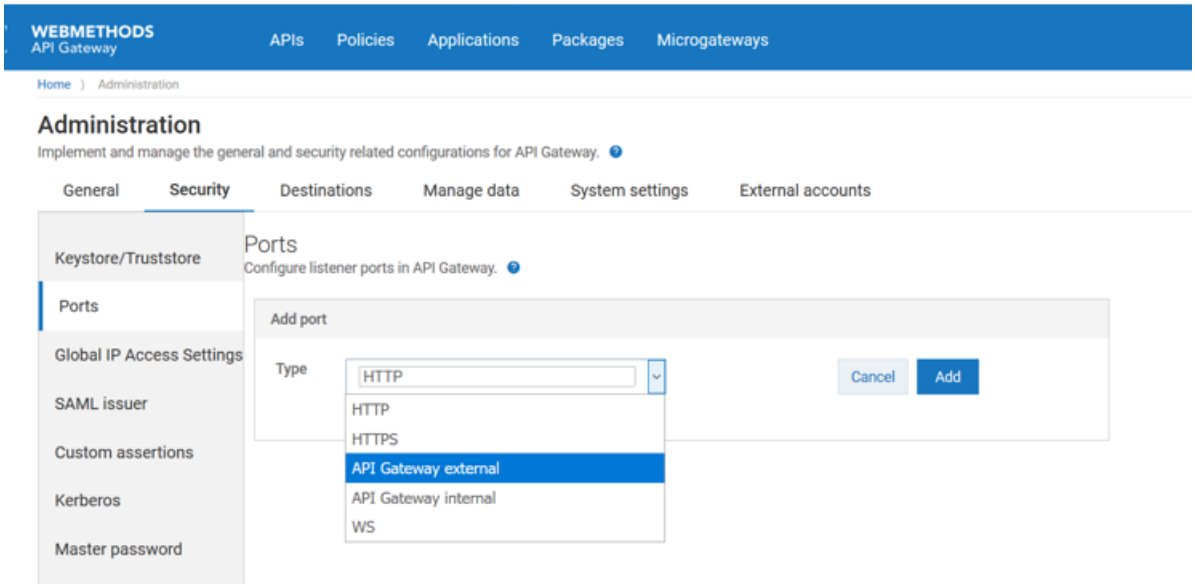
   The external client requests come to an external port, which delegates the request to its paired port, that is the registration port.

   a. Expand the menu options in the title bar and select **Administration**.

   b. Select **Security** > **Ports** and click on **Add Ports**.



   c. Select the type of port as **API Gateway external** and click **Add**.



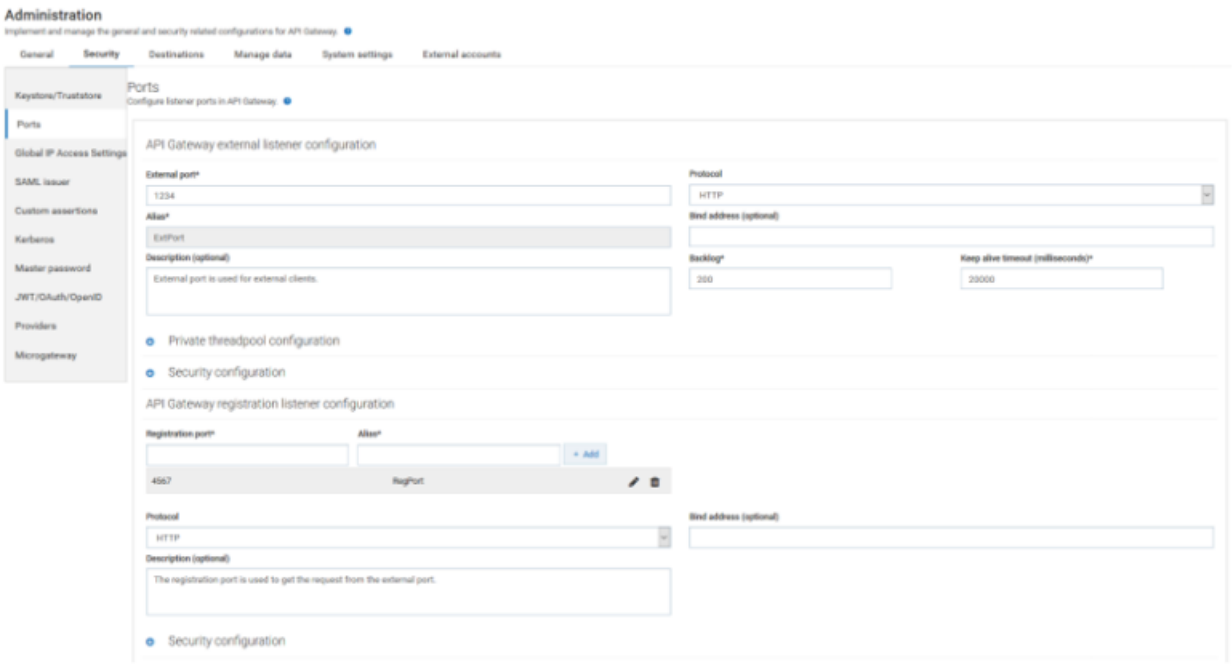
d. Provide the following information:

- **External port**: `1234`
- **Alias**: `ExtPort`

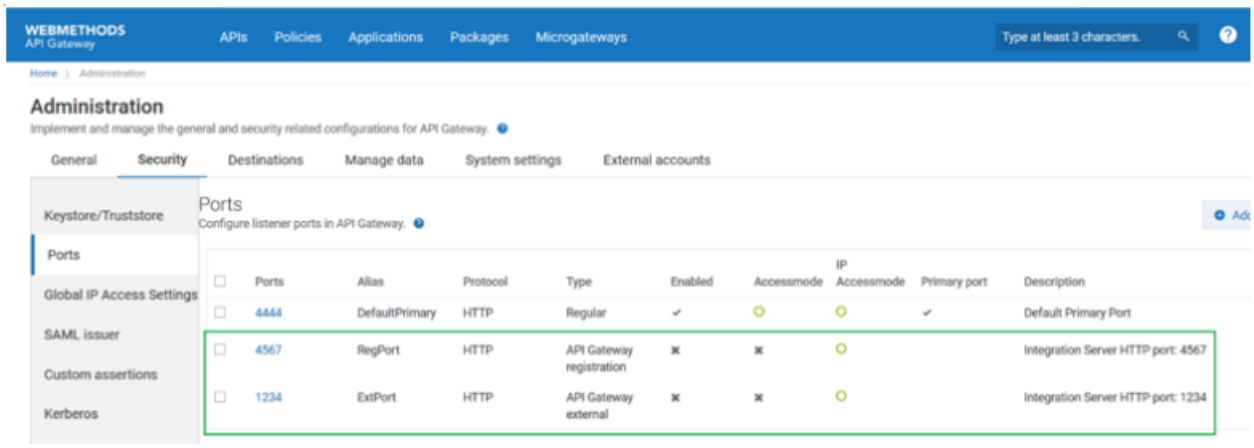
e. Under API Gateway registration listener configuration, provide the following information:

- **Registration port**: `4567`
- **Alias**: `RegPort`

f. Click **Add**.



g. Click **Add** button at the end of the page to add the external port.



4. Create an internal port on API Gateway.

   In this use case, a single API Gateway instance is used. In the case of creating an internal port in the internal server, the same API Gateway server acts as an internal server, which listens and pulls the requests queuing up in the registration port.

   a. Expand the menu options in the title bar and select **Administration**.

   b. Select **Security** > **Ports**.

   c. Click on **Add Ports** and select **API Gateway internal** and click **Add**.

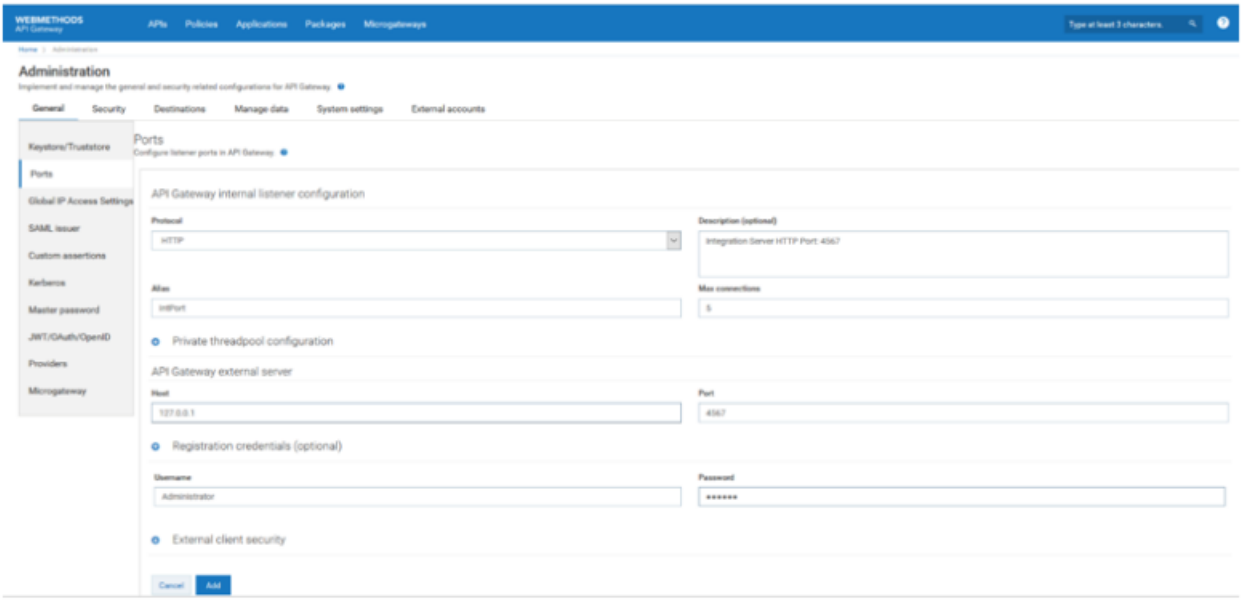
   d. Provide the following value for Alias, `IntPort`.

e. Under API Gateway external server, provide the following information:

- **Host**: `127.0.0.1`
- **Port**: `4567`

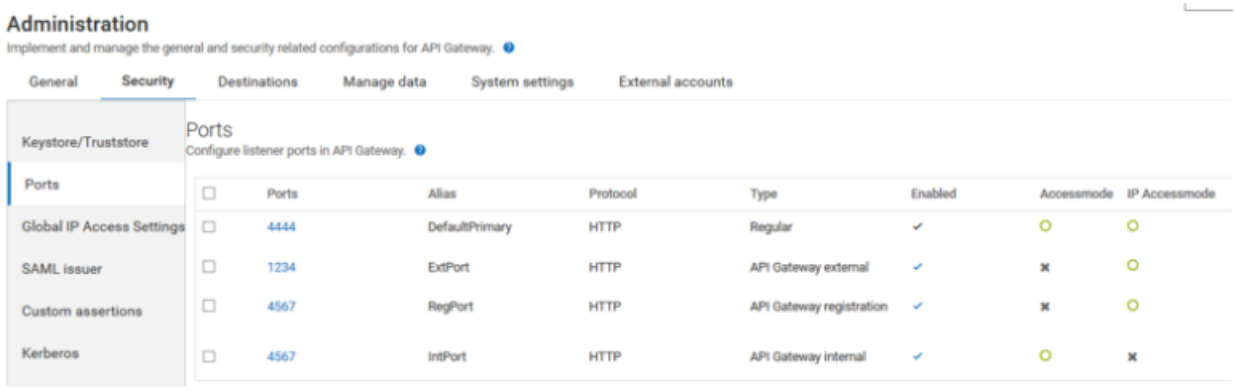
f. Under Registration credential(optional), type the username as `Administrator` and password as `manage`.

g. Click **Add**.



5. Enable the ports.

   Enable the ports after all the ports are created. Click on the **x** under **Enabled** column for the ports.



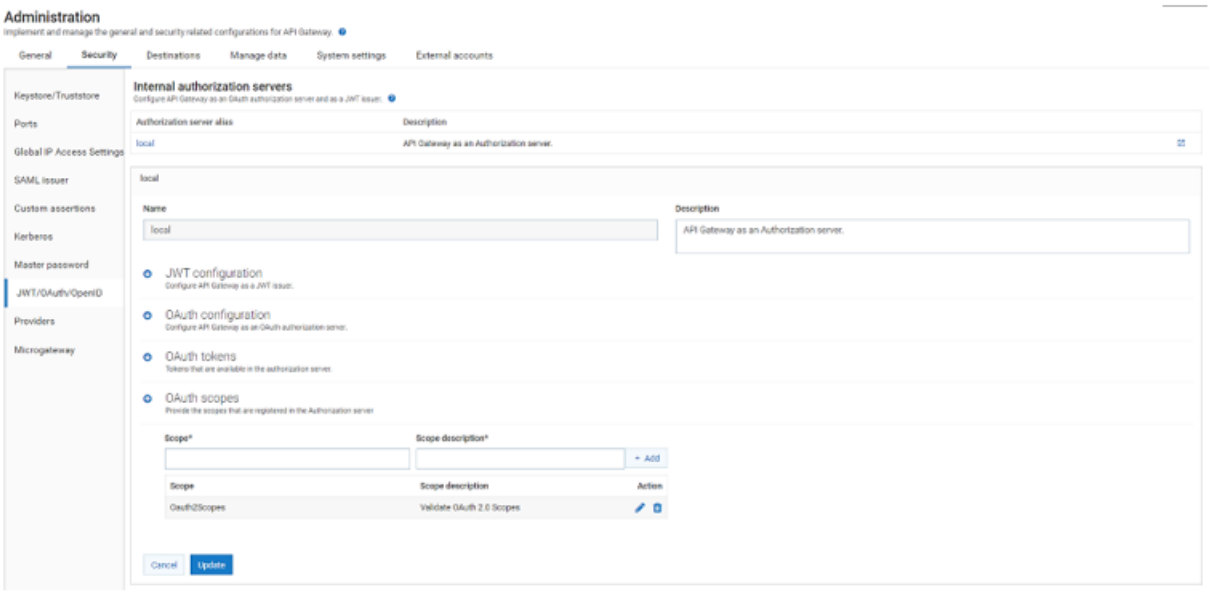
Step 2: Configure API Gateway as an internal authorization server

Here, you configure API Gateway with the required information to act as an internal authorization server for OAuth or JWT depending on what authentication protocol you want to use to identify

and authorize a client application. You can also define the required scopes that provide a way to limit the amount of access that is granted to an access token. In this xample, use the default configuration as it is and define one OAuth Scope (OAuth2Scopes) to limit the resource usage.

1. Expand the menu options in the title bar and select **Administration**.
2. Select **Security** -> **JWT/OAuth/OpenID** and click **local**.

   This displays all the configurable settings for the Authorization Server.
3. Click **OAuth scopes** to add Scope.
4. Provide the following information and click Add:
   - **Scope**: `OAuth2Scopes`
   - **Scope description**: `Validate OAuth 2.0 Scopes`
5. Click **Update**.



Step 3: Create an API with strategy

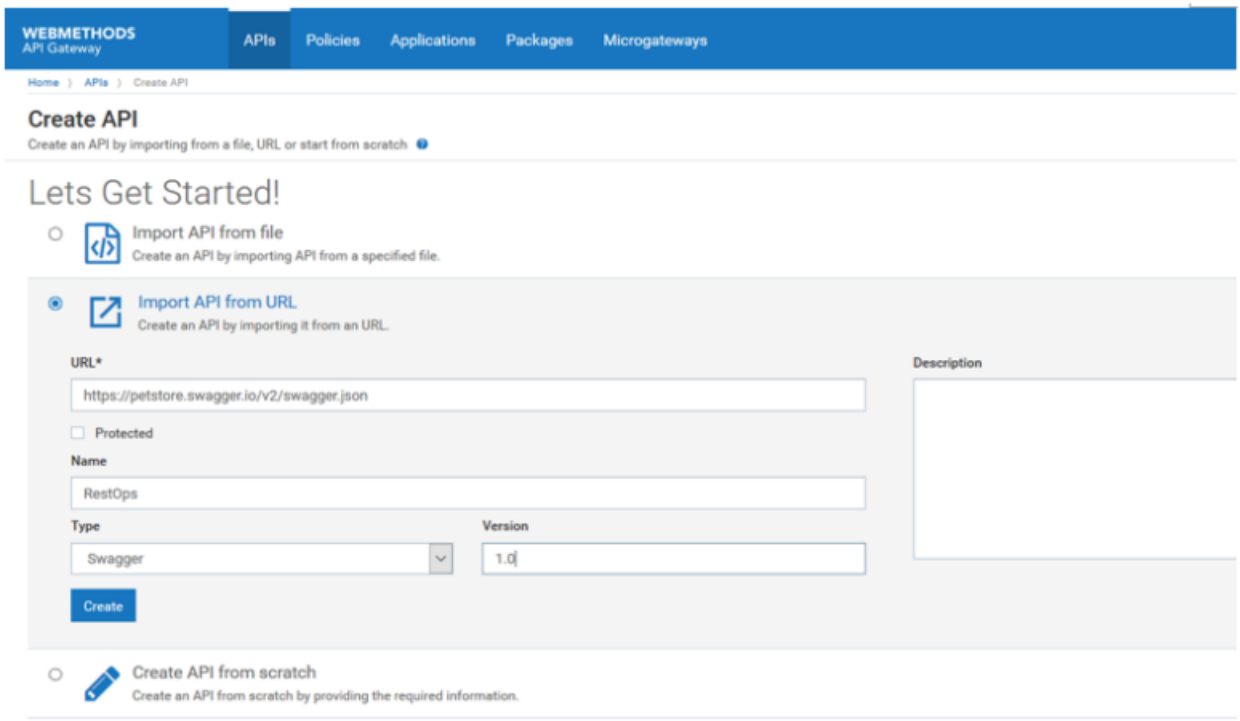
In this example, you create an API by importing from the URL `https://petstore.swagger.io/v2/swagger.json`.

1. In the **APIs** tab, click **Create API**.
2. Select **Import API from URL**.
3. Provide the following details.
   - **Name**: `RestOps`
   - **Type**: `Swagger`

- **Version**: `1.0`

4. Click **Create**.

   The API is created and the API details page for the API appears.



5. Enforce OAuth 2 policy on the API.

   You enforce OAuth2 policy on the RestOps API. This policy ascertains that an OAuth token is required to access this API.

   a. Click **API** in the title navigation panel.

   b. Click **RestOps**.

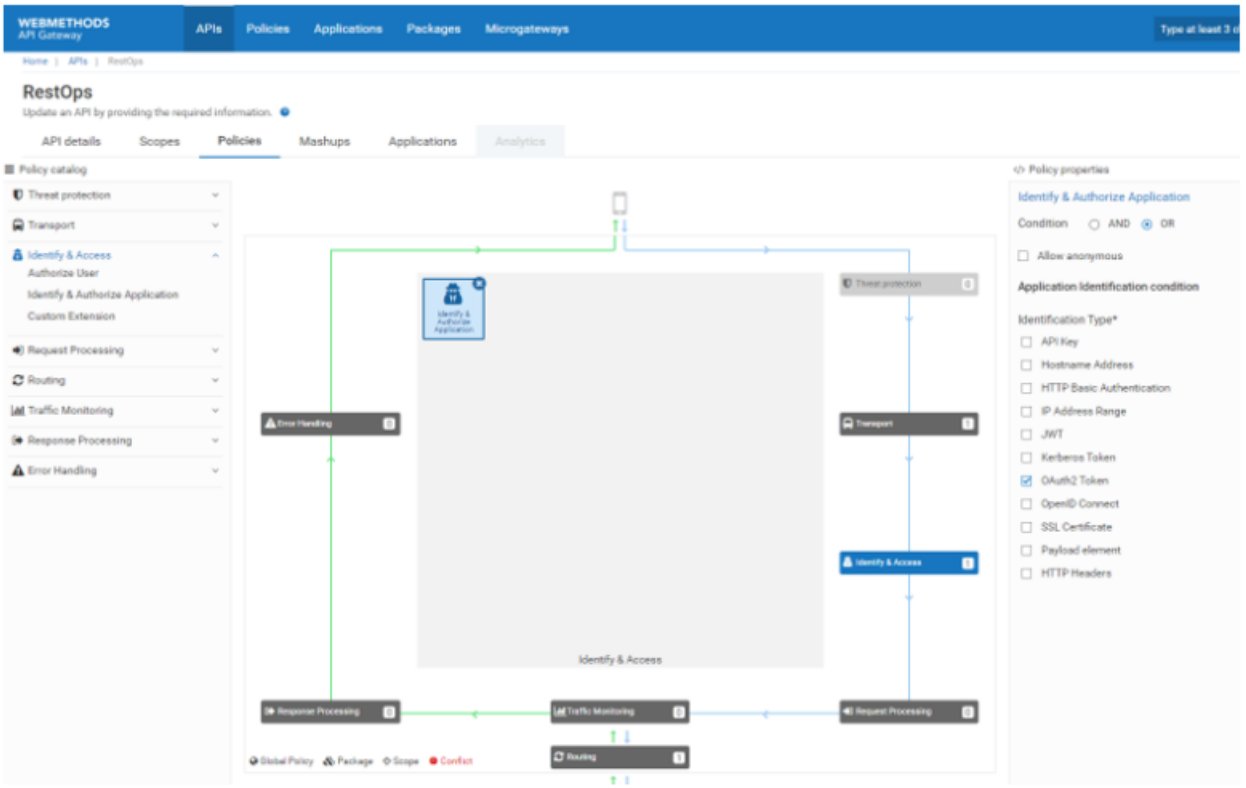
   c. Click the **Policies** tab.

   d. Click on **Edit** to add the policy to API.

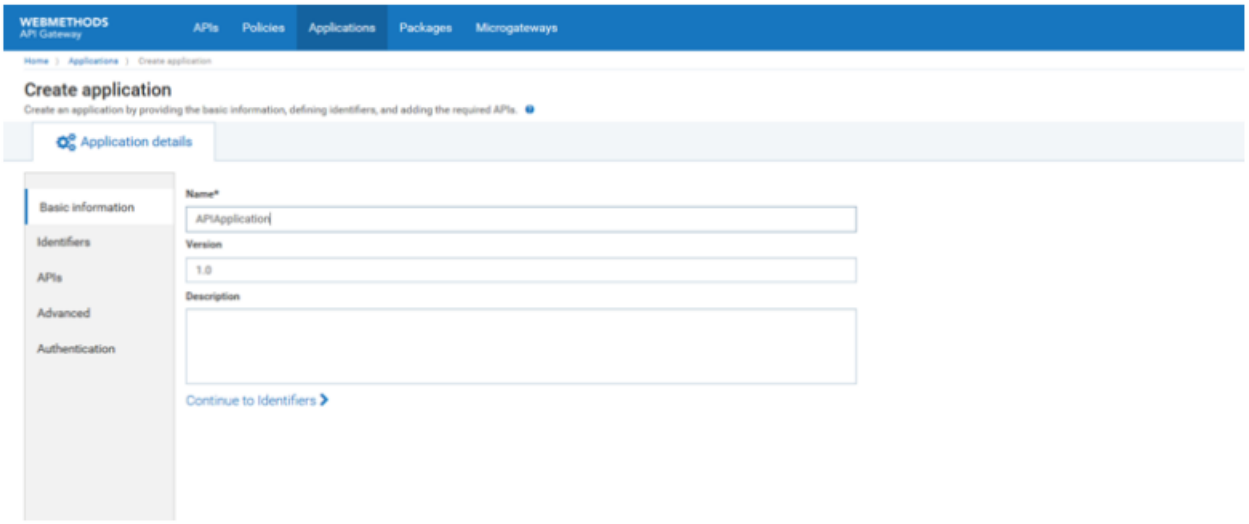
   e. Click **Identify & Access** in the Policy catalog.

   f. Select the **OAuth2 Token** check box in the **Identification Type** field of the Application Identification section and save the changes.

   g. Click **Activate** to activate the API on the API details page of the API.



6. Create an Application in API Gateway with strategy and register it to an API.
   a. Click **Applications** in the title navigation bar.
   b. Click **Create application**.
   c. Provide the name as `APIApplication`.
   d. Click **Continue to Identifiers**.



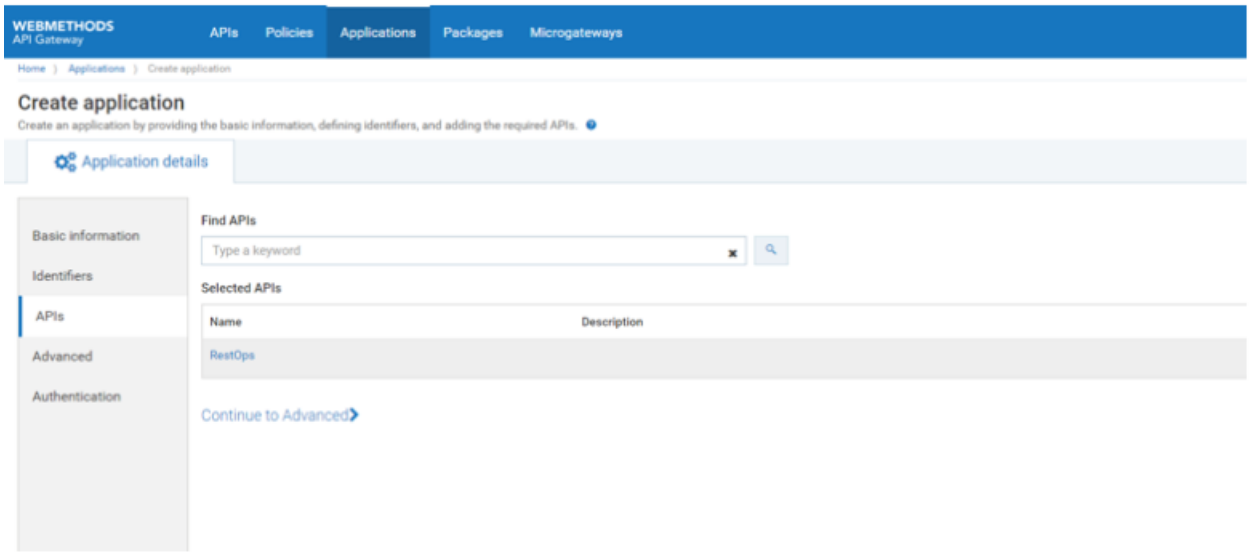
e. Click on **Continue to APIs**.

f. Search the API RestOps, by typing `RestOps` in the **Find APIs** text box.

The RestOps API appears in the **Selected APIs** section.

g. Select the RestOpsAPI.

h. Click **Continue to Advanced**.



i. Click **Continue to Authentication**.

j. Click **Create strategy**.

A strategy is a way to authenticate the incoming request and provide multiple authentication mechanisms or multiple authorization servers for a single authentication scheme. You can create multiple strategies authorized by an API for an application.

k. Provide the name as `AppStrategy`.

l. Enable the toggle button **Generate credentials** to generate the credentials dynamically in the authorization server.

The client-id and client-secret get created automatically.

m. Select `Confidential` from the **Application type** drop-down list.

n. Specify the redirect URIs that the authorization server can use to redirect the resource owner's browser during the grant process. In this example, provide `www.example.com`, which is not a valid URL.

o. Select the required Grant types. In this example, the selected grant types are `authorization_code` and `client_credentials`.

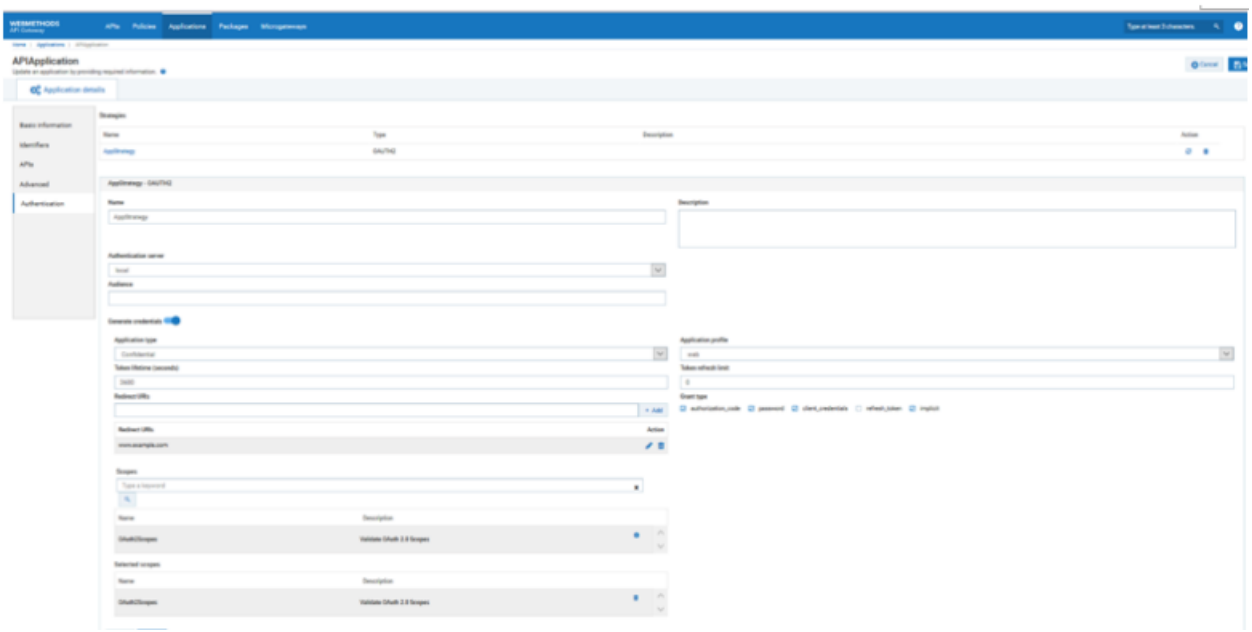
p. Provide `OAuth2Scopes` in the **Scope** text box and click the search icon.

The matching OAuth2Scopes appear.

q. Click **+** sign to add the scope to the Strategy.

r. Click **Add** button at the bottom to add the Strategy.

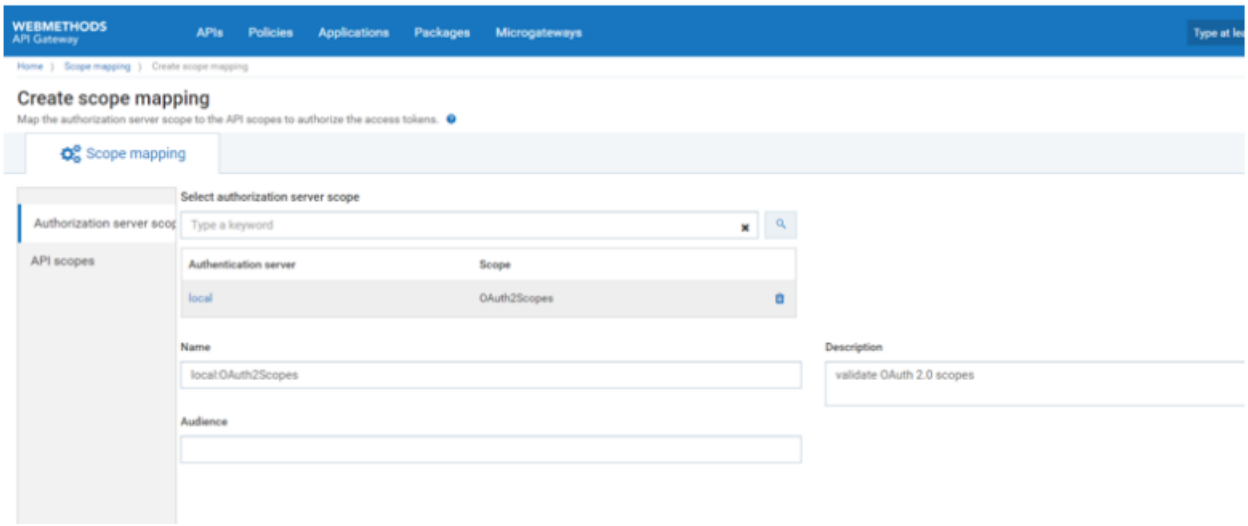
s. Click **Save**.



Step 4: Map OAuth scopes

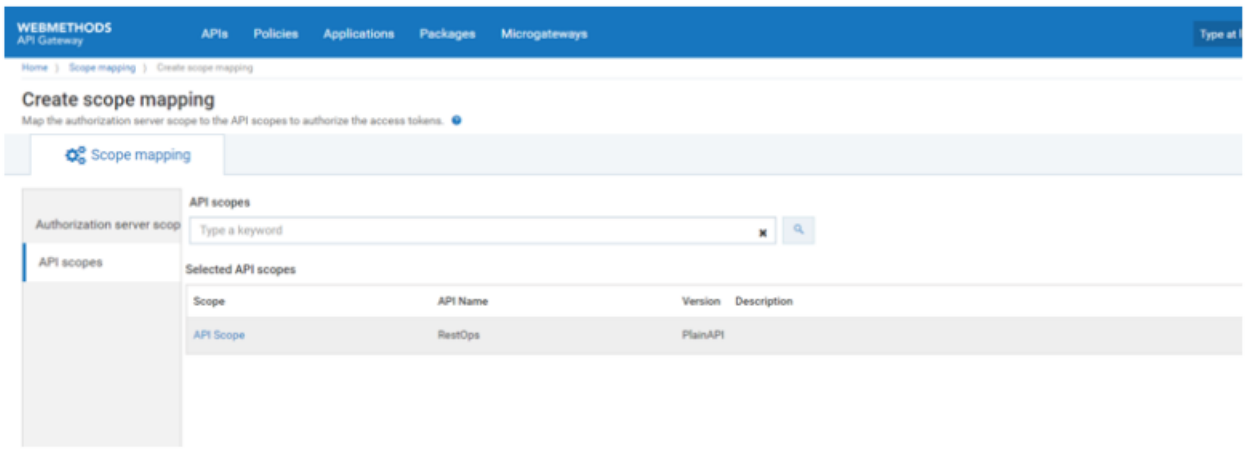
After registering an application, you must map the scope defined in the Authorization server with the APIs in API Gateway to authorize the access tokens to be used to access the protected resources. You can map either a complete API or parts (resources or methods) of an API to the scope or add the scope details and modify the scope details as required from the OAuth or OpenID scopes page. In this example you select the OAuth2Scopes scope.

1. Expand the menu options in the title bar and select **OAuth/OpenID** scopes.
2. Click **Map scope**.
3. Type `OAuth2Scopes` in **Select authorization server scope** and select the listed Authorization server scope from the search list populated.



4. Click **API scopes**.
5. Type RestOps or `API Scope`, which is to be linked to the authorization server, in **API scopes search** text box.
6. Click **Save**.

   The authorization server scope is mapped to the selected API scopes and the authorization scope is listed in the scopes list.

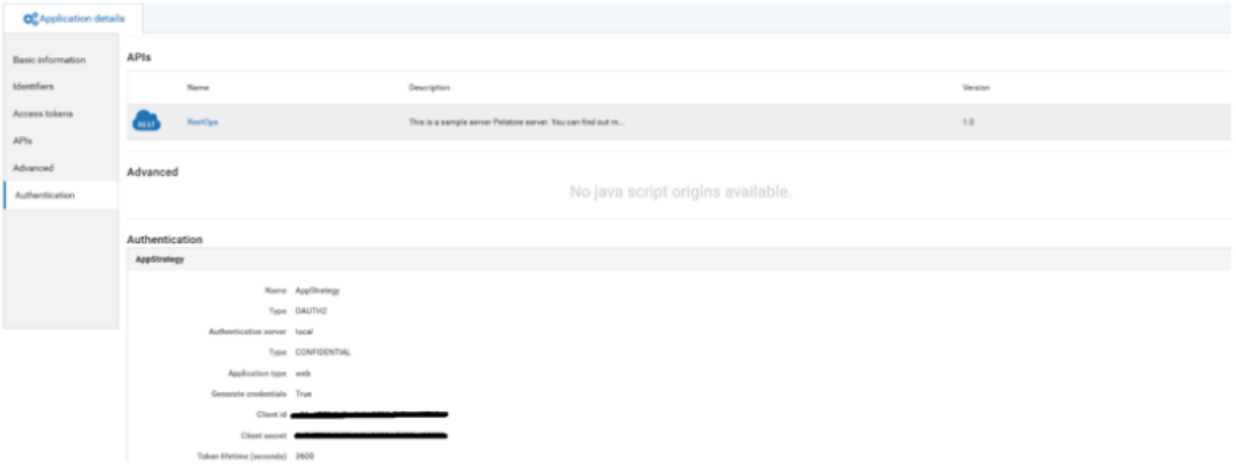


Step 5: Get authorization code

As the grant type chosen is Client credentials, the Client Id and Client secret is required to get the bearer token.

1. In the title bar, click on **Application**.
2. Click the Application `APIApplication`.
3. Click **AppStrategy**.

4. Note the Client id and Client secret.



Step 6: Get the bearer token

Get the OAuth bearer token using external ports.

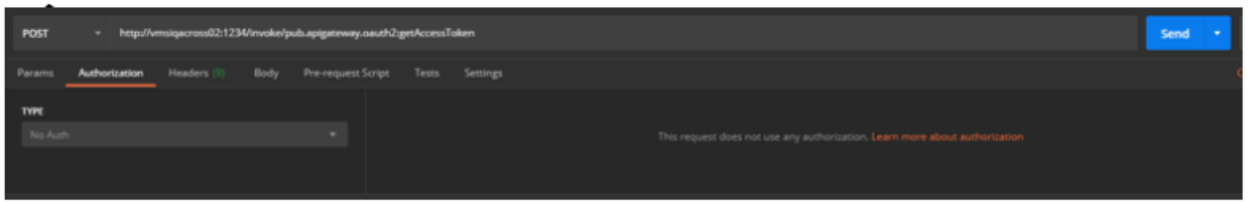
1. Use any Rest client to retrieve the token.

   In the following example, Postman client is used.

2. Make a **POST** call to the following URL, with the hostname of the system where API Gateway is installed in place of localhost and use external port instead of default port.

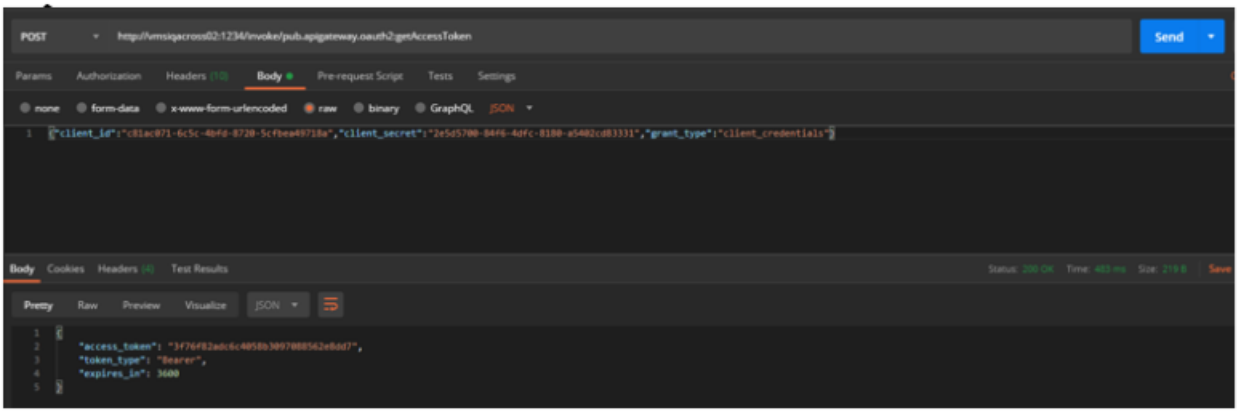
   `http://machinename:1234 /invoke/pub.apigateway.oauth2/getAccessToken`

   `http://localhost:1234 /invoke/pub.apigateway.oauth2/getAccessToken`



3. Provide the following `payload`, with the required `client id` and `client secret` and `grant type`, in the body of the request section. In headers, provide Content-Type as `application/json` (if not specified).

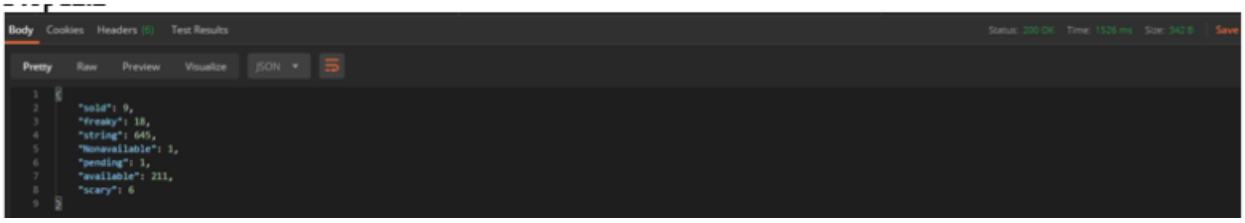
The access token that can be used to access the required application is displayed in the response section.



Step 7: Invoke API

Invoke API using the bearer token. Add the OAuth2 access token as a Bearer token in the request header and send to the API. A successful response, *200 OK* is received from the API with the desired response.

The response is as shown.



Defining Multpile OAuth 2.0 Scopes in API Gateway

Scope is a mechanism in OAuth 2.0 to limit an application's access to a user's account. Scope is required to get an access token. A *scope* is the definition of the resources the client application can access on behalf of a resource owner. In API Gateway, scope is defined for methods or resources of an API. You can apply Scope to an API from the **Identity and Access Policy** page.

Types of Scopes

- **OAuth Scope:** Defined in Authorization Server

- **API Scope:** Scope defined in API

If you do not define scope for an API, a global scope is provided by default. This global scope is applicable for all the methods or resources of the API .

Actors

- Developers with basic knowledge on webMethods API Gateway, Integration Server, OAuth2 architecture
- Customers with basic knowledge on webMethods API Gateway, Integration Server, OAuth2 architecture

Before you begin

Ensure that you have:

- Installed Integration Server with API Gateway
- Knowledge on any REST Client
- API Gateway up and running

Basic Flow

This section explains the following three flows :

- Creating OAuth Scopes in the Local Authorization Server and mapping this scope to the Global Scope for an API
- Creating OAuth Scopes in External Authorization Server and mapping this scope to the Global Scope for an API
- Creating multiple OAuth Scopes in Local Authorization Server and mapping this scope to the Scope defined in the API

Creating OAuth scopes in local authorization server and mapping this scope to the global Scope for an API

1. Create OAuth scope in local authorization server .

   You can create the OAuth scope using the **Authorization Server** page.

   a. Expand the menu options in the title bar and select **Administration**

   b. Select **Security** > **JWT/OAuth/OpenID**. Click **local**. All configurable settings for the local authorization server appears

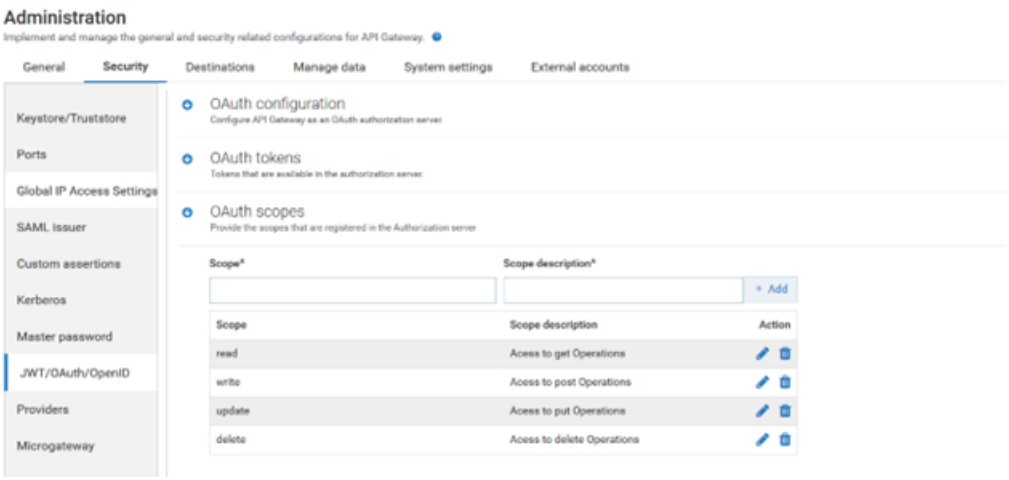
   c. Click **local**

   d. Click **OAuth Scope** to add Scope

   e. Provide the values in the following fields

- **Scope:** OAuth2Scopes
- **Scope description:** OAuth2Scopes

f. Click **Add**.



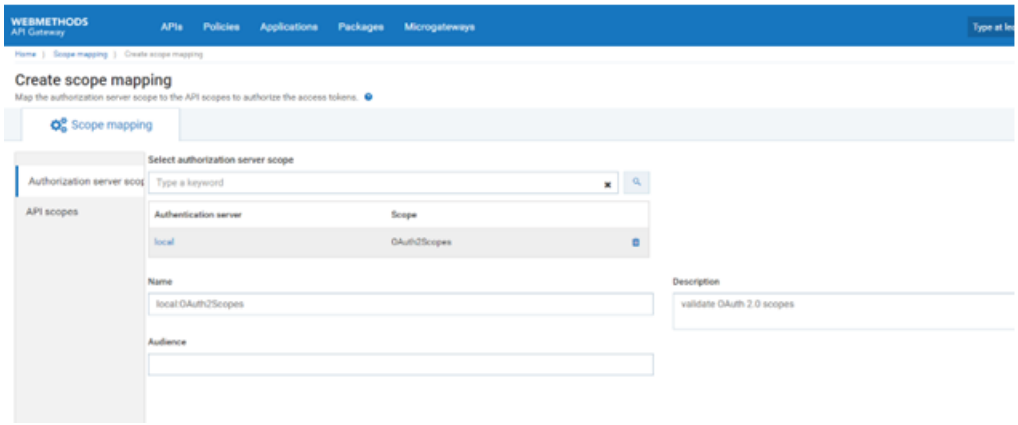
2. Map the OAuth scope to the global scope.

   You must map the OAuth scope to the global scope or to a scope defined for the methods or resources of the API. Here, as the scope for the methods or resources of the API is not created, the methods or resources of the API is not restricted.The global scope is applicable to all the methods or resources of the API.

   In this example, consider the global scope API scopes. Map the OAuth scope to the API scopes for an API *RestOps*.

   a. Expand the menu options in the title bar and select **OAuth/OpenID scopes**.

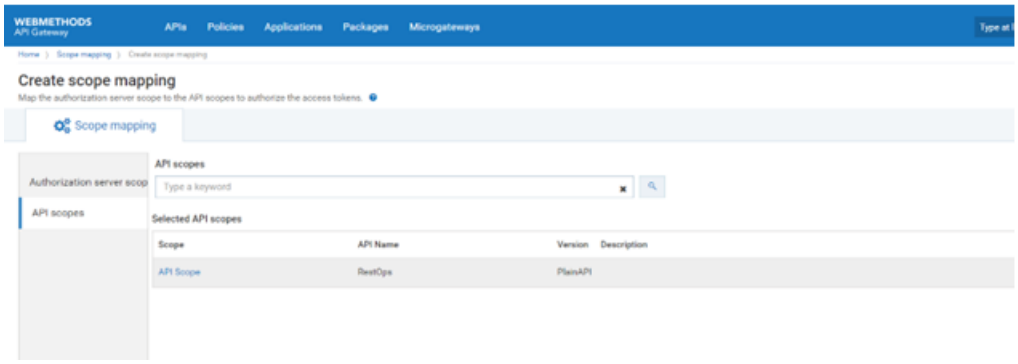
   b. Click**Maps scope**. Type OAuth2Scopes in **Select authorization server scope** and select the listed authorization server scope from list.



   c. Click **API scopes**

d. Type either the the API RestOps or API Scope in **API scopes**. This is the Global scope, which is to be linked to the authorization server.

e. Save the changes. This maps the authorization server scope to the selected API scopes and lists the authorization scope in the scopes list.



For details on other configurations and how to invoke the API, see " Securing APIs using OAuth 2.0 in API Gateway with Local Authorization Server" on page 132

Creating OAuth scopes in External Authorization Server and mapping this scope to the Global Scope for an API

1. Create OAuth scope in external authorization server.

   In the authorization page the external authorization server is defined. Initially scope has to be created in the external authorization server. The same scope needs to be created or gets automatically listed with the discovery URL. Here the external authorization server is OKTA.

   a. Create scope getscope in OKTA.

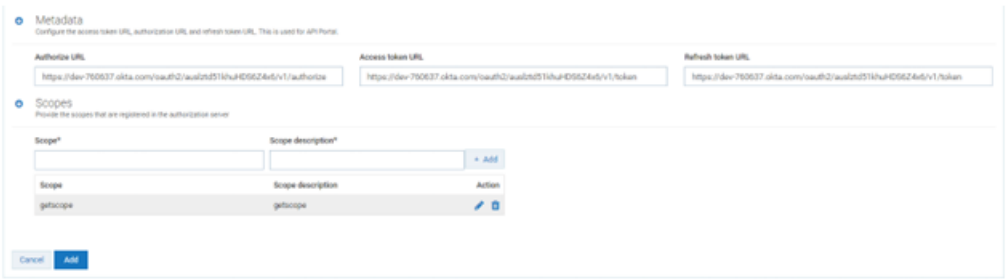
   b. Expand the menu options in the title bar and select**Administration**

   c. Select**Security** > **JWT/OAuth/OpenID.**. Click **local**. This displays all the configurable settings for the Authorization Server

   d. Click on **okta-oauth-server**

   e. Click **Scopes**

   f. Click on **Scopes**. Verify getscope is present in the list.



2. Map the external OAuth scope to the global scope.

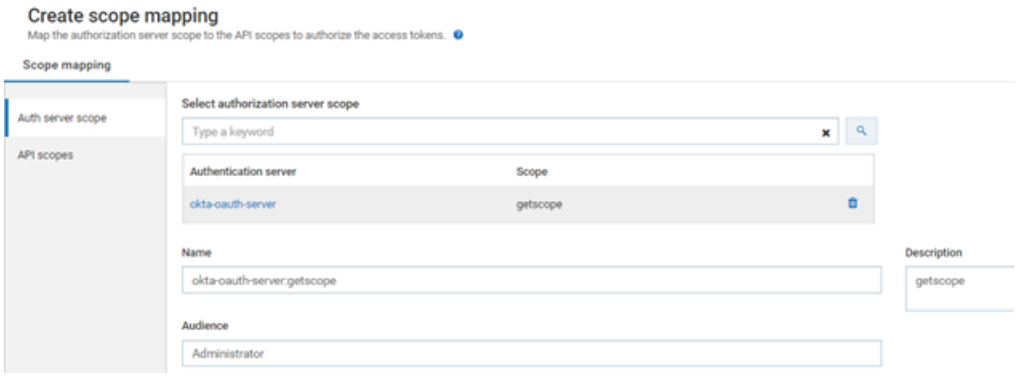
   You must map the scope specified in the OKTA Authorization server with the APIs in API Gateway to authorise the access tokens to access the protected resources.

   a. Expand the menu options in the title bar and select **OAuth/OpenID scopes**.

   b. Click **Map scope**.

   c. Type getscope in **select authorization server scope** and select the listed authorization server scope from the search list populated.

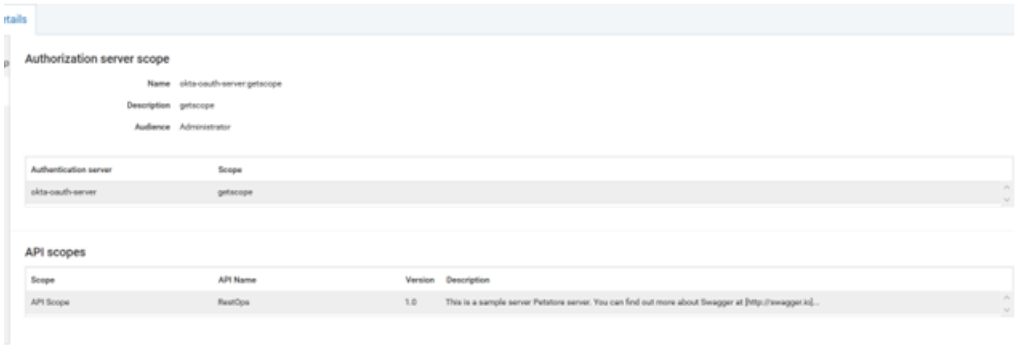
   d. Provide Audience : **Administrator**



   e. Click **API scopes**.

   f. Type `RestOps` or `API scope`, which is to be linked to the authorization server, in API scopes search text box.

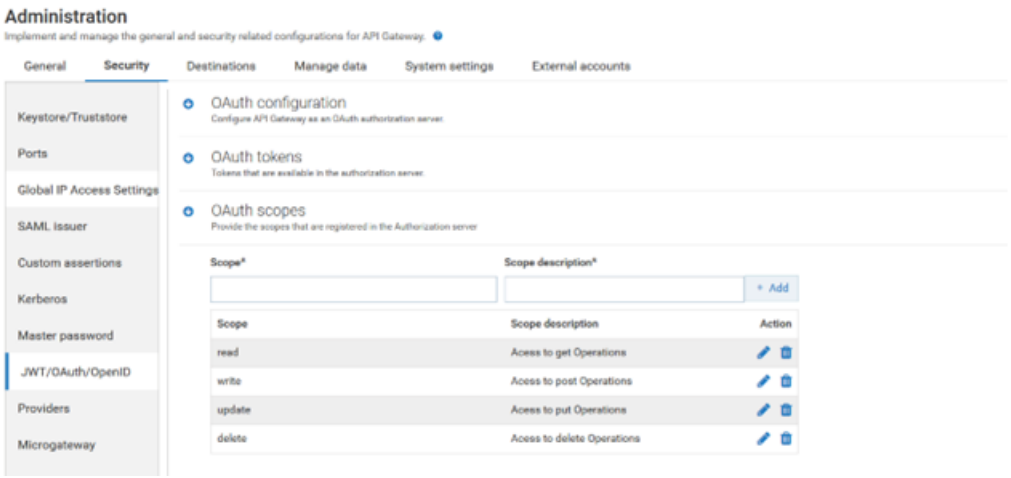
   g. Save the changes. This maps the authorization server scope to the selected API scopes and lists the authorization scope in the scope list



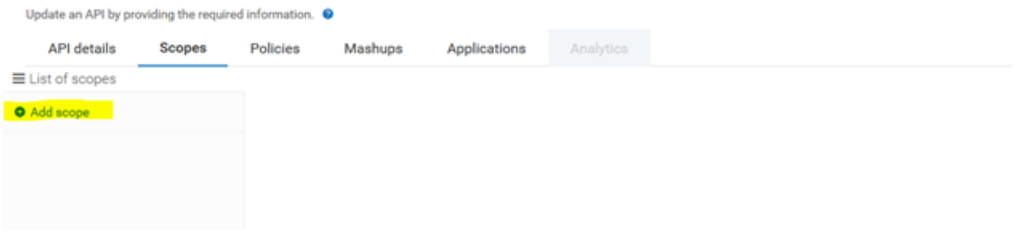
   For details on other configurations and how to invoke the API, see " Securing APIs using OAuth 2.0 in API Gateway using Third Party Authorization Server" on page 155.

Creating multiple OAuth Scopes in Local Authorization Server and mapping this scope to the scope defined in the API

1. Create multiple OAuth scopes in local authorization server.
   a. Expand the menu options in the title bar and select **Administration**
   b. Select **Security** > **JWT/OAuth/OpenID**. Click**Local**. This displays all the configurable settings for the authorization server
   c. Click on **Local**.
   d. Click **OAuth scopes** to add scope. Add the following OAuth scopes `read`, `write`, `modify`, and `delete` and click **Add**.
   e. Click **Update**.



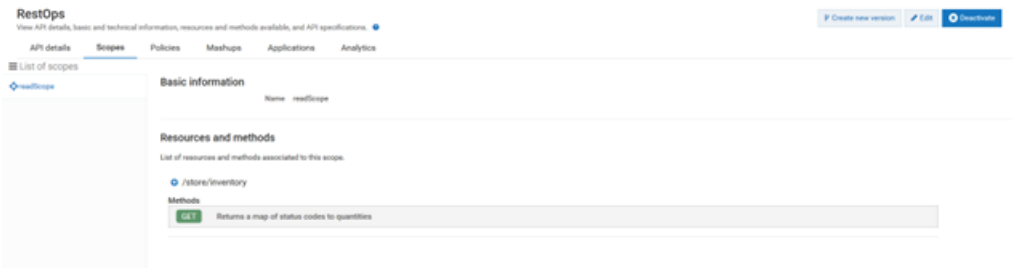
2. Create scopes for the API.
   a. Navigate to **APIs** in the title navigation bar.
   b. Click **RestOps**.
   c. Click **Scopes**.
   d. Click **Edit**.
   e. Click **Add Scope**.



f. Provide the name: *readScope*

g. In **Resources and methods** select GET for /store/inventory method. You can select other GET methods as well.

h. Click **Save**.

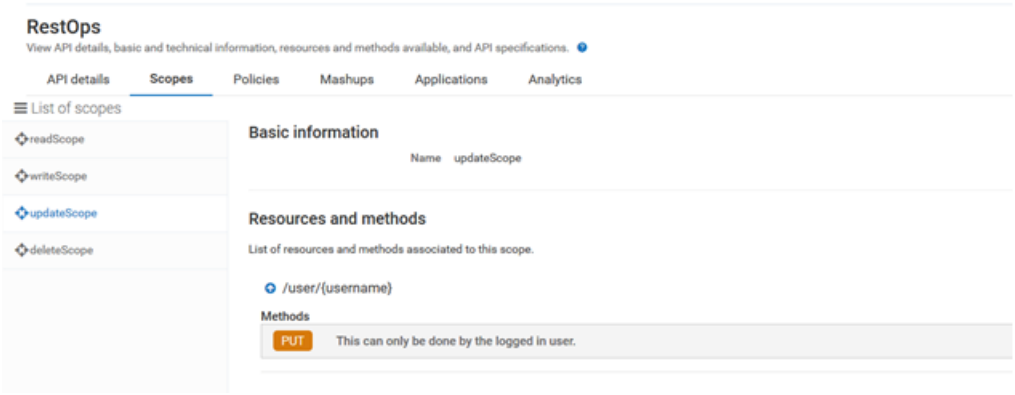


i. Click **Edit**.

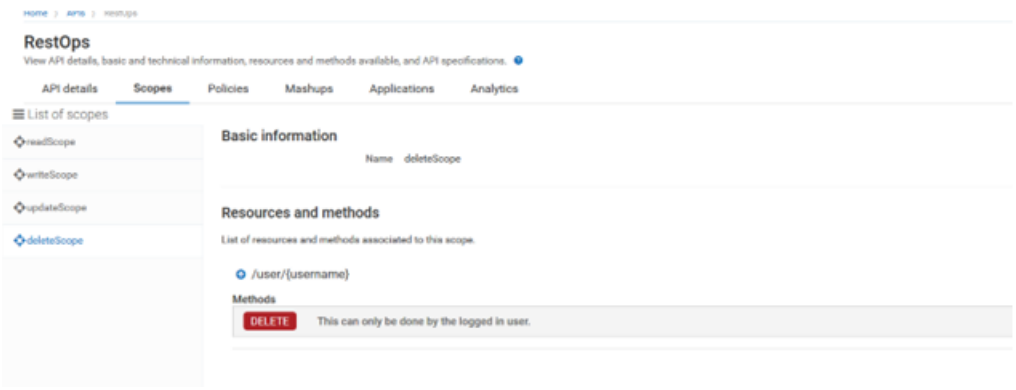
j. Click **Add Scope**.

k. Provide name: `writeScope` and select POST in /user method.

l. Create scopes updateScope and select PUT in /user/{username}



m. Create scope deleteScope and select DELETE in /user/{username}



3. Enforce OAuth2 policy on the API RestOps with the required scope.

   This policy ascertains that a OAuth token is required to access this API.

   a. Click **API** in the title navigation bar.

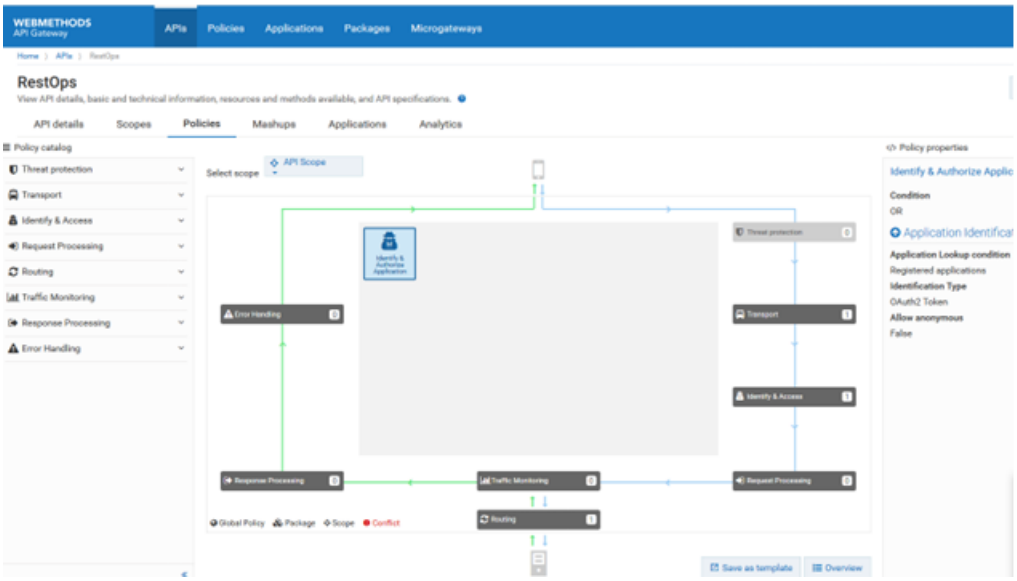
   b. Click **RestOps**.

   c. Click **Policies**.

   d. Click **Edit**.

   e. Click **Identify & Access** from the Policy catalog section .

   f. Click **Save**.

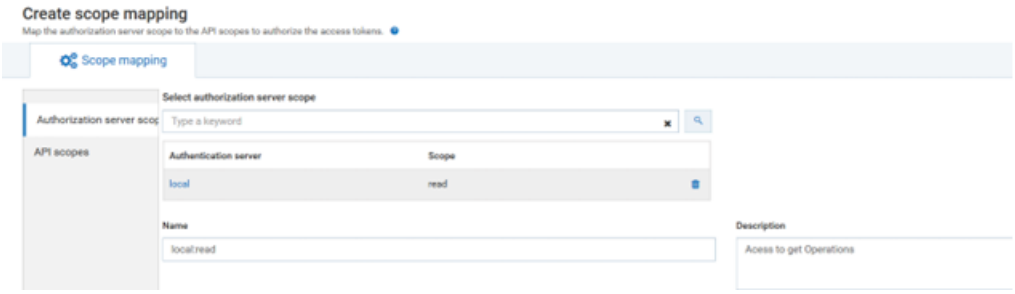


4. **Map the OAuth scope to the API-level scope.**

   Here, you map the OAuth scope that was defined in the local Authorization server with the API-level scope defined in the RestOps API.

   a. Expand the menu options in the title bar and select **OAuth/OpenID** scopes.

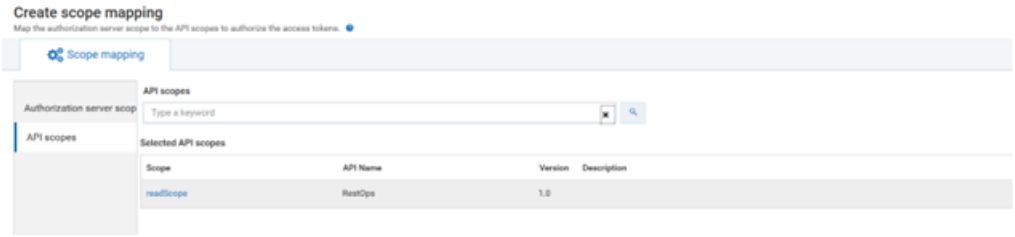
b. Click Map Scope, type `read` in select authorization server scope and select the listed authorization server scope from the search list populated.



c. Click **API scopes.**

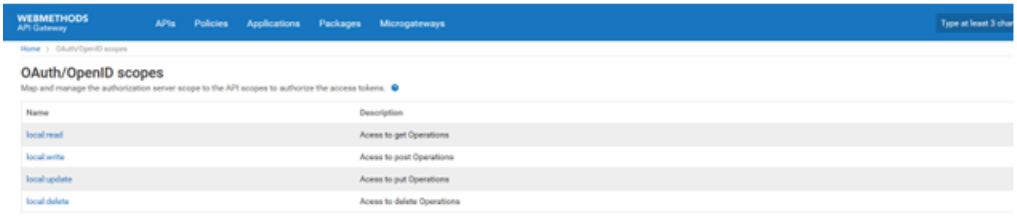
d. Type `Restops` or `readScope`, which is to be linked to the authorization server, in API scopes search text box.

e. Save the changes. This maps the authorization server scope to the selected API scopes and lists the authorization scope in the scopes list.



f. Similarly Map

- write OAuth Scope to *writeScope* of API.
- update OAuth Scope to *updateScope* of API.
- delete OAuth Scope to *delete Scope* of API.

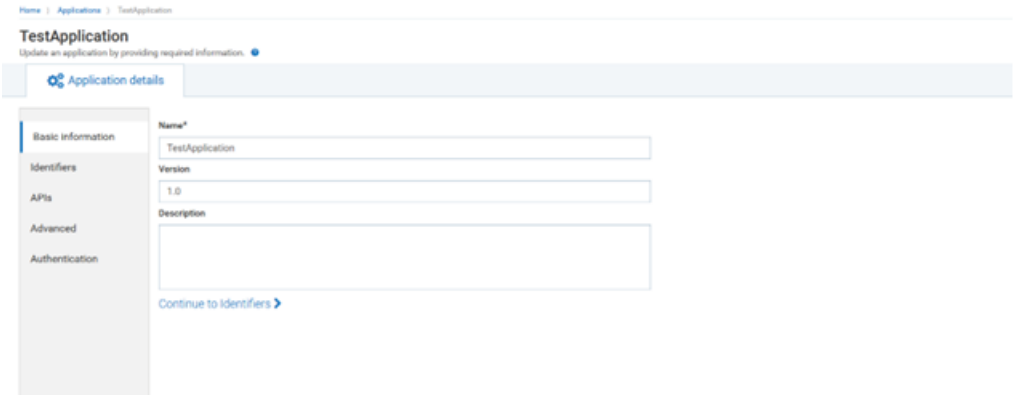


5. Configure application and strategy.

a. Click **Applications** in the title navigation bar.

b. Click **Create application** provide the Name: `TestApplication`.

c. Click **Continue to Identifiers**.



d. Click on **Continue to APIs**.

e. Type `RestOps` in the **Find APIs** text box. The API is listed in the drop down box.

f. Select the `RestOps` API.

g. Click **Continue to Advanced**. In the Advanced page, no input is required.

h. Click on **Continue to Authentication**.

i. Click **Create Strategy**. A strategy is a way to authenticate the incoming request and provides multiple authentication mechanisms or multiple authorization servers for a single authentication scheme. You can create multiple strategies authorized by an API for an application.

j. Provide the Name: `TestStrategy`

k. Enable the toggle button **Generate credentials** to dynamically generate the client in the authorization server. By enabling the toggle button client-id and client-secret are created automatically.

l. Select **Confidential** as application type.

m. Specify the redirect URIs that the authorization server can use to redirect the resource owner's browser during the grant process. In this case, you are provided with www.example.com which is not a valid URL.

n. Select the required Grant types, `authorization_code` and `client_credentials`.

o. Provide the following scopes in the scope text box and click the search icon.

   a. read, click **Add**

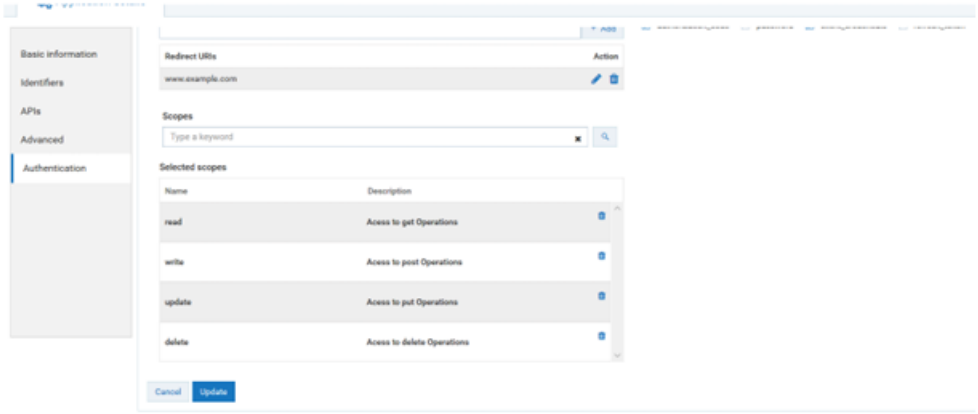
   b. write, click **Add**

   c. update, click **Add**

   d. delete, click **Add**

p. Click **Add** to add the trsategy.

q. Click **Save**.



6. Get the authorization code.

   You require the client_id and client_secret to get the authorization code.

   a. Navigate to Applications in the title Menu.

   b. Click on **TestApplication** .

   c. Under Authentication, click `TestStrategy` strategy.

   d. Copy the client id.

   e. Invoke the following url to get the authorization code

   ```
   http://<machinename>:4444/invoke/pub.oauth/authorize?
   client_id=01fc7b74-1f56-48d5-81fe-bcd6e895d40f&
   redirect_uri=www.example.com&response_type=code&state=121
   ```

   f. If you select read, you get the authorization code only for read scope

   Example `http://<machinename>:4444/invoke/wm.server.oauth/www.example.com?code=4c4f499e6b894103972bd12c6e8e49d7&grant_type=authorization_code&redirect_uri=www.example.com&state=121&scope=read`

   g. If you select only write, you get the authorization code only for write scope

Example `http://<machinename>:4444/invoke/wm.server.oauth/www.example.com?code=e295940d2daa4ac3887575400c81b78b&grant_type=authorization_code&redirect_uri=www.example.com&state=121&scope=write`

h. If you select all, you will get the authorization code only for read+write+update+delete scope

Example `http://<machinename>:4444/invoke/wm.server.oauth/www.example.com?code=3cbcfb3523624675a54925bd96b56bea&grant_type=authorization_code&redirect_uri=www.example.com&state=121&scope=read+write+update+delete`

Retrieving OAuth Token

You must retrieve an OAuth token to access an API that is OAuth protected.

To retrieve an OAuth token

1. Open your REST client.
2. Make a POST call to the following URL, with the hostname of the system where API Gateway is installed in place of localhost:

```
http://localhost:5555/invoke/pub.apigateway.oauth2/getAccessToken
```

For example

```
http://10.2.120.14:5555/invoke/pub.apigateway.oauth2/getAccessToken
```

3. Provide the following payload, with the required client id and client secret, in the **Request** section:

```
{
            "grant_type":"client_credentials",
            "client_id":"client id",
            "client_secret":"client secret"
}
```

You can find Client id and Client secret in the **Authentication** section of the **Application details** page.

For example

```
{
            "grant_type":"client_credentials",
            "client_id":"0abcd80e-f009-4a38-b52e-e663b2e18e5b",
            "client_secret":"3bd9c383-813e-40d4-b876-67c4da7c71cc"
}
```

The access token that can be used to access the required application is displayed in the **Response** section.

Sample response

```
{
    "access_token": "c9a39e14e6a84be0b228bc9bcb76ad99",
    "token_type": "Bearer",
    "expires_in": 3600
}
```

Secure API using OAuth2 with refresh token workflow

When using the authorization code grant type to get the access token, you need to get the permission from the resource owners at least for the first time. In the subsequent attempts to get the access token, if you do not want to get the permission from the resource owners, then you can use the refresh token.

This use case explains how to secure the API using OAuth2 authentication strategy. It also explains the refresh token workflow in detail.

Configuring OAuth2 Authentication with Refresh Token

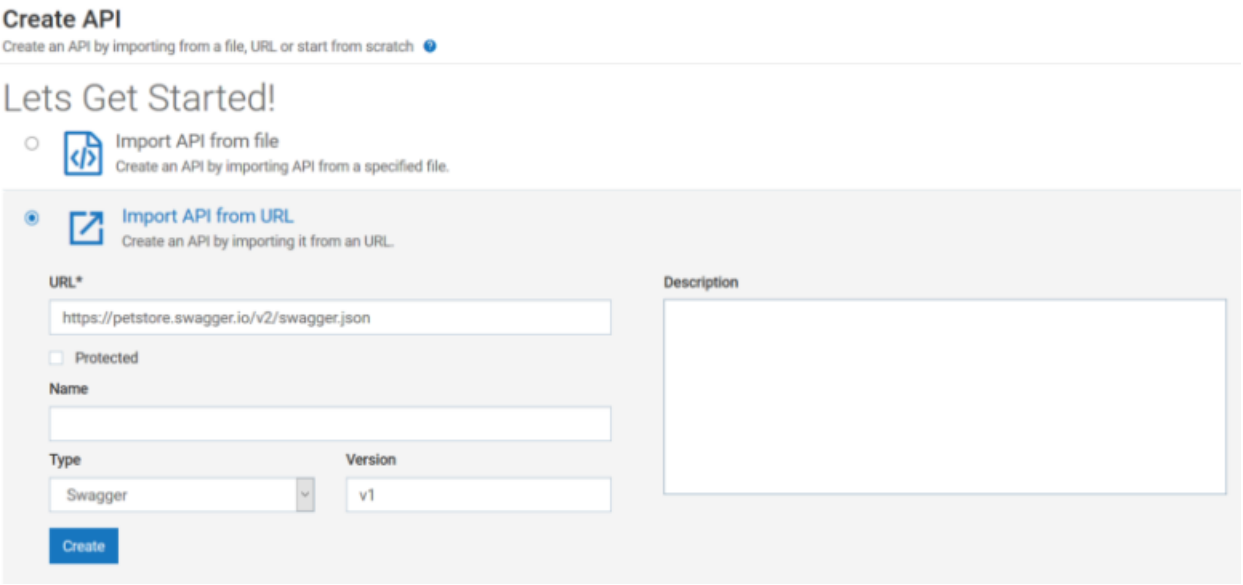
This use case explains how to secure the API using OAuth2 authentication strategy with *authorization_code* and *refresh_token* grant types.

The use case starts when you create an API and ends when you create an application strategy with OAuth2 authentication scheme.

To configure OAuth2 Authentication with Refresh Token

1. Create an API.

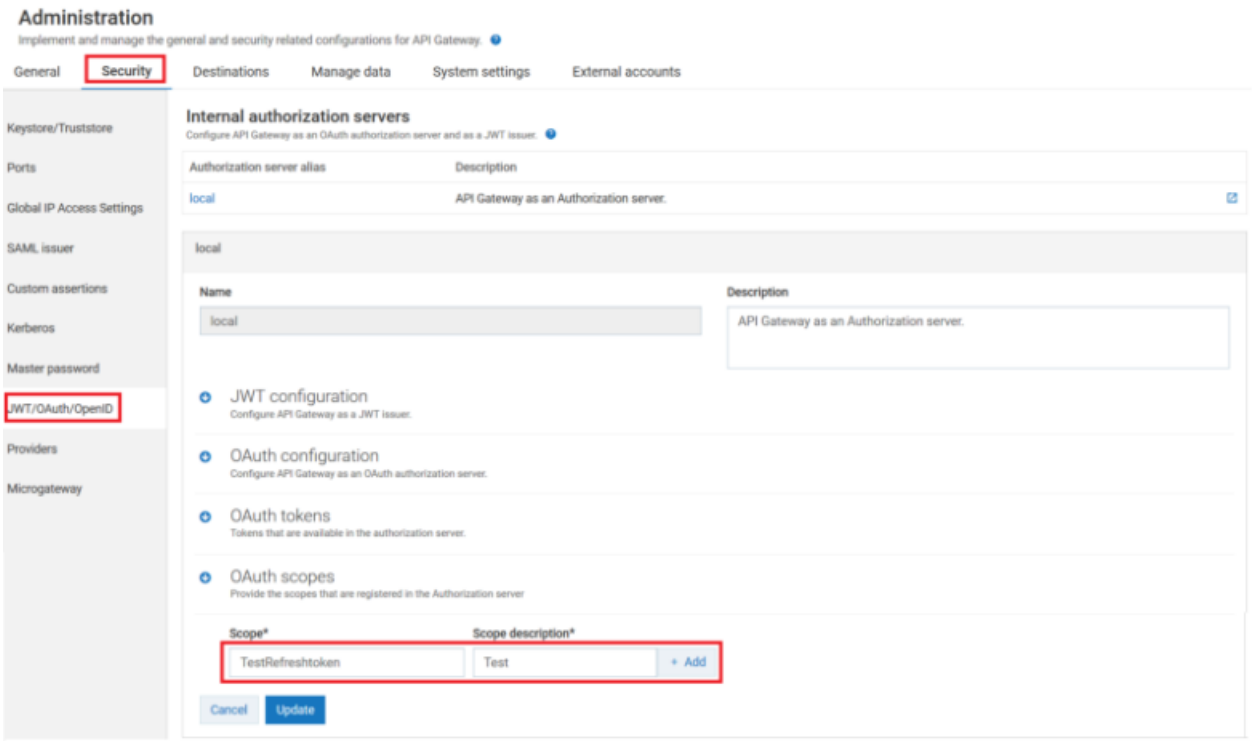
   For details about creating an API, see "Creating a REST API from Scratch" on page 20.



2. Enable the OAuth2 token identification type in the **Identify & Authorize** policy.

For details about **Identify & Authorize** policy, see "Identify & Authorize" on page 111.

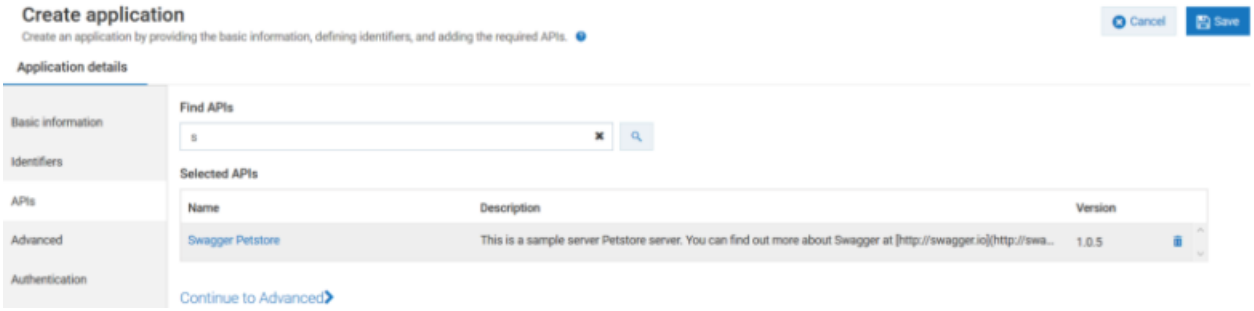
3. Create OAuth scope in the local authorization server.



4. Map the OAuth scope to the API scope.

For details about mapping OAuth scope, see *webMethods API Gateway Administration*.

5. Create an application with OAuth2 authentication strategy.

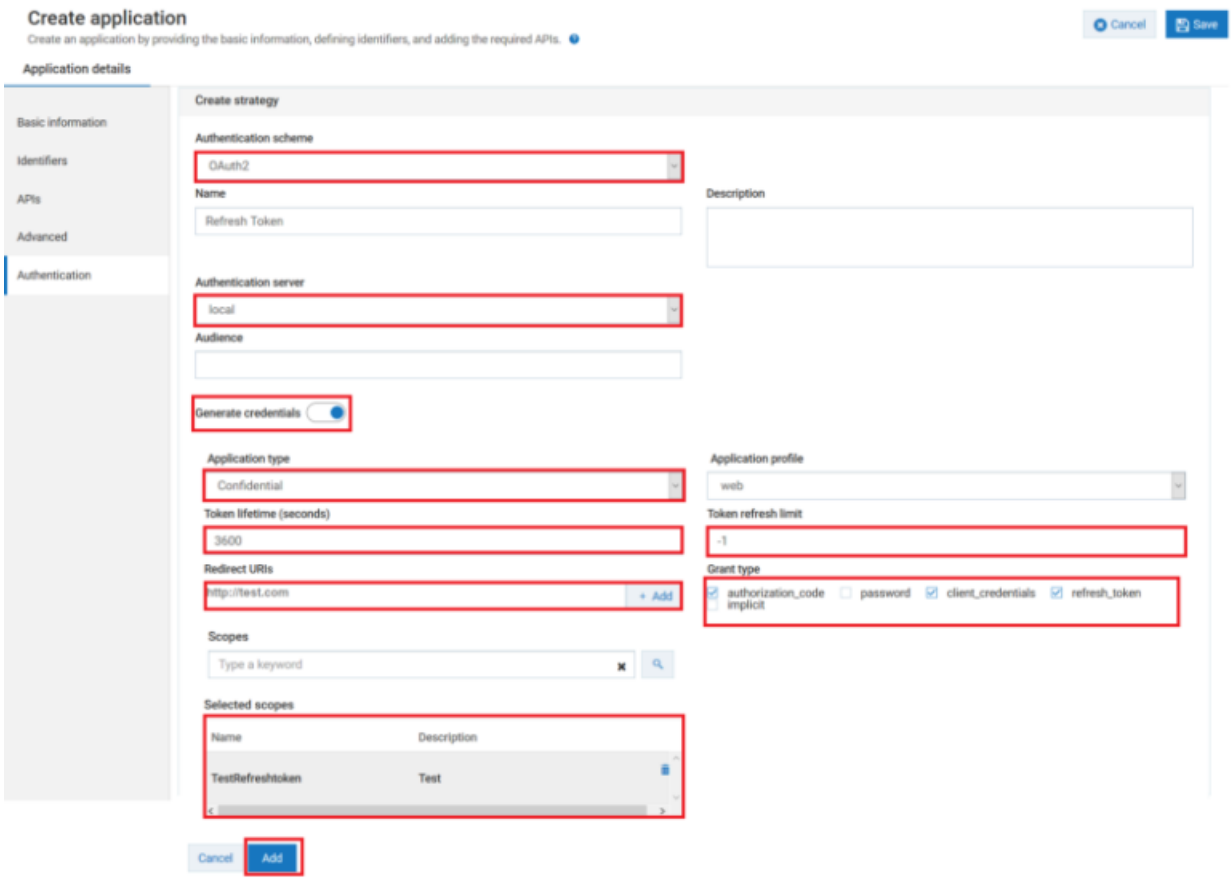


a. Create a new application.

   For details about creating an application, see "Creating an Application" on page 80.

b. Associate the application with the API that you have created.

c. Click the **Authentication** tab to create strategy with OAuth2 authentication.



d. Select the **Authentication schemes** as *OAUTH2*.

e. Specify the **Authentication server** as *local*.

f. Enable the **Generate credentials** toggle button to generate the client dynamically in the authorization server and provide the following information:

   a. Select the **Application Type** as *Confidential*. A confidential client is an application that can keep a client password confidential to the world. This client password is assigned to the client app by the authorization server. This password is used to identify the client to the authorization server, to avoid fraud. An example of a confidential client could be a web app, where no one but the administrator can get access to the server, and see the client password.

   b. Select the application profile from the **Application profile** drop-down menu. For example, web.

   c. Specify the duration in seconds for which the access token is active in the **Token lifetime (seconds)**.

   d. Specify the number of times you can use the refresh token in the **Token refresh limit** to get a new acceswws token.

Note:
To use refresh token unlimitedly, specify the limit as -1.

e. Specify the URIs that the authorization server can use to redirect the resource owner's browser during the grant process. You can add multiple URIs by clicking **+Add**.

f. Specify the grant type to be used to generate the credentials. For this specific use case, we have selected *authorization_code*, *client_credentials*, and *refresh_token*, which are dynamically populated from the authorization server.

Note:
Make sure you have selected *refresh_token* **grant_type**, if you want to get the refresh tokens.

g. Select the scopes that are to be mapped for the authentication strategy.

h. Click **Add** to save the strategy.

i. Click **Save** to save the application.

Refresh Token Process Flow

This use case explains the following workflow:

1. How to get the access token with resource owner permission?
2. How to get the access token without resource owner permission using refresh token in the subsequent attempts?

How to get the access token with resource owner permission?

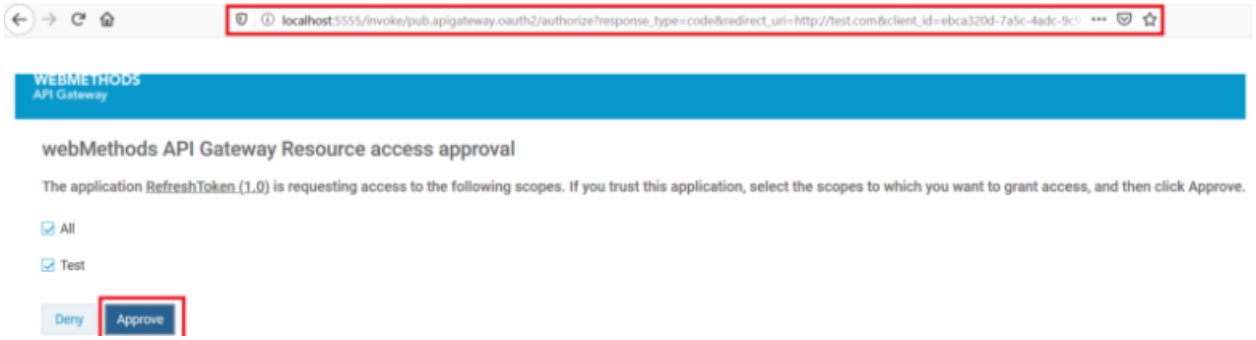
This use case starts when you get the authorization code and ends when you access then API.

To get access token using authorization code grant type (With resource owner permission)

1. Get authorization code.

   a. Click the http(s)://hostname:port/invoke/pub.apigateway.oauth2/authorize?response_type=code&redirect_uri=<*redirectURI*>&client_id=<*Client ID*>.

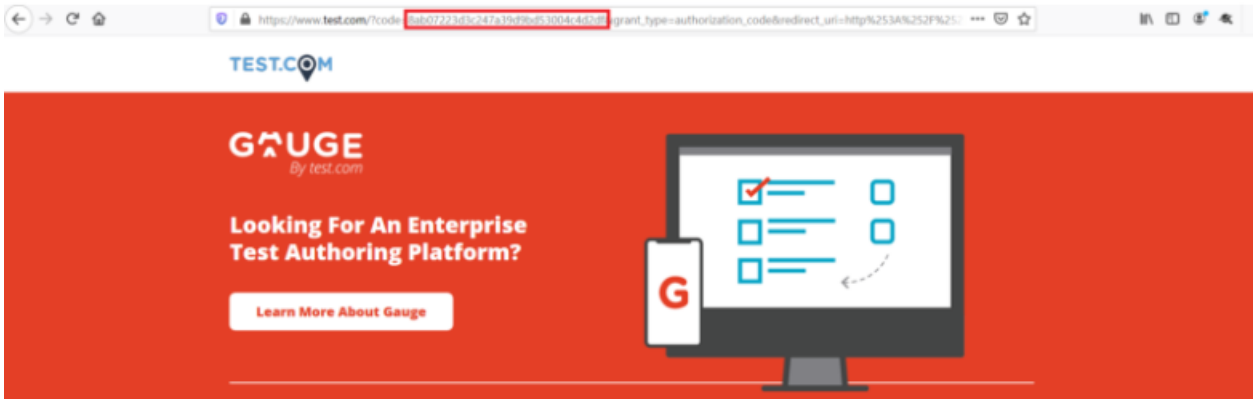
   Note:
   Make sure you have replaced the <*redirectURI*> and <*ClientID*> in the above mentioned URL. You can get the redirect URI and client ID from the **Authentication** tab of the **Application** screen.



b. Click the **Approve** button.

c. Provide the credentials of your API Gateway instance.

You are re-directed to the redirect URI as per to the configuration. The below screenshot is just a sample, you will be redirected to a different URL based on your configuration and so the screenshot varies accordingly. If the given redirect URI is not a valid web page, you may get a *Page not found* error, which is fine, because you can get the authorization code value from the browser URL.



d. Make a note of the authorization code that is displayed in the address bar of the browser. As highlighted in the above image's URL, you can see the authorization code in the *code=* field of the URL.

2. Get Access Token.

a. Invoke the access token endpoint.

Request: POST http(s):// hostname:port /invoke/pub.apigateway.oauth2/getAccessToken

In the **Authorization** tab, select the authorization type as *Basic Auth*. Provide the client ID as username and client secret as password. You can get the client ID and client secret in the **Authentication** tab of the **Application** screen.

Sample request body

```
{
                "redirect_uri":"http://test.com",
                "scope":"email",
                "grant_type":"authorization_code",
                "code":"4b4b16c68f1c4b6fa7f26e0cb00b5daa"
}
```

Note:
You must replace the redirect_URI, scope, and code with appropriate values. For the code field value, make sure you use the authorization code that you have noted down in the previous step.

Sample response body

```
{
     "scope": "TestRefreshtoken",
     "access_token":
"c92b6227a19c46f1a6545bf370bb6ee6e30ff87957ef4b1aaa9577f7e86e4bd7",
     "refresh_token":
"f78dd4fc5b8d4d799cf066427e828e26ce7e3723e4334416a7b9cd8a274e6947",
     "token_type": "Bearer",
     "expires_in": 3600
}
```

3. Access API using the REST API client.

   In the **Authorization** tab, select the authorization type as *Bearer Token* and provide the access token that you get from the response payload of the previous step.

How to get the access token without resource owner permission using refresh token in the subsequent attempts?

This use case starts when you get the authorization code and ends when you access the API.

To get access token using refresh token (Without resource owner permission)

When the access token expires and if you need to access the same API, you need to get another access token. If you have refresh token, you can get a new access token without getting the permission from the resource owner.

1. Invoke the refresh token endpoint.

   Request: POST http(s)://hostname:port/invoke/pub.oauth/refreshAccessToken

   In the **Authorization** tab, select the authorization type as *Basic Auth*. Provide the client ID as username and client secret as password. You can get the client ID and client secret in the **Authentication** tab of the **Application** screen.

   Sample request body

```
{
               "grant_type":"refresh_token",
"refresh_token":"f78dd4fc5b8d4d799cf066427e828e26ce7e3723e4334416a7b9cd8a274e6947"
}
```

Note:
Make sure you have replaced the refresh token that you got from the Step 2 using " How to get the access token with resource owner permission?" on page 206 use case.

Sample response body

```
{
     "grant_type": "refresh_token",
          "refresh_token":
"f78dd4fc5b8d4d799cf066427e828e26ce7e3723e4334416a7b9cd8a274e6947",
      "scope": "TestRefreshtoken ",
       "access_token":
"c102bcaebecf451ca705bf54d26fae732ea9790a0ff64a87a010b3875b4b8da2",
      "token_type": "Bearer",
      "expires_in": 3600
}
```

2. Access API using the REST API client.

   In the **Authorization** tab, select the authorization type as *Bearer Token* and provide the access token that you get from the response payload of the previous step.

JWT Authentication Use case and Workflow

JSON Web Token is a JSON-based open standard (RFC 7519) means of representing a set of information to be securely transmitted between two parties. A set of information is the set of claims (claim set) represented by the JWT. A claim set consists of zero or more claims represented by the name-value pairs, where the names are strings and the values are arbitrary JSON values. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure, enabling the claims to be digitally signed. JWTs can be signed using a shared secret (with HMAC algorithm), or a public or private key pair using RSA.

API Gateway can generate a JWT token itself or validate the JWT token generated by a trusted third-party server. API Gateway uses the RSA-based JWT to provide stronger integrity protection to JWTs when API Gateway is the issuer of the token. The JSON-based access tokens contain one or more claims. A claim is any piece of information that serves as an unique identifier, and that the token issuer who generated the token has verified. API Gateway extracts the claims from the JWT, identifies the application and then authorizes access to the protected resource.

Note:
JWT authentication is supported for both REST and SOAP APIs. API Gateway does not support Base 64 encoded JWT tokens.

Use case 1: JWT authentication with API Gateway as a JWT issuer

This describes the high level workflow for the scenario where API Gateway can generate the JSON Web Token itself.

1. Configure API Gateway as an internal authorization server.

For a complete procedure on configuring API Gateway as an internal authorization server, see *webMethods API Gateway Administration*.

2. Enforce the Identify & Authorize policy on the API.

   Ensure to select JWT. For more details, see "Identify & Authorize" on page 111.

3. Associate an application with the API.

   You can create a new application or use an existing one. Ensure that you add the required claims while creating the application, which you would use to validate the access token. For a complete procedure on creating an application with a strategy, see "Creating an Application" on page 80.

4. Activate the API.

   User on invoking the API uses the JWT identification method to access the protected resource.

5. You get the JWT in one of the following ways (with or without claims), which you can pass as a bearer token to invoke the API.

   - **Retrieve JWT Token** - For a complete procedure on retrieving a JWT token, see "Retrieving JWT Token" on page 213.
   - **Retrieve JWT Token with Claims** - For a complete procedure on retrieving a JWT token with claims, see "Retrieving JWT Token with Claim" on page 214.

Use case 2: API Gateway with an external JWT issuer

This describes the high level workflow for the scenario where API Gateway accepts JSON Web Token generated by a trusted third-party server.

1. Configure an external authorization server.

   For a complete procedure on configuring an external authorization server, see *webMethods API Gateway Administration*.

2. Enforce the Identify & Authorize policy on the API.

   Ensure to select JWT. For more details, see "Identify & Authorize" on page 111.

3. Associate an application with the API.

   You can create a new application or use an existing one. Ensure that you add the required claims while creating the application, which you would use to validate the access token and the external authorization server that would be the JWT issuer. For a complete procedure on creating an application with a strategy, see "Creating an Application" on page 80. For information about configuring authorization server for the applications created from API Portal, see "OAuth Authentication in API Gateway" on page 123.

4. Activate the API.

   User on invoking the API uses the JWT identification method to access the protected resource.

5. Pass the JWT as a bearer token to invoke the API.

Use case 3: JWT authentication with API Gateway for applications registered from API Portal

This use case describes the high-level workflow for the scenario where API Gateway generates the JSON Web Tokens for the applications registered from API Portal. From API Portal, you can create applications for APIs that require JWT access tokens to access them and test APIs from API Portal.

1. Configure an internal or external authorization server in API Gateway.

   For a complete procedure on configuring API Gateway as an internal authorization server, see *webMethods API Gateway Administration*.

   For a complete procedure on configuring an external authorization server, see *webMethods API Gateway Administration*.

2. Create an API.

   For a complete procedure on creating APIs, see " Defining and Managing APIs" on page 10.

3. Enforce the Identify & Authorize policy on the API.

   Ensure to select JWT. For more details, see "Identify & Authorize" on page 111.

4. Provide the name of the authorization server in the **watt.server.oauth.authServer.alias** settings in the Administration section of API Gateway.

5. Publish the API to API Portal.

6. Log in to API Portal.

7. Open the API that you published from the **API gallery** page.

8. Click **Get access token** from the right pane of the **API details** page request an access token to access and use the API.

9. In the **Request API access token** dialog box, provide the **Application name** and **Application description**. The application is created and listed in the **Applications** page.

10. Click **Try API**.

11. Select the required application from the **Application** drop-down list in the left pane.

12. Select the resource, that you want to try, from the left pane.

13. In the **Authorization** tab, select `JWT` from the **Authorization type** drop-down list.

14. Do one of the following:

    - Provide your Integration Sever credentials in the **User name**, **Password** field, and click **Get token**. Select a token from the available list of tokens, and click **Update**.
    - Provide the JWT token or select one from the available list of tokens, and click **Update**.

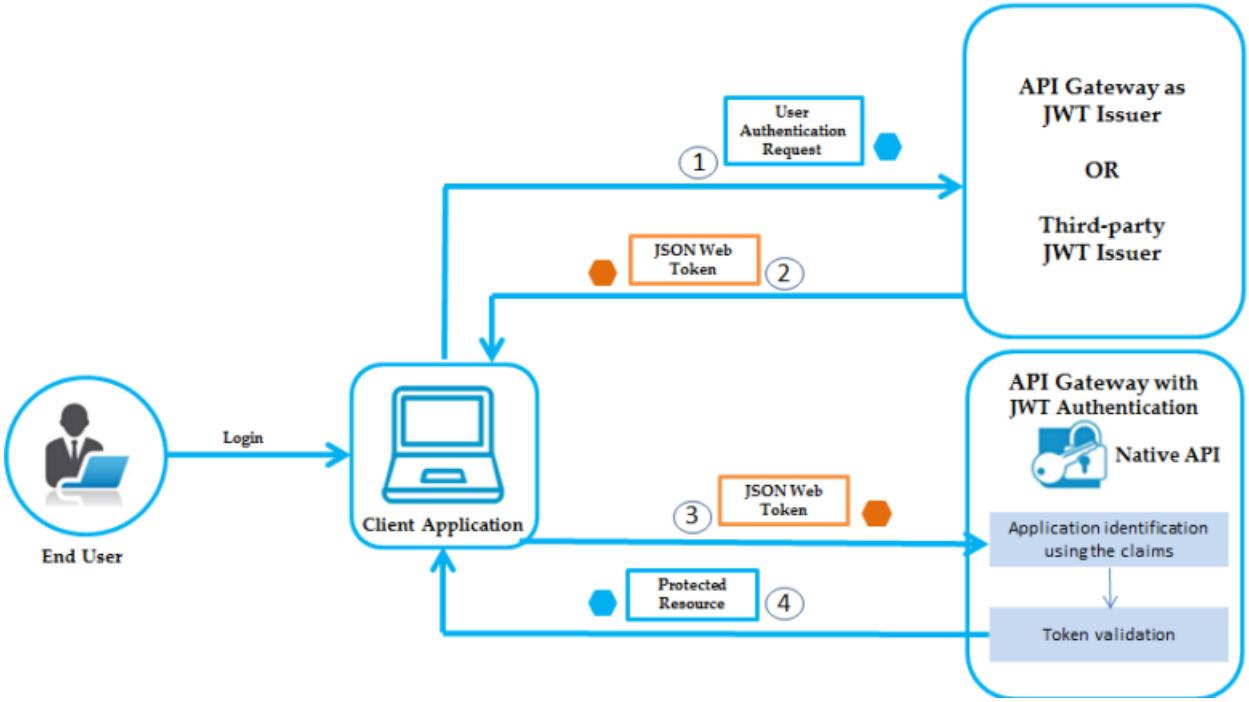
    The bearer token value appears in the **Value** field of the **Header** tab.

Note:
If you are using a REST client like Postman or SoapUI to create an consumer application and invoke a REST API, then you must generate the application authentication using static or dynamic payload, and provide the bearer token value to invoke the API. But, if you are using API Portal to register a consumer application, this process is made simple using the **Get token** feature in the **Try API** section of API Portal.

15. Click **Send**. The response for the selected method appears.

JWT Authorization Workflow

The flow of authorization requests and responses between the end user, client application, JWT issuer, and resource server is as depicted in the following figure.



The JWT authorization workflow is as follows:

1. The end user logs in, the client application sends an authentication request to API Gateway or to any third-party JWT issuer, to obtain a JWT token.
2. If API Gateway is the JWT issuer, then it validates the user or the application. If the user or application credentials are valid, API Gateway generates the JSON token using a private key that was specified in the JWT configuration, and sends the generated token to the client.

   If the user credentials are invalid, API Gateway returns a specific error response.
3. Client sends the generated JSON token in the HTTP Authorization request header as a Bearer token to access the protected API in API Gateway.

4. API Gateway first identifies the application based on claims from the JWT, then validates the JWT using the public certificate of the issuer (the issuer can be API Gateway or a third-party issuer) and provides access to the protected resources.

   If the validation fails, API Gateway returns a specific error response.

   Note:

   - If API Gateway has generated the JSON token, it validates the signature using a public certificate that was specified in the JWT configuration. Else, if the HTTP request is sent from a third-party JWT issuer, API Gateway validates the token using a public certificate or the JWKS URI of the issuer.
   - When a Policy violation event is logged in case of expired Oauth2 tokens, the application that is associated turn in to **Unknown**.

Retrieving JWT Token

You can retrieve JWT using one of the following ways:

- **Retrieve with static payload**: This method is used to retrieve an access token for a general access.
- **Retrieve using an Application Id**: This method is used to retrieve an access token to be used for a particular application.

