

webMethods API Gateway Administration

Version 10.15

October 2022

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.

Copyright © 2016-2024 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <https://softwareag.com/licenses/>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <https://softwareag.com/licenses/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at <https://softwareag.com/licenses/> and/or in the root installation directory of the licensed product(s).

Document ID: YAI-AG-1015-20240102

Table of Contents

About this Documentation.....	5
Document Conventions.....	6
Online Information and Support.....	6
Data Protection.....	7
1 Deployment.....	9
Concepts.....	10
Deployment Configurations.....	24
2 Operating API Gateway.....	95
Administering API Gateway through API Gateway User Interface.....	96
Starting and Stopping API Gateway.....	96
Data Management.....	98
Monitoring API Gateway.....	176
General Administration Configuration.....	322
Users, Groups, and Teams.....	425
Destination Configuration.....	469
Audit Logging.....	514
System Settings.....	519
Configuring External Accounts.....	533
Configuration Types and Properties.....	541
3 Security Configuration.....	547
Overview of Security Configuration in API Gateway.....	548
Keystore and Truststore.....	548
Ports.....	555
Global IP Access Settings For Ports.....	577
SAML Issuer.....	583
Custom Assertions.....	587
Kerberos Settings.....	596
Master Password Management.....	598
OAuth, JWT, and OpenID Configuration.....	604
Threat Protection Policies.....	622
Securing API Gateway Communication using TLS.....	633
Troubleshooting Tips: Securing API Data Store (Elasticsearch).....	659
4 Container-based Provisioning.....	661
Docker Configuration.....	662
Kubernetes Support.....	687
5 High Availability, Disaster Recovery, and Fault Tolerance.....	701
High Availability.....	702

High Availability and Disaster Recovery.....	706
High Availability and Fault Tolerance.....	714
6 Performance Tuning and Scaling.....	755
Hardware and Product Configurations.....	756
Changing the JVM Heap Size to Tune API Gateway Performance.....	770
Data Separation.....	771
Scaling.....	779
7 Remove User Data from API Gateway.....	785
Removing User Data.....	786
8 API Gateway Configuration with Command Central.....	791
Overview.....	792
Install API Gateway using Command Central.....	793
Manage API Data Store Configurations in Command Central.....	795
Manage API Gateway Product Configurations in Command Central.....	804
Manage Inter-component and Cluster configurations.....	813
Command Line to Manage API Data Store.....	819
Troubleshooting Tips: API Gateway Configuration with Command Central.....	821

About this Documentation

■ Document Conventions	6
■ Online Information and Support	6
■ Data Protection	7

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

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies service names and locations in the format <i>folder.subfolder.service</i> , APIs, Java classes, methods, properties.
<i>Italic</i>	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 Deployment

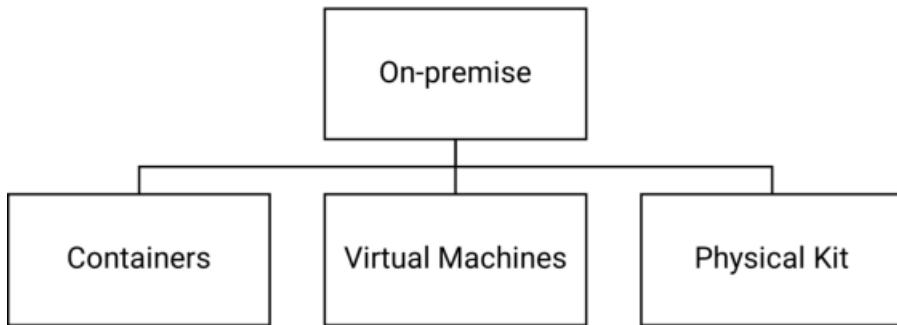
■ Concepts	10
■ Deployment Configurations	24

Concepts

This section provides an overview of the deployment options for API Gateway. Choose a deployment option that best fits your business needs.

On-Premise Deployment Options

You can deploy API Gateway on-premise in the following ways:



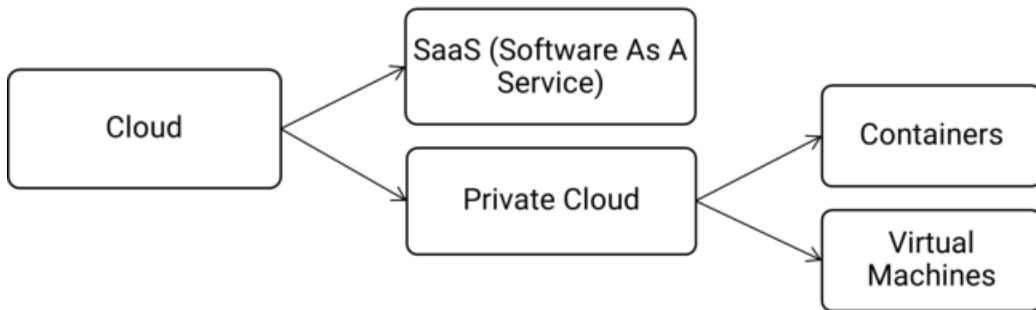
Note:

Running API Gateway in containers is optimal as you can leverage container orchestration platforms like Kubernetes to orchestrate your deployment, although deployment on virtual and physical machines are supported.

Cloud Deployment Options

You can deploy API Gateway on the cloud in two ways: either on the SaaS cloud or your private cloud.

- If you choose the SaaS cloud, Software AG hosts and maintains API Gateway, and manages the infrastructure, which is also the recommended deployment option. Alternatively, you can choose to deploy API Gateway on a private cloud, which is also fully supported.
- If you choose to deploy API Gateway on a private cloud, you can run the application in containers or on virtual machines. Running API Gateway in containers is favorable as the containers can be orchestrated with container orchestration platforms like Kubernetes.



Security Considerations

Regardless of your deployment infrastructure, it is essential that you secure your API management platform. This section provides the guidelines to choose an API Gateway architecture based on your security requirements.

On-premise Deployments

Typically, any on-premise deployment comprises two or more layers. The outer most layer, generally called the Demilitarized Zone (DMZ) protects the inner layer called the trusted zone or green zone against denial of service, SQL injection, and other malicious attacks. The trusted zone hosts services. For such a deployment architecture whose major concern is threat protection, you can implement the DMZ platform security using **API Gateway Standard Edition**.

DMZ platform security implemented using the API Gateway Standard Edition provides the following capabilities:

- Protects the API Management platform from malicious attacks such as Denial of Service (DoS).
- Protects the APIs from common web vulnerabilities, such as SQL, or JSON injection attacks, and so on.
- Boosts the API security by restricting the attackers from sending malicious payloads. For example, large payloads, nested convoluted data structures, and so on.
- Scans the attachments that are part of APIs by integrating with the enterprise anti-virus software through a standard protocol (ICAP).

In addition to threat protection, if you require capabilities such as policy enforcement, request-response transformation, mediation, conditional error processing, and so on, choose the **API Gateway Advanced Edition**.

For more information about the two flavors of API Gateway, see “[API Gateway Editions](#)” on page 12.

Note:

- The API Gateway Standard Edition is meant for threat protection only and not for deploying APIs.
- Web Application Firewall (WAF) provides general protection against threats whereas API Gateway Standard Edition provides security designed to protect the deployed APIs. For more information, see “[API Gateway Editions](#)” on page 12.

Cloud Deployments

Private cloud vendors follow cloud-native procedure such as Defense in Depth. Most vendors offer guidelines, but it might vary across providers. They implement combination of security measures such as networking filters, firewall rules, Web Application Firewall (WAF), and so on. As a result, most of the threats are mitigated, eliminating the need for API Gateway Standard Edition. However, you can add an additional layer of security by setting up an external port and threat protection policies directly on API Gateway Advanced Edition.

API Gateway Editions

You can deploy API Gateway in two editions based on your license.

- **API Gateway: Standard Edition.** This edition of API Gateway offers only API protection.
- **API Gateway: Advanced Edition.** This edition of API Gateway offers both API protection and mediation capabilities.



API Gateway Standard Edition key points	API Gateway Advance Edition key points
Applicable mainly to on-premise deployments.	Applicable to all the deployments.
Protects the API Gateway platform from the malicious attacks. For example, Denial of Service (DoS), Global DoS, Injection Attacks, and so on.	Provides security, mediation and other policy enforcement. For example, request-response transformation, conditional error processing, and so on.

API Gateway Standard Edition key points	API Gateway Advance Edition key points
---	--

Typically, this layer is just a gate keeper and no APIs can be deployed in the standard edition server.

Typically, this layer hosts all the APIs and therefore, it is the main service virtualization layer delivering the intended business value.

For more information about the capabilities available in the Standard and Advanced Editions of API Gateway, see [“API Gateway Standard and Advanced Editions Capability Matrix” on page 91](#)

API Gateway Standard Edition vs Web Application Firewall

This section explains the need for API Gateway Standard Edition, in addition to other software that already exist for the DMZ security such as Web Application Firewall (WAF). API Gateway Standard Edition is required for the following reasons:

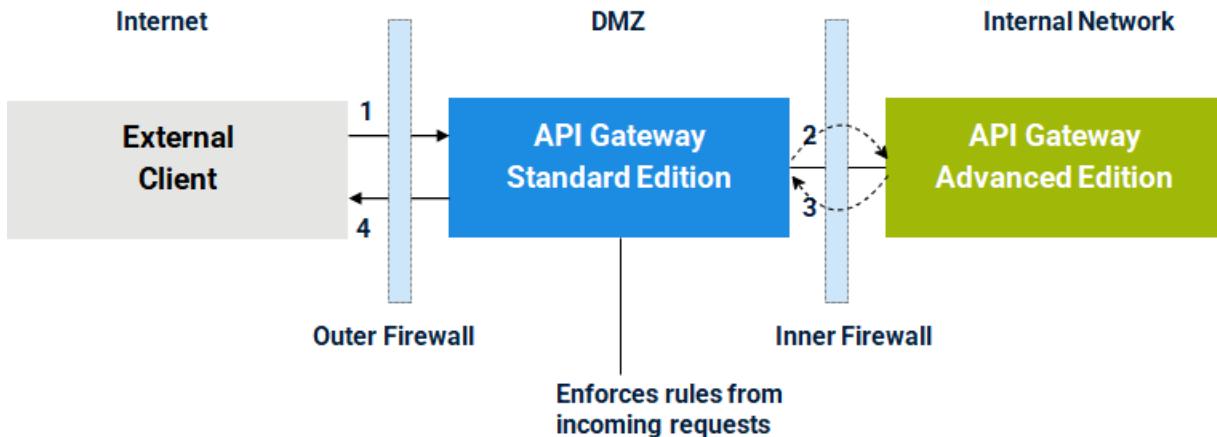
- WAF serves a wider set of edge security concerns and its features vary across products.
- API Gateway Standard Edition provides the necessary threat protection capabilities applicable in the context of exposing APIs to the external world. There may be an overlap of the features between the API Gateway and the WAF. However, API Gateway Standard Edition is not a replacement for WAF.
- If you already have a WAF arrangement in place, depending on the comprehensiveness of its capabilities, you may decide not to use the API Gateway Standard Edition. In such a case, you might need to punch a hole in the inner firewall to allow the API Gateway-specific traffic, which is not optimal in comparison to the reverse invoke capability of the API Gateway Standard Edition, which is considered more secure as you do not have to punch holes in the inner firewall.
- Alternatively, you can combine WAF and API Gateway Standard Edition to leverage the best of both the worlds.

Reverse Invoke in API Gateway

This section explains what is reverse invoke and how it works in API Gateway.

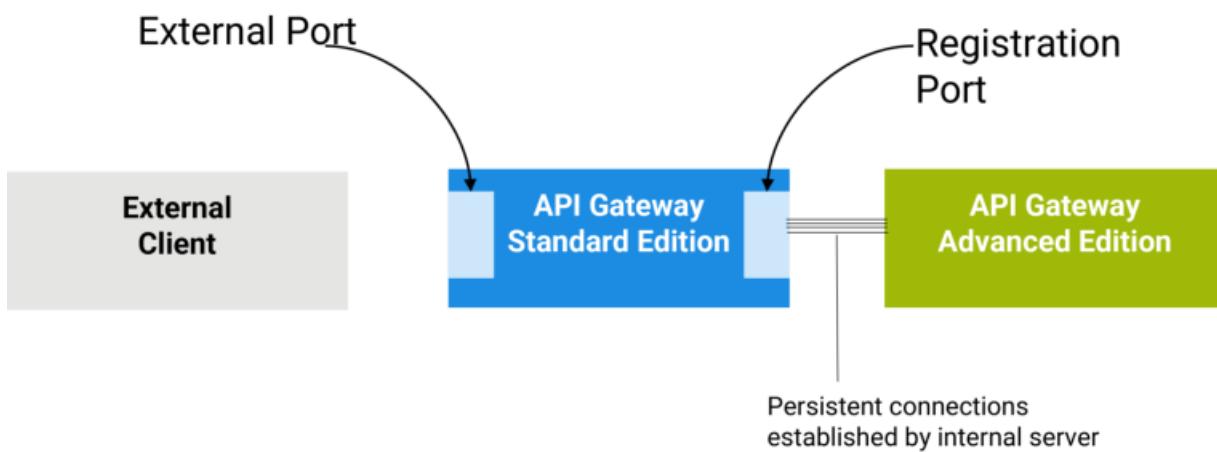
What is Reverse Invoke?

The reverse invoke flow is as follows:



1. External clients send the API requests to the API Gateway Standard Edition Server in the DMZ.
2. The API Gateway Standard Edition Server collects client information from each request and evaluates the request against any rules that is defined. Those requests, which do not violate a rule are passed to the API Gateway Advanced Edition server.
3. The API Gateway Advanced Edition server processes the requests and sends the responses to the API Gateway Standard Edition Server.
4. The API Gateway Standard Edition server then forwards the responses back to the client.

How does Reverse Invoke work?



1. API Gateway Standard Edition server uses an external port to listen to the API requests from external clients.
2. API Gateway Standard Edition server maintains its connection with the API Gateway Advanced Edition server through a registration port. For security purposes, the API Gateway Advanced Edition server initiates the outbound connections to the registration port.

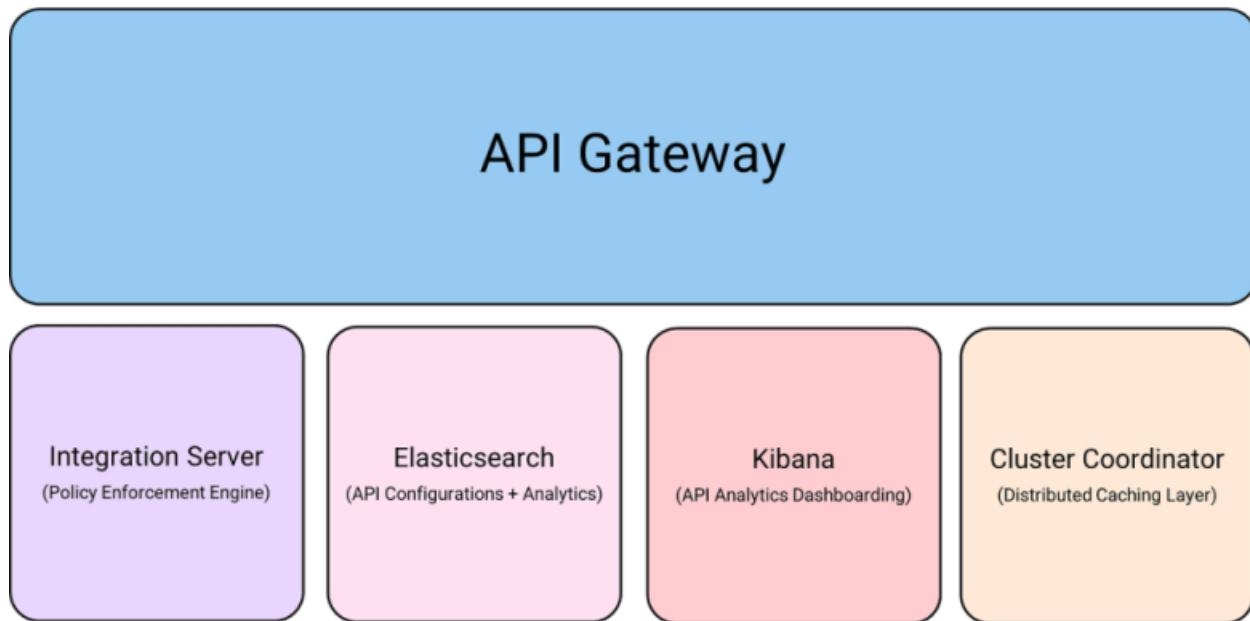
3. By limiting the connections to just those established by the API Gateway Advanced Edition server, this arrangement makes it difficult for attackers to directly penetrate the internal network, even if they subvert a system in the DMZ.
4. For maximum benefit, Software AG highly recommends that you configure the inner firewall to deny all inbound connections. With this configuration, you isolate the servers on the corporate network from the DMZ. This capability is the main advantage of using API Gateway Standard Edition server over traditional third-party proxy servers.

Note:

The reverse invoke method is used in **Paired Deployment**. For more information on paired deployment setup, see "["Paired Deployment" on page 36](#)

API Gateway Components

API Gateway consists of the following components:



The following table lists the purpose and characteristics of the API Gateway components.

Component	Purpose	Characteristics
Integration Server	Policy Enforcement Engine	<ul style="list-style-type: none"> ■ Stateless engine ■ Two or more instances can operate as a cluster ■ Supports vertical and horizontal scaling ■ It runs an embedded Apache Tomcat server to host API Gateway console UI

Component	Purpose	Characteristics
Elasticsearch	Persistence layer for API configurations and API analytics	<ul style="list-style-type: none"> ■ Stateful persistence ■ Externalized Elasticsearch is supported ■ Needs quorum (odd number of instances) to operate in a cluster <p>Note: Best practice is to separate the API analytics in a dedicated cluster.</p> <ul style="list-style-type: none"> ■ Supports horizontal scaling and data-sharing
Kibana	Dashboard for the API analytics	<ul style="list-style-type: none"> ■ Stateless engine ■ Externalized Kibana is supported ■ Two instances can operate in a cluster ■ Scaling more than two is generally not required
Cluster Coordinator	In-Memory Distributed Caching	<ul style="list-style-type: none"> ■ Stateful engine for cluster coordination and service result caching ■ Two options: <ul style="list-style-type: none"> Ignite Terracotta Server Array (TSA) <ul style="list-style-type: none"> ■ Ignite is embedded and scales as you scale the Integration Server. ■ TSA operates in an active-passive setup, typically, two instances are sufficient. Scaling more than two is not required and an anti-pattern .

Note:

All the components can run in containers.

For more information on the supported versions of external Elasticsearch, Kibana, and Terracotta Server Array, see “[API Gateway, Elasticsearch, Kibana, and TSA Compatibility Matrix](#)” on page [92](#).

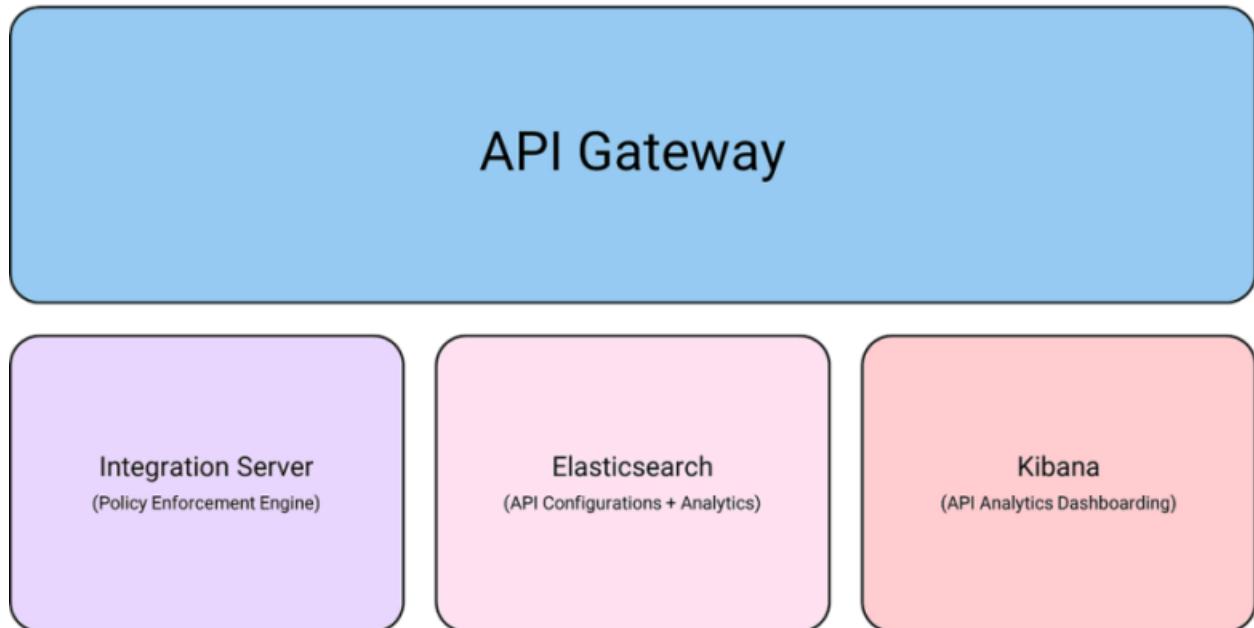
Deployment Models

The following sections explain the API Gateway deployment models based on its purpose. For example, API Gateway is set up differently for development and production environments.

Single-node Deployment

Development environment is the environment, where the developers develop APIs. It is typically individual developer's workstation. And hence it is single-node deployment.

API Gateway single-node deployment consists of the following components:



Note:

Cluster coordinator is skipped as it is not required for single node deployment.

Single-node deployment is possible on virtual machines or containers.

- To deploy API Gateway on a virtual machine, you have to install API Gateway manually using Software AG installer.
- To deploy API Gateway in a container, you have to download docker images from the Container Registry or build your own images. For more details, see "["Docker Configuration" on page 662](#)".
However, it is easy to run an instance in a container in comparison to a virtual machine.

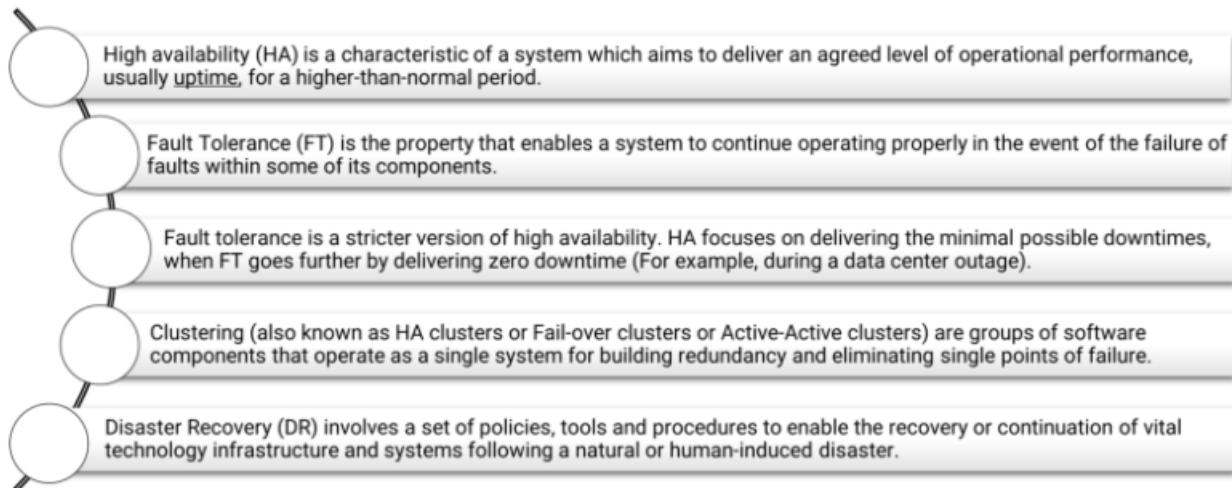
For more information about how to set up an API Gateway instance, see the section "Install API Gateway" in the *webMethods API Gateway Quick Start*.

Multi-node Cluster Deployment

Environments that require High Availability (such as production) can be network of geographically distributed API Gateway instances within or across data centers, typically, either on-premises or cloud, in a virtual machine or containerized set-up.

High Availability, Disaster Recovery, and Fault Tolerance

Production environments require increased uptime and business continuity, which depend on High Availability, Disaster Recovery, and Fault Tolerance capabilities explained in the following diagram.

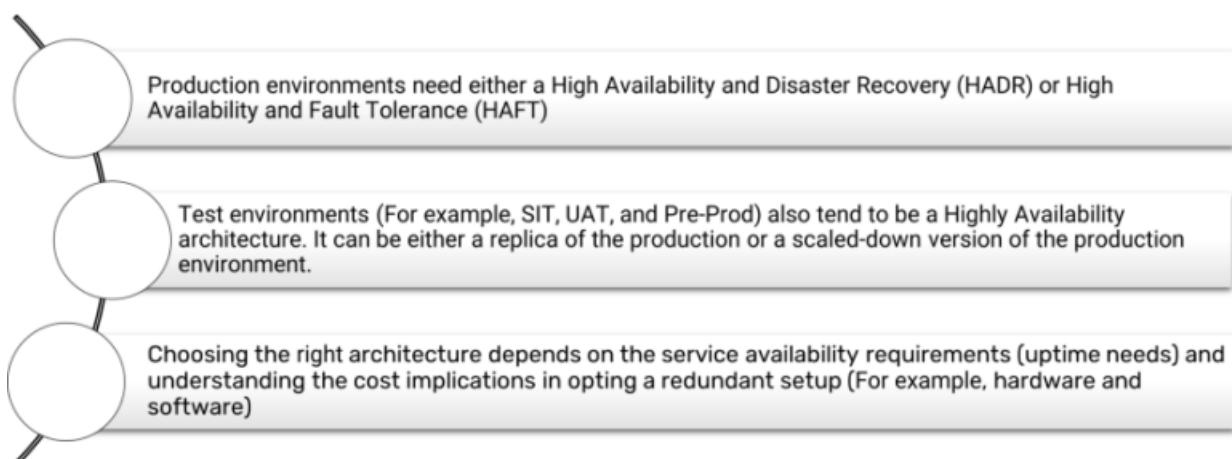


Note:

Unlike high availability and fault tolerance, disaster recovery deals with catastrophic consequences that render entire IT infrastructures unavailable rather than single component failures. Since disaster recovery involves data and technology, its main objective is to recover data as well as get infrastructure components up and running within the shortest time frame after an unpredicted event.

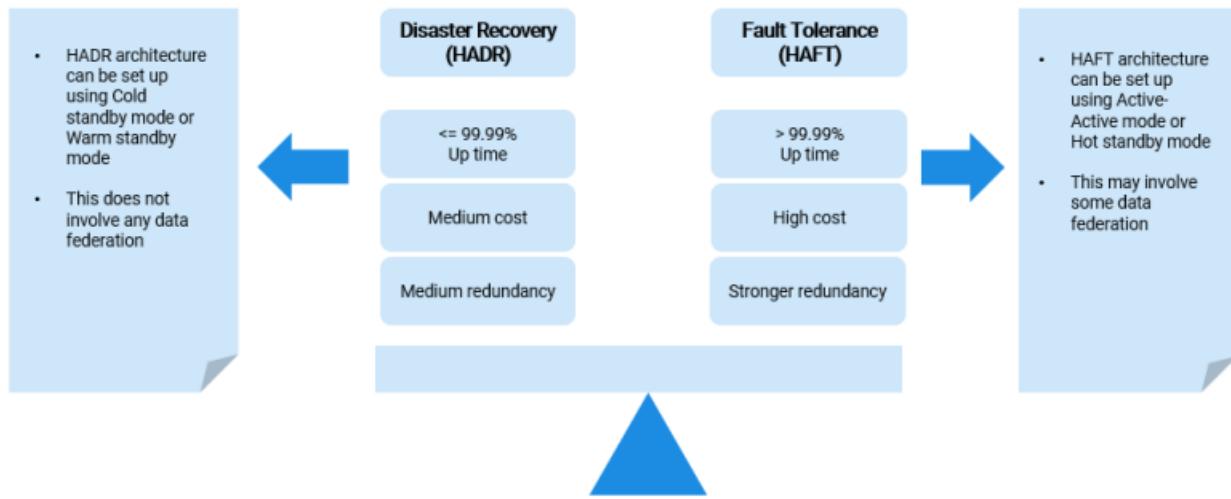
Deployment Recommendations

Software AG recommends the following cluster deployment options for the production and test environments:



Disaster Recovery vs Fault Tolerance

Both High Availability and Disaster Recovery (HADR) and High Availability and Fault Tolerance (HAFT) architectures ensure that the application runs without any system degradation. However, the unique attributes that differentiate them are cost, design, redundancy level, and behavior on component faults or failures.

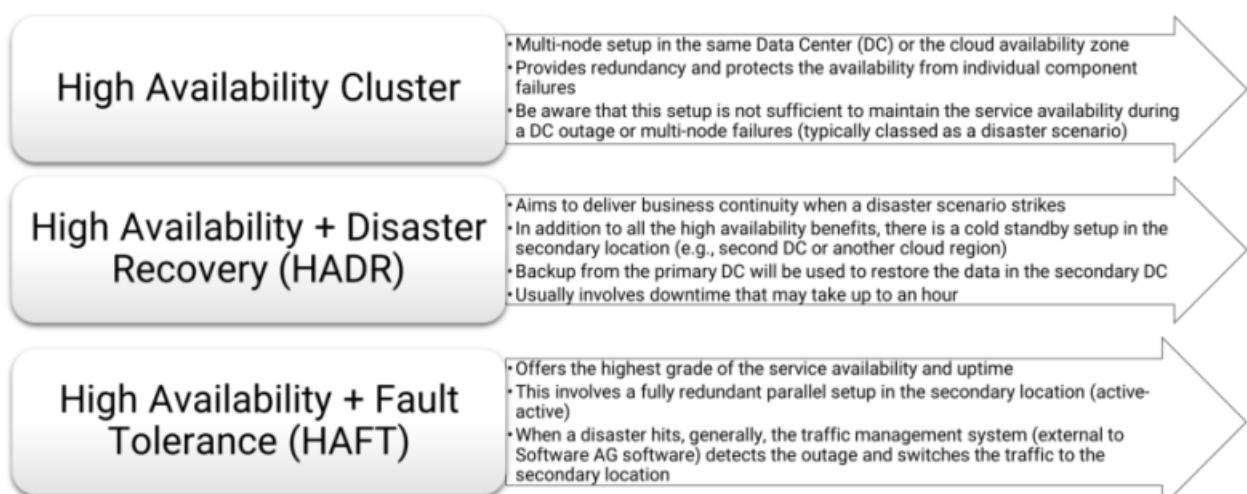


Cluster Deployment Options

To achieve high availability, you can cluster your API Gateway. You can cluster API Gateway nodes using one of the following cluster coordinators:

- **Apache Ignite** operates as an embedded server and scales seamlessly with each node of the API Gateway runtime.
- **Terracotta Server Array (TSA)** operates in an active-passive setup. Typically, two instances are sufficient.

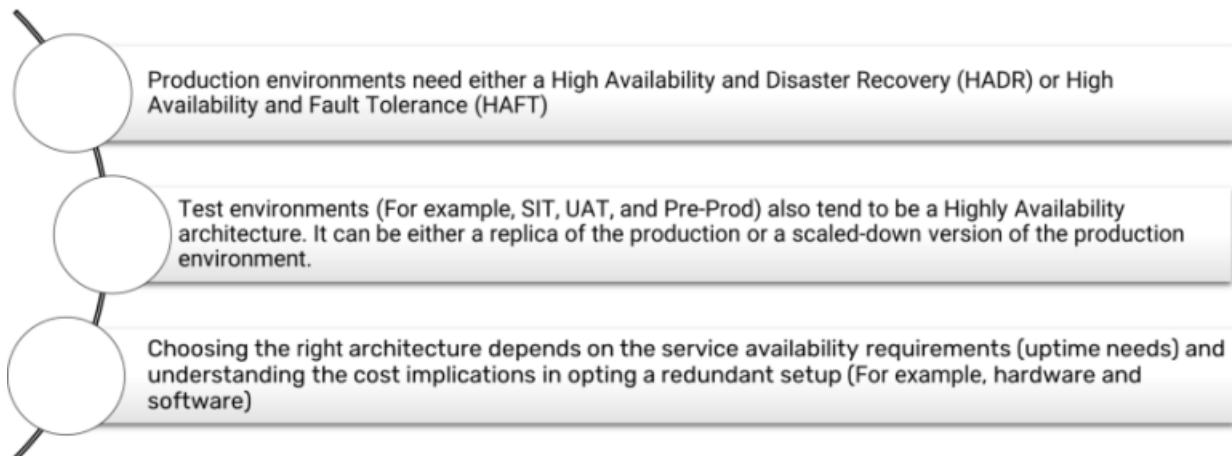
The following image depicts the most popular choices in cluster deployment:



For more information about:

- High Availability Cluster, see “[Cluster Deployment](#)” on page 24.
- High Availability + Disaster Recovery, see “[High Availability and Disaster Recovery](#)” on page 706.
- High Availability + Fault Tolerance, see “[High Availability and Fault Tolerance](#)” on page 714.

Software AG recommends the following cluster deployment options for the production and test environments:



External Elasticsearch and Kibana

Even though API Gateway has embedded Elasticsearch and Kibana, API Gateway provides an option to use external Elasticsearch and Kibana. The key points are as follows:

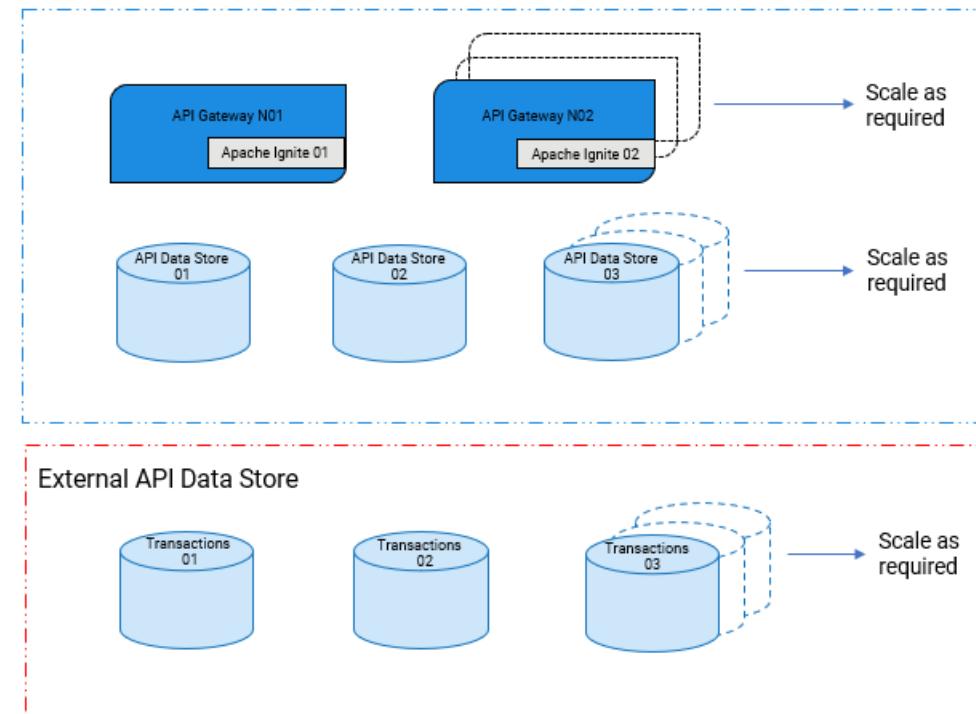
- You can scale each of the component as per your need (as opposed to scaling the embedded bundle).
- You can separate the analytics data from the API configuration data as per the recommended best practice.
- You can improve the efficiency with the data backup or restore procedures.
- It also caters to have different Recovery Time Objective (RTO) and Recovery Point Objective (RPO) for each of the components. And so it helps in restoring API configuration data, which is much more important and tend to have aggressive RTO when compared to the analytics.
- Check the “[compatibility matrix](#)” on page 92 for the supported version(s) of the software.

For details on how to connect to external Elasticsearch and external Kibana, see “[Connecting to an External Elasticsearch](#)” on page 56 and “[Connecting to an External Kibana](#)” on page 60 sections respectively.

Reference Architectures

The following sections explain the various recommended API Gateway deployment architecture with respect to high availability.

Recommended High Availability Architecture

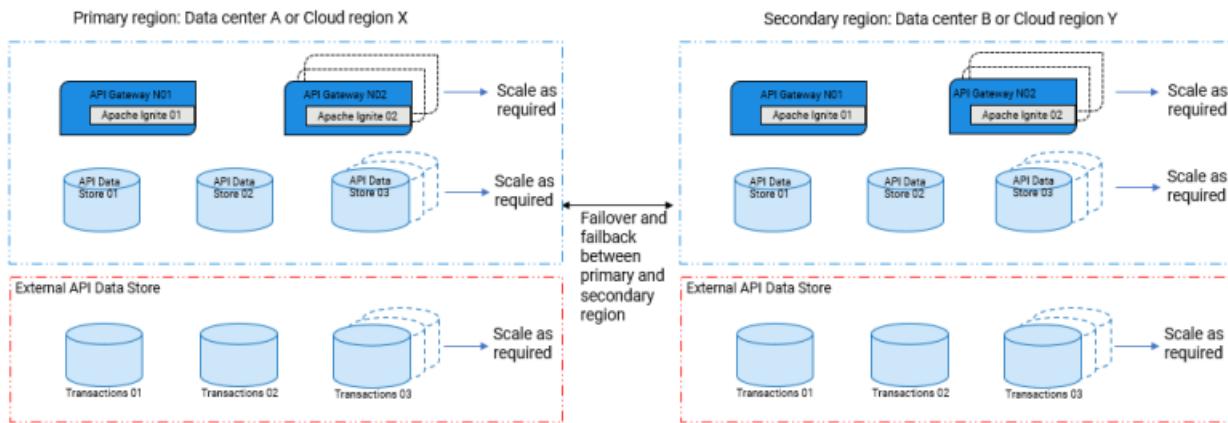


Note:

Alternatively, you can also use the Terracotta Server Array cluster coordinator in active-passive quorum.

High Availability and Disaster Recovery (HADR) solution

The architecture of HADR is as follows:



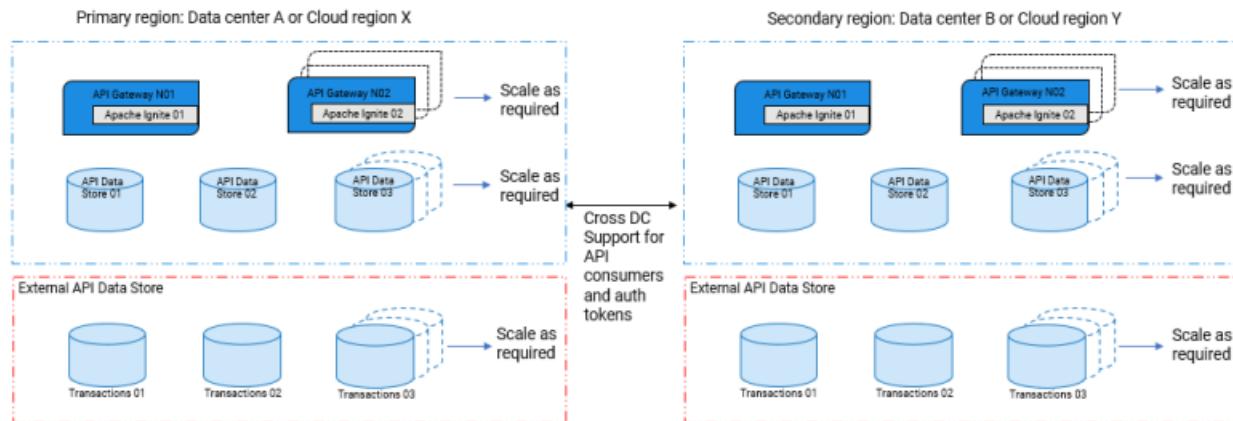
The keypoints about HADR solution are as follows:

- Use this setup, if the Recovery Time Objective (RTO) ranges from 15 minutes to an hour or more.
- Each data center or the cloud region hosts an independent cluster of its own.
- Primary data center serves the traffic and is exposed to the client and you must turn on the secondary data center only when the primary data center goes down due to a disaster.
- Backup snapshots are taken on a periodic basis and stored in a suitable externalized storage (For example, AWS S3 or Azure Blobs).
 - For **Cold standby mode**. Secondary data center remains switched off until failover operation. Hence, this approach saves cost.
 - For **Warm standby mode**. API Data Store is operational in the secondary data center. As API Data Store is operational and data is restored from the latest backup snapshots in secondary data center, this approach reduces RTO, but the cost is higher compared to Cold standby mode.
- After restoring the snapshot, all the API Gateway nodes must be started.
- This architecture supports failback to primary data center.

For more details, see “[High Availability and Disaster Recovery](#)” on page 706.

High Availability and Fault Tolerance (HAFT) solution

The architecture of HAFT is as follows:



The keypoints about HAFT solution are as follows:

- Use this setup, if the RTO is highly demanding, which ranges from few seconds or minutes.
- Each data center or the cloud region hosts an independent or isolated cluster of its own.
- HAFT solution can be set up using **Hot standby mode** or **Active-Active mode**. The following table explains the difference between the modes.
 - For **Hot standby mode**. You can have only two data centers. Out of which, only the primary data center serves the client request. Hence, this approach is cost effective compared to Active-Active mode.
 - For **Active-Active mode**. You can have N (Minimum 3) number of data centers and all the data centers serve the client request. Hence, the cost is more compared to Hot standby mode.
- API assets should be synchronized across the data center through CI-CD deployments.
- Cross-DC federation support offered by API Gateway is limited to API consumers and auth tokens.
- API transactions are local to the cluster where the traffic is served and not aggregated. For aggregated transactions, use a centralized API Data Store.

For more details, see [“High Availability and Fault Tolerance” on page 714](#).

API Analytics

You can view API analytics using one of the following dashboards:

- **API Gateway Dashboard.**
- **Custom Dashboard.** You can implement this outside the Software AG stack. (For example, using your own setup of Elasticsearch and Kibana).

The data storage location determines the dashboard . If you store analytics data in API Data store, you can view the analytics using the default embedded dashboards. By default, API Gateway is shipped with a rich set of purpose-built dashboards. In addition, API Gateway provides custom

dashboard capabilities. Further requirements can be addressed using the custom dashboard capability.

Alternatively, if you prefer to have full control of where the data is stored and how to view dashboard, Software AG recommends you to configure an external Elasticsearch destination and build your own custom dashboards. In such cases, you cannot view the API analytics from the API Gateway UI and it is very important to set up the data housekeeping procedure. For details, see “[Data Housekeeping](#)” on page 101.

Containerization

API Gateway supports containerization. For more information about container-based provisioning, see “[Docker Configuration](#)” on page 662

Product Configurations

For more information about hardware and product configuration guidelines, see “[Hardware and Product Configurations](#)” on page 756.

API Gateway can be configured on startup through a set of external configuration files. For more information, see “[Externalizing Configurations](#)” on page 62

The following sections provide deployment configurations for API Gateway.

Deployment Configurations

The following sections provide deployment configurations for API Gateway.

Cluster Deployment

This section provides information about nodes and clusters in API Gateway and how to configure an API Gateway cluster after you have installed the product. For installation procedures for the product, see *Installing webMethods Products On Premises*.

API Gateway supports clustering to achieve horizontal scalability and reliability.

Each API Gateway cluster node holds all the API Gateway components including API Gateway user interface, the API Gateway package running in webMethods Integration Server, and an API Data Store instance for storing assets. The API Data Store is either embedded in the API Gateway instance or the API Gateway instances store there assets in a separate or external Elasticsearch cluster. A load balancer distributes the incoming requests to the cluster nodes. You can synchronize the nodes either through Apache Ignite or Terracotta Server Array.

Note:

API Gateway does not require an external RDBMS for clustering.

As each node of an API Gateway cluster offers the same functionality, nodes can be added or removed from an existing cluster. The synchronization of any new node happens automatically.

The synchronization includes configuration items, and runtime assets like APIs, policies, and applications. The synchronized runtime assets become active automatically.

Cluster Deployment Options

You can cluster API Gateway nodes using one of the following cluster coordinators:

- Apache Ignite for a peer-to-peer clustered setup.

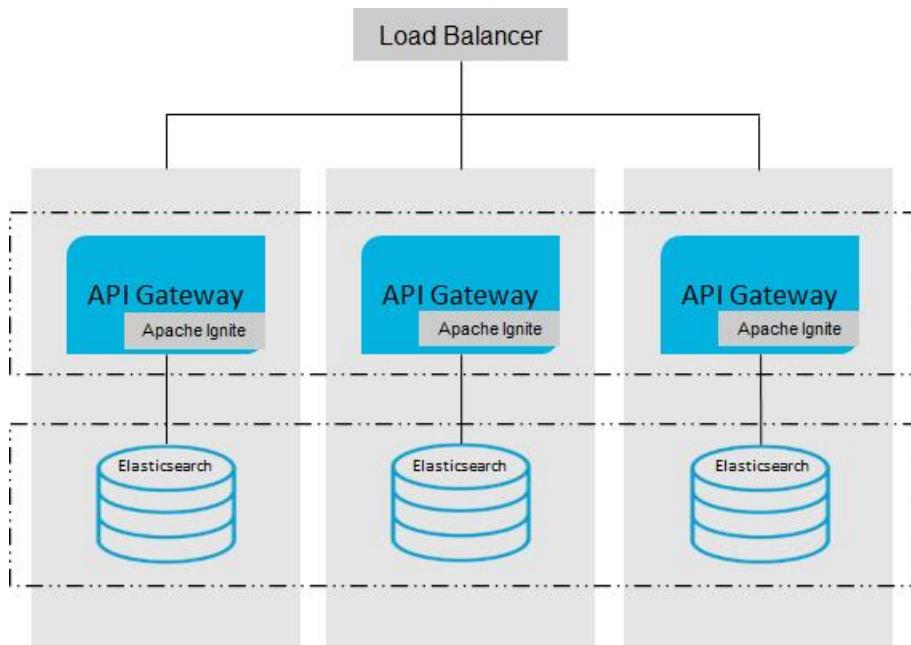
Here, the API Gateway servers are connected to each other, and synchronized through distributed caches.

- Terracotta Server Array (TSA).

Here, the API Gateway servers do not contact each other, rather each API Gateway server connects to the TSA, which has its own runtime, typically deployed on its own machine. The TSA manages the distributed caches that the API Gateway servers use to synchronize.

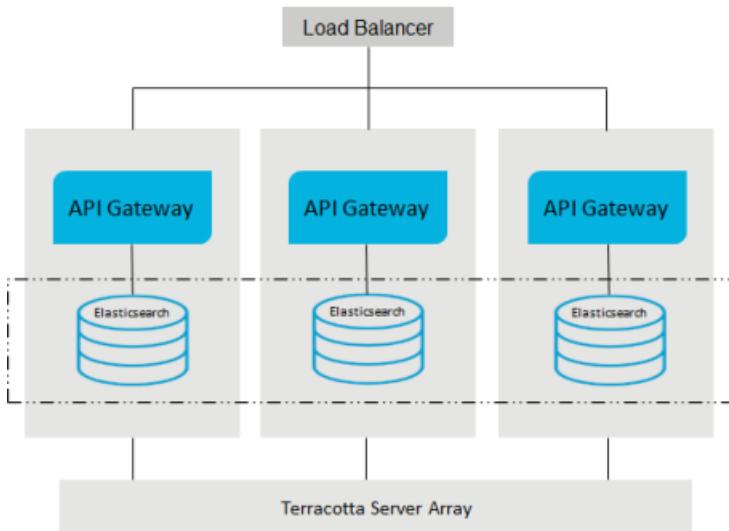
In the options mentioned, an API Gateway cluster can be deployed as a traditional on-premise installation on real or virtual machines, as Docker containers, or in Kubernetes clusters.

Peer-to-peer clustering using Apache Ignite



This setup depicts a peer to peer clustered environment using Apache Ignite and API Data Store clustering that is defined across the embedded API Data Store instances. In this setup, Apache Ignite runs embedded inside each API Gateway instance and is used for distributed caching.

Clustered environment using TSA



This setup depicts a clustered environment using TSA and API Data Store (Elasticsearch) clustering that is defined across the embedded API Data Store instances. TSA is used for distributed caching in this setup.

API Gateway Cluster Configuration

You can enable API Gateway clustering through the API Gateway user interface.

Alternatively, you can set up cluster configurations through externalized configuration files. For details, see “[Using the Externalized Configuration Files](#)” on page 62.

Note:

You cannot configure an API Gateway cluster across multiple data centers, because API Data Store (Elasticsearch) cannot be clustered across multiple data centers.

Enabling Clustering for API Gateway through the User Interface

Keep the following points in mind when enabling clustering for API Gateway.

- You must have API Gateway administrator privileges to enable clustering.
- For a cluster configuration, the embedded API Data Store instances should also be clustered using standard Elasticsearch clustering properties, by modifying the SAG_root/InternalDataStore/config/elasticsearch.yml file on each instance.
- If you are using Terracotta Server Array for clustering, ensure the following:
 - To be in the same cluster, API Gateways must use the same Terracotta Server Array URLs and the same cluster name.
 - An enterprise can have more than one cluster. To isolate multiple clusters on the same network, each cluster must have a different cluster name or different Terracotta Server Array or both.

- Before you enable clustering using Terracotta, perform the following steps to ensure that Terracotta Server Array is ready for use by the API Gateways in the cluster.
- Install the Terracotta Server Array if you have not done so already. For installation instructions, see *Installing Software AG Products*.

Note:

Before you install your Terracotta Server Array, you have to decide how many Terracotta Servers will make up the array and whether or not you want to mirror those servers. To guide your decision, review the sections on high availability and architecture in the product documentation for the Terracotta Server Array or consult your Software AG representative.

- Configure the tc-config.xml file on the Terracotta Server Array. For details on installing and changing a Terracotta license, see *webMethods Integration Server Administrator's Guide*.

➤ To enable clustering

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Clustering**.
3. Click the **Enabled** toggle button to  state to enable the cluster.
4. Provide the following information:

Field	Description
Select cluster type	<p>Specifies the cluster type.</p> <p>You can select one of the following:</p> <ul style="list-style-type: none"> ■ None. Specifies a non-clustered set up. ■ Ignite. Specifies you are using Apache Ignite for the clustering set up. ■ Terracotta. Specifies you are using Terracotta Server Array for the clustering set up.
Cluster name	<p>Specifies the name of the cluster to which API Gateway belongs.</p> <p>Keep the following in mind when specifying the cluster name:</p> <ul style="list-style-type: none"> ■ The cluster name cannot include any periods ". ". API Gateway converts any periods in the name to underscores when you save the cluster configuration.

Field	Description
	<ul style="list-style-type: none"> ■ The cluster name cannot exceed 32 characters. If it does, API Gateway uses the first 20 characters of the supplied name and then a hash for the remaining characters.
Session time out (in minutes)	<p>Specifies the number of minutes an inactive session is retained in the clustered session store.</p> <p>The default is 60 minutes.</p> <p>Set the clustered session timeout value to be longer than the session timeout value, which governs how long a server keeps session information in its memory. (API Gateway displays the session timeout on the Administration > System settings > Configuration page.)</p>
Action on clustering failure	<p>Specifies how API Gateway responds when an error at start up prevents API Gateway from joining the cluster. Select one of the following:</p> <ul style="list-style-type: none"> ■ Start as stand-alone API Gateway. API Gateway starts as a stand-alone, unclustered API Gateway if it encounters any errors that prevent it from joining the cluster at start up. API Gateway continues to receive requests on inbound ports and serve requests. This is the default setting. ■ Shut down API Gateway. API Gateway shuts down if it encounters any errors that prevent it from joining the cluster at start up. ■ Enter quiesce mode on stand-alone. API Gateway starts as a stand-alone, unclustered API Gateway in quiesce mode if it encounters any errors that prevent it from joining the cluster at start up. When API Gateway is in quiesce mode, only the diagnostic port and quiesce port are enabled.
Discovery port	<p><i>This field is available and required when you select Ignite as the cluster type.</i></p> <p>Specifies the port that is used to discover the members that participate in the cluster. Each cluster member opens this port in order to be discoverable. Each API Gateway server tries to contact other servers through this port.</p> <p>When you specify the port number, make sure the port number ranges between 1024 to 65535.</p>
Communication port	<p><i>This field is available and required when you select Ignite as the cluster type.</i></p>

Field	Description
	<p>Specifies the port that is used to communicate between distributed caches. Each server opens this port to be able to receive information. Each server distributes cache entries through this port to other cluster members.</p> <p>When you specify the port number, make sure the port number ranges between 1024 to 65535.</p>
Port range	<p><i>This field is available and required when you select Ignite as the cluster type.</i></p> <p>Specifies the number of ports to include in the range available to multiple cluster members on the same host. The default value is 0.</p> <p>The Port range parameter applies to both the Discovery port and the Communication port.</p> <p>Generally, each cluster member is deployed on its own host. However, for trial or demo purposes you might have multiple cluster members on the same host. In such a situation, each server requires its own discovery and communication port, and must know the other servers' ports. You can use the Port range parameter to configure this scenario. This parameter defines the size of a port range. For example, if you configure the port with port number 10100 and set Port range as 5, the resulting port range that is available is 10100 to 10105. When a server starts, it picks an unused port from the range 10100 to 10105, and uses the port to communicate with the other servers. Another server on start up can use an unused port from the same port range to communicate with the other servers. This allows all servers to use the same cluster configuration.</p>
Provisioning target	<p><i>This field is available and required when you select Ignite as the cluster type.</i></p> <p>Specifies whether you want to use host name or Kubernetes service as the provisioning target.</p> <p>Select one of the following as provisioning targets and provide the required details:</p> <ul style="list-style-type: none"> ■ Hostnames. Provide the name of the participating host in the cluster and click . ■ You can add multiple host names by clicking . ■ Kubernetes. Provide the following details:

Field	Description
	<ul style="list-style-type: none"> ■ Servicename. Provide the name of the service that exposes the API Gateway deployment. ■ Namespace. Provide the name of the Kubernetes namespace within which the API Gateway is deployed.
Terracotta server array URLs	<p><i>This field is available and required when you select Terracotta as the cluster type.</i></p> <p>Provide the URL (<i>host:port</i>) of the TSA to use with the specific API Gateway's cluster and click .</p> <p>You can add multiple TSA URLs by clicking .</p>

5. Click **Save**.
6. Restart API Gateway.
7. Log on as administrator and navigate to **Administration > General > Clustering** and verify that all nodes in the cluster are displayed under **Cluster hosts**.

Note:

You must enable clustering on all nodes in a cluster for the nodes to be included in the cluster.

Terracotta Server Array Configuration

API Gateway requires a Terracotta Server array installation if you select Terracotta server array based clustering. For more information, see *webMethods Integration Server Clustering Guide* and the Terracotta documentation located at <http://www.terracotta.org/>

A sample Terracotta configuration file is as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>

<tc:tc-config xmlns:tc="http://www.terracotta.org/config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <tc-properties>
    <property name="l2.nha.dirtydb.autoDelete" value="true"/>
    <property name="l2.nha.dirtydb.rolling" value="2"/>
    <property name="logging.maxFileSize" value="512"/>
    <property name="logging.maxBackups" value="20"/>
    <property name="l2.nha.tcgroupcomm.reconnect.timeout" value="10000"/>
    <property name="l2.l1reconnect.timeout.millis" value="10000"/>
  </tc-properties>

  <servers>
    <mirror-group group-name="group1">
      <server host="${host}" name="server1" bind="0.0.0.0">
```

```

<data>/opt/softwareag/tsa/server-data</data>
<logs>/opt/softwareag/tsa/server-logs</logs>
<index>/opt/softwareag/tsa/server-index</index>
<authentication/>

<dataStorage size="2g">
    <offheap size="2g"/>
</dataStorage>

</server>

<server host="${host}" name="server2" bind="0.0.0.0">

<data>/opt/softwareag/tsa/server-data</data>
<logs>/opt/softwareag/tsa/server-logs</logs>
<index>/opt/softwareag/tsa/server-index</index>
<authentication/>
<dataStorage size="2g">
    <offheap size="2g"/>
</dataStorage>

</server>

</mirror-group>

<garbage-collection>
    <enabled>true</enabled>
    <verbose>false</verbose>
    <interval>3600</interval>
</garbage-collection>

<restartable enabled="false"/>
<failover-priority>AVAILABILITY</failover-priority>

<client-reconnect-window>360</client-reconnect-window>

</servers>

<clients>
    <logs>logs-%i</logs>
</clients>

</tc:tc-config>

```

API Data Store Cluster Configuration

When running embedded in API Gateway, the API Data store instances have to be clustered by modifying the `SAG_root/InternalDataStore/config/elasticsearch.yml` file within each API Gateway instance. You must provide the cluster configurations in the `elasticsearch.yml` file in the `SAG_root/InternalDataStore/config/` folder before starting the Elasticsearch for the very first time. When you start Elasticsearch, the node auto-bootstraps itself into a new cluster. You cannot change the configuration after bootstrap and thus, Elasticsearch does not merge separate clusters together after they have formed, even if you subsequently try and configure all the nodes into a single cluster. For more information, see <https://www.elastic.co/guide/en/elasticsearch/reference/8.2/index.html>.

Configuring Elasticsearch Cluster

Before you start, ensure that the Elasticsearch is not started after API Gateway installation.

➤ To configure an Elasticsearch cluster

1. If you have started API Gateway before setting up the Elasticsearch cluster configuration, perform the following steps before proceeding with the configuration:
 - Log off and exit from API Gateway.
 - Delete the nodes folder from the *Installation Location\InternalDataStore\data* folder.
 - Make the necessary cluster configuration and start API Gateway.
 - Start Elasticsearch.

A node is created in the Elasticsearch cluster.

2. Open **elasticsearch.yml** from *SAG_root\InternalDataStore\config\elasticsearch.yml* in any node that you want to cluster.

The following configuration is a sample of how the configuration appears initially.

```
cluster.name:"SAG_EventDataStore"
node.name: node1
path.logs: SAG_root\InternalDataStore\logs
network.host:0.0.0.0
http.port:9240
discovery.seed_hosts: ["node1:9340"]
transport.port:9340
path.repo:['SAG_root\InternalDataStore/archives']
cluster.initial_master_nodes:["node1"]
xpack.ml.enabled: false
xpack.security.enabled: false
xpack.security.transport.ssl.enabled: false
xpack.security.http.ssl.enabled: false
action.destructive_requires_name: false
```

discovery.seed_hosts. Provide a list of nodes to the Elasticsearch that it should try to contact. Once the node contacts a member of the unicast list, it receives a full cluster state that lists all nodes in the cluster. It then proceeds to contact the master and join the cluster.

path.repo. This is a mandatory configuration for performing backup and restore. This is the location where the Elasticsearch writes the snapshots to. Hence, it is important to have a location that is accessible to all the nodes. The location must be network file system, S3 or Azure in the clustered setup.

cluster.initial_master_nodes. This parameter must be set so that when you start a cluster for the first time cluster bootstrapping is performed. The parameter must contain the names of the master-eligible nodes in the initial cluster and must be defined on every master-eligible node in the cluster. This setting helps prevent split-brain, the existence of two masters in a single cluster.

`action.destructive_requires_name`. This parameter is set to be *false* so that the migration and backup processes can be performed.

Elasticsearch provides an option to configure the locations where you would want to store your data and logs. Ensure that you specify the locations that are accessible and have enough disk space. It is also important to monitor and ensure basic house keeping of the data location by planning an effective data retention strategy. You can change the defaults using the following configuration:

```
path.data: /var/lib/elasticsearch
path.logs: <IS\_Installed\_Location>/InternalDataStore/logs
// These values can be changed
```

Elasticsearch, by default, binds to loop back address and hence it is important to change it for a production deployment. For more details on this configuration, see <https://www.elastic.co/guide/en/elasticsearch/reference/8.2/modules-network.html>

3. Provide the name of the cluster in the **cluster.name** property.

Nodes with same cluster names form a cluster. That is, if there are three nodes in the cluster, the value in the **cluster.name** property must be same across all three nodes. In other words, Elasticsearch forms a cluster with nodes that have the same **cluster.name**.

For example,

```
cluster.name:"SAG_EventDataStore"
```

4. Provide the names of all participating nodes, as seen in the **node.name** property, and the ports they use, as seen in the **http.port** property, in the **discovery.seed_hosts** property in the following format:

```
host_name:port_name
```

If there are three nodes in the cluster, the value in the **discovery.seed_hosts** property is as in the following example:

```
discovery.seed_hosts: ["node1:9340","node2:9340","node3":"9340"]
```

The names of all nodes appear in the **cluster.initial_master_nodes** property. The node name displayed in this property is same as seen in the **node.name** property.

Sample configuration of a node is as follows:

```
cluster.name:"SAG_EventDataStore"
node.name: node1
path.logs: SAG_root\InternalDataStore/logs
network.host:0.0.0.0
http.port:9240
discovery.seed_hosts: ["hostname1:9340","hostname2:9340","hostname3:9340"]
transport.port:9340
path.repo:['SAG_root\InternalDataStore/archives']
cluster.initial_master_nodes:["node1","node2","node3"]
xpack.ml.enabled: false
xpack.security.enabled: false
xpack.security.transport.ssl.enabled: false
```

```
xpack.security.http.ssl.enabled: false
action.destructive_requires_name: false
```

The specified nodes are clustered.

Adding New Nodes to an Elasticsearch Cluster

This section explains how to add a new node to an Elasticsearch cluster. You can add nodes to a cluster by configuring new nodes to find an existing cluster and start them up.

For example, consider that a new node, *node 4*, is added to a cluster that already has three nodes in it namely, *node1*, *node2*, and *node3*.

➤ To add new node to a cluster

1. Open **elasticsearch.yml** located at `SAG_root/InternalDataStore/config`, where the new node is being added.

The following configuration is a sample of how the configuration appears initially.