To retrieve a JWT token

1. Open your internet browser.
2. Perform one of the following steps to retrieve access token:
   - To retrieve the access token with static payload, provide the following URL in the browser, with the IP of API Gateway in place of local host:

     ```
     http://localhost:5555/rest/pub/apigateway/jwt/getJsonWebToken
     ```

   - To retrieve the access token for a particular application, provide the following URL, with the IP of API Gateway and required application Id:

     ```
     http://localhost:5555/rest/pub/apigateway/jwt/getJsonWebToken?
     app_id=applicationId
     ```

     For example,

     ```
     https://localhost:5556/rest/pub/apigateway/jwt/getJsonWebToken?
     app_id=9502c862-9e67-4726-bc13-598df42c7fb6
     ```

   The JWT token is displayed:

The subject claim of the token generated by making a GET call will be the username of user who calls the JWT endpoint.

Note:
You must use HTTPS protocol when retrieving JWT token. If you want to use the HTTP protocol, you must set the **pg_JWT_isHTTPS** setting in the **Administration** > **Extended Settings** to *false*.

Retrieving JWT Token with Claim

When you retrieve a JWT token for a particular application, the application is authenticated using the application identifiers provided in the request, such as, APIKey, Username, or Host name, and then a token is generated with application id as a subject.

For example, consider multiple developers using an application to retrieve an access token. In such a scenario, each user can have a claim that can be used to identify the user who made a particular transaction.

To retrieve a JWT token with claim

1. Open your REST client.

2. Make a POST call to the following URL, with the IP address of the system where API Gateway is installed in place of localhost:

   ```
   http://localhost:5555/gateway/security/getJsonWebToken
   ```

   For example,

   ```
   http://localhost:5555/rest/pub/apigateway/jwt/getJsonWebToken
   ```

3. Provide your claim identifiers in the **Request** section:

   ```
   { "claimsSet": { "identifier": "value"} }
   ```

   For example,

   ```
   { "claimsSet": { "name": "username", "company": "organization" } }
   ```

   Note:

Before invoking this service, ensure that the authorization server is configured and the scope mapping is done.

The access token is displayed in the **Response** section. The subject claim of the token generated by making a POST call will be the ID of the identified application.

OpenID Authentication Use case and Workflow

OpenID Connect is an open standard and decentralized authentication protocol that extends on the OAuth 2.0 authorization framework. It combines the capability of Open ID in verifying the client's identity and OAuth's capability of accessing the client's resources.

In case of OpenID support in API Gateway, you can use the OpenID authentication protocol to identify and authorize a client application to access the protected resources in one of the following ways (these are explained in detail in the Usecase section.):

- Use just the access tokens (that is OAuth token) to invoke the protected resources.
- Use the ID token (that gives information about the user) to invoke the protected resources in one of the following ways:
  - Present the ID token to exchange it for an access token and use the access token to access the protected resources.
  - Use the ID token directly to access the protected resources.

API Gateway does not act as a OpenID Connect server but can validate the tokens issued by other OpenID Connect servers.

The following internal API is used for getting an access token for an ID token.

exchangeIDToken

Method: **POST**

URL: `http://host:port/gateway/security/exchangeIDToken`

Payload

```
{
  "gatewayScopes": ["OktaTenant1:inventory"],Identify & Authorize in the

  "idToken": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjQwYzZiMDliNDQ5NjczNDUzYzNkYTY"
  "expiry": 3000
}
```

For details on scopes, see *webMethods API Gateway Administration*.

Note:
The getOpenIDtoken call is deprecated and is no more available from the API Gateway release 10.3 onwards.

Use case 1: OpenID authentication using OpenID Connect Provider

This describes the high level workflow for using the OpenID authentication protocol to identify and authorize a client application to access the protected resources.

1. Configure a Provider if you are using the Dynamic client registration. Else you can proceed to step 2.

   For a complete procedure on configuring a provider, see *webMethods API Gateway Administration* .

2. Configure an external authorization server.

   Ensure you configure the external authorization server with the introspection URL and OAuth scopes. For a complete procedure on configuring an external authorization server, see *webMethods API Gateway Administration*.

3. Map the scopes.

   For a complete procedure on mapping scopes, see *webMethods API Gateway Administration*.

4. Enforce the Identify & Authorize policy on the API.

   Ensure to select OpenID Connect or JWT as options. For more details, see “Identify & Authorize” on page 111.

5. Associate an application with the API.

   You can create a new application or use an existing one. Ensure that the application associated contains the strategy for OpenID authentication. While creating a strategy you can associate it with the scopes that are available to be used while using dynamic client registration. For a complete procedure on creating an application with a strategy, see “Creating an Application” on page 80.

6. Activate the API.

   User on invoking the API uses the access token or the ID token provided by the provider to access the protected resource.

7. User can access the protected resources in one of the following ways:

   - The user presents the access token to API Gateway and on validation accesses the protected resource.
   - The user presents the ID token to API Gateway to exchange it for an access token (if the user has configured the OpenID Connect option in step 4). The client then presents the access token to API Gateway and on validation accesses the protected resource.

     The following internal API is used for getting an access token for an ID token.

     exchangeIDToken

     Method: **POST**

URL: `http://host:port/gateway/security/exchangeIDToken`

Payload

```
{
  "gatewayScopes": ["OktaTenant1:inventory"],
  "idToken": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjQwYzZiMDliNDQ5NjczNDUzYzNkYTY"
  "expiry": 3000
}
```

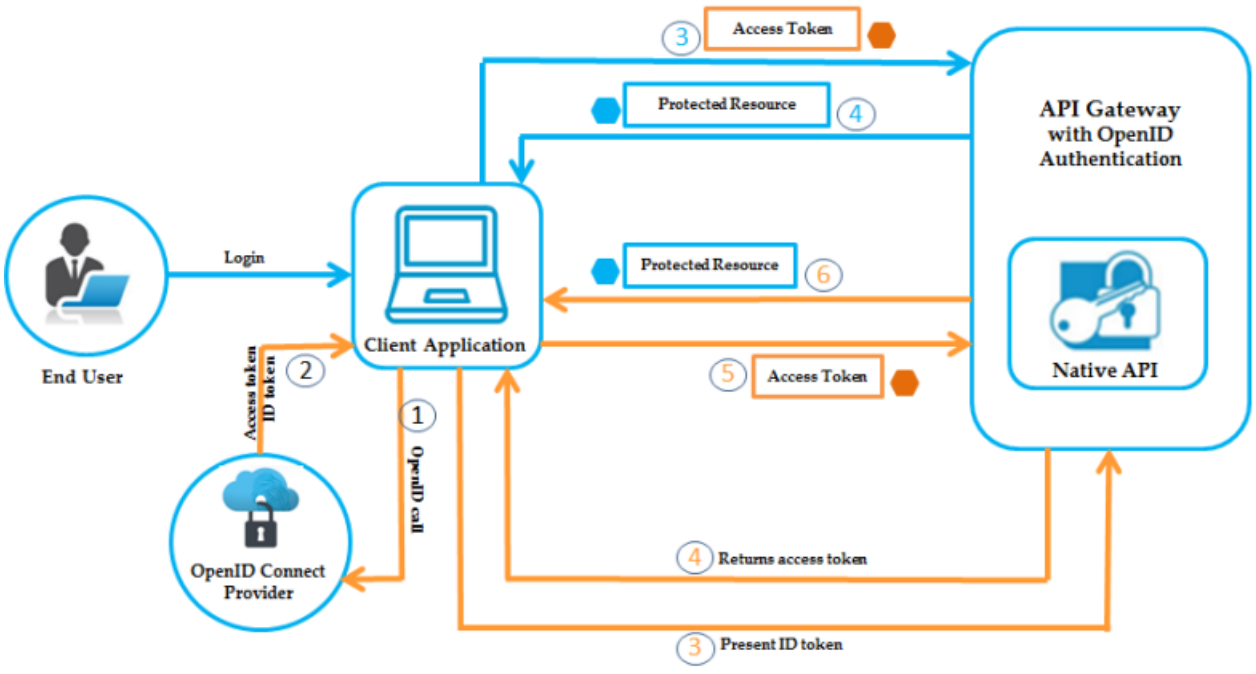
- The user presents the ID token as a JWT directly to API Gateway (if the user has configured the JWT option in step 4), and on validation accesses the protected resource.

OpenID Authorization Worklow

The OpenID Connect support in API Gateway provides two different ways for a client to access a protected resource depending on whether the provider has provided an access token or an ID token. The workflow diagram depicts both these cases. The first 2 steps are same in both the cases, the arrows in blue depict the flow where an access token is used to access the protected resource, and the arrows in orange depict the flow where an ID token is used to access the protected resource.

OpenID authorization workflow using the OpenID Connect Provider

The flow of authorization requests and responses between the end user, client application, OpenID Connect provider, and resource server is as depicted in the following figure. The client application makes an OpenID call to the OpenID Connect provider and receives an access token or an ID token in the response. It uses these tokens to access the protected resources.



OpenID authorization workflow using the access token provided by the Open ID Connect Provider

1. The client makes an OpenID call to the OpenID connect Provider.

2. The OpenID Connect Provider provides an access token to the client.
3. The client application presents the access token received from the OpenID Connect Provider to send HTTP requests to API Gateway.
4. API Gateway then performs the following:
   a. Identifies the application using the clientId.
   b. Validates the token locally or remotely if it is not possible locally.
   c. Checks if the requested resource is part of the scopes in the token.
   d. Checks the audience.

   API Gateway provides access to the protected resource if all the validations are done. If the access token is valid, API Gateway provides access to the protected resource. If the access token is expired, authorization server returns a specific error response. The client application can then use Refresh Token to request a new access token. The Authorization Server returns a new access token that can be used to access the protected resource.

OpenID authorization workflow using the ID token provided by the Open ID Connect Provider

1. The client makes an OpenID call to the OpenID Connect Provider.
2. The OpenID Connect Provider provides an ID token to the client.
3. The client application presents the ID token received from the OpenID Connect Provider to API Gateway.
4. API Gateway validates the ID token and returns an access token to the client application.

   The following internal API is used for getting an access token for an ID token.

   exchangeIDToken

   Method: **POST**

   URL: `http://host:port/gateway/security/exchangeIDToken`

   Payload

```
{
  "gatewayScopes": ["OktaTenant1:inventory"],
  "idToken": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjQwYzZiMDliNDQ5NjczNDUzYzNkYTY"
  "expiry": 3000
}
```

   For details on mapping scopes, see *webMethods API Gateway Administration*.
5. The client then uses this access token to send HTTP requests to API Gateway.
6. API Gateway then performs the following:
   a. Identifies the application using the clientId.
   b. Validates the token locally or remotely if it is not possible locally.

c. Checks if the requested resource is part of the scopes in the token.

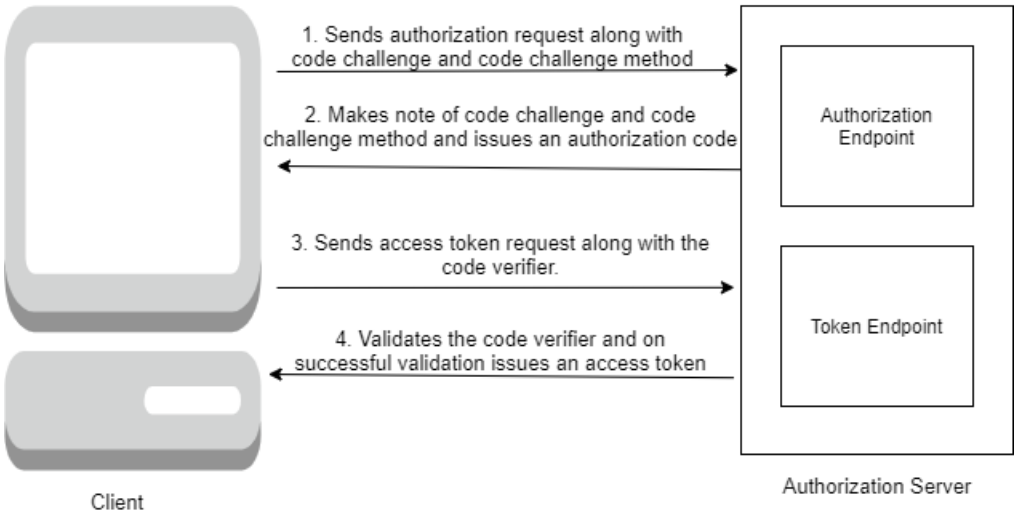
d. Checks the audience.

API Gateway provides access to the protected resource if all the validations are done. If the access token is valid, API Gateway provides access to the protected resource. If the access token is expired, authorization server returns a specific error response. The client application can then use Refresh Token to request a new access token. The Authorization Server returns a new access token that can be used to access the protected resource.

Note:
The user can present the ID token directly as a JWT to access the protected resources in case the ID token is provided on configuring the JWT property in the Identify & Authorize policy enforced on the API.

Securing Access Token Calls with PKCE

PKCE (Proof Key for Code Exchange) is a mechanism that prevents **Authorization Code Interception** attacks and makes OAuth 2.0 authorization code grant more secure for public clients. The client application should give proof to the authorization server that the authorization code belongs to the client application. Only then the authorization server issues an access token for the client application.

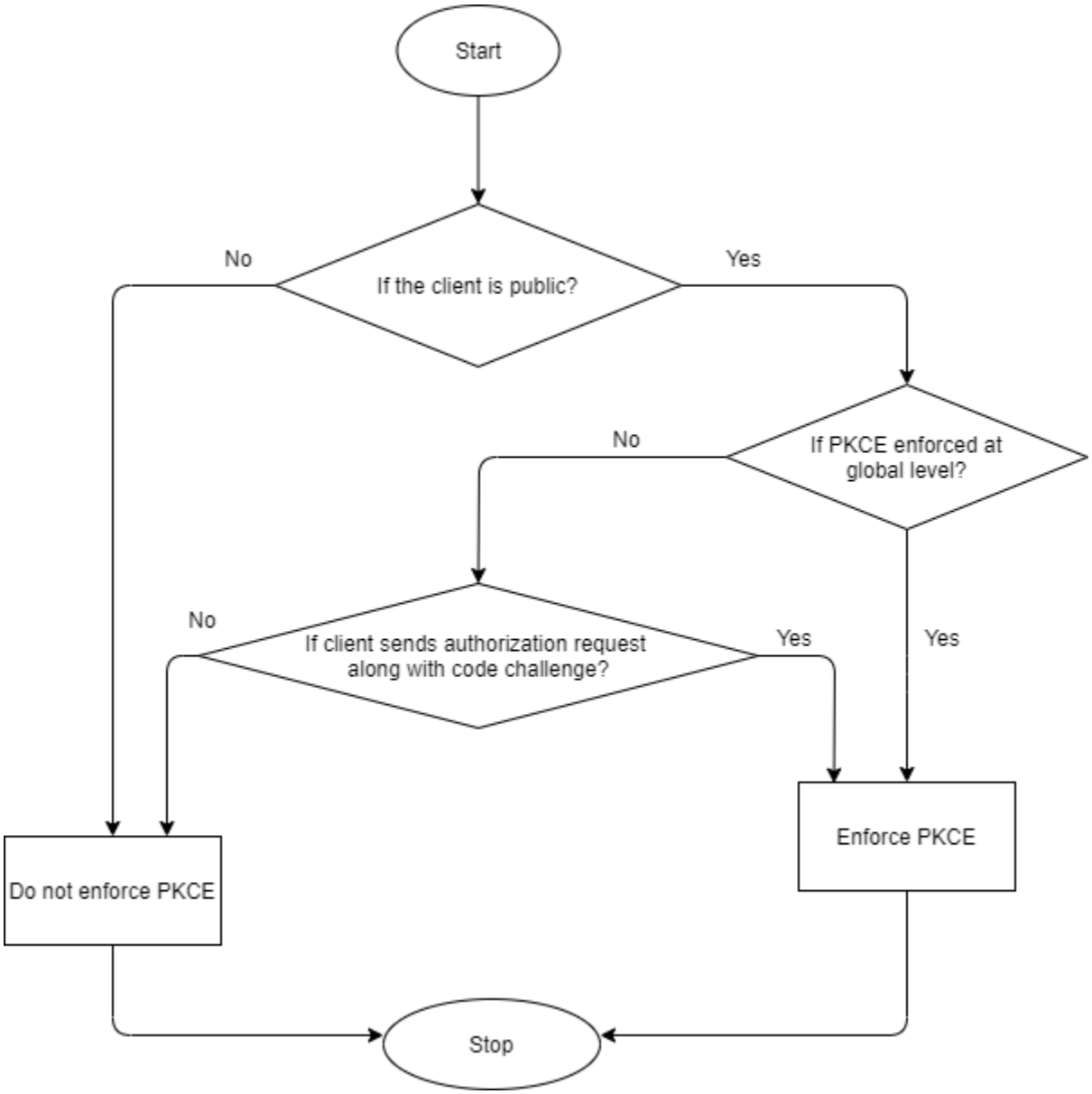


The PKCE flow works with these parameters:

- **Code Verifier**. The code verifier should be a high-entropy cryptographic random string with a minimum of 43 characters and a maximum of 128 characters.
- **Code Challenge**. The code challenge is created by SHA256 hashing the code verifier and then applying base64 URL encoding of the resulting hash. If the client cannot do the hashing and encoding transformation, it can use the code challenge method as *plain* where the code challenge is same as code verifier
- **Code Challenge Method**. This is an optional parameter. If the client uses SHA256 hashing the code challenge method value must be *S256*. If no hashing is done, then the code challenge

method value must be *plain*. If the code challenge method value is not passed in the client request then plain would be considered as default value.

The flow chart explains the Get Access Token workflow in API Gateway using authorization code grant type with PKCE.



How do I enforce PKCE globally?

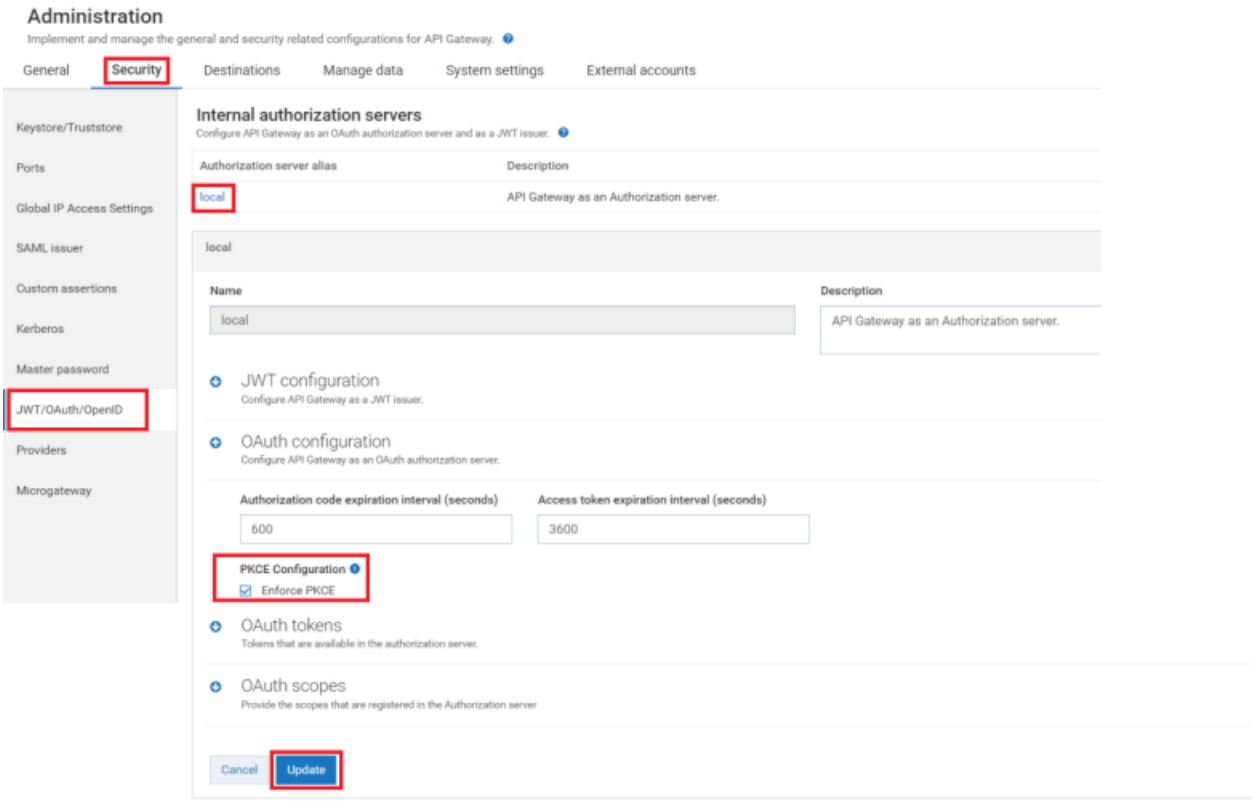
This use case explains how to enforce PKCE globally in the local authorization server. When you enforce PKCE at global level, then it is applied for all the public OAuth2.0 clients of local authorization server.

This use case starts when you want to enable the PKCE workflow and ends when you get the access token on successful validation.

To enforce the PKCE at global level

1. Expand the menu icon, in the title bar, and select **Administration**.
2. Select **Security** > **JWT/OAuth/OpenID**.

   The Authorization servers section displays a list of available internal and external authorization servers.



3. In the **Internal authorization servers** section, click `local`.
4. Expand the **OAuth configuration** section, select the **Enforce PKCE** checkbox.
5. Click the **Update** button.

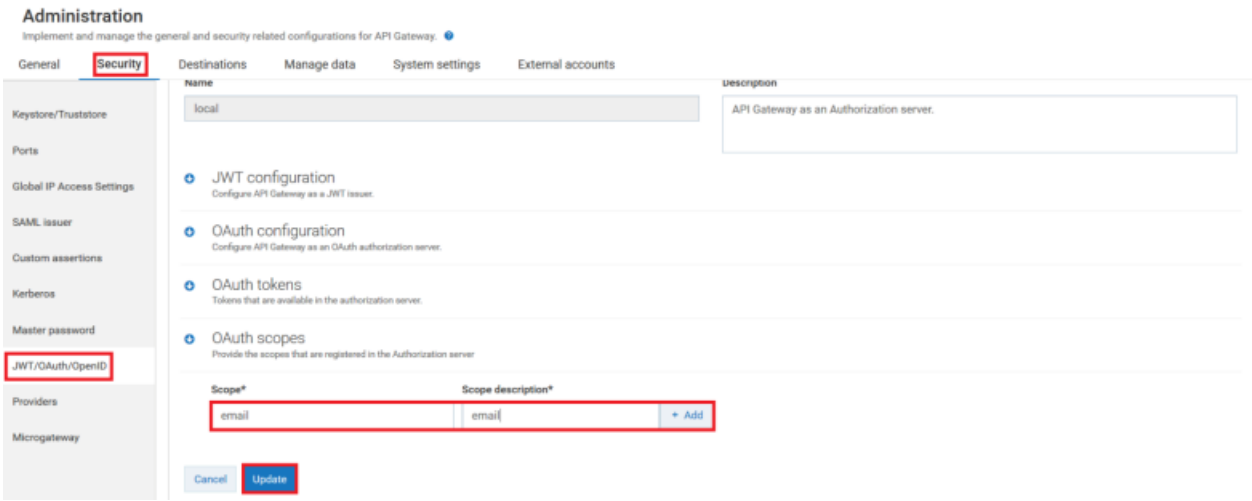
   Once you enforce PKCE, you get access token only on successful validation of code verifier.

How do I secure the access token with Authorization Code (With PKCE) grant type using postman?

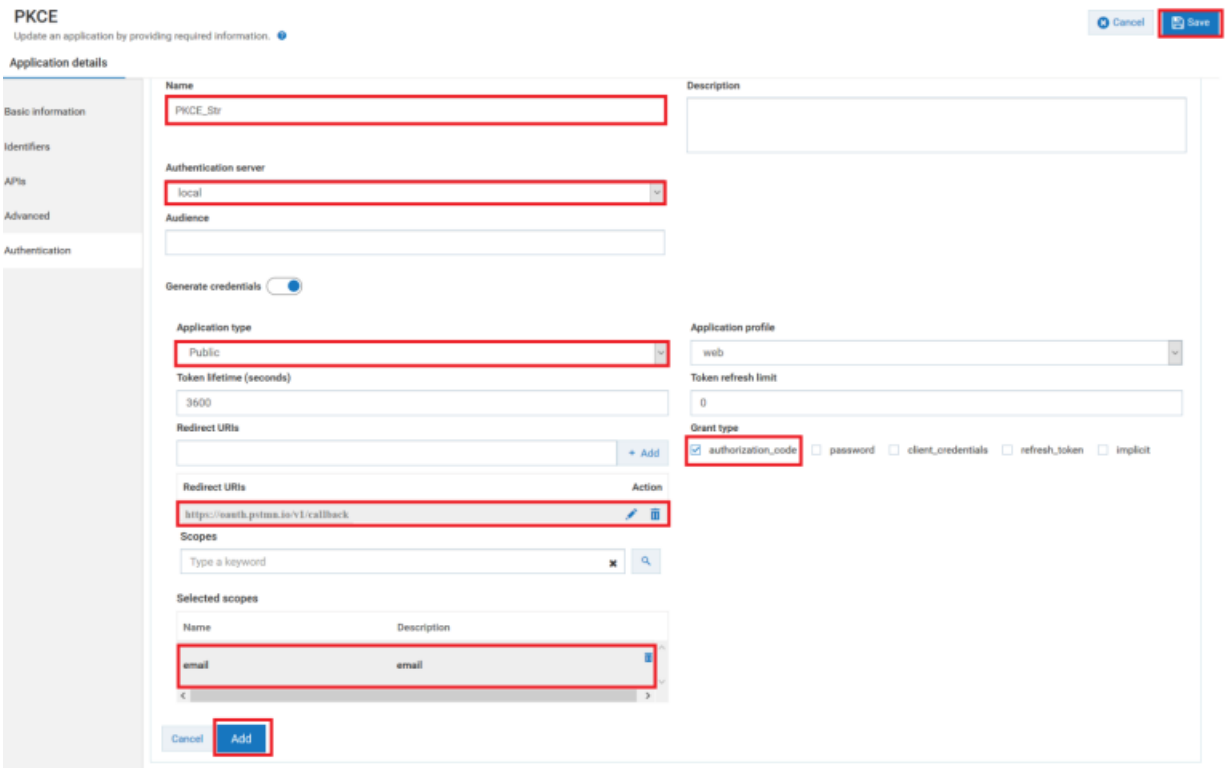
This use case starts when you enforce the PKCE and ends when you get access the token securely using postman.

To secure the access token

1. Create OAuth scope in the local authorization server.



2. Create an application with OAuth2 authentication strategy. For details about creating an application, see "Creating an Application" on page 80.



   a. Click the **Authentication** tab to create a strategy with OAuth2 authentication.

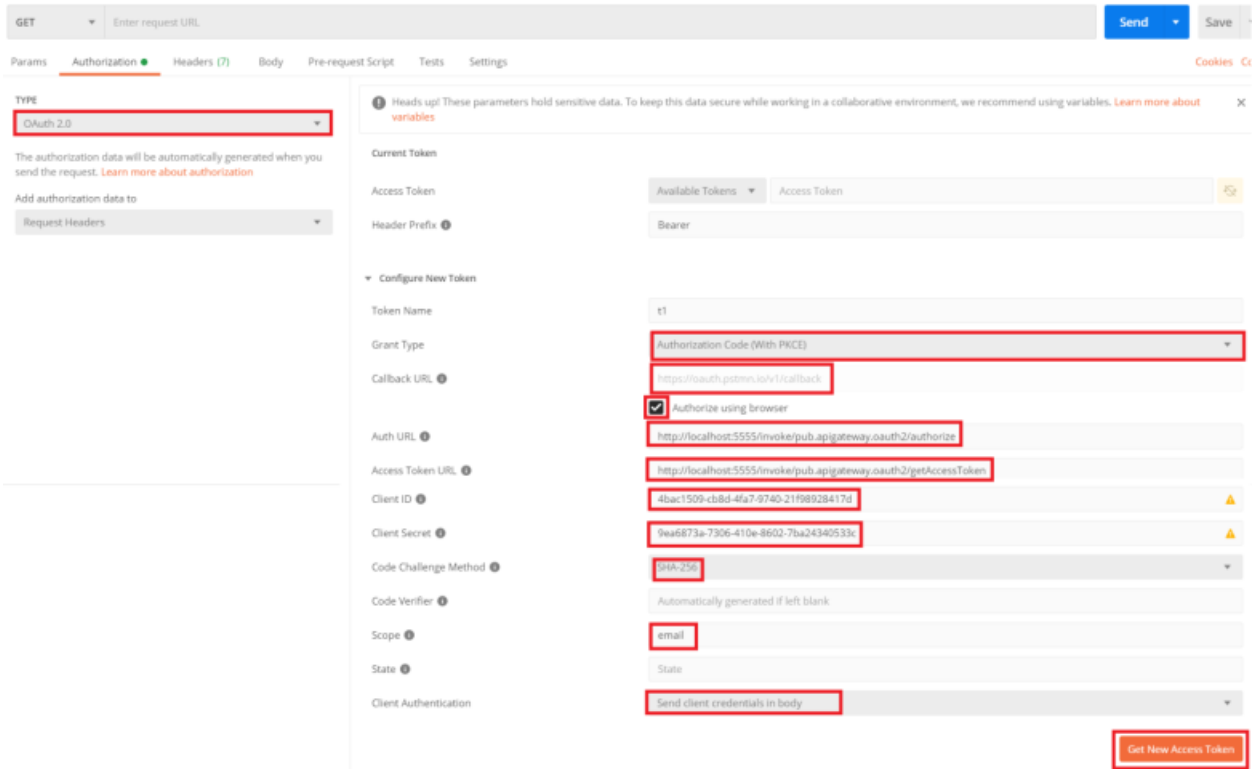
      Make sure you have selected the following mandatory fields for this use case:

- Select the **Authentication schemes** as `OAUTH2`.
- Specify the **Authentication server** as `local`.
- Select the **Application Type** as `Public`.
- Specify the grant type to be used to generate the credentials. For this specific use case, you must select authorization_code, which is dynamically populated from the authorization server.
- Specify the postman `https://oauth.pstmn.io/v1/callback` URL as redirect URI.
- Specify the OAuth scope that you have created for the local authorization server in Step 1.

b. Click **Add** to save the strategy.

c. Click **Save** to save the application.

3. In the Postman, under the **Authorization** tab, select the authorization type as `OAuth2.0` from the **TYPE** drop-down menu.



a. In the **Configure New Token** section, select the grant type as `Authorization Code (With PKCE)`.

b. Type the redirect URL as `https://oauth.pstmn.io/v1/callback` in the Callback URL text box.

c. Select the **Authorize using browser** check box.

d. Type the authorization URL as `http(s)://hostname:port/invoke/pub.apigateway.oauth2/authorize` in the **Auth URL** text box.

e. Type the `http(s)://hostname:port/invoke/pub.apigateway.oauth2/getAccessToken` in the **Access Token URL** text box.

f. Type the client ID and client secret in the **Client ID** and **Client Secret** text boxes respectively.

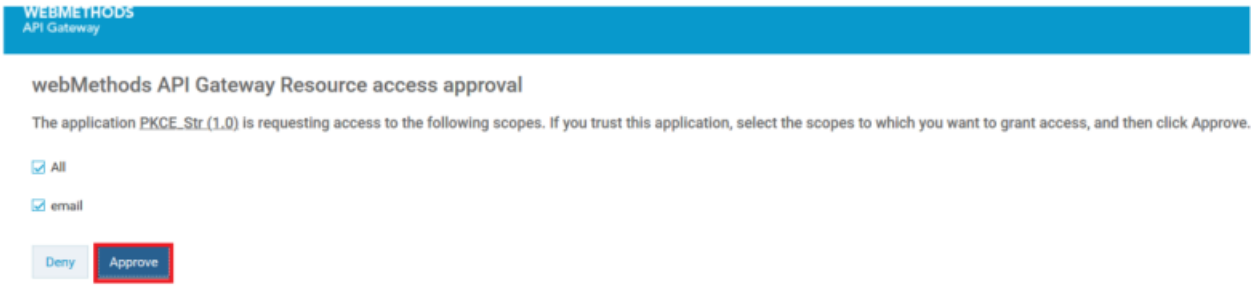
> **Note:**
> You can get the client ID and client secret from the **Authentication** tab of the **Application** screen.

g. Select the hashing method used to generate the code challenge from the **Code Challenge Method** drop down menu.

h. Specify the OAuth scope that you have created for the local authorization server in Step 1 in the **Scope** text box.

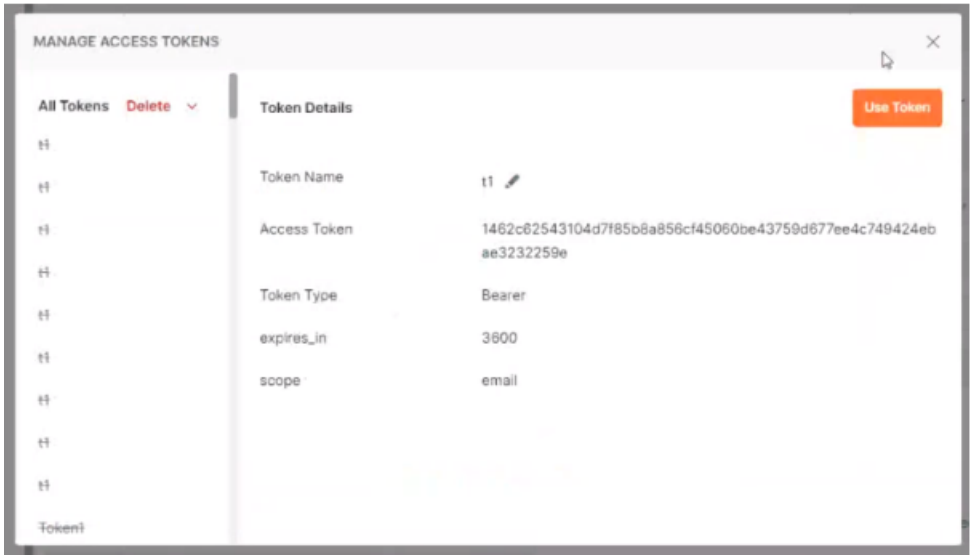
i. Select the client authentication as `Send client credentials in body`.

j. Click the **Get New Access Token** button.



k. Click the **Approve** button.

The **MANAGE ACCESS TOKENS** pop-up window displays the access token.



How do I secure the access token by directly calling API Gateway's REST APIs?

This use case starts when you enforce the PKCE and ends when you get access the token securely using REST APIs.

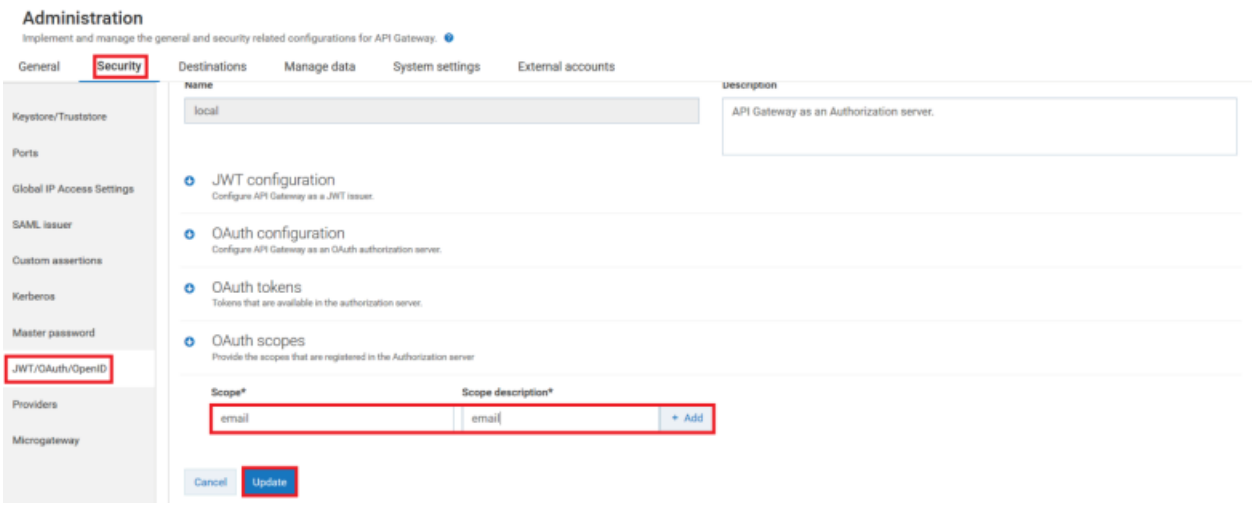
Before you begin

Ensure that you have:

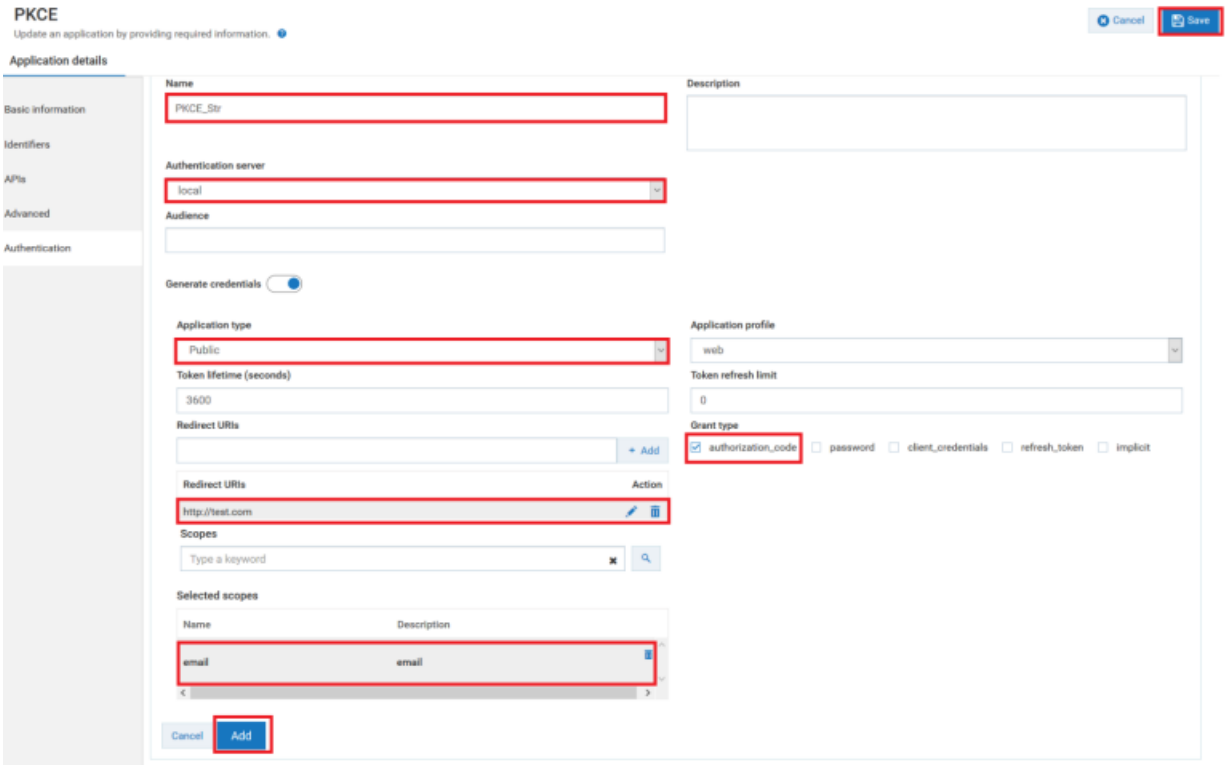
- generated Code Challenge and Code Verifier using the JAR file. For details about how to generate code challenge and code verifier, see " How do I generate code verifier and code challenge using JAR files? " on page 229.
- enforced PKCE at global level.

To secure the access token

1. Create OAuth scope in the local authorization server.



2. Create an application with OAuth2 authentication strategy. For details about creating an application, see "Creating an Application" on page 80.



a. Click the **Authentication** tab to create a strategy with OAuth2 authentication.

   Make sure you have selected the following mandatory fields for this use case:

   - Select the **Authentication schemes** as `OAUTH2`.
   - Specify the **Authentication server** as `local`.
   - Select the **Application Type** as `Public`.

- Specify the grant type to be used to generate the credentials. For this specific use case, you must select authorization_code, which are dynamically populated from the authorization server.
- Specify the URIs that the authorization server can use to redirect the resource owner's browser during the grant process. You can add multiple URIs by clicking +Add.
- Specify the OAuth scope that you have created for the local authorization server in Step 1.

b. Click **Add** to save the strategy.

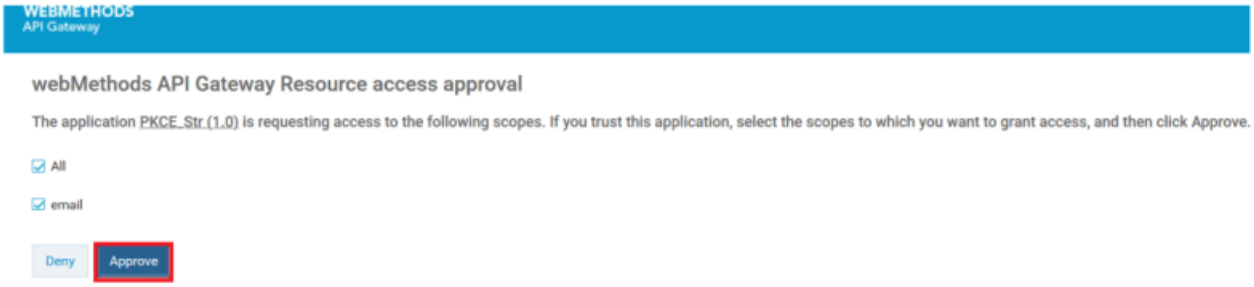
c. Click **Save** to save the application.

3. Get authorization code.

a. Call the authorize endpoint using a REST client.

```
http(s)://hostname:port/invoke/pub.apigateway.oauth2/authorize?
response_type=code&redirect_uri=<redirectURI>&client_id=<Client
ID>&code_challenge=<Code_Challenge>&code_challenge_method=S256
```

Note:
Make sure you have replaced the *redirectURI , ClientID, and Code_Challenge* in the above mentioned URL. You can get the *redirect URI* and *client ID* from the **Authentication** tab of the **Application** screen.



b. Click the **Approve** button.

c. Provide the credentials of API Gateway user to approve the request.

You are re-directed to the redirect URI as per the configuration in the application strategy. The screenshot below is just a sample, you are redirected to a different URL based on your configuration, so the screenshot varies accordingly. If the given redirect URI is not a valid web page, you might get a **Page not found** error, which is fine, because we get the authorization code value from the browser URL.

Authorization code : 0025abe9f96d4901b61340344c29a576

d. Make a note of the authorization code.

> **Note:**
> If the redirect URL screen is not able to display the authorization code, then you can take it from the address bar of the browser. As highlighted in the above image's URL, you can see the authorization code in the code=field of the URL.

e. Click **Save** to save the application.

4. Get access token.

a. Invoke the access token endpoint using a REST client.

Request: POST `http(s)://hostname:port/invoke/pub.apigateway.oauth2/getAccessToken`.

In the **Authorization** tab, select the authorization type as `Basic Auth` . Provide the client ID as username and client secret as the password. You can get the client ID and client secret in the **Authentication** tab of the **Application** screen.

Sample request body

```
{
                "redirect_uri":"http://test.com",
                "scope":"email",
                "grant_type":"authorization_code",
                "code":"0025abe9f96d4901b61340344c29a576",

"code_verifier":"a4793f15479a4c5697f93b44d055ab6cbd16be50400a4591892f914b1a256da8",
                "client_id":"374b1fae-4405-411b-85a0-6e1ab90923ba"
}
```

> **Note:**
> You must replace the redirect_URI, scope, code, and code_verifier with appropriate values. For the code field, make sure you use the authorization code you noted down in the step 3.d.

Sample response body

```
{
    "access_token":
"b5b33bc9c57945f388010f8caf5fe9b6b14abef468d346e68e0cd374c0df60d7",
    "token_type": "Bearer",
    "expires_in": 3600
```

```
}
```

How do I enforce PKCE selectively for each access token call?

You can enforce PKCE specific to each GET access token call. To perform this use case, you must clear the **Enforce PKCE** check box in the **Administration > Security > JWT/OAuth/OpenID** screen. When you disable the PKCE global option, by default PKCE is not verified. But if you send the authorize request with the code challenge and code challenge method parameters, you get an access token with PKCE verification even though you have not enforced PKCE.

How do I generate code verifier and code challenge using JAR files?

If you want to secure the access token by directly calling REST APIs in API Gateway, you have to generate the code verifier and code challenger using JAR files.

Before you begin

Ensure that you have JShell, which is available as part of JDK from JDK9.

To generate code verifier and code challenge

1. Invoke the JShell file in the *Install_Dir*\common\lib directory with class path set to wm-isclient.jar using the below command:

   ```
   C:\> jshell -c c:\ Install_Dir\common\lib\wm-isclient.jar
   ```

2. Import the PKCE class file using the following command:

   ```
   jshell> import com.softwareag.util.PKCE;
   ```

3. Create code verifier using the following command:

   ```
   jshell> PKCE.createCodeVerifier();
   ```

   The code verifier is generated as follows:

   ```
   $2==>"95b4efde52b141d1bde8a7bfc23bdb244728fdd70d4a4be5b110866cfc218db7"
   ```

4. Create code challenger using the following command:

   ```
   jshell> PKCE.createCodeChallenge("code_verifier","S256");
   ```

   Note:
   Replace the *code_verifier* parameter with the code verifier string that you generated in the previous step.

   The code challenge is generated using SHA 256 hashing method as follows:

   ```
   $3==>"tMTWyt3W5QtaPIqNkqAHLTGZnN0aPopp2fsLrUFdAC0"
   ```

Inbound Auth - Message

An API Provider can use this policy to enforce authentication on the API. When this policy is configured for an API, API Gateway expects the clients to pass the authentication credentials through the payload message that will be added to the request and sent to the native API. API Gateway supports a wide range of authentication schemes, such as X.509 Certificate, WSS Username, SAML, and Kerberos, in addition to signing and encryption, at the message-level.

Note:
Message-level authentication can be used to secure inbound communication of only SOAP APIs.

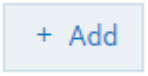
The table lists the properties that you can specify for this policy:

Property	Description
Binding Assertion	Specifies the type of binding assertion required for the message transfer between the recipient and the initiator.
Require Encryption. Specifies that a request's XML element, which is represented by an XPath expression or by parts of a SOAP request such as the SOAP body or the SOAP headers, be encrypted.	
Encrypted Parts	Click **+ Add encrypted part** to add the required properties. This allows you to encrypt parts of a SOAP request such as the SOAP body or the SOAP headers. Provide the following information: ■ **Entire SOAP Body**. Specifies encryption of the entire SOAP body. ■ **All SOAP Attachments**. Specifies encryption of all the SOAP attachments. In the SOAP Header section, provide the following information: ■ **Header Name**. Specifies the name for the SOAP header field. ■ **Header Namespace**. Specifies the namespace of the SOAP header to be encrypted. You can add more SOAP headers by clicking  .
Encrypted Elements	Click **+ Add encrypted element** to add the required properties. Select this option to encrypt the entire element, which is represented by an XPath expression. Provide the following information: **XPath**. Specifies the XPath expression in the API request.

Property	Description
	In the Namespace section, provide the following information: ■ **Namespace Prefix**. Specifies the namespace prefix of the element to be encrypted. ■ **Namespace URI**. Specifies the generated XPath element. You can add more namespace prefixes and URIs by clicking + Add.
Require Signature. Specifies that a request's XML element, which is represented by an XPath expression or by parts of a SOAP request such as the SOAP body or the SOAP headers, be signed.	
Signed Elements	Click **+ Add require signature** to add the required properties. Select this option to sign the entire element represented by an XPath expression. Provide the following information: **XPath**. Specifies the XPath expression in the API request. For the Namespace section, provide the following information: ■ **Namespace Prefix**. Specifies the namespace prefix of the element to be signed. ■ **Namespace URI**. Specifies the generated XPath element. You can add more namespace prefixes and URIs by clicking + Add.
Signed Parts	Click **+ Add signed part** to add the required properties. Select this option to sign parts of a SOAP request such as the SOAP body or the SOAP headers. Provide the following information: ■ **Entire SOAP Body**. Specifies signing of the entire SOAP body. ■ **All SOAP Attachments**. Specifies signing of all the SOAP attachments. For the SOAP Header section, provide the following information: ■ **Header Name**. Specifies the name for the SOAP header field. ■ **Header Namespace**. Specifies the Namespace of the SOAP header to be signed. You can add more namespace prefixes and URIs by clicking + Add.

Property	Description
Validate SAML Audience URIs. Validates the audience restriction in the conditions section of the SAML assertion. It verifies whether any of the valid audience URI within a valid condition element in SAML assertion matches with any of the configured URI. If two conditions are available, then one of the audience URIs in the first condition, and one of the audience URIs in the second condition must match with any of the configured URIs in this policy for the SOAP API. This property is used in the following scenarios: ■ When the native API is enforced with the SAML policy, and the service provider wants to delegate audience restriction validation to API Gateway. ■ When **Require SAML Token** assertion is defined for the SOAP API in API Gateway.	
URI	Specifies the SAML audience URI.
Match Criteria	Select one of the following options: ■ **Allow Sublevels**. Any one of the audience URI in the incoming SAML assertion either has to be an exact match or it can have sub paths to the configured URI. For example, if `http://yahoo.com` is configured as the **URI** and the **Allow Sublevels** option is selected, the audience URI has `http://yahoo.com/mygroup` and condition is matched because the main URI matches with the configured URI (`http://yahoo.com`). The extra path `mygroup` is a sublevel path. ■ **Exact match**. Any one of the audience URI in the incoming SAML assertion is verified for the exact match with the configured URI. For example, if `http://yahoo.com` is configured as the **URI** and the **Exact match** option is selected, the audience URI must be configured with `http://yahoo.com` in order to match the condition. This is selected by default. For more information on audience URI, see conditions and audience restriction sections in the SAML specification available in the https://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf location.
Token Assertions	Select the type of token assertion required to authenticate the client. Select any of the following: ■ **Require X.509 Certificate**. Mandates that there should be a wss x.509 token in the incoming SOAP request. ■ **Require WSS Username token**. Mandates that there should be a WSS username token in the incoming SOAP request. Uses WS-Security authentication to validate user names and passwords that are transmitted in the SOAP message header for the WSS Username token.

Property	Description
	■ **Kerberos Token Authentication**. Mandates that there should be a Kerberos token in the incoming SOAP request. Authenticates the client based on the Kerberos token. API Gateway extracts the Kerberos token from the SOAP body and validates the token with the KDC using SPN credentials configured by the provider for the API. If the Kerberos token sent by the client is valid, API Gateway forwards the request to the native service and the response to the client. ■ **Service Principal Name**. Specifies a valid SPN, which is the name type to use while authenticating an incoming client principal name. The specified value is used by the client or the server to obtain a service ticket from the KDC server. **Note:** API Gateway supports the *username* format for Service Principal Names (SPNs). This format represents the principal name as a named user defined in LDAP used for authentication to the KDC. ■ **Service Principal Password**. Specifies a valid password of the Service Principal Name user or the Service Principal Name host. ■ **Require SAML Token**. Mandates that there should be a SAML token in the incoming SOAP request. Uses a Security Assertion Markup Language (SAML) assertion token to validate applications. Provide the following information: ■ **SAML Version**. Specifies the supported SAML version. Available values are **SAML 1.0**, **SAML 2.0** ■ **SAML Subject Configuration**. Select one of the following: ■ **Bearer of Token**. Select the bearer method when the client wants a security token to be issued without a proof of possession. ■ **Holder of Key - Symmetric**. Select the Holder of Key (Symmetric) method when either the client or the server has to generate security tokens such as X509 tokens. A symmetric key is established using the security token. You can use this token to sign and encrypt parts and elements. ■ **Holder of Key - Public**. Select the Holder of Key (Public) method when both the client and the server have security token such as X509 certificates. In this method, the client

Property	Description
	uses its private key to sign and the recipient's (API Gateway) public key to encrypt. ■ **WS-Trust Version**. Specifies the WS-Trust version to be used. Available values are **WS-Trust 1.0**, **WS-Trust 1.3** ■ **Encrypt Signature**. Select **Yes** to encrypt the signature. ■ **Issuer Address**. Specifies the SAML issuer address. ■ **Metadata Reference Address**. Specifies the address from where the metadata reference document is obtained. ■ **Algorithm Suite**. Specifies the applicable algorithm suite. ■ **Key**. Specifies the Key type of the security token template. ■ **Value**. Specifies a value for the request token. You can add more values for the key-value pair by clicking  . ■ **Custom Token Assertion**. Type a search string, select a custom token assertion name to authenticate the client, and click  to add. You can add more custom token assertions in a similar way. Click the **Custom Token Assertion** arrow to see a list of all custom token assertions available in API Gateway. Click  to delete the custom token assertion added.
Require Timestamp	Specifies that the time stamps be included in the request header. API Gateway checks the time stamp value against the current time to ensure that the request is not an old message. This serves to protect your system against attempts at message tampering, such as replay attacks.

Authorize User

This policy authorizes incoming requests against a list of users, a list of groups, or users who belong to LDAP groups registered in API Gateway.

Note:
LDAP groups cannot be authorized using the List of Groups configuration option. To authorize a user who belongs to an LDAP group, you must first create a team containing one or more

LDAP groups and then authorize the user using List of Teams configuration option in this policy.

Use this policy in conjunction with an authentication policy (for example, Require HTTP Basic Authentication, Require WSS Username Token).

The table lists the parameters of this policy and how they are applied to authorize the incoming requests.

Property	Description
List of Users	Authorizes applications against a list of users registered in API Gateway. Type a search string, select a user, and click + Add to add. You can add one or more users. Click to delete the user added.
List of Groups	Authorizes applications against a list of groups registered in API Gateway. Type a search string, select a group, and click + Add to add. You can add one or more groups. Click to delete the group added.
List of Teams	Authorizes applications against a list of teams registered in API Gateway. Type a search string, select a team, and click + Add to add. You can add one or more teams. Click to delete a team.

Request Processing

These policies are used to specify how the request message from an application has to be transformed or pre-processed and configure the masking criteria for the data to be masked before it is submitted to the native API. This is required to protect the data and accommodate differences between the message content that an application is capable of submitting and the message content that a native API expects. The policies included in this stage are:

- Invoke webMethods IS
- Request Transformation

- Validate API Specification
- Data Masking
- Custom Extension

  Custom Extension policies allow you to handle requirements that might not be provided by the out-of-the-box policies. You can add these custom extensions into API Gateway policy stages. To learn more about Custom Extension, see “Custom Policy Extension” on page 383.

Validate API Specification

This policy validates the incoming request against API's various specifications such as schema, query parameters, path parameters, cookie parameters, content-types, and HTTP Headers referenced in their corresponding formats as follows:

- The schema is available as part of the API definition. The schema for SOAP API are imported through WSDL and for REST APIs it can be through swagger, RAML or can be uploaded by the user when an API is created from scratch.
- The query parameters, path parameters, cookie parameters, and content- types are available as part of the API definition.
- The HTTP Headers are specified in the Validate API Specification policy page.

The request sent to the API by an application must conform with the structure or format expected by the API. The incoming requests are validated against the API specifications in this policy to conform to the structure or format expected by the API.

Various API specifications validated are:

- **Schema:**

  Schema validation for REST API and SOAP API:

  The incoming requests are validated against the schema provided in the API definition. The schema defines the elements and attributes and specifies the data types of these elements to ensure that only appropriate data is allowed through to the API. API Gateway does not validate the payload, if the payload is sent as a stream.

  For a REST API, the schema can be added inline or uploaded in the Components section on the API details page. For details on how to add the schema inline or upload, see “Creating a REST API from Scratch” on page 20.

  The schema type for validation is selected based on:

  - The Content-Type header when the policy is added in the Request processing stage.
  - The Accept header when the policy is added in the Response processing stage.

  If the header or payload is missing the schema validation is skipped.

  The table lists the default Content type/Accept header and schema validation type mapping.

Content-type/Accept	Schema validation type
application/json	JSON schema
application/json/badgerfish	
application/xml	XML schema
text/xml	
text/html	
text/plain	Regular expression

For a SOAP API, the WSDL and the referenced schema must be provided in a zip format. The JSON schema validation is supported for the operations that are exposed as REST.

Note:
If schema mapping is not found for a content-type of the request in the API, the behavior is as follows:

- If schema mapping is not available in a REST API or SOAP-to-REST transformed API, the validation is skipped.
- If application/json is mapped to XML schema in the API definition, then the JSON content in the request is validated against XML schema to provide a backward compatibility support for APIs migrated from the 10.1 version.
- If only XML schema mappings exist for any of the content-types, the payload is converted into XML and validated against all the XML schemas. If the payload is valid against one of the schemas, the validation is successful.
- If the payload is not XML convertible, the validation is not performed and the request is allowed to reach the native API.

Schema validation for GraphQL API:

The incoming query or mutation payloads are validated against the GraphQL schema type system.

- **Query Parameters:**

  This is applicable only to a REST API. The incoming requests are validated against the query parameters specified in the API definition.

- **Path Parameters:**

  This is applicable only to a REST API. The incoming requests are validated against the path parameters specified in the API definition.

- **Content-types:**

  This is not applicable to a GraphQL API. The incoming requests are validated against the content-types specified in the API definition.

Note:

When Content-type validation is selected for a SOAP API, the validation fails in case of SOAP to REST scenarios and displays an error with 500 status code instead of 400 as displayed in the other scenarios.

- **Cookie Parameters:**

  This is not applicable to a GraphQL API. The incoming requests are validated against the cookie parameters specified in the API definition.

- **HTTP Headers:**

  This is not applicable to a GraphQL API. The incoming requests are validated against the HTTP Headers specified in this policy to conform to the HTTP headers expected by the API. If the HTTP Headers are not specified in this policy, API Gateway uses the Headers defined in the API specification.

The runtime invocations that fail the specification validation are considered as policy violations. You can view such policy violation events in the dashboard.

The table lists the API specification properties, you can specify for this policy, to be validated:

Property	Description
Schema	Validates the request payload against the appropriate schema. Provide the following additional features for XML schema validation: *This is not applicable to a GraphQL API.* ■ **Feature name**. Specifies the name of the feature for XML parsing when performing XML schema validation. Select the required feature names from the list: ■ `GENERATE_SYNTHETIC_ANNOTATIONS` ■ `ID_IDREF_CHECKING` ■ `IDENTITY_CONSTRAINT_CHECKING` ■ `IGNORE_XSL_TYPE` ■ `NAMESPACE_GROWTH` ■ `NORMALIZE_DATA` ■ `ROOT_ELEMENT_DECL` ■ `ROOT_TYPE_DEF` ■ `SIGMA_AUGMENT_PSVI` ■ `SCHEMA_DV_FACTORY` ■ `SCHEMA_ELEMENT_DEFAULT`

Property	Description
	■ SCHEMA_LOCATION ■ SCHEMA_NONS_LOCATION ■ SCHEMA_VALIDATOR ■ TOLERATE_DUPLICATES ■ ENPARSED_ENTITY_CHECKING ■ VALIDATE_ANNOTATIONS ■ XML_SCHEMA_FULL_CHECKING ■ XMLSCHEMA_VALIDATION For details about XML parsing features, see http://xerces.apache.org/xerces2-j/features.html and for details about the exact constants, see https://xerces.apache.org/xerces2-j/javadocs/xerces2/org/apache/xerces/parsers/XML11Configuration.html. ■ **Feature value**. Specifies whether the feature value is **True** or **False**. **Schema validation for GraphQL API:** The incoming query or mutation payloads are validated against the GraphQL schema type system.
Query Parameters	*This is applicable only to a REST API*. Validates the query parameters in the incoming request against the query parameters defined in that request's API Specification.
Path Parameters	*This is applicable only to a REST API*. Validates the path parameters in the incoming request against the path parameters defined in that request's API Specification.
Cookie Parameters	*This is not applicable to a GraphQL API*. Validates the cookie parameters in the incoming request against the cookie parameters defined in that request's API Specification.
Content-types	*This is not applicable to a GraphQL API*. Validates the content-types in the incoming request against the content-types defined in that request's API Specification.
HTTP Headers	*This is not applicable to a GraphQL API*. Validates the HTTP header parameters in the incoming request against the HTTP headers defined in that request's API Specification. Provide the following information: ■ **Condition**. Specifies the logical operator to use to validate multiple HTTP headers in the incoming API requests.

Property	Description
	Available values are: ■ **AND**. API Gateway accepts only the requests that contain all configured HTTP headers. ■ **OR**. This is selected by default. API Gateway accepts requests that contain at least one configured HTTP header. ■ **HTTP Header Key**. Specifies a key that must be passed through the HTTP header of the incoming API requests. ■ **Header Value**. *Optional*. Specifies the corresponding key value that could be passed through the HTTP header of the incoming API requests. As this property supports variable framework, you can make use of the available variables to specify the header value. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373. You can add more HTTP headers by clicking  .

Request Transformation

This policy enables you to configure several transformations on the request messages from clients into a format required by the native API before it is submitted to the native API.

The transformations include Header, Query Parameter, Path Parameter transformation, HTTP Method transformation, Payload transformation, and Advanced transformation. You can configure conditions according to which the transformations are executed.

The table lists the properties that you can specify for this policy:

Property	Description
Condition	Conditions are used to specify when the policy has to be executed. You can add multiple conditions with logical operators. Available values are: ■ **AND**. API Gateway transforms the requests that comply with all the configured conditions. ■ **OR**. This is selected by default. API Gateway transforms the requests that comply with any one configured condition. Click **Add Condition** and provide the following information and click  .

Property	Description
	■ **Variable**: Specifies the variable type with a syntax. ■ **Operator**: Specifies the operator to use to relate variable and the value. You can select one of the following: ■ **Equals** ■ **Equals ignore case** ■ **Not equals** ■ **Not equals ignore case** ■ **Contains** ■ **Not Contains** ■ **Exists** ■ **Not Exists** ■ **Range** ■ **Greater Than** ■ **Less Than** ■ **Value**: Specifies a plain value or value with a syntax. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
Transformation Configuration: Specifies various transformations to be configured.	
Header/Query/Path Transformation for REST API and **Header Transformation** for SOAP API	Specifies the Header, Query or path transformation to be configured for incoming requests. You can add or modify header, query or path transformation parameters by providing the following information: ■ **Variable**. Specifies the variable type with a syntax. ■ **Value**. Specifies a plain value or value with a syntax. You can add multiple variables and corresponding values by clicking  . You can remove any header, query, or path transformation parameters by typing the plain value or value with a syntax. **Note:**

Property	Description
	Software AG recommends you not to modify the headers `${request.headers.Content-Length}` and `${request.headers.Content-Encoding}` as API Gateway adds the right values for these headers before sending the response back to client. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373. **Note:** Payload transformation does not happen automatically for content-type transformation. When you change the content type, ensure that you do payload transformation. For example, if you change the content-type header from application/xml to application/json, you must also change the respective payload from application/xml to application/json.
Method transformation for REST API	Specifies the method transformation to be configured for incoming requests. Select any of the HTTP Method listed: ■ **GET** ■ **POST** ■ **PUT** ■ **DELETE** ■ **HEAD** ■ **CUSTOM** **Note:** When **CUSTOM** is selected, the HTTP method in incoming request is sent to the native service. When other methods are selected, the selected method is used in the request sent to the native service. **Note:** Only Method Transformation happens when configured, but you have to take care of adding payload during transformations involving method change like GET to POST, and so on.
Payload Transformation	Specifies the payload transformation to be configured for incoming requests. **Note:** API Gateway does not process the payload, if the payload is sent as a stream.

Property	Description

	Provide the following information:
	■ **Payload Type**. Specifies the content-type of payload, to which you want to transform. The **Payload** field renders the respective payload editor based on the selected content-type.
	■ **Payload**. Specifies the payload transformation that needs to be applied for the incoming requests.
	As this property supports variable framework, you can make use of the available variables to transform the request messages.
	For example, consider the native API accepting two integer values `value1` and `value2`, and you want to pass these two values from API Gateway to the native API, you can configure the payload field as follows:
	`{`
	`"value1" : 12,`
	`"value2" : 34`
	`}`
	You can also configure the payload field using one or more variables by using variable framework. Let us see another syntax. For example, for the same native API seen in the previous example, if your client sends both the values through headers `val1` and `val2`, and you want to add it to payload for the native API to recognize the input, you can do so by configuring the payload field as follows:
	`{`
	`"value1" :${request.headers.val1},`
	`"value2" :${request.headers.val2}`
	`}`
	For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
	Note:
	If your payload content-type is different from the incoming payload's content-type, you need to transform the content-type of the header using **Header Transformation**.
	■ Click **+ Add xslt document** to add an xslt document and provide the following information:
	■ **XSLT file**. Specifies the XSLT file used to transform the request messages as required.
	Click **Browse** to browse and select a file.
	■ **Feature Name**. Specifies the name of the XSLT feature.

Property	Description

	■ **Feature value**. Specifies the value of the XSLT feature.
	
	You can add more XSLT features and xslt documents by clicking  .
	
	Note:
	API Gateway supports XSLT 1.0 and XSLT 2.0.
	
	■ Click **+ Add xslt transformation alias** and provide the following information:
	
	■ **XSLT Transformation alias**. Specifies the XSLT transformation alias.
	
	When the incoming request is in JSON, you can use a XSLT file similar to the below sample:
	
	`<?xml version="1.0" ?>`
	`<xsl:stylesheet version="1.1"`
	`xmlns:xsl="http://www.w3.org/1999/XSL/Transform">`
	`<xsl:output method="xml"/>`
	`<xsl:template match="/" >`
	`<xsl:element name="fakeroot">`
	`<xsl:element name="fakenode">`
	`<!-- Apply your transformation rules based on the request from the Client-->`
	`</xsl:element>`
	`</xsl:element>`
	`</xsl:template>`
	`</xsl:stylesheet>`
	
	When the incoming request is in XML, you can use a XSLT file similar to the below sample:
	
	`<?xml version="1.0" ?>`
	`<xsl:stylesheet version="1.1"`
	`xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`
	`xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">`
	`<xsl:output method="xml"/>`
	`<xsl:template match="/" >`
	`<xsl:element name="soapenv:Envelope">`
	`<xsl:element name="soapenv:Body">`
	`<!-- Apply your transformation rules based on the request from the Client-->`
	`</xsl:element>`
	`</xsl:element>`
	`</xsl:template>`
	`</xsl:stylesheet>`

Advanced Transformation	Specifies the advanced transformation to be configured for incoming requests. Provide the following information:

Property	Description
	■ **webMethods IS Service**. Specify the webMethods IS service to be invoked to process the request messages. You can add multiple services by clicking  . **Note:** The webMethods IS service must be running on the same Integration Server as API Gateway. ■ **Run as User**. Specifies the authentication mode to invoke the IS service. If this field is left blank the incoming credentials of the user, identified by API Gateway, are used to authenticate and invoke the IS service. You can also specify a particular user, you want API Gateway to use to invoke the IS service. ■ **Comply to IS Spec**. Mark this as `true` if you want the input and the output parameters to comply to the IS Spec present in pub.apigateway.invokeISService.specifications folder in wmAPIGateway package. ■ **webMethods IS Service alias**. Specifies the webMethods IS service alias to be invoked to pre-process the request messages.
Transformation Metadata: Specifies the metadata for transformation of the incoming requests. For example, the namespaces configured in this section can be used when you provide the syntax for XPath ${request.payload.xpath} For example: ${request.payload.xpath[//ns:emp/ns:empName]}	
Namespace	Specifies the namespace information to be configured for transformation. Provide the following information: ■ **Namespace Prefix**. The namespace prefix of the payload expression to be validated. For example, specify the namespace prefix as `SOAP_ENV`. ■ **Namespace URI**. The namespace URI of the payload expression to be validated. For example, specify the namespace URI as `http://schemas.xmlsoap.org/soap/envelope/`. This declaration defines SOAP_ENV as an alias for the namespace: http://schemas.xmlsoap.org/soap/envelope/. **Note:** You can add multiple namespace prefixes and URIs by clicking  .

Invoke webMethods IS

This policy pre-processes the request messages and transforms the message into the format required by the native API or performs some custom logic, before API Gateway sends the requests to the native APIs.

For example, you might need to accommodate differences between the message content that a client is capable of submitting and the message content that a native API expects. For example, if the client submits an order record using a slightly different structure than the structure expected by the native API, you can use this action to process the record submitted by the client to the structure required by the native API.

If `Comply to IS Spec` parameter is configured as `true`, API Gateway invokes the IS Service with IS specification in the path pub.apigateway.invokeISService.specifications:RequestSpec for Request Processing.

The following are the input and output parameters for REST, SOAP, and WebSocket APIs as specified in the above IS specification. Input parameters can be used to access the existing values of the request while output parameters can be used to modify/write the values to the request.

	Parameter name	Description
Input parameters	**headers**	Headers in incoming request. Data type: Document
	query	Query parameters in incoming request (this is applicable for REST API only). Data type: Document
	path	Path parameter of the incoming request (this is applicable for REST API only). Data type: String
	httpMethod	HTTP Method of the incoming request (this is applicable for REST API only). Data type: String
	payload	Payload of the incoming request. Data type: String
	payloadObject	The payload for binary content types like multi-part / form-data. Data type: Object
	MessageContext	The message context object of the request. Data type: Object

	Parameter name	Description
	apiName	Name of the API invoked by the request. Data type: String
	apiVersion	Version of the API invoked by the request. Data type: String
	requestUrl	URL of the request. Data type: String
	ipInfo	Contains IP information of the request. Data type: Document
	websocketInfo	Websocket related information of the request. Data type: Document
	correlationID	Correlation ID of the request/response. This is unique and same for a request and response. Data type: String
	customFieldsMap	Custom transactional fields can be added to the transactional events using this field. For more information, see **Adding Custom Fields to Transactional Events** section. Data type: Document
	authorization	Authorization information of the request. For more information, see **Accessing authorization values hidden after IAM policy** section. Data type: Document
Output parameters	**headers**	Headers in incoming request. Data type: Document
	query	Query parameters in incoming request (this is applicable for REST API only). Data type: Document
	path	Path parameter of the incoming request (this is applicable for REST API only). Data type: String

Parameter name	Description
httpMethod	HTTP Method of the incoming request (this is applicable for REST API only). Data type: String
payload	Payload of the incoming request. Data type: String
payloadObject	The payload for binary content types like multi-part / form-data. Data type: Object
MessageContext	The message context object of the request. Data type: Object
customFieldsMap	Custom transactional fields can be added to the transactional events using this field. For more information, see **Adding Custom Fields to Transactional Events** section. Data type: Document

By default the "query" pipeline variable is a key value pair, where the value is of type string. But, if the incoming request contains multiple values for the same query parameter and if you want to access those multiple values using **webMethods IS Service**, you have to ensure two things:

1. Make sure that you have checked the **Repeat** check box for query parameter in the **Add Resource Parameter** section of the API details screen.
2. To access or transform multiple values of that query parameter, you have to insert string list (instead of string) under the "query" pipeline variable in the webMethods IS Service.

Note:

- For SOAP to REST APIS, the payload contains the transformed SOAP request.
- Payload transformation does not happen automatically for content-type transformation. When you change the content type, ensure to do payload transformation also as part of IS Service. For example, if you change the content-type header from application/xml to application/json using IS service, you must also change the respective payload from application/xml to application/json
- Only Method Transformation happens when configured, but you have to take care of adding payload during transformations involving method change like GET to POST, and so on.
- When `Comply to IS spec` is `true`, you can change the values of headers, query, payload, and so on, programatically using Message Context, as well as using the pipeline variables given. Software AG recommends you not to change those values directly in Message Context, as the values in output pipeline variables are written to Message Context after the invocation of IS Service.

If `Comply to IS Spec` parameter is set to `false`, API Gateway invokes the IS Service with the same input and output parameters supported in 10.1 and the earlier versions:

- proxy.name
- JSONRESTContentString (REST only)
- SOAPEnvelope (SOAP only)
- EnvelopeString (SOAP only)

The table lists the properties that you can specify for this policy:

Property	Description
Invoke webMethods Integration Server Service	

Add invoke webMethods Integration Server service	Specifies the webMethods IS service to be invoked to pre-process the request messages and the authentication mode for the IS service.
	
	Provide the following information:
	
	■ **webMethods IS Service**. Specify the webMethods IS service to be invoked to pre-process the request messages.
	
	The webMethods IS service must be running on the same Integration Server as API Gateway.
	
	Note:
	If an exception occurs when invoking the webMethods IS service, by default API Gateway displays the status code as *500* and error message as *Internal Server Error*.
	
	You can set custom status code and error message by setting the following properties in the message context of the webMethods IS service:
	■ service.exception.status.code
	■ service.exception.status.message
	
	The sample code is given below:
	
	`IDataCursor idc = pipeline.getCursor();`
	`MessageContext context =`
	`(MessageContext)IDataUtil.get(idc,"MessageContext");`
	`if(context != null)`
	`{`
	`context.setProperty("service.exception.status.code",`
	`404);`
	`context.setProperty("service.exception.status.message",`
	`"Object Not Found");`
	`throw new ServiceException();`
	`}`
	
	Note:

Property	Description
	If ServiceException or FlowException occurs when invoking webMethods IS Service, the message given in the exception is displayed to the client. If any other exception occurs, a generic error message is displayed to the client. ■ **Run as User**. Specifies the authentication mode to invoke the IS service. If this field is left blank the incoming credentials of the user, identified by API Gateway, are used to authenticate and invoke the IS service. You can also specify a particular user, you want API Gateway to invoke the IS service. **Note:** It is the responsibility of the user who activates the API to review the value configured in **Run as User** field to avoid misuse of this configuration. ■ **Comply to IS Spec**. Mark this as `true` if you want the input and the output parameters to comply to the IS Spec present in pub.apigateway.invokeISService.specifications folder in wmAPIGateway package. **Note:**Software AG recommends users to configure the policy with `Comply to IS Spec` as `true`, as you can read or change the values of headers, and so on, without having to read from or write to the message context.
webMethods IS Service alias	Specifies the webMethods IS service alias to be invoked to pre-process the request messages. Start typing the webMethods alias name, select the alias from the type-ahead search results displayed and click + Add to add one or more aliases. You can use the delete icon to delete the added aliases from the list.

Adding Custom Fields to Transactional Events

This section explains you how to add custom fields to the transactional events.

1. Create webMethods IS service by specifying the *pub.apigateway.utils:customFieldInTransactionEventSpec* as a specification reference.
2. In the webMethods IS service, set the required custom fields in the *customFieldsMap* output variable.

3. Once when *customFieldsMap* gets created, the custom fields will be available in the transactional events.

4. Invoke the API with the Invoke webMethods IS policy.

Note:
You can also add the custom fields to the transactional events from API Gateway by configuring the *customTransactionFields.FIELD_NAME* custom variable in the **Custom Extension** policy. For more details, see "How Do I Define a Custom Variable?" on page 397.

Accessing authorization values hidden after IAM policy

By default, API Gateway removes all the authorization related information from client request (for example authorization header) once the IAM policy is engaged. The information like authorization header can be added back to the request sent to native API using Outbound Authentication policy in the Routing stage. However, if the you want to extract the authorization information at the request processing stage for sending the authorization values using a different header to the native API for audit purposes, or performing some business logic in IS Service based on the authorization values, then you can access the authorization values using the `authorization` pipeline variable.

The following table lists the supported authorization values:

Name	Type	Description
clientId	String	clientId identified after the OAuth / JWT / OpenID token is authenticated.
userName	String	Name of the user identified after the IAM policy.
issuer	String	Issuer identified from the JWT token.
authHeader	String	Value of the incoming authorization header sent by client. **Note:** If the authorization header has bearer tokens (such as OAuth, OpenID, or JWT), then the `authHeader` pipeline variable is empty. For such cases, Software AG recommends to use the `incomingToken` pipeline variable.
incomingToken	String	Value of the token in case the incoming authorization header contains a bearer token.
audience	String	Audience identified from the incoming JWT token.
apiKey	String	API Key sent from client.
claims	Document (Key-value pair)	Contains the claims present in the JWT token. You can provide the claim name to access the claim value.
certificates	Object List	Client certificates used for SSL connectivity.

Note:
All the above mentioned authorization values except certificates can be accessed using authorization pipeline variable.

Accessing client certificates used for SSL connectivity

You can now access the client certificates used for SSL Connectivity in the Invoke webMethods IS Service (comply to IS Spec = true) using pipeline authorization > certificates.

Since certificates are not string data type, you need to write JAVA code to convert the pipeline variable certificates into accessible certificate format (Java X509Certificate) and you can read the values using the methods supported by X509Certificate.

The below sample code converts the pipeline variable certificates to X509Certificate:

```
import java.security.cert.X509Certificate;
IDataCursor cursor = pipeline.getCursor();
IData authIData = IDataUtil.getIData(cursor, "authorization");
IDataCursor authCursor = authIData.getCursor();
X509Certificate[] certificates = (X509Certificate[])
IDataUtil.getObjectArray(authCursor, "certificates");
```

The following watt parameters control the certification verification

- **watt.net.ssl.client.hostnameverification**

  When API Gateway server acts as a HTTPS client, this parameter specifies whether API Gateway should restrict outbound HTTPS connections only when a valid hostname is found in the server's certificate. If you set this parameter to true, API Gateway verifies if the hostname is present in the server's certificate. If this verification fails, an error is logged and the connection is aborted. If you set this parameter to false, API Gateway skips the hostname verification. By default, this parameter is set to false.

- **watt.security.ssl.ignoreExpiredChains**

  This parameter specifies whether API Gateway server ignores expired CA certificates in a certificate chain it receives from an Internet resource (that is, a web server, another API Gateway server). If you set this parameter to true, API Gateway, ignores the expired CA certificates. However, API Gateway allows SSL connection to be established, even if the certificate is expired. Note that this is less secure than denying connections when a certificate is expired. If you set this parameter to false, API Gateway does not ignore the expired CA certificates and a connection cannot be established, if a certificate is expired. By default, this parameter is set to false.

- **watt.security.ssl.client.ignoreEmptyAuthoritiesList**

  When API Gateway acts as a client, this parameter specifies if API Gateway sends a certificate chain, after a remote SSL server returns an empty list of trusted authorities. If you set this parameter to true, API Gateway ignores the empty trusted authorities list and sends its chain anyway. If you set this parameter to false, API Gateway requires presentation of trusted certificates before sending out its certificate chain. By default, this parameter is set to false.

Data Masking

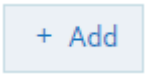
Data masking is a technique whereby sensitive data is obscured in some way to render it safe and to protect the actual data while having a functional substitute for occasions when the real data is not required.

This policy is used to mask sensitive data at the application level. At the application level you must have an Identify and Access policy configured to identify the application for which the masking is applied. If no application is specified then it will be applied for all the other requests. Fields can be masked or filtered in the request messages received. You can configure the masking criteria as required for the XPath, JSONPath, and Regex expressions based on the content-type. This policy can also be applied at the API scope level.

The table lists the content-type and masking criteria mapping.

Content-type	Masking Criteria
application/xml text/xml text/html	XPath
application/json application/json/badgerfish	JSONPath
text/plain	Regex

The table lists the masking criteria properties that you can configure to mask the data in the request messages received:

Property	Description
Consumer Applications	*Optional.* Specifies the applications for which the masking criterion has to be applied. Start typing the application name, select the application from the type-ahead search results displayed, and click  to add one or more applications. For example: If there is a DataMasking(DM1) criteria created for application1 a second DataMasking(DM2) for application2 and a third DataMasking(DM3) with out any application, then for a request that comes from consumer1 the masking criteria DM1 is applied, for a request that comes from consumer2 DM2 is applied. If a request comes with out any application or from any other application except application1 and application2 DM3 is applied.

Property	Description
	You can use the delete icon to delete the added applications from the list.
XPath: Specifies the masking criteria for XPath expressions in the request messages.	
Masking Criteria	Click **Add masking criteria** and provide the following information and click **Add**: ■ **Query expression**. Specify the query expression that has to be masked or filtered. ■ **Masking Type**. Specifies the type of masking required. You select either **Mask** or **Filter**. Selecting **Mask** replaces the value with the given value (the default value being ********). Selecting **Filter** removes the field completely. ■ **Mask Value**. Appears only if you have selected the **Masking Type** as **Mask**. Provide a mask value. You can add multiple masking criteria. As **Query expression** and **Mask Value** properties support variable framework, you can use the available variables. In case of query expression, if you provide variable syntax, the XPath is applied on the payload using the value that is resolved from the variable given. For example, if you provide a query expression as `${request.headers.myxpath}` and the corresponding mask value as `${request.headers.var1}`, and if the incoming request header `myxpath` is configured with value `//ns:cardNumber`, then the card number derived from the payload is masked with the header value in `var1` . For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373. ■ **Namespace**. Specifies the following Namespace information: ■ **Namespace Prefix**. The namespace prefix of the payload expression to be validated. ■ **Namespace URI**. The namespace URI of the payload expression to be validated. **Note:** You can add multiple namespace prefix and URI by clicking +Add.

Property	Description
JSONPath: This is applicable only for REST API. Specifies the masking criteria for JSONPath expressions in the request messages.	
Masking Criteria	Click **Add masking criteria** and provide the following information and click **Add**: ■ **Query expression**. Specify the query expression that has to be masked or filtered. ■ **Masking Type**. Specifies the type of masking required. You select either **Mask** or **Filter**. Selecting **Mask** replaces the value with the given value (the default value being ********). Selecting **Filter** removes the field completely. ■ **Mask Value**. Appears only if you have selected the **Masking Type** as **Mask**. Provide a mask value. As **Query expression** and **Mask Value** properties support variable framework, you can use the available variables. In case of query expression, if you provide variable syntax, the JSONPath is applied on the payload using the value that is resolved from the variable given. For example, if you provide a query expression as `${request.headers.myjsonpath}` and the corresponding mask value as `${request.headers.var1}` , and if the incoming request header `myjsonpath` is configured with value `$.cardNumber`, then the card number derived from the payload is masked with the header value in `var1` . For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
Regex: Specifies the masking criteria for regular expressions in the request messages.	
Masking Criteria	Click **Add masking criteria** and provide the following information and click **Add**: ■ **Query expression**. Specify the query expression that has to be masked or filtered. ■ **Masking Type**. Specifies the type of masking required. You select either **Mask** or **Filter**. Selecting **Mask** replaces the value with the given value (the default value being ********). Selecting **Filter** removes the field completely. ■ **Mask Value**. Appears only if you have selected the **Masking Type** as **Mask**. Provide a mask value.

Property	Description
	As **Query expression** and **Mask Value** properties support variable framework, you can use the available variables. In case of query expression, if you provide variable syntax, the regex is applied on the payload using the value that is resolved from the variable given. For example, if you provide a query expression as `${request.headers.myregex}` and the corresponding mask value as `${request.headers.var1}`, then the regex is applied using the value configured in the request header `myregex` and the derived value is masked with the header value in `var1`. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
Apply for transaction Logging	Select this option to apply masking criteria for transactional logs. **Note:** For REST enabled SOAP services ■ Use *JSONPath.* To mask the incoming request of application/json content-type. ■ Use *XPath of transformed SOAP request.* To mask native service request.
Apply for payload	Select this option to apply masking criteria for request payload in the following scenarios: ■ incoming request from the client. ■ outgoing request to the native service.

Routing

The policies in this stage enforce routing of requests to target APIs based on the rules you can define to route the requests and manage their respective redirections according to the initial request path. The policies included in this stage are:

- Content-based Routing
- Conditional Routing
- Dynamic Routing
- Load Balancer Routing
- Straight Through Routing
- Custom HTTP Header

- Outbound Auth - Transport
- Outbound Auth - Message
- JMS/AMQP Routing
- JMS/AMQP Properties
- Custom Extension

  Custom Extension policies allow you to handle requirements that might not be provided by the out-of-the-box policies. You can add these custom extensions into API Gateway policy stages. To learn more about Custom Extension, see "Custom Policy Extension" on page 383.

In cases where the internal server is protected by a firewall, the endpoint in the routing policy that is applied should be configured as apigateway://*registrationPort-aliasname/relative path of the API*. Here the registration port alias name is the alias name configured for the external registration port to communicate with the internal port.

Straight Through Routing

When you select the Straight Through routing protocol, the API routes the requests directly to the native service endpoint you specify. If your entry protocol is HTTP or HTTPS, you can select the Straight Through routing policy.

The table lists the properties that you can specify for this policy:

Property	Value
Endpoint URI	Specifies the URI of the native API endpoint to route the request to in case all routing rules evaluate to False. Service registries that have been added to the API Gateway instance are also included in the list. If you choose a service registry, API Gateway sends a request to the service registry to discover the IP address and port at which the native service is running. API Gateway replaces the service registry alias in the **Endpoint URI** with the IP address and port returned by the service registry. For example, if your service is hosted at the URL: `http://host:port/abc/`, you need to configure the Endpoint URI as: `http://${ServiceRegistryName}/abc/`. As this property supports variable framework, you can make use of the available variables. For example, you can configure the endpoint URI using hard coded URL, simple alias, endpoint alias, and variable syntax or any of these combination. If you define the endpoint URI as `http://${myAliasHost}:${request.headers.nativeport}/${sys:resource-path}`, where the `${myAliasHost}` variable syntax is used to define the

Property	Value
	simple alias and the `${request.headers.nativeport}` variable syntax is used to define the native port based on the request. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373. For a REST API, the `${sys:resource-path}` alias in the **Endpoint URI** is replaced by the resources and query parameters of the native service. For a GraphQL API, the `${sys:query_string}` alias in the **Endpoint URI** is replaced by the query string of the native service. **Note:** If you use endpoint alias, make sure the endpoint alias is created before you define it in the policy. For example, if you define ${alias} syntax in the policy before creating the `alias` as endpoint alias, API Gateway considers ${alias} as custom variable or simple alias and tries to resolve against those. So in that case, after creating endpoint alias you have to edit and save the API or policy to associate ${alias} syntax with the endpoint alias.
HTTP Method	This is applicable to REST-based APIs. Specifies the available routing methods: **GET**, **POST**, **PUT**, **DELETE**, and **CUSTOM** (default). When **CUSTOM** is selected, the HTTP method in the incoming request is sent to the native service. When other methods are selected, the selected method is used in the request sent to the native service. **Note:** Software AG recommends to use **Request Transformation > Method Transformation** to achieve this as other transformations can also be done under the same policy.
Soap Optimization Method	This is applicable for SOAP-based APIs. Specifies the optimization methods that API Gateway can use to parse SOAP requests to the native API. Select one of the following options: ■ **MTOM**. API Gateway uses the Message Transmission Optimization Mechanism (MTOM) to parse SOAP requests to the API.

Property	Value
	■ **SwA**. API Gateway uses the SOAP with Attachment (SwA) technique to parse SOAP requests to the API. ■ **None**. API Gateway does not use any optimization method to parse the SOAP requests to the API. This is selected by default.
HTTP Connection Timeout (seconds)	Specifies the time interval (in seconds) after which a connection attempt times out. The precedence of the Connection Timeout configuration is as follows: 1. If you specify a value for the **Connection timeout** field in routing endpoint alias, then the **Connection timeout** value specified in the Endpoint alias section takes precedence over the timeout values defined at the API level and the global level. 2. If you specify a value 0 for the **Connection timeout** field in routing endpoint alias, then API Gateway uses the value specified in the **Connection timeout** field in the routing protocol processing step of an API. The Read Timeout value specified at an API level takes precedence over the global configuration. 3. If you specify a value 0 or do not specify a value for the **Connection timeout** field in the routing protocol processing step at the API level or specify a value 0 at an alias level, then API Gateway uses the value specified in this `pg.endpoint.connectionTimeout` property. 4. If you do not specify any value for `pg.endpoint.connectionTimeout`, then API Gateway uses the default value of 30 seconds.
Read Timeout (seconds)	Specifies the time interval (in seconds) after which a socket read attempt times out. The precedence of the Read Timeout configuration is as follows: 1. If you specify a value for the **Read timeout** field in routing endpoint alias, then the **Read timeout** value specified in the Endpoint alias section takes precedence over the timeout values defined at the API level and the global level. 2. If you specify a value 0 for the **Read timeout** field in routing endpoint alias, then API Gateway uses the value specified in the **Read Timeout** field in the routing protocol processing step of an API. The Read Timeout value specified at an API level takes precedence over the global configuration.

Property	Value
	3. If you specify a value 0 or do not specify a value for the **Read timeout** field in the routing protocol processing step at the API level or specify a value 0 at an alias level, then API Gateway uses the value specified in this `pg.endpoint.readTimeout` property. 4. If you do not specify any value for `pg.endpoint.readTimeout`, then API Gateway uses the default value of 30 seconds.
Pass WS-Security Headers	This is applicable for SOAP-based APIs. Selecting this indicates that API Gateway should pass the WS-Security headers of the incoming requests to the native API.
SSL configuration. Configures keystore, key alias, and truststore for securing connections to native APIs.	
Keystore Alias	Specifies the keystore alias configured in API Gateway. This value (along with the value of Client Certificate Alias) is used for performing SSL client authentication. Lists all available keystores. If you have not configured any keystore, the list is empty.
Key Alias	Specifies the alias for the private key, which must be stored in the keystore specified by the keystore alias.
Truststore Alias	Specifies the alias for the truststore that contains the list of CA certificates that API Gateway uses to validate the trust relationship with the native API. If you do not configure any truststore alias, it implies that API Gateway does not validate the certificates provided by native APIs.
Service Registry Configuration	
Service Discovery Endpoint Parameter	Values required for constructing the discovery service URI. ■ **Parameter**: An alias that you have included in the discovery service URI while adding the service registry to API Gateway. ■ **Value**: Specifies a value with a syntax. The alias specified in Path Parameter is substituted with this value when invoking the discovery service. For example: if the service registry configuration of the service registry that you have selected in **Endpoint URI** has **Service discovery path** set to `/catalog/service/{serviceName}` (and the `{serviceName}` alias is intended for passing the service name), you

Property	Value
	must enter `{serviceName}` as **Parameter** and the name of the service as **Value**. As the **Value** field supports variable framework, you can make use of the available variables as path parameters. For example, if you provide a parameter as `{serviceName}`(in **Service discovery path** while you define a service register) and the corresponding values for the path parameter as `${request.header.var1}`, the value in var 1 retrieved from the request header will substitute the service name. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

Custom HTTP Header

You can use this policy to route requests based on the custom HTTP headers specified for the outgoing message to the native service.

The table lists the properties that you can specify for this policy:

Property	Description
HTTP Header Key	Specifies the HTTP header key in the requests.
Header Value	Specifies the Header value contained in the requests. As this property supports variable framework, you can use the available variables to specify the header value. For example, if you provide a header value as `${request.header.token1}`, the header value in `token1` is sent in the outgoing message to authenticate the backend services . For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

You can add multiple entries for the Header key-value pair by clicking 

.

Outbound Auth - Transport

When the native API is protected and expects the authentication credentials to be passed through transport headers, you can use this policy to provide the credentials that will be added to the request and sent to the native API. API Gateway supports a wide range of authentication schemes, such as Basic Authentication, Kerberos, NTLM, and OAuth, at the transport-level.

Note:

Transport-level authentication can be used to secure inbound communication of both the SOAP APIs and the REST APIs.

The table lists the properties that you can specify for this policy:

Property	Description
Authentication scheme	Select one of the following schemes for outbound authentication at the transport level: ■ **Basic**. Uses basic HTTP authentication details to authenticate the client. ■ **Kerberos**. Uses Kerberos credentials for authentication. ■ **NTLM**. Uses NTLM configuration for authentication. ■ **OAuth2**. Uses OAuth token details to authenticate the client. ■ **JWT**. Uses JSON web token details to authenticate the client. ■ **Anonymous**. Authenticates the client without any credentials. ■ **Alias**. Uses the configured alias name for authentication.
Authenticate using	Select one of the following modes to authenticate the client: ■ **Custom credentials**. Uses the values specified in the policy to obtain the required token to access the native API. ■ **Delegate incoming credentials**. Uses the values specified in the policy by the API providers to select whether to delegate the incoming token or act as a normal client. ■ **Incoming HTTP Basic Auth credentials**. Uses the incoming user credentials to retrieve the authentication token to access the native API. ■ **Incoming kerberos credentials**. Uses the incoming kerberos credentials to access the native API. ■ **Incoming OAuth token**. Uses the incoming OAuth2 token to access the native API. ■ **Incoming JWT**. Uses the incoming JSON Web Token (JWT) to access the native API. ■ **Transparent**. Enables NTLM handshake between client and native API. API Gateway does not perform any authentication before passing the incoming requests to native API. It simply passes the incoming credentials to native API. The NTLM authentication happens at the native API.

Property	Description
Basic	Uses the HTTP authentication details to authenticate the client. API Gateway supports the following modes of HTTP authentication: ■ **Custom credentials** ■ **Incoming HTTP Basic Auth credentials** Provide the following credentials: ■ **User Name**. Specifies the user name. ■ **Password**. Specifies the password of the user. ■ **Domain** . Specifies the domain in which the user resides.

Kerberos	Uses the Kerberos credentials to authenticate the client.
	API Gateway supports the following modes of Kerberos authentication:
	■ **Custom credentials**
	■ **Delegate incoming credentials**
	■ **Incoming HTTP basic auth credentials**
	■ **Incoming kerberos credentials**
	Provide the following credentials:
	■ **Client principal**. Provide a valid client LDAP user name.
	■ **Client password**. Provide a valid password of the client LDAP user.
	■ **Service principal**. Provide a valid SPN. The specified value is used by the client to obtain a service ticket from the KDC server.
	■ **Service Principal Name Form**. The SPN type to use while authenticating an incoming client principal name. Select any of the following:
	  ■ **User name**. Specifies the username form.
	  ■ **Hostbased**. Specifies the host form.

NTLM	Uses the NTLM credentials to authenticate the client. API Gateway supports the following modes of NTLM authentication: ■ **Custom credentials**

Property	Description
	■ **Incoming HTTP basic auth credentials** ■ **Transparent** Provide the following credentials: ■ **User Name**. Specifies the user name. ■ **Password**. Specifies the password of the user. ■ **Domain** . Specifies the domain in which the user resides.
OAuth2	Uses the OAuth2 token to authenticate the client. API Gateway supports the following modes of NTLM authentication: ■ **Custom credentials** ■ **Incoming OAuth token** **OAuth2 token**. Specifies the client's OAuth2 token.
JWT	Uses the JSON Web Token (JWT) to authenticate the client. If the native API is enforced to use JWT for authenticating the client, then API Gateway enforces the need for a valid JWT in the outbound request while accessing the native API. API Gateway supports the **Incoming JWT** mode of JWT authentication.
Alias	Uses the configured alias to authenticate the client. Provide the name of the configured alias.

When you configure an API with an inbound authentication policy, and a client sends a request with credentials, API Gateway uses the credentials for the inbound authentication. When sending the request to native server, API Gateway removes the already authenticated credentials when no outbound authentication policy is configured.

If as an API provider you want to use the same credentials for authentication at both API Gateway and native server, you should configure the outbound authentication policy to pass the incoming credentials to the native service. If you do not configure an outbound authentication policy, API Gateway removes the incoming credentials, as it is meant for API Gateway authentication only.

However, when both the inbound authentication policy and outbound authentication policy are not configured, API Gateway just acts as a proxy and forwards the credentials to the native service. Since the credentials are not meant for API Gateway (as no inbound auth policy is configured), API Gateway forwards the credentials to native service (unless there are different settings configured in outbound authentication policy, for example, custom credentials or Anonymous).

Content-based Routing

If you have a native API that is hosted at two or more endpoints, you can use the content-based routing protocol to route specific types of messages to specific endpoints. You can route messages to different endpoints based on specific values that appear in the request message. You might use this capability, for example, to determine which operation the consuming application has requested, and route requests for complex operations to an endpoint on a fast machine. For example, if your entry protocol is HTTP or HTTPS, you can select the Content-based routing. The requests are routed according to the content-based routing rules you create. You may specify how to authenticate requests.

Note:
As the content-based routing policy's capabilities can also be configured using conditional routing policy, the content-based routing policy will be deprecated in future releases and the configurations will be migrated to conditional routing policy. Hence, Software AG recommends to use conditional routing policy over content-based routing policy.

The table lists the properties that you can specify for this policy:

Property	Description
Default Route To: Specifies the URLs of two or more native services in a pool to which the requests are routed.	
Endpoint URI	Specifies the URI of the native API endpoint to route the request to in case all routing rules evaluate to False. Service registries that have been added to the API Gateway instance are also included in the list. If you choose a service registry, API Gateway sends a request to the service registry to discover the IP address and port at which the native service is running. API Gateway replaces the service registry alias in the **Endpoint URI** with the IP address and port returned by the service registry. For example, if your service is hosted at the URL: `http://host:port/abc/`, you need to configure the Endpoint URI as: `http://${ServiceRegistryName}/abc/`. As this property supports variable framework, you can make use of the available variables. For example, you can configure the endpoint URI using hard coded URL, simple alias, endpoint alias, and variable syntax or any of these combination. If you define the endpoint URI as `http://${myAliasHost}:${request.headers.nativeport}/${sys:resource-path}`, where the `${myAliasHost}` variable syntax is used to define the simple alias and the `${request.headers.nativeport}` variable syntax is used to define the native port based on the request.

Property	Description
	For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373. **Note:** If you use endpoint alias, make sure the endpoint alias is created before you define it in the policy. For example, if you define ${alias} syntax in the policy before creating the `alias` as endpoint alias, API Gateway considers ${alias} as custom variable or simple alias and tries to resolve against those. So in that case, after creating endpoint alias you have to edit and save the API or policy to associate ${alias} syntax with the endpoint alias.
HTTP Method	This is applicable for REST-based APIs. Specifies the available routing methods: **GET**, **POST**, **PUT**, **DELETE**, and **CUSTOM** (default). When **CUSTOM** is selected, the HTTP method in the incoming request is sent to the native service. When other methods are selected, the selected method is used in the request sent to the native service. **Note:** Software AG recommends to use **Request Transformation > Method Transformation** to achieve this as other transformations can also be done under the same policy.
SOAP Optimization Method	This is applicable for SOAP-based APIs. Specifies the optimization methods that API Gateway can use to parse SOAP requests to the native API. Select one of the following options: ■ **MTOM**. API Gateway uses the Message Transmission Optimization Mechanism (MTOM) to parse SOAP requests to the API. ■ **SwA**. API Gateway uses the SOAP with Attachment (SwA) technique to parse SOAP requests to the API. ■ **None**. API Gateway does not use any optimization method to parse the SOAP requests to the API. This is selected by default.
HTTP Connection Timeout (seconds)	Specifies the time interval (in seconds) after which a connection attempt times out. The precedence of the Connection Timeout configuration is as follows:

Property	Description
	1. If you specify a value for the **Connection timeout** field in routing endpoint alias, then the **Connection timeout** value specified in the Endpoint alias section takes precedence over the timeout values defined at the API level and the global level. 2. If you specify a value 0 for the **Connection timeout** field in routing endpoint alias, then API Gateway uses the value specified in the **Connection timeout** field in the routing protocol processing step of an API. The Read Timeout value specified at an API level takes precedence over the global configuration. 3. If you specify a value 0 or do not specify a value for the **Connection timeout** field in the routing protocol processing step at the API level or specify a value 0 at an alias level, then API Gateway uses the value specified in this `pg.endpoint.connectionTimeout` property. 4. If you do not specify any value for `pg.endpoint.connectionTimeout`, then API Gateway uses the default value of 30 seconds.
Read Timeout (seconds)	Specifies the time interval (in seconds) after which a socket read attempt times out. The precedence of the Read Timeout configuration is as follows: 1. If you specify a value for the **Read timeout** field in routing endpoint alias, then the **Read timeout** value specified in the Endpoint alias section takes precedence over the timeout values defined at the API level and the global level. 2. If you specify a value 0 for the **Read timeout** field in routing endpoint alias, then API Gateway uses the value specified in the **Read Timeout** field in the routing protocol processing step of an API. The Read Timeout value specified at an API level takes precedence over the global configuration. 3. If you specify a value 0 or do not specify a value for the **Read timeout** field in the routing protocol processing step at the API level or specify a value 0 at an alias level, then API Gateway uses the value specified in this `pg.endpoint.readTimeout` property. 4. If you do not specify any value for `pg.endpoint.readTimeout`, then API Gateway uses the default value of 30 seconds.
Pass WS-Security Headers	This is applicable for SOAP-based APIs.

Property	Description
	Selecting this indicates that API Gateway should pass the WS-Security headers of the incoming requests to the native API.
SSL Configuration. Configures keystore, key alias, and truststore for securing connections to native APIs.	
Keystore Alias	Specifies the keystore alias configured in API Gateway. This value (along with the value of Client Certificate Alias) is used for performing SSL client authentication. Lists all available keystores. If you have not configured any keystore, the list is empty.
Key Alias	Specifies the alias for the private key, which must be stored in the keystore specified by the keystore alias.
Truststore Alias	Specifies the alias for the truststore that contains the list of CA certificates that API Gateway uses to validate the trust relationship with the native API. If you do not configure any truststore alias, it implies that API Gateway does not validate the certificates provided by native APIs.
Service Registry Configuration	
Service Discovery Endpoint Parameter	Values required for constructing the discovery service URI. ■ **Parameter**: An alias that you have included in the discovery service URI while adding the service registry to API Gateway. ■ **Value**: Specifies a value for the path parameter. The alias specified in Path Parameter is substituted with this value when invoking the discovery service. For example: if the service registry configuration of the service registry that you have selected in **Endpoint URI** has **Service discovery path** set to `/catalog/service/{serviceName}` (and the `{serviceName}` alias is intended for passing the service name), you must enter `{serviceName}` as **Parameter** and the name of the service as **Value**. As the **Value** field supports variable framework, you can make use of the available variables as path parameters. For example, if you provide a parameter as `{serviceName}`(in **Service discovery path** while you define a service registry) and the corresponding values for the path parameter as `${request.header.var1}`, the value in `var1` retrieved from the request header substitutes the service name.

Property	Description
	For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
Rule: Defines the routing decisions based on one of the following routing options. Click **Add Rule** and provide the following information.	
Payload Identifier	Specifies using the payload identifier to identify the client, extract the custom authentication credentials supplied in the request represented using the payload identifier, and verify the client's identity. In the Payload identifier section, click **Add payload identifier**, provide the following information, and click **Add**. ■ **Expression type**. Specifies the type of expression, which is used for identification. You can select one the following expression type: ■ **XPath**. Provide the following information: ■ **Payload Expression**. Specifies the payload expression that the specified XPath expression type in the request has to be converted to. For example: /name/id ■ **Namespace Prefix**. The namespace prefix of the payload expression to be validated. ■ **Namespace URI**. The namespace URI of the payload expression to be validated. **Note:** You can add multiple namespace prefix and URI by clicking +Add. ■ **JSONPath**. Provide the **Payload Expression** that specifies the payload expression that the specified JSONPath expression type in the request has to be converted to. For example: $.name.id ■ **Text**. Provide the **Payload Expression** that specifies the payload expression that the specified Text expression type in the request has to be converted to. For example: any valid regular expression. You can add multiple payload identifiers as required. **Note:**

Property	Description
	Only one payload identifier of each type is allowed. For example, you can add a maximum of three payload identifiers, each being of a different type.
Route To. Specifies the Endpoint URI of native APIs in a pool to which the requests are routed.	
Endpoint URI	Specifies the URI of the native API endpoint to route the request to. You can use service registries in a similar manner as described in the main **Endpoint URI** above. For example, if your service is hosted at the URL: `http://host:port/abc/`, you need to configure the Endpoint URI as: `http://${ServiceRegistryName}/abc/`. As this property supports variable framework, you can make use of the available variables. For example, you can configure the endpoint URI using hard coded URL, simple alias, endpoint alias, and variable syntax or any of these combination. If you define the endpoint URI as `http://${myAliasHost}:${request.headers.nativeport}/${sys:resource-path}`, where the `${myAliasHost}` variable syntax is used to define the simple alias and the `${request.headers.nativeport}` variable syntax is used to define the native port based on the request. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373. **Note:** If you use endpoint alias, make sure the endpoint alias is created before you define it in the policy. For example, if you define ${alias} syntax in the policy before creating the `alias` as endpoint alias, API Gateway considers ${alias} as custom variable or simple alias and tries to resolve against those. So in that case, after creating endpoint alias you have to edit and save the API or policy to associate ${alias} syntax with the endpoint alias.
HTTP Method	This is applicable for REST-based APIs. Specifies the available routing methods: **GET**, **POST**, **PUT**, **DELETE**, and **CUSTOM** (default). When **CUSTOM** is selected, the HTTP method in the incoming request is sent to the native service. When other methods are selected, the selected method is used in the request sent to the native service.
Soap Optimization Method	This is applicable for SOAP-based APIs.

Property	Description
	Specifies the optimization methods that API Gateway can use to parse SOAP requests to the native API. Select one of the following options: ■ **MTOM**. API Gateway uses the Message Transmission Optimization Mechanism (MTOM) to parse SOAP requests to the API. ■ **SwA**. API Gateway uses the SOAP with Attachment (SwA) technique to parse SOAP requests to the API. ■ **None**. API Gateway does not use any optimization method to parse the SOAP requests to the API. This is selected by default.
HTTP Connection Timeout (seconds)	Specifies the time interval (in seconds) after which a connection attempt times out. The precedence of the Connection Timeout configuration is as follows: 1. If you specify a value for the **Connection timeout** field in routing endpoint alias, then the **Connection timeout** value specified in the Endpoint alias section takes precedence over the timeout values defined at the API level and the global level. 2. If you specify a value 0 for the **Connection timeout** field in routing endpoint alias, then API Gateway uses the value specified in the **Connection timeout** field in the routing protocol processing step of an API. The Read Timeout value specified at an API level takes precedence over the global configuration. 3. If you specify a value 0 or do not specify a value for the **Connection timeout** field in the routing protocol processing step at the API level or specify a value 0 at an alias level, then API Gateway uses the value specified in this `pg.endpoint.connectionTimeout` property. 4. If you do not specify any value for `pg.endpoint.connectionTimeout`, then API Gateway uses the default value of 30 seconds.
Read Timeout (seconds)	Specifies the time interval (in seconds) after which a socket read attempt times out. The precedence of the Read Timeout configuration is as follows: 1. If you specify a value for the **Read timeout** field in routing endpoint alias, then the **Read timeout** value specified in the

Property	Description
	Endpoint alias section takes precedence over the timeout values defined at the API level and the global level. 2. If you specify a value 0 for the **Read timeout** field in routing endpoint alias, then API Gateway uses the value specified in the **Read Timeout** field in the routing protocol processing step of an API. The Read Timeout value specified at an API level takes precedence over the global configuration. 3. If you specify a value 0 or do not specify a value for the **Read timeout** field in the routing protocol processing step at the API level or specify a value 0 at an alias level, then API Gateway uses the value specified in this `pg.endpoint.readTimeout` property. 4. If you do not specify any value for `pg.endpoint.readTimeout`, then API Gateway uses the default value of 30 seconds.
Pass WS-Security Headers	This is applicable for SOAP-based APIs. Selecting this indicates that API Gateway should pass the WS-Security headers of the incoming requests to the native API.
SSL Configuration	Configures keystore, key alias, and truststore for securing connections to native APIs. Provide the following information: ■ **Keystore Alias**. Specifies the keystore alias configured in API Gateway. This value (along with the value of Client Certificate Alias) is used for performing SSL client authentication. Lists all available keystores. If you have not configured any keystore, the list is empty. ■ **Key Alias**. Specifies the alias for the private key, which must be stored in the keystore specified by the keystore alias. ■ **Truststore Alias**. Specifies the alias for the truststore that contains the list of CA certificates that API Gateway uses to validate the trust relationship with the native API. If you do not configure any truststore alias, it implies that API Gateway does not validate the certificates provided by native APIs.
Service Registry Configuration	
Service Discovery Endpoint Parameter	Values required for constructing the discovery service URI.

Property	Description
	■ **Parameter**: An alias that you have included in the discovery service URI while adding the service registry to API Gateway. ■ **Value**: Specifies a value for the path parameter. The alias specified in Path Parameter is substituted with this value when invoking the discovery service. For example: if the service registry configuration of the service registry that you have selected in **Endpoint URI** has **Service discovery path** set to `/catalog/service/{serviceName}` (and the `{serviceName}` alias is intended for passing the service name), you must enter `{serviceName}` as **Parameter** and the name of the service as **Value**. As the **Value** field supports variable framework, you can make use of the available variables as path parameters. For example, if you provide a parameter as `{serviceName}`(in **Service discovery path** while you define a service register) and the corresponding values for the path parameter as `${request.header.var1}`, the value in var 1 retrieved from the request header will substitute the service name. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

Conditional Routing

If you have a native API that is hosted at two or more endpoints, you can use the condition-based protocol to route specific types of messages to specific endpoints. The requests are routed according to the condition-based routing rules you create. For example, if your entry protocol is HTTP or HTTPS, you can select conditional routing specifying HTTP or HTTPS. A routing rule specifies where requests should be routed to, and the criteria to use to route. You may also specify how to authenticate requests.

Note:
The context-based routing policy is renamed and it's capabilities are included in conditional routing policy. You can use this policy to configure to route the requests conditionally based on variable types.

The following table provides the existing options of routing till API Gateway version 10.5 and their corresponding variable syntax to choose the same option in API Gateway version 10.7.

10.5 Conditional Variable	10.5 Condition Operator	10.7 Transformation Variable	10.7 Transformation Condition Operator
Consumer		${request.application.id}	Equals

10.5 Conditional Variable	10.5 Condition Operator	10.7 Transformation Variable	10.7 Transformation Condition Operator
Date	Before	${date}	Lesser than
	After		Greater than
Time	Before	${time}	Lesser than
	After		Greater than
Predefined System Context Variables			
User	Equal to	${user}	Equals
Inbound HTTP Method	Not equal to	${inboundHttpMethod}	Not equals
Routing Method		${routingMethod}	
Inbound Content Type		${inboundContentType}	
Inbound Accept		${inboundAccept}	
Inbound Protocol		${inboundProtocol}	
Inbound Request URI		${inboundRequestURI}	
Inbound IP		${inboundIP	
Gateway Hostname		${gatewayHostname}	
Gateway IP		${gatewayIP}	
Operation Name		${operationName}	
Custom Context Variables			
mx:var1	Equal to	${var1}	Equals
PROTOCOL_HEADERS[KEY]	Not equal to	${request.headers.KEY}	Not Equals
SOAP_HEADERS[INDEX]	Lesser than	${soapHeaders[INDEX}	Lesser than
	Greater than		Greater than
IPV4	-	${inboundIP}	Range
IPV6	-	${inboundIP}	Range

The table lists the properties that you can specify for this policy:

Property	Description
Route To	Specifies the URLs of two or more native services in a pool to which the requests are routed.

Property	Description
Endpoint URI	Specifies the URI of the native API endpoint to route the request to in case all routing rules evaluate to False. Service registries that have been added to the API Gateway instance are also included in the list. If you choose a service registry, API Gateway sends a request to the service registry to discover the IP address and port at which the native service is running. API Gateway replaces the service registry alias in the **Endpoint URI** with the IP address and port returned by the service registry. For example, if your service is hosted at the URL: `http://host:port/abc/`, you need to configure the Endpoint URI as: `http://${ServiceRegistryName}/abc/`. As this property supports variable framework, you can make use of the available variables. For example, you can configure the endpoint URI using hard coded URL, simple alias, endpoint alias, and variable syntax or any of these combination. If you define the endpoint URI as `http://${myAliasHost}:${request.headers.nativeport}/${sys:resource-path}`, where the `${myAliasHost}` variable syntax is used to define the simple alias and the `${request.headers.nativeport}` variable syntax is used to define the native port based on the request. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373. **Note:** If you use endpoint alias, make sure the endpoint alias is created before you define it in the policy. For example, if you define ${alias} syntax in the policy before creating the `alias` as endpoint alias, API Gateway considers ${alias} as custom variable or simple alias and tries to resolve against those. So in that case, after creating endpoint alias you have to edit and save the API or policy to associate ${alias} syntax with the endpoint alias.
HTTP Method	This is applicable to REST-based APIs. Specifies the available routing methods: **GET**, **POST**, **PUT**, **DELETE**, and **CUSTOM** (default). When **CUSTOM** is selected, the HTTP method in the incoming request is sent to the native service. When other methods are selected, the selected method is used in the request sent to the native service. **Note:**

Property	Description
	Software AG recommends to use **Request Transformation > Method Transformation** to achieve this as other transformations can also be done under the same policy.
SOAP Optimization Method	This is applicable for SOAP-based APIs. Specifies the optimization methods that API Gateway can use to parse SOAP requests to the native API. Select one of the following options: ■ **MTOM**.API Gateway uses the Message Transmission Optimization Mechanism (MTOM) to parse SOAP requests to the API. ■ **SwA**. API Gateway uses the SOAP with Attachment (SwA) technique to parse SOAP requests to the API. ■ **None**. API Gateway does not use any optimization method to parse the SOAP requests to the API. This is selected by default.
HTTP Connection Timeout (seconds)	Specifies the time interval (in seconds) after which a connection attempt times out. The precedence of the Connection Timeout configuration is as follows: 1. If you specify a value for the **Connection timeout** field in routing endpoint alias, then the **Connection timeout** value specified in the Endpoint alias section takes precedence over the timeout values defined at the API level and the global level. 2. If you specify a value 0 for the **Connection timeout** field in routing endpoint alias, then API Gateway uses the value specified in the **Connection timeout** field in the routing protocol processing step of an API. The Read Timeout value specified at an API level takes precedence over the global configuration. 3. If you specify a value 0 or do not specify a value for the **Connection timeout** field in the routing protocol processing step at the API level or specify a value 0 at an alias level, then API Gateway uses the value specified in this `pg.endpoint.connectionTimeout` property. 4. If you do not specify any value for `pg.endpoint.connectionTimeout`, then API Gateway uses the default value of 30 seconds.
Read Timeout (seconds)	Specifies the time interval (in seconds) after which a socket read attempt times out.

Property	Description
	The precedence of the Read Timeout configuration is as follows: 1. If you specify a value for the **Read timeout** field in routing endpoint alias, then the **Read timeout** value specified in the Endpoint alias section takes precedence over the timeout values defined at the API level and the global level. 2. If you specify a value 0 for the **Read timeout** field in routing endpoint alias, then API Gateway uses the value specified in the **Read Timeout** field in the routing protocol processing step of an API. The Read Timeout value specified at an API level takes precedence over the global configuration. 3. If you specify a value 0 or do not specify a value for the **Read timeout** field in the routing protocol processing step at the API level or specify a value 0 at an alias level, then API Gateway uses the value specified in this `pg.endpoint.readTimeout` property. 4. If you do not specify any value for `pg.endpoint.readTimeout`, then API Gateway uses the default value of 30 seconds.
Pass WS-Security Headers	This is applicable for SOAP-based APIs. Selecting this indicates that API Gateway should pass the WS-Security headers of the incoming requests to the native API.
SSL Configuration. Configures keystore, key alias, and truststore for securing connections to native APIs.	
Keystore Alias	Specifies the keystore alias configured in API Gateway. This value (along with the value of Client Certificate Alias) is used for performing SSL client authentication. Lists all available keystores. If you have not configured any keystore, the list is empty.
Key Alias	Specifies the alias for the private key, which must be stored in the keystore specified by the keystore alias.
Truststore Alias	Specifies the alias for the truststore that contains the list of CA certificates that API Gateway uses to validate the trust relationship with the native API. If you do not configure any truststore alias, it implies that API Gateway does not validate the certificates provided by native APIs.
Service Registry Configuration	

Property	Description
Service Discovery Endpoint Parameter	Values required for constructing the discovery service URI. ■ **Parameter**: An alias that you have included in the discovery service URI while adding the service registry to API Gateway. ■ **Value**: Specifies a value with a syntax. The alias specified in Path Parameter is substituted with this value when invoking the discovery service. For example: if the service registry configuration of the service registry that you have selected in **Endpoint URI** has **Service discovery path** set to `/catalog/service/{serviceName}` (and the `{serviceName}` alias is intended for passing the service name), you must enter `{serviceName}` as **Parameter** and the name of the service as **Value**. As the **Value** field supports variable framework, you can make use of the available variables as path parameters. For example, if you provide a parameter as `{serviceName}`(in **Service discovery path** while you define a service register) and the corresponding values for the path parameter as `${request.header.var1}`, the value in var 1 retrieved from the request header will substitute the service name. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
Rule. Defines the routing decisions based on one of the following routing options.	
Name	Provide a name for the rule.
Condition Operator	Specifies the condition operator to be used. Select one of the following operators: ■ **OR**. Specifies that one of the set conditions should be applied. ■ **AND**. Specifies all the set conditions should be applied.

Add Condition	Specify the context variables for processing client requests.
	■ **Variable**: Specifies the variable type.
	■ **Operator**: Specifies the operator to use to relate variable and the value. You can select one of the following:
	  ■ **Equals**
	  ■ **Equals ignore case**

Property	Description
	■ **Not equals** ■ **Not equals ignore case** ■ **Contains** ■ **Not Contains** ■ **Exists** ■ **Not Exists** ■ **Range** ■ **Greater Than** ■ **Less Than** ■ **Value**: Specifies a plain value or value with a syntax. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
Route To. Specifies the endpoint URI of native services in a pool to which the requests are routed.	
Endpoint URI	Specifies the URI of the native API endpoint to route the request to. You can use service registries in a similar manner as described in the main **Endpoint URI** above. As this property supports variable framework, you can make use of the available variables. For example, you can configure the endpoint URI using hard coded URL, simple alias, endpoint alias, and variable syntax or any of these combination. If you define the endpoint URI as `http://${myAliasHost}:${request.headers.nativeport}/${sys:resource-path}`, where the `${myAliasHost}` variable syntax is used to define the simple alias and the `${request.headers.nativeport}` variable syntax is used to define the native port based on the request. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373. **Note:** If you use endpoint alias, make sure the endpoint alias is created before you define it in the policy. For example, if you define ${alias} syntax in the policy before creating the `alias` as endpoint alias, API Gateway considers ${alias} as custom variable or simple alias and tries to resolve against those. So in that case, after creating endpoint alias you have to edit and save the API or policy to associate ${alias} syntax with the endpoint alias.

Property	Description
HTTP Method	This is applicable for REST-based APIs. Specifies the available routing methods: **GET**, **POST**, **PUT**, **DELETE**, and **CUSTOM** (default). When **CUSTOM** is selected, the HTTP method in the incoming request is sent to the native service. When other methods are selected, the selected method is used in the request sent to the native service.
Soap Optimization Method	This is applicable for SOAP-based APIs. Specifies values to enable SSL authentication for SOAP APIs. Select one of the following options: ■ **MTOM**. API Gateway uses the Message Transmission Optimization Mechanism (MTOM) to parse SOAP requests to the API. ■ **SwA**. API Gateway uses the SOAP with Attachment (SwA) technique to parse SOAP requests to the API. ■ **None**. API Gateway does not use any optimization method to parse the SOAP requests to the API. This is selected by default.
HTTP Connection Timeout (seconds)	Specifies the time interval (in seconds) after which a connection attempt times out. The precedence of the Connection Timeout configuration is as follows: 1. If you specify a value for the **Connection timeout** field in routing endpoint alias, then the **Connection timeout** value specified in the Endpoint alias section takes precedence over the timeout values defined at the API level and the global level. 2. If you specify a value 0 for the **Connection timeout** field in routing endpoint alias, then API Gateway uses the value specified in the **Connection timeout** field in the routing protocol processing step of an API. The Read Timeout value specified at an API level takes precedence over the global configuration. 3. If you specify a value 0 or do not specify a value for the **Connection timeout** field in the routing protocol processing step at the API level or specify a value 0 at an alias level, then API Gateway uses the value specified in this `pg.endpoint.connectionTimeout` property.

Property	Description
	4. If you do not specify any value for `pg.endpoint.connectionTimeout`, then API Gateway uses the default value of 30 seconds.
Read Timeout (seconds)	Specifies the time interval (in seconds) after which a socket read attempt times out. The precedence of the Read Timeout configuration is as follows: 1. If you specify a value for the **Read timeout** field in routing endpoint alias, then the **Read timeout** value specified in the Endpoint alias section takes precedence over the timeout values defined at the API level and the global level. 2. If you specify a value 0 for the **Read timeout** field in routing endpoint alias, then API Gateway uses the value specified in the **Read Timeout** field in the routing protocol processing step of an API. The Read Timeout value specified at an API level takes precedence over the global configuration. 3. If you specify a value 0 or do not specify a value for the **Read timeout** field in the routing protocol processing step at the API level or specify a value 0 at an alias level, then API Gateway uses the value specified in this `pg.endpoint.readTimeout` property. 4. If you do not specify any value for `pg.endpoint.readTimeout`, then API Gateway uses the default value of 30 seconds.
Pass WS-Security Headers	This is applicable for SOAP-based APIs. Selecting this indicates that API Gateway should pass the WS-Security headers of the incoming requests to the native API.
SSL Configuration	Configures keystore, key alias, and truststore for securing connections to native APIs. Provide the following information: ■ **Keystore Alias**. Specifies the keystore alias configured in API Gateway. This value (along with the value of Client Certificate Alias) is used for performing SSL client authentication. Lists all available keystores. If you have not configured any keystore, the list is empty. ■ **Key Alias**. Specifies the alias for the private key, which must be stored in the keystore specified by the keystore alias.

Property	Description
	■ **Truststore Alias**. Specifies the alias for the truststore that contains the list of CA certificates that API Gateway uses to validate the trust relationship with the native API. If you do not configure any truststore alias, it implies that API Gateway does not validate the certificates provided by native APIs.
Service Registry Configuration	
Service Discovery Endpoint Parameter	Values required for constructing the discovery service URI. ■ **Parameter**: An alias that you have included in the discovery service URI while adding the service registry to API Gateway. ■ **Value**: Specifies a value with a syntax. The alias specified in Path Parameter is substituted with this value when invoking the discovery service. For example: if the service registry configuration of the service registry that you have selected in **Endpoint URI** has **Service discovery path** set to `/catalog/service/{serviceName}` (and the `{serviceName}` alias is intended for passing the service name), you must enter `{serviceName}` as **Parameter** and the name of the service as **Value**. As the **Value** field supports variable framework, you can make use of the available variables as path parameters. For example, if you provide a parameter as `{serviceName}`(in **Service discovery path** while you define a service register) and the corresponding values for the path parameter as `${request.header.var1}`, the value in var 1 retrieved from the request header will substitute the service name. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

Dynamic Routing

This policy enables API Gateway to support dynamic routing of virtual aliases based on policy configuration. The configured policies are enforced on the request sent to an API and these requests are forwarded to the dynamic endpoint based on specific criteria that you specify.

Note:
As the dynamic routing policy's capabilities can also be configured using conditional routing policy, the dynamic routing policy will be deprecated in future releases and the configurations will be migrated to conditional routing policy. Hence, Software AG recommends to use conditional routing policy over dynamic routing policy. In future version, when dynamic routing

is migrated to conditional routing policy ${sys:dyn_Endpoint} will be replaced with ${dynamicEndpoint} system variable.

The table lists the properties that you can specify for this policy:

Property	Description
Route To. Specifies the URLs of two or more native services in a pool to which the requests are routed.	
Endpoint URI	Specifies the URI of the native API endpoint to route the request to in case all routing rules evaluate to False. Service registries that have been added to the API Gateway instance are also included in the list. If you choose a service registry, API Gateway sends a request to the service registry to discover the IP address and port at which the native service is running. API Gateway replaces the service registry alias in the **Endpoint URI** with the IP address and port returned by the service registry. For example, if your service is hosted at the URL: `http://host:port/abc/`, you need to configure the Endpoint URI as: `http://${ServiceRegistryName}/abc/`. As this property supports variable framework, you can make use of the available variables. For example, you can configure the endpoint URI using hard coded URL, simple alias, endpoint alias, and variable syntax or any of these combination. If you define the endpoint URI as `http://${myAliasHost}:${request.headers.nativeport}/${sys:resource-path}`, where the `${myAliasHost}` variable syntax is used to define the simple alias and the `${request.headers.nativeport}` variable syntax is used to define the native port based on the request. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373. **Note:** If you use endpoint alias, make sure the endpoint alias is created before you define it in the policy. For example, if you define ${alias} syntax in the policy before creating the `alias` as endpoint alias, API Gateway considers ${alias} as custom variable or simple alias and tries to resolve against those. So in that case, after creating endpoint alias you have to edit and save the API or policy to associate ${alias} syntax with the endpoint alias.
HTTP Method	This is applicable to REST-based APIs. Specifies the available routing methods: **GET**, **POST**, **PUT**, **DELETE**, and **CUSTOM** (default).

Property	Description
	When **CUSTOM** is selected, the HTTP method in the incoming request is sent to the native service. When other methods are selected, the selected method is used in the request sent to the native service. **Note:** Software AG recommends to use **Request Transformation > Method Transformation** to achieve this as other transformations can also be done under the same policy.
SOAP Optimization Method	This is applicable for SOAP-based APIs. Specifies the optimization methods that API Gateway can use to parse SOAP requests to the native API. Select one of the following options: ■ **MTOM**. API Gateway uses the Message Transmission Optimization Mechanism (MTOM) to parse SOAP requests to the API. ■ **SwA**. API Gateway uses the SOAP with Attachment (SwA) technique to parse SOAP requests to the API. ■ **None**. API Gateway does not use any optimization method to parse the SOAP requests to the API. This is selected by default.
HTTP Connection Timeout (seconds)	Specifies the time interval (in seconds) after which a connection attempt times out. The precedence of the Connection Timeout configuration is as follows: 1. If you specify a value for the **Connection timeout** field in routing endpoint alias, then the **Connection timeout** value specified in the Endpoint alias section takes precedence over the timeout values defined at the API level and the global level. 2. If you specify a value 0 for the **Connection timeout** field in routing endpoint alias, then API Gateway uses the value specified in the **Connection timeout** field in the routing protocol processing step of an API. The Read Timeout value specified at an API level takes precedence over the global configuration. 3. If you specify a value 0 or do not specify a value for the **Connection timeout** field in the routing protocol processing step at the API level or specify a value 0 at an alias level, then API Gateway uses the value specified in this `pg.endpoint.connectionTimeout` property.

Property	Description
	4. If you do not specify any value for `pg.endpoint.connectionTimeout`, then API Gateway uses the default value of 30 seconds.
Read Timeout (seconds)	Specifies the time interval (in seconds) after which a socket read attempt times out. The precedence of the Read Timeout configuration is as follows: 1. If you specify a value for the **Read timeout** field in routing endpoint alias, then the **Read timeout** value specified in the Endpoint alias section takes precedence over the timeout values defined at the API level and the global level. 2. If you specify a value 0 for the **Read timeout** field in routing endpoint alias, then API Gateway uses the value specified in the **Read Timeout** field in the routing protocol processing step of an API. The Read Timeout value specified at an API level takes precedence over the global configuration. 3. If you specify a value 0 or do not specify a value for the **Read timeout** field in the routing protocol processing step at the API level or specify a value 0 at an alias level, then API Gateway uses the value specified in this `pg.endpoint.readTimeout` property. 4. If you do not specify any value for `pg.endpoint.readTimeout`, then API Gateway uses the default value of 30 seconds.
Pass WS-Security Headers	This is applicable for SOAP-based APIs. Selecting this indicates that API Gateway should pass the WS-Security headers of the incoming requests to the native API.
SSL Configuration. Configures keystore, key alias, and truststore for securing connections to native APIs.	
Keystore Alias	Specifies the keystore alias configured in API Gateway. This value (along with the value of Client Certificate Alias) is used for performing SSL client authentication. Lists all available keystores. If you have not configured any keystore, the list is empty.
Key Alias	Specifies the alias for the private key, which must be stored in the keystore specified by the keystore alias.
Truststore Alias	Specifies the alias for the truststore that contains the list of CA certificates that API Gateway uses to validate the trust relationship with the native API.

Property	Description
	If you do not configure any truststore alias, it implies that API Gateway does not validate the certificates provided by native APIs.
Service Registry Configuration	
Service Discovery Endpoint Parameter	Values required for constructing the discovery service URI. ■ **Parameter**: An alias that you have included in the discovery service URI while adding the service registry to API Gateway. ■ **Value**: Specifies a value with a syntax. The alias specified in Path Parameter is substituted with this value when invoking the discovery service. For example: if the service registry configuration of the service registry that you have selected in **Endpoint URI** has **Service discovery path** set to `/catalog/service/{serviceName}` (and the `{serviceName}` alias is intended for passing the service name), you must enter `{serviceName}` as **Parameter** and the name of the service as **Value**. As the **Value** field supports variable framework, you can make use of the available variables as path parameters. For example, if you provide a parameter as `{serviceName}`(in **Service discovery path** while you define a service register) and the corresponding values for the path parameter as `${request.header.var1}`, the value in var 1 retrieved from the request header will substitute the service name. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
Rule. Defines the routing decisions based on one of the following routing options.	
Route Using	Defines the dynamic URL based on the HTTP header value sent by the client or the context variable value. Select one of the following: ■ **Header**: Select and specify the **Name** required. This header name is configured by the API provider and is used to decide the routing decisions at the API level. The request message must be routed to the dynamic URL generated from the HTTP header value. ■ **Context**: The API providers must provide IS service in the policy, Invoke webMethods Integration Server. IS service would perform custom manipulations and set the value for the Context Variable ROUTING_ENDPOINT. API Gateway takes this

Property	Description
	ROUTING_ENDPOINT value as the native endpoint value and performs the routing. ■ **Name**. This field is displayed only when you select **Header** as the routing method. Type a name for the Routing header. API Gateway expects this header name in the incoming request that invokes the API.
Route To	Specifies the endpoint URI of native services in a pool to which the requests are routed. Provide the following information: ■ **Endpoint URI** . Specifies the URI of the native API endpoint to route the request to. You can use service registries in a similar manner as described in the main **Endpoint URI** above. As this property supports variable framework, you can make use of the available variables. For example, you can configure the endpoint URI using hard coded URL, simple alias, endpoint alias, and variable syntax or any of these combination. If you define the endpoint URI as `http://${myAliasHost}:${request.headers.nativeport}/${sys:resource-path}`, where the `${myAliasHost}` variable syntax is used to define the simple alias and the `${request.headers.nativeport}` variable syntax is used to define the native port based on the request. You can also use the system-defined alias `${sys:dyn-Endpoint}`. When you use the system-defined alias, the variables are replaced at runtime by the **Header** value or the **Context** value, selected as the **Route To** option. Consider the following URL with the system-defined alias: `http://HOSTNAME:5555/rest/com/softwareag/mediator/samples/dynamicRouting/validateDynamicURI/${sys:dyn-Endpoint}` Now, if the incoming request has **Header** value as **resource**, the `${sys:dyn-Endpoint}` alias in the URL is replaced by the **Header** value and the effective URL is `http://HOSTNAME:5555/rest/com/softwareag/mediator/samples/dynamicRouting/validateDynamicURI/resource`. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

Property	Description
	Note: If you use endpoint alias, make sure the endpoint alias is created before you define it in the policy. For example, if you define ${alias} syntax in the policy before creating the `alias` as endpoint alias, API Gateway considers ${alias} as custom variable or simple alias and tries to resolve against those. So in that case, after creating endpoint alias you have to edit and save the API or policy to associate ${alias} syntax with the endpoint alias. - **HTTP Method**. This applicable to REST-based APIs. Specifies the available routing methods: **GET**, **POST**, **PUT**, **DELETE**, and **CUSTOM** (default). When **CUSTOM** is selected, the HTTP method in the incoming request is sent to the native service. When other methods are selected, the selected method is used in the request sent to the native service. - **HTTP Connection Timeout (seconds)**. Specifies the time interval (in seconds) after which a connection attempt times out. The precedence of the Connection Timeout configuration is as follows: 1. If you specify a value for the **Connection timeout** field in routing endpoint alias, then the **Connection timeout** value specified in the Endpoint alias section takes precedence over the timeout values defined at the API level and the global level. 2. If you specify a value 0 for the **Connection timeout** field in routing endpoint alias, then API Gateway uses the value specified in the **Connection timeout** field in the routing protocol processing step of an API. The Read Timeout value specified at an API level takes precedence over the global configuration. 3. If you specify a value 0 or do not specify a value for the **Connection timeout** field in the routing protocol processing step at the API level or specify a value 0 at an alias level, then API Gateway uses the value specified in this `pg.endpoint.connectionTimeout` property. 4. If you do not specify any value for `pg.endpoint.connectionTimeout`, then API Gateway uses the default value of 30 seconds.

Property	Description
	■ **Read Timeout (seconds)**. Specifies the time interval (in seconds) after which a socket read attempt times out. The precedence of the Read Timeout configuration is as follows: 1. If you specify a value for the **Read timeout** field in routing endpoint alias, then the **Read timeout** value specified in the Endpoint alias section takes precedence over the timeout values defined at the API level and the global level. 2. If you specify a value 0 for the **Read timeout** field in routing endpoint alias, then API Gateway uses the value specified in the **Read Timeout** field in the routing protocol processing step of an API. The Read Timeout value specified at an API level takes precedence over the global configuration. 3. If you specify a value 0 or do not specify a value for the **Read timeout** field in the routing protocol processing step at the API level or specify a value 0 at an alias level, then API Gateway uses the value specified in this `pg.endpoint.readTimeout` property. 4. If you do not specify any value for `pg.endpoint.readTimeout`, then API Gateway uses the default value of 30 seconds. ■ **SSL Configuration**. Configures keystore, key alias, and truststore for securing connections to native APIs. Provide the following information: ■ **Keystore Alias**. Specifies the keystore alias configured in API Gateway. This value (along with the value of Client Certificate Alias) is used for performing SSL client authentication. Lists all available keystores. If you have not configured any keystore, the list is empty. ■ **Key Alias**. Specifies the alias for the private key, which must be stored in the keystore specified by the keystore alias. ■ **Truststore Alias**. Specifies the alias for the truststore that contains the list of CA certificates that API Gateway uses to validate the trust relationship with the native API. If you do not configure any truststore alias, it implies that API Gateway does not validate the certificates provided by native APIs.
SOAP Optimization Method	This is applicable for SOAP-based APIs.

Property	Description
	Specifies the optimization methods that API Gateway can use to parse SOAP requests to the native API. Select one of the following options: ■ **MTOM**. API Gateway uses the Message Transmission Optimization Mechanism (MTOM) to parse SOAP requests to the API. ■ **SwA**. API Gateway uses the SOAP with Attachment (SwA) technique to parse SOAP requests to the API. ■ **None**. API Gateway does not use any optimization method to parse the SOAP requests to the API. This is selected by default.
Pass WS-Security Headers	This is applicable for SOAP-based APIs. Selecting this indicates that API Gateway should pass the WS-Security headers of the incoming requests to the native API.

Load Balancer Routing

If you have an API that is hosted at two or more endpoints, you can use the load balancing option to distribute requests among the endpoints. Requests are distributed across multiple endpoints. The requests are routed based on the round-robin strategy. The load for a service is balanced by directing requests to two or more services in a pool, until the optimum level is achieved. The application routes requests to services in the pool sequentially, starting from the first to the last service without considering the individual performance of the services. After the requests have been forwarded to all the services in the pool, the first service is chosen for the next loop of forwarding.

If the entry protocol is HTTP or HTTPS, you can select the Load Balancer routing.

The table lists the properties that you can specify for this policy:

Property	Description
Route To. Specifies the URLs of two or more native services in a pool to which the requests are routed.	
Endpoint URI	Specifies the URI of the native API endpoint to route the request to in case all routing rules evaluate to False. Service registries that have been added to the API Gateway instance are also included in the list. If you choose a service registry, API Gateway sends a request to the service registry to discover the IP address and port at which the native service is running. API Gateway replaces the service

Property	Description
	registry alias in the **Endpoint URI** with the IP address and port returned by the service registry. For example, if your service is hosted at the URL: `http://host:port/abc/`, you need to configure the Endpoint URI as: `http://${ServiceRegistryName}/abc/`. As this property supports variable framework, you can make use of the available variables. For example, you can configure the endpoint URI using hard coded URL, simple alias, endpoint alias, and variable syntax or any of these combination. If you define the endpoint URI as `http://${myAliasHost}:${request.headers.nativeport}/${sys:resource-path}`, where the `${myAliasHost}` variable syntax is used to define the simple alias and the `${request.headers.nativeport}` variable syntax is used to define the native port based on the request. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373. **Note:** If you use endpoint alias, make sure the endpoint alias is created before you define it in the policy. For example, if you define ${alias} syntax in the policy before creating the `alias` as endpoint alias, API Gateway considers ${alias} as custom variable or simple alias and tries to resolve against those. So in that case, after creating endpoint alias you have to edit and save the API or policy to associate ${alias} syntax with the endpoint alias.
HTTP Method	This is applicable to REST APIs. Specifies the available routing methods: **GET**, **POST**, **PUT**, **DELETE**, and **CUSTOM** (default). When **CUSTOM** is selected, the HTTP method in the incoming request is sent to the native service. When other methods are selected, the selected method is used in the request sent to the native service.
SOAP Optimization Method	This is applicable for SOAP-based APIs. Specifies the optimization methods that API Gateway can use to parse SOAP requests to the native API. Select one of the following options: ■ **MTOM**. API Gateway uses the Message Transmission Optimization Mechanism (MTOM) to parse SOAP requests to the API.

Property	Description
	■ **SwA**. API Gateway uses the SOAP with Attachment (SwA) technique to parse SOAP requests to the API. ■ **None**. API Gateway does not use any optimization method to parse the SOAP requests to the API. This is selected by default.
HTTP Connection Timeout (seconds)	Specifies the time interval (in seconds) after which a connection attempt times out. The precedence of the Connection Timeout configuration is as follows: 1. If you specify a value for the **Connection timeout** field in routing endpoint alias, then the **Connection timeout** value specified in the Endpoint alias section takes precedence over the timeout values defined at the API level and the global level. 2. If you specify a value 0 for the **Connection timeout** field in routing endpoint alias, then API Gateway uses the value specified in the **Connection timeout** field in the routing protocol processing step of an API. The Read Timeout value specified at an API level takes precedence over the global configuration. 3. If you specify a value 0 or do not specify a value for the **Connection timeout** field in the routing protocol processing step at the API level or specify a value 0 at an alias level, then API Gateway uses the value specified in this `pg.endpoint.connectionTimeout` property. 4. If you do not specify any value for `pg.endpoint.connectionTimeout`, then API Gateway uses the default value of 30 seconds.
Read Timeout (seconds)	Specifies the time interval (in seconds) after which a socket read attempt times out. The precedence of the Read Timeout configuration is as follows: 1. If you specify a value for the **Read timeout** field in routing endpoint alias, then the **Read timeout** value specified in the Endpoint alias section takes precedence over the timeout values defined at the API level and the global level. 2. If you specify a value 0 for the **Read timeout** field in routing endpoint alias, then API Gateway uses the value specified in the **Read Timeout** field in the routing protocol processing step of an API. The Read Timeout value specified at an API level takes precedence over the global configuration.

Property	Description
	3. If you specify a value 0 or do not specify a value for the **Read timeout** field in the routing protocol processing step at the API level or specify a value 0 at an alias level, then API Gateway uses the value specified in this `pg.endpoint.readTimeout` property. 4. If you do not specify any value for `pg.endpoint.readTimeout`, then API Gateway uses the default value of 30 seconds.
Suspend duration (seconds)	A numeric timeout value (in seconds). The default value is 30. This property specifies the time, in seconds, for which API Gateway temporarily suspends an endpoint, whenever Read time-out or Connection time-out occurs for the endpoint, and routes the request to the next configured endpoint in this time interval. For example: If you have 3 endpoints configured endpoint #1, endpoint #2, and endpoint #3, the suspend duration is configured as 60 seconds for endpoint #2, and there is a Read Timeout or Connection Timeout for endpoint #2, then API Gateway temporarily suspends endpoint #2 for 60 seconds. In this time interval API Gateway skips endpoint #2 while routing the requests to the configured endpoints. Request 1 -> endpoint #1 Request 2 -> endpoint #3 (endpoint #2 is suspended for 60 seconds and hence the request is sent to endpoint #3 Request 3 -> endpoint #1
Pass WS-Security Headers	This is applicable for SOAP-based APIs. Selecting this indicates that API Gateway should pass the WS-Security headers of the incoming requests to the native API.
SSL Configuration. Configures keystore, key alias, and truststore for securing connections to native APIs.	
Keystore Alias	Specifies the keystore alias configured in API Gateway. This value (along with the value of Client Certificate Alias) is used for performing SSL client authentication. Lists all available keystores. If you have not configured any keystore, the list is empty.
Key Alias	Specifies the alias for the private key, which must be stored in the keystore specified by the above keystore alias.

Property	Description
Truststore Alias	Specifies the alias for the truststore that contains the list of CA certificates that API Gateway uses to validate the trust relationship with the native API. If you do not configure any truststore alias, it implies that API Gateway does not validate the certificates provided by native APIs.
Service Registry Configuration	
Service Discovery Endpoint Parameter	Values required for constructing the discovery service URI. ■ **Parameter**: An alias that you have included in the discovery service URI while adding the service registry to API Gateway. ■ **Value**: Specifies a value with a syntax. The alias specified in Path Parameter is substituted with this value when invoking the discovery service. For example: if the service registry configuration of the service registry that you have selected in **Endpoint URI** has **Service discovery path** set to `/catalog/service/{serviceName}` (and the `{serviceName}` alias is intended for passing the service name), you must enter `{serviceName}` as **Parameter** and the name of the service as **Value**. As the **Value** field supports variable framework, you can make use of the available variables as path parameters. For example, if you provide a parameter as `{serviceName}`(in **Service discovery path** while you define a service register) and the corresponding values for the path parameter as `${request.header.var1}`, the value in var 1 retrieved from the request header will substitute the service name. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

Failover behavior during load balancing

When an endpoint that is configured in Load balancer routing returns any of these exceptions - ConnectException, MalformedURLException, NoRouteToHostException, ProtocolException, SocketTimeoutException, UnknownHostException, UnknownServiceException - then API Gateway treats the endpoint to be inactive and routes to the next endpoint as per the round-robin strategy. In this case, the endpoint is suspended for the duration mentioned in the `suspendDuration` parameter (default is 30s), which indicates the duration to suspend the endpoint without repeatedly trying to reach it.

In this way API Gateway tries to invoke all the endpoints configured in the load balance routing. If all endpoints return downtime error, API Gateway returns a `Service is down` error.

If an endpoint returns an exception other than the Downtime exception then that exception is sent to the client and the remaining endpoints are not invoked.

You can control the behavior of considering Downtime exceptions only for load balancing through the extended property `pg.lb.failoverOnDowntimeErrorOnly`, which you can set through **Administration** > **General** > **Extended settings** page. The default value of this property is `true`. If you set the value to `false` all failures from the endpoint are treated as downtime and load balancing takes place.

Outbound Auth - Message

When the native API is protected and expects the authentication credentials to be passed through payload message, you can use this policy to provide the credentials that is added to the request and sent to the native API. API Gateway supports a wide range of authentication schemes, such as WSS Username, SAML, and Kerberos, in addition to signing and encryption at the message-level.

Note:
Message-level authentication can be used to secure outbound communication of only SOAP APIs.

The table lists the properties that you can specify for this policy:

Property	Description
Authentication scheme	Select one of the following schemes for outbound authentication at the message level: ■ **WSS username**. Uses WSS credentials authenticate the client. ■ **SAML**. Uses SAML issuer configuration details for authentication. ■ **Kerberos**. Uses Kerberos credentials for authentication. ■ **None**. Authenticates the client without any authentication schemes. ■ **Alias**. Uses the configured alias name for authentication. ■ **Remove WSS headers**. Uses the WSS headers for authentication.
Authenticate using	Select one of the following modes to authenticate the client: ■ **Custom credentials**. Uses the values specified in the policy to obtain the required token to access the native service. ■ **Incoming HTTP Basic Auth credentials**. Uses the incoming user credentials to retrieve the authentication token to access the native API

Property	Description
	■ **Delegate incoming credentials**. Uses the values specified in the policy by the API providers to select whether to delegate the incoming token or act as a normal client.
WSS username	Uses the WSS credentials to authenticate the client. Provide the following credentials: ■ **User Name**. Specifies the user name. ■ **Password**. Specifies the password of the user.

Kerberos	Uses the Kerberos credentials to authenticate the client.
	Provide the following information:
	■ **Client principal**. Provide a valid client LDAP user name.
	■ **Client password**. Provide a valid password of the client LDAP user.
	■ **Service principal**. Provide a valid SPN. The specified value is used by the client to obtain a service ticket from the KDC server.
	■ **Service Principal Name Form**. The SPN type to use while authenticating an incoming client principal name. Select any of the following:
	  ■ **User name**. Specifies the username form.
	  ■ **Hostbased**. Specifies the host form.

SAML	Provide the SAML issuer that is configured.
Signing Configurations	Uses the signing configuration details to authenticate the client. Provide the following information: ■ **Keystore Alias**. Specifies a user-specified text identifier for an API Gateway keystore. The alias points to a repository of private keys and their associated certificates. ■ **Key Alias**. Specifies the alias for the private key, which must be stored in the keystore specified by the keystore alias.
Encryption Configurations	Uses the encryption configuration details to authenticate the client. Provide the following information: ■ **Truststore alias**. Specifies the alias for the truststore. The truststore contains the trusted root certificate for the CA that signed the API Gateway certificate associated with the key alias.

Property	Description
	■ **Certificate alias**. Provide a text identifier for the certificate associated with the truststore alias. API Gateway populates the certificate alias list with the certificate aliases from the selected truststore alias.
Alias	Uses the configured alias to authenticate the client. Provide the name of the configured alias.
Stage	Specify a stage, if you want the configuration to be applicable to a specific stage.

When you configure an API with an inbound authentication policy, and a client sends a request with credentials, API Gateway uses the credentials for the inbound authentication. When sending the request to native server, API Gateway removes the already authenticated credentials when no outbound authentication policy is configured.

If as an API provider you want to use the same credentials for authentication at both API Gateway and native server, you should configure the outbound authentication policy to pass the incoming credentials to the native service. If you do not configure an outbound authentication policy, API Gateway removes the incoming credentials, as it is meant for API Gateway authentication only.

However, when both the inbound authentication policy and outbound authentication policy are not configured, API Gateway just acts as a proxy and forwards the credentials to the native service. Since the credentials are not meant for API Gateway (as no inbound auth policy is configured), API Gateway forwards the credentials to native service (unless there are different settings configured in outbound authentication policy, for example, custom credentials or Anonymous).

JMS/AMQP Policies

To configure API Gateway for JMS with Message broker native protocol support or JMS with AMQP protocol you need to:

- Create one or more JNDI provider aliases to specify where API Gateway can look up when it needs to create a connection to JMS provider or specify a destination for sending or receiving messages.
- Create one or more connection aliases that encapsulate the properties that API Gateway needs to create a connection with the JMS provider.

JMS/AMQP Routing

You can use this policy when you want to specify a JMS queue or topic to which API Gateway submits the request, and the destination where the response should be routed to where API Gateway waits to listen to the response from the native API.

For example, you can use this policy when you have a native API that is exposed over AMQP or JMS and that requires clients to communicate with the server using other protocols. This policy allows you to bridge protocols between the client and the native API.

You can apply the JMS/AMQP routing policy to both REST and SOAP APIS. The following sections explain their usage.

Use case 1: Using the JMS/AMQP routing policy (JMS with a message broker native protocol) for a SOAP API

This describes the high level workflow for the scenario where you use the JMS/AMQP routing policy (JMS with a message broker native protocol) for a SOAP API.

1. Create an alias to a JNDI Provider For a detailed procedure, see *webMethods API Gateway Administration*.
2. Configure API Gateway to use a JMS connection alias to establish an active connection between API Gateway and the JMS provider. For a detailed procedure, see *webMethods API Gateway Administration*.
3. Create a WS (Web Service) JMS Provider endpoint alias and configure the Alias, Description, Type (Provider), Transport Type (JMS) fields and JMS Transport Properties. For a detailed procedure, see *webMethods API Gateway Administration*.
4. A WS (Web Service) endpoint trigger is created when you configure WS (Web Service) JMS Provider endpoint alias. This trigger consists of the input source details like Queue name or Topic name. You can update the WS (Web Service) endpoint trigger, as required. For detailed procedures, see *webMethods API Gateway Administration*.
5. Select the required API.
6. Click **Edit**.
7. In the API Details section click **Policies**.
8. Enforce the **JMS/AMQP SOAP Routing** policy with the following properties configured.
   a. Specify the connection URL for connecting to the JMS provider.
   b. Specify a queue name where a reply to the message must be sent.
   c. Provide a priority of this JMS message.
   d. Provide expiration time of the JMS message.
   e. Specify the message delivery mode for the request message.

   For details on the **JMS/AMQP SOAP Routing** policy, see " JMS/AMQP SOAP Routing" on page 301.
9. Click Save.

   The enforced policy **JMS/AMQP SOAP Routing** with the required configuration now allows any java client to communicate with the API asynchronously.

Use case 2: Using the JMS/AMQP routing policy (JMS with AMQP protocol) for a SOAP API

This describes the high level workflow for the scenario where you use the JMS/AMQP routing policy (JMS with AMQP protocol) for a SOAP API.

1. Configure API Gateway to use JMS with AMQP protocol.

   Configure JNDI settings and JMS settings as per the client you are using to achieve JMS over AMQP protocol support.

   Note:
   For a sample procedure on configuring API Gateway to use JMS with AMQP protocol using Apache qpid libraries, see "Configuring API Gateway for JMS with AMQP Protocol" on page 106.

2. Select the required API.
3. Click **Edit**.
4. In the API Details section click **Policies**.
5. Enforce the **JMS/AMQP SOAP Routing** policy with the following properties configured.
   a. Specify the connection URL for connecting to the JMS provider.
   b. Specify a queue name where a reply to the message must be sent.
   c. Provide a priority for this AMQP message.
   d. Provide expiration time of the AMQP message.
   e. Specify the message delivery mode for the request message.

   For details on the **JMS/AMQP SOAP Routing** policy, see " JMS/AMQP SOAP Routing" on page 301.

6. Click **Save**.

   The enforced policy **Enable JMS/AMQP** with the required configuration now allows all the clients such as Python, Ruby, Java, and Dotnet to communicate with the API asynchronously.

Use case 3: Using the JMS/AMQP routing policy (JMS with a message broker native protocol) for a REST API

This describes the high level workflow for the scenario where you use the JMS/AMQP routing policy (JMS with a message broker native protocol) for a REST API.

1. Create an alias to a JNDI Provider For a detailed procedure, see *webMethods API Gateway Administration*.

2. Configure API Gateway to use a JMS connection alias to establish an active connection between API Gateway and the JMS provider. For a detailed procedure, see *webMethods API Gateway Administration*.
3. Select the required API.
4. Click **Edit**.
5. In the API Details section click **Policies**.
6. Enforce the **JMS/AMQP REST Routing** policy with the following properties configured.
   a. Specify the connection alias that contains the configuration information needed to establish a connection to a specific JMS provider.
   b. Specify the destination to which the request message is sent.
   c. Specify the destination type to which the request message is sent.
   d. Specify the destination to which the response message is sent.
   e. Specify the type of destination, queue or topic, to which the response message is sent.
   f. Provide expiration time of the JMS message.
   g. Provide the time for which API Gateway listens for the response message.
   h. Specify the message delivery mode for the request message.

   For details on the **JMS/AMQP REST Routing** policy, see "JMS/AMQP REST Routing" on page 304.
7. Click **Save**.

   The enforced policy **JMS/AMQP REST Routing** with the required configuration now allows any java client to communicate with the API asynchronously.

Use case 4: Using the JMS/AMQP routing policy (JMS with AMQP protocol) for a REST API

This describes the high level workflow for the scenario where you use the JMS/AMQP routing policy (JMS with AMQP protocol) for a REST API.

1. Configure API Gateway to use JMS with AMQP protocol.

   Configure JNDI settings and JMS settings as per the client you are using to achieve JMS over AMQP protocol support.

   Note:
   For a sample procedure on configuring API Gateway to use JMS with AMQP protocol using Apache qpid libraries, see "Configuring API Gateway for JMS with AMQP Protocol" on page 106.
2. Select the required API.

3. Click **Edit**.
4. In the API Details section click **Policies**.
5. Enforce the **JMS/AMQP REST Routing** policy with the following properties configured.
   a. Specify the connection alias that contains the configuration information needed to establish a connection to a specific JMS provider.
   b. Specify the destination to which the request message is sent.
   c. Specify the destination type to which the request message is sent.
   d. Specify the destination to which the response message is sent.
   e. Specify the type of destination, queue or topic, to which the response message is sent.
   f. Provide expiration time of the AMQP message.
   g. Provide the time for which API Gateway listens for the response message.
   h. Specify the message delivery mode for the request message.

   For details on the **JMS/AMQP REST Routing** policy, see "JMS/AMQP REST Routing" on page 304.
6. Click **Save**.

   The enforced policy **Enable JMS/AMQP** with the required configuration now allows all the clients such as Python, Ruby, Java, and Dotnet to communicate with the API asynchronously.

JMS/AMQP SOAP Routing

If you are using JMS with Message broker native protocol support ensure that following actions are performed before using the Enable JMS/AMQP policy:

- Create an alias to a JNDI Provider For a detailed procedure, see *webMethods API Gateway Administration*.
- Configure API Gateway to use a JMS connection alias to establish an active connection between API Gateway and the JMS provider. For a detailed procedure, see *webMethods API Gateway Administration*.
- Create a WS (Web Service) JMS Provider endpoint alias and configure the Alias, Description, Type (Provider), Transport Type (JMS) fields and JMS Transport Properties. For a detailed procedure, see *webMethods API Gateway Administration*.
- Configure a WS (Web Service) endpoint trigger. For detailed procedures, see *webMethods API Gateway Administration*.

If you are using JMS with AMQP protocol support, ensure the following before using the Enable JMS/AMQP policy:

- You have configured API Gateway for JMS with AMQP. For details, see "Configuring API Gateway for JMS with AMQP Protocol" on page 106.

The table lists the properties that you can specify for this policy:

Property	Description
Connection URL	Provide a connection alias for connecting to the JMS provider (for example, an Integration Server alias or a JNDI URL). The connection URL contains various elements that construct the destination and other connection specific parameters. The structure of the connection URL is: `<protocol>:<lookupVariant>:<destination>?<parameters>` where ■ *protocol*. Specify the name of the transport protocol. The default value is JMS. ■ *lookupVariant*. Specify the destination type such as queue or topic. The default value is queue. ■ *destination*. Specify the destination name of the JMS Provider. For dynamic queue the destination name is: `dynamicQueues/<Queue name>` ■ *Parameters* ■ `wm-wsendpointalias`. Specify the JMS consumer endpoint alias. This parameter is required for API Gateway to look up the JMS consumer alias and send the request to the specified queue. ■ `jndiInitialContextFactory`. Specify the initial context factory for the JNDI look up. For example: `org.apache.activemq.jndi.ActiveMQInitialContextFactory` for ActiveMQ ■ `jndiConnectionFactoryName`. Specify the connection factory look up name. For example: ■ `ConnectionFactory` for ActiveMQ if you are using the JMS with broker native protocol. ■ `qpidConnectionFactory` for ActiveMQ if you are using the JMS with AMQP protocol. ■ `jndiURL`. Specify the Provider URL for the Active MQ to connect to API Gateway. For example: ■ `tcp://vmmeddemo03:61616` for ActiveMQ if you are using the JMS with broker native protocol. ■ The file path location of the properties file, for example, Install directory`\IntegrationServer\lib\jars\amqp.properties` if you are using JMS with AMQP protocol. ■ `targetService`. Specify the API Gateway API name. This parameter is required if you are sending the request to another API in API Gateway that uses JMS as the entry protocol. Sample: With consumer endpoint alias

Property	Description

	`jms:queue:dynamicQueues/MyTestQueue?`
	`wm-wsendpointalias=JMSConsumerEndpointAlias&target`
	`Service=EchoS_VS_JMS_IN`
	
	Sample: With JNDI lookup parameters
	
	`jms:queue:dynamicQueues/MyTestQueue?`
	`jndiConnectionFactoryName=ConnectionFactory`
	`&jndiInitialContextFactory=org.apache.`
	`activemq.jndi.ActiveMQInitialContextFactory`
	`&targetService=EchoS_VS_JMS_IN`
	
	Sample: With JNDI lookup parameters for AMQP protocol
	
	`jms:queue:dynamicQueues/MyTestQueue?`
	`jndiConnectionFactoryName=qpidConnectionFactory`
	`&jndiInitialContextFactory=org.apache.qpid.jms.`
	`jndi.JmsInitialContextFactory`
	`&targetService=EchoS_VS_JMS_IN`

Reply To Destination	Specify a queue name where a reply to the message must be sent.
Priority	Type an integer that represents the priority of this JMS or AMQP message with respect to other messages that are in the same queue. The priority value determines the order in which the messages are routed. The lowest priority value is 0 and the highest priority value is 9. The messages with this priority value are executed first. ■ Priority values 0 through 9. ■ The default priority for a JMS or AMQP message is 0.
Time to Live (ms)	Provide a numeric value that specifies the expiration time (in milliseconds) of the JMS or AMQP message. If the time-to-live is specified as zero, expiration is set to zero which indicates the message does not expire. The default value is 0.
Delivery Mode	The message delivery mode for the request message. This is the delivery mode that web service clients must specify in the JMS or AMQP message that serves as the request message for the web service. The available options are: ■ **Non-persistent**. Indicates that the request message is not persistent. The message might be lost if the JMS provider fails. ■ **Persistent**. Indicates that the request message should be persistent. The message is not lost if the JMS provider fails.

JMS/AMQP REST Routing

If you are using JMS with Message broker native protocol support ensure that following actions are performed before using the Enable JMS/AMQP policy:

- Create an alias to a JNDI Provider For a detailed procedure, see *webMethods API Gateway Administration*.
- Configure API Gateway to use a JMS connection alias to establish an active connection between API Gateway and the JMS provider. For a detailed procedure, see *webMethods API Gateway Administration*.

If you are using JMS with AMQP protocol support, ensure the following before using the Enable JMS/AMQP policy:

- You have configured API Gateway for JMS with AMQP. For details, see "Configuring API Gateway for JMS with AMQP Protocol" on page 106.

The table lists the properties that you can specify for this policy:

Property	Description
Connection Alias Name	Specifies the name of the connection alias. Each connection alias contains the configuration information needed to establish a connection to a specific JMS provider.
Destination Name	Specify the name of the destination to, which the request message is sent. As this property supports variable framework, you can use the available variables to specify the destination name. For example, you can provide a destination name as `${request.header.var1}`. The destination name used in `var1` is where the Queue or Topic that is created in Universal Messaging stores the events. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
Destination Type	Specify the destination type to which the request message is sent.
Reply To Name	Specify the name of the destination to, which the response message is sent. As this property supports variable framework, you can use the available variables to specify the destination name. For example, you can provide a destination name as `${request.header.dest1}`. The destination name used in `dest1` is the Queue or Topic that is created dynamically in Universal Messaging. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

Property	Description
Reply To Type	Specifies the type of destination to which the response message is sent. Select one of the following source type: ■ **QUEUE**. Indicates that the response message is sent to a particular queue. ■ **TOPIC**. Indicates that the response message is sent to a particular topic.
Time to Live (ms)	Provide a numeric value that specifies the expiration time (in milliseconds) of the JMS or AMQP message. If the time-to-live is specified as zero, expiration is set to zero which indicates the message does not expire. The default value is 0.
Time to Wait (ms)	Defines the time in milliseconds for which API Gateway listens to the Reply To Queue or Topic for the response message.
Delivery Mode	The message delivery mode for the request message. This is the delivery mode that web service clients must specify in the JMS or AMQP message that serves as the request message for the web service. The available options are: ■ **Non-persistent**. Indicates that the request message is not persistent. The message might be lost if the JMS provider fails. ■ **Persistent**. Indicates that the request message should be persistent. The message is not lost if the JMS provider fails.

JMS/AMQP Properties

The JMS/AMQP Properties policy can be configured to set AMQP or JMS Properties, a few standard AMQP or JMS Headers, and HTTP Transport Headers in the outgoing JMS message that is being sent from the proxy API to the native API.

AMQP or JMS headers are part of the JMS message that are used by both clients and providers. They are used to identify a message and to route the message to the applicable JMS Providers or consumers.

You can add HTTP Headers such as API Key, Authorization header, and so on. This is useful when the native API is configured with the Enable AMQP/JMS policy and the proxy API wants to pass the security headers over to that native API.

Every JMS message includes JMS/AMQP properties that are always passed from provider to client. The purpose of the properties is to convey extra information to the client outside the normal content of the message body. Additionally, JMS/AMQP property values are set exclusively by the consumer application. When a client receives a message, the properties are in read-only mode. If a client tries to modify any of the properties, a MessageNotWriteableException occurs.

The properties are standard Java name or value pairs. The property names must conform to the message selector syntax specifications defined in the message interface. Property fields are most often used for message selection and filtering. By using a property field, a message consumer can interrogate the property field and perform message filtering and selection. When this action is configured for a proxy API, API Gateway uses the JMS or AMQP properties to authenticate client requests before submitting to the native APIs. JMS or AMQP headers can also be set using properties, however, JMS or AMQP properties take precedence over headers.

The JMS/AMQP properties section has separate policies that you can configure for REST and SOAP APIs. They are as follows:

- JMS/AMQP REST Properties
- JMS/AMQP SOAP Properties

The table lists the properties that you can specify for this policy:

Property	Description
JMS Property Key	Specify the JMS property key.
JMS Property Value	Specify the JMS property value for the specified key.

As both these properties support variable framework, you can use the available variables to specify the JMS property key and value.

For example, if you provide a property key as `${request.header.token1}` and the corresponding property value as `${request.header.token2}`, then the value in `token1` and `token2` passes security headers to the native API.

For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

Predefined JMS Properties

Property categories	Property	Description
Run-time settings	- `jms.deliveryMode` - `jms.priority` - `jms.timeToLive` - `jms.messageType`	If the `jms.messageType` is set to `TextMessage`, the SOAP envelope in the request is sent as a text message to the JMS queue instead of byte stream.
Standard JMS headers	- `JMSType` - `JMSCorrelationID` - `JMSXGroupID`	The following headers are not applicable. If they are added an error response would be sent at runtime: - `JMSMessageID`

Property categories	Property	Description
	■ JMSXGroupSeq	■ JMSExpiration ■ JMSRedelivered ■ JMSTimestamp ■ JMSDeliveryMode ■ JMSPriority ■ JMSReplyTo ■ JMSDestination
Application specific properties	■ SOAPJMS_requestURI ■ SOAPJMS_bindingVersion ■ SOAPJMS_soapAction ■ SOAPJMS_targetService ■ SOAPJMS_contentType	

Mapping AMQP messages to JMS

Header

Field name	Description
durable	When receiving a message, the `durable` field of header MUST be mapped to the `JMSDeliveryMode` header of the Message. If the `durable` field of header is set to `false` or is not set then the `JMSDeliveryMode` MUST be taken to be NONPERSISTENT. When the `durable` field of header is set to `true` the `JMSDeliveryMode` of the Message MUST be taken to be PERSISTENT.
priority	This field is mapped to the `JMSPriorityheader` of the Message. JMS Priority is specified as being of type `int` despite the valid values only being 0-9. AMQP allows for the `priority` field of header to be any valid `ubyte` value. When receiving a message with the `priority` field of header greater than 9, the `JMSPriority` MUST be set to 9. If the `priority` field of header is unset then the `JMSPriority` MUST be taken to be DEFAULT_PRIORITY that is, the value 4).
ttl	This field defines the number of milliseconds for which a given message is considered live. There is no direct equivalent for the ttl field of header in the JMS specification.

Field name	Description
	If and only if the `absolute-expiry-time` field of properties is not set,`JMSExpiration` SHOULD be based on the `ttl` field of header if set, by summing it with the current time in milliseconds since the Unix Epoch
first acquirer	This field does not have a direct equivalent within the JMS specification, although `JMSRedelivered` is related, and so vendor property JMS_AMQP_FIRST_ACQUIRER SHOULD be used.
delivery-count	This field is mapped to the JMS-defined `JMSXDeliveryCount` property and `JMSRedelivered` header of the Message as follows. AMQP uses the `delivery-count` field of header to track previously failed delivery attempts for a message, with the first delivery attempt having a value of zero, and soon. `JMSXDeliveryCount` is defined as a Java `int` count of delivery attempts, set by the provider on receive, where the first delivery attempt has value 1, the second has value 2 and so on. The value of `JMSXDeliveryCount` property is thus equal to `delivery-count + 1`. The `JMSRedelivered` header MUST be considered to be true if and only if the delivery-count field of header has a value greater than 0.

Properties

Field name	Description
message-id	This field is equivalent to the JMSMessageID header of the Message. The `JMSMessageID` value is a Java `String` where as the `message-id` field of properties is defined as being of type providing `message-id`, that is `message-id-ulong`, `message-id-uuid`, `message-id-binary` or `message-id-string`. The JMS client library MUST prefix **ID:** to the value of the `message-id` field of properties before returning it as the `JMSMessageID` value.
user-id	This field is mapped to the JMS-defined `JMSXUserID` property of the Message. `JMSXUserID` is specified as being of type `String`, while the `user-id` field of `properties` field is specified as type `binary`. To maintain end-to-end fidelity for this property implementations SHOULD convert between AMQP `binary` and Java `String` by using the UTF-8 Unicode[UNICODE63] character encoding.

Field name	Description
to	This field is mapped to the `JMSDestination` header of the Message. `JMSDestination` is defined as being of the JMS `Destination` type, while the `to` field of properties requires an `address-string`. If the `to` field of properties was not set on a received message, the `JMSDestination` header value SHOULD be derived from the Destination to which the receiving consumer was established.
subject	This field is mapped to the `JMSType` header of the Message.
reply-to	This field is mapped to the JMSReplyTo header of the Message. `JMSReplyTo` is defined as being of the `JMSDestination` type, while the `reply-to` field of properties requires an `address-string`.
correlation-id	This field is mapped to the `JMSCorrelationID` header of the Message. The `JMSCorrelationID` value is a Java `String` where as the `correlation-id` field of properties is defined as being of type providing `message-id`, that is `message-id-ulong`, `message-id-uuid`, `message-id-binary` or `message-id-string`. Where the `correlation-id` field of properties for the received message is of type `message-id-string` and the `boolean` message annotation with `symbol` key of `x-opt-app-correlation-id` is either not set or is `false`, then the `correlation-id` field of properties MUST be formatted as a `JMSMessageID`, that is the client library MUST prefix **ID:** to the value before returning it as the `JMSCorrelationID` value.
content-type	This field does not have an equivalent within the JMS specification, and so the vendor property JMSAMQPCONTENTTYPESHOULD be used.
content-encoding	This field does not have an equivalent within the JMS specification, and so the vendor property JMSAMQPCONTENTENCODINGSHOULD be used.
absolute-expiry-time	This field is mapped to the `JMSExpiration` head of the Message. If the `absolute-expiry-time` field of properties is set, then `JMSExpiration` MUST have the equivalent Java `long` value, representing the time at which the message expires, in milliseconds since the Unix Epoch. If the `absolute-expiry-time` field of properties is not set then `JMSExpiration` SHOULD be based on the `ttl` field of header instead if set.
creation-time	This field is mapped to the `JMSTimestamp` header of the Message. If the `creation-time` field of properties is not set, then `JMSTimestamp` MUST have the value zero. If the `creation-time` field of properties

Field name	Description
	field is set, then `JMSTimestamp` MUST have the equivalent Java `long` value, representing the time at which the message was sent or created, in milliseconds since the Unix Epoch.
group-id	This field is mapped to the JMS-defined `JMSXGroupID` property of the Message.
group-sequence	This field is mapped to the JMS-defined `JMSXGroupSeq` property of the Message. As the `group-sequence` field of properties is an `uint` and `JMSXGroupSeq` is an `int`, `group-sequence` values in the range 2^{31} to 2^{32}-1 inclusive MUST be mapped to `JMSXGroupSeq` values in the range -2^{31} to -1 inclusive.
reply-to-group-id	This field does not have an equivalent within the JMS specification, and so the vendor property JMS_AMQP_REPLY_TO_GROUP_ID MUST be used.

For more information on AMQP properties and JMS to AMQP mapping properties, see https://www.oasis-open.org/committees/download.php/56418/amqp-bindmap-jms-v1.0-wd06.pdf.

Traffic Monitoring

The policies in this stage provide ways to enable logging request and response payload, enable monitoring run-time performance conditions for APIs and applications, enforce limits for the number of service invocations during a specified time interval and send alerts to a specified destination when the performance conditions are violated, and enable caching of the results of API invocations depending on the caching criteria defined. The policies included in this stage are:

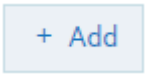
- Log Invocation
- Monitor Performance
- Monitor SLA
- Traffic Optimization
- Service Result Cache

Log Invocation

This policy enables logging requests or responses to a specified destination. This action also logs other information about the requests or responses, such as the API name, operation name, the Integration Server user, a timestamp, and the response time.

The table lists the properties that you can specify for this policy:

Property	Description
Store Request Headers	Logs all request headers.
Store Request Payload	Logs all request payloads.
Store Response Headers	Logs all response headers.
Store Response Payload	Logs all response payloads.
Compress Payload Data	Compresses the logged payload data. For details about payload compression and how to uncompress a payload, see "Uncompressing a payload" on page 312.
Log Generation Frequency	Specifies how frequently to log the payload. Select one of the following options: ■ **Always**. Logs all requests and responses. ■ **On Failure**. Logs only the failed requests and responses. ■ **On Success**. Logs only the successful responses and requests.
Destination	Specifies the destination where to log the payload. Select the required options: ■ **API Gateway** ■ **API Portal** ■ **Audit Log** Audit log destination can be configured as *DB* or *File* in the **Administration** > **Destinations** screen. Software AG recommends to use *DB* when you choose Audit Log as the destination to log transactions through Log Invocation policy. If you choose *File*, warnings appear in the log file since a few of the transaction log fields are not compatible with Audit log file destination such as BLOB types. For more information, see **Configure Audit Logging** section in *webMethods Audit Logging Guide*. ■ **CentraSite** **Note:** This option is applicable only for the APIs published from CentraSite to API Gateway. ■ **Digital Events**

Property	Description
	■ **Elasticsearch** ■ **Email** (you can add multiple email addresses by clicking ). **Note:** If an email alias is available, you can type the email alias in the **Email Address** field with the following syntax, ${emailaliasname}. For example, if test is the email alias, then type ${test}. ■ **JDBC** ■ **Local Log**: You can select the severity of the messages to be logged (logging level) from the **Log Level** drop-down list. The available log levels are ERROR, INFO, and WARN. **Note:** ■ Set the Integration Server Administrator's logging level for API Gateway to match the logging levels specified for the run-time actions (go to **Settings** > **Logging** > **Server Logger**). For example, if a Log Invocation action is set to the logging level of Error, you must also set Integration Server Administrator's logging level for API Gateway to Error. If the action's logging level is set to a low level (Warning-level or Information level), but Integration Server Administrator's logging level for API Gateway is set to a higher level (Error-level), then only the higher-level messages are written to the log file. ■ Entries posted to the local log are identified by a product code of YAI and suffixed with the initial alphabet of the logging level selected. For example, for an error level, the entry appears as [YAI.0900.0002E]. ■ **SNMP** ■ List of destinations configured using the **Custom destinations** section. For details on publishing to custom destinations, see *webMethods API Gateway Administration*.

Uncompressing a payload

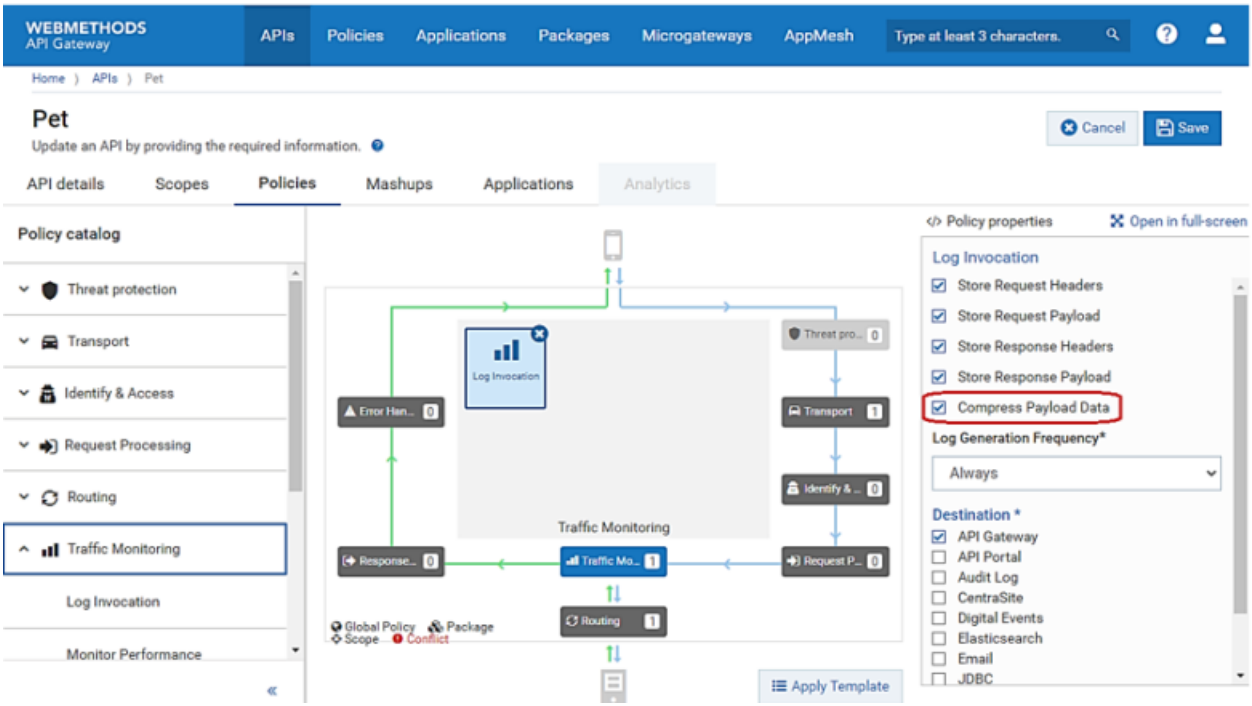
Payload compression helps you to optimize the storage by reducing the size of the actual payload. It improves the performance while rendering the analytics information in the dashboard.

The request and response payload of the API Gateway API and native API is compressed in the encoded form.

To generate the data and uncompress the payload.

1. Ensure you have an API enforced with a **Log invocation policy** with the property **Compress payload data** selected.

   See the following example where an API is enforced with a Log invocation policy with Compress payload data selected.



2. Invoke the same API using an external REST client such as Postman or SoapUI to see the API transaction.

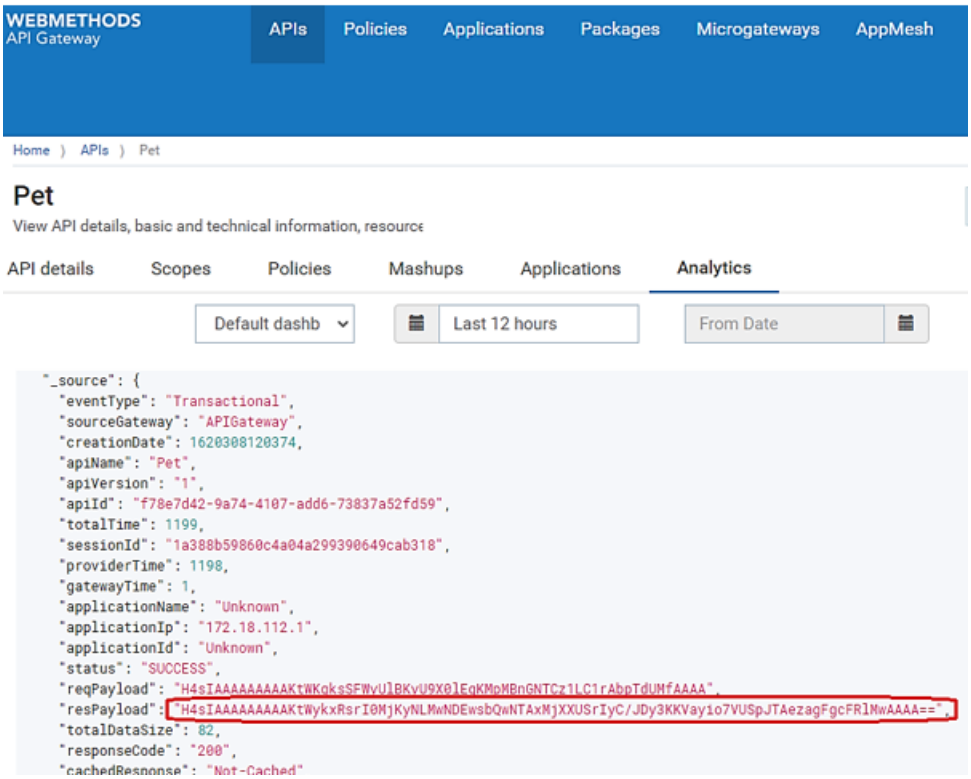
   TransactionalEvent is generated every time an API invocation happens.

3. Click **Analytics** of the same API in API Gateway UI.

   This displays the different types of events generated in the dashboard. For details about analytics, see *webMethods API Gateway Administration*.

4. Select **Runtime events** and click to expand your transaction.

5. Click **JSON** or **Table** and copy the encoded string (value) of the request or response payload that you want to uncompress.



6. Pass the copied string as an *input* to the following Java program.

```
public static String uncompressString(String zippedBase64Str)  throws IOException
 {
String unCompressedPayload = null;
byte[] bytes = Base64.getDecoder().decode(zippedBase64Str);
GZIPInputStream zi = null;
try{
zi = new GZIPInputStream(new ByteArrayInputStream(bytes));
unCompressedPayload = IOUtils.toString(zi);
}finally{
IOUtils.closeQuietly(zi);
}
return unCompressedPayload;
}
```

See the following example, where an encoded string from the request payload is passed as an *input* to the Java program.

```
import org.apache.commons.io.IOUtils;



import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.util.Base64;
import java.util.zip.GZIPInputStream;


public class MyClass {
    public static void main(String[] args) throws IOException {
        String str = "H4sIAAAAAAAAAKtWykxRsrI0MjKyNLMwNDEwsbQwNTAxMjXXUSrIyC/JDy3KKVayio7VUSpJTAezagFgcFRlMwAAAA==";
        System.out.println(uncompressString(str));
    }


    public static String uncompressString(String zippedBase64Str) throws IOException {
        String unCompressedPayload = null;
        byte[] bytes = Base64.getDecoder().decode(zippedBase64Str);
        GZIPInputStream zi = null;
        try{
            zi = new GZIPInputStream(new ByteArrayInputStream(bytes));
            unCompressedPayload = IOUtils.toString(zi);
        }finally{
            IOUtils.closeQuietly(zi);
        }
        return unCompressedPayload;
    }
}
```

Input

```
{"id":9222968140498504257,"photoUrls":[],"tags":[]}
```

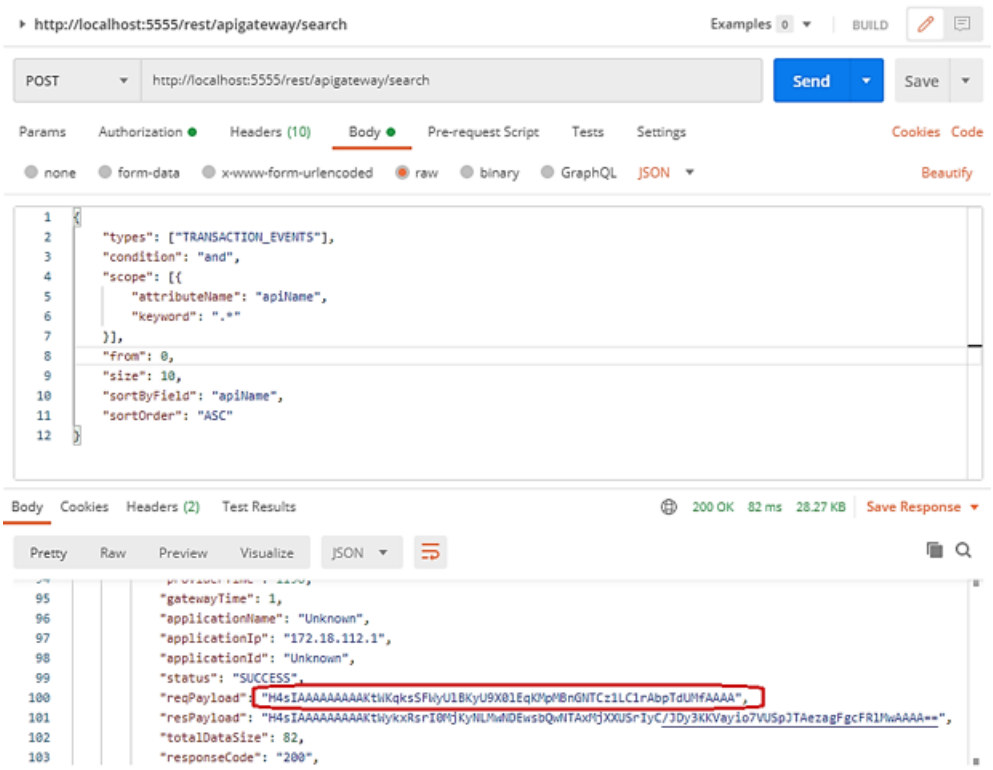
Output

The Java *output* contains the uncompressed payload.

Note:
This code snippet is applicable only for the payload compressed by the log invocation policy.

You can also query the data using the REST endpoints from the swagger file APIGatewaySearch.json and uncompress the payload with the same code snippet.

For details about the REST endpoints, see *webMethods API Gateway Developer's Guide*.



Traffic Optimization

This policy limits the number of API invocations during a specified time interval, and sends alerts to a specified destination when the performance conditions are violated. You can use this policy to avoid overloading the back-end services and their infrastructure, to limit specific clients in terms of resource usage, and so on.

The Traffic optimization policy generates two types of events when the specified limit is breached:

- **Policy violation event**. Indicates the violations that occur for an API. If there are 100 violations, then 100 policy violation events are generated.
- **Monitor event**. Controlled by the alert frequency configuration specified in the policy.

The table lists the properties that you can specify for this policy:

Property	Description
Limit Configuration	
Rule name	Specifies the name of throttling rule to be applied. By default, the **Total Request Count** appears as the rule name and you cannot modify this.
Operator	Specifies the operator that connects the rule to the value specified. By default, the **Greater Than** operator is selected. It indicates that the

Property	Description
	throttling rule is applied when the Total Request Count for an API is greater than (exceeds the limit specified for) the value specified in the **Value** field.
Value	Specifies the value of the request count beyond which the policy is violated. When multiple requests are made at the same time, it might be possible that this limit applied to trigger an alert is not strictly adhered to. There is no loss observed in the invocation counts data, but there might be a minor delay in aggregating the count. The invocation count gets incremented, only when API Gateway receives the response from the native API. For example, if you have set the limit at 5 with an interval alert of 1 minute and if you invoke 5 requests in parallel, out of which for 1 of the request the native API delays sending the response to API Gateway. In such cases, the invocation count would still be 4 as the native API is yet to send the response to API Gateway. There is a minor delay in aggregating the count due to native API response delay. Hence, API Gateway allows additional invocation. However, when the invocation count exceeds 5 an alert is triggered.
Destination	Specifies the destination to log the alerts. Select the required options: ■ **API Gateway** ■ **API Portal** ■ **CentraSite** **Note:** Applicable only for the APIs published from CentraSite to API Gateway. ■ **Digital Events** ■ **Elasticsearch** ■ **Email** (you can add multiple email addresses by clicking + Add). **Note:** If an email alias is available, you can type the email alias in the **Email Address** field with the following syntax, ${emailaliasname}. For example, if test is the email alias, then type ${test}.

Property	Description
	■ **JDBC** ■ **Local Log**: You can select the severity of the messages to be logged (logging level) from the **Log Level** drop-down list. The available log levels are ERROR, INFO, and WARN. **Note:** ■ Set the Integration Server Administrator's logging level for API Gateway to match the logging levels specified for the run-time actions (go to **Settings** > **Logging** > **Server Logger**). For example, if a Log Invocation action is set to the logging level of Error, you must also set Integration Server Administrator's logging level for API Gateway to Error. If the action's logging level is set to a low level (Warning-level or Information level), but Integration Server Administrator's logging level for API Gateway is set to a higher level (Error-level), then only the higher-level messages are written to the log file. ■ Entries posted to the local log are identified by a product code of YAI and suffixed with the initial alphabet of the logging level selected. For example, for an error level, the entry appears as [YAI.0900.0002E]. ■ **SNMP** ■ List of destinations configured using the **Custom destinations** section. For details on publishing to custom destinations, see *webMethods API Gateway Administration*.
Alert Interval	Specifies the interval of time for the limit to be reached. The timer starts once the first API is activated and resets after the configured time interval. If an API is deactivated the interval gets reset, and on API activation the time interval starts afresh.
Unit	Specifies the unit of measurement of the **Alert Interval** configured, to monitor performance, before sending an alert. For example: ■ **Minutes** ■ **Hours** ■ **Days** ■ **Calendar Week**. The time interval starts on the first day of the week and ends on the last day of the week. By default, the start day of the week is set to Monday. For example:

Property	Description
	■ If an API is activated on a Wednesday and **Alert Interval** is set to 1, the time interval ends on Sunday, that is, 5 days. ■ If an API is activated on a Wednesday and **Alert Interval** is set to 2, the time interval still ends on Sunday, but the period is two calendar weeks, that is 12 days. You can change the start day of the week using the extended setting `startDayOfTheWeek` in the **Administration** > **General** > **Extended settings** section. Restart the API Gateway server for the changes to take effect. ■ **Calendar Month**. The time interval starts on the first day of the month and ends on the last day of the month. For example: ■ If an API is activated in the month of August and **Alert Interval** is set to 1, the time interval ends on the last day of August. ■ If an API is activated in the month of August and **Alert Interval** is set to 2, the time interval ends in two calendar months, that is on the last day of September.
Alert Frequency	Specifies the frequency at which the alerts are issued and the monitor events are logged. Specify one of the options: ■ **Only Once**. Triggers an alert every time the specified condition is violated and logs a monitor event for the alert interval specified. ■ **Every Time**. Triggers an alert every time the specified condition is violated and logs multiple monitor events based on the number of API invocations.
Alert Message	Specifies the text message to be included in the alert.
Consumer-specific throttling	Specifies whether the configured invocation limit has to apply to all consumer applications together or to each application individually. **Note:** This field is applicable only when you select the required applications from the **Consumer Applications** field. Possible values: ■ If this option is selected, each application specified in the **Consumer Applications** field is allowed to invoke the API the

Property	Description
	specified number of times within the given time limit. For example, consider the following: You have specified consumer applications *Consumer1* and *Consumer2* and you allow 100 invocations per minute. Then, *Consumer1* can perform up to 100 invocations per minute and *Consumer2* as well can perform the same without policy violation. ■ If this option is not selected, the specified invocation limit is applied to all consumer applications together. For example, consider the following: You have specified consumer applications *Consumer1* and *Consumer2* and you allow 100 invocations per minute. Then, both consumer applications together can invoke the API for 100 times per minute without policy violation.
Consumer Applications	Specifies the consumer applications for which the policy is applied. Do one of the following: ■ Select the required consumer applications. Type a keyword to find the required application and click + to add it. ■ Select an option to apply the configured invocation limit: ■ **All consumers**. To apply the invocation limit to all consumers. All consumer applications - including the registered, global applications, and default applications - to invoke the API the specified number of times within the given time limit. That is, the specified invocation limit is shared by all consumer applications. For example, if you specify `1000` invocations per minute for an API, then the total number of invocations performed by all consumer applications in a minute cannot exceed 1000. ■ **All registered consumers**. To apply the invocation limit to all registered consumers. Allows all registered consumer applications to invoke the API the specified number of times within the given time limit. That is, the specified invocation limit is shared by all registered consumer applications. The registered consumer applications are the applications that are registered with API to which the policy is applied. For example. if you specify `1000` invocations per minute for an API, then the total number of invocations performed by all

Property	Description
	registered consumer applications in a minute cannot exceed 1000. ■ **All non-registered consumers**. To apply the invocation limit to all non-registered consumers. Allows all non-registered consumer applications to invoke the API for the specified number of times within the given time limit. That is, the specified invocation limit is shared by all non-registered application. The non-registered applications are the applications that are not registered with the API to which the policy is applied. For example, if you specify `1000` invocations per minute for an API, then the total number of invocations performed by all non-registered consumer applications in a minute cannot exceed 1000. **Note:** You can invoke an API using a non-registered consumer application only if the `Global applications` or `Global applications and DefaultApplication` option is selected from the **Application Lookup Control** field of the corresponding **Identify & Authorize** policy. ■ **Each consumer**. To apply the invocation limit to each consumer. Allows each consumer application to invoke the API the specified number of times within the given time limit. That is, the specified invocation limit is applicable individually for each consumer application. For example, if you specify `1000` invocations per minute for an API, then each consumer application can perform 1000 invocations in a minute ■ **Each registered consumer**. To apply the invocation limit to each registered consumer. Allows each registered consumer application to invoke the API the specified number of times within the given time limit. That is, the specified invocation limit is applicable individually for each registered consumer application. For example, if you specify `1000` invocations per minute, then each registered consumer application can perform 1000 invocations in a minute. ■ **Each non-registered consumer**. To apply the invocation limit to each non-registered consumer. Allows each non-registered consumer application to invoke the API the

Property	Description
	specified number of times within the given time limit. That is, the specified invocation limit is applicable individually for each non-registered consumer application. For example, if you specify `1000` invocations per minute, then each non-registered consumer application can perform 1000 invocations in a minute. The invocation limit applied to newly added applications depends on the application type. The time limit starts when the associated API is activated or the application is associated to the policy. For example, consider the following: API-level Traffic Optimization policy configured with invocation limit as `500` for `5 minutes`, for **All registered consumers**. The available registered consumers are *Consumer1* and *Consumer2*. When you activate the policy, the two consumers together perform 200 invocations in 2 minutes. Register a new consumer application, *Consumer3* to the list after 2 minutes from the start time. Then, all three consumers perform the remaining 300 invocations during the last 3 minutes without violating the policy. Let us see one more example. Global Traffic Optimization policy configured with invocation limit as `100` for `15 minutes`, for **Each consumer**. The available consumers are *Consumer1* and *Consumer2*. When you activate the policy, the each consumer performs 100 invocations in 10 minutes. Add a new consumer application, *Consumer3* to the list after 2 minutes from the start time. Then, the new consumer can also perform 100 invocations during the last 5 minutes without violating the policy. **Note:** If you select `All consumers` or `Each consumer` from this field and specify `Registered applications` from the **Application Lookup Control** field of the corresponding **Identify & Authorize** policy, then only the registered applications are allowed to invoke the API. In such scenarios, only the registered consumer applications can utilize the specified invocation limit. You can invoke APIs using non-registered consumer applications only if the `Global applications` or `Global applications and DefaultApplication` option is selected from the **Application Lookup Control** field of the corresponding **Identify & Authorize** policy.

Traffic Optimization Policy Enforcement Scenarios

You can enforce more than one Traffic Optimization policy for an API based on your requirement. This section lists a sample requirement scenario and the possible solution:

How do I?

- Allow 1000 invocation per minute for each registered consumer application.
- Allow 500 invocation for 15 minutes for all non-registered consumer applications.

Possible solution

Create two Traffic Optimization policies with the following specifications

- **Policy 1**
  - **Value**. `1000`.
  - **Alert Interval**. `1`.
  - **Alert Frequency**. `minute`.
  - **Consumer Applications**. `Each registered consumer`.
- **Policy 2**
  - **Value**. `500`.
  - **Alert Interval**. `15`.
  - **Alert Frequency**. `minute`.
  - **Consumer Applications**. `All non-registered consumers`.

Service Result Cache

This policy enables caching of the results of API invocations depending on the caching criteria defined. You can define the elements for which the API responses are to be cached based on the criteria such as HTTP Header, XPath, Query parameters, and so on. You can also limit the values to store in the cache using a whitelist. For the elements that are stored in the cache, you can specify other parameters such as Time to Live and Maximum Response Payload Size.

Caching the results of an API request increases the throughput of the API call and improves the scalability of the API.

The cache criteria applicable for a SOAP-based API request are HTTP Header and XPATH. The cache criteria applicable for a REST-based API request are HTTP Header and Query parameters. The caching works only for GET methods for REST APIs.

Note:
If there are no values set for any of the criteria, then, by default, all the SOAP requests and GET requests for the Rest API are based on the URL.

The table lists the properties that you can specify for this policy:

Property	Description
Cache Criteria. Specifies the criteria that API Gateway uses to determine the request component, that is, the actual payload based on which the results of the API invocation are cached.	
HTTP Header	Uses the HTTP header in the API request. You can use this criterion for APIs that accept payloads only in HTTP format. **Header Name**. Specifies the HTTP header name. **Cache responses only for these values**. API Gateway caches the API responses only for requests whose cache criteria match with those set for the action, and whose criteria evaluation results in any one of the values in this list. You can add multiple entries by clicking  . **Note:** If this field is empty, all the values that satisfy the criterion are cached.
Query Parameters	You can use this criterion for REST-based API requests. Specifies the names and values of the query parameters to filter the incoming requests and cache the results based on the names and values specified. **Parameter Name**. Specifies the parameter name. **Cache responses only for these values**. API Gateway caches the API responses only for requests whose cache criteria match those set for the action, and whose criteria evaluation results in any one of the values in this list. You can add multiple entries by clicking  .

XPath	You can use this criterion for SOAP-based API requests whose payload is a SOAP envelope. Uses the XPath expression in the API request.
	
	■ **Name Space**. Specifies the namespace of the XPath expression.
	  ■ **Prefix**. Specifies the prefix for the namespace.
	  ■ **URI**. Specifies the namespace URI.
	
	You can add multiple entries by clicking  .
	
	XPath Expression. Specifies the XPath expression in the API request.
	
	Cache responses only for these values. API Gateway caches the API responses only for requests whose cache criteria match those set for the action, and whose criteria evaluation results in any one of the values in this list. You can add multiple entries by clicking  .

Property	Description
	Note: If this field is empty, all the values that satisfy the criteria are cached.
Time to Live (e.g., 5d 4h 1m)	Specifies the lifespan of the elements in the cache after which the elements are considered to be out-of-date. The time is specified in terms of days, hours, and minutes; for example, 5d 4h 1m. If you do not specify any value, the Time to Live is considered to be unlimited (does not expire). If you set the value to 0d 0h 0m, the API results are not cached. The default time format is minutes if the input is a number.
Maximum Response Payload Size (in KB)	Specifies the maximum payload size for the API in kilo bytes. The value -1 stands for unlimited payload size.

Example of enforcing caching criteria:

Cache criteria	HTTP Header	Query parameters	XPATH	Values
C1	Header1			h1, h2
C2	Header2			
C3		query1		q1, q2

In the example, there are two HTTP headers and one query parameter as cache criteria. The HTTP Header **Header2** has no values specified. Hence, all the incoming requests with the HTTP Header **Header2** are cached.

When there are multiple cache criteria, the following behaviour is observed in the cache result:

- If the incoming request R1 matches criteria C1, then the result is cached. API Gateway responds to any further incoming request R1 that matches criteria C1 from the cache.
- If the incoming request R1 matches criteria C1 and C2, then the result is cached as a new request.
- If you configure multiple cache criteria, and if one or more cache criteria match, then the result is cached. The criteria are matched with the cached results while caching the request, and it follows the AND condition among the matched criteria.

Monitor Performance

This policy monitors a set of run-time performance conditions for an API, and sends alerts when the performance conditions are violated. However, this policy monitors run-time performance at

the API level. Parameters like success count, fault count and total request count are immediate monitoring parameters and the evaluation happens immediately after the limit is breached. The rest of the parameters are Aggregated monitoring parameters whose evaluation happens once the configured interval is over. If there is a breach in any of the parameters, an event notification (Monitor event) is sent to the configured destination. In a single policy, multiple action configurations behave as AND condition. The OR condition can be achieved by configuring multiple policies.

The table lists the properties that you can specify for this policy:

Property	Value
Action Configuration. Specifies the type of action to be configured.	
Name	Specifies the name of the metric to be monitored. You can select one of the available metrics: ■ **Availability**. Indicates whether the native API is available to the clients as specified in the current interval. API Gateway calculates the availability of the native API based on the alert interval specified and it is calculated from the instant the API activation takes place. The availability of the API is calculated as = (time for which the native API is up / total interval of time) x 100. This value is measured in %. For example, if you set **Availability** as less than 90, then whenever the availability of the native API falls below 90%, in the specified time interval, API Gateway generates an alert. Suppose, the alert interval is set as 1 minute (60 seconds) and if there are 7 API invocations at various times in that 1 minute with a combination of up and down as shown in the table, the availability is calculated as follows:

Request #	Invocation time (the second at which the API is invoked)	Service status	Up time
1	5	Up	5 (from start to now)
2	15	Up	10 (between 1 and 2)
3	30	Down	15 (between 2 and 3)
4	40	Down	0 (since last state is Down)
5	45	Up	0
6	50	Down	5 (between 5 and 6)

Property	Value

Request #	Invocation time (the second at which the API is invoked)	Service status	Up time
7	55	Up	0
			5 (remaining 5 seconds considered as Up inline with last state)
	Total		40 (Availability is 67%)

As the availability of the native API calculated is 66.67% and falls below 90%, API Gateway generates an alert. The API is considered to be down for the ongoing request when API Gateway receives a connection related error from the native API in the outbound call. If the API does not respond with an HTTP response, then it is considered as down.

- **Average Response Time**. Indicates the average time in milliseconds taken by the service to complete all invocations in the current interval. The average is calculated from the instant the API activation takes place for the configured interval.

  For example, if you set an alert for Average response time greater than 30 ms with an interval of 1 minute then on API activation, the monitoring interval starts and the average of the response time of all runtime invocations for this API in 1 minute is calculated. If this is greater than 30 ms, then a monitor event is generated. If this is configured under Monitor Performance, then all the runtime invokes are taken into account.

- **Fault Count**. Indicates the number of faults returned in the current interval. The HTTP status codes greater than or equal to 400, returned from API Gateway are considered as fault request transactions. This includes the downtime errors as well.

- **Maximum Response Time**. Indicates the maximum time in milliseconds to respond to a request in the current interval.

- **Minimum Response Time**. Indicates the minimum time in milliseconds to respond to a request in the current interval.

- **Success Count**. Indicates the number of successful requests in the current interval.

Property	Value
	■ **Total Request Count**. Indicates the total number of requests (successful and unsuccessful) in the current interval.
Operator	Specifies the operator applicable to the metric selected. Select one of the available operator: **Greater Than**, **Less Than**, **Equals To**.
Value	Specifies the alert value for which the monitoring is applied.
Destination	Specifies the destination where the alert is to be logged. Select the required options: ■ **API Gateway** ■ **API Portal** ■ **CentraSite** **Note:** This option is applicable only for the APIs published from CentraSite to API Gateway. ■ **Digital Events** ■ **Elasticsearch** ■ **Email** (you can add multiple email addresses by clicking + Add). **Note:** If an email alias is available, you can type the email alias in the **Email Address** field with the following syntax, ${emailaliasname}. For example, if test is the email alias, then type ${test}. ■ **JDBC** ■ **Local Log**: You can select the severity of the messages to be logged (logging level) from the **Log Level** drop-down list. The available log levels are ERROR, INFO, and WARN. **Note:** ■ Set the Integration Server Administrator's logging level for API Gateway to match the logging levels specified for the run-time actions (go to **Settings** > **Logging** > **Server Logger**). For example, if a Log Invocation action is set to the

Property	Value
	logging level of Error, you must also set Integration Server Administrator's logging level for API Gateway to Error. If the action's logging level is set to a low level (Warning-level or Information level), but Integration Server Administrator's logging level for API Gateway is set to a higher level (Error-level), then only the higher-level messages are written to the log file. ■ Entries posted to the local log are identified by a product code of YAI and suffixed with the initial alphabet of the logging level selected. For example, for an error level, the entry appears as [YAI.0900.0002E]. ■ **SNMP** ■ List of destinations configured using the **Custom destinations** section. For details on publishing to custom destinations, see *webMethods API Gateway Administration*.
Alert Interval	Specifies the time period in which to monitor performance before sending an alert if a condition is violated. The timer starts once the API is activated and resets after the configured time interval. If an API is deactivated the interval gets reset, and on API activation the time interval starts afresh.
Unit	Specifies the unit of measurement of the **Alert Interval** configured, to monitor performance, before sending an alert. For example: ■ **Minutes** ■ **Hours** ■ **Days** ■ **Calendar Week**. The time interval starts on the first day of the week and ends on the last day of the week. By default, the start day of the week is set to Monday. For example: ■ If an API is activated on a Wednesday and **Alert Interval** is set to 1, the time interval ends on Sunday, that is, 5 days. ■ If an API is activated on a Wednesday and **Alert Interval** is set to 2, the time interval still ends on Sunday, but the period is two calendar weeks, that is 12 days. You can change the start day of the week using the extended setting `startDayOfTheWeek` in the **Administration** > **General** > **Extended settings** section. Restart the API Gateway server for the changes to take effect.