```
cluster.name:"SAG_EventDataStore"
node.name: node4
path.logs: SAG_root\InternalDataStore\logs
network.host:0.0.0.0
http.port:9240
discovery.seed_hosts: ["node4:9340"]
transport.port:9340
path.repo: ['SAG_root\InternalDataStore/archives']
cluster.initial_master_nodes:["node4"]
xpack.ml.enabled: false
xpack.security.enabled: false
xpack.security.transport.ssl.enabled: false
xpack.security.http.ssl.enabled: false
action.destructive_requires_name: false
```

2. Provide the name of the node, as seen in the **node.name** property, and port number used by the node, as seen in the **http.port** property, in the **discovery.seed_hosts** property in the following format:

```
host_name:port_name
```

For example

```
node4:9340
```

Sample configuration after providing the new node details:

```
cluster.name:"SAG_EventDataStore"
cluster.initial_master_nodes:["node1","node2","node3"]
node.name: node4
path.logs: SAG_root\InternalDataStore\logs
network.host:0.0.0.0
http.port:9240
discovery.seed_hosts: ["node1:9340","node2:9340","node3":"9340","node4:9340"]
transport.port:9340
```

```

path.repo: ['SAG_root\InternalDataStore/archives']
xpack.ml.enabled: false
xpack.security.enabled: false
xpack.security.transport.ssl.enabled: false
xpack.security.http.ssl.enabled: false
action.destructive_requires_name: false

```

- Save the configuration. The new node is added to the cluster.

Note:

When you restart an Elasticsearch cluster, you must restart the master node first.

If you want to remove a node from a cluster do the following:

- Open the **elasticsearch.yml** file located at *SAG_root/InternalDataStore/config/*.
- Remove the node listed in the format `host_name:port_name` in the **discovery.seed_hosts** property.
- Save the **elasticsearch.yml** file and restart the Elasticsearch cluster. The specified node is now removed from the cluster.

Load Balancer Configuration

You can use a custom load balancer for an API Gateway cluster. Here you use the load balancer nginx.

On a Linux machine, the load balancer configuration file */etc/nginx/nginx.conf* is as follows:

```

user    nginx;
worker_processes  1;
error_log  /var/log/nginx/error.log  debug;
pid      /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include      /etc/nginx/mime.types;
    default_type application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile      on;
    #tcp_nopush    on;
    keepalive_timeout  65;
    gzip  on;

    upstream apigateway {
        server localhost:5555;
        server localhost:5556;
    }
}

```

```
    server localhost:5557;
}

server {
    listen 8000;
    location / {
        proxy_pass http://apigateway;
    }
}
}
```

Use `sudo nginx -s reload` or `sudo nginx -s start` to reload or start nginx. In a test environment, the command `nginx-debug` is used for greater debugging. The load needs to be exposed through the firewall that is protecting the host the firewall is running on. Load balancer should be configured to ensure sticky UI sessions.

Ports Configuration

By default, API Gateway does provide synchronization of the port configuration across API Gateway cluster nodes. If you do not want the ports to be synchronized across API Gateway cluster nodes, set the `portClusteringEnabled` parameter available under **Username > Administration > General > Extended settings** in API Gateway to `false`.

Note:

When this parameter is set to `true`, all the existing port configurations except the diagnostic port (9999) and the primary port (5555) are removed.

Synchronization of ports configuration does not cover temporary disconnects of a node, therefore, to get a node synchronized, you must restart it. Also, if you do not remove the port configuration, the port can be re-synchronized by performing another update on the same configuration. Therefore, to activate the ports synchronization, do the following:

1. Set the `portClusteringEnabled` parameter to `true`.
2. Restart all the cluster nodes.

Configuring Multiple Instances of API Gateway in a Single Installation

The instance creation script can be used to create another instance of API Gateway in the same installation. While creating another instance you can choose your preferred HTTP and HTTPS port for the API Gateway web application using `web.http.port` and `web.https.port` respectively and the back-end REST service endpoint using `primary.port` option.

To create a new instance, run the following command:

```
is_instance.sh create -Dprimary.port=5656 -Dinstance.name=APIGateway
-Dweb.http.port=7474 -Dweb.https.port=7575 -Dpackage.list=WmAPIGateway
```

Paired Deployment

You can configure paired deployment using a reverse invoke setup.

Reverse Invoke Deployment for Paired Gateway Setup

Reverse invoke deployment allows you to securely expose your API end points without exposing the backend APIs or services. You can configure reverse invoke by initiating a connection from the backend servers of the API Gateway present in the demilitarized zone (DMZ).

In a normal configuration, your API Gateway accepts requests directly from the clients in DMZ zone which can cause network security issues. With reverse invoke setup, an additional API Gateway is used to enhance security. The additional API Gateway is placed in the insecure DMZ and the actual API Gateway that interacts with the native services, resides in the more secure green zone.

In a reverse invoke deployment scenario, the external clients send requests to the DMZ API Gateway. These requests are received by the external port of the DMZ API Gateway and forwarded to the registration port. The green zone API Gateway interacts with the registration port and receives the requests, processes the requests through the native service and sends back the responses to the registration port of the DMZ API Gateway. The responses are then forwarded to the external port of DMZ API Gateway and from there to the external clients.

Note:

- If a request is made to the external port and if the API is not available, the request is delegated to the registration port. The listener port configured on the green zone API Gateway listens to the registration port and picks up this request (reverse invoke), processes it, and then sends back the response to the DMZ API Gateway.
- If a request is made to the external port and if the API exists locally, the DMZ API Gateway processes the request.
- The registration port and the external port operate independently. If you define the registration port with the HTTP protocol, you can still configure the external port with the HTTPS protocol.

For more information on ports, see [“Ports” on page 555](#).

Configuring Reverse Invoke Setup

In this scenario, you can configure the threat protection rules in API Gateway server (Standard Edition) located in the DMZ. In the API Gateway instance located in the green zone, you can configure the authentication, authorization, and mediation rules prior to routing the requests to the native API.

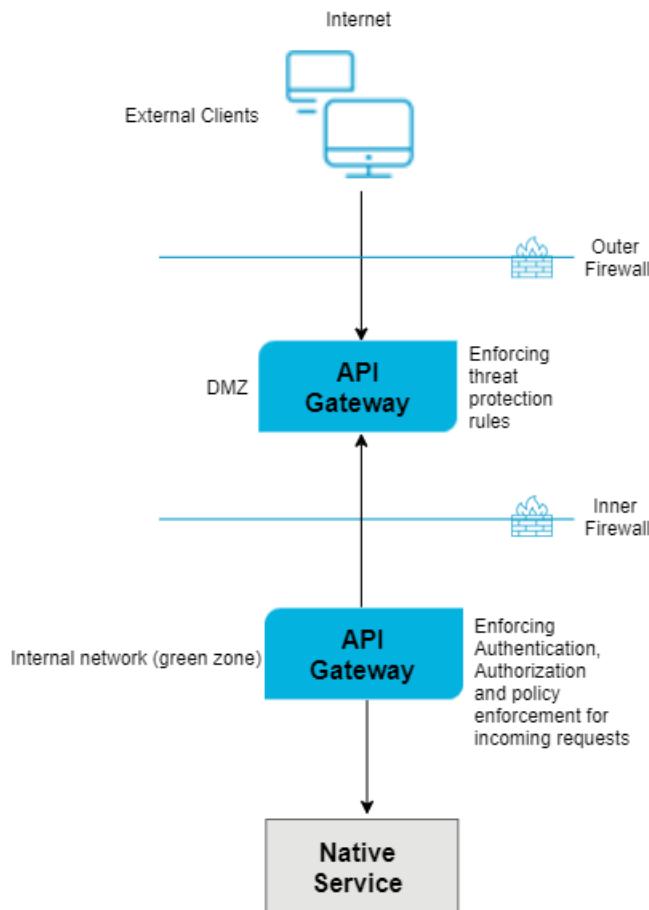
Note:

For a HA setup in API Gateway Standard Edition in DMZ, two nodes are adequate, and Software AG recommends not to cluster the API Gateway Standard Edition. Hence, policies, threat protection rules and configurations must be created or deployed to individual nodes to keep the nodes synchronized.

The following figure describes how reverse invoke works. The client requests are sent to the API Gateway instance in DMZ. These requests are present on the registration port. The green zone API Gateway listens to these requests through the listener port, processes the request through the native service application and responds back to the API Gateway instance in DMZ. The API Gateway instance in DMZ responds to the external clients.

Important:

A connection between API Gateway Server in DMZ and the API Gateway Server in Green zone is available except when a request is being made to the API Gateway in green zone or a response is being returned from the API Gateway in green zone. In other words, DMZ API Gateway connection utilization is I/O bound. Therefore, if you expect large, simultaneous transactions, increase the number of registered connections accordingly.



➤ To configure reverse invoke

1. Configure external and registration ports on API Gateway in DMZ.
 - a. Log on to API Gateway as an Administrator user.
 - b. Expand the menu options icon, in the title bar, and select **Administration**.
 - c. Navigate to **Security > Ports**.
 - d. Click **Add ports**.
 - e. Select **API Gateway external** option from the **Type** drop-down menu.

- f. Click **Add**.
- g. Provide the following information in the **API Gateway external listener configuration** to configure the External port.
- **External port.** Specifies the port number you want to use for the external port.
Use a number that is not already in use. This is the port that clients connect to through your outer firewall.
 - **Alias.** Specifies an alias for the port.
An alias must be between 1 and 255 characters in length and include one or more of the following: letters (a-z, A-Z), numbers (0-9), underscore (_), period (.), and hyphen (-).
 - **Description (optional).** A description of the port.
 - **Protocol.** Specifies the protocol to use for this port (HTTP or HTTPS).
If you select HTTPS, additional security and credential boxes appear for which you have to provide the required values.
 - **Bind address (optional).** Specifies the IP address to which to bind this port.
Specify a bind address if your machine has multiple IP addresses and you want the port to use this specific address. If you do not specify a bind address, API Gateway picks one for you.
 - **Backlog.** Specifies the number of requests that can remain in the queue for an enabled port before API Gateway begins rejecting requests.
The default is 200. The maximum value is 65535.
 - **Keep alive timeout.** Specifies when to close the connection if the server has not received a request from the client within this timeout value (in milliseconds) or when to close the connection if the client has explicitly placed a close request with the server.
The default value is 20000ms.

Note:

For more information on ports, see “[Ports](#)” on page 555.

- h. If you want to configure m-TLS, select **HTTPS** in the **Protocol** field under **API Gateway external listener configuration** and select one of the following options in the **Client authentication** field, in the **Security configuration** section.
- **Request client certificate.** This option requests for a certificate from the client. However, even if the client does not provide a valid certificate, the connection is established.
 - **Require client certificate.** This option requests for a certificate from the client. If the client does not provide a valid certificate, the connection is not established. If you select

this option, you must also configure the following fields in the **Listener specific credentials** section.

- **Keystore alias.** Select a Keystore.
- **Key alias(signing).** Select a Key alias.
- **Truststore alias.** Select Truststore.

- i. Provide the required information to configure the registration port, in the **API Gateway registration listener configuration** section.

The important fields to be configured are **Registration port**, **Alias**, and **Protocol**. For more information on ports, see “[Ports](#)” on page 555.

- j. Configure the **Keystore alias**, **Key alias**, and **Truststore alias** fields, in the **Listener specific credentials** section.
2. Click **Add**.
3. Click the  icon in the **Enabled** column next to the external and registration ports to enable them.

The port is enabled and a success message appears.

4. Execute the following steps in the green zone API Gateway.
 - a. Create an API Gateway internal port.
 - b. Select **HTTPS** in the **Protocol** field.
 - c. In the API Gateway external server section, type the hostname of the DMZ API Gateway in the **Host** field.
 - d. Type the port number of the API Gateway registration port of DMZ API Gateway in the **Port** field.
 - e. In the Registration credentials section, provide the following information.
 - **Keystore alias.** Select a Keystore.
 - **Key alias(signing).** Select a Key alias.
 - **Truststore alias.** Select Truststore.
5. Configure the internal port of the API Gateway in green zone with the registration port of API Gateway in DMZ.
6. Configure load balancer URL in the green zone API Gateway.

- a. Expand the menu options icon, in the title bar, and select **Administration**.
- b. Navigate to **General > Load balancer**.

Provide the configured external server host and port or an external load balancer URL. The API endpoints expose this port for external consumers. If you have a Load Balancer, then the requests from the Load Balancer must be directed to API Gateway's external port.

For more information on load balancers, see "[Clusters and Load Balancers](#)" on page 322.

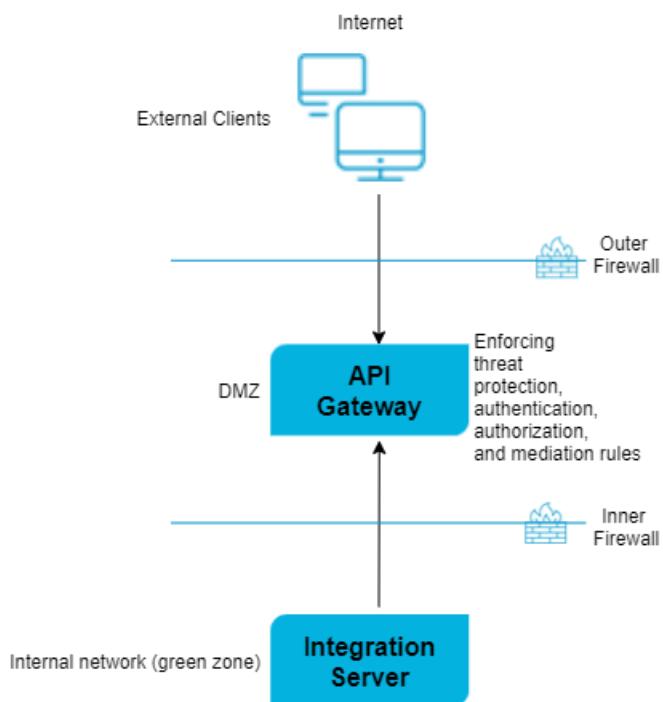
7. Create an API in the internal API Gateway Server with routing protocol and endpoint as the native API. For more information on how to create APIs, see *webMethods API Gateway User's Guide*.
8. You can now access the API by using the URL in the format
`http://externalserver:externalport/gateway/api-name/resource-path`.

Paired Deployment with Integration Server in Green Zone

This is another scenario of paired deployment using reverse invoke in which you use an integration server in green zone.

In this scenario, you can configure threat protection, authentication, authorization, and mediation rules in the API Gateway instance present in DMZ.

The following image describes the working method. The client requests are sent to the API Gateway in DMZ. These requests are present on the registration port. The Integration Server in green zone listens to the registration port through the listener port, processes the requests, and responds back to the API Gateway instance in DMZ. The API Gateway instance in DMZ responds to the external clients.



➤ To configure reverse invoke

1. Configure external and registration ports on API Gateway in DMZ.
 - a. Log on to API Gateway as an Administrator user.
 - b. Expand the menu options icon, in the title bar, and select **Administration**.
 - c. Navigate to **Security > Ports**.
 - d. Click **Add ports**.
 - e. Select **API Gateway external** option from the **Type** drop-down menu.
 - f. Click **Add**.
 - g. Provide the following information in the **API Gateway external listener configuration** to configure the External port.
 - **External port.** Specifies the port number you want to use for the external port.
Use a number that is not already in use. This is the port that clients connect to through your outer firewall.
 - **Alias.** Specifies an alias for the port.

An alias must be between 1 and 255 characters in length and include one or more of the following: letters (a -z, A-Z), numbers (0-9), underscore (_), period (.), and hyphen (-).

- **Description (optional).** A description of the port.
- **Protocol.** Specifies the protocol to use for this port (HTTP or HTTPS).

If you select HTTPS, additional security and credential boxes appear for which you have to provide the required values.

- **Bind address (optional).** Specifies the IP address to which to bind this port.
- Specify a bind address if your machine has multiple IP addresses and you want the port to use this specific address. If you do not specify a bind address, API Gateway picks one for you.
- **Backlog.** Specifies the number of requests that can remain in the queue for an enabled port before API Gateway begins rejecting requests.
- The default is 200. The maximum value is 65535.
- **Keep alive timeout.** Specifies when to close the connection if the server has not received a request from the client within this timeout value (in milliseconds) or when to close the connection if the client has explicitly placed a close request with the server.
- The default value is 20000ms.

Note:

For more information on ports, see “[Ports](#)” on page 555.

- h. If you want to configure m-TLS, select **HTTPS** in the **Protocol** field under **API Gateway external listener configuration** and select one of the following options in the **Client authentication** field, in the **Security configuration** section.
 - **Request client certificate.** This option requests for a certificate from the client. However, even if the client does not provide a valid certificate, the connection is established.
 - **Require client certificate.** This option requests for a certificate from the client. If the client does not provide a valid certificate, the connection is not established. If you select this option, you must also configure the following fields in the **Listener specific credentials** section.
 - **Keystore alias.** Select a Keystore.
 - **Key alias(signing).** Select a Key alias.
 - **Truststore alias.** Select Truststore.
- i. Provide the required information to configure the registration port, in the **API Gateway registration listener configuration** section.

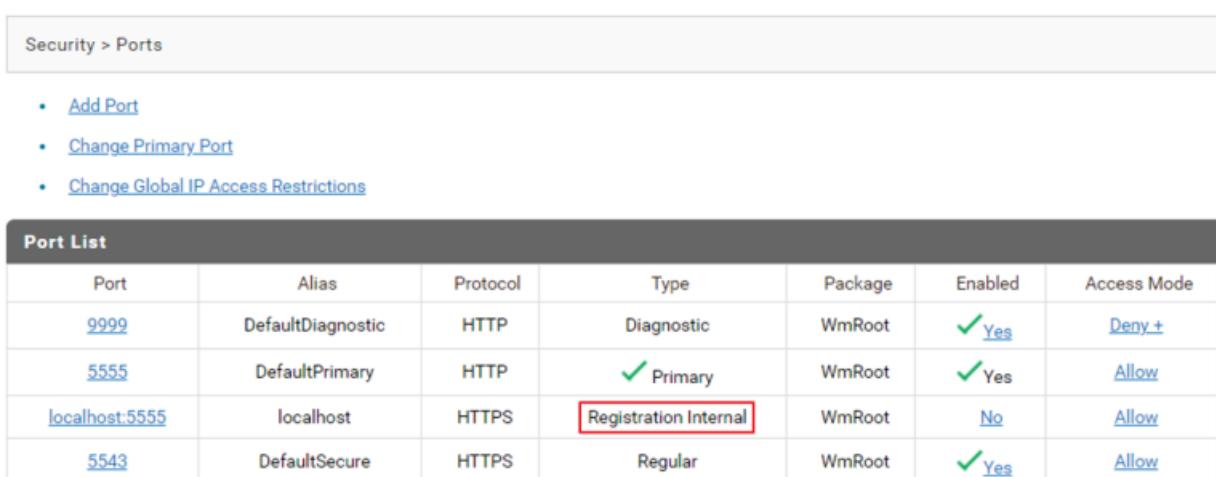
The important fields to be configured are **Registration port**, **Alias**, and **Protocol**. For more information on ports, see “[Ports](#)” on page 555.

- j. Configure the **Keystore alias**, **Key alias**, and **Truststore alias** fields. in the **Listener specific credentials** section.
 2. Click **Add**.
 3. Click the ✕ icon in the **Enabled** column next to the external and registration ports to enable them.
- The port is enabled and a success message appears.
4. Configure Load Balancer URL in API Gateway.
 - a. Expand the menu options icon, in the title bar, and select **Administration**.
 - b. Navigate to **General > Load Balancer**.

Provide the configured external port or an external Load Balancer URL. The API endpoints have this port for external consumers. If you have a Load Balancer, then the requests from the Load Balancer must be directed to API Gateway's External port.

For more information on load balancers, see “[Clusters and Load Balancers](#)” on page 322.

5. In the green zone Integration Server, perform the following configurations to set up two-way SSL.
 - a. Navigate to **Server > Ports**.
 - b. Select the **Registration Internal** port of the API Gateway.



The screenshot shows the 'Ports' configuration screen. At the top, there are three navigation links: 'Add Port', 'Change Primary Port', and 'Change Global IP Access Restrictions'. Below this is a table titled 'Port List' with columns: Port, Alias, Protocol, Type, Package, Enabled, and Access Mode. There are four rows of data:

Port	Alias	Protocol	Type	Package	Enabled	Access Mode
9999	DefaultDiagnostic	HTTP	Diagnostic	WmRoot	Yes	Deny +
5555	DefaultPrimary	HTTP	Primary	WmRoot	Yes	Allow
localhost:5555	localhost	HTTPS	Registration Internal	WmRoot	No	Allow
5543	DefaultSecure	HTTPS	Regular	WmRoot	Yes	Allow

- c. Click **Edit HTTPS Port Configuration**.

Security > Ports > View Internal Server Details

- [Return to Ports](#)
- [Edit HTTPS Port Configuration](#)

Internal Server	
Protocol	HTTPS
Package Name	WmRoot
Alias	localhost
Description (optional)	Integration Server HTTPS port: 5555
Max Connections	5
Enterprise Gateway Server	
Host	localhost
Port	5555
Registration Credentials (Optional)	

- d. Select the Yes option in the **Enable** field.
- e. Configure the fields, as required.
- f. In the Registration credentials section, configure the **Keystore Alias** and **Truststore Alias** fields.
- g. Select **Require Client Certificates** in the **Client Authentication** field.
- h. Click **Save Changes**.

- [Return to Ports](#)

Internal Server	
Enable	<input checked="" type="radio"/> Yes <input type="radio"/> No
Protocol	<input type="radio"/> HTTP <input checked="" type="radio"/> HTTPS
Package Name	WmRoot <input type="button" value="▼"/>
Alias	localhost
Description (optional)	Integration Server HTTPS port: 5555
Max Connections	5
Threadpool	Enable
Enterprise Gateway Server	
Host	localhost
Port	5555
Registration Credentials (Optional)	
User Name	<input type="text"/>
Password	<input type="password"/>
Use JSSE	<input checked="" type="radio"/> Yes <input type="radio"/> No
Keystore Alias	DEFAULT_IS_KEYSTORE <input type="button" value="▼"/>
Key Alias	ssos <input type="button" value="▼"/>
Truststore Alias	DEFAULT_WMCLOUD_TRUSTSTORE <input type="button" value="▼"/>
External Client Security	
Client Authentication	Require Client Certificates <input type="button" value="▼"/>
	Must match Client Authentication setting on the Enterprise Gateway Server External Port.
<input type="button" value="Save Changes"/>	

6. Configure API routing endpoints with registration port alias.

a. Create an API in API Gateway.

For more information on how to create APIs, see *webMethods API Gateway User's Guide*.

Here, the internal server is an Integration Server and to use the reverse invoke functionality, you must modify the routing endpoint of the API created on the API Gateway instance in DMZ as shown in the below syntax.

```
apigateway://{{REG_PORT_ALIAS}}/rest/api/resource
```

If the internal server is not an Integration Server, you can specify the routing endpoint as regular endpoint, where the service is hosted.

Note:

If the routing points to an API that resides in API Gateway, the end point is as follows.

```
apigateway://{{REG_PORT_ALIAS}}/gateway/api/resource which in turn invokes the native service.
```

7. Configure Internal Server Port on the Integration Server in green zone.

a. Configure the Internal Server Port in the Integration Server where the native API resides.

b. Provide the details of API Gateway and Registration port.

8. You can now access the API by using the URL in the format

http://externalserver:externalport/gateway/api-name/resource-path.

Important: A connection between API Gateway Server in DMZ and the internal server in Green zone is available except when a request is being made to the internal server in green zone or a response is being returned from the internal server in green zone. In other words, DMZ API Gateway connection utilization is I/O bound. Therefore, if you expect large, simultaneous transactions, increase the number of registered connections accordingly.

Importing a Certificate and Mapping to User

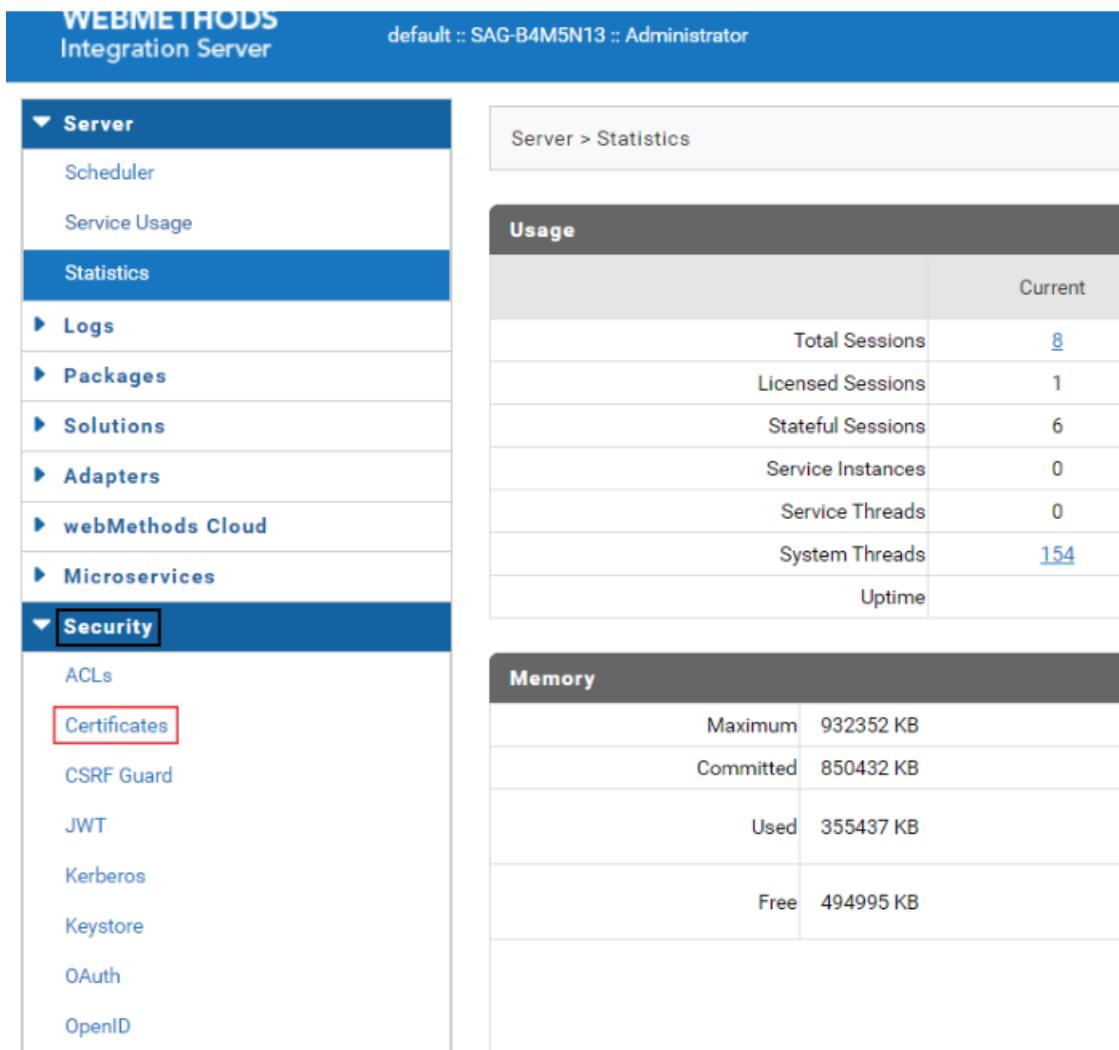
You can import client certificates and CA signing certificates through Integration Server Administrator to keep them on file, map them to particular user accounts, and specify how they are to be used. The user mapping to the certificate must be performed on the external server.

Keep the following points in mind before importing and mapping certificates:

- To create an SSL connection between Integration Server and an internet resource that serves as a client, you have to import a copy of the client's SSL signing certificate (CA certificate).
- Although Integration Server supports loading certificates for LDAP users, Software AG recommends using central user management and then configuring LDAP and certificates in My webMethods Server.

➤ To import a client certificate and map it to a user

1. Open the Integration Server Administrator.
2. Navigate to **Security > Certificates**.



The screenshot shows the 'Statistics' section of the webMethods Integration Server. The left sidebar has a 'Certificates' item highlighted with a red box. The main content area shows usage statistics:

Usage		Current
Total Sessions	8	
Licensed Sessions	1	
Stateful Sessions	6	
Service Instances	0	
Service Threads	0	
System Threads	154	
		Uptime

Memory	
Maximum	932352 KB
Committed	850432 KB
Used	355437 KB
Free	494995 KB

3. Click **Configure Client Certificates**.

The **Configure Client Certificates** window is displayed.

4. Type the path of the certificate that you want to import, in the **Certificate Path** field.

Note:

The certificate must be on the same machine on which the Integration Server is running.

5. Type a user name or click search icon to search for and select a user.

To search a user, perform one of the following tasks, once you click the search icon:

- To select a local user, select **Local** in the **Provider list**. Select the local user to which you want to map the certificate. If you have not configured an external user directory, you cannot view the Provider list.
 - To select a user from an external directory (LDAP or a central user directory), select the user directory that you want to search, in the **Provider list**. In the **Search** field, type the criteria to find a user and click **Go**. Select the user to whom you want to map the certificate.
6. Select one of the following options from the **Usage** field.
- **SSL Authentication**. Use the certificate to represent the client's authentication credentials when making an SSL connection with Integration Server.
 - **Verify**. Use the certificate's public key to verify the authenticity of documents, messages, or streams originating from the client and containing a digital signature.
 - **Encrypt**. Use the certificate's public key to encrypt outgoing documents, messages, or streams from Integration Server to the client.
 - **Verify and Encrypt**. Use the same certificate to verify the authenticity of documents, messages, or streams originating from the client and containing a digital signature, and to encrypt outgoing documents, messages, or streams from Integration Server to the client.
 - **Message Authentication**. Use the certificate to represent the client's authentication credentials when making an SSL connection with Integration Server, when using message-level rather than transport-level authentication. For example, with web service messages whose SOAP message headers contain SSL certificate information.
7. Click **Import Certificate**.

The screenshot shows the 'Security > Certificates > Configure Client Certificates' page. On the left, there is a sidebar with links like Server, Scheduler, Service Usage, Statistics, Logs, Packages, Solutions, Adapters, webMethods Cloud, Microservices, and Security. The Security link is highlighted. The main area shows the 'Import Certificate' dialog with the following details:

Import Certificate	
Certificate Path	<input type="text" value="C:\Myfiles\APICertificate"/>
Example Paths	<input type="text" value="C:\folder\file.der (location must be accessible to Integration Server)
\\server\folder\file.der (location must be accessible to Integration Server)
folder\file.der (relative to Integration Server instance directory)"/>
User	<input type="text" value="Administrator"/> <input type="button" value="..."/>
Usage	<input type="text" value="SSL Authentication"/> <input type="button" value="..."/>
<input type="button" value="Import Certificate"/>	

Troubleshooting Tips: API Gateway and DMZ Connectivity

I see several requests waiting for a registration connection

This might occur when the network and connections are unreliable.

On the internal server (Green Zone), configure the property `watt.server.rg.internalregistration.timeout` that controls how long the API Gateway server waits for a connection to the internal server before closing an unresponsive connection to the API Gateway server.

To configure this property:

1. Navigate to **User menu > Administration > General > Extended settings**.
2. Click **Show and hide keys**.
3. Select `watt.server.rg.internalregistration.timeout` property in the watt properties section.
4. Provide a suitable value as follows:
 - Provide a value 0 when the network and connections are reliable, so that the connection between internal server and API Gateway never times out. This is the default value of this property.
 - Provide a non-zero value when the network and connections are unreliable or when the connections breakdown. This value specifies the time after which API Gateway stops trying for a unresponsive internal server connection so that the connections time out.
5. Click **Save**.

Considerations while configuring the `watt.server.rg.internalregistration.timeout` property:

- When you set the property to a value within which if API Gateway does not receive any requests from DMZ, then registration internal ports are auto refreshed. When the connectivity between API Gateway and DMZ is broken after a refresh (disabled and enabled), set the internal ports manually to resolve the issue.
- When you set the property to a value that is lower than `watt.net.socketpool.sweeperInterval`, the internal server closes the connection to the API Gateway server and re-establishes a new connection regularly.

The property `watt.net.socketpool.sweeperInterval` specifies the frequency, in seconds, at which the socket pool sweeper performs. The socket pool sweeper sends a ping request to all API Gateway connections and HTTP client connections. During a sweep it removes any invalid HTTP client connections. By default, the sweeper sends a ping request every 60 seconds.

As a good practice, Software AG recommends enabling the `watt.net.socketpool.sweeperInterval` setting, if you are using the `watt.server.rg.internalregistration.timeout` property. Set the value of `watt.server.rg.internalregistration.timeout` on the internal server to a value greater than the ping values defined by the `watt.net.socketpool.sweeperInterval` server configuration parameter on the API Gateway server.

In addition to the above parameter setting also increase the connection from internal server by ensuring that you have enough threads configured in both DMZ and internal server to handle the load.

I see client requests on the API Gateway server (DMZ) waiting indefinitely for a connection to the internal server (Green zone).

This might occur when the connections breakdown.

Configure the property `watt.server.rg.internalsocket.timeout` that controls how long the API Gateway server waits for a connection to the internal server and returns a HTTP 500-Internal Server Error to the requesting client.

To configure this property:

1. Navigate to **User menu > Administration > General > Extended settings**.
2. Click **Show and hide keys**.
3. Select `watt.server.rg.internalsocket.timeout` property in the watt properties section.
4. Provide a suitable value.
 - If a connection to the internal server becomes available within the specified timeout period, API Gateway server forwards the request to the internal server.
 - If a connection does not become available before the timeout elapses, API Gateway server returns a HTTP 500-Internal Server error to the requesting client.
5. Click **Save**.

I see client authentication is not enforced for APIs invoked on API Gateway server (DMZ) if requests are sent to the external port.

For an API invocation on API Gateway server (DMZ) if requests are sent to the external port you may observe that there is no client authentication performed and you may observe that the enforced IAM policies fail.

To enable client authentication for requests that sent to the external port, you must set the `watt.server.revInvoke.proxyMapUserCerts` property as follows:

1. Navigate to **User menu > Administration > General > Extended settings**.
2. Click **Show and hide keys**.
3. Select `watt.server.revInvoke.proxyMapUserCerts` property in the watt properties section and set it to true.

I see some of the internal API invocations are processed in the API Gateway server (DMZ) instead of API Gateway server (Green zone) when using an API Gateway Advanced Edition.

This is observed when you have the paired deployment setup and you have enforced only threat protection policies in API Gateway server (DMZ) and all other policies in internal server (Green zone) and you have configured port access restrictions to allow access only to the APIs hosted on the API Gateway (say with `/gateway/`, `/ws/`, and so on). In such a case you must provide access

to the following APIs in case the APIs are protected by security policies such as OAuth, OpenId or JWT. Allowing access to these endpoints is important for API Portal and API consumers to access API Gateway to retrieve the tokens.

- pub.apigateway.oauth2:getAccessToken
- pub.apigateway.oauth2/getAccessToken
- pub/apigateway/oauth2/getAccessToken
- secure.apigateway.oauth2/approve
- secure.apigateway.oauth2:approve
- secure/apigateway/oauth2/approve
- pub.apigateway.oauth2:authorize
- pub.apigateway.oauth2/authorize
- pub/apigateway/oauth2/authorize
- pub/apigateway/openid/getOpenIDToken
- pub/apigateway/openid/openIDCallback
- pub/apigateway/jwt/getJsonWebToken
- /pub/apigateway/jwt/certs
- /pub/apigateway/jwt/configuration
- /pub/apigateway/jwt/thirdPartyConfiguration

By default, if API Gateway server (DMZ) has API Gateway Advanced Edition then API requests are processed on the API Gateway server in DMZ but the actual enforcement happens in the internal server. Hence, these API requests must be processed on internal server. To resolve this, configure the extended setting `forwardInternalAPIsRequest` as follows:

1. Navigate to **User menu > Administration > General > Extended settings**.
2. Click **Show and hide keys**.
3. Select the extended setting `forwardInternalAPIsRequest` and set it to true.

I see that the internal APIs can be accessed through the External Port of API Gateway server deployed in DMZ

API Gateway supports the safe exposure of APIs by featuring threat protection and enforcing identity and access management policies. The reverse invoke capabilities of API Gateway supports the exposure of APIs that are running entirely behind a firewall or where just the service implementing the APIs is protected by a firewall.

In cases where you can access the internal APIs through the external port of API Gateway server in DMZ, you can block requests to internal APIs in API Gateway as follows:

1. Configure the following ports in API Gateway located in DMZ:

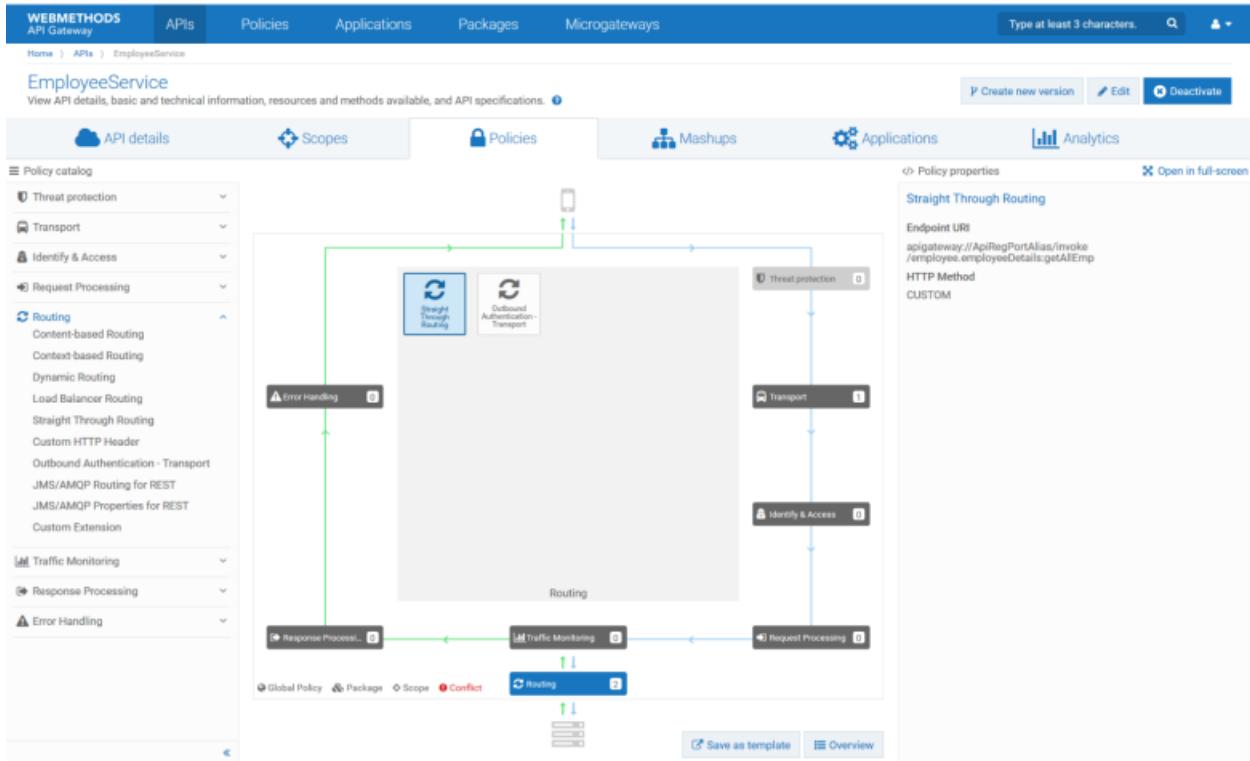
- 5500: API Gateway default HTTP regular port
- 9200: API Gateway external port (ExtPortAlias)
- 9201: API Gateway registration port (DefaultRegPortAlias)
- 9202: API Gateway registration port (ApiRegPortAlias)

The screenshot shows the 'Administration' section of the webMethods API Gateway. The left sidebar has 'Ports' selected under 'Keystore/Truststore'. The main area is titled 'Ports' with the sub-instruction 'Configure listener ports in API Gateway.' Below is a table listing five ports:

Ports	Alias	Protocol	Type	Enabled	Accessmode	IP Accessmode	Primary port	Description
5500	DefaultPrimary	HTTP	Regular	✓	○	○	✓	Default Primary Port
9201	DefaultRegPortAlias	HTTP	API Gateway registration	✓	✗	○		Integration Server HTTP port: 9201
9202	ApiRegPortAlias	HTTP	API Gateway registration	✓	✗	○		Integration Server HTTP port: 9202
9200	ExtPortAlias	HTTP	API Gateway external	✓	✗	○		Integration Server HTTP port: 9200
9201	internal	HTTP	API Gateway internal	✓	○	✗		Integration Server HTTP port: 9201

With the above configuration, the API Gateway instance in DMZ receives requests on the external port ExtPortAlias. These requests are forwarded to the registration port DefaultRegPortAlias. DefaultRegPortAlias is connected to an API Gateway internal port that is defined on the API Gateway in the DMZ. This ensures that requests are not forwarded to the Integration Server or API Gateway in the green zone, but processed within the API Gateway in DMZ. The default registration port is the first one defined for an external port. There is no specific naming required.

2. Configure the routing policies in the API Gateway instance located in DMZ, as follows.



To forward API requests to the backend services, the API routing policies must point to the ApiRegPortAlias, the second registration port alias defined for the external port.

The endpoint URI of the Straight Through Routing policy leverages the apigateway scheme and references the ApiRegPortAlias. The resource path of the endpoint URI points to the sample flow service employee running on the Integration Server in green zone. This flow service can not be invoked directly from the DMZ external port. All non-API requests are routed to the internal port of the API Gateway in DMZ where the backend flow services are not defined.

3. Configure the internal port of the Integration Server in green zone as follows.

Port	Alias	Protocol	Type	Package	Enabled	Access Mode	IP Access	Advanced	Delete	Description
9999	DefaultDiagnostic	HTTP	Diagnostic	WmRoot	✓ Yes	Deny+	Allow	Edit	X	Default Diagnostic Port
DMZHost:9202	Internal	HTTP	Registration Internal	WmRoot	✓ Yes	Allow	Not Applicable	Edit	X	Integration Server HTTP port: 9201
5555	DefaultPrimary	HTTP	✓ Primary	WmRoot	✓ Yes	Allow	Allow	Edit	X	Default Primary Port

The registration port ApiRegPortAlias is associated with the internal port of the Integration Server in the green zone. The figure depicts the port configuration screen with the internal port connected referencing the registration port ApiRegPortAlias on the API Gateway in DMZ.

My internal server has run out of registration connections

My internal server has run out of registration port and I see the following error message.
number requests waiting for a registration connection.

Resolution

Each connection consumes a thread, either from the API Gateway in green zone's common thread pool or from the internal listener's private thread pool, if one is defined. The consumed thread can only be used to process requests from API Gateway in DMZ.

If you have defined a private thread pool for the internal registration listener, the number of connections you can specify in the Max Connections box is limited to the maximum number of threads allowed in the private thread pool for this listener.

If you have multiple internal registration listeners, each with its own private thread pool, the same rule applies for each internal registration listener.

If you have not defined a private thread pool for an internal registration listener, a reasonable limit for the Max Connections box is 75% of the number of server threads specified in Server Thread Pool Max Threads box on the Settings > Resources page. If you have multiple internal registration listeners and none of them have private thread pools, the sum of all connections specified in the Max Connections boxes for these listeners should not exceed 75% of the number of server threads specified in Server Thread Pool Max Threads.

A thread remains open unless it is closed by a firewall, a network glitch, or an exception.

Connecting to an External Elasticsearch

This section explains the changes that you must make in the config.properties file to enable API Gateway to communicate with the external Elasticsearch.

You can also connect to an external Elasticsearch through externalized configurations. For more details, see “[Externalizing Configurations](#)” on page 62.

Note:

If you use an external Elasticsearch with same version as API Data Store, then you can use the Kibana or dashboard that is shipped with API Gateway, else they have to be configured separately. If you have configured Elasticsearch externally, then you have to configure Kibana externally. To know the compatible Kibana and Filebeat (Beats) versions for your Elasticsearch, see <https://www.elastic.co>.

Important:

When you use an external Elasticsearch, you must use the Elasticsearch plugin mapper size. Without this plugin, API Gateway does not start. To install the mapper size plugin, use the command `sudo bin/elasticsearch-plugin install mapper-size`. If you have deployed a clustered environment and have configured external Elasticsearch on multiple nodes, you must install this plugin on all the nodes that use external Elasticsearch.

➤ To connect to an external Elasticsearch

1. Navigate to `WmAPIGateway/config/resources/elasticsearch/config.properties`

The config.properties file contains all the properties and Elasticsearch configurations.

2. Configure the following properties:

Property and Description

pg.gateway.elasticsearch.autostart

This property specifies whether the Elasticsearch starts automatically. If an external Elasticsearch is configured it has to be manually started. This property needs to be set to false to avoid API Data Store starting automatically.

Default value: true

pg.gateway.elasticsearch.client.http.response.size

This property specifies the response size, in MB, for API Gateway Elasticsearch client.

Default value: 100

pg.gateway.elasticsearch.config.location

This property specifies the location of the config file if you want to read port details from some other Elasticsearch config file

Property and Description

pg.gateway.elasticsearch.hosts

Mandatory

This property lists Elasticsearch hosts and ports. The values are comma separated.

Default value: localhost:9240

Note:

Once a host is added to this property, this is the value that is used to connect to Elasticsearch and the host configured in gateway-es-store.xml is not considered.

pg.gateway.elasticsearch.http.keepAlive

Mandatory

This property creates the persistent connection between client and server.

Default value: true

pg.gateway.elasticsearch.http.connectionTimeout

Mandatory

This property specifies the time, in milliseconds, after which the connection times out.

Default value: 10000

pg.gateway.elasticsearch.http.socketTimeout

Mandatory

This property specifies the wait time, in milliseconds, for a reply once the connection to Elasticsearch is established after which it times out.

Default value: 30000

pg.gateway.elasticsearch.http.maxRetryTimeout

Mandatory

This property specifies the wait time, in milliseconds, for retries after which it times out.

Default value: 100000

It is advisable to set max retry time for a request to (number of nodes * socketTimeOut)+connectionTimeout

pg.gateway.elasticsearch.http.keepAlive.maxConnections

Mandatory

Property and Description

This property specifies the maximum number of persistent connections that can be established between an API Gateway and Elasticsearch cluster.

Default value: 50

pg.gateway.elasticsearch.http.keepAlive.maxConnectionsPerRoute

Mandatory

This property specifies the maximum number of persistent connections that can be established per HTTP route to an Elasticsearch server.

Default value: 15

pg.gateway.elasticsearch.http.username

This property specifies the user name to connect to Elasticsearch using basic authentication.

pg.gateway.elasticsearch.http.password

This property specifies the password to connect to Elasticsearch using basic authentication.

pg.gateway.elasticsearch.https.keystore.filepath

This property specifies the Keystore file path for establishing HTTPS communication with Elasticsearch.

pg.gateway.elasticsearch.https.truststore.filepath

This property specifies the truststore file path for establishing HTTPS communication with Elasticsearch.

pg.gateway.elasticsearch.https.keystore.password

This property specifies the Keystore password for establishing HTTPS communication with Elasticsearch.

pg.gateway.elasticsearch.https.keystore.alias

This property specifies the Keystore alias for establishing HTTPS communication with Elasticsearch.

pg.gateway.elasticsearch.https.truststore.password

This property specifies the truststore password for establishing HTTPS communication with Elasticsearch.

pg.gateway.elasticsearch.https.enabled

This property specifies whether you want to enable or disable the HTTPS communication with Elasticsearch.

Default value: false

Property and Description

If this property is set to false none of the above properties related to HTTPS are respected.

pg.gateway.elasticsearch.outbound.proxy.enabled

This property specifies whether you want to enable or disable outbound proxy communication.

Default value: true

pg.gateway.elasticsearch.outbound.proxy.alias

This property specifies the outbound proxy alias name used to connect to Elasticsearch.

pg.gateway.elasticsearch.https.enforce.hostname.verification

This property enforces the host name verification for SSL communication.

Default value: false

pg.gateway.elasticsearch.sniff.enable

Mandatory

This property enables sniffers to add the other nodes in an Elasticsearch cluster to the client so that the client can talk to all nodes.

Default value: true

This configuration must be set to *false* if you are changing the network when API Gateway or Elasticsearch is running.

pg.gateway.elasticsearch.tenantId

This property allows you to specify a tenant name of your choice. This value must be same across all nodes.

The default value of this property is the Integration Server instance name. So, ensure that you provide same name for all Integration Server nodes in a cluster.

If you modify this value, you must edit the value in all nodes and restart the API Gateway server for the change to take effect.

pg.gateway.elasticsearch.sniff.timeInterval

Mandatory

This property enables adding the newly added Elasticsearch cluster nodes to existing REST client in a specified time interval in milliseconds.

Default value: 60000

3. Restart API Gateway for the HTTP client to take effect.

Note:

If hosts and ports are changed for Elasticsearch then you have to update the appropriate Elasticsearch configuration for Kibana separately and restart the Elasticsearch server as well as Kibana.

You can also externalize the Elasticsearch tenant ID and configuration by using a master configuration file. For details, see “[Externalizing Configurations](#)” on page 62.

Troubleshooting Tips: External Elasticsearch

Kibana dashboard does not work after configuring external Elasticsearch.

Kibana dashboard is not working after configuring external Elasticsearch.

Resolution:

1. Set the **apigw.kibana.autostart** setting in the **uiconfiguration.properties** file to false.
This step prevents the Elasticsearch configuration being reset to *localhost*.
2. In the **kibana.yml** file found in the *SAGInstallDir\profiles\IS_default\apigateway\dashboard\config* location, update the following field to the corresponding external Elasticsearch URL:
`elasticsearch.hosts: ["http://localhost:9240"]`

Replace **localhost** with the system name or IP address where Elasticsearch is running and **9240** with the corresponding Elasticsearch port number.

3. In the **uiconfiguration.properties** file found in the *SAGInstallDir\profiles\IS_default\apigateway\config* folder, provide the same external Elasticsearch URL in the following field:
`apigw.es.url=http://localhost:11140`

This must be same as the value provided in the previous step.

4. Start Kibana by running the **kibana.bat** (Windows) or **kibana.sh** (Linux) file from the following location: *SAGInstallLocation\profiles\IS_default\apigateway\dashboard\bin*.

Note:

As the Kibana autostart is disabled in the first step, you must start Kibana manually everytime.

Alternative resolution:

- You can externalize the external Elasticsearch and Kibana configurations. For information on externalizing, see “[Externalizing Configurations](#)” on page 62.

Connecting to an External Kibana

Considerations when you configure an External Kibana:

- Ensure the Kibana version is compatible with the Elasticsearch version as Kibana and Elasticsearch have a one-to-one mapping. For details on version compatibility, see [Support Matrix](#).
- Turn off Kibana auto start in one of the following ways:
 - By using Externalized configuration files. For details, see “[Using the Externalized Configuration Files](#)” on page 62. Software AG recommends using this configuration.
 - By setting the property **apigw.kibana.autostart** to false located in `C:\API Gateway instance\profiles\IS_default\apigateway\config\uiconfiguration.properties`.

You can have one of the following Kibana configurations:

- Default Kibana connected to API Data Store.
- External Kibana connected to API Data Store.

You can configure this setup as follows:

For an external Kibana to connect to API Data Store you have to configure the following properties in the `kibana.yml` file where you have installed the external Kibana.

Note:

The default index is **.kibana** and it is not configured by users.

Property	Description
<code>server.port: port number</code>	Specifies which server port to use. Example: 9405
<code>server.host: server host IP address or host name</code>	Specifies the host to bind the server to. The default value is <code>localhost</code> , which means the remote machines will not be able to connect. To allow connections for remote users you must set this parameter to a non-loopback address. Example: "0.0.0.0"
<code>server.basePath: server path of the proxy</code>	Specifies the proxy setting to render the charts from the external Kibana in API Gateway UI. The server path you specify must not end with a /. Value: "/apigatewayui/dashboardproxy"
<code>elasticsearch.hosts: http://hostname:port</code>	Specifies the URLs of the Elasticsearch instance to use for all your queries. Example: ["http://localhost:9240"]

You can find these values in the `kibana.yml` file of the internal Kibana installed location `C:\API Gateway instance\profiles\IS_default\apigateway\dashboard\config`. You can copy these values in the `kibana.yml` file of the external Kibana in the respective installed location.

If you are using a Kibana version different than the one shipped with API Gateway that is compatible with the Elasticsearch version, you have to specify the Kibana version in the `config.json` file located at `C:\API Gateway instance\IntegrationServer\instances\default\packages\WmAPIGateway\config\resources\kibana\config\7\`. For details on version compatibility, see [Support Matrix](#).

- Embedded Kibana connected to External Elasticsearch to store all API Gateway assets then configure the following:

Open the `kibana.yml` file located at `C:\API Gateway instance\profiles\IS_default\apigateway\dashboard\config` and specify the external Elasticsearch host and port details, which the Kibana has to connect to, as follows:

```
# The Elasticsearch instance to use for all your queries.  
elasticsearch.hosts: ["http://host_name:port"]
```

- External Kibana connected to an external Elasticsearch.

You can configure this setup by using externalized configuration files. For details, see “[Using the Externalized Configuration Files](#)” on page 62.

Note:

When using external Elasticsearch and external Kibana the startup of the components must follow the following order:

1. Start Elasticsearch and verify the cluster health.
2. Start the API Gateway service.
3. After API Gateway is started and available, start Kibana.

Externalizing Configurations

API Gateway can be configured on startup through a set of external configuration files. These files are used to manage and provision configurations from a centralized location. The externalized configuration can be specified either within a single file providing all the necessary configurations or multiple files for individual configurations. By using multiple files the configurations can be split. For example, the cluster configuration can be specified separately from the Kibana or Terracotta configuration. The externalized configuration can be in YAML or properties files format.

To consider the externalized configuration files during the API Gateway startup, the configuration files need to be referenced in the master configuration file, `config-sources.yml`.

Both the master configuration and external configuration files are located in the `SAGInstallDir\IntegrationServer\instances\instance_name\packages\WmAPIGateway\resources\configuration` folder.

Using the Externalized Configuration Files

The API Gateway administrator provides configuration settings in one or more external configuration files and creates the master configuration file listing the external configuration files. On startup, API Gateway reads config-sources.yml file and loads all the external configuration source files that it references. The settings in the externalized configuration files override the respective internal configuration settings (such as uiconfiguration.properties, server.cnf). Once the API Gateway configuration space is updated, the rest of the API Gateway package gets loaded with the updated configuration settings.

Note:

For settings that are not given in the externalized configuration files, API Gateway use the settings given in the internal configuration files.

The below sample externalized configuration file template contains the configuration settings that the API Gateway administrator wants to externalize. The given external configuration settings overwrite the respective internal configuration settings. For the configuration settings that are not specified in the externalized configuration file, the settings given in the respective internal configuration files take precedence.

```
apigw:
  elasticsearch:
    .....
  analyticsDataStore:
    .....
  kibana:
    .....
  filebeat:
    .....
  cluster:
    .....
  users:
    .....
  masterpassword:
    .....
  ui:
    .....
  alias:
    .....
```

Elasticsearch Configuration

Note:

Install and run Elasticsearch, version 8.2.3, if you are configuring an external Elasticsearch.

The Elasticsearch configuration and details section contains all the necessary properties for an Elasticsearch HTTP client using which API Gateway connects to either an externally running Elasticsearch server or to the Elasticsearch-powered API Data Store in API Gateway. The key configurations are as follows:

- **tenantId.** The API Gateway tenant id, using which the Elasticsearch indices are created for that tenant.
- **hosts.** A comma separated list of Elasticsearch instances. Example: host1:9200,host2:9240.

- **autostart.** *Optional.* If the value is not provided, by default it would be `false`. API Gateway would connect to the given external Elasticsearch hosts. If the value is set to `true`, API Gateway automatically starts the API Data Store. In this case, the hosts should point to API Data Store host and port. The default host for the API Data Store is `localhost:9240`.
- **http.** The basic authentication credentials and HTTP-connection specific properties.
- **https.** If the `enabled` property within `https` is set to `true`, API Gateway uses the other `https` properties to connect to the configured hosts.
- **sniff.** These properties help in adding a new Elasticsearch node to the Elasticsearch cluster.
- **outboundproxy.** Outbound proxy settings between API Gateway and Elasticsearch.
- **clientHttpResponseBodySize.** Maximum Response payload size in MB.

A sample Elasticsearch configuration is as follows:

```
apigw:
  elasticsearch:
    tenantId: apigateaway
    hosts: localhost:9200
    autostart: false
    http:
      username: elastic
      password: changeme
      keepAlive: true
      keepAliveMaxConnections: 10
      keepAliveMaxConnectionsPerRoute: 100
      connectionTimeout: 1000
      socketTimeout: 10000
      maxRetryTimeout: 100000
    https:
      enabled: false
      keystoreFilepath: C:/softwares/elasticsearch/config/keystore-new.jks
      truststoreFilepath: C:/softwares/elasticsearch/config/truststore-new.ks
      keystoreAlias: root-ca
      keystorePassword: 6572b9b06156a0ff778c
      truststorePassword: manage
      enforceHostnameVerification: false
    sniff:
      enable: false
      timeInterval: 1000
    outboundProxy:
      enabled: false
      alias: somealias
    clientHttpResponseBodySize: 1001231
```

Analytics Data Store Configuration

Note:

Install and run Elasticsearch, version 8.2.3, if you are configuring an external Elasticsearch.

The analytics data store configuration supports setting the analytics data store to store analytics data separately from the core data. It also contains the required properties to establish a

communication channel between API Gateway and analytics data store to exchange data. The key configurations are as follows:

- **tenantId.** The API Gateway tenant id, using which the Elasticsearch indices are created for that tenant.
- **hosts.** A comma separated list of Elasticsearch instances. Example: host1:9200, host2:9240.
- **http.** The basic authentication credentials and HTTP-connection specific properties.
- **https.** The HTTPS-connection specific properties. If the enabled property within https is set to true, API Gateway uses the other https properties to connect to the configured hosts.
- **sniff.** These properties help in adding a new Elasticsearch node to the Elasticsearch cluster.
- **outboundproxy.** Outbound proxy settings between API Gateway and Elasticsearch.
- **clientHttpResponseBodySize.** Maximum response payload size in MB.

A sample analytics data store configuration is as follows:

```
---
apigw:
  analyticsDataStore:
    tenantId: default
    hosts: localhost:9241
    http:
      username: analyticsdatastore
      password: changeme
      keepAlive: true
      keepAliveMaxConnections: 60
      keepAliveMaxConnectionsPerRoute: 20
      connectionTimeout: 2000
      socketTimeout: 40000
      maxRetryTimeout: 200000
    https:
      enabled: false
      keystoreFilepath: C:/softwares/analyticsdatastore/config/demouser-keystore.jks
      truststoreFilepath: C:/softwares/analyticsdatastore/config/truststore.jks
      keystoreAlias: cn=demouser
      keystorePassword: 6572b9b06156a0ff778c
      truststorePassword: 2c0820e69e7dd5356576
      enforceHostnameVerification: false
    sniff:
      enable: false
      timeInterval: 70000
    outboundProxy:
      enabled: false
      alias: somealias
    clientHttpResponseBodySize: 200
```

Kibana Configuration

Note:

Install Kibana, version 8.2.3, if configuring an external instance of Kibana.

The Kibana configuration supports setting the Kibana server URL, which can point to either the one that is run by API Gateway or any externally running server. It also contains the SSL certificate related settings that would be used to connect to the SSL protected Elasticsearch server. The key configurations are as follows:

- `dashboardInstance`. The Kibana server URL in the format `scheme://hostname:port`. Example: `http://vmabc:5601`.
- `autostart`. *Optional*. If the value is not provided, by default it would be `false`. API Gateway would connect to the given external Kibana server. If the value is set to `true`, API Gateway automatically starts the internal Kibana server. In this case, the hosts should point to internal Kibana server host and port. The default value is `http://localhost:9405`.
- `sslCA`. A list of paths to the PEM file for the certificate authority for the Elasticsearch instance.
- `sslCert`. The path to the PEM format certificate for SSL client authentication.
- `sslKey`. The client certificate key used for client authentication. These files are used to verify the identity of Kibana to the Elasticsearch server when it is SSL protected.

A sample Kibana configuration is as follows:

```
apigw:
  kibana:
    dashboardInstance: http://localhost:9405
    autostart: true
    elasticsearch:
      sslCA: C:/softwares/elasticsearch/config/SAG-B1HPWT2.pem
      sslCert: C:/softwares/elasticsearch/config/SAG-B1HPWT2.crt
      sslKey: C:/softwares/elasticsearch/config/SAG-B1HPWT2.key
```

Filebeat Configuration

The Filebeat configuration supports configuring the SSL certificate related settings that are used to connect to the SSL protected Elasticsearch server. The key configurations are as follows:

- `sslCA`. A list of paths to the PEM file for the certificate authority for the Elasticsearch instance.
- `sslCert`. The path to the PEM format certificate for SSL client authentication.
- `sslKey`. The client certificate key used for client authentication. These files are used to verify the identity of Kibana to Elasticsearch server when it is SSL protected.

A sample Filebeat configuration is as follows:

```
apigw:
  filebeat:
    output:
      elasticsearch:
        sslCA: C:/softwares/elasticsearch-version/config/SAG-B1HPWT2.pem
        sslCert: C:/softwares/elasticsearch-version/config/SAG-B1HPWT2.crt
        sslKey: C:/softwares/elasticsearch-version/config/SAG-B1HPWT2.key
```

Cluster Configuration

This section describes the cluster configuration steps for both peer-to-peer clustering using Apache Ignite or Terracotta Server Array.

Peer-to-peer cluster configuration based on Apache Ignite technology

With peer-to-peer clustering the cluster members synchronize by using distributed caches. Therefore, each API Gateway server requires two ports. Hence, these ports have to be accessible for communication. If you have deployed the cluster on a traditional on-premise installation with real or virtual machines then the firewall must be open to the server ports.

In order to form a cluster, each cluster member requires information about possible other servers. If the cluster is deployed on a traditional on-premise installation, or as Docker containers, then you have to specify the names of the participating hosts. This is an initial member list. If you want to scale the cluster, you can include servers with host name that is not in the initial list. Such a server contacts any server in the initial list, and then negotiates joining the cluster.

If the cluster is deployed in a Kubernetes environment, the cluster specification requires the Kubernetes namespace and the name of the Kubernetes service that exposes the API Gateway deployment. New cluster members are then detected by checking the endpoints attached to the Kubernetes service.

In order to analyze the service endpoints you must start the API Gateway deployment with a service account with specific permissions. For details about the service account permissions, see ["API Gateway Clustering on Kubernetes" on page 692](#).

The cluster configuration contains peer-to-peer cluster and the related Elasticsearch cluster settings.

Note:

The cluster configuration for Elasticsearch clustering settings is only applicable to API Data Store.

The key configurations are as follows:

- `aware`, `name`, `sessTimeout`, `actionOnStartupError`. These parameters have the same meaning as in Terracotta clustering. However, note that in this case they will not be applied to the server `watt` properties.
- `ignite.discoveryPort`. The port that is used to discover the participating cluster members. Each cluster member opens this port in order to be discoverable. Each server tries to contact other servers through this port. The `discoveryPort` must be different from the `communicationPort`.
- `ignite.communicationPort`. The port that is used to communicate between distributed caches. Each server opens this port to receive information and distribute cache entries to other cluster members. The `communicationPort` must be different from the `discoveryPort`.
- `ignite.portRange`. The number of ports to include in the range available to multiple cluster members on the same host. The default value of the `portRange` parameter is 0.

The `portRange` parameter applies to both the `discoveryPort` and the `communicationPort`. When you specify a non-zero `portRange` value, ensure that the resulting ranges for the discovery port and the communication port do not overlap.

Generally, each cluster member is deployed on its own host. However, for trial or demo purposes you might have multiple cluster members on the same host. In such a situation, each server requires its own discovery and communication port, and must know the other servers' ports. You can use the `portRange` parameter to configure this scenario. The `portRange` is a number that defines the size of a port range. For example, if you configure the port with port number 10100 and `portRange` as 5, the resulting port range that is available is 10100 to 10105. When a server starts, it picks an unused port from the range 10100 to 10105, and uses the port to communicate with the other servers. Another server on start up can use an unused port from the same port range to communicate with the other servers. This allows all servers to use the same cluster configuration.

- `ignite.hostnames`. The list of initial host names participating in the cluster. The list is comma-separated.
- `ignite.k8sServiceName`. The name of the service that exposes the API Gateway deployment, in case of a Kubernetes cluster.
- `ignite.k8sNamespace`. The name of the Kubernetes namespace within which the API Gateway is deployed, in case of a Kubernetes cluster.

A sample cluster configuration for an on-premise or Docker deployment is as follows:

```
apigw:  
  cluster:  
    aware:      true  
    name:       APIGatewayCluster  
    sessTimeout: 60  
    actionOnStartupError: standalone  
    ignite:  
      hostnames:      daeirnd33974,daeirnd33562,daeirnd33974  
      discoveryPort: 10100  
      communicationPort: 10400
```

A sample cluster configuration for Kubernetes cluster is as follows:

```
apigw:  
  cluster:  
    aware:      true  
    name:       APIGatewayCluster  
    sessTimeout: 60  
    actionOnStartupError: standalone  
    ignite:  
      k8sServiceName:   api-gateway-service  
      k8sNamespace:    api-gateway-namespace  
      discoveryPort: 10100  
      communicationPort: 10400
```

A sample cluster configuration for multiple servers on the same host is as follows:

```
apigw:  
  cluster:  
    aware:      true
```

```

name:          APIGatewayCluster
sessTimeout: 60
actionOnStartupError: standalone
ignite:
  hostnames:      localhost
  discoveryPort: 10100
  communicationPort: 10400
  portRange:      3

```

Cluster configuration using Terracotta Server Array

Note:

Install and run the Terracotta server (a version that is compatible with API Gateway 10.11) for clustering API Gateway instances.

The cluster configuration contains the Terracotta cluster settings. The key configurations are as follows:

- aware, name, tsaUrls, sessTimeout, actionOnStartupError. All are required Terracotta cluster settings in the server watt properties.
- terracottaLicenseFileName. The Terracotta server license file name. The file should be present in the folder `SAGInstallDir/common/conf`. API Gateway uses this file to join the Terracotta cluster.

A sample Cluster configuration is as follows:

```

apigw:
  cluster:
    aware: true
    name: APIGatewayTSAcluster
    tsaUrls: VMYAI105BVT06:9510
    terracottaLicenseFileName: terracotta-license.key
    sessTimeout: 20
    actionOnStartupError: standalone

```

For `terracottaLicenseFileName` parameter a valid license file should be present in the `SAGInstallDir/common/conf` location, otherwise the parameter is ignored.

Note:

When cluster settings are given in the configuration files, the API Gateway server, on startup, updates the internal settings with the values from the configuration files but the API Gateway node does not join the cluster. You must restart the server for the cluster settings to become effective and for the API Gateway node to join the cluster.

User Configuration

The user configuration supports configuring user and group information on the API Gateway server. By default, the local users created are assigned to the **Everybody** group. The key configurations are as follows:

- `firstName`. First name of the user.

- `lastName`. Last name of the user.
- `password`. Password of the user.
- `emailAddresses`. List of email addresses of the user.
- `active`. Active status of the user.
- `language`. Preferred language of the user.
- `groups`. Names of the groups the user belongs to.

A sample user configuration is as follows:

```
---  
apigw:  
  users:  
    tstk:  
      firstName: "stark"  
      lastName: "Koop"  
      password: "oops"  
      emailAddresses: [ tstk@sag.com, tstk@sag.co.uk ]  
      active: true  
      groups:  
        - "group1"  
        - "group2"  
    fred:  
      firstName: "Fred"  
      lastName: "Barker"  
      password: "oops"  
      emailAddresses: [ fred@sag.com ]  
      active: true  
      groups: [group1,group2]  
    bob:  
      firstName: "Bob"  
      lastName: "Tate"  
      password: "oops"  
      emailAddresses: [ bob@sag.com ]  
      active: true
```

Master Password Configuration

In API Gateway, you would be using passwords while enforcing security related policies, while connecting to various destinations such as, API Portal, CentraSite, Email, and SNMP, while configuring the security-related aliases, configuring outbound proxy servers, and so on.

To protect these passwords API Gateway encrypts them. By default, it encrypts them using Password-Based Encryption (PBE) standard, also known as PKCS5. This encryption method requires the use of an encryption key or master password that you specify.

The master password configuration supports configuring of encryption key or master password on the API Gateway server. The key configurations are as follows:

- `expiry`. Expiry interval for the master password.
- `oldPassword`. Current password of the user.

- newPassword . New password of the user.

A sample master password configuration is as follows:

```
---
apigw:
  masterpassword:
    expiry: "60"
    oldPassword: "old1"
    newPassword: "new1"
```

UI Configuration

The UI configuration supports configuring HTTP and HTTPS port on the API Gateway server.

The key HTTP and HTTPS configurations are as follows:

Parameters	Description
maxHttpHeaderSize This parameter is applicable to HTTP and HTTPS.	Specifies the maximum size of the request and response HTTP header, specified in bytes. If not specified, this attribute is set to 8192 (8 KB).
connectionTimeout This parameter is applicable to HTTP.	This property specifies the time, in milliseconds, after which the connection times out.
server This parameter is applicable to HTTP and HTTPS.	This property overrides the server header for the HTTP response.
maxSpareThreads This parameter is applicable to HTTP and HTTPS.	Specifies the maximum number of request processing threads the connector creates, which determines the maximum number of simultaneous requests that API Gateway server can handle.
disableUploadTimeout This parameter is applicable to HTTP and HTTPS.	This flag allows the servlet container to use a different, usually longer connection timeout during data upload.
minSpareThreads This parameter is applicable to HTTP and HTTPS.	Specifies the minimum number of threads always kept running. This includes both active and idle threads.
redirectPort This parameter is applicable to HTTP and HTTPS.	When a SSL port is required by the client, the request is redirected to this port number.

Parameters	Description
acceptCount This parameter is applicable to HTTP and HTTPS.	Specifies the maximum queue length for incoming connection requests when all possible request processing threads are in use. Any requests received when the queue is full is refused. The default value is 100.
port This parameter is applicable to HTTP and HTTPS.	Specifies the TCP port number on which this connector creates a server socket and awaits incoming connections.
enableLookups This parameter is applicable to HTTP and HTTPS.	Set to <code>true</code> if you want calls to <code>request.getRemoteHost()</code> to perform DNS lookup in order to return the actual host name of the remote client. Set to <code>false</code> to skip the DNS lookup and return the IP address in string form instead (thereby improving performance).
enabled This parameter is applicable to HTTP and HTTPS.	Enable or disable API Gateway web-app port.
alias This parameter is applicable to HTTP and HTTPS.	Identifies a proxy server and a port on the server through which you want to route requests.
sslProtocol This parameter is applicable to HTTPS.	Specifies the SSL protocols to use. A single value may enable multiple protocols. For more information, see JVM documentation. If not specified, the default is <code>TLS</code> .
sslEnabled This parameter is applicable to HTTPS.	Use this attribute to enable SSL traffic on a connector. The default value is <code>false</code> . To turn on SSL handshake or encryption or decryption on a connector set this value to <code>true</code> . When turning this value to <code>true</code> , set the scheme and the secure attributes to pass the correct <code>request.getScheme()</code> and <code>request.isSecure()</code> values to the servlets.
keystoreType This parameter is applicable to HTTPS.	The keystore file type to use for the server certificate. If not specified, the default value is <code>JKS</code> .
keystoreFile This parameter is applicable to HTTPS.	The pathname of the keystore file where the server certificate is stored. By default, the pathname is the file <code>.keystore</code> in the operating system home directory of the user that is running Tomcat. If the <code>keystoreType</code> doesn't need a file use " " (empty string) for this parameter.
keystorePass This parameter is applicable to HTTPS.	The password used to access the specified keystore file.

Parameters	Description
scheme This parameter is applicable to HTTPS.	Set this attribute to the name of the protocol you wish to have returned by calls to <code>request.getScheme()</code> .
secure This parameter is applicable to HTTPS.	Set this attribute to <code>true</code> if you wish to have calls to <code>request.isSecure()</code> to return <code>true</code> for requests received by this connector.

For more information on HTTP and HTTPS port configuration, see [Apache Tomcat documentation](#).

A sample UI configuration is as follows:

```
---
apigw:
  ui:
    http:
      maxHttpHeaderSize: "8192"
      connectionTimeout: "20001"
      server: "SoftwareAG-Runtime"
      maxSpareThreads: "78"
      disableUploadTimeout: "true"
      minSpareThreads: "23"
      redirectPort: "19073"
      acceptCount: "102"
      port: "11072"
      enableLookups: "false"
      enabled: "true"
      alias: "defaultHttp"
    https:
      maxHttpHeaderSize: "8192"
      server: "SoftwareAG-Runtime"
      maxSpareThreads: "72"
      disableUploadTimeout: "true"
      minSpareThreads: "22"
      acceptCount: "104"
      port: "11075"
      enableLookups: "false"
      enabled: "true"
      alias: "defaultHttps"
      sslProtocol: "tls"
      sslEnabled: "true"
      keystoreType: "xxv"
      keystoreFile: "Test2"
      keystorePass: "geheim"
      scheme: "xScheme"
      secure: "true"
```

Aliases Configuration

An alias in API Gateway holds environment-specific property values to use in policy routing configuration. API Gateway aliases can be defined through external configuration. The alias configuration supports configuring aliases on the API Gateway server.

The key configurations are as follows:

- **name**. A unique name for the alias.
- **description**. A description of the alias.
- **type**. Type of the alias. The following aliases configuration are supported:
 - simple
 - endpoint
 - httpTransportSecurityAlias
 - soapMessageSecurityAlias
 - samlIssuerAlias
 - authServerAlias
 - webmethodsAlias
 - transformationAlias
 - serviceRegistryAlias
 - clientMetadataMapping
 - awsConfigurationAlias
 - isConfigurationAlias
- **owner**. Owner of the alias.

Sample configuration of the supported aliases are as follows:

simple alias configuration:

```
apigw:  
  aliases:  
    simpleAlias1:  
      type: "simple"  
      value: "vmspar02w"
```

endpoint alias configuration:

```
apigw:  
  aliases:  
    endpointAlias1:  
      type: "endpoint"  
      endPointURI: "http://vmspar02w:9998"  
      connectionTimeout: 30  
      readTimeout: 30  
      suspendDurationOnFailure: 0  
      optimizationTechnique: "None"  
      passSecurityHeaders: false  
      keystoreAlias: "ksAlias"
```

httpTransportSecurityAlias configuration:

```
apigw:
  aliases:
    httpSec1:
      type: "httpTransportSecurityAlias"
      authType: "HTTP_BASIC"
      authMode: "INCOMING_HTTP_BASIC_AUTH"
      httpAuthCredentials:
        userName: "Bob"
        password: "R3Vlc3NjdA=="
        domain: "EUR"
```

isConfigurationAlias configuration:

```
apigw:
  aliases:
    myIsAlias:
      type: "isConfigurationAlias"
      url: "http://localhost:5555"
      username: "Administrator"
      password: "bXlwYXNz"
      keystoreAlias: "DEFAULT_IS_KEYSTORE"
      keyAlias: "ssos"
      packageName: "WmAPIGateway"
      folderName: "test2"
      importSwaggerBasedOnTags: "false"
      enableMTOM: "true"
      enforceWSICompliance: "true"
      validateSchemaWithXerces: "true"
      contentModelComplianceForWSDL: "Lax"
```

authServerAlias configuration:

```
apigw:
  aliases:
    testAuthServer:
      type: "authServerAlias"
      description: "Test Auth server"
      tokenGeneratorConfig:
        expiry: "0"
        accessTokenExpInterval: "3600"
        authCodeExpInterval: "600"
        algorithm: "null"
      supportedGrantTypes: ["authorization_code","client_credentials","implicit"]
      authServerType: "EXTERNAL"
      localIntrospectionConfig:
        issuer: "testIssuer"
        trustStoreAlias: "DEFAULT_IS_TRUSTSTORE"
        certificateAlias: "sso"
        jwksuri: "http://mytest.com"
        description: "Issuer description"
      remoteIntrospectionConfig:
        introspectionEndpoint: "http://myendpoint"
        clientId: "1234"
        clientSecret: "c2VjcmV0"
        user: "Administrator"
      metadata:
        authorizeURL: "http://softwareag.com/authorize"
        accessTokenURL: "http://softwareag.com/accessToken"
        refreshTokenURL: "http://softwareag.com/authorize/refresh"
      sslConfig:
```

```

keyStoreAlias: "DEFAULT_IS_KEYSTORE"
keyAlias: "ssos"
trustStoreAlias: "DEFAULT_IS_TRUSTSTORE"
discoveryURL: "http://softwareag.com/.well-known/openid-configuration"

```

The properties of the supported aliases are as follows:

Type	Parameters
simple	value. Value of the simple alias.
endpoint	<ul style="list-style-type: none"> ■ <code>endPointURI</code>. The default URI or components of the URI such as service name. ■ <code>connectionTimeout</code>. Time interval (in seconds) after which a connection attempt times out. ■ <code>readTimeout</code>. Time interval (in seconds) after which a socket read attempt times out. ■ <code>suspendDurationOnFailure</code>. Time to suspend the request upon a failure. ■ <code>optimizationTechnique</code>. Type of optimization technique used for SOAP messages. ■ <code>passSecurityHeaders</code>. Boolean value whether to pass security headers or not. ■ <code>keystoreAlias</code>. Keystore alias name that is used for the signing or encryption. ■ <code>keyAlias</code>. Key alias in the particular keyStore . ■ <code>truststoreAlias</code>. Truststore alias name to validate the server certificate.
httpTransportSecurityAlias	<ul style="list-style-type: none"> ■ <code>authType</code>. The type of authentication you want to use while communicating with the native API. <ul style="list-style-type: none"> ■ The authentication types supported are: <code>HTTP_BASIC</code>, <code>NTLM</code>, <code>OAUTH2</code>, <code>KERBEROS</code>, <code>JWT</code>, <code>ALIAS</code>, and <code>REMOVE_INCOMING_HTTP_HEADERS</code>. ■ <code>authMode</code>. Authentication mode to use while communicating with the native API. <ul style="list-style-type: none"> ■ The authentication modes supported are: <code>NEW</code>, <code>INCOMING_HTTP_BASIC_AUTH</code>, <code>INCOMING_WSS_USER</code>, <code>INCOMING_X509</code>, <code>DELEGATE_INCOMING</code>, <code>INCOMING_OAUTH_TOKEN</code>, <code>INCOMING_JWT</code>, <code>TRANSPARENT</code>, and <code>INCOMING_KERBEROS</code>.

Type	Parameters
	<ul style="list-style-type: none"> ■ <code>httpAuthCredentials</code>. Credentials to use for HTTP authentication. The authentication credentials supported are: <ul style="list-style-type: none"> ■ <code>userName</code>. Specify a username to access the native API. ■ <code>password</code>. Specify a password to access the native API. ■ <code>domain</code>. Specify a domain to access the native API. ■ <code>kerberosCredentials</code>. Credentials to use for Kerberos authentication. The authentication credentials supported are: <ul style="list-style-type: none"> ■ <code>clientPrincipal</code>. A unique identity to which Kerberos can assign tickets. ■ <code>clientPassword</code>. Password for the client principal. ■ <code>servicePrincipal</code>. A unique identifier of a service instance. ■ <code>servicePrincipalNameForm</code>. The format in which you want to specify the principal name of the service that is registered with the principal database. <code>servicePrincipalNameForm</code> value can be <code>hostbased</code> or <code>username</code>. ■ <code>requestDelegateToken</code>. Boolean value whether the token needs to be delegated or not. ■ <code>oauth2Token</code>. OAuth2 token to use for authentication.
<code>transformationAlias</code>	<ul style="list-style-type: none"> ■ <code>fileName</code>. Name of the file. ■ <code>content</code>. Content of the file.
<code>serviceRegistryAlias</code>	<ul style="list-style-type: none"> ■ <code>endpointURI</code>. Endpoint to use to communicate with the service registry. ■ <code>heartBeatInterval</code>. API Gateway pings the service registry on the configured interval for every API. ■ <code>username</code>. Username to use in the basic authentication when communicating with the service registry. ■ <code>password</code>. Password to use in the basic authentication when communicating with the service registry.

Type	Parameters
	<ul style="list-style-type: none"> ■ <code>keystoreAlias</code>. A keystore is a repository of private key. This keystore contains the private key to use for the SSL communication with the service registry. ■ <code>keyAlias</code>. The key alias is the private key to use for signing when using SSL communication with the service registry. ■ <code>trustStoreAlias</code>. A truststore is a repository of public keys. This truststore contains the public key of the service registry to use for the SSL communication with the service registry. ■ <code>customHeaders</code>. Custom headers to send while communicating with the service registry. ■ <code>discoveryInfo</code>. Contains information like resource path and HTTP method to use while discovering a service in service registry. ■ <code>registrationInfo</code>. Contains information like resource path and HTTP method to use while registering a service in service registry. ■ <code>deRegistrationInfo</code>. Contains information like resource path and HTTP method to use while de-registering a service from service registry. ■ <code>serviceRegistryType</code>. Contains the information about the type of service registry. ■ <code>connectionTimeout</code>. The time interval (in seconds) after which a connection attempt times out while communicating with service registry. ■ <code>readTimeout</code>. The time interval (in seconds) after which a socket read attempt times out while communicating with service registry.
<code>awsConfigurationAlias</code>	<ul style="list-style-type: none"> ■ <code>region</code>. The configured AWS instance region detail. ■ <code>accessKey</code>. The access key ID for the AWS instance. This is used to sign the requests. ■ <code>secretKey</code>. The secret access key for the AWS instance. This is used to sign the requests.
<code>samlIssuerAlias</code>	<ul style="list-style-type: none"> ■ <code>issuerCommunicationMode</code>. Mode of communication to the STS.

Type	Parameters
	<ul style="list-style-type: none"> ■ <code>issuerPolicy</code>. The webMethods Integration Server service name. ■ <code>issuerAuthScheme</code>. The authentication type to use for communicating to STS. ■ <code>issuerAuthMode</code>. Mode of communication to STS. ■ <code>wssCredentials</code>. Credentials for the WSS Username token. ■ <code>kerberosCredentials</code>. Credentials for the Kerberos token. ■ <code>endpoint</code>. The endpoint URI of the STS. ■ <code>samlVersion</code>. SAML version used for authentication. ■ <code>wsTrustVersion</code>. WS-Trust version that API Gateway must use to send the RST to the SAML issuer. ■ <code>appliesTo</code>. Specify the scope for which this security token is required. ■ <code>extendedParameters</code>. Extensions to the <code>wst:RequestSecurityToken</code> element for requesting specific types of keys, algorithms, or key and algorithms, as specified by a given policy in the return tokens. ■ <code>signAndEncryptConfig</code>. Private and public keys to use signature and encryption.
<code>webmethodsAlias</code>	<ul style="list-style-type: none"> ■ <code>serviceName</code>. The webMethods Integration Server service name. ■ <code>runAsUser</code>. The user name you want API Gateway to use to invoke the IS service. ■ <code>complyToISSpec</code>. Set to true, if you want the input and the output parameters to comply to the IS Spec specified.
<code>clientMetadataMapping</code>	<ul style="list-style-type: none"> ■ <code>providerName</code>. Name of the provider. ■ <code>implNames</code>. Map of specification names to the implementation names of the service provider. ■ <code>extendedValuesV2</code>. List of headers that needs to be sent along with the client management request.

Type	Parameters
	<ul style="list-style-type: none"> ■ <code>generateCredentials</code>. Specifies whether API Gateway should generate <code>clientId</code> and <code>client secret</code>. ■ <code>supportedApplicationTypes</code>. List of <code>application_type</code> values supported by the authorization server provider.
<code>soapMessageSecurityAlias</code>	<ul style="list-style-type: none"> ■ <code>authType</code>. Type of authentication. ■ <code>authMode</code>. Mode of authentication ■ <code>wssCredentials</code>. Credentials required for the WSS Username token. ■ <code>kerberosCredentials</code>. Credentials for the Kerberos token. ■ <code>samlIssuerConfig</code>. SAML issuer configuration name. ■ <code>signAndEncryptConfig</code>. Private and public keys to use for signature and encryption.
<code>isConfigurationAlias</code>	<ul style="list-style-type: none"> ■ <code>url</code>. URL of the Integration Server. ■ <code>username</code>. User credentials required to access the Integration Server instance. ■ <code>password</code>. Password required to access the Integration Server instance. ■ <code>keystoreAlias</code>. The text identifier for the Integration Server keystore file. The keystore contains the private keys and certificates (including the associated public keys) of Integration Server. ■ <code>keyAlias</code>. The alias for a specific key in the specified keystore. ■ <code>packageName</code>. Default package name where the alias is published. ■ <code>folderName</code>. Default folder name where the alias is published.
<code>authServerAlias</code>	<ul style="list-style-type: none"> ■ <code>description</code>. Description of the auth server. ■ <code>tokenGeneratorConfig</code>. Specifies the token information that would be added as a bearer token in the HTTP request for client authentication. ■ <code>supportedGrantTypes</code>. Specifies the list of grant types that are supported by API Gateway. The grant types supported are:

Type	Parameters
	<ul style="list-style-type: none"> ■ authorization_code, client_credentials, and implicit. ■ localIntrospectionConfig. Specifies the introspection endpoint to check that access tokens used in client requests are currently active and are valid to invoke the protected resources. ■ sslConfig. Specifies the SSL configuration information.

Consolidating Externalized Configuration Files

You can consolidate the configurations of different inter-components and cluster in a single configuration file.

A sample consolidated configuration file is as follows:

```

apigw:
  elasticsearch:
    tenantId: "apigateway"
    hosts: "localhost:9240"
    autostart: "true"
    http:
      username: ""
      password: "@secure.elasticsearch.http.password"
      keepAlive: "true"
      keepAliveMaxConnections: 10
      keepAliveMaxConnectionsPerRoute: 100
      connectionTimeout: 1000
      socketTimeout: 10000
      maxRetryTimeout: 100000
    https:
      enabled: "false"
      truststoreFilepath: "C:/softwares/elasticsearch-version/config/truststore-new.ks"

      keystoreAlias: "root-ca"
      truststorePassword: "@secure.elasticsearch.http.truststore.password"
      enforceHostnameVerification: "false"
    sniff:
      enable: "false"
      timeInterval: 1000
    outboundProxy:
      enabled: "false"
      alias: "esoutboundproxyalias"
    clientHttpResponseSize: 1001231
  kibana:
    dashboardInstance: "http://localhost:9405"
    autostart: "true"
  elasticsearch:
    sslCA: "C:/softwares/elasticsearch-version/config/SAG-B1HPWT2.pem"
  filebeat:
    output:
      elasticsearch:
        sslCA: "C:/softwares/elasticsearch-version/config/SAG-B1HPWT2.pem"

```

```
analyticsDataStore:
  tenantId: default
  hosts: "localhost:9241"
  http:
    username: analyticsdatastore
    password: changeme
    keepAlive: true
    keepAliveMaxConnections: 60
    keepAliveMaxConnectionsPerRoute: 20
    connectionTimeout: 2000
    socketTimeout: 40000
    maxRetryTimeout: 200000
  https:
    enabled: false
    keystoreFilepath: C:/softwares/analyticsdatastore/config/demouser-keystore.jks
    truststoreFilepath: C:/softwares/analyticsdatastore/config/truststore.jks
    keystoreAlias: cn=demouser
    keystorePassword: 6572b9b06156a0ff778c
    truststorePassword: 2c0820e69e7dd5356576
    enforceHostnameVerification: false
  sniff:
    enable: false
    timeInterval: 70000
  outboundProxy:
    enabled: false
    alias: somealias
  clientHttpResponseSize: 200
cluster:
  aware: "true"
  name: "APIGatewayTSAcluster"
  tsaUrls: "VMYAI105BVT06:9510"
  terracottaLicenseFileName: "terracotta-license.key"
  sessTimeout: "20"
  actionOnStartupError: "standalone"
users:
  tstk:
    firstName: "stark"
    lastName: "Koop"
    password: "oops"
    emailAddresses: [ tstk@sag.com, tstk@sag.co.uk ]
    active: true
    groups:
      - "group1"
      - "group2"
  fred:
    firstName: "Fred"
    lastName: "Barker"
    password: "oops"
    emailAddresses: [ fred@sag.com ]
    active: true
    groups: [group1,group2]
  bob:
    firstName: "Bob"
    lastName: "Tate"
    password: "oops"
    emailAddresses: [ bob@sag.com ]
    active: true
masterpassword:
  expiry: "60"
  oldPassword: "old1"
```

```
    newPassword: "new1"
ui:
  http:
    maxHttpHeaderSize: "8192"
    connectionTimeout: "20001"
    server: "SoftwareAG-Runtime"
    maxSpareThreads: "78"
    disableUploadTimeout: "true"
    minSpareThreads: "23"
    redirectPort: "19073"
    acceptCount: "102"
    port: "11072"
    enableLookups: "false"
    enabled: "true"
    alias: "defaultHttp"
  https:
    maxHttpHeaderSize: "8192"
    server: "SoftwareAG-Runtime"
    maxSpareThreads: "72"
    disableUploadTimeout: "true"
    minSpareThreads: "22"
    acceptCount: "104"
    port: "11075"
    enableLookups: "false"
    enabled: "true"
    alias: "defaultHttps"
    sslProtocol: "tls"
    sslEnabled: "true"
    keystoreType: "xxv"
    keystoreFile: "Test2"
    keystorePass: "geheim"
    scheme: "xScheme"
    secure: "true"
aliases:
  simpleAlias1:
    type: "simple"
    value: "vmspar02w"
  endpointAlias1:
    type: "endpoint"
    endPointURI: "http://vmspar02w:9998"
    connectionTimeout: 30
    readTimeout: 30
    suspendDurationOnFailure: 0
    optimizationTechnique: "None"
    passSecurityHeaders: false
    keystoreAlias: "ksAlias"
  httpSec1:
    type: "httpTransportSecurityAlias"
    authType: "HTTP_BASIC"
    authMode: "INCOMING_HTTP_BASIC_AUTH"
    httpAuthCredentials:
      userName: "Bob"
      password: "R3Vlc3Njda=="
      domain: "EUR"
  myIsAlias:
    type: "isConfigurationAlias"
    url: "http://localhost:5555"
    username: "Administrator"
    password: "bXlwYXNz"
    keystoreAlias: "DEFAULT_IS_KEYSTORE"
```

```

keyAlias: "ssos"
packageName: "WmAPIGateway"
folderName: "test2"
importSwaggerBasedOnTags: "false"
enableMTOM: "true"
enforceWSICompliance: "true"
validateSchemaWithXerces: "true"
contentModelComplianceForWSDL: "Lax"
testAuthServer:
  type: "authServerAlias"
  description: "Test Auth server"
  tokenGeneratorConfig:
    expiry: "0"
    accessTokenExpInterval: "3600"
    authCodeExpInterval: "600"
    algorithm: "null"
supportedGrantTypes: ["authorization_code","client_credentials","implicit"]
authServerType: "EXTERNAL"
localIntrospectionConfig:
  issuer: "testIssuer"
  trustStoreAlias: "DEFAULT_IS_TRUSTSTORE"
  certificateAlias: "sso"
  jwksuri: "http://mytest.com"
  description: "Issuer description"
remoteIntrospectionConfig:
  introspectionEndpoint: "http://myendpoint"
  clientId: "1234"
  clientSecret: "c2VjcmV0"
  user: "Administrator"
metadata:
  authorizeURL: "http://softwareag.com/authorize"
  accessTokenURL: "http://softwareag.com/accessToken"
  refreshTokenURL: "http://softwareag.com/authorize/refresh"
sslConfig:
  keyStoreAlias: "DEFAULT_IS_KEYSTORE"
  keyAlias: "ssos"
  trustStoreAlias: "DEFAULT_IS_TRUSTSTORE"

```

Similarly, you can consolidate separate property files into a single file as shown in the following sample.

```

apigw.elasticsearch.tenantId=apigateway
apigw.elasticsearch.autostart=true
apigw.elasticsearch.hosts=localhost:9240
apigw.elasticsearch.clientHttpResponseSize=1001231
apigw.elasticsearch.http.keepAlive=true
.
.
.

apigw.kibana.dashboardInstance=http://localhost:9405
apigw.kibana.elasticsearch.sslCert=/path/to/your/client.crt
apigw.kibana.elasticsearch.sslKey=/path/to/your/client.key
apigw.kibana.elasticsearch.sslCA=C:/softwares/elasticsearch-version/config/SAG-B1HPWT2.pem
.
.
.
.

apigw.filebeat.output.elasticsearch.sslCert=/path/to/your/client.crt
apigw.filebeat.output.elasticsearch.sslKey=/path/to/your/client.key
apigw.filebeat.output.elasticsearch.sslCA=C:/softwares/elasticsearch-version/config/SAG-B1HPWT2.pem

```

```
.
.
.

apigw.cluster.tsUrls=daeirnd33974:9510
apigw.cluster.actionOnStartupError=standalone
apigw.cluster.name=APIGatewayTSAcluster
apigw.cluster.sessTimeout=20
apigw.cluster.terracottaLicenseFileName=terracotta-license.key
```

Master configuration YAML file and its usage

The master configuration file, config-sources.yml, contains the paths, metadata, and properties for the other configuration files. The master configuration file and the other configuration files should be present in the folder `SAGInstallDir\IntegrationServer\instances\instance_name\packages\WmAPIGateway\resources\configuration`. The master configuration file can contain references to both YAML and Properties file types.

The master configuration file is read by API Gateway on startup. Using this file API Gateway reads the different configurations provided in the folder. If any entry has an invalid file name or path it is ignored but the error is logged into the API Gateway logs.

A sample master configuration file is as follows:

```
#####
# This is the master configuration file which contains the configuration
# source definitions.
#
#----- Sources configuration -----
sources:
#----- YAML file configuration source -----
- type: YAML
  allowEdit: true
  properties:
    location: allExternal-settings.yml
#----- Properties file configuration source -----
#- type: PROPERTIES
#  allowEdit: true
#  properties:
#    location: system-settings.properties
#
#----- END -----
```

The table lists and explains the properties of a configuration file source entry.

Property	Detail
type	<p>Indicates the type of the configuration source. The applicable types are YAML, PROPERTIES and CC_YAML.</p> <ul style="list-style-type: none"> ■ YAML. A YAML configuration file. ■ PROPERTIES. A properties configuration file. ■ cc_YAML. A YAML configuration file, which is reserved for Command Central updates.

Property	Detail
	<p>Note: Command Central support for API Gateway is deprecated and will be removed in future releases.</p>
allowEdit	<p>Indicates whether this file can be updated from API Gateway and is useful for hiding passwords.</p> <p>Valid values are <code>true</code> and <code>false</code>.</p> <ul style="list-style-type: none"> ■ If the value is set to <code>true</code>, it hides the clear text passwords. ■ If the value is set to <code>false</code>, it displays the clear text passwords.
properties	<p>Properties that enable API Gateway to connect to the defined configuration source. For the 10.5 release only the <code>location</code> property is supported.</p> <ul style="list-style-type: none"> ■ <code>location</code>. An absolute or relative path to a component-specific configuration file. In case of relative path, the file would be located relative to the system-defined location <code>SAGInstallDir\IntegrationServer\instances\instance_name\packages\WmAPIGateway\resources\configuration</code>. <p>Note:</p> <ul style="list-style-type: none"> ■ For the CC_YAML file type, the location is fixed as <code>cc-config.yml</code>. This file must not be modified manually as it is updated directly by Command Central. Instead, use the Command Central interfaces to modify this file. ■ Command Central support for API Gateway is deprecated and will be removed in future releases.

Note:

The master configuration filename `config-sources.yml` is system-defined. A file with a different name is not treated as the master configuration file.

Hiding Clear Text Passwords in Configuration Files

To prevent unauthorized users from reading the credentials in the configuration files and other potential threats, the Administrator can enable hiding of such secrets by setting the `allowEdit` flag to `true` in the master configuration file. When `allowEdit` is set to `true` the secret values in the configuration files are stored in the Password manager and the plain text values in the files are replaced with the Password manager keys on API Gateway startup. After this, a user can see only the password keys in the files. On startup, API Gateway would retrieve the passwords for those settings from the Password manager using those keys and hence it is advised not to alter any of the password manager key values in the file. The passwords can be modified at any time and the same are replaced with the Password manager keys in the next API Gateway startup.

This table provides the list of the settings and their respective Password manager keys.

Setting	Password manager key replacement
apigw: elasticsearch: http: username: elastic	@secure.elasticsearch.http.password
apigw: elasticsearch: https: keystorePassword: 6572b9b06156a0ff778c	@secure.elasticsearch.http.keystore.password
apigw: elasticsearch: https: truststorePassword: 6572b9b06156a0ff778c	@secure.elasticsearch.http.truststore.password

Properties File Support for Externalized Configurations

In addition to YAML files, configurations can be saved in Properties files as well. The property names are the same as those in the YAML configuration files. The property names in Properties files are delimited by a "." for forming the property name. For example, the `tenantId` property under `apigw > elasticsearch` in YAML, can be specified as `apigw.elasticsearch.tenantId` in the properties file.

A sample Properties file is as follows:

```
apigw.elasticsearch.tenantId=default
apigw.elasticsearch.autostart=false
apigw.elasticsearch.hosts=vmabc\:9240
apigw.elasticsearch.http.password=admin123
apigw.elasticsearch.http.username=admin
apigw.kibana.dashboardInstance=http://localhost:9405
apigw.kibana.elasticsearch.sslCert=/path/to/your/client.crt
```

Environment Variables Support for Externalized configurations

All the supported externalized configurations can be defined through environment variables. The environment variable names are the same as the property names. Instead of the `.` delimiter the `_` delimiter is used.

The main purpose of the environment variables is to inject a configuration into an API Gateway container during startup.

A sample externalized configuration with environment variable is as follows:

```
apigw_elasticsearch_tenantId=default
apigw_elasticsearch_autostart=false
apigw_elasticsearch_hosts=vmabc\:9240
apigw_elasticsearch_http_password=admin123
apigw_elasticsearch_http_username=admin
apigw_kibana_dashboardInstance=http://localhost:9405
apigw_kibana_elasticsearch_sslCert=/path/to/your/client.crt
```

Configuring Multiple Configuration Files and Its Effects

The master configuration file can have many entries (0 to N) for defining multiple configuration files as configuration sources. When such a file is used to start API Gateway, the configuration values from all the files would be merged into a single effective configuration. If the same configuration value is present in two files, then the value in the file which has a higher preference is given priority. The order of preference is in the reverse order in which they are defined in the master configuration file, that is, the configuration values that are defined in the last configuration file entry would have the highest preference. A sample use case is explained below.

Assume `file1.yml` has the following configurations.

```
apigw:  
  elasticsearch:  
    tenantId: default
```

And, `file2.properties` has the following configurations.

```
apigw.elasticsearch.tenantId=apigateway
```

And, `file3.yml` has the following configurations.

```
apigw:  
  elasticsearch:  
    http:  
      username: admin  
      password: admin123  
  kibana:  
    dashboardInstance: http://localhost:5601
```

Then the combined configuration that becomes effective is as follows.

Effective config.yml configuration:

```
apigw:  
  elasticsearch:  
    tenantId: apigateway  
    http:  
      username: admin  
      password: admin123  
  kibana:  
    dashboardInstance: http://localhost:5601
```

Limitation

If you have defined cluster configuration in the externalized configuration file, on startup the API Gateway server updates the internal settings with the values from the externalized configuration files but the node in the cluster will not be updated. API Gateway server restart is required for the cluster settings to become effective and to join the cluster.

Default Scenario

By default, on start API Gateway reads the master configuration file and loads all the defined configuration source files referenced in the master configuration file. If the master configuration config-sources.yml file does not exist or is not valid, API Gateway falls back to its default behavior, that is, the values defined in the internal configuration file become effective. Similarly, if any of the configuration files does not exist or is not valid, then those files are ignored and API Gateway uses the corresponding internal configuration file. The API Gateway server startup is not blocked in the above scenarios. Instead, the error logs are logged into API Gateway application logs for debugging purpose.

Note:

To view the error logs, enable *Debug* level for the **Externalized Configuration** facility in the logging settings.

A sample log for an API Gateway instance using externalized configurations is as follows:

```
[302]2019-08-16 11:19:02 IST [YAI.0013.8889I] [default][SAG-G43RXF2] Configuration loaded from configuration sources. APIGatewayConfig:  
APIGatewayConfig{elasticsearch=Elasticsearch{tenantId='apigw', hosts='localhost:9200', autostart='null', http=null, https=null, sniff=null, outboundProxy=null, clientHttpResponseSize=null, pendingRestart='null'}, kibana=null, filebeat=null, cluster=null}  
  
[301]2019-08-16 11:19:02 IST [YAI.0013.8889I] [default][SAG-G43RXF2] APIGatewayConfig loaded from ConfigurationSource{type=PROPERTIES, allowEdit=true, properties={location=components.properties}}:  
APIGatewayConfig{elasticsearch=Elasticsearch{tenantId='apigw', hosts='null', autostart='null', http=null, https=null, sniff=null, outboundProxy=null, clientHttpResponseSize=null, pendingRestart='null'}, kibana=null, filebeat=null, cluster=null}  
  
[300]2019-08-16 11:19:02 IST [YAI.0013.8887D] [default][SAG-G43RXF2] Debug: Retrieving configuration from Properties file source: ConfigurationSource{type=PROPERTIES, allowEdit=true, properties={location=components.properties}}  
  
[299]2019-08-16 11:19:02 IST [YAI.0013.8889I] [default][SAG-G43RXF2] APIGatewayConfig loaded from ConfigurationSource{type=YAML, allowEdit=true, properties={location=components.yml}}:  
APIGatewayConfig{elasticsearch=Elasticsearch{tenantId='null', hosts='localhost:9200', autostart='null', http=null, https=null, sniff=null, outboundProxy=null, clientHttpResponseSize=null, pendingRestart='null'}, kibana=null, filebeat=null, cluster=null}  
  
[298]2019-08-16 11:19:02 IST [YAI.0013.8887D] [default][SAG-G43RXF2] Debug: Retrieving configuration from YAML file source: ConfigurationSource{type=YAML, allowEdit=true, properties={location=components.yml}}  
  
[297]2019-08-16 11:19:02 IST [YAI.0013.8887D] [default][SAG-G43RXF2] Debug: Loading configuration from sources: [ConfigurationSource{type=YAML, allowEdit=true, properties={location=components.yml}}, ConfigurationSource{type=PROPERTIES, allowEdit=true, properties={location=components.properties}}]  
  
[293]2019-08-16 11:19:01 IST [YAI.0013.8889I] [default][SAG-G43RXF2] Configuration loaded from configuration sources. APIGatewayConfig:  
APIGatewayConfig{elasticsearch=Elasticsearch{tenantId='apigw', hosts='localhost:9200', autostart='null', http=null, https=null, sniff=null, outboundProxy=null, clientHttpResponseSize=null, pendingRestart='null'}, kibana=null, filebeat=null, cluster=null}
```

```
[292]2019-08-16 11:19:01 IST [YAI.0013.8889I] [default][SAG-G43RZF2] APIGatewayConfig loaded from ConfigurationSource{type=PROPERTIES, allowEdit=true, properties={location=components.properties}}:  
APIGatewayConfig{elasticsearch=Elasticsearch{tenantId='apigw', hosts='null', autostart='null', http=null, https=null, sniff=null, outboundProxy=null, clientHttpResponseSize=null, pendingRestart='null'}, kibana=null, filebeat=null, cluster=null}  
  
[291]2019-08-16 11:19:01 IST [YAI.0013.8887D] [default][SAG-G43RZF2] Debug: Retrieving configuration from Properties file source: ConfigurationSource{type=PROPERTIES, allowEdit=true, properties={location=components.properties}}  
  
[290]2019-08-16 11:19:01 IST [YAI.0013.8889I] [default][SAG-G43RZF2] APIGatewayConfig loaded from ConfigurationSource{type=YAML, allowEdit=true, properties={location=components.yml}}:  
APIGatewayConfig{elasticsearch=Elasticsearch{tenantId='null', hosts='localhost:9200', autostart='null', http=null, https=null, sniff=null, outboundProxy=null, clientHttpResponseSize=null, pendingRestart='null'}, kibana=null, filebeat=null, cluster=null}  
  
[289]2019-08-16 11:19:01 IST [YAI.0013.8887D] [default][SAG-G43RZF2] Debug: Retrieving configuration from YAML file source: ConfigurationSource{type=YAML, allowEdit=true, properties={location=components.yml}}  
  
[288]2019-08-16 11:19:01 IST [YAI.0013.8887D] [default][SAG-G43RZF2] Debug: Loading configuration from sources: [ConfigurationSource{type=YAML, allowEdit=true, properties={location=components.yml}}, ConfigurationSource{type=PROPERTIES, allowEdit=true, properties={location=components.properties}}]
```

system-settings.yml

API Gateway ships with a default configuration file `system-settings.yml`, which contains the default values for the inter-component and cluster configurations. The API Gateway Administrator can start API Gateway with the original (default) configuration values by referring to this file in the master configuration file (`config-sources.yml`) with a higher preference.

For more externalized configuration samples, see <https://github.com/SoftwareAG/webmethods-api-gateway/tree/master/samples/externalconfigurations>.

Troubleshooting tips: Externalized Configuration

The following checkpoints may resolve any issues, you may encounter, while externalizing configurations.

- Check whether the master `config-sources.yml` file exists and it is a valid YAML file.
- Check whether the locations of the configuration files are correctly configured in the master configuration file.
- Check whether the configuration files are valid YAML files.
- Check whether the configuration files contain the right structure and names for the settings as provided in the templates.

- Check whether the configured external instance (Elasticsearch or Kibana) is running before starting API Gateway.
- Check for the logs by enabling debug level of the **Externalized Configuration** facility in the logging settings.

API Gateway Standard and Advanced Editions Capability Matrix

This table lists the capabilities available in the Standard and the Advanced Editions of API Gateway.

Feature	Standard Edition	Advanced Edition
Users and Roles	Administrators	Administrators and API Provider
Administration	Yes	Yes
	<ul style="list-style-type: none"> ■ Ports ■ License management ■ Load balancing ■ Keystore configuration 	
Administration	No	Yes
	<ul style="list-style-type: none"> ■ Extended settings 	
Alias management	No	Yes
Service management	No	Yes
Policy management	Yes	Yes
	<ul style="list-style-type: none"> ■ Threat protection rules 	
Policy management	No	Yes
	<ul style="list-style-type: none"> ■ Global policies ■ Policy templates 	
Export and Import	No	Yes
	<ul style="list-style-type: none"> ■ APIs ■ Global policies 	
Application management	No	Yes
Plans and packages	No	Yes
Analytics	Yes	Yes

Feature	Standard Edition	Advanced Edition
■ Threat protection rule violations		
Analytics	No	Yes
■ Service ■ Applications ■ Consumers		
Clustering and auto synchronization	No	Yes

You can view the type of license by selecting **Username > About**. The information is displayed under Product Information section. You can change the type of license at any time from the Standard Edition to the Advanced Edition.

Note:

For details about API Gateway License management see, *webMethods Integration Server Administrator's Guide*

API Gateway, Elasticsearch, Kibana, and TSA Compatibility Matrix

As stated earlier, API Gateway uses Elasticsearch as its primary data storage. The compatible Elasticsearch versions for the API Gateway versions depend on the API Gateway data type.

API Gateway data can be broadly classified into following four types:

- **Core data.** This type includes APIs, Applications, Policies, Plans, Packages, Administration Settings, Security Configurations (Keystores/Trustores) & Tokens (OAuth/API Keys).
- **Transaction data.** This type includes the runtime transactions events and metrics data.
- **Application logs**
- **Audit logs**

The table below lists the Elasticsearch versions and corresponding Kibana and Terracotta Server Array (TSA) versions that support the storage of core data and transaction data of the available API Gateway versions:

API Gateway	Compatible Elasticsearch versions (Core data level)	Compatible Elasticsearch versions (Transaction data level)	Compatible Kibana version	Compatible TSA version
			8.2.3	4.3.4 through 4.3.9
10.15	8.2.3	All	8.2.3	4.3.4 through 4.3.9
10.11	7.13.0	All	7.13.0	4.3.4 through 4.3.9
10.7	7.7	All	7.7	4.3.4 through 4.3.9
10.5	7.2.0	All	7.2.0	4.3.4 through 4.3.8
10.4	5.6.4, 2.3.2	All	5.6.x, 4.5.x	4.3.4 through 4.3.8
10.3	5.6.4, 2.3.2	All	5.6.x, 4.5.x	4.3.4 through 4.3.8
10.2	5.6.4, 2.3.2	All	5.6.x, 4.5.x	4.3.4 through 4.3.8
10.1	2.3.2	All	4.5.x	4.3.4 through 4.3.8
9.12	2.3.2	All	4.5.x	4.3.4 through 4.3.8

API Gateway 10.15 includes

- Elasticsearch 8.2.3
- Kibana 8.2.3
- Filebeat 8.2.3

Note:

Ensure that you do not use the Java 17 that is inbuilt with the API Data Store for any other purpose.

2 Operating API Gateway

■ Administering API Gateway through API Gateway User Interface	96
■ Starting and Stopping API Gateway	96
■ Data Management	98
■ Monitoring API Gateway	176
■ General Administration Configuration	322
■ Users, Groups, and Teams	425
■ Destination Configuration	469
■ Audit Logging	514
■ System Settings	519
■ Configuring External Accounts	533
■ Configuration Types and Properties	541

Administering API Gateway through API Gateway User Interface

This section describes the various administrative tasks that you can perform through the API Gateway User Interface. The supported administrative tasks are as follows:

- **General configuration:** load balancing, extended settings, service faults, approval settings, proxy server aliases, and URL aliases.
- **Security configuration:** keystores and trustore, ports, SAML issuer, custom assertions, OAuth 2.0, Kerberos settings, JWT, and OpenID provider.
- **Destination configuration:** targets to which the events and performance metrics data is sent.
- **Data Management:** configure the event types and interval at which data can be archived or purged.
- **System setting configuration:** configure system-level configuration parameters and communicate them across nodes in the cluster.
- **External account configuration:** add and configure service registries, AWS accounts, and Integration Server instances.

You can access the API Gateway UI in the following ways:

- Navigate to `http://host:port` where port is the HTTP port of API Gateway configured during installation. For example, `http://host:9072`.
- Log on to Integration Server administration console and click the home button of `WmAPIGateway` package under **Packages > Management** menu.
- Log on to Integration Server administration console and click **API Gateway** under **Solutions** menu.

Starting and Stopping API Gateway

You can start and stop the API Gateway either by using

- scripts or
- user interface

Starting and Stopping API Gateway Using Scripts

You can use the predefined batch files to start API API Gateway. Use the **startup.bat** file to start API Gateway. When you use scripts to start API Gateway, the start process is initiated immediately. You do not have the option to hold the process until all the active sessions end. This method starts API Gateway immediately.

➤ To start and stop API Gateway using scripts

1. Open Command Prompt.
2. Navigate to `C:\SAGInstallDir\IntegrationServer\instances\default\bin`.
3. Run **shutdown.bat** to stop API Gateway.
4. Run **startup.bat** to start API Gateway.

In the default deployment scenario, when you start API Gateway instance, it automatically starts the API Data Store and Kibana components as these components are bundled together. If API Data Store and Kibana fails to start on their own, you have to start these components manually as follows:

- Run the **startup.bat** file located at `SAGInstallDir\instance_name\InternalDataStore\bin` to start the API Data Store.
 - Run the **shutdown.bat** file located at `SAGInstallDir\instance_name\InternalDataStore\bin` to stop the API Data Store.
- Run the **kibana.bat** (Windows) or **kibana.sh** (Linux) file located at `SAGInstallDir\profiles\IS_default\apigateway\dashboard\bin` to start the Kibana.

Shut down or Restart API Gateway Using User Interface

You can restart API Gateway through the Integration Server user interface. You can also restart API Gateway in the Quiesce mode if you want to end all the active sessions before API Gateway restart. This method may take more time to restart (as compared to using scripts) based on the options you select.

➤ To restart API Gateway from Integration Server user interface

1. Open a browser and type `localhost:5555`.

Note:

If you have changed the port number during installation, type the new port number.

This launches the WebMethods Integration Server Administrator page.

2. Click  and select **Shutdown or restart**.



The **Shutdown or restart** page appears.

Shut down or restart



3. In the **Shut Down or Restart** section, select one of the following options:
 - **After all sessions end** and click **Shut Down** or **Restart** button to either shut down or restart API Gateway after all the active sessions are completed.
 - **Immediately** and click **Shut Down** button to either shut down or restart API Gateway immediately.

Important:

You must use the **Immediately** option only if your API Gateway has a clustered configuration. With clustered configuration, all the active sessions are transferred to another API Gateway node. If you select the **Immediately** option with a clustered configuration, all your active sessions are lost.

4. Click one of the following buttons to restart API Gateway:
 - **Shut Down.** Select this option to shut down API Gateway normally.
 - **Restart.** Select this option to restart API Gateway normally.
 - **Restart in Quiesce Mode.** Select this option to restart API Gateway in quiesce mode.

Starting in quiesce mode allows you to run only few specific packages. If you restart API Gateway in quiesce mode, you can only use those packages that are designated to run under quiesce mode. This mode speeds up API Gateway as only selected packages are running. You can exit this mode anytime by clicking the **Exit Quiesce Mode** button.

Data Management

As you create and publish APIs, and administer API Gateway settings as a part of your business, the data generated from API Gateway gradually grows. The growing volume of data must be properly managed for optimal access and usage of the required data.

For an effective data management practice, Software AG strongly recommends that you perform the following :

- **Data housekeeping.** Helps you maintain a hassle-free database and enhances your data access experience.

- **Data backup.** Helps you protect your data and lets you recover the data when there is a data loss.

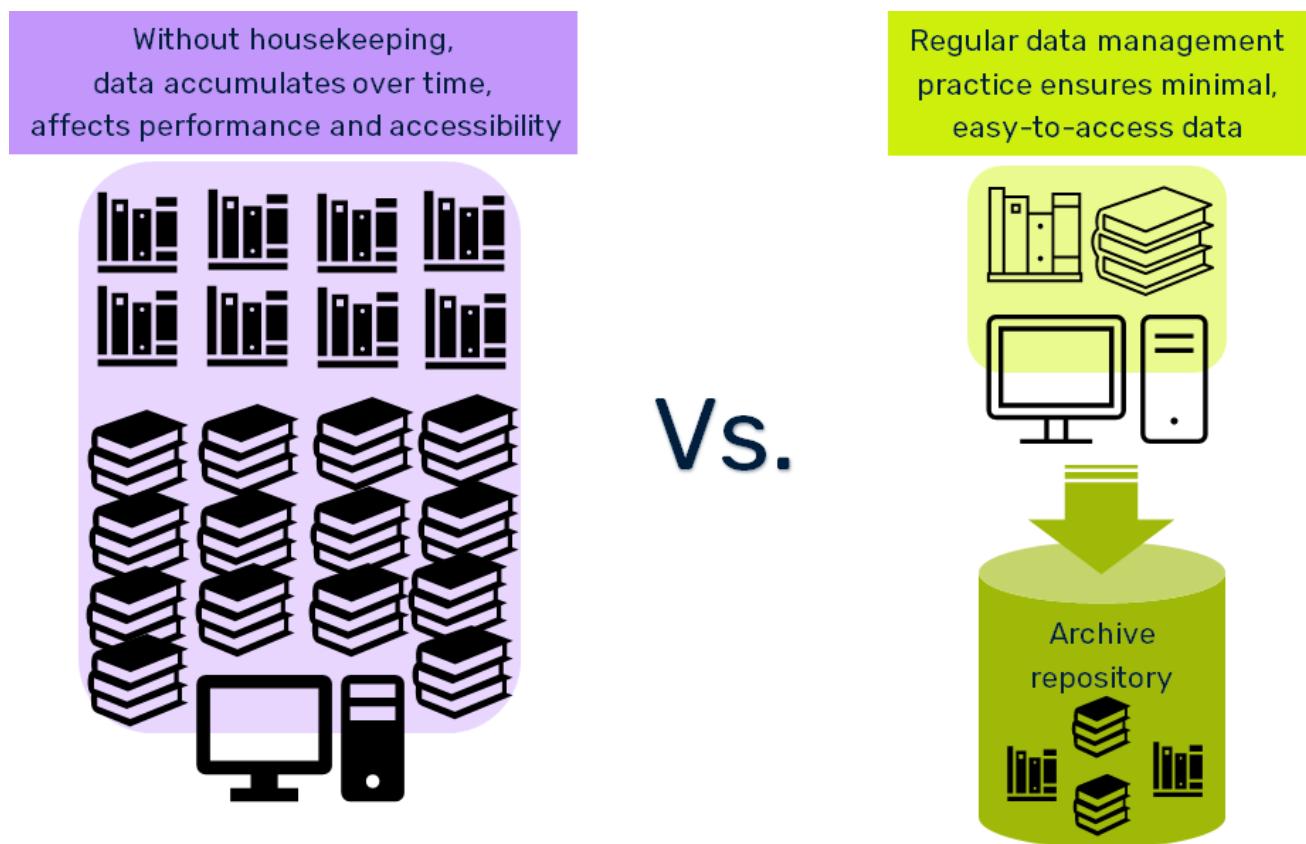
The following sections describe these use cases in detail.

Data housekeeping

Data housekeeping is to limit the data maintained by API Gateway to avoid any capacity or performance problems.

API Gateway records analytics and log data for every API invocation. As a result, the amount of data maintained by API Gateway increases significantly.

Housekeeping involves maintaining the recent analytics and log data in API Gateway. Hence, the older or obsolete data is removed or purged. You can archive the log and analytics data before purging it. The archives can be used to meet long-term data retention requirements for forensic analysis, legal, or compliance purposes.

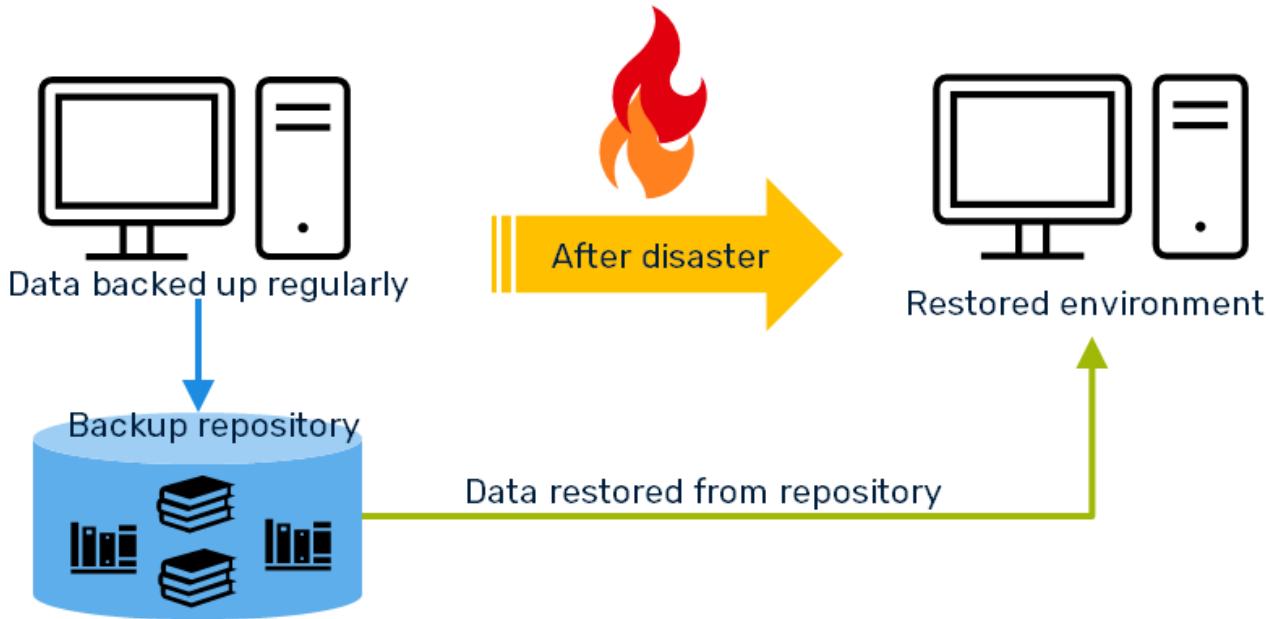


Data backup for data recovery

To protect against accidental loss of data, you should take regular backups of your data.

Data losses can occur in many forms, from hardware failures to data center outages, human errors, security threats, or other unexpected events. Legal compliance requirements or business continuity planning may require measures that allow systematic handling of such data loss scenarios.

Backup refers to the practice of making periodic copies of important data to a separate storage area. If the original data is lost or damaged, then use a copy to recover it.



Types of Data and their Storage

Before we learn how to manage data, it is important to learn the types of data in API Gateway and their storage details. The following table lists the data that you need to manage:

Data type	How is it stored? Where is it stored?	
API Gateway assets	Documents	API Data Store (Elasticsearch)
API Gateway analytics	Documents	API Data Store (Elasticsearch)
Platform data (configurations)	Files	On server nodes (File system)
Platform logs	Files	On server nodes (File system)

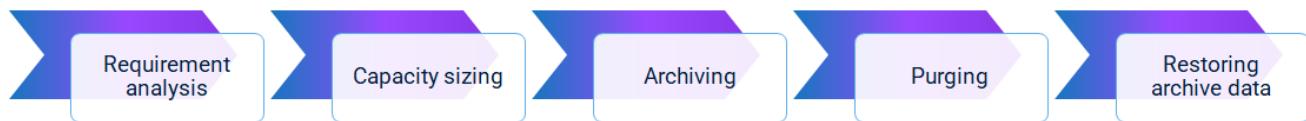
Difference between Archive and Backup

This comparison helps you determine the right option to manage the data in your environment:

Archive	Backup
When is it used?	
Where is it stored?	
To meet long-term data retention requirements for forensic analysis, legal or compliance reasons.	To restore data in event of data loss, damage, error or corruption.
Dedicated storage area that is cost-optimized where the restoring time can be longer.	Safe and foolproof repository. Usually, a network or cloud location.

Data Housekeeping

This section provides guidelines and procedures on how to perform housekeeping of your data and it covers the following:



Requirements

The housekeeping requirements depend on your data size and the number of transactions recorded per day. You need to analyze the existing data and the approximate transactional data that may be recorded on a daily basis. Based on your analysis, you must set the housekeeping parameters. Some of the questions that you need to answer during your analysis are as follows:

- **API transactions considerations:**
 - What is the data retention period to be set? (for example, 90 days, 120 days, and so on).
 - Do you need a copy of the older data for long-term retention? (for legal compliance requirements).
 - Would you ever require restoring data for a particular period? (for forensic analysis).
 - What would be the storage locations if you have to export archived data?
- **Server log considerations.** What is the data retention period to be set? (for example, 90 days, 120 days, and so on).
- **Audit log considerations.** What is the data retention period to be set? (for example, 90 days, 120 days, and so on).

Note:

The above questions are just samples that come across as the most common requirements. You can extend the list with other appropriate questions that will help you determine the complete set of data retention requirements.

Housekeeping approaches for API transaction data

You can use one of the following approaches to housekeep your API transaction data.

- **Archive.** Archive is the process of moving data that is no longer actively used, to a separate storage location for long-term retention. You may require archived data for future reference, forensic analysis, or regulatory compliance.
- **Purge.** Purge is the process of freeing up space in API Data Store by deleting obsolete data, not required by the system (data older than the defined retention-period).

Capacity sizing

If you decide to setup archiving, you must first analyze the capacity sizing requirements.

The archiving process has a few capacity sizing requirements. The size of the memory and the required storage depends on how much data is stored for every archive interval and the data retention period.

Some of the factors to be considered are as follows:

- **What is the archive interval?**

The archiving frequency to be practiced. This factor impacts the memory sizing.

- **Should the archives include API payload?**

Inclusion of API payload details such as the headers, parameters, request, response, and so on impacts memory and disk sizing.

- **What is the archive retention period?**

This factor impacts disk sizing.

You must consider other factors based on your data archival requirements.

Purge does not require any additional capacity sizing.

Archive considerations

Ensure that you:

- Use a dedicated storage area for archiving that is stored outside the API data store.
- Schedule the archive process to be run during non-peak periods as it is generally resource intensive and may affect the performance.
- Perform the process of deleting the older archives after the defined retention period (for example, after 2 years) either manually or using scripts.

Purge considerations

The pre-requisites for the purge process are as follows:

- For long-term data retention needs, to prevent data loss, you must archive the older data before initiating the purge activity.
- You must schedule the purge process to be run during non-peak periods.

CAUTION:

Purge results in irrecoverable loss of data, unless the data is archived.

Archive and purge methods

You can automate the archive and purge operations using REST APIs. Alternatively, you can also archive and purge manually from the API Gateway UI. Software AG strongly recommends you to use APIs and automate the process of archive and purge.

Archive and Purge using API

You can use the APIs provided by API Gateway to archive and purge data. This method is easy and helps you automate the archive and purge operations.

Archive transaction and Audit data

You can archive outdated data (for example, data that is older than a year) for forensic analysis and decision making. Archived data is not the same as backup data. Software AG recommends that you use a proper naming convention for every archive that you create. You can specify the period (like from 1 June to 1 July) and the type of events to be archived. For the list of events that you can archive, see ["List of events that can be archived or purged" on page 114](#).

You can use the following REST API to archive data:

```
curl -X POST -u "Administrator:manage" -H "content-type:application/json" -H
"Accept:application/json"
"http://localhost:5555/rest/apigateway/apitransactions/archives?"time interval"
```

The following API archives data of all event types for the period, 3 June 2021 to 4 June 2021:

```
curl -X POST -u "Administrator:manage" -H "content-type:application/json" -H
"Accept:application/json"
"http://localhost:5555/rest/apigateway/apitransactions/archives?
from=2021-06-03%2000:00:00&until=2021-06-04%2000:00:00&eventType=ALL"
```

You can schedule archiving using cron jobs or any other scheduling methods. Archiving is a resource-intensive operation. You must not schedule it during peak hours.

You can monitor the status of an archive job using the following API:

```
curl -X GET -u "Administrator:manage" -H "content-type:application/json" -H
"Accept:application/json"
"http://localhost:5555/rest/apigateway/apitransactions/jobs/9c0eeefde-dc26-4cb7-b0eb-dfe8f3a8a545"
```

The above command returns the following output, if the archive job completes successfully:

```
{
  "status": "Completed",
  "Filename": "\default-2021-06-14-1623648788446"
}
```

If the archive job fails, the status field in the above output, displays **Failed**. You must configure alerts for failures to get notified about the failed archive jobs. Common reasons for failure include health of the Elasticsearch cluster, load on the system, and so on. You can look into server logs and analyze the failure reasons.

Purging data

You can schedule and automate the purge process. You can also purge data manually through the API Gateway UI. To learn more about how to purge data manually, refer to the ["Archive and Purge using UI" on page 109](#) section. You can purge the following data using commands.

- Analytics data
- Backup snapshots
- Obsolete or empty indexes
- Expired OAuth tokens
- Archive data

Purge Analytics Data

You can purge analytics data based on timeline or size. As an example of timeline based purging, you can purge data older than an year. As an example of size-based purging, you can purge data greater than 100 GB.

Timeline based purging

You can use the following API to purge the analytics data of the specified event type and period:

```
curl -X DELETE -u "Administrator:manage" -H "Accept:application/json"
"http://localhost:5555/rest/apigateway/apitransactions?
action=purge&eventType=eventtype&olderThan=timeline"
```

For the list of events that you can specify with the API, see ["List of events that can be archived or purged" on page 114](#).

The **olderthan** field is the timeline field and can have one of the following values:

Timeline Name Syntax	Example	Usage
Year	<number>Y	<1>Year Purges all data up to last 1 year
Month	<number> M	<1> M Purges all data up to last 1 month

Timeline Name Syntax		Example	Usage
Days	<number>d	<1>d	Purges all data up to last day
Time	<number>h<number>m<number>s	14h30m2s\	Purge all data up to the given time

You can monitor the status of a purge job using the following API:

```
http://localhost:5555/rest/apigateway/apitransactions/jobs/<job_id>
```

If the purge is successful, you get the following output.

```
{
"status": "Completed",
"Filename": "File_name"
}
```

Purge based on size

You can purge data based on size. When the size of an index exceeds the specified 25 GB limit, you must roll over the index. When you roll over an index, a new index is created. When you have new indexes, you can purge the old indexes. For example, if you have set maximum size for analytics data as 300 GB, maximum size of an index to be 25 GB, and if your data grows to 325 GB, then you have 13 indexes and the size of each index is 25 GB. Each index contains a primary and a replica shard. So, when the size of the primary shard of an index equals 12.5 GB, the size of the replica index will also be 12.5 GB. The total size of the index will be 25 GB. Hence, you must check the size of the primary shard of an index to decide whether the index needs to be rolled over.

You must regularly monitor the indexes that need to be purged. For information on calculating index size, see [“Calculating index size” on page 306](#).

If you regularly roll over indexes, it becomes easier to find the oldest indexes and purge them. Purging older and obsolete indexes ensure the quick recovery of disk space.

Perform the following steps to purge an index:

- Find the oldest index using the following API:

```
curl -X GET
http://localhost:9240/_cat/indices/gateway_default_analytics_transactionalevents*?h=i&s=i:desc
```

Note:

The above API returns the list of indexes in descending order of index name. API Gateway follows the pattern, gateway_default_transactionalevents_epoch_00000n, where the date and time is represented in the epoch format and 'n' denotes any number starting from 1, which increments during rollover.

API Gateway returns the following pattern, aliasname_yyyyMMddhhmm-00000n when no target index suffix parameter is provided during roll over. If a target index suffix parameter is provided during rollover, API Gateway returns aliasname_<targetIndexSuffix>

- Delete the index returned in the previous step using the following API:

```
curl -X DELETE http://localhost:9240/indexname
```

3. To ensure that an index is deleted, run the following command with the required index name:

```
curl -v -X GET http://localhost:9240/indexname
```

If the deletion is successful, the above API returns the status code 404.

Purge backup snapshots

Data backups are created to safeguard data in a repository for restoring in case of any disasters. The backup snapshots created over a period of time occupies a considerable disk space. Hence, it is essential to purge backup snapshots that are older than the data retention period.

For information on purging backup snapshots, see [“Deleting a Backup File” on page 145](#).

Purge obsolete or empty indexes

API Gateway may have empty indexes due to roll-over and purge operations. It is essential to cleanup the empty indexes. You can delete an index if there are multiple indexes and the index to be deleted is not a write index. Software AG recommends that you perform the purge operation after the scheduled backups.

You can use the following API to check the documents stored by the indexes:

```
curl -X GET "http://localhost:9240/_cat/indices/  
gateway_default_analytics_transactionalevents*s=docs.count:asc&h=i,docs.count"
```

The API returns the following response:

```
gateway_default_analytics_transactionalevents_202106111722 0  
gateway_default_analytics_transactionalevents_1622725174422-000001 2
```

If an index's response value is more than 0, it implies that index is not empty and must not be deleted.

You can use the following API to check if the indexes are write index:

```
curl -X GET  
"http://localhost:9240/_cat/aliases/gateway_default_analytics_transactionalevents?  
h=i,is_write_index&s=is_write_index:asc"
```

The above API returns the following response:

```
gateway_default_analytics_transactionalevents_1622725174422-000001 false  
# gateway_default_analytics_transactionalevents_202106111722 true
```

If an index has a value **true**, it implies that the index is a write index and should not be deleted.

You can use the following API to delete an index:

```
curl -X DELETE http://localhost:9240/indexname
```

You can schedule the purge operation of indexes using a cron job or some other scheduling method. You can schedule index purging on a daily basis. You can monitor the index delete index by using the following API:

```
curl -v -X GET http://localhost:9240/indexname
```

If the deletion is successful, the above API returns status code 404. You must configure alerts for failed purge jobs. When a purge job fails, you must check the Elasticsearch logs to troubleshoot.

Purge expired OAuth tokens

You can use the following API to delete expired OAuth tokens.

```
https://<>/invoke/pub.oauth:removeExpiredAccessTokens
```

You can schedule the purge operation of indexes using a cron job or some other scheduling method. You can schedule OAuth token purging on a daily basis. You must configure alerts for failed purge jobs. When a purge job fails, you must check the server logs.

Purge Archive Data

You must delete the archive data after it reaches the maximum retention period. There is no API to clear the archive data. You must delete archives manually. You can delete archives on a daily basis.

Important:

API Gateway does not perform the purge operation immediately. Once you initiate the purge process, API Gateway starts to mark the files for deletion. Once all the files are marked, the status of the purge operation shows 100% and this implies that the files are deleted internally. However, the actual disk space which was occupied by the purged files is not freed-up, even after the purge status shows 100%. When the purge status is 100%, it implies that the files are internally deleted and it may take up more time to free up the disk space.

If you want to free up the disk space immediately after initiating the purge process, use the following REST API:

```
http://Elasticsearch_host:Elasticsearch_port/_target_indexes/_forcemerge?  
only_expunge_deletes=true
```

where,

- **<target>**. Comma-separated list of data streams, indexes, and aliases used to limit the request. This parameter supports wildcards (*). To target all data streams and indices, exclude this parameter or use * or _all.
- **only_expunge_deletes**. Boolean parameter. When set to *true*, it expunges only those segments containing document deletions.

Sample command

```
http://localhost:9240/gateway_default_analytics_transactionalevents/_forcemerge?only_expunge_deletes=true
```

You can use the Force Merge API on the following indexes in API Data Store or an Elasticsearch:

- Transaction events (gateway_default_transactionalevents)

- Lifecycle events (gateway_default_lifecycleevents)
- Performance metrics (gateway_default_performancemetrics)
- Monitor events (gateway_default_monitorevents)
- Threat Protection events (gateway_default_threatprotectionevents)
- Policy violation events (gateway_default_policyviolationevents)
- Error events (gateway_default_errorevents)
- Audit events (gateway_default_auditlogs)
- Application logs (gateway_default_log)
- Mediator trace span (gateway_default_mediatortracespan and gateway_default_serverlogtracespans)

To learn more about this API, see [Elasticsearch documentation](#) on Force Merge API.

Creating Rollover of indexes that exceed the maximum size

This section explains the steps to find out the indexes exceeding maximum size and roll them over using REST APIs.

For information about the roll over process using the **apigatewayUtil** script and the roll over conditions, see “[Creating Rollover of an Index](#)” on page 133.

➤ To create rollover of an index that has exceeded the given size

1. Run the following command with the required index name:

```
curl -X POST
  "http://localhost:9240/gateway_tenant_index_name/_rollover/new_index_name"
  -d '{"settings": {"index.number_of_shards":1,
    "index.number_of_replicas":1}, "conditions":{"max_size" : "max_index_size"}}'
```

For example, you can run the following command to roll over the **analytics transactional events** index once it exceeds 12.5 GB:

```
curl -X POST
  "http://localhost:9240/gateway_tenant_analytics_transactionalevents/_rollover/new_index_name"
  -d '{"settings": {"index.number_of_shards":1,
    "index.number_of_replicas":1}, "conditions":{"max_size" : "13421772800b"}}'
```

or, run the following command to roll over the **gateway_default_mediatortracespan** tracer index once it exceed 2.5 GB.