Property	Value
	■ **Calendar Month**. The time interval starts on the first day of the month and ends on the last day of the month. For example: ■ If an API is activated in the month of August and **Alert Interval** is set to 1, the time interval ends on the last day of August. ■ If an API is activated in the month of August and **Alert Interval** is set to 2, the time interval ends in two calendar months, that is on the last day of September.
Alert Frequency	Specifies how frequently to issue alerts for the counter-based metrics (Total Request Count, Success Count, Fault Count). Select one of the options: ■ **Only Once**. Triggers an alert only the first time one of the specified conditions is violated. ■ **Every Time**. Triggers an alert every time one of the specified conditions is violated.
Alert Message	Specifies the text to be included in the alert.

Monitor SLA

This policy monitors a set of run-time performance conditions for an API, and sends alerts to a specified destination when the performance conditions are violated. This policy enables you to monitor run-time performance for one or more specified applications. You can configure this policy to define a **Service Level Agreement (SLA)**, which is a set of conditions that defines the level of performance that an application should expect from an API. You can use this policy to identify whether the API threshold rules are met or exceeded. For example, you might define an agreement with a particular application that sends an alert to the application if responses are not sent within a certain maximum response time. You can configure SLAs for each API or application combination.

Parameters like success count, fault count and total request count are immediate monitoring parameters and the evaluation happens immediately after the limit is breached. The rest of the parameters are Aggregated monitoring parameters whose evaluation happens once the configured interval is over. If there is a breach in any of the parameters, an event notification (Monitor event) is sent to the configured destination. In a single policy, multiple action configurations behave as AND condition. The OR condition can be achieved by configuring multiple policies.

The table lists the properties that you can specify for this policy:

Property	Value
Action Configuration. Specifies the type of action to be configured.	

Property	Value
Name	Specifies the name of the metric to be monitored. You can select one of the available metrics: ■ **Availability**. Indicates whether the native API is available to the clients as specified in the current interval. API Gateway calculates the availability of the native API based on the alert interval specified and it is calculated from the instant the API activation takes place. The availability of the API is calculated as = (time for which the native API is up / total interval of time) x 100. This value is measured in %. For example, if you set **Availability** as less than 90, then whenever the availability of the native API falls below 90%, in the specified time interval, API Gateway generates an alert. Suppose, the alert interval is set as 1 minute (60 seconds) and if there are 7 API invocations at various times in that 1 minute with a combination of up and down as shown in the table, the availability is calculated as follows:

Request #	Invocation time (the second at which the API is invoked)	Service status	Up time
1	5	Up	5 (from start to now)
2	15	Up	10 (between 1 and 2)
3	30	Down	15 (between 2 and 3)
4	40	Down	0 (since last state is Down)
5	45	Up	0
6	50	Down	5 (between 5 and 6)
7	55	Up	0
			5 (remaining 5 seconds considered as Up inline with last state)
	Total		40 (Availability is 67%)

As the availability of the native API calculated is 66.67% and falls below 90%, API Gateway generates an alert. The API is considered to be down for the ongoing request when API Gateway receives a

Property	Value
	connection related error from the native API in the outbound call. If the API does not respond with an HTTP response, then it is considered as down. - **Average Response Time**. Indicates the average time taken in milliseconds (ms) by the service to complete all invocations in the current interval. The average is calculated from the instant the API activation takes place for the configured interval. For example, if you set an alert for Average response time greater than 30 ms with an interval of 1 minute then on API activation, the monitoring interval starts and the average of the response time of all runtime invocations for this API in 1 minute is calculated. If this is greater than 30 ms, then a monitor event is generated. If this is configured under Monitor SLA policy with an option to configure applications so that application specific SLA monitoring can be done, then the monitoring for the average response time is done only for the specified application. - **Fault Count**. Indicates the number of faults returned in the current interval. The HTTP status codes greater than or equal to 400, returned from API Gateway are considered as fault request transactions. This includes the downtime errors as well. - **Maximum Response Time**. Indicates the maximum time in milliseconds (ms) to respond to a request in the current interval. - **Minimum Response Time**. Indicates the minimum time in milliseconds (ms) to respond to a request in the current interval. - **Success Count**. Indicates the number of successful requests in the current interval. - **Total Request Count**. Indicates the total number of requests (successful and unsuccessful) in the current interval.
Operator	Specifies the operator applicable to the metric selected. Select one of the available operator: **Greater Than**, **Less Than**, **Equals To**.
Value	Specifies the alert value for which the monitoring is applied.
Destination	Specifies the destination where the alert is to be logged. Select the required options: - **API Gateway** - **API Portal**

Property	Value
	■ **CentraSite** **Note:** This option is applicable only for the APIs published from CentraSite to API Gateway. ■ **Digital Events** ■ **Elasticsearch** ■ **Email** (you can add multiple email addresses by clicking + Add). **Note:** If an email alias is available, you can type the email alias in the **Email Address** field with the following syntax, ${emailaliasname}. For example, if test is the email alias, then type ${test}. ■ **JDBC** ■ **Local Log**: You can select the severity of the messages to be logged (logging level) from the **Log Level** drop-down list. The available log levels are ERROR, INFO, and WARN. **Note:** ■ Set the Integration Server Administrator's logging level for API Gateway to match the logging levels specified for the run-time actions (go to **Settings** > **Logging** > **Server Logger**). For example, if a Log Invocation action is set to the logging level of Error, you must also set Integration Server Administrator's logging level for API Gateway to Error. If the action's logging level is set to a low level (Warning-level or Information level), but Integration Server Administrator's logging level for API Gateway is set to a higher level (Error-level), then only the higher-level messages are written to the log file. ■ Entries posted to the local log are identified by a product code of YAI and suffixed with the initial alphabet of the logging level selected. For example, for an error level, the entry appears as [YAI.0900.0002E]. ■ **SNMP** ■ List of destinations configured using the **Custom destinations** section. For details on publishing to custom destinations, see *webMethods API Gateway Administration*.

Property	Value
Alert Interval	Specifies the time period (in minutes) in which to monitor performance before sending an alert if a condition is violated. The timer starts once the API is activated and resets after the configured time interval. If and API is deactivated the interval gets reset and on API activation its starts afresh.
Unit	Specifies the unit of measurement of the **Alert Interval** configured, to monitor performance, before sending an alert. For example: ■ **Minutes** ■ **Hours** ■ **Days** ■ **Calendar Week**. The time interval starts on the first day of the week and ends on the last day of the week. By default, the start day of the week is set to Monday. For example: ■ If an API is activated on a Wednesday and **Alert Interval** is set to 1, the time interval ends on Sunday, that is, 5 days. ■ If an API is activated on a Wednesday and **Alert Interval** is set to 2, the time interval still ends on Sunday, but the period is two calendar weeks, that is 12 days. You can change the start day of the week using the extended setting `startDayOfTheWeek` in the **Administration** > **General** > **Extended settings** section. Restart the API Gateway server for the changes to take effect. ■ **Calendar Month**. The time interval starts on the first day of the month and ends on the last day of the month. For example: ■ If an API is activated in the month of August and **Alert Interval** is set to 1, the time interval ends on the last day of August. ■ If an API is activated in the month of August and **Alert Interval** is set to 2, the time interval ends in two calendar months, that is on the last day of September.
Alert Frequency	Specifies how frequently to issue alerts for the counter-based metrics (Total Request Count, Success Count, Fault Count). Select one of the options:

Property	Value
	■ **Only Once**. Triggers an alert only the first time one of the specified conditions is violated. ■ **Every Time**. Triggers an alert every time one of the specified conditions is violated.
Alert Message	Specifies the text to be included in the alert.
Consumer Applications	Specifies the application to which this Service Level Agreement applies. You can type a search term to match an application and click + Add to add it. You can add multiple applications or delete an added application by clicking  .

Response Processing

These policies are used to specify how the response message from the API has to be transformed or pre-processed and configure the masking criteria for the data to be masked before it is submitted to the application. This is required to protect the data and accommodate differences between the message content that an API is capable of submitting and the message content that an application expects. The policies included in this stage are:

- Invoke webMethods IS
- Response Transformation
- Validate API Specification
- CORS
- Data Masking
- Custom Extension

  Custom Extension policies allow you to handle requirements that might not be provided by the out-of-the-box policies. You can add these custom extensions into API Gateway policy stages. To learn more about Custom Extension, see *webMethods API Gateway User's Guide*.

Validate API Specification

This policy validates the responses against API's various specifications such as schema, content-types, and HTTP Headers referenced in their corresponding formats as follows:

- The schema is available as part of the API definition. The schema for SOAP API are imported through WSDL and for REST APIs it can be through swagger, RAML or can be uploaded by the user when an API is created from scratch.

- The content- types are available as part of the API definition. FOR SOAP APIs these are imported through WSDL and for REST APIs it can be through swagger, RAML or can be uploaded by the user.
- The HTTP Headers are specified in the Validate API Specification policy page.

The response sent to the API by an application must conform with the structure or format expected by the API. The responses from the native API are validated against the API specifications in this policy to conform to the structure or format expected by the API.

Various API specifications validated are:

- **Schema:**

  The responses from the native API are validated against the schema provided in the API definition. API Gateway does not validate the payload, if the payload is sent as a stream.

  The schema defines the elements and attributes and specifies the data types of these elements to ensure that only appropriate data is allowed through to the API. For a REST API, the schema can be added inline or uploaded in the Components section on the API Details page. For details on how to add the schema inline or upload, see "Creating a REST API from Scratch" on page 20.

  The schema type for validation is selected based on:

  - The Content-Type header when the policy is added in the Request processing stage.
  - The Accept header when the policy is added in the Response processing stage.

  If the header or payload is missing the schema validation is skipped.

  The table lists the default Content type/Accept header and schema validation type mapping.

Content-type/Accept	Schema validation type
application/json	JSON schema
application/json/badgerfish	
application/xml	XML schema
text/xml	
text/html	
text/plain	Regular expression

  For a SOAP API, the WSDL and the referenced schema must be provided in a zip format. The JSON schema validation is supported for the operations that are exposed as REST.

- **Content-types:**

  The responses from the native API are validated against the content-types specified in the API definition.

- **HTTP Headers:**

The responses from the native API are validated against the HTTP Headers specified in this policy to conform to the HTTP headers expected by the API. If the HTTP Headers are not specified in this policy, API Gateway uses the Headers defined in the API specification.

The run-time invocations that fail the specification validation are considered as policy violations. Such policy violation events that are generated can be viewed in the dashboard.

The table lists the API specification properties, you can specify for this policy, to be validated:

Property	Description
Schema	Validates the response payload against the appropriate schema. Provide the following additional features for XML schema validation: ■ **Feature name**. Specifies the name of the feature for XML parsing when performing XML schema validation. Select the required feature names from the list: ■ GENERATE_SYNTHETIC_ANNOTATIONS ■ ID_IDREF_CHECKING ■ IDENTITY_CONSTRAINT_CHECKING ■ IGNORE_XSL_TYPE ■ NAMESPACE_GROWTH ■ NORMALIZE_DATA ■ ROOT_ELEMENT_DECL ■ ROOT_TYPE_DEF ■ SIGMA_AUGMENT_PSVI ■ SCHEMA_DV_FACTORY ■ SCHEMA_ELEMENT_DEFAULT ■ SCHEMA_LOCATION ■ SCHEMA_NONS_LOCATION ■ SCHEMA_VALIDATOR ■ TOLERATE_DUPLICATES ■ ENPARSED_ENTITY_CHECKING ■ VALIDATE_ANNOTATIONS ■ XML_SCHEMA_FULL_CHECKING

Property	Description
	■ `XMLSCHEMA_VALIDATION` For details about XML parsing features, see http://xerces.apache.org/xerces2-j/features.html and for details about the exact constants, see https://xerces.apache.org/xerces2-j/javadocs/xerces2/org/apache/xerces/parsers/XML11Configuration.html. ■ **Feature value**. Specifies whether the feature value is **True** or **False**.
Content-types	Validates the content-types in the incoming response against the content-types defined in that response's API Specification.
HTTP Headers	Validates the HTTP header parameters in the incoming response against the HTTP headers defined in that response's API Specification. Provide the following information: ■ Condition: Specifies the logical operator to use to validate multiple HTTP headers in the incoming API responses. Available values are: ■ **AND**. API Gateway accepts only the responses that contain all configured HTTP headers. ■ **OR**. This is selected by default. API Gateway accepts responses that contain at least one configured HTTP header. ■ **HTTP Header Key**. Specifies a key that must be passed through the HTTP header of the incoming API responses. ■ **Header Value**. Optional. Specifies the corresponding key value that could be passed through the HTTP header of the incoming API responses. As this property supports variable framework, you can make use of the available variables to specify the header value. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373. You can add more HTTP headers by clicking +Add.

Response Transformation

This policy specifies the properties required to transform response messages from native APIs into a format required by the client.

The transformations include Header transformation, Status transformation, Payload transformation, and Advanced transformation. You can configure conditions according to which the transformations are executed

The table lists the properties that you can specify for this policy:

Property	Description
Condition	Conditions are used to specify when the policy has to be executed. You can add multiple conditions with logical operators. Available values are: ■ **AND**. API Gateway transforms the responses that comply with all the configured conditions ■ **OR**. This is selected by default. API Gateway transforms the responses that comply with any one configured condition. Click **Add Condition** and provide the following information and click  . ■ **Variable**. Specifies the variable type with a syntax. ■ **Operator**. Specifies the operator to use to relate variable and the value. You can select one of the following: ■ **Equals** ■ **Equals ignore case** ■ **Not equals** ■ **Not equals ignore case** ■ **Contains** ■ **Not Contains** ■ **Exists** ■ **Not Exists** ■ **Range** ■ **Greater Than** ■ **Less Than** ■ **Value**. Specifies a plain value or value with a syntax. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
Transformation Configuration. Specifies various transformations to be configured.	
HeaderTransformation	Specifies the header, query or path transformation to be configured for the responses received from the native API.

Property	Description
	You can add or modify header, query or path transformation parameters by providing the following information: ■ **Variable**. Specifies the variable type with a syntax. ■ **Value**. Specifies a plain value or value with a syntax. You can add multiple variables and corresponding values by clicking  . You can remove any header, query, or path transformation parameters by typing the plain value or value with a syntax. **Note:** Software AG recommends you not to modify the headers `${response.headers.Content-Length}` and `${response.headers.Content-Encoding}` as API Gateway adds the right values for these headers before sending the response back to client. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373. **Note:** Payload transformation does not happen automatically for content-type transformation. When you change the content type, ensure to do payload transformation. For example, if you change the content-type header from application/xml to application/json, you must also change the respective payload from application/xml to application/json.
Status transformation	Specifies the status transformation to be configured for the responses received from the native API. Provide the following information: ■ **Code**. Specifies the status code that is sent in the response to the client. For example if you want to transform status code as 201, provide 201 in the **Code** field. ■ **Message**. Specifies the Status message that is sent in the response to the client. As both these properties support variable framework, you can make use of the available variables to transform the response code and message.

Property	Description
	For example *You have submitted successfully* can be used to transform the original *OK* status message. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

Payload Transformation	Specifies the payload transformation to be configured for the responses received from the native API.
	
	Note:
	API Gateway does not process the payload, if the payload is sent as a stream.
	
	Provide the following information:
	
	■ **Payload Type**. Specifies the content-type of payload, to which you want to transform. The **Payload** field renders the respective payload editor based on the selected content-type.
	
	■ **Payload**. Specifies the transformation that needs to be applied for the response.
	
	As this property supports variable framework, you can make use of the available variables to transform the response messages.
	
	For example, consider the client accepting two integer values `value1` and `value2`, and you want to pass these two values from API Gateway to the client, you can configure the payload field as follows:
	
	`{`
	`"value1" : 12,`
	`"value2" : 34`
	`}`
	
	You can also configure the payload field using one or more variables by using variable framework. Let us see another syntax. For example, for the same API seen in the previous example, if your native sends both the values through headers `val1` and `val2`, and you want to add it to payload for the client to recognize the input, you can do so by configuring the payload field as follows:
	
	`{`
	`"value1" :${response.headers.val1},`
	`"value2" :${response.headers.val2}`
	`}`
	
	For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

Property	Description

	Note:
	If your payload content-type is different from the incoming payload's content-type, you need to transform the content-type of the header using **Header Transformation**.
	■ Click **+ Add xslt document** to add an xslt document and provide the following information:
	■ **XSLT file**. Specifies the XSLT file used to transform the response messages as required.
	Click **Browse** to browse and select a file.
	■ **Feature Name**. Specifies the name of the XSLT feature.
	■ **Feature value**. Specifies the value of the XSLT feature.
	You can add more XSLT features and xslt documents by clicking  .
	Note:
	API Gateway supports XSLT 1.0 and XSLT 2.0.
	■ Click **+ Add xslt transformation alias** and provide the following information:
	■ **XSLT Transformation alias**. Specifies the XSLT transformation alias
	When you receive the response in JSON, you can use a XSLT file similar to the below sample:
	`<?xml version="1.0" ?>`
	`<xsl:stylesheet version="1.1"`
	`xmlns:xsl="http://www.w3.org/1999/XSL/Transform">`
	`<xsl:output method="xml"/>`
	`<xsl:template match="/" >`
	`<xsl:element name="fakeroot">`
	`<xsl:element name="fakenode">`
	`<!-- Apply your transformation rules based on the response received from the native API-->`
	`</xsl:element>`
	`</xsl:element>`
	`</xsl:template>`
	`</xsl:stylesheet>`
	When you receive the response in XML, you can use a XSLT file similar to the below sample:
	`<?xml version="1.0" ?>`
	`<xsl:stylesheet version="1.1"`

Property	Description
	<pre> xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"> <xsl:output method="xml"/> <xsl:template match="/" > <xsl:element name="soapenv:Envelope"> <xsl:element name="soapenv:Body"> <!-- Apply your transformation rules based on the response received from the native API--> </xsl:element> </xsl:element> </xsl:template> </xsl:stylesheet></pre>
Advanced Transformation	Specifies the advanced transformation to be configured for the responses received from the native API.. Provide the following information: ■ **webMethods IS Service**. Specify the webMethods IS service to be invoked to process the response messages. You can add multiple services by clicking  . **Note:** The webMethods IS service must be running on the same Integration Server as API Gateway. ■ **Run as User**. Specifies the authentication mode to invoke the IS service. If this field is left blank the incoming credentials of the user, identified by API Gateway, are used to authenticate and invoke the IS service. You can also specify a particular user, you want API Gateway to use to run the IS service. ■ **Comply to IS Spec**. Mark this as `true` if you want the input and the output parameters to comply to the IS Spec present in pub.apigateway.invokeISService.specifications folder in wmAPIGateway package. ■ **webMethods IS Service alias**. Specifies the webMethods IS service alias to be invoked to pre-process the request messages.
Transformation Metadata. Specifies the metadata for transformation of the responses received from the native API. For example, the namespaces configured in this section can be used when you provide the syntax for XPath ${response.payload.xpath} For example: ${response.payload.xpath[//ns:emp/ns:empName]}	
Namespace	Specifies the namespace information to be configured for transformation.

Property	Description
	Provide the following information: ■ **Namespace Prefix**. The namespace prefix of the payload expression to be validated. For example, specify the namespace prefix as `SOAP_ENV`. ■ **Namespace URI**. The namespace URI of the payload expression to be validated. For example, specify the namespace URI as `http://schemas.xmlsoap.org/soap/envelope/`. This declaration defines SOAP_ENV as an alias for the namespace: http://schemas.xmlsoap.org/soap/envelope/. **Note:** You can add multiple namespace prefix and URI by clicking  .

Invoke webMethods IS

This policy processes the native API's response messages into the format required by the application, before API Gateway returns the responses to the application.

If `Comply to IS Spec` parameter is configured as `true`, API Gateway invokes the IS Service with IS specification in the path pub.apigateway.invokeISService.specifications:ResponseSpec (for Response Processing).

Note:
The pipeline variables in the Invoke IS service include only the response headers and they do not include request headers (sent by the client or the ones added during the request transformation step). To access the request headers in the Invoke IS Service flow, use the **pub.flow:getTransportInfo** service.

The following are the input and output parameters for REST, SOAP, and WebSocket APIs as specified in the above IS Specification. Input parameters can be used to access the existing values of the response while output parameters can be used to modify/write the values to the response.

	Parameter name	Description
Input parameters	**headers**	Headers in response. Data type: Document
	payload	Payload of the response. Data type: String

	Parameter name	Description
	payloadObject	The payload for binary content types like multi-part / form-data. Data type: Object
	statusCode	Status code of the response. Data type: String
	statusMessage	Status message of the response. Data type: String
	MessageContext	The message context object of the response. Data type: Object
	apiName	Name of the API invoked by the response. Data type: String
	requestUrl	URL of the response. Data type: String
	ipInfo	Contains IP information of the response. Data type: Document
	websocketInfo	Websocket related information of the response. Data type: Document
	correlationID	Correlation ID of the request/response. This is unique and same for a request and response. Data type: String
	customFieldsMap	Custom transactional fields can be added to the transactional events using this field. For more information, see **Adding Custom Fields to Transactional Events** section. Data type: Document
Output parameters	**headers**	Headers in response. Data type: Document
	payload	Payload of the response. Data type: String

Parameter name	Description
payloadObject	The payload for binary content types like multi-part / form-data. Data type: Object
statusCode	Status code of the response. Data type: String
statusMessage	Status message of the response. Data type: String
MessageContext	The message context object of the response. Data type: Object
customFieldsMap	Custom transactional fields can be added to the transactional events using this field. For more information, see **Adding Custom Fields to Transactional Events** section. Data type: Document

Note:

- For SOAP to REST APIS, the payload contains the transformed JSON response.
- Payload transformation does not happen automatically for content-type transformation. When you change the content type, ensure to do payload transformation also as part of IS Service.
- When `Comply to IS spec` is true, you can change the values of headers, query, payload, and so on, programatically using Message Context, as well as using the pipeline variables given. Software AG recommends you not to change those values directly in Message Context, as the values in output pipeline variables are written to Message Context after the invocation of IS Service.

If `Comply to IS Spec` parameter is set to `false`, API Gateway invokes the IS Service with the same input and output parameters supported in 10.1 and the earlier versions::

- proxy.name
- JSONRESTContentString (REST only)
- SOAPEnvelope (SOAP only)
- EnvelopeString (SOAP only)

The table lists the properties that you can specify for this policy:

Property	Description
Invoke webMethods Integration Server Service	

Add invoke webMethods Integration Server service	Specifies the webMethods IS service to be invoked to process the response messages and the authentication mode for the IS service.
	
	Provide the following information:
	
	■ **webMethods IS Service**. Specify the webMethods IS service to be invoked to pre-process the response messages.
	
	The webMethods IS service must be running on the same Integration Server as API Gateway
	
	Note:
	If an exception occurs when invoking the webMethods IS service, by default API Gateway displays the status code as *500* and error message as *Internal Server Error*.
	
	You can set custom status code and error message by setting the following properties in the message context of the webMethods IS service:
	■ service.exception.status.code
	■ service.exception.status.message
	
	The sample code is given below:
	`IDataCursor idc = pipeline.getCursor();`
	`MessageContext context =`
	`(MessageContext)IDataUtil.get(idc,"MessageContext");`
	`if(context != null)`
	`{`
	`context.setProperty("service.exception.status.code",`
	`404);`
	`context.setProperty("service.exception.status.message",`
	`"Object Not Found");`
	`throw new ServiceException();`
	`}`
	
	Note:
	If ServiceException or FlowException occurs when invoking webMethods IS Service, the message given in the exception is displayed to the client. If any other exception occurs, a generic error message is displayed to the client.
	
	■ **Run as User**. Specifies the authentication mode to invoke the IS service. If this field is left blank the incoming credentials of the user, identified by API Gateway, are used to authenticate and invoke the IS service. You can also specify a particular user, you want API Gateway to invoke the IS service.

Property	Description
	Note: It is the responsibility of the user who activates the API to review the value configured in **Run as User** field to avoid misuse of this configuration. ■ **Comply to IS Spec**. Mark this as `true` if you want the input and the output parameters to comply to the IS Spec present in pub.apigateway.invokeISService.specifications folder in wmAPIGateway package. **Note:**Software AG recommends users to configure the policy with `Comply to IS Spec` as `true`, as you can read or change the values of headers, and so on, without having to read from or write to the message context.
webMethods IS Service alias	Specifies the webMethods IS service alias used to invoke the webMethods IS service to pre-process the response messages. Start typing the webMethods alias name, select the alias from the type-ahead search results displayed and click + Add to add one or more aliases. You can use the delete icon to delete the added aliases from the list.

Adding Custom Fields to Transactional Events

This section explains you how to add custom fields to the transactional events.

1. Create webMethods IS service by specifying the *pub.apigateway.utils:customFieldInTransactionEventSpec* as a specification reference.
2. In the webMethods IS service, set the required custom fields in the *customFieldsMap* output variable.
3. Once when *customFieldsMap* gets created, the custom fields will be available in the transactional events.
4. Invoke the API with the Invoke webMethods IS policy.

Note:
You can also add the custom fields to the transactional events from API Gateway by configuring the *customTransactionFields.FIELD_NAME* custom variable in the **Custom Extension** policy. For more details, see *webMethods API Gateway User's Guide*.

Data Masking

Data masking is a technique whereby sensitive data is obscured in some way to render it safe and to protect the actual data while having a functional substitute for occasions when the real data is not required.

This policy is used to mask sensitive data at the application level. At the application level you must have an Identify and Access policy configured to identify the application for which the masking is applied. If no application is specified then it will be applied for all the other responses. Fields can be masked or filtered in the response messages to be sent. You can configure the masking criteria as required for the XPath, JSONPath, and Regex expressions based on the content-types. This policy can also be applied at the API scope level.

Note:
API Gateway does not mask the payload, if the payload is sent as a stream.

The table lists the content-type and masking criteria mapping.

Content-type	Masking Criteria
application/xml	XPath
text/xml	
text/html	
application/json	JSONPath
application/json/badgerfish	
text/plain	Regex

The table lists the masking criteria properties that you can configure to mask the data in the response messages:

Property	Description
Consumer Applications	*Optional.* Specifies the applications for which the masking criterion has to be applied. Start typing the application name, select the application from the type-ahead search results displayed, and click + Add to add one or more applications. For example: If there is a DataMasking(DM1) criteria created for application1 a second DataMasking(DM2) for application2 and a third DataMasking(DM3) with out any application, then for a request that comes from consumer1 the masking criteria DM1 is applied, for a request that comes from consumer2 DM2 is applied. If a request comes with out

Property	Description
	any application or from any other application except application1 and application2 DM3 is applied. You can use the delete icon to delete the added applications from the list.
XPath: Specifies the masking criteria for XPath expressions in the response messages.	
Masking Criteria	Click **Add masking criteria** and provide the following information and click **Add**: ■ **Query expression**. Specify the query expression that has to be masked or filtered. ■ **Masking Type**. Specifies the type of masking required. You select either **Mask** or **Filter**. Selecting **Mask** replaces the value with the given value (the default value being ********). Selecting **Filter** removes the field completely. ■ **Mask Value**. Appears only if you have selected the **Masking Type** as **Mask**. Provide a mask value. You can add multiple masking criteria. As **Query expression** and **Mask Value** properties support variable framework, you can use the available variables. In case of query expression, if you provide variable syntax, the XPath is applied on the payload using the value that is resolved from the variable given. For example, if you provide a query expression as `${request.headers.myxpath}` and the corresponding mask value as `${request.headers.var1}`, and if the incoming request header `myxpath` is configured with value `//ns:cardNumber`, then the card number derived from the payload is masked with the header value in `var1` . For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373. ■ **Namespace**. Specifies the following Namespace information: ■ **Namespace Prefix**. The namespace prefix of the payload expression to be validated. ■ **Namespace URI**. The namespace URI of the payload expression to be validated **Note:**

Property	Description
	You can add multiple namespace prefix and URI by clicking +Add.
JSONPath. This is applicable only for REST API. Specifies the masking criteria for JSONPath expressions in the response messages.	
Masking Criteria	Click **Add masking criteria** and provide the following information and click **Add**: ■ **Query expression**. Specify the query expression that has to be masked or filtered. ■ **Masking Type**. Specifies the type of masking required. You select either **Mask** or **Filter**. Selecting **Mask** replaces the value with the given value (the default value being ********). Selecting **Filter** removes the field completely. ■ **Mask Value**. Appears only if you have selected the **Masking Type** as **Mask**. Provide a mask value. As **Query expression** and **Mask Value** properties support variable framework, you can use the available variables. In case of query expression, if you provide variable syntax, the JSONPath is applied on the payload using the value that is resolved from the variable given. For example, if you provide a query expression as `${request.headers.myjsonpath}` and the corresponding mask value as `${request.headers.var1}` , and if the incoming request header `myjsonpath` is configured with value `$.cardNumber`, then the card number derived from the payload is masked with the header value in `var1` . For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
Regex. Specifies the masking criteria for regular expressions in the response messages.	
Masking Criteria	Click **Add masking criteria** and provide the following information and click **Add**: ■ **Query expression**. Specify the query expression that has to be masked or filtered. ■ **Masking Type**. Specifies the type of masking required. You select either **Mask** or **Filter**. Selecting **Mask** replaces the value with the given value (the default value being ********). Selecting **Filter** removes the field completely.

Property	Description
	■ **Mask Value**. Appears only if you have selected the **Masking Type** as **Mask**. Provide a mask value. As **Query expression** and **Mask Value** properties support variable framework, you can use the available variables. In case of query expression, if you provide variable syntax, the regex is applied on the payload using the value that is resolved from the variable given. For example, if you provide a query expression as `${request.headers.myregex}` and the corresponding mask value as `${request.headers.var1}`, then the regex is applied using the value configured in the request header `myregex` and the derived value is masked with the header value in `var1` . For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
Apply for transaction Logging	Select this option to apply masking criteria for transactional logs. When you select this option the transactional log for the response is masked on top of response sent to the client.
Apply for payload	Select this option to apply masking criteria for response payload in the following scenarios: ■ response received from the native service. ■ response sent to the client. **Note:** When you select this option it automatically masks the data in the transactional log.

CORS

The Cross-Origin Resource Sharing (CORS) mechanism supports secure cross-domain requests and data transfers between browsers and web servers. The CORS standard works by adding new HTTP headers that allow servers to describe the set of origins that are permitted to read that information.

This policy provides CORS support that uses additional HTTP headers to let a client or an application gain permission to access selected resources. An application or a client makes a cross-origin HTTP request when it requests a resource from a different domain, protocol, or port than the one from which the current request originated.

If you want to apply this policy in API Gateway at API level, make sure you have set the *watt.server.cors.enabled* property to *false*. CORS policy is not supported at scope-level. \

Note:
Both the Integration Server CORS policy and API Gateway CORS policy cannot coexist. When you enforce the CORS policy at Integration Server level, CORS enforcment is done for all requests. The preflight requests are handled by the Integration Server before even it reaches API Gateway.

This policy is applicable for REST, SOAP, and ODATA APIs.

The table lists the CORS response specifications, you can specify for this policy:

Property	Description
Allowed Origins	Specifies the origin from which the responses originating are allowed. syntax for the origin: *scheme*://*host*:*port* You can add multiple origins by clicking +Add. You can also provide Regular expressions for allowed origins. Allowed origins can also be specified in the Advanced section under Applications. Allowed origins of applications registered with this API are also allowed to access this API.
Allow Headers	Specifies the Headers that are allowed in the request. You can add multiple headers that are to be allowed by clicking +Add.
Expose Headers	Specifies the headers that be exposed to the user on request failure. You can add multiple headers that are to be allowed by clicking +Add.
Allow Credentials	Specifies whether API Gateway includes the Access-Control-Allow-Credentials header.
Allowed Methods	Specifies the methods that are allowed in the request. Specify one or more of the following: **GET**, **POST**, **PUT**, **DELETE**, and **PATCH**.
Max Age	Specifies the age for which the preflight response is valid.

A corresponding HTTP header is set for all the values above as per the specification. For additional information, see https://www.w3.org/TR/cors/.

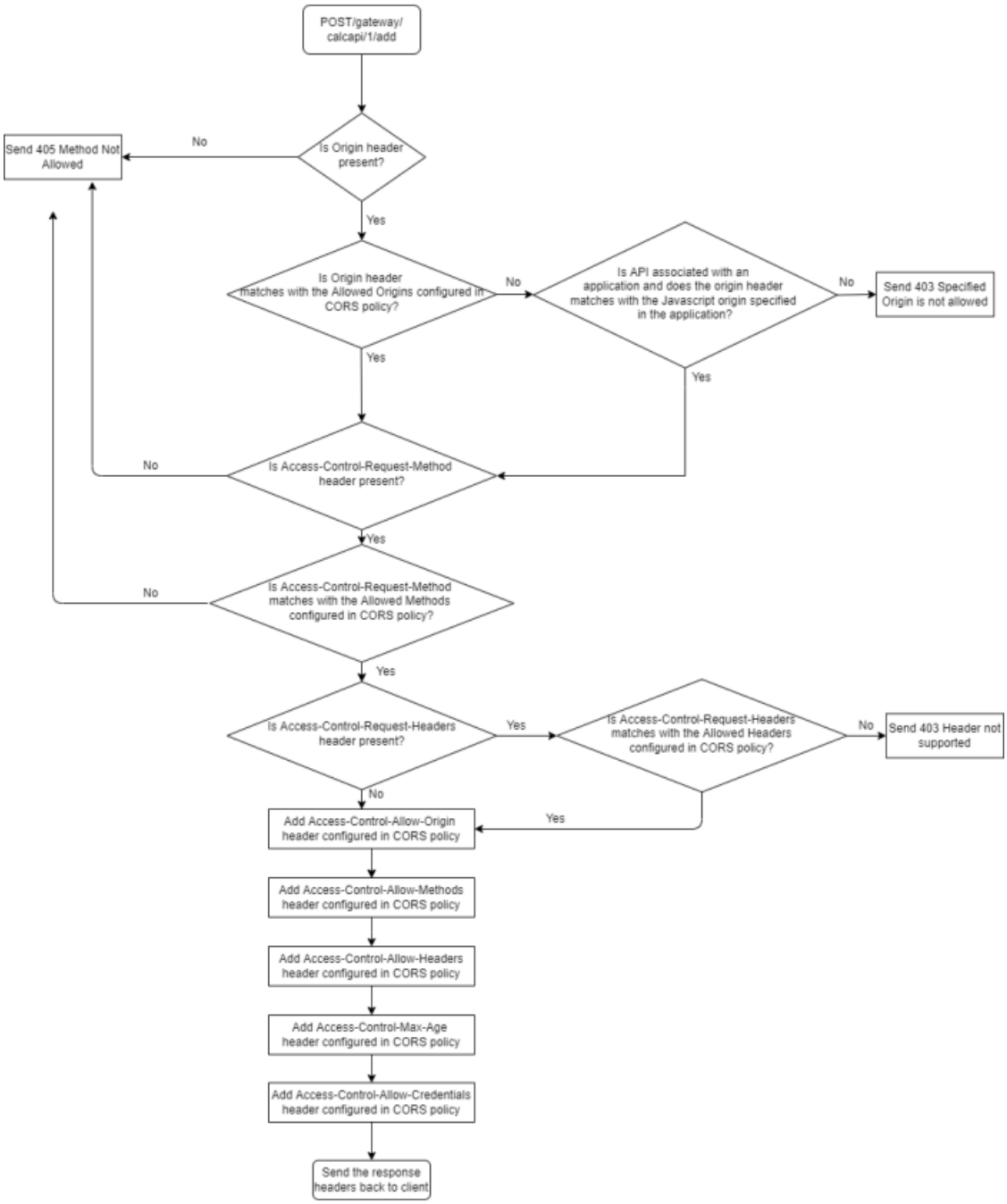
API Gateway handles CORS preflight request and CORS request differently. To know more about the work flow of CORS preflight and CORS request refer the respective flowchart.

CORS Preflight Request

A CORS preflight request is a HTTP request that a browser sends before the original CORS request to check whether the API Gateway server will permit the actual CORS request. CORS preflight request uses OPTIONS method and includes these headers as part of the request sent from the browser to API Gateway:

1. Origin
2. Access-Control-Request-Method
3. Access-Control-Request-Headers

The following flow chart explains the flow of the CORS preflight request received in API Gateway:



The following table shows the various use cases of the CORS preflight request originating from browser and how API Gateway responds to each CORS preflight requests:

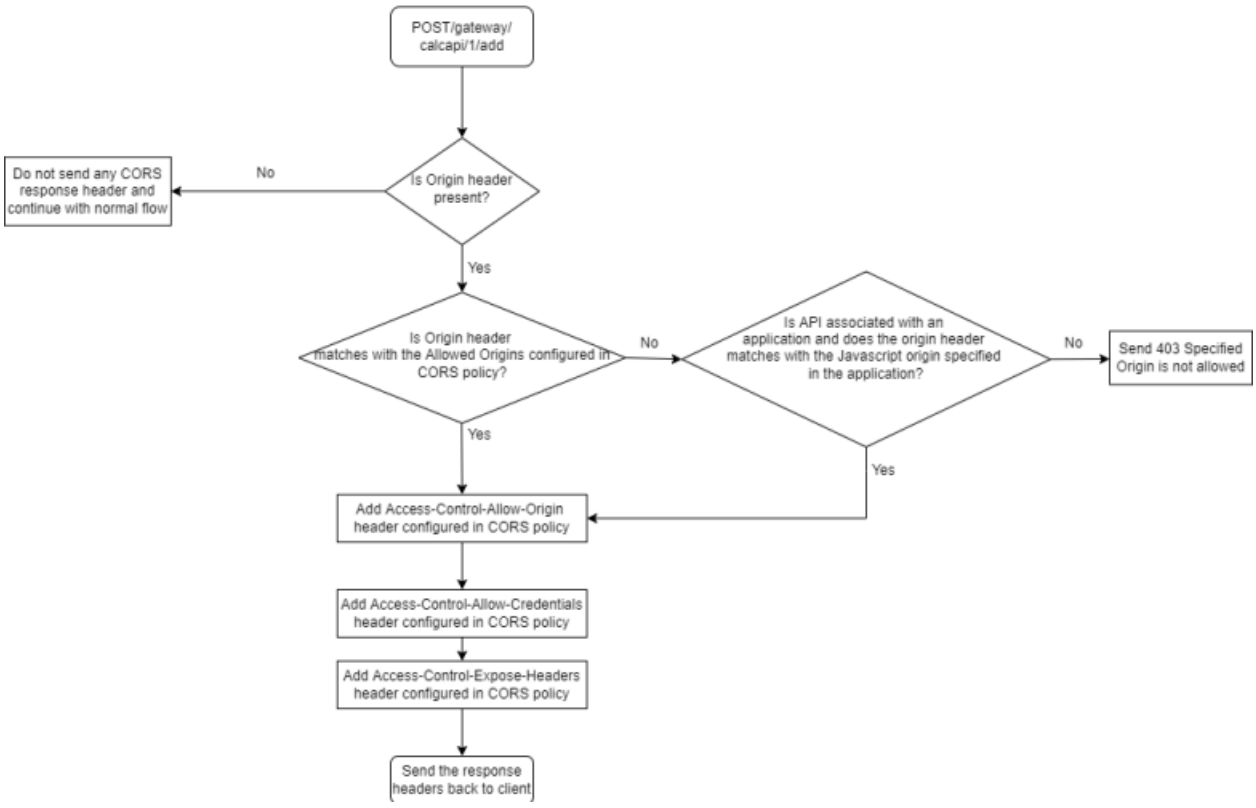
#	CORS Preflight request headers from browser	Configured CORS Policy in API Gateway	API Gateway sends the respective response to browser
1	Origin: **http://test.com** Access-Control-Request-Method : POST Access-Control-Request-Headers : test1,test2	Access-Control-Allow-Origin : **http://test2.com** Access-Control-Allow-Methods : POST,GET,PUT Access-Control-Allow-Headers : test1,test2	Sends **403 Specified Origin is not allowed** status, as the Origin header (http://test.com) from the browser does not match with the Access-Control-Allow-Origin (http://test2.com) configured in the CORS policy.
2	Origin: http://test2.com Access-Control-Request-Method : **DELETE** Access-Control-Request-Headers : test1,test2	Access-Control-Allow-Origin : http://test2.com Access-Control-Allow-Methods : **POST,GET,PUT** Access-Control-Allow-Headers : test1,test2	Sends **405 Method Not Allowed** status, as the Access-Control-Request-Method header (DELETE) from the browser does not match with the Access-Control-Allow-Methods (POST,GET,PUT) configured in the CORS policy.
3	Origin: http://test2.com Access-Control-Request-Method : POST Access-Control-Request-Headers : **test3**	Access-Control-Allow-Origin : http://test2.com Access-Control-Allow-Methods : POST,GET,PUT Access-Control-Allow-Headers : **test1,test2**	Sends **403 Header Not Supported**, as the Access-Control-Request-Headers header (test3) from the browser does not match with the Access-Control-Allow-Headers (test1,test2) configured in the CORS policy.
4	Origin: **http://test2.com** Access-Control-Request-Method : **POST** Access-Control-Request-Headers : **test1**	Access-Control-Allow-Origin : **http://test2.com** Access-Control-Allow-Methods : **POST** Access-Control-Allow-Headers : **test1**, test2 Access-Control-Max-Age: 100 Access-Control-Allow-Credentials : true	Sends **200 OK** status with the following headers: ■ Access-Control-Allow-Origin : http://test2.com ■ Access-Control-Allow-Methods : POST,GET,PUT ■ Access-Control-Allow-Headers : test1,test2 ■ Access-Control-Max-Age: 100

#	CORS Preflight request headers from browser	Configured CORS Policy in API Gateway	API Gateway sends the respective response to browser
		Access-Control-Expose-Headers : header1,header2	Since the origin, methods, and headers from the browser matches with configured CORS policy in API Gateway.
5	Origin: **http://test2.com** Access-Control-Request-Method : **POST** Access-Control-Request-Headers : **test1**	Access-Control-Allow-Origin : **http://test1.com** Access-Control-Allow-Methods : **POST** Access-Control-Allow-Headers : **test1**, test2 Access-Control-Max-Age: 100 Access-Control-Allow-Credentials : true Access-Control-Expose-Headers : header1,header2 In addition, if you have specified the Javascript origins in the application as **http://test2.com**	Sends **200 OK** status with the following headers: ■ Access-Control-Allow-Origin : http://test2.com ■ Access-Control-Allow-Methods : POST,GET,PUT ■ Access-Control-Allow-Headers : test1,test2 ■ Access-Control-Max-Age: 100 Even though the origin header from the browser does not match with configured CORS policy, it matches with the configured javascript origins in the application.

CORS Request

A CORS request is a HTTP request that includes an *Origin* header. When API Gateway receives the CORS request, the *Origin* header in the CORS request is verified against the *Access-Control-Allow-Origin* configured in the CORS policy, if it matches then API Gateway allows to access the resources.

The following flow chart explains the flow of the CORS request received in API Gateway:



The following table shows the various use cases of the CORS request originating from browser and how API Gateway responds to each CORS requests:

#	CORS Request headers from browser	Configured CORS Policy in API Gateway	API Gateway sends the respective response to browser
1	Origin: **http://test.com**	Access-Control-Allow-Origin : **http://test2.com** Access-Control-Allow-Methods : POST,GET,PUT Access-Control-Allow-Headers : test1,test2 Access-Control-Max-Age: 100 Access-Control-Allow-Credentials : true Access-Control-Expose-Headers : header1,header2	Sends **403 Specified Origin is not allowed** status, as the Origin header (http://test.com) from the browser does not match with the Access-Control-Allow-Origin (http://test2.com) configured in the CORS policy.
2	Origin: **http://test2.com**	Access-Control-Allow-Origin : **http://test2.com**	Sends **200 OK** status with the following headers:

#	CORS Request headers from browser	Configured CORS Policy in API Gateway	API Gateway sends the respective response to browser
		Access-Control-Allow-Methods : POST,GET,PUT Access-Control-Allow-Headers : test1,test2 Access-Control-Max-Age: 100 Access-Control-Allow-Credentials : true Access-Control-Expose-Headers : header1,header2	■ Access-Control-Allow-Origin : http://test2.com ■ Access-Control-Allow-Credentials : true ■ Access-Control-Expose-Headers : header1,header2 ■ Access-Control-Allow-Credentials : true Since the Origin header (http://test2.com) from the browser matches with the Access-Control-Allow-Origin (http://test2.com) configured CORS policy in API Gateway.
3	Origin: **http://test2.com** Access-Control-Request-Method : **POST** Access-Control-Request-Headers : **test1**	Access-Control-Allow-Origin : **http://test1.com** Access-Control-Allow-Methods : **POST** Access-Control-Allow-Headers : **test1**, test2 Access-Control-Max-Age: 100 Access-Control-Allow-Credentials : true Access-Control-Expose-Headers : header1,header2 In addition, if you have specified the Javascript origins in the application as **http://test2.com**	Sends **200 OK** status with the following headers: ■ Access-Control-Allow-Origin : http://test2.com ■ Access-Control-Allow-Methods : POST,GET,PUT ■ Access-Control-Allow-Headers : test1,test2 ■ Access-Control-Max-Age: 100 ■ Access-Control-Allow-Credentials : true Even though the origin header from the browser does not match with configured CORS policy, it matches with the configured javascript origins in the application.

Note:

- If native service supports CORS mechanism and if you have not configured the CORS policy in API Gateway, then API Gateway goes to pass-through security mode and forwards the CORS request to the native service.
- If native service supports CORS mechanism and if you have also configured the CORS policy in API Gateway, then API Gateway takes precedence in handling the CORS request.

Error Handling

The policy in this stage enables you to specify the error conditions, lets you determine how these error conditions are to be processed. You can also mask the data while processing the error conditions. The policies included in this stage are:

- Conditional Error Processing
- Data Masking
- Custom Extension

  Custom Extension policies allow you to handle requirements that might not be provided by the out-of-the-box policies. You can add these custom extensions into API Gateway policy stages. To learn more about Custom Extension, see “Custom Policy Extension” on page 383.

Conditional Error Processing

Error Handling is the process of passing an exception message issued as a result of a run-time error to take any necessary actions. This policy returns a custom error message (and the native provider's service fault content) to the application when the API Gateway or native provider returns a service fault. You can configure conditional error processing and use variables to create custom error messages.

The table lists the properties that you can specify for this policy:

Property	Description
Condition	Conditions are used to specify when the policy has to be executed. You can add multiple conditions with logical operators. Available values are: ■ **AND**. API Gateway transforms the error responses that comply with all the configured conditions ■ **OR**. This is selected by default. API Gateway transforms the error responses that comply with any one configured condition. Click **Add Condition** and provide the following information and click  . ■ **Variable**. Specifies the variable type with a syntax.

Property	Description
	■ **Operator**. Specifies the operator to use to relate variable and the value. You can select one of the following: ■ **Equals** ■ **Equals ignore case** ■ **Not equals** ■ **Not equals ignore case** ■ **Contains** ■ **Not Contains** ■ **Exists** ■ **Not Exists** ■ **Greater Than** ■ **Less Than** ■ **Value**. Specifies a plain value or value with a syntax. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
Pre-Processing. Specifies how the error response is to be processed before this policy processes it.	
Invoke webMethods Integration Server Service	Specify the webMethods IS service to pre-process the error message. Provide the following information ■ **webMethods IS Service**. Specify the webMethods IS service to be invoked to pre-process the error messages. You can add multiple entries for webMethods IS service by clicking  . ■ **Run as User**. Specifies the authentication mode to invoke the IS service. If this field is left blank the incoming credentials of the user, identified by API Gateway, are used to authenticate and invoke the IS service. You can also specify a particular user, you want API Gateway to use to run the IS service. ■ **Comply to IS Spec**. Mark this as `true`, if you want the input and the output parameters to comply to the IS Spec present in pub.apigateway.invokeISService.specifications folder in the wmAPIGateway package.

Property	Description
	■ **webMethods IS Service alias**. Start typing the webMethods alias name and select the alias from the type-ahead search results displayed to add one or more aliases.
XSLT Transformation	Provide the XSLT file and feature you want to use to transform the service error response. Click **Browse** to select the XSLT file and upload it. Provide the following information for the XSLT feature: ■ **Feature Name**. Specifies the name of the XSLT feature. ■ **Feature Value**. Specifies the value for the feature. You can add multiple entries for feature name and value by clicking + Add. **Note:** API Gateway supports XSLT 1.0 and XSLT 2.0.
Transformation Configuration. Specifies various transformations to be configured.	
Header Transformation	Customizes the list of headers in the error response that is sent to the client. You can add or modify header parameters by providing the following information: ■ **Variable**. Specifies the variable type with a syntax. ■ **Value**. Specifies a plain value or value with a syntax. You can add multiple variables and corresponding values by clicking + Add. You can remove any header by typing the plain value or value with a syntax. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
Status Transformation	Specifies the status transformation to be configured for the error responses. Provide the following information: ■ **Code**. Specifies the status code that is sent in the response to the client.

Property	Description
	For example if you want to transform status code as 403, provide 403 in the **Code** field. ■ **Message**. Specifies the Status message that is sent in the response to the client. For example *The data you are looking for is not found* can be used to transform the original *404 Not Found* status message.
Define custom variables	Defines a custom variable name to a complex variable expression or constant value. This can be particularly useful when you want to use this complex expression multiple times in the error payload transformation or when you want to use a short notation for a complex variable expression. Provide the following information: ■ **Variable**. Specifies the variable type with a syntax. ■ **Value**. Specifies a plain value or value with a syntax. For example if you provide a variable as id and the corresponding value as ${response.payload.jsonPath[$.id]}, this creates a custom variable that can be used in failure message transformation. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
Failure Message. Specifies the custom failure message format that API Gateway should send to the application.	
Failure Messages	Specifies the payload transformation to be configured for the error responses. ■ Click **text** and specify the payload to use to transform the error response messages as required. ■ Click **json** and specify the payload to use to transform the error response messages as required. ■ Click **xml** and specify the payload to use to transform the error response messages as required. **Note:** For a SOAP API, select the type **text** and provide the failure message to be included in the faultstring of the SOAP response. Failure message in type **json**, **xml** are not used for the SOAP response.

Property	Description
	As this property supports variable framework, to transform the error response messages you can make use of the available variables in addition to the custom variables defined in this policy. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373. ■ Click **Send Native Provider Fault Message** to send the native failure message to the application without applying payload transformation. ■ This field is not applicable for APIs when they participate in API mashups.
Post-Processing. Specifies how the error response sent by the native service is to be processed before sending the same to the application.	
Invoke webMethods Integration Server Service	Specify the webMethods IS Service for post-processing the error message. Provide the following information ■ **webMethods IS Service**. Specify the webMethods IS service to be invoked to post-process the error messages. You can add multiple entries for webMethods IS service by clicking  . **Note:** The pipeline variables in the Invoke IS service include only the response headers and they do not include request headers (sent by the client or the ones added during the request transformation step). To access the request headers in the Invoke IS Service flow, use the **pub.flow:getTransportInfo** service. ■ **Run as User**. Specifies the authentication mode to invoke the IS service. If this field is left blank the incoming credentials of the user, identified by API Gateway, are used to authenticate and invoke the IS service. You can also specify a particular user, you want API Gateway to use to run the IS service. ■ **Comply to IS Spec**. Mark this as `true` if you want the input and the output parameters to comply to the IS Spec present in pub.apigateway.invokeISService.specifications folder in wmAPIGateway package. **Note:**

Property	Description
	The response headers sent by the native service and API Gateway does not include any info regarding the request headers or the headers added during the request transformation. Use the pub.flow:getTransportInfo flow service to retrieve the request headers in the pipeline. ■ **webMethods IS Service alias**. Start typing the webMethods alias name and select the alias from the type-ahead search results displayed to add one or more aliases.
XSLT Transformation	Provide the XSLT file that you want to use to transform the service error response. Provide the following information for the XSLT feature: ■ **Feature Name**. Specifies the name of the XSLT feature. ■ **Feature Value**. Specifies the value for the feature. You can add multiple entries for feature names and values by clicking  . **Note:** API Gateway supports XSLT 1.0 and XSLT 2.0.

Transformation Metadata. Specifies the metadata for transformation of the error responses received from the native API. For example, the namespaces configured in this section can be used when you provide the syntax for XPath ${response.payload.xpath} For example: ${response.payload.xpath[//ns:emp/ns:empName]}

Namespace	Specifies the namespace information to be configured for transformation. This is applicable only for XML transformation. Provide the following information: ■ **Namespace Prefix**. The namespace prefix of the payload expression to be validated. For example, specify the namespace prefix as `SOAP_ENV`. ■ **Namespace URI**. The namespace URI of the payload expression to be validated. For example, specify the namespace URI as `http://schemas.xmlsoap.org/soap/envelope/`. This declaration defines SOAP_ENV as an alias for the namespace: http://schemas.xmlsoap.org/soap/envelope/. **Note:**

Property	Description
	You can add multiple namespace prefixes and URIs by clicking + Add.

Data Masking

Data masking is a technique whereby sensitive data is obscured in some way to render it safe and to protect the actual data while having a functional substitute for occasions when the real data is not required.

This policy is used to mask sensitive data in the custom error messages being processed and sent to the application. Fields can be masked or filtered in the error messages. You can configure the masking criteria as required for the XPath, JPath, and Regex expressions. This policy can also be applied at the API scope level.

Note:
API Gateway does not mask the payload, if the payload is sent as a stream.

The table lists the masking criteria properties that you can configure to mask the data in the request messages received:

Property	Description
Consumer Applications	Specifies the applications for which the masking criterion has to be applied. Start typing the application name, select the application from the type-ahead search results displayed, and click + Add to add one or more applications. You can use the delete icon to delete the added applications from the list.
XPath. Specifies the masking criteria for XPath expressions in the error messages.	
Masking Criteria	Click **Add masking criteria** and provide the following information and click **Add**: ■ **Query expression**. Specify the query expression that has to be masked or filtered. ■ **Masking Type**. Specifies the type of masking required. You select either **Mask** or **Filter**. Selecting **Mask** replaces the value with the given value (the default value being ********). Selecting **Filter** removes the field completely.

Property	Description
	■ **Mask Value**. Appears only if you have selected the **Masking Type** as **Mask**. Provide a mask value. You can add multiple masking criteria. As **Query expression** and **Mask Value** properties support variable framework, you can use the available variables. In case of query expression, if you provide variable syntax, the XPath is applied on the payload using the value that is resolved from the variable given. For example, if you provide a query expression as `${request.headers.myxpath}` and the corresponding mask value as `${request.headers.var1}`, and if the incoming request header `myxpath` is configured with value `//ns:cardNumber`, then the card number derived from the payload is masked with the header value in `var1` . For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373. ■ **Namespace**. Specifies the following Namespace information: ■ **Namespace Prefix**. The namespace prefix of the payload expression to be validated. ■ **Namespace URI**. The namespace URI of the payload expression to be validated **Note:** You can add multiple namespace prefix and URI by clicking +Add.
JSONPath. This is applicable only for REST API. Specifies the masking criteria for JSONPath expressions in the error messages.	
Masking Criteria	Click **Add masking criteria** and provide the following information and click **Add**: ■ **Query expression**. Specify the query expression that has to be masked or filtered. ■ **Masking Type**. Specifies the type of masking required. You select either **Mask** or **Filter**. Selecting **Mask** replaces the value with the given value (the default value being ********). Selecting **Filter** removes the field completely. ■ **Mask Value**. Appears only if you have selected the **Masking Type** as **Mask**. Provide a mask value.

Property	Description
	As **Query expression** and **Mask Value** properties support variable framework, you can use the available variables. In case of query expression, if you provide variable syntax, the JSONPath is applied on the payload using the value that is resolved from the variable given. For example, if you provide a query expression as `${request.headers.myjsonpath}` and the corresponding mask value as `${request.headers.var1}` , and if the incoming request header `myjsonpath` is configured with value `$.cardNumber`, then the card number derived from the payload is masked with the header value in `var1` . For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
Regex. Specifies the masking criteria for regular expressions in the error messages.	
Masking Criteria	Click **Add masking criteria** and provide the following information and click **Add**: ■ **Query expression**. Specify the query expression that has to be masked or filtered. ■ **Masking Type**. Specifies the type of masking required. You select either **Mask** or **Filter**. Selecting **Mask** replaces the value with the given value (the default value being ********). Selecting **Filter** removes the field completely. ■ **Mask Value**. Appears only if you have selected the **Masking Type** as **Mask**. Provide a mask value. As **Query expression** and **Mask Value** properties support variable framework, you can use the available variables. In case of query expression, if you provide variable syntax, the regex is applied on the payload using the value that is resolved from the variable given. For example, if you provide a query expression as `${request.headers.myregex}` and the corresponding mask value as `${request.headers.var1}` , then the regex is applied using the value configured in the request header `myregex` and the derived value is masked with the header value in `var1` . For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

Property	Description
Apply for transaction Logging	Select this option to apply masking criteria for transactional logs. When you select this option the transactional log for the response is masked on top of response sent to the client.
Apply for payload	Select this option to apply masking criteria for payload. When you select this option the payload in the response sent to the client is masked. **Note:** When you select this option it automatically masks the data in the transactional log.

Policy Validation and Dependencies

The following table lists the following:

- Policy dependencies; whether a policy must be used in conjunction with another particular policy.
- Conflicting or incompatible policies.
- Whether a policy can be included multiple times in a single API. If a policy cannot be included multiple times in a single API, API Gateway selects one, depending on the precedence of the policy at the enforcement level, for the effective policy and processes at run-time.

Policy	Applicable API Type	Dependent Policy	Mutually Exclusive Policy	Can include multiple times in an API?
Authorize User	REST SOAP	Identify & Authorize	None.	No. API Gateway includes only one policy in the effective policy.
Conditional Error Processing	REST SOAP	None.	None.	Yes. API Gateway includes all Conditional Error Processing policies in the effective policy.
Conditional Routing	REST SOAP	None.	Straight Through Routing, Load Balancer Routing, Dynamic Routing, Content-based Routing	No. API Gateway includes only one policy in the effective policy.

Policy	Applicable API Type	Dependent Policy	Mutually Exclusive Policy	Can include multiple times in an API?
Content-based Routing	REST SOAP	None.	Straight Through Routing, Load Balancer Routing, Dynamic Routing, Conditional Routing	No. API Gateway includes only one policy in the effective policy.
Custom HTTP Header	REST SOAP	None.	None.	No. API Gateway includes only one policy in the effective policy.
Data Masking (Error Handling)	REST SOAP	None.	None.	No. API Gateway includes only one policy in the effective policy.
Data Masking (Response Processing)	REST SOAP	None.	None.	No. API Gateway includes only one policy in the effective policy.
Data Masking (Request Processing)	REST SOAP	None.	None.	No. API Gateway includes only one policy in the effective policy.
Dynamic Routing	REST SOAP	None.	Straight Through Routing, Load Balancer Routing, Content-based Routing, Conditional Routing	No. API Gateway includes only one policy in the effective policy.
Enable HTTP / HTTPS	REST SOAP GraphQL	None.	None.	No. API Gateway includes only one policy in the effective policy.
Enable JMS / AMQP	REST SOAP	None	None	No. API Gateway includes only one policy in the effective policy.

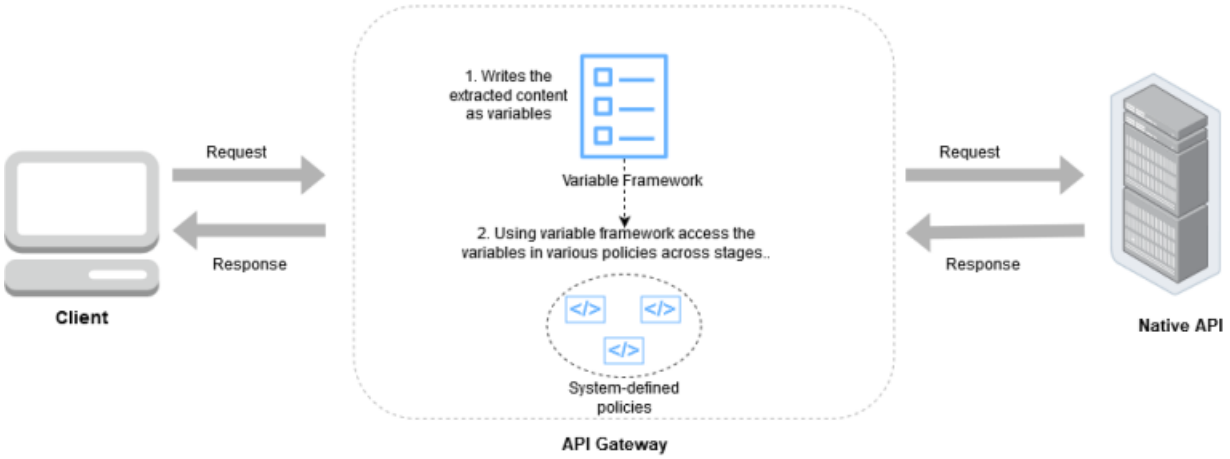
Policy	Applicable API Type	Dependent Policy	Mutually Exclusive Policy	Can include multiple times in an API?
Identify & Authorize	REST SOAP GraphQL	Inbound Auth - Message policy is required if **Identification Type** is configured as **WS Security Username Token** or WS Security X.509 Certificate or **Kerberos Token** for SOAP-based APIs.	None.	No. API Gateway includes only one policy in the effective policy.
Inbound Auth - Message	SOAP	None.	None.	No. API Gateway includes only one policy in the effective policy.
Invoke webMethods IS (Response Processing)	REST SOAP	None.	None.	Yes. API Gateway includes all Invoke webMethods IS policies in the effective policy.
Invoke webMethods IS (Request Processing)	REST SOAP	None.	None.	Yes. API Gateway includes all Invoke webMethods IS policies in the effective policy.
JMS/AMQP REST Properties	REST	JMS/AMQP REST Routing	None	No. API Gateway includes only one policy in the effective policy.
JMS/AMQP SOAP Properties	SOAP	JMS/AMQP SOAP Routing	None.	No. API Gateway includes only one policy in the effective policy.
JMS/AMQP REST Routing	REST	None	Straight Through Routing, Dynamic Routing, Content-based Routing, Conditional Routing	No. API Gateway includes only one policy in the effective policy.

Policy	Applicable API Type	Dependent Policy	Mutually Exclusive Policy	Can include multiple times in an API?
JMS/AMQP REST Routing	SOAP	None.	Straight Through Routing, Dynamic Routing, Content-based Routing, Conditional Routing	No. API Gateway includes only one policy in the effective policy.
Load Balancer Routing	REST SOAP	None.	Straight Through Routing, Dynamic Routing, Content-based Routing, Conditional Routing	No. API Gateway includes only one policy in the effective policy.
Log Invocation	REST SOAP GraphQL	None.	None.	Yes. API Gateway includes all Log Invocation policies in the effective policy.
Monitor Performance	REST SOAP	None.	None.	Yes. API Gateway includes all Monitor Performance policies in the effective policy.
Monitor SLA	REST SOAP	Identify & Authorize	None.	Yes. API Gateway includes all Monitor Service Level Agreement policies in the effective policy.
Outbound Auth - Message	SOAP	None.	None.	No. API Gateway includes only one policy in the effective policy.
Outbound Auth - Transport	REST SOAP	None.	None.	No. API Gateway includes only one policy in the effective policy.
Response Transformation	REST SOAP	None.	None.	Yes. API Gateway includes all XSLT Transformation

Policy	Applicable API Type	Dependent Policy	Mutually Exclusive Policy	Can include multiple times in an API?
				policies in the effective policy.
Request Transformation	REST SOAP	None.	None.	Yes. API Gateway includes all XSLT Transformation policies in the effective policy.
Service Result Cache	REST SOAP	None.	None.	No. API Gateway includes only one policy in the effective policy.
Set Media Type	REST	None.	None.	No. API Gateway includes only one policy in the effective policy.
Straight Through Routing	REST SOAP GraphQL	None.	Load Balancer Routing, Dynamic Routing, Content-based Routing, Conditional Routing	No. API Gateway includes only one policy in the effective policy.
Traffic Optimization	REST SOAP	Identify & Authorize	None.	Yes. API Gateway includes all Traffic Optimization policies in the effective policy.
Validate API Specification (Response Processing)	REST SOAP	None.	None.	No. API Gateway includes only one policy in the effective policy.
Validate API Specification (Request Processing)	REST SOAP GraphQL	None.	None.	No. API Gateway includes only one policy in the effective policy.

Variable Framework

This figure depicts how the variable framework is used to access the variables in various policies across stages.



Note:
Any variables that access the request or response payloads cannot be used in the variable framework, if the payload is sent as a stream.

The following table summarizes the keywords that are used to define the variable syntaxes:

Variable keyword	Description
`paramStage`	Defines the stage of the system defined policy. Possible values are: ■ `request` ■ `response`
`paramType`	Defines the specific parameter of the request or response. Possible values are: ■ `payload` ■ `headers` ■ `query` ■ `path` ■ `httpMethod` ■ `statusCode` ■ `statusMessage`

Variable keyword	Description
`queryType`	Defines the query that can be applied over string elements like payload. Possible values are: - `xpath` - `jsonPath` - `regex`

The following variable types are available in the request or response stages:

- `${paramStage.paramType}`

  You can use this syntax to access the following string variables: `path`, `statusCode`, `statusMessage`, `httpMethod`.

  Examples: `${request.path}`, `${response.statusCode}`

- `${paramStage.paramType.paramName}`

  You can use this syntax to access map types, such as query, headers, and path.

  Example: `${request.query.var1}`, `${response.header.Content-Type}`, `${request.path.name}`.

- `${paramStage.paramType.queryType[queryValue]}`

  You can use this syntax to access the payload. Examples:

  - `${request.payload.xpath[//ns:emp/ns:empName]}`

    Where "`//ns:emp/ns:empName`" is the XPath to be applied on the payload if contentType is application/xml, text/xml, or text/html.

  - `${response.payload.jsonPath[$.cardDetails.number]}`

    Where `$.cardDetails.number` is the jsonPath to be applied on payload if contentType is application/json or application/json/badgerfish.

  - `${request.payload.regex[[0-9]+]}`

    Where `[0-9]+` is the regular expression to be applied on the payload if contentType is text/plain.

  - `${request.isSoapToRest}` or `${response.isSoapToRest}`

    This variable returns True if the current invoke is REST invoke for a SOAP API. Else it returns False.

  Note:
  While `xpath` and `jsonPath` are applicable only to payload, `regEx` can be used with both `payload` and `path`.

- `${paramStage[stepName].paramType.paramName]}`

You can use this syntax to access the header or payload in the request or response stage.

Example:

Variable: `${response.headers.id}`

Value: `${response[customExtension].payload.jsonPath[$.id]}`

This transformation adds a header to the response with the name id, and its value is derived from the JSON payload that is sent from the external callout as per the JSON path.

The following sections summarize the variables that are available in API Gateway as part of variable framework template in addition to the existing predefined system context and custom context variables:

Request Variables

Variables that allow you to extract and reuse values in the request processing stage.

Variable Syntax	Description
${request.headers.NAME} Example: `${request.headers.Content-Type}`	Gets the value of the header name in the request.
$ {request.query.NAME} Example: `${request.query.var1}`	Gets the value of the query name in the request.
${request.path}	Gets the value of the path in the request.
${request.path.regex[EXPR]} Example: `${request.path.regex[0]}`	Gets the value of the path in the request.
${request.httpMethod}	Gets the method in the request.
${request.payload.xpath[EXPR]} Example:`${request.payload.xpath[//ns:emp/ns:empName]}`, where //ns:emp/ns:empName is the xpath to be applied on the payload if contentType is application/xml.	Gets the value after applying a xpath expression on the request path. **Note:** The namespace URI for the prefixes you have configured in the xpath expression are resolved using namespaces configured in the metadata section in the policy or using the namespaces configured through *XpathNamespaces* custom variable in the custom extension policy.
${request.payload.jsonPath[EXPR]}	Gets the value after applying a JSON expression on the request path.

Variable Syntax	Description
Example: `${request.payload.jsonPath[$.cardDetails.number]}` where $.cardDetails.number is the jsonPath to be applied on the payload if contentType is application/json. Provide the following variable, if there is a blank space in the parameter name **${request.payload.jsonPath[$.['param name']]}** For example, if the parameter name is *first name*, then provide the variable as `${request.payload.jsonPath[$.['first name']]}`.	
${request.payload.regex[EXPR]} Example: `${request.payload.regex[[0-9]+]}` where [0-9]+ is the regex to be applied on the payload if contentType is text/plain	Gets the value after applying a regular expression on the request path.
${request.authorization.clientId}	Gets the value of the client ID identified from the authorization header by the configured IAM policy. This value is available only if the relevant IAM policy is configured.
${request.authorization.issuer}	Gets the value of the issuer identified from the authorization header by the configured IAM policy. This value is available only if the relevant IAM policy is configured.
${request.authorization.userName}	Gets the value of the username identified from the authorization header by the configured IAM policy. This value is available only if the relevant IAM policy is configured.
${request.authorization.authHeader}	Gets the value of the authorization header by the configured IAM policy. This value is available only if the relevant IAM policy is configured. **Note:** If the authorization header has bearer tokens (such as OAuth, OpenID, or JWT), then you cannot use this variable. In such cases, Software AG recommends to use the

Variable Syntax	Description
	${request.authorization.incomingToken} variable.
${request.authorization.apiKey}	Gets the value of the API key from the authorization header by the configured IAM policy. This value is available only if the relevant IAM policy is configured.
${request.authorization.incomingToken}	Gets the value of the incoming token from the authorization header by the configured IAM policy. This value is available only if the relevant IAM policy is configured.
${request.authorization.audience}	Gets the value of the audience from the authorization header by the configured IAM policy. This value is available only if the relevant IAM policy is configured.
${request.authorization.claims.CLAIM_NAME} Example: ${request.authorization.claims.emp.company}	Gets the value for the claim name from the claims identified from the Authorization header by the configured IAM policy. This value is available only if the relevant IAM policy is configured
${request.correlationID}	Gets the correlation ID for this request.
${request.application.id}	Gets the ID of the application identified for this request.
${request.application.name}	Gets the name of the application identified for this request.
${request.application.version}	Gets the version ID of the application identified for this request.
${request.application.claims.CLAIM_NAME} Example:`${request.application.claims.sample}`	Gets the value of the claim name for the claims identifier configured in the application identified for this request.
${request.application.partnerId}	Gets the partner ID of the application identified for this request.
${request.application.description}	Gets the description of the application identified for this request.
${request.application.hostname[NUMBER]} Example: `${request.application.hostname[0]}`	Gets the value of the hostname identifier in the specified index for the application identified for this request.

Variable Syntax	Description
${request.application.payloadIdentifier[NUMBER]} Example:`${request.application.payloadIdentifier[1]}`	Gets the value of the payload identifier in the specified index for the application identified for this request.
${request.application.team[NUMBER]} Example: `${request.application.team[0]}`	Gets the value of the team identifier in the specified index for the application identified for this request.
${request.application.token[NUMBER]} Example:`${request.application.token[1]}`	Gets the value of the token identifier in the specified index for the application identified for this request.
${request.application.username[NUMBER]} Example:`${request.application.username[0]}`	Gets the value of the username identifier in the specified index for the application identified for this request.
${request.application.wssUsername[NUMBER]} Example:`${request.application.wssUsername[0]}`	Gets the value of the wssUsername identifier in the specified index for the application identified for this request.
${request.application.headers.HEADER_NAME} Example:`${request.application.headers.Accept}`	Gets the value of the header name for the headers identifier configured in the application identified for this request.
${request.payload.native.xpath [EXPR]}	In SOAP to REST context, this variable returns the SOAP request to be sent to the native API.

Response variables

Variables that allow you to extract and reuse values in the response processing stage.

Variable Syntax	Description
${response.headers.NAME} Example: `${response.headers.Accept}`	Gets the value of the header name in the response.
${response.statusCode}	Gets the value for the status code for the response.
${response.statusMessage}	Gets the value for the status message in the response
${response.payload.xpath[EXPR]} Example:`${response.payload.xpath[//ns:emp/ns:empName]}` where //ns:emp/ns:empName is the xpath to be applied on the payload if contentType is application/xml	Gets the value of the payload from the specified xpath of the response. **Note:**

Variable Syntax	Description
	The namespace URI for the prefixes you have configured in the xpath expression are resolved using namespaces configured in the metadata section in the policy or using the namespaces configured through *XpathNamespaces* custom variable in the custom extension policy.
${response.payload.jsonPath[EXPR]} Example:`${response.payload.jsonPath[$.cardDetails.number]}` where $.cardDetails.number is the jsonPath to be applied on the payload if contentType is application/json	Gets the value of the payload from the specified jsonPath of the response.
${response.payload.regex[EXPR]} Example: `${ response.payload.regex[[0-9]+]}` where [0-9]+ is the regex to be applied on the payload if contentType is text/plain	Gets the value of the payload from the specified regex of the response.
${response.payload.native.xpath [EXPR]}	In SOAP to REST context, this variable returns the native SOAP response, returned by the native SOAP API.

API Gateway evaluates and supports the array expressions in JSON path.

Example: Consider the following payload.

```
{
"firstName":"John",
"lastName":"doe",
"age":26,
"address":
{"streetAddress":"naist street","city":"Nara","postalCode":"630-0192"}
,
"phoneNumbers":[
{"type":"iPhone","number":"0123-4567-8888"}
,
{"type":"home","number":"0123-4567-8910"}
]
}
```

Following are the responses for the expressions after evaluating the array expressions in JSON path.

Expressions	Response
$.phoneNumbers[1].type	"home"
$.phoneNumbers[0,1].type or $.phoneNumbers[:2].type	["iPhone","home"]
$.phoneNumbers[0,1] or $.phoneNumbers[:2]	[{"type":"iPhone","number":"0123-4567-8888"} \{"type":"home","number":"0123-4567-8910"}]
$..firstName	["John"]
$.firstName	"John"
$.address.city	"Nara"

System Context Variables

API Gateway provides predefined system context variables and the values of these variables are extracted from the client request.

Variable Syntax	Description
${apiId}	Get the value of the API ID.
${apiName}	Get the name of the API.
${apiVersion}	Get the version of the API.
${packageId}	Get the value of the package ID.
${planId}	Get the value of the plan ID.
${customTransactionFields.FIELD_NAME} Example: `${customTransactionFields.sample}`	Provides you an option to get or set custom fields to the transactional events for this request. To set the custom fields, you can configure the *customTransactionFields.FIELD_NAME* custom variable in **Custom Extension** policy.
${providerTime}	Gets the time taken in milliseconds between the request sent to native server and response received from native server for this transaction.
${date}	Gets the date when the request gets invoked.
${dynamicEndpoint}	Gets the value of the ROUTING_ENDPOINT context variable set using pub.apigateway.ctxvar:setContextVariable
${time}	Gets the time when the request gets invoked.

Variable Syntax	Description
${user}	Gets the value of the user ID who sends the request.
${inboundHttpMethod} Example: `GET`	Gets the value the HTTP method used by the client to send the request.
${routingMethod} Example: `POST`	Gets the value of the HTTP method used by the API Gateway in the routing policy to send the request to native API.
${InboundContentType} Example: `application/json`	Gets the content type of the request.
${inboundAccept} Example: `*/*`	Gets the accept header in the incoming request from the client.
${inboundProtocol}	Gets the protocol of the request.
${inboundRequestURI} For example, if the API is invoked: `http://host:port/gateway/API` then the expected value of this variable would be `/gateway/API`. For a REST API, the URL also includes query string parameters. For example, if the following API is invoked: `http://host:port/gateway/cars?vin=1234` the expected value of this variable would be `/gateway/cars?vin1234`.	A partial reference to an API (for HTTP and HTTPS only). The protocol, host and port are not part of the value.
${inboundIP} Example: `210.178.9.0`	Gets the value of the client IP address used to send the request.
${gatewayHostname} Example: `uk.myhost.com`	Gets the API Gateway host name.
${gatewayIP} Example: `198.168.1.9`	Gets API Gateway IP address.
${operationName} Example: `addInts`	Gets the value of API operation selected from the request. Operation names are available only for SOAP APIs. It is empty for REST API.

Variable Syntax	Description
${nativeEndpoint} Example: `http://host:port/Service`	Gets the value of the native endpoints from the request. It returns value only after executing the routing policy.

In addition, the variable framework also supports the following variables:

- **${jms.headers.NAME}**
- **${jms.query.NAME}**
- **${jms.path}**
- **${jms.path.regex[EXPR]}**
- **${jms.httpMethod}**
- **${jms.payload.xpath[EXPR]}**
- **${jms.payload.jsonPath[EXPR]}**
- **${jms.payload.regex[EXPR]}**
- **${jms.statusCode}**

Note:
You can use these variables when you want to use JMS/AMQP so that transformation can be applied for the JMS/AMQP values. For example, if you set the path parameter as `jms.path.petid`and the corresponding value as `jms.header.h1`, then if the request contains the header value h1, the value h1 is replaced by the path parameter petid.

Enhancements to Variable Framework Support

Until API Gateway version 10.5, the variable framework was supported in API Mashup, Request Transformation, Response Transformation, Conditional Error Processing, and Custom Extension policies.

In API Gateway version 10.7 the existing variable framework is enhanced to support the following use cases:

- Simple aliases can be accessed across all stages using variable framework. For example: ${simpleAlias}.
- The existing custom and system context variables are now accessible using variable framework. As part of variable framework, the custom context variables that were defined using ctxVar IS service are merged into custom variables. The syntax for accessing the system context variables or custom context variables using variable framework is similar to the custom variables. Example : ${variableName}. For details on how the system and custom context variables were declared in API Gateway version 10.5, see "Conditional Routing" on page 273.

  Note:

As like the earlier versions of API Gateway, you can also define and access the custom-context variable or custom-variable using the ctxVar IS Service. For details, see *webMethods API Gateway User's Guide* .

- Both system context variables and custom variables (that includes custom context variables) are accessible across all policy parameters that support variables.

Custom Policy Extension

API Gateway provides a range of out-of-the-box policies to address common API management requirements like security, transformation, validation, error processing, and so on. In addition, API Gateway provides an option to add custom extensions or custom variables.

Custom Extensions

You can add these custom extensions into API Gateway policy stages to handle a requirement that might not be handled by any of the existing policies. You can use custom extensions in conjunction with the existing policies across stages. For example, if you want to invoke a third-party API or call an external endpoint during any stage of API processing, you can add custom logic in the corresponding policy stage and use it as required.

API Gateway supports the following custom extension types:

- **External endpoint**

  Use this custom extension when you have an external endpoint exposed, which can be configured and invoked during any stage in API processing.

  For example, if a native API expects the request in a certain format and the client application sends the request in a different format, you can add a custom extension to modify the incoming request to the required format before sending it to the native API.

- **webMethods IS service**

  Use this custom extension when you want to invoke the webMethods IS policy.

- **AWS Lambda**

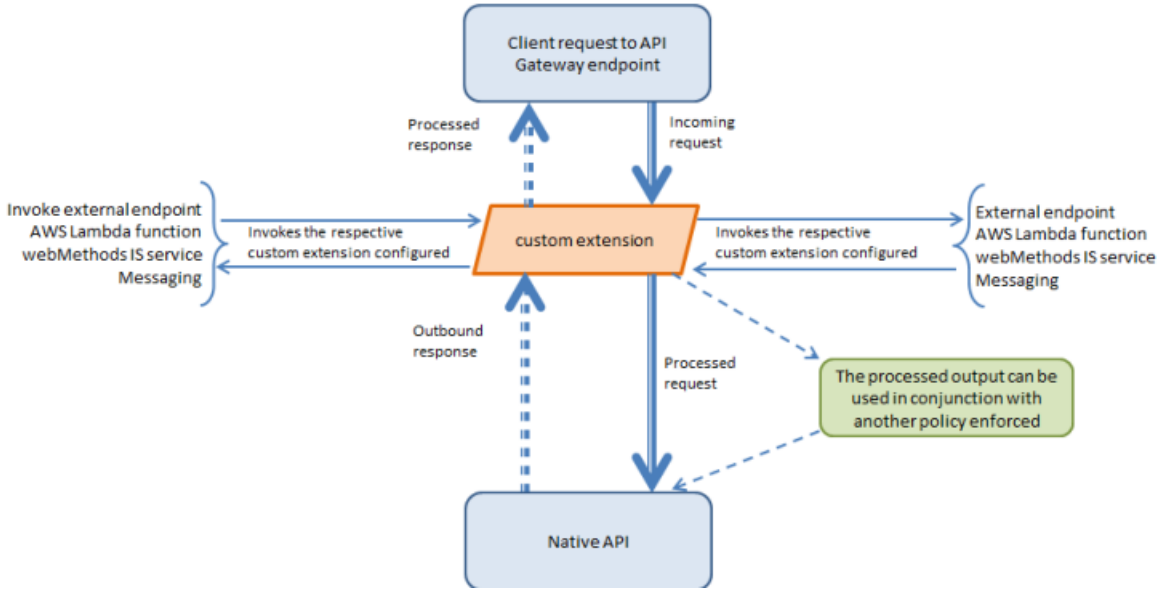
  Use this custom extension to invoke an Amazon Web Services (AWS) Lambda function and use the business logic built-in the Lambda function in any stage of API processing.

- **Messaging**

  Use this custom extension when you want to send some data to a queue or topic during any stage in API processing and a system can read the message from the queue or topic and process it asynchronously.

Custom extensions are applicable to the REST, SOAP, and OData API types. Custom extensions are supported at all levels such as, API, Scope, Global and can be added in any or all policy enforcement stages except the transport policy and the traffic monitoring policy stages.

The figure depicts a sample workflow for custom extension support in the request and response processing stages in API Gateway.



Custom Variables

You can configure custom variables under custom extension policy. You can assign a value or a variable expression to a custom variable which can be used in other policy parameters. Custom variable also provides option to set custom field to the transactional events. To set the custom fields, you have to define customTransactionFields.FIELD_NAME custom variable. It also provides an option to configure namespaces for XPath expressions. To configure the namespaces you have to define XpathNamespaces custom variable.

How Do I Invoke an API through HTTP or HTTPS using Custom Extension?

This use case explains how to invoke a service through HTTP or HTTPS using custom extension. The custom extension configured can be enforced in any of the policy stages and used during API processing.

The use case starts when you have an API that has to be enforced with a custom extension and ends when you successfully invoke the API with the custom extension enforced.

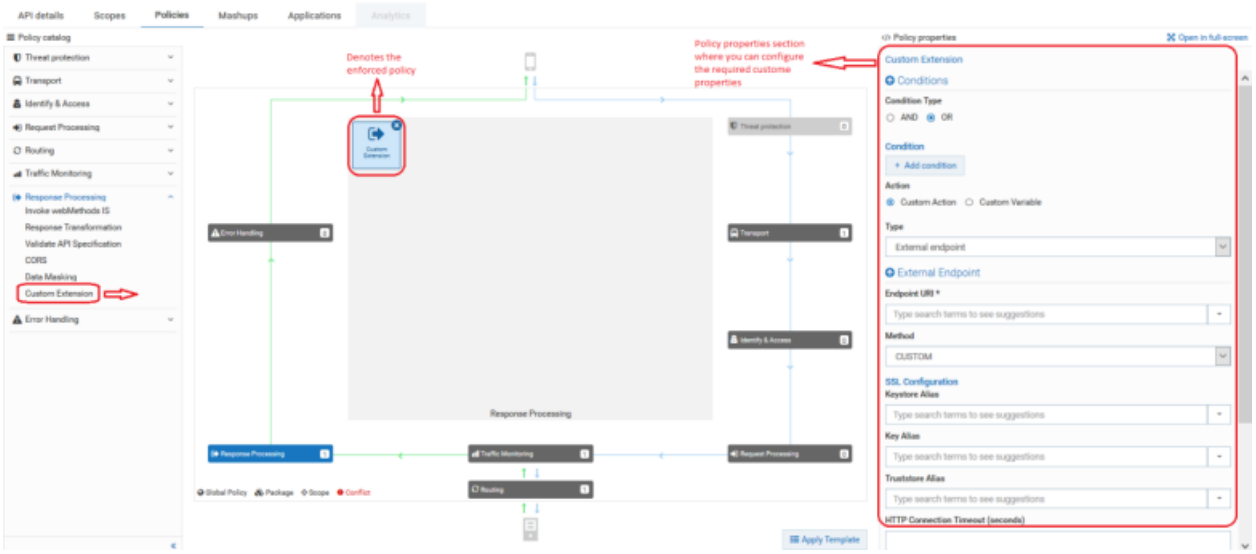
To invoke a service through HTTP or HTTPS using custom extension

1. Ensure you have the external endpoint URL to be invoked during API processing using a custom extension.
2. Click **APIs** on the title navigation bar.
3. Click the required API.

   The API details page appears.
4. Click **Edit**.
5. Select **Policies**.

6. Click ***Required Policy stage*** **> Custom Extension.**

   This adds the custom extension policy where you can configure the required properties.



   Click Open in full-screen to open the policy properties section in a full page.

7. Provide the following information in the **Conditions** section, as required:

Property	Description
Condition	Conditions are used to specify when the policy has to be executed. You can add multiple conditions with logical operators. Available values are: ■ **AND**. API Gateway executes this policy when all the configured conditions comply in the respective policy stage ■ **OR**. This is selected by default. API Gateway executes this policy when any one of the configured conditions complies. Click **Add Condition** and provide the following information and click **Add** . ■ **Variable**. Specifies the variable type with a syntax. ■ **Operator**. Specifies the operator to use to relate variable and the value. You can select one of the following: ■ **Equals** ■ **Equals ignore case** ■ **Not equals**

Property	Description
	■ **Not equals ignore case** ■ **Contains** ■ **Exists** ■ **Range** ■ **Greater Than** ■ **Less Than** ■ **Value**. Specifies a plain value or value with a syntax. For details about the variables available in API Gateway, see "Variable Framework" on page 373.

8. Click **Custom Action**.
9. Select **External endpoint** in the custom extension **Type** field.
10. Provide the following information in the External Endpoint section, as required:

Property	Description
Endpoint URI	Provide the external endpoint URI that you want to invoke.
Method	Specify the method exposed by the API. Available values are: **PUT**, **POST**, **GET**, **DELETE**, **HEAD**, **CUSTOM**. **Note:** If you select **CUSTOM**, the HTTP method in the incoming request is sent to the native API.
SSL Configuration	Specifies the required SSL configuration details of the external endpoint. Provide the following information: ■ **Keystore Alias**. Specifies the keystore alias. For details on Keystore configuration, see *webMethods API Gateway Administration*. ■ **Key Alias**. Specifies the alias for the private key, which must be stored in the keystore specified by the keystore alias. ■ **Truststore Alias**. Specifies the alias for the truststore. For details on Truststore configuration, see *webMethods API Gateway Administration*.

Property	Description
	■ **HTTP Connection Timeout (seconds)**. Specifies the time interval (in seconds) after which a connection attempt to the external endpoint URL times out. ■ **Read Timeout (seconds)**. Specifies the time interval (in seconds) after which a socket read attempt times out.
Path Parameters	Specifies the path parameter you want to configure to your custom extension. Provide the following information: ■ **Path Parameter Name**. Species the name of the path parameter you want to configure in your custom extension. This path parameter name should be present in the endpoint URL enclosed with `{}` to be replaced at runtime. For example, define external URL as `http://host/authors/{id}/books` and provide `id` as path parameter name with the value you need to populate at runtime. ■ **Path Parameter Value**. Specifies the value for the path parameter specified.

11. Configure the custom properties of the custom extension as required.

    For details about the custom extension properties and their descriptions, see "Custom Extension Properties" on page 400.

12. Click **Save**.

    The API is saved with the added custom extension.

13. Invoke the API.

    The applied custom extension invokes the mentioned HTTP or HTTPS endpoint and processes as configured.

How Do I Invoke an IS Service using a Custom Extension?

This use case explains how to invoke an IS service using custom extension in one of the policy stages and enforce during API processing.

For example you may want to process the request messages and transform them into a format required by the native API or perform some custom logic before API Gateway sends the requests to the native API.

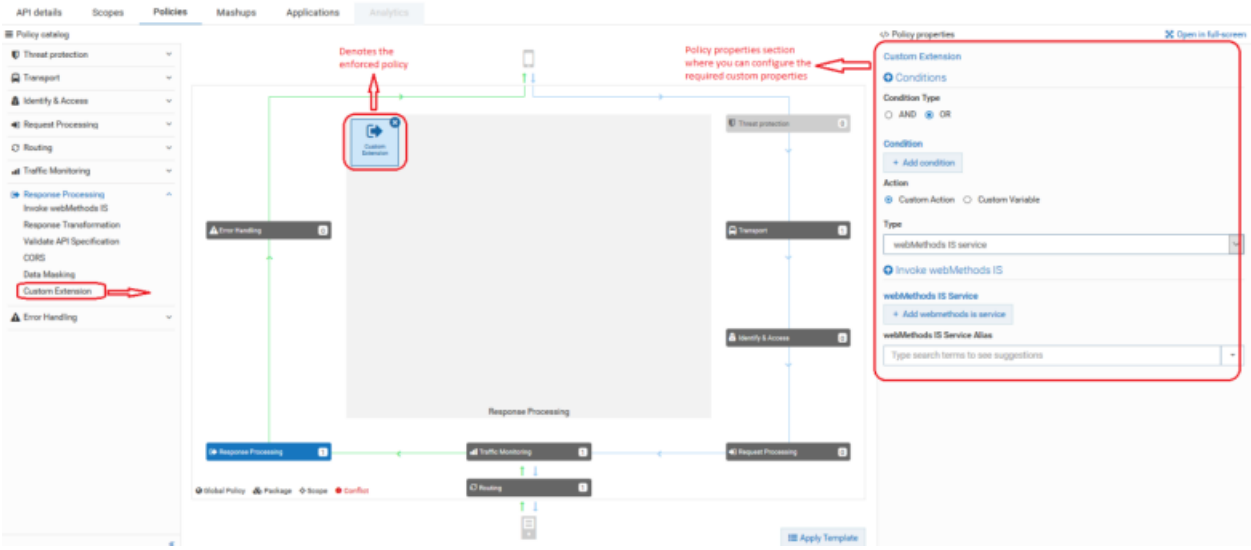
The use case starts when you have an API which has to be enforced with a messaging custom extension and ends when you successfully invoke the API with the custom extension enforced.

To invoke an IS service using custom extension

1. Click **APIs** on the title navigation bar.
2. Click the required API.

   The API details page appears.
3. Click **Edit**.
4. Select **Policies**.
5. Click ***Any policy stage*** **> Custom Extension.**

   This adds the custom extension policy where you can configure the required properties.



Click Open in full-screen to open the policy properties section in a full page.

6. Provide the following information in the **Conditions** section, as required:

Property	Description
Condition	Conditions are used to specify when the policy has to be executed. You can add multiple conditions with logical operators. Available values are: ■ **AND**. API Gateway executes this policy when all the configured conditions comply in the respective policy stage ■ **OR**. This is selected by default. API Gateway executes this policy when any one of the configured conditions complies.

Property	Description
	Click **Add Condition** and provide the following information and click **Add**. ■ **Variable**. Specifies the variable type with a syntax. ■ **Operator**. Specifies the operator to use to relate variable and the value. You can select one of the following: ■ **Equals** ■ **Equals ignore case** ■ **Not equals** ■ **Not equals ignore case** ■ **Contains** ■ **Exists** ■ **Range** ■ **Greater Than** ■ **Less Than** ■ **Value**. Specifies a plain value or value with a syntax. For details about the variables available in API Gateway, see "Variable Framework" on page 373.

7. Click **Custom Action**.
8. Select **webMethods IS service** in the custom extension **Type** field.
9. Provide the following information in the Invoke webMethods IS section, as required:

Property	Description
webMethods IS Service	Specify the webMethods IS service to be invoked to process the messages. The webMethods IS service must be running on the same Integration Server as API Gateway. **Note:** If an exception occurs when invoking the webMethods IS service, by default API Gateway displays the status code as *500* and error message as *Internal Server Error*.

Property	Description

	You can set custom status code and error message by setting the following properties in the message context of the webMethods IS service:
	■ service.exception.status.code
	■ service.exception.status.message
	The sample code is given below:
	`IDataCursor idc = pipeline.getCursor();`
	`MessageContext context =`
	`(MessageContext)IDataUtil.get(idc,"MessageContext");`
	`if(context != null)`
	`{`
	`context.setProperty("service.exception.status.code", 404);`
	`context.setProperty("service.exception.status.message",`
	`"Object Not Found");`
	`throw new ServiceException();`
	`}`
	Note:
	If ServiceException or FlowException occurs when invoking webMethods IS Service, the message given in the exception is displayed to the client. If any other exception occurs, a generic error message is displayed to the client.

Run As User	Specifies the authentication mode to invoke the IS service. If this field is left blank the incoming credentials of the user, identified by API Gateway, are used to authenticate and invoke the IS service. You can also specify a particular user, you want API Gateway to invoke the IS service.
Comply to IS Spec	Select this property to mark it true, if you want the input and the output parameters to comply to the IS Spec present in pub.apigateway.invokeISService.specifications folder in wmAPIGateway package.
webMethods IS Service Alias	Specifies the webMethods IS service alias to be invoked to process the messages. Start typing the webMethods alias name, select the alias from the type-ahead search results displayed, and click +Add to add one or more aliases.

10. Click **Save**.

    The API is saved with the added custom extension.

11. Invoke the API.

The applied custom extension invokes the IS service and processes as configured.

How Do I Invoke an AWS Lambda Function using Custom Extension?

This use case explains how to invoke an AWS Lambda function using custom extension. The custom extension configured can be enforced in any of the policy stages and used during API processing.

The use case starts when you have an API that has to be enforced with a custom extension and ends when you successfully invoke the API with the custom extension enforced.

To invoke an AWS Lambda function using custom extension

1. Create a Lambda function and ensure it is active.

   For details on how to create an AWS Lambda function, see https://docs.aws.amazon.com/lambda/latest/dg/getting-started.html.

2. Configure AWS alias.

   For details on how to configure an AWS alias, see *webMethods API Gateway Administration*.

3. Click **APIs** on the title navigation bar.

4. Click the required API.

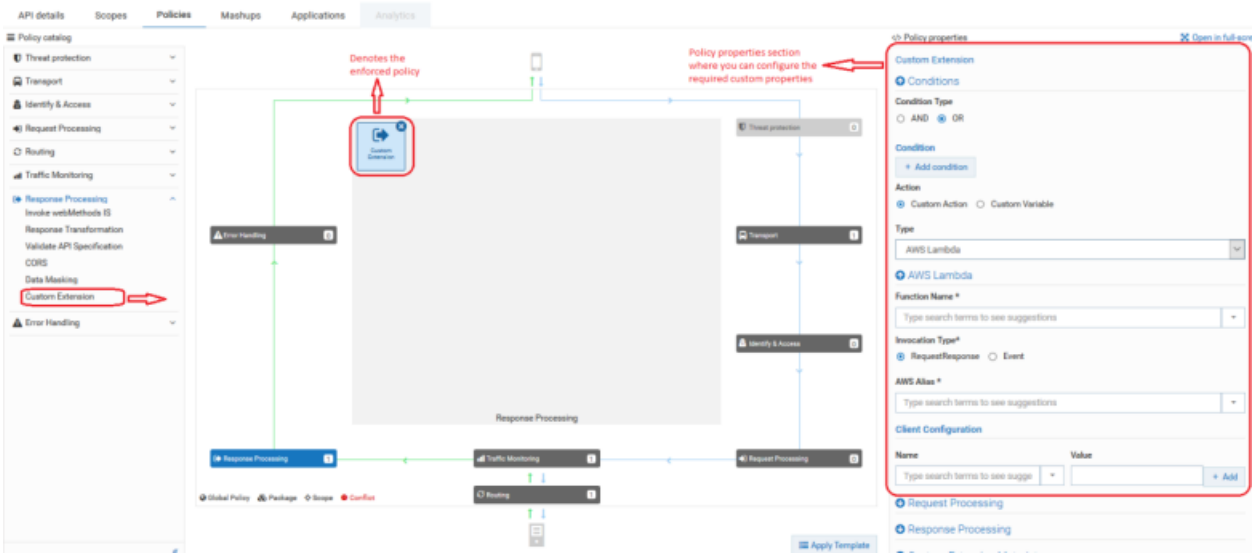
   The API details page appears.

5. Click **Edit**.

6. Select **Policies**.

7. Click ***Required Policy stage*** **> Custom Extension.**

   This adds the custom extension policy where you can configure the required properties.



Click Open in full-screen to open the policy properties section in a full page.

8. Provide the following information in the **Conditions** section, as required:

Property	Description
Condition	Conditions are used to specify when the policy has to be executed. You can add multiple conditions with logical operators. Available values are: ■ **AND**. API Gateway executes this policy when all the configured conditions comply in the respective policy stage ■ **OR**. This is selected by default. API Gateway executes this policy when any one of the configured conditions complies. Click **Add Condition** and provide the following information and click **Add**. ■ **Variable**. Specifies the variable type with a syntax. ■ **Operator**. Specifies the operator to use to relate variable and the value. You can select one of the following: ■ **Equals** ■ **Equals ignore case** ■ **Not equals** ■ **Not equals ignore case** ■ **Contains** ■ **Exists** ■ **Range** ■ **Greater Than** ■ **Less Than** ■ **Value**. Specifies a plain value or value with a syntax. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

9. Click **Custom Action**.
10. Select **AWS Lambda** in the custom extension **Type** field.
11. Provide the following information in the AWS Lambda section, as required:

Property	Description
Function Name	Provide the AWS Lambda function name you want to invoke. As this property supports variable framework, you can use the available variables. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
Invocation Type	Specify the AWS invocation type, asynchronous or synchronous. Available options are: ■ **RequestResponse** (synchronous type) ■ **Event** (asynchronous type)
AWS Alias	Provide the AWS alias configured for the AWS account.
Client Configuration	Provide the following client configuration details and click +Add. ■ **Name**. Start typing the client property name and select the required property from the type-ahead search results displayed. API Gateway supports the following properties that you can configure: Socket timeout(ms), Connection timeout(ms), Request timeout(ms), Connection expiration timeout(ms), Maximum Connection idle time(ms), Client execution timeout(ms), Server error retry count, Enable throttle retries, Maximum client retry count, TCP send buffer size hints, TCP receive buffer size hints, Enable gzip requests, Enable Expect-Continue, Enable host prefix injection, Enable Keep-alive, Enable, Response metadata caching, Response metadata cache size, and Signature Algorithm. ■ **Value**. Provide a value for the client property specified. You can configure multiple properties. For details about the supported client properties, see the following AWS documents: ■ https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/section-client-configuration.html ■ https://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/ClientConfiguration.html

12. Configure the custom properties of the custom extension as required.

    For details about the custom extension properties and their descriptions, see "Custom Extension Properties" on page 400.

13. Click **Save**.

The API is saved with the added custom extension.

14. Invoke the API.

    The applied custom extension invokes the AWS lambda function and processes as configured.

How Do I Invoke an API Asynchronously through JMS/AMQP using a Custom Extension?

This use case explains how to add messaging as a custom extension in one of the policy stages and invoke a service asynchronously during API processing.

You want to use the AMQP messaging setup to send some data to a queue during request processing using the configured custom extension. This data that is sent can then be read from a queue, processed, and sent in an asynchronous way.

The use case starts when you have an API which has to be enforced with a messaging custom extension and ends when you successfully invoke the API with the custom extension enforced.

To invoke an API asynchronously through JMS/AMQP using custom extension

1. Ensure you have a JMS/AMQP environment set up with the required connection alias configured.

   For details on setting up the JMS/AMQP setup, see *webMethods API Gateway Administration*.

2. Click **APIs** on the title navigation bar.
3. Click the required API.

   The API details page appears.

4. Click **Edit**.
5. Select **Policies**.
6. Click ***Any policy stage*** **> Custom Extension.**

   This adds the custom extension policy where you can configure the required properties.

7. Provide the following information in the **Conditions** section, as required:

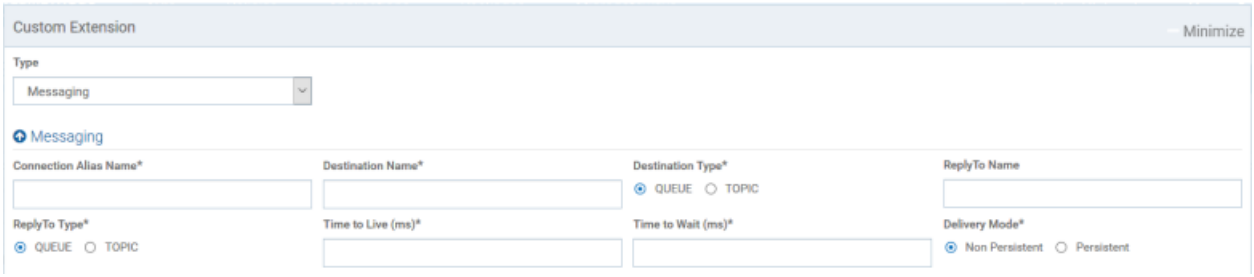
Property	Description
Condition	Conditions are used to specify when the policy has to be executed. You can add multiple conditions with logical operators. Available values are: ■ **AND**. API Gateway executes this policy when all the configured conditions comply in the respective policy stage ■ **OR**. This is selected by default. API Gateway executes this policy when any one of the configured conditions complies. Click **Add Condition** and provide the following information and click **Add**. ■ **Variable**. Specifies the variable type with a syntax. ■ **Operator**. Specifies the operator to use to relate variable and the value. You can select one of the following: ■ **Equals** ■ **Equals ignore case** ■ **Not equals** ■ **Not equals ignore case** ■ **Contains** ■ **Exists**

Property	Description
	■ **Range** ■ **Greater Than** ■ **Less Than** ■ **Value**. Specifies a plain value or value with a syntax. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

8. Click **Custom Action**.
9. Select **Messaging** in the custom extension **Type** field.
10. Provide the following information in the Messaging section, as required:

Property	Description
Connection Alias Name	Name of the connection alias you have configured. You can configure the connection alias under Administration > Messaging section. For details on how to configure the connection alias, see *webMethods API Gateway Administration*.
Destination Name	Specify the destination to which the request message is sent.
Destination Type	Specify the destination type to which the request message is sent.
Reply To Name	Specify the destination to which the response message is sent.
Reply To Type	Specifies the destination type to which the response message is sent. Select one of the following types: ■ **QUEUE**. Indicates that the response message is sent to a particular queue. ■ **TOPIC**. Indicates that the response message is sent to a particular topic.
Time to Live (ms)	Provide a numeric value that specifies the expiration time (in milliseconds) of the JMS or AMQP message. If the time-to-live is specified as zero, expiration is set to zero, which indicates that the message does not expire.
Time to Wait (ms)	Defines the time in milliseconds for which API Gateway listens to the Reply To Queue or Topic for the response message.

Property	Description
Delivery Mode	The message delivery mode for the request message. This is the delivery mode that web service clients must specify in the JMS or AMQP message that serves as the request message for the web service. Select one of the following modes: ■ **Non-Persistent**. Indicates that the request message is not persistent. The message might be lost if the JMS provider fails. ■ **Persistent**. Indicates that the request message should be persistent. The message is not lost if the JMS provider fails.



11. Configure the custom properties of the custom extension as required.

    For details on the custom extension properties and their description, see "Custom Extension Properties" on page 400.

12. Click **Save**.

    The API is saved with the added custom extension..

13. Invoke the API.

    The applied custom extension calls the queue or topic that is configured.

How Do I Define a Custom Variable?

This use case explains how to define custom variable using custom extension. The defined custom variable can be used in any of the subsequent policy stages during API processing.

The use case starts when you have to define a custom variable, which is not available in API Gateway and ends when you successfully defined and accessed the variable in the subsequent policy stages.

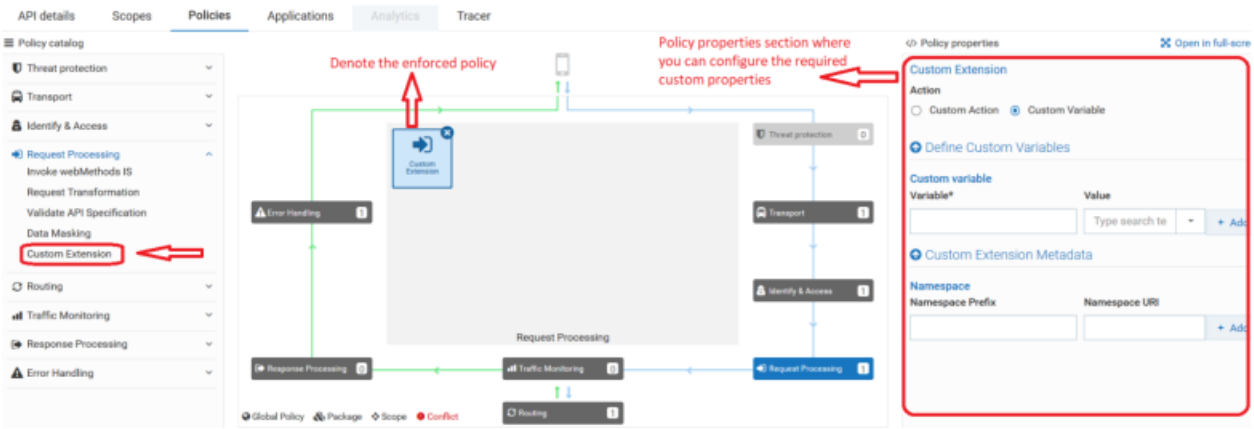
To define a custom variable using custom extension

1. Click **APIs** on the title navigation bar.
2. Click the required API.

The API details page appears.

3. Click **Edit**.
4. Select **Policies**.
5. Click ***Required Policy stage*** **> Custom Extension.**

   This adds the custom extension policy where you can configure the required properties.



Click Open in full-screen to open the policy properties section in a full page.

6. Provide the following information in the **Conditions** section, as required:

Property	Description
Condition	Conditions are used to specify when the policy has to be executed. You can add multiple conditions with logical operators. Available values are: ■ **AND**. API Gateway executes this policy when all the configured conditions comply in the respective policy stage. ■ **OR**. This is selected by default. API Gateway executes this policy when any one of the configured conditions complies. Click **Add Condition** and provide the following information and click **Add**. ■ **Variable**. Specifies the variable type with a syntax. ■ **Operator**. Specifies the operator to use to relate variable and the value. You can select one of the following: ■ **Equals** ■ **Equals ignore case**

Property	Description
	■ **Not equals** ■ **Not equals ignore case** ■ **Contains** ■ **Exists** ■ **Range** ■ **Greater Than** ■ **Less Than** ■ **Value**. Specifies a plain value or value with a syntax. For details about the variables available in API Gateway, see "Variable Framework" on page 373 .

7. Click **Custom Variable**.
8. Provide the following information in the **Define Custom Variables** section, as required:

Property	Description
Custom Variable	Specify the custom variable with a syntax to be accessed across subsequent stages and click **Add**. ■ **Variable**. Specifies the custom variable with a syntax. ■ **Value**. Specifies a plain value or value with a syntax. For example, if you want to use the client's request related information like content-type header at response stage, you can define the ${clientContentType} custom variable to store the ${request.headers.Content-Type} variable. The ${clientContetType} custom variable can be accessed in any other policy across subsequent stages such as response or error processing stage. For details about the variables available in API Gateway, see "Variable Framework" on page 373.

9. Provide the following information in the **Custom Extension Metadata** section, as required. This is applicable only for XML transformation:

Property	Description
Namespace Prefix	Provide the namespace prefix of the payload expression to be validated.

Property	Description
	For example, specify the namespace prefix as `SOAP_ENV`.
Namespace URI	Provide the namespace URI of the payload expression to be validated. For example, specify the namespace URI as `http://schemas.xmlsoap.org/soap/envelope/`. This declaration defines SOAP_ENV as an alias for the namespace: http://schemas.xmlsoap.org/soap/envelope/. **Note:** You can add multiple namespace prefixes and URIs by clicking **Add**.

10. Click **Save**.

    The API is saved with the added custom variables.

11. Invoke the API.

    The custom variables are defined and can be accessed in the subsequent policy stages.

Custom Extension Properties

The table lists the properties that you can specify for a custom extension.

Request Processing Section

The table lists the custom extension properties you can configure in the Request processing section:

Property	Description
Payload	Provide the request payload to be sent to the custom extension in one of the following ways: ■ Type the request payload in the text box. For details on the data objects and variables available in the Request Processing section that you can use to configure, see "Data Objects and Variables Available in API Gateway" on page 402. ■ Click ⁘ and select one of the following and provide the required information: ■ **Inline Request**. Type the required payload. ■ **Load from Schema**. Click **Browse** to upload a JSON or XML schema file and click **Save**.

Property	Description
Headers	Provide the following information, if you want to configure the headers you need to send to the custom extension. By default, no headers are sent to the custom extension. ■ Select **Use incoming headers** to use the header content in the incoming requests from the client. ■ Provide the **Header Name** and the **Header Value** in the incoming client request that has to be processed.
Query Parameters	Provide the following information, if you want to configure query parameters you need to send to the custom extension. ■ Provide the **Query Parameter Name** and the **Query Parameter Value** in the incoming client request that has to be processed. For details on the data objects and variables available in the Request Processing section that you can use to configure, see "Data Objects and Variables Available in API Gateway" on page 402.

Response Processing section

The table lists the custom extension properties you can configure in the Response processing section:

Property	Description
Copy the entire response	Select to copy the entire response received from the external call out. This response is used in the subsequent step by using `${request.payload}` or `${response.payload}`. **Note:** Do not select this if you are using AWS Lambda custom extension with invocation type as **Event** as there is no response returned.
Abort API execution in case of failure	Select to abort the API execution when the external callout encounters any failures. If you do not select this option, API Gateway logs the failure and continues with the processing.
Transformation	Specify the following custom variables with a syntax to be accessed from the response of the custom extension and click **Add**. ■ **Variable**. Specifies the variable type with a syntax. ■ **Value**. Specifies a value with a syntax.

Property	Description
	For example if you provide a variable as `${var}` and the corresponding value as `${response[customExtension].payload.jsonPath[$.id]}`, this transformation evaluates the JSON path from the custom policy response payload to get the value of the attribute `id`. The evaluated value is assigned to the variable `var` given in the **Variable** field. You can use the ${var} syntax in the subsequent policies that support variable framework. For details about the data objects and variables available in the Response Processing section that you can use to configure, see "Data Objects and Variables Available in API Gateway" on page 402.
Custom extension metadata	This is used for XML transformation. ■ **Namespace Prefix**. Provide the namespace prefix of the payload expression to be validated. ■ **Namespace URI**. Provide the namespace URI of the payload expression to be validated.

Custom Extension Metadata section

The table lists the custom extension properties you can configure in the Custom Extension Metadata section. This is applicable only for XML transformation.

Property	Description
Namespace Prefix	Provide the namespace prefix of the payload expression to be validated.
Namespace URI	Provide the namespace URI of the payload expression to be validated.

For details about the data objects and variables that you can use to configure, see "Data Objects and Variables Available in API Gateway" on page 402.

Data Objects and Variables Available in API Gateway

The following table summarizes the data objects and variables that are available in API Gateway:

Object or Variable type	Possible values
`paramStage`	■ `request` ■ `response`
`paramType`	■ `payload` or `body` ■ `headers` ■ `query`

Object or Variable type	Possible values
	■ path ■ httpMethod ■ statusCode ■ statusMessage
queryType	■ xpath ■ jsonPath ■ regex

The following data objects are available in the request processing or response processing steps:

- `${paramStage.paramType}`

  You can use this syntax to access the following string variables: `path`, `statusCode`, `statusMessage`, `httpMethod`. Examples: `${request.path}`, `${response.statusCode}`

- `${paramStage.paramType.paramName}`

  You can use this syntax to access map types, such as query, headers, and path. Example: `${request.query.var1}`, `${response.header.Content-Type}`, `${request.path.name}`.

- `${paramStage.paramType.queryType[queryValue]}`

  You can use this syntax to query a paramType. Examples:

  - `${request.payload.xpath[//ns:emp/ns:empName]}`

    Where "`//ns:emp/ns:empName`" is the XPath to be applied on the payload if contentType is application/xml, text/xml, or text/html.

  - `${response.payload.jsonPath[$.cardDetails.number]}`

    Where `$.cardDetails.number` is the jsonPath to be applied on payload if contentType is application/json or application/json/badgerfish.

  - `${request.payload.regex[[0-9]+]}`

    Where `[0-9]+` is the regular expression to be applied on the payload if contentType is text/plain.

  Note:
  While `xpath` and `jsonPath` are applicable only to payload, `regEx` can be used with both `payload` and `path`.

- `${paramStage[stepName].paramType.paramName]}`

  You can use this syntax to access header or payload from the response of the custom extension in the response processing step.

Example:

Variable: `${response.headers.id}`

Value: `${response[customExtension].payload.jsonPath[$.id]}`

This transformation adds a header to the response with name id and its value is derived from the json payload that is sent from the external callout as per the json path.

- You can define your own variables in the Transformation variables field in the response processing step.

  Examples: `${key}`, `${value}`. The custom transformation variables that you define are available in subsequent steps.

Request and Response Transformation Policies

Transformation policy enables you to configure several transformations on the requests from the clients into a format required by the native API, or to transform the response by the native API into a format required by the client.

The transformations include Header, Query Parameter, Path Parameter transformation, HTTP Method transformation, Payload transformation, and Advanced transformation. The transformations are applied based on the configurations provided in the transformation policies.

When can you use transformation policies?

You can use transformation policies:

- When the API Provider wants to read the contents of the request and response to do audit logging, or trigger a notification based on the contents of the request.
- When the API Provider wants to modify the request before forwarding the request to native API as the native API wants to identify all incoming requests from API Gateway. In such case the provider can configure the Request transformation policy to add a header to all requests before they get routed to the native API.

Pre-Requisites

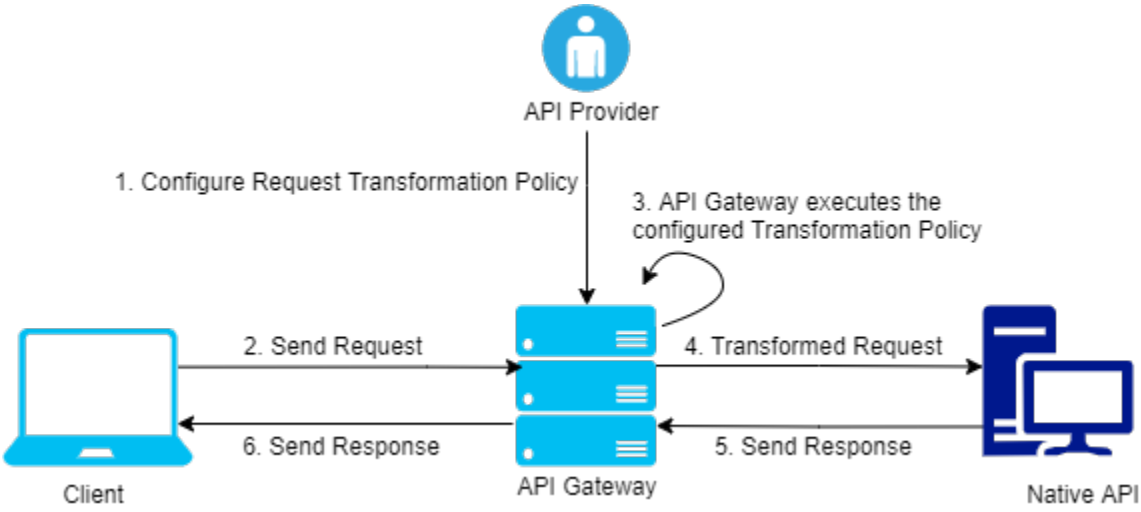
- Install API Gateway advanced edition 10.2 or higher.
- Basic understanding of API Gateway and policy enforcement.
- Ensure that you have the Manage API privilege.

How do I transform a request using Request Transformation Policy?

Use the Request Transformation policy to modify the contents of an incoming request such as headers, payload, query parameters, path parameters, HTTP method using the configurations given by the API Provider.

The request transformation workflow is as follows:

1. The API Provider configures the Request Transformation policy in the Request Processing stage of API Gateway. The API provider configures the details about when and how to transform the contents of an incoming request.
2. The client sends the request to API Gateway.
3. API Gateway applies the transformations configured by the API Provider and transforms the incoming request.
4. API Gateway sends the transformed request to the native API.
5. Native API processes the transformed request and sends the response to API Gateway.
6. API Gateway forwards the response to the client.



Consider a scenario where you have a legacy REST API (employeeApi) that does not adhere to the REST API standards. For example, it accepts functional information such as employee name through a header `employeeName` instead of accepting them through query or path parameters and you want to modify the API to REST standards.

To configure request transformation policy:

1. Click **APIs** in the title navigation bar.

   A list of available APIs appears.

2. Select a Rest API from the list of APIs and click **Edit**.

3. Select **Policies > Request Processing > Request Transformation**.

   The Request Transformation details page appears.

4. In the **Condition** section, select **OR**.

The configured transformation is applied when at least one of the conditions is satisfied.

Note:
The condition can also be set to AND operator. The configured transformation is applied only when all the set conditions are satisfied.

5. Click **Add Condition** to configure the conditions to evaluate the contents on the request.

   a. Specify the **Variable**. Example, Content-Type.

   b. Specify the **Operator** to use to relate variable and the value provided. Example, **Equals**.

   c. Specify the **Value**. Example, application/json.

      When you select the operator - **Equals**, the Condition checks if the **Variable**: Content-Type is equal to the **Value**: application/json.

   d. Click **Add**.

6. Select **Transformation Configuration > Header/Query/Path transformation**.

   The Header/Query/Path transformation details page appears.

7. In **Add/Modify** section, add the variable and set its value.

   Here, native API accepts employee name through header `${request.headers.employeeName}` and you want the native API to accept these values through the query parameter `${request.query.name}` and expose this change to the client without exposing the query parameter.

   To achieve this, set the variable and the value parameters as follows:

   a. **Variable**: `${request.headers.employeeName}`

   b. **Value**: `${request.query.name}`

   c. Click **Add**.

   Note:
   For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

8. In the **Remove** section, add `${request.query.name}` to remove the query parameter from the request so that it does not reach the native API.

9. Click **Save**.

This request transformation policy configuration allows the native API to accept the header values through query parameters. The native API accepts the header values through the query parameters

by transforming the query parameters to header parameters and then removing the query parameter from the incoming request.

Request Transformation Policy Properties

The table lists the properties that you can specify for the Request Transformation policy:

Property	Description
Condition	Conditions are used to specify when the policy has to be executed. You can add multiple conditions with logical operators. Available values are: ■ **AND**. API Gateway transforms the requests that comply with all the configured conditions. ■ **OR**. This is selected by default. API Gateway transforms the requests that comply with any one configured condition. Click **Add Condition** and provide the following information and click  . ■ **Variable**: Specifies the variable type with a syntax. ■ **Operator**: Specifies the operator to use to relate variable and the value. You can select one of the following: ■ **Equals** ■ **Equals ignore case** ■ **Not equals** ■ **Not equals ignore case** ■ **Contains** ■ **Exists** ■ **Range** ■ **Greater Than** ■ **Less Than** ■ **Value**: Specifies a plain value or value with a syntax. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
Transformation Configuration: Specifies various transformations to be configured.	

Property	Description
Header/Query/Path Transformation for REST API and **Header Transformation** for SOAP API	Specifies the Header, Query or path transformation to be configured for incoming requests. You can add or modify header, query or path transformation parameters by providing the following information: ■ **Variable**. Specifies the variable type with a syntax. ■ **Value**. Specifies a plain value or value with a syntax. You can add multiple variables and corresponding values by clicking  . You can remove any header, query, or path transformation parameters by typing the plain value or value with a syntax. **Note:** Software AG recommends you not to modify the headers `${request.headers.Content-Length}` and `${request.headers.Content-Encoding}` as API Gateway adds the right values for these headers before sending the response back to client. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373. **Note:** Payload transformation does not happen automatically for content-type transformation. When you change the content type, ensure that you do payload transformation. For example, if you change the content-type header from application/xml to application/json, you must also change the respective payload from application/xml to application/json.
Method transformation for REST API	Specifies the method transformation to be configured for incoming requests. Select any of the HTTP Method listed: ■ **GET** ■ **POST** ■ **PUT** ■ **DELETE** ■ **HEAD** ■ **CUSTOM**

Property	Description
	Note: When **CUSTOM** is selected, the HTTP method in incoming request is sent to the native service. When other methods are selected, the selected method is used in the request sent to the native service.
	Note: Only Method Transformation happens when configured, but you have to take care of adding payload during transformations involving method change like GET to POST, and so on.

Payload Transformation	Specifies the payload transformation to be configured for incoming requests.
	
	Provide the following information:
	
	■ **Payload Type**. Specifies the content-type of payload, to which you want to transform. The **Payload** field renders the respective payload editor based on the selected content-type.
	
	■ **Payload**. Specifies the payload transformation that needs to be applied for the incoming requests.
	
	As this property supports variable framework, you can make use of the available variables to transform the request messages.
	
	For example, consider the native API accepting two integer values `value1` and `value2`, and you want to pass these two values from API Gateway to the native API, you can configure the payload field as follows:
	
	`{`
	`"value1" : 12,`
	`"value2" : 34`
	`}`
	
	You can also configure the payload field using one or more variables by using variable framework. Let us see another syntax. For example, for the same native API seen in the previous example, if your client sends both the values through headers `val1` and `val2`, and you want to add it to payload for the native API to recognize the input, you can do so by configuring the payload field as follows:
	
	`{`
	`"value1" :${request.headers.val1},`
	`"value2" :${request.headers.val2}`
	`}`
	
	For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

Property	Description

Note:
If your payload content-type is different from the incoming payload's content-type, you need to transform the content-type of the header using **Header Transformation**.

- Click **+ Add xslt document** to add an xslt document and provide the following information:
  - **XSLT file**. Specifies the XSLT file used to transform the request messages as required.

    Click **Browse** to browse and select a file.
  - **Feature Name**. Specifies the name of the XSLT feature.
  - **Feature value**. Specifies the value of the XSLT feature.

    You can add more XSLT features and xslt documents by clicking 

.

Note:
API Gateway supports XSLT 1.0 and XSLT 2.0.

- Click **+ Add xslt transformation alias** and provide the following information:
  - **XSLT Transformation alias**. Specifies the XSLT transformation alias

When the incoming request is in JSON, you can use a XSLT file similar to the below sample:

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.1"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="xml"/>
   <xsl:template match="/" >
   <xsl:element name="fakeroot">
   <xsl:element name="fakenode">
        <!-- Apply your transformation rules based on the
request from the Client-->
 </xsl:element>
 </xsl:element>
   </xsl:template>
</xsl:stylesheet>
```

When the incoming request is in XML, you can use a XSLT file similar to the below sample:

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.1"
```

Property	Description
	<pre> xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"> <xsl:output method="xml"/> <xsl:template match="/" > <xsl:element name="soapenv:Envelope"> <xsl:element name="soapenv:Body"> <!-- Apply your transformation rules based on the request from the Client--> </xsl:element> </xsl:element> </xsl:template> </xsl:stylesheet></pre>
Advanced Transformation	Specifies the advanced transformation to be configured for incoming requests. Provide the following information: ■ **webMethods IS Service**. Specify the webMethods IS service to be invoked to process the request messages. You can add multiple services by clicking  . For details about usage of Invoke webMethods IS policy in versions 10.2 and higher, see "Invoke webMethods IS Policy" on page 421. **Note:** The webMethods IS service must be running on the same Integration Server as API Gateway. ■ **Run as User**. Specifies the authentication mode to invoke the IS service. If this field is left blank the incoming credentials of the user, identified by API Gateway, are used to authenticate and invoke the IS service. You can also specify a particular user, you want API Gateway to use to invoke the IS service. ■ **Comply to IS Spec**. Mark this as `true` if you want the input and the output parameters to comply to the IS Spec present in pub.apigateway.invokeISService.specifications folder in wmAPIGateway package. ■ **webMethods IS Service alias**. Specifies the webMethods IS service alias to be invoked to pre-process the request messages.
Transformation Metadata: Specifies the metadata for transformation of the incoming requests. For example, the namespaces configured in this section can be used when you provide the syntax for XPath ${request.payload.xpath} For example: ${request.payload.xpath[//ns:emp/ns:empName]}	
Namespace	Specifies the namespace information to be configured for transformation.

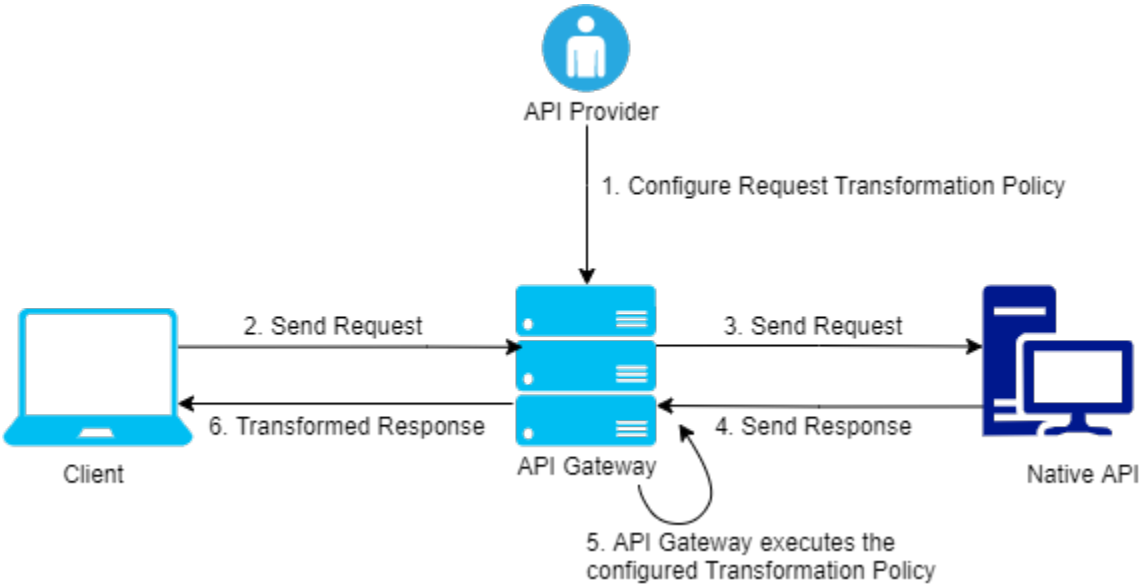
Property	Description
	Provide the following information: ■ **Namespace Prefix**. The namespace prefix of the payload expression to be validated. For example, specify the namespace prefix as `SOAP_ENV`. ■ **Namespace URI**. The namespace URI of the payload expression to be validated. For example, specify the namespace URI as `http://schemas.xmlsoap.org/soap/envelope/`. This declaration defines SOAP_ENV as an alias for the namespace: http://schemas.xmlsoap.org/soap/envelope/. **Note:** You can add multiple namespace prefixes and URIs by clicking  .

How do I transform a request and its response using Transformation Policy?

Use the Response Transformation policy to modify the contents of an outgoing response such as headers, payload, query parameters, path parameters, HTTP method using the configurations given by the API Provider.

The response transformation workflow is as follows:

1. The API Provider configures the Response Transformation policy in the Response Processing stage of API Gateway. The API provider configures the details about when and how to transform contents of an outgoing response.
2. The client sends the request to API Gateway.
3. API Gateway forwards the request to native API.
4. Native API processes the request and sends response to API Gateway.
5. API Gateway applies the transformations configured by the API Provider and transforms the outgoing response.
6. API Gateway forwards the transformed response to the client.



Consider a scenario, where a native API URL is moved permanently or temporarily, the native API sends a 301 or 302 status code, and also sends the new address in the location header. However, when API Gateway comes across the 301 or 302 status code, API Gateway reads the status code and the location header, and redirects the request to new address mentioned in the location header. API Gateway, then sends the response from the new address to the client. This is how 3xx status code is handled in API Gateway.

In this scenario, if you do not want API Gateway to do the redirection, instead you want the clients to receive the 3xx status code, and then do the redirection. This can be achieved by using the Status Transformation policy in the Response Processing stage.

To achieve this transformation:

1. Change the native API to send an intermediate 2xx status code instead of 3xx status code, for request from API Gateway.

   For example, a demo service package contains a couple of REST services - source and destination.

   The REST service source is moved to a new address and it sends a 301 status along with location header. However, it sends 297 status code with the location header for requests from API Gateway. The location header contains the address for destination, which is the new address of the moved resource.

2. Configure the API in API Gateway with a Request Transformation policy to send a request header **requestOrigin** with the value **APIGateway**. To configure the request transformation policy, perform the following steps:

   a. Click **APIs** in the title navigation bar.

      A list of available APIs appears.

b. Select a Rest API from the list of APIs and click **Edit**.

c. Select **Policies > Request Processing > Request Transformation**.

The Request Transformation details page appears.

d. In the **Condition** section, select **OR**.

The configured transformation is applied when at least one of the conditions is satisfied.

> **Note:**
> The condition can also be set to AND operator. The configured transformation is applied only when all the set conditions are satisfied.

e. Click **Add Condition** to configure the conditions to evaluate the contents on the request.

f. Specify the **Variable**. Example, Content-Type.

g. Specify the **Operator** to use to relate variable and the value provided. Example, **Equals**.

h. Specify the **Value**. Example, application/json.

When you select the operator - **Equals**, the Condition checks if the **Variable**: Content-Type is equal to the **Value**: application/json.

i. Click **Add**.

j. Select **Transformation Configuration > Header/Query/Path transformation**.

The Header/Query/Path transformation details page appears.

k. In **Add/Modify** section, add the variable and set its value.

Set the **Variable** and **Value** parameters as follows:

- **Variable**: `${request.headers.requestOrigin}`
- **Value**: APIGateway

> **Note:**
> For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

l. Click **Save**.

This Request Transformation policy allows the API in API Gateway to send a request header **requestOrigin** with the value **APIGateway**. This will help the native API identify the request from API Gateway and send the response code 297.

3. Configure the API in API Gateway with the Status Transformation policy to transform the 297 status code to 301 status code. To configure the status transformation policy, perform the following steps:

   a. Click **APIs** in the title navigation bar.

      A list of available APIs appears.

   b. Select a Rest API from the list of APIs and click **Edit**.

   c. Select **Policies > Response Processing > Response Transformation** .

      The Response Transformation details page appears.

   d. In the **Condition** section, select **OR**.

      The configured transformation is applied when at least one of the conditions is satisfied.

      Note:
      The condition can also be set to AND operator. The configured transformation is applied only when all the set conditions are satisfied.

   e. Click **Add Condition** to configure the conditions to evaluate the contents on the request.

   f. Specify the **Variable**. Example, `${response.statusCode}`.

      Note:
      For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

   g. Specify the **Operator** to use to relate variable and the value provided. Example, **Equals**.

   h. Specify the **Value**. Example, 297.

      When you select the operator - **Equals**, the Condition checks if the **Variable**: `${response.statusCode}` is equal to the **Value**: 297.

   i. Click **Add**.

   j. Select **Transformation Configuration > Status transformation**.

      The Status transformation details page appears.

   k. Specify the Code and Message values that you would like in the response.

      Set the **Code** and **Message** parameters as follows:

      - **Code**: 301

- **Message**: Moved Permanently

l. Click **Save**.

This transformation policy allows the clients to receive the 301 status code, and then redirect to the new address mentioned in location header.

Response Transformation Policy Properties

The table lists the properties that you can specify for the Response Transformation policy:

Property	Description

Condition	Conditions are used to specify when the policy has to be executed. You can add multiple conditions with logical operators.
	
	Available values are:
	
	- **AND**. API Gateway transforms the responses that comply with all the configured conditions
	- **OR**. This is selected by default. API Gateway transforms the responses that comply with any one configured condition.
	
	Click **Add Condition** and provide the following information and click  .
	
	- **Variable**. Specifies the variable type with a syntax.
	- **Operator**. Specifies the operator to use to relate variable and the value. You can select one of the following:
	  - **Equals**
	  - **Equals ignore case**
	  - **Not equals**
	  - **Not equals ignore case**
	  - **Contains**
	  - **Exists**
	  - **Range**
	  - **Greater Than**
	  - **Less Than**
	- **Value**. Specifies a plain value or value with a syntax.

Property	Description
	For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
Transformation Configuration. Specifies various transformations to be configured.	
HeaderTransformation	Specifies the header, query or path transformation to be configured for the responses received from the native API. You can add or modify header, query or path transformation parameters by providing the following information: ■ **Variable**. Specifies the variable type with a syntax. ■ **Value**. Specifies a plain value or value with a syntax. You can add multiple variables and corresponding values by clicking  . You can remove any header, query, or path transformation parameters by typing the plain value or value with a syntax. **Note:** Software AG recommends you not to modify the headers `${response.headers.Content-Length}` and `${response.headers.Content-Encoding}` as API Gateway adds the right values for these headers before sending the response back to client. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373. **Note:** Payload transformation does not happen automatically for content-type transformation. When you change the content type, ensure to do payload transformation. For example, if you change the content-type header from application/xml to application/json, you must also change the respective payload from application/xml to application/json.
Status transformation	Specifies the status transformation to be configured for the responses received from the native API. Provide the following information: ■ **Code**. Specifies the status code that is sent in the response to the client. For example if you want to transform status code as 201, provide 201 in the **Code** field.

Property	Description
	■ **Message**. Specifies the Status message that is sent in the response to the client. As both these properties support variable framework, you can make use of the available variables to transform the response code and message. For example *You have submitted successfully* can be used to transform the original *OK* status message. For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.

Payload Transformation	Specifies the payload transformation to be configured for the responses received from the native API.
	
	Provide the following information:
	
	■ **Payload Type**. Specifies the content-type of payload, to which you want to transform. The **Payload** field renders the respective payload editor based on the selected content-type.
	
	■ **Payload**. Specifies the transformation that needs to be applied for the response.
	
	As this property supports variable framework, you can make use of the available variables to transform the response messages.
	
	For example, consider the client accepting two integer values `value1` and `value2`, and you want to pass these two values from API Gateway to the client, you can configure the payload field as follows:
	
	`{`
	`"value1" : 12,`
	`"value2" : 34`
	`}`
	
	You can also configure the payload field using one or more variables by using variable framework. Let us see another syntax. For example, for the same API seen in the previous example, if your native sends both the values through headers `val1` and `val2`, and you want to add it to payload for the client to recognize the input, you can do so by configuring the payload field as follows:
	
	`{`
	`"value1" :${response.headers.val1},`
	`"value2" :${response.headers.val2}`
	`}`

Property	Description

	For details about the variables available in API Gateway, see "Variables Available in API Gateway" on page 373.
	
	Note:
	If your payload content-type is different from the incoming payload's content-type, you need to transform the content-type of the header using **Header Transformation**.
	
	■ Click **+ Add xslt document** to add an xslt document and provide the following information:
	
	■ **XSLT file**. Specifies the XSLT file used to transform the response messages as required.
	
	Click **Browse** to browse and select a file.
	
	■ **Feature Name**. Specifies the name of the XSLT feature.
	
	■ **Feature value**. Specifies the value of the XSLT feature.
	
	You can add more XSLT features and xslt documents by clicking  .
	
	Note:
	API Gateway supports XSLT 1.0 and XSLT 2.0.
	
	■ Click **+ Add xslt transformation alias** and provide the following information:
	
	■ **XSLT Transformation alias**. Specifies the XSLT transformation alias
	
	When you receive the response in JSON, you can use a XSLT file similar to the below sample:
	
	`<?xml version="1.0" ?>`
	`<xsl:stylesheet version="1.1"`
	`xmlns:xsl="http://www.w3.org/1999/XSL/Transform">`
	`<xsl:output method="xml"/>`
	`<xsl:template match="/" >`
	`<xsl:element name="fakeroot">`
	`<xsl:element name="fakenode">`
	`<!-- Apply your transformation rules based on the response received from the native API-->`
	`</xsl:element>`
	`</xsl:element>`
	`</xsl:template>`
	`</xsl:stylesheet>`
	
	When you receive the response in XML, you can use a XSLT file similar to the below sample:

Property	Description
	<?xml version="1.0" ?> <xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"> <xsl:output method="xml"/> <xsl:template match="/" > <xsl:element name="soapenv:Envelope"> <xsl:element name="soapenv:Body"> <!-- Apply your transformation rules based on the response received from the native API--> </xsl:element> </xsl:element> </xsl:template> </xsl:stylesheet>
Advanced Transformation	Specifies the advanced transformation to be configured for the responses received from the native API.. Provide the following information: ■ **webMethods IS Service**. Specify the webMethods IS service to be invoked to process the response messages. You can add multiple services by clicking  . For details about usage of Invoke webMethods IS policy in versions 10.2 and higher, see "Invoke webMethods IS Policy" on page 421. **Note:** The webMethods IS service must be running on the same Integration Server as API Gateway. ■ **Run as User**. Specifies the authentication mode to invoke the IS service. If this field is left blank the incoming credentials of the user, identified by API Gateway, are used to authenticate and invoke the IS service. You can also specify a particular user, you want API Gateway to use to run the IS service. ■ **Comply to IS Spec**. Mark this as `true` if you want the input and the output parameters to comply to the IS Spec present in pub.apigateway.invokeISService.specifications folder in wmAPIGateway package. ■ **webMethods IS Service alias**. Specifies the webMethods IS service alias to be invoked to pre-process the request messages.

Property	Description
Transformation Metadata. Specifies the metadata for transformation of the responses received from the native API. For example, the namespaces configured in this section can be used when you provide the syntax for XPath ${response.payload.xpath} For example: ${response.payload.xpath[//ns:emp/ns:empName]}	
Namespace	Specifies the namespace information to be configured for transformation. Provide the following information: ■ **Namespace Prefix**. The namespace prefix of the payload expression to be validated. For example, specify the namespace prefix as `SOAP_ENV`. ■ **Namespace URI**. The namespace URI of the payload expression to be validated. For example, specify the namespace URI as `http://schemas.xmlsoap.org/soap/envelope/`. This declaration defines SOAP_ENV as an alias for the namespace: http://schemas.xmlsoap.org/soap/envelope/. **Note:** You can add multiple namespace prefix and URI by clicking  .

Invoke webMethods IS Policy

This policy pre-processes the request messages and transforms the message into the format required by the native API or performs some custom logic, before API Gateway sends the requests to the native APIs.

For example, you might need to accommodate differences between the message content that a client is capable of submitting and the message content that a native API expects. For example, if the client submits an order record using a slightly different structure than the structure expected by the native API, you can use this action to process the record submitted by the client to the structure required by the native API.

This policy also processes the native API's response messages into the format required by the application, before API Gateway returns the responses to the application.

The transformations using Invoke webmethods IS policy include Header, Query Parameter, Path Parameter transformation, HTTP Method transformation, Payload transformation, Status Code, and Status Message.

When can you use Invoke webmethods IS policy?

You can use Invoke webmethods IS policy:

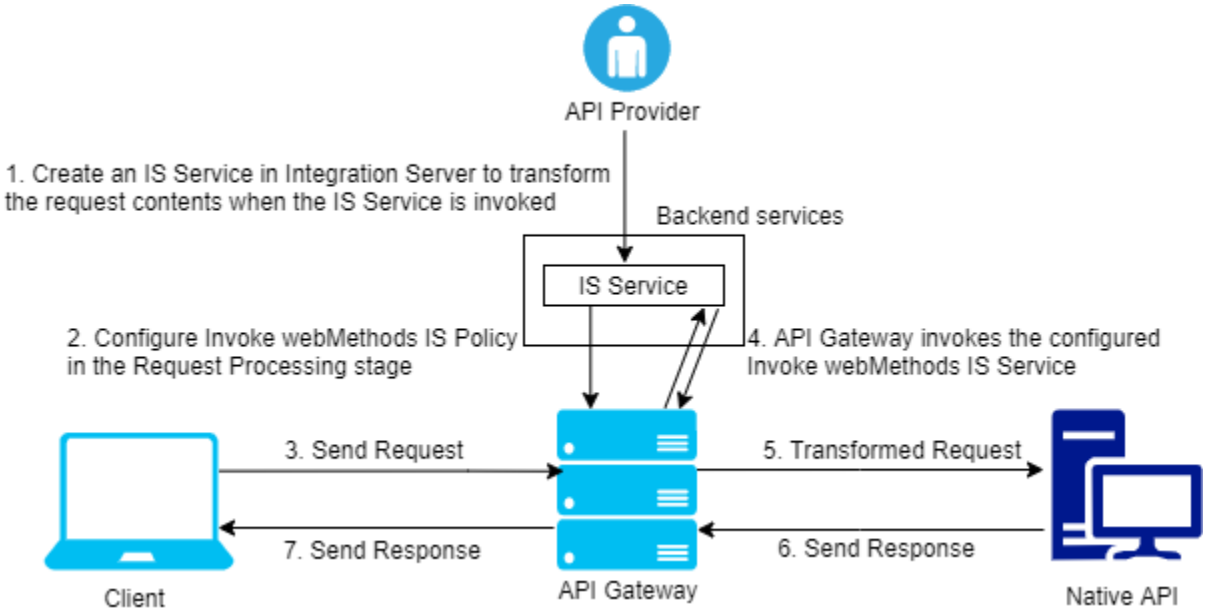
- When as an API Provider wants to read the contents of the request and response to do audit logging, or trigger a notification based on the contents of the request.
- When the API Provider wants to modify the request before forwarding the request to native API as the native API wants to identify all incoming requests from API Gateway. In such case the API Provider can configure the Invoke webmethods IS policy to add a header to all requests before they get routed to the native API.
- When the API Provider wants to achieve complex use cases of transformation by writing an Invoke IS Service.
- When the API Provider wants to write some custom logic using Java code to do the transformation.

How do I transform a request using Invoke webMethods IS policy?

Use the Invoke webMethods IS policy to modify the contents of an incoming request such as headers, payload, query parameters, path parameters, HTTP method using the configurations given by the API Provider.

The Invoke webMethods IS workflow is as follows:

1. The API Provider creates an IS Service in Integration Server in which API Gateway is running. The API Provider configures the IS Service to transform the request contents as per their need.
2. The API Provider configures the Invoke webMethods IS policy in the Request Processing stage of API Gateway with the created IS Service.
3. The client sends the request to API Gateway.
4. API Gateway invokes the webMethods IS Service configured by the API Provider. The IS Service transforms the request contents as defined by the API Provider.
5. API Gateway sends the transformed request to the native API.
6. Native API processes the transformed request and sends the response to API Gateway.
7. API Gateway forwards the response to the client.



To configure Invoke webMethods IS policy in the Request Processing stage:

1. Click **APIs** in the title navigation bar.

   A list of available APIs appears.

2. Select a Rest API from the list of APIs and click **Edit**.

3. Select **Policies > Request Processing > Request Transformation**.

   The Request Transformation section appears.

4. In the **Condition** section, select **OR**.

   The configured transformation is applied when at least one of the conditions is satisfied.

   Note:
   The condition can also be set to AND operator. The configured transformation is applied only when all the set conditions are satisfied.

5. Click **Add Condition** to configure the conditions to evaluate the contents on the request.

   a. Specify the **Variable**. Example, Content-Type.

   b. Specify the **Operator** to use to relate variable and the value provided. Example, **Equals**.

   c. Specify the **Value**. Example, application/json.

When you select the operator - **Equals**, the Condition checks if the **Variable**: Content-Type is equal to the **Value**: application/json.

d. Click **Add**.

6. Select **Transformation Configuration > Advanced Transformation**.

   The Advanced Transformation section appears.

7. In **webMethods IS Service** section, click **+ Add webmethods is service**.

8. Provide the following information.

   - **webMethods IS Service**. Specify the webMethods IS service to be invoked to process the request messages.

     You can add multiple services by clicking 

.

     Note:
     The webMethods IS service must be running on the same Integration Server as API Gateway.

   - **Run as User**. Specifies the authentication mode to invoke the IS service. If this field is left blank the incoming credentials of the user, identified by API Gateway, is used to authenticate and invoke the IS service. You can also specify a particular user, you want API Gateway to use to invoke the IS service.

   - **Comply to IS Spec**. Mark this as `true` if you want the input and the output parameters to comply to the IS Spec present in pub.apigateway.invokeISService.specifications folder in wmAPIGateway package.

   - **webMethods IS Service alias**. Specifies the webMethods IS service alias to be invoked to pre-process the request messages.

   Note:
   For details about the variables available in API Gateway, see "Invoke webMethods IS Policy Properties for Request Processing" on page 424.

9. Click **Save**.

This Invoke webMethods IS policy modifies the contents of an incoming request based on the IS Service invoked.

Invoke webMethods IS Policy Properties for Request Processing

If `Comply to IS Spec` parameter is configured as `true`, API Gateway invokes the IS Service with IS specification in the path pub.apigateway.invokeISService.specifications:RequestSpec for Request Processing

The following are the input and output parameters for REST, SOAP, and WebSocket APIs as specified in the above IS Specification.

API type	Input parameters	Output parameters
REST	■ headers ■ query ■ payload ■ path ■ httpMethod ■ messageContext ■ apiName ■ requestUrl ■ correlationID (this is unique for request and response)	■ headers ■ query ■ payload ■ path ■ httpMethod ■ messageContext
SOAP	■ headers ■ payload ■ messageContext ■ apiName ■ payloadObject ■ requestUrl ■ correlationID (this is unique for request and response)	■ headers ■ payload ■ messageContext ■ payloadObject
WebSocket	■ headers ■ payload (this is applicable when the message type is Text) ■ payloadObject (this is applicable when the message type is Binary) ■ messageContext ■ apiName ■ requestUrl ■ websocketInfo	■ headers ■ payload ■ messageContext ■ payloadObject

API type	Input parameters	Output parameters
	■ correlationID (this is unique for request and response)	

By default the "query" pipeline variable is a key value pair, where the value is of type string. But, if the incoming request contains multiple values for the same query parameter and if you want to access those multiple values using **webMethods IS Service**, you have to ensure two things:

1. Make sure that you have checked the **Repeat** check box for query parameter in the **Add Resource Parameter** section of the API details screen.
2. To access or transform multiple values of that query parameter, you have to insert string list (instead of string) under the "query" pipeline variable in the webMethods IS Service.

Note:

- For SOAP to REST APIS, the payload contains the transformed SOAP request.
- Payload transformation does not happen automatically for content-type transformation. When you change the content type, ensure to do payload transformation also as part of IS Service. For example, if you change the content-type header from application/xml to application/json using IS service, you must also change the respective payload from application/xml to application/json
- Only Method Transformation happens when configured, but you have to take care of adding payload during transformations involving method change like GET to POST, and so on.
- When `Comply to IS spec` is `true`, you can change the values of headers, query, payload, and so on, programatically using Message Context, as well as using the pipeline variables given. Software AG recommends you do not change those values directly in Message Context, as the values in output pipeline variables are written to Message Context after the invocation of IS Service.

If `Comply to IS Spec` parameter is set to `false`, API Gateway invokes the IS Service with the same input and output parameters supported in 10.1 and the earlier versions:

- proxy.name
- JSONRESTContentString (REST only)
- SOAPEnvelope (SOAP only)
- EnvelopeString (SOAP only)

The table lists the properties that you can specify for this policy:

Property	Description
Invoke webMethods Integration Server Service	
Add invoke webMethods Integration Server service	Specifies the webMethods IS service to be invoked to pre-process the request messages and the authentication mode for the IS service. Provide the following information:

Property	Description

	■ **webMethods IS Service**. Specify the webMethods IS service to be invoked to pre-process the request messages.
	
	The webMethods IS service must be running on the same Integration Server as API Gateway .
	
	Note:
	If an exception occurs when invoking the webMethods IS service, by default API Gateway displays the status code as *500* and error message as *Internal Server Error*.
	
	You can set custom status code and error message by setting the following properties in the message context of the webMethods IS service:
	■ service.exception.status.code
	■ service.exception.status.message
	
	The sample code is given below:
	`IDataCursor idc = pipeline.getCursor();`
	`MessageContext context =`
	`(MessageContext)IDataUtil.get(idc,"MessageContext");`
	`if(context != null)`
	`{`
	`context.setProperty("service.exception.status.code",`
	`404);`
	`context.setProperty("service.exception.status.message",`
	`"Object Not Found");`
	`throw new ServiceException();`
	`}`
	
	Note:
	If ServiceException or FlowException occurs when invoking webMethods IS Service, the message given in the exception is displayed to the client. If any other exception occurs, a generic error message is displayed to the client.
	
	■ **Run as User**. Specifies the authentication mode to invoke the IS service. If this field is left blank the incoming credentials of the user, identified by API Gateway, are used to authenticate and invoke the IS service. You can also specify a particular user, you want API Gateway to invoke the IS service.
	
	Note:
	It is the responsibility of the user who activates the API to review the value configured in **Run as User** field to avoid misuse of this configuration.
	
	■ **Comply to IS Spec**. Mark this as `true` if you want the input and the output parameters to comply to the IS Spec present in

Property	Description
	pub.apigateway.invokeISService.specifications folder in wmAPIGateway package. **Note:**Software AG recommends users to configure the policy with `Comply to IS Spec` as `true`, as you can read or change the values of headers, and so on, without having to read from or write to the message context.
webMethods IS Service alias	Specifies the webMethods IS service alias to be invoked to pre-process the request messages. Start typing the webMethods alias name, select the alias from the type-ahead search results displayed and click + Add to add one or more aliases. You can use the delete icon to delete the added aliases from the list.

Adding Custom Fields to Transactional Events

This section explains you how to add custom fields to the transactional events.

1. Create webMethods IS service by specifying the *pub.apigateway.utils:customFieldInTransactionEventSpec* as a specification reference.
2. In the webMethods IS service, set the required custom fields in the *customFieldsMap* output variable.
3. Once when *customFieldsMap* gets created, the custom fields will be available in the transactional events.
4. Invoke the API with the Invoke webMethods IS policy.

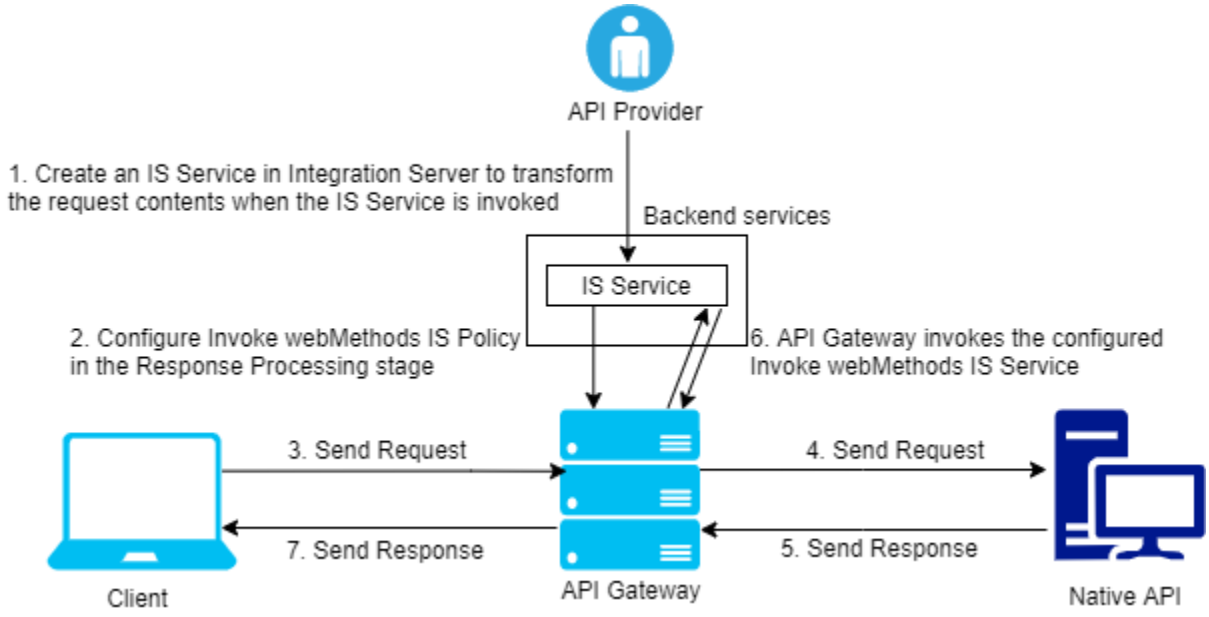
Note:
You can also add the custom fields to the transactional events from API Gateway by configuring the *customTransactionFields.FIELD_NAME* custom variable in the **Custom Extension** policy. For more details, see "How Do I Define a Custom Variable?" on page 397.

How do I transform a response using Invoke webMethods IS policy?

Use the Invoke webMethods IS policy to modify the contents of an outgoing response such as headers, payload, query parameters, path parameters, HTTP method using the configurations given by the API Provider.

The Invoke webMethods IS workflow is as follows:

1. The API Provider creates an IS Service in Integration Server in which API Gateway is running. The API Provider configures the IS Service to transform the response contents as per their need.
2. The API Provider configures the Invoke webMethods IS policy in the Response Processing stage of API Gateway with the created IS Service.
3. The client sends the request to API Gateway.
4. API Gateway forwards the request to native API.
5. Native API processes the request and sends the response to API Gateway.
6. API Gateway invokes the webMethods IS Service configured by the API Provider. The IS Service transforms the response contents as defined by the API Provider.
7. API Gateway forwards the transformed response to the client



To configure Invoke webMethods IS policy in the Response Processing stage:

1. Click **APIs** in the title navigation bar.

   A list of available APIs appears.

2. Select a Rest API from the list of APIs and click **Edit**.

3. Select **Policies > Response Processing > Response Transformation**.

   The Response Transformation section appears.

4. In the **Condition** section, select **OR**.

The configured transformation is applied when at least one of the conditions is satisfied.

Note:
The condition can also be set to AND operator. The configured transformation is applied only when all the set conditions are satisfied.

5. Click **Add Condition** to configure the conditions to evaluate the contents on the response.

   a. Specify the **Variable**. Example, Content-Type.

   b. Specify the **Operator** to use to relate variable and the value provided. Example, **Equals**.

   c. Specify the **Value**. Example, application/json.

      When you select the operator - **Equals**, the Condition checks if the **Variable**: Content-Type is equal to the **Value**: application/json.

   d. Click **Add**.

6. Select **Transformation Configuration > Advanced Transformation**.

   The Advanced Transformation section appears.

7. In **webMethods IS Service** section, click **+ Add webmethods is service**.

8. Provide the following information.

   - **webMethods IS Service**. Specify the webMethods IS service to be invoked to process the request messages.

     You can add multiple services by clicking 

.

     Note:
     The webMethods IS service must be running on the same Integration Server as API Gateway.

   - **Run as User**. Specifies the authentication mode to invoke the IS service. If this field is left blank the incoming credentials of the user, identified by API Gateway, are used to authenticate and invoke the IS service. You can also specify a particular user, you want API Gateway to use to invoke the IS service.

   - **Comply to IS Spec**. Mark this as `true` if you want the input and the output parameters to comply to the IS Spec present in pub.apigateway.invokeISService.specifications folder in wmAPIGateway package.

   - **webMethods IS Service alias**. Specifies the webMethods IS service alias to be invoked to pre-process the request messages.

   Note:

For details about the variables available in API Gateway, see "Invoke webMethods IS Policy Properties for Response Processing" on page 431.

9. Click **Save**.

This Invoke webMethods IS policy modifies the contents of an outgoing response to the client based on the IS Service invoked.

Invoke webMethods IS Policy Properties for Response Processing

If `Comply to IS Spec` parameter is configured as `true`, API Gateway invokes the IS Service with IS specification in the path pub.apigateway.invokeISService.specifications:ResponseSpec (for Response Processing)

The following are the input and output parameters for REST, SOAP, and WebSocket APIs as specified in the above IS Specification.

API type	Input parameters	Output parameters
REST	■ headers ■ payload ■ messageContext ■ statusCode ■ statusMessage ■ apiName ■ requestUrl ■ correlationID (this is unique for request and response)	■ headers ■ payload ■ messageContext ■ statusCode ■ statusMessage
SOAP	■ headers ■ payload ■ messageContext ■ statusCode ■ statusMessage ■ apiName ■ payloadObject ■ requestUrl	■ headers ■ payload ■ messageContext ■ statusCode ■ statusMessage

API type	Input parameters	Output parameters
	■ correlationID (this is unique for request and response)	
WebSocket	■ headers ■ payload (this is applicable when the message type is Text) ■ payloadObject (this is applicable when the message type is Binary) ■ messageContext ■ apiName ■ requestUrl ■ websocketInfo ■ correlationID (this is unique for request and response)	■ headers ■ payload ■ messageContext ■ payloadObject

Note:

- For SOAP to REST APIS, the payload contains the transformed JSON response.
- Payload transformation does not happen automatically for content-type transformation. When you change the content type, ensure to do payload transformation also as part of IS Service.
- When `Comply to IS spec` is true, you can change the values of headers, query, payload, and so on, programatically using Message Context, as well as using the pipeline variables given. Software AG recommends you do not change those values directly in Message Context, as the values in output pipeline variables are written to Message Context after the invocation of IS Service.

If `Comply to IS Spec` parameter is set to `false`, API Gateway invokes the IS Service with the same input and output parameters supported in 10.1 and the earlier versions::

- proxy.name
- JSONRESTContentString (REST only)
- SOAPEnvelope (SOAP only)
- EnvelopeString (SOAP only)

The table lists the properties that you can specify for this policy:

Property	Description
Invoke webMethods Integration Server Service	

Add invoke webMethods Integration Server service	Specifies the webMethods IS service to be invoked to process the response messages and the authentication mode for the IS service.
	
	Provide the following information:
	
	■ **webMethods IS Service**. Specify the webMethods IS service to be invoked to pre-process the response messages.
	
	The webMethods IS service must be running on the same Integration Server as API Gateway
	
	Note:
	If an exception occurs when invoking the webMethods IS service, by default API Gateway displays the status code as *500* and error message as *Internal Server Error*.
	
	You can set custom status code and error message by setting the following properties in the message context of the webMethods IS service:
	■ service.exception.status.code
	■ service.exception.status.message
	
	The sample code is given below:
	`IDataCursor idc = pipeline.getCursor();`
	`MessageContext context =`
	`(MessageContext)IDataUtil.get(idc,"MessageContext");`
	`if(context != null)`
	`{`
	`context.setProperty("service.exception.status.code",`
	`404);`
	`context.setProperty("service.exception.status.message",`
	`"Object Not Found");`
	`throw new ServiceException();`
	`}`
	
	Note:
	If ServiceException or FlowException occurs when invoking webMethods IS Service, the message given in the exception is displayed to the client. If any other exception occurs, a generic error message is displayed to the client.
	
	■ **Run as User**. Specifies the authentication mode to invoke the IS service. If this field is left blank the incoming credentials of the user, identified by API Gateway, are used to authenticate and invoke the IS service. You can also specify a particular user, you want API Gateway to invoke the IS service.

Property	Description
	Note: It is the responsibility of the user who activates the API to review the value configured in **Run as User** field to avoid misuse of this configuration. ■ **Comply to IS Spec**. Mark this as `true` if you want the input and the output parameters to comply to the IS Spec present in pub.apigateway.invokeISService.specifications folder in wmAPIGateway package. **Note:**Software AG recommends users to configure the policy with `Comply to IS Spec` as `true`, as you can read or change the values of headers, and so on, without having to read from or write to the message context.
webMethods IS Service alias	Specifies the webMethods IS service alias used to invoke the webMethods IS service to pre-process the response messages. Start typing the webMethods alias name, select the alias from the type-ahead search results displayed and click + Add to add one or more aliases. You can use the delete icon to delete the added aliases from the list.

Adding Custom Fields to Transactional Events

This section explains about how to add custom fields to the transactional events.

1. Create webMethods IS service by specifying the *pub.apigateway.utils:customFieldInTransactionEventSpec* as a specification reference.
2. In the webMethods IS service, set the required custom fields in the *customFieldsMap* output variable.
3. Once when *customFieldsMap* gets created, the custom fields will be available in the transactional events.
4. Invoke the API with the Invoke webMethods IS policy.

Note:
You can also add the custom fields to the transactional events from API Gateway by configuring the *customTransactionFields.FIELD_NAME* custom variable in the **Custom Extension** policy. For more details, see "How Do I Define a Custom Variable?" on page 397.

System Context Variables

API Gateway provides predefined system context variables and you can declare your own custom context variables. Any context variable state defined during the inbound request processing steps is available during the outbound response processing steps. To set, get, or remove the predefined context variables, use "The API for Context Variables" on page 437 provided in API Gateway.

The table lists the predefined system context variables that you can configure in the conditional routing policy through the API Gateway user interface.

System Context Variable Name	Description
`User`	The identified API Gateway user for the current request.
`Inbound HTTP method`	The HTTP method used by the client to send the request. For example, GET, POST, PUT, DELETE, and PATCH.
`Routing method`	The HTTP method used by the routing policy when you select CUSTOM as the HTTP method. If you do not define this context variable, then the method used is from the Inbound HTTP method.
`Inbound content type`	Content type of the request.
`Inbound accept`	Accept header in the incoming request from the client.
`Inbound protocol`	The protocol of the request. For example, HTTP or HTTPS.
`Inbound request URI`	A partial reference to an API (for HTTP and HTTPS only). The protocol, host and port are not part of the value. For example, if the API is invoked: http://*host*:*port*/gateway/API then the expected value of this variable would be /gateway/API. For a REST API, the URL also includes query string parameters. For example, if the following API is invoked: http://*host*:*port*/gateway/cars?vin=1234 the expected value of this variable would be /gateway/cars?vin1234.
`Inbound IP`	The Client IP address used to send the request.
`Gateway hostname`	API Gateway host name.
`Gateway IP`	API Gateway IP address.
`Operation name`	Operation name for SOAP APIs. It is empty for REST API.

System Context Variable Name	Description
`Native Endpoint`	Retrieves the native endpoint in the incoming request from the client.

The table lists the predefined context variables that you can set or get in API Gateway using an IS service. For details, see "The API for Context Variables" on page 437.

Context Variable Name	Description
`CONSUMER_APPLICATION`	The name of the consumer application accessing the API.
`INTERVAL_FAULT_COUNT`	The number of service faults for the interval.
`INTERVAL_SUCCESS_COUNT`	The number of success counts for a given API.
`INTERVAL_TOTAL_COUNT`	The total number of counts for a given service.
`AVG_SUCCESS_TIME`	The average amount of time it took the service to complete all invocations in the current interval. This is measured from the moment API Gateway receives the request until the moment it returns the response to the caller. **Note:** By default, average response time does not include metrics for failed invocations.
`FASTEST_SUCCESS_INVOKE`	Minimum Response Time. **Note:** By default, Minimum Response Time does not include metrics for failed invocations.
`SLOWEST_SUCCESS_INVOKE`	Maximum Response Time. **Note:** By default, Maximum Response Time does not include metrics for failed invocations.
`SOAP_HEADERS`	Contains an array of the SOAP header elements in the request.
`PROTOCOL_HEADERS`	Contains a map of key-value pairs in the request, where the values are provided as strings.
`SERVICE_NAME`	The name of the service.
`NATIVE_PROVIDER_ERROR`	The reason returned by the native provider in the case where it produced a SOAP fault. This will not contain API Gateway errors such as security policy enforcement errors. This variable only contains the reason text wrapped in a SOAP fault.

Context Variable Name	Description
	Note: When you use this variable in Conditional Error Processing message that you specify in the Response Processing step, note the following: if a request is denied due to security policy enforcement, the fault handler variable $ERROR_MESSAGE would contain a native service provider error message or other error messages that result from enforced security assertions. However, $NATIVE_PROVIDER_ERROR is null in this case.
`ROUTING_ENDPOINT`	API Gateway takes the ROUTING_ENDPOINT value from the message context and replaces the *${sys:dyn-Endpoint}* variable in the **Route Through** field of dynamic routing policy configuration.

The API for Context Variables

API Gateway provides an IS service that you can use to:

- Set, get, declare, and remove custom context variables.
- Set and get the predefined system context variables. (It is not allowed to declare or remove the predefined system context variables.)

API Gateway provides the following JAVA services, which are defined in the class ISMediatorRuntimeFacade.java:

- pub.apigateway.ctxvar:getContextVariable
- pub.apigateway.ctxvar:setContextVariable
- pub.apigateway.ctxvar:declareContextVariable
- pub.apigateway.ctxvar:removeContextVariable

pub.apigateway.ctxvar:getContextVariable

Use this JAVA service to retrieve a context variable's value and assign it to a pipeline variable. All parameter names are case-sensitive.

Parameter	Pipeline Type	Data Type	Description	Examples
MessageContext	in	Object ref	This object is inserted into the pipeline by API Gateway.	N/A
varName	in	String	Context variable name (system or custom).	For system context variable, use just the variable name to

Parameter	Pipeline Type	Data Type	Description	Examples
				get its value. For example, `PROTOCOL_HEADERS`. For custom context variable, use the prefix "**mx:**" with the variable name to get its value. For example, `mx:CUSTOM_VAR`
serValue	out	Object ref	Java.io.serializable value. (Usually a string).	

The table lists the predefined system context variables and its syntax used to get system context variables using `pub.apigateway.ctxvar:getContextVariable`.

System Context Variable Name	ctxVar IS Service Syntax	Set or Get Supported
`User`	USER	Supports get
`Inbound HTTP method`	INBOUND_HTTP_METHOD	Supports get
`Routing method`	ROUTING_METHOD	Supports get
`Inbound content type`	MESSAGE_TYPE	Supports get
`Inbound accept`	BUILDER_TYPE	Supports get
`Inbound protocol`	INBOUND_PROTOCOL	Supports get
`Inbound request URI`	INBOUND_REQUEST_URI	Supports get
`Inbound IP`	INBOUND_IP	Supports get
`Gateway hostname`	MEDIATOR_HOSTNAME	Supports get
`Gateway IP`	MEDIATOR_IP	Supports get
`Operation name`	OPERATION	Supports get
`Native Endpoint`	NATIVE_ENDPOINT	Supports get **Note:** This variable returns native endpoint value, only after Routing policy gets executed.
`Protocol headers`	PROTOCOL_HEADERS[xxx]	Supports set and get
`SOAP headers`	SOAP_HEADERS[xxx]	Supports set and get

Notes on getting and setting the PROTOCOL_HEADERS

All context variable values are typed as either `string` or `int` except for the predefined context variables, PROTOCOL_HEADERS, which is of the type `IData`. You can set or get value for PROTOCOL_HEADERS in one of the following ways:

- **set or get the entire structure.**

  To set the entire structure, you must:

  - Set the `varName` parameter in `pub.apigateway.ctxvar:setContextVariable` to `PROTOCOL_HEADERS`.
  - Use the method `ISMediatorRuntimeFacade.setContextVariableValue()`.

  To get the entire structure, you must:

  - Set the `varName` parameter in `pub.apigateway.ctxvar:getContextVariable` to `PROTOCOL_HEADERS`.
  - Use the method `ISMediatorRuntimeFacade.getContextVariableValue()`.

  If the varName is set to PROTOCOL_HEADERS, you get or set the entire IData structure containing all of the transport headers. The key is the transport header name (for example, Content-Type) and the value is a String. The IData object for PROTOCOL_HEADERS contains a set of string values where each IData string key matches the header name in the transport headers map. The set of possible keys includes the HTTP v1.1 set of headers as well as any custom key-value pairs you might have defined.

  Alternatively, you can set the varName parameter to address a specific element in the array. For example, setting it to PROTOCOL_HEADERS[Content-Type] would apply to the Content-Type transport header.

- **set or get a nested value.**

  Set a nested value in one of the following ways:

  - Set the `varName` parameter in `pub.apigateway.ctxvar:setContextVariable` to `PROTOCOL_HEADERS[`*`arrayElement`*`]`, where `[`*`arrayElement`*`]` refers to a specific element. For example, `PROTOCOL_HEADERS[Content-Type]` (to indicate the first array element in the set).
  - Alternatively, use the method `ISMediatorRuntimeFacade.setContextVariableValue()`. Use this method only if you are writing a JAVA service and you want to access it through the JAVA source code.

  Get a nested value in one of the following ways:

  - Set the `varName` parameter in `pub.apigateway.ctxvar:getContextVariable` to `PROTOCOL_HEADERS[`*`arrayElement`*`]`, where `[`*`arrayElement`*`]` refers to a specific element. For example, `PROTOCOL_HEADERS[Content-Type]` (to indicate the first array element in the set).
  - Alternatively, use the method `ISMediatorRuntimeFacade.getContextVariableValue()`. Use this method only if you are writing a JAVA service and you want to access it through the JAVA source code.

You can set or get a nested value inside PROTOCOL_HEADERS through an additional keyName. In this case, the object reference is *not* an IData object. For PROTOCOL_HEADERS, the keyName must match the transport header name in a case-sensitive manner (for example, `PROTOCOL_HEADERS[Content-Type]` or `PROTOCOL_HEADERS[Authorization]`). In this case, the Serializable value will be a string.

pub.apigateway.ctxvar:setContextVariable

Use this JAVA service to set a value on a context variable. The pipeline variable containing the context variable value should be an object reference that implements java.io.Serializable. All parameter names are case-sensitive.

Parameter	Pipeline Type	Data Type	Description	Examples
MessageContext	in	Object ref	This object is inserted into the pipeline by API Gateway.	N/A
varName	in	String	Context variable name (predefined or custom).	`PROTOCOL_HEADERS` `mx:CUSTOM_VAR`
serValue	in	Object ref	Java.io.serializable value. (Usually a string).	

pub.apigateway.ctxvar:declareContextVariable

Use this JAVA service to declare a *custom* context variable. All custom-defined context variables must be declared in a custom namespace that is identified by using the prefix `mx` (for example, `mx:CUSTOM_VARIABLE`). All parameter names are case-sensitive.

Note:
It is not legal to use this service to declare the predefined context variables; you can only declare custom variables.

Parameter	Pipeline Type	Data Type	Description
ctxVar	in	Object ref	The document type defining the context variable object. Use the ctxVar Document Type provided in the JAVA service pub.apigateway.ctxvar:ctxVar and map it to this input variable. Define the name (for example, mx:CUSTOM_VARIABLE), the schema_type (string or int), and isReadOnly (true or false).
ctxVar	out	Object ref	The set Context variable document type.

Parameter	Pipeline Type	Data Type	Description
varNameQ	out	Object ref	javax.xml.namespace.QName value. The QName of the variable.

Note the following:

- After declaring the context variable, you can use the setContext variable to set a value on the context variable.
- You do *not* need to declare the following kinds of context variables:
  - The predefined context variables provided by API Gateway. If you attempt to declare an existing predefined context variable, an error will occur.
  - Any custom context variable that you define in a routing rule that you create in the conditional routing step.
- Any custom context variables that you explicitly declare in source code using the API will have a declaration scope of SESSION.
- Any custom context variable's state that is defined during the inbound request processing steps will still be available during the outbound response processing steps.
- All context variable values are typed as either `string` or `int` (excluding the PROTOCOL_HEADERS variables, which are of the type `IData`).
- Valid names should be upper case (by convention) and must be a valid JAVA Identifier. In general, use alpha-numerics, $ or _ symbols to construct these context names. Names with punctuation, whitespace or other characters will be considered invalid and will fail deployment.
- All custom context variables must be declared in a custom namespace that is identified by using an `mx` prefix (for example, `mx:CUSTOM_VARIABLE`).
- To reference a custom context variable in a flat string, you need to prepend a $ symbol to the context variable name to indicate that variable's value should be referenced. Think of this usage as being similar to the `&` address operation for C variables.

  An expression that references a custom context variable might look like this:

  `$mx:TAXID=1234` or `$mx:ORDER_SYSTEM_NAME="Pluto"`

  Notice that the values of the data type "int" are not enclosed in quotation marks, while the values of the data type "string" are. The quotation marks are only needed if a context variable *expression* (as opposed to a reference) is defined.
- Referencing an undefined context variable does not result in an error.
- Once a variable has been declared it cannot be declared again.

pub.apigateway.ctxvar:removeContextVariable

Use this JAVA service to remove a *custom* context variable from a request or response session. All parameter names are case-sensitive.

Note:
Keep the following points in mind:

- It is not legal to use this service to remove any predefined context variables; you can only remove custom variables.
- Attempting to remove a non-existent context variable will *not* result in an error.

Parameter	Pipeline Type	Data Type	Description	Examples
MessageContext	in	Object ref	This object is inserted into the pipeline by API Gateway.	N/A
varName	in	String	Custom context variable name.	`mx:CUSTOM_VAR`

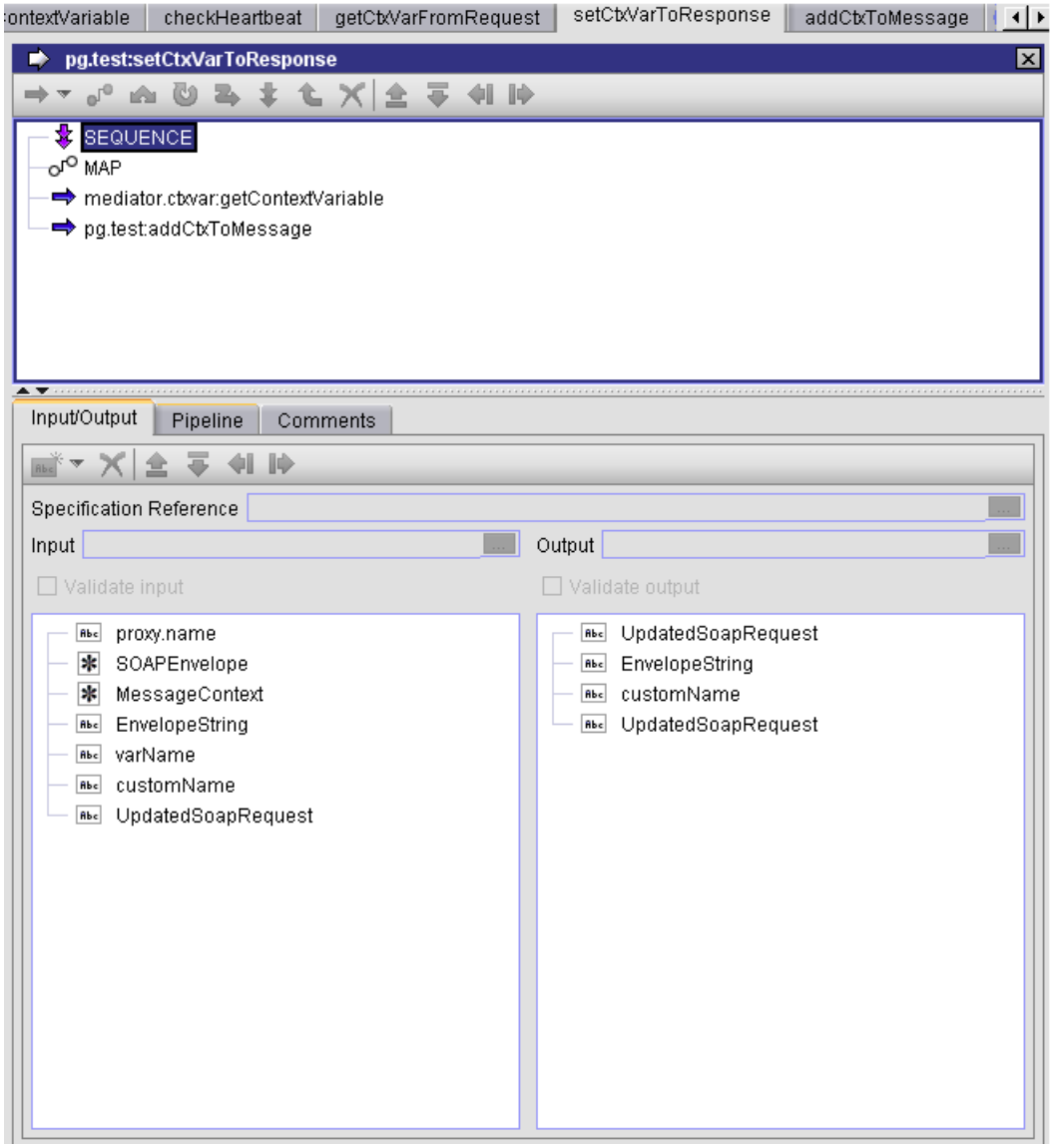
Sample Flow Service: Getting a Context Variable Value

This flow service sets the value of a custom context variable to be used in a response.

This flow service declares a pipeline variable named `customName`, which is set to the value `mx:COMP_TEST`.

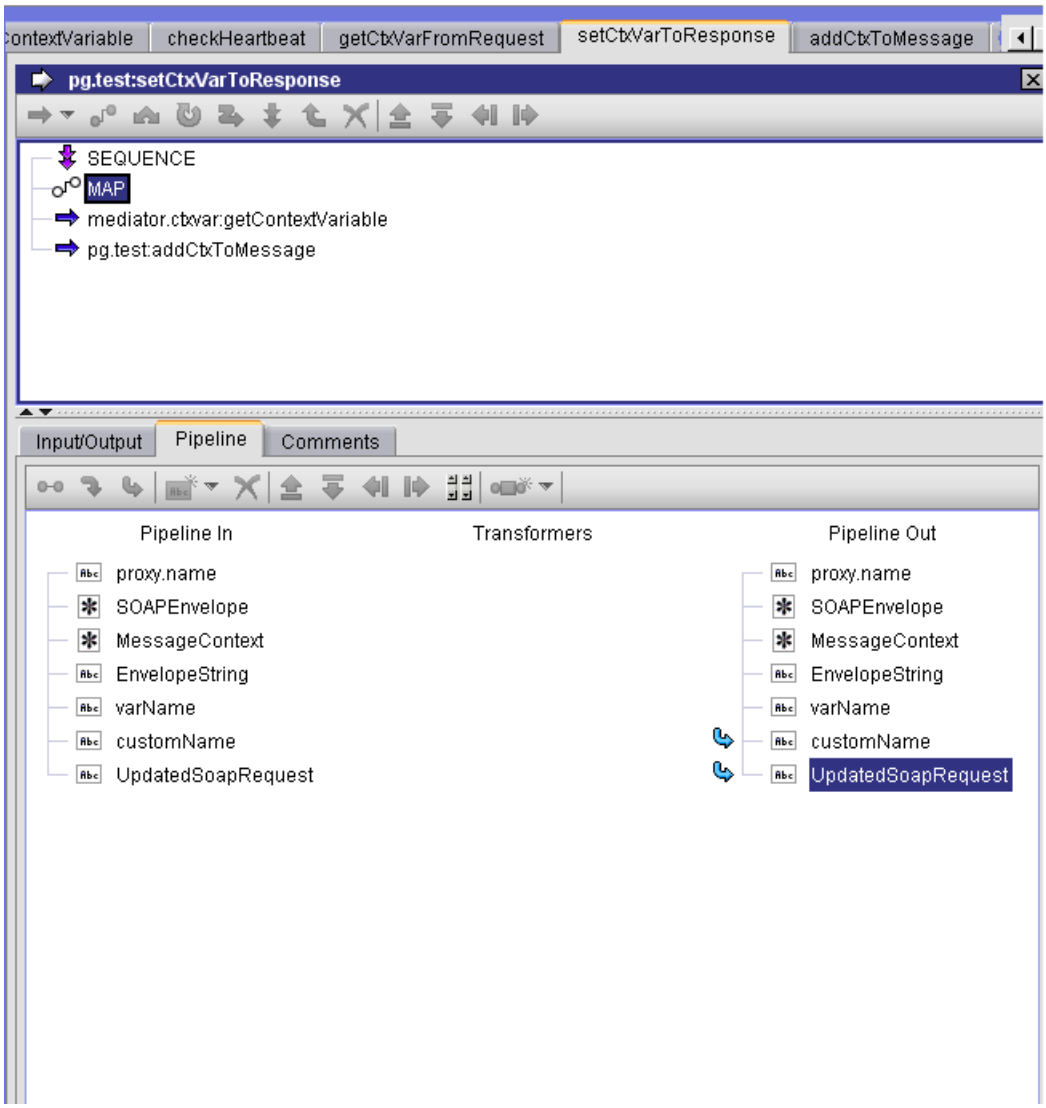
This flow service will retrieve the context variable for `customName` and create an element for its context variable value in the response message return to the consumer.

Step 1. Declaring customName



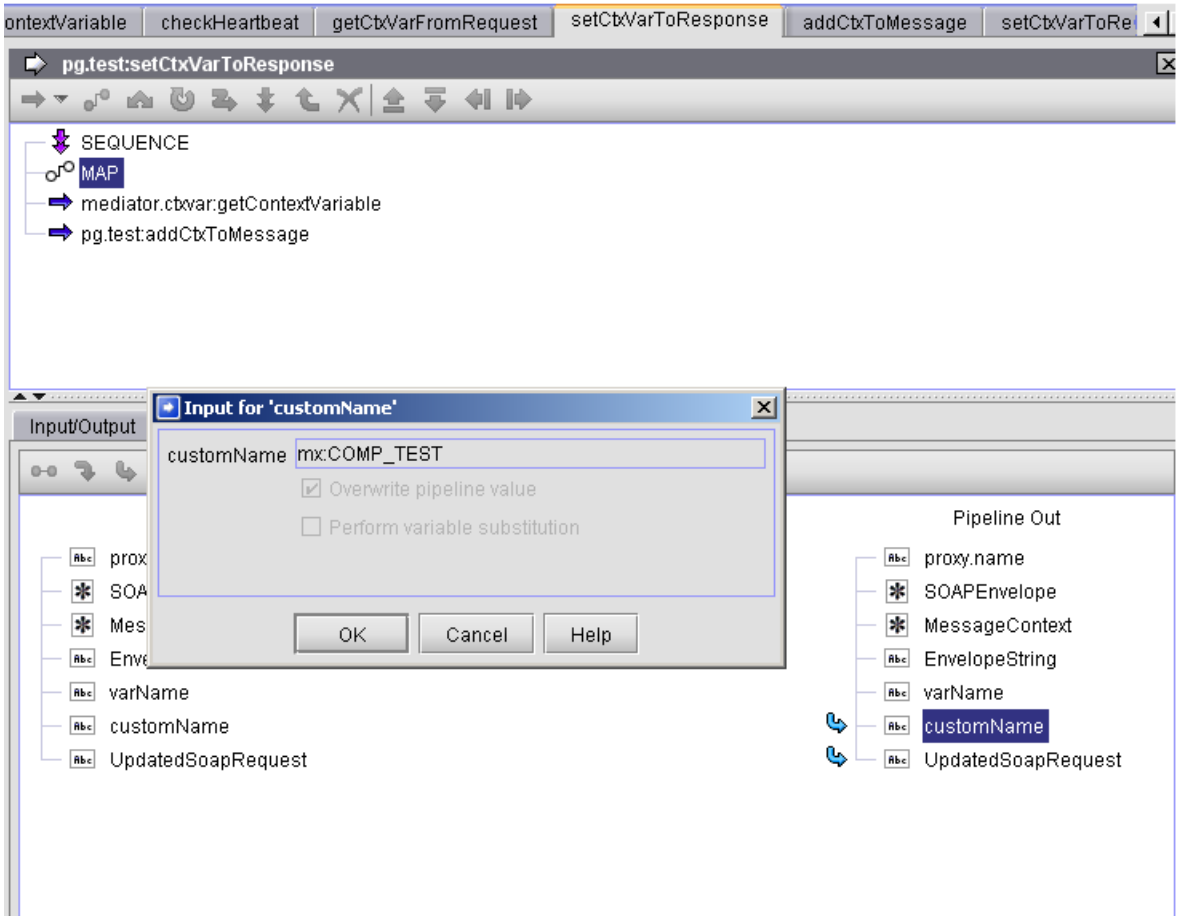
You can define the `customName` variable value to be `mx:COMP_TEST` so you can use this variable to lookup the custom variable name that was seeded in the previous example.

Step 2. Setting customName to mx:COMP_TEST



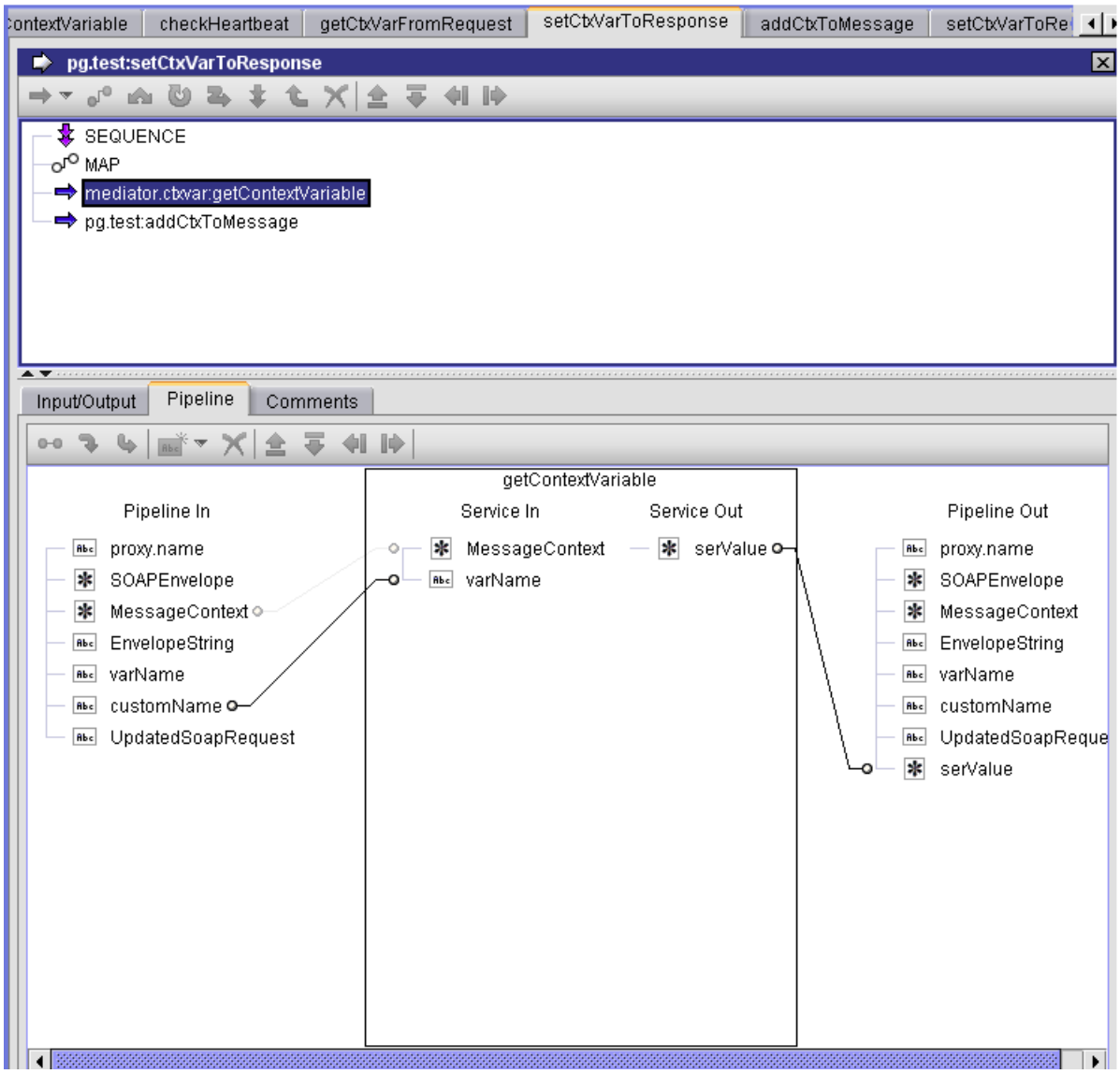
Clicking on the `customName` pipeline variable displays the name.

Step 3. Displaying the value of customName



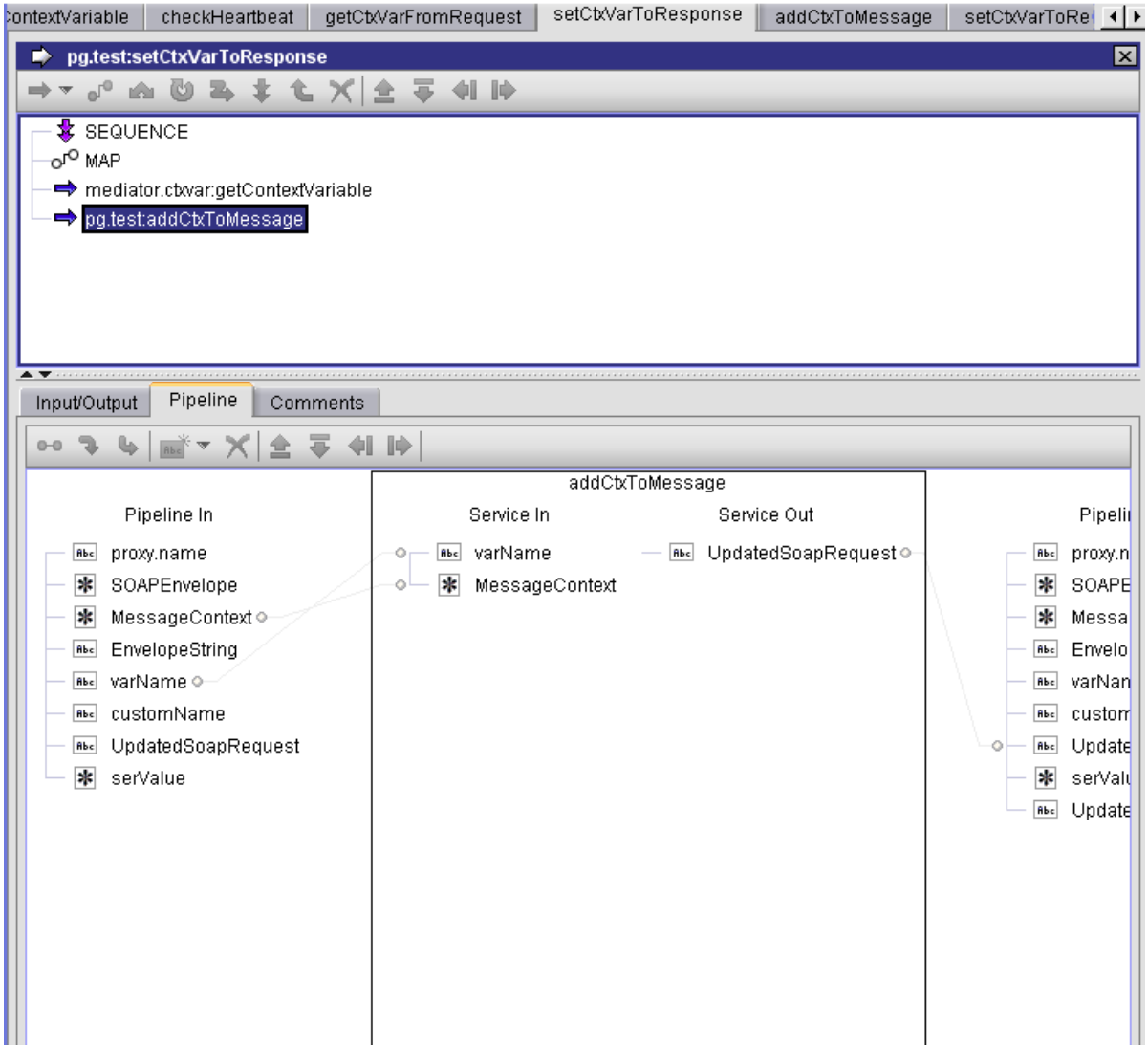
The call to `pub.mediator.ctxvar:getContextVariable` retrieves the value of the custom context variable from the context variable map.

Step 4. Calling meditor.ctxvar:getContextVariable



This is just a sample JAVA service that takes the context variable and creates a top-level element in the response message using the same name and value.

Step 5. Sample service using the context variable



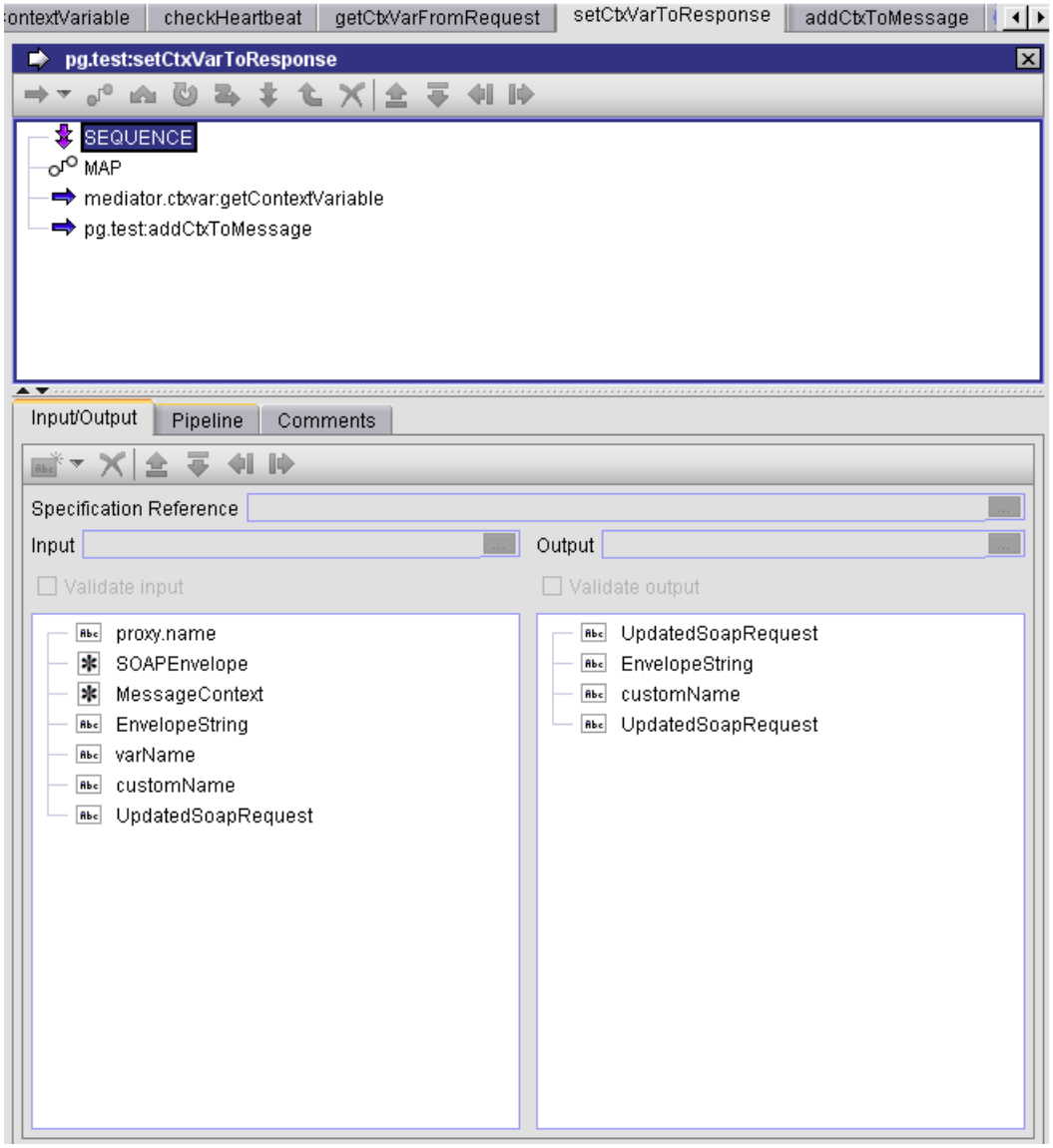
Sample Flow Service: Setting a Context Variable Value

This flow service sets the value of a custom context variable to be used in a response.

This flow service declares a pipeline variable named `customName`, which is set to the value `mx:COMP_TEST`.

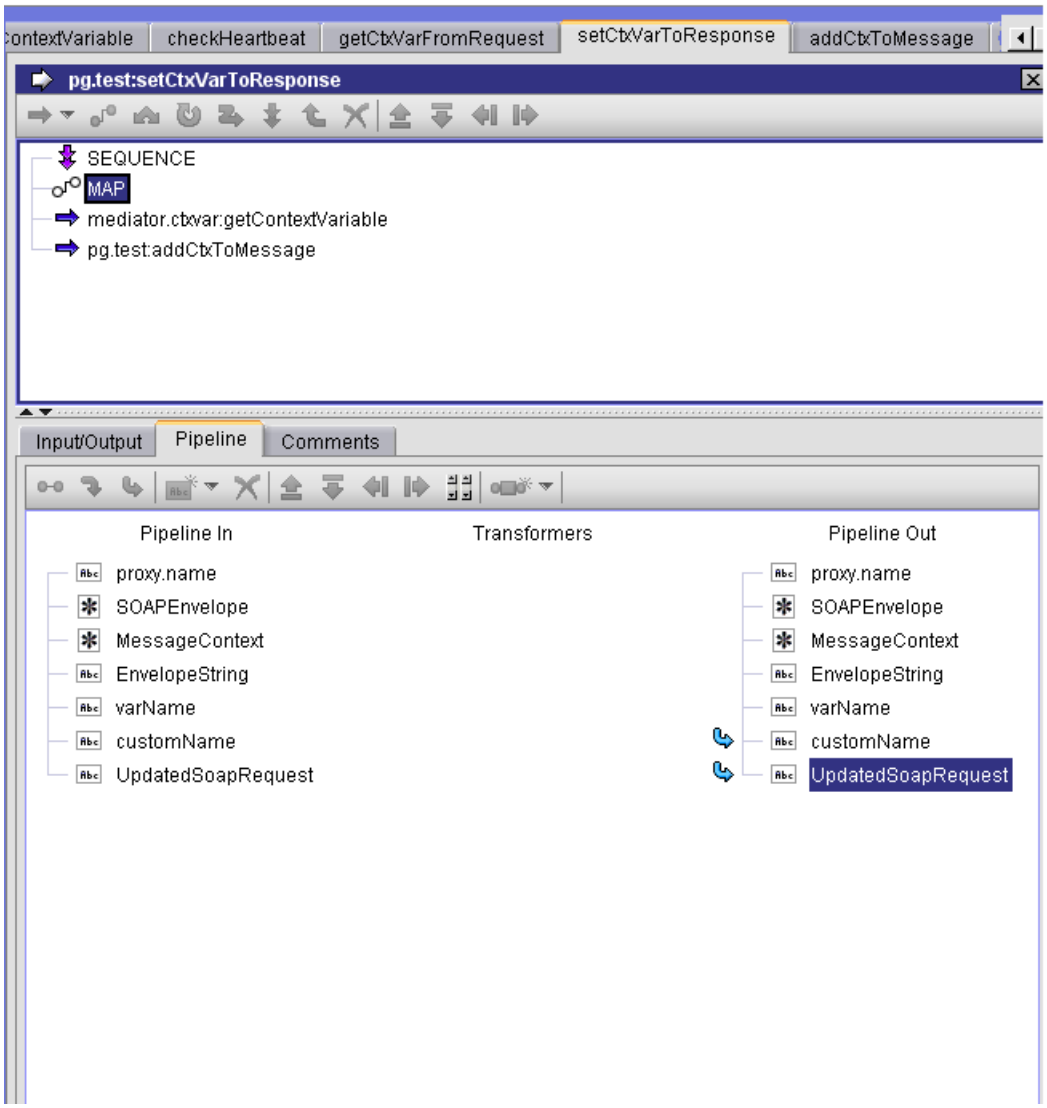
This flow service retrieves the context variable for `customName` and create an element for its context variable value in the response message return to the consumer.