```
curl -X POST
  "http://localhost:9240/gateway_tenant_mediatortracespan/_rollover/new_index_name"
  -d '{"settings": {"index.number_of_shards":1,
    "index.number_of_replicas":1}, "conditions":{"max_size" : "2684354560b"}}'
```

Sample output:

```
{
  "acknowledged": false,
  "shards_acknowledged": false,
  "old_index": "gateway_default_analytics_transactionalevents_202106091512",
  "new_index": "gateway_default_analytics_transactionalevents_202106091514",
```

```

    "rolled_over": false,
    "dry_run": false,
    "conditions": [
        "[max_size: 12gb)": false
    ]
}

```

Restoring Archived Data using API

Restoring is the process of copying backed up data from an archive and restoring it in the database to replace lost or damaged data. You can restore the archived data in the API Gateway database as required.

You can use the REST API, apitransactions/archives to restore archived data.

View the list of available archive files using the following REST API:

```

curl -XGET
"http://daeyaix1bvt01:5555/rest/apigateway/apitransactions/archives"
-H 'accept:application/json' -H "Accept: application/json" -u username:password

```

Sample response

```
{"archiveFiles":["default-2021-10-13-1634096748481","default-2021-10-13-1634096810028",
"default-2021-10-13-1634096871380","default-2021-10-13-1634096933197",
"default-2021-10-13-1634097054614","default-2021-10-13-1634097248461"]}
```

Restore the required archive file using the following REST API:

```

curl -XPOST
"http://localhost:5555/rest/apigateway/apitransactions/archives/filename"
-H "accept:application/json" -u username:password

```

Sample command

```

curl -XPOST
"http://localhost:5555/rest/apigateway/apitransactions/archives/default-2017-02-09-1486644396751"
-H "accept:application/json" -u Administrator:manage

```

Archive and Purge using UI

This section explains the steps to archive or purge data using the application UI.

For the list of events that you can archive or purge, see “[List of events that can be archived or purged](#)” on page 114.

Note:

You can archive or purge the data in the API Data Store. If you use an external Elasticsearch as a Data store for API Gateway, the archive or purge operations are performed against that data store. These operations do not work on the data stores that are configured as destinations for transactions & logs.

Archiving Data

To archive data in the API Gateway database, you must have the **Manage purge and restore runtime event functional** privilege in API Gateway.

Archiving is the process of moving data that is no longer actively used for long-term retention so that it can be used at a later time. You can archive data based on the type of data or the age of the data in the API Gateway database.

➤ To archive data

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Manage Data**.
3. Click **Archive and purge**.
4. Select the event types for which the data has to be archived.
For the list of events that you can archive or purge, see “[List of events that can be archived or purged](#)” on page 114.
5. Select **Archive**.
6. 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.

If you select both **From date** and **To date**, all the data in the selected range is archived.

Home > Administration

Administration

Implement and manage the general and security related configurations for API Gateway. [?](#)

General Security Destinations Manage data System settings External accounts

Archive and purge

Restore Configuration

Archive and purge data
Archive and/or purge your data. [?](#)

Event type
 All

Action
 Archive Purge

Selection

Range
 Older than

From date To date

[Cancel](#) [Submit](#)

API Gateway archives the selected type of data for the given date range.

- Select **Older than**. Type a number and a time period. For example, if you type 1Y, data older than 1 year is selected for archiving. If you select 1M, data older than one month is selected for archiving.

The screenshot shows the 'Manage data' tab selected in the navigation bar. On the left, a sidebar lists 'Archive and purge', 'Restore', and 'Configuration'. The main area is titled 'Archive and purge data' with the sub-instruction 'Archive and/or purge your data.' Below this are sections for 'Event type' (set to 'All'), 'Action' (checkboxes for 'Archive' and 'Purge'), and 'Selection' (radio buttons for 'Range' and 'Older than', with 'Older than' selected and a value of '1y,1M,1d,1h,1m,1s' entered). At the bottom are 'Cancel' and 'Submit' buttons.

7. Click **Submit**.

The archive job is triggered and a job is created in the Job listing table.

By default, the archives are stored in the following location: `SAGInstallPath/profiles/IS_default/workspace/temp/default`.

You can modify this location from the **Configuration** tab on the **Manage Data** section. To learn more about the procedure, see “[Archive and Purge Configuration](#)” on page 116. Alternatively, you can also use the **backupsharedFilelocation** property in the **Extended settings** section to modify the location.

You can provide the maximum number of backups that can be archived using the **maxBackupslimit** property in the **Extended settings** section. The default value of this setting is 10. If you try to archive a backup after you reach the limit, you receive an error. To avoid errors, you can increase the default value from 10 to a higher value. If you do not provide a value in this field, then infinite number of archives are stored.

Purging Data

To purge data from the API Gateway database, you must have the Manage purge and restore runtime event functional privilege in API Gateway.

Purging is the process of systematically deleting unwanted data from the database. You can purge data based on the type of data or the age of the data in the API Gateway database.

➤ To purge data

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Manage Data**.
3. Click **Archive and purge**.
4. Select the event types for which the data has to be purged.
For the list of events that you can archive or purge, see “[List of events that can be archived or purged](#)” on page 114.
5. Select **Purge**.
6. Select one of the following options to purge (delete) the required data.
 - Select **Range**. Select a period during which you want the data to be purged.
 - To purge selected types of data from a particular date till the current date, select the required date in the **From date** field.
 - To purge selected types of data from the beginning (events start date) till a particular date, select the required date in the **To date** field.

If you select both **From date** and **To date**, all the data in the selected range is archived.
 - Select **Older than**. Type a number and a time period. For example, if you type 1Y, data older than 1 year is selected for purging. If you select 1M, data older than one month is selected for purging.
7. Click **Submit**.

The purge operation is triggered and a job is created in the job table. You can monitor the status of the purge operation, in the job table.

Restoring Archived Data using UI

To restore data in the API Gateway database, you must have the Manage purge and restore runtime event functional privilege in API Gateway.

➤ To restore archived data

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Manage Data**.

3. Click **Restore**.

You can view a list of archives in the job table.

4. Click the Restore icon for the archive job from which you want to restore data.

Job ID	Archive filename	Created
e1d13e55-5c60-409e-9a1b-4d3cddeb1648	default-2021-07-27-1627378452912	2021-07-27 09:34:12 GMT

You can view the status of the restore job on the Archive and Purge page.

Job ID	Action	Archive filename	Status	Created
87b20d60-a751-4028-b171-5fd96933324	purge	N/A	Completed	2021-07-27 09:38:34 GMT
e1d13e55-5c60-409e-9a1b-4d3cddeb1648	archive	default-2021-07-27-1627378452912	Completed	2021-07-27 09:34:12 GMT
df34d7be-bd5d-4f45-8629-9bce46ff952f	restore	default-2021-07-27-1627378452912	In Progress	2021-07-27 09:48:20 GMT

The selected data is restored in the API Gateway database, once the job is completed.

List of events that can be archived or purged

The following data can be archived or purged using **Archive & Purge** in API Gateway database:

Event name	Description
Audit logs	Archives or purges the auditable event data.
Error events	Archives or purges the runtime error event data.
Lifecycle events	Archives or purges the API Gateway's lifecycle data.
Monitor events	Archives or purges the runtime monitoring data.

Event name	Description
Performance metrics	Archives or purges the runtime performance metrics data.
Policy violation events	Archives or purges the policy compliance violation data.
Threat protection events	Archives or purges the threat protection data.
Transaction events	Archives or purges the API transactional data.
Application logs	Archives or purges the aggregated logs.
Mediator trace span	Archives or purges the stage-wise trace data.
Server log trace	Archives or purges the server log data.
Request response trace	Archives or purges the aggregated request and response trace data.

Job Listing

You can view the status and other details of each archive and purge job in the job listing. You can view this table only if you have created at least one archive job or purge job. The job listing table displays the following details.

Job ID	Action	Archive filename	Status	Created
87b20d60-a751-4028-b171-5f0d96933324	purge	N/A	In Progress (16 %)	2021-07-27 09:38:34 GMT
e1d13e55-5c60-409e-9a1b-4d3cddeb1648	archive	default-2021-07-27-1627378452912	Completed	2021-07-27 09:34:12 GMT

Column Name	Description
Job ID	Displays an auto-assigned unique ID for each archive and purge job.
Action	Specifies the action performed by the task. It displays one of the following actions based on the action performed:

Column Name	Description
	<ul style="list-style-type: none"> ■ Archive ■ Purge ■ ArchiveandPurge ■ Restore
Archive filename	Displays the name of the archive file, created. By default, the name is default-archive creation date-archive creation time . You cannot change the default name. For a purge job the file name would display as N/A.
Status	Displays the status of the archive or purge job. The status can be one of the following: <ul style="list-style-type: none"> ■ Failed. The archive or purge job failed. ■ Completed. The archive or purge job is successfully completed. ■ In Progress. The archive or purge job is running.
Created	Displays the date and time when the archive or purge job starts.
Refresh	Allows you to refresh the status. When you click refresh for an archive or purge job, the value of the Status column is updated.

Archive and Purge Configuration

By default, the archived files are stored in the `installation_location/profiles/IS_default/workspace/temp/default` location. However, you can store the archives in your preferred location. You can use the **backupsharedFilelocation** field to set a different location to save archived files. This setting performs the same operation as the **backupsharedFilelocation** extended setting.

Software AG recommends you to configure an alternate storage location for archived files. If you use the default location, it uses the API Gateway disk space.

➤ To configure the archive and purge settings

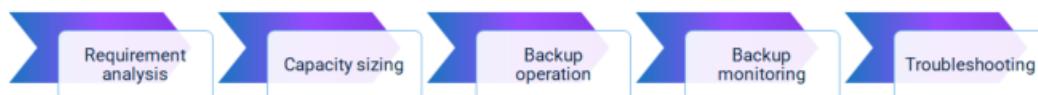
1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Manage Data**.
3. Click **Configuration**.
4. Type a new location to store the archives, in the **backupsharedFilelocation** field.

You must provide the complete path of the location in which you want to store the archives. API Gateway creates a folder called **default**, in the specified location and stores all the archive files in **default** folder, created in the specified location. For example, if you specify the location as C:\SoftwareAG\APIGateway, a folder called **default** is created in the APIGateway folder and archives are stored in this location.

5. **Purge throttle per second** controls the number of documents to be purged per second. The default value is 1000. If the date range you selected for purge has 10000 documents, then it takes 10 seconds to complete the purge operation. The default value of 1000 must work for most of the cases. Increasing the value will increase the purge performance (speed). On a flip side, this may cause memory pressure on API Data store (Elasticsearch) if the size of the documents being purged are high.

Data Backup

This section covers the following topics related to data backup:



Backup Requirements Analysis

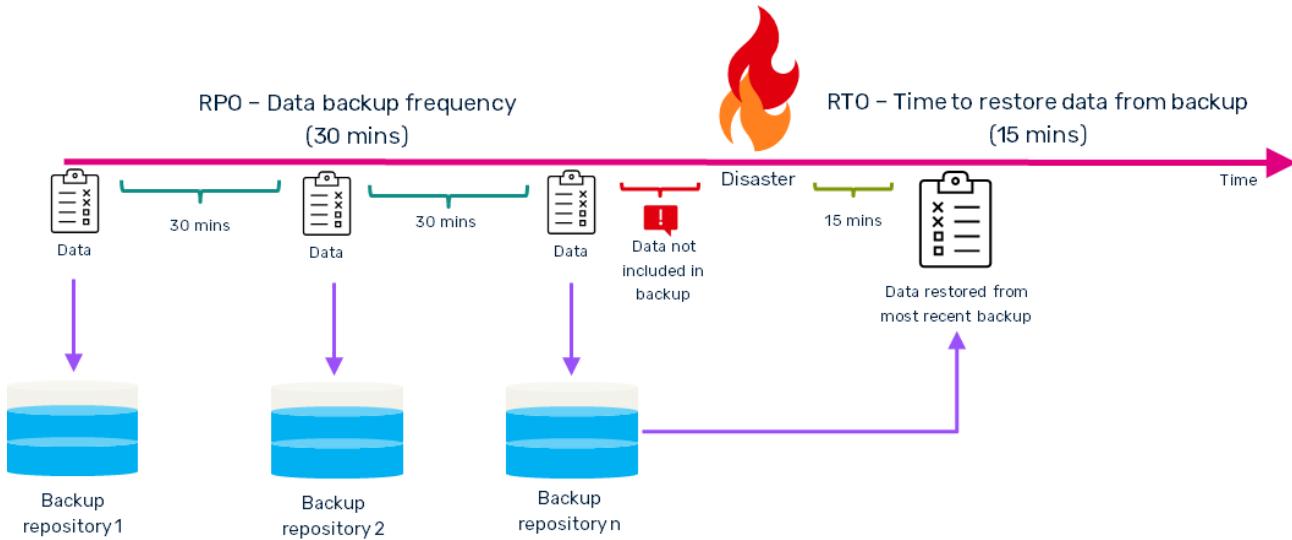
To analyze the backup requirements, you must first determine the following values:

- **Recovery Point Objective** (RPO): Determines the frequency in which the backups must be created. Basically, this value is the age of files that must be recovered from backup for a business operation to resume after a disaster.

For example, if the age of data, that you can afford to lose, just before a disaster is 30 minutes then the RPO is 30 minutes. It means that you have to create a backup for every 30 minutes.

- **Recovery Time Objective** (RTO): Determines the time that data must be restored from backup.

For example, if data has to be restored in 15 minutes from the time of disaster, then the RTO is 15 minutes.



The RPO value could be different for each data type. For example, the RPO for the API assets data could be one hour whereas the RPO for API analytics could be 24 hours.

Based on your backup requirements, you must ensure the pre-requisites are available.

Backup Repository Capacity Sizing

The size of backup repository depends on the size of data stored in API Data Store on a regular basis. The following questions help you calculate the backup repository size:

- **What is the backup frequency?**

The backup frequency impacts the memory size and it has to be taken into account for analyzing memory size.

The recommended backup frequency for:

- **Assets data.** More frequent. For example, 1 hour. The assets data is very critical for the business to run seamlessly; and it is lesser in size.
- **Analytics data.** Lesser frequent than the assets data. For example, 24 hours. The analytics data is larger in size.

- **Are the API payload details included in the backup?**

If you choose to backup the headers, parameters, requests, and responses of API payloads, it impacts the memory and disk size.

- **How long the backup snapshots to be stored?**

The retention period of backup snapshots impacts the disk size requirement. You could retain backup snapshots for a maximum of 30 to 90 days based on your requirement after which you can archive or purge.

Backup Considerations

Software AG recommends that you follow these points for a seamless backup and restore process:

1. Store the backup of API assets, API analytics, and platform data in different repositories, preferably outside the API Data Store folder.
2. Schedule your backup periodically.
3. Additional Elasticsearch plugins might be required if you want to store the backup files in one of the supported cloud storage spaces.

Backup Operation

For creating a complete API Gateway data backup, you must create a backup of

- **Assets data and Analytics data.** You can create a backup of the assets and analytics manually or by scheduling a cron job. The frequency in which the assets and analytics data is backed up is different. So, they are taken in separate snapshots. For information on taking a backup of API Data Store, see “[Data Store Backup](#)” on page 119.

Software AG recommends that you automate the backup process.

- **Platform Data.** You can take a backup of the platform data using the **apigatewayUtil** script or manually take a backup of the configuration files. For more information on creating backup of the platform data, see “[Platform Data Backup](#)” on page 156.

Typically, there is no need to back up the platform data periodically as these are mostly one-time configurations required at the time of setup or provisioning. You can perform a file backup when there are changes. For details on restoring platform data, see “[Restoring Platform Data backed up using the Script](#)” on page 170 or “[Restoring the manually backed up Platform data](#)” on page 170.

Data Store Backup

The API Data Store is backed up using the *apigatewayUtil* script. This script comes along with the installation of the application, and you can find the same from the `SAGInstallDir/IntegrationServer/instances/instance_name/packages/WmAPIGateway/cli/bin` folder.

Taking backup using the script involves the following steps:



API Gateway backups are snapshots of data taken incrementally. This backup process only copies data to the repository that was not already copied by an earlier snapshot, avoiding unnecessary duplication of work or storage space. Hence, you can safely take snapshots very frequently with minimal overhead. For example, if you create backup files daily, the backup file you create has only the data that has undergone change and the data created since your last backup, that is the previous day.

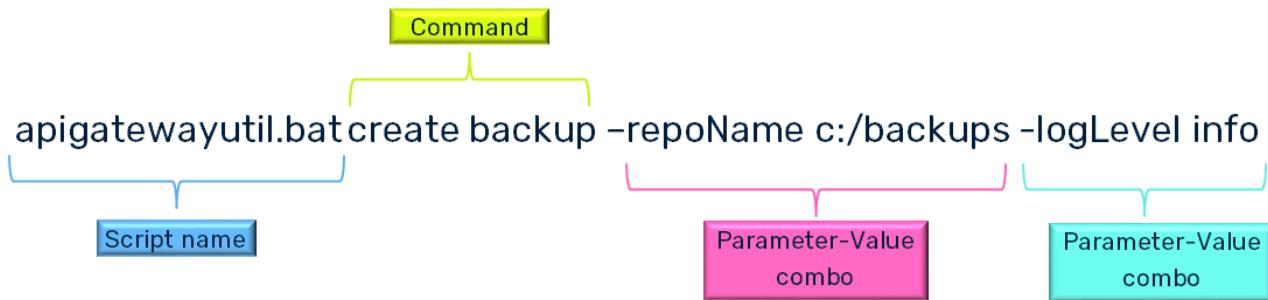
This process of incrementing the data with snapshots only applies within a single repository because no data is shared between repositories.

Format to run apigatewayutil Script

The *apigatewayUtil* script is always followed by:

- a command, based on the operation that has to be performed. For example, **create backup** to create a backup and **list backup** to view the list of backup files.
- list of parameters to support the operation along with their corresponding values. For example, **include assets** to include assets in the backup file.

For example,



If you need information on the parameters that you can use with the *apigatewayUtil* script and their syntaxes, run the following command:

Linux

```
./apigatewayUtil -help
```

Windows

```
apigatewayUtil.bat -help
```

Backup Repository

A repository is a folder, network location, or cloud location where you store the data backup files. The default repository that comes with the API Gateway installation is located at *SAGInstallDir/InternalDataStore/archives*. This repository is called **default**. However, you can create additional repositories for backup storage based on your requirements.

Configure the required repositories using the **configure** command. You can configure more than one repository in one Elasticsearch instance with unique names. You can specify the required repository name using the **repoName** parameter when performing backup and restore operations.

You can configure your backup repository in one of the following locations:

- Local file system
- Network file system
- Cloud

Configuring a Local File System Repository

The local file system functions as the default repository. If you do not create a repository, the backup files are stored in the `SAGInstallDir/InternalDataStore/archives/` folder, by default.

Note:

For a clustered API Gateway setup, ensure that the backup repository must be accessible for all cluster nodes.

1. From the command prompt, go to `SAGInstallDir/IntegrationServer/instances/instance_name/packages/WmAPIGateway/cli/bin`.
2. Run the following command to configure local file system repository:

Linux

```
./apigatewayUtil.sh configure fs_path -path backup_location
```

Windows

```
apigatewayUtil.bat configure fs_path -path backup_location
```

For example,

```
apigatewayUtil.bat configure fs_path -path D:/localapigatewayBackup
```

This command creates a backup folder called `localapigatewayBackup` in the D drive.

3. Restart API Data Store.

You can now use the configured local repository to store backup files. For information about backing up, [“Creating API Data Store Backup” on page 126](#).

4. *Optional.* Run the following command to list the available list of repositories and verify whether the repository you created appears in the list:

Linux

```
./apigatewayUtil.sh list manageRepo
```

Windows

```
apigatewayUtil.bat list manageRepo
```

Run the following command to list the available list of analytics data store repositories and verify whether the repository you created appears in the list:Linux

```
./apigatewayUtil.sh list manageRepo -analyticsDS true
```

Windows

```
apigatewayUtil.bat list manageRepo -analyticsDS true
```

You can specify the log file location and log level for the repository creation using the **logFileLocation** and **logLevel** parameters. For information on these parameters, see [“Specifying Log File Details” on page 148](#).

5. *Optional.* Run the following command to delete analytics data store repository:

Linux

```
./apigatewayUtil.sh delete manageRepo -analyticsDS true -repoName analyticsRepo
```

Windows

```
apigatewayUtil.bat delete manageRepo -analyticsDS true -repoName analyticsRepo
```

For example,

```
apigatewayUtil.bat delete manageRepo -analyticsDS true -repoName myrepo
```

This command deletes the local repository named `myrepo`.

You can specify the log file location and log level for deleting the repository using the **logFileLocation** and **logLevel** parameters. For information on these parameters, see “[Specifying Log File Details](#)” on page 148.

Configuring a Network File System Repository

You can specify a folder that is safe in the network as a repository to store your backup files.

If you are performing this for a clustered setup, ensure that all nodes in the cluster can access the repository.

➤ To configure a network file system repository:

1. From the command prompt, go to `SAGInstallDir/IntegrationServer/instances/instance_name/packages/WmAPIGateway/cli/bin`.
2. Run the following command to create a network repository:

Linux

```
./apigatewayUtil.sh configure fs_path -path network_location
```

Windows

```
apigatewayUtil.bat configure fs_path -path network_location
```

For example,

```
apigatewayUtil.bat configure fs_path -path //10.2.35.121/apigatewayBackup
```

This command creates a repository called `localapigatewayBackup` at the specified network location.

You can specify the log file location and log level for the repository creation using the **logFileLocation** and **logLevel** parameters. For information on these parameters, see “[Specifying Log File Details](#)” on page 148.

3. Restart API Data Store.

You can now use the configured network repository to store backup files. For information on backing up, “[Creating API Data Store Backup](#)” on page 126.

4. *Optional.* Run the following command to list the available list of repositories and verify whether the repository you created appears in the list:

Linux

```
./apigatewayUtil.sh list manageRepo
```

Windows

```
apigatewayUtil.bat list manageRepo
```

Run the following command to list the available list of analytics data store repositories and verify whether the repository you created appears in the list:

Linux

```
./apigatewayUtil.sh list manageRepo -analyticsDS true
```

Windows

```
apigatewayUtil.bat list manageRepo -analyticsDS true
```

You can specify the log file location and log level for the repository listing using the **logFileLocation** and **logLevel** parameters. For information on these parameters, see “[Specifying Log File Details](#)” on page 148.

5. *Optional.* Run the following command to delete analytics data store repository:

Linux

```
./apigatewayUtil.sh delete manageRepo -analyticsDS true -repoName analyticsRepo
```

Windows

```
apigatewayUtil.bat delete manageRepo -analyticsDS true -repoName analyticsRepo
```

For example,

```
apigatewayUtil.bat delete manageRepo -analyticsDS true -repoName myrepo
```

This command deletes the local repository named `myrepo`.

You can specify the log file location and log level for deleting the repository using the **logFileLocation** and **logLevel** parameters. For information on these parameters, see “[Specifying Log File Details](#)” on page 148.

Configuring a Cloud Repository in Amazon S3

You can configure the following types of cloud repositories:

- Amazon S3

- Azure storage. For information on setting up a Azure cloud repository, see <https://www.elastic.co/guide/en/cloud/current/ec-azure-snapshotting.html>.

This section explains the steps to configure an Amazon S3 repository. To configure a repository, ensure that you have installed the Amazon S3 plugin in Elasticsearch.

➤ To configure an Amazon S3 cloud repository

1. From the command prompt, go to `SAGInstallDir/InternalDataStore/bin`.
2. Run the following command to create Elasticsearch keystore:

Linux

```
./elasticsearch-keystore create
```

Windows

```
elasticsearch-keystore.bat create
```

3. Run the following command to add the Amazon S3 repository access key to your Elasticsearch keystore:

Linux

```
./elasticsearch-keystore add s3.client.default.access_key
```

Windows

```
elasticsearch-keystore.bat add s3.client.default.access_key
```

4. When prompted for the Amazon S3 repository access key, type the access key value and press **Enter**.

```
Enter value for s3.client.default.access_key: 123-test-123d-123
```

5. Run the following command to add the Amazon S3 repository secret key:

Linux

```
./elasticsearch-keystore add s3.client.default.secret_key
```

Windows

```
elasticsearch-keystore.bat add s3.client.default.secret_key
```

6. When prompted for the Amazon S3 repository secret key, type the secret key value and press **Enter**.

```
Enter value for s3.client.default.secret_key: tests1232sk12312t
```

Important:

You must restart Elasticsearch once you provide the secret key.

7. Go to `SAGInstallDir/IntegrationServer/instances/instance_name/packages/WmAPIGateway/cli/bin/conf`.

- Open the `gateway-s3-repo.cnf` file and specify Amazon S3 values for the following fields:

```
type=s3
bucket=<s3-bucket-name>
region=<s3-region>
access_key=<s3-access-key>
secret_key=<s3-secret-key>
base_path=<s3-base-path>
```

For example,

```
type=s3
bucket=s3 bucket
region=ap-south-1
access_key=apikey
secret_key=secretKey
base_path=basepath/repo
```

Note:

The **base_path** value must not start or end with an slash (\).

- Run the following command:

Linux

```
./apigatewayUtil.sh configure manageRepo -file file_path
```

Windows

```
apigatewayUtil.bat configure manageRepo -file file_path
```

For example,

```
./apigatewayUtil.sh configure manageRepo -file
SAGInstallDir/IntegrationServer/instances/
instance_name/packages/WmAPIGateway/cli/bin/conf/gateway-s3-repo.cnf
```

Run the following command to configure the analytics data store repository:

Linux

```
./apigatewayUtil.sh configure manageRepo -analyticsDS true -repoName analyticsRepo
```

Windows

```
apigatewayUtil.bat configure manageRepo -analyticsDS true -repoName analyticsRepo
```

For example,

```
apigatewayUtil.bat configure manageRepo -analyticsDS true -repoName myrepo
```

This command creates a repository named `myrepo` to backup the analytics data store data.

You can specify the log file location and log level for the repository creation using the **logFileLocation** and **logLevel** parameters. For information on these parameters, see ["Specifying Log File Details" on page 148](#).

- Restart API Data Store.

You can now use the configured cloud repository to store backup files. For information on backing up, [“Creating API Data Store Backup” on page 126](#).

11. *Optional.* Run the following command to list the available list of repositories and verify whether the repository you created appears in the list:

Linux

```
./apigatewayUtil.sh list manageRepo
```

Windows

```
apigatewayUtil.bat list manageRepo
```

Run the following command to list the available list of analytics data store repositories and verify whether the repository you created appears in the list: Linux

```
./apigatewayUtil.sh list manageRepo -analyticsDS true
```

Windows

```
apigatewayUtil.bat list manageRepo -analyticsDS true
```

You can specify the log file location and log level for the repository listing using the **logFileLocation** and **logLevel** parameters. For information on these parameters, see [“Specifying Log File Details” on page 148](#).

12. *Optional.* Run the following command to delete analytics data store repository:

Linux

```
./apigatewayUtil.sh delete manageRepo -analyticsDS true -repoName analyticsRepo
```

Windows

```
apigatewayUtil.bat delete manageRepo -analyticsDS true -repoName analyticsRepo
```

For example,

```
apigatewayUtil.bat delete manageRepo -analyticsDS true -repoName myrepo
```

This command deletes the local repository named `myrepo`.

You can specify the log file location and log level for deleting the repository using the **logFileLocation** and **logLevel** parameters. For information on these parameters, see [“Specifying Log File Details” on page 148](#).

Creating API Data Store Backup

After you have configured the backup repositories, you can create backups.

Pre-requisites:

- Ensure that you have the API Gateway running during backup.

- Ensure that you configure a repository to store your backup files. For information about creating and configuring a repository, see “[Backup Repository](#)” on page 120.

Note:

Before you start the analytics data store backup you have to:

- Configure the apigw.analytics.ds.hosts property in the analyticsds.properties file located at *SAGInstallDirectory\IntegrationServer\instances\instance_name\packages\WmAPIGateway\config\resources\elasticsearch*. For example, apigw.analytics.ds.hosts=localhost:9241.
- Configure the Hostname and Port property in the gateway-analytics-ds.xml file located at *SAGInstallDirectory\IntegrationServer\instances\instance_name\packages\WmAPIGateway\config\resources\beans*. Sample code to set the hostname and port is as follows:

```
<constructor-arg index="0">
    <list>
        <value>hostname:port</value>
    </list>
</constructor-arg>
```

➤ To take backup

1. From the command prompt, go to

SAGInstallDir/IntegrationServer/instances/default/packages/WmAPIGateway/cli/bin.

Note:

Replace default with the corresponding instance name.

2. Run the following command to create a backup in the default location:

Linux

```
./apigatewayUtil.sh create backup
```

Windows

```
apigatewayUtil.bat create backup
```

The following sample shows the creation of a backup on Windows. The backup is created in the default location, *SAGInstallDir/InternalDataStore/archives/*.

```
C:\SoftwareAG\IntegrationServer\instances\default\packages\WmAPIGateway\cli\bin>apigatewayUtil.bat
create backup
Initiated backup process for default-2021-may-17-22-23-2-266000000 file
```

You can include the following parameters to customize the backup as per your requirements.

Parameter	Description
name	Name of the backup file. You can provide alphanumeric values and the following special characters: !, @, \$, %, (,). The name can be based on date and time for unique identification.

Parameter	Description
	<p>If you do not provide any name for the backup file, the script generates a name with the current time and the value provided for the tenant parameter. For example, <i>default-2021-may-17-22-23-2-266000000</i>.</p> <p>Note: Do not use uppercase in the name.</p>
tenant	<p>Name of the tenant in which the data exists.</p> <p>If you do not provide this parameter, the value is picked from <code>pg.gateway.elasticsearch.tenantId</code> property in config.properties file located at <code>IntegrationServer\instances\instance_name\packages\WmAPIGateway\config\resources\elasticsearch</code>.</p>
repoName	<p>Name of the repository where the backup must be saved.</p> <p>If you do not provide this parameter, the value is picked from the <i>tenant</i> parameter.</p>
include	<p>Type of data you want to include in the backup file. All indexes, irrespective of their types, are included in the backup file if you do not provide any parameters.</p> <p>Available options for this parameter are:</p> <ul style="list-style-type: none"> ■ Assets. To include all API Gateway assets in the backup being created. ■ Audit. To include the audit data in the backup. ■ Analytics. To include analytics data in the backup. ■ License. To include the license data in the backup. This data provides the transaction usage analytics across all stages of the application. ■ Log. To include log data in the backup. The log index stores the application logs when log aggregation is enabled in API Gateway. It stores logs at Integration Server level, API Data Store level, and platform level. <p>If you include one or more of the above entries in your backup, the corresponding item is added as a suffix to your backup file name. That is, if you provide <i>include assets audit</i> parameter in your backup command, and if your backup name is <i>backup01</i>, then the backup file name is created as <i>backup01 -assets audit</i>.</p> <p>If you have different retention period for core data and analytics data, then you can use this parameter to create backup of a particular type of data. For example, if you have an RPO of 30 mins for core data and 4 hours for all data (including analytics), then you run the following command for every 30 mins and the full backup command every 4 hours:</p>

Parameter	Description
	<pre>apigatewayUtil.bat create backup -tenant mytenant -name mytenant_2021June031230 -include assets -repoName myrepo</pre>
	<p>The names of indexes that you can provide with this parameter for the above data types are listed in the section, “List of Indexes that can be included in backup” on page 149.</p>
analyticsDS	<p>Option to create a backup of the analytics data store. Specify as true to create a backup of the analytics data store.</p>
	<p>The sample command to create a backup of the analytics data store is as follows:</p>
	<pre>apigatewayUtil.bat create backup -analyticsDS true -reponame myrepo</pre>
	<p>You can use the <code>include</code> parameter with this command to include the different types of data that you want to have in the backup file.</p>
	<p>If you do not provide any options for the <code>include</code> parameter:</p>
	<ul style="list-style-type: none"> ■ with <code>-analyticsDS true</code>. Audit data, analytics data, and log data is included in the backup. ■ without <code>-analyticsDS true</code>. Only assets data is included in the backup of API Data Store. If you want include other data besides assets, for example analytics data, use the <code>-include</code> parameter explicitly.
debug	<p>Option to specify whether you want to activate debugging when creating a backup. Available levels are:</p>
	<ul style="list-style-type: none"> ■ <code>true</code>. Displays detailed information on the backup process when the script is run. In most cases, the output printed with the debug parameter is helpful to troubleshoot the issue. ■ <code>false</code>. Does not display detailed information. The value is considered as <code>false</code>, when this parameter is not provided.
logLevel	<p>Level of log that you want to create. Log levels indicate the severity of logs. Available levels are:</p>
	<ul style="list-style-type: none"> ■ <code>Info</code>. Provides the list of regular events that occur during the process. These events are informative. ■ <code>Debug</code>. Provides the events that could be useful, if you have to debug the process. ■ <code>Warning</code>. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process.

Parameter	Description
-----------	-------------

- **Error**. Indicates the events that stop the functionality from working as designed.
- **Trace**. Provides the list of events in a much detailed manner that could be useful for debugging.

You can specify one of the log level with the **LogLevel** parameter. For example, to create a log file of **Warning** level when listing backup files, you can run the following command:

```
apigatewayUtil.bat list backup -logLevel warning
```

When you provide **Error** as the log level, then only the error level logs are saved. When you provide **Debug** as the log level, then **Debug, Info, Warning** and **Error** level logs are saved. When you provide **Trace** as log level, then all level logs are saved.

This parameter is optional. If you do not specify the parameter, then the **Info** level logs are saved by default.

logFileLocation Location where you want to save the log file.

For example, to save the log file in `C:/apiglogs/backups`, you can provide the location as seen in the following example:

```
apigatewayUtil.bat create backup -name samplebackup  
-logFileLocation C:/apiglogs/backups
```

This parameter is optional. If you do not specify the parameter, the logs are saved in the following location `SAGInstallDir/IntegrationServer/instances/instance_name/packages/WmAPIGateway/cli/logs/APIGWUtility.log`.

The parameters listed above are optional. To create a backup file in the repository that you configured, use the **repoName** parameter to provide the repository name. That is,

```
apigatewayUtil.bat create backup -repoName repository_name
```

For example,

```
apigatewayUtil.bat create backup -repoName s3_repo
```

After running the script, you can check the backup status using the `status backup` command and view the list of backup files in your repository. For information on verifying backup status, see “[Verifying Backup Status](#)” on page 142 and for information on viewing the list of backup files, see “[Viewing Backup Files List](#)” on page 138.

Important:

From the Data housekeeping perspective, Software AG recommends that you schedule the execution of these commands through cron jobs or other scheduling options. The frequency of the backup depends on your RPO needs.

Advanced Backup Options

The following sections explain the advanced backup options.

Creating Backup of specific Indexes

API Data Store contains multiple indexes. Different indexes are used to store different types of data. So, to take a backup of a particular data, you can take a backup of a specific index or set of indexes. For the list of indexes available in API Data Store, see “[List of Indexes that can be included in backup](#)” on page 149.

➤ To take a backup of the required indexes

1. From the command prompt, go to

`SAGInstallDir/IntegrationServer/instances/default/packages/WmAPIGateway/cli/bin.`

Note:

Replace `default` with the corresponding instance name.

2. Run the following command to create a backup of specific indexes in the default location:

Linux

```
./apigatewayUtil.sh create backup -indices list of indexes separated by comma
```

Windows

```
apigatewayUtil.bat create backup -indices list_of_indexes separated by comma
```

The following sample shows the creation of a backup of the file indexes, `gateway_default_aliases-000001` and `gateway_default_truststores-000001`, on Windows. The backup is created in the default location, `SAGInstallDir/InternalDataStore/archives/`.

```
C:\SoftwareAG\IntegrationServer\instances\default\packages\WmAPIGateway\cli\bin>apigatewayUtil.bat
  create backup
  -indices gateway_default_aliases-000001, gateway_default_truststores-000001
  Initiated backup process for default-2021-may-17-22-23-2-266000000 file
```

For the list of indexes that you can provide with the **indices** parameter, see “[List of Indexes that can be backed up individually](#)” on page 153.

Similar to the list of parameters used during backup creation, you can provide parameters except the **include** parameter. So, the list of parameters to customize the backup of specific indexes as per your requirements are:

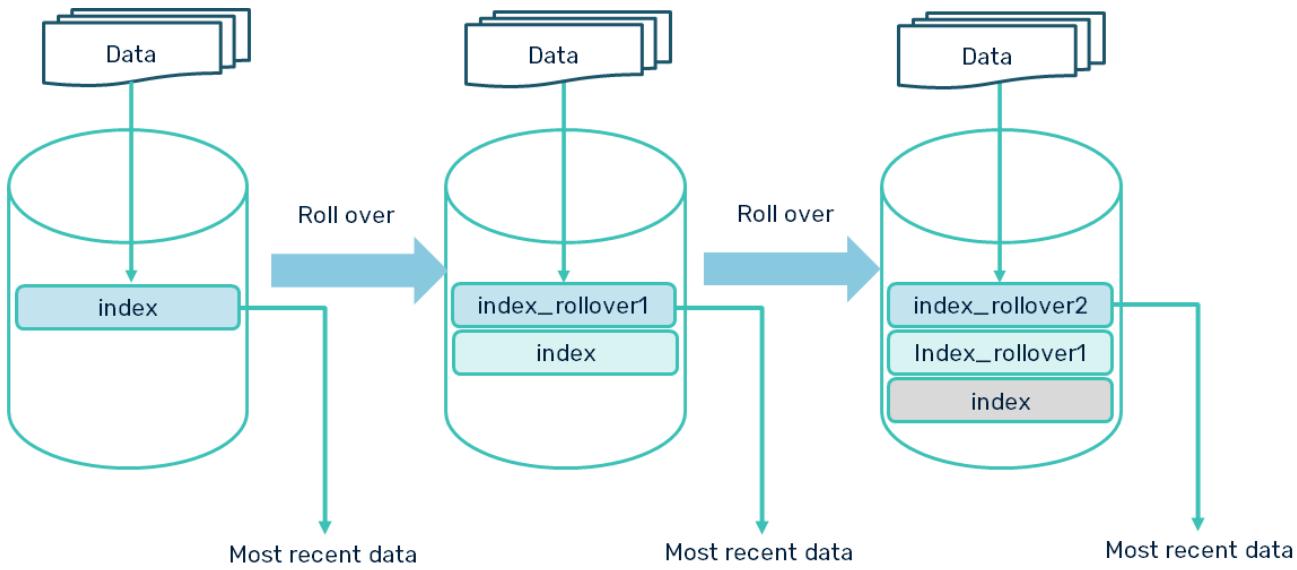
Parameter	Description
name	Name of the backup file. You can provide alphanumeric values and the following special characters: !, @, \$, %, , (,).

Parameter	Description
	<p>If you do not provide any name for the backup file, the script generates a name with the current time and the value provided for the tenant parameter. For example, <i>default-2021-may-17-22-23-2-266000000</i>.</p> <p>Note: Do not use uppercase in the name.</p>
tenant	<p>Name of the tenant in which the data exists.</p> <p>If you do not provide this parameter, the value is picked from <code>pg.gateway.elasticsearch.tenantId</code> property in config.properties file located at <code>IntegrationServer\instances\instance_name\packages\WmAPIGateway\config\resources\elasticsearch</code>.</p>
repoName	<p>Name of the repository where the backup must be saved.</p> <p>If you do not provide this parameter, the value is picked from the <i>tenant</i> parameter.</p>
debug	<p>Option to specify whether you want to activate debugging when creating a backup. Available levels are:</p> <ul style="list-style-type: none"> ■ <code>true</code>. Displays detailed information on the backup process when the script is run. In most cases, the output printed with the debug parameter is helpful to troubleshoot the issue. ■ <code>false</code>. Does not display detailed information. The value is considered as false, when this parameter is not provided.
analyticsDS	<p>Option to create a backup of the analytics data store of a specific index or set of indexes. Specify as true to create a backup of the analytics data store of a specific index or set of indexes.</p> <p>The sample command to create a backup of the analytics data store of specific index or set of indexes is as follows:</p> <pre data-bbox="612 1594 1362 1657"><code>apigatewayUtil.bat create backup -analyticsDS true -indices list_of_indexes separated by comma</code></pre>
logLevel	<p>Level of log that you want to create. Log levels indicate the severity of logs. Available levels are:</p> <ul style="list-style-type: none"> ■ <code>Info</code>. Provides the list of regular events that occur during the process. These events are informative.

Parameter	Description
	<ul style="list-style-type: none"> ■ Debug. Provides the events that could be useful, if you have to debug the process. ■ Warning. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process. ■ Error. Indicates the events that stop the functionality from working as designed. ■ Trace. Provides the list of events in a much detailed manner that could be useful for debugging. <p>You can specify one of the log level with the LogLevel parameter. For example, to create a log file of Warning level when listing backup files, you can run the following command:</p> <pre>apigatewayUtil.bat list backup -logLevel warning</pre> <p>When you provide Error as the log level, then only the error level logs are saved. When you provide Debug as the log level, then Debug, Info, Warning and Error level logs are saved. When you provide Trace as log level, then all level logs are saved.</p> <p>This parameter is optional. If you do not specify the parameter, then the Info level logs are saved by default.</p>
logFileLocation	<p>Location where you want to save the log file.</p> <p>For example, to save the log file in <code>C:/apiglogs/backups</code>, you can provide the location as seen in the following example:</p> <pre>apigatewayUtil.bat create backup -name samplebackup -logFileLocation C:/apiglogs/backups</pre> <p>This parameter is optional. If you do not specify the parameter, the logs are saved in the following location <code>SAGInstallDir/IntegrationServer/instances/instance_name/packages/WmAPIGateway/cli/logs/APIGWUtility.log</code>.</p>

Creating Rollover of an Index

When data on a particular index exceeds a certain limit, it is essential to rollover the index and create a new index. In API Gateway, it is essential to monitor the indexes for transactional events. For transactional events, you must rollover the index, when the index size exceeds 25 GB. When an index is rolled over, a new index is created with two primary and a replica for each shard.



You can perform rollover for the following indexes:

- performancemetrics
- policyviolationevents
- monitorevents
- errorevents
- threatprotectionevents
- transactionalevents
- lifecycleevents
- auditlogs
- log
- serverlogtrace
- mediatortrace
- requestresponsetrace

➤ To create rollover of an index

1. From the command prompt, go to
`SAGInstallDir/IntegrationServer/instances/default/packages/WmAPIGateway/cli/bin.`

Note:

Replace `default` with the corresponding instance name.

2. Run the following command to create a backup of specific indexes in the default location:

Linux

```
./apigatewayUtil.sh rollover index -type name of the index to roll over
```

Windows

```
apigatewayUtil.bat rollover index -type name of the index to roll over
```

The following sample shows the roll over of index, *performancemetrics*, on Windows.

```
C:\SoftwareAG\IntegrationServer\instances\default\packages\WmAPIGateway\cli\bin>apigatewayUtil.bat
    rollover index
    -type performancemetrics
Rollover is completed successfully
Old Index: gateway_default_analytics_performancemetrics_202108140608
New Index: gateway_default_analytics_performancemetrics_202108140612
Rolled over: true
Dry Run: false
Conditions: [{}]
```

You can include the following optional parameters to customize the roll over command as per your requirements.

Parameter	Description
tenant	Name of the tenant in which the data exists. If you do not provide this parameter, the value is picked from <code>pg.gateway.elasticsearch.tenantId</code> property in config.properties file located at <code>IntegrationServer\instances\instance_name\packages\WmAPIGateway\config\resources\elasticsearch</code> .
targetIndexSuffix	Suffix to be included in the name of the rolled over index. If you do not provide this parameter, the rollover timestamp is added as a suffix to the rolled over index. You can provide a suffix such as the date distinguish the index. For example, if you provide the following command:
	<pre>apigatewayutil.bat rollover index -type performancemetrics -targetIndexSuffix 15aug2021</pre> <p>the suffix is added to the rolled over index</p> <pre>Rollover is completed successfully Old Index: gateway_default_analytics_performancemetrics_14aug2021 New Index: gateway_default_analytics_performancemetrics_15aug2021 Rolled over: true Dry Run: false Conditions: [{}]</pre>
dryRun	Option to check whether the current index matches with any of the specified conditions.

Parameter	Description
	<p>Possible values are:</p> <ul style="list-style-type: none"> ■ <i>true</i>. Performs check on the current index against the specified conditions. ■ <i>false</i>. Does not check the current index. This is the default value and the value of the parameter is considered as <i>false</i>, if do not specify this parameter. <p>You can specify one of the conditions listed in the section.</p> <p>For example, to check if the size of the current log index is <i>6 GB</i>, you can run the command as seen here:</p> <pre>apigatewayutil.bat rollover index -type log -dryRun true -maxSize 6GB</pre> <p>Sample result:</p> <pre>Dry run executed successfully. Old Index: gateway_default_log_1628582905777-000001 New Index: gateway_default_log_202108271018 Rolled over: false Dry Run: true Conditions: [[max_size: 6gb]=false]]</pre> <p>Note: This parameter does not perform rollover. It displays results based on the check performed.</p>
analyticsDS	<p>Option to rollover indices in the analytics data store. Specify as <i>true</i> to rollover indices in the analytics data store.</p> <p>The sample command to rollover indices in analytics data store is as follows:</p> <pre>apigatewayUtil.bat rollover index -analyticsDS true -type log</pre>
logLevel	<p>Level of log that you want to create. Log levels indicate the severity of logs. Available levels are:</p> <ul style="list-style-type: none"> ■ Info. Provides the list of regular events that occur during the process. These events are informative. ■ Debug. Provides the events that could be useful, if you have to debug the process. ■ Warning. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process.

Parameter	Description
	<ul style="list-style-type: none"> ■ Error. Indicates the events that stop the functionality from working as designed. ■ Trace. Provides the list of events in a much detailed manner that could be useful for debugging. <p>You can specify one of the log level with the LogLevel parameter. For example, to create a log file of Warning level when listing backup files, you can run the following command:</p> <pre>apigatewayUtil.bat list backup -logLevel warning</pre> <p>When you provide Error as the log level, then only the error level logs are saved. When you provide Debug as the log level, then Debug, Info, Warning and Error level logs are saved. When you provide Trace as log level, then all level logs are saved.</p> <p>This parameter is optional. If you do not specify the parameter, then the Info level logs are saved by default.</p>
logFileLocation	<p>Location where you want to save the log file.</p> <p>For example, to save the log file in <code>C:/apiglogs/backups</code>, you can provide the location as seen in the following example:</p> <pre>apigatewayUtil.bat create backup -name samplebackup -logFileLocation C:/apiglogs/backups</pre> <p>This parameter is optional. If you do not specify the parameter, the logs are saved in the following location <code>SAGInstallDir/IntegrationServer/instances/instance_name/packages/WmAPIGateway/cli/logs/APIGWUtility.log</code>.</p>

The specified indexes are rolled over when any one of these conditions is satisfied.

Conditions that can be given with rollover command

In addition to the above parameters you can provide the following conditions based on which the specified indexes must be rolled over:

Condition	Description
maxAge	<p>Maximum age of the indexes.</p> <p>The specified indexes are rolled over if they are older than the value provided for this condition. For the list of possible values for this field, see https://www.elastic.co/guide/en/elasticsearch/reference/master/api-conventions.html#time-units.</p>

Condition	Description
	<p>For example, to roll over the monitorevents index if it is older than 2 minutes, you can provide</p> <pre data-bbox="600 346 1078 403">apigatewayutil.bat rollover index -type monitorevents -maxAge 2m</pre>
maxDocs	<p>Maximum number of the documents in the indexes.</p> <p>The specified indexes are rolled over if the number of documents in the indexes are more than or equal to the value provided for this condition.</p> <p>For example, to roll over the monitorevents index if the number of documents in the index is more than or equal to 100, you can provide</p> <pre data-bbox="600 747 1078 804">apigatewayutil.bat rollover index -type monitorevents -maxDocs 100</pre>
maxSize	<p>Maximum size of the indexes.</p> <p>The specified indexes are rolled over if their size is equal to or more than the value provided in this condition. For the list of possible values for this field, see https://www.elastic.co/guide/en/elasticsearch/reference/master/api-conventions.html#byte-units.</p> <p>For example, to roll over the monitorevents index if its size is more than 1 GB, you can provide</p> <pre data-bbox="600 1191 1078 1248">apigatewayutil.bat rollover index -type monitorevents -maxAge 1gb</pre>
maxPrimaryShardSize	<p>Maximum size of the primary shard of the indexes.</p> <p>The specified indexes are rolled over if the size of their primary shards is equal to or more than the value provided for this condition.</p> <p>For example, to roll over the monitorevents index if the size of the index's primary shard is more than or equal to 1 GB, you can provide</p> <pre data-bbox="600 1613 1233 1670">apigatewayutil.bat rollover index -type monitorevents -maxPrimaryShardSize 1GB</pre>

Viewing Backup Files List

You can view the list of backup files in a location using the `list backup` command.

➤ To view the list of backup files

- From the command prompt, go to

`SAGInstallDir\IntegrationServer\instances\default\packages\WmAPIGateway\cli\bin.`

Note:

Replace `default` with the corresponding instance name.

- Run the following command:

Linux

```
./apigatewayUtil.sh list backup
```

Windows

```
apigatewayUtil.bat list backup
```

The available list of backup files appears as follows:

```
C:\SoftwareAG\IntegrationServer\instances\default\packages\WmAPIGateway\cli\bin>apigatewayUtil.bat
list backup
Backups available in default are
default-2021-april-12-11-38-4-420000000
sample
backup12april-analytics
default-2021-april-12-16-49-30-247000000
default-2021-may-17-22-23-2-266000000
default-2021-may-17-22-24-53-611000000
```

You can provide the following parameters based on your requirement:

Parameter	Description
tenant	Name of the tenant for which you want to view the list of backup files. If you do not provide this parameter, the value is picked from <code>pg.gateway.elasticsearch.tenantId</code> property in config.properties file located at <code>IntegrationServer\instances\instance_name\packages\WmAPIGateway\config\resources\elasticsearch</code> .
repoName	Name of the repository for which you want to view the list of backup files. If you do not provide this value when running the <code>list</code> command, then the value is picked from the <code>tenant</code> parameter and the list of backup files from that repository appears.
status	Option to specify the status of backup files and filter backup files based on their status. For example, if you want to view

Parameter	Description
	<p>the list of backup files whose status is Partial, then you can run the command as seen here:</p> <pre data-bbox="600 346 1385 388">apigatewayUtil.bat list backup -status partial</pre> <p>For information on possible states of backup files, see "Verifying Backup Status" on page 142.</p>
verbose	<p>Option to display detailed status of the backup files in a given repository. Possible values are:</p> <ul style="list-style-type: none"> ■ true. The backup files appear with the following details: <ul style="list-style-type: none"> ■ snapshot. Name of the backup file. ■ status. Status of the backup process. ■ startTime. Time when the backup process was initiated. ■ endTime. Time when the backup process was completed. ■ Duration. Time taken for the backup creation. ■ Indices. Name of the indexes included in the backup. ■ Successful shards. Number of successful shards in backup. ■ Failed shards. Number of failed shards in backup. ■ Total shards. Total number of shards in backup. ■ false. The backup files appear without the details listed above. <p>When you do not provide this parameter, the value for the parameter is considered as false, the backup files appear without the list of details seen above.</p>
format	<p>Option to specify the format in which the details must appear. Works in combination with the verbose parameter. Available options are:</p> <ul style="list-style-type: none"> ■ JSON ■ Text <p>For example, if you run the following command, the backup status details are displayed in plain text format:</p>

Parameter	Description
	<pre>apigatewayUtil.bat status backup -name samplebackup -verbose true -format text</pre>
analyticsDS	<p>Option to view the list of backup files of the analytics data store. Specify as true to view the list of backup files of the analytics data store.</p> <p>The sample command to view the list of backup files of the analytics data store is as follows:</p> <pre>apigatewayUtil.bat list backup -analyticsDS true</pre>
logLevel	<p>Level of log that you want to create. Log levels indicate the severity of logs. Available levels are:</p> <ul style="list-style-type: none"> ■ Info. Provides the list of regular events that occur during the process. These events are informative. ■ Debug. Provides the events that could be useful, if you have to debug the process. ■ Warning. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process. ■ Error. Indicates the events that stop the functionality from working as designed. ■ Trace. Provides the list of events in a much detailed manner that could be useful for debugging. <p>You can specify one of the log level with the logLevel parameter. For example, to create a log file of Warning level when listing backup files, you can run the following command:</p> <pre>apigatewayUtil.bat list backup -logLevel warning</pre> <p>When you provide Error as the log level, then only the error level logs are saved. When you provide Debug as the log level, then Debug, Info, Warning and Error level logs are saved. When you provide Trace as log level, then all level logs are saved.</p> <p>This parameter is optional. If you do not specify the parameter, then the Info level logs are saved by default.</p>
logFileLocation	<p>Location where you want to save the log file.</p> <p>For example, to save the log file in C:/apiglogs/backups, you can provide the location as seen in the following example:</p>

Parameter	Description
	<pre>apigatewayUtil.bat create backup -name samplebackup -logFileLocation C:/apiglogs/backups</pre> <p>This parameter is optional. If you do not specify the parameter, the logs are saved in the following location <code>SAGInstallDir/IntegrationServer/instances/instance_name/packages/WmAPIGateway/cli/logs/APIGWUtility.log</code>.</p>

Verifying Backup Status

When you run the `create backup` command, the backup process is initiated. You can then verify the status of a backup file using the `status backup` command. This command checks the status of a backup file and displays one of the following results:

Status	Description
Success	The backup file is successfully created with the given parameters.
Failed	The backup process has failed. You can diagnose the cause of failure using the <code>-debug true</code> parameter along with the <code>create backup</code> command or by checking the logs. For more information, see " Troubleshooting a Failed Backup " on page 159.
In progress	The backup process is still in progress. This message usually appears when the backup data is large and you verify the status as soon as you initiate the backup process.
Partial	The backup of the data is partially complete. You can troubleshoot such instances by following the steps given in " Troubleshooting a Failed Backup " on page 159.
Unavailable	The specified backup file does not exist. Verify the file name and repository.

➤ To verify the status of a backup

- From the command prompt, go to

`SAGInstallDir/IntegrationServer/instances/default/packages/WmAPIGateway/cli/bin`.

Note:

Replace `default` with the corresponding instance name.

- Run the following command:

Linux

```
./apigatewayUtil.sh status backup -name backup-file-name
```

Windows

```
apigatewayUtil.bat status backup -name backup-file-name
```

The backup status appears as shown below:

```
C:\SoftwareAG\IntegrationServer\instances\default\packages\WmAPIGateway\cli\bin>apigatewayUtil.bat
status backup -name
default-2021-april-12-11-38-4-420000000
The status of the default-2021-april-12-11-38-4-420000000 is SUCCESS
```

You can specify tenant and repository when verifying the backup status via the following parameters:

Parameter	Description
tenant	Name of the tenant that you want to verify backup status. If you do not provide this parameter, the value is picked from <code>pg.gateway.elasticsearch.tenantId</code> property in config.properties file located at <code>IntegrationServer\instances\instance_name\packages\WmAPIGateway\config\resources\elasticsearch</code> .
repoName	Name of the repository where the backup can be found. If you do not provide this parameter, the value is picked from the <i>tenant</i> parameter.
verbose	Option to display detailed status of the backup files in a given repository. Possible values are: <ul style="list-style-type: none"> ■ true. The backup files appear with the following details: <ul style="list-style-type: none"> ■ snapshot. Name of the backup file. ■ status. Status of the backup process. ■ startTime. Time when the backup process was initiated. ■ endTime. Time when the backup process was completed. ■ Duration. Time when for the backup creation. ■ Indices. Name of the indexes included in the backup. ■ Successful shards. Number of successful shards in backup. ■ Failed shards. Number of failed shards in backup. ■ Total shards. Number of successful shards in backup.

Parameter	Description
	<ul style="list-style-type: none"> ■ <code>false</code>. The backup files appear without the details listed above. <p>When you do not provide this parameter, the value for the parameter is considered as <code>false</code>, the backup files appear without the list of details seen above.</p>
format	<p>Option to specify the format in which the details must appear. Works in combination with the verbose parameter. Available options are:</p> <ul style="list-style-type: none"> ■ JSON ■ Text <p>For example, if you run the following command, the backup status details are displayed in plain text format:</p> <pre data-bbox="587 823 1382 897">apigatewayUtil.bat status backup -name samplebackup -verbose true -format text</pre>
analyticsDS	<p>Option to get the backup status of the analytics data store. Specify as true to get the backup status of the analytics data store.</p> <p>The sample command to get the backup status of the analytics data store is as follows:</p> <pre data-bbox="587 1140 1382 1214">apigatewayUtil.bat status backup -analyticsDS true -name samplebackup</pre>
logLevel	<p>Level of log that you want to create. Log levels indicate the severity of logs. Available levels are:</p> <ul style="list-style-type: none"> ■ Info. Provides the list of regular events that occur during the process. These events are informative. ■ Debug. Provides the events that could be useful, if you have to debug the process. ■ Warning. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process. ■ Error. Indicates the events that stop the functionality from working as designed. ■ Trace. Provides the list of events in a much detailed manner that could be useful for debugging.

Parameter	Description
	<p>You can specify one of the log level with the logLevel parameter. For example, to create a log file of Warning level when listing backup files, you can run the following command:</p> <pre>apigatewayUtil.bat list backup -logLevel warning</pre> <p>When you provide Error as the log level, then only the error level logs are saved. When you provide Debug as the log level, then Debug, Info, Warning and Error level logs are saved. When you provide Trace as log level, then all level logs are saved.</p> <p>This parameter is optional. If you do not specify the parameter, then the Info level logs are saved by default.</p>
logFileLocation	<p>Location where you want to save the log file.</p> <p>For example, to save the log file in <code>C:/apiglogs/backups</code>, you can provide the location as seen in the following example:</p> <pre>apigatewayUtil.bat create backup -name samplebackup -logFileLocation C:/apiglogs/backups</pre> <p>This parameter is optional. If you do not specify the parameter, the logs are saved in the following location <code>SAGInstallDir/IntegrationServer/instances/instance_name/packages/WmAPIGateway/cli/logs/APIGWUtility.log</code></p>

If you run the **status backup** command without the **tenant** and **repoName** parameters, then the given backup file is checked in the default tenant and repository.

Deleting a Backup File

You can delete backups that are older than your data retention period and no more required.

As stated earlier, API Gateway backups are snapshots of data taken incrementally. These snapshots are logically independent from each other, even within a single repository. Hence, deleting a backup file does not affect the integrity of any other backup file.

CAUTION:

Use the **delete backup** command instead of deleting manually.

➤ To delete a backup file

- From the command prompt, go to `SAGInstallDir/IntegrationServer/instances/default/packages/WmAPIGateway/cli/bin`.

Note:

Replace `default` with the corresponding instance name.

- Run the following command to delete a backup file:

Linux

```
./apigatewayUtil.sh delete backup -name backup-file-name
```

Windows

```
apigatewayUtil.bat delete backup -name backup-file-name
```

Tip:

To determine the backup files for deleting, you can view the list of backup files with the **verbose** parameter set as true. For information on viewing the list of backup files, see [“Viewing Backup Files List” on page 138](#).

You can provide the following parameters based on your requirements:

Parameter	Description
tenant	<p>Name of the tenant in which the backup file is located.</p> <p>If you do not provide this parameter, the value is picked from <code>pg.gateway.elasticsearch.tenantId</code> property in config.properties file located at <code>IntegrationServer\instances\instance_name\packages\WmAPIGateway\config\resources\elasticsearch</code>.</p>
repoName	<p>Name of the repository from which you want to delete the backup file.</p> <p>If you do not provide this parameter, the value is picked from the <i>tenant</i> parameter.</p>
olderThan	<p>Option to delete the backup files that were created earlier than the given number of days. You must provide this parameter with the required number of days as seen in the following example:</p> <pre>apigatewayUtil.bat delete backups -olderThan 10</pre> <p>This command deletes the backup files that were created earlier than the past 10 days.</p>
analyticsDS	<p>Option to delete a backup of the analytics data store. Specify as true to delete a backup of the analytics data store.</p> <p>The sample command to delete a backup of the analytics data store is as follows:</p> <pre>apigatewayUtil.bat delete backup -analyticsDS true -name samplebackup</pre>

Parameter	Description
logLevel	<p>Level of log that you want to create. Log levels indicate the severity of logs. Available levels are:</p> <ul style="list-style-type: none"> ■ Info. Provides the list of regular events that occur during the process. These events are informative. ■ Debug. Provides the events that could be useful, if you have to debug the process. ■ Warning. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process. ■ Error. Indicates the events that stop the functionality from working as designed. ■ Trace. Provides the list of events in a much detailed manner that could be useful for debugging. <p>You can specify one of the log level with the LogLevel parameter. For example, to create a log file of Warning level when listing backup files, you can run the following command:</p> <pre>apigatewayUtil.bat list backup -logLevel warning</pre> <p>When you provide Error as the log level, then only the error level logs are saved. When you provide Debug as the log level, then Debug, Info, Warning and Error level logs are saved. When you provide Trace as log level, then all level logs are saved.</p> <p>This parameter is optional. If you do not specify the parameter, then the Info level logs are saved by default.</p>
logFileLocation	<p>Location where you want to save the log file.</p> <p>For example, to save the log file in <code>C:/apiglogs/backups</code>, you can provide the location as seen in the following example:</p> <pre>apigatewayUtil.bat create backup -name samplebackup -logFileLocation C:/apiglogs/backups</pre> <p>This parameter is optional. If you do not specify the parameter, the logs are saved in the following location <code>SAGInstallDir/IntegrationServer/instances/instance_name/packages/WmAPIGateway/cli/logs/APIGWUtility.log</code>.</p>

Specifying Log File Details

Log files created during backup and restore are used to monitor the process and diagnose problems encountered during the process. You can specify the required level of log and the location where the log file must be saved using the following parameters with the **apigatewayUtil** script:

Parameter	Description
logLevel	<p>Allows you to specify the required level of log that you want to create. Log levels indicate the severity of logs. Available levels are:</p> <ul style="list-style-type: none"> ■ Info. Provides the list of regular events that occur during the process. These events are informative. ■ Debug. Provides the events that could be useful, if you have to debug the process. ■ Warning. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process. ■ Error. Indicates the events that stop the functionality from working as designed. ■ Trace. Provides a much detailed events that could be useful for debugging. <p>You can specify one of the log level with the logLevel parameter. For example, to create a log file of Warning level when listing backup files, you can run the following command:</p> <pre>apigatewayUtil.bat list backup -logLevel warning</pre> <p>When you provide Error as the log level, then only the error level logs are saved. When you provide Debug as the log level, then Debug, Info, Warning and Error level logs are saved. When you provide Trace as log level, then all level logs are saved.</p> <p>This parameter is optional. If you do not specify the parameter, then the Info level logs are saved by default.</p>
logFileLocation	<p>Allows you to provide the location where you want to save the log file.</p> <p>For example, to save the log file in <code>C:/apiglogs/backups</code>, you can provide the location as seen in the following example:</p> <pre>apigatewayUtil.bat create backup -name samplebackup -logFileLocation C:/apiglogs/backups</pre> <p>This parameter is optional. If you do not specify the parameter, the logs are saved in the following location <code>SAGInstallDir/</code></p>

Parameter	Description
	IntegrationServer/instances/instance_name/packages/WmAPIGateway/cli/logs/APIGWUtility.log.

Viewing Exit Codes for Backup Script Operations

When you automate running the **apigatewayUtil** script, the script returns an exit code to indicate the process status. The following table lists the possible exit codes:

Exit codes	Description
0	Command execution successful.
1	Command execution failed due to some internal error.
8	Backup process in progress
9	Status of the backup process is failed.
10	Status of the backup process is success.
11	Backup file not found.
12	Specified backup files are successfully deleted.
13	Unable to delete the specified backup files.
14	Unable to roll over indexes.

➤ To view the exit codes in command prompt

1. Run the following command immediately after running the **apigatewayUtil** script with any command:

```
echo %ERRORLEVEL%
```

For example,

```
C:\SoftwareAG_1011\IntegrationServer\instances\default\packages\WmAPIGateway\cli\bin>
apigatewayUtil.bat create backup -name backup-apis
Initiated backup process for backup-apis
C:\SoftwareAG_1011\IntegrationServer\instances\default\packages\WmAPIGateway\cli\bin>echo
%ERRORLEVEL%
0
```

List of Indexes that can be included in backup

This sections lists the indexes that you can provide with the **include** parameter to include them in a backup file .

Index type	Index name
Analytics	gateway_default_monitorevents gateway_default_lifecycleevents gateway_default_threatprotectionevents gateway_default_policyviolationevents gateway_default_performancemetrics gateway_default_errorevents gateway_default_transactionalevents
Assets	gateway_default_approver gateway_default_strategies gateway_default_plans gateway_default_searchquery gateway_default_jndisettings gateway_default_publishinfo gateway_default_applications gateway_default_assertion gateway_default_stages gateway_default_uipolicy gateway_default_apis gateway_default_oauth2token gateway_default_rollback gateway_default_migrationinfos gateway_default_microgatewayregistrationinfo gateway_default_microgatewayassets gateway_default_outboundproxysettings gateway_default_truststores gateway_default_registeredapplications gateway_default_policyactions

Index type	Index name
	gateway_default_kerberossettings
	gateway_default_oauth2materializedtoken
	gateway_default_ispackages
	gateway_default_policy
	gateway_default_subscriptions
	gateway_default_oauth2scopedata
	gateway_default_gatewayscopes
	gateway_default_promotions
	gateway_default_quotaaccumulator
	gateway_default_jmsconnectionalias
	gateway_default_oauth2accesstoken
	gateway_default_urlaliases
	gateway_default_keystore
	gateway_default_portalgateways
	gateway_default_accessprofiles
	gateway_default_documents
	gateway_default_users
	gateway_default_jmstrigger
	gateway_default_aliases
	gateway_default_deploymentmap
	gateway_default_oauth2clientregistration
	gateway_default_configurations
	gateway_default_passmandata
	gateway_default_internalsettings
	gateway_default_egpolicyindexlists
	gateway_default_groups
	gateway_default_oauth2scopes

Index type	Index name
	gateway_default_oauth2refreshtoken
	gateway_default_approvalconfiguration
	gateway_default_webserviceendpointalias
	gateway_default_approvalrequest
	gateway_default_reversemaps
	gateway_default_packages
	gateway_default_accesscontrollist
	gateway_default_rule
	gateway_default_promotionsets
	gateway_default_proxybypass
	gateway_default_bindingassertion
	gateway_default_tokenassertion
	gateway_default_mediatortracespan
	gateway_default_serverlogtracespans
	gateway_default_requestresponsetracespans
Dashboard	gateway_default_dashboard
License	gateway_default_notifications
	gateway_default_apiusagedetails
	gateway_default_monthlyaggregateddetails
	gateway_default_licensemetrics
	gateway_default_notificationcriteria
Audit	gateway_default_auditlogs
Log	gateway_default_log
Trace Cache	gateway_default_cachestatistics
	gateway_default_serverlogtracespans
	gateway_default_mediatortracespan
	gateway_default_requestresponsetracespans

List of Indexes that can be backed up individually

This section lists the indexes that you can back up individually or combined together using the **indices** parameter:

- license_licensemetrics
- rule
- microgatewayassets
- stages
- approver
- urlaliases
- oauth2authcode
- registeredapplications
- migrationinfos
- analytics_monitorevents
- hints
- truststores
- kerberossettings
- audit_auditlogs
- outboundproxysettings
- dashboard-event-log
- accesscontrollist
- analytics_policyviolationevents
- jmtrigger
- serverlogtracespans
- microgatewayregistrationinfo
- appmesh
- analytics_performancemetrics
- keystore
- mediatortracespan
- appmesh_deployment_api

- strategies
- analytics_threatprotectionevents
- gatewayscopes
- aliases
- internalsettings
- portalgateways
- ispackages
- tokenassertion
- deploymentmap
- groups
- jndisettings
- oauth2token
- uipolicy
- license_monthlyaggregateddetails
- license_notifications
- license_apiusagedetails
- approvalconfiguration
- analytics_lifecycleevents
- configurations
- oauth2refreshtoken
- documents
- searchquery
- plans
- policy
- oauth2scopedata
- egpolicyindexlists
- license_notificationcriteria
- promotionsets
- applications

- rollback
- requestresponsetracespans
- webhooks
- passmandata
- oauth2accesstoken
- bindingassertion
- dashboard
- assertion
- publishinfo
- analytics_transactionalevents
- oauth2materializedtoken
- policyactions
- log
- apis
- accessprofiles
- quotaaccumulator
- promotions
- analytics_errorevents
- jmsconnectionalias
- webserviceendpointalias
- packages
- proxybypass
- reversemaps
- oauth2clientregistration
- cache_cachestatistics
- approvalrequest
- subscriptions
- oauth2scopes
- users

For information on creating a backup of required indexes, see “[Creating Backup of specific Indexes](#)” on page 131.

Platform Data Backup

Software AG recommends that you backup the platform data immediately after the initial setup and once after every platform configuration deployment changes. For example, Changing the JVM heap memory.

When you perform platform data backup of a cluster, you must backup the data of each node though the configuration is expected to be consistent among the nodes.

The list of API Gateway configuration data and their respective location is as follows:

Configuration	File name	Location	Is this supported by externalization?
Elasticsearch configuration	elasticsearch.yml	<code>SAGInstallDir/</code> <code>InternalDataStore/config/</code>	No
Elasticsearch client configuration	config.properties	<code>SAGInstallDir/</code> <code>IntegrationServer/</code> <code>instances/ instance_name/</code> <code>packages/WmAPIGateway/</code> <code>config/ resources/</code> <code>elasticsearch/</code>	Yes
Kibana configuration	kibana.yml	<code>SAGInstallDir/profiles/</code> <code>instance_name/ apigateway/</code> <code>dashboard/config/</code>	No
UI configurations	uiconfiguration.properties	<code>SAGInstallDir/</code> <code>profiles/instance_name/</code> <code>apigateway/config/</code>	No
Master password	mpw.dat	<code>SAGInstallDir/profiles/</code> <code>instance_name/</code> <code>configuration/security/</code> <code>passman/</code>	Yes
Server Ports configuration	If the portClusteringEnabled extended setting is set to false, you must create the server ports in each instance.		No
WebApp settings	com.softwareag.catalina.connector. http.pid-apigateway.properties	<code>SAGInstallDir/</code> <code>profiles/instance_name/</code> <code>configuration/</code>	Yes

Configuration	File name	Location	Is this supported by externalization?
	com.softwareag.catalina.connector. https.pid-apigateway.properties	com.softwareag.platform. config.propsloader/	
custom_wrapper.conf	Custom settings on memory	SAGInstallDir/profiles/ instance_name/ configuration/	No

You can back up the platform data using the **export platformConfiguration** command. This script backs up the platform configurations that are covered using Externalized configuration feature (indicated in the last column of the above table). You can backup other configurations manually.

- For information on using the **export platformConfiguration** command, see “[Creating Platform Data Backup using Script](#)” on page 157.
- For the list of data that must be backed up manually, see “[Creating Platform Data Backup Manually](#)” on page 159.

Creating Platform Data Backup using Script

The script saves the platform data backup in the YML format.

Pre-requisites:

- Ensure that you have the API Gateway running during backup.
- Ensure that you have the repository where you want to save the platform data backup.

➤ To take backup of platform data

1. From the command prompt, go to
`SAGInstallDir/IntegrationServer/instances/instance_name/packages/WmAPIGateway/cli/bin.`
2. Run the following command to create a backup in the default location:

Linux

```
./apigatewayUtil.sh export platformConfiguration -url URL of the instance
-username User name to access the URL -password Password to access the URL
-filePath Location where the backup must be saved
```

Windows

```
apigatewayUtil.bat export platformConfiguration -url URL of the instance
-username User name to access the URL -password Password to access the URL
-filePath Location where the backup must be saved
```

You can include the following parameters to customize the backup as per your requirements.

Parameter	Description
password	Password required to access the specified instance (from where the platform data backup is taken). If you want to avoid providing the password in clear text when running the apigatewayutil script, you can skip this parameter. When you run the command without the password parameter, then you will be prompted to enter the password. You can provide the password in hidden characters.
logLevel	<p>Allows you to specify the required level of log that you want to create. Log levels indicate the severity of logs. Available levels are:</p> <ul style="list-style-type: none"> ■ Info. Provides the list of regular events that occur during the process. These events are informative. ■ Debug. Provides the events that could be useful, if you have to debug the process. ■ Warning. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process. ■ Error. Indicates the events that stop the functionality from working as designed. ■ Trace. Provides a much detailed events that could be useful for debugging. <p>You can specify one of the log level with the logLevel parameter. For example, to create a log file of Warning level when listing backup files, you can run the following command:</p> <pre>apigatewayUtil.bat list backup -logLevel warning</pre> <p>When you provide Error as the log level, then only the error level logs are saved. When you provide Debug as the log level, then Debug, Info, Warning and Error level logs are saved. When you provide Trace as log level, then all level logs are saved.</p> <p>This parameter is optional. If you do not specify the parameter, then the Info level logs are saved by default.</p>
logFileLocation	<p>Allows you to provide the location where you want to save the log file. For example, to save the log file in <code>C:/apiglogs/backups</code>, you can provide the location as seen in the following example:</p> <pre>apigatewayUtil.bat create backup -name samplebackup -logFileLocation C:/apiglogs/backups</pre>

Parameter	Description
	This parameter is optional. If you do not specify the parameter, the logs are saved in the following location <code>SAGInstallDir\IntegrationServer\instances\instance_name\packages\WmAPIGateway\cli\logs\APIGWUtility.log</code>

Creating Platform Data Backup Manually

The backup script, **apigatewayUtil**, does not backup the following configuration data. For the backup of configuration data, you need to create copies of the following files in a Version Control system or a Network File Storage system:

Configuration	File name	File location
UI configurations	uiconfiguration.properties	<code>SAGInstallDir\profiles\instance_name\apigateway\config\</code>
WebApp settings	com.softwareag.catalina.connector. http.pid-apigateway.properties com.softwareag.catalina.connector. https.pid-apigateway.properties	<code>SAGInstallDir\profiles\instance_name\configuration\com.softwareag.platform.config.propsloader\</code>
Custom ESB packages	File name specified by users.	Location used by users to save the file. Generally, the customized packages are saved in the following location: <code>SAGInstallDir\IntegrationServer\tenant\packages\</code>
custom_wrapper.conf	Custom settings on memory	<code>SAGInstallDir\profiles\instance_name\configuration\</code>

Backup folders monitoring

You must manually monitor the folders, in which you maintain the backup files, to identify any possible failures. This process cannot be automated and thus, this has to manually monitored.

Troubleshooting a Failed Backup

To diagnose any problems and troubleshoot the failing backup operations, you can activate debug logging when creating a backup. The debug log is activated by setting the debug parameter to true.

Linux

```
./apigatewayUtil.sh create backup -debug true
```

Windows

```
apigatewayUtil.sh create backup -debug true
```

The activated debug logging provides detailed information on the backup process on the console. You can find further details in the logs created in the `SAGInstallDir\InternalDataStore\logs` folder.

Important:

When taking backup in synchronous mode, if you encounter a socket time out error then backup in asynchronous mode.

Troubleshooting Tips: API Data Store Backup

I see an error while executing apigatewayUtil.bat create backup command

The apigatewayUtil.bat create backup command fails to execute with the following BeanCreationException error:

```
org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'elasticSearchClient' defined in URL file:E:/ESB/WM/IntegrationServer/instances/default/packages/WmAPIGateway/config/resources/beans/gateway-es-store.xml:Invocation of init method failed; nested exception is NoNodeAvailableException[None of the configured nodes are available:  
[
```

Resolution:

Ensure that you have provided a valid elasticsearch hostname and port number in the `config.properties` file located at `SAGInstallDir\IntegrationServer\instances\default\packages\WmAPIGateway\config\resources\elasticsearch`.

When the Elasticsearch is clustered, I am unable to backup the API Gateway database

When I attempt to backup the API Gateway database with the following commands, they fail:

- apigatewayUtil.bat create backup
- apigatewayUtil.bat status backup
- apigatewayUtil.bat list backup
- apigatewayUtil.bat list manageRepo

This is because of the `path.repo` specified in the `elasticsearch.yml`. The `path.repo` was pointing to a local file instead of a shared file location.

Resolution:

In the Elasticsearch cluster environment, it is mandatory to specify a shared file location as `path.repo` in the `elasticsearch.yml` located at `SAGInstallDir\InternalDataStore\config`.

Error occurs when I access the backup repository

When I access the backup repository after some configuration changes, I see the following error message:

Error While getting all snapshots or repository - default.Message-[default] could not read repository data from index blb.

In such scenarios, you can delete the repository and create a fresh one to store your backup files by following the steps given in Resolution 1. Or, you can move the existing backup files to a new repository by following the steps in Resolution 2 .

Resolution 1:

To delete the problematic repository and create a new one

- Run the following command to delete the repository.

Linux

```
./apigatewayUtil.sh delete manageRepo -repoName repository_name
```

Windows

```
apigatewayUtil.bat delete manageRepo -repoName repository_name
```

Example :