Step 1. Declaring customName



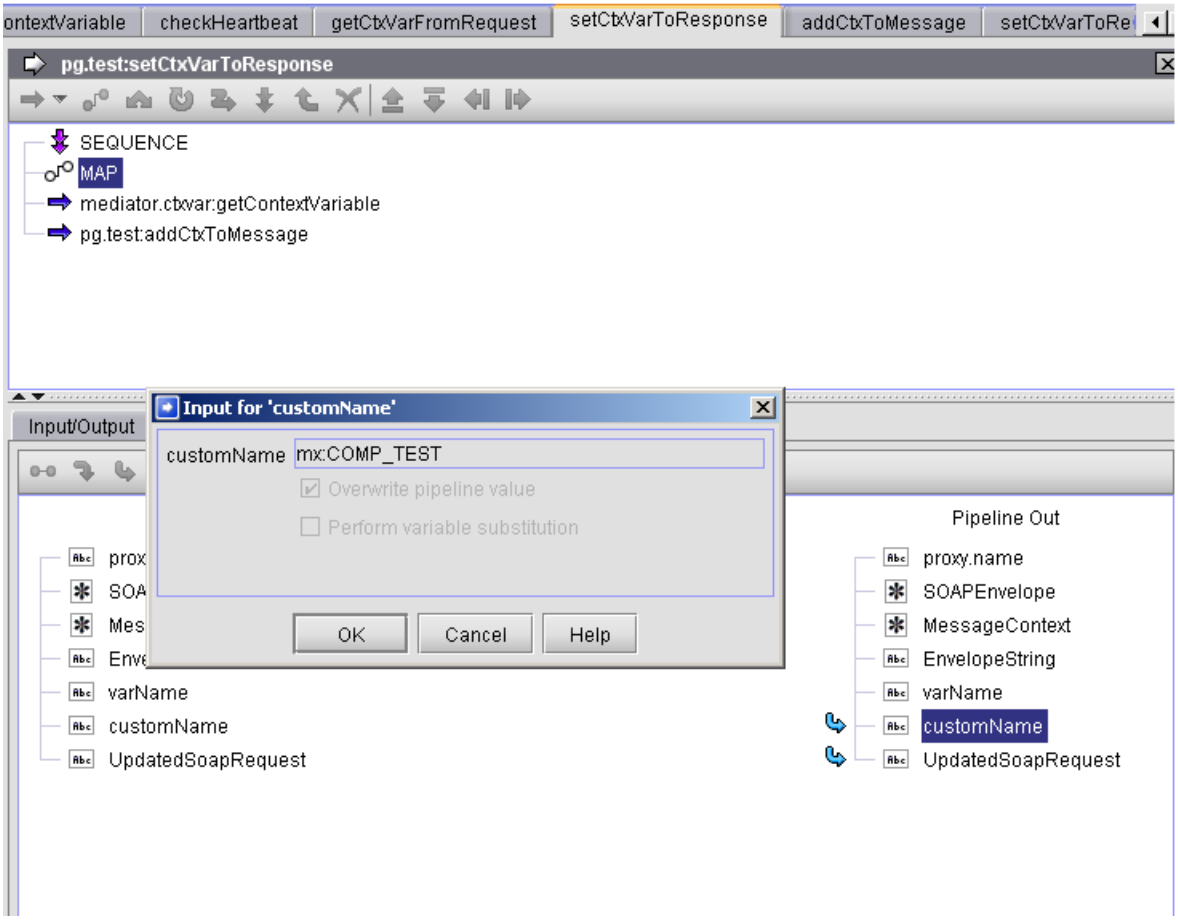
You define the `customName` variable value to be `mx:COMP_TEST` so you can use this variable to lookup the custom variable name that was seeded in the previous example.

Step 2. Setting customName to mx:COMP_TEST



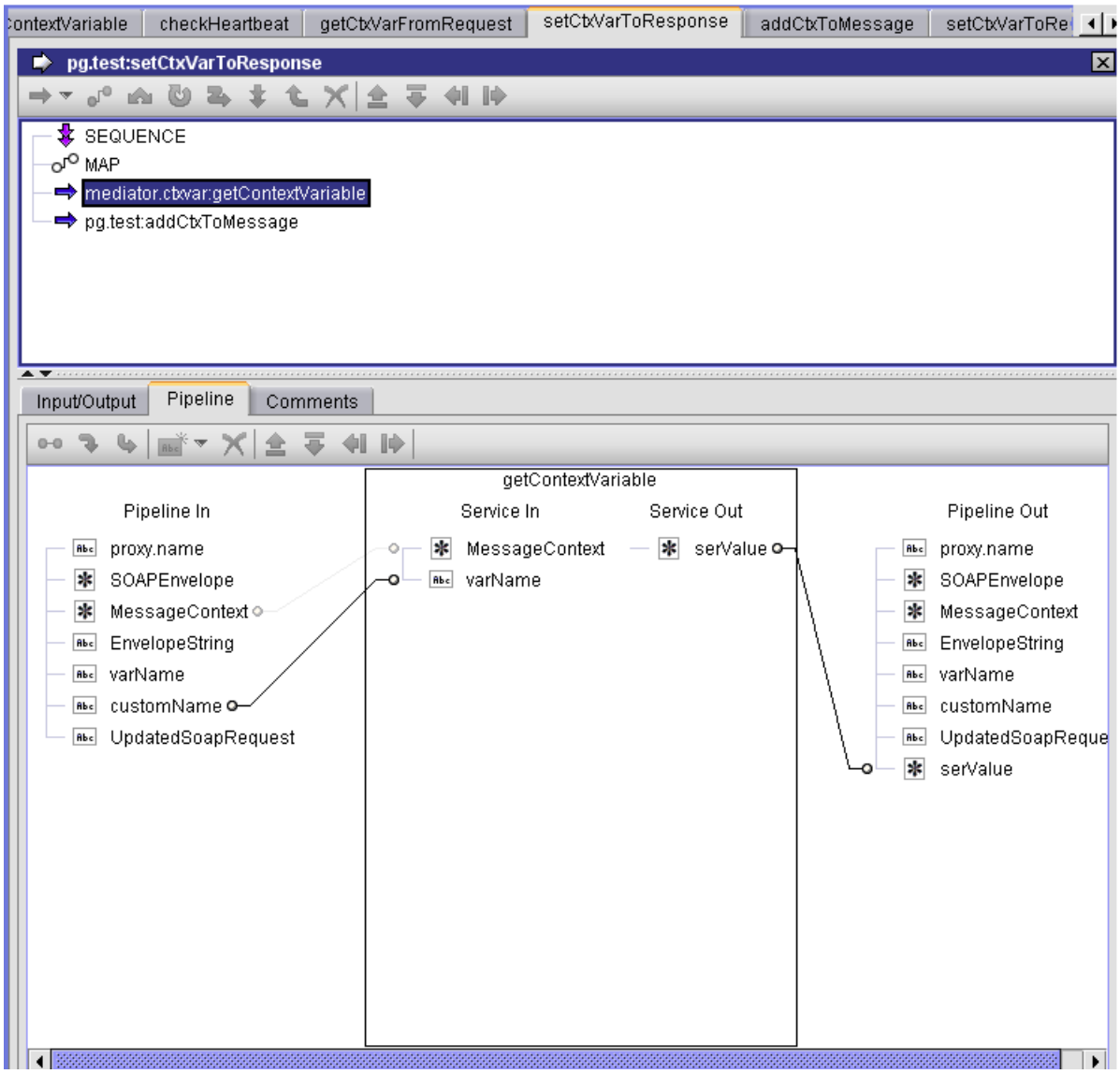
Clicking on the `customName` pipeline variable displays the name.

Step 3. Displaying the value of customName



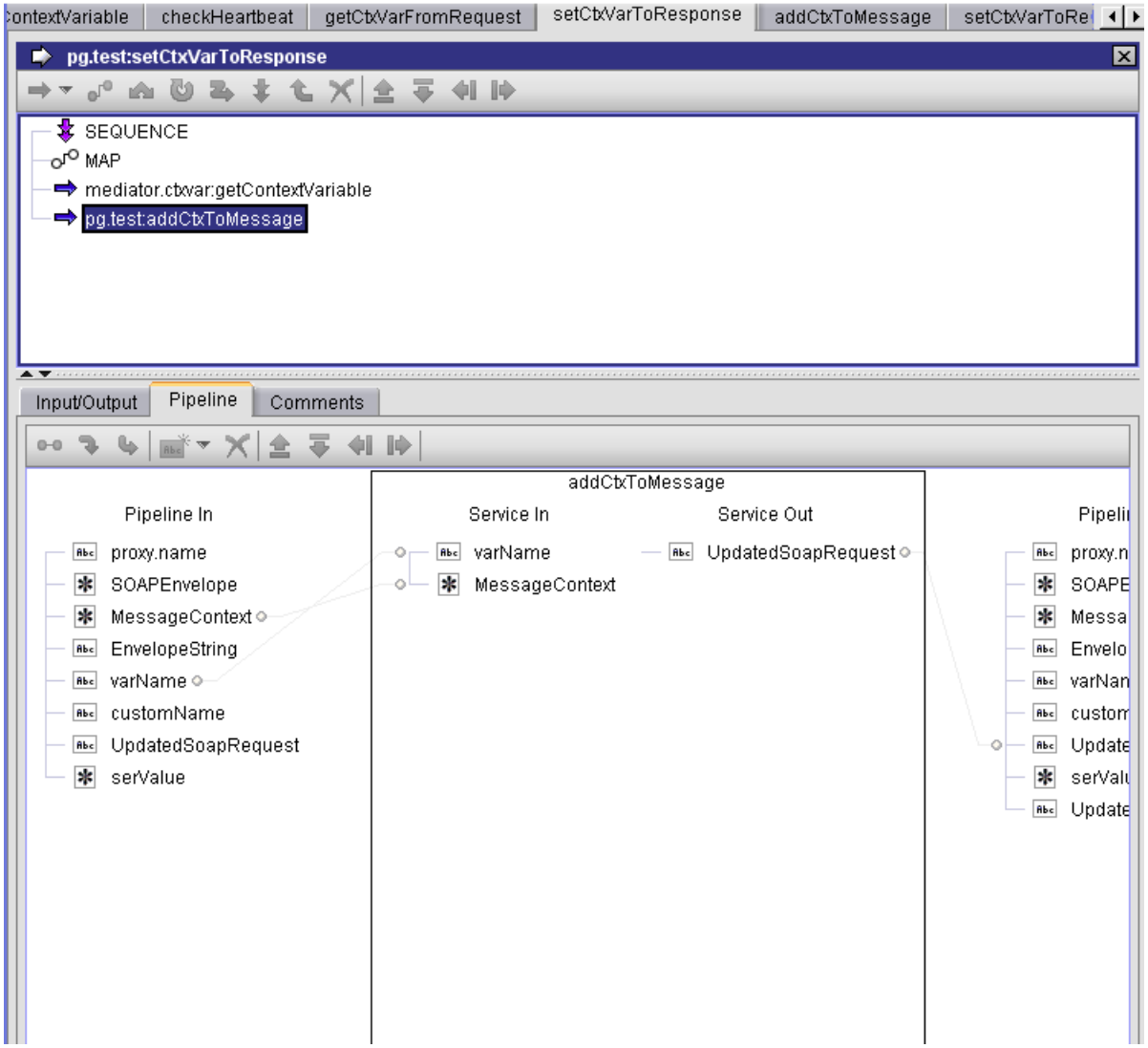
The call to `pub.mediator.ctxvar:getContextVariable` retrieves the value of the custom context variable from the context variable map.

Step 4. Calling meditor.ctxvar:getContextVariable



This is just a sample JAVA service that takes the context variable and creates a top-level element in the response message using the same name and value.

Step 5. Sample service using the context variable



Assigning a Policy to an API

Ensure that the API is in **Edit** mode before you start assigning a policy to the API.

To assign a policy to an API

1. Click **APIs** in the title navigation bar.
2. Select the required API.
3. Click the **Policies** tab.
4. Select the policy stage and the required policy.

   The policy is displayed in the infographic with its properties displayed in properties section.

5. Provide the properties for the selected policy.

6. Click **Save**.

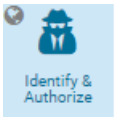
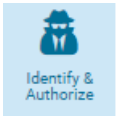
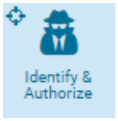
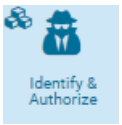
   The policy is assigned to the API.

Viewing API Policy Details

The **Policies** tab on the API details page specifies the set of policies that are applied for that particular API.

The API can have a set of policies that are configured globally through a policy, or directly through a policy template or a scope-level policy.

The global policy in an API details page has each of its policies differentiated using a specific icon from the rest of the policies that are defined at the API-level and scope-level. The icon in the policy indicates the Identify & Authorize policy's enforcement level within an API:

Icon	Description
	Policy is applied from a global policy. This policy is applicable across all resources / methods / operations of all APIs.
	Policy is applied from a policy template or at the API definition. This policy is applicable across all resources / methods / operations of that particular API.
	Policy is applied for the API's scope. This policy is applicable across a set of resources / methods / operations of that particular API.
	Policy is applied through an active package definition. This policy is applicable across all resources / methods / operations of that particular API.

Unlike the policy defined at API-level or scope-level, the policy defined as part of a global policy cannot be edited or deleted through the details page of an API.

To view the policy details of an API

1. Click **APIs** in the title navigation bar.

   A list of all registered APIs appears.

2. Select the required API.

3. Click the **Policies** tab.

   The Infographic view displays policies configured for the API.

   When this API is associated with one or more plans through active packages, a list of the **Identify & Authorize** policies and **Threat Protection** policies that are inherited from the corresponding plans and enforced on the API also appears. The inherited policies are differentiated using the package icon. The **Identify & Authorize** policy, always, has the **Identification Type** set to `API Key`.

4. Click Overview.

   A list of all available policies enforced on the API appears.

Modifying API Policy Details

Ensure that the API is in **Edit** mode before you modify a policy that is assigned to the API.

To modify the policy details of an API

1. Click **APIs** in the title navigation bar.
2. Select the required API.
3. Click the **Policies** tab.
4. Select the policy stage, and the required policy.

   The Infographic view displays policies configured for the API.

5. You can do one of the following:

   - Add more policies to the API. Select the policy stage and add the required policy. Configure the properties for the newly added policy as required.
   - Modify the already configured policy. Select the required policy and modify the properties as required.
   - Delete policies from the API. To remove a policy, click the **x** icon.

6. Click **Save**.

Aliases

An alias in API Gateway holds stage-specific property values that can be shared by multiple policy configurations. Aliases referenced by policy configurations are substituted during runtime. Changing an alias value affects all referencing policies. Aliases are referenced through a name

therefore alias names have to be unique within an API Gateway. The corresponding alias value is substituted in place of an alias name during run-time. Thus the same alias can be referred to in multiple policies and the change in a particular alias would affect all the policy properties. Aliases have optional stage information, in addition to a name and value, which allows to define stage-specific aliases in a multi-stage environment. For details about stage-specific aliases, see *webMethods API Gateway Staging and Promotion*. An alias definition without stage information applies to the API Gateway instance where the alias is defined.

Not all policies support the full set of aliases that are available in API Gateway. Some aliases are applicable only with certain policies and for certain policy parameters. For details, see "Supported Alias and Policy Combinations" on page 467.

You can create six types of alias:

- Simple alias
- Endpoint alias
- HTTP transport security alias
- SOAP message security alias
- webMethods IS Service alias
- XSLT Transformation alias

Creating a Simple Alias

You must have the API Gateway's manage aliases functional privilege assigned to perform this task.

A simple alias holds simple key property values. The name of the alias can be used in the configuration of the properties of a routing policy or an email destination for the Log Invocation, Monitor SLA, Monitor Performance, and Traffic Optimization policies.

To create a simple alias

1. Expand the menu options icon  , in the title bar, and select **Aliases**.
2. Click **Create alias**.
3. In the Basic information section, provide the following information:

Field	Description
Name	Name of the alias.
Type	Select **Simple alias**.
Description	Description of the alias.

4. Click **Technical information** and specify a value in the **Default value** field.

   Note:
   You can specify multiple email addresses, if you are creating an email alias, for example, abc@gmail.com, test@gmail.com, and so on.

5. Specify a stage, if you want the alias to be applicable to a specific stage.

6. Click **Save**.

   Note:
   If you want to configure this alias in the routing policies, you can follow the syntax ${aliasname}. For example, if you want to route it to an endpoint http/mydevenv.com:7000/api, you can create a simple alias with the name mystage and its value being http/mydevenv.com:7000. The endpoint URL can be specified in the properties as ${mystage}/api.

Creating an Endpoint Alias

You must have the API Gateway's manage aliases functional privilege assigned to perform this task.

An endpoint alias stores the endpoint value along with additional properties such as connection timeout, read timeout, whether to pass security headers or not, keystore alias, key alias, and so on.

To create an endpoint alias

1. Expand the menu options icon  , in the title bar, and select **Aliases**.

2. Click **Create alias**.

3. In the Basic information section, provide the following information:

Field	Description
Name	Name of the alias.
Type	Select **Endpoint alias**.
Description	Description of the alias.

4. Click **Technical information** and provide the following information:

Field	Description
Optimization technique	This is applicable only for a SOAP API. Specify the optimization technique for the SOAP request received. Select any one of the following: ■ **None**. This is the default value. API Gateway does not use any optimization method to parse the SOAP requests to the API. ■ **MTOM**. Indicates that API Gateway expects to receive a request with a Message Transmission Optimization Mechanism (MTOM) attachment and forwards the attachment to the native service. ■ **SWA**. Indicates that API Gateway expects to receive a SOAP with Attachment (SWA) request and forwards the attachment to the native service.
Pass WS-Security Headers	Passes the security header.
Endpoint URI	Specify the default URI or components of the URI such as service name.
Connection timeout	Specify the time interval (in seconds) after which a connection attempt times out. The precedence of the Connection Timeout configuration is as follows: a. If you specify a value for the **Connection timeout** field in routing endpoint alias, then the **Connection timeout** value specified in the Endpoint alias section takes precedence over the timeout values defined at the API level and the global level. b. If you specify a value 0 for the **Connection timeout** field in routing endpoint alias, then API Gateway uses the value specified in the **Connection timeout** field in the routing protocol processing step of an API. The Read Timeout value specified at an API level takes precedence over the global configuration. c. If you specify a value 0 or do not specify a value for the **Connection timeout** field in the routing protocol processing step at the API level or specify a value 0 at an alias level, then API Gateway uses the value specified in this `pg.endpoint.connectionTimeout` property.

Field	Description
	d. If you do not specify any value for `pg.endpoint.connectionTimeout`, then API Gateway uses the default value of 30 seconds.
Read timeout	Specify the time interval (in seconds) after which a socket read attempt times out. The precedence of the Read Timeout configuration is as follows: a. If you specify a value for the **Read timeout** field in routing endpoint alias, then the **Read timeout** value specified in the Endpoint alias section takes precedence over the timeout values defined at the API level and the global level. b. If you specify a value 0 for the **Read timeout** field in routing endpoint alias, then API Gateway uses the value specified in the **Read Timeout** field in the routing protocol processing step of an API. The Read Timeout value specified at an API level takes precedence over the global configuration. c. If you specify a value 0 or do not specify a value for the **Read timeout** field in the routing protocol processing step at the API level or specify a value 0 at an alias level, then API Gateway uses the value specified in this `pg.endpoint.readTimeout` property. d. If you do not specify any value for `pg.endpoint.readTimeout`, then API Gateway uses the default value of 30 seconds.
Keystore alias	Specifies the keystore alias configured in API Gateway. This value (along with the value of Client Certificate Alias) is used for performing SSL client authentication. Lists all available keystores. If you have not configured any keystore, the list is empty.
Key alias	Specifies the alias for the private key, which must be stored in the keystore specified by the keystore alias.
Truststore alias	Specifies the alias for the truststore that contains the list of CA certificates that API Gateway uses to validate the trust relationship with the native API. If you do not configure any truststore alias, it implies that API Gateway does not validate the certificates provided by native APIs.
Stage	Specify a stage, if you want the alias to be applicable to a specific stage.

5. Click **Save**.

Creating an HTTP Transport Security Alias

You must have the API Gateway's manage aliases functional privilege assigned to perform this task.

An HTTP Transport security alias contains transport level security information required while accessing the native API. Transport level security that are supported in API Gateway outbound are as follows:

- HTTP Basic authentication
- OAuth2 authentication
- NTLM authentication
- Kerberos authentication
- JWT authentication

To create an HTTP transport secure alias

1. Expand the menu options icon  , in the title bar, and select **Aliases**.
2. Click **Create alias**.
3. In the Basic information section, provide the following information:

Field	Description
Name	Name of the alias.
Type	Select **HTTP transport security alias**.
Description	Description of the alias.

4. Click **Technical information** and provide the following information:

Field	Description
Authentication scheme	Specify the type of authentication you want to use while communicating with the native API. Select one of the following: - **Basic**. Uses basic authentication (user name and password).

Field	Description
	■ **Kerberos**. Uses Kerberos authentication. ■ **NTLM**. Uses NTLM authentication. ■ **OAuth2**. Uses OAuth2 authentication. ■ **JWT**. Uses JWT authentication.
For the Authentication type **Basic**, authenticate using the following:	
Custom credentials	Specifies the values provided in the policy required to access the native API. Provide the following information: ■ **Username**. Specify a username to access the native API. ■ **Password**. Specify a password to access the native API. ■ **Domain**. Specify a domain to access the native API.
Incoming HTTP basic auth credentials	No properties required. Considers the incoming HTTP basic authentication credentials.
For Authentication type **Kerberos**, authenticate using any of the following:	

Custom credentials	Specifies the values provided in the policy required to obtain the Kerberos token to access the native API.
	Provide the following information:
	■ **Client principal**. A valid client LDAP user name.
	■ **Client password**. A valid password of the client LDAP user.
	■ **Service principal**. A valid Service Principal Name (SPN). The specified value is used by the client to obtain a service ticket from the KDC server.
	■ **Service principal nameform**. Specifies the format in which you want to specify the principal name of the service that is registered with the principal database. Select one of the following:
	  ■ **Username**. Represents the principal name as a named user defined in LDAP used for authentication to the KDC.
	  ■ **Hostbased**. Represents the principal name using the service name and the host name, where host name is the host computer.

Field	Description

Delegate incoming credentials	Specifies the values provided in the policy required by the API providers to select whether to delegate the incoming Kerberos token or act as a normal client.
	
	Provide the following information:
	
	■ **Client principal**. A valid client LDAP user name.
	
	■ **Client password**. A valid password of the client LDAP user.
	
	■ **Service principal**. A valid Service Principal Name (SPN). The specified value is used by the client to obtain a service ticket from the KDC server.
	
	■ **Service principal nameform**. Specifies the format in which you want to specify the principal name of the service that is registered with the principal database. Select one of the following:
	
	  ■ **Username**. Represents the principal name as a named user defined in LDAP used for authentication to the KDC.
	
	  ■ **Hostbased**. Represents the principal name using the service name and the host name, where host name is the host computer.

Incoming HTTP basic auth credentials	Specifies the incoming HTTP basic authentication credentials in the transport header of the incoming request for client principal and client password.
	
	Provide the following information:
	
	■ **Service principal**. A valid Service Principal Name (SPN). The specified value is used by the client to obtain a service ticket from the KDC server.
	
	■ **Service principal nameform**. Specifies the format in which you want to specify the principal name of the service that is registered with the principal database. Available values are:
	
	  ■ **Username**. Represents the principal name as a named user defined in LDAP used for authentication to the KDC.
	
	  ■ **Hostbased**. Represents the principal name using the service name and the host name, where host name is the host computer.

Incoming kerberos credentials	No properties required. Considers the incoming kerberos credentials.
For Authentication type **NTLM**, authenticate using any of the following:	

Field	Description
Custom credentials	Specifies the credentials that are required for the NTLM handshake. Provide the following information: ■ **Username**. Name of a consumer who is available in the Integration Server on which API Gateway is running. ■ **Password**. A valid password of the consumer. ■ **Domain**. The domain used by the server to authenticate the consumer.
Incoming HTTP basic auth credentials	No properties required. Considers the incoming HTTP basic authentication credentials.
Transparent	No properties required.
For the Authentication type **OAuth2**, authenticate using any of the following:	
Custom credentials	Specifies the OAuth2 token value that would be added as bearer token in the transport header while accessing the native API.
Incoming OAuth token	Considers the incoming OAuth token to access the native API.
For Authentication type **JWT**, authenticate using any of the following:	
Incoming JWT	Considers the incoming JSON web token to access the native API.

5. Specify a stage, if you want the alias to be applicable to a specific stage.

6. Click **Save**.

Creating a SOAP Message Security Alias

You must have the API Gateway's manage aliases functional privilege assigned to perform this task.

A SOAP message security alias contains message level security information that is requires to access the native API. If the native service is enforced with any WS security policy, API Gateway enforces those policies in the outbound request while accessing the native API using the configuration parameters specified in the alias.

To create SOAP message secure alias

1. Expand the menu options icon  , in the title bar, and select **Aliases**.

2. Click **Create alias**.

3. In the Basic information section, provide the following information:

Field	Description
Name	Name of the alias.
Type	Select **SOAP message secure alias**.
Description	Description of the alias.

4. Click **Technical information** and provide the following information:

Field	Description
Authentication scheme	Specify the type of authentication scheme you want to use to authenticate the client. Available values are: ■ **None**. Does not use any authentication types to authenticate the client. ■ **WSS Username**. Generates a WSS username token and sends it in the soap header to the native API. ■ **Kerberos**. Fetches a Kerberos token and sends it to the native API. ■ **SAML**. Fetches a SAML token and sends it to the native API.
For Authentication scheme **None**. Does not require any properties.	
For Authentication type **WSS Username**, authenticate using any of the following:	
Custom credentials	Specifies the values provided in the policy to be used to obtain the WSS username token to access the native API. Provide the following information: ■ **Username**. Specifies a username used to generate the WSS username token. ■ **Password**. Specifies the password used to generate the WSS username token.
For Authentication type **Kerberos**, authenticate using any of the following:	
Custom Credentials	Uses the Basic authentication credentials coming in the transport header of the incoming request for client principal and client password.

Field	Description

	Provide the following information:
	■ **Client principal**. A valid client LDAP user name.
	■ **Client password**. A valid password of the client LDAP user.
	■ **Service principal**. A valid Service Principal Name (SPN). The specified value is used by the client to obtain a service ticket from the KDC server.
	■ **Service principal nameform**. Specifies the format in which you want to specify the principal name of the service that is registered with the principal database. Select one of the following:
	  ■ **Username**. Represents the principal name as a named user defined in LDAP used for authentication to the KDC.
	  ■ **Hostbased**. Represents the principal name using the service name and the host name, where host name is the host computer.

Delegate incoming credentials	Specifies the values provided in the policy to be used by the API providers to select whether to delegate the incoming Kerberos token or act as a normal client.
	Provide the following information:
	■ **Client principal**. A valid client LDAP user name.
	■ **Client password**. A valid password of the client LDAP user.
	■ **Service principal**. A valid Service Principal Name (SPN). The specified value is used by the client to obtain a service ticket from the KDC server.
	■ **Service principal nameform**. Specifies the format in which you want to specify the principal name of the service that is registered with the principal database. Available values are:
	  ■ **Username**. Represents the principal name as a named user defined in LDAP used for authentication to the KDC.
	  ■ **Hostbased**. Represents the principal name using the service name and the host name, where host name is the host computer.

Incoming HTTP basic auth credentials	Specifies the incoming HTTP basic authentication credentials to access the native API. Provide the following information:

Field	Description
	■ **Service principal nameform**. Specifies the format in which you want to specify the principal name of the service that is registered with the principal database. Select one of the following: ■ **Username**. Represents the principal name as a named user defined in LDAP used for authentication to the KDC. ■ **Hostbased**. Represents the principal name using the service name and the host name, where host name is the host computer.
For Authentication type **SAML**	
SAML issuer configuration	Specifies the SAML issuer configuration that is used by the API Gateway to fetch the SAML token which is then added in the SOAP header and sent to the native API. This field is visible and required only if you have configured a SAML issuer in **Administration** > **Security** > **SAML issuer** section.
Signing configurations	
Keystore alias	Specify the keystore that needs to be used by API Gateway while sending the request to the native API. A keystore is a repository of private key and its corresponding public certificate.
Key alias	The key alias is the private key that is used sign the request sent to the native API.
Encryption configurations	
Truststore alias	Select the truststore to be used by API Gateway when sending the request to the native API. Truststore is a repository that holds all the trusted public certificates.
Certificate alias	Select the certificate from the truststore that is used to encrypt the request that is sent to the native API.
Stage	Specify a stage, if you want the alias to be applicable to a specific stage.

5. Click **Save**.

Creating a webMethods Integration Server Service Alias

You must have the API Gateway's manage aliases functional privilege assigned to perform this task.

A webMethods Integration Server service alias holds the IS service value. The name of the alias can be used to invoke the Invoke webMethods IS policy for request and response processing.

To create a webMethods IS service alias

1. Expand the menu options icon  , in the title bar, and select **User management**.
2. Click **Create alias**.
3. In the Basic information section, provide the following information:

Field	Description
Name	Name of the alias.
Type	Select **webMethods IS Service alias**.
Description	Description of the alias.

4. Click **Technical information** and provide the following information:

Field	Description
Service name	Specify the IS service name. **Note:** The IS service must be available in the Integration Server, to which the aliases are deployed.
Comply to IS Spec **(pub.apigateway.invokeISService .specifications)**	Select . **Comply to IS Spec**, if you want the input and the output parameters to comply to the IS Spec specified.
Stage	Specify a stage, if you want the alias to be applicable to a specific stage.

5. Click **Save**.

Creating an XSLT Transformation Alias

You must have the API Gateway's manage aliases functional privilege assigned to perform this task.

An XSLT transformation alias holds a list of XSLT style sheets. The name of the alias can be used in the XSLT Transformation policies for request and response processing.

To create a transformation alias

1. Expand the menu options icon  , in the title bar, and select **User management**.
2. Click **Create alias**.
3. In the Basic information section, provide the following information:

Field	Description
Name	Name of the alias.
Type	Select **XSLT Transformation alias**.
Description	Description of the alias.

4. Click **Technical information** and browse and select an XSLT style sheet in the **Select transformation file** field.
5. Specify a stage, if you want the alias to be applicable to a specific stage.
6. Click **Save**.

Supported Alias and Policy Combinations

API Gateway provides a set of aliases whose runtime-specific environment variables can be used in configuring the policy routing endpoints, routing rules, endpoint connection properties, and outbound authentication tokens. The types of aliases whose properties you can use for the policy configurations are:

- Simple alias
- Endpoint alias
- HTTP transport security alias
- SOAP message security alias
- webMethods IS Service alias
- XSLT Transformation alias

Not all policies support the full set of aliases that are available in API Gateway. Some aliases are applicable only with certain policies and for certain policy parameters. For example, a *Simple* alias applies to the routing and traffic monitoring policies, whereas an *Endpoint* alias applies only to the routing policies. When you define a Straight Through Routing policy with a simple alias, the alias property is defined using the Endpoint URI field. When you define the same Straight Through Routing policy with an endpoint alias, the alias property is defined using a set of fields - Endpoint

URI, SOAP Optimization Method, HTTP Connection Timeout, Read Timeout, Pass WS-Security Headers, and Keystore Alias.

The following table identifies the policies and policy parameters that each alias type supports:

Simple Alias

Policy Name	Policy Parameter Name
Straight Through Routing	In the **Straight Through Routing** definition: ■ Endpoint URI
Content-based Routing	In the default and custom **Route To** rule definitions: ■ Endpoint URI
Conditional Routing	In the default and custom **Route To** rule definitions: ■ Endpoint URI
Load Balancer Routing	In the **Route To** rule definition: ■ Endpoint URI
Dynamic Routing	In the default and custom **Route To** rule definitions: ■ Endpoint URI
Log Invocation	In the **Email Destination** section: ■ Email Address
Monitor Performance	In the **Email Destination** section: ■ Email Address
Monitor SLA	In the **Email Destination** section: ■ Email Address
Traffic Optimization	In the **Email Destination** section: ■ Email Address

Endpoint Alias

Policy Name	Policy Parameter Name
Straight Through Routing	In the **Straight Through Routing** definition: ■ Endpoint URI ■ SOAP Optimization Method (Applicable only for SOAP APIs)

Policy Name	Policy Parameter Name
	■ HTTP Connection Timeout ■ Read Timeout ■ Pass WS-Security Headers (Applicable only for SOAP APIs) ■ Keystore Alias ■ Key Alias
Content-based Routing	In the default and custom **Route To** rule definitions: ■ Endpoint URI ■ SOAP Optimization Method (Applicable only for SOAP APIs) ■ HTTP Connection Timeout ■ Read Timeout ■ Pass WS-Security Headers (Applicable only for SOAP APIs) ■ Keystore Alias ■ Key Alias
Conditional Routing	In the default and custom **Route To** rule definitions: ■ Endpoint URI ■ SOAP Optimization Method (Applicable only for SOAP APIs) ■ HTTP Connection Timeout ■ Read Timeout ■ Pass WS-Security Headers (Applicable only for SOAP APIs) ■ Keystore Alias ■ Key Alias
Load Balancer Routing	In the **Route To** rule definition: ■ Endpoint URI ■ SOAP Optimization Method (Applicable only for SOAP APIs) ■ HTTP Connection Timeout ■ Read Timeout ■ Pass WS-Security Headers (Applicable only for SOAP APIs) ■ Keystore Alias

Policy Name	Policy Parameter Name
	■ Key Alias
Dynamic Routing	In the default and custom **Route To** rule definitions: ■ Endpoint URI ■ SOAP Optimization Method (Applicable only for SOAP APIs) ■ HTTP Connection Timeout ■ Read Timeout ■ Pass WS-Security Headers (Applicable only for SOAP APIs) ■ Keystore Alias ■ Key Alias

HTTP Transport Security Alias

Policy Name	Policy Parameter Name
Outbound Auth - Transport	In the **Authentication** scheme: ■ Alias

SOAP Message Security Alias (Applicable only for SOAP APIs)

Policy Name	Policy Parameter Name
Outbound Auth - Message	In the **Authentication** scheme: ■ Alias

webMethods IS Service Alias

Policy Name	Policy Parameter Name
Invoke webMethods IS (Request Processing)	webMethods IS Service Alias
Invoke webMethods IS (Response Processing)	webMethods IS Service Alias

XSLT Transformation Alias

Policy Name	Policy Parameter Name
Request Transformation (Request Processing)	Transformation Configuration ■ Payload Transformation ■ XSLT Transformation alias
Response Transformation (Response Processing)	Transformation Configuration ■ Payload Transformation ■ XSLT Transformation alias

Global Policies

Important:
API Gateway's Standard Edition License does not support the functionality of Global Policies. You can create and manage global policies only using the Advanced Edition License.

Global policies are a set of policies that are associated globally to all APIs or the selected set of APIs. Global policies are supported for SOAP and REST APIs but not supported for GraphQL API.

By associating policies globally to all APIs or the selected set of APIs, the administrator can ensure that a set of policies is applied to the selected APIs by default. The administrator can, for example, define a global policy that attaches a WS-Security (WSS) authentication to all SOAP API endpoints within a specific IP range. In this case, any client request from the specific IP range automatically inherits the security configuration defined in the global policy for SOAP APIs.

API Gateway provides a system global policy, **Transaction logging**, which is bundled with the product. By default, the policy is in the **Inactive** state. The transaction logging policy has standard filters and log invocation policy, that log request or response payloads to a specified destination. You can modify this policy to include additional filters or modify the policy properties, but you cannot delete this policy. You can activate this policy in the **Polices** > **Global policies** page or through the global policy details page. Activating the policy enforces it on all APIs in API Gateway based on the configured filters, and logs transactions across all the APIs. If you have multiple log invocation policies assigned to an API, the policies are compiled into a single policy and one transaction log is created per destination.

Global Policy Matrix

This table lists the stage-specific policies that can be configured as global policy for different types of APIs at the global level.

Note:
The **Policy configuration** page displays only the policies that are common to one or more API types selected in the global policy filter.

Stages	Policies
Transport	■ Enable bulkhead - This policy can be enforced to configure the maximum number of concurrent requests that the APIs can process. ■ Enable HTTP/HTTPS - This policy can be enforced for all types of API. But the SOAP versions 1.1 and 1.2 are applicable only for SOAP-based APIs. The SOAP 1.1 and SOAP 1.2 sub types are not available in UI when the REST and ODATA APIs are selected. **Note:** Software AG recommends to create a separate policy for each API type. ■ Set Media Type - This policy is applicable only for a REST request and the policy name is not listed in **Policy configuration** page when the SOAP and ODATA APIs are selected. ■ Enable JMS/AMQP - This policy is applicable only for SOAP APIs and the policy name is not listed in **Policy configuration** page when the REST and ODATA APIs are selected.
Identity & Access	■ Authorize User, Identify & Authorize - These policies can be enforced to any API Type. ■ Inbound Auth - Message - This policy is applicable only for SOAP-based APIs and the policy name is not listed in **Policy configuration** page when the REST and ODATA APIs are selected.
Request Processing	■ Invoke webMethods IS, Validate API Specification, Data Masking - These policies can be enforced to any API Type. ■ Request Transformation - This policy is applicable only for SOAP and REST APIs. and not for ODATA services. When all three API types are selected, Request Transformation policy cannot be applied at the global level.
Routing	■ Custom HTTP Header, Outbound Auth - Transport, Outbound Auth - Message. The Routing stage policies can be applied at a global level for all types of API.
Traffic Monitoring	■ Log Invocation, Monitor Performance, Monitor SLA, Traffic Optimization, and Service Result Cache. The Traffic Monitoring stage policies can be applied at a global level for all types of API.
Response Processing	■ Invoke webMethods IS, Validate API Specification, Data Masking - These policies can be enforced to any API Type. ■ Response Transformation - This policy can be enforced only for SOAP and REST APIs and the policy name is not listed in **Policy configuration** page when ODATA API type is selected. ■ CORS - This policy can be enforced only for REST and ODATA APIs and the policy name is not listed in **Policy configuration** page when SOAP-based API is selected.

Stages	Policies
Error handling	Conditional Error Processing and Data Masking. The Error handling stage policies can be applied at a global level for all types of API.

Creating a Global Policy

You must have the API Gateway's manage global policies functional privilege assigned.

To create a global policy you must perform the following high-level steps:

1. **Create a new global policy**: During this step, you specify the basic details of the global policy.
2. **Optionally refine the scope of the policy**: During this step, you can specify additional criteria to narrow the set of APIs to which the global policy applies.
3. **Configure the policies**: During this step, you associate one or more policies, and assign values to each of the associated policy's properties.
4. **Activate the policy**: During this step, you put the new global policy into effect.

To create a global policy

1. Click **Policies** in the title navigation bar.
2. Click the **Global policies** tab.

   A list of all available global policies appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of policies you want to display in a page.
3. In the **Policies** page, click the **Create global policy** button.

   If you do not see the **Create global policy** button, it is probably because you do not have the API Gateway Administrator role to create a global policy in API Gateway.

   This opens the Create global policy page with the default **Policy details** tab.
4. In the Basic information section, provide the required information as follows:

Field	Description
Name	Name of the global policy.
Description	Description of the global policy.

   You can save the global policy by clicking **Save** at this stage and add the filters and policy configuration at a later time.
5. Click **Continue to filters >**.

Alternatively, you can click **Filters** in the left navigation panel.

6. To filter APIs by API type, select one or more API types.

   Available API types are **REST**, **SOAP**, and **OData**. The global policy would apply to the APIs specified by the filter.

7. *This is applicable to REST APIs.* To filter APIs by HTTP methods, select one or more HTTP methods.

   Available HTTP methods are **GET**, **POST**, **PUT**, **DELETE**, **PATCH**, and **HEAD**. The global policy would apply to the APIs that have the methods specified by the filter.

   For details about the HTTP methods, see " Refining the Scope of a Global Policy" on page 477.

8. To filter APIs by attributes:

   a. Select an attribute

      Available attributes are **API name**, **API description**, **API version**, **API tag**, **Resource/Operation tag**, **Method tag**.

   b. Select a comparison operator.

   c. Specify the match string.

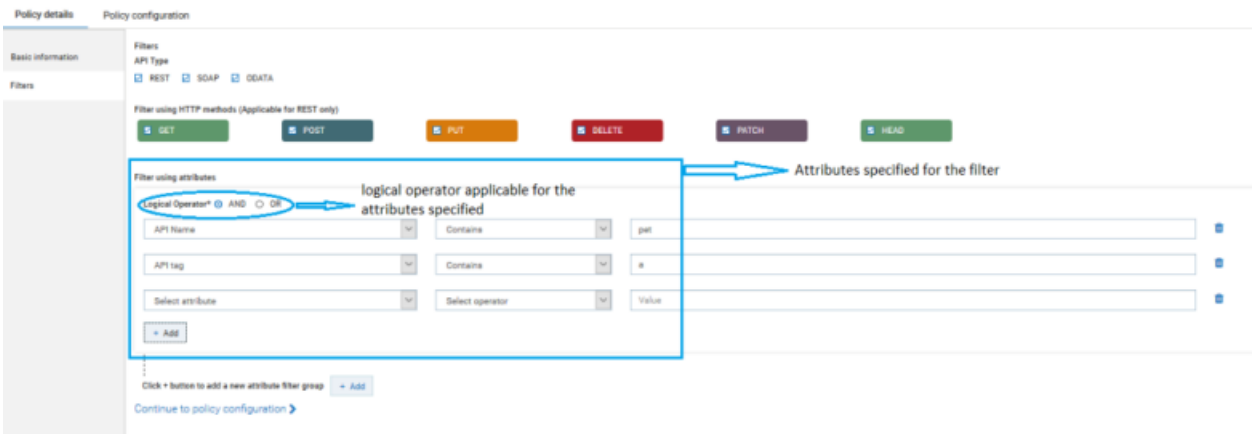
   d. Click **+ Add**.

      You can add multiple criteria by clicking the **+ Add** button and repeating the above steps.

   e. Select the logical conjunction (**AND**) or disjunction (**OR**) operation to apply when multiple criteria are specified for the global policy. The default value is **AND**.

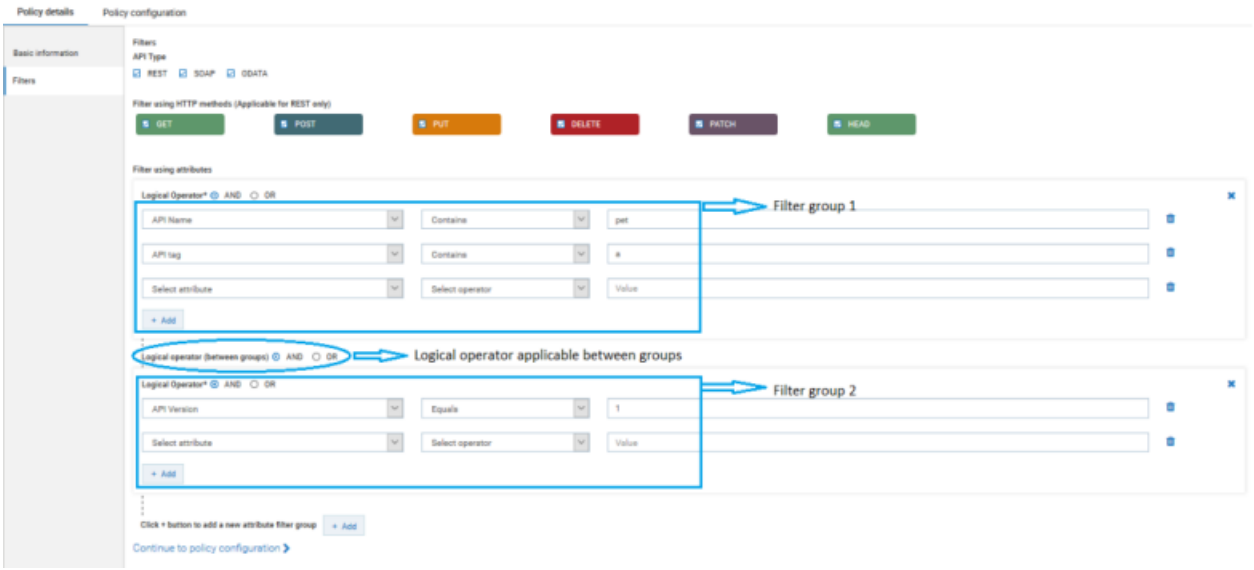
   The global policy would apply to the APIs that match the attributes specified in the filter. For details about attributes, see " Refining the Scope of a Global Policy" on page 477.

   Example: To apply the global policy to APIs that match the criteria API name that contains `pet` and API tag that contains `a`, you can configure a filter as follows:



9. To add multiple attribute filter groups, as required, click the **+Add** button. and specify the logical conjunction (**AND**) or disjunction (**OR**) operation to apply between filter groups. The global policy would apply to the APIs that match the filter groups specified in the filter.

   Example: To apply the global policy to APIs that match criteria API name that contains `pet` and API tag that contains `a` in filter group 1 and API version that equals `1` in filter group 2, you can configure a filter as follows:



   You can save the global policy by clicking **Save** at this stage and configure policies at a later time.

10. Click **Continue to policy configuration >**.

    Alternatively, you can click the **Policy configration** tab.

11. In the Policy configuration section, you can select the policies and configure the properties for each policy that you want API Gateway to enforce when it applies this global policy.

    For details, see " Configuring Properties for a Global Policy" on page 481.

12. Click **Save**.

    The global policy is created and displays the policy details page.

You can now activate the global policy. For details, see " Activating a Global Policy" on page 484.

Modifying the Scope of a Global Policy

You must have the API Gateway's manage global policies functional privilege assigned.

Scope refers to the set of properties that determine a selected set of APIs for the enforcement of the policy. For a global policy, scope is determined by the policy's property **API type** in the **Policy details** tab.

API Type	Description
REST	Global policy is applied on all REST APIs in API Gateway.
SOAP	Global policy is applied on all SOAP APIs in API Gateway.
ODATA	Global policy is applied on all OData APIs in API Gateway.

To modify the scope of a global policy

1. Click **Policies** in the title navigation bar.
2. Click the **Global policies** tab.

   A list of all available global policies appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of policies you want to display in a page.
3. Select the required policy.

   The global policy details page appears.
4. Click **Edit**.

   If you do not see the **Edit** button, it is probably because you do not have the API Gateway Administrator role to modify the scope of a global policy in API Gateway.
5. In the Filters section, specify the following:

   a. In the **API type** section, select the API types (**REST**, **SOAP**, **ODATA**, or all) to which you want to apply the policy.

   b. *Optional*. In the Filter using attributes section, specify additional selection criteria to narrow the set of APIs to which this policy will be applied. For details, see " Refining the Scope of a Global Policy" on page 477.

6. Click **Save**.

Refining the Scope of a Global Policy

You must have the API Gateway's manage global policies functional privilege assigned.

If you want to further restrict the set of APIs to which the global policy is applied, you can specify additional selection criteria in the Filter section of the API details page. Using the Filter section, you can filter APIs by Name, Description, Version attributes, HTTP Methods (applicable only for REST APIs), API tag (applicable for all selected API types), Resource/Operation tag (applicable for REST and SOAP APIs) and Method tag (applicable for a REST API). For details about the API types and their components for which you can add a tag, see "Adding Tags to an API" on page 56. If you specify no filter criteria, API Gateway applies the global policy to all the selected APIs.

If the specified attribute does not apply for the selected API type, it is not evaluated for that API type alone. For example, if you specify Resource/Operation tag = `secure` and select all API types, REST, SOAP, and ODATA, then while evaluating the condition for each API, the expression evaluates only for SOAP and REST API and does not evaluate the filter for OData API.

Filtering by Name, Description, Version and Tag attributes

You can filter APIs based on their Name, Description, Version, API tag, Resource/Operation tag and Method tag attributes using any of the following comparison operators:

Comparison Operators	Description
Equals	Selects APIs whose Name, Description, Version or Tag value matches a given string of characters. For example, use this operator to apply a policy only to REST APIs with the Name or Description value `4G Mobile Store`.
Not Equals	Selects APIs whose Name, Description, Version or Tag value does not match a given string of characters. For example, use this operator to apply a policy only to all REST APIs except those with the Name, Description, or Tag value `Mobile`.
Contains	Selects APIs whose Name, Description or Tag value includes a given string of characters anywhere within the attribute's value. For example, use this operator to apply a policy to REST APIs that had the word `Mobile` anywhere in their Name, Description, or Tag attribute.
Starts with	Selects APIs whose Name, Description, or Tag value begins with a given string. For example, use this operator to apply a policy only to REST APIs whose Name, Description, or Tag begins with the characters `4G`.
Ends with	Selects APIs whose Name, Description, or Tag value ends with a given string. For example, use this operator to apply a policy only

Comparison Operators	Description
	to REST APIs whose Name, Description, or Tag value ends with the characters `Store`.

When specifying match strings for the comparison operators described above, keep the following points in mind:

- Match strings *are not case-sensitive*. If you define a filter for names that start with `ABC` it select names starting with `abc` and `Abc`.
- Wildcard characters are not supported. That is, you cannot use characters such as * or % to represent *any sequence of characters*. These characters, if present in the match string, are simply treated as literal characters that are to be matched.

Filtering by HTTP Methods (Applicable only for REST APIs)

- You can optionally restrict a policy to specific HTTP methods of the REST APIs by specifying the options GET, POST, PUT, DELETE, PATCH, and HEAD.

HTTP Methods	Description
GET	Policy applies only to HTTP GET requests for any resource in the API. For example, use this option to apply a policy to resources of a REST API during an incoming GET request.
POST	Policy applies only to HTTP POST requests for any resource in the API. For example, use this option to apply a policy to resources of a REST API during an incoming POST request.
PUT	Policy applies only to HTTP PUT requests for any resource in the API. For example, use this option to apply a policy to resources of a REST API during an incoming PUT request.
DELETE	Policy applies only to HTTP DELETE requests for any resource in the API. For example, use this option to apply a policy to resources of a REST API during an incoming DELETE request.
PATCH	Policy applies only to HTTP PATCH requests for any resource in the API. For example, use this option to apply a policy to resources of a REST API during an incoming PATCH request.
HEAD	Policy applies only to HTTP HEAD requests for any resource in the API. For example, use this option to apply a policy to resources of a REST API during an incoming HEAD request.

To refine the scope of a global policy

1. Click **Policies** in the title navigation bar.

2. Click the **Global policies** tab.

   A list of all available global policies appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of policies you want to display in a page.

3. Select the required policy.

   The global policy details page appears.

4. Click **Edit**.

   If you do not see the **Edit** button, it is probably because you do not have the API Gateway Administrator role to refine the scope of a global policy in API Gateway.

5. Click Filters.

6. To filter by API types, select the API types by which you want to filter APIs.

7. *Applicable only for REST APIs.* To filter by HTTP methods, in the Filter using HTTP methods section, select the HTTP methods by which you want to filter APIs with appropriate incoming requests.

8. To filter by Name, Description, Version, or Tags perform the following steps in the Filter using attributes section:

   a. Select an attribute to filter the APIs to which you want to apply the global policy.

      Available attributes: **API name**, **API description**, **API version**, **API tag**, **Resource/Operation tag**, **Method tag**.

   b. Select the comparison operator.

   c. Specify the match string in the third field.

   d. To specify additional criteria, click the **Add** button and repeat the above steps.

   e. Select the logical conjunction (**AND**) or disjunction (**OR**) operation to apply when multiple criteria are specified for the global policy. The default value is AND.

   You can add multiple attribute filter groups by clicking the **+Add** button. You can also specify the logical conjunction (**AND**) or disjunction (**OR**) operation to apply between filter groups.

9. Click **Save** to save the updated policy.

Associating Policies to a Global Policy

You must have the API Gateway's manage global policies functional privilege assigned.

The **Policy Configuration** tab on the Global Policy details page specifies the policy stages and the list of policies (applicable for each stage) that you want API Gateway to execute when it enforces the global policy.

When modifying the list of policies for a global policy, API Gateway validates the policies to ensure that there are no policy conflicts.

To modify the list of policies of a global policy

1. Click **Policies** in the title navigation bar.
2. Click the **Global Policies** tab.

   A list of all available global policies appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of policies you want to display in a page.
3. Select the required policy.

   The Global Policy details page appears.
4. Click **Edit**.

   If you do not see the **Edit** button, it is probably because you do not have the API Gateway Administrator role to modify the list of policies of a global policy in API Gateway.
5. Select the policy's **Policy Configuration** tab.

   The policy information is provided in the following sections:

   - Policy catalog - Transport, Identify and Access, Request Processing, Routing, Traffic Monitoring, Response Processing, Error Handling
   - Infographic - List of applied policies
   - Policy properties - Collection of policy properties
6. In the Policy catalog section, click the chevron to expand the required policy stage.

   This displays a list of policies that are classified under the particular stage.
7. In the expanded list of policies, select the policies that you want API Gateway to execute when it applies this global policy. To select a policy, click the **Add** (+) icon next to the policy name. The selected policies are displayed in the Infographic section.

   When you select the policies for the global policy, keep the following points in mind:

   - The policies shown in the Policy catalog section are determined by the API types that you have specified on the Filters section of the Global Policy Details page.

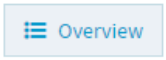
     If you do not see a policy that you need, that policy is probably not compatible with the API type that you selected in the Filters section.

- If necessary, you can click the **Policy Details** tab and change your API type selection.

Use the **x** icon in any individual policy to remove that particular policy from the Infographic section.

8. To configure the properties for any new policies that you might have added to the Infographic section in the preceding steps or to make any necessary updates to the properties for existing policies in the global policy, see " Configuring Properties for a Global Policy" on page 481.

9. Click **Save**.

10. Click 

to view the complete list of policies in the updated policy.

    The **Overview** button is located in the lower right-corner of the Infographic section.

    To exit the overview, click the **Close** icon.

Configuring Properties for a Global Policy

You must have the API Gateway's manage global policies functional privilege assigned.

The **Policy Configuration** tab on the Global Policy details page specifies the list of policies that are applicable for each policy stage in the Policy catalog section. Each policy in the Infographic section has properties that you must set to configure the policy's enforcement behavior.

To configure the properties for a global policy

1. Click **Policies** in the title navigation bar.

2. Click the **Global Policies** tab.

    A list of all available global policies appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of policies you want to display in a page.

3. Select the required policy.

    The Global Policy details page appears.

4. Click **Edit**.

    If you do not see the **Edit** button, it is probably because you do not have the API Gateway Administrator role to configure the properties of a global policy in API Gateway.

5. Select the policy's **Policy Configuration** tab.

    The policy information is provided in the following sections:

- Policy catalog - Transport, Identify and Access, Request Processing, Routing, Traffic Monitoring, Response Processing, Error Handling
- Infographic - List of applied policies
- Policy properties - Collection of policy properties

6. In the Policy catalog section, click the chevron to expand the required policy stage.

   This displays a list of policies that are classified under the particular stage.

7. In the Infographic section, do the following for each policy in the list:

   a. Select the policy whose properties you want to examine or set.

   b. In the Policy properties section, set the values for the policy's properties as necessary.

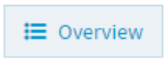
   Note:
   Required properties are marked with an asterisk.

8. Click **Open in full-screen** to view the policy's properties in full screen mode.

   The **Open in full-screen** link is located in the upper right-hand corner of the **Policy Configuration** tab. Set the properties of the displayed policy, and then click **OK**.

   To exit out of full screen mode, click the **Minimize** icon.

9. Click **Save**.

10. Click 

to view the complete list of policies in the updated policy.

    The **Overview** button is located in the lower right-corner of the Infographic section.

    To exit the overview, click the **Close** icon.

Viewing List of Global Policies and Policy Details

The **Global Policies** tab displays a list of all globally available policies in API Gateway. Global policies are listed alphabetically by name.

In addition to viewing the list of policies, you can also examine the details of a policy, create a copy of the template, activate, and delete a global policy in the **Global Policies** tab.

To view the policy list and properties of a global policy

1. Click **Policies** in the title navigation bar.

2. Click the **Global Policies** tab.

A list of all available global policies appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of policies you want to display in a page.

The **Global Policies** tab provides the following information about each policy:

Column	Description
Name	Name of the global policy.
Description	The description for the global policy.

You can also perform the following operations on a global policy:

- Activate a policy to begin enforcing runtime behaviors.
- Deactivate a policy to suspend enforcement of runtime behaviors.
- Create a new copy of the policy.
- Delete a policy to remove it from API Gateway.

3. Select the required policy whose details you want to examine.

The Global Policy details page appears. The policy details are displayed in the following tabs:

- **Policy Details**: This tab contains a summary of basic information such as name, description, scope of the policy as to when the policy will apply, applicable APIs, and other information.
- **Policy Configuration**: This tab contains the policy stages, applied policies, as well as the configuration details of individual policies.

Modifying Global Policy Details

You must have the API Gateway's manage global policies functional privilege assigned.

You use the Global Policy details page to examine and modify the properties of a policy.

When modifying the details of a global policy, keep the following points in mind:

- You will not be allowed to save the policy unless all of its properties have been set.
- On successful modification of the policy details for an active global policy, the policy changes apply with immediate effect in all the active APIs that are applicable for this global policy.
- You will not be allowed to remove an individual policy (for example, Identify & Authorize) from the active global policy, if the global policy is already applied to an active API, and if the Identify & Authorize is a dependent policy for another policy (for example, Traffic Optimization) that is applied for the API.
- If modification of the policy details for an active global policy fails, API Gateway issues an error message with details of the incompatible or conflicting policies.

To modify the properties of a global policy

1. Click **Policies** in the title navigation bar.
2. Click the **Global Policies** tab.

   A list of all available global policies appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of policies you want to display in a page.
3. Select the required policy.

   The Global Policy details page appears. The policy details are displayed in the following tabs:

   - **Policy Details**: This tab contains a summary of basic information such as name, description, scope of the policy as to when the policy will apply, applicable APIs, and other information.
   - **Policy Configuration**: This tab contains the policy stages, applied policies, as well as the configuration details of individual policies.
4. Click **Edit**.

   If you do not see the **Edit** button, it is probably because you do not have the API Gateway Administrator role to modify the properties of a global policy in API Gateway.
5. On the **Policy Details** tab, modify the basic properties, selection criteria, and the applicable APIs as necessary.
6. On the **Policy Configuration** tab, modify the policy list and the policy's configuration properties as necessary.
7. When you have completed the required modifications in the Global Policy details page, click **Save** to save the updated policy.
8. Click **Overview** to view the complete list of policies in the updated policy.

   The **Overview** button is located in the lower right-corner of the Infographic section.

   To exit the overview, click the **Close** icon.

Activating a Global Policy

You must have the API Gateway's activate global policies functional privilege assigned.

Global policies are not enforced until they are activated.

When you activate a global policy, be aware that:

- When a global policy becomes active, API Gateway begins enforcing it immediately in all the applicable APIs that are currently in the **Active** state. You can suspend enforcement of a policy by switching it to the **Inactive** state as described in " Deactivating a Global Policy" on page 485.
- Activation of a global policy fails if there is a conflict in the effective policy validation in at least one of the active APIs that are applicable for this policy. API Gateway reports the conflict, and the global policy can only be activated when the conflict is resolved.

To determine whether a global policy is active or inactive, examine the policy's **Active** indicator on the **Policies** > **Global Policies** tab. The icon in the **Active** column indicates the policy's activation state as follows:

Icon	Description
	Policy is active.
	Policy is inactive.

The activation state of a policy is also reported in the Global Policy Details page.

To activate a global policy

1. Click **Policies** in the title navigation bar.
2. Click the **Global Policies** tab.

   A list of all available global policies appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of policies you want to display in a page.
3. Select the required policy.

   The Global Policy details page appears.
4. Click **Activate**.

   If you do not see the **Activate** button, it is probably because you do not have the API Gateway Administrator role to activate a global policy, or the policy is already in an **Active** state in API Gateway.

Deactivating a Global Policy

You must have the API Gateway's activate global policies functional privilege assigned.

Deactivating a global policy causes API Gateway to suppress enforcement of the policy.

You usually deactivate a policy to suspend enforcement of a particular policy (temporarily or permanently).

To deactivate a policy, you change the policy to the **Inactive** state. At a later time, you can begin enforcing a global policy by switching it to the **Active** state as described in " Activating a Global Policy" on page 484.

When you deactivate a global policy, be aware that:

- Deactivation of a global policy fails if there is a conflict in the effective policy validation in at least one of the active APIs that are applicable for this policy. API Gateway reports the conflict, and the global policy can only be activated when the conflict is resolved.

To determine whether a global policy is active or inactive, examine the policy's **Active** indicator on the **Policies** > **Global Policies** tab. The icon in the **Active** column indicates the policy's activation state as follows:

Icon	Description
	Policy is active.
	Policy is inactive.

The deactivation state of a policy is also reported in the Global Policy Details page.

To deactivate a global policy

1. Click **Policies** in the title navigation bar.
2. Click the **Global Policies** tab.

   A list of all available global policies appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of policies you want to display in a page.
3. Select the required policy.

   The Global Policy details page appears.
4. Click **Deactivate**.

   If you do not see the **Deactivate** button, it is probably because you do not have the API Gateway Administrator role to deactivate a global policy, or the policy is already in an **Inactive** state in API Gateway.

Deleting a Global Policy

You must have the API Gateway's manage global policies functional privilege assigned.

You delete a global policy to remove it from API Gateway permanently.

To delete a global policy, the following conditions must be satisfied:

- The policy must not be in-progress.
- The policy must be inactive.

To delete a global policy

1. Click **Policies** in the title navigation bar.
2. Click the **Global Policies** tab.

   A list of all available global policies appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of policies you want to display in a page.
3. Click the **Delete** ( ) icon for the required policy.

   If you do not see the **Delete** button, it is probably because you do not have the API Gateway Administrator role to delete a global policy, or the policy is in an **Active** state in API Gateway.
4. Click **Yes** in the confirmation dialog.

Copying a Global Policy

You must have the API Gateway's manage global policies functional privilege assigned.

A global policy can become quite complex, especially if it includes many policies. Instead of creating a new policy from scratch, it is sometimes easier to copy an existing policy that is similar to the one you need and edit the copy.

When you create a copy of a global policy, be aware that:

- When API Gateway creates a copy of a policy, the new copy of the policy is identical to the original one.
- Like all new policies, the copied policy is marked as **Inactive**.
- There is no expressed relationship between the copy and the original policy (that is, API Gateway does not establish any type of association between the two policies).

In general, a copied policy is no different from a policy that you create from scratch.

To copy a global policy

1. Click **Policies** in the title navigation bar.
2. Click the **Global Policies** tab.

   A list of all available global policies appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of policies you want to display in a page.

3. Click the **Copy** icon for the required policy.

   If you do not see the **Copy** button, it is probably because you do not have the API Gateway Administrator role to create the copy of a global policy in API Gateway.

4. In the **Copy of Global Policy** dialog box, provide the required information for each of the displayed data fields:

Field	Description
Name	Name of the global policy. API Gateway automatically adds the name of the existing global policy to the **Name** field. You can change the name of the policy to suit your needs. But you cannot leave this field empty.
Description	The description for the global policy.

5. Click **Copy** to save the new policy.

6. Modify the new policy as necessary and then save it.

Activate the new policy when you are ready to put it into effect.

Exporting Global Policies

You must have the API Gateway's export assets functional privilege assigned.

Note:
For more information about exporting and importing global policies,

To export the global policies

1. Click **Policies** in the title navigation bar.

2. Select **Global Policies**.

3. Click  to export a single policy.

   Alternatively, you can select multiple APIs to be exported simultaneously by clicking the checkboxes adjacent to the names of the API.

   Click  and select **Export** from the drop-down list.

   The browser prompts you to either open or save the export archive.

4. Select the appropriate option and click **OK**.

Scope-level Policies

You can define policies at the API-level or scope-level for an API. API-level policies are processed for all incoming requests to the API. Scope-level policies are processed only for incoming requests that apply to a specific scope in the API. Any policy you specify at the API-level is overridden by the policy defined at the scope-level if the policies are the same. In contrast, the API-level policies will not affect the scope-level policies. But if there are policies applied at the global-level (through a global policy) for the API, then those policies will override every other policy configured at the API-level.

The scope-level policies for an API provide a granular enforcement of policies at the resource-level, method-level, or both for the REST API, or at the operation-level for the SOAP API.

Note:
Scope-level policies are not supported for OData APIs.

An API can have zero or more scope-level policies. When you define the scope-level policies for an API, keep the following points in minds:

- For a policy (for example, Identify & Authorize) that can appear only once in an API, if the same policy is already applied through the API details page, API Gateway prompts you with a warning message that the scope-level policy takes precedence over the API-level policy, and is enforced on the API at run-time.
- For a policy (for example, Monitor SLA) that can appear multiple times in an API, if the same policy is already applied to the API through a global policy, API Gateway prompts you with a warning message that the global policy takes precedence over the scope-level policy, and is enforced on the API at run-time.
- If a resource or method or operation has the same policy (for example, Require HTTP / HTTPs) applied through different scopes, API Gateway prompts you with an error message and sets the focus to the conflicting policies. You must remove the required policy from the individual scope(s) to resolve the conflicts.

API Gateway supports scope-level policies only for the following stages:

- Identify and Access: All policies in this stage are supported.
- Request Processing: Only Data Masking policy in this stage is supported.
- Traffic Monitoring: All policies in this stage are supported.
- Response Processing: Only Data Masking policy in this stage is supported.
- Error Handling: Only Data Masking policy in this stage is supported.

For information on the usage scenarios of policies configured for the scopes of an API, see "Example: Usage Scenarios of API Scopes" on page 498.

API Scopes

API definitions can be complex and span across multiple REST resources and methods, or SOAP operations for an API. To reduce the complexity of an API definition, you can define scopes and impose a set of policies on each scope to suit your requirements.

A scope represents a logical grouping of REST resources, methods, or both, and SOAP operations in an API. You can then enforce a specific set of policies on each individual scope in the API.

An API can have a set of declared scopes. The available scopes for an API are listed in the **Scopes** tab of the API details page.

Creating an API Scope

Scopes enable you to group a set of REST resources, methods, or both, and SOAP operations for an API.

A scope consists of a name, description, and zero or more resources, methods, or operations. An API can have zero or more scopes.

You can define a set of policies and configure its properties for each individual scope. These policies apply to each of the resources, methods, or operations that are associated to the scope.

Instructions throughout the remainder of this guide use the term *scope-level policy* when referring to a set of policies configured for an individual scope of the API.

Note:
Ensure that you have a unique set of resources, methods, or operations in every scope in the API.

To create a scope

1. Click **APIs** in the title navigation bar.

   A list of APIs available in API Gateway appears.

2. Click the name of the required API.

   This opens the API details page.

3. Click **Edit**.

   If the API is active, API Gateway prompts you to deactivate it.

4. Click the **Scopes** tab.

   This displays a list of scopes available in the API.

5. In the List of scopes section, click **Add scope**.

6. In the Basic information section, provide the required information for each data field that appears:

Field	Description
Name	Name of the scope. A scope name must be unique within an API. **Note:** API Gateway automatically adds the name `New Scope` to the **Name** field. You can change the name of the scope to suit your needs. But you cannot leave this field empty.
Description	Description of the scope.

7. *Applicable only for REST APIs*. In the Resources and methods section, select the resources, methods, or both, you want to associate to this scope.

   When selecting a resource or method for the scope definition, you can select whether you want some or all of the methods within that resource to be selected as well.

8. Applicable only for SOAP APIs. In the Operations section, select the operations you want to associate to this scope.

9. Click **Save**.

   The scope is created and listed in the List of scopes section.

Post-requisites:

- Activate the API when you are ready to put it into effect.
- To apply and configure policies for this API scope, see " Creating a Scope-level Policy" on page 494.

Viewing List of API Scopes and Scope Details

The **Scopes** tab in the API details page displays a list of all available scopes in the API.

In addition to viewing the list of scopes, you can also examine and modify the details of a scope, and delete a scope in the **Scopes** tab.

To view the scope list and properties of a scope

1. Click **APIs** in the title navigation bar.

   A list of APIs available in API Gateway appears.

2. Click the name of the required API.

This opens the API details page.

3. Click the **Scopes** tab.

   This displays a list of scopes available in the API.

4. In the List of scopes section, click the name of the scope you want to examine.

   This opens the details of the scope. The scope details appear in the following sections:

   - Basic information: This section contains a summary of basic information such as name and description of the scope.
   - Resources and methods: Applicable only for REST APIs. This section contains a collection of REST resources, methods, or both, that are associated to the scope.
   - Operations: Applicable only for SOAP APIs. This section contains a collection of SOAP operations that are associated to the scope.

Modifying API Scope Details

You use the **Scopes** tab in the API details page to examine and modify the details of a scope.

To modify the properties of a scope

1. Click **APIs** in the title navigation bar.

   A list of APIs available in API Gateway appears.

2. Click the name of the required API.

   This opens the API details page.

3. Click **Edit**.

   If the API is active, API Gateway displays a warning message to let you know that the API is active.

4. Click the **Scopes** tab.

   This displays a list of scopes available in the API.

5. In the List of scopes section, click the name of the scope you want to modify.

   This opens the details of the scope. The scope details appears in the following sections:

   - Basic information: This section contains a summary of basic information such as name and description of the scope.
   - Resources and methods: Applicable only for REST APIs. This section contains a collection of REST resources, methods, or both, that are associated to the scope.

- Operations: Applicable only for SOAP APIs. This section contains a collection of SOAP operations that are associated to the scope.

6. Modify the basic properties, applicable resources, methods, or operations of the scope.
7. Click **Save**.

   Activate the API, if it is not active, to put it into effect.

Deleting an API Scope

You delete a scope to remove it from the API permanently.

When a scope is deleted from the API definition, API Gateway deletes the existing associations between the scope and the collection of resources, methods, or operations in the API. But, the collection of resources, methods, or operations continue to exist in the API.

To delete a scope

1. Click **APIs** in the title navigation bar.

   A list of all registered APIs appears.
2. Click the name of the required API.

   This opens the API details page.
3. Click **Edit**.

   If the API is active, API Gateway displays a warning message to let you know that the API is active.
4. Click the **Scopes** tab.

   This tab displays a list of scopes available with the API.
5. In the List of scopes section, locate the name of the scope you want to delete.
6. Click the **Delete** ( ) icon next to the scope name.
7. Click **Yes** in the confirmation dialog.

   The scope is removed from the List of scopes section.
8. Click **Save** to save the updated API.

   Activate the API, if it is not active, to put it into effect.

Creating a Scope-level Policy

You create a policy for the API scope, to enforce the specific set of policies on a collection of resources, methods, or both, or operations that are associated to the scope. An API can have zero or more scope-level policies.

To create a scope-level policy

1. Click **APIs** in the title navigation bar.

   This displays a list of APIs available in API Gateway.

2. Click the name of the required API.

   This opens the API details page.

3. Click **Edit**.

   If the API is active, API Gateway displays a warning message to let you know that the API is active.

4. Click the **Policies** tab.

   This displays a list of scopes and policies available in the API.

5. In the **API Scope** box, select the scope for which you want to create a policy.

6. In the Policy catalog section, click the chevron to expand the required policy stage.

   This displays a list of policies that are classified under the particular stage.

7. In the expanded list of policies, select the policies that you want to associate with this scope. To select a policy, click the **Add** (+) icon next to the policy name. The selected policies are displayed in the **Infographic** section.

   When you select the policies for the scope-level policy, keep in mind that the policies shown in the **Policy catalog** section are determined by the type of the displayed API. If you do not see a policy that you need, that policy is probably not compatible with this API.

   Use the **Delete** (X) icon in any individual policy to remove that particular policy from the **Infographic** section.

8. In the Infographic section, do the following for each policy in the list:

   a. Select the policy whose properties you want to examine or set.

   b. In the Policy properties section, set the values for the policy's properties as necessary.

Note:
Required properties are marked with an asterisk.

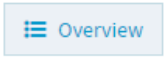
9. Click **Open in full-screen** to view the policy's properties in full-screen mode.

   The **Open in full-screen** link is located in the upper right-corner of the **Policies** tab.

10. Set the properties of the displayed policy, and then click **OK**.

    To exit out of full-screen mode, click the **Minimize** icon.

11. Click **Save** to create the new scope-level policy.

    Click 

to view the complete list of policies in the updated API. Activate the API, if it is not active, to put it into effect.

Viewing List of Scope-level Policies and Policy Details

The Infographic section displays the list of policies that are associated to a selected scope in the API's **Policies** tab.

To view the scope-level policies and properties of a policy

1. Click **APIs** in the title navigation bar.

   This displays a list of APIs available in API Gateway.

2. Click the name of the required API.

   This opens the API details page.

3. Click the **Policies** tab.

   This displays a list of scopes and policies available in the API.

4. In the **API Scope** box, select the scope whose policy details you want to examine.

5. In the Infographic section, do the following for each policy in the list:

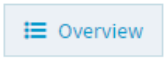
   a. Select the policy whose properties you want to examine.

   b. In the Policy properties section, examine the values for the policy's properties as required.

6. Click **Open in full-screen** to view the policy's properties in full-screen mode.

   The **Open in full-screen** link is located in the upper right-corner of the **Policies** tab. Examine the properties of the displayed policy, and then click **OK**.

To exit out of full-screen mode, click the **Minimize** icon.

7. Click  to view the complete list of policies in the updated API.

   The **Overview** button is located in the lower right-corner of the Infographic section.

   To exit the overview, click the **Close** icon.

Modifying Scope-level Policy Details

The API can have a set of policies that are configured globally through a global policy, or directly through a policy template, or a set of individual policies at the API-level or scope-level.

To customize the policy configurations at the scope-level, you need to apply the policies that are available for the API's scope, and then configure the properties of the individual policies to suit the needs of runtime behavior of that particular API.

You use the **Policies** tab to examine and modify the properties of a policy at the scope-level.

To modify the properties of a scope-level policy

1. Click **APIs** in the title navigation bar.

   This displays a list of APIs available in API Gateway.

2. Click the name of the required API.

   This opens the API details page.

3. Click **Edit**.

   If the API is active, API Gateway displays a warning message to let you know that the API is active.

4. Click the **Policies** tab.

   This displays a list of scopes and policies available in the API.

5. In the **API Scope** box, select the scope whose policy details you want to modify.

6. On the Infographic section, modify the policy list and the policy's configuration properties as necessary.

   Use the **Delete** (X) icon in any individual policy to remove that particular policy from the **Infographic** section.

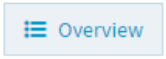
7. Click **Open in full-screen** to view the policy's properties in full-screen mode.

   The **Open in full-screen** link is located in the upper right-corner of the **Policies** tab.

8. Modify the properties of the displayed policy, and then click **OK**.

   To exit full-screen mode, click the **Minimize** icon.

9. When you have completed the required modifications for the scope-level policy, click **Save** to save the updated scope-level policy.

   Click 

to view the complete list of policies in the updated API. Activate the API, if it is not active, to put it into effect.

Deleting a Scope-level Policy

You delete a policy at the scope-level to remove the association between the policy and a scope.

When deleting a scope-level policy in the API details page, keep the following points in mind:

- When a scope is deleted from the API details, API Gateway removes the policies that were associated with the deleted scope.

To delete a scope-level policy

1. Click **APIs** in the title navigation bar.

   A list of all registered APIs appears.

2. Select the required API.

   This opens the API details page.

3. Click **Edit**.

   If the API is active, API Gateway displays a warning message to let you know that the API is active.

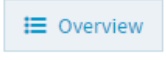
4. Click the **Policies** tab.

   A list of scopes and policies available with the API appears.

5. In the **API Scope** box, select the scope whose policy you want to remove.

6. On the Infographic section, click the **x** icon in any individual policy to remove that particular policy from the scope.

7. When you have removed the policy, click **Save** to save the updated scope-level policy.

   Click 

to view the complete list of policies in the updated API. Activate the API, if it is not active, to put it into effect.

Example: Usage Scenarios of API Scopes

API Provider can restrict the enforcement of policies at the resource-level or method-level for a REST API, and at the operations-level for a SOAP API. This policy enforcement on the resources, methods, or operations of the API will apply in addition to the default enforcement of policies at the global-level and the user-defined enforcement of policies at the API-level.

Consider you have a REST API, for example, *PhoneStore API*, with a collection of resources and methods.

Resource Name	Resource Path	Supported Methods
Resource A	/phones/orders	GET
		POST
Resource B	/phones/orders/{order-id}	GET
		PUT
		DELETE
Resource C	/phones/orders/{order-id}/paymentdetails	GET
		POST

The following section demonstrates the application of scopes and the policy enforcement using Resource C: /phones/orders/{order-id}/paymentdetails of the PhoneStore API.

You can create scopes in the PhoneStore API, and define the individual scopes with a specific set of resources, methods, or both.

Scope Name	Applied Resource	Applied Method
PAYMENT Scope	Resource C: /phones/orders/{order-id}/paymentdetails	
WRITE Scope	Resource C: /phones/orders/{order-id}/paymentdetails	POST

Assume you have an API-level policy which enforces an Identify & Authorize policy with HTTP Basic Authentication for the PhoneStore API. Now, you might need to have different authentication mechanisms for different methods and resources (collectively, scopes) of the PhoneStore API, depending on the level of access you need.

For example, you might want to enforce an Identify & Authorize policy for the Resource C in PAYMENT Scope to enforce secured access to the data. You might also want to apply an Identify & Authorize policy with API Key authentication and Traffic Optimization policy (with 5 API invocations per minute), in particular, for the POST method of the Resource C in WRITE Scope to enforce a higher-level of secured access and manipulation of the REST data.

API-level / Scope-level Policy	Applied Policies
API-level Policy	Identify & Authorize policy with HTTP Basic Authentication
Scope-level Policy for PAYMENT Scope	Identify & Authorize policy
Scope-level Policy for WRITE Scope	Identify & Authorize policy with API Key Traffic Optimization

The API Scopes definition looks like this:

API-level / Scope-level Policy	Applied Policies
API-level Policy	Identify & Authorize policy with HTTP Basic Authentication
Policy for PAYMENT Scope	Resource C: /phones/orders/{order-id}/paymentdetails Identify & Authorize policy
Policy for WRITE Scope	Resource C: /phones/orders/{order-id}/paymentdetails Method: POST Identify & Authorize policy with API Key Traffic Optimization

The precedence of the policy enforcement effective for an API at run-time is as follows:

1. Global Policy Enforcement
2. Method-level Policy Enforcement (REST APIs) -OR- Operation-level Policy Enforcement (SOAP APIs)
3. Resource-level Policy Enforcement (REST APIs)
4. API-level Policy Enforcement

The specific aspect of processing during the handling of an API invocation at run-time in API Gateway can be best understood with the following scenarios:

Scenario A: Invoke GET method on the Resource C: /phones/orders/{order-id}/paymentdetails

- Global Policy: Not applicable
- Method-level Policy: Not applicable
- Resource-level Policy(s): Identify & Authorize
- API-level Policy: Identify & Authorize policy with HTTP Basic Authentication

As per the precedence of policy enforcement, the Inbound Authentication - Transport at the resource-level and the Identify & Authorize policy with HTTP Basic Authentication at the API-level are enforced at run-time.

The effective policy set enforced on the API for the GET method at run-time includes:

- Identify & Authorize
- Identify & Authorize policy with HTTP Basic Authentication

Scenario B: Invoke POST method on the Resource C: /phones/orders/{order-id}/paymentdetails in WRITE Scope

- Global Policy: Not applicable
- Method-level Policy(s): (1) Identify & Authorize policy with API Key (2) Traffic Optimization
- Resource-level Policy(s): Identify & Authorize
- API-level Policy: Identify & Authorize policy with HTTP Basic Authentication

As per the precedence of policy enforcement, the Identify & Authorize policy with API Key at the method-level takes precedence over the Identify & Authorize policy with HTTP Basic Authentication at the API-level, and is enforced at run-time.

The effective policy set enforced on the API for the POST method at run-time includes:

- Identify & Authorize
- Identify & Authorize policy with API Key
- Traffic Optimization

Now, consider that you apply an active Global Policy that has the Identify & Authorize policy with Hostname Address for all REST APIs (including our PhoneStore API).

Scenario C: Invoke POST method on the Resource C: /phones/orders/{order-id}/paymentdetails in WRITE Scope

- Global Policy: Identify & Authorize policy with Hostname Address
- Method-level Policy(s): (1) Identify & Authorize policy with API Key (2) Traffic Optimization
- Resource-level Policy(s): Identify & Authorize
- API-level Policy: Identify & Authorize policy with HTTP Basic Authentication

As per the precedence of policy enforcement, the Identify & Authorize policy with Hostname Address applied through the global policy takes precedence over every other Identify & Authorize policy that is applied at the method-level and the API-level, and is enforced at run-time.

The effective policy set enforced on the API for the POST method at run-time includes:

- Identify & Authorize
- Identify & Authorize policy with Hostname Address

- Traffic Optimization

Resolving Scope Conflicts

When you save an API, API Gateway combines the scopes specified with the set of policies defined at the API-level, and on saving the API, API Gateway applies the policies to the API at various enforcement levels. API Gateway validates the scope list to ensure that it contains no conflicting or incompatible policies. If the list contains conflicts or inconsistencies, API Gateway prompts you with an error message.

Consider that you modify the existing UPDATE scope to include a POST method for Resource C. The API Scopes definition now looks like this:

API-level / Scope-level Policy	Applied Policies
API-level Policy	Identify & Authorize policy with HTTP Basic Authentication
Policy for PAYMENT Scope	Resource C: /phones/orders/{order-id}/paymentdetails Identify & Authorize policy
Policy for WRITE Scope	Resource C: /phones/orders/{order-id}/paymentdetails Method: POST Identify & Authorize policy with API Key Traffic Optimization
Policy for UPDATE Scope	Resource C: /phones/orders/{order-id}/paymentdetails Method: POST Identify & Authorize policy with API Key

Scenario D: Save the updated PhoneStore API.

- Global Policy: Not applicable
- Method-level Policy(s): (1) Identify & Authorize policy with API Key (2) Identify & Authorize policy with IP Address Range (3) Traffic Optimization
- Resource-level Policy(s): Identify & Authorize
- API-level Policy: Identify & Authorize policy with HTTP Basic Authentication

As per the precedence of policy enforcement, the Identify & Authorize policy at the method-level in WRITE and UPDATE Scopes take precedence over the Identify & Authorize policy at the API-level. But the Identify & Authorize policy with the API Key and IP Address Range authentications that are applied at the method-level results in a policy conflict.

To resolve the conflicts, you can choose one of the following workaround:

- **Option 1**: Remove the existing association between the POST method and the WRITE Scope or UPDATE Scope through the API Scope details.
- **Option 2**: Delete the WRITE Scope or UPDATE Scope.
- **Option 3**: Remove the Identify & Authorize policy from the WRITE Scope or UPDATE Scope.

Policy Templates

Important:
API Gateway's Standard Edition License does not support policy templates. You can create and manage policy templates only using the Advanced Edition License.

Policy templates are a set of policies that can be associated directly with an individual API. The direct association of the policy template with an API provides the flexibility to alter the policy's configurations to suit the individual API requirements.

To apply a policy template to an API, modify the details page of the API, and apply the selected policy template.

Creating a Policy Template

You must have the API Gateway's manage policy templates functional privilege assigned.

To create a policy template you must perform the following high-level steps:

1. **Create a new policy template**: During this step, you specify the basic details of the policy template.
2. **Configure the policies**: During this step, you associate one or more policies with the template, and assign values to each of the associated policy's properties.

To create a policy template

1. Click **Policies** in the title navigation bar.
2. Click the **Policy Templates** tab.

   A list of all available policy templates appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of templates you want to display in a page.
3. In the **Policies** page, click the **Create Policy Template** button.

   This opens the Create Policy Template page with the default **Policy Details** tab.
4. In the Basic Information section, provide the required information for each of the displayed data fields:

Field	Description
Name	Name of the policy template.
Description	Description of the policy template.

5. Click **Continue to policy configuration**.

6. In the **Policy Configuration** tab, select the policies and configure the properties for each policy that you want API Gateway to execute when it applies this policy template.

   For details, see “Associating Policies with a Policy Template” on page 503 and “ Configuring Properties for a Policy Template” on page 504.

7. Click **Save**.

Associating Policies with a Policy Template

You must have the API Gateway's manage policy templates functional privilege assigned.

The **Policy Configuration** tab on the Policy Template details page specifies the set of policy stages and the list of policies (applicable for each stage).

To modify the list of policies of a policy template

1. Click **Policies** in the title navigation bar.

2. Click the **Policy Templates** tab.

   A list of all available policy templates appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of templates you want to display in a page.

3. Select the required template.

   The Policy Template details page appears.

4. Click **Edit**.

5. Click the **Policy Configuration** tab.

   The policy template information is provided in the following sections:

   - Policy catalog - Transport, Identify and Access, Request Processing, Routing, Traffic Monitoring, Response Processing, Error Handling
   - Infographic - List of applied policies
   - Policy properties - Collection of policy properties

6. In the Policy catalog section, click the chevron to expand the required policy stage.

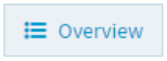
   This displays a list of policies that are classified under the particular stage.

7. In the expanded list of policies, select the policies that you want API Gateway to execute when it applies this policy template. To select a policy, click the **Add** (+) icon next to the policy name. The selected policies are displayed in the Infographic section.

   Use the **Delete** (X) icon in any individual policy to remove that particular policy from the Infographic section.

8. To configure the properties for any new policies that you might have added to the Infographic section in the preceding steps or to make any necessary updates to the properties for existing policies in the policy template, see " Configuring Properties for a Policy Template" on page 504.

9. When the list of policies is complete and you have configured all of the properties for the policies correctly, click **Save** to save the updated policy template.

10. Click 

to view the complete list of policies in the updated policy template.

    The **Overview** button is located in the lower right-corner of the Infographic section. In addition, you can view the configured properties for the individual policies.

    To exit the overview, click the **Close** icon.

Configuring Properties for a Policy Template

You must have the API Gateway's manage policy templates functional privilege assigned.

The **Policy Configuration** tab on the Policy Template details page specifies the list of policies that are applicable for each policy stage in the Policy catalog section. Each policy in the Infographic section has properties that you must set to configure the policy's enforcement behavior.

To configure the properties for a policy template

1. Click **Policies** in the title navigation bar.

2. Click the **Policy Templates** tab.

   A list of all available policy templates appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of templates you want to display in a page.

3. Select the required template.

   The Policy Template details page appears.

4. Click **Edit**.

5. Click the **Policy Configuration** tab.

   The policy template information is provided in the following sections:

   - Policy catalog - Transport, Identify and Access, Request Processing, Routing, Traffic Monitoring, Response Processing, Error Handling
   - Infographic - List of applied policies
   - Policy properties - Collection of policy properties

6. In the Policy catalog section, click the chevron to expand the required policy stage.

   This displays a list of policies that are classified under the particular stage.

7. In the Infographic section, do the following for each policy in the list:

   a. Select the policy whose properties you want to examine or set.

   b. In the Policy catalog section, set the properties as necessary.

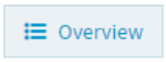
   > **Note:**
   > Required properties are marked with an asterisk.

8. Click **Open in full-screen** to view the policy's properties in full screen mode.

   The **Open in full-screen** link is located in the upper right-hand corner of the **Policy Configuration** tab. Set the properties of the displayed policy, and then click **OK**.

   To exit full screen mode, click the **Minimize** icon.

9. After you configure the properties for all of the policies in the Infographic section, click **Save** to save the updated policy template.

10. Click 

to view the complete list of policies in the updated policy template.

    The **Overview** button is located in the lower right-corner of the Infographic section.

    To exit the overview, click the **Close** icon.

Viewing List of Policy Templates and Template Details

The **Policy Templates** tab displays a list of all available policy templates in API Gateway. Policy templates are listed alphabetically by name.

In addition to viewing the list of policy templates, you can also examine the details of a template, create a copy of the template, and delete a policy template in the **Policy Templates** tab.

To view the policy template list and properties of a policy template

1. Click **Policies** in the title navigation bar.

2. Click the **Policy Templates** tab.

   A list of all available policy templates appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of templates you want to display in a page. This tab provides the following information about each template:

Column	Description
Name	Name of the policy template.
Description	The description for the policy template.

   You can also perform the following operations on a policy template:

   - Create a new copy of the policy template.
   - Delete a policy template to remove it from API Gateway.

3. Select the required policy template.

   The Policy Template details page appears. The policy template details are displayed in the following tabs:

   - **Policy Details**: This tab contains a summary of basic information such as the name and description of the policy template.
   - **Policy Configuration**: This tab contains the policy stages, applied policies, as well as the configuration details of individual policies.

Modifying Policy Template Details

You must have the API Gateway's manage policy templates functional privilege assigned.

You use the Policy Template details page to examine and modify the properties of a policy template.

To modify the properties of a policy template

1. Click **Policies** in the title navigation bar.

2. Click the **Policy Templates** tab.

   A list of all available policy templates appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of templates you want to display in a page.

3. Select the required template.

The Policy Template details page appears. The policy template details are displayed in the following tabs:

- **Policy Details**: This tab contains a summary of basic information such as name and description of the policy template.
- **Policy Configuration**: This tab contains the policy stages, applied policies, as well as the configuration details of individual policies.

4. Click **Edit**.
5. On the **Policy Details** tab, modify the basic properties of the policy as necessary.
6. On the **Policy Configuration** tab, modify the policy list and the policy's configuration properties as necessary.
7. When you have completed the required modifications on the Policy Template details page, click **Save** to save the updated policy template.

   If update of a policy template fails, API Gateway displays a pop-up style error message.
8. Click **Overview** to view the complete list of policies in the updated policy template.

   The **Overview** button is located in the lower right-corner of the Infographic section.

   To exit the overview, click the **Close** icon.

Deleting a Policy Template

You must have the API Gateway's manage policy templates functional privilege assigned.

You delete a policy template to remove it from API Gateway permanently.

To delete a policy template

1. Click **Policies** in the title navigation bar.
2. Click the **Policy Templates** tab.

   A list of all available policy templates appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of templates you want to display in a page.
3. Click the **Delete** ( ) icon for the required template.
4. Click **Yes** in the confirmation dialog.

Copying a Policy Template

You must have the API Gateway's manage policy templates functional privilege assigned.

A policy template can become quite complex, especially if it includes many policies. Instead of creating a new policy template from scratch, it is sometimes easier to copy an existing template that is similar to the one you need and edit the copy.

When you create a copy of a policy template, be aware that:

- When API Gateway creates a copy of a policy template, the new copy of the policy template is identical to the original one.
- There is no expressed relationship between the copy and the original policy (that is, API Gateway does not establish any type of association between the two policy templates).

In general, a copied policy template is no different from a policy template that you create from scratch.

To copy a policy template

1. Click **Policies** in the title navigation bar.
2. Click the **Policy Templates** tab.

   A list of all available policy templates appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of templates you want to display in a page.
3. Click the **Copy** icon for the required template.
4. In the **Copy of Policy Template** dialog box, provide the required information for each of the displayed data fields:

Field	Description
Name	Name of the policy template. API Gateway automatically adds the name of the existing policy template to the **Name** field. You can change the name of the template to suit your needs. But you cannot leave this field empty.
Description	The description for the policy template.

5. Click **Copy** to save the new policy template.
6. Modify the new policy template as necessary and then save it.

Applying a Policy Template on the API Details Page

You must have the API Gateway's manage APIs functional privilege assigned.

The **Policies** tab on the API details page specifies the set of policies that API Gateway will execute when an application requests access to that particular API.

The API can have a set of policies that are applied through a global policy, through a policy template, through a scope-level policy, and through API-level policies.

To customize the policy configurations for an API using a policy template, you need to apply the template (containing a set of policies), and then configure the properties of the individual policies to suit the runtime requirements for that API.

To apply a policy template on the API details page

1. Click **APIs** in the title navigation bar.

   A list of all registered APIs appears.

2. Select the required API.

3. Click **Edit**.

4. Click the **Policies** tab.

   The API's policy information is provided in the following sections:

   - Policy stages - Threat Protection, Transport, Identify and Access, Request Processing, Routing, Traffic Monitoring, Response Processing, Error Handling
   - Infographic - List of applied policies
   - Policy properties - Collection of policy properties

5. Click **Apply template** located in the lower right-corner of the **Infographic** section.

   This opens the **Apply template** dialog box.

6. In the **Template chooser**, select one or more policy templates that you want to apply to the API.

   You can choose to display the details of an individual policy template by clicking the **Info** icon. This option populates the list of policies that are defined in the particular template.

7. Select one or more policy templates that you want API Gateway to execute at run-time.

8. Click **Next**.

   You must have at least one template selected to use the **Next** button.

9. In the **Apply Templates to API** wizard, review the list of policies and the configuration details of the associated policies.

   - If necessary, you can click **Previous** to return to the **Template chooser** wizard and change your template selections.
   - If at any time you wish to abandon all your changes and return to the **Policies** tab, click **Cancel**.

10. Click **Apply**.

    If you have one or more policy conflicts, API Gateway displays the conflicting/incompatible policies with a **Conflict** icon. You can choose to resolve the policy conflicts and do a **Apply**, or simply continue to **Apply with conflicts**.

    If you choose the continue with conflicts, API Gateway sets the focus on the conflicting policies with Conflict () icon displayed next to the policy names in the Infographic section and the corresponding policy stages.

    API Gateway will redirect you to the **Policies** tab. The newly applied policy template comprising a set of policies and the policy properties is displayed in the Infographic section.

11. After you apply the required policy templates, click **Save** to save the updated API.

Post-requisites:

Activate the API when you are ready to put it into effect.

Modifying a Policy Template on the API Details Page

You must have the API Gateway's manage policy templates functional privilege assigned.

The **Policies** tab on the API details page specifies the set of policies that API Gateway executes when an application requests access to that particular API.

The API can have a set of policies that are applied through a global policy, through a policy template, through a scope-level policy, and through API-level policies.

To modify the details of a policy template on the API details page

1. Click **APIs** in the title navigation bar.

   A list of all registered APIs appears.

2. Select the required API.

3. Click **Edit**.

   If the API is active, API Gateway displays a warning message to let you know that the API is active.

4. Click the **Policies** tab.

   The API's policy information is provided in the following sections:

   - Policy catalog - Threat Protection, Transport, Identify and Access, Request Processing, Routing, Traffic Monitoring, Response Processing, Error Handling
   - Infographic - List of applied policies
   - Policy properties - Collection of policy properties

5. In the Infographic section, do the following for each policy in the list:

   a. Select the policy whose properties you want to examine or set.

   b. In the Policy properties section, set the properties as necessary.

   > **Note:**
   > Required properties are marked with an asterisk.

   c. Use the **Delete** (X) icon in any individual policy to remove that particular policy from the Infographic section.

6. Click **Open in full-screen** to view policy properties in full screen mode.

   The **Open in full-screen** button is located in the upper right-hand corner of the **Policy Configuration** tab.

7. Set the properties of the displayed policy, and then click **OK**.

   To exit full screen mode, click the **Minimize** icon.

8. Click **Save** to save the updated API.

   Activate the API, if it is not active, to put it into effect.

Saving Policy Definition of an API as Policy Template

You must have the API Gateway's manage policy templates functional privilege assigned.

The **Policies** tab on the API details page specifies the set of policies that API Gateway will execute when an application requests access to that particular API.

The API can have a set of policies that are applied through a global policy, through a policy template, through a scope-level policy, and through API-level policies.

You can save the current policy definition of an API as a new policy template. At a later time, you can reuse this policy template in other APIs. For more information, see " Applying a Policy Template on the API Details Page" on page 509.

To save policy definition as policy template

1. Click **APIs** in the title navigation bar.

   A list of all registered APIs appears.

2. Select the required API.

3. Click the **Policies** tab.

   The API's policy information is provided in the following sections:

   - Policy catalog - Threat Protection, Transport, Identify and Access, Request Processing, Routing, Traffic Monitoring, Response Processing, Error Handling
   - Infographic - List of applied policies
   - Policy properties - Collection of policy properties

4. Click **Save as template** located in the lower right-hand corner of the Infographic section.

5. In the **Save as template** dialog box, provide the required information for each of the displayed data fields:

Field	Description
Name	Name of the policy template.
Description	Description of the policy template.

6. Click **Save**.

Change Ownership of Assets

Assets such as APIs and applications in API Gateway have an option where the ownership of the asset can be changed. Applications have confidential data like API key and client certificates which only the owner can view. Therefore, if the owner of an asset has to take up a different responsibility or leave the organization, no other user can view the secrets of the asset. The edit option available on the asset details page, enables the transfer of ownership of the asset to another user, so that the new owner of the asset can access or view the confidential data of the asset. API Gateway provides an option to configure an approval process for the assets' ownership change. Approval and auditing contribute to the governance of change ownership.

Before you begin

Ensure that you have:

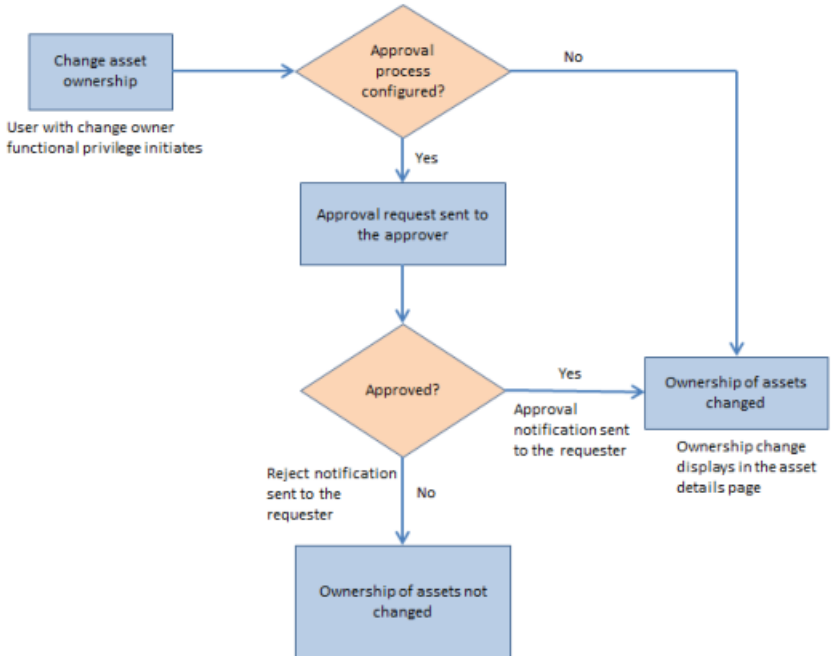
- API Gateway advanced edition version 10.5 or higher installed.

- Basic understanding of API Gateway and its related components like the API Gateway user interface.
- Change owner/team privilege.

  For details on functional privileges available, see *webMethods API Gateway Administration*.
- The change owner approval process configured and enabled if you want to enforce an approval process for ownership changes of assets.

  For details on configuring the approval process, see *webMethods API Gateway Administration*.

The figure depicts the workflow for changing ownership of assets.



How Do I Change the Ownership of an API?

This use case explains how to change the ownership of an API. You can configure an approval process for the change of ownership to take effect, if required.

The use case starts when you have an API that requires a change of owner and ends when you successfully change the API's ownership.

In this example, an API *petstore* is owned by *user1*. The ownership of *petstore* has to be changed to *user2* through an approval process.

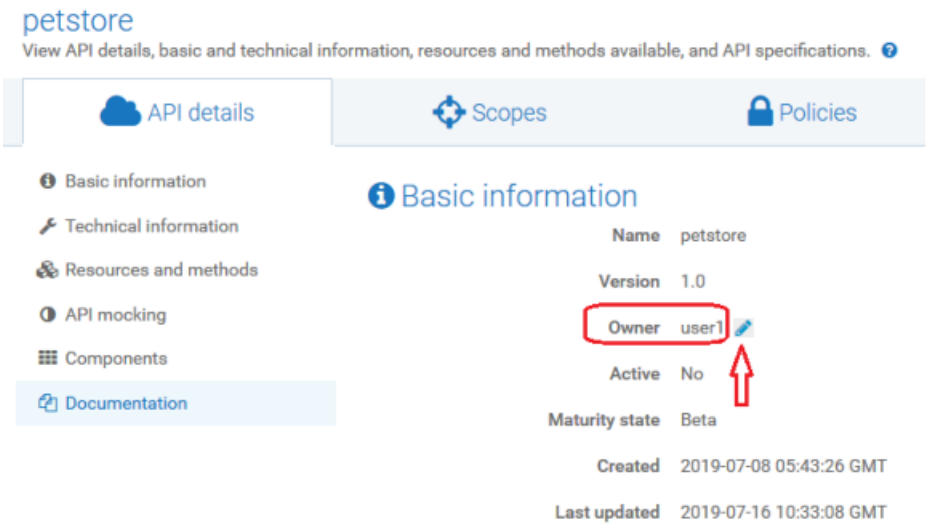
Before you begin

Ensure that you have the change owner privilege.

To change the ownership of an API

1. Log on to API Gateway as a user with the change owner privilege.
2. Click **APIs** on the title navigation bar.
3. Click **petstore**.

   The API details page appears. The owner of the API *petstore* is *user1* as displayed in the Basic information section.

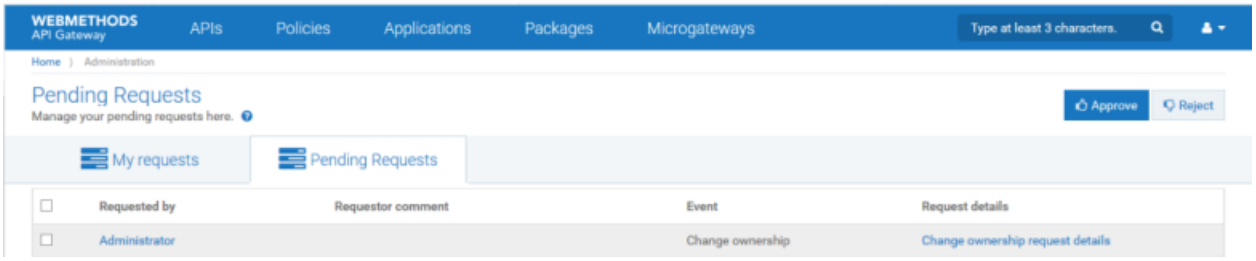


4. Click **change**.
5. Select `user2` from the list and click ✓.

   The change approval process is initiated.

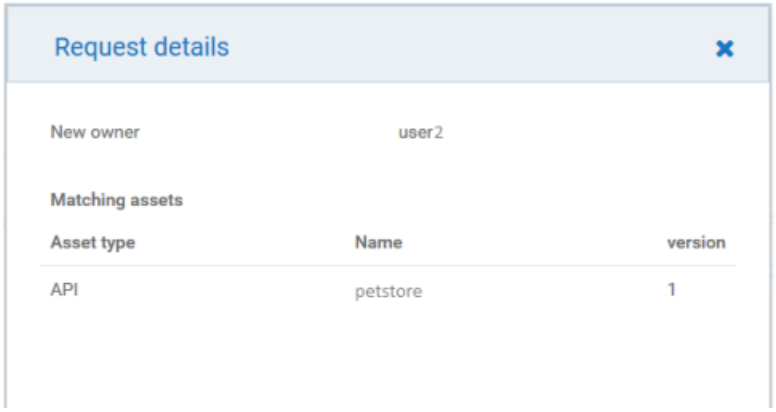
   Note:
   If the approval flow is not configured, the owner of the API changes to *user2* and a success message appears. Skip to step 8.

6. An approval request is sent to the approver.
7. The approver approves the request that resides in the Pending Requests section of the API Gateway UI.



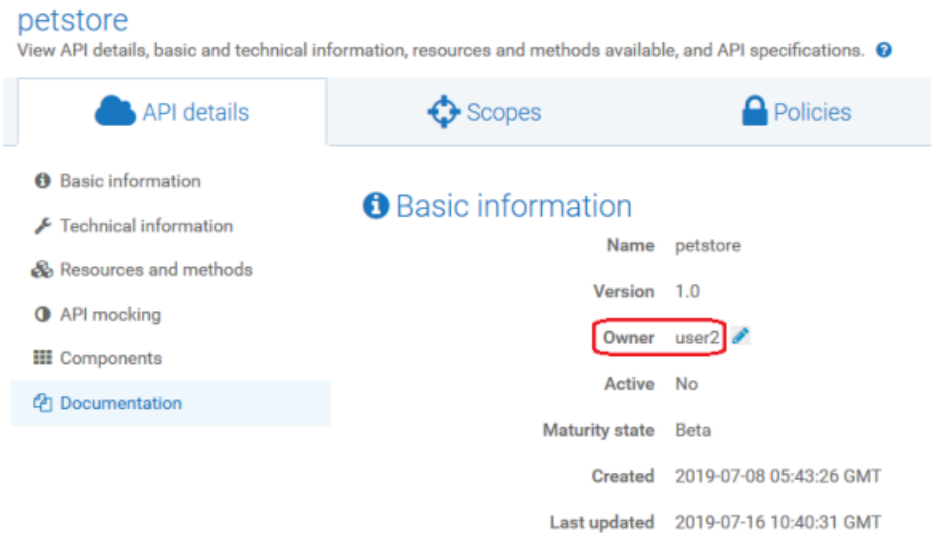
Note:
The approver can click **Reject** to reject the request for ownership change if the request is invalid. A reject notification is sent to the requester and the ownership of *petstore* remains with *user1*.

Click **Change ownership request details** to view the request details. The Request details dialog box appears.



The approval notification is sent to the requester.

8. The owner of the API *petstore* is changed from *user1* to *user2*.



How Do I Change the Ownership of an Application?

This use case explains how to change the ownership of an application. You can configure an approval process for the change of ownership to take effect, if required.

The use case starts when an application requires a change of owner and ends when you successfully change the application's ownership.

In this example, an application *App1* is owned by the team *Administrators*. The ownership of *App1* has to be changed to *DevTeam* through an approval process.

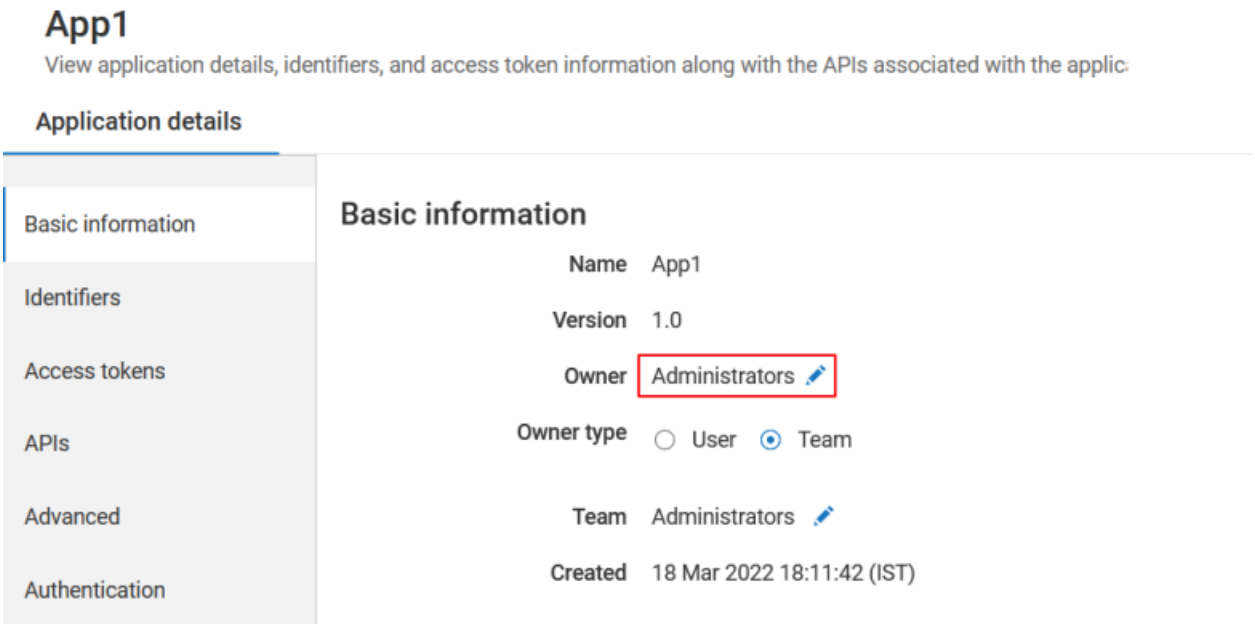
Before you begin

Ensure that you have the change owner privilege.

To change the ownership of an application

1. Log on to API Gateway as a user with the change owner privilege.
2. Click **Applications** on the title navigation bar.
3. Click the required application **app1**.

   The application details page appears. The owner of the application *App1* is *Administrators* as displayed in the Basic information section.



4. Click .
5. Select *Team*.

6. Select `DevTeam` from the list and click  .

   The change approval process is initiated.

   Note:
   If the approval flow is not configured, the owner of the application changes and a success message appears. Skip to step 8.

7. An approval request is sent to the approver.

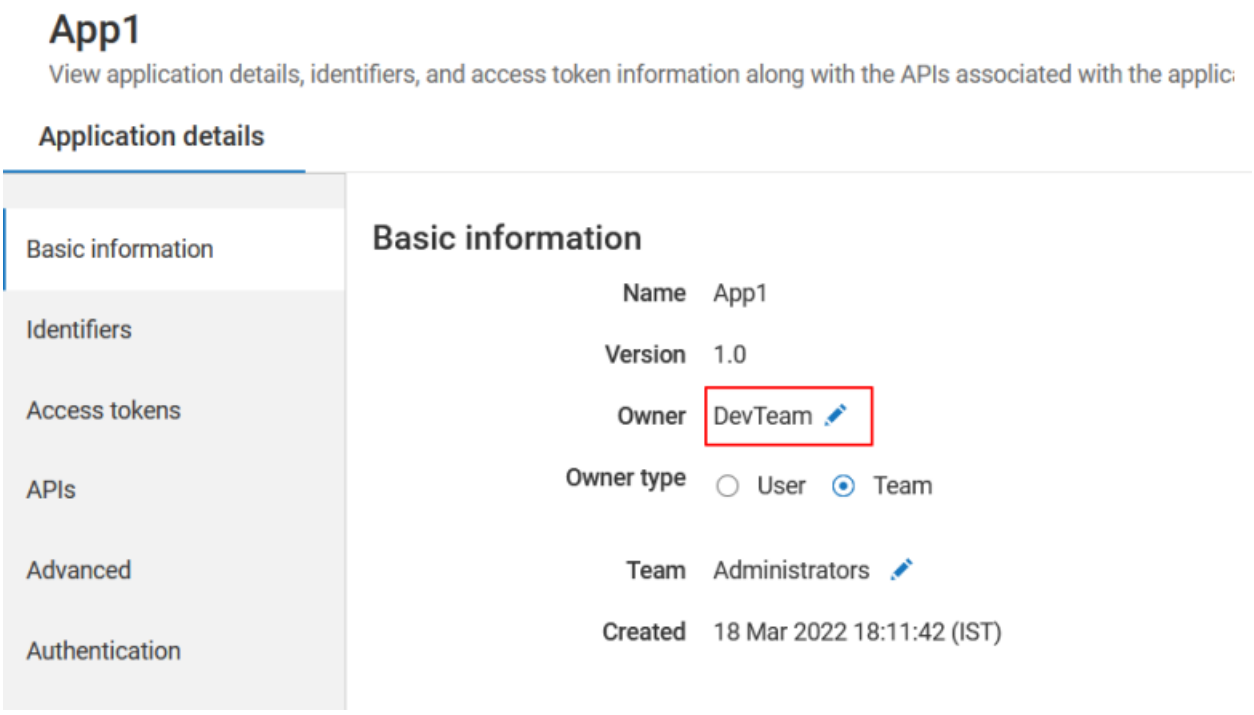
8. The approver approves the request that resides in the Pending Requests section of the API Gateway UI.

   Note:
   The approver can click **Reject** to reject the request for ownership change if the request is invalid. A reject notification is sent to the requester and the ownership does not change.

   Click **Change ownership request details** to view the request details. The Request details dialog box appears.

   The approval notification is sent to the requester.

9. The owner of the application *App1* is changed from *Administrators* to *DevTeam*.



Alternate steps

- You can select *User* or *Team* in the **Owner type** field (in *Step 5*)

  If you select *Team*, the list of teams appear in the field.

How Do I Change the Ownership of Multiple Assets?

It is convenient to change the asset ownership for multiple assets with a single REST request than doing it separately for individual assets. This use case explains how to change the ownership of multiple assets by sending a REST request. You can configure an approval process, if required, for the change of ownership to take effect.

The use case starts when multiple assets require change of owner and ends when you successfully change the ownership of the assets to another user.

To change the ownership of multiple assets

1. Use the following REST request to change the asset ownership to a new user.

```
POST http://host:port/rest/apigateway/assets/owner
Content-Type: application/json
{
     "assetType": "*", (API/APPLICATION)
     "assetIds": ["*"],
     "currentOwner": "user1",
     "newOwner": "user2"
}
```

   Provide the following information in the REST request:

   - `assetType`. Specifies the asset type for which you want to change the owner. Available values are `API`, `APPLICATION`, or the wildcard `*`. The wildcard `*` specifies all the assets, APIs and applications owned by the user specified in `currentOwner`.
   - `assetIds`. Specifies the ID of the assets specified in `assetType`.

     Note:
     This is optional. `assetIds` is not required if you specify `currentOwner`.

   - `currentOwner`. Specifies the user name of the owner of the assets specified in the `assetType` field.

     Note:
     If both `currentOwner` and `assetIds` are specified, both are validated. For example, consider *user1* and *user2* are owners of *assetID1* and *assetID2* respectively. In the request payload, if you include *assetID1* and *assetID2* in the `assetIds` field and *user1* in the **currentOwner** field, then only *assetID1* ownership changes.

   - `newOwner`. Specifies the user name of the user who would be the new owner of the assets specified.

   Example 1: If *user1* owns two assets, an API *petstore* and application *app1*, and you want the ownership to be transferred to *user2*, send a REST request as follows:

```
POST http://localhost:5555/rest/apigateway/assets/owner
Content-Type: application/json
{
     "assetType": "*", (API/APPLICATION)
```

```
     "currentOwner": "user1",
     "newOwner": "user2"
}
```

This request transfers the ownership of all the assets owned by *user1* to *user2*.

The change approval process is initiated.

Example 2: *user1* owns three APIs, *api1*, *api2*, and *api3* and 2 applications, *app1* and *app2*. If you want the ownership of *api1*, *api2*, and *app1* to be transferred to *user2*, send a REST request as follows:

```
POST http://localhost:5555/rest/apigateway/assets/owner
Content-Type: application/json
{
     "assetType": "*", (API/APPLICATION)
     "assetIds": ["apiID1, apiID2, appID1"],
     "currentOwner": "user1",
     "newOwner": "user2"
}
```

where *apiID1*, *apiID2*, and *appID1* are asset IDs of *api1*, *api2*, and *app1* respectively.

The change approval process is initiated.

Note:
If the approval flow is not configured, the ownership of the assets changes from *user1* to *user2*. Skip to step 4.

2. An approval request is sent to the approver.

   The approval request contains information of all the assets whose ownership needs to change and the new owners' name.

3. The approver approves the request in the Pending Requests section of the API Gateway UI.

   The approval notification is sent to the requester.

4. The owner of the assets is changed from *user1* to *user2*.

Debugging API

With Trace API support, you can monitor the complete life cycle of the runtime requests within API Gateway. This use case explains how to trace an API call in API Gateway. You can perform tracing for any runtime requests. Inspecting the failed runtime requests help you to debug and troubleshoot your API calls. You can trace REST, SOAP, and OData API calls only.

On enabling the tracer for an API, you can view the list of runtime requests that invoked the API. For each request, you can view

- the list of policies that were invoked in each stage
- time taken to execute the stage and its corresponding policies
- policy configured at the time of invocation

- values that were passed as input before the execution of the policies and values that were transformed at the end of the policy execution
- conditions and transformations that were applied and performed at the time of invocation
- server log captured at the time of invocation

> **Note:**
> Server logs are captured based on the log level settings enabled for runtime requests. To capture detailed logs during tracing, set the log level to `DEBUG` or `TRACE` for all the required stages in the Integration Server.

Important considerations when you trace an API:

- When you create a new API version from an API for which tracing is enabled, by default tracing is disabled in the newly versioned API.
- When you import an API with the **Overwrite** option selected as `All` or `Custom - API`, and if the API already exists after you import the API, by default the trace is disabled. You have to enable trace explicitly.
- When you promote an API with the option **Overwrites assets except alias that already exist on the selected target stages** selected, by default after you promote the API to the target instance, the trace is disabled. You have to enable trace explicitly.
- API Gateway does not support tracing for threat protection policies and rules.
- API Gateway does not support tracing for Microgateway groups.

The following policies are covered as part of trace API:

- Transport
  - Enable HTTP/HTTPS
  - Set Media Type
- Identify & Access
  - Authorize Users
  - Identify & Authorize
  - Custom Extension
- Request Processing
  - Invoke webMethods IS
  - Request Transformation
  - Data Masking
  - Custom Extension
- Routing

- Straight Through Routing
- Custom HTTP Header
- Outbound Auth - Transport
- Custom Extension

- Response Processing
  - Invoke webMethods IS
  - Response Transformation
  - CORS
  - Data Masking
  - Custom Extension
- Error Handling
  - Data Masking
  - Custom Extension

Note:
You can enable or disable tracing for an API by using the Service Management REST API. For more details about this API, see *webMethods API Gateway Developer's Guide*.

How do I enable tracing?

This use case starts when you want to enable trace for an API and ends when you view the trace details for that API.

Before you begin

Ensure that you have:

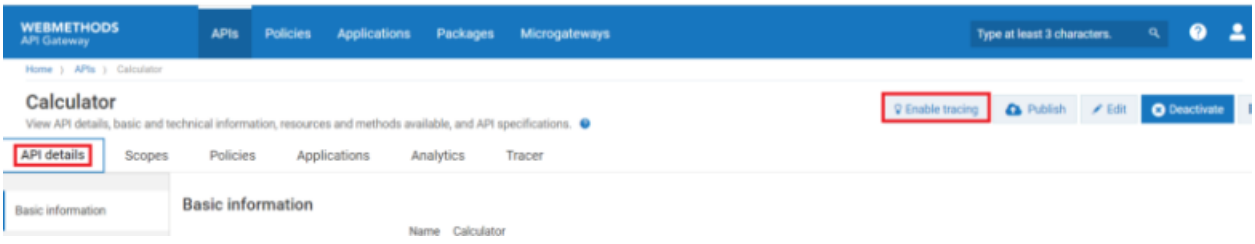
- *Manage APIs* privilege.
- Activated the API before you enable the tracer.

To enable tracing

1. Click **APIs** in the title navigation bar.
2. Click an API for which you want to enable the trace.

   The **API details** page displays the basic information, technical information, resources and methods, and specification for the selected API.
3. Click the **Enable tracing** button.

Once you have enabled the tracer, the API details page displays the warning message, `This API has tracing enabled. Tracing impacts performance and storage, hence disable tracing when it is not needed.`



4. Click the **Tracer** tab to view the trace details

   The Trace API page displays the **Runtime events**, **Policies applied**, and **Event tracer details** sections.

> **Note:**
> When you enable tracer, API Gateway captures a large amount of data, which might impact the performance and availability of the product. Hence Software AG strongly recommends you to disable the tracer when not needed and employ data house keeping procedures. For more information about Data housekeeping, see *webMethods API Gateway Administration* .

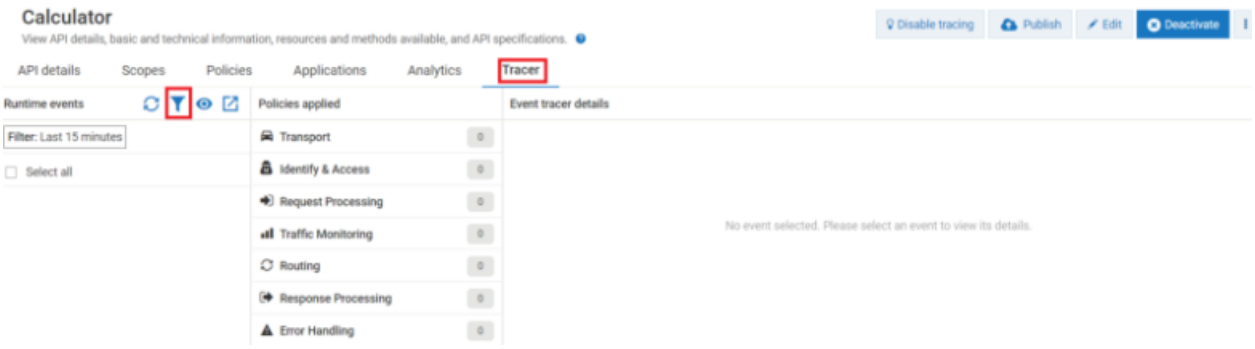
How do I filter the runtime request?

This use case starts when you want to filter the client request based on its runtime events and ends when you view the trace details of the filtered client request.

To filter the runtime event

1. Click the **Tracer** tab.

   The Trace API page displays the **Runtime events**, **Policies applied**, and **Event tracer details** sections.

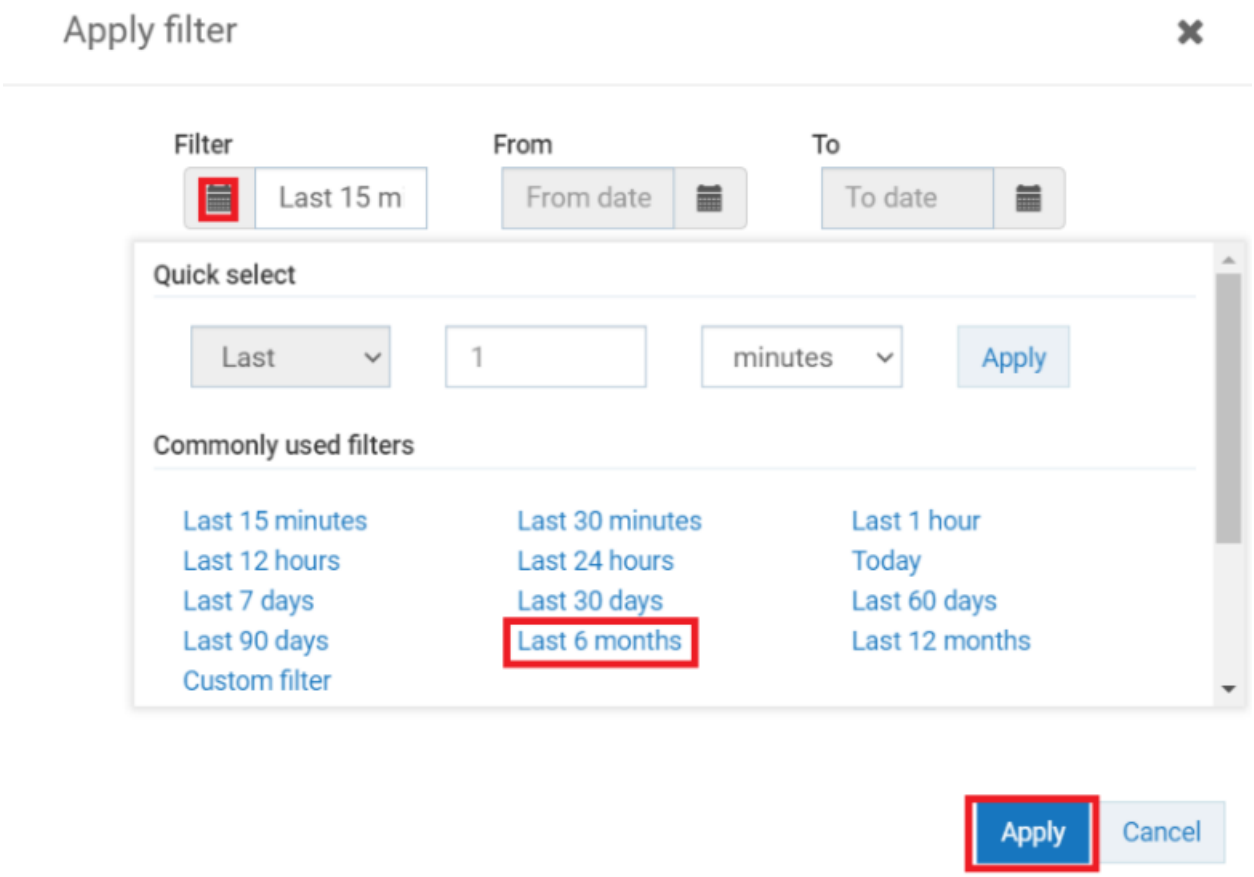


2. In the **Runtime events** section, click  to filter the runtime events.

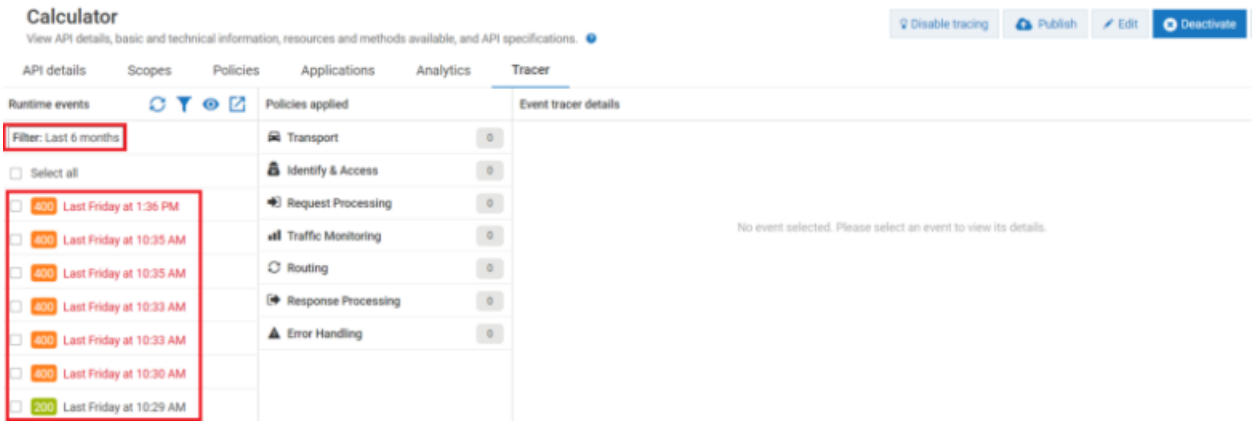
   The **Apply filter** pop-up window displays.

3. To filter the runtime events, click  and select the time interval using the options:

   - **Quick select**. Specify the time interval. Click **Apply** to filter the runtime events based on the time interval.
   - **Commonly used**. Select a commonly used time interval, and the filter is applied automatically. To view the runtime events between a time interval, click **Custom range** > **From Date** > **To Date** > **Apply**.



The **Runtime events** section displays the list of runtime events based on the applied filter.



The runtime events are displayed using various legends to indicate the different types of requests along with their status code.

The below table displays the legends and their description:

Legends	Description
2XX	Successful API calls
4XX	Failed API calls due to client-side errors
5XX	Failed API calls due to Server-side errors
3XX	Redirection calls
1XX	Informational calls

How do I view the trace details?

This use case starts when you want to view the stage-wise and policy-wise trace details about the selected client request and ends when you view the trace details at the policy level.

Before you begin

Ensure that you have:

- *Manage APIs* and *Activate APIs* privileges.
- Invoked the API after you have enabled the tracer.

To view the trace details

1. In the **Runtime events** section, click the client request for which you want to view the trace details.

   The Trace API page refreshes and populates data in the **Policies applied** and **Event tracer details** sections. By default, the **Event tracer details** section displays the **General Information**, **API request and response**, and **Server logs** sections. Under the **API request and response** section, you have the following sub-sections:

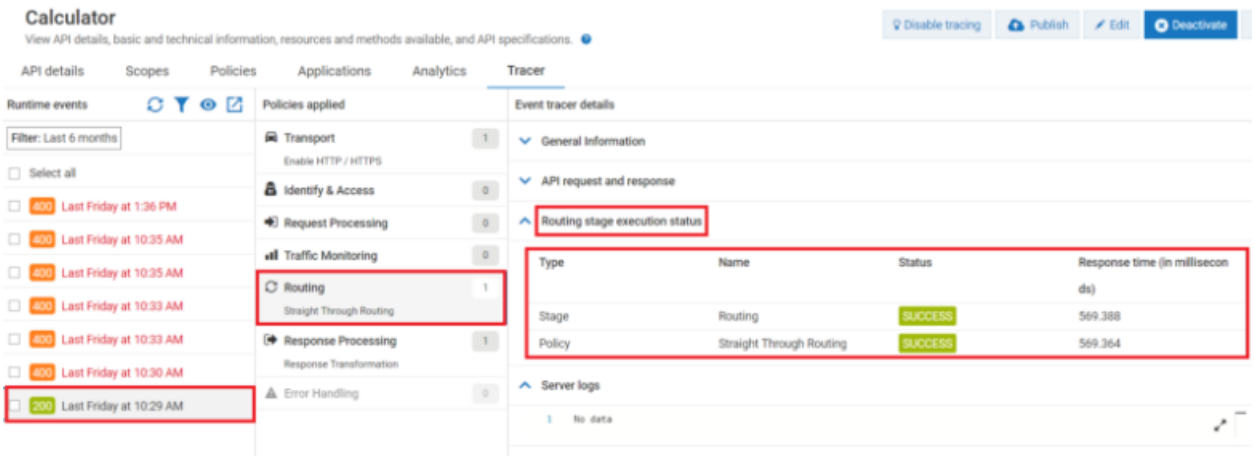
   - **Request sent by client**. Displays the request headers and request body sent by the client to API Gateway.
   - **Response sent to client**. Displays the response headers and response body sent to the client from API Gateway.

- **Request sent to native service** . Displays the request headers and request body sent to the native API from API Gateway.
- **Response sent by native service**. Displays the response headers and response body sent by the native API to API Gateway.

Note:
If the request and response body has streaming content, the tracer does not capture the streaming content even if you have enabled the tracer.

2. In the **Policies applied** section, click the stage name for which you want to view the trace details.

   The Trace API page refreshes the **Event tracer details** section with the ***stage_ name*** **stage execution status** section displaying the status and response time of the stage and policies that are enforced during API invocation.



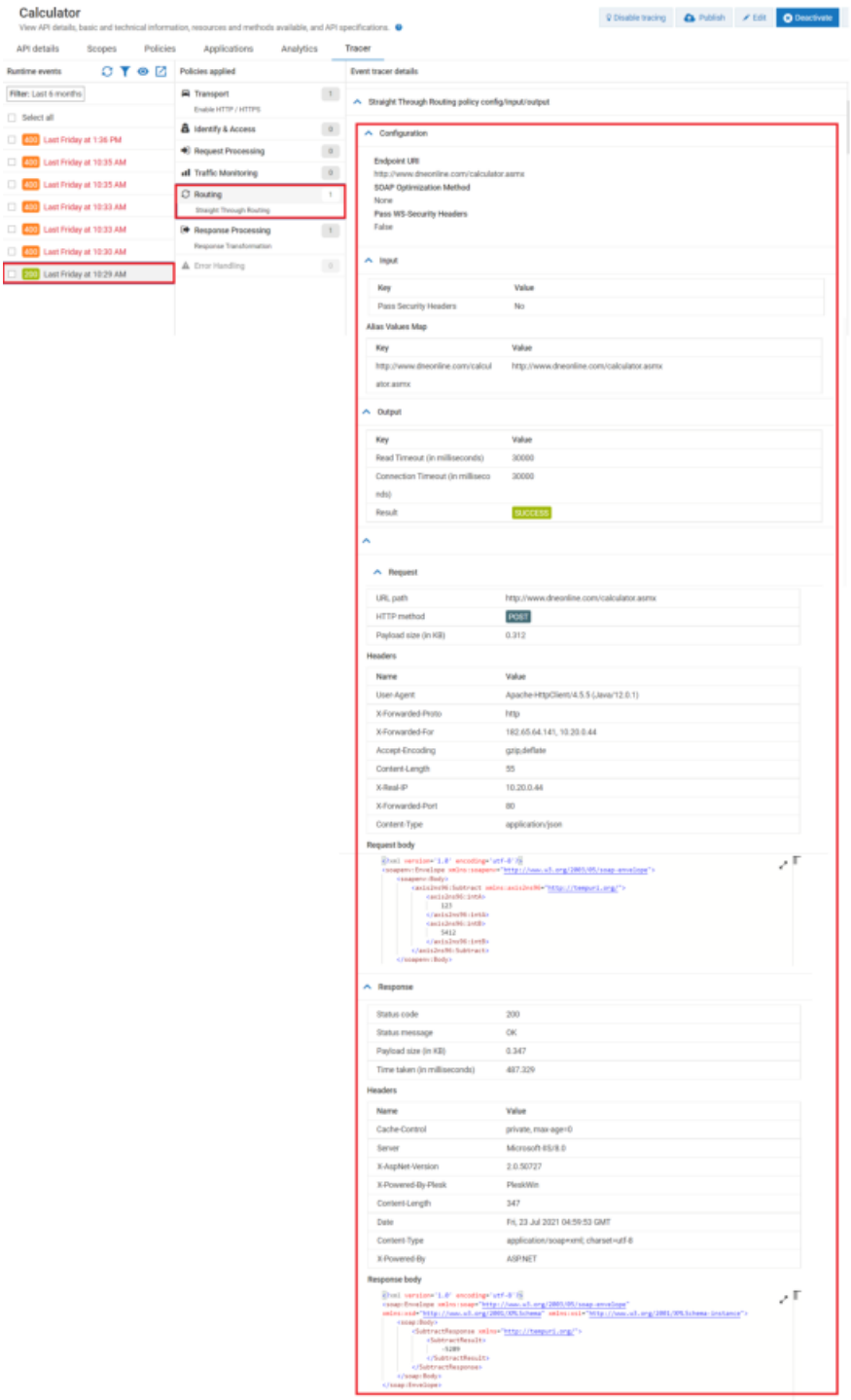
Note:

- In the **Policies applied** section, if no policies is enforced in a stage during the invocation then that stage is disabled. In this use case, **Error Handling** stage is disabled.
- The **Status** column indicates that whether the corresponding policy is invoked or not.

  If the **Status** column displays SUCCESS, it indicates the corresponding policy is enforced successfully during invocation but it does not mean that the conditions specified in the policy are matched. You can click the respective policy to know more details on how the conditions were applied during invocation.

3. In the **Policies applied** section, click the policy name for which you want to view the trace details.

   The Trace API page refreshes the **Event tracer details** sections with the ***policy_name*** **policy config/input/output** section displaying the configuration details, values that were passed as input before the enforcement of the policies and values that were transformed at the end of the policy enforcement, conditions and transformations that were applied and performed at the time of invocation, and payloads.



Note:
The template of the ***policy_name*** **policy config/input/output** section varies based on the policy.

How do I inspect failed runtime requests using tracer?

This use case starts when you want to inspect the failed runtime request and ends when you debug and troubleshoot the failed API requests.

To inspect the failed runtime request

1. In the **Runtime events** section, click the client request for which you want to inspect the trace details.

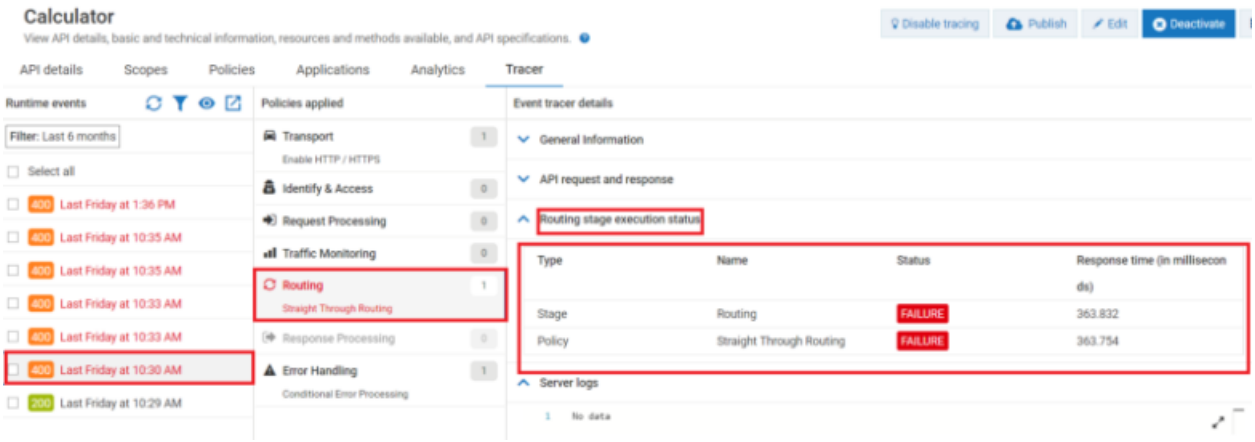
   The Trace API page refreshes and populates data in the **Policies applied** and **Event tracer details** sections. By default, the **Event tracer details** section displays the **General Information**, **API request and response**, and **Server logs** sections. Under the **API request and response** section, you have the following sub-sections:

   - **Request sent by client** . Displays the request headers and request body sent by the client to API Gateway.
   - **Response sent to client**. Displays the response headers and response body sent to the client from API Gateway.
   - **Request sent to native service** . Displays the request headers and request body sent to the native API from API Gateway.
   - **Response sent by native service**. Displays the response headers and response body sent by the native API to API Gateway.

   Note:
   If the request and response body has streaming content, the tracer does not capture the streaming content even if you have enabled the tracer.

2. In the **Policies applied** section, click the stage name highlighted in red for which you want to inspect the trace details.

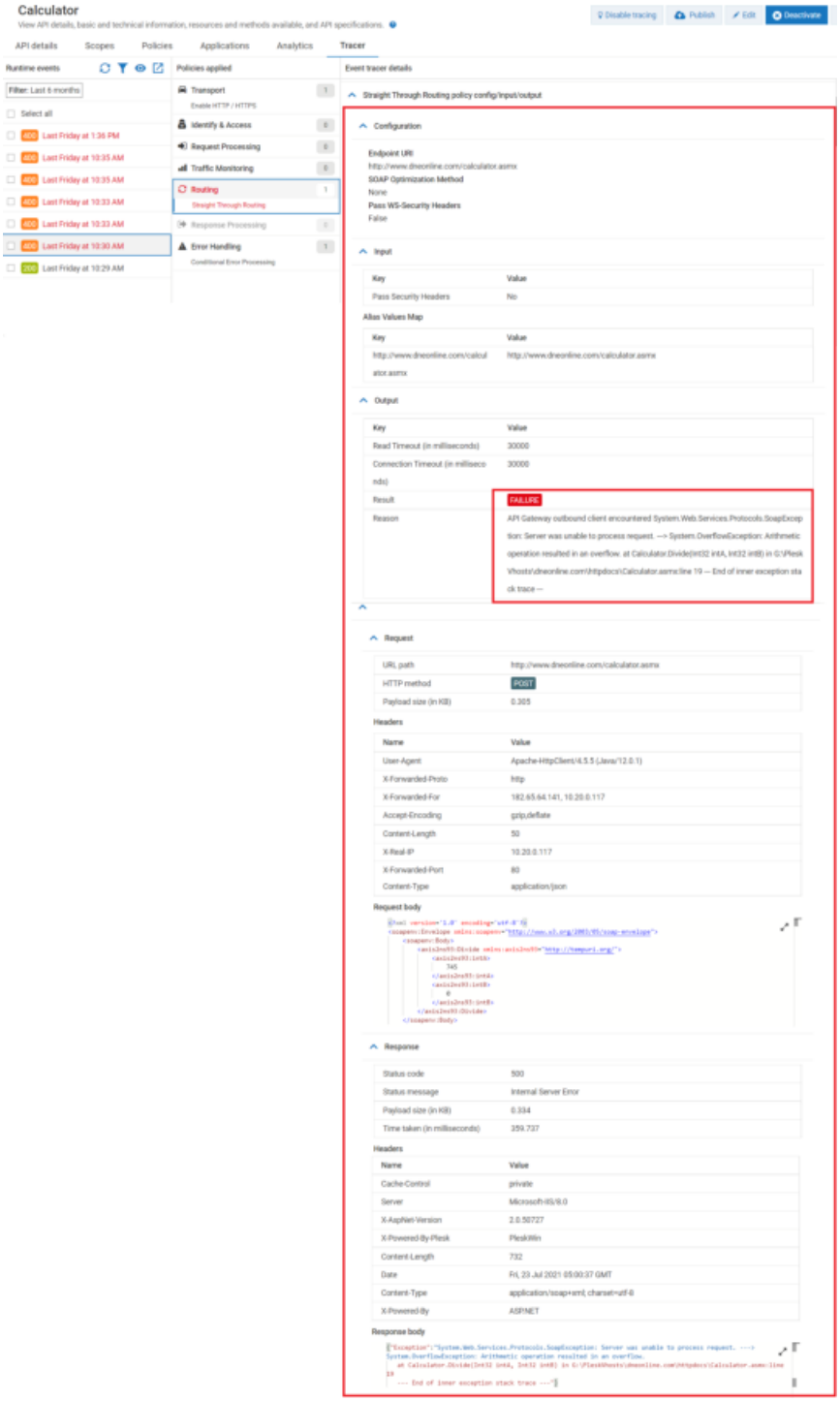
   The Trace API page refreshes the **Event tracer details** section with the ***stage_ name*** **stage execution status** section displaying the status and response time of the stage and policies that failed during API invocation.



Note:
In the **Policies applied** section, if no policies in a stage is enforced during the invocation then that stage is disabled. In this use case, **Response Processing** stage is disabled and it is not enforced as the API invocation fails in the **Routing** stage. The **Error Handling** stage was enforced in order to handle the **Routing** stage failure.

3. In the **Policies applied** section, click the policy name request highlighted in red for which you want to inspect the trace details.

   The Trace API page refreshes the **Event tracer details** sections with the ***policy_name*** **policy config/input/output** section displaying the configuration details, values that are passed during the enforcement of that policy, transformation conditions, and payloads. It also highlights the exact location where the policy invocation failed along with the failure reason.



Note:
The template of the ***policy_name*** **policy config/input/output** section varies based on the policy.

How do I import runtime requests?

This use case starts when you want to import the client request from any other API Gateway instance to your API Gateway instance and ends when you view the trace details for the imported request.

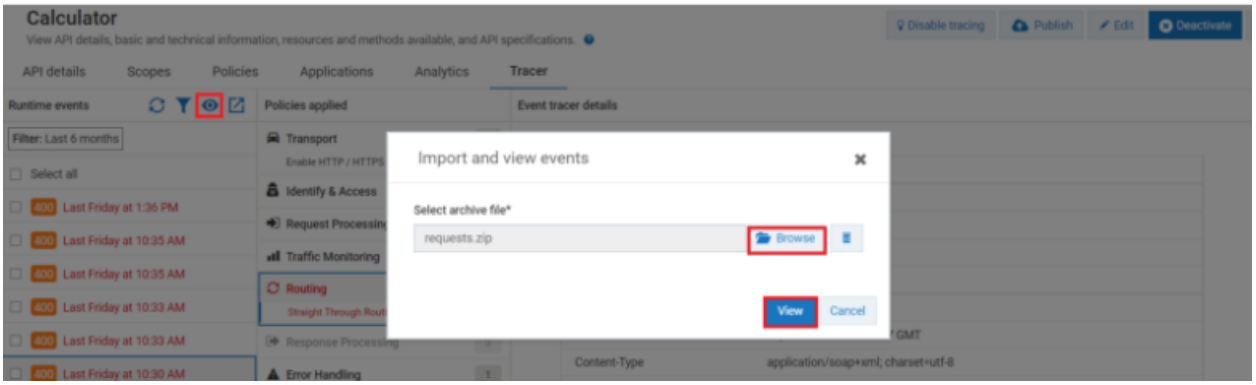
Before you begin

Ensure that the imported request's API ID matches with the API ID to which you import the request. The API type must also match with the API to which you import the archived request. If the imported request's API ID or API type does not match with the existing API, API Gateway rejects the import request.

To import the runtime request

1. Click the **Tracer** tab.

   The Trace API page displays the **Runtime events**, **Policies applied**, and **Event tracer details** sections.



2. In the **Runtime events** section, click to import the archived runtime request.

   The **Import and view events** pop-up window displays.

3. Browse the runtime request file that you want to import.

   Note:
   Make sure the file that you import does not exceed 50 MB.

4. Click the **View** button.

   The imported request gets displayed in the **Runtime events** section.

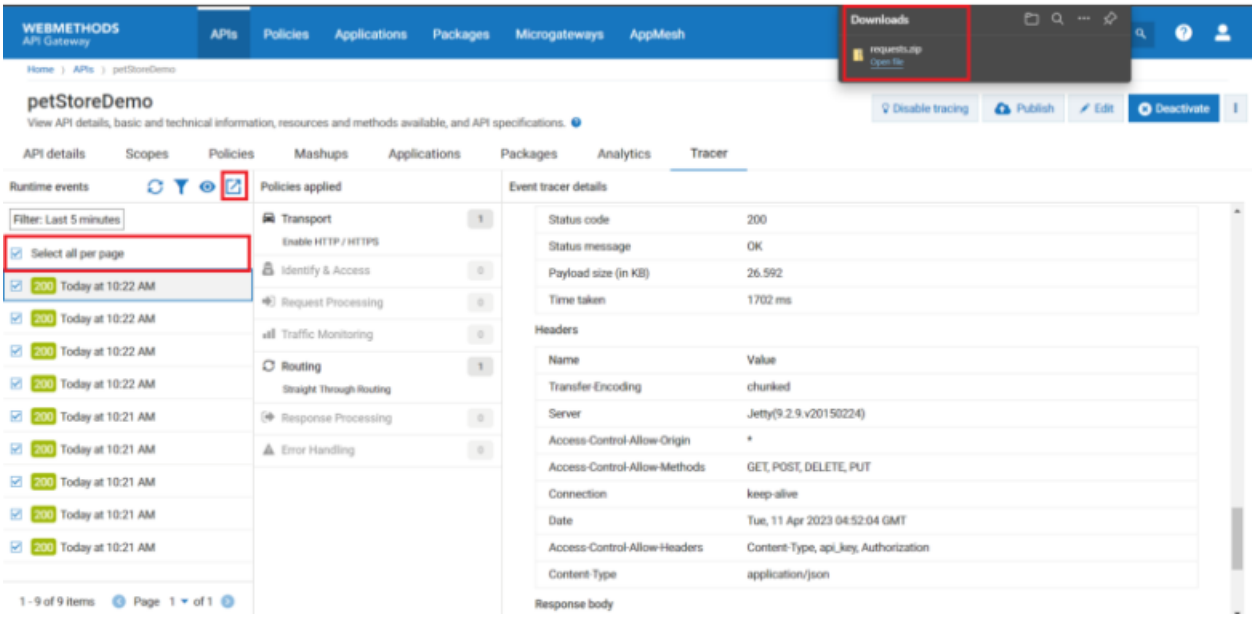
How do I export or download runtime requests?

This use case starts when you want to export the client request from your API Gateway instance to your local machine and ends when you import the request in another API Gateway instance.

To export the runtime request

1. Click the **Tracer** tab.

   The Trace API page displays the **Runtime events**, **Policies applied**, and **Event tracer details** sections.



2. In the **Runtime events** section, select the runtime event that you want to export.

   Note:
   The **Runtime events** section lists only 20 runtime events per page. When you click **Select all per page** check box, all the runtime events of the API do not get selected. Instead, the 20 runtime events that are listed in that particular page gets selected.

3. Click  to export the runtime request.

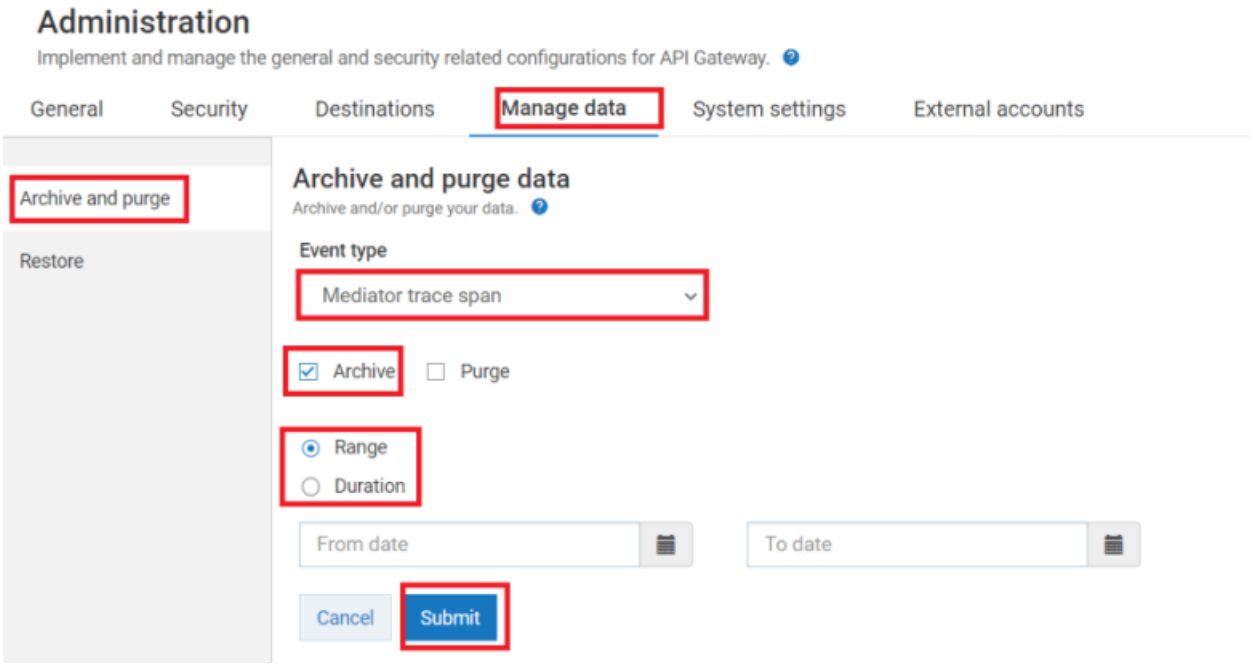
   The selected request is downloaded to your local machine in a predefined location.

How do I archive or purge the tracer details?

This use case starts when you want to archive or purge the tracer details and ends when you have successfully archived or purged the tracer details.

To archive or purge the trace details

1. Expand the menu icon in the title bar and select **Administration** > **Manage Data** > **Archive and purge**.



2. Select

   - the event type as `Mediator trace span` to archive or purge the stage-wise mediator details captured when you enabled the tracer.
   - the event type as `Server log trace` span to archive or purge the server logs that were captured when you enabled the tracer.
   - the event type as `Request response trace` span to archive or purge the requests and response logs that were captured when you enabled the tracer.

   Note:
   Ensure that all the three event types mentioned are archived or purged, so that the tracer does not impact the performance of API Gateway.

3. Click either the

   - **Archive** check box to archive the tracer data.
   - **Purge** check box to purge the tracer data.

4. Select one of the following options to archive the required data.

   - Select **Range**. Select a period during which you want the data to be archived.

- To archive selected types of data from a particular date till the current date, select the required date in the **From date** field.
- To archive selected types of data from the beginning (events start date) till a particular date, select the required date in the **To date** field.

  API Gateway archives the selected type of data for the specified date range.

- Select **Duration**. Type the maximum time after which you want the data to be archived.

  API Gateway archives the selected types of data after the time specified in years, months, days, hours, minutes, or seconds (1y, 1m, 1d, 1H, 1M, 1S).

5. Click the **Submit** button.

   Based on your selection, API Gateway archives or purges the trace details.

How do I archive and purge the tracer details using REST API Calls?

API Gateway provides the following REST API and the resources to archive and purge the trace details:

To archive the trace details use the following REST API calls:

- `POST/rest/apigateway/apitransactions?action=archive&eventType=serverLogTraceSpan&from=yyyy-MM-dd HH:mm:ss&until=yyyy-MM-dd HH:mm:ss`

  This archives the server logs that were captured when you enabled the tracer for the specified range.

- `POST/rest/apigateway/apitransactions/archives?action=archive&eventType=serverLogTraceSpan&olderThan= 7d`

  This archives the server logs that were captured when you enabled the tracer for the specified duration.

  With this REST API call you can archive the server logs that were captured during the last 7 days. Similarly, you can archive the last months and years trace details by specifying the **olderThan** as `2y` for 2 years and `3M` for 3 months. You can specify the number of days, years, and months as per your need.

To purge the trace details use the following REST API calls:

- `DELETE/rest/apigateway/apitransactions?action=purge&eventType=serverLogTraceSpan&from=yyyy-MM-dd HH:mm:ss&until=yyyy-MM-dd HH:mm:ss`

  This deletes the server logs that were captured when you enabled the tracer for the specified range.

- `DELETE/rest/apigateway/apitransactions?action=purge&eventType=serverLogTraceSpan&olderThan= 7d`

  This deletes the server logs that were captured when you enabled the tracer for the specified duration.

  With this REST API call you can delete the server logs that were captured during the last 7 days. Similarly, you can delete the last months and years trace details by specifying the **olderThan** as `2y` for 2 years and `3M` for 3 months. You can specify the number of days, years, and months as per your need.

To archive and purge the trace details use the following REST API calls:

- `DELETE/rest/apigateway/apitransactions?action=archiveAndPurge&eventType=serverLogTraceSpan&from= yyyy-MM-dd HH:mm:ss&until=yyyy-MM-dd HH:mm:ss`

  This archives and deletes the server logs that were captured when you enabled the tracer for the specified range.

- `DELETE/rest/apigateway/apitransactions?action=archiveAndPurge&eventType=serverLogTraceSpan&olderThan= 7d`

  This archives and deletes the server logs that were captured when you enabled the tracer for the specified duration.

  With this REST API call you can archive and delete the server logs that were captured during the last 7 days. Similarly, you can archive and delete the last months and years trace details by specifying the **olderThan** as `2y` for 2 years and `3M` for 3 months. You can specify the number of days, years, and months as per your need.

Note:
In all these REST API calls,

- if you want to archive and purge the stage-wise mediator details, specify the **eventType** as `mediatorTraceSpan`.
- if you want to archive and purge the requests and response logs, specify the **eventType** as `requestResponseTraceSpan`.

When you make these REST API calls, you can see the job ID in the response, which is used in the following REST API call to retrieve the status of the job.

To view the status of archive or purge jobs:

You can view the status of archive or purge jobs using the following REST API call:

`GET/rest/apigateway/apitransactions/jobs/JobID` retrieves the status of the specified job ID.

Sample request: GET/rest/apigateway/apitransactions/jobs/ca108bf0-34f3-4726-83a0-2eab4f8b947

Sample response payload:

```
{
    "status": "Completed",
    "action": "purge",
```

```
    "jobId": "ca108bf0-34f3-4726-83a0-2eab4f8b9473",
    "creationDate": "2021-08-16 09:07:35 GMT",
    "totalDocuments": 4456,
    "deletedDocuments": 4456
}
```

API Mashups

Overview

Servers that provide an API may expose a vast set of functionality. However, each individual service in the API usually provides a very specific functionality. While this is usually effective, sometimes it is useful or required to consolidate a few services and expose them as a single service. In other situations, you might want to extend a service with the functionality provided by an external API. API mashups address these requirements for grouping services and exposing them as a single service.

Note:
Currently, API Gateway supports API mashups for REST APIs only. You can define a mashup only in a REST API and only REST APIs can be included in the mashup.

The APIs that are included in an API mashup (participating APIs) can be connected to each other in the following ways:

- **API chaining**. Two or more participating APIs are connected and invoked in a sequence—one after the other.
- **API aggregation**. Two or more participating APIs are connected to a common aggregator step and invoked in the specified sequence—one after the other. The aggregator step captures the response of the aggregated APIs. The aggregator step enables you to:
  - Collate the responses and pass to the next step.
  - Process the responses and pass the processed data to the next step.

Usage scenario: API chaining

Assume an API that provides information about courses offered by different universities in a given location. This API provides a service that returns the list of universities for a given course name and postal code. This service could be:

```
GET /universities?course=medicine&postalcode=600012
```

The provider of the API wants to extend this API for use in mobile applications that have access to users' location. As mobile applications can access a user's location in terms of longitude and latitude, this involves first retrieving the postal code for the users' current location and then passing that information to the existing API.

Suppose there is a publically available API that returns the postal code based on longitude and latitude values. This service could be:

```
GET /postalcode?lat=331&long=22324321
```

If this public API meets other requirements, such as security, performance, and usage limits, it can be utilized to deliver the required functionality.

Using an API mashup, you can create and expose a single service that calls both services: the external service that returns the postal code and the existing service that provides the list of universities. The resulting service could be:

```
GET /universities?course=medicine&lat=331&long=22324321
```

Usage scenario: API aggregation

Assume an IT services provider that provides hosting and cloud services to its customers. Users can create accounts for the different types of services that they need to use: bare metal servers, Virtual Private Servers, platforms as a service, and so on. A customer has multiple types of accounts. The statement for each type of account is returned by a different API. The API provider wants to provide a single API that consolidates the statements of a given customer and returns a single response with all the information.

Key Features of a REST API Mashup

- An API mashup allows you to orchestrate multiple resources and methods and expose the behavior as a single service. In a regular method that is not a mashup, API Gateway applies all the enforced policies and then routes the request to the native endpoint. In the case of a mashup, API Gateway still applies all the enforced policies in the request flow till routing; but thereafter, it starts the orchestration flow defined in the mashup. After the orchestration flow ends, all the policies defined for that method are applied in the response flow—in the same way as a regular method.
- API mashups are defined at the method level. You can edit any REST API and define a mashup for one or more methods within it.
- You can include any REST API defined within API Gateway in the mashup.
- The entire framework that API Gateway provides to a regular REST API method is available to an API mashup method. Therefore, you can utilize query parameters, path parameters, aliases, variables, payload transformations using XSLT transforms, transformations using webMethods IS services, and custom pipeline variables.

Considerations for Creating an API Mashup

- By default, the policies of an API that is participating in an API mashup are not enforced when it is invoked within the API mashup. However, if you select the **Should execute Outbound policies** option, the outbound security policies of the participating API are enforced in the context of the API mashup.
- The following are specific to a mashup step and are not automatically passed from one step to another:
  - Headers
  - Query parameters

- Path parameters
- Payload

  However, you can add parameters in a mashup step to access data from any of the previous steps or another source.

  An exception to this rule is the first step (the first participating service) in a mashup, which receives the complete request sent by the client.

- A participating API cannot have reverse invoke routing.

Structure of an API Mashup

An API mashup consists of one or more mashup steps, and each step invokes an API. A mashup step defines the request for the API that it invokes. A step can use the data objects provided by API Gateway to access data in the initial request sent to the operation that has the mashup and any of the previous steps.

The following table summarizes the data objects and variables that are available in API Gateway.

Object/Variable Type	Possible values
`paramStage`	- `request` - `response`
`paramType`	- `payload` or `body` - `headers` - `query` - `path` - `httpMethod` - `statusCode` - `statusMessage`
`queryType`	- `xpath` - `jsonPath` - `regex`

The following data objects are available to a mashup step:

- `${paramStage.paramType}`

  You can use this syntax to access the following string variables: `path`, `statusCode`, `statusMessage`, `httpMethod`. Examples: `${request.path}`, `${response.statusCode}`

- `${paramStage.paramType.paramName}`

You can use this syntax to access map types, such as query, headers, and path. Example: `${request.query.var1}`, `${response.header.Content-Type}`, `${request.path.name}`.

- `${paramStage.paramType.queryType[queryValue]}`

  This syntax can be used to query a paramType. Examples:

  - `${request.payload.xpath[//ns:emp/ns:empName]}`

    Where `//ns:emp/ns:empName` is the XPath to be applied on the payload if contentType is `application/xml`, `text/xml`, or `text/html`.

  - `${response.payload.jsonPath[$.cardDetails.number]}`

    Where `$.cardDetails.number` is the jsonPath to be applied on payload if contentType is `application/json` or `application/json/badgerfish`.

  - `${request.payload.regex[[0-9]+]}`

    Where `[0-9]+` is the regular expression to be applied on the payload if contentType is `text/plain`.

  Note:
  While `xpath` and `jsonPath` are applicable only to payload, `regEx` can be used with both `payload` and `path`.

- `${paramStage[stepName].paramType.queryType[queryValue]}`

  You can use this syntax to access data in any step. For example, you can use the following syntax to access the payload of a step named `createAPI`: `${response[createAPI].payload.jsonPath[$.apiResponse.api.id]}`.

- You can define your own variables using the **Custom Pipeline variables** field:

  Examples: **${key}**, **${value}**. The custom pipeline variables that you define are available in subsequent steps.

Note:
Data objects from any of the steps of the mashup can also be accessed by response processing policies and error processing policies of the API that contains the mashup.

Creating an API Mashup

To create a mashup you require:

- The API must include the resource and the method in which you want to add the API mashup.
- The participating APIs (that you want to include in the mashup) must exist in the API Gateway instance.

To create a mashup in a REST API

1. Click **APIs** in the title navigation bar.

   A list of all registered APIs appears.

2. Click the required API.

   The API details page appears.

3. Click **Edit**.

4. Click **Mashups**.

   The Mashups tab appears. It displays the resources in the API and their methods on the left and an empty (Default routing) routing diagram.

   Note:
   If the API does not have any mashup, the Mashup tab displays the list of resources only in the **Edit** mode; the tab is empty in the view mode.

5. In the **List of resources**, click the resource in which you want to include the mashup.

   The resource tab expands and the methods included in the resource are displayed.

6. Click the toggle button to enable the method in which you want to create the mashup.

   Note:
   If you use the toggle button and disable a method that has a mashup, the mashup definition for that method is immediately cleared.

7. Click **Add invoke** to add a mashup step.

   a. Connect the step to **Start**.

      The **Start** and **Stop** terminators and all steps have connection points that you can connect to the other steps and terminators. p

      To select a connection point and connect it to another connection point:

      a. Hover the mouse over the top or bottom of the step or terminator till the connection point is highlighted.

      b. Click the connection point and drag to the other step or terminator.

   b. Configure the step properties as desired.

      The Mashup Routing panel that appears on the right side of the mashup canvas displays the properties for the selected step. You can configure the following properties using the Mashup Routing panel:

Section	Field	Description
Mashup step name		Provide a name for the mashup step that is unique within the mashup.
API Endpoint		The API endpoint that you want to invoke in the mashup step. The API must be published on the current API Gateway instance.
	API Gateway API	The endpoint of the API that you want to use. You can type a few letters and select from the autocomplete list.
	Resource	The resource in the API that you want to use. You can type a few letters and select a resource from the autocomplete list.
	Method	The specific method of the resource that you want to invoke.
	Execute outbound authentication policy	Select if you want the outbound security policies of the participating API to be enforced in the context of an API mashup.
Headers		
	Use incoming Headers	Select to use the headers in the incoming request.
	Custom Headers	Custom headers that you can add in addition or instead of the incoming headers. Each custom header must have the following fields: ■ **Header Name** ■ **Header Value**
Query Parameters		Provide the following values: ■ **Query Parameter Name** ■ **Query Parameter Value**
Path Parameters		Provide the following values: ■ **Path Parameter Name** ■ **Path Parameter Value**
Payload		Type the **Payload**.

Section	Field	Description
	XSLT Document	Click **Add xslt document** and select the XSLT file for transforming the payload. Provide the following values: ■ **XSLT File** ■ **Feature Name** ■ **Feature value** For information about transforming the payload using XSLT, see "Request Transformation" on page 240.
	XSLT Transformation alias	Click **Add xslt transformation alias** and select an existing XSLT transformation alias.
Advanced Transformation		
	webMethods IS service	Click **Add webMethods IS service** and provide the following values: ■ **webMethods IS Service** ■ **Run As User** ■ Select **Comply to IS Spec** For information about these fields and using the webMethods IS Service, see "Invoke webMethods IS" on page 246.
	webMethods IS Service Alias	For information about the webMethods IS Service Alias, see "Invoke webMethods IS" on page 246.
Transformation Metadata		
	Namespace	Provide the following values: ■ **Namespace Prefix** ■ **Namespace URI** For information about transformation metadata, see "Request Transformation" on page 240.

Section	Field	Description
Custom Pipeline Variables		You can use custom pipeline variables to hold values that need to be used in another step of the API mashup. Provide the following values: ■ **Name** ■ **Value** For more information, see "Structure of an API Mashup" on page 540.

Note:
In several fields, such as **Header Value** within custom headers, **Query Parameter Value**, and **Path Parameter Value**, you can use values from previous steps and other data using the variable and alias framework provided by API Gateway. For more information, see "Structure of an API Mashup" on page 540.

8. Click **Add aggregator** to add an aggregator step.

Note:
You can also add an aggregator step by connecting two invocation steps to the same previous step. An aggregator step is automatically added after the steps when you connect the second step to the same previous step.

9. If you have added the aggregator by clicking **Add aggregator**, add the following connections:

   a. Connect the steps that need to be aggregated to the aggregator step.

   b. Connect the aggregator step to the next step.

10. To add additional steps to the aggregated block, complete the following steps:

    a. To add a new step to the aggregated block, click **Add invoke** and connect the new step to the same previous step.

       You can configure the properties of the new step immediately or later. For details on configuring the step properties, see step 7.

    b. To add an existing step to the aggregated block, delete the connections of the step, if any and then connect the step to the previous step for the aggregated block and the aggregator step.

11. Click the mashup step and configure the properties of the mashup step as desired.

    You can configure the mashup step properties using the Mashup Aggregator action panel that appears on the right side of the mashup canvas when you click the aggregator step. You can configure the following properties using the Mashup Aggregator action panel:

Section	Field	Description
Headers		
	Use incoming Headers	Select to use the headers in the incoming request.
	Custom Headers	Custom headers that you can add in addition or instead of the incoming headers. Each custom header must have the following fields: ■ **Header Name** ■ **Header Value**
Query Parameters		Provide the following values: ■ **Query Parameter Name** ■ **Query Parameter Value**
Path Parameters		Provide the following values: ■ **Path Parameter Name** ■ **Path Parameter Value**
Payload		Type the **Payload**.
	XSLT Document	Click **Add xslt document** and select the XSLT file for transforming the payload. Provide the following values: ■ **XSLT File** ■ **Feature Name** ■ **Feature value**
	XSLT Transformation alias	Click **Add xslt transformation alias** and select an existing XSLT transformation alias.
Advanced Transformation		
	webMethods IS Service	Click **Add webMethods IS service** and provide the following values: ■ **webMethods IS Service** ■ Select a **Run As User** ■ Select **Comply to IS Spec**

Section	Field	Description
		For information about these fields and using the webMethods IS Service, see "Invoke webMethods IS" on page 246.
	webMethods IS Service Alias	Select an existing webMethods IS service alias.
Transformation Metadata		
	Namespace	Provide the following values: ■ **Namespace Prefix** ■ **Namespace URI**
Custom Pipeline Variables		You can use custom pipeline variables to hold values that need to be used in another step of the API mashup. Provide the following values: ■ **Name** ■ **Value** For more information, see "Structure of an API Mashup" on page 540.
Mashup Response Transformation		■ Select **Aggregate response** ■ **Payload**

Note:
In several fields, such as **Header Value** within custom headers, **Query Parameter Value**, and **Path Parameter Value**, you can use values from previous steps and other data using the variable and alias framework provided by API Gateway. For more information, see "Structure of an API Mashup" on page 540.

12. Add, configure, and connect additional API invocation steps and API aggregator steps as desired.

13. Click **Save**.

    The mashup is created for the selected method.

Note:
You must activate the API to make the mashup available to client applications. For more information about activating an API, see "Activating an API" on page 566.

SOAP to REST Transformation

SOAP APIs are commonly used to expose data within enterprises. With the rapid adoption of the REST APIs, API providers must be able to provide RESTful interfaces to their existing SOAP APIs instead of creating new REST APIs. Using the API Gateway SOAP to REST transformation feature, the API provider can either expose the parts of the SOAP API or expose the complete SOAP API with RESTful interface. API Gateway allows you to customize the way the SOAP operations are exposed as REST resources. Additionally, the Swagger or RAML definitions can be generated for these REST interfaces.

Activating SOAP to Rest Transformation

You must have the Manage APIs functional privilege assigned to perform this task.

To activate SOAP to REST transformation for a SOAP operation

1. Click **APIs** in the title navigation bar.
2. Select the required API from the list of available APIs.

   The API details page for the selected API appears.
3. Click **Edit**.
4. Click **REST transformation**.

   A list of SOAP operations already exposed to the consumers as well as to be transformed from SOAP to REST appears. By default, all the SOAP operations are in inactive state.
5. Click  to activate the SOAP to REST transformation for the SOAP operations.

   Alternatively, you can activate the SOAP to REST transformation for multiple SOAP operations simultaneously by clicking the **Transform all operations** activation toggle button.
6. Select the operation to edit the SOAP operations.
7. In the Transformation Configuration section, configure the following settings:

   - **Use Schema for XML to JSON transformation**

     If you select this checkbox, the XML schema (present in the WSDL) defines the data type of the entity. The data type can either be String, Int, Double, Float, or Boolean. In the response from the native server, if an entity is of a different data type other than the ones defined in the XML schema, API Gateway returns it is as a String data type and not an error.

If you do not select this checkbox, API Gateway does not honor the XML schema during transformation. Instead, API Gateway derives the data type of entities based on the native service response.

By default, this checkbox is not selected for any SOAP API. If you have migrated your APIs from another instance of API Gateway , the value of this checkbox will depend on the value of the Extended property **pg.soapToRest.typeConvertorEnabled** in the source API Gateway of the migrated APIs. The **pg.soapToRest.typeConvertorEnabled** property specifies whether the key values in a SOAP request must be converted to their primitive type when a SOAP API is transformed to REST API.

Note:
Date and Enumeration data types are also considered to be strings. When API Gateway cannot determine the data type of any value, it considers the data type to be a string.

- **Use default values from schema**

  If you select this checkbox, API Gateway considers the default values provided in the XML schema, if there are no values present in a request or response. If the request or response has some value, this value overrides the default value from XML schema.

  If you do not select this checkbox, API Gateway does not consider the default values present in the XML schema even if there are no values present in the request or response.

- **Remove operation name in response**

  If you select this checkbox, the root node is not passed as a part of SOAP to REST response and only the JSON is passed. Root node is generally the SOAP operation name or SOAP operation response name, present in the XML schema. This check box is applicable only to JSON responses.

  If you do not select this checkbox, JSON response is accompanied by the root node. By default, this check box is not selected for any SOAP API.

8. Click **Save**.

   The API details page for the selected API appears.

9. Click **REST transformation**.

   A list of REST resources for the SOAP operations appears. Click on each resource to view the details that are available as REST definitions.

Modifying the REST Definitions for SOAP Operations

You must have the Manage APIs functional privilege assigned to perform this task.

To modify the REST definitions for SOAP operation

1. Click **APIs** in the title navigation bar.

2. Select the required API from the list of available APIs.

   The API details page for the selected API appears.

3. Click **Edit**.

4. Click **REST transformation**.

   A list of SOAP operations already exposed to the consumers as well as to be transformed from SOAP to REST appears.

5. Click  to activate SOAP to REST transformation, for the required operation.

   Alternatively, you can activate the SOAP to REST transformation for all the SOAP operations simultaneously by clicking the **Transform all operations** activation toggle button.

6. Provide the following information:

Field	Description
Resource name	Name of the resource. The existing name of the SOAP operation automatically appears, you can modify this name.
Resource path	Path of the resource. The existing path of the SOAP operation automatically appears, you can modify this path.
Tags	Tags to add to your operation. Select tags for the operation from the drop-down menu.
Description	Description of the resource. A few lines to describe the resource. This is an optional field.

7. Click **+ Add Parameter** and provide the following information to add the required resource level parameters:

Field	Description
Name	Name of the parameter.
Description	Description of the parameter.
Type	Specifies the parameter type. Available values are: **Path**, **Query-string**.
Data type	Specifies the data type. Available values are: **String**, **Date**, **Date time**, **Integer**, **Double**, **Boolean**.
Required	Specifies the parameter is required if selected.
Repeat	Applicable to parameters of type query. The query parameter value can take comma separated array values.
Value	Specifies the possible value.
XPath	Specifies how the request parameter must be mapped to the SOAP payload that is sent to the native SOAP service. For example, `/soapenv:Envelope/soapenv:Body/axis:sayHello/axis:name`, or `//axis:name` (If the SOAP request has only one element such as name).
Namespace prefix	Specifies the namespace prefix of the element that appears in the **XPath**.
Namespace URI	Specifies the namespace URI for the XPath element. You can add more namespace prefixes and namespace URIs by clicking  .

   You can add more parameters by clicking  .

8. Select one of the available methods: **GET**, **POST**, **PUT**, or **DELETE**. By default, **POST** is selected.

   By default, API Gateway generates the sample JSON request and response based on the XML schema definitions of the SOAP API. Additionally, you can provide a schema and modify the generated sample.

9. Click **Add Request** and provide the schema and a sample for the content-type.

10. Click **Add Response** and select the status code from the drop-down and provide a description for the status code selected.

    Additionally, to add a content-type to the status code selected, click the status code to which you want to add a content-type and select the **Content type**. Provide a schema and a sample for the content-type selected. By default, status code 200 is automatically generated by the system.

11. Click **Save**.

Supported Content-types and Accept Headers

The following table specifies the content-type available for the HTTP methods:

HTTP Method	Content-types	Accept Headers

GET	`application/x-www-form-urlencoded`	`application/json`
		`application/xml or text/xml`
		`multipart/form-data or multipart/mixed`

POST	`application/json`
	`application/xml or text/xml`
	`multipart/form-data or multipart/mixed`
	`application/x-www-form-urlencoded`
		`application/json`
		`application/xml or text/xml`
		`multipart/form-data or multipart/mixed`

PUT	`application/json`
	`application/xml or text/xml`
	`multipart/form-data or multipart/mixed`
	`application/x-www-form-urlencoded`
		`application/json`
		`application/xml or text/xml`
		`multipart/form-data or multipart/mixed`

DELETE	`application/x-www-form-urlencoded`	`application/json`
		`application/xml or text/xml`
		`multipart/form-data or multipart/mixed`

Note:
If a content-type is not specified, then the request verifies the value of the `Set Media Type` parameter. If the value of the `Set Media Type` parameter is not defined, then by default, for POST and PUT HTTP methods, the `application/json` content-type is used. Whereas for GET and DELETE HTTP methods, the `application/x-www-form-urlencoded` content-type is used.

REST API Endpoints

After providing the information required for the SOAP to REST transformation and activating the API, the API can be invoked as either SOAP or REST API.

The REST transformation of the SOAP API does not change the API name. The only change to the SOAP invocation is that the *resource-path-for-the-operation* is appended:

`/ws/API-NAME/version-number/resource-path-for-the-resource`

Note:
The REST-enabled SOAP API cannot be invoked using the `/rest` directive.

Samples for REST Request

application/json

The following table provides the samples of the REST request for the `application/json` content-type application and the equivalent SOAP request after transformation from REST to SOAP:

	Request	Equivalent SOAP Request
Consists of only one element (qualified namespaces)	`{ "name":"user1" }`	`<soapenv:Envelope xmlns:soapenv= "http://schemas.xmlsoap.org/soap/envelope/" xmlns:axis="http://ws.apache.org/axis2"> <soapenv:Body> <axis:sayHello> <axis:name>user1/axis:name> </axis:sayHello> </soapenv:Body> </soapenv:Envelope>`
Consists of only one element (non-qualified namespaces)	`{ "name":"user1" }`	`<soapenv:Envelope xmlns:soapenv= "http://schemas.xmlsoap.org/soap/envelope/" xmlns:axis="http://ws.apache.org/axis2"> <soapenv:Body> <axis:sayHello> <name>user1</name> </axis:sayHello> </soapenv:Body> </soapenv:Envelope>`
Consists of multiple elements	`{ "a":"1", "b" : 2 }`	`<soapenv:Envelope xmlns:soapenv= "http://schemas.xmlsoap.org/soap/envelope/"> <soapenv:Body> <addInts> <a>1</a><b>2</b> </addInts> </soapenv:Body> </soapenv:Envelope>`

application/xml and text/xml

The following table provides the samples of the REST request for the `application/xml and text/xml` content-type application and the equivalent SOAP request after transformation from REST to SOAP:

	Request	Equivalent SOAP Request
Consists of only one element and namespace added by the client	`<axis:name xmlns: axis= "http://ws.apache.org/axis2" >user1</axis:name>`	`<soapenv:Envelope xmlns:soapenv= "http://schemas.xmlsoap.org/soap/envelope/" xmlns:axis="http://ws.apache.org/axis2"> <soapenv:Body> <axis:sayHello> <axis:name>user1</axis:name> </axis:sayHello> </soapenv:Body> </soapenv:Envelope>`
Consists of only one element and client does not send the Namespace	`<someOtherNamespace :name xmlns:toMed= "http: //someOtherNamespace" >user1 </someOtherNamespace :name>`	`<soapenv:Envelope xmlns:soapenv= "http://schemas.xmlsoap.org/soap/envelope/" xmlns:axis="http://ws.apache.org/axis2"> <soapenv:Body> <axis:sayHello> <axis:name>user1</axis:name> </axis:sayHello> </soapenv:Body> </soapenv:Envelope>`
Consists of only one element and the client sends a different namespace to API Gateway	`<toMed:name xmlns: toMed="http://tOMed" >user1</toMed:name>`	`<soapenv:Envelope xmlns:soapenv= "http://schemas.xmlsoap.org/soap/envelope/" xmlns:axis="http://ws.apache.org/axis2"> <soapenv:Body> <axis:sayHello> <axis:name>user1</axis:name> </axis:sayHello> </soapenv:Body> </soapenv:Envelope>`
Multiple XML elements	`<addInts> <a>2</a> <b>3</b> </addInts>`	`<soapenv:Envelope xmlns:soapenv= "http://schemas.xmlsoap.org/soap/envelope/"> <soapenv:Body> <addInts> <a>2</a><b>3</b> </addInts> </soapenv:Body> </soapenv:Envelope>`

Path and Query Parameters

The following table provides the samples of the REST request having path and query parameters and the equivalent SOAP request after transformation from REST to SOAP:

	Request	Equivalent SOAP Request

Simple query or path parameter	`/ws/CalcService/add/{num1}`
	or
	`/ws/CalcService/add?num1=10`
		`<ws:addInts xmlns:ws="http:test"><num1>10</num1></ws:addInts>s`

Multiple query or path parameters	`/ws/CalcService/add/{num1}/{num2}`
	or
	`/ws/CalcService/add?num1=10&num2=3`
	or
	`/ws/CalcService/add/{num1}&num2=3`
		`<ws:addInts xmlns:ws="http:test"><num1></num1><num2></num2></ws:addInts>`

Hierarchical elements	`/ws/CalcService/add/{num1}/anotherNumber/{num2}` `/ws/CalcService/add/{num1}/anotherNumber/{num2}`	`<ws:addInts xmlns:ws="http:test"><num1></num1> <num2></num2></ws:addInts>`

multipart/form-data

If you send the `multipart/form-data` content-type as the REST request, then you have to optimize the method to be used. This optimization is based on the value specified in the `SOAP Optimization Method` parameter available in Routing policy. The default optimization type is Message Transmission Optimization Mechanism (MTOM). For example, API Gateway converts REST requests with `multipart/form-data` and `multipart/mixed` types as follows:

1. The Multipurpose Internet Mail Extensions (MIME) parts that have a content ID or name that match the elements of type `base64Binary` or `hexBinary` in the schema are added as attachments to the outbound request.
2. Parts other than the content ID or name types are converted into XML depending on the content-type of the MIME part. The `application/xml` and `application/json` content-types are converted. If API Gateway is unable to process the MIME part, it wraps the MIME part inside an XML element with the name of the content ID.

Limitations

The following limitations apply when you perform a SOAP to REST transformation:

- When the API provider defines the mapping for the SOAP operation to the REST resource or method, API Gateway allows the provider to specify either the path and the query parameters

or the body but not both. These mappings are used when transforming the incoming REST request to the SOAP request.

- If both path and query parameters and body are sent in the incoming REST request, then the path and the query parameters are ignored.
- If your REST resource accepts the `text/xml` content-type, then you cannot modify the default resource path and resource name automatically generated by the system. This name must be same as the SOAP operation name. However, this limitation is not applicable for other content-types.
- The HTTP method filters of the global policy are not applicable to the REST transformed method of the SOAP API.
- The REST (REST transformed SOAP operations) resources do not appear as general REST resources when filtered in the **Scopes** section of the API in API Gateway.
- You cannot apply the **Inbound Authentication-Message** policy to the SOAP operation enabled as REST.
- The SOAP services that have Web Services Interoperability Organization (WS-I) non-compliant WSDLs cannot be REST-enabled.

API First Implementation

APIs form the nerve center of software applications. So, it is very important for the providers to be clear about what they would provide and for the consumers to be clear about what they want to consume. Better understanding of APIs guarantee an excellent output. API First is all about the establishment of a common agreement between the providers and consumers. Thus, this design helps both the parties to be on the same page.

When adapting API First approach, API developers start the API development with the API contract and work on the implementation part at a later stage. This approach of prioritizing the API design over its implementation is beneficial to both, providers and consumers.

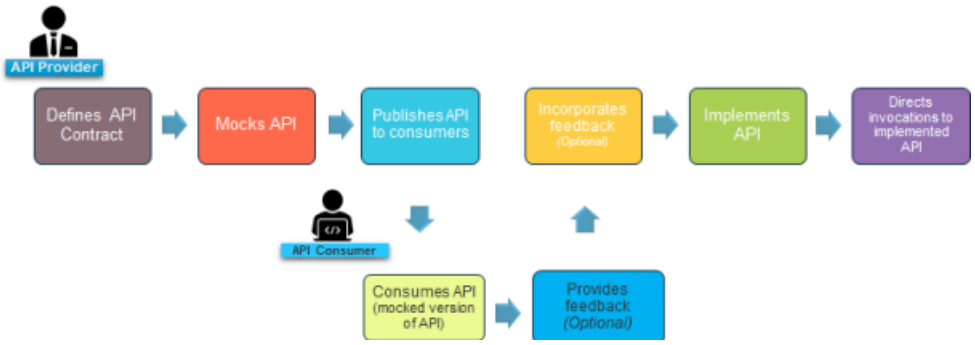
In conventional scenarios, providers expose APIs to their consumers only after the API is implemented. Consumers test the API and let the providers know their feedback about the API. Providers must then revisit the API to incorporate the feedback received from their consumers. You can optimize this process by adapting API First design.

When following API First approach, consumer does not have to wait for the provider to implement the API. Consumers can proceed with their application development using the exposed API. The implementation status of API does not have an impact on consumers as they receive the designated

responses for their requests through the mocked API. So, the API development and the application development can take place at the same time.

Once the provider implements the API, the end-point is updated to divert the invocations to the actual implementation instead of mocked response. The provider can then disable mocking.

The following diagram explains the flow of API development as per the API First design:



As per the API First design, providers expose their API to consumers when the development is underway.

API First Design using API Gateway

Starting API Gateway 10.5, the application provides seamless support for API First approach for your APIs.

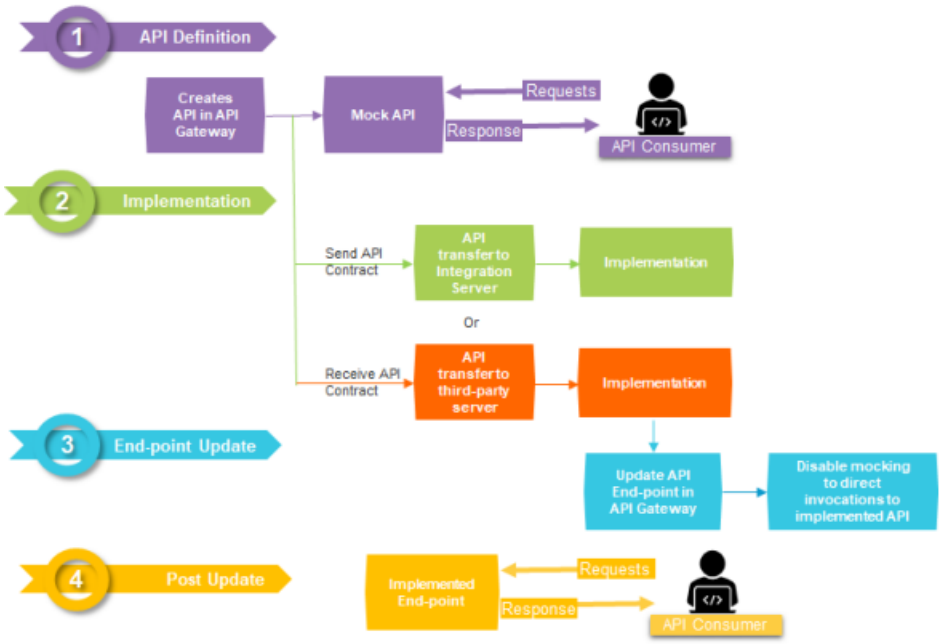
Using API Gateway, you can define API contract for the APIs and download provider specification for the APIs that you create. As a provider, you would not want to expose all resources and methods of an API to consumers. The API Contract given to the consumer has only the part of API exposed to the consumer whereas the provider specification contains the complete specification. This is useful for providers to implement the API.

You can enable mocking and activate the API for consumption. The mocked version of API returns respective responses for the consumer requests. This ensures the required end-user experience to the consumers.

When the API is ready to be implemented, you can implement the API in Integration Server or any other implementation server. If you are using Integration Server, you can add the required Integration Server instance in API Gateway. You can add multiple Integration Server instances and publish your API to the required instance. If you are using Integration Server, you can send the API contract from API Gateway. Else, the API contract has to be retrieved from API Gateway.

After implementation, you can update the actual implementation end-point to API Gateway. This step is mandatory to disable API mocking and divert the invocations to the actual end-point.

The following workflow shows the high-level workflow of API First implementation approach using API Gateway:



API First Implementation using Integration Server

This use case explains the steps involved in adapting API First from Integration Server. When an API created in API Gateway is implemented in Integration Server, then the API Contract is sent from API Gateway to Integration Server.

The use case starts when you create an API in API Gateway and ends when you communicate the API implementation endpoint to API Gateway.

In this example, the *APIFirst* API is created in API Gateway and implemented in the Integration Server instance, *IS1* that is configured in API Gateway.

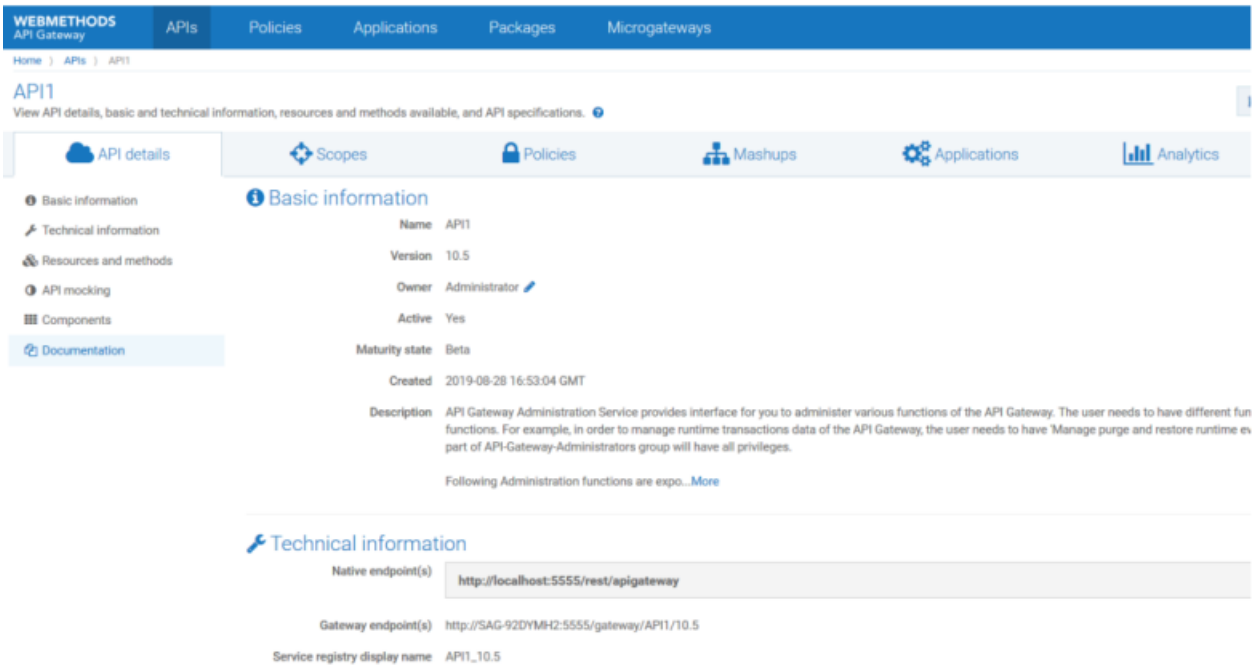
Before you begin

- Ensure that you have the Manage API privilege.
- Configure the required Integration Server instances in API Gateway for implementing your APIs. For details about configuring Integration Server instances, see *webMethods API Gateway Administration*.

To adapt API First design using Integration Server

1. Log on to API Gateway.
2. Click **APIs** in the title navigation bar.

   A list of all existing APIs appears.
3. Click **Create API** to create an API with required API documentation.

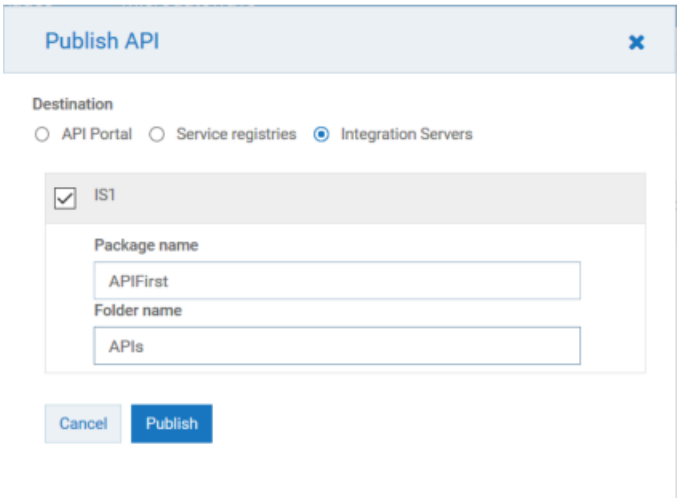


4. Click **Policies** and define required policies for the API.
5. Click **Enable Mocking** to mock and generate API mock responses.

   This step enables the API to send responses to the requests received from consumers.
6. From the APIs page, click **Publish** for the *APIFirst* API.

   The **Publish API** dialog box appears.
7. Select **Integration Servers**.

   The list of configured Integration Server instances appears.



8. Select the *IS1* instance from the list.

9. In the **Package Name** and **Folder Name** fields, provide the package name and folder name of the IS instance in which the API must be implemented.

   The API along with the API contract is published to Integration Server.

10. After implementing the API in Integration Server, invoke the REST end-point to communicate API implemented endpoint to API Gateway:

```
PUT http://<API Gateway host>:<port>/rest/apigateway/apis/{apiId}/implementation
{
  "maturityState": "string",
  "nativeBaseURLs": [
    "string"
  ]
}
```

   You can provide required values for the parameters in the above command. For information on parameters, see "List of Parameters used in API Implementation" on page 563.

   Example:

```
PUT http://10.2.151.149:5555/rest/apigateway/apis/
94dfd243-dd54-4d7e-8ba5-396ffaf6fe4e/implementation
{
"nativeBaseURLs":["https://10.2.35.125:5556/ws/srvs:Calculator/
CalculatorHttpSoap11Endpoint",
"http://10.2.151.149:5555/ws/srvs:Calculator/CalculatorHttpSoap11Endpoint"],
"maturityStatus" : "Implemented"
}
```

   For details about the REST API, see the swagger file APIGatewayServiceManagement.json, located at `SAG_Install_Directory`/IntegrationServer/instances/*default*/packages/WmAPIGateway/resources/apigatewayservices/APIGatewayServiceManagement.json. For more information about Service Management, see *webMethods API Gateway Developer's Guide*.

   As a result of the REST call, the mocking of the API is disabled and the consumers requests are directed to the actual implementation.

API First Implementation using a Third-party Server

This use case explains the steps involved in adapting API First approach using a third-party implementation server.

The use case starts when you create an API in API Gateway and ends when you communicate the API implementation endpoint to API Gateway.

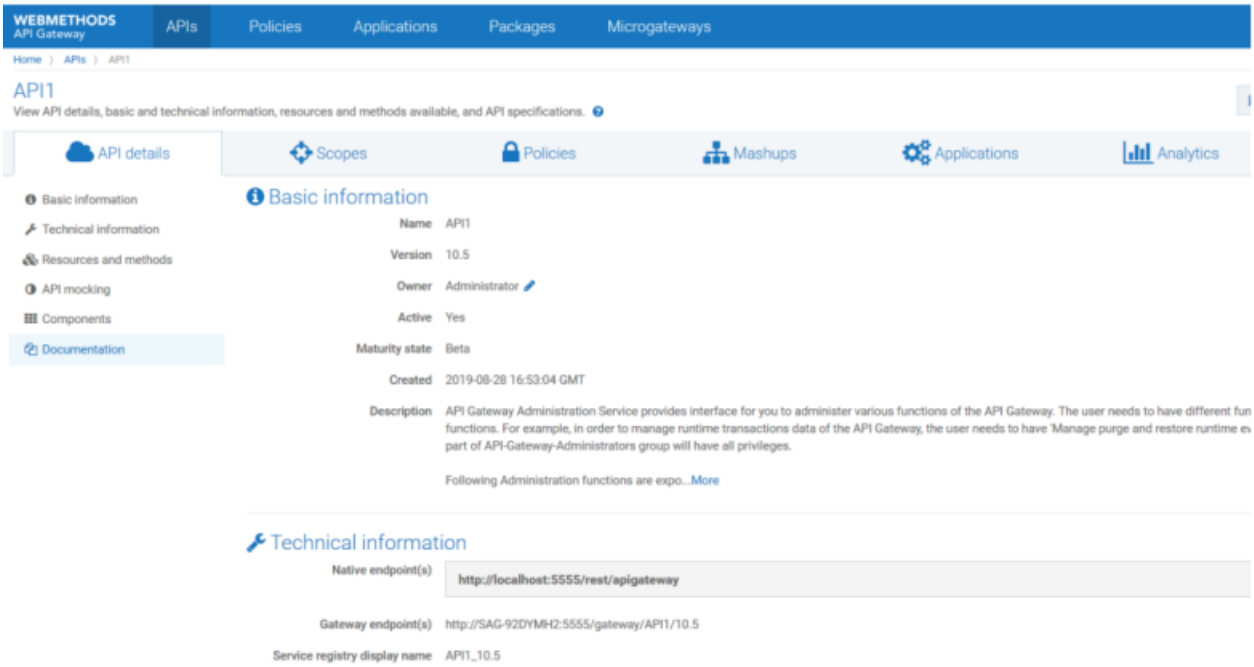
Before you begin

- Ensure that you have the Manage API privilege in API Gateway.
- Configure a third-party implementation server for implementing your APIs.

To adapt API First design using a third-party implementation server

1. Log on to API Gateway.
2. Click **APIs** in the title navigation bar.

   A list of all existing APIs appears.
3. Click **Create API** to create an API with required API documentation.



4. Click **Policies** and define required policies for the API.
5. Click **Enable Mocking** to mock and generate API mock responses.
6. Using an external REST client such as Postman or SoapUI, run the below command to search for the API in API Gateway for implementation:

```
POST http://<API Gateway host>:<port>/rest/apigateway/search
{
 "types" : ["api"],
  "scope" : [
                {
                    "attributeName" : "maturityState",
                    "keyword" : "ToBeImplemented"
                }
            ]
}
```

   The maturityState parameter in the above command is used search for APIs based on their maturity state. In this use case, you must search for APIs that are to be implemented. Hence,

you can provide the *ToBeImplemented* value for the parameter. This command returns the list of APIs that are yet to be implemented.

7. Using the API Id of the API that you want to implement, run the following command to retrieve the API contract from API Gateway:

```
GET http://<host>:<port>/rest/apigateway/apis/{apiId}/
providerspecification?format=swagger
```

The value for the format parameter can be *swagger*, *raml*, or *openapi* for REST APIs; and *wsdl* for SOAP APIs.

Note:
You can search for an API based on its maturity status in API Gateway using the following command:

```
POST http://<API Gateway host>:<port>/rest/apigateway/search
{
 "types" : ["api"],
  "scope" : [
                {
                    "attributeName" : "maturityState",
                    "keyword" : "ToBeImplemented"
                }
            ]
}
```

8. Implement the API in the required implementation server.

9. After implementation, invoke the REST end-point to communicate API implemented endpoint to API Gateway:

```
PUT http://<API Gateway host>:<port>/rest/apigateway/apis/{apiId}/implementation
{
  "maturityState": "string",
  "nativeBaseURLs": [
    "string"
  ]
}
```

You can provide required values for the parameters in the above command. For information on parameters, see "List of Parameters used in API Implementation" on page 563.

Example:

```
PUT http://10.2.151.149:5555/rest/apigateway/apis/
94dfd243-dd54-4d7e-8ba5-396ffaf6fe4e/implementation
{
"nativeBaseURLs":["https://10.2.35.125:5556/ws/srvs:Calculator/
CalculatorHttpSoap11Endpoint",
"http://10.2.151.149:5555/ws/srvs:Calculator/CalculatorHttpSoap11Endpoint"],
"maturityStatus" : "Implemented"
}
```

For details about the REST API, see the swagger file APIGatewayServiceManagement.json, located at `Install directory`/IntegrationServer/instances/default/packages/

WmAPIGateway/resources/apigatewayservices/APIGatewayServiceManagement.json. For more information about Service Management, see *webMethods API Gateway Developer's Guide*.

As an outcome of the REST call, the mocking of the API is disabled and API Gateway starts requests for the actual implementation.

List of Parameters used in API Implementation

The following are some of the parameters used during API implementation:

Parameter	Purpose
nativeBaseURLs	Endpoint URLs of the native service. This parameter is mandatory to route the requests to this implemented API. The existing endpoint values of the routing policies of the API are replaced with the URLs given against this parameter. You can provide multiple HTTP and HTTPS URLs for this parameter. The URLs that you provide for this parameter appears under the **Native endpoint(s)** section in the **Technical information** page of API Gateway. The first URL among the list of URLs is used in the routing policies by this update call. If you want to use any other URL in the routing policies, you can update the API policies accordingly.
maturityState	Indicates the maturity state of APIs. Use this parameter to search for an API based on its maturity state and retrieve the API for implementation. Also, you can use this to update the maturity state of an API after implementation. Typically, the value of this parameter would be the consecutive state defined in the **apiMaturityStatePossibleValues** extended setting configuration. **For example,** If any of the following states are configured in the **apiMaturityStatePossibleValues** setting : Design, Implementation, Testing, Production; and current state of an API is *Implementation*, then you must specify *Testing* as the parameter value because that would be next stage as per the configuration.

Troubleshooting Tips: Implement APIs

I see errors when API Gateway parses huge responses received from the native SOAP API

I see the following errors when API Gateway parses huge responses received from the native SOAP API:

com.fasterxml.aalto.WFCException: Unexpected end-of-input when trying to parse

com.fasterxml.aalto.WFCException: Unexpected end-of-input when trying to parse CHARACTERS at [row,col {unknown-source}]: [13621,577]

com.fasterxml.aalto.WFCException: 500 for SOAP APIs exchanging bigger payloads

Resolution:

To avoid encountering errors while parsing large responses from the native API, change the `enablesoapValidation` property in the `axis2.xml` file located at *`Install_dir`*`\IntegrationServer\instances\default\config\wss\` in one of the following ways:

- By commenting out the line

```
<!--<parameter name="enableSoapValidation">true</parameter> -->
```

- By setting the property `enablesoapValidation` to `false`

```
<parameter name="enableSoapValidation">false</parameter>
```

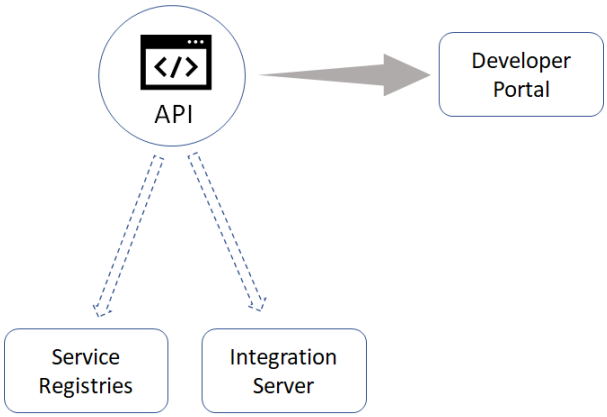
You must restart the API Gateway server for the change to take effect. The impact of this change is that the SOAP API request and responses are no more validated if they are compliant with the SOAP specification.