```
apigatewayUtil.bat delete manageRepo -repoName myrepo
```

The repository is deleted.

- Create a new backup repository. For details on the commands see “[Creating Backup of specific Indexes](#)” on page 131.

You can now store your backup in a new repository.

Resolution 2:

To move the backup files to another repository

- Stop API Data Store.
- Create a new repository to move the required backup files.
- Copy the backup to the new repository (file system copying).
- Start API Data Store.
- Verify if the backup was created. For details on the command see “[Verifying Backup Status](#)” on page 142.

When I modify a Network File System repository, how do I ensure that all Internal Data Store yaml files are updated without corrupting the repository

In such scenarios, you can copy the existing backup file to the new repository or if you do not need the backup file you can delete the backup file in the existing repository.

Resolution 1:

To copy the backup file to the new repository

- Stop API Data Store.
- Copy all the existing backup files to a new repository(file system copy).
- Start API Data Store.

Note:

Software AG recommends not to run any command before copying the old backup.

Resolution 2:

To delete the backup file in the existing repository

- Stop API Data Store.
- Delete all old repository.

Linux

```
./apigatewayUtil.sh delete manageRepo -repoName repository_name
```

Windows

```
apigatewayUtil.bat delete manageRepo -repoName repository_name
```

Example :

```
apigatewayUtil.bat delete manageRepo -repoName myrepo
```

The repository is deleted.

- Start API Data Store.

Restore Operation

When you setup a new API Gateway instance or a node, you can restore the data that you have backed up from an earlier instance to recreate the same instance.

Pre-requisites

Before you perform a data restore, ensure that the:

- Backup snapshots are taken as per the specified RPO.
- Required snapshots are available in the corresponding repositories.

- API Gateway components are up and running.

Similar to the backup operation, you must perform the following processes for complete restoring of the API Gateway data:

Restore using the apigatewayUtil script - to restore the API Data Store. For information on the script, see [“Restoring Data Store Backup” on page 163](#).

Manual restore - to restore the API Gateway configuration data. For more information, see [“Restoring Platform Data backed up using the Script” on page 170](#) or [“Restoring the manually backed up Platform data” on page 170](#).

Restoring Data Store Backup

You can restore the data from backup snapshots using the **apigatewayUtil** script.

Important:

If your restore operation fails for some reason and if the error message instructs you to use the **perform_open_indices** command, follow the steps given in the section [“Troubleshooting a Failed Restore” on page 170](#).

➤ To restore API Data Store

1. From the command prompt, go to
`SAGInstallDir/IntegrationServer/instances/default/packages/WmAPIGateway/cli/bin.`

Note:

Replace `default` with the corresponding instance name.

2. Run the following command to restore a backup file:

Linux

```
./apigatewayUtil.sh restore backup -name backup-file-name
```

Windows

```
apigatewayUtil.bat restore backup -name backup-file-name
```

The specified backup is restored. Since, the **repoName** parameter is not specified in the above command, the value is picked from the *tenant* parameter.

You can include the following parameters as per your requirements:

Parameter	Description
tenant	Name of the tenant from which you want to restore. If you do not provide this parameter, the value is picked from <code>pg.gateway.elasticsearch.tenantId</code> property in config.properties file located

Parameter	Description
	at <code>IntegrationServer\instances\instance_name\packages\WmAPIGateway\config\resources\elasticsearch</code> .
repoName	<p>Name of the repository from which you want to restore the backup file.</p> <p>If you do not provide this parameter, the value is picked from the <code>tenant</code> parameter.</p> <p>In case you have more than one repository, you must provide this parameter with the name of the repository that you want to restore from.</p>
sync	<p>Option to specify whether the restore process is performed in a synchronously or asynchronously. The possible values are:</p> <ul style="list-style-type: none"> ■ <code>true</code>. The restore process is synchronous. That is, the script will wait until the restore is completed. Depending on the size of the data being restored, this could be a time-consuming process. ■ <code>false</code>. The process is asynchronous. That is, the script does not wait for the completion of the restore operation. Rather, the restore request will be issued, and the script will complete. This is the default value. You can later check the status of the restore process using the status restore command. For information on viewing the restore status, see “Verifying Restore Status” on page 167.
srcTenant	<p>Option to specify whether that the tenant name in the backup file is different from the tenant name on which the restore is being performed. Hence, this must be used in combination with the <code>tenant</code> parameter that specifies the tenant name to which the data backup is being restored. That is,</p> <pre data-bbox="401 1220 866 1332">apigatewayUtil.sh restore backup -name backup_file_name -srcTenant backup_tenant_name -tenant tenant_name</pre>
	If you do not provide the <code>tenant</code> parameter, then the script assumes the tenant name as the value for <code>srcTenant</code> .
Note: The <code>srcTenant</code> , <code>aggregate</code> , and <code>restoreClusterState</code> parameters must be used with caution and only in cases where it is deemed necessary. For normal incremental backup and restore operations, these parameters do not play a significant role.	
include	<p>Option to include any of the following based on your input:</p> <ul style="list-style-type: none"> ■ <code>analytics</code>. Restores the analytics data, logs, and events data. ■ <code>assets</code>. Restores assets. ■ <code>license</code>. Restores license metrics.

Parameter	Description
	<ul style="list-style-type: none"> ■ <code>audit</code>. Restores audit logs. ■ <code>log</code>. Restores log data. <p>You can provide one or more of the above values separated by commas, and without spaces. For example, to restore analytics and assets, you can provide</p> <pre style="background-color: #f0f0f0; padding: 5px;">./apigatewayUtil.sh restore backup -name backup_file_name -repoName repository_name --include analytics,assets</pre>
analyticsDS	<p>Option to restore a backup of the analytics data store. Specify as true to restore a backup of the analytics data store.</p> <p>The sample command to restore a backup of the analytics data store is as follows:</p> <pre style="background-color: #f0f0f0; padding: 5px;">apigatewayUtil.bat restore backup -analyticsDS true -name backup_file_name</pre> <p>You can use the <code>include</code> parameter with this command to include the different types of data that you want to restore.</p> <p>If you do not provide any options for the <code>include</code> parameter:</p> <ul style="list-style-type: none"> ■ with <code>-analyticsDS true</code>. Audit data, analytics data, and log data is restored. ■ without <code>-analyticsDS true</code>. Only assets data is restored for API Data Store. If you want include other data besides assets, for example analytics data, use the <code>-include</code> parameter explicitly.
aggregate	<p>Option to specify whether the existing license, logs, audit, and analytics data should be merged with the restored data. The possible values are:</p> <ul style="list-style-type: none"> ■ <code>true</code>. The existing data is merged with the restored data. If some analytics data is present in the current instance, and you restore from a backup, which also has analytics data, and if you provide <code>-aggregate true</code>, then the script restores the analytics data from the backup into a new index and includes it in the index alias for analytics. So you can find the existing analytics data as well the analytics data from the restored backup file in your instance. ■ <code>false</code>. The existing data is replaced with the restored data. This is the default value. <p>Note: This parameter is not applicable for assets data. That is, you cannot aggregate assets data. Hence, if you had provided -include assets in your restore command, then the parameter is ignored because it is not applicable.</p>

Parameter	Description
restoreClusterState	<p>Option to specify whether the global state settings such as templates and cluster state must be restored. The cluster state includes information such as persistent cluster settings, index templates, pipelines, and so on. It must be restored only in a new Elasticsearch instance. The possible values for this parameter are:</p> <ul style="list-style-type: none"> ■ <i>true</i>. The global state settings are restored. ■ <i>false</i>. The global state settings are not restored. This is the default value. <p>Do not use this parameter for normal restore operations. Use only when restoring backup from a different Elasticsearch or when the <code>-srcTenant</code> parameter is specified.</p>
logLevel	<p>Level of log that you want to create. Log levels indicate the severity of logs. Available levels are:</p> <ul style="list-style-type: none"> ■ Info. Provides the list of regular events that occur during the process. These events are informative. ■ Debug. Provides the events that could be useful, if you have to debug the process. ■ Warning. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process. ■ Error. Indicates the events that stop the functionality from working as designed. ■ Trace. Provides the list of events in a much detailed manner that could be useful for debugging. <p>You can specify one of the log level with the LogLevel parameter. For example, to create a log file of Warning level when listing backup files, you can run the following command:</p> <pre>apigatewayUtil.bat list backup -logLevel warning</pre> <p>When you provide Error as the log level, then only the error level logs are saved. When you provide Debug as the log level, then Debug, Info, Warning and Error level logs are saved. When you provide Trace as log level, then all level logs are saved.</p> <p>This parameter is optional. If you do not specify the parameter, then the Info level logs are saved by default.</p>
logFileLocation	<p>Location where you want to save the log file.</p> <p>For example, to save the log file in <code>C:/apiglogs/backups</code>, you can provide the location as seen in the following example:</p>

Parameter	Description
	<pre>apigatewayUtil.bat create backup -name samplebackup -logFileLocation C:/apiglogs/backups</pre> <p>This parameter is optional. If you do not specify the parameter, the logs are saved in the following location <code>SAGInstallDir/IntegrationServer/instances/instance_name/packages/WmAPIGateway/cli/logs/APIGWUtility.log</code></p>

The parameters listed above are optional. To restore a backup from the repository that you configured, use the **repoName** parameter, and provide the repository name. That is,

Linux

```
./apigatewayUtil.sh restore backup -name backup_file_name  
-repoName repository_name
```

Windows

```
apigatewayUtil.bat restore backup -name backup12april -repoName S3_repo
```

Note:

If you are providing the **srcTenant** parameter, or setting the **aggregate** or **restoreClusterState** as true, then Software AG recommends that you take a backup of the node that you need to restore before performing the restore.

3. Restart API Data Store.

In cluster setups, restart all nodes in the cluster.

Verifying Restore Status

When you run the `restore backup` command, the restore process is initiated. You can then verify the status of a restore operation using the `status restore` command. This command checks the status of the given backup file and displays one of the following results:

Status	Description
Success	The restore of the given backup is successfully restored with the given parameters.
Failed	The restore process has failed. If the error message instructs to perform open indices, follow the steps in the section, "Troubleshooting a Failed Restore" on page 170 .
In progress	The restore process is still in progress. This usually happens when the data being restore is large and you verify the status as soon as you initiate the restore process.

Status	Description
Completed with warnings	The restore process is completed. However, there are some warning messages. You can check the warning messages in the log file found in the <code>SAGInstallDir\InternalDataStore\logs</code> folder.

➤ To verify the status of a restore operation

1. From the command prompt, go to `SAGInstallDir\IntegrationServer\instances\instance_name\packages\WmAPIGateway\cli\bin\.`.
2. Run the following command:

Linux

```
./apigatewayUtil.sh status restore -name backup-file-name
```

Windows

```
apigatewayUtil.bat status restore -name backup-file-name
```

The restore status appears as shown below:

```
C:\SoftwareAG\IntegrationServer\instances\default\packages\WmAPIGateway\cli\bin>apigatewayutil.bat
status restore -name sample
The restore of sample is completed successfully
```

You can include the following parameters as per your requirements:

Parameter	Description
tenant	Name of the tenant that you want to verify restore status. If you do not provide this parameter, the value is picked from <code>pg.gateway.elasticsearch.tenantId</code> property in config.properties file located at <code>IntegrationServer\instances\instance_name\packages\WmAPIGateway\config\resources\elasticsearch</code> .
repoName	Name of the repository that you want to verify. If you do not provide this parameter, the value is picked from the <code>tenant</code> parameter.
analyticsDS	Option to get the restore status of the analytics data store. Specify as true to get the restore status of the analytics data store. The sample command to get the restore status of the analytics data store is as follows:

Parameter	Description
	<pre>apigatewayUtil.bat status restore -analyticsDS true -name samplebackup</pre>

logLevel	<p>Allows you to specify the required level of log that you want to create. Log levels indicate the severity of logs. Available levels are:</p> <ul style="list-style-type: none"> ■ Info. Provides the list of regular events that occur during the process. These events are informative. ■ Debug. Provides the events that could be useful, if you have to debug the process. ■ Warning. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process. ■ Error. Indicates the events that stop the functionality from working as designed. ■ Trace. Provides a much detailed events that could be useful for debugging. <p>You can specify one of the log level with the LogLevel parameter. For example, to create a log file of Warning level when listing backup files, you can run the following command:</p> <pre>apigatewayUtil.bat list backup -logLevel warning</pre> <p>When you provide Error as the log level, then only the error level logs are saved. When you provide Debug as the log level, then Debug, Info, Warning and Error level logs are saved. When you provide Trace as log level, then all level logs are saved.</p> <p>This parameter is optional. If you do not specify the parameter, then the Info level logs are saved by default.</p>
----------	---

logFileLocation	<p>Allows you to provide the location where you want to save the log file.</p> <p>For example, to save the log file in <code>C:/apiglogs/backups</code>, you can provide the location as seen in the following example:</p> <pre>apigatewayUtil.bat create backup -name samplebackup -logFileLocation C:/apiglogs/backups</pre> <p>This parameter is optional. If you do not specify the parameter, the logs are saved in the following location <code>SAGIInstallDir/IntegrationServer/instances/instance_name/packages/WmAPIGateway/cli/logs/APIGWUtility.log</code>.</p>
-----------------	---

If you run the `status restore` command without the **tenant** and **repoName** parameters, then system checks for the given backup file in the default tenant and repository.

Restoring Platform Data backed up using the Script

➤ To restore the platform data backed up using the script

1. Copy the YML file, that was exported with the platform data, to the location `SAGInstallDir\IntegrationServer\instances\instance_name\packages\WmAPIGateway\resources\configuration`.

Alternatively, you can copy the backup YML file to a location and specify the location in the Master configuration file, `config-sources.yml`, located at `SAGInstallDir\IntegrationServer\instances\instance_name\packages\WmAPIGateway\resources\configuration`.

2. Restart API Gateway.

In cluster setups, restart all nodes in the cluster.

Restoring the manually backed up Platform data

➤ To restore the platform data backed up manually

1. Copy the required configuration files from the repository, to the respective location in the API Gateway installation. For the list of configuration files and their locations, see the table provided in the section, “[Platform Data Backup](#)” on page 156.
2. Restart API Gateway.

In cluster setups, restart all nodes in the cluster.

Troubleshooting a Failed Restore

When a restore fails, you can analyze and understand the reason for failure from the error message that appears after the operation. If the error message instructs you to use the **perform_open_indices** command, follow the steps given in this section.

When you perform a restore operation, the indices in the API Data Store are closed to avoid any overwriting of data. After the restore process is over, the indices are opened so that the application starts using the Data Store. However, if a restore operation fails, you must open the indices for the regular operations to continue. For example, if a restore operation fails due to insufficient memory, then the error message that appears instructs you to perform open indices.

If the error message does not display enough details, you can check the logs for further details. You can check the logs saved during the process in the `SAGInstallDir\InternalDataStore\logs` folder.

➤ To open indices

1. From the command prompt, go to
`SAGInstallDir/IntegrationServer/instances/instance_name/packages/WmAPIGateway/cli/bin.`
2. Run the following command to delete a backup file:

Linux

```
./apigatewayUtil.sh perform open_indices
```

Windows

```
apigatewayUtil.bat perform open_indices
```

You can specify the log file location and log level for the above operation using these parameters:

Parameter	Description
analyticsDS	<p>Option to open indices in analytics data store. Specify as true to open indices in analytics data store.</p> <p>The sample command to open indices in analytics data store is as follows:</p> <pre>apigatewayUtil.bat perform open_indices -analyticsDS true</pre>
logLevel	<p>Allows you to specify the required level of log that you want to create. Log levels indicate the severity of logs. Available levels are:</p> <ul style="list-style-type: none"> ■ Info. Provides the list of regular events that occur during the process. These events are informative. ■ Debug. Provides the events that could be useful, if you have to debug the process. ■ Warning. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process. ■ Error. Indicates the events that stop the functionality from working as designed. ■ Trace. Provides a much detailed events that could be useful for debugging. <p>You can specify one of the log level with the LogLevel parameter. For example, to create a log file of Warning level when listing backup files, you can run the following command:</p> <pre>apigatewayUtil.bat list backup -LogLevel warning</pre>

Parameter	Description
	<p>When you provide Error as the log level, then only the error level logs are saved. When you provide Debug as the log level, then Debug, Info, Warning and Error level logs are saved. When you provide Trace as log level, then all level logs are saved.</p> <p>This parameter is optional. If you do not specify the parameter, then the Info level logs are saved by default.</p>
logFileLocation	<p>Allows you to provide the location where you want to save the log file.</p> <p>For example, to save the log file in <code>C:/apiglogs/backups</code>, you can provide the location as seen in the following example:</p> <pre>apigatewayUtil.bat create backup -name samplebackup -logFileLocation C:/apiglogs/backups</pre> <p>This parameter is optional. If you do not specify the parameter, the logs are saved in the following location <code>SAGInstallDir/IntegrationServer/instances/instance_name/packages/WmAPIGateway/cli/logs/APIGWUtility.log</code>.</p>

Adding New Nodes to an Elasticsearch Cluster

This section explains how to add a new node to an Elasticsearch cluster. You can add nodes to a cluster by configuring new nodes to find an existing cluster and start them up.

For example, consider that a new node, *node 4*, is added to a cluster that already has three nodes in it namely, *node1*, *node2*, and *node3*.

➤ To add new node to a cluster

1. Open **elasticsearch.yml** located at `SAG_root/InternalDataStore/config`, where the new node is being added.

The following configuration is a sample of how the configuration appears initially.

```
cluster.name:"SAG_EventDataStore"
node.name: node4
path.logs: SAG_root\InternalDataStore\logs
network.host:0.0.0.0
http.port:9240
discovery.seed_hosts: ["node4:9340"]
transport.port:9340
path.repo:['SAG_root\InternalDataStore/archives']
cluster.initial_master_nodes:["node4"]
xpack.ml.enabled: false
```

```
xpack.security.enabled: false
xpack.security.transport.ssl.enabled: false
xpack.security.http.ssl.enabled: false
action.destructive_requires_name: false
```

2. Provide the name of the node, as seen in the **node.name** property, and port number used by the node, as seen in the **http.port** property, in the **discovery.seed_hosts** property in the following format:

host_name:port_name

For example

node4:9340

Sample configuration after providing the new node details:

```
cluster.name:"SAG_EventDataStore"
cluster.initial_master_nodes:["node1","node2","node3"]
node.name: node4
path.logs: SAG_root\InternalDataStore/logs
network.host:0.0.0.0
http.port:9240
discovery.seed_hosts: ["node1:9340","node2:9340","node3":9340,"node4:9340"]
transport.port:9340
path.repo:['SAG_root\InternalDataStore/archives']
xpack.ml.enabled: false
xpack.security.enabled: false
xpack.security.transport.ssl.enabled: false
xpack.security.http.ssl.enabled: false
action.destructive_requires_name: false
```

3. Save the configuration. The new node is added to the cluster.

Note:

When you restart an Elasticsearch cluster, you must restart the master node first.

If you want to remove a node from a cluster do the following:

1. Open the **elasticsearch.yml** file located at *SAG_root/InternalDataStore/config/*.
2. Remove the node listed in the format host_name:port_name in the **discovery.seed_hosts** property.
3. Save the **elasticsearch.yml** file and restart the Elasticsearch cluster. The specified node is now removed from the cluster.

Troubleshooting Tips: API Data Store

API Gateway package is not accessible from Integration Server.

The following error message appears:

com.softwareag.apigateway.core.exceptions.DataStoreException

This problem could be because the **defaultEncoding** extended setting is modified.

CAUTION:

Do not modify this value. If you modify this value, your API Gateway instance will not function as this value is used in encoding all API Gateway transactions. If you migrate from one setup to another, ensure you have specified the same value for the **defaultEncoding** setting as the source instance. If these values are not the same, the target API Gateway instance does not start.

This error message appears: com.softwareag.apigateway.core.exceptions.DataStoreException:
com.softwareag.apigateway.core.exceptions.DataStoreException:

Resolution:

Set the value of the **defaultEncoding** extended setting as UTF-8.

The Event data store on API Gateway is using a lot of disk space.

The Elasticsearch JVM is unable to allocate memory for internal objects. Either other process in the machine are consuming more memory or Elasticsearch is not given sufficient heap space.

Also JVM has written these information in write.lock file, which is solely used by Elasticsearch for its internal purpose. Elasticsearch expects this file to be of size 0 and should not be modified. Since jvm has written the data, it is showing that as error and filling the disk with log.

Resolution:

Increase the Event data store JVM heap size.

I have exceeded the limit for total fields [1000] in index [gateway_default_analytics_]

The following error message appears:

2019-09-19 00:29:02 UTC [YAI.0300.9999E] error while saving doc Index - gateway_default_analytics, typeName - transactionalEvents: POST
http://10.177.129.5:9241/gateway_default_analytics/transactionalEvents: HTTP/1.1 400 Bad Request
{"error":{"root_cause":[{"type":"illegal_argument_exception","reason":"Limit of total fields [1000] in index [gateway_default_analytics] has been exceeded"}],"type":"illegal_argument_exception","reason":"Limit of total fields [1000] in index [gateway_default_analytics] has been exceeded"},"status":400}

Resolution:

Increase the limit for total fields.

```
PUT /gateway_default_analytics/_settings{"index.mapping.total_fields.limit": 20000}
```

I experienced a low disk space issue and my API Gateway has stopped working for WRITE operations.

The following error message appears:

Exception: [WARN][o.e.c.r.a.DiskThresholdMonitor] [localhost1568897216386] flood stage disk watermark [95%] exceeded on
 [BOf6SQe2SwyI93vi4RIBNQ][localhost1568897216386][C:\SoftwareAG\InternalDataStore\data\nodes\0] free: 2.4gb[2.4%], all indices on this node will be marked read-onlySaving an API -> error message ("Saving API failed. com.softwareag.apigateway.core.exceptions.DataStoreException: Error while saving the document. doc Id - 6d5c7ac0-574a-4a53-acba-a738f21e3142, type name - _doc, message - "index [gateway_default_policy] blocked by: [FORBIDDEN/12/index read-only / allow delete (api)];" ")

Resolution:

1. Purge the API Gateway data logs and other unwanted data from Elasticsearch. For information on data purging, see "[Archive and Purge using UI](#)" on page 109.
2. Run the following cURL command on the Elasticsearch immediately after you have freed up atleast 15% of the disk space.

```
curl -XPUT -H "Content-Type: application/json" http://localhost:9240/_all/_settings -d '{"index.blocks.read_only_allow_delete": null}'
```

My Elasticsearch server is not starting. I get a "bootstrap checks failed" error.

The following error message appears:

[2020-03-25T09:09:20,298][INFO][o.e.b.BootstrapChecks] [itsbebel00471.jnj.com1585050877659] bound or publishing to a non-loopback address, enforcing bootstrap checks
 [2020-03-25T09:09:20,299][ERROR][o.e.b.Bootstrap] [itsbebel00471.jnj.com1585050877659] node validation exception [1] bootstrap checks failed [1]: system call filters failed to install; check the logs and fix your configuration or disable system call filters at your own risk.

Resolution:

Add bootstrap.system_call_filter: false setting to elasticsearch.yml

When I access the audit logs, the internal datastore, crashes.

The following error message appears:

[2020-03-03T10:03:33,857][ERROR][o.e.ExceptionsHelper] [daeipresal43558.eur.ad .sag1580968109910] fatal error at org.elasticsearch.ExceptionsHelper.lambda\$maybeDieOnAnotherThread\$2(ExceptionsHelper.java:310) at java.util.Optional.ifPresent(Optional.java:159) at org.elasticsearch.ExceptionsHelper.maybeDieOnAnotherThread(ExceptionsHelper.java:300) at org.elasticsearch.http.netty4.Netty4HttpRequestHandler.exceptionCaught(Netty4HttpRequestHandler.java:76) [2020-03-03T10:03:33,858][ERROR][o.e.ExceptionsHelper] [daeipresal43558.eur.ad .sag1580968109910] fatal error at org.elasticsearch.ExceptionsHelper.lambda\$maybeDieOnAnotherThread\$2(ExceptionsHelper.java:310) at java.util.Optional.ifPresent(Optional.java:159) at

```
org.elasticsearch.ExceptionsHelper.maybeDieOnAnotherThread(ExceptionsHelper.java:300) at  
org.elasticsearch.http.netty4.Netty4HttpRequestHandler.exceptionCaught(Netty4HttpRequestHandler.java:76)  
[2020-03-03T10:03:33,867][ERROR][o.e.b.ElasticsearchUncaughtExceptionHandler] [  
daeipresal43558.eur.ad .sag1580968109910] fatal error in thread [Thread-176], exiting  
java.lang.OutOfMemoryError: Java heap space
```

Resolution:

Set the -XX:MaxDirectMemorySize property to 4095m.

Monitoring API Gateway

Monitoring is an important part of maintaining the reliability, availability, and performance of API Gateway. The API Gateway monitoring service enables you to

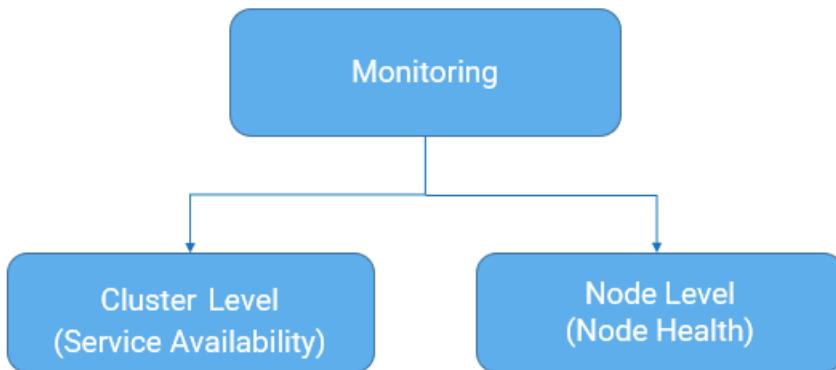
- View the health and performance of the application.
- Discover, analyze, and debug the application issues before they adversely impact the system. This improves business continuity and reduce application downtime.
- Understand the behavior of the application, scale for business volumes, and benefit from the cost optimization.

Through API Gateway monitoring, you can monitor the health and resources of the following API Gateway components:

- API Gateway Server. For details about the monitoring specifics, see [“Monitoring API Gateway” on page 189](#).
- API Data Store. For details about the monitoring specifics, see [“Monitoring API Data Store” on page 303](#).
- Terracotta Server Array. For details about the monitoring specifics, see [“Monitoring Terracotta” on page 314](#).
- Kibana. For details about the monitoring specifics, see [“Monitoring Kibana” on page 319](#).

You can monitor API Gateway at:

- Cluster-level. For details about the monitoring specifics, see [“Cluster-level Monitoring” on page 177](#).
- Node-level. For details about the monitoring specifics, see [“Node-level Monitoring” on page 183](#).



In a Highly Available (HA) set up, multiple API Gateway instances operate as a cluster. The HA set up ensures that there is no single point of failure in a multiple node deployment such as multiple virtual machines or Kubernetes pods.

- The cluster-level monitoring ensures service availability, that is, availability of access and functionality (serving API requests) of API Gateway. Service availability must be measured only at the cluster-level, for example, one node down in the cluster does not mean a service outage.
- The node-level monitoring ensures the ability of a particular instance of API Gateway components to serve the functionality (API requests) in the node. Node-health must be dealt within isolation by setting up probes per instance of each API Gateway component.

Cluster-level Monitoring

The cluster-level monitoring ensures service availability, that is, availability of access and functionality (serving API requests) of API Gateway. Through cluster-level monitoring, you can check:

- If the runtime is available and ready to serve the traffic.
- If the API Gateway administrator console is accessible.

How do I monitor the cluster health of API Gateway?

You can set up the Readiness Probe, Runtime Service Health Probe, and Administration Service Health Probe to monitor the cluster health.

Requirement	Type of Impact	Solution
For API Gateway, is there an endpoint that returns <i>yes</i> or <i>no</i> about its service availability, that is, readiness for serving the incoming API requests?	<i>Business Impact.</i> To know if there is an outage in API Gateway.	Use Readiness Probe .

Requirement	Type of Impact	Solution
For API Gateway, is there an endpoint that indicates the availability of the administrator user console?	<i>Operational Impact.</i> To know if the administrator user console is available.	Use Administration Service Health Probe .
For API Gateway, is there an endpoint that indicates the cluster health and its details?	<i>Technical Impact.</i> To know the details about where the fault lies when there is a cluster failure.	Use Runtime Service Health Probe .

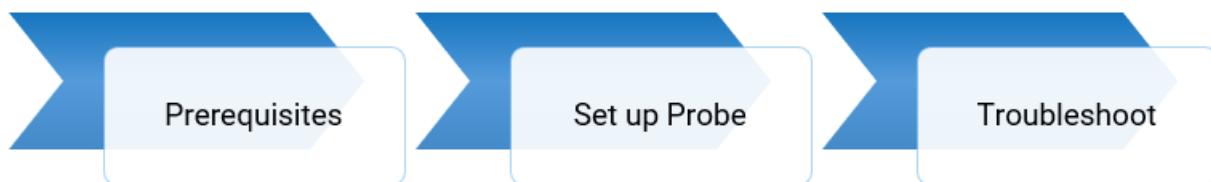
How do the probes help in cluster-level monitoring?

	Readiness Probe	Runtime Service Health Probe	Administration Service Health Probe
What is it?	Indicates if the traffic-serving port of API Gateway is ready to accept requests.	Reports on the overall cluster health and indicates if the components of API Gateway are in an operational state.	Indicates if the API Gateway administrator console is available and accessible.
When is it used?	To continuously check and report on the service availability of API Gateway.	To continuously report on the cluster health with the details of the components involved in clustering.	To continuously report on the availability of the administrator console and API analytics.

Note:

The points in the table are also applicable to scenarios where the cluster health is NOT OK, for example, API Data Store or Terracotta failure. Such scenarios do not always mean an outage. API Gateway may still be able to process the requests.

How do I set up probes?



Prerequisites:

- You must have a valid API Gateway user credential for using the Readiness Probe, Runtime Service Health Probe, and Administration Service Health Probe.
- All the cluster-level probes must be setup to target API Gateway load balancer endpoint.

- Software AG recommends to set up a dedicated port for monitoring with an appropriate private thread pool.

Readiness Probe at Cluster-Level

To monitor the readiness of API Gateway, that is to check if API Gateway is ready to accept the requests, use the following REST endpoint:

`GET /rest/apigateway/health`

The following table shows the response code and the description.

Response	Description
200 OK	Readiness check is successful. Readiness probe continues to reply OK if API Gateway remains in an operational state to serve the requests.
500 Internal server error	Readiness check failed and denotes a problem.
timeout or no response as the request did not reach the probe	Several factors can contribute to the delay when the Readiness Probe initiates, which may result in the timeout errors. To know the reasons for timeout errors, see “ Causes for timeout errors ” on page 303 for more information.

Note:

As this is a Readiness Probe and only the response status code is essential, by design, JSON payload is not returned in the response for both success and failure scenarios.

Runtime Service Health Probe at Cluster-Level

To monitor the runtime service health of API Gateway, that is to check the cluster health of API Gateway, use the following REST endpoint:

`GET /rest/apigateway/health/engine`

The following table shows the response code and the description.

Response	Description
200 OK	Runtime service health check is successful. The response is 200 OK when all APIs are activated after a startup.
500 Internal server error	Runtime service health check failed and denotes a problem. The response JSON indicates the problem.
timeout or no response as the request did not reach the probe	Several factors can contribute to the delay when the Runtime Service Health Probe initiates, which may result in the timeout errors. To know the reasons for timeout errors, see “ Causes for timeout errors ” on page 303 for more information.

The response JSON of each health check request displays a status field in the response.

The overall status of API Gateway cluster can be *green*, *yellow*, and *red*.

Status	Description
green	Indicates that the cluster is in a healthy state.
yellow	Indicates that API Gateway does not have adequate resources to run.
red	Indicates the cluster failure and an outage.

The overall status of API Gateway cluster is assessed based on the API Data Store status, API Gateway resource status, and the cluster status within nodes.

API Data Store status

Status	Description
green	Indicates that API Data Store is in a healthy state. When the status of API Data Store signals <i>green</i> or <i>yellow</i> , the overall status of API Gateway is <i>green</i> .
red	Indicates cluster failure and an outage. When the status of API Data Store signals <i>red</i> , the overall status of API Gateway is <i>red</i> .
yellow	Indicates a node failure in the cluster. However, the cluster is still functioning and operational.

API Gateway resource status

Status	Description
green	Indicates that API Gateway resource types like memory, disk space, and service threads are available to run.
yellow	Indicates that API Gateway does not have adequate resources to run. When the API Gateway resource status is <i>yellow</i> , the overall status of API Gateway is <i>yellow</i> .

Cluster status within nodes

Status	Description
green	Indicates that cluster is in a healthy state. The cluster status is <i>green</i> only when Terracotta Server Array is up and running. When the status of the cluster signals <i>green</i> , the overall status of API Gateway is <i>green</i> .
red	Indicates cluster failure and an outage. When the status of the cluster signals <i>red</i> , the overall status of API Gateway is <i>red</i> .

A sample HTTP response is as follows:

```
{
  "status": "green",
  "elasticsearch": {
    "cluster_name": "api_gateway_cluster",
    "status": "green",
    "number_of_nodes": "3",
    "number_of_data_nodes": "3",
    "timed_out": "false",
    "active_shards": "200",
    "initializing_shards": "0",
    "unassigned_shards": "0",
    "task_max_waiting_in_queue_millis": "0",
    "node": "localhost:9240",
    "response_time_ms": "4"
  },
  "is": {
    "status": "green",
    "diskspace": {
      "status": "up",
      "free": "14206386176",
      "inuse": "17994313728",
      "threshold": "3220069990",
      "total": "32200699904"
    },
    "memory": {
      "status": "up",
      "freemem": "420766624",
      "maxmem": "2147483648",
      "threshold": "161061273",
      "totalmem": "1610612736"
    },
    "servicethread": {
      "status": "up",
      "avail": "397",
      "inuse": "3",
      "max": "400",
      "threshold": "40"
    },
    "response_time_ms": "309"
  },
  "cluster": {
    "status": "green",
    "isClusterAware": "true",
    "nodes": "3",
    "response_time_ms": "518"
  },
  "runtime": {
    "status": "green",
    "start_mode": "up"
  }
}
```

The overall cluster status of API Gateway is green since all components work as expected.

Administration Service Health Probe at Cluster Level

To check the availability and health status of the API Gateway administration service (UI, Dashboards) at the cluster level, use the following REST endpoint:

GET /rest/apigateway/health/admin

The following table shows the response code and the description.

Response	Description
200 OK	Administration service health check is successful.
500 Internal server error	Denotes a problem. The response JSON indicates the problem.
timeout or no response as the request did not reach the probe	Several factors can contribute to the delay when you initiate the Administration Service Health Probe, which may result in the timeout errors. To know the reasons for timeout errors, see "Causes for timeout errors" on page 303 for more information.

The overall Administration Service Health Probe status can be green or red based on the API Gateway administration service's status and Kibana status.

Kibana status

Status	Description
green	Indicates that Kibana's port is accessible. When the status signals green, the overall status of Administration Service Health Probe is green.
red	Indicates that either Kibana's port is inaccessible or Kibana's communication with API Data Store is not established. When the status signals red, the overall status of Administration Service Health Probe is red.

API Gateway administration service status

Status	Description
green	Indicates that API Gateway administration service is available. When the status signals green, the overall status of Administration Service Health Probe is green.
red	Indicates that API Gateway administration service is not available. When the status signals red, the overall status of Administration Service Health Probe is red.

A sample HTTP response is as follows:

```
{
```

```

    "status": "green",
    "ui": {
        "status": "green",
        "response_time_ms": "40"
    },
    "kibana": {
        "status": {
            "overall": {
                "state": "green",
                "nickname": "Looking good",
                "icon": "success",
                "uiColor": "secondary"
            }
        },
        "response_time_ms": "36"
    }
}

```

The overall status is green since API Gateway administration service and Kibana is in a healthy state.

Troubleshooting: Cluster-level Monitoring

During cluster-level monitoring, when you encounter a problem, check if the server nodes are up and running. Using a test client such as Postman, hit a particular node rather than the external load balancer endpoint. If it works, then it means that the load balancer is experiencing issues or there can be connectivity issues between the load balancer and the nodes.

Node-level Monitoring

The node-level monitoring ensures the ability of a particular instance of API Gateway components to serve the functionality (API requests) in the node. The node health is monitored by setting up probes per instance of each API Gateway component in the node.

The node-health monitoring enables you to check the following during the application start-up:

- If the bootstrap of the node is completed.
- If the node joined the cluster.
- If the node is ready to serve the requests.
- If the node is unhealthy, it identifies the problem and checks if the server component requires a restart.

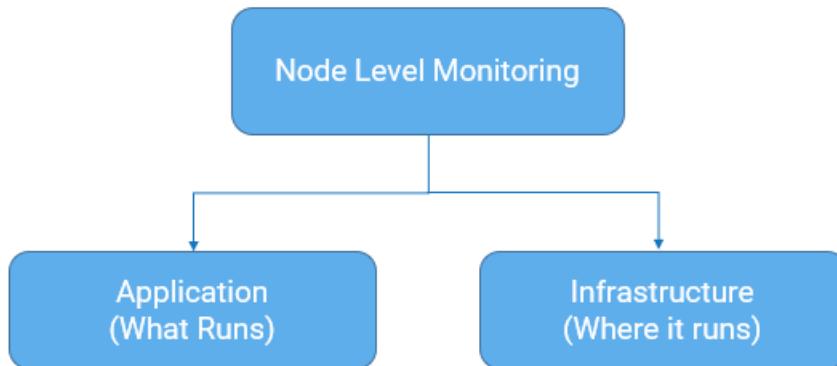
The node-health monitoring enables you to check the following when the application in the node runs:

- If the application is available.
- If the application is under load.
- If the application is performing well.

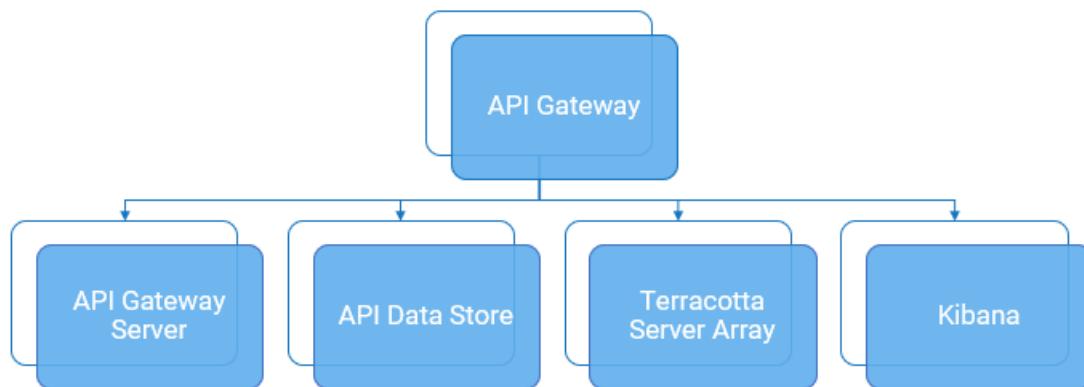
- If there are any errors.
- If the application is unhealthy, it identifies the problem and checks if the server component requires a restart.

You can monitor the node-health at:

- Application level
- Infrastructure level



Application level



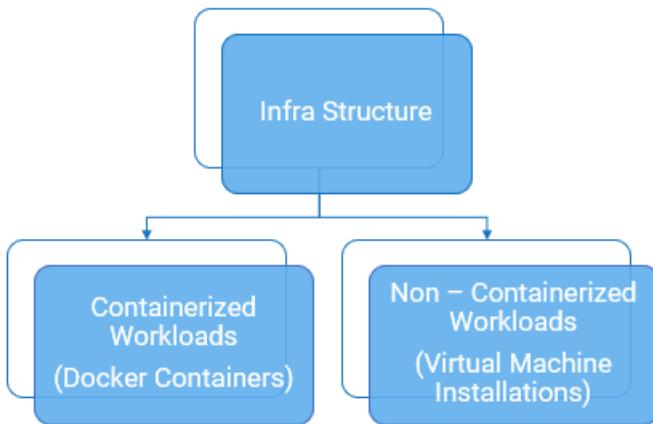
At the application level, you can monitor the state, that is cluster status and console access of the application along with the resource utilization of the application of the following API Gateway components:

- API Gateway Server
- API Data Store
- Terracotta Server Array
- Kibana

Note:

- It is not required that all the components are hosted on the same node. Few components such as API Data Store, Terracotta Server Array and so on can be hosted on dedicated nodes.
- You can check both the cluster status and console access only for API Gateway server and only the cluster status for API Data Store and Terracotta Server Array.

Infrastructure level



Note:

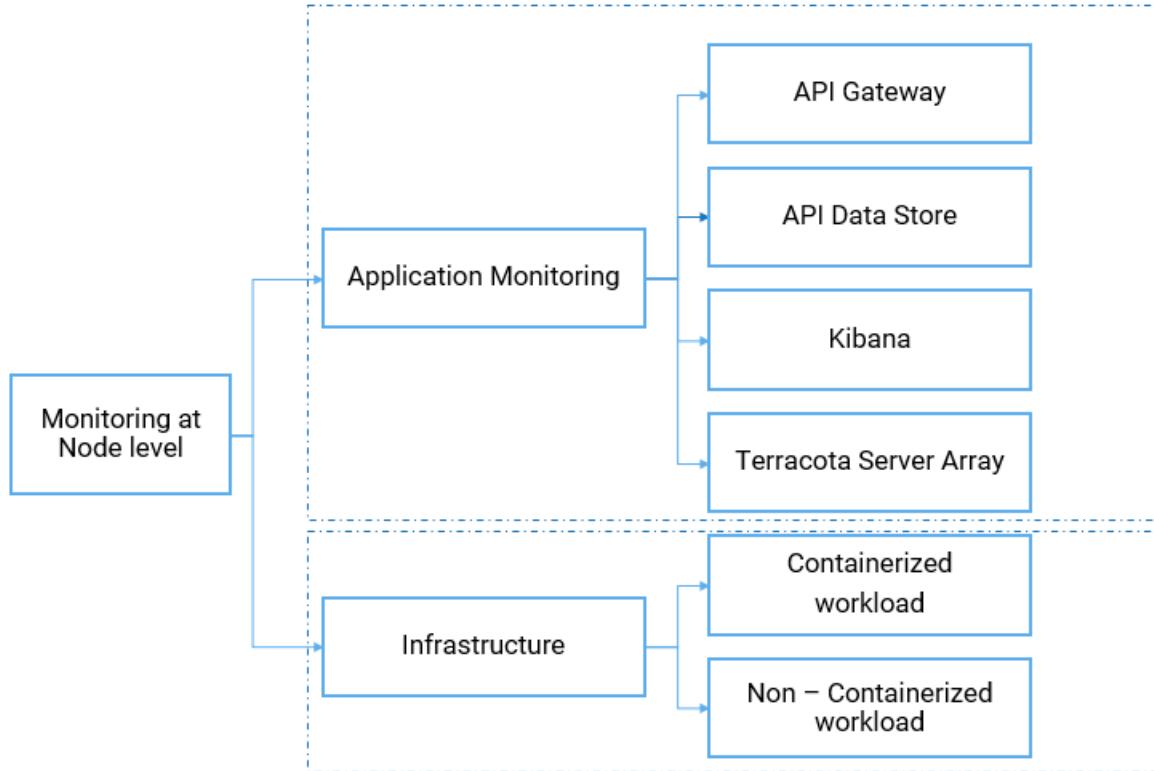
Startup probe is available in the recent versions of Kubernetes and is a recommended alternative to the Runtime Service Health Probe during the bootstrapping of the application. The endpoint is the same as that of the Runtime Service Health Probe but the Startup probe in Kubernetes itself is configured with slightly different characteristics like *initial delay* for the first check, *failure threshold* and so on.

At the Infrastructure level, you can monitor both containerized workloads (Docker containers) and non-containerized workloads (Virtual machine installations).

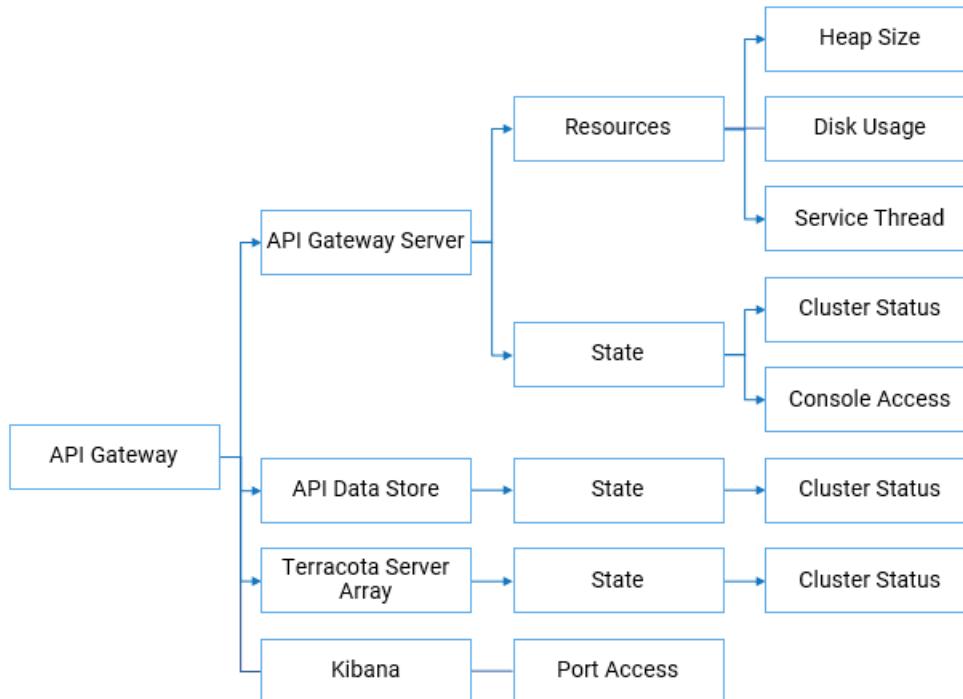
Note:

Most health checks are infrastructure-agnostic and can be used on both the types of infrastructure.

The overall node-level monitoring is explained in the following diagram.



The node-level monitoring in detail with the state and resource information of each component is explained in the following diagram.



How do I monitor the node health of API Gateway?

You can set up the Readiness probe, Runtime Service Health Probe, and Administration Service Health Probe to monitor the overall health status of a particular node of API Gateway.

Requirement	Impact	Solution
For a running instance of an API Gateway or its components, is there an endpoint that returns <i>yes</i> or <i>no</i> about its readiness for serving the incoming API requests?	For Load Balancer or Kubernetes service to know if the node is healthy for it to be added to the load distribution pool for routing the incoming requests.	Use Readiness Probe .
For a running instance of an API Gateway or its components, is there an endpoint that returns the metrics about the application and the resource utilization at the level of the container or the virtual machine?	To monitor capacity and performance.	Set up Metrics collection .
For a running instance of API Gateway or its components, is there an endpoint that indicates the cluster health and the status of the runtime components?	To know the details about where the fault lies when there is a cluster failure, so that the node can be health and the status of the runtime manually restarted. Kubernetes does it automatically.	Use Runtime Service Health Probe and Administration Service Health Probe .

How do the probes help in node-level monitoring?

	Readiness Probe	Runtime Service Health Probe	Administration Service Health Probe
What is it?	Indicates if the traffic-serving port of a particular API Gateway node is ready to accept requests.	Reports on the overall cluster health and indicates if the components of a particular API Gateway node are in an operational state.	Indicates if the API Gateway administrator consoles are available and accessible on a particular API Gateway node.
When is it used?	To check if a particular node of API Gateway can be added to the load balancer.	To check if a node requires a restart or if there is a problem that needs immediate attention.	
How does it help in containerized workloads?	If the Readiness Probe fails, the pod is removed from the endpoints of the service. Hence, traffic is not served until it is ready. The pod is not shutdown or restarted.		
How does it help in	If the Readiness Probe fails, the node can be removed from the endpoints of the load balancer. Hence, traffic is not served until it becomes ready.		

Readiness Probe	Runtime Service Health Probe	Administration Service Health Probe
non-containerized setup?		

Monitoring

The sections in the following diagram are explained in detail as part of monitoring API Gateway components.



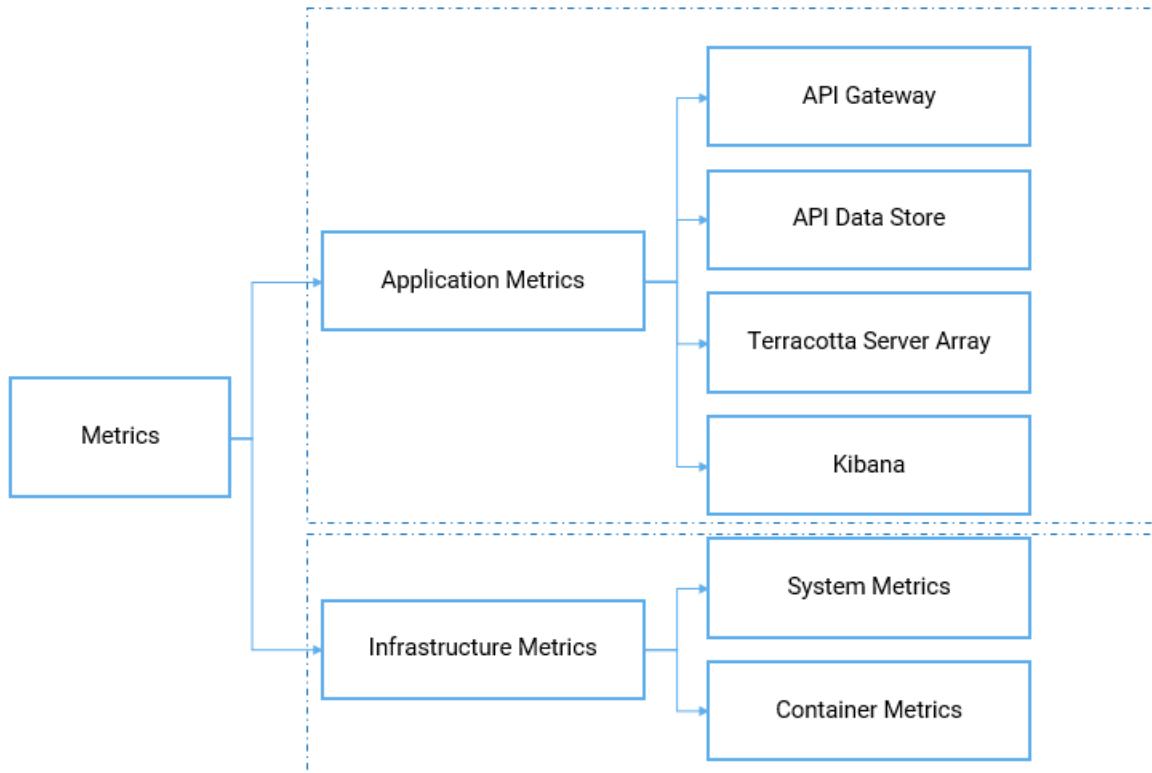
Metrics

You can monitor the resources of API Gateway server, API Data Store, Terracotta, and Kibana with various metrics. A metric is a measurement related to health, capacity, or performance of a given resource such as CPU, Disk, and so on. Metrics are classified into two types:

- Application metrics. This refers to the metrics related to your API Gateway component.
- Infrastructure metrics. This refers to the infrastructure where the application runs. This is further classified into System metrics and Container Metrics. At the infrastructure level, you can monitor both containerized workloads (Docker containers) with container metrics and non-containerized workloads (Virtual machine installations) with system metrics.

Software AG offers you the capability to monitor both application metrics and infrastructure metrics. You can gain insight into the consumption and availability of resources, which in turn helps you identify and analyze the root cause and debug the issues quickly. This helps you to determine when to scale up the applications. This improves the overall business continuity and reduce the application downtime.

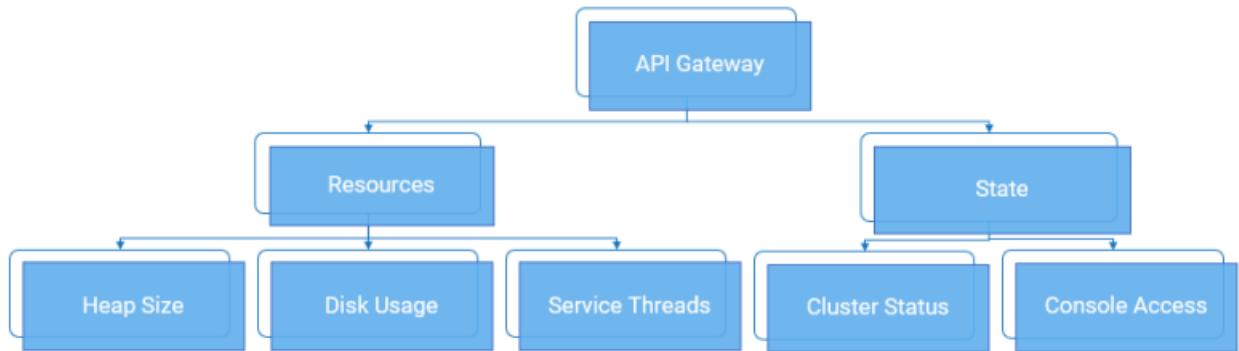
The following diagram explains the different types of metrics.



The following sections describe how you can check the availability and monitor each component in the node individually.

Monitoring API Gateway

You can monitor API Gateway by monitoring its resources and its state. You monitor the resources using the Prometheus metrics and the state by monitoring the health of API Gateway and its components



As part of application monitoring, you can monitor the state, that is the cluster status and console access of API Gateway along with the resources.

Monitoring API Gateway through Prometheus Metrics

How do I collect Prometheus metrics?

To check the usage of the application and system parameters, use the following metrics endpoint: GET /metrics. When the endpoint is called, API Gateway gathers metrics and returns the data in the Prometheus format.

Note:

Prometheus is a non- Software AG dashboarding tool that helps in trend analysis. For more information, see <https://prometheus.io/>.

Prometheus metrics are exposed through the following endpoint.

```
[http|https]://host:port/metrics
```

The metrics endpoint by default is available on the following ports:

- Default primary port (http). 5555
- Default secure port (https). 5543
- Default diagnostic port (debug port). 9999

A sample for the metrics endpoint is as follows:

```
http://server:5555/metrics
```

The metrics you monitor have the following characteristics:

- The metrics are maintained separately for each cluster node
- The metrics are not persisted. They are reinitialized on API Gateway restart.

Authentication for the metrics endpoint

- By default, the authentication is disabled when running API Gateway as Docker container.
- For on-premise installations, the following environment variable can be set to switch off the authentication for the metrics endpoint:

```
SAG_IS_METRICS_ENDPOINT_ACL=Anonymous
```

The endpoint also exposes the Integration Server Prometheus metrics. For more details on the Integration Server Prometheus metrics, see *Developing Microservices with webMethods Microservices Runtime*.

Exposing API Gateway Prometheus Metrics over a dedicated port

The metrics endpoint can be made available on a custom port. After creating the port, add the following service to the port's allow list:

```
wm.server.query:getPrometheusStats
```

Similarly, the metrics endpoint can be removed from the default ports (5555 or 5543 or 9999) by removing the service from the allow or deny lists.

Application Metrics

Monitor the following Prometheus metrics to analyze API Gateway health.

- Threads statistics
- Service errors
- Memory usage of JVM
- HTTP or HTTPS requests

Note:

The threshold values, configurations, and severities that are mentioned throughout this section are the guidelines that Software AG suggests for an optimal performance of API Gateway. You can modify these thresholds or define actions based on your operational requirements.

For details about how to generate thread dump, heap dump and log locations, see [“Troubleshooting: Monitoring API Gateway” on page 299](#).

If the metrics return an exceeded threshold value, consider the severity as mentioned and perform the possible actions that Software AG recommends to identify and debug the problem and contact Software AG for further support.

Monitor the Threads statistics

Metric	Description
<code>sag_is_service_threads</code>	<p>Checks the percentage of total number of threads used for service execution where the threads are obtained from the server thread pool.</p> <p>If the threads usage is above the recommended threshold value for more than 15 minutes, consider the severity as mentioned:</p> <ul style="list-style-type: none"> ■ Above 80% threshold, Severity: <i>ERROR</i> ■ Above 90% threshold, Severity: <i>CRITICAL</i> <p>The steps to identify the causes of higher threads usage are as follows:</p> <ol style="list-style-type: none"> 1. Identify the process that consumes the highest number of threads. 2. Generate the thread dump. 3. Analyze thread dump to identify the thread locks. 4. Analyze the logs of all API Gateway instances in the node.

Monitor the Service errors

Metric	Description
sag_is_number_service_errors	<p>Checks the number of services that results in errors or exceptions.</p> <p>If service errors are encountered, consider the severity as <i>ERROR</i>.</p> <p>The steps to identify the causes of service errors are as follows:</p> <ol style="list-style-type: none"> 1. Check the cluster status of API Gateway using the following REST endpoint: GET /rest/apigateway/health/engine to know if API Gateway is healthy and is in a cluster mode. 2. Check the server logs for any exception from <code>SAGInstallDirectory\IntegrationServer\instances\instance_name\logs\server.log</code>.

Monitor the Memory usage of JVM

Metric	Description
sag_is_used_memory_bytes	<p>Checks the percentage of total used memory of JVM.</p> <p>If the memory usage is above the recommended threshold value for more than 15 minutes, consider the severity as mentioned:</p> <ul style="list-style-type: none"> ■ Above 80% threshold, Severity: <i>ERROR</i> ■ Above 90% threshold, Severity: <i>CRITICAL</i> <p>The steps to identify the causes of higher memory usage of JVM are as follows:</p> <ol style="list-style-type: none"> 1. Check the cluster status of API Gateway using the following REST endpoint: GET /rest/apigateway/health/engine to know if API Gateway is healthy and is in a cluster mode. 2. Generate the heap dump. 3. Analyze the logs of all the API Gateway instances. 4. Identify the server that has an issue and restart the server if required. 5. Perform the following actions after restarting the server: <ul style="list-style-type: none"> a. Check for the readiness of API Gateway. b. Check the cluster status of API Gateway using the following REST endpoint: GET /rest/apigateway/health/engine to know if API Gateway is healthy and is in a cluster mode. c. Check the resource availability of all the required system resources like memory, heap, and disk. d. Check API Data Store connectivity with API Gateway server.

Metric	Description
	e. Check the Terracotta client logs for errors in Terracotta communication for a cluster set up.

Monitor the HTTP or HTTPS requests

Metric	Description
<code>sag_is_http_requests</code>	<p>Checks the percentage of total number of HTTP or HTTPS requests since the last statistics poll.</p> <p>The statistics poll interval is controlled by the <code>watt.server.stats.pollTime</code> server configuration parameter and the default interval is 60 seconds.</p> <p>If the total number of HTTP or HTTPS requests since the last statistics poll is above the threshold limit that is based on the Throughput Per Second (TPS) value, consider the severity as <i>ERROR</i>.</p>

API Operational Metrics

API Gateway provides metric statistics for API calls, error, and error rates. The API-level monitoring measures the availability of the deployed APIs. For example, error rates and performance (latency). You can use these metrics to measure the service and business availability.

The key metrics monitored are:

- Error rates
 - API transaction error rate per API and the aggregated value
 - API execution error rate per API and the aggregated value
 - Backend API errors per API and the aggregated value
 - Errors arising from the inter component interactions (such as API Gateway to Elasticsearch)
- Performance (latency)
 - API performance per API
 - API Gateway performance and Backend API performance
 - Aggregated latency introduced by API Gateway

The API-level metrics you monitor have the following characteristics:

- The metrics continue to exist when an API is deactivated or activated.
- The metrics for a deleted API are no longer reported.

Note:

- The metric count starts from zero when the server starts.
- The API invocations are counted per node and not for the complete cluster.
- The API invocations are counted only within an API Gateway instance.
- After scraping, that is after sending the `/metrics` request, all Gauge values are reset.

The following tables lists the API-level metrics and the server-level metrics and their corresponding Prometheus metric types and labels that are associated with a metric that you use for monitoring.

API-level Metrics

The API-level metrics monitor the API transaction error rates and API performance. This is monitored based on the metric type you use like API name, API version, the response code or the tenant that receives the API invocation request.

Prometheus metric name	Description	Label
sag_apigw_api_exec_error_count	<p>This is an error rate related metric.</p> <p>Counts the number of API invocations that fail due to an unexpected error. This does not include policy violations and backend failures.</p>	<ul style="list-style-type: none"> ■ api_name ■ api_version ■ env
Prometheus metric type: Counter		
sag_apigw_api_backend_error_count	<p>This is an error rate related metric. Counts the number of backend service failures during API invocations with a response code greater or equal to 300 and classifies the failures by its response code with the label 3xx, 4xx, 5xx or connect. The connect reason means that the backend service cannot be reached, for example, due to network outages.</p>	<ul style="list-style-type: none"> ■ api_name ■ api_version ■ code (can have values 3xx, 4xx, 5xx, connect) ■ env
Prometheus metric type: Counter		
sag_apigw_api_tx_error_count	<p>This is an error rate related metric. Counts the number of API invocations with a response code greater or equal to 300 and classifies the invocations by its response code with the label 3xx, 4xx or 5xx.</p>	<ul style="list-style-type: none"> ■ api_name ■ api_version ■ code (can have values 3xx, 4xx, 5xx) ■ env ■ error (can have values internal, policy, backend)
Prometheus metric type: Counter		

Prometheus metric name	Description	Label
sag_apigw_api_avg_latency	<p>This is a latency related metric.</p> <p>Measures the average time spent by the API invocations, which does not include the time spent in backend services, classified by its response code with the label 2xx, 3xx, 4xx or 5xx.</p> <p>Prometheus metric type: Gauge</p>	<ul style="list-style-type: none"> ■ api_name ■ api_version ■ code (can have values 2xx, 3xx, 4xx, 5xx) ■ env
sag_apigw_api_avg_backend_response_time	<p>This is a latency related metric.</p> <p>Measures the average time spent by the backend services while performing an API invocation request, classified by its response code with the label 2xx, 3xx, 4xx, 5xx or connect.</p> <p>Prometheus metric type: Gauge</p>	<ul style="list-style-type: none"> ■ api_name ■ api_version ■ code (can have values 2xx, 3xx, 4xx, 5xx, connect) ■ env
sag_apigw_api_avg_response_time	<p>This is a latency related metric.</p> <p>Measures the average time spent by the API invocations, classified by its response code with the label 2xx, 3xx, 4xx or 5xx. This includes the time spent in API Gateway and by the backend service.</p> <p>Prometheus metric type: Gauge</p>	<ul style="list-style-type: none"> ■ api_name ■ api_version ■ code (can have values 2xx, 3xx, 4xx, 5xx) ■ env
sag_apigw_apicalls_total	<p>The total number of API invocations per HTTP response code.</p> <p>Prometheus metric type: Counter</p>	<ul style="list-style-type: none"> ■ code ■ env

Server-level Metrics

The server-level metrics monitor the API invocation errors due to inter-component connectivity. This is monitored based on the metric type you use such as component and tenant that receives the API invocation request.

Prometheus metric name	Description	Label
sag_apigw_tx_error_count	<p>This is an error rate related metric.</p> <p>Counts the number of API invocations with a response code</p>	<ul style="list-style-type: none"> ■ code (can have values 3xx, 4xx, 5xx)

Prometheus metric name	Description	Label
	<p>greater or equal to 300 and classifies the invocations by its response code with the label 3xx, 4xx or 5xx.</p> <p>Prometheus metric type: Counter</p>	<ul style="list-style-type: none"> ■ env ■ error (can have values internal, policy, backend)
sag_apigw_backend_error_count	<p>This is an error rate related metric. Counts the number of backend service failures during all API invocations with a response code greater or equal to 300 and classifies the failures by its response code with the label 3xx, 4xx, 5xx or connect. The connect reason means that the backend service cannot be reached, for example, due to network outages.</p> <p>Prometheus metric type: Counter</p>	<ul style="list-style-type: none"> ■ code (can have values 3xx, 4xx, 5xx, connect) ■ env
sag_apigw_component_error_count	<p>This is an error rate related metric. Counts the number of exceptions that occur when interacting with components or destinations.</p> <p>Prometheus metric type: Counter</p>	<ul style="list-style-type: none"> ■ component ■ env
sag_apigw_exec_error_count	<p>This is an error rate related metric. Counts the number of API invocations that fail due to an unexpected error (for example NPE) and provides an aggregated number.</p> <p>Prometheus metric type: Counter</p>	<ul style="list-style-type: none"> ■ env
sag_apigw_avg_response_time	<p>This is a latency related metric. Measures the average time spent by all API invocations, classified by its response code with the label 2xx, 3xx, 4xx or 5xx. This includes the time spent in API Gateway and by the backend services</p> <p>Prometheus metric type: Gauge</p>	<ul style="list-style-type: none"> ■ code (can have values 2xx, 3xx, 4xx, 5xx) ■ env

Prometheus metric name	Description	Label
sag_apigw_avg_latency	<p>This is a latency related metric.</p> <p>Measures the average time spent by all API invocations, which does not include the time spent in backend services, classified by its response code with the label 2xx, 3xx, 4xx or 5xx.</p>	<ul style="list-style-type: none"> ■ code (can have values 2xx, 3xx, 4xx, 5xx) ■ env

Prometheus metric type: Gauge

Prometheus labels

Prometheus label	Description
api_name	The name of the API for which the API invocation failed.
api_version	The version of the API for which the API invocation failed.
component	The name of the API Gateway component that cannot be reached.
code	The code label shows the HTTP response code for the API calls counted. For each HTTP response code that occurred during the lifetime of the API Gateway server, the metrics response contains a separate counter entry.
env	<p>The name of the customer environment.</p> <p>The value of the env label is taken from the pg.gateway.elasticsearch.tenantId property in the config.properties file located at SAGInstallDir/IntegrationServer/instances/instance_name/packages/WmAPIGateway/config/resources/elasticsearch.</p>
error	The type of error encountered, such as internal error, policy violation, or backend error.

Example: Sample Metrics

Here are a few example of sample metrics.

- Sample transaction error count metrics are as follows:

In this example there are no unexpected execution errors counted for the particular API.

```
# HELP sag_apigw_api_exec_error_count Number of unexpected execution errors
# TYPE sag_apigw_api_exec_error_count counter
sag_apigw_api_exec_error_count{api version="1.0",env="default",api
name="TestMetricsRest"} 0 1646997393690
```

In this example there are no unexpected execution errors counted for none of the executing APIs.

```
# HELP sag_apigw_exec_error_count Number of aggregated execution errors
# TYPE sag_apigw_exec_error_count counter
sag_apigw_exec_error_count{env="default"} 0 1646997393690
```

- Sample aggregated backend errors count metrics are as follows:

In this example there is 1 connection error counted for the backend service of the particular API.

```
# HELP sag_apigw_api_backend_error_count Number of backend errors
# TYPE sag_apigw_api_backend_error_count counter
sag_apigw_api_backend_error_count{code="connect",api version="1.0",env="default",api name="TestMetricsRest"} 1 1646997393690
```

In this example there is 1 connection error counted for backend services of all executing APIs.

```
# HELP sag_apigw_backend_error_count Number of aggregated backend errors
# TYPE sag_apigw_backend_error_count counter
sag_apigw_backend_error_count{code="connect",env="default"} 1 1646997393690
```

- Sample transaction error count metrics is as follows:

In this example there is 1 transaction error counted for a particular API. This is an ordinary policy error with the response code 4xx.

```
# HELP sag_apigw_api_tx_error_count Number of transaction errors
# TYPE sag_apigw_api_tx_error_count counter
sag_apigw_api_tx_error_count{code="4xx",api version="1.0",env="default",error="policy",api name="TestMetricsRest"} 1 1646997393690
```

In this example there is 1 transaction error counted for a particular API. This is a backend service error with the response code 5xx.

```
# HELP sag_apigw_api_tx_error_count Number of transaction errors
# TYPE sag_apigw_api_tx_error_count counter
sag_apigw_api_tx_error_count{code="5xx",api version="1.0",env="default",error="backend",api name="TestMetricsRest"} 1 1646997393690
```

- Sample API Gateway latency metrics is as follows:

In this example 1 millisecond is the time spent in API Gateway performing an API invocation request for the incoming API requests measured with response code 4xx

```
# HELP sag_apigw_api_avg_latency Average API Gateway latency
# TYPE sag_apigw_api_avg_latency gauge
sag_apigw_api_avg_latency{code="4xx",api version="1.0",env="default",api name="TestMetricsRest"} 1 1646997393690
```

In this example 2328 milliseconds is the time spent in API Gateway performing an API invocation request for the incoming API requests measured with response code 5xx

```
# HELP sag_apigw_api_avg_latency Average API Gateway latency
```

```
# TYPE sag_apigw_api_avg_latency gauge
sag_apigw_api_avg_latency{code="5xx",api version="1.0",env="default",api name="TestMetricsRest"} 2328 1646997393690
```

- Sample backend response time metrics is as follows. In this example 2315 milliseconds is the average time spent by the backend services while performing an API invocation request; measured with the label code=connect. The connect reason means that the backend service cannot be reached, for example, due to network outages.