3 Publish APIs

Why Publish APIs?

After you build your APIs, you have to publish them to make the APIs available for consumption by developers and consumers. Publishing an API enables developers and consumers to consume the functionalities exposed through your API. They can then integrate the functionalities, the APIs provide, into applications easily. Creating and developing an API in API Gateway does not make it accessible to your users. You must first activate the API before publishing it to a portal so that the gateway endpoint is available for developers and consumers to invoke the API.

API Gateway allows you to publish APIs to Developer Portal from where they are available for consumption by developers and consumers.



Optionally, API Gateway also allows you to publish the APIs to the following destinations:

- Service registries. This enables applications to dynamically locate an API Gateway instance that can process that API.
- Integration Server. This is used in API first implementation approach.

In addition to publishing, API Gateway provides you with capabilities to customize your APIs before publishing them. You can customize APIs depending on how you want to restrict exposure of specific information the APIs handle as follows:

- Restrict the exposure of specific resources, methods, and operations of an API to other applications.
- Define a custom gateway endpoint by customizing the URL of the gateway endpoint. Consumers can use the customized URL to access the API.

The following sections describe how you can activate an API, customize the gateway endpoint, and publish APIs to different destinations.

Activating an API

You must first activate the API before publishing it to a portal so that the gateway endpoint is available for developers and consumers to invoke the API.

You must have the Activate/Deactivate APIs functional privilege assigned to perform this task. You can activate an API in the Manage APIs page. Alternatively you can also activate the API from the API Details page.

To activate an API

1. Click **APIs** in the title navigation bar.

   A list of available APIs appears.

2. Do one of the following:

   - Click the toggle button, in the corresponding column of the API to be activated, to change the status to  to activate the API.
   - Select the API to open the API details page. Click **Activate**.

3. Click **Yes** in the confirmation dialog box.

   The API is now activated. The Gateway endpoint is now available, which can be used by the consumers of this API. You can now publish the API to the required destination and expose the API for consumption by the consumers.

   You can modify API details or update the API, except the name and version of the API, when the API is in the active state. Active APIs are replaced during deployment with zero downtime and do not break ongoing requests. The updated APIs do not become effective for ongoing requests.

   Note:

   - If there is an error while saving after updating an active API, the API becomes inactive but the changes are saved.
   - Once the API is activated, you can define the custom gateway endpoints. For more information about gateway endpoints, see " Gateway Endpoints" on page 572.
   - Once the API is activated, you can enable the tracer. For more information about how to enable the tracer and view the tracing details, see "Debugging API" on page 519"Debugging API" on page 519.

WSDLs in API Gateway

When you activate a SOAP API in API-Gateway, the API exposes a link to the WSDL describing the API Gateway usage. The format of the link is as follows:

```
http://apigw-host:apigw-port/ws/<service-name>/1?wsdl
```

For example: http://myhost:5555/ws/Hello_Service/1?wsdl

If the WSDL imports more files, for example, sub WSDLs or XML schemas, then these files can be accessed through:

```
http://<apigw-host>:<apigw-port>/ws/<service-name>/1/<id>?xsd=<name>
```

For example: http://myhost:5555/ws/Hello_Service/1/53fe951a-2c04-4283-8b2d-8ee2957531b1?xsd=A

During this action, unlike all other parts of the WSDL, the <service> section is completely regenerated. For each activated HTTP or HTTPS port, depending on the API's transport policy, one or more endpoint entries are generated into the WSDL. By default, the following entries are present:

- Usual entry with service name and version number
- Mediator-compatible entry with service name and original port name
- One entry for each custom endpoint

Port names

The port names are numbered through the different entries to ensure uniqueness. Moreover, when the original port name has a *http* or *https* specifier at its end, then this specifier is taken over to the generated port name.

Examples

Example 1: With a single active HTTP port

```
<service name="Hello_Service">
        <port name="Hello_Port2" binding="tns:Hello_Binding">
            <soap:address location="http://myhost:5555/ws/Hello_Service/1"/>
        </port>
        <port name="Hello_Port" binding="tns:Hello_Binding">
          <soap:address location="http://myhost:5555/ws/Hello_Service.Hello_Port/1"/>
        </port>
    </service>
```

You can add custom endpoints to the API, for example, per UI. The customized values (prefix, servicename, version) appear in the <service> section.

Example 2: With an additional custom endpoint

```
<service name="Hello_Service">
        <port name="Hello_Port3" binding="tns:Hello_Binding">
            <soap:address location="http://myhost:5555/ws/Hello_Service/1"/>
        </port>
        <port name="Hello_Port2" binding="tns:Hello_Binding">
            <soap:address location="http://myhost:5555/myprefix/myservice/5"/>
        </port>
        <port name="Hello_Port" binding="tns:Hello_Binding">
          <soap:address location="http://myhost:5555/ws/Hello_Service.Hello_Port/1"/>
        </port>
    </service>
```

Example 3: With an additional HTTPS port that gets enabled and switched on in the API's transport policy

```
<service name="Hello_Service">
        <port name="Hello_Port" binding="tns:Hello_Binding">
          <soap:address location="http://myhost:5555/ws/Hello_Service.Hello_Port/1"/>
        </port>
        <port name="Hello_Port3" binding="tns:Hello_Binding">
          <soap:address location="https://myhost:5559/ws/Hello_Service.Hello_Port/1"/>
```

```
        </port>
        <port name="Hello_Port2" binding="tns:Hello_Binding">
            <soap:address location="http://myhost:5555/ws/Hello_Service/1"/>
        </port>
        <port name="Hello_Port4" binding="tns:Hello_Binding">
            <soap:address location="https://myhost:5559/ws/Hello_Service/1"/>
        </port>
        <port name="Hello_Port5" binding="tns:Hello_Binding">
            <soap:address location="http://myhost:5555/myprefix/myservice/5"/>
        </port>
        <port name="Hello_Port6" binding="tns:Hello_Binding">
            <soap:address location="https://myhost:5559/myprefix/myservice/5"/>
        </port>
    </service>
```

Parameters for refining the exposed port entries

Use the parameter **wsdlPortLayout** in **Administration** > **Extended settings** section to refine the exposed port entries. The parameter can have the following values:

- `service-port`

  All the port entries are exposed. This is the default value and the results as explained in the examples above apply.

- `service-only`

  Only one port (with servicename or version) is exposed. When this value is set, only the simple endpoint with the service name is generated. Example 3 would now look as follows:

```
<service name="Hello_Service">
        <port name="Hello_Port" binding="tns:Hello_Binding">
            <soap:address location="http://myhost:5555/ws/Hello_Service/1"/>
        </port>
        <port name="Hello_Port2" binding="tns:Hello_Binding">
            <soap:address location="https://myhost:5559/ws/Hello_Service/1"/>
        </port>
        <port name="Hello_Port3" binding="tns:Hello_Binding">
            <soap:address location="http://myhost:5555/myprefix/myservice/5"/>
        </port>
        <port name="Hello_Port4" binding="tns:Hello_Binding">
            <soap:address location="https://myhost:5559/myprefix/myservice/5"/>
        </port>
    </service>
```

- `mediator-comp`

  When this value is set, the entries generated are in mediator compatibility mode. Note that custom endpoints do not appear in this case. The entries look as follows:

```
<service name="Hello_Service">
       <port name="Hello_Portsoaphttp" binding="tns:Hello_Binding">
           <soap:address

location="http://myhost:5555/ws/Hello_Service.Hello_Portsoaphttp/1"/>
       </port>
       <port name="Hello_Portsoaphttps" binding="tns:Hello_Binding">
```

```
          <soap:address

location="https://myhost:5559/ws/Hello_Service.Hello_Portsoaphttps/1"/>
       </port>
   </service>
```

Deactivating an API

You can deactivate an API in the Manage APIs page. Alternatively you can also deactivate the API from the API Details page.

You must have the Activate/Deactivate APIs functional privilege assigned to perform this task.

To deactivate an API

1. Click **APIs** in the title navigation bar.

   A list of available APIs appears.

2. Click the toggle button, in the corresponding column of the API to be deactivated, to change the status to  to deactivate the API

3. Click **Yes** in the confirmation dialog box.

   The API is now deactivated. The API is no more available to be consumed by consumers.

Exposing a REST API to Applications

The API Provider can restrict the exposure of specific resources and methods of a REST API to other applications.

Consider you have a native REST API created in API Gateway with resources - Resource A, Resource B, and Resource C. You might want to expose Resource A and Resource C, and restrict the visibility of Resource B to other applications. You can use the **Expose to consumers** button to switch on the visibility of Resource A and Resource C and switch off the visibility of Resource B as required. Similarly, you can restrict the visibility of one or more methods within each individual resource.

If an application attempts to invoke the Resource C in the above REST API, API Gateway returns a HTTP response code 404.

By default, the **Expose to consumers** button is switched on for all resources and methods of the REST API. Once the API is activated, all of its resources and methods are exposed to registered applications. If you do not want a particular set of resources and methods, or a set of methods in a particular resource to be hidden for registered applications, switch off the **Expose to consumers** button in the REST API definition.

Note:

Be aware that API Gateway does not allow you to activate a REST API if none of the methods in the API are selected for exposing to other applications. You must select at least one method of the REST API to enforce runtime invocations.

To expose a set of resources and methods of the REST API

1. Click **APIs** in the title navigation bar.

   A list of APIs available in API Gateway appears.

2. Click the name of the required API.

   This opens the API details page.

3. Click **Edit**.

   If the API is active, API Gateway displays a warning message to let you know that the API is active.

4. Click Resources and methods.

   This displays a list of resources and methods available in the API.

   a. To select a resource, switch on the **Expose to consumers** button next to the resource URI.

      You can select one or more resources to expose to other applications.

   b. To select a method within the resource, click on the resource path. In the expanded list of methods, switch on the **Expose to consumers** button next to the method name.

      You can select one or more methods to expose to other applications.

5. Click **Save** to save the updated API.

   Activate the API, if it is not active, to put it into effect.

Exposing a SOAP API and GraphQL API to Applications

The API Provider can restrict the exposure of specific operations of a SOAP API and GraphQL API to other applications.

Consider you have a native SOAP API or GraphQL API created in API Gateway with operations - Operation A, Operation B, and Operation C. You might want to expose the Operation A and Operation C, and restrict the visibility of Operation B to other applications. You can use the **Expose to consumers** button to switch on the visibility of Operation A and Operation C and switch off the visibility of Operation B, as required.

If an application attempts to invoke the Operation B in the SOAP API or GraphQL API, API Gateway returns a HTTP response code 404 for SOAP API and response code 400 for GraphQL API.

By default, the **Expose to consumers** action is switched on for all operations of the SOAP API and GraphQL API. Once the API is activated, exposed operations are available for use in the registered applications. If you do not want a particular set of operations to be hidden for registered applications, switch off **Expose to consumers** in the SOAP API or GraphQL API definition.

Note:
API Gateway will not allow you to activate a SOAP API or GraphQL API if none of the operations in the API are selected for exposure to other applications. You must select at least one operation of the SOAP API or GraphQL API to enforce runtime invocations.

To expose a set of operations of the SOAP API or GraphQL API

1. Click **APIs** in the title navigation bar.

   This displays a list of APIs available in API Gateway.

2. Click the name of the required API.

   This opens the **API details** page.

3. Click **Edit**.

   If the API is active, API Gateway displays a warning message to let you know that the API is active.

4. Click **Operations**.

   This displays a list of operations available in the API.

   To select an operation, switch on the **Expose to consumers** action next to the operation URI. You can select one or more operations to expose to other applications.

5. Click **Save** to save the updated API.

   Activate the API, if it is not active, to put it into effect.

Gateway Endpoints

Gateway endpoint is the URL that is used to access an API through API Gateway. By default, API Gateway provides a default gateway endpoint for all active APIs. The default gateway endpoint is in the *protocol://host:port*/{defaultPrefix}/{apiName}/{apiVersion}/{resourcePath} format.

When there is a need to access the API using a different endpoint, you can define a custom gateway endpoint. In custom gateway endpoints, you can customize the portion of the URL between *port* and *resourcePath*.

Custom gateway endpoints can be specific to an API or a global template can be defined through global gateway endpoint.

How do I Define API-specific Gateway Endpoints?

This use case explains how to define custom gateway endpoints specific to an API. You can define more than one custom gateway endpoint to an API. Custom gateway endpoints can be added for all types of APIs such as REST, SOAP, OData, and WebSocket.

The use case starts when you want to define API specific gateway endpoint and ends when you have created the API specific gateway endpoint.

Here are some points that you need to consider, when you define API specific gateway endpoint:

- Custom gateway endpoints cannot be created for the APIs that have blank space or special characters in API name or API version.
- Gateway endpoint is case-sensitive.
- Gateway endpoint cannot start with pre-defined prefixes such as *rest* or *invoke* .
- URL path of one custom gateway endpoint cannot start with the URL path of the another custom gateway endpoint or default gateway endpoint. For example, if any of the API has a custom endpoint with URL path `abc/custom`, you cannot have another custom gateway endpoint with URL path `abc/customendpoint`. Similarly, if any of the API has a default gateway endpoint `gateway/myAPI/v1`, you cannot have custom endpoint with URL path `gateway/myAPI`. However, it is possible to have two valid custom gateway endpoints with URL paths `abc/custom1` and `abc/custom2`, because here one of the URL path is not the extension of another URL path.
- In order to use the gateway endpoints feature, the *watt.server.url.alias.partialMatching* property needs to be *true* . By default, this property is set to *true* .
- API Gateway internally creates the URL aliases, when you create a custom gateway endpoint. These internal URL aliases are hidden from the API Gateway users, and are displayed only in the Integration Server. Software AG recommends that you do not modify any URL alias through Integration Server.
- A gateway endpoint can use following variables, which are resolved dynamically:
  - ${defaultPrefix} - resolves based on API type. For REST and OData the defaultPrefix is gateway, SOAP the defaultPrefix is ws, and Websockets the defaultPrefix is websocket.
  - ${apiName} - replaces with the API name value.

    For example, when a gateway endpoint uses ${apiName} variable, and if you change the API name, it automatically gets reflected in the gateway endpoint.
  - ${apiVersion} - replaces with the API version value.

    Note:
    If you want to use a gateway endpoint across all versions of an API, Software AG recommends you to use the ${apiVersion} variable so that the gateway endpoint becomes unique across different versions.

Important:
At any given point, API Gateway does not allow you to provide the same gateway endpoint for different APIs nor different versions of same API. Hence, make sure that you provide an unique gateway endpoint, so that it does not match with any of the existing APIs' default or custom gateway endpoints.

Before you begin

Ensure that you have:

- *Manage APIs* functional privilege.
- Activated the API.

To define API-specific gateway endpoints

1. Click **APIs** in the title navigation bar.

   A list of all registered APIs appears.

   Note:
   You can manage the gateway endpoints of an API, directly from the view mode of the API details screen.

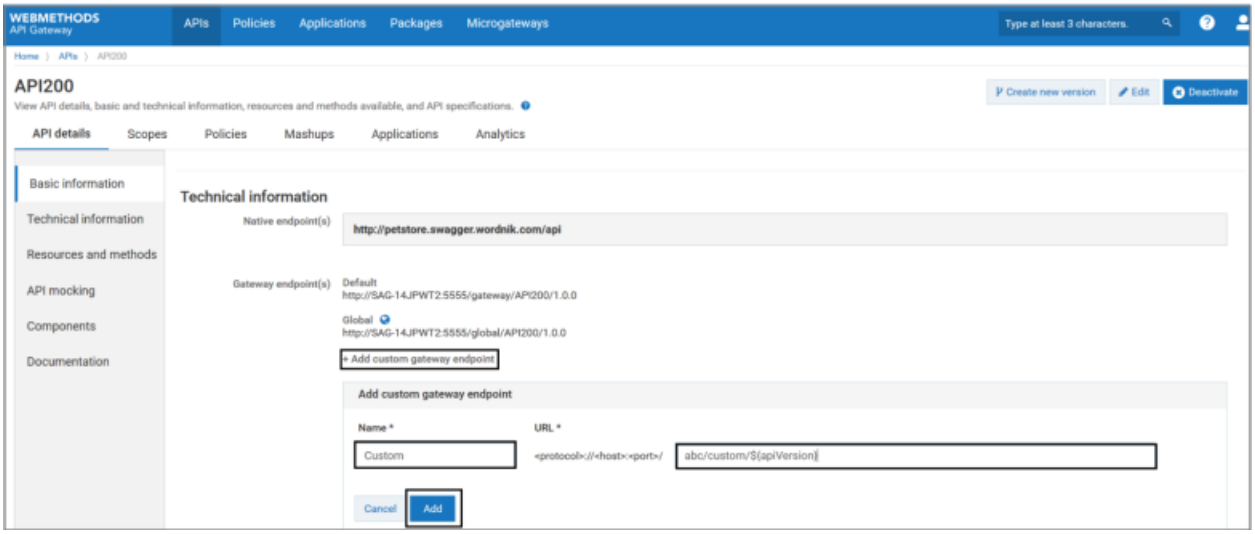
2. Click the corresponding API for which you want to customize the gateway endpoint.

   The API details page appears.

3. Click **Technical information**.

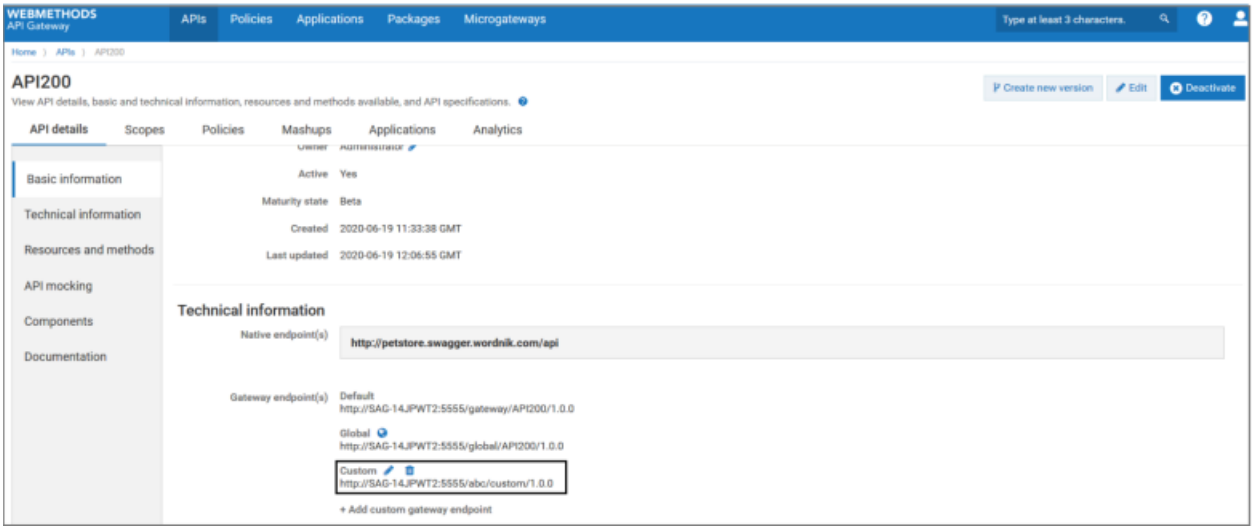
4. Click **+Add custom gateway endpoint** and provide the following information.

Field	Description
Name	Specifies the name for the custom gateway endpoint. A gateway endpoint name must be unique within an API.
URL	Specifies the custom gateway endpoint. The gateway endpoint URL cannot include a space, nor can it include the following special characters: # % ? ' " < \



5. Click **Save**.

   The added custom gateway endpoint appears in the **Gateway endpoint(s)** field of the API details page. In addition to the default gateway endpoint, you can access the API using this custom gateway endpoint.



Note:
You can edit or delete the gateway endpoint from API details page either by clicking the or icon corresponding to the gateway endpoint that you want to edit or delete.

How do I Define Global Gateway Endpoint?

This use case explains how to define global gateway endpoint. The global gateway endpoint creates gateway endpoint template for all APIs. Each API inherits this global endpoint in addition to the default and custom endpoints of an API.

The use case starts when you want to define global gateway endpoint and ends when you have created the global gateway endpoint.

Global gateway endpoint is not supported for the APIs that have blank space or special characters in API name or API version.

In order to generate a unique gateway endpoint for each API version, the global gateway endpoint template must use the following variables:

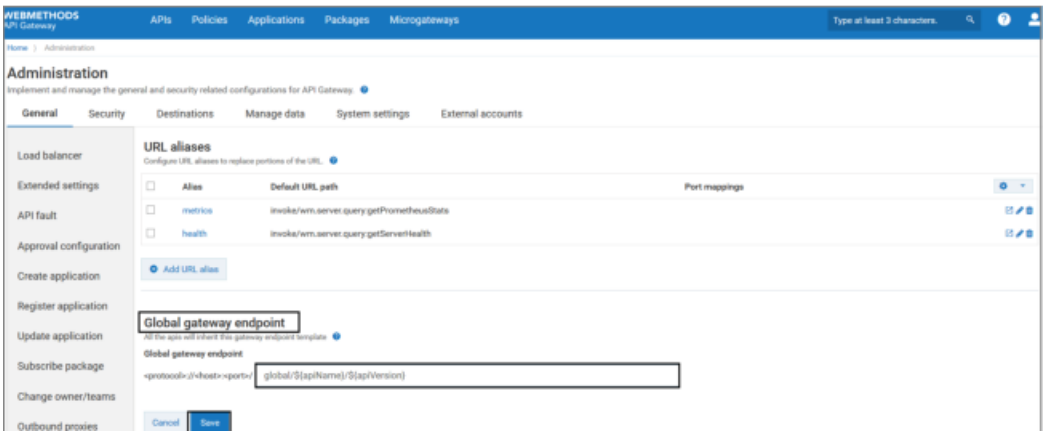
- ${apiName}
- ${apiVersion)

Before you begin

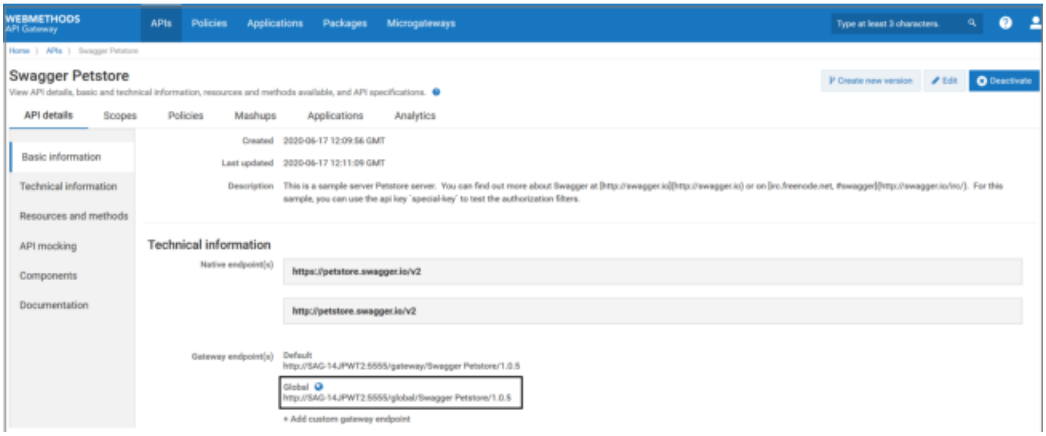
Ensure that you have *Manage APIs* functional privilege.

To define global gateway endpoint

1. Expand the menu options icon , in the title bar, and select **Administration**.
2. Select **General > URL aliases**.
3. In the **Global gateway endpoint** section, provide the global gateway endpoint that you want to define across the APIs.
4. Click **Save**.



The added global gateway endpoint appears in the **Gateway endpoint(s)** field of the API details page of all APIs. In addition to the default and API-specific gateway endpoints, you can access your APIs using this global gateway endpoint.



How do I Edit Global Gateway Endpoint?

This use case explains you how to edit the global gateway endpoint. You can edit the global gateway endpoint, when you want to change or update the existing global gateway endpoint template for all the APIs.

The use case starts when you want to edit global gateway endpoint and ends when you have updated the global gateway endpoint.

To edit global gateway endpoint

1. Expand the menu options icon , in the title bar, and select **Administration**.
2. Select **General > URL aliases**.
3. In the **Global gateway endpoint** section, update the value specified in the **Global gateway endpoint** field.
4. Click **Save**.

   The updated global gateway endpoint appears in the **Gateway endpoint(s)** field of the API details page. All the APIs can be accessed using the updated global gateway endpoint.

   Note:
   You cannot access the APIs using the older global gateway endpoint.

How do I Delete Global Gateway Endpoint?

This use case explains you how to delete the global gateway endpoint. You can delete the global gateway endpoint, when you do not want to access any of your APIs using the existing global gateway endpoint template.

The use case starts when you want to delete global gateway endpoint and ends when you have deleted the global gateway endpoint.

To delete global gateway endpoint

1. Expand the menu options icon  , in the title bar, and select **Administration**.
2. Select **General > URL aliases**.
3. In the **Global gateway endpoint** section, delete the value specified in the **Global gateway endpoint** field.
4. Click **Save**.

   The global gateway endpoint is removed from the **Gateway endpoint(s)** field of the API details page and you cannot access any of your APIs using global gateway endpoint.

Other Gateway Endpoint Usecases

Publishing APIs to API Portal

Just like publishing the default gateway endpoints, you can also publish the custom gateway endpoints to API Portal. Published custom gateway endpoints can be accessed through the API Portal interface.

Supporting Custom Prefix in CentraSite deployed APIs

When you virtualize a service in CentraSite, you can replace the default prefix of an invocation alias with custom prefix. When you publish such services to API Gateway, the custom prefix that was specified in CentraSite are supported in API Gateway by automatically adding the custom gateway endpoint to the respective API.

Publishing APIs to API Portal

Publishing an API to API Portal sends the SOAP and REST APIs to API Portal on which they are exposed for testing and user consumption.

Note:
API Gateway does not support publishing GraphQL API to API Portal.

The process of publishing an API to API Portal is initiated from API Gateway and is carried out on the API Portal server.

Doing this involves the following high-level steps:

- **Step 1**: You initiate the publish process by selecting the API to be published, specify the API endpoints to be visible to the consumers, and the API Portal communities in which the API is to be published.
- **Step 2**: API Gateway publishes the API to each of the specified API Portal communities.

- **Step 3**: During bulk publishing of APIs, the process continues even if API Gateway encounters a failure with API Portal.

When publishing an API to the API Portal destination, keep the following points in mind:

- The API Portal destination must be configured in API Gateway.
- You must have the Publish to API Portal functional privilege.
- You cannot publish an API if it is in inactive state. You have to activate the API before publishing it.

Publishing a Single API to API Portal

Pre-requisites:

You must have the Publish to API Portal functional privilege assigned to perform this task.

To publish an API to API Portal

1. Click **APIs** in the title navigation bar.

   A list of all APIs appears.

2. Click the **Publish** icon for the API that you want to publish. Alternatively, you can click the **Publish** button from the API details page of the required API.

3. Select the API endpoints that need to be visible to the consumers.

   At least one endpoint should be selected before publishing the API.

4. Select the API type if you want to publish a REST-enabled SOAP API.

   When the REST transformation is enabled for a SOAP API in API Gateway, you can publish the REST-enabled SOAP API to API Portal in one of the following ways:

   - **Publish as REST**. Default. The API is published as a REST API to API Portal. The REST resources and methods which correspond to the transformed SOAP operations are also published to API Portal.
   - **Publish as SOAP**. The API is published as a SOAP API with the SOAP operations to API Portal.
   - **Publish as REST and SOAP**. When both the options are selected, the API is published as a REST API as well as a SOAP API in API Portal and marked as a HYBRID API.

   Note:
   The **Publish as** option is available only if the REST transformation is enabled for the SOAP API.

5. Select the communities to which the API needs to be published.

By default, an API is published to the Public Community of API Portal.

Note:
If an API is already a part of the package published to a community then you cannot remove it from that community.

6. Click **Publish**.

   The API along with the selected endpoints is published to API Portal and available for the consumers to consume it.

   A REST-enabled SOAP API is published to API Portal based on the selected API type:

   - **REST API.** The API Details view displays the published API as a REST API with the defined REST resources and methods.
   - **SOAP API.** The API Details view displays the published API as a SOAP API with the defined SOAP operations.
   - **HYBRID API.** The API Details view, by default, displays the published API as a REST API with the REST resources and methods. There is an option **SOAP** that can be selected to display the published API as a SOAP API with the SOAP operations.

   Once an API is published, the **Publish** icon changes to **Republish** icon.

   You can unpublish an API once it is published by clicking the **Unpublish** icon.

Publishing Multiple APIs to API Portal in a Single Operation

Pre-requisites:

You must have the Publish to API Portal functional privilege assigned to perform this task.

You can bulk publish APIs to API Portal.

To publish multiple APIs to API Portal in a single operation

1. Click **APIs** in the title navigation bar.

   A list of all APIs appears.

2. Select the APIs that you want to publish.

   By default, all the respective API endpoints are internally selected to be visible to the consumers.

3. In the **Menu**  icon, click **Publish**.

4. Select the communities to which the APIs have to be published.

   By default, the APIs are published to the Public Community of API Portal.

5. Click **Publish**.

   The APIs along with their associated endpoints are published to API Portal and available for the consumers to consume.

   If you have selected several APIs where one or more of them are REST-enabled SOAP APIs in API Gateway, then these SOAP APIs are published as REST APIs along with their specific REST endpoints in API Portal.

6. Examine the **Publish APIs report** window and check for any errors that occurred during the publishing process.

   The **Publish APIs report** window displays the following information:

Field	Description
Name	The name of the published API.
Version	The version of the published API.
Status	The status of the publishing process. The available values are: ■ Success ■ Failure
Description	A descriptive information if the API publishing process fails or if a warning occurs.

   API Gateway writes these results to the **Audit logs** dashboard, so you can view them later.

7. Click **Download the detailed report here** to download the detailed report as an HTML file.

Unpublishing APIs from API Portal

After you publish an API to API Portal, the API remains published and available on API Portal for consumption until you manually unpublish the API.

You can unpublish a SOAP or REST API from API Portal to suspend its interaction, testing, and user consumption in API Portal.

Unpublishing a Single API from API Portal

Pre-requisites:

You must have the Publish to API Portal functional privilege assigned to perform this task.

To unpublish an API from API Portal

1. Click **APIs** in the title navigation bar.

A list of all APIs appears.

2. Click the **Unpublish** icon for the API that you want to unpublish. Alternatively, you can click the **Unpublish** button from the API details page of the required API.

   The **Unpublish API** dialog box is displayed.

3. Select **API Portal** in **Destination**.

4. Select **Unpublish**.

5. Click **Yes** in the confirmation dialog.

   The API is unpublished from the API Portal destination. The API is no longer available on API Portal for testing and user consumption.

   Once an API is unpublished, the **Republish** icon changes to **Publish** icon.

   You can publish an API once it is unpublished by clicking the **Publish** icon.

Unpublishing Multiple APIs from API Portal in a Single Operation

Pre-requisites:

You must have the Publish to API Portal functional privilege assigned to perform this task.

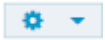
You can bulk unpublish APIs from API Portal.

To unpublish multiple APIs from API Portal in a single operation

1. Click **APIs** in the title navigation bar.

   A list of all APIs appears.

2. Select the APIs that you want to unpublish.

3. In the **Menu**  icon, click **Unpublish**.

4. Click **Yes** in the confirmation dialog.

   The selected APIs are unpublished from API Portal.

5. Examine the **Unpublish APIs report** window and check for any errors that occurred during the unpublishing process.

   The **Unpublish APIs report** window displays the following information:

Parameter	Description
Name	The name of the unpublished API.
Status	The status of the unpublishing process. The available values are: ■ Success ■ Failure
Description	A descriptive information if the API unpublishing process fails or if a warning occurs.

API Gateway writes these results to the **Audit logs** dashboard, so you can view them later.

6. Click **Download the detailed report here** to download the detailed report as an HTML file.

Publishing APIs to Service Registries

Publishing an API to a service registry enables applications to dynamically locate an API Gateway instance that can process that API.

When publishing an API to a service registry, keep the following points in mind:

- Before you publish an API to a service registry destination, you must add the service registry to the API Gateway instance from where you want to publish.
- You must have the **Publish API to service registry** functional privilege to publish APIs to a service registry.
- You can publish only active APIs. You cannot publish APIs that are in the inactive state.
- An API that is published to a service registry:
  - Is automatically de-registered from the service registry if the API is deactivated in API Gateway. When the API is activated again, it is automatically registered on the same service registry.
  - Is automatically de-registered from the service registry if the API Gateway instance from where it was registered goes down. When the API Gateway instance comes up again, the API is registered on the same service registry.
- In a cluster of API Gateway nodes, only the API Gateway instance from where you publish an API is added to the service registry. You have to separately publish the API from each API Gateway instance that the service registry can use for an API.

  Note:
  Similarly, you have to separately unpublish the API from each API Gateway instance from where you want to unpublish the API.

- If a load balancer has been configured for the API Gateway cluster, APIs from all instances are registered using the load balancer URL.

Publishing a Single API to Service Registries

Pre-requisites:

You must have the **Publish API to service registry** functional privilege assigned to perform this task.

To publish an API to service registries

1. Click **APIs** in the title navigation bar.

   The list of APIs defined in API Gateway appears.

2. Click the **Publish** icon for the API that you want to publish. Alternatively, you can click the Publish button from the API details page of the required API.

3. Select **Service Registries**.

   The list of service registries that have been added to API Gateway is displayed.

4. Select the service registry to which you want to publish the API.

   The list of endpoints in the selected API are displayed.

5. Select the endpoints that you want to publish to the selected service registry.

6. Repeat the previous two steps to publish the API to additional service registries.

7. Click **Publish**.

   Once an API is published, the **Publish** icon changes to **Republish** icon.

   You can unpublish a published API by clicking the **Unpublish** icon.

Publishing Multiple APIs to Service Registries in a Single Operation

Pre-requisites:

You must have the **Publish API to service registry** functional privilege assigned to perform this task.

Note:

When you publish multiple APIs to one or more service registries in a single operation, all endpoints in the APIs are published. To selectively publish endpoints within an API, you must publish the API separately as a single API.

To publish multiple APIs to service registries in a single operation

1. Click **APIs** in the title navigation bar.

   The list of APIs defined in API Gateway appears.

2. Select the APIs that you want to publish.

3. On the  menu, click **Publish**.

4. Select **Service Registries**.

   The list of service registries that have been added to API Gateway is displayed.

5. Select the service registry to which you want to publish the API and click **Publish**.

   Once an API is published, the **Publish** icon changes to **Republish** icon.

   You can unpublish a published API by clicking the **Unpublish** icon.

Unpublishing APIs from a Service Registry

You can manually unpublish APIs that you had previously published on service registries.

You must consider the following points before unpublishing an API from a service registry:

- You must have the **Publish API to service registry** functional privilege to unpublish APIs from a service registry.
- There is no option to unpublish individual endpoints. When you manually unpublish an API, all the endpoints in that API are unpublished from the selected service registries.
- As both—API publishing to service registries and API unpublishing to service registries—are specific to the current API Gateway instance, APIs are unpublished only for the API Gateway instance from where you unpublish. Therefore, if the same API was published from other instances of API Gateway, it continues to be available on the service registries from those API Gateway instances.

APIs may also get unpublished automatically from service registries, as described below.

Automatic Unpublishing of APIs

API Gateway automatically, but temporarily unpublishes an API in the following situations:

- When you deactivate an API after publishing it to a service registry.

Note:
When you reactivate the API, the temporarily unpublished endpoints are published again to the original service registries.

- When you disable or delete an API Gateway port that has endpoints that have been published to a service registry.

  Note:
  When you enable or add back the port again, the temporarily unpublished endpoints are published again to the original service registries.

Unpublishing a Single API from Service Registries

Pre-requisites:

You must have the **Publish API to service registry** functional privilege assigned to perform this task.

To unpublish an API from Service Registries

1. Click **APIs** in the title navigation bar.

   A list of all APIs appears.

2. Click the **Unpublish** icon for the API that you want to unpublish.

   The **Unpublish API** dialog box is displayed.

3. Select **Service registries** in **Destination**.

   The list of service registries to which the API was published is displayed.

4. Select the service registries from which you want to unpublish the API.

5. Select **Force unpublish** to mark the API as unpublished in API Gateway even if the unpublish fails on the selected service registries.

   The API is unpublished from the selected service registries. The API is no longer available on selected service registries for testing and user consumption.

   Once an API is unpublished, the **Republish** icon changes to **Publish** icon.

Unpublishing Multiple APIs from Service Registries in a Single Operation

Pre-requisites:

You must have the **Publish API to service registry** functional privilege assigned to perform this task.

You can bulk unpublish APIs from one or more service registries.

To unpublish multiple APIs from service registries in a single operation

1. Click **APIs** in the title navigation bar.

   A list of all APIs appears.

2. Select the APIs that you want to unpublish.

3. In the **Menu**  icon, click **Unpublish**.

4. Select **Service registries** in **Destination**.

   The list of service registries to which the APIs were published is displayed.

5. Select the service registries from which you want to unpublish the selected APIs.

6. Select **Force unpublish** to mark the APIs as unpublished in API Gateway even if the unpublish fails on the destination service registries.

7. Examine the **Unpublish APIs report** window and check for any errors that occurred during the unpublishing process.

   The **Unpublish APIs report** window displays the following information:

Parameter	Description
Name	The name of the unpublished API.
Status	The status of the unpublishing process. The available values are: ■ Success ■ Failure
Description	A descriptive information if the API unpublishing process fails or if a warning occurs.

   API Gateway writes these results to the **Audit logs** dashboard, so you can view them later.

8. Click **Download the detailed report here** to download the detailed report as an HTML file.

The APIs are unpublished from the selected service registries for the current API Gateway instance. Once an API is unpublished, the **Republish** icon changes to **Publish** icon.

Publishing APIs to Integration Server

Publishing an API to an Integration Server enables applications to dynamically locate an API Gateway instance that can process that API.

When publishing an API to an Integration Server instance, keep the following points in mind:

- Before you publish an API to an Integration Server destination, you must add the Integration Server instance to the API Gateway instance from where you want to publish.
- You must have the **Manage API** functional privilege to publish APIs to Integration Server.
- You can publish only active APIs. You cannot publish APIs that are in the inactive state.
- An API that is published to a Integration Server:
  - Is automatically de-registered from Integration Server if the API is deactivated in API Gateway. When the API is activated again, it is automatically registered on the same Integration Server.
  - Is automatically de-registered from Integration Server if the API Gateway instance from where it was registered goes down. When the API Gateway instance comes up again, the API is registered on the same Integration Server.
- In a cluster of API Gateway nodes, only the API Gateway instance from where you publish an API is added to the Integration Server. You have to separately publish the API from each API Gateway instance that the Integration Server can use for an API.

  Note:
  Similarly, you have to separately unpublish the API from each API Gateway instance from where you want to unpublish the API.
- If a load balancer has been configured for the API Gateway cluster, APIs from all instances are registered using the load balancer URL.

Publishing a Single API to Integration Server

Pre-requisites:

You must have the **Manage API** functional privilege assigned to perform this task.

To publish an API to Integration Server

1. Click **APIs** in the title navigation bar.

   The list of APIs defined in API Gateway appears.

2. Click the **Publish** icon for the API that you want to publish.

   Alternatively, you can click the Publish button from the API details page of the required API.

3. Select **Integration Servers**.

   The list of configured Integration Server instances appears.

4. Select the Integration Server to which you want to publish the API.

5. Provide the package name and folder name of the IS instance in which the API must be implemented

6. Repeat the previous two steps to publish the API to additional Integration Server instances.

7. Click **Publish**.

   Once an API is published, the **Publish** icon changes to **Republish** icon.

   You can unpublish a published API by clicking the **Unpublish** icon.

Publishing Multiple APIs to Integration Server in a Single Operation

Pre-requisites:

You must have the **Manage API** functional privilege assigned to perform this task.

To publish multiple APIs to Integration Server in a single operation

1. Click **APIs** in the title navigation bar.

   The list of APIs defined in API Gateway appears.

2. Select the APIs that you want to publish.

3. On the  menu, click **Publish**.

4. Select **Integration Servers**.

   The list of Integration Servers that have been added to API Gateway appears.

5. Select the Integration Server to which you want to publish the API and click **Publish**.

   Once an API is published, the **Publish** icon changes to **Republish** icon.

   You can unpublish a published API by clicking the **Unpublish** icon.

Unpublishing APIs from Integration Server

You can manually unpublish APIs that you had previously published on Integration Server.

You must consider the following points before unpublishing an API from an Integration Server instance:

- You must have the **Manage API** functional privilege to unpublish APIs from Integration Server.
- As both—API publishing to Integration Server and API unpublishing to Integration Server—are specific to the current API Gateway instance, APIs are unpublished only for the API Gateway instance from where you unpublish. Therefore, if the same API was published from other instances of API Gateway, it continues to be available on the Integration Server instance from those API Gateway instances.

APIs may also get unpublished automatically from Integration Server, as described below.

Automatic Unpublishing of APIs

API Gateway automatically, but temporarily unpublishes an API in the following situations:

- When you deactivate an API after publishing it to Integration Server.
- When you disable or delete an API Gateway port that has endpoints that have been published to Integration Server.

Unpublishing a Single API from Integration Server

Pre-requisites:

You must have the **Manage API** functional privilege assigned to perform this task.

To unpublish an API from Integration Server

1. Click **APIs** in the title navigation bar.

   A list of all APIs appears.

2. Click the **Unpublish** icon for the API that you want to unpublish.

   The **Unpublish API** dialog box is displayed.

3. Select **Integration Servers** in **Destination**.

   The list of Integration Servers to which the API was published is displayed.

4. Select the Integration Server from which you want to unpublish the API.

5. Select **Force unpublish** to mark the API as unpublished in API Gateway even if the unpublish fails on the selected Integration Server.

   The API is unpublished from the selected Integration Server. The API is no longer available on selected Integration Server for testing and user consumption.

   Once an API is unpublished, the **Republish** icon changes to **Publish** icon.

Unpublishing Multiple APIs from Integration Server in a Single Operation

Pre-requisites:

You must have the **Manage API** functional privilege assigned to perform this task.

You can bulk unpublish APIs from one or more Integration Servers.

To unpublish multiple APIs from Integration Server in a single operation

1. Click **APIs** in the title navigation bar.

   A list of all APIs appears.

2. Select the APIs that you want to unpublish.

3. In the **Menu**  icon, click **Unpublish**.

4. Select **Integration Servers** in **Destination**.

   The list of Integration Servers to which the APIs were published is displayed.

5. Select the Integration Server from which you want to unpublish the selected APIs.

6. Select **Force unpublish** to mark the APIs as unpublished in API Gateway even if the unpublish fails on the destination Integration Server.

7. Examine the **Unpublish APIs report** window and check for any errors that occurred during the unpublishing process.

   The **Unpublish APIs report** window displays the following information:

Parameter	Description
Name	The name of the unpublished API.
Status	The status of the unpublishing process. The available values are: ■ Success ■ Failure
Description	A descriptive information if the API unpublishing process fails or if a warning occurs.

   API Gateway writes these results to the **Audit logs** dashboard, so you can view them later.

8. Click **Download the detailed report here** to download the detailed report as an HTML file.

The APIs are unpublished from the selected Integration Server for the current API Gateway instance. Once an API is unpublished, the **Republish** icon changes to **Publish** icon.

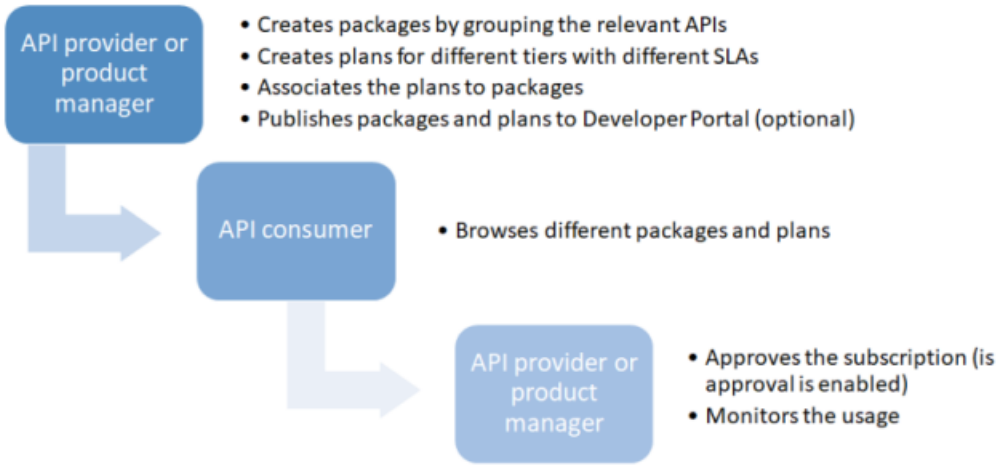
4 Monetize APIs

API Monetization

Once you create and configure your APIs in API Gateway, you can create a monetization strategy for your APIs.

API Gateway allows you to create packages and plans. As an API provider or API product manager, you can configure the packages and plans as per your organization requirements to monetize your APIs.

The general flow to monetize APIs is as follows:



When you add an API to a package for monetization, the **API key** authentication mechanism is automatically added to the IAM policy at API level. If the API already contains an IAM policy that has two authentication mechanisms with the **AND** condition, then the condition is switched to **OR**. This ensures that the monetization is supported when certain consumer applications access the API by just using the API key.

In a typical API monetization solution, you have the following components. You can create an end-to-end monetization experience by integrating these components.

- **API Gateway**. You can create APIs, packages and plans and host them in API Gateway. In addition, you can enforce quota and rate limits and monitor the API usage.

  Note:
  If you are configuring API consumption rate limits for any purpose other than monitoring, apply the Traffic optimization policy, instead of packaging. For information about the policy, see "Traffic Optimization" on page 316.

- **Billing solution**. You can translate the APIs, Packages, and Plans into billable products. You can perform customer information and billing related activities here, based on the usage quota.

- **API Portal**. Here consumers find and subscribe to the API packages and plans.

Note:
With API Gateway 10.11, you can create a subscription by using the Subscription REST API. For more details about this API, see *webMethods API Gateway Administration*.

API Gateway does not support a billing solution. However, it provides the following extensions, which you can use to integrate with a billing system.

- APIs to create, read, update, and delete the APIs, packages, plans, and subscriptions.
- Extensible model that enables extending meta data for packages, plans, and subscriptions to store additional (billing or consumer related) data.
- Auditing and lifecycle provides support to track the changes to assets. You can use the Search API to retrieve the audit data or you can configure the audit data to be pushed to different destinations as and when there is a change. For more details about Search API, see *webMethods API Gateway Administration*. For more details about custom destination, see *webMethods API Gateway Administration*.
- API Gateway monitors the usage and transactions. APIs are available to retrieve the monitoring and transactions data or you can configure API Gateway to push this data to different destinations. Alerts can be configured to be sent to different destinations for different metrics. To learn more, see *webMethods API Gateway Administration*.
- API Gateway provides APIs to retrieve the usage information for an API or a subscription. You can use this data to determine the quota usage and for billing purposes.

  Note:
  To view usage metrics, you must either add log invocation policy to each API or use global policy to generate transaction events.

On the API Gateway to API Portal integration, API Gateway provides support for publishing APIs, packages and plans to API Portal and also provides support for creating subscriptions from the API Portal. Additionally, API Gateway pushes API transactions to API Portal.

Packages and Plans

An API Package refers to a logical grouping of multiple APIs from a single API provider. A package can contain one or more APIs and an API can belong to more than one package. You must have the API Gateway 's manage packages and plans functional privilege assigned to manage API packages and plans.

An API Plan is the contract proposal presented to consumers who are about to subscribe to APIs. Plans are offered as tiered offerings with varying availability guarantees, SLAs or cost structures associated with them. An API package can be associated with multiple plans at a time. This helps the API providers in providing tiered access to their APIs to allow different service levels and pricing plans. Though you can edit or delete a plan that has subscribers, Software AG recommends you not to do so.

Note:
Package and plan subscriptions can be done only through API Portal.

You can create packages and plans, associate a plan with a package, associate APIs with a package, view the list of packages, package details, and APIs and plans associated with the package in the API Gateway user interface

Creating a Package

You must have the manage packages and plans functional privilege assigned to perform this task.

An API Package refers to a logical grouping of multiple APIs from a single API provider. A package can contain one or more APIs and an API can belong to more than one package. You can subscribe to a package from the API Portal or using the Subscription APIs.

You can create an API Package from the Manage packages and plans page.

To create an API Package

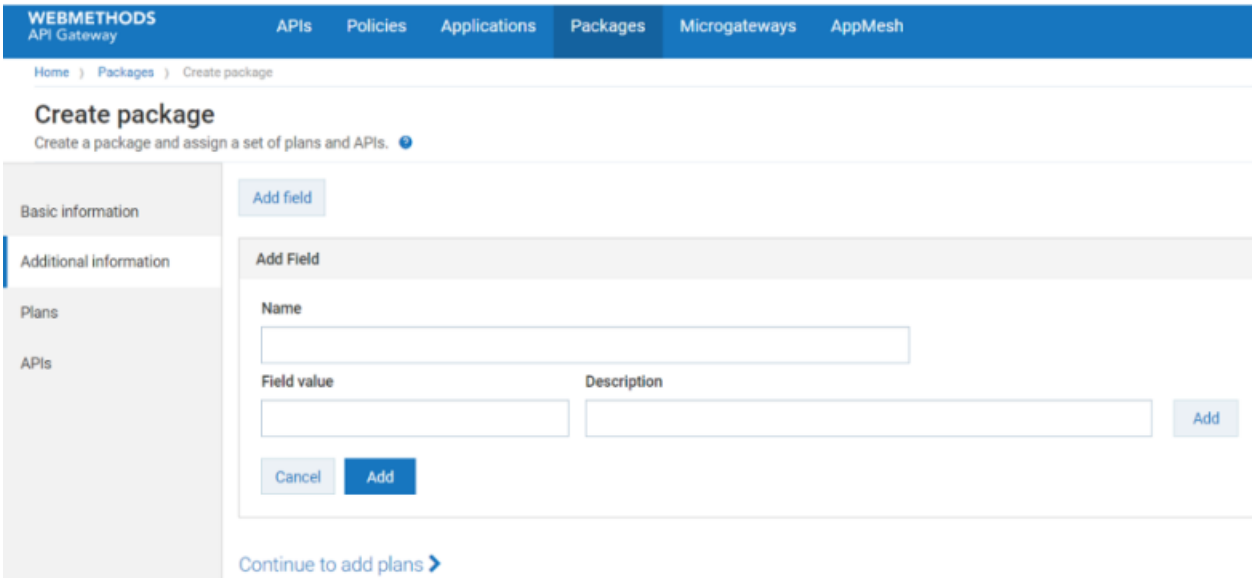
1. Click **Packages** in the title navigation bar.
2. Click **Create** in the Manage packages and plans section.
3. Select **Package**.
4. Click **Create**.
5. Provide the following information in the Basic information section:

Field	Description
Name	Name of the API package.
Version	Version assigned for the API package.
Team	Team to which the application must be assigned to. You can select more than one team. To remove a team, click the icon next to the team.
Description	A brief description for the API package.
Icon	An icon that is displayed for the API package. Click Browse and select the required image to be displayed as the icon for the API package. The icon size should not be more than 100 KB.

You can save the API package at this point and add the plans at a later time. The above fields are basic fields, provided by API Gateway.

6. Click **Additional information** to create custom fields for your package.

You can use these fields to extend meta data for Packages to store additional (billing/consumer related) data. For example, you can create an additional field called Category, which determines the category of a package. You can add drop-down values like gold, silver, and bronze. So you can now categorize packages as gold package, silver package, and so on.



7. Click **Add field** to create a new custom field.
8. (Optional) Click **Add** to add multiple custom fields.
9. Provide the following information:

Field	Description
Name	Name of the custom field.
Field Value	Value for the custom field.
Description	A brief description for the custom field.

10. Click **Save**.
11. Click **Plans** in the left navigation pane.
12. Select the plans that are to be associated with the API package.

    You can save the API package at this point and add APIs at a later time.
13. Click **Continue to add APIs**.

    Alternatively, click **APIs** in the left navigation pane.

When you add an API to a package for monetization, the **API key** authentication mechanism is automatically added to the IAM policy at API level. If the API already contains an IAM policy that has two authentication mechanisms with the **AND** condition, then the condition will be switched to **OR**. This ensures the monetization is supported when certain consumers access the API by just using the API key.

14. Type characters in the search box and click the search icon to search for the required APIs.

    A list of APIs that contain the characters specified in the search box appears.

15. Select the required APIs to be associated with the Package and click **+** to add them.

    You can delete the APIs from the package by clicking the **Delete** icon adjacent to the API in the API list.

16. Click **Save**.

Creating a Plan

You must have the API Gateway's manage packages and plans functional privilege assigned to perform this task.

An API Plan is the contract proposal presented to consumers who are about to subscribe to APIs. Plans are offered as tiered offerings with varying availability guarantees, SLAs or cost structures associated with them. An API package can be associated with multiple plans at a time. This helps the API providers in providing tiered access to their APIs to allow different service levels and pricing plans. Though you can edit or delete a plan that has subscribers, Software AG recommends you not to do so.

You can create packages and plans, associate a plan with a package, associate APIs with a package, view the list of packages, package details, and APIs and plans associated with the package in the API Gateway user interface.

You can create a plan from the Manage packages and plans page.

To create a plan

1. Click **Packages** in the title navigation bar.

2. Click **Create** in the Manage packages and plans section.

3. Select **Plan**.

4. Click **Create**.

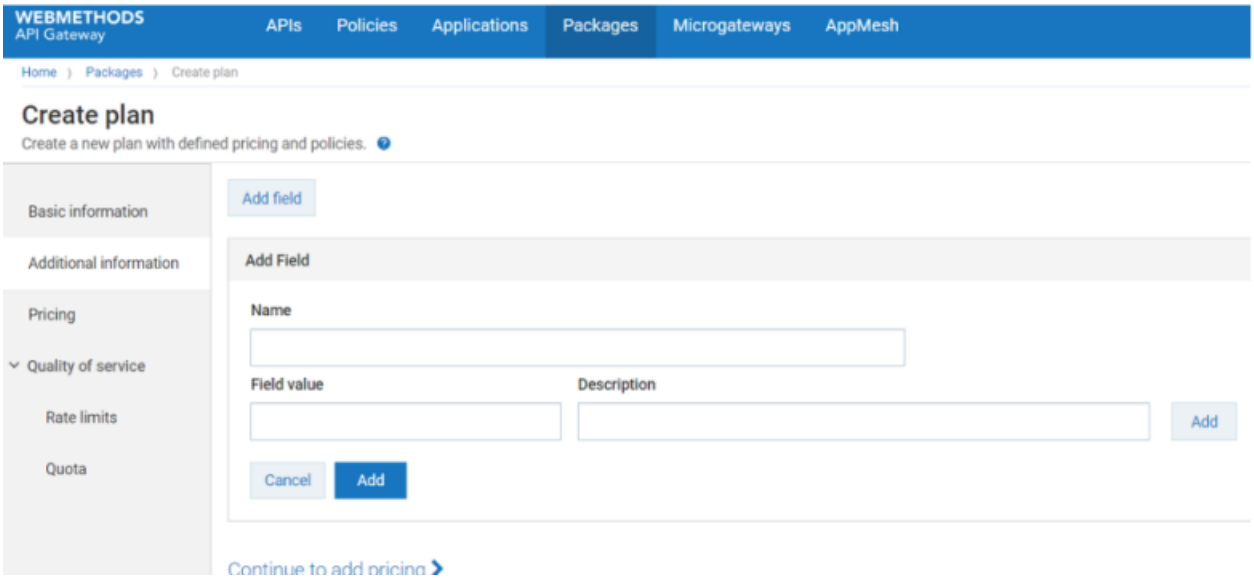
5. Provide the following information in the Basic information section:

Field	Description
Name	Name of the plan.
Version	Version assigned for the plan.
Team	Team to which the application must be assigned to. You can select more than one team. To remove a team, click the icon next to the team.
Description	A brief description for the plan.
Icon	An icon that is displayed for the plan. Click Browse and select the required image to be displayed as the icon for the plan. The icon size should not be more than 100 KB.

You can save the API package at this point and add the plans at a later time. The above fields are basic fields, provided by API Gateway. You can add additional information in the **Additional information** section.

6. Click **Additional information** to create custom fields for your plan.

   You can use these fields to extend meta data for Packages to store additional (billing/consumer related) data. For example, you can have a field called plan type. This field can have drop-down values called prepaid and postpaid. You can categorize all the plans as either prepaid or postpaid plans.



7. Click **Add field** to create a new custom field.

8. (Optional) Click **Add** to add multiple custom fields.

9. Provide the following information:

Field	Description
Name	Name of the custom field.
Field Value	Value for the custom field.
Description	A brief description for the custom field.

10. Click **Save**.

11. Click **Pricing** in the left navigation pane. .

12. Provide the following information in the Pricing section:

Field	Description
Cost	Specifies the cost for the plan.
Terms	Specifies the terms of conditions for the pricing.
License	Specifies the license information.

You can save the plan at this point and provide traffic optimization configurations at a later time.

13. Click **Continue to Quality of Service**.

Alternatively, click **Rate limits** in the left navigation pane.

Note:
If you are configuring API consumption rate limits for any purpose other than monitoring, apply the Traffic optimization policy, instead of packaging. For information about the policy, see "Traffic Optimization" on page 316.

14. Click **+ Add Rule**.

15. Provide the following information in the Create Rule section:

Field	Description
Maximum request count	Specifies the maximum number of requests handled. Value provided should be an integer.
Interval	Specifies the value for the interval for which the maximum request count is handled.

Field	Description
	Value provided should be an integer.
Interval unit	Specifies the unit of measurement of the time interval. For example: ■ **Minutes** ■ **Hours** ■ **Days** ■ **Calendar Week**. The plan starts on the first day of the week and ends on the last day of the week. By default, the start day of the week is set to Monday. For example: ■ If you subscribe to a package on a Wednesday and **Interval** is set to 1, the validity of the plan ends on Sunday, that is, 5 days. ■ If you subscribe to a package on a Wednesday and **Interval** is set to 2, the validity of the plan still ends on Sunday, but the validity of the plan is two calendar weeks, that is 12 days. You can change the start day of the week using the extended setting `startDayOfTheWeek` in the **Administration** > **General** > **Extended settings** section. Restart the API Gateway server for the changes to take effect. ■ **Calendar Month**. Starts on the first day of the month and ends on the last day of the month. For example: ■ If you subscribe to a package in the month of August and **Interval** is set to 1, the validity of the plan ends on the last day of August. ■ If you subscribe to a package in the month of August and **Interval** is set to 2, the validity of the plan ends in two calendar months, that is on the last day of September.
Alert frequency	Specifies how frequently to send alerts to API Gateway destination when the **Rate limits** condition is violated. Select one of the options: ■ **Only Once**. Triggers an alert only the first time one of the specified conditions is violated.

Field	Description
	■ **Every Time**. Triggers an alert every time one of the specified conditions is violated.
Violation message	Specifies the text that appears when the rule is violated.

16. Click **Ok**.

    This creates the rule and displays it in the Configured rules table. Click **+ Add rule** to add more rules. You can edit or delete the rules by clicking the **Edit** and the **Delete** icons respectively.

    At a later time, when this plan is applied to an API through a package, the rules that you configured for this plan are enforced on the applied API.

17. Click **Quota** and provide the following information in the Quota settings section.

Field	Description
Maximum request quota	Specifies the maximum number of requests handled. Value provided should be an integer.
Block on breach	When selected, it specifies that the access to the API is blocked when there is a rule violation. Also, a notification is sent to API Gateway destination depending on the **Alert frequency**. By default, this option is not selected.
Interval	Specifies the value for the interval for which the maximum request quota is handled. Value provided should be an integer.
Interval unit	Specifies the unit of measurement of the time interval. For example: ■ **Minutes** ■ **Hours** ■ **Days** ■ **Calendar Week**. The plan starts on the first day of the week and ends on the last day of the week. By default, the start day of the week is set to Monday. For example: ■ If you subscribe to a package on a Wednesday and **Interval** is set to 1, the validity of the plan ends on Sunday, that is, 5 days.

Field	Description
	■ If you subscribe to a package on a Wednesday and **Interval** is set to 2, the validity of the plan still ends on Sunday, but the validity of the plan is two calendar weeks, that is 12 days. You can change the start day of the week using the extended setting `startDayOfTheWeek` in the **Administration** > **General** > **Extended settings** section. Restart the API Gateway server for the changes to take effect. ■ **Calendar Month**. Starts on the first day of the month and ends on the last day of the month. For example: ■ If you subscribe to a package in the month of August and **Interval** is set to 1, the validity of the plan ends on the last day of August. ■ If you subscribe to a package in the month of August and **Interval** is set to 2, the validity of the plan ends in two calendar months, that is on the last day of September.
Alert frequency	Specifies how frequently to send alerts to API Gateway destination when the **Quota** condition is violated. Select one of the options: ■ **Only Once**. Triggers an alert only the first time one of the specified conditions is violated. ■ **Every Time**. Triggers an alert every time one of the specified conditions is violated.
Violation message	Specifies the text that displays when the policy is violated.
Notification settings	Specifies whether notifications are to be sent on rule violations. Enable the toggle button to enable the notifications and provide the following information: ■ **Notify after (in %)**. Provide a value which is a number. A notification is sent to the configured email IDs once the total request count reaches the % value as provided in the maximum quota value. ■ **Violation message**. Provide the content of the mail that is sent to the configured email Ids once the quota request count reaches the limit specified.

Field	Description
	■ **Email Ids**. Provide an email Id of the recipient to which notifications have to be sent once the quota request count reaches the limit specified. Click +Add to add multiple recipients. **Note:** The SMTP settings under **Administrator settings** > **Destinations** has to be provided for an email to be sent. ■ **Send Digital Events** ■ **Custom destination**. Select custom destinations to which the notification must be sent. You can select multiple custom destinations. The custom destinations displayed in this field are populated from the custom destinations, configured in the **Administration > Destinations > Custom destinations** page.

Important:
Incase of a server crash or restart, the quota status is determined by the value set in the **pgmen.quotaSurvival.addLostIntervals** property and works as follows:

- If the property is set to false, remaining time in quota is retained even after a restart or crash. For example, If quota is of 60 minutes and 7 minutes was used before the server crash or restart, then quota remaining time of 53 minutes is retained after server crash or restart, if the property is set to false.
- If the property is set to true, and if the sum of time between server shutdown and restart and quota elapsed time does not exceed the interval of the subscription, the quota usage value is retained. In this case the remaining quota time is calculated as {current interval cycle - (elapsed time + (start time - shutdown time))}. For example, if the current subscription duration is 1 month and if the server starts on the 10th day of the cycle and restarts on the 12th day of the cycle, the remaining quota time is calculated as {30 - (10 + (12-10))} = 18 days.
- If the property is set to true, and if the sum of time between server shutdown and restart and quota elapsed time exceeds the interval of the subscription, a new interval is created. Quota usage value is not retained in this case.

18. Click **Save**.

The plan is created and listed in the list of plans.

Activating a Package

You must have the API Gateway's activate/deactivate packages assigned to perform this task.

You can activate a package so that a consumer can try out APIs in the package with the package level token. When the consumer requests a token from API Portal, the request is processed in API Gateway and a token is sent back to API Portal. This token is visible to the consumer on the Access

Token page. The consumer can test the APIs in the package with this token on the API Try out page.

To activate a package

1. Click **Packages** in the title navigation bar.

   A list of all packages appears with their status as **Inactive** or **Active**.

2. Click the activation toggle button for the package.

   The package is now activated.

   Alternatively you can click **Activate** on the Packages details page to activate the package.

Publishing a Package

You must have the API Gateway's publish to API Portal functional privilege assigned to perform this task.

You can publish a package to the configured destination, for example API Portal. Once the package is published, the APIs associated with the package are available to consumers. The package level token is applicable to all APIs associated with the package. The consumers do not have to request an access token for individual APIs to consume them.

Ensure the following before publishing a package:

- A destination is configured.
- The package is active.
- The package has at least one plan and API associated with it.
- The APIs associated with the package is published to the destination.

To publish a package

1. Click **Packages** in the title navigation bar.

   A list of all packages appears.

2. Click the **Publish** icon for the package that has to be published.

3. Select the communities to which the package needs to be published.

   By default, a package is published to the Public Community of API Portal.

   Note:
   The list of communities displayed are those that are common to which the APIs associated with this package are already published to.

4. Click **Publish**.

   A success messages is displayed when the package is successfully published. The package is now published to the destination, for example API Portal, that is configured and is available on API Portal to consumers.

You can unpublish a package once it is published by clicking the **Unpublish** icon for the required package.

Viewing List of Packages and Package Details

You can view the list of packages in the Packages section of the Manage packages and plans page from where you can create, delete, and select a package to view its details.

To view the package list and package details

1. Click **Packages** in the title navigation bar.

   A list of all packages appears. You can perform various operations like activating a package, publishing or unpublishing a package, and deleting a package.

2. Select a package.

   The basic information, and the associated plans and APIs for the selected package appears in the package details page.

Viewing List of Plans and Plan Details

You can view the list of plans in the Plans section of the Manage packages and plans page from where you can create, delete, and select a plan to view its details.

To view the plan list and plan details

1. Click **Packages** in the title navigation bar.

2. Click **Plans**.

   A list of all plans appears. You can delete a plan by clicking the **Delete** icon for the respective plan.

3. Select a plan.

   The basic information, the pricing, and Quality of service associated with the selected plan appears in the plan details page.

Viewing a List of Subscriptions

In the **Manage packages and plans** page, the **Subscriptions** tab lists the applications and the associated package name, plan, used quota, start time, end time, and the remaining period of the subscription. The **Subscriptions** tab lists only the packages and plans that are subscribed from API Portal.

In the **Subscription** tab, you can also search for the subscriptions by name, package name, and plan name.

Modifying a Package

You must have the API Gateway's manage packages and plans assigned to perform this task.

You can modify the basic information, include or exclude plans and APIs of the package. You can modify a package when it is either in active or inactive state. If you modify a package when it is in the active state, the following points are applicable:

- If you remove an API from the package, subscribers cannot leverage the service of that API.
- If you add an API to a package, subscribers can leverage the service of the API without performing any setup.
- If a package's plan has active subscribers, you cannot remove that plan from the package.

To modify a package

1. Click **Packages** in the title navigation bar.

   A list of all packages appears.

2. Select a package.

   The basic information, and the associated plans and APIs for the selected package appear on the package details page.

3. Click **Edit**.

   The package details appear.

   Note:
   The **Edit** option is available only if the package is in inactive state.

4. You can modify the information related to the package, as required, in the Basic information section.

5. Click **Plans** in case you want to modify the plans associated with the package.

   A list of plans associated with the package and list of available plans appears.

6. You can do the following:

   - Add more plans to the package by selecting plans listed in the available plans list.
   - Delete the plans from the package by clearing the check box of the plan associated with the package.

7. Click **APIs** in case you want to modify the APIs associated with the package.

   A list of APIs associated with the package and a search box to search for APIs that need to be added to the package appear.

8. You can do one of the following:

   - Add more APIs to the package. You can search for APIs using the search box and click **+** adjacent to the API to add it
   - Delete the APIs from the package by clicking the **Delete** icon adjacent to the API in the APIs list.

9. Click **Save**.

   This saves the modified package.

Modifying a Plan

You must have the API Gateway's manage packages and plans functional privilege assigned to perform this task.

You can modify a plan to change the pricing details and Quality of service associated with the plan. You can modify a plan when the package associated with the plan is active or inactive. If you modify a plan when it is in the active state, the following points are applicable:

- The quota usage data is not reset for the existing customers. However, you can explicitly reset or modify the quota usage. If you modify the quota usage, a new cycle is initiated for all the subscribers.
- If you modify the Rate limits or pricing, it does not impact the quota usage.

To modify a plan

1. Click **Packages** in the title navigation bar.

   A list of all packages appears.

2. Click **Plans**.

   A list of all plans appears.

3. Select a plan.

The plan details page displays the basic information, pricing details, and the Quality of service associated with the plan.

4. Click **Edit**.

   The plan details appear with fields that you can edit.

5. You can modify the information related to the plan, as required, in the Basic information section.

6. Click **Pricing** in case you want to modify the pricing model associated with the plan.

7. Modify the pricing plan as required.

8. Click **Rate limits** if you want to modify the rules associated with the plan.

   A list of rules associated with the plan appears.

9. You can do one of the following:

   - Add more rules to the plan. Click **Add rule** to create and add rules to the plan.
   - Modify the already configured rule. Click the **Edit** icon for the rule listed in the **Configured rules** list and modify the details as required.
   - Delete rules from the plan. Click the **Delete** icon adjacent to the rule in the **Configured rules** list.

10. Click **Quota settings** if you want to modify the quota settings for the plan.

11. Modify the quota settings as required.

12. Click **Save**.

    This saves the modified plan.

Deleting a Package

You must have the API Gateway's manage packages and plans assigned to perform this task.

You can delete a package from the Package list that appears on the Manage packages and plans page. You can not delete a package if it is in active state. You have to deactivate it before deleting it.

To delete a package

1. Click **Packages** in the title navigation bar.

   A list of all packages appears.

2. Click the **Delete** icon for the package that has to be deleted.

3. Click **Yes** in the confirmation dialog.

Deleting a Plan

You must have the API Gateway's manage packages and plans functional privilege assigned to perform this task.

You can delete a plan from the Plans list that appears in the Plans section of the Manage packages and plans page. You can delete a plan only if it is not associated with a package. You have to disassociate the plan with the package before deleting it.

To delete a plan

1. Click **Packages** in the title navigation bar.

2. Click **Plans**.

   A list of plans appears.

3. Click the **Delete** icon for the plan that has to be deleted.

4. Click **Yes** in the confirmation dialog.

5 Monitor APIs

Analytics

API Gateway provides monitoring capabilities to monitor API Gateway and API usage by collecting and analyzing data about the availability and performance of an API. This helps in identifying problems that impact users. In addition to monitoring the performance of APIs, you may also want to get an insight into how developers are using the published APIs. This data provides a better understanding of any improvements that might be required to enhance the API usage or performance.

The analytics dashboard in the API Gateway UI displays a variety of charts to provide an overview of API Gateway performance and its API usage. The data for these dashboards come from the API Gateway destination store. The dashboard has various filters that you can apply depending on what you want to monitor. API Gateway also provides the capability to create a custom dashboard.

- **API Gateway dashboard**. Displays API Gateway-wide analytics such as Summary of APIs, API usage, API trends, the top performing API and the non-performing API analytics, audit logs, applications and package related event information. Click > **Analytics** to access API Gateway-wide analytics. For details about the API Gateway dashboard, see *webMethods API Gateway Administration*.
- **API-specific dashboard**. Displays API specific analytics such as API invocation trends by response time, success and failure rates, API performance, consumer or application traffic for a specific API. This can be accessed from the API details page.
- **Custom dashboards**. Displays API Gateway-wide analytics or API specific analytics as configured. Click > **Analytics** to access API Gateway-wide analytics. A custom dashboard is a collection of visualizations. You can add the visualizations as per your requirement and compile the visualizations as a custom dashboard. For details about creating custom dashboard, see *webMethods API Gateway Administration*.

The dashboard view depends on the events and metrics generated in API Gateway and their types. An event is a kind of notification or alert generated by the API Gateway Metrics and Event Notification module. Various types of events are generated based on the behavior of the transactions in the system. Events generated by API Gateway are real-time events made persistent in the store and sent to configured destinations.

API-specific Dashboard

You can view the API-specific dashboard by navigating to the API details page and click **Analytics**.

Select the API-specific dashboard from the drop-down list. The dashboard displays the following analytics based on the metrics monitored.

To filter the API-specific analytics, select the time interval using the options:

- **Quick select**. Specify the time interval. Click **Apply** to filter the analytics based on the time interval.

- **Commonly used**. Select a commonly used time interval, and the filter is applied automatically. To view the API-specific analytics between a time interval, click **Custom range** > **From Date** > **To Date** > **Apply**.
- **Recently used**. Select a recently used time interval, and the filter is applied automatically.

When you log in and view the analytics, the last used time interval is saved for each dashboard. When you view the dashboard again, the last used time interval for that dashboard is applied. The last used time interval is valid for the current session only.

For the specified time interval, you can also filter based on an API. The API drop-down list displays all the APIs. On selecting an API, the data displayed is for the selected API.

You can click on the specific event in the list under Legend to view the specific event in any of the widgets. You can view additional details for an event by hovering the cursor over a particular color in the graphical representations.

Metric	Description
Events over time	Displays the trending of events generated by the selected API over time.
API invocations	Displays the number of times the API was invoked during the specified time.
API invocations - Status wise	Displays the number of successful API invocations and failed API invocations during the specified time. API invocations is the sum of successful API invocations and failed API invocations.
API invocation pattern	Displays API invocation over period of time during the specified time interval in the form of a line graph.
Native service performance	Displays information on how fast the native service responds to the request received in the specified time based on the data in the transactional event.
Gateway vs Provider time	Displays the comparison between `gatewayTime` performance and `providerTime` performance.
Response code trend	Displays the trend based on the response codes received from various events for the API during the specified time.
API trend by response	Displays the trending of the selected API based on the response time from the performance metrics for that API.
Success vs Failure	Displays the trending of API based on its success rate as compared to its failure rate in the performance metrics for the specified time.
Runtime events	Displays the run time event details for the selected API. Displays information on the event type, date when the event

Metric	Description
	was created, the agent on which the event was generated, description of the alert generated, the source of event, and the application that generated the event.
Service result cache	Displays a bar graph showing the number of responses served from cache and the number of responses fetched from the native service at the operation level for the selected API during the specified time. The **Service result cache** metric graphical representation is not supported for GraphQL API.
Method level invocations	Displays the method level invocations per operation for the API during the specified time. You can hover the cursor over the stacked bar chart to view the various methods invoked per operation or resource and also the operations or resources for the selected API during the specified time. The **Service result cache** metric graphical representation is not supported for GraphQL API.

6 Microservices

Manage Microservices

In comparison to a monolithic architecture, where all processes are tightly coupled and run as a single service, a microservices architecture structures an application as a collection of services that are loosely coupled and independently deployable. The microservices architecture enables the rapid, frequent, and reliable delivery of large, complex applications. Each component service in a microservices architecture can be developed, deployed, operated, and scaled without affecting the functioning of other services.

The adoption of the microservices architecture pattern drives the need for lightweight gateways or microgateways. webMethods Microgateway gives control over a microservice landscape by enforcing policies that perform authentication, traffic monitoring, and traffic management. The lightweight nature of a Microgateway allows a flexible deployment to avoid gaps or bottlenecks in policy enforcement.

Businesses are adopting microservices for agility and scalability. In managing the complexity of distributed microservices environments, the microservices architecture might run into operational challenges, such as service discovery, connectivity, security, and fault tolerance. This is where a service mesh helps in providing critical capabilities that provide a solution to the operational challenges you face. webMethods AppMesh is just such a solution that allows you to apply an application context to service mesh or microservice deployments.

The following sections describe how API Gateway facilitates management of deployed Microgateways and configuration of the AppMesh infratructure to manage microservices.

Microgateways

API Gateway enables you to monitor the Microgateways that are connected to it. You can view the active APIs and detailed analytics for each Microgateway that is connected to the API Gateway.

The **Microgateways management** page displays all Microgateway groups that are connected to the API Gateway. A Microgateway group enables you to group Microgateways that have some common element, such as domain (finance or human resources) or type of APIs (external-facing or internal use). For each Microgateway group, the **Microgateways management** page displays the following information:

- Name and Description of the Microgateway group.
- The number of Microgateways that are part of the group.
- The number of APIs that are available in that group.

You can perform the following operation on this page:

- Click **View details** to view more information about a Microgateway group.
- Click **Analytics** to view the **Analytics** tab of a Microgateway group. For information about Microgateway Analytics, see " Microgateway Group Analytics" on page 618.

Note:

For information about installing, configuring, and using Microgateways, see the *webMethods Microgateway User's Guide*.

Microgateway Groups

A Microgateway group is a collection of Microgateway instances that are grouped based on a common domain or API type. The **Microgateway groups** page displays the available groups and the Microgateways that are included in a particular group. The page displays the following information for each Microgateway group:

- **Basic information** section includes
  - Name of the group
  - Description
  - Number of APIs in the group
- **Microgateways** section includes the following details of each Microgateway instance in the group:
  - Host name
  - HTTP and HTTPS ports that the Microgateway uses to expose the APIs that are provision on it
  - A description of the Microgateway
  - The number of APIs available on the Microgateway

To add a Microgateway to the group, you have to add the following information to the `custom-settings.yml` file:

```
microgatewayPool:
    microgatewayPoolName: poolNameHere
    microgatewayPoolDescription: poolDescriptionHere
```

Where *poolNameHere* is the name of the group and *poolDescriptionHere* is an optional description of the group. If a *poolNameHere* is not provided, the Microgateway is added to the **Default** group.

Note:
For more information about `custom-settings.yml`, see the *webMethods Microgateway User's Guide*.

You can perform the following operations on this page:

- Click  to delete a Microgateway from API Gateway. You can also delete multiple Microgateways.
- Click the Microgateway name to view more information about it.

Microgateway Group Analytics

The Microgateway group **Analytics** tab displays detailed analytics based on the data cumulatively received from the Microgateways in a group. This tab displays the following information:

- **Overall events**. Displays a pie chart that lists different events being monitored. Each of these event categories is depicted with different colors.
- **Application Activity**. Displays a pie chart to indicate activities based on applications. You can view the number of APIs that are authorized with applications and the number of APIs that are not authorized using any applications.
- **API Invocation**. Displays a pie chart to indicate the number of invocations made to each API present in the group.
- **Runtime events**. Displays the run time event details such as time when the event was generated, API Name, the application that generated the event, event type, description of the alert generated due to the event, status, and the source of event.
- **Payload size**. Displays the payload size of the request and responses during data transfer in the specified time. This data is picked up from the transactional event that is triggered when a log invocation policy is applied to the API.

You can perform the following operations on this page:

- **Apply filters**. The **Analytics** tab provides filters that you can use to view selective data or events. You can use the displayed duration filter and add a custom filter using the filter query builder.
  - To apply a duration filter, select the time interval from the drop-down list, and click **Apply filter** to filter the analytics based on the time interval chosen. To specify a custom duration, select **Custom** from the drop-down list, select the required **From Date** and **To Date** values, and click **Apply filter**.
  - You can also add filters based on a filter query. To add a filter based on a filter query, click **Add a filter**, select the desired field, operator and value, and click **Save**.
- **View specific events**. Click on the specific event in the list under Legend to view the specific event in any of the widgets. You can view additional details for an event by hovering the cursor over a particular color in the graphical representations.

Microgateway Details

The Microgateway details page provides information about a particular Microgateway.

The **Microgateway Info** tab includes two sections:

- The **Basic information** section provides information about the Microgateway.

- The **APIs** tab section provides the information of the APIs provisioned on that Microgateway. Clicking an API opens the **API details** page. The active Microgateway endpoints of the API are also displayed in the **API details** page.

Note:
All the Service Registries to which a Microgateway is publishing an API must be configured in API Gateway.

You can perform the following operations on this page:

- Click an API to view the API details.
- Click **Analytics** to view detailed analytics based on the data received from an individual Microgateway. The tab includes the following analytic graphs:
  - Overall events
  - Application activity
  - Runtime events
  - Payload size
- Similar to the Microgateway Group Analytics tab, you can apply required filters and view specific events. For more information about the widgets and instructions to view graphs, see " Microgateway Group Analytics" on page 618.

Deleting Microgateway Instances

When you stop a Microgateway instance, the instance is deleted from API Gateway automatically. But, if a Microgateway stops abruptly, the corresponding instance remains stale in API Gateway. You can remove such stale instances by deleting them.

Important:
When deleting Microgateways, ensure that they are not in *Running* status. Deleting an instance removes it completely from API Gateway. For information on checking the status of a Microgateway, see *Creating API Gateway Asset Archives using the Command Line* section in the *Microgateway User's Guide*.

You can delete one Microgateway or multiple instances from a Microgateway group at the same time.

Deleting a Microgateway

To delete a Microgateway

1. Click **Microgateways** in the title navigation bar.
2. Click the required Microgateway group.

The Microgateway group details appears.

3. From the **Microgateways** section, click  next to the required Microgateway.

   A warning message appears.

4. Click **Yes** to delete.

Deleting Multiple Microgateways

To delete multiple Microgateways

1. Click **Microgateways** in the title navigation bar.
2. Click the required Microgateway group.

   The Microgateway group details appears.

3. From the **Microgateways** section, select the Microgateways that you want to delete by selecting the check boxes next to the required host names.
4. Click  and select **Delete** from the drop-down list.

   A warning message appears.

5. Click **Yes** to delete the selected Microgateways.

   The selected Microgateways are deleted and the **Delete Microgateways report** appears.

6. Click **Download the delete report here** to download the report.

   The report displays the following details of the deleted Microgateways.

   - Host name
   - HTTP or HTTPS port name
   - Status

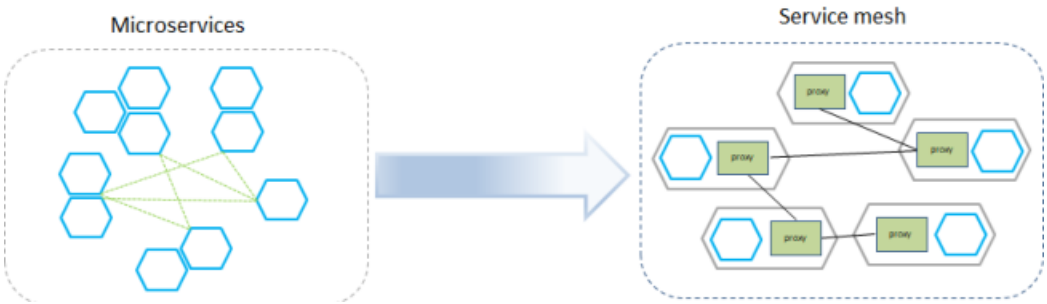
AppMesh Support

Businesses are adopting microservices for agility and scalability. In managing the complexity of distributed microservices environments, the microservices-based architecture might run into operational challenges, such as service discovery, connectivity, security, and fault tolerance. This is where a service mesh helps in providing critical capabilities that provide a solution for the operational challenges you face. For example, the collaboration of services within a microservice architecture requires the exchange of requests. In case a service is overloaded by requests, the

service mesh reroutes the requests to address the overload situation for optimizing the services to work together.

As an application develops, new services are added; this complicates the communication network, increases chances of failure, and adds to the complexity of finding where the problem occurred. A service mesh makes handling the complex network easier as it captures the service-to-service communication details. In a service mesh, the requests between the microservices are routed through the proxies in its own infrastructure layer.

The figure below depicts the microservice environment on the left and the microservice with the service mesh infrastructure on the right. The microservices have individual proxies deployed alongside each service, in a separate container. The service-to-service communication is routed through these proxies.



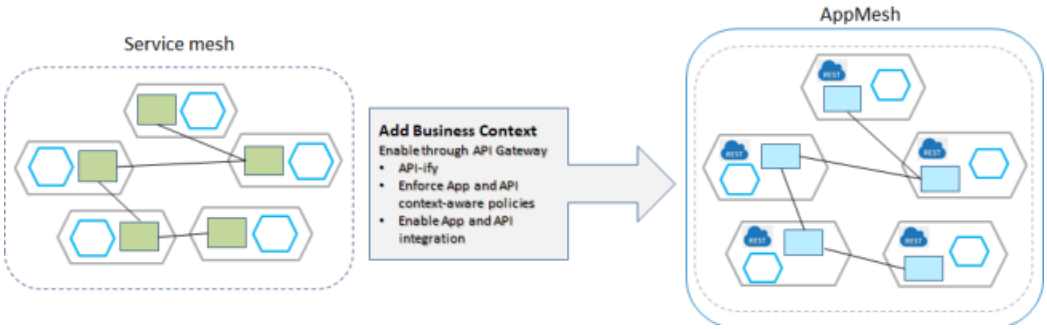
Since the service mesh is built into the application, it helps in fast and easy communication amongst the services with less downtime as the application grows in size.

Why AppMesh?

Though the service mesh helps in managing a complex landscape of microservices, there is a limitation when it comes to application awareness. It is difficult to achieve application-level enforcement on the requests before they reach the microservices. webMethods AppMesh provides the required solution of applying an application context to a service mesh or microservice deployments.

webMethods AppMesh extends the service mesh platform by providing application awareness through the *APIfy* action on the microservices, where it provides an API face to the microservices. This enables the reuse, governance, consumption, landscape management capabilities, and drives the API-led integration of microservices.

The figure below depicts an AppMesh deployment, wherein a business context is added to the service mesh through API Gateway. Each service is *APIfied* and has a Microgateway injected as a sidecar.



Features and Benefits

AppMesh allows your organizations to manage microservices-led applications to:

- **Gain better control**. Group and manage microservices as business applications. Create, manage, and deliver new applications quickly.
- **Govern applications centrally**. Add context to your microservices and API landscape.
- **Deliver without disruption**. Enhance your application without making changes to existing services.

In detail, AppMesh provides several critical functions, which include:

- Discovering services.
- Creating, managing, and delivering new applications quickly.
- Applying business rules to drive application-specific behavior.
- Deep visibility into how the application is running and who is using it.

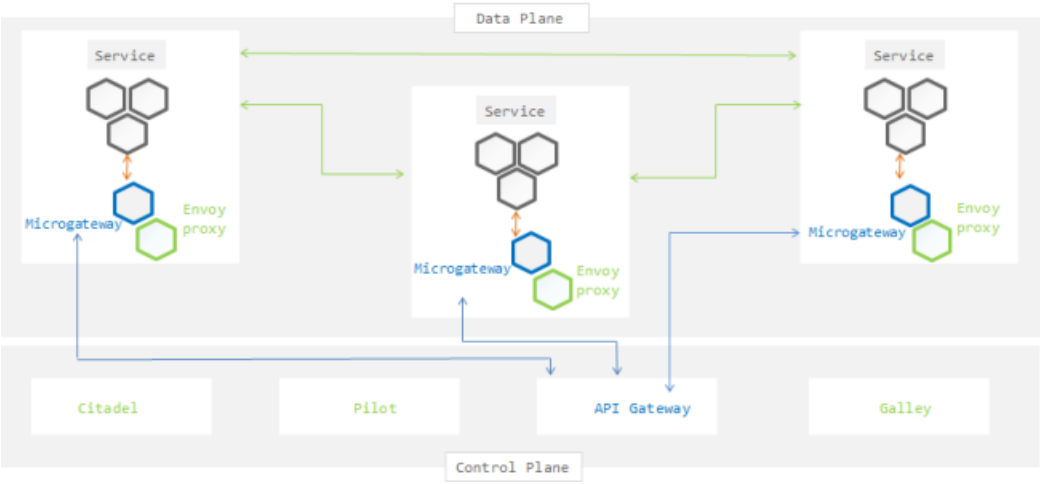
Istio-based AppMesh

Istio is an open source service mesh platform that provides a way to control how microservices share data with one another and is designed to run in a variety of environments; Kubernetes being one of them. Istio support is added to a service by deploying an Envoy proxy that sits alongside a service and routes requests to and from other proxies.

API Gateway provides the capability to discover services in a Kubernetes-based Istio deployment. It allows to *APIfy* a service, and deploy it back to the Kubernetes environment. After deploying the services back into the Kubernetes environment, you can provision the APIs and service updates from the API Gateway user interface, through the Microgateway injected as a sidecar for the service in the Kubernetes pod.

The APIs that AppMesh creates as a result of the *APIfy* action are directly linked to and are hosted by Microgateway, and are used to enforce the policies to the service-to-service communication, which is called the East-West traffic.

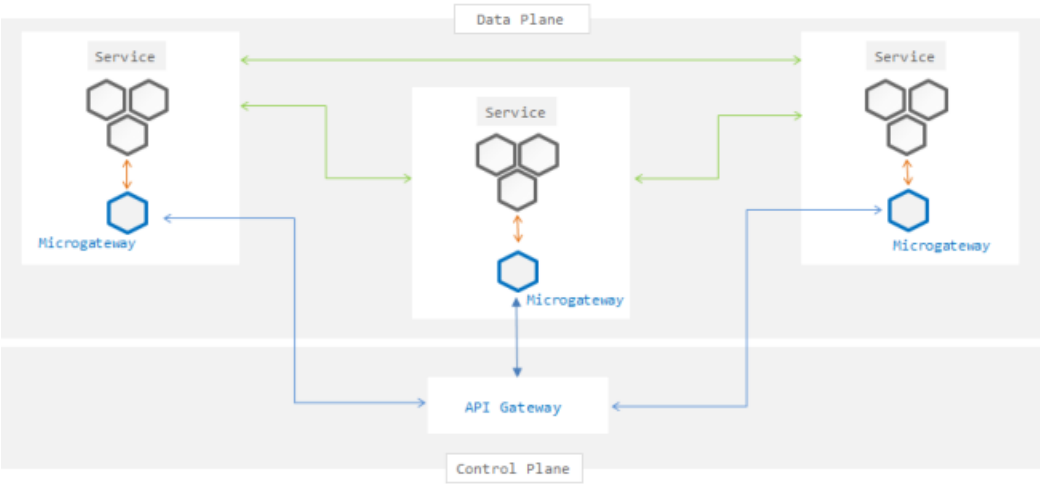
The figure below depicts the Istio-based AppMesh architecture where the communication between the pods is through the envoy proxy and the Microgateway injected into the pod communicates with the API Gateway.



Kubernetes-based AppMesh

You can deploy AppMesh in a Kubernetes environment even without a service mesh or Istio deployment. API Gateway provides the capability to discover services, *APIfy* a service, and deploy it back to the Kubernetes environment. On deploying the AppMesh, the Microgateway is injected as a sidecar for the service in the Kubernetes pods. The Microgateway injected into the pod acts as a proxy for the services for the inter-service communication. You can now provision the APIs and service updates, from the API Gateway user interface, through the Microgateway injected as a sidecar for the service in the Kubernetes pod.

The figure below depicts the Kubernetes-based AppMesh architecture without a service mesh deployment. The Microgateway injected into the pod as a sidecar communicates with API Gateway and any updates to the APIs or policies enforced on the services are provisioned through the Microgateway into the Kubernetes pod.



Supported Platforms

webMethods AppMesh supports the following platforms:

- Istio on Azure Kubernetes
- Istio on Kubernetes
- Istio on Rancher

API Gateway supports Kubernetes versions 1.9 to 1.17.0, and Istio versions 1.5 and 1.6.

AppMesh Licensing

The API Gateway license is extended with the AppMesh feature.

You can configure the AppMesh feature license in the **Administration** > **General** > **License** > **Configuration** section. For details about configuring API Gateway license, see *webMethods API Gateway Administration*.

You can view the AppMesh license details in the **Administration** > **General** > **License** > **Details** section. For details about viewing license details, see *webMethods API Gateway Administration*.

If the API Gateway license does not contain the AppMesh feature support, the following functions are not available in an API Gateway instance:

- AppMesh tab in the API Gateway user interface.
- Service mesh configuration section under **Administration** > **External accounts** in the API Gateway user interface.

Configure API Gateway to Connect to a Service Mesh Environment

To discover the services and deploy AppMesh, you must configure API Gateway to connect to the service mesh environment where the services reside.

This configuration section is visible in the API Gateway user interface if you have the required AppMesh license and the *Manage general administration configurations* privileges.

Before you begin

Ensure that you have:

- Manage general administration configurations privileges.
- Valid AppMesh license.
- Kubernetes client configuration file and its location to set up the connection between API Gateway and the service mesh.

For details about Kubernetes in general and the Kubernetes client configuration file, see *Kubernetes documentation*.

- Valid namespaces in Kubernetes with or without the Istio service mesh environment setup.
- Docker image for Microgateway pushed to a registry that is reachable by your Kubernetes environment.

  For details about how to create the Microgateway image, see "Creating a Microgateway Image" on page 626.

To configure API Gateway to connect to a service mesh environment

1. Click  and select **Administration**.
2. Click **External accounts** > **Service mesh**.
3. Click **Browse**, select and upload the required Kubernetes client configuration file.

   On successful upload of the file, the cluster name and the cluster endpoint details appear in the table.

   API Gateway supports a single Kubernetes cluster and context. If multiple cluster or contexts exist, AppMesh uses the context present in the current context field of the Kubernetes client configuration file.
4. Provide the following details required for AppMesh configuration:

Field	Description
API Gateway URL	*Optional*. Specify the API Gateway URL of the API Gateway instance. This is required to set up the communication channel between API Gateway and the Microgateway that is injected into the Kubernetes pod. If you do not configure the API Gateway URL, the default value is picked up from the Load balancer URLs that are configured under **Administration** > **Load balancer** in the following precedence: a. First of the HTTPS Load balancer URL, if configured. b. First of the HTTP Load balancer URL, if configured. c. Default host name with 5555 as the default port.
API Gateway username	The username of the API Gateway instance.

Field	Description
API Gateway password	The password of the API Gateway instance.
Microgateway image	Specify the location of the Microgateway image to deploy Microgateway as a sidecar in the Kubernetes pod.
Microgateway port	Specify the port the Microgateway listens on.
Namespaces	Specify the Kubernetes namespace to monitor using AppMesh. You can add multiple namespaces. If you do not provide any namespace, then the default namespace `default`, that is present in Kubernetes environment is picked up.

5. Click **Save configuration**.

   The service mesh environment is configured, and the communication between API Gateway and the service mesh is enabled.

You can now proceed with discovery of services and deploying AppMesh.

Creating a Microgateway Image

The Microgateway image is required to deploy the Microgateway as a sidecar in the Kubernetes pods. The Microgateway has to be present in the registry repository for it to be available for deployment as a sidecar into a Kubernetes pod.

1. Run the following commands to build the required Microgateway image:

```
./microgateway.sh createDockerFile --docker_dir . -p 9090
docker build -t your-repo:mcgw-app-mesh -f Microgateway_DockerFile
docker push your-repo:mcgw-app-mesh
```

The Microgateway image can now be used to inject Microgateway as a sidecar in the Kubernetes pods while deploying AppMesh.

AppMesh Deployment

This section describes how microservices are discovered and deployed in AppMesh.

Before you begin

You must have a Kubernetes environment with or without service mesh configured, and an AppMesh environment set up in API Gateway.

Stages in AppMesh deployment

1. “Service Discovery” on page 627

2. "APIfy" on page 628
3. "Update API Definition and Policies" on page 628
4. "Deploy AppMesh" on page 629

Service Discovery

AppMesh uses the Kubernetes REST API to search for services or deployments from the Kubernetes environment for the configured namespaces.

You can view all the discovered services in the API Gateway user interface in the **AppMesh** tab.

A list of microservices created in the Kubernetes environment as deployments, present in the configured AppMesh namespaces appears.

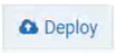
To view the service details, click **View details**. The service details page displays the following information:

Service Details	Components
Basic information	■ **Service name**. Name of the microservice. ■ **Namespace**. Name of the namespace added in the microservice. ■ **Internal endpoints**. These are the native endpoints of the microservice, which is present in the routing policy of the API, that are only reachable within the cluster. These endpoints are created in the Kubernetes environment as services. ■ **External endpoints**. These are the service endpoints that are used by the external client to invoke an API.
Deployment details	■ **Deployment configuration**. Provides the YAML deployment configuration. ■ **Deployment flow**. Provides the microservice pod traffic details.
Service mesh sidecar	Provides the following service mesh proxy details: ■ Virtual services that are associated with the service ■ Destination rules ■ Authorization policies ■ Envoy filters **Note:** For more information about service mesh proxy details, see https://istio.io/latest/docs.
Microgateway sidecar	■ **API**. Provides a link to the API details page.

Service Details	Components
	■ **Microgateway**. Provides a link to Microgateway groups.

You can perform the following actions in the service details page:

- **APIfy** 

- **Deploy** 

- **Undeploy** 

APIfy

APIfy is the process of giving an API face to the Kubernetes service. APIfy creates an empty API with the endpoint that API Gateway receives from the service.

Click **APIfy** to APIfy a microservice. An API is created for the microservice in API Gateway and you can access it using the **APIs** tab or the **API** link in the **Microgateway sidecar** section of the service details page.

The API created by default has a single resource with resource path (/) and the routing endpoint is the first internal endpoint of the service.

Note:
Only one API can be created for a microservice.

Update API Definition and Policies

The API created after you APIfy a microservice, may need updates to the API definition and API policies (if any). You can update the API definition with the OpenAPI, Swagger, or RAML files.

To update the API definition and policies

1. Click **APIs** in the title navigation bar.

   Note:
   Alternatively, you can navigate to the API details page using the **API** link in the **Microgateway sidecar** section of the service details page.

2. Select an API from the list of APIs.

3. Click and select **Update**.

   The Update API window appears.

4. Update the API definition in one of the following ways:

   - By importing the API definition from a file.
   - By importing the API definition from a URL.

   For more information about how to update APIs, see *webMethods API Gateway User's Guide*.

5. Click **Update**.

   The updated API definition must match the API implemented by the microservices. The REST resources and the available REST operations are enforced by the injected Microgateway.

   Service requests against undefined resources or operations are rejected.

Deploy AppMesh

After updating the API definition, the service is deployed and the policies that are assigned in API Gateway are injected to the Kubernetes pod as a Microgateway sidecar.

Click **Deploy** to deploy AppMesh.

Note:
Before you deploy a service, you must APIfy it and the service must contain an API in API Gateway.

After you deploy AppMesh, you can view the injected Microgateway details in one of the following ways:

- Using the **Microgateway** link in the **Microgateway sidecar** section of the service details page.
- Using the **Microgateways** tab.

Note:

- There is a downtime in the initial deployment, due to the Kubernetes service definition update.
- Services for deployments that have their target ports, which are referenced with the container ports, are not supported by AppMesh. As AppMesh injects an additional container to the deployment, it causes an ambiguity in the referenced service target ports.

Undeploy AppMesh

The undeploy action removes the injected Microgateway from the microservice deployment, and corrects the service definition to point to the microservice.

Click **Undeploy** to undeploy AppMesh.

Note:
There is a downtime in the undeployment, due to the Kubernetes service definition update.

Provisioning of API and Policy Updates

To provision the API definition and policy updates for a Microgateway deployed in the Kubernetes pod, you have to update the API and redeploy AppMesh.

To provision API and policy updates in the AppMesh environment

1. Click **AppMesh**.

   A list of microservices in the Kubernetes environment present in the namespaces, configured in the AppMesh configuration, and those that expose a nodePort and the corresponding deployments appears.

2. Click **View details** to view the service details.

3. Open the corresponding API of the microservice.

4. Update the API definition of the API.

   a. Update the API definition in one the following ways:

      - By importing the API definition from a file.
      - By importing the API definition from a URL.

      For more information about how to update APIs, see "Updating APIs" on page 48.

   b. Update the required API policies, if any.

      For more information about how to update policies, see "Modifying API Policy Details" on page 454.

5. Click **AppMesh** to view the microservices updated with the API definition and API policies.

6. Click **View details** to view the service details.

7. Click **Deploy**.

   The service is redeployed and Microgateway is injected to the Kubernetes pod.

To allow redeployment updates to occur with zero downtime of the pods, the Kubernetes out-of-the-box support through rolling updates is used. This ensures that the deployment does not break the current requests, and no requests are dropped due to a pod failure.

The Kubernetes rolling updates strategy used in AppMesh redeployment has the following parameters:

- `RollingUpdate`. New pods are added gradually, and old pods are terminated gradually.

- `maxSurge`. The number of pods that can be created above the desired amount of pods during an update.
- `maxUnavailable`. The number of pods that can be unavailable during the update process.

Sample Rolling Update strategy you must add in the deployment descriptor that allows for maximum available pods is as follows:

```
strategy:
   type: RollingUpdate
   rollingUpdate:
     maxUnavailable: 0
     maxSurge: 1
```

7 Accessibility Profile

Web Content Accessibility Guidelines

API Gateway supports Web Content Accessibility Guidelines (WCAG) through a separate UI profile called *Accessibility profile*. The *Accessibility profile* is a read-only profile with limited coverage in terms of number of screens as well as the functionalities. Users can access API Gateway accessibility profile using the following URL:

`http://hostname:9071/apigatewayui/accessibility.jsp`

Currently the following screens are available with this profile:

- API Gateway Login page
- API List page
- API Details page