```
# HELP sag_apigw_api_avg_backend_response_time Average backend service duration
# TYPE sag_apigw_api_avg_backend_response_time gauge
sag_apigw_api_avg_backend_response_time{code="connect",api version="1.0",env="default",api name="TestMetricsRest"} 2315 1646997393690
```

- Sample average response time metrics is as follows:

In this example 1 millisecond is the average response time spent for the incoming API requests measured with response code 4xx

```
# HELP sag_apigw_api_avg_response_time Average request duration
# TYPE sag_apigw_api_avg_response_time gauge
sag_apigw_api_avg_response_time{code="4xx",api version="1.0",env="default",api name="TestMetricsRest"} 1 1646997393690
```

In this example 2328 milliseconds is the average response time spent for the incoming API requests measured with response code 5xx

```
# HELP sag_apigw_api_avg_response_time Average request duration
# TYPE sag_apigw_api_avg_response_time gauge
sag_apigw_api_avg_response_time{code="5xx",api version="1.0",env="default",api name="TestMetricsRest"} 2328 1646997393690
```

- Sample apicall metrics is as follows. In this example 32 API calls are measured with the HTTP response code 200.

```
# HELP sag_apigw_apicalls_total Total number of API invocations per response code
# TYPE sag_apigw_apicalls_total counter
sag_apigw_apicalls_total {code="200" ,env="default"} 32 1635169035001
```

Monitoring API Gateway Health

How do I monitor the health of API Gateway?



Prerequisites:

- You must have a valid API Gateway user credential for using the Readiness Probe, Runtime Service Health Probe, and Administration Service Health Probe.
- All the node level probes must be setup to target the local instance, typically, localhost.

- Software AG recommends to set up a dedicated port for monitoring with an appropriate private thread pool.

Readiness Probe at Node-Level

To monitor the readiness of API Gateway, that is to check if the traffic-serving port of a particular API Gateway node is ready to accept requests, use the following REST endpoint:

`GET /rest/apigateway/health`

The following table shows the response code and the description.

Response	Description
200 OK	Readiness check is successful. Readiness probe continues to reply OK if API Gateway remains in an operational state to serve the requests.
500 Internal server error	Readiness check failed and denotes a problem.
timeout or no response as the request did not reach the probe	Several factors can contribute to the delay when the Readiness Probe initiates, which may result in the timeout errors. To know the reasons for timeout errors, see “ Causes for timeout errors ” on page 303 for more information.

Note:

As this is a Readiness Probe and only the response status code is essential, by design, JSON payload is not returned in the response for both success and failure scenarios.

Runtime Service Health Probe

To monitor the runtime service health of API Gateway, that is to check the overall cluster health and to identify if the components of a particular API Gateway node are in an operational state, use the following REST endpoint:

`GET /rest/apigateway/health/engine`

The following table shows the response code and the description.

Response	Description
200 OK	Runtime service health check is successful.
500 Internal server error	Runtime service health check failed and denotes a problem. The response JSON indicates the problem.
timeout or no response as the request did not reach the probe	Several factors can contribute to the delay when the Runtime Service Health Probe initiates, which may result in the timeout errors. To know the reasons for timeout errors, see “ Causes for timeout errors ” on page 303 for more information.

The response JSON of each health check request displays a status field in the response.

The overall status of API Gateway node can be green ,yellow, or red.

Status	Description
green	Indicates that the cluster within the node is in a healthy state.
yellow	Indicates that API Gateway does not have adequate resources to run.
red	Indicates the cluster failure in the node and an outage.

The overall status of API Gateway node is assessed based on the API Data Store status, API Gateway resource status, and the Terracotta server status.

API Data Store status

Status	Description
green	Indicates that API Data Store is in a healthy state. When the status of API Data Store signals green or yellow, the overall status of API Gateway is green.
red	Indicates that API Data Store is not in a healthy state. When the status of API Data Store signals red, the overall status of API Gateway is red.
yellow	Indicates a node failure in the cluster. However, the cluster is still functioning and operational.

API Gateway resource status

Status	Description
green	Indicates that API Gateway resource types like memory, disk space, and service threads are available to run.
yellow	Indicates that API Gateway does not have adequate resources to run. When the API Gateway resource status is yellow, the overall status of API Gateway is yellow.

Terracotta Server Array status

Status	Description
green	Indicates that Terracotta server is in a healthy state. When the status of Terracotta server signals green, the overall status of API Gateway is green.
red	Indicates that Terracotta server is not in a healthy state. When the status of Terracotta server signals red, the overall status of API Gateway is red.

A sample HTTP response is as follows:

```
{  
    "status": "green",  
    "elasticsearch": {  
        "cluster_name": "SAG_EventDataStore",  
        "status": "yellow",  
        "number_of_nodes": "1",  
        "number_of_data_nodes": "1",  
        "timed_out": "false",  
        "active_shards": "95",  
        "initializing_shards": "0",  
        "unassigned_shards": "92",  
        "task_max_waiting_in_queue_millis": "0",  
        "port_9240": "ok",  
        "response_time_ms": "526"  
    },  
    "is": {  
        "status": "green",  
        "diskspace": {  
            "status": "up",  
            "free": "908510568448",  
            "inuse": "104799719424",  
            "threshold": "101331028787",  
            "total": "1013310287872"  
        },  
        "memory": {  
            "status": "up",  
            "freemem": "425073672",  
            "maxmem": "954728448",  
            "threshold": "92222259",  
            "totalmem": "922222592"  
        },  
        "servicethread": {  
            "status": "up",  
            "avail": "72",  
            "inuse": "3",  
            "max": "75",  
            "threshold": "7"  
        },  
        "response_time_ms": "258"  
    },  
    "terracotta": {  
        "status": "green",  
        "nodes": "1",  
        "healthy_nodes": "1",  
        "response_time_ms": "22"  
    }  
}
```

The overall engine status is green since all components work as expected.

Administration Service Health Probe

To check the availability and health status of the API Gateway administration service (UI, Dashboards) on a particular API Gateway node, use the following rest endpoint:

GET /rest/apigateway/health/admin

The following table shows the response code and the description.

Response	Description
200 OK	Administration service health check is successful.
500 Internal server error	Denotes a problem. The response JSON indicates the problem.
timeout or no response as the request did not reach the probe	Several factors can contribute to the delay when you initiate the Administration Service Health Probe, which may result in the timeout errors. To know the reasons for timeout errors, see "Causes for timeout errors" on page 303 for more information.

The overall Administration Service Health Probe status can be `green` or `red` based on the API Gateway administration service's status and Kibana's status.

Kibana status

Status	Description
<code>green</code>	Indicates that Kibana's port is accessible. When the status signals <code>green</code> , the overall status of Administration Service Health Probe is <code>green</code> .
<code>red</code>	Indicates that either Kibana's port is inaccessible or Kibana's communication with API Data Store is not established. When the status signals <code>red</code> , the overall status of Administration Service Health Probe is <code>red</code> .

API Gateway administration service status

Status	Description
<code>green</code>	Indicates that API Gateway administration service is available. When the status signals <code>green</code> , the overall status of Administration Service Health Probe is <code>green</code> .
<code>red</code>	Indicates that API Gateway administration service is not available. When the status signals <code>red</code> , the overall status of Administration Service Health Probe is <code>red</code> .

A sample HTTP response is as follows:

```
{
  "status": "green",
  "ui": {
    "status": "green",
    "response_time_ms": "40"
  },
  "kibana": {
    "status": {
      "port": "green"
    }
  }
}
```

```

        "overall": {
            "state": "green",
            "nickname": "Looking good",
            "icon": "success",
            "uiColor": "secondary"
        }
    },
    "response_time_ms": "36"
}
}

```

The overall status is green since API Gateway administration service and Kibana is in a healthy state.

System Metrics

Monitor the following system metrics to analyze API Gateway health:

- CPU usage
- Disk usage
- Memory usage

Monitor the CPU usage

Metric	Description
sag_is_server_proc_sys_percent	Checks the percentage of CPU used by the Operating System.
sag_is_server_proc_cpu_percent	<p>Checks the percentage of CPU used by the Integration Server JVM.</p> <p>If the CPU usage of both the metrics is above the recommended threshold value for more than 15 minutes, consider the severity as mentioned:</p> <ul style="list-style-type: none"> ■ Above 80% threshold, Severity: <i>ERROR</i> ■ Above 90% threshold, Severity: <i>CRITICAL</i> <p>The steps to identify the causes of higher CPU usage is as follows:</p> <ol style="list-style-type: none"> 1. Identify the process that consumes the highest CPU. 2. Generate the thread dump. 3. Analyze thread dump to identify the thread locks. 4. Analyze the logs of all the API Gateway instances in the node. 5. If CPU spikes happen due to excess load, Software AG recommends you to monitor the load and scale up and scale down API Gateway if required. For more details about scaling, see ""Scaling" on page 779.

Monitor the Disk usage

Metric	Description
<code>sag_is_server_total_disk_mbytes</code>	Checks the percentage of total available disk space in megabytes.
<code>sag_is_server_used_disk_mbytes</code>	<p>Checks the percentage of used disk space in megabytes.</p> <p>If the disk usage of both the metrics is above the recommended threshold value, consider the severity as mentioned:</p> <ul style="list-style-type: none"> ■ Above 80% threshold, Severity: <i>ERROR</i> ■ Above 90% threshold, Severity: <i>CRITICAL</i> <p>The steps to identify the causes of higher disk usage are as follows:</p> <ol style="list-style-type: none"> 1. The events archived in API Gateway are stored in the temp directory in the following location: <code>SAGInstallDirectory\profiles\IS_instance_name\workspace\temp</code>. Check the size of the temp directory and clean up the space to reduce the disk usage. 2. Check if the log rotation works as configured for the following file types: server, audit, error, session, wrapper, osgi, and API Gateway and check the size of the log files that consume more disk space to know if it is greater than the configured values. 3. Purge the events periodically to clean up the disk space for optimal performance of API Gateway. <p>For more details about Purging, see “Archive and Purge using API” on page 103.</p>

Monitor the Memory usage

Metric	Description
<code>sag_is_server_total_memory_mbytes</code>	Checks the percentage of total amount of physical memory available in megabytes.
<code>sag_is_server_used_memory_mbytes</code>	<p>Checks the percentage of total amount of physical memory used in megabytes.</p> <p>If the memory usage of both the metrics is above the recommended threshold value for more than 15 minutes, consider the severity as mentioned:</p> <ul style="list-style-type: none"> ■ Above 80% threshold, Severity: <i>ERROR</i> ■ Above 90% threshold, Severity: <i>CRITICAL</i> <p>The steps to identify the causes of higher memory usage is as follows:</p>

Metric	Description
	<ol style="list-style-type: none"> 1. Identify the process that consumes more memory. 2. Check the cluster status of API Gateway using the following REST endpoint: GET /rest/apigateway/health/engine to know if API Gateway is healthy and responding. 3. Generate the heap dump. 4. Analyze the logs of all the API Gateway instances and identify the file that consumes more memory. 5. Identify the server that has an issue and restart the server if required. 6. Perform the following actions after restarting the server: <ol style="list-style-type: none"> a. Check for the readiness of API Gateway. b. Check the cluster status of API Gateway using the following REST endpoint: GET /rest/apigateway/health/engine to know if API Gateway is healthy and is in a cluster mode. c. Check the resource availability of all the required system resources like memory, heap, disk. d. Check the Terracotta client logs for errors in Terracotta communication for a cluster set-up.

For more details about the API Gateway metrics, see *Developing Microservices with webMethods Microservices Runtime*.

Log monitoring

In addition to the metrics, to monitor the logs regularly, perform the following steps:

1. Check for the availability of all logs frequently.
2. Check if the log rotation works as configured for all file types.
3. Check the size of the log file to know if it is greater than the configured values.

To monitor the logs in different levels, check the availability of logs in FATAL, ERROR or WARNING level.

Monitoring API Gateway through Dashboards

The analytics dashboard in 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 would want to monitor. API Gateway also provides 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.
- **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.

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 event 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.

These are the types of events generated in API Gateway:

- **Transactional event :** Provides a summary of each runtime transaction in the system. It is generated when a Log Invocation policy is included for the API. For example, if an API has the policy attached to it, then for every invoke the system generates a transaction event. API Gateway provides a system global policy, Transaction logging, which are pre-configured in the product. This policy is, by default, deactivated. The transaction logging policy has standard filters and a log invocation policy that logs request or response payloads to a specified destination.
- **Error event:** Provides details of an error that occurred during an API invoke. This event is generated whenever there is an error in the system during a runtime service invocation. This is configured as part of destination configuration.
- **Monitoring event:** Provides a summary of event details along with the breach information when there is a threshold breach in any of the configured parameters. Monitoring could be done based on various parameters such as Total Request Count, Total Success Count, Response Time, and Availability. Monitoring can be done at the consumer application level too so that each consumer can be tracked individually. These events are generated when a Monitor Performance and Monitor SLA Policy is included for the API. In addition, the Traffic Optimization policy generates these events for every API invocation only when there is a breach in the parameters configured in the policy or once per alert interval based on the alert frequency configured in the policy.
- **Policy violation event:** Provides a summary of the policy violations that occurred in the system. When a policy attached to an API is violated, the system generates the policy violation event for alerting the provider. The Identity and Access, Authorization, and Schema Validation policies generate these events. This is configured as part of destination configuration. In addition, the Traffic Optimization policy generates these events for every API invocation only when there is a breach in the parameters configured in the policy.
- **Lifecycle event:** Provides a summary of the life cycle of the API Gateway instance. Whenever the instance is started or stopped, a life cycle notification is generated. This is configured as part of destination configuration.

- **Threat protection event:** Provides a summary of the threat protection filter and rule violations. When a filter or rule is violated, the system generates the threat protection violation event. For more details, see "[Configuring Alert Settings](#)" on page 632.

Note:

Internationalization is not supported in API Gateway dashboards.

API Gateway Dashboard

The dashboard displays the API Gateway-wide analytics based on the metrics monitored. Click



> **Analytics** to access API Gateway-wide analytics.

To filter the API Gateway-wide 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 Gateway-wide 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.

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.

In the **Applications** dashboard, you can filter the data using the filter for Applications in the specified time interval. The Applications drop-down list displays all the applications. When you select an application, its data is displayed. By default, the data displayed is for all the applications.

In the **Packages** dashboard, you can filter the data using the filter for Packages in the specified time interval. The Packages drop-down list displays all the packages. When you select a package, its data are displayed. By default, the data displayed is for all the packages.

In the **Audit logs** dashboard, you can filter the data using the filter for Audit logs in the specified time interval. It displays the data of all the auditable events.

In the **Cache statistics** dashboard, you can filter the cache statistics data depending on the Node name and Application type specified in the specified time interval.

In the **Application logs** dashboard, you can filter the application logs depending on the node, origin of log and so on in the specified time interval. Click **Download** to download the aggregated logs, the logs collected from different sources such as API Gateway server logs, API Gateway UI logs, API Data Store logs, dashboard logs and platform logs. The downloaded logs would contain the logs filtered as per the time interval filter applied.

Note:

- The Summary, Trends, and Application analytics are visible only in API Gateway Full Edition. Threat protection analytics is the only data visible in API Gateway Firewall Edition. The threat protection analytics information is visible only if you select the Alert destination as flow service in **Policies > Threat protection > Alert settings** section.

Category	Metric	Description
Summary	Overall events	Displays a pie chart that lists different events being monitored and each of these event categories is depicted with different colors.
	Application activity	Displays the application activity in API Gateway during the specified time.
	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.
Trends	Package performance	Displays a pie chart depicting package performance during the specified time. The different colors in the pie chart depict different packages this API belongs to.
	Events over time	Displays the trending of events generated by the APIs across API Gateway over time.
	API trend by success	Displays the trending of APIs based on their success rate in the performance metrics.
	API trend by failure	Displays the trending of APIs based on their failure rate in the performance metrics.
Applications	Overall error trends	Displays a graph depicting the performance of all the APIs in the system based on the error event generated. Each of these event categories is depicted with different colors.
	Events per application	Displays a pie chart that depicts the activity of events per application being monitored and each

Category	Metric	Description
		of these categories is depicted with different colors.
	Violations per application	Displays the number of violations per application based on the events generated such as monitoring, SLA violation, and policy violations.
	Activity rate of consumed packages	This bar chart displays the package that the selected application has consumed (when an application is chosen in the filter). Hover the cursor over the bar chart to see the number of invocations to the package using the specified application.
	Activity rate for consumed APIs	Displays the activity rate for all the APIs that are consumed by the application during the specified time.
	Runtime events	Displays the run time event details such as API Name, event type, date when the event was created, the agent on which the event was generated, description of the alert generated due to the event, the source of event, and the application that generated the event.
Packages	Package invocations	Displays the number of package invocations during the specified time.
	Trending subscription for package	Displays the trending subscriptions for the package based on the number of invocations. The different colors in the donut pie chart depict the trending behavior of the different applications in the package.
	Trending APIs in the package	Displays the number of invocations for an API for an application for the selected package over the specified time interval.
Threat protection	Threat protection filters	Displays the graphical representation of the events based on the filter violations during the specified time.
	Threat protection rules	Displays the graphical representation of the events based on the rule violations during the specified time.

Category	Metric	Description
	Threat protection events	Displays the threat protection event details such as Time, filter name, rule name, resource path, server host, and request time.
Audit logs	Time	Displays the time the event occurred.
	User	Displays the name of the user who caused the event.
	Status	Displays the current status of the transaction. The available values are: <ul style="list-style-type: none"> ■ SUCCESS ■ FAILURE
	Source machine	Displays the host name of the machine on which the event occurred.
	Object type	Displays the type of API Gateway object on which the event occurred. The available values are: <ul style="list-style-type: none"> ■ ACCESS_PROFILE_MANAGEMENT ■ ALIAS_MANAGEMENT ■ ANALYTICS_MANAGEMENT ■ API_MANAGEMENT ■ APPLICATION_MANAGEMENT ■ APPROVALS_MANAGEMENT ■ GROUPS_MANAGEMENT ■ PACKAGE_MANAGEMENT ■ PLAN_MANAGEMENT ■ PROMOTION_MANAGEMENT ■ POLICY_MANAGEMENT ■ USER_MANAGEMENT
	Object	Displays the UUID that uniquely identifies the object in the database.
	Message	Displays the success message or error message as a result of the event.

Category	Metric	Description
	Client IP address	Displays the IP address of the machine on which the event occurred.
	Action	<p>Displays the type of action for the event. The available values are:</p> <ul style="list-style-type: none"> ■ LOGIN ■ LOGOUT ■ CREATE ■ UPDATE ■ DELETE ■ ACTIVATE ■ DEACTIVATE
	Payload	Displays the content of data payload for the event.
Cache statistics:	Cache counts	Displays the hit, miss, and eviction count for API invocations across API Gateway.
	Cache usage statistics	Displays the cache usage size and the free size as a bar chart.
Application logs	Application logs saved search	<p>Displays a table that lists the cumulative logs collected across sources with details of each log that is collected in the time interval specified in the filter.</p> <p>These are the details displayed in the form of a table:</p> <ul style="list-style-type: none"> ■ Time. Specifies the date and time when the log was collected. ■ node. Specifies the node from which the log is generated. ■ fileType. Specifies the file type to which the logs belong. The following are the file types to which a log can belong: <ul style="list-style-type: none"> ■ APIGatewayServerLogs ■ APIGatewayUILogs ■ PlatformLogs

Category	Metric	Description
		<ul style="list-style-type: none"> ■ OSGILogs ■ WrapperLogs ■ InternalDataStoreLogs ■ DashboardLogs <p>■ logLevel. Specifies the log level.</p> <p>■ message. Displays the actual message for the event for which the log was saved.</p> <p>■ correlationId. Specifies the correlation id that applies to the API Gateway server logs with which you can identify a particular request.</p> <p>You can expand each entry to view details of the actual log in the tabular or a JSON format.</p> <p>In addition you can create a filter to display the logs based on their id, index, type, correlation id, and so on. This helps in analyzing the events effectively.</p>
Source vs log level		<p>Displays the log data per source per log level for the specified time interval.</p> <p>The data is displayed in the form of a pie chart.</p> <p>Hover the cursor over the piechart to view the following details.</p> <p>The inner section of the pie chart displays the number of logs collected per file type. The corresponding outer section displays the log levels for the logs collected for that file type.</p>
Log level tag cloud		<p>Displays the log levels available and shortcut filters to filter the logs by log levels.</p> <p>Click on one of the log levels. You now see the logs for the specified log level in the table under Application logs saved search and the distribution of the selected logs per sources that produced them in the pie chart under Source vs log level.</p>
Custom dashboards	Create your own visualizations and compile the visualizations as custom dashboards. To create custom dashboards, see “ Creating custom dashboards ” on page 214.	

Category	Metric	Description
		<p>You can export and import the assets (visualizations and dashboards) from external Kibana to API Gateway custom dashboard, and API Gateway custom dashboard to external Kibana. The export and import are possible between API Gateway instances running on the same tenant. API Gateway does not support importing the assets across different tenants.</p> <p>Note: You can import the assets created in Kibana 7.7.1 in to API Gateway.</p>

Creating custom dashboards

Pre-requisites:

You must have the API Gateway's manage custom dashboards functional privilege assigned to manage the custom dashboards in **Global analytics**.

The **Custom dashboards** has two options:

- **View.** To view custom dashboards. You can select the custom dashboard that you want to view from the drop-down list.
- **Build.** To build custom dashboards. Here, you can create and add visualizations to build custom dashboards.

Note:

- Software AG recommends that you do not modify any default visualizations. If you modify any default visualizations, an upgrade or restart of API Gateway server overwrites the modifications.
- Improper visualizations or long running queries in the visualizations can impact the performance of the Elasticsearch.

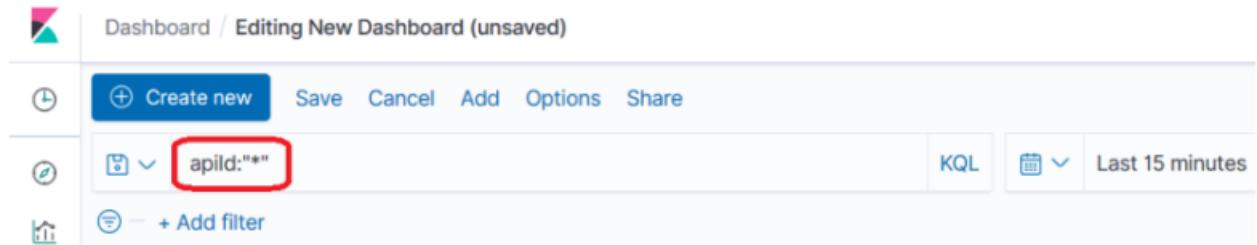
You can create custom dashboards for both API Gateway-wide analytics and API-specific analytics

from the  > **Analytics** page. You can create the visualizations and dashboards in the Kibana interface, and the dashboards are rendered in API Gateway user interface.

➤ To create custom dashboards

1. Click  > **Analytics**.
2. Click **Custom dashboards** > **Build**.
3. Build custom dashboard. For instructions, see Kibana documentation.

For API-specific custom dashboard, type `apiId:"*"` in the search box and then save the dashboard. This filter creates a custom dashboard specific to API-specific analytics.



To view the custom dashboard for API Gateway-wide analytics, click **View** and select the dashboard from the drop-down list.

To view the custom dashboard for API-specific analytics, click **APIs** > *API name* > **Analytics**. Select the custom dashboard from the drop-down list.

Kibana Limitations

- In the Analytics dashboard, after applying a filter and refreshing the page, the filter remains visible in the user interface, but the corresponding filtered results are not being displayed. Furthermore, if you open the link in a new browser, the filter is not retained or displayed.
- Due to a vulnerability, the functionality of **Permalinks** and **Embedcodes**, which are accessible on the user interface, is currently inactive.
- Within the runtime events section, you have the option to improve an event's visibility by selecting "Toggle column in table." This choice leads to the inclusion of the chosen field in the runtime events column, rendering it visible. It is crucial to recognize that this customization is fleeting and will reset after a server restart, causing the field to no longer appear.
- Expanding and incorporating a field as a column in the transactional event table within Analytics works as intended. However, when you download a CSV file, only the default columns are included. This is a Kibana functionality. When the Download as CSV is submitted, Kibana downloads the fields present on the original definition of the dashboard and the custom fields added are not considered.

Runtime Events and Metrics Data Model

API Gateway generates runtime events and Key Performance Indicator (KPI) metrics for the currently active APIs. The types of runtime events that API Gateway can generate are:

Events	Description
Lifecycle	A Lifecycle event is generated each time the API Gateway instance is started or shut down.
Error	An Error event is generated each time an invocation of API results in an error.
Policy Violation	A Policy Violation event is generated each time an invocation of API violates a policy that was configured for the API.

Events	Description
Transaction	A Transaction event is generated each time an API is invoked (successfully or unsuccessfully).
Monitor	A Monitor event is generated when a configured SLA parameter, such as the average response time, fault count, availability, and so on, is breached for the API.

KPI metrics are used to monitor the run-time execution of APIs. Metrics include the maximum response time, average response time, fault count, availability of APIs, and so on. If you include runtime monitoring policies, the policies will monitor the KPI metrics for APIs, and can send alerts to various destinations when user-specified performance conditions for an API are violated. The KPI metrics that API Gateway can generate are:

Metric	Reports on...
Availability	The percentage of time that an API was available during the current interval. A value of 100 indicates that the API was always available. Only the time when the API is unavailable counts against this metric. If invocations fail due to policy violations, this parameter could still be as high as 100.
Average Response Time	The average amount of time it took the API 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 client.
Fault Count	The number of failed invocations in the current interval.
Maximum Response Time	The maximum amount of time it took the API to complete an invocation in the current interval.
Minimum Response Time	The minimum amount of time it took the API to complete an invocation in the current interval.
Successful Request Count	The number of successful API invocations in the current interval.
Total Request Count	The total number of requests for each API running in API Gateway in the current interval.

You can configure API Gateway to publish the runtime events and metrics data to different destinations. The following sections describe the runtime events and metrics data model for each of these destinations:

- API Gateway
- API Portal
- Audit Log
- CentraSite

- Elasticsearch
- Email
- JDBC
- Local Log

API Gateway

The runtime events and metrics payload is generated by API Gateway at run-time. The columns that make up the events and metrics data model for API Gateway are listed below:

Transactional Events

Column	Description
apild	The unique identifier for the API. Example: c0f84954-9732-11e5-b9f4-f159eafe47b1
apiName	Name of the API in which the event occurred. Example: pet1
apiVersion	The system-assigned version identifier for the API. Example: 1.0
applicationId	The unique identifier for the application associated with the API invocation. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
applicationIp	IP address of the application associated with the API invocation. Example: 10.20.248.33
applicationName	Name of the application associated with the API invocation. An application name is populated as unknown when API Gateway is unable to identify the application using a security policy that is configured for the API. Example: SampleApplication
callbackRequest	Indicates whether the event is generated for a callback request. Possible values are: <ul style="list-style-type: none"> ■ true. This denotes that the event is generated for a callback request.

Column	Description
	<ul style="list-style-type: none"> ■ false. This denotes that the event is generated for a normal response.
correlationID	<p>The unique identifier that is automatically generated for every request coming to API Gateway and can be used to query the log.</p> <p>Example: MED38e9cfa4-2348-408b-9462-124b2181c1a6:656</p>
creationDate	<p>Date and time when the event was generated in API Gateway.</p> <p>Example: 1501671101509</p>
customFields	<p>The custom fields an API Provider can provide to log a new field and value for a transaction event.</p> <p>Example: {"customfield": "customvalue"}</p>
errorOrigin	<p>The origin of error.</p> <p>Example: Nativeservice</p>
eventType	<p>The type of event that occurred.</p> <p>Example: Transactional</p>
externalCalls	<p>List the external calls from API Gateway. These external calls can be to a native service or service registry.</p> <p>Example:</p> <pre>[{ "externalCallType": "SERVICE_REGISTRY_CALL", "externalURL": "http://service.registry.com", "callDuration": 49, "callStartTime": 1562244570486, "callEndTime": 1562244570535, "responseCode": "200" }, { "externalCallType": "NATIVE_SERVICE_CALL", "externalURL": "https://petstore.swagger.io/v2/store/inventory", "callDuration": 1285, "callStartTime": 1562244569252, "callEndTime": 1562244570537, "responseCode": "200" }]</pre>
gatewayTime	<p>Duration in milliseconds, to process a request by API Gateway. This does not include native service processing duration.</p> <p>gatewayTime = totalTime - providerTime</p> <p>Example: 20</p>

Column	Description
httpMethod	The HTTP method used to invoke the API. Example: GET
nativeHttpMethod	The HTTP method used to invoke the native service. Example: GET
nativeReqPayload	The native service request data. Example: <pre>{ "param1" : "value1", "param2" : 10 }</pre>
nativeRequestHeaders	Request header in the incoming request from the API Gateway to native service. Example: <pre>{ "Authorization": "*****", "Accept": "*/*", "Authorization": "*****", "Accept": "*/*", "Cache-Control": "no-cache", "User-Agent": "PostmanRuntime/7.13.0", "Postman-Token": "381424fa-e3b3-4058-8df9-4abf9d72c899", "postmanHeader": "hello", "accept-encoding": "gzip, deflate", "Content-Type": "application/x-www-form-urlencoded" }</pre>
nativeResPayload	The native service response data. Example: <pre>{ "id": 2, "category": { "id": 2, "name": "string" }, "name": "pysen", "photoUrls": ["string"], "tags": [{ "id": 0, "name": "string" }], "status": "available" }</pre>

Column	Description
nativeResponseHeaders	<p>Response header in the outgoing response from the native service to API Gateway.</p> <p>Example:</p> <pre>{ "Server": "Jetty(9.2.9.v20150224)", "Access-Control-Allow-Origin": "*", "Access-Control-Allow-Methods": "GET, POST, DELETE, PUT", "Connection": "close", "Date": "Fri, 07 Jun 2019 12:44:13 GMT", "Access-Control-Allow-Headers": "Content-Type, api_key, Authorization", "Content-Type": "application/json" }</pre>
nativeURL	<p>URL of the native service.</p> <p>Example: http://petstore.swagger.io/v2/pet/2</p>
operationName	<p>Name of the API operation that is invoked.</p> <p>Example: Using a Calculator API, you can perform various operations such as addition, subtraction, multiplication, and division. When an addition operation is invoked in API Gateway, then the operation field name is populated as addInts.</p>
queryParameters	<p>This is applicable only for REST APIs. Query parameters present in the incoming REST request.</p> <p>Example: {"status":"available"}</p>
reqPayload	<p>The API request payload data.</p> <p>Example: RequestPayload</p>
requestHeaders	<p>Request header in the incoming request from the client.</p> <p>Example:</p> <pre>{ "Cache-Control": "max-age=0", "Accept": "text/plain, application/json; q=0.9, image/webp, image/apng, */*; q=0.8", "Upgrade-Insecure-Requests": "1", "Connection": "keep-alive", "User-Agent": "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36", "Host": "mcdaso02:5555", "Accept-Encoding": "gzip, deflate", "Accept-Language": "en-US, en; q=0.9, ta; q=0.8",</pre>

Column	Description
	<pre>"Content-Type":"application/x-www-form-urlencoded" }</pre>
responseCode	The HTTP response status code that indicates success or failure of the requested operation. Example: 200
responseHeaders	Response header in the outgoing response. Example: <pre>{ "Server": "Jetty(9.2.9.v20150224)", "Access-Control-Allow-Origin": "*", "Access-Control-Allow-Methods": "GET,POST, DELETE, PUT", "Connection": "close", "Date": "Fri, 30 Mar 2018 08:25:45 GMT", "Access-Control-Allow-Headers": "Content-Type, api_key, Authorization", "Content-Type": "application/xml" }</pre>
serverID	The API Gateway server on which the transaction event occurred. This column is currently not used. It appears as NULL or as an empty string. Example: SampleHost:80
sessionId	A string the API Gateway server generates to uniquely identify each session. This is either the IS session token or the automatically generated GUID if the token is missing from the message context. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
sourceGateway	Source of event generation. Possible values: APIGateway or Microgateway .
sourceGatewayNode	Source API Gateway's IP address. Example: 10.0.75.1
status	Status of the API request. Possible values are: SUCCESS, FAILURE
totalContentSize	The total combined size of request and response payloads in bytes. Example: 100
totalTime	Time in milliseconds required to invoke the API provider. This time includes the overhead incurred by API Gateway. Overhead includes

Column	Description
	<p>security overhead for encryption, decryption, and load-balance retries.</p> <p>Example: 120</p>

Error Events

Column	Description
apiId	<p>The unique identifier for the API.</p> <p>Example: c0f84954-9732-11e5-b9f4-f159eafe47b1</p>
apiName	<p>Name of the API in which the event occurred.</p> <p>Example: SampleAPI</p>
apiVersion	<p>The system-assigned version identifier for the API.</p> <p>Example: 1.0</p>
applicationId	<p>The unique identifier for the application associated with the API invocation.</p> <p>Example: c0f84954-9732-11e5-b9f4-f159eafe47b2</p>
applicationIp	<p>IP address of the application associated with the API invocation.</p> <p>Example: 10.20.248.33</p>
applicationName	<p>Name of the application associated with the API invocation.</p> <p>An application name is populated as unknown when API Gateway is unable to identify the application using a security policy that is configured for the API.</p> <p>Example: SampleApplication</p>
correlationID	<p>The unique identifier that is automatically generated for every request coming to API Gateway and can be used to query the log.</p> <p>Example: MED38e9cfa4-2348-408b-9462-124b2181c1a6:656</p>
creationDate	<p>Date and time when the event was generated in API Gateway.</p> <p>Example: 1501671101509</p>
errorDesc	<p>Message that describes the error that occurred.</p> <p>Example: Invocation for SampleAPI was rejected based on policy violation, response code: 503</p>

Column	Description
eventType	The type of event that occurred. Example: Error Event
operationName	Name of the API operation that is invoked. Example: Using a Calculator API, you can perform various operations such as addition, subtraction, multiplication, and division. When an addition operation is invoked in API Gateway, then the operation field name is populated as addInts .
responseCode	The HTTP response status code that indicates success or failure of the requested operation. Example: 200
sessionId	A string the API Gateway server generates to uniquely identify each session. This is either the IS session token or the automatically generated GUID if the token is missing from the message context. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
sourceGateway	Source of event generation. Possible values: APIGateway or Microgateway .
sourceGatewayNode	Source API Gateway's IP address. Example: 10.0.75.1
userAgent	Name of the client used to invoke the API. Example: Postman

Monitoring Events

Column	Description
alertDesc	Text of the alert message sent to a configured destination when the performance conditions are violated. The alert message is specified in the policy definition of an API. Example: EnforcePolicy-HardLimit
alertType	The type of alert generated for the event. Possible values are: Monitor, sla
apilid	The unique identifier for the API. Example: c0f84954-9732-11e5-b9f4-f159eafe47b1

Column	Description
apiName	Name of the API in which the event occurred. Example: SampleAPI
apiVersion	The system-assigned version identifier for the API. Example: 1.0
applicationId	The unique identifier for the application associated with the API invocation. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
applicationIp	IP address of the application associated with the API invocation. Example: 10.20.248.33
applicationName	Name of the application associated with the API invocation. An application name is populated as unknown when API Gateway is unable to identify the application using a security policy that is configured for the API. Example: SampleApplication
creationDate	Date and time when the event was generated in API Gateway. Example: 1501671101509
eventSource	The source where the event occurred. Example: API_Gateway_Instance
eventType	The type of event that occurred. Example: Monitor Event
httpMethod	The HTTP method used to invoke the API. Example: GET
operationName	Name of the API operation that is invoked. Example: Using a Calculator API, you can perform various operations such as addition, subtraction, multiplication, and division. When an addition operation is invoked in API Gateway, then the operation field name is populated as addInts .
sessionId	A string the API Gateway server generates to uniquely identify each session. This is either the IS session token or the automatically generated GUID if the token is missing from the message context. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2

Column	Description
sourceGateway	Source of event generation. Possible values: APIGateway or Microgateway .
sourceGatewayDetails	Details of events generated only from Microgateway. The details include Microgateway Id, Source Gateway Host, Source Gateway Port, Source Gateway Version, Microgateway pool.

Lifecycle Events

Column	Description
creationDate	Date and time when the event was generated in API Gateway. Example: 1501671101509
eventType	The type of event that occurred. Example: LifeCycle
gatewayStatus	Status of the API Gateway instance. Possible values are: STARTED or STOPPED
sourceGateway	Source of event generation. Possible values: APIGateway or Microgateway .
sourceGatewayDetails	Details of events generated only from Microgateway. The details include Microgateway Id, Source Gateway Host, Source Gateway Port, Source Gateway Version, Microgateway pool.

Policy Violation Events

Column	Description
alertDesc	Text of the alert message sent to a configured destination when the performance conditions are violated. The alert message is specified in the policy definition of an API. Example: A violation was detected for policy (Unknown-Policyuser): application could not be identified. Anonymous access is not allowed for this service!
alertSource	Name of the API Gateway policy that generated the alert message. Example: Unknown-Policy

Column	Description
alertType	The type of alert generated for the event. Example: PolicyViolation
apiId	The unique identifier for the API. Example: c0f84954-9732-11e5-b9f4-f159eafe47b1
apiName	Name of the API in which the event occurred. Example: SampleAPI
apiVersion	The system-assigned version identifier for the API. Example: 1.0
applicationId	The unique identifier for the application associated with the API invocation. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
applicationIp	IP address of the application associated with the API invocation. Example: 10.20.248.33
applicationName	Name of the application associated with the API invocation. An application name is populated as unknown when API Gateway is unable to identify the application using a security policy that is configured for the API. Example: SampleApplication
creationDate	Date and time when the event was generated in API Gateway. Example: 1501671101509
eventType	The type of event that occurred. Example: Policy Violation Event
httpMethod	The HTTP method used to request the API access. Example: GET
operationName	Name of the API operation that is invoked. Example: Using a Calculator API, you can perform various operations such as addition, subtraction, multiplication, and division. When an addition operation is invoked in API Gateway, then the operation field name is populated as addInts .

Column	Description
responseCode	The HTTP response status code that indicates success or failure of the requested operation. Example: 200
sessionId	A string the API Gateway server generates to uniquely identify each session. This is either the IS session token or the automatically generated GUID if the token is missing from the message context. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
sourceGateway	Source of event generation. Possible values: APIGateway or Microgateway .
sourceGatewayNode	Source API Gateway's IP address. Example: 10.0.75.1
userAgent	Name of the client used to invoke the API. Example: Postman

Performance Metrics

Column	Description
apild	The unique identifier for the API. Example: c0f84954-9732-11e5-b9f4-f159eafe47b1
apiName	Name of the API in which the event occurred. Example: SampleAPI
apiVersion	The system-assigned version identifier for the API. Example: 1.0
availability	The percentage of time that an API was available during the current interval. A value of 100 indicates that the API was always available. If invocations fail due to policy violations, this parameter could still be as high as 100. Example: 100
avgResponseTime	The average amount of time it took the API to complete each invocation in the current interval. Response time is measured from the moment API Gateway receives the request until the moment it returns the response to the caller.

Column	Description
	Example: 135
creationDate	Date and time when the event was generated in API Gateway. Example: 1501671101509
eventType	The type of event that occurred. Example: Performance Metrics Event
faultCount	The number of failed API invocations in the current interval. Example: 10
intervalStart	The starting date and time from which you want to examine metrics. Example: 1526294632172
intervalStop	The ending date and time until which you want to examine metrics. Example: 1526294632182
maxResponseTime	The maximum amount of time (in milliseconds) it took for the API to complete an invocation in the current interval. Example: 343
minResponseTime	The minimum amount of time (in milliseconds) it took for the API to complete an invocation in the current interval. Example: 10
sourceGateway	Source of event generation. Possible values: APIGateway or Microgateway .
successCount	The number of successful API invocations in the current interval. Example: 100
totalCount	The total number of API invocations (successful and unsuccessful) in the current interval. Example: 110

Threat Protection Events

Column	Description
id	The unique identifier for an event.

Column	Description
	Example: 8e05267a-45c9-45f0-a3dd-8b2ee1e98ca2
alertAction	A helpful action taken on the API for the alert. Example: DENY
creationDate	Date and time when the event was generated in API Gateway. Example: 1501671101509
eventType	The type of event that occurred. Example: Transactional
filterName	Name of the threat protection filter. Example: DoSFilter
message	If the API invocation failed, message that describes the error that occurred. Example: Global Denial of Service limits were reached: Maximum requests limit of 2 in 120 seconds has been exceeded.
requestHost	Hostname of the machine from which the API access request was submitted. Example: 10.60.34.152
requestTime	Date and time the request was submitted. Example: 1501671101509
requestType	The type of request that was received for the API. Example: ALL
resourcePath	The relative URI path of a resource that was used for API invocation. Example: invoke/pub.date/getCurrentDate
ruleName	The API Gateway rule that triggered the event. Example: GlobalDoSRule
serverHost	The name or IP address of the machine on which the thread protection server is running. Example: 10.60.34.83
serverPort	The port number on which the thread protection server is configured to listen for incoming requests.

Column	Description
	Example: 8911

API Portal

The runtime events and metrics payload generated by API Gateway at run-time is published to the configured API Portal destination. The columns that make up the events and metrics data model for API Portal are listed below:

Transactional Events

Column	Description
apiId	The unique identifier for the API. Example: c0f84954-9732-11e5-b9f4-f159eafe47b1
apiName	Name of the API in which the event occurred. Example: SampleAPI
apiVersion	The system-assigned version identifier for the API. Example: 1.0
consumerId	The unique identifier for the consumer associated with the API invocation. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
consumerIp	IP address of the consumer associated with the API invocation. Example: 10.1.1.211
consumerName	Name of the consumer associated with the API invocation. A consumer name is populated as unknown when API Gateway is unable to identify the consumer using a security policy that is configured for the API. Example: SampleApplication
correlationID	The unique identifier that is automatically generated for every request coming to API Gateway and can be used to query the log. Example: MED38e9cfa4-2348-408b-9462-124b2181c1a6:656
creationDate	Date and time when the event was generated in API Gateway. Example: 1501671101509

Column	Description
customFieldsrequest	The custom fields an API Provider can provide to log a new field and value for a transaction event. Example: {"customfield":"customvalue"}
errorOrigin	The origin of error. Example: Nativeserivce
eventType	The type of event that occurred. Example: Transactional
externalCalls	List the external calls from API Gateway. These external calls can be to a native service or service registry. Example: <pre>[{ "externalCallType": "SERVICE_REGISTRY_CALL", "externalURL": "http://service.registry.com", "callDuration": 49, "callStartTime": 1562244570486, "callEndTime": 1562244570535, "responseCode": "200" }, { "externalCallType": "NATIVE_SERVICE_CALL", "externalURL": "https://petstore.swagger.io/v2/store/inventory", "callDuration": 1285, "callStartTime": 1562244569252, "callEndTime": 1562244570537, "responseCode": "200" }]</pre>
gatewayTime	Duration in milliseconds, to process a request by API Gateway. This does not include native service processing duration. gatewayTime = totalTime - providerTime Example: 20
operationName	Name of the API operation that is invoked. Example: Using a Calculator API, you can perform various operations such as addition, subtraction, multiplication, and division. When an addition operation is invoked in API Gateway, then the operation field name is populated as addInts .
providerTime	Time in milliseconds required for API Gateway to invoke a native provider and receive a response. This time includes the overhead incurred by API Gateway. Overhead includes the time it takes for a

Column	Description
	provider to process a request and return a response, plus any network latency to or from the provider. Subtracting total time from provider time must give a rough indicator of the API Gateway overhead.
	Example: 20
queryParameters	This is applicable only for REST APIs. Query parameters present in the incoming REST request.
	Example: {"status": "available"}
requestHeaders	Request header in the incoming request from the client.
	Example:
<pre>{ "Cache-Control": "max-age=0", "Accept": "text/plain, application/json; q=0.9, image/webp, image/apng, */*; q=0.8", "Upgrade-Insecure-Requests": "1", "Connection": "keep-alive", "User-Agent": "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36", "Host": "mcdaso02:5555", "Accept-Encoding": "gzip, deflate", "Accept-Language": "en-US,en;q=0.9,ta;q=0.8", "Content-Type": "application/x-www-form-urlencoded" }</pre>	
response	The API response payload data.
	Example: <ResponsePayload>
responseCode	The HTTP response status code that indicates success or failure of the requested operation.
	Example: 200
responseHeaders	Response header in the outgoing response.
	Example:
<pre>{ "Server": "Jetty(9.2.9.v20150224)", "Access-Control-Allow-Origin": "*", "Access-Control-Allow-Methods": "GET, POST, DELETE, PUT", "Connection": "close", "Date": "Fri, 30 Mar 2018 08:25:45 GMT", "Access-Control-Allow-Headers": "Content-Type, api_key, Authorization", "Content-Type": "application/xml" }</pre>	

Column	Description
nativeHttpMethod	The HTTP method used to invoke the native service.
	Example: GET
nativeRequestHeaders	Request header in the incoming request from the API Gateway to native service.
	Example:
	<pre>{ "Authorization": "*****", "Accept": "*/*", "Authorization": "*****", "Accept": "*/*", "Cache-Control": "no-cache", "User-Agent": "PostmanRuntime/7.13.0", "Postman-Token": "381424fa-e3b3-4058-8df9-4abf9d72c899", "postmanHeader": "hello", "accept-encoding": "gzip, deflate", "Content-Type": "application/x-www-form-urlencoded" }</pre>
nativeRequestPayload	The native service request data.
	Example:
	<pre>{ "param1" : "value1", "param2" : 10 }</pre>
nativeResponsePayload	The native service response data.
	Example:
	<pre>{ "id":2, "category": { "id":2, "name":"string" }, "name":"pysen", "photoUrls":["string"], "tags": [{ "id":0, "name":"string" }], "status":"available" }</pre>
nativeURL	URL of the native service.
	Example: http://petstore.swagger.io/v2/pet/2

Column	Description
sessionId	A string the API Gateway server generates to uniquely identify each session. This is either the IS session token or the automatically generated GUID if the token is missing from the message context. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
sourceGatewayNode	Source API Gateway's IP address. Example: 10.0.75.1
status	Status of the API request. Possible values are: SUCCESS, FAILURE
targetName	Name of the API Gateway instance reporting the event. Example: API_Gateway_Instance
totalContentSize	The total combined size of request and response payloads in bytes. Example: 100
totalTime	Time in milliseconds required to invoke the API provider. This time includes the overhead incurred by API Gateway. Overhead includes security overhead for encryption, decryption, and load-balance retries. Example: 120

Error Events

Column	Description
apiId	The unique identifier for the API. Example: c0f84954-9732-11e5-b9f4-f159eafe47b1
apiName	Name of the API in which the event occurred. Example: SampleAPI
apiVersion	The system-assigned version identifier for the API. Example: 1.0
consumerId	The unique identifier for the consumer associated with the API invocation. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
consumerIp	IP address of the consumer associated with the API invocation.

Column	Description
	Example: 10.20.248.33
consumerName	<p>Name of the consumer associated with the API invocation.</p> <p>A consumer name is populated as unknown when API Gateway is unable to identify the consumer using a security policy that is configured for the API.</p>
	Example: SampleApplication
correlationID	<p>The unique identifier that is automatically generated for every request coming to API Gateway and can be used to query the log.</p>
	Example: MED38e9cfa4-2348-408b-9462-124b2181c1a6:656
creationDate	<p>Date and time when the event was generated in API Gateway.</p>
	Example: 1501671101509
errorDesc	<p>Message that describes the error that occurred.</p> <p>Service invocation for SampleAPI was rejected based on policy violation, response code: 503</p>
eventType	<p>The type of event that occurred.</p>
	Example: Error Event
operationName	<p>Name of the API operation that is invoked.</p> <p>Example: Using a Calculator API, you can perform various operations such as addition, subtraction, multiplication, and division. When an addition operation is invoked in API Gateway, then the operation field name is populated as addInts.</p>
responseCode	<p>The HTTP response status code that indicates success or failure of the requested operation.</p>
	Example: 503
sessionId	<p>A string the API Gateway server generates to uniquely identify each session. This is either the IS session token or the automatically generated GUID if the token is missing from the message context.</p>
	Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
targetName	<p>Name of the API Gateway instance reporting the event.</p>
	Example: API_Gateway_Instance

Monitoring Events

Column	Description
alertDesc	<p>Text of the alert message sent to a configured destination when the performance conditions are violated. The alert message is specified in the policy definition of an API.</p> <p>Example: EnforcePolicy-HardLimit</p>
alertSource	<p>Name of the API Gateway policy that generated the alert message.</p> <p>Example: Unknown-Policy</p>
alertType	<p>The type of alert generated for the event.</p> <p>Example: sla</p>
apilid	<p>The unique identifier for the API.</p> <p>Example: c0f84954-9732-11e5-b9f4-f159eafe47b1</p>
apiName	<p>Name of the API in which the event occurred.</p> <p>Example: SampleAPI</p>
apiVersion	<p>The system-assigned version identifier for the API.</p> <p>Example: 1.0</p>
applicationId	<p>The unique identifier for the application associated with the API invocation.</p> <p>Example: c0f84954-9732-11e5-b9f4-f159eafe47b2</p>
applicationIp	<p>IP address of the application associated with the API invocation.</p> <p>Example: 10.20.248.33</p>
applicationName	<p>Name of the application associated with the API invocation.</p> <p>An application name is populated as unknown when API Gateway is unable to identify the application using a security policy that is configured for the API.</p> <p>Example: SampleApplication</p>
creationDate	<p>Date and time when the event was generated in API Gateway.</p> <p>Example: 1501671101509</p>
eventType	<p>The type of event that occurred.</p> <p>Example: Monitor Event</p>

Column	Description
monitorAttr	The monitored attribute which has breached the configured SLA. Example: AVGRESPONSETIME GT 1.0, SUCCESSCOUNT EQ 3, REQUESTCOUNT GT 10
operationName	Name of the API operation that is invoked. Example: Using a Calculator API, you can perform various operations such as addition, subtraction, multiplication, and division. When an addition operation is invoked in API Gateway, then the operation field name is populated as addInts .
sessionId	A string the API Gateway server generates to uniquely identify each session. This is either the IS session token or the automatically generated GUID if the token is missing from the message context. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
targetName	Name of the API Gateway instance reporting the event. Example: API_Gateway_Instance

Lifecycle Events

Column	Description
creationDate	Date and time when the event was generated in API Gateway. Example: 1501671101509
eventStatus	Status of the API Gateway instance. Possible values are: STARTED or STOPPED
eventType	The type of event that occurred. Example: LifeCycle
targetName	Name of the API Gateway instance reporting the event. Example: API_Gateway_Instance

Policy Violation Events

Column	Description
alertDesc	<p>Text of the alert message sent to a configured destination when the performance conditions are violated. The alert message is specified in the policy definition of an API.</p> <p>Example: A violation was detected for policy (Unknown-Policyuser): application could not be identified. Anonymous access is not allowed for this service!</p>
alertSource	<p>Name of the API Gateway policy that generated the alert message.</p> <p>Example: Unknown-Policy</p>
alertType	<p>The type of alert generated for the event.</p> <p>Example: PolicyViolation</p>
apild	<p>The unique identifier for the API.</p> <p>Example: c0f84954-9732-11e5-b9f4-f159eafe47b1</p>
apiName	<p>Name of the API in which the event occurred.</p> <p>Example: SampleAPI</p>
apiVersion	<p>The system-assigned version identifier for the API.</p> <p>Example: 1.0</p>
consumerId	<p>The unique identifier for the consumer associated with the API invocation.</p> <p>Example: c0f84954-9732-11e5-b9f4-f159eafe47b2</p>
consumerIp	<p>IP address of the consumer associated with the API invocation.</p> <p>Example: 10.20.248.33</p>
consumerName	<p>Name of the consumer associated with the API invocation.</p> <p>A consumer name is populated as unknown when API Gateway is unable to identify the consumer using a security policy that is configured for the API.</p> <p>Example: SampleApplication</p>
creationDate	<p>Date and time when the event was generated in API Gateway.</p> <p>Example: 1501671101509</p>
eventType	<p>The type of event that occurred.</p>

Column	Description
	Example: Policy Violation Event
operationName	Name of the API operation that is invoked. Example: Using a Calculator API, you can perform various operations such as addition, subtraction, multiplication, and division. When an addition operation is invoked in API Gateway, then the operation field name is populated as addInts .
responseCode	The HTTP response status code that indicates success or failure of the requested operation. Example: 200
sessionId	A string the API Gateway server generates to uniquely identify each session. This is either the IS session token or the automatically generated GUID if the token is missing from the message context. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
targetName	Name of the API Gateway instance reporting the event. Example: API_Gateway_Instance

Performance Metrics

Column	Description
apilid	The unique identifier for the API. Example: c0f84954-9732-11e5-b9f4-f159eafe47b1
apiName	Name of the API in which the event occurred. Example: SampleAPI
apiVersion	The system-assigned version identifier for the API. Example: 1.0
availability	The percentage of time that an API was available during the current interval. A value of 100 indicates that the API was always available. If invocations fail due to policy violations, this parameter could still be as high as 100. Example: 100
avgResponseTime	The average amount of time it took the API to complete each invocation in the current interval. Response time is measured from

Column	Description
	the moment API Gateway receives the request until the moment it returns the response to the caller. Example: 135
creationDate	Date and time when the event was generated in API Gateway. Example: 1501671101509
eventType	The type of event that occurred. Example: Performance Metrics Event
faultCount	The number of failed API invocations in the current interval. Example: 10
includeFaults	Includes failed API invocations. Possible values are: true, false
intervalStart	The starting date and time from which you want to examine metrics. Example: 2015-08-26 04:13:35 PM
intervalStop	The ending date and time until which you want to examine metrics. Example: 2015-08-26 04:13:45 PM
maxResponseTime	The maximum amount of time (in milliseconds) it took for the API to complete an invocation in the current interval. Example: 343
minResponseTime	The minimum amount of time (in milliseconds) it took for the API to complete an invocation in the current interval. Example: 10
operationName	Name of the API operation that is invoked. Example: Using a Calculator API, you can perform various operations such as addition, subtraction, multiplication, and division. When an addition operation is invoked in API Gateway, then the operation field name is populated as addInts .
successCount	The number of successful API invocations in the current interval. Example: 100
targetName	Name of the API Gateway instance reporting the event. Example: API_Gateway_Instance

Column	Description
totalCount	The total number of API invocations (successful and unsuccessful) in the current interval.
	Example: 110

Audit Log

The runtime events and metrics payload generated by API Gateway at run-time is published to the configured Audit Log destination. The columns that make up the events and metrics data model for Audit Log are listed below:

Transactional Events

Column	Description
API_ID	The unique identifier for the API. Example: ec1473cc-40a0-479e-9126-474a917c3c89
API_NAME	Name of the API in which the event occurred. Example: SampleAPI
API_VERSION	The system-assigned version identifier for the API. Example: 1.0
AUDITTIMESTAMP	Date and time when the event was written to the log. Example: 2017-08-07 07:22:22
CONSUMER_IP	IP address of the consumer associated with the API invocation. Example: 10.60.37.42
CONSUMER_NAME	Name of the consumer associated with the API invocation. A consumer name is populated as unknown when API Gateway is unable to identify the consumer using a policy that is configured for the API. Example: SampleApplication
CONTEXTID	The unique identifier for the current context information API Gateway uses to connect related entries from different logs. This column is currently not used. It appears as NULL or as an empty string. Example: 81546147-41a8-4998-8150-02ba67bb08c2

Column	Description
CORRELATIONID	The unique identifier that is automatically generated for every request coming to API Gateway and can be used to query the log. Example: MED38e9cfa4-2348-408b-9462-124b2181c1a6:656
CUSTOMFIELDS	The custom fields an API Provider can provide to log a new field and value for a transaction event. Example: {"customfield": "customvalue"}
ERROR_ORIGIN	The origin of error. Example: Nativeservice
EVENT_PK	The primary key (PK) that uniquely identifies the event that occurred. Example: 1
EXTERNAL_CALLS	List the external calls from API Gateway. These external calls can be to a native service or service registry. Example: <pre>[{ "externalCallType": "SERVICE_REGISTRY_CALL", "externalURL": "http://service.registry.com", "callDuration": 49, "callStartTime": 1562244570486, "callEndTime": 1562244570535, "responseCode": "200" }, { "externalCallType": "NATIVE_SERVICE_CALL", "externalURL": "https://petstore.swagger.io/v2/store/inventory", "callDuration": 1285, "callStartTime": 1562244569252, "callEndTime": 1562244570537, "responseCode": "200" }]</pre>
INSERTTIMESTAMP	Date and time when the event was generated in API Gateway. Example: 2017-08-07 07:22:22
MSGID	The ID assigned to the message by the API provider. This column is currently not used. Example: 361dc2f8-a60b-fc21-8545-9b07fce1a479
NATIVE_ENDPOINT	The endpoint URL of the native API that is invoked.

Column	Description
	Example: http://petstore.swagger.io/v2/pet/55
NATIVE_HTTP_METHOD	The HTTP method used to invoke the native service. Example: GET
NATIVE_REQUEST_HEADERS	Request header in the incoming request from the API Gateway to native service.
	<p>Example:</p> <pre>{ "Authorization": "*****", "Accept": "*/*", "Authorization": "*****", "Accept": "*/*", "Cache-Control": "no-cache", "User-Agent": "PostmanRuntime/7.13.0", "Postman-Token": "381424fa-e3b3-4058-8df9-4abf9d72c899", "postmanHeader": "hello", "accept-encoding": "gzip, deflate", "Content-Type": "application/x-www-form-urlencoded" }</pre>
NATIVE_REQ_PAYLOAD	The native service request data. Example:
	<pre>{ "param1" : "value1", "param2" : 10 }</pre>
NATIVE_RESPONSE_HEADERS	Response header in the outgoing response from the native service to API Gateway. Example:
	<pre>{ "Server": "Jetty(9.2.9.v20150224)", "Access-Control-Allow-Origin": "*", "Access-Control-Allow-Methods": "GET, POST, DELETE, PUT", "Connection": "close", "Date": "Fri, 07 Jun 2019 12:44:13 GMT", "Access-Control-Allow-Headers": "Content-Type, api_key, Authorization", "Content-Type": "application/json" }</pre>
NATIVE_RES_PAYLOAD	The native service response data.
	Example:

Column	Description
	<pre>{ "id":2, "category": [{ "id":2, "name":"string" }, "name":"pysen", "photoUrls":["string"], "tags": [{ "id":0, "name":"string" }], "status":"available" } }</pre>
NATIVE_URL	<p>URL of the native service.</p> <p>Example: http://petstore.swagger.io/v2/pet/2</p>
OPERATION_NAME	<p>Name of the API operation or resource that is invoked.</p> <p>Example: /pet/{petId}</p>
PROVIDER_TIME	<p>Time in milliseconds required for API Gateway to invoke a native provider and receive a response. This time includes the overhead incurred by API Gateway. Overhead includes the time it takes for a provider to process a request and return a response, plus any network latency to or from the provider. Subtracting total time from provider time must give a rough indicator of the API Gateway overhead.</p> <p>Example: 1336</p>
QUERY_PARAMETERS	<p>This is applicable only for REST APIs. Query parameters present in the incoming REST request.</p> <p>Example: {"status":"available"}</p>
REQUEST_HEADERS	<p>Request header in the incoming request from the client.</p> <p>Example:</p> <pre>{ "Cache-Control":"max-age=0", "Accept":"text/plain,application/json; q=0.9,image/webp,image/apng,*/*; q=0.8", "Upgrade-Insecure-Requests":"1", "Connection":"keep-alive", "User-Agent":"Mozilla/5.0(Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181Safari/537.36", "Host":"petstore.swagger.io", "DNT":"1", "Accept-Encoding":"gzip, deflate", "Accept-Language":"en-US,en;q=0.9" }</pre>

Column	Description
	<pre>"Host":"mcdaso02:5555", "Accept-Encoding":"gzip,deflate", "Accept-Language":"en-US,en;q=0.9,ta;q=0.8", "Content-Type":"application/x-www-form-urlencoded" }</pre>
RESPONSE_HEADERS	<p>Response header in the outgoing response.</p> <p>Example:</p> <pre>{ "Server":"Jetty(9.2.9.v20150224)", "Access-Control-Allow-Origin":"*", "Access-Control-Allow-Methods":"GET,POST,DELETE, PUT", "Connection":"close", "Date":"Fri, 30 Mar 2018 08:25:45 GMT", "Access-Control-Allow-Headers":"Content-Type,api_key, Authorization", "Content-Type":"application/xml" }</pre>
ROOTCONTEXTID	<p>The unique identifier for the root context information API Gateway uses to connect related entries from different logs.</p> <p>This column is currently not used. It appears as NULL or as an empty string.</p> <p>Example: 81546147-41a8-4998-8150-02ba67bb08c2</p>
SERVID	<p>The API Gateway server on which the transaction event occurred.</p> <p>This column is currently not used. It appears as NULL or as an empty string.</p> <p>Example: SampleHost:80</p>
SERVICE_NAME	<p>Name of the service in which the event occurred.</p> <p>Example: Swagger_Petstore</p>
SESSION_ID	<p>A string the API Gateway server generates to uniquely identify each session. This is either the IS session token or the automatically generated GUID if the token is missing from the message context.</p> <p>Example: 6dfcd849198c4a7e96b4ff89bc2deaf5</p>
SOURCE_GATEWAY_NODE	<p>Source API Gateway's IP address.</p> <p>Example: 10.0.75.1</p>
STATUS	<p>Status of the API request.</p> <p>Possible values are: SUCCESS, FAILURE</p>

Column	Description
TOTAL_TIME	Time in milliseconds required to invoke the API provider. This time includes the overhead incurred by API Gateway. Overhead includes security overhead for encryption, decryption, and load-balance retries. Example: 1042

CentraSite

The runtime events and metrics payload generated by API Gateway at run-time is published to the configured CentraSite destination. The columns that make up the events and metrics data model for CentraSite are listed below:

Transactional Events

Column	Description
ApiUserVersion	The system-assigned version identifier for the API. Example: 1.0
Consumer	Name of the consumer associated with the API invocation. A consumer name is populated as unknown when API Gateway is unable to identify the consumer using a policy that is configured for the API. Example: SampleApplication
ConsumerId	The unique identifier for the consumer associated with the API invocation. Example: be8b27d6-8f79-4c6e-b06c-a628d2ba30c3
Consumer IP Address	IP address of the consumer associated with the API invocation. Example: 10.60.20.169
CorrelationID	The unique identifier that is automatically generated for every request coming to API Gateway and can be used to query the log. Example: MED38e9cfa4-2348-408b-9462-124b2181c1a6:656
Created Time	Date and time when the event was generated in API Gateway. Example: 2017-08-09 01:27:45 AM
CustomFields	The custom fields an API Provider can provide to log a new field and value for a transaction event.

Column	Description
	Example: {"customfield":"customvalue"}
ErrorOrigin	The origin of error. Example: Nativeservice
Event Type	The type of event that occurred. Example: Transaction Event
External Calls	List the external calls from API Gateway. These external calls can be to a native service or service registry. Example: <pre>[{ "externalCallType": "SERVICE_REGISTRY_CALL", "externalURL": "http://service.registry.com", "callDuration": 49, "callStartTime": 1562244570486, "callEndTime": 1562244570535, "responseCode": "200" }, { "externalCallType": "NATIVE_SERVICE_CALL", "externalURL": "https://petstore.swagger.io/v2/store/inventory", "callDuration": 1285, "callStartTime": 1562244569252, "callEndTime": 1562244570537, "responseCode": "200" }]</pre>
Native HTTP Method	The HTTP method used to invoke the native service. Example: GET
Native Request Headers	Request header in the incoming request from the API Gateway to native service. Example: <pre>{ "Authorization": "*****", "Accept": "*/*", "Authorization": "*****", "Accept": "*/*", "Cache-Control": "no-cache", "User-Agent": "PostmanRuntime/7.13.0", "Postman-Token": "381424fa-e3b3-4058-8df9-4abf9d72c899", "postmanHeader": "hello", "accept-encoding": "gzip, deflate", "Content-Type": "application/x-www-form-urlencoded" }</pre>

Column	Description
Native Req Payload	<p>The native service request data.</p> <p>Example:</p> <pre>{ "param1" : "value1", "param2" : 10 }</pre>
Native Response Headers	<p>Response header in the outgoing response from the native service to API Gateway.</p> <p>Example:</p> <pre>{ "Server": "Jetty(9.2.9.v20150224)", "Access-Control-Allow-Origin": "*", "Access-Control-Allow-Methods": "GET, POST, DELETE, PUT", "Connection": "close", "Date": "Fri, 07 Jun 2019 12:44:13 GMT", "Access-Control-Allow-Headers": "Content-Type, api_key, Authorization", "Content-Type": "application/json" }</pre>
Native Res Payload	<p>The native service response data.</p> <p>Example:</p> <pre>{ "id": 2, "category": { "id": 2, "name": "string" }, "name": "pysen", "photoUrls": ["string"], "tags": [{ "id": 0, "name": "string" }], "status": "available" }</pre>
Native URL	<p>URL of the native service.</p> <p>Example: http://petstore.swagger.io/v2/pet/2</p>
PartnerId	<p>The unique identifier for the partner that generated the audit record.</p> <p>Example: unknown</p>

Column	Description
Provider Round Trip Time	<p>Time in milliseconds required for API Gateway to invoke a native provider and receive a response. This time includes the overhead incurred by API Gateway. Overhead includes the time it takes for a provider to process a request and return a response, plus any network latency to or from the provider. Subtracting total time from provider time must give a rough indicator of the API Gateway overhead.</p> <p>Example: 1700</p>
QueryParameters	<p>This is applicable only for REST APIs. Query parameters present in the incoming REST request.</p> <p>Example: {"status":"available"}</p>
RequestHeaders	<p>Request header in the incoming request from the client.</p> <p>Example:</p> <pre>{ "Cache-Control": "max-age=0", "Accept": "text/plain,application/json; q=0.9,image/webp,image/apng,*/*; q=0.8", "Upgrade-Insecure-Requests": "1", "Connection": "keep-alive", "User-Agent": "Mozilla/5.0(Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181Safari/537.36", "Host": "mcdaso02:5555", "Accept-Encoding": "gzip,deflate", "Accept-Language": "en-US,en;q=0.9,ta;q=0.8", "Content-Type": "application/x-www-form-urlencoded" }</pre>
RequestPayload	<p>The API request payload data.</p> <p>Example: <RequestPayload></p>
ResponseHeaders	<p>Response header in the outgoing response.</p> <p>Example:</p> <pre>{ "Server": "Jetty(9.2.9.v20150224)", "Access-Control-Allow-Origin": "*", "Access-Control-Allow-Methods": "GET,POST,DELETE, PUT", "Connection": "close", "Date": "Fri, 30 Mar 2018 08:25:45 GMT", "Access-Control-Allow-Headers": "Content-Type,api_key, Authorization", "Content-Type": "application/xml" }</pre>

Column	Description
ResponsePayload	The API response payload data. Example: < ResponsePayload >
Source Gateway Node	Source API Gateway's IP address. Example: 10.0.75.1
Total Round Trip time	Time in milliseconds required to invoke the API provider. This time includes the overhead incurred by API Gateway. Overhead includes security overhead for encryption, decryption, and load-balance retries. Example: 1707
Gateway	Name of the API Gateway instance reporting the event. Example: API_Gateway_Instance
Request Status	Status of the API request. Possible values are: SUCCESS, FAILURE

Error Events

Column	Description
ApiUserVersion	The system-assigned version identifier for the API. Example: 1.0
Consumer	Name of the consumer associated with the API invocation. A consumer name is populated as unknown when API Gateway is unable to identify the consumer using a policy that is configured for the API. Example: SampleApplication
ConsumerId	The unique identifier for the consumer associated with the API invocation. Example: be8b27d6-8f79-4c6e-b06c-a628d2ba30c3
Consumer IP Address	IP address of the consumer associated with the API invocation. Example: 10.60.20.169
CorrelationID	The unique identifier that is automatically generated for every request coming to API Gateway and can be used to query the log.

Column	Description
	Example: MED38e9cfa4-2348-408b-9462-124b2181c1a6:656
Created Time	Date and time when the event was generated in API Gateway. Example: 2017-08-09 08:24:04 AM
Error Source	The source where the error occurred. Example: e1cc3c7b-495d-11e7-a5a6-88cf17308ba4
Error Description	Message that describes the error that occurred. Example: Resource / not found
Event Type	The type of event that occurred. Example: Error Event
Gateway	Name of the API Gateway instance reporting the event. Example: API_Gateway_Instance

Monitoring Events

Column	Description
PolicyName	Name of the policy that is enforced on the API. Example: Log Invocation
Alert Description	Text of the alert message sent to a configured destination when the performance conditions are violated. The alert message is specified in the policy definition of an API. Example: MSLA_ALERT MESSAGE
Alert Source	Name of the API Gateway policy that generated the alert message. Example: Monitorpolicy, EnforcePolicy-HardLimit
Alert Type	The type of alert generated for the event. Possible values are: Monitor, Sla
apiName	Name of the API in which the event occurred. Example: pet1
apiVersion	The system-assigned version identifier for the API. Example: 1.0

Column	Description
applicationId	The unique identifier for the application associated with the API invocation. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
applicationIp	IP address of the application associated with the API invocation. Example: 10.20.248.33
applicationName	Name of the application associated with the API invocation. An application name is populated as unknown when API Gateway is unable to identify the application using a security policy that is configured for the API. Example: SampleApplication
creationDate	Date and time when the event was generated in API Gateway. Example: 1501671101509
Monitored Attribute	The monitored attribute which has breached the configured SLA. Example: AVGRESPONSETIME GT 1.0, SUCCESSCOUNT EQ 3, REQUESTCOUNT GT 10
native_endpoint	The endpoint URL of the native API that is invoked. Example: http://petstore.swagger.io/v2/pet/55
operationName	Name of the API operation that is invoked. Example: Using a Calculator API, you can perform various operations such as addition, subtraction, multiplication, and division. When an addition operation is invoked in API Gateway, then the operation field name is populated as addInts .
sessionId	A string the API Gateway server generates to uniquely identify each session. This is either the IS session token or the automatically generated GUID if the token is missing from the message context. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
targetName	Name of the API Gateway instance reporting the event. Example: API_Gateway_Instance

Policy Violation Events

Column	Description
ApiUserVersion	The system-assigned version identifier for the API. Example: 1.0
Alert Source	Name of the API Gateway policy that generated the alert message. Example: Unknown-Policy
Alert Type	The type of alert generated for the event. Example: PolicyViolation
Alert Description	Text of the alert message sent to a configured destination when the performance conditions are violated. The alert message is specified in the policy definition of an API. Example: A violation of policy was detected : Unable to identify the application for the request
Consumer	Name of the consumer associated with the API invocation. A consumer name is populated as unknown when API Gateway is unable to identify the consumer using a policy that is configured for the API. Example: unknown
ConsumerId	The unique identifier for the consumer associated with the API invocation. Example: unknown
Consumer IP Address	IP address of the consumer associated with the API invocation. Example: 10.60.37.118
Created Time	Date and time when the event was generated in API Gateway. Example: 2017-08-09 08:25:52 AM
Event Type	The type of event that occurred. Example: Policy Violation Event
Gateway	Name of the API Gateway instance reporting the event. Example: API_Gateway_Instance

Lifecycle Events

Column	Description
TimeStamp	Date and time when the event was generated in API Gateway. Example: 2017-08-26 04:13:35 PM
Target	Name of the API Gateway instance reporting the event. Example: API_Gateway_Instance
LifeCycleStatus	Status of the API Gateway instance. Possible values are: STARTED or STOPPED
LifeCycleAlertDescription	The alert notification message for the lifecycle event. Example: Alert_Message

Performance Metrics

Column	Description
AVG_RESP_TIME	The average amount of time it took the API to complete each invocation in the current interval. Response time is measured from the moment API Gateway receives the request until the moment it returns the response to the caller. Example: 1376
FAULT_COUNT	The number of failed API invocations in the current interval. Example: 1
INCLUDE_FAULTS	Includes failed API invocations. Possible values are: true , false
INTERVAL_START	The starting date and time from which you want to examine metrics. Example: 02 Aug 2017 10:51:31 GMT
INTERVAL_STOP	The ending date and time until which you want to examine metrics. Example: 02 Aug 2017 10:52:31 GMT
MAX_RESP_TIME	The maximum amount of time (in milliseconds) it took for the API to complete an invocation in the current interval. Example: 1401

Column	Description
MIN_RESP_TIME	The minimum amount of time (in milliseconds) it took for the API to complete an invocation in the current interval. Example: 1352
OPERATION_NAME	Name of the API operation that is invoked. Example: /pet/{petId}
SERVICE_KEY	The Universally Unique Identifier (UUID) for the service in which the event occurred. This column is currently not used by APIs created in API Gateway. It is used to support the APIs that are migrated from CentraSite or Mediator to API Gateway.
SUCCESS_COUNT	The number of successful API invocations in the current interval. Example: 1
TARGET_NAME	Name of the API Gateway instance reporting the event. Example: API_Gateway_Instance
totalCount	The total number of API invocations (successful and unsuccessful) in the current interval. Example: 2

Elasticsearch

The runtime events and metrics payload generated by API Gateway at run-time is published to the configured Elasticsearch destination. The columns that make up the events and metrics data model for Elasticsearch are listed below:

Transactional Events

Column	Description
apild	The unique identifier for the API. Example: af70b2de-c9c5-4f40-94be-7d8622743e42
apiName	Name of the API in which the event occurred. Example: SampleAPI
apiVersion	The system-assigned version identifier for the API. Example: 1.0

Column	Description
applicationId	The unique identifier for the application associated with the API invocation. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
applicationIp	IP address of the application associated with the API invocation. Example: 10.60.37.42
applicationName	Name of the application associated with the API invocation. An application name is populated as unknown when API Gateway is unable to identify the application using a security policy that is configured for the API. Example: SampleApplication
cachedResponse	Indicates whether the response is sent to the client from the cached data present in API Gateway through the Service result caching policy or the response is received from Native service and sent to client. Possible values are: Cached, Not-Cached
correlationID	The unique identifier that is automatically generated for every request coming to API Gateway and can be used to query the log. Example: MED38e9cfa4-2348-408b-9462-124b2181c1a6:656
creationDate	Date and time when the event was generated in API Gateway. Example: 1501671101509
customFields	The custom fields an API Provider can provide to log a new field and value for a transaction event. Example: {"customfield": "customvalue"}
errorOrigin	The origin of error. Example: Nativeservice
eventType	The type of event that occurred. Example: Transactional
externalCalls	List the external calls from API Gateway. These external calls can be to a native service or service registry. Example: <pre>[{ "externalCallType": "SERVICE_REGISTRY_CALL", "externalURL": "http://service.registry.com",</pre>

Column	Description
	<pre> "callDuration":49, "callStartTime":1562244570486, "callEndTime":1562244570535, "responseCode": "200" }, { "externalCallType":"NATIVE_SERVICE_CALL", "externalURL": "https://petstore.swagger.io/v2/store/inventory", "callDuration":1285, "callStartTime":1562244569252, "callEndTime":1562244570537, "responseCode":"200" }] </pre>
httpMethod	<p>The HTTP method used to invoke the API.</p> <p>Example: GET</p>
isCallbackRequest	<p>Indicates whether the event is generated for a callback request.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> ■ true. This denotes that the event is generated for a callback request. ■ false. This denotes that the event is generated for a normal response.
messageType	<p>This is applicable only for WebSocket APIs. This indicates the type of a WebSocket message.</p> <p>Possible values are: binary, text</p>
nativeHttpMethod	<p>The HTTP method used to invoke the native service.</p> <p>Example: GET</p>
nativeRequestHeaders	<p>Request header in the incoming request from the API Gateway to native service.</p> <p>Example:</p> <pre> { "Authorization": "*****", "Accept": "*/*", "Authorization": "*****", "Accept": "*/*", "Cache-Control": "no-cache", "User-Agent": "PostmanRuntime/7.13.0", "Postman-Token": "381424fa-e3b3-4058-8df9-4abf9d72c899", "postmanHeader": "hello", "accept-encoding": "gzip, deflate", "Content-Type": "application/x-www-form-urlencoded" } </pre>

Column	Description
	}
nativeRequestPayload	The native service request data. Example: <pre>{ "param1" : "value1", "param2" : 10 }</pre>
nativeResponseHeaders	Response header in the outgoing response from the native service to API Gateway. Example: <pre>{ "Server": "Jetty(9.2.9.v20150224)", "Access-Control-Allow-Origin": "*", "Access-Control-Allow-Methods": "GET, POST, DELETE, PUT", "Connection": "close", "Date": "Fri, 07 Jun 2019 12:44:13 GMT", "Access-Control-Allow-Headers": "Content-Type, api_key, Authorization", "Content-Type": "application/json" }</pre>
nativeResponsePayload	The native service response data. Example: <pre>{ "id": 2, "category": { "id": 2, "name": "string" }, "name": "pysen", "photoUrls": ["string"], "tags": [{ "id": 0, "name": "string" }], "status": "available" }</pre>
nativeURL	URL of the native service. Example: http://petstore.swagger.io/v2/pet/2
operationName	Name of the API operation that is invoked.

Column	Description
	Example: /pet/{petId}
origin	This is applicable only for WebSocket APIs. The origin of the request. Possible values are: client, server
packageId	The unique identifier for the API package. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
packageName	Name of the API package. Example: Travel Package
planId	The unique identifier for the API plan. Example: d0f84954-9732-11e5-b9f4-f159eafe47b2
planName	Name of the API plan. Example: Gold Plan
providerTime	Time in milliseconds required for API Gateway to invoke a native provider and receive a response. This time includes the overhead incurred by API Gateway. Overhead includes the time it takes for a provider to process a request and return a response, plus any network latency to or from the provider. Subtracting total time from provider time must give a rough indicator of the API Gateway overhead. Example: 1367
queryParameters	This is applicable only for REST APIs. Query parameters present in the incoming REST request. Example: {"status":"available"}
reqPayload	The API request payload data. Example: <RequestPayload>
requestHeaders	Request header in the incoming request from the client. Example: <pre>{ "Cache-Control": "max-age=0", "Accept": "text/plain, application/json; q=0.9, image/webp, image/apng, */*; q=0.8", "Upgrade-Insecure-Requests": "1", "Connection": "keep-alive", "User-Agent": "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36", }</pre>

Column	Description
	<pre>"Host":"mcdaso02:5555", "Accept-Encoding":"gzip,deflate", "Accept-Language":"en-US,en;q=0.9,ta;q=0.8", "Content-Type":"application/x-www-form-urlencoded" }</pre>
resPayload	<p>The API response payload data.</p> <p>Example: <ResponsePayload></p>
responseCode	<p>The HTTP response status code that indicates success or failure of the requested operation.</p> <p>Example: 404</p>
responseHeaders	<p>Response header in the outgoing response.</p> <p>Example:</p> <pre>{ "Server":"Jetty(9.2.9.v20150224)", "Access-Control-Allow-Origin":"*", "Access-Control-Allow-Methods":"GET,POST,DELETE, PUT", "Connection":"close", "Date":"Fri, 30 Mar 2018 08:25:45 GMT", "Access-Control-Allow-Headers":"Content-Type,api_key, Authorization", "Content-Type":"application/xml" }</pre>
serverID	<p>The API Gateway server on which the transaction event occurred.</p> <p>This column is currently not used. It appears as NULL or as an empty string.</p> <p>Example: SampleHost:80</p>
sourceGatewayDetails	<p>Details of events generated only from Microgateway. The details include Microgateway Id, Source Gateway Host, Source Gateway Port, Source Gateway Version, Microgateway pool.</p>
sourceGatewayNode	<p>Source API Gateway's IP address.</p> <p>Example: 10.0.75.1</p>
sourceGateway	<p>Source of event generation.</p> <p>Possible values: APIGateway or Microgateway.</p>
status	<p>Status of the API request.</p> <p>Possible values are: SUCCESS, FAILURE</p>

Column	Description
totalContentSize	The total combined size of request and response payloads in bytes. Example: 51
totalTime	Time in milliseconds required to invoke the API provider. This time includes the overhead incurred by API Gateway. Overhead includes security overhead for encryption, decryption, and load-balance retries. Example: 1401

Error Events

Column	Description
apiId	The unique identifier for the API. Example: af70b2de-c9c5-4f40-94be-7d8622743e42
apiName	Name of the API in which the event occurred. Example: SampleAPI
apiVersion	The system-assigned version identifier for the API. Example: 1.0
applicationId	The unique identifier for the application associated with the API invocation. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
applicationIp	IP address of the application associated with the API invocation. Example: 10.60.37.42
applicationName	Name of the application associated with the API invocation. An application name is populated as unknown when API Gateway is unable to identify the application using a security policy that is configured for the API. Example: SampleApplication
correlationID	The unique identifier that is automatically generated for every request coming to API Gateway and can be used to query the log. Example: MED38e9cfa4-2348-408b-9462-124b2181c1a6:656
creationDate	Date and time when the event was generated in API Gateway. Example: 1501671101509

Column	Description
errorDesc	Message that describes the error that occurred. Example: Native service provider error. Code : 404
eventSource	The source where the event occurred. Example: API_Gateway_Instance
eventType	The type of event that occurred. Example: Error
httpMethod	The HTTP method used to invoke the API. Example: GET
operationName	Name of the API operation that is invoked. Example: /pet/{petId}
responseCode	The HTTP response status code that indicates success or failure of the requested operation. Example: 404
sourceGatewayDetails	Details of events generated only from Microgateway. The details include Microgateway Id, Source Gateway Host, Source Gateway Port, Source Gateway Version, Microgateway pool.
sourceGatewayNode	Source API Gateway's IP address. Example: 10.0.75.1
sourceGateway	Source of event generation. Possible values: APIGateway or Microgateway .

Monitoring Events

Column	Description
alertDesc	Text of the alert message sent to a configured destination when the performance conditions are violated. The alert message is specified in the policy definition of an API. Example: EnforcePolicy-HardLimit
alertSource	Name of the API Gateway policy that generated the alert message. Example: Monitorpolicy

Column	Description
alertType	The type of alert generated for the event. Example: Monitor
apiId	The unique identifier for the API. Example: af70b2de-c9c5-4f40-94be-7d8622743e42
apiName	Name of the API in which the event occurred. Example: SampleAPI
apiVersion	The system-assigned version identifier for the API. Example: 1.0
applicationId	The unique identifier for the application associated with the API invocation. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
applicationIp	IP address of the application associated with the API invocation. Example: 10.60.37.42
applicationName	Name of the application associated with the API invocation. An application name is populated as unknown when API Gateway is unable to identify the application using a security policy that is configured for the API. Example: SampleApplication
creationDate	Date and time when the event was generated in API Gateway. Example: 1501671101509
eventSource	The source where the event occurred. Example: API_Gateway_Instance
eventType	The type of event that occurred. Example: Monitor
httpMethod	The HTTP method used to request the API access. Example: GET
monitorAttr	The monitored attribute which has breached the configured SLA. Example: AVGRESPONSETIME GT 1.0, SUCCESSCOUNT EQ 3, REQUESTCOUNT GT 10

Column	Description
operationName	Name of the API operation that is invoked. Example: /pet/{petId}
responseCode	The HTTP response status code that indicates success or failure of the requested operation. Example: 200
sourceGateway	Source of event generation. Possible values: APIGateway or Microgateway.
sourceGatewayDetails	Details of events generated only from Microgateway. The details include Microgateway Id, Source Gateway Host, Source Gateway Port, Source Gateway Version, Microgateway pool.

Policy Violation Events

Column	Description
alertDesc	Text of the alert message sent to a configured destination when the performance conditions are violated. The alert message is specified in the policy definition of an API. Example: A violation was detected for policy (Unknown-Policyuser): application could not be identified. Anonymous access is not allowed for this service!
alertSource	Name of the API Gateway policy that generated the alert message. Example: Unknown-Policy
alertType	The type of alert generated for the event. Example: PolicyViolation
apild	The unique identifier for the API. Example: af70b2de-c9c5-4f40-94be-7d8622743e42
apiName	Name of the API in which the event occurred. Example: SampleAPI
apiVersion	The system-assigned version identifier for the API. Example: 1.0
applicationId	The unique identifier for the application associated with the API invocation.

Column	Description
	Example: 9434e90d-65c3-4e37-8ccb-595b8df3e645
applicationIp	IP address of the application associated with the API invocation. Example: 10.60.37.42
applicationName	Name of the application associated with the API invocation. An application name is populated as unknown when API Gateway is unable to identify the application using a security policy that is configured for the API. Example: SampleApplication
creationDate	Date and time when the event was generated in API Gateway. Example: 1501671101509
eventSource	The source where the event occurred. Example: API_Gateway_Instance
eventType	The type of event that occurred. Example: PolicyViolation
httpMethod	The HTTP method used to request the API access. Example: GET
operationName	Name of the API operation that is invoked. Example: /pet/{petId}
responseCode	The HTTP response status code that indicates success or failure of the requested operation. Example: 503
sourceGateway	Source of event generation. Possible values: APIGateway or Microgateway.
sourceGatewayDetails	Details of events generated only from Microgateway. The details include Microgateway Id, Source Gateway Host, Source Gateway Port, Source Gateway Version, Microgateway pool.

Performance Metrics

Column	Description
apiId	The unique identifier for the API. Example: af70b2de-c9c5-4f40-94be-7d8622743e42
apiName	Name of the API in which the event occurred. Example: SampleAPI
apiVersion	The system-assigned version identifier for the API. Example: 1.0
availability	The percentage of time that an API was available during the current interval. A value of 100 indicates that the API was always available. If invocations fail due to policy violations, this parameter could still be as high as 100. Example: 100.0
avgResponseTime	The average amount of time it took the API to complete each invocation in the current interval. Response time is measured from the moment API Gateway receives the request until the moment it returns the response to the caller. Example: 1376
creationDate	Date and time when the event was generated in API Gateway. Example: 1501671101509
eventType	The type of event that occurred. Example: PerformanceData
faultCount	The number of failed API invocations in the current interval. Example: 1
includeFaults	Includes failed API invocations. Possible values are: true, false
intervalStart	The starting date and time from which you want to examine metrics. Example: 02 Aug 2017 10:51:31 GMT
intervalStop	The ending date and time until which you want to examine metrics. Example: 02 Aug 2017 10:52:31 GMT

Column	Description
maxResponseTime	The maximum amount of time (in milliseconds) it took for the API to complete an invocation in the current interval. Example: 1401
minResponseTime	The minimum amount of time (in milliseconds) it took for the API to complete an invocation in the current interval. Example: 1352
sourceGatewayDetails	Details of events generated only from Microgateway. The details include Microgateway Id, Source Gateway Host, Source Gateway Port, Source Gateway Version, Microgateway pool.
sourceGateway	Source of event generation. Possible values: APIGateway or Microgateway .
operationName	Name of the API operation that is invoked. Example: /pet/{petId}
successCount	The number of successful API invocations in the current interval. Example: 1
totalCount	The total number of API invocations (successful and unsuccessful) in the current interval. Example: 2

Email

The runtime events and metrics payload generated by API Gateway at run-time is published to the configured Email destination. The columns that make up the events and metrics data model for Email are listed below:

Transactional Events

Column	Description
apiName	Name of the API in which the event occurred. Example: SampleAPI
consumerId	The unique identifier for the consumer associated with the API invocation. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2

Column	Description
consumerName	<p>Name of the consumer associated with the API invocation.</p> <p>A consumer name is populated as unknown when API Gateway is unable to identify the consumer using a security policy that is configured for the API.</p> <p>Example: SampleApplication</p>
correlationID	<p>The unique identifier that is automatically generated for every request coming to API Gateway and can be used to query the log.</p> <p>Example: MED38e9cfa4-2348-408b-9462-124b2181c1a6:656</p>
Description	<p>Message that describes the date and time the API was invoked and the application associated with the API invocation.</p> <p>Example: Invoked at 4/24/18 1:50 PM Consumer Name: Unknown Consumer ID: Unknown</p>
ErrorOrigin	<p>The origin of error.</p> <p>Example: Nativeservicve</p>
External Calls	<p>List the external calls from API Gateway. These external calls can be to a native service or service registry.</p> <p>Example:</p> <pre>[{ "externalCallType": "SERVICE_REGISTRY_CALL", "externalURL": "http://service.registry.com", "callDuration": 49, "callStartTime": 1562244570486, "callEndTime": 1562244570535, "responseCode": "200" }, { "externalCallType": "NATIVE_SERVICE_CALL", "externalURL": "https://petstore.swagger.io/v2/store/inventory", "callDuration": 1285, "callStartTime": 1562244569252, "callEndTime": 1562244570537, "responseCode": "200" }]</pre>
Native Endpoint	<p>The endpoint URL of the native API being invoked.</p> <p>Example: http://petstore.swagger.io/v2/pet/55</p>
Native HTTP Method	<p>The HTTP method used to invoke the native service.</p> <p>Example: GET</p>

Column	Description
Native Request Headers	Request header in the incoming request from the API Gateway to native service. Example: <pre>{ "Authorization": "*****", "Accept": "*/*", "Authorization": "*****", "Accept": "*/*", "Cache-Control": "no-cache", "User-Agent": "PostmanRuntime/7.13.0", "Postman-Token": "381424fa-e3b3-4058-8df9-4abf9d72c899", "postmanHeader": "hello", "accept-encoding": "gzip, deflate", "Content-Type": "application/x-www-form-urlencoded" }</pre>
Native Request Payload	The native service request data. Example: <pre>{ "param1" : "value1", "param2" : 10 }</pre>
Native Response Headers	Response header in the outgoing response from the native service to API Gateway. Example: <pre>{ "Server": "Jetty(9.2.9.v20150224)", "Access-Control-Allow-Origin": "*", "Access-Control-Allow-Methods": "GET, POST, DELETE, PUT", "Connection": "close", "Date": "Fri, 07 Jun 2019 12:44:13 GMT", "Access-Control-Allow-Headers": "Content-Type, api_key, Authorization", "Content-Type": "application/json" }</pre>
Native Response Payload	The native service response data. Example: <pre>{ "id": 2, "category": { "id": 2, "name": "string" } }</pre>

Column	Description
	<pre>}, "name": "pysen", "photoUrls": ["string"], "tags": [{ "id": 0, "name": "string" }], "status": "available" }</pre>
Native URL	URL of the native service. Example: http://petstore.swagger.io/v2/pet/2
Operation/Resource Name	Name of the operation or resource that is being invoked on the API. Example: /pet/{petId}
Policy Action Name	Name of the runtime policy that is enforced on the API. Example: Log Invocation
Source Gateway Node	Source API Gateway's IP address. Example: 10.0.75.1
Status	Status of the API invocation. Possible values are: SUCCESS, FAILURE
Version	The system-assigned version identifier for the API. Example: 1.0

Monitoring Events

Column	Description
alertDesc	Text of the alert message sent to a configured destination when the performance conditions are violated. The alert message is specified in the policy definition of an API. Example: EnforcePolicy-HardLimit
alertSource	The type of alert generated for the event. Example: Monitor
alertType	Text of the alert message sent to a configured destination when the performance conditions are violated. The alert message is specified in the policy definition of an API.

Column	Description
	Example: Test
apiGWHostName	Name of the host which serves the request. Example: SAG-HS09MG2
apiName	Name of the API in which the event occurred. Example: SampleAPI
apiVersion	The system-assigned version identifier for the API. Example: 1.0
applicationId	The unique identifier for the application associated with the API invocation. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2
applicationName	Name of the application associated with the API invocation. An application name is populated as unknown when API Gateway is unable to identify the application using a security policy that is configured for the API. Example: SampleApplication
monitorAttr	The monitored attribute which has breached the configured SLA. Example: AVGRESPONSETIME GT 1.0, SUCCESSCOUNT EQ 3, REQUESTCOUNT GT 10
Native Endpoint	The endpoint URL of the native API that is being invoked. Example: http://petstore.swagger.io/v2/pet/55

JDBC

The events and metrics payload generated by API Gateway at run-time is published to the configured JDBC destination. The columns that make up the events and metrics data model for JDBC are listed below:

API Gateway supports three types of database, Oracle, DB2 and MSSQL. Based on the database selected, API Gateway publishes the events and metrics payload to the JDBC destination.

Transactional Events

Column Description	Oracle	DB2	MSSQL
API_ID	Varchar(256)	Varchar(256)	NVarchar(256)

Column Description	Oracle	DB2	MSSQL
The unique identifier for the API. Example: c0f84954-9732-11e5-b9f4-f159eafe47b1			
API_NAME	Varchar(256)	Varchar(256)	NVarchar(256)
Name of the API in which the event occurred. Example: SampleAPI			
API_VERSION	Varchar(256)	Varchar(256)	NVarchar(256)
The system-assigned version identifier for the API. Example: 1.0			
APPLICATION_ID	Varchar(256)	Varchar(256)	NVarchar(256)
The unique identifier for the application associated with the API invocation. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2			
APPLICATION_IP	Varchar(64)	Varchar(64)	NVarchar(64)
IP address of the application associated with the API invocation. Example: 10.20.248.33			
APPLICATION_NAME	Varchar(128)	Varchar(128)	NVarchar(128)
Name of the application associated with the API invocation. An application name is populated as unknown when API Gateway is unable to identify the application using a security policy that is configured for the API.			
Example: SampleApplication			
AUDITTIMESTAMP	TIMESTAMP	TIMESTAMP	DATETIME
Date and time when the event was written to the log.			
BINDING_NAME	Varchar(256)	Varchar(256)	NVarchar(256)
Name of the binding that identifies a specific access URI. This column is currently not used. It appears as NULL or as an empty string.			

Column Description	Oracle	DB2	MSSQL
CACHED_RESPONSE	Varchar(12)	Varchar(12)	NVarchar(12)
Indicates whether the response is sent to the client from the cached data present in API Gateway through the Service result caching policy or the response is received from Native service and sent to client.	Possible values are: Cached, Not-Cached		
CONSUMER_ID	Varchar(256)	Varchar(256)	NVarchar(256)
The unique identifier for the consumer associated with the API invocation.	Example: c0f84954-9732-11e5-b9f4-f159eafe47b2		
CONSUMER_IP	Varchar(64)	Varchar(64)	NVarchar(64)
IP address of the consumer associated with the API invocation.	Example: 10.20.248.33		
CONSUMER_NAME	Varchar(128)	Varchar(128)	NVarchar(128)
Name of the consumer associated with the API invocation.	A consumer name is populated as unknown when API Gateway is unable to identify the consumer using a policy that is configured for the API.		
Example: SampleApplication			
CONTEXTID	CHAR(36)	CHAR(36)	NCHAR(36)
The unique identifier for the current context information API Gateway uses to connect related entries from different logs.	This column is currently not used. It appears as NULL or as an empty string.		
Example: 81546147-41a8-4998-8150-02ba67bb08c2			
CORRELATIONID	Varchar(256)	Varchar(256)	NVarchar(256)
The unique identifier that is automatically generated for every request coming to API Gateway and can be used to query the log.			

Column Description	Oracle	DB2	MSSQL
Example: MED38e9cfa4-2348-408b-9462-124b2181c1a6:656			
CUSTOM_FIELD	BLOB	BLOB	IMAGE
The custom fields an API Provider can provide to log a new field and value for a transaction event.			
Example: {"customfield":"customvalue"}			
ERROR_ORIGIN	Varchar(256)	Varchar(256)	NVarchar(256)
The origin of error.			
EVENT_CREATE_TS	TIMESTAMP	TIMESTAMP	DATETIME
Date and time when the event was generated in API Gateway.			
This is not the time the database performed its insert (for example, this is calculated by the Mediator policy engine and not a database function).			
EVENT_SOURCE			
EVENT_TYPE	Varchar(256)	Varchar(256)	NVarchar(256)
The type of event that occurred.			
Example: Transactional			
EVENT_USERNAME	Varchar(256)	Varchar(256)	NVarchar(256)
Name of the user on API Gateway that invoked the API.			
EXTERNAL_CALLS	BLOB	BLOB	IMAGE
List the external calls from API Gateway. These external calls can be to a native service or service registry.			
Example:			
<pre>[{ "externalCallType": "SERVICE_REGISTRY_CALL", "externalURL": "http://service.registry.com", "callDuration": 49, "callStartTime": 1562244570486, "callEndTime": 1562244570535, "responseCode": "200" }]</pre>			

Column Description	Oracle	DB2	MSSQL
<pre>}, { "externalCallType": "NATIVE_SERVICE_CALL", "externalURL": "https://petstore.swagger.io/v2/store /inventory", "callDuration":1285, "callStartTime":1562244569252, "callEndTime":1562244570537, "responseCode":"200" }]</pre>			
HTTP_METHOD	Varchar(8)	Varchar(8)	NVarchar(8)
The HTTP method used to invoke the API.			
Example: GET			
INSERTTIMESTAMP	TIMESTAMP	TIMESTAMP	DATETIME
Date and time when the event was generated in API Gateway.			
MSGID	CHAR(36)	CHAR(36)	NCHAR(36)
The ID assigned to the message by the API provider.			
This column is currently not used.			
NATIVE_ENDPOINT	Varchar(4000)	Varchar(4000)	NVarchar(4000)
The endpoint URL of the native API that is invoked.			
Example: http://petstore.swagger.io/v2/pet/55			
NATIVE_HTTP_METHOD	Varchar2(20)	Varchar(20)	Varchar(20)
The HTTP method used to invoke the native service.			
Example: GET			
NATIVE_REQUEST_HEADERS	BLOB	BLOB	IMAGE
Request header in the incoming request from the API Gateway to native service.			
Example:			
<pre>{ "Authorization": "*****", "Accept": "*/*", "Authorization": "*****", "Accept": "*/*", "Cache-Control": "no-cache",</pre>			

Column Description	Oracle	DB2	MSSQL
<pre>"User-Agent": "PostmanRuntime/7.13.0", "Postman-Token": "381424fa-e3b3-4058-8df9-4abf9d7c899", "postmanHeader": "hello", "accept-encoding": "gzip, deflate", "Content-Type": "application/x-www-form-urlencoded" }</pre>			
NATIVE_REQUEST_PAYLOAD	CLOB	CLOB	Varchar(max)
The native service request data.			
Example:			
<pre>{ "param1" : "value1", "param2" : 10 }</pre>			
NATIVE_RESPONSE_HEADERS	BLOB	BLOB	IMAGE
Response header in the outgoing response from the native service to API Gateway.			
Example:			
<pre>{ "Server":"Jetty(9.2.9.v20150224)", "Access-Control-Allow-Origin":"*", "Access-Control-Allow-Methods": "GET, POST, DELETE, PUT", "Connection":"close", "Date": "Fri, 07 Jun 2019 12:44:13 GMT", "Access-Control-Allow-Headers": "Content-Type, api_key,Authorization", "Content-Type": "application/json" }</pre>			
NATIVE_RESPONSE_PAYLOAD	CLOB	CLOB	Varchar(max)
The native service response data.			
Example:			
<pre>{ "id":2, "category": { "id":2, "name":"string" }, "name":"pysen", "photoUrls":["string"], "tags":</pre>			

Column Description	Oracle	DB2	MSSQL
[{ "id":0, "name":"string" }], "status":"available" }			
NATIVE_URL	CLOB	CLOB	Varchar(max)
URL of the native service.			
Example: http://petstore.swagger.io/v2/pet/2			
OPERATION_NAME	Varchar(256)	Varchar(256)	NVarchar(256)
Name of the API operation or resource that is invoked.			
Example: /pet/{petId}			
ORG_KEY	Varchar(128)	Varchar(128)	NVarchar(128)
The Globally Unique Identifier (GUID) for an organization.			
This column is currently not used by APIs created in API Gateway. It is used to support the APIs that are migrated from CentraSite or Mediatorrr to API Gateway.			
PACKAGE_ID	Varchar(256)	Varchar(256)	NVarchar(256)
The unique identifier for the API package.			
Example: c0f84954-9732-11e5-b9f4-f159eafe47b2			
PACKAGE_NAME	Varchar(256)	Varchar(256)	NVarchar(256)
Name of the API package.			
Example: Travel Package			
PARENTCONTEXTID	CHAR(36)	CHAR(36)	NCHAR(36)
The unique identifier for the parent context information API Gateway uses to connect related entries from different logs.			
This column is currently not used. It appears as NULL or as an empty string.			
PARTNER_ID	Varchar(256)	Varchar(256)	NVarchar(256)

Column Description	Oracle	DB2	MSSQL
The unique identifier for the partner that generated the audit record.			
This column is currently not used. It appears as NULL or as an empty string.			
PLAN_ID	Varchar(256)	Varchar(256)	NVarchar(256)
The unique identifier for the API plan.			
Example: d0f84954-9732-11e5-b9f4-f159eafe47b2			
PLAN_NAME	Varchar(256)	Varchar(256)	NVarchar(256)
Name of the API plan.			
Example: Gold Plan			
PROVIDER_TIME	NUMERIC	INT	INTEGER
Time in milliseconds required for API Gateway to invoke a native provider and receive a response. This time includes the overhead incurred by API Gateway. Overhead includes the time it takes for a provider to process a request and return a response, plus any network latency to or from the provider. Subtracting total time from provider time must give a rough indicator of the API Gateway overhead.			
Example: 1367			
QUERY_PARAMS	BLOB	BLOB	IMAGE
This is applicable only for REST APIs. Query parameters present in the incoming REST request.			
Example: {"status": "available"}			
REQUEST	BLOB	BLOB	IMAGE
The API request payload data.			
Example: <RequestPayload>			
REQUEST_HEADERS	BLOB	BLOB	IMAGE
Request header in the incoming request from the client.			
RESPONSE	BLOB	BLOB	IMAGE
The API response payload data.			

Column Description	Oracle	DB2	MSSQL
Example: < ResponsePayload >			
RESPONSE_CODE	Numeric	INT	INTEGER
The HTTP response status code that indicates success or failure of the requested operation.			
Example: 200			
RESPONSE_HEADERS	BLOB	BLOB	IMAGE
Response header in the outgoing response.			
ROOTCONTEXTID	CHAR(36)	CHAR(36)	NCHAR(36)
The unique identifier for the root context information API Gateway uses to connect related entries from different logs.			
This column is currently not used. It appears as NULL or as an empty string.			
Example: 81546147-41a8-4998-8150-02ba67bb08c2			
SERVID	Varchar(450)	Varchar(450)	NVarchar(450)
The API Gateway server on which the transaction event occurred.			
This column is currently not used. It appears as NULL or as an empty string.			
SERVICE_KEY	Varchar(128)	Varchar(128)	NVarchar(128)
The Universally Unique Identifier (UUID) for the service in which the event occurred.			
This column is currently not used by APIs created in API Gateway. It is used to support the APIs that are migrated from CentraSite or Mediator to API Gateway.			
SERVICE_NAME	Varchar(256)	Varchar(256)	NVarchar(256)
Name of the service in which the event occurred.			
Example: SampleAPI			
SERVICE_VERSION	Varchar(256)	Varchar(256)	NVarchar(256)
The system-assigned version identifier for the service.			

Column Description	Oracle	DB2	MSSQL
Example: 1.0			
SESSION_ID	Varchar(128)	Varchar(128)	NVarchar(128)
A string the API Gateway server generates to uniquely identify each session. This is either the IS session token or the automatically generated GUID if the token is missing from the message context.			
Example: c0f84954-9732-11e5-b9f4-f159eafe47b2			
SOURCE_GATEWAY_NODE	Varchar2(256)	Varchar(256)	Varchar(256)
Source API Gateway's IP address.			
Example: 10.0.75.1			
STATUS	Varchar(128)	Varchar(128)	NVarchar(128)
Status of the API request.			
Possible values are: SUCCESS, FAILURE			
TARGET_NAME	Varchar(32)	Varchar(32)	NVarchar(32)
Name of the API Gateway instance reporting the event.			
Example: API_Gateway_Instance			
TOTAL_DATA_SIZE	NUMERIC	INT	INTEGER
The total combined size of request and response payloads in bytes.			
Example: 100			
TOTAL_TIME	NUMERIC	INT	INTEGER
Time in milliseconds required to invoke the API provider. This time includes the overhead incurred by API Gateway. Overhead includes security overhead for encryption, decryption, and load-balance retries.			
Example: 120			
USER_AGENT	Varchar2(256)	Varchar(256)	NVarchar(256)
Name of the client used to invoke the API.			
Example: Postman			

Error Events

Column Description	Oracle	DB2	MSSQL
API_ID	Varchar(256)	Varchar(256)	NVarchar(256)
The unique identifier for the API.			
Example: c0f84954-9732-11e5-b9f4-f159eafe47b1			
API_NAME	Varchar(256)	Varchar(256)	NVarchar(256)
Name of the API in which the event occurred.			
Example: SampleAPI			
API_VERSION	Varchar(256)	Varchar(256)	NVarchar(256)
The system-assigned version identifier for the API.			
Example: 1.0			
APPLICATION_ID	Varchar(256)	Varchar(256)	NVarchar(256)
The unique identifier for the application associated with the API invocation.			
Example: c0f84954-9732-11e5-b9f4-f159eafe47b2			
APPLICATION_IP	Varchar(64)	Varchar(64)	NVarchar(64)
IP address of the application associated with the API invocation.			
Example: 10.20.248.33			
APPLICATION_NAME	Varchar(128)	Varchar(128)	NVarchsar(128)
Name of the application associated with the API invocation.			
An application name is populated as unknown when API Gateway is unable to identify the application using a security policy that is configured for the API.			
Example: SampleApplication			
BINDING_NAME	Varchar(256)	Varchar(256)	Varchar(256)
Name of the binding that identifies a specific access URI.			
This column is currently not used. It appears as NULL or as an empty string.			

Column Description	Oracle	DB2	MSSQL
CONSUMER_ID	Varchar(256)	Varchar(256)	NVarchar(256)
The unique identifier for the consumer associated with the API invocation.			
Example: c0f84954-9732-11e5-b9f4-f159eafe47b2			
CONSUMER_IP	Varchar(64)	Varchar(64)	NVarchar(64)
IP address of the consumer associated with the API invocation.			
Example: 10.20.248.33			
CONSUMER_NAME	Varchar(128)	Varchar(128)	NVarchar(128)
Name of the consumer associated with the API invocation.			
A consumer name is populated as unknown when API Gateway is unable to identify the consumer using a policy that is configured for the API.			
Example: SampleApplication			
CORRELATION_ID	Varchar2(256)	Varchar(256)	Varchar(256)
The unique identifier that is automatically generated for every request coming to API Gateway and can be used to query the log.			
Example: MED38e9cfa4-2348-408b-9462-124b2181c1a6:656			
ERROR_SOURCE	Varchar(256)	Varchar(256)	NVarchar(256)
The source where the error occurred.			
Example: e1cc3c7b-495d-11e7-a5a6-88cf17308ba4			
ERROR_DESC	Varchar(4000)	Varchar(4000)	NVarchar(4000)
Message that describes the error that occurred.			
Example: Resource / not found			
EVENT_CREATE_TS	TIMESTAMP	TIMESTAMP	DATETIME
Date and time when the event was generated in API Gateway.			

Column Description	Oracle	DB2	MSSQL
This is not the time the database performed its insert (for example, this is calculated by the Mediator policy engine and not a database function).			
EVENT_SOURCE	Varchar(80)	Varchar(80)	NVarchar(80)
The source where the event occurred.			
EVENT_TYPE	Varchar(256)	Varchar(256)	NVarchar(256)
The type of event that occurred.			
Example: Error Event			
EVENT_USERNAME	Varchar(80)	Varchar(80)	NVarchar(80)
Name of the user on API Gateway that invoked the API.			
HTTP_METHOD	Varchar(8)	Varchar(8)	NVarchar(8)
The HTTP method used to invoke the API.			
Example: GET			
NATIVE_ENDPOINT	Varchar(4000)	Varchar(4000)	NVarchar(4000)
The endpoint URL of the native API that is invoked.			
Example: http://petstore.swagger.io/v2/pet/55			
OPERATION_NAME	Varchar(256)	Varchar(256)	NVarchar(256)
Name of the API operation or resource that is invoked.			
Example: /pet/{petId}			
ORG_KEY	Varchar(128)	Varchar(128)	NVarchar(128)
The Globally Unique Identifier (GUID) for an organization.			
This column is currently not used by APIs created in API Gateway. It is used to support the APIs that are migrated from CentraSite or Mediator to API Gateway.			
RESPONSE_CODE	Numeric	INT	INTEGER
The HTTP response status code that indicates success or failure of the requested operation.			
Example: 200			

Column Description	Oracle	DB2	MSSQL
SERVICE_KEY The Universally Unique Identifier (UUID) for the service in which the event occurred. This column is currently not used by APIs created in API Gateway. It is used to support the APIs that are migrated from CentraSite or Mediator to API Gateway.	Varchar(128)	Varchar(128)	NVarchar(128)
SERVICE_NAME Name of the service in which the event occurred. Example: SampleAPI	Varchar(256)	Varchar(256)	NVarchar(256)
SERVICE_VERSION The system-assigned version identifier for the service. Example: 1.0	Varchar(256)	Varchar(256)	NVarchar(256)
SESSION_ID A string the API Gateway server generates to uniquely identify each session. This is either the IS session token or the automatically generated GUID if the token is missing from the message context. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2	Varchar(128)	Varchar(128)	NVarchar(128)
TARGET_NAME Name of the API Gateway instance reporting the event. Example: API_Gateway_Instance	Varchar(64)	Varchar(64)	NVarchar(64)
USER_AGENT Name of the client used to invoke the API. Example: Postman	Varchar2(256)	Varchar(256)	NVarchar(256)

Monitoring Events

Column Description	Oracle	DB2	MSSQL
USER_AGENT Name of the client used to invoke the API.	Varchar2(256)	Varchar(256)	NVarchar(256)

Column Description	Oracle	DB2	MSSQL
Example: Postman			
ALERT_DESC	Varchar(256)	Varchar(256)	NVarchar(256)
Text of the alert message sent to a configured destination when the performance conditions are violated. The alert message is specified in the policy definition of an API.			
Example: EnforcePolicy-HardLimit			
ALERT_SOURCE	Varchar(256)	Varchar(256)	NVarchar(256)
Name of the API Gateway policy that generated the alert message.			
Example: MonitorPolicy			
ALERT_TYPE	Varchar(128)	Varchar(128)	NVarchar(128)
The type of alert generated for the event.			
Possible values are: Monitor, SLA			
API_ID	Varchar(256)	Varchar(256)	NVarchar(256)
The unique identifier for the API.			
Example: c0f84954-9732-11e5-b9f4-f159eafe47b1			
API_NAME	Varchar(256)	Varchar(256)	NVarchar(256)
Name of the API in which the event occurred.			
Example: SampleAPI			
API_VERSION	Varchar(256)	Varchar(256)	NVarchar(256)
The system-assigned version identifier for the API.			
Example: 1.0			
APPLICATION_ID	Varchar(256)	Varchar(256)	NVarchar(256)
The unique identifier for the application associated with the API invocation.			
Example: c0f84954-9732-11e5-b9f4-f159eafe47b2			
APPLICATION_IP	Varchar(64)	Varchar(64)	NVarchar(64)
IP address of the application associated with the API invocation.			

Column Description	Oracle	DB2	MSSQL
Example: 10.20.248.33			
APPLICATION_NAME	Varchar(128)	Varchar(128)	NVarchsar(128)
Name of the application associated with the API invocation.			
An application name is populated as unknown when API Gateway is unable to identify the application using a security policy that is configured for the API.			
Example: SampleApplication			
BINDING_NAME	Varchar(256)	Varchar(256)	NVarchar(256)
Name of the binding that identifies a specific access URI.			
This column is currently not used. It appears as NULL or as an empty string.			
creationDate	Varchar2(256)	Varchar(256)	NVarchar(256)
Date and time when the event was generated in API Gateway.			
Example: 1501671101509			
EVENT_TYPE	Varchar(256)	Varchar(256)	NVarchar(256)
The type of event that occurred.			
Example: Monitor Event			
EVENT_USERNAME	Varchar(80)	Varchar(80)	NVarchar(80)
Name of the user on API Gateway that invoked the API.			
MONITOR_ATTR	Varchar(256)	Varchar(256)	NVarchar(256)
The monitored attribute which has breached the configured SLA.			
Example: AVGRESPONSETIME GT 1.0, SUCCESSCOUNT EQ 3, REQUESTCOUNT GT 10			
NATIVE_ENDPOINT	Varchar(4000)	Varchar(4000)	NVarchar(4000)
The endpoint URL of the native API that is invoked.			
Example: http://petstore.swagger.io/v2/pet/55			

Column Description	Oracle	DB2	MSSQL
OPERATION_NAME Name of the API operation or resource that is invoked. Example: /pet/{petId}	Varchar(256)	Varchar(256)	NVarchar(256)
RESPONSE_CODE The HTTP response status code that indicates success or failure of the requested operation. Example: 200	Numeric	INT	INTEGER
SESSION_ID A string the API Gateway server generates to uniquely identify each session. This is either the IS session token or the automatically generated GUID if the token is missing from the message context. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2	Varchar(128)	Varchar(128)	NVarchar(128)
TARGET_NAME Name of the API Gateway instance reporting the event. Example: API_Gateway_Instance	Varchar(64)	Varchar(64)	NVarchar(64)

Policy Violation Events

Column Description	Oracle	DB2	MSSQL
ALERT_DESC Text of the alert message sent to a configured destination when the performance conditions are violated. The alert message is specified in the policy definition of an API. Example: EnforcePolicy-HardLimit	Varchar(256)	Varchar(256)	NVarchar(256)
ALERT_SOURCE Name of the API Gateway policy that generated the alert message. Example: PolicyViolationPolicy	Varchar(256)	Varchar(256)	NVarchar(256)

Column Description	Oracle	DB2	MSSQL
ALERT_TYPE The type of alert generated for the event. Example: PolicyViolation	Varchar(128)	Varchar(128)	NVarchar(128)
API_ID The unique identifier for the API. Example: c0f84954-9732-11e5-b9f4-f159eafe47b1	Varchar(256)	Varchar(256)	NVarchar(256)
API_NAME Name of the API in which the event occurred. Example: SampleAPI	Varchar(256)	Varchar(256)	NVarchar(256)
API_VERSION The system-assigned version identifier for the API. Example: 1.0	Varchar(256)	Varchar(256)	NVarchar(256)
APPLICATION_ID The unique identifier for the application associated with the API invocation. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2	Varchar(256)	Varchar(256)	NVarchar(256)
APPLICATION_IP IP address of the application associated with the API invocation. Example: 10.20.248.33	Varchar(64)	Varchar(64)	NVarchar(64)
APPLICATION_NAME Name of the application associated with the API invocation. An application name is populated as unknown when API Gateway is unable to identify the application using a security policy that is configured for the API. Example: SampleApplication	Varchar(128)	Varchar(128)	NVarchar(128)
BINDING_NAME Name of the binding that identifies a specific access URI.	Varchar(256)	Varchar(256)	NVarchar(256)

Column Description	Oracle	DB2	MSSQL
This column is currently not used. It appears as NULL or as an empty string.			
CONSUMER_ID The unique identifier for the consumer associated with the API invocation. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2	Varchar(256)	Varchar(256)	NVarchar(256)
CONSUMER_IP IP address of the consumer associated with the API invocation. Example: 10.20.248.33	Varchar(32)	Varchar(32)	NVarchar(32)
CONSUMER_NAME Name of the consumer associated with the API invocation. A consumer name is populated as unknown when API Gateway is unable to identify the consumer using a policy that is configured for the API. Example: SampleApplication	Varchar(128)	Varchar(128)	NVarchar(128)
EVENT_CREATE_TS Date and time when the event was generated in API Gateway. This is not the time the database performed its insert (for example, this is calculated by the Mediator policy engine and not a database function).	TIMESTAMP	TIMESTAMP	DATETIME
EVENT_SOURCE The source where the event occurred.	Varchar(80)	Varchar(80)	NVarchar(80)
EVENT_TYPE The type of event that occurred. Example: Policy Violation	Varchar(256)	Varchar(256)	NVarchar(256)
EVENT_USERNAME Name of the user on API Gateway that invoked the API.	Varchar(80)	Varchar(80)	NVarchar(80)

Column Description	Oracle	DB2	MSSQL
HTTP_METHOD The HTTP method used to invoke the API. Example: GET	Varchar(8)	Varchar(8)	NVarchar(8)
NATIVE_ENDPOINT The endpoint URL of the native API that is invoked. Example: http://petstore.swagger.io/v2/pet/55	Varchar(4000)	Varchar(4000)	NVarchar(4000)
OPERATION_NAME Name of the API operation that is invoked. Example: /pet/{petId}	Varchar(256)	Varchar(256)	NVarchar(256)
ORG_KEY The Globally Unique Identifier (GUID) for an organization. This column is currently not used by APIs created in API Gateway. It is used to support the APIs that are migrated from CentraSite or Mediator to API Gateway.	Varchar(128)	Varchar(128)	NVarchar(128)
RESPONSE_CODE The HTTP response status code that indicates success or failure of the requested operation. Example: 200	Numeric	INT	INTEGER
SERVICE_KEY The Universally Unique Identifier (UUID) for the service in which the event occurred. This column is currently not used by APIs created in API Gateway. It is used to support the APIs that are migrated from CentraSite or Mediator to API Gateway.	Varchar(128)	Varchar(128)	NVarchar(128)
SERVICE_NAME Name of the service in which the event occurred. Example: SampleAPI	Varchar(256)	Varchar(256)	NVarchar(256)
SERVICE_VERSION	Varchar(256)	Varchar(256)	NVarchar(256)

Column Description	Oracle	DB2	MSSQL
The system-assigned version identifier for the service. Example: 1.0			
SESSION_ID A string the API Gateway server generates to uniquely identify each session. This is either the IS session token or the automatically generated GUID if the token is missing from the message context. Example: c0f84954-9732-11e5-b9f4-f159eafe47b2	Varchar(128)	Varchar(128)	NVarchar(128)
TARGET_NAME Name of the API Gateway instance reporting the event. Example: API_Gateway_Instance	Varchar(64)	Varchar(64)	NVarchar(64)
USER_AGENT Name of the client used to invoke the API. Example: Postman	Varchar2(256)	Varchar(256)	NVarchar(256)

Performance Metrics

Column Description	Oracle	DB2	MSSQL
API_ID The unique identifier for the API. Example: c0f84954-9732-11e5-b9f4-f159eafe47b1	Varchar(256)	Varchar(256)	NVarchar(256)
API_NAME Name of the API in which the event occurred. Example: SampleAPI	Varchar(256)	Varchar(256)	NVarchar(256)
API_VERSION The system-assigned version identifier for the API. Example: 1.0	Varchar(256)	Varchar(256)	NVarchar(256)
AVAIL	NUMBER	NUMBER (6, INTEGER 2)	

Column Description	Oracle	DB2	MSSQL
The percentage of time that an API was available during the current interval. A value of 100 indicates that the API was always available. If invocations fail due to policy violations, this parameter could still be as high as 100.			
Example: 100			
AVG_RESP	NUMBER	INT	INTEGER
The average amount of time it took the API to complete each invocation in the current interval. Response time is measured from the moment API Gateway receives the request until the moment it returns the response to the caller.			
Example: 1376			
BINDING_NAME	Varchar(256)	Varchar(256)	NVarchar(256)
Name of the binding that identifies a specific access URI.			
This column is currently not used. It appears as NULL or as an empty string.			
EVENT_CREATE_TS	TIMESTAMP	TIMESTAMP	DATETIME
Date and time when the event was generated in API Gateway.			
This is not the time the database performed its insert (for example, this is calculated by the Mediator policy engine and not a database function).			
EVENT_SOURCE	Varchar(80)	Varchar(80)	NVarchar(80)
The source where the event occurred.			
EVENT_TYPE	Varchar(256)	Varchar(256)	NVarchar(256)
The type of event that occurred.			
Example: Performance Data			
FAULT_COUNT	NUMBER	INT	INT
Total number of error invocations for a specified API within the configured metrics interval.			
Example: 1			

Column Description	Oracle	DB2	MSSQL
INCLUDEFAULTS	CHAR(1)	CHAR(1)	NCHAR(1)
Includes failed API invocations. Possible values are: true, false			
INTERVAL_START	TIMESTAMP (6)	TIMESTAMP	DATETIME
The starting date and time from which you want to examine metrics.			
Example: 1526294632172			
INTERVAL_STOP	TIMESTAMP (6)	TIMESTAMP	DATETIME
The ending date and time until which you want to examine metrics.			
Example: 1526294632182			
LIVELY	Char(1)	Char(1)	NChar(1)
Boolean. Availability of an the API at the end of the current interval.			
MAX_RESP	NUMBER	INT	INTEGER
The maximum amount of time (in milliseconds) it took for the API to complete an invocation in the current interval.			
Example: 1401			
MIN_RESP	NUMBER	INT	INTEGER
The minimum amount of time (in milliseconds) it took for the API to complete an invocation in the current interval.			
Example: 1352			
NATIVE_ENDPOINT	Varchar(4000)	Varchar(4000)	NVarchar(4000)
The endpoint URL of the native API that is invoked.			
Example: http://petstore.swagger.io/v2/pet/55			
OPERATION_NAME	Varchar(256)	Varchar(256)	NVarchar(256)
Name of the API operation or resource that is invoked.			
Example: /pet/{petId}			

Column Description	Oracle	DB2	MSSQL
ORG_KEY The Globally Unique Identifier (GUID) for the organization. This column is currently not used by APIs created in API Gateway. It is used to support the APIs that are migrated from CentraSite or Mediator to API Gateway.	Varchar(128)	Varchar(128)	NVarchar(128)
SERVICE_KEY The Universally Unique Identifier (UUID) for the service in which the event occurred. This column is currently not used by APIs created in API Gateway. It is used to support the APIs that are migrated from CentraSite or Mediator to API Gateway.	Varchar(128)	Varchar(128)	NVarchar(128)
SERVICE_NAME Name of the service in which the event occurred. Example: SampleAPI	Varchar(256)	Varchar(256)	NVarchar(256)
SERVICE_VERSION The system-assigned version identifier for the service. Example: 1.0	Varchar(256)	Varchar(256)	NVarchar(256)
SUCCESS_COUNT The number of successful API invocations in the current interval. Example: 1	NUMBER	INT	INTEGER
TARGET_NAME Name of the API Gateway instance reporting the event. Example: API_Gateway_Instance	Varchar(64)	Varchar(64)	NVarchar(64)
TOTAL_COUNT The total number of API invocations (successful and unsuccessful) in the current interval. Example: 2	NUMBER	INT	INTEGER
USER_AGENT Name of the client used to invoke the API.	Varchar2(256)	Varchar(256)	NVarchar(256)

Column Description	Oracle	DB2	MSSQL
Example: Postman			

Local Log

The runtime events and metrics payload generated by API Gateway at run-time is published to the configured Local Log destination. The columns that make up the events and metrics data model for Local Log are listed below:

Transactional Events

Column	Description
ApiName	Name of the API in which the event occurred. Example: SampleAPI
ApiVersion	The system-assigned version identifier for the API. Example: 1.0.0
ApplicationID	The unique identifier for the application associated with the API invocation. Example: 7908eb44-d107-4670-929d-89111fc9347c
ApplicationIP	IP address of the application associated with the API invocation. Example: 10.60.37.42
ApplicationName	Name of the application associated with the API invocation. An application name is populated as unknown when API Gateway is unable to identify the application using a security policy that is configured for the API. Example: SampleApplication
CorrelationID	The unique identifier that is automatically generated for every request coming to API Gateway and can be used to query the log. Example: MED38e9cfa4-2348-408b-9462-124b2181c1a6:656
CustomFields	The custom fields an API Provider can provide to log a new field and value for a transaction event. Example: {"customfield": "customvalue"}
ErrorOrigin	The origin of error. Example: Nativeservice

Column	Description
EventSource	The source where the event occurred. Example: API_Gateway_Instance
NativeHttpMethod	The HTTP method used to invoke the native service. Example: GET
nativeRequestPayload	The native service request data. Example: <pre>{ "param1" : "value1", "param2" : 10 }</pre>
nativeResponsePayload	The native service response data. Example: <pre>{ "id":2, "category": { "id":2, "name":"string" }, "name":"pysen", "photoUrls":["string"], "tags": [{ "id":0, "name":"string" }], "status":"available" }</pre>
Native URL	URL of the native service. Example: http://petstore.swagger.io/v2/pet/2
Operation/Resource name	Name of the API operation or resource that is invoked. Example: /pet
Partner ID	The unique identifier for the partner that generated the audit record. Example: unknown
SessionId	
queryParams	This is applicable only for REST APIs. Query parameters present in the incoming REST request.

Column	Description
	Example: {"status":"available"}
RequestHeaders	Request header in the incoming request from the client.
	Example:
<pre>{ "Cache-Control": "max-age=0", "Accept": "text/plain,application/json;q=0.9,image/webp,image/apng,*/*;q=0.8", "Upgrade-Insecure-Requests": "1", "Connection": "keep-alive", "User-Agent": "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36", "Host": "mcdaso02:5555", "Accept-Encoding": "gzip,deflate", "Accept-Language": "en-US,en;q=0.9,ta;q=0.8", "Content-Type": "application/x-www-form-urlencoded" }</pre>	
ResponseHeaders	Response header in the outgoing response.
	Example:
<pre>{ "Server": "Jetty(9.2.9.v20150224)", "Access-Control-Allow-Origin": "*", "Access-Control-Allow-Methods": "GET,POST,DELETE,PUT", "Connection": "close", "Date": "Fri, 30 Mar 2018 08:25:45 GMT", "Access-Control-Allow-Headers": "Content-Type,api_key,Authorization", "Content-Type": "application/xml" }</pre>	
Session Id	A string the API Gateway server generates to uniquely identify each session. This is either the IS session token or the automatically generated GUID if the token is missing from the message context.
	Example: 81439d366e874bc79d9f81490e30e6e0
Source Gateway Node	Source API Gateway's IP address.
	Example: 10.0.75.1
TargetEPR	The endpoint URL of the native API that is invoked.
	Example: http://petstore.swagger.io/v2/pet/55

Monitoring Events

Column	Description
alertDesc	<p>Text of the alert message sent to a configured destination when the performance conditions are violated. The alert message is specified in the policy definition of an API.</p> <p>Example: EnforcePolicy-HardLimit</p>
alertSource	<p>Name of the API Gateway policy that generated the alert message.</p> <p>Example: Unknown-Policy</p>
apiName	<p>Name of the API in which the event occurred.</p> <p>Example: SampleAPI</p>
apiVersion	<p>The system-assigned version identifier for the API.</p> <p>Example: 1.0</p>
applicationId	<p>The unique identifier for the application associated with the API invocation.</p> <p>Example: c0f84954-9732-11e5-b9f4-f159eafe47b2</p>
applicationIp	<p>IP address of the application associated with the API invocation.</p> <p>Example: 10.20.248.33</p>
applicationName	<p>Name of the application associated with the API invocation.</p> <p>An application name is populated as unknown when API Gateway is unable to identify the application using a security policy that is configured for the API.</p> <p>Example: SampleApplication</p>
creationDate	<p>Date and time when the event was generated in API Gateway.</p> <p>Example: 1501671101509</p>
eventType	<p>The type of event that occurred.</p> <p>Example: Policy Violation Event</p>
monitorAttr	<p>The monitored attribute which has breached the configured SLA.</p> <p>Example: AVGRESPONSETIME GT 1.0, SUCCESSCOUNT EQ 3, REQUESTCOUNT GT 10</p>
Native Endpoint	The endpoint URL of the native API that is invoked.

Column	Description
	Example: http://petstore.swagger.io/v2/pet/55
Operation/Resource name	Name of the API operation or resource that is invoked.
	Example: /pet

Troubleshooting: Monitoring API Gateway

- During API Gateway monitoring, when you encounter a problem, check if the components of the node or the containers are up and running.
- To identify the process ID of the application and to generate thread dump and heap dump for monitoring various system and application metrics, see:
 - “How Do I Generate Thread Dump?” on page 299.
 - “How Do I Generate Heap Dump?” on page 301.
- API Gateway and API Data Store logs are located at the following locations:
 - *SAGInstallDirectory\IntegrationServer\instances\instance_name\logs*
 - *SAGInstallDirectory\profiles\IS_instance_name\logs*
 - *SAGInstallDirectory\InternalDataStore\logs*
- For information about the logs, see “Application Log Configurations” on page 389.

How Do I Generate Thread Dump?

Thread dumps are vital artifacts to diagnose CPU spikes, deadlocks, memory problems, unresponsive applications, poor response times, and other system problems. Thread dump is used to analyze thread contention issues and it provides information on the exact status of each thread and information about the call stack of each thread. There are various platforms to generate a thread dump. This section explains few platforms from where you can take a thread dump. You can choose any platform according to your requirement.

Prerequisites:

- Java 7 version and above is considered for generating dumps.
- To check the Process ID (PID) of the API Gateway JAVA application:

Linux: Run the following command `ps -ef | grep java`
 Windows: JAVA PID is available in the Task Manager

Generate Thread Dump using jstack

jstack is an effective command line tool to capture thread dumps. The jstack tool is located in the `JRE or JDK_HOME\bin` folder.

Run the following command to generate thread dump:

```
jstack -l pid > file_path
```

where:

pid. Process Id of the application, whose thread dump should be generated.

file_path. File path, where the thread dump has to be generated.

Example:

```
jstack -l 37320 > /opt/tmp/threadDump.txt
```

37320 is the PID and the opt/tmp/threadDump.txt is the location where the thread dump of the process is generated.

Note:

The following shell script generates the thread dumps automatically multiple times:

```
i=1
while [ $i -le 5 ]
do
    echo jstack -l <PID> > ThreadDump$i
    sleep 10
    i=`expr $i + 1`
done
```

Generate Thread Dump using jVisualVM

Java VisualVM is a graphical user interface tool that provides detailed information about the applications while they are running on a specified Java Virtual Machine (JVM). jVisualVM is located in *JDK_HOME\bin\jvisualvm.exe*

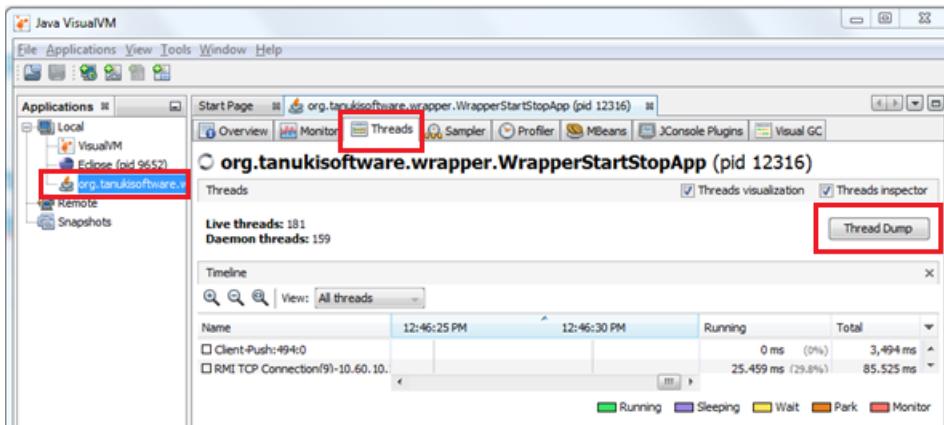
1. Launch the jVisualVM.
2. Select your java application from the list in the left pane.

The left pane of the window lists all the java applications that are running on your system.

3. Click on the **Threads** tab in the right pane.
4. Click **Thread Dump** button.

The thread dump is generated.

Example:



Note:

This tool also has the capability to generate thread dumps from the java processes that are running in remote host as well. To connect to the jVisualVM, the java process (remote process) must be started with the JMX port using the following command:

```
java -Dcom.sun.management.jmxremote.port=3333 \
-Dcom.sun.management.jmxremote.ssl=false \
-Dcom.sun.management.jmxremote.authenticate=false \
YourJavaApp
```

Generate Thread Dump using JCMD

The JCMD tool is located in the `JRE or JDK_HOME\bin` folder. To generate thread dump using JCMD:

Run the following JCMD command to generate thread dump:

```
jcmd <pid> Thread.print > file_path
```

where:

pid. Process Id of the application, whose thread dump should be generated.

file_path. File path, where the thread dump is generated.

Example:

```
jcmd 37320 Thread.print > /opt/tmp/threadDump.txt
```

37320 is the PID and the `/opt/tmp/threadDump.txt` is the location where the thread dump of the process is generated.

How Do I Generate Heap Dump?

Heap Dumps are vital artifacts to diagnose memory-related problems such as slow memory leaks, garbage collection problems, and `java.lang.OutOfMemoryError`. They are also vital artifacts to optimize the memory consumption. There are various platforms to generate a heap dump. This section explains few platforms from where you can generate a heap dump. You can choose any platform according to your requirement.

Prerequisites:

- Java 7 version and above is considered for generating dumps.
- To check the Process ID (PID) of the API Gateway JAVA application :

Linux: Run the following command `ps -ef | grep java`
Windows: JAVA PID is available in the Task Manager

Generate Heap Dump using jmap

The jmap tool is located in the *JRE or JDK_HOME\bin* folder. jmap tool generates heap dumps into a specified file location. To generate a heap dump:

Run the following command:

```
jmap -dump:format=b,file=file_path.bin PID
```

where:

pid. Process Id of the application, whose heap dump should be generated.

file_path. File path, where the heap dump has to be generated.

Example:

```
jmap -dump:format=b,file=/opt/tmp/heapdump.bin 37320
```

37320 is the PID and the opt/tmp/heapdump.txt is the location where the heap dump of the process is generated.

Generate Heap Dump using jVisualVM

Java VisualVM is a graphical user interface tool that provides detailed information about the applications while they are running on a specified Java Virtual Machine (JVM). jVisualVM is located in *JDK_HOME\bin\jvisualvm.exe*

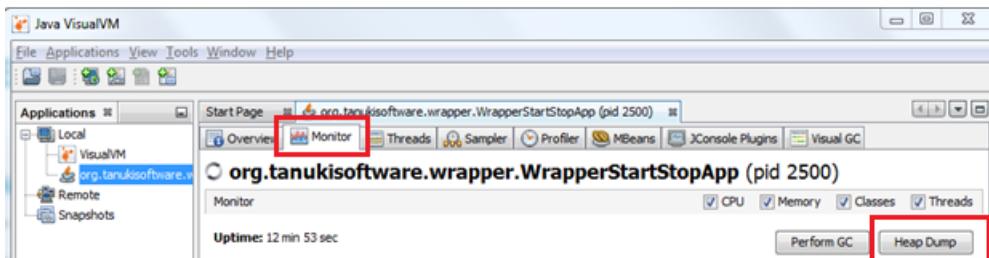
1. Launch the jVisualVM.
2. Select your java application from the list in the left pane.

The left pane of the window lists all the java applications that are running on your system.

3. Click on the **Monitor** tab in the right pane.
4. Click **Heap Dump** button.

The Heap dump is generated.

Example:



HeapDumpOnOutOfMemoryError

When an application runs out of memory, it is ideal to generate heap dump right at that point to diagnose the problem. You can identify the objects that were occupying the memory and also the percentage of memory they were occupying when `java.lang.OutOfMemoryError` occurred. The following JVM parameter can be set while starting the Java application to generate the heap dump whenever Out Of Memory (OOM) exception occurs in the application.

```
-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=file_path
```

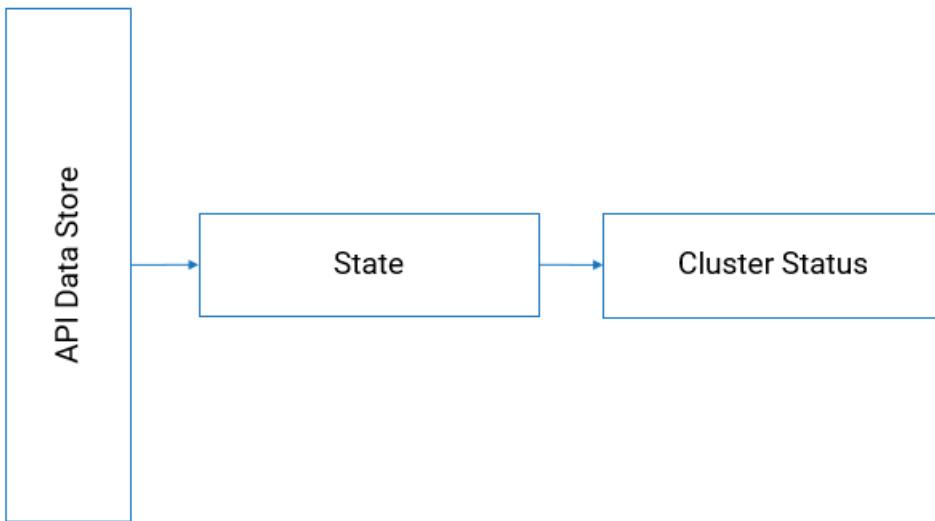
where `file_path` is the location and the name of the file, where the heap dump is generated.

Causes for timeout errors

The following factors can contribute to the delay when the Readiness Probe, Runtime Service Health Probe, or Administration Service Health Probe initiates resulting in the timeout errors.

- Poor capacity (CPU-RAM) or sizing of the components.
- Nodes are not up and running.
- Poor configurations may result in license validation issues, or ports connectivity issues.
- Firewall or proxy issues may result in communication failure between the components.
- Virtualized or containerized environments may run slowly due to overprovisioning.
- Other programs that run on the machines such as antivirus, and so on.

Monitoring API Data Store



As part of application monitoring, you can monitor the state, that is the cluster status of API Data Store along with the resources.

How do I monitor the health of API Data Store?



Prerequisites:

- You must have a valid API Gateway user credential for using the Readiness Probe and Liveness Probe.
- All the node level probes must be setup to target the local instance, typically, localhost.
- Software AG recommends to set up a dedicated port for monitoring with an appropriate private thread pool.

Readiness Probe at Node-Level

To monitor the readiness of API Data Store, that is to check if API Data Store has started successfully, use the following REST endpoint:

GET `HTTP://host:port/_cluster/health`

The following table shows the response code and the description.

Response	Description
200 OK	Readiness check is successful.
500 Internal server error	Readiness check failed and denotes a problem. The response JSON indicates the problem.

Response	Description
	<p>If readiness probe fails, you can perform one of the following actions:</p> <ul style="list-style-type: none"> ■ If you have installed API Gateway directly, check the API Data Store logs to find the status or exception. ■ If you have installed API Gateway through docker image or Kubernetes, ensure that the existing pod is resolved or a new pod is created (automatically) and ready for serving the requests.
timeout or no response as the request did not reach the probe	<p>Several factors can contribute to the delay when the Readiness Probe initiates, which may result in the timeout errors. To know the reasons for timeout errors, see "Causes for timeout errors" on page 303 for more information.</p>

Liveness Probe at Node-Level

As API Data Store works in a cluster-based environment, the result of the Liveness Probe is determined by the cluster health. You can check the cluster status using the same endpoint mentioned for the Readiness Probe.

How do I collect metrics?

Metrics collection is reported in the Prometheus data format. Prometheus is a non- Software AG dashboarding tool that helps in trend analysis. For more information, see <https://prometheus.io/>. The Prometheus metrics names can differ in your environment if you are using a different Prometheus exporter like ES exporter.

Application Metrics

Monitor the following metrics to analyze API Data Store health.

- Index size
- Cluster health
- Number of shards
- GC monitoring

Note:

The threshold values, configurations, and severities that are mentioned throughout this section are the guidelines that Software AG suggests for an optimal performance of API Data Store. You can modify these threshold values or define actions based on your operational requirements.

For details about how to generate thread dump and heap dump, see ["Troubleshooting: Monitoring API Data Store" on page 314](#).

If the metrics return an exceeded threshold value, consider the severity as mentioned and perform the possible actions that Software AG recommends to identify and debug the problem and contact Software AG for further support.

Index Size

Storing all data in a single index will slow down Elasticsearch's performance. Hence, the data must be split into multiple smaller indexes and stored. Advantages of small indexes include:

- **Faster start-up of Elasticsearch.** Multiple smaller indexes instead of one huge index allows Elasticsearch to start up faster.
- **Faster response.** When you store all data in a single index, then Elasticsearch slows down since it spends a lot of time in shard allocation. Chunking of data in smaller units helps in avoiding this time consumption.

Each index has two divisions; the primary shard and the replica shard. The data is first stored in primary shard. Elasticsearch replicates the data in the primary shard as replica shard. For example, when you allot 25 GB for an index, the space is equally divided for both divisions of an index. As per the example, the size of all indexes total up to a maximum of 300 GB. That is, 150 GB is for primary data and the 150 GB for replica shards. Replication of primary data enables Elasticsearch to make it highly available.

When data on a particular index exceeds a certain limit, it is essential to roll over the index and create a new index. The acceptable size limit of an index depends on its type. Software AG recommends that you specify 25 GB (12.5 for each shard) for the transactional events indexes and 5 GB (2.5 for each shard) for tracer indexes. For the list of tracer indexes, see "["List of Indexes that can be included in backup" on page 149](#)".

It is essential to monitor the transactional events indexes to prevent them exceeding 25 GB of size. For information on calculating index size, see "["Calculating index size" on page 306](#)".

You must rollover an index when the size of the primary shard is 12.5 GB. That is, if the size of the primary index is 12.5 GB, then the size of replica will also be 12.5 GB. Hence, you must monitor the size of primary index and perform rollovers as and when required.

When you rollover an index, a new index is created with a primary and a replica for each shard. The naming convention of the new index is `Index_name_YYYYMMDDHHMM`. For example, `gateway_default_analytics_transactionalEvents_YYYYMMDDHHMM`. For information on creating a rollover, see "["Creating Rollover of an Index" on page 133](#)".

Calculating index size

The query used to calculate the index size returns the primary shard of an index. Hence, you must calculate the actual index size by multiplying the returned size by two. For example, if you want to purge indexes that are beyond 25 GB, then you must purge the indexes whose size are 12.5 GB.

1. Run the following command:

```
http://localhost:9240/_cat/indices/gateway_tenant_index_name?  
v&s=i&format=json&pretty
```

For example,

```
http://localhost:9240/_cat/indices/
gateway_default_analytics_transactionalevents_1639736462002-000001?
v&s=i&format=json&pretty
```

Sample output.:

```
[  
{  
  "health" : "yellow",  
  "status" : "open",  
  "index" : "gateway_default_analytics_transactionalevents_1639736462002-000001",  
  "uuid" : "2tmWIIAcQ1KeSqIg9iPU0g",  
  "pri" : "5",  
  "rep" : "1",  
  "docs.count" : "663",  
  "docs.deleted" : "0",  
  "store.size" : "909.8kb",  
  "pri.store.size" : "909.8kb"  
}  
]
```

API Data Store Cluster Health

To ensure optimal health and performance of API Data Store, Software AG recommends monitoring the API Data Store cluster health regularly.

Command	Description
<code>curl -X GET http://localhost:9240/_cluster /health?pretty</code>	This command retrieves API Data Store cluster health status.
<code>\$.status</code>	This JSON path expression retrieves the cluster health status from the response.
<code>\$.number_of_nodes</code>	This JSON path expression retrieves the number of nodes in the cluster from the response.

The response JSON of the health check request displays a status field in the response. The status can have the values green, yellow or red. The cluster health status is displayed based on the following color codes:

Status	Description
green	Indicates that the cluster is in a healthy state. When API Data Store is handling huge data, it takes some time to display the cluster health status.
yellow	Indicates that the cluster is not in a healthy state. Identify the cause and rectify it. During this time, API Data Store processes the requests for the index that is available. If there are unassigned shards, then identify the unassigned shards, check the reason for the unallocation and resolve the issue.

Status	Description
red	<ul style="list-style-type: none"> ■ Run the following command to retrieve the list of unassigned shards. <pre data-bbox="572 340 1356 487">curl -X GET "http://localhost:9240/_cat/shards?h=index,shard,primaryOrReplica,state,docs,store,ip,node,segments.count,unassigned.at,unassigned.details,unassigned.for,unassigned.reason,help,s=index&v"</pre> ■ Run the following command to check the unallocated reason for specific shards. <pre data-bbox="572 614 1356 720">curl -X GET "http://localhost:9240/_cluster/allocation/explain" -d '{ "index" :"index name", "primary" : "true false", "shard": "shardnumber" }' reason,help,s=index&v"</pre>

A sample HTTP response is as follows:

```

{
  "cluster_name": "SAG_apidastore_cluster",
  "status": "green",
  "timed_out": false,
  "number_of_nodes": 3,
  "number_of_data_nodes": 3,
  "active_primary_shards": 101,
  "active_shards": 202,
  "relocating_shards": 0,
  "initializing_shards": 0,
  "unassigned_shards": 0,
  "delayed_unassigned_shards": 0,
  "number_of_pending_tasks": 0,
  "number_of_in_flight_fetch": 0,
  "task_max_waiting_in_queue_millis": 0,
  "active_shards_percent_as_number": 100.0
}

```

The overall cluster status is green since all API Data Store nodes work as expected.

Number of shards

To ensure proper allocation of shards to nodes, Software AG recommends to monitor the number of shards regularly.

Command	Description
<pre>curl -X GET "http://localhost:9240/_cluster /health?pretty"</pre>	<p>This command retrieves the number of shards on API Data Store. If the total number of active shards from the response exceeds the <code>heap space * nodes * 20</code> count, then increase the heap space of API Data Store nodes or add a new API Data Store node. For more information on adding a new API Data Store node, see “Adding New Nodes to an Elasticsearch Cluster” on page 34.</p> <p>API Data Store considers a maximum of 20 active shards per GB of heap space as healthy. Perform any of the following actions to maintain the total number of active shards:</p> <ul style="list-style-type: none"> ■ Scale up the API Data Store node. ■ If you are not able to scale up the API Data Store node, then increase the heap size as the last option. The heap space should not be more than half of system memory (RAM). For example, if the system memory is 16 GB, you can allocate a maximum of 8 GB for API Data Store. <p>To increase the heap space, modify the parameters <code>Xms2g</code> and <code>Xmx2g</code> in the <code>jvm.options</code> file located at <code>SAG_Install_Directory\InternalDataStore\config</code>.</p>

Garbage Collection (GC) Monitoring

The GC metric provides the GC run-time in seconds. You must check GC run-time once every five minutes. The average GC run-time should not exceed one second.

Metric	Description
■ <code>elasticsearch_jvm_gc_collection_seconds_sum</code>	The quotient of both the metrics gives the GC run time.
■ <code>elasticsearch_jvm_gc_collection_seconds_count</code>	If the quotient is more than 1 second, it implies that GC is taking longer time to run and this slows down API Data Store request processing. You must collect the logs and get the mapping of API index and transaction index.

Infrastructure Metrics

Infrastructure metrics include system metrics and container metrics. For information about container metrics, see [“Container Metrics” on page 313](#).

System Metrics

Monitor the following system metrics to analyze API Data Store health.

- CPU usage

- Disk usage
- Memory usage

Monitor the CPU usage

To ensure that CPU is not over utilized, you must monitor the CPU health regularly. You can monitor the CPU usage at two levels: process level and OS level. If the process level CPU is utilized beyond the threshold limits, you can share the load. However, if the OS level CPU has reached its limits, you must contact your IT team.

Command / Metric	Description
<code>curl -X GET http://localhost:9240 /_nodes/stats/process?pretty</code>	This command retrieves the CPU utilization by the API Data Store pods.
<code>\$.nodes.nodeid .process.cpu.percent</code>	<p>This JSON path expression retrieves the percentage of CPU usage by an API Data Store pod.</p> <p>If a pod is using 80% of the CPU space for more than 15 minutes, consider the severity as <i>WARNING</i> and perform the following steps to identify the causes of higher CPU usage.</p> <ol style="list-style-type: none"> 1. Identify the process that consumes the highest CPU. 2. Generate the thread dump. 3. Analyze the thread dump to identify the thread locks. <p>If a pod is using 90% of the CPU space for more than 15 minutes, look for the following Prometheus metrics:</p> <ul style="list-style-type: none"> ■ <code>elasticsearch_os_cpu_percent</code> ■ <code>elasticsearch_process_cpu_percent</code>
elasticsearch_os_cpu_percent	<p>If <code>elasticsearch_os_cpu_percent</code> is more than 90%, consider the severity as <i>CRITICAL</i> and perform the following steps to identify the causes of higher CPU usage.</p> <ol style="list-style-type: none"> 1. Restart the pod. 2. Check the readiness and liveliness of the pod.
elasticsearch_process_cpu_percent	<p>If <code>elasticsearch_process_cpu_percent</code> is more than 90%, consider the severity as <i>CRITICAL</i> and add a new node to the cluster. To learn more about how to add a new API Data Store node, see “Adding New Nodes to an Elasticsearch Cluster” on page 34.</p>

Monitor the Disk usage

To ensure that all nodes have enough disk space, Software AG recommends to monitor the disk space regularly.

Command	Description
<code>curl -X GET http://localhost:9240 /_nodes/stats/fs</code>	This command retrieves the disk space of the API Data Store nodes. It lists the disk space available in all nodes.
	For more information about Elasticsearch node statistics, see Elasticsearch documentation .
<code>\$.nodes..fs.total.total_in_bytes</code>	This JSON path expression retrieves the total disk space.
<code>\$.nodes..fs.total.free_in_bytes</code>	This JSON path expression retrieves the free disk space.
<code>.nodes..fs.total.available_in_bytes</code>	This JSON path expression retrieves the available disk space.

Disk-based shard allocations

Note:

500GB(HA) / 150GB(single node) is used as an example for maximum data retention here.

Command	Description
<code>curl -X GET http://localhost:9240/_cluster/settings?pretty</code>	This command retrieves the configured disk-based shard allocations in API Data Store. To learn more about disk-based shard allocations, see Elasticsearch documentation .
	The shard allocation is based on the thresholds known as <code>Low</code> , <code>High</code> , and <code>Flood</code> watermark.
Shard allocation: <code>Low</code> <code>\$.persistent.cluster.routing.allocation.disk.watermark.low</code>	The default threshold for this level is 80%. Once the threshold is reached, API Data Store does not allocate new shards to nodes that have used more than 80% disk space. You can calculate if the disk usage is low by using the expression (<code>average disk usage of the API Data Store cluster / standalone</code>). If the result of this expression exceeds the defined threshold (80%), the disk has reached the <code>Low</code> stage. If your disk usage has reached <code>low</code> , perform the following steps: <ol style="list-style-type: none"> 1. Query the transaction event index size and verify if the index is above 525 GB(HA) / 175 GB(single node). If it is already breached, monitor if the purge scripts are running and the index size is decreasing. 2. Verify if the size of each transaction event index is equal to the sum of used space (range of 25 GB). If this does not match, some other external items like increased logs size or heap dump are occupying a lot of space. Clear the logs and heap dump. 3. Repeat the above steps until the transaction event index is less than 525 GB and the average disk usage of the cluster becomes less than 80%.
Shard allocation: <code>High</code>	The default threshold for this level is 85%. Once the threshold is reached, API Data Store attempts to relocate shards away from a node whose disk

Command	Description
<code>\$.persistent.cluster.routing.watermark.high</code>	<p><code>\$.persistent.cluster.routing.usage</code> is above 85%. You can calculate if the disk usage is low by using the expression (average disk usage of the API Data Store cluster / standalone). If the result of this expression exceeds the defined threshold (85%), the disk has reached the <code>High</code> stage. If your disk usage has reached <code>High</code>, perform the following steps:</p>
	<ol style="list-style-type: none"> 1. Query the transaction event index and verify if the index is above 525 GB(HA) / 175 GB(single node). If it is already breached, monitor if the purge scripts are running and the index size is decreasing. 2. Verify if the size of each transaction event index is equal to the sum of used space (range of 25 GB). If this does not match, some other external items like increased logs size or heap dump are occupying a lot of space. Clear the logs and heap dump. 3. Repeat the above steps until the transaction event index is less than 525 GB and the average disk usage of the cluster becomes less than 85%
<code>Shard allocation: Flood</code> <code>\$.persistent.cluster.routing.allocation.disk.watermark.flood</code>	<p>The default threshold for this level is 90%. Once the threshold is reached, API Data Store enforces a read-only index block (<code>index.blocks.read_only_allow_delete</code>) on every index that has one or more shards allocated on the node that has at least one disk exceeding the <code>flood</code> stage. This is the last resort to prevent nodes from running out of disk space.</p>
	<p>You can calculate if the disk usage is in <code>flood</code> stage, by using the expression (average disk usage of the API Data Store cluster / standalone). If the result of this expression exceeds the defined threshold (90%), the disk is in the <code>Flood</code> stage. If your disk usage has reached the <code>Flood</code> stage, perform the following steps:</p>
	<ul style="list-style-type: none"> ■ Monitor the purging of data and ensure the purging happens and the disk space occupancy is reduced. ■ If this situation is due to a spike in the requests count and size, follow up with the customer to understand the reason for the sudden spike and inform the customer to compress the payload for transaction logging or not to store the request or response.
<code>curl -X GET http://localhost:9240/_nodes/stats/metric</code>	<p>This command retrieves information about specific metrics like <code>fs</code>, <code>http</code>, <code>os</code>, <code>process</code>, and so on.</p> <p>For more information about the corresponding metrics, see Elasticsearch documentation.</p>

Monitor the Memory usage

Command	Description
<code>http://HOST:9240/_nodes/nodeid/stats/os</code>	This URL retrieves the memory status utilized by the API Data Store pods.
<code>http://URL/nodes?v&full_id=true&h=id,name,ip</code>	This URL retrieves the node id of the corresponding API. This returns the node id, node name, and the node IP address.
<code>\$.nodes.nodeid.os.mem.free_percent</code>	<p>This JSON expression retrieves the percentage of memory that is free. If a pod is using 85% of the available memory, consider the severity as WARNING, and identify the process that consumes more memory and generate the heap dump.</p> <p>If a pod is using 90% of the available memory, consider the severity as CRITICAL, and perform the following steps to identify the reason.</p> <ol style="list-style-type: none"> 1. Identify the process that consumes more memory. 2. Generate the heap dump. 3. Restart the pod. 4. Check the readiness and liveness of the pod.

Container Metrics

If you have installed API Gateway through Docker or Kubernetes, Software AG recommends monitoring the following metrics to check if API Data Store container is healthy. When the metrics exceed the threshold value, consider the severity as mentioned below and perform the possible actions that Software AG recommends to identify, analyze, and debug the problem.

Metric	Description
PodNotReady	If the pod status is not ready for more than 10 minutes, consider the severity as CRITICAL and check the pod console log to find a status or exception. Ensure that either the issue with the existing pod is resolved or a new pod is created.
DeploymentReplicas Mismatch	If the pod replicas' count is not equal to number of pods in ready state, even after 10 minutes, consider the severity as CRITICAL and check the pod console log, and identify and resolve the new pod provisioning issue.
NodeNotReady	If a newly created or scaled pod is not ready in the Kubernetes cluster even after 15 minutes of deployment, consider the severity as CRITICAL and check the autoscaling settings and node provisioning events, logs, and identify and resolve the issue discovered from the logs.
StatefulSetReplicas Mismatch	If the Statefulset replicas mismatch for longer than 5 minutes, consider the severity as CRITICAL and check the pod console logs to find the status or exception and resolve the same.

Metric	Description
	<p>Note: Statefulset is a workload API that manages the deployment and scaling of a set of pods.</p>
PodCrashLooping	If API Data Store pods are stopping and restarting continuously for more than 10 minutes, consider the severity as <i>CRITICAL</i> and describe the pod status and check for any error. Restart the pod and check the startup logs. Check the availability of system resources and cluster health.
PVC_Usage	If only 10% of the persistent volume is free at any given point of time, consider the severity as <i>CRITICAL</i> and check cluster health and perform the same clean up that you would perform for the API Data Store metrics.
PVC_Error	If the persistent volume status shows XXX at any given point of time, consider the severity as <i>CRITICAL</i> and check the API Data Store cluster status.

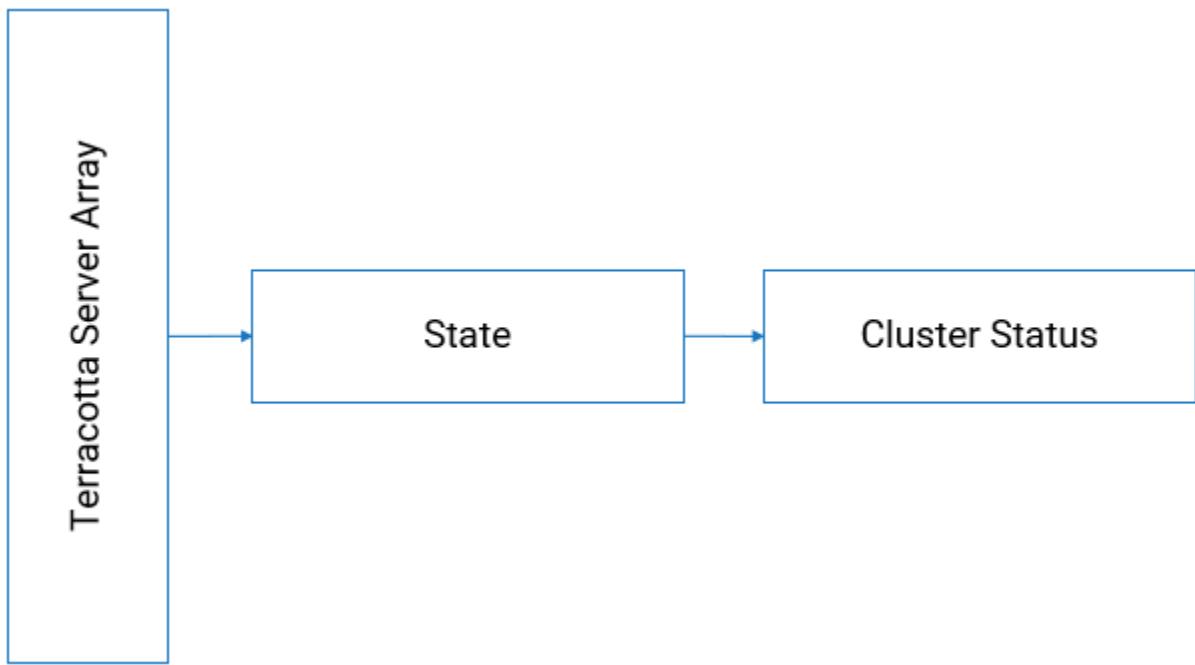
Analyze Trend

You can use external tools for dashboarding operations and visualizing metrics and logs.

Troubleshooting: Monitoring API Data Store

- During API Data Store monitoring, when you encounter any problem, check if the components of the node or the containers are up and running.
- To identify the process ID of the application and to generate thread dump and heap dump for monitoring various system and application metrics:
 - see “[How Do I Generate Thread Dump?](#)” on page 299.
 - see “[How Do I Generate Heap Dump?](#)” on page 301.
- API Data Store logs are located at `SAGInstallDirectory\InternalDataStore\logs`

Monitoring Terracotta



As part of application monitoring, you can monitor the state, that is the cluster health of Terracotta Server Array.

How do I monitor the health of Terracotta Server Array?

Liveness Probe at Node-Level

To monitor the liveness of Terracotta server, that is to check the cluster health status of Terracotta, run the following script:

```
SAGInstallDirectory/Terracotta/server/bin/server-stat.sh
```

Check the following condition from the response to verify the liveness of the server.

```
<server>.health = OK
AND
<server>.role = ACTIVE/PASSIVE
```

Following is one of the responses based on which Terracotta instance (active or passive), the health check is done:

```
server.health: OK
server.role: ACTIVE
server.initialState: START-STATE
server.state: ACTIVE-COORDINATOR
server.port: 9540
server.group name: TSA API Gateway
```

or

```
server.health: OK
server.role: PASSIVE
server.initialState: START-STATE
```

```
server.state: PASSIVE-STANDBY
server.port: 9540
server.group name: TSA API Gateway
```

The following table shows the response code and the description.

Response	Description
200 OK	Runtime service health check is successful.
500 Internal server error	Runtime service health check failed and denotes a problem. If the Runtime service health check fails, restart Terracotta server.
timeout or no response as the request did not reach the probe	Several factors can contribute to the delay when the Runtime Service Health Probe initiates, which may result in the timeout errors. To know the reasons for timeout errors, see "Causes for timeout errors" on page 303 for more information.

Readiness Probe - Node Level

To monitor the readiness of Terracotta server, that is to check if Terracotta server is ready to serve the requests, use the same script that is mentioned for the liveness check and monitor the readiness.

Infrastructure Metrics

Infrastructure metrics include system metrics and container metrics. For information about container metrics, see ["Container Metrics" on page 318](#).

System Metrics

Monitor the following metrics to analyze the health of Terracotta server.

- CPU usage
- Disk usage
- Memory usage

If the metrics return an exceeded threshold value, consider the severity as mentioned below and perform the possible actions that Software AG recommends to identify and debug the problem and contact Software AG for further support.

Note:

The threshold values, configurations, and severities that are mentioned throughout this section are the guidelines that Software AG suggests for an optimal performance of API Gateway. You can modify these thresholds or define actions based on your operational requirements.

To generate thread dump and heap dump for monitoring various system metrics, see ["Troubleshooting: Monitoring Terracotta Server Array" on page 319](#).

Monitor	Description
CPU usage	<p>If the CPU usage of the system is above the recommended threshold value, consider the severity as mentioned:</p> <p>Above 80% threshold for 15 minutes continuously, Severity: <i>WARNING</i></p> <p>Above 90% threshold for 15 minutes continuously, Severity: <i>CRITICAL</i></p> <p>The steps to identify the causes of higher CPU usage are as follows:</p> <ol style="list-style-type: none"> 1. Identify the process that consumes the highest CPU. 2. Generate the thread dump. 3. Analyze the thread dump and logs to identify the problem. 4. Monitor the process closely. If the process fails, it should recreate. 5. Check if the active-passive quorum is intact using the following script: <code>SAGInstallDirectory/Terracotta/server/bin/server-stat.sh</code> 6. Check if API Gateway clients can establish the connection to Terracotta cluster using the following REST endpoint <code>GET /rest/apigateway/health/engine</code>
Disk usage	<p>If the disk usage of the Terracotta server shows a higher value, rotate logs based on a fixed size and fix the number of rotated files to be persisted.</p>
Memory usage	<p>If the memory usage is above the recommended threshold value, consider the severity as mentioned:</p> <p>Above 80% threshold, Severity: <i>WARNING</i></p> <p>Above 90% threshold, Severity: <i>CRITICAL</i></p> <p>The steps to identify the causes of higher memory usage are as follows:</p> <ul style="list-style-type: none"> ■ Identify the process that consumes more memory. ■ Start the Terracotta Management Console (TMC) and check the heap usage, off-heap usage and warnings. ■ Analyze the memory dump and Terracotta logs to identify the issue. ■ Monitor the process closely. ■ Check if the active-passive quorum is intact using the following script: <code>SAGInstallDirectory/Terracotta/server/bin/server-stat.sh</code> ■ Check if API Gateway clients can establish the connection to Terracotta cluster using the following REST endpoint <code>GET /rest/apigateway/health/engine</code>

Container Metrics

If you have installed Terracotta through Docker or Kubernetes, Software AG recommends monitoring the following metrics to check if Terracotta container is healthy

Metric	Description
PodNotReady	If the status of the pod is <i>not</i> ready for more than 10 minutes, consider the severity as <i>CRITICAL</i> and perform the following actions to identify the problem: <ol style="list-style-type: none"> 1. Check the console logs of the pod to find the status for any exception. 2. Identify issue with the provisioning of the pod.
StatefulSet ReplicasMismatch	If the Statefulset replicas mismatch is longer than 5 minutes, consider the severity as <i>CRITICAL</i> and perform the following actions to identify the problem: <ol style="list-style-type: none"> 1. Check the console logs of the pod to find the status for any exception. 2. Identify issue with the provisioning of the pod.
PodRestarting	Checks and creates a new pod if the application inside the pod is not up. If the application inside the pod is <i>not</i> up in 1 minute, consider the severity as <i>CRITICAL</i> . Kubernetes creates a new pod to maintain the availability. Perform the following actions to identify the problem: <ol style="list-style-type: none"> 1. Check the previous logs of the pod and ensure that you check the logs for all the pods that are running. 2. Check the Terracotta client logs for errors in Terracotta communication, if the tenant is in cluster mode.

Pod restart verification procedure

For any reason, if the pod is restarted, check the following to verify the health of the new pod.

1. Wait for 150 seconds for the alternate pod (passive) to take an active role.
2. If the alternate pod does not take an active role, it can lead to 2 active pods under following circumstances:

- The passive pod can turn active and also the new pod can turn active simultaneously.

Note:

Terracotta heals itself by sending a zap signal to one of the pods.

- When 2 pods are active, it may be due to the reason that the pod that was transitioning from passive to active is stuck and in this case, its readiness or liveness checks returns an

unhealthy status and an appropriate action that is defined for an unhealthy pod is performed.

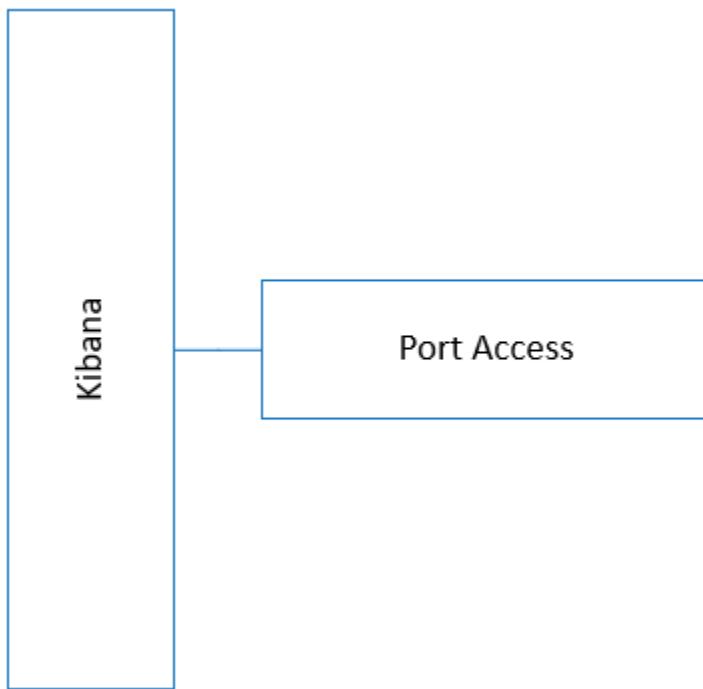
Analyze Trend

You can use external tools for dashboarding operations and visualizing metrics and logs.

Troubleshooting: Monitoring Terracotta Server Array

- During Terracotta Server Array monitoring, when you encounter a problem, check if the components of the node or the containers are up and running.
- Terracotta logs are located at:
 - Client log: *SAGInstallDirectory\IntegrationServer\instances\instance_name\logs*
 - Server log: *SAGInstallDirectory\tsa*
- To generate thread dump and heap dump for monitoring various system and application metrics:
 - see “[How Do I Generate Thread Dump?](#)” on page 299.
 - see “[How Do I Generate Heap Dump?](#)” on page 301.

Monitoring Kibana



Health check

You can ping the following port to check the health of Kibana.

`tcp-socket:9405`

If the port is accessible, you can understand that Kibana is in a healthy state.

Troubleshooting Tips: Monitoring Performance

I see performance degradation when the Log Invocation policy is configured to log API Data Store or an external Elasticsearch

When the Log Invocation policy is configured for APIs, API Gateway displays only some transaction events in the Analytics tab of an API, while ignoring others. This issue occurs during times of high transaction event loads.

Resolution:

Increase the following extended settings and watt properties values to get the desired performance improvements:

- `events.collectionPool.maxThreads`
- `events.collectionPool.minThreads`
- `events.collectionQueue.size`
- `events.reportingPool.maxThreads`
- `events.reportingPool.minThreads`
- `events.reportingQueue.size`

Increase the following JVM heap size in the `wrapper.conf` file located at `SAGIInstallDir\profiles\IS_default\configuration`:

- `wrapper.java.initmemory`
- `wrapper.java.maxmemory`

I see that the runtime events widget does not render transaction events with a huge payload

Runtime events widget under API-specific analytics dashboard does not display the transactional events and a shard failure error appears when there are huge Base64 encoded payloads.

Resolution:

To avoid encountering errors while handling huge Base64 encoded payloads:

- Set the `index.highlight.max_analyzed_offset` index level setting to maximum.

Sample to set the `index.highlight.max_analyzed_offset` index level setting to maximum is as follows:

```
curl -X PUT "localhost:9240/<index_name>/settings?pretty" -H 'Content-Type: application/json' -d' { "index.highlight.max_analyzed_offset" :50000000 }
```

- Set the `indices.query.bool.max_clause_count` property value to 2048 in the `elasticsearch.yml` file located at `SAG_InstallDir/InternalDataStore/config`.

Increasing the `indices.query.bool.max_clause_count` property value to 2048 does not impact the transactional flow performance. But by increasing this value, you are allowing API Data Store to run queries and aggregations over many fields which might consume more resources to run these queries.

I see that I am not able to substitute credentials when using Kibana keystore

Substituting credentials when using Kibana keystore does not work in the API Gateway UI. When navigating to the API Gateway **Analytics** page, the user is asked for Elasticsearch credentials in an authentication window.

Resolution:

To resolve this issue, use plain text passwords while authenticating Kibana keystore.

I see the low disk space issue, and API Gateway stops working for the WRITE operations.

This error occurs when there is low disk space.

The following error messages are seen in the **SAG_EventDataStore.log** file in the `SAGInstallDir\InternalDataStore\logs`:

- Exception: [WARN] [o.e.c.r.a.DiskThresholdMonitor] [localhost1568897216386] flood stage disk watermark [95%] exceeded on [BOf6SQe2SwyI93vi4RIBNQ] [localhost1568897216386] [C:\SoftwareAG\InternalDataStore\data\nodes\0] free: 2.4gb [2.4%], all indices on this node will be marked read-only.
- Saving an API -> error message ("Saving API failed.
com.softwareag.apigateway.core.exceptions.DataStoreException: Error while saving the document. doc Id - 6d5c7ac0-574a-4a53-acba-a738f21e3142, type name - _doc, message - "index [gateway_default_policy] blocked by: [FORBIDDEN/12/index read-only / allow delete (api)];")

Resolution:

- Increase the disk space in all nodes or clean up disk space by clearing unwanted data. If you do not perform this step, then there is a chance that this error might appear again.
- Make a REST call to:

```
curl -XPUT -H "Content-Type: application/json" http://localhost:9200/_all/_settings -d' {"index.blocks.read_only_allow_delete": null}
```

You can optimize the usage of disk space using the Watermark property of Elasticsearch. For information about the property, see
<https://www.elastic.co/guide/en/elasticsearch/reference/current/disk-allocator.html>

General Administration Configuration

In addition to the administrative tasks like Security configuration, Data Management, Destination configuration, External account configuration, API Gateway supports various general administration configurations and system settings.

You must have the API Gateway's manage general administration configurations functional privilege assigned to perform the following configurations in the general configuration section of API Gateway:

- Configure API Gateway to communicate with a load balancer that allows API Gateway to provide the endpoints of the API with the load balancer URL.
- Configure the extended settings which are advanced parameters required for your server to operate properly.
- Configure API fault settings for errors being returned by the API Gateway to the applications.
- Configure approval settings.
- Configure Proxy server aliases.
- Configure URL aliases.
- Configure the custom content-types.
- Configure cache to boost performance.
- Configure log levels and the destination where the aggregated logs are collected.
- Configure API Gateway and Terracotta license file path.
- Configure cluster settings.
- Configure API callback request settings.

Clusters and Load Balancers

Load balancing enables the distribution of messages received across the API Gateway nodes. Clustering helps in attaining the high availability functionality as well as load balancing. In addition, a cluster provides fault tolerance that ensures recovery of events and metrics in case a node goes down. If you have a web service that is hosted at two or more endpoints, you can use the load balancing option to distribute requests across the nodes. Requests are distributed across multiple nodes and are routed based on the round-robin execution strategy.

In a load balanced system, calls from an application are distributed across two or more different instances of API Gateway, referred to as nodes. Each node is an instance of API Gateway running on an instance of Integration Server.

If you cluster API Gateway instances, you must configure API Gateway with the load balancer URL appropriately for it to report the API address at the load balancer URL instead of having direct access address with API Gateway.

Load Balancer URLs

Load balancer URLs are the URLs of the load balancer where the requests are sent by the applications. When an API is activated on a node of a cluster, the endpoint of the API is provided as the load balancer URL instead of the Gateway end point. API Gateway stores this load balancer URL endpoint. Since all nodes of an API Gateway cluster are synchronized with APIs, each API Gateway accepts the message when routed from the load balancer to any node in the cluster.

A load balancer URL consists of a host name (or IP address) and port number of the load balancer in the following format:

`http://hostname:portnumber`

or

`http://IP-address:portnumber`

For example, if the host name of the load balancer is ExampleHost, and its port number is 80, the load balancer URL would be `http://ExampleHost:80`

You must configure any one node in a cluster with the same load balancer URL. The load balancer URLs are automatically synchronized on all the nodes in a cluster.

Note:

- When a Load Balancer URL is updated in API Gateway, a logout and re-login or opening a new session reflects the updated artifacts URL under Specifications. The artifacts downloaded without a new session also show the updated endpoints.
- You can also configure the load balancer URL to use the Web application URL, HTTPS protocol, or WebSocket port configuration.

Configuring Load Balancer URLs

You have to configure the load balancer URLs to define the endpoints for API Gateway.

➤ To configure the load balancer URLs

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Load balancer**.
3. Provide the following information in the API runtime invocation URLs section:

Field	Description
Load balancer URL (HTTP)	Specifies the primary HTTP load balancer URL. For the URL, you can specify either the IP address or host name of the load balancer with the port number, as follows: <code>http://IP-address:portnumber or http://hostname:portnumber</code>
Load balancer URL (HTTPS)	Specifies the primary HTTPS load balancer URL. For the URL, you can specify either the IP address or host name of the load balancer with the port number, as follows: <code>http://IP-address:portnumber or https://hostname:portnumber</code>
Load balancer URL (WS)	Specifies the WebSocket load balancer URL. For the URL, you can specify either the IP address or host name of the load balancer with the port number, as follows: <code>http://IP-address:portnumber or https://hostname:portnumber</code>

4. Provide the following information in the Web application URLs section:

Field	Description
Web application load balancer URL	Specifies the Web application load balancer URL. If a value is not specified, then API Gateway uses the default hostname and port number during publish of an API Gateway asset from CentraSite to API Gateway. For example, <code>http://myHostname:9072</code> .

5. Click **Save**.

Configuring Extended Settings

You must have the API Gateway's manage user administration functional privilege assigned to configure the extended settings.

You can configure advanced parameter settings in the Extended settings section. These parameters affect the operation of your server. You must not change these settings unless requested to do so by Software AG Global Support. You can configure the watt parameter settings in this section.

➤ To configure the extended settings

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Extended settings**.

3. Click **Show and hide keys**. This displays all the configurable parameters.
4. You can configure any of the following parameters in the Extended keys section by providing the required values. The configured values are listed under Extended settings at the top of the page.

Parameter and Description

allowEGInvokeOnly

Specifies whether the SOAP APIs with Transport policy set to `http`, can be invoked using the reverse invoke method when you set the external port as `https`, and the Registration and Internal ports as `http`. Ensure you enable this setting in the system where the SOAP API is created.

Note:

This setting affects only the behavior of SOAP APIs. You can invoke the REST APIs using the reverse invoke method, irrespective of this setting, given the above said conditions are true.

Possible values:

- `true`. You can invoke SOAP APIs using the reverse invoke method if the external port is set as `https`, and the Registration and Internal ports are set as `http`.
- `false`. You cannot invoke SOAP APIs using the reverse invoke method.

allowExceedMaxWindowSize

Specifies whether the number of records retrieved by Elasticsearch in a single request exceeds the configured value or not.

Possible values:

- `true`. The number of records retrieved in a single request can exceed the maximum value configured in Elasticsearch. This is the default value.
- `false`. Displays an error message when the number of records retrieved in a single request exceeds the configured value.

apiDocumentsRestrictedExtension

Specifies the list of restricted file extensions to prevent users from uploading files with those extensions as the input document for an API. For example, a file with the `.exe` file extension could contain executable code that runs on demand when it is downloaded. If files with the `.exe` file extension are restricted, users cannot upload a file with the `.exe` extension in API Gateway.

By default, several standard file extensions are blocked, including any file extensions that are treated as executable files by Windows Explorer. The file extensions blocked by default are:

- `.bat` - Batch file

Parameter and Description

- .bin - Binary file
- .dll - Windows dynamic link library
- .exe - Executable program

You can remove any or all of the default file extensions.

apiDocumentsUploadSizeLimitInMB

Specifies the maximum document size, in MB, that can be uploaded as an input document for an API.

Default value is 5.

This prevents users from uploading huge files that might slow down the system.

apig_MENConfiguration_tickInterval

Specifies the time interval (in seconds) between each interval processor iteration.

The value, you provide, must be an evenly divisible fraction of the smallest policy interval, which is one minute.

Default value is 15.

Note:

Exercise caution when you modify this setting as this is a system level setting.

apig_rest_service_redirect

Specifies whether the incoming API requests must be redirected to the directives, /ws and /mediator for providing Mediator compatible endpoints.

Possible values:

- true. The incoming API requests are redirected to the directives, /ws or /mediator to provide Mediator compatible endpoints.
 - false. The incoming API requests are not redirected to the /ws or /mediator directives. This is the default value.
-

apig_schemaValidationPoolSize

Specifies the pool size for the XML schema parsers. API Gateway uses parsers to validate the XML payload against the XML schema when you have selected Schema in the **Validate API Specification** policy of the API.

The default value is 10. Provide a bigger value to increase the performance of API Gateway in validating the XML schema of APIs.

apiGroupingPossibleValues

Parameter and Description

Specifies the names of API groups. You can organize your APIs by associating them to the relevant API groups. The groups provided, by default, are:

- Finance Banking and Insurance
- Sales and Ordering
- Search
- Transportation and Warehousing

You can add, edit, or delete API groups based on your requirement.

apiKeyExpirationPeriod

Specifies the time for which an API Key is valid. You can provide the value in seconds, minutes, days, months, or years. For example, 8 seconds, 8s, 10 months, 10m, 15 minutes, 15min.

The expiration date is computed as follows:

- When a new application is created: Expiration date = The time when an application is created + The value specified in the apiKeyExpirationPeriod parameter.
- When an API access key is regenerated: Expiration date = The time when the API key is regenerated + The value specified in the apiKeyExpirationPeriod parameter.

If you do not specify a value, then the API key never expires.

apiKeyHeader

Specifies the HTTP header name from which API Gateway retrieves the API key from incoming client requests.

The default value is x-Gateway-APIKey.

apiMaturityStatePossibleValues

Specifies the API maturity state values that can be set for an API. You can search for APIs based on their maturity status.

The default values provided are Beta, Deprecated, Experimental, Production, and Test.

appMesh.microgateway.logLevel

Specifies the log level of Microgateways that are deployed through API Gateway.

The default value is ERROR.

Possible values: TRACE, DEBUG, INFO, WARN, ERROR, FATAL

backupSharedFileLocation

Parameter and Description

Specifies the file location where the data backup file has to be archived. The default location is SAGInstallPath/profiles/IS_default/workspace/temp/default.

The files are saved with the corresponding timestamp in the specified location. Only the run-time events are included in the archives.

clusterNotifierCacheStaleInterval

Specifies the time interval after which data in the ClusterNotifierCache is considered stale and is removed from the cache.

The default value is 900 seconds. If you provide a non-numeric value, API Gateway interprets the value as the default value of 900 seconds. If you provide a value less than 60 seconds, API Gateway interprets the value as the lower limit of 60 seconds.

The ClusterNotifierCache maintains a data structure which has an entry for each active member in a cluster. This data structure is used to communicate changes on API definitions, applications, policies, and so on, between the cluster members. When a cluster member shuts down gracefully it removes its entry from the data structure. However, when a cluster member process is killed the entry remains, and other cluster members continue posting notifications to the entry. In order to avoid endless growing resulting in performance degradation the cluster data structure is monitored for absent cluster members. If a cluster member has not reacted within the configured `clusterNotifierCacheStaleInterval` it is regarded as stale, and its data is removed from the `ClusterNotifierCache`.

customCertificateHeader

Specifies the header name of the request header in which the client certificate can be passed. API Gateway checks for the existence of this header and fetches the certificate and identifies the application.

The default value is `X-Client-Cert`.

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.

Parameter and Description

PEM formatted certificates are defined to be Base64 encoded. Here, the certificate with delimiters are Base64 encoded to avoid the stripping off of the delimiters.

decodeAllDelimitersInURI

Specifies whether the encoded characters of URL in the incoming client requests must be decoded or not. The URLs are encoded as per the URI specs or user's requirements.

Possible values:

- true. The encoded characters in the URL of incoming requests are decoded.
- false. The encoded characters in the URL of the incoming client requests are not decoded. This is the default value.

defaultEncoding

Specifies the format for encoding the design time and run time invocation data.

The default value is UTF-8.

If you want to modify this value, Software AG recommends that you do it before starting API Gateway.

You can change the value in the gateway-core.xml file located in the folder:

SAGInstallDir\IntegrationServer\instances\instance-name\packages\WnAPIGateway\config\resources\beans.

The property to be modified is: <entry key="defaultEncoding" value="UTF-8"/>.

Note:

Changing this value after starting API Gateway might make API Gateway non-functional. If API Gateway is clustered, then the same value should be updated in all nodes of API Gateway.

defaultLanguage

Specifies the display language for the user interface of API Gateway. You can change the display language to your preferred language at any time.

The default value is en.

defaultSearchResultSize

Specifies the default size in the search request for the events, and metrics data.

The default value is 1000. If you provide a value -1 it displays all the search results.

disableRemoteEntityReference

Specifies whether remote entity references are resolved when creating a SOAP API.

Possible values:

Parameter and Description

- `true`. Disables the resolving of remote entity references when creating SOAP APIs.
- `false`. Enables the resolving of remote entity references when creating SOAP APIs.

enableHotdeploy

Specifies whether the hot deploy functionality is enabled or disabled. When this setting is enabled, you can modify APIs that are active.

Possible values:

- `true`. Enables the hot deploy function. This is the default value.
- `false`. Disables the hot deploy function.

Note:

Ensure to refresh your browser when you modify this setting to reflect the changes to the ongoing user sessions.

enableImportBackup

This parameter provides an option to secure an API before overwriting it.

Available values are:

- `true`. If you set the value to `true`, the existing API is restored in the event of an error during the import. This is the default value.
- `false`. If you set the value to `false`, the feature to secure an existing API before overwriting is disabled. If the overwrite fails due to an error during the import, the existing API has to be deleted.

enableTeamWork

Specifies whether the Team Support feature in API Gateway is enabled.

Possible values:

- `true`. Enables the Team Support feature.
- `false`. Disables the Team Support featured. This is the default value.

esScrollTimeout

Specifies the time, in milliseconds, that the search results for each request must be kept active in Elasticsearch.

When the **allowExceedMaxWindowSize** setting is enabled, the **Scroll** feature of the Elasticsearch is enabled to allow Elasticsearch to accept multiple search requests and return multiple results. That is, you can perform multiple search requests using the same query till you get the desired number of records from Elasticsearch.

Parameter and Description

When you send a search request, Elasticsearch returns the result and keeps the result active for the time specified in the **esScrollTimeOut** setting. If a request exceeds the time specified in the **esScrollTimeOut** setting, then the subsequent search requests also fail with a *Invalid Scroll ID* error message. For more information on the Scroll feature, refer <https://www.elastic.co/guide/en/elasticsearch/reference/current/search-request-body.html#request-body-search-scroll>.

events.collectionPool.maxThreads

Specifies the maximum number of threads to be used for the event data collection pool.

Each thread in this pool is assigned a task of collecting the events like transactions, error and performance metrics, and processing them for sending to destinations such as API Gateway, Elasticsearch, API Portal, and so on.

Specifying more number of threads implies that the processing of events for sending to the desired destinations is faster. At the same time, it increases the usage of system resources, which could result in slower service execution.

This value must be greater than or equal to the value of **events.CollectionPool.minThreads**.

Default value is 8.

events.collectionPool.minThreads

Specifies the minimum number of threads to be used for the event data collection pool.

Each thread in this pool is assigned a task of collecting the events like transaction, error and performance metrics and processing them for sending to the desired destinations such as API Gateway, Elasticsearch, API Portal, and so on.

Specifying less number of threads implies that processing of events for sending to the desired destinations is slower.

Default value is 1.

events.collectionQueue.size

Specifies the size of the collection queue to be used during event data collection. This is used in connection with the **events.collectionPool.minThreads** and **events.collectionPool.maxThreads** settings.

When events like transaction, error events, and performance metrics are generated during API invocations, they are put in the collection queue for further processing. Each thread in the collection pool is assigned a task of collecting these events and processing them for sending to the desired destinations such as API Gateway, Elasticsearch, API Portal, and so on.

If the queue capacity is reached, then any additional event data would be lost. Hence it is better to increase this size when there is an increase in incoming traffic.

Parameter and Description

Specifying a large collection queue size and small collection thread pool size might cause delay in processing the events data but ensures all the data are processed.

On the other hand, specifying a small collection queue size and large collection thread pool size might cause faster processing of the events data but keeps the CPUs busier and at the same time when the traffic increases there is a possibility of data loss when the collection queue size is full.

Default value is 10000.

events.reportingPool.maxThreads

Specifies the maximum number of threads to be used for the event data reporting pool.

Each thread in this pool is assigned a task of sending the events like transaction, error and performance metrics to the desired destinations such as API Gateway, Elasticsearch, API Portal, and so on.

Specifying more number of threads implies that sending of events to the desired destinations is faster. At the same time it increases the usage of system resources, which could result in slower service execution.

This value must be greater than or equal to the value of **events.ReportingPool.minThreads**.

Default value is 4.

events.reportingPool.minThreads

Specifies the minimum number of threads to be used for the event data reporting pool.

Each thread in this pool is assigned a task of sending the events like transaction, error and performance metrics to the desired destinations such as API Gateway, Elasticsearch, API Portal, and so on.

Specifying less number of threads implies that sending of events to the desired destinations is slower.

Default value is 2.

events.reportingQueue.size

Specifies the size of the reporting queue to be used during event data reporting. This is used in connection with the **events.reportingPool.minThreads** and **events.reportingPool.maxThreads** settings.

When events like transaction, error events, and performance metrics are generated during API invocations, they are put in the collection queue for further processing. Each thread in the collection pool is assigned a task of collecting these events, processing them and put in the reporting queue for sending to the desired destinations such as API Gateway, Elasticsearch, API Portal, and so on.

Parameter and Description

If the reporting queue capacity is reached, then any additional event data would be lost. Hence it is better to increase this size when there is an increase in incoming traffic.

Specifying a large reporting queue size and small reporting thread pool size might cause delay in processing the events data but ensures all the data are processed.

On the other hand, specifying a small reporting queue size and large reporting thread pool size might cause faster processing of the events data but keeps the CPUs busier and at the same time when the traffic increases there is a possibility of data loss when the collection queue size is full.

Default value is 5000.

eventsRefreshInterval

Specifies the refresh interval for the events indices in seconds.

The default value is 10.

forwardInternalAPIsRequest

This parameter is required in the case of a paired gateway deployment scenario using an Advanced Edition license at DMZ.

Specifies whether the incoming requests are forwarded to internal APIs that are deployed in the green zone.

Possible values:

- true. API Gateway forwards the incoming requests to the internal APIs that are deployed in the green zone.
- false. API Gateway does not forward the incoming requests. This is the default value.

Following are the internal APIs and their URIs for which this parameter is required and its value must be set to true:

- getOAuthToken - /pub/apigateway/oauth2/getAccessToken
- OAuth Authorization - /pub/apigateway/oauth2/authorize
- getOpenIdToken - /pub/apigateway/openid/getOpenIDToken
- openIDCallbackService - /pub/apigateway/openid/openIDCallback
- getJWTToken - /pub/apigateway/jwt/getJsonWebToken

forwardqueryParams

Specifies whether API Gateway should forward the query parameter sent by client and query parameters configured in Request Processing stage to the native service if the \${sys:resource_path} variable is not present in the Routing policy URL.

Parameter and Description

- true. API Gateway forwards the query parameters sent by client and query parameters configured in Request Processing stage to the native service even if the \${sys:resource_path} is not present in the Routing policy URL.
- false. API Gateway does not forward the query parameters to the native service if the \${sys:resource_path} is not available in the Routing policy. This is the default value.

gatewayClientInvokingToken

Specifies the value of the client token that can invoke the backend APIs in API Gateway.

The default value is apigateway.

invokeESB_asUser

Specifies the username of the user who runs IS services via policies configured in API Gateway.

If the Run as User value is included in the policy level for Invoke webMethods IS service section, then the value provided in the policy overrides the user provided in this setting.

If no value is provided in either of the above mentioned fields, then the user authenticated from the incoming request is used for invoking the IS service.

maxAllowedZipFileSize

Specifies the maximum size of zip files that can be uploaded to create an API from the **Create API** screen.

Default value is 100000000.

maxBackupsLimit

Specifies the maximum number of backups that are archived. The default value is 10. If you do not provide a value, then infinite number of archives are kept.

The archives are saved in the location provided in the **backupSharedFileLocation** setting.

maxRegexLengthInSearchQuery

Specifies the maximum length of Regex parameter that you can use in Regex expression query.

Default value is 37000. This value can be increased based on the requirement.

maxWindowSize

Specifies the maximum number of search results that Elasticsearch can return for a single search request.

Using the **index.max_result_window** property in Elasticsearch, you can configure the maximum number of records to be retrieved in a single Elasticsearch request. To know more about the property, refer

<https://www.elastic.co/guide/en/elasticsearch/reference/current/index-modules.html>.

Parameter and Description

Default value is 10000.

If the value configured in the **index.max_result_window** setting of Elasticsearch is different from that of the default value, then it is recommended that you provide the same value in this field.

If you have a value greater than the configured value, then Elasticsearch displays an error message. Also, you must enable the **allowExceedMaxWindowSize** setting if the total number of results to be retrieved is more than the value specified here.

For example, if you have specified 1000 in this setting and the total number of records to be retrieved is 20000, then API Gateway sends 20 requests to retrieve all records provided the **allowExceedMaxWindowSize** setting is enabled. In the above example, if the **allowExceedMaxWindowSize** setting is not enabled, an error message is displayed as the value is lesser than the default value.

paginationPossibleValues

Specifies the list of possible values of the pagination size, that is, number of items listed per page.

The default values displayed for pagination options are 10, 20, 30, 40, 50. For example, if you select 20, then 20 items are displayed per page.

For example, if you change the values to 5, 10, 15, 20, 25 then pagination options displayed are 5, 10, 15, 20 and 25. So now when you select 15, the items displayed per page would be 15.

On modifying the value, you have to logout and re-login for the changes to reflect.

pg_Cache_autoScalerRunInterval

Specifies the run interval, in minutes, for the Auto Scaler thread.

Default value is 120 minutes.

The Auto Scaler thread checks for system memory load and adjusts the percentage of objects kept in the cache automatically.

pg_Cache_averageObjectSize

Specifies the value used to calculate the average size, in bytes, of the objects that can be loaded into cache.

The default value is 64.

Software AG recommends you not to modify this value.

pg_Cache_boundedCacheResizeRunInterval

Specifies the run interval, in minutes, for the bounded cache resize thread.

Parameter and Description

The default value is 30 minutes.

The bounded cache stores a predefined number of objects in memory. The cache value is configured in terms of percentage and it varies based on the new objects added to database. The bounded cache resize thread computes the memory size at the configured interval. For example, if you specify 50, the cache memory is calculated once every 50 minutes.

pg_Cache_maxCacheSize

Specifies the maximum number of objects that are kept in the unbounded cache memory.

The default value is 1048576.

Internally, unbounded cache is also a bounded cache with a configured maximum limit.

pg_Cache_minCachePercent

Specifies the minimum cache percent value. API Gateway maintains the configured memory size. That is, if the value that is computed by the auto scaler thread goes below this value, API Gateway resets the cache percent to the value specified for this setting.

The default value is 20.

The auto scaler thread computes the percentage of objects to be stored in cache. If it computes a lower percentage, say 2%, the value is reset based on the pg_Cache_minCachePercent value.

pg_Cache_minCacheSize

Specifies the minimum number of objects that are kept in the unbounded cache memory.

The default value is 1024.

For example, if there are 20 objects, and cache size is set to 100%, then the computed cache size is 20. When the actual size becomes lower than the value specified in this setting, then the value is reset to the pg_Cache_minCacheSize value.

pg_Cache_statisticsProcessorRunInterval

Specifies the run interval, in minutes, for the statistics processor thread. The statistics processor thread stores the cache statistics are in Elasticsearch.

The default value is 15 minutes.

The cache statistics analytics page displays details about the cache statistics.

pg_Dataspace_GossipInterval

Specifies how frequently each node should gossip with one another.

By default, the value is set to 3 seconds.

pg_Dataspace_TimeToFail

Parameter and Description

Specifies the maximum permissible interval between two consecutive gossips.

By default, the value is set to 30 seconds.

pg_Dataspace_WarmupTime

Specifies the maximum permissible rehashing interval from start-up or shut down of the server.

By default, the value is set to 300 seconds.

pg_JWT_clock_skew_seconds

Specifies the clock difference between API Gateway and the external authorization servers that must be adjusted, when validating a JWT token. Provide the value in seconds. The value must be between 0 - 600 (10 minutes).

Most often, there is a slight time difference between API Gateway and the external authorization servers. This could affect the token authorization when validating the expiry (exp) claim or not before (nbf) claim. Hence, you need to provide the number of seconds to be adjusted between the two parties for a seamless authorization process.

For example, a user generates a JWT token at 6:00 AM, which is valid till 7:00 AM. Its nbf claim value is also 6:00 AM. That is, you can use the token only after 6:00 AM till 7:00 AM. In such a scenario, consider that the API Gateway is 120 seconds behind the authorization server. If you present the token at 6:01 AM of the authorization server that will be 5:59 AM of API Gateway, and hence the token validation will fail as the nbf claim condition is not satisfied.

Similarly, consider that API Gateway is 120 seconds ahead. The user generates a token at 6:00 AM and presents the token at 6:59 AM, then the token validation will fail as the expiry claim condition is not satisfied (as API Gateway's time is 7:01 already).

To overcome such scenarios, you can specify the time difference that has to be adjusted when validating tokens from external authorization servers.

In the examples given above, the token validations will pass if the time difference between the two parties is adjusted.

Note:

The value provided for the setting is applicable for all external authorization servers that you have configured with your API Gateway installation. If you want to provide an exclusive value for a server, then you need to provide the same using the **API Gateway > Administration > Security > JWT/OAuth/OpenID screen**.

pg_JWT_isHTTPS

Specifies whether the transport protocol over which the JSON Web Tokens (JWTs) are granted authorization is restricted to HTTPS.

Possible values:

Parameter and Description

- true. Restricts the transport protocol to HTTPS. This is set by default.
 - false. Allows HTTP and HTTPS transport protocols.
-

pg_oauth2_createDefaultScopes

Specifies whether default scopes must be created.

Possible values:

- true. When local authorization server is configured as the default authorization server, then API Gateway automatically creates a scope in Integration Server during API creation, and creates a scope mapping between the OAuth scope in the local authorization server and the API. This setting is useful in cases where a service is published from Centrasite.
 - false. API Gateway does not create a scope automatically. Users must manually create and map scopes for the services published from Centrasite. This is the default value.
-

pg_oauth2_isHTTPS

Specifies whether the transport protocol over which the OAuth 2.0 access tokens are granted authorization is restricted to HTTPS.

Possible values:

- true. Restricts the transport protocol to HTTPS. This is set by default.
 - false. Allows HTTP and HTTPS protocols.
-

pg_OpenID_isHTTPS

Specifies the transport protocol over which the OpenID (ID) tokens are granted authorization is restricted to HTTPS.

Possible values:

- true. Restricts the transport protocol to HTTPS. This is set by default.
 - false. Allows HTTP and HTTPS protocols.
-

pg_xslt_disableDoctypeDeclarations

Specifies whether the xslt doc type declarations must be disabled or not.

Possible values:

- true. Disables the xslt doc type declarations. This is the default value.
 - false. Enables the xslt doc type declarations.
-

pg_xslt_enableDOM

Specifies whether the DOM parsing must be enabled.

Parameter and Description

Possible values:

- `true`. Enables the DOM parsing.
- `false`. Disables the DOM parsing and enables other parsers.

pg_xslt_enableSecureProcessing

Specifies whether the use of extensions must be disabled.

Possible values:

- `true`. Enables the use of extensions. This is the default value.
- `false`. Disables the use of extensions.

pg.3pSnmpSender.sendDelay

This is an internal parameter. Do not modify.

pg.bulkhead.statusCode

Specifies the status code that is returned when rejecting unprocessed requests. When the number of concurrent requests to an API exceeds the configured bulkhead limit, then this status code is displayed.

If you do not specify any value, then the **503** status code is displayed.

pg.bulkhead.statusMessage

Specifies the message that is returned when rejecting unprocessed requests. When the number of concurrent requests to an API exceeds the configured bulkhead limit, then this status message is displayed.

If you do not specify any value, then the **Service unavailable** status message is displayed.

pg.cs.snmpTarget.base64Encoded

This is an internal parameter. Do not modify.

pg.cs.snmpTarget.connTimeout

Specifies the number of milliseconds before an inactive connection to the SNMP target server is closed. If set to 0, the socket remains open indefinitely.

pg.cs.snmpTarget.maxRequestSize

Specifies the maximum size (in bytes) for SNMP traps.

The default value is 10485760.

pg.cs.snmpTarget.retries

Parameter and Description

Specifies the number of times to resend SNMP traps upon failure.

Default value is 1.

This parameter works with **pg.cs.snmpTarget.sendTimeOut** to determine the delay in re-sending SNMP traps to malfunctioning SNMP servers (that is, it retries *sendTimeOut).

This means that if the retries parameter is set to 3, and the **sendTimeOut** parameter is set to 500 milliseconds, there is a 1.5 second delay before the thread sending the alert is available to send another event. Malfunctioning event destinations could delay the amount of time it takes API Gateway to report events, or it could cause discarded events when the queue reaches its maximum level.

pg.cs.snmpTarget.sendTimeOut

Specifies the time (in milliseconds) to wait before the SNMP trap times out because the server destination is not responding.

This value schedules a timer that resends an SNMP event that has not yet completed when it expires. You must set a timeout value that ensures that the trap is sent to the SNMP server within the time frame. This parameter does not abort an event that is in progress. Set this parameter higher than the default when sending traps with large payloads. The default value is 500.

This parameter works with **pg.cs.snmpTarget.retries** to determine the delay in resending SNMP traps to malfunctioning SNMP servers (that is, it retries *sendTimeOut).

This means that if the retries parameter is set to 3, and the sendTimeOut parameter is set to 500 milliseconds, there is a 1.5 second delay before the thread sending the alert is available to send another event. Malfunctioning event destinations could delay the amount of time it takes API Gateway to report events, or it could cause discarded events when the queue reaches its maximum level.

pg.default.enable.oldVersion

Specifies whether the oldest version of an API has to be enabled when the version of an API is not specified.

For example, when you have an API that is versioned, the client can specify the version number in the URL to invoke that specific version of the API, that is, API-NAME/Version-number. When the client does not specify the version number, API Gateway defaults to the latest version of the API and invokes it. You can use this parameter to change this behavior such that API Gateway defaults to the oldest version of API and invokes it.

Available values are:

- true. When you set this parameter as true and invoke an API without specifying the version number, then API Gateway defaults to the oldest version of the specified API and invokes it.

Parameter and Description

- `false`. This is the default value. When you set this parameter as `false` and invoke an API without specifying the version number, then API Gateway defaults to the latest version of the specified API and invokes it.

pg.endpoint.connectionTimeout

Specifies the time interval (in seconds) after which an HTTP connection attempt times out.

The default value is 30 seconds.

This is a global property that applies to the endpoints of all APIs. If you prefer to specify a connection timeout for the endpoints of an API individually, set the `HTTP Connection Timeout` parameter in the API's Routing Protocols processing step, which would then take precedence over `pg.endpoint.connectionTimeout`.

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.
- d. If you do not specify any value for `pg.endpoint.connectionTimeout`, then API Gateway uses the default value of 30 seconds.

pg.endpoint.readTimeout

Specifies the time interval (in seconds) after which a socket read attempt times out. Default value: 30 seconds.

This is a global property that applies to all APIs. If you prefer to specify a read timeout for APIs individually, set the Read Timeout field in the API's Routing Protocols processing step, which would then take precedence over `pg.endpoint.readTimeout`.

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

Parameter and Description

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.
-

pg.lb.failoverOnDowntimeErrorOnly

Specifies API Gateway's behavior of endpoints in a load-balanced routing scenario.

Possible values:

- `true`. Load balancing does not happen when the service fault encountered in the response is a downtime error. This is the default value.
For example, the following are some Downtime exceptions and for which fail overs happen:
`ConnectException`, `MalformedURLException`, `NoRouteToHostException`,
`ProtocolException`, `SocketTimeoutException`, `UnknownHostException`,
`UnknownServiceException`, Web Service not available, The service cannot be found.
 - `false`. Load balancing happens whenever a service fault is encountered in the response coming from endpoint 1 and API Gateway immediately tries the next configured endpoint. There is no distinction on the type of fault present in the response from endpoint.
-

pg.MTOMStreaming.cachedFiles.delete.interval

Specifies the interval (in seconds) to delete the cached MTOM files.

Default value is 3600 seconds. This property takes effect only when the **pg.suppress.IS.Icm** setting is set to `true`.

pg.nativeServer.validatePrivateIPs

Specifies whether to validate the native server endpoint against the private IP configuration during invocation of an API.

Possible values:

- `true`. If this is set to true, then the API invocation is not allowed if the native endpoint uses any private IPs other than the ones configured in `/rest/apigateway/configurations/whiteListingIPs`
- `false`. If this is set to false, no private IP validation is done for the native endpoint.

Default value is `false` for on-prem and `true` for cloud installations.

pg.overwrite.users.withSameLoginId

Parameter and Description

Specifies whether API Gateway must validate and overwrite a user based on their login ID or UUID when importing or promoting the user or team from one stage to another, provided the logged-in credentials (login ID) of both instances are the same.

Possible values:

- **true**. API Gateway validates and overwrites the user with the user's login ID, if the user is not matched with the UUID during import or promotion of a user or teams. The default value is **true**.
 - **false**. API Gateway validates and overwrites the user with the UUID.
-

pg.removeSSNID

Specifies whether to include the set-cookie header in the response header. The set-cookie header contains SSNID value sent from the native service.

Possible values:

- **true**. The client that invokes API Gateway API, does not receive the set-cookie header that contains ssnid value.
 - **false**. The client that invokes API Gateway API, receives the set-cookie header that contains ssnid value if the native service sends this in the response.
-

pg.runtime.extended.stackSize

Specifies the thread space size allocated for critical functions to run. You can configure the thread space size as required by setting a value for this setting. The size is specified in MB.

Default value: 20

If you provide an invalid value, API Gateway considers the default value.

pg.schema.cacheOnDeploy

Specified whether XML schemas that are required for schema validation of APIs must be pre-loaded for APIs that have the Validate API Specification policy with Schema enabled. The XML schemas are pre-loaded during the start-up or deployment in Mediator. This feature ensures that the first invocation of APIs take longer duration.

Possible values:

- **true**. If an API has the Validate API Specification policy with Schema enabled then the schema is loaded in the memory during API activation.
- **false**. The schema is loaded during the first invocation of API.

Default value is **false**.

pg.security.allowedhostnames

Parameter and Description

Validates the host header in an incoming request against the host names configured in this setting. Provide a comma-separated list of host names that you want the host header in the incoming request to be validated against.

This setting is required especially in scenarios where, when an API is invoked by adding a dummy host in the Host header. The invocation gets through the proxy API and API Gateway returns success response, which can pose as a vulnerability.

If the host in the request header is not in the list of host names configured in this setting, API Gateway returns a 400 Bad Request with an error response `Invalid Host` header.

If you do not include any host names and leave the setting blank, the host header in the incoming request is not validated.

pg.security.honourPortAccessModeSettings

Specifies whether the access mode settings configured in the **Administration > Security > Ports** section must be enforced on the HTTP and HTTPS ports.

In addition to the default port, you can add ports in API Gateway using which you can consume APIs. You can configure the access mode of the ports to determine whether a port can be used to access an API or not. You can either allow or deny the access of all APIs through a port. When you allow access of APIs using a port by default, you can specify a list of APIs that must be denied access over the port. Also, if you deny the access of APIs using a port, you can specify a list of APIs that can be allowed to access using the port. For details on creating ports and allowing or denying access to a port, see “[Ports](#)” on page 555.

Possible values:

- `true`. Access of REST and OData APIs through the HTTP and HTTPS ports is allowed or denied based on the access mode settings configured from the **Administration > Security > Ports** section.
- `false`. The access mode setting specified for the ports are not applied. This is the default value.

Prior 10.7, the access mode setting of ports was applicable only for the SOAP APIs. Starting 10.7, the access mode is applicable for REST and OData APIs as well. The default value of the **pg.security.honourPortAccessModeSettings** setting is `false`. That is, the access mode configuration is imposed only on SOAP APIs. So, the existing users need not modify their settings if they want the access mode setting to be enforced only on the SOAP APIs. However, if you want to enforce the access mode setting on REST and OData APIs, you can change the value of this setting to `true`.

pg.snmp.communityTarget.base64Encoded

Specifies whether to use a third-party SNMPv1 community-based connection.

The default value is `false`.

Parameter and Description

When this property is set to true, the Community name of 3rd Party SNMP destination configuration is expected as base64 encoded.

pg.snmp.communityTarget.maxRequestSize

Specifies the maximum size (in bytes) for SNMP traps.

The default value is 65535.

pg.snmp.communityTarget.retries

Specifies the number of times to resend SNMP traps upon failure.

Default value is 1.

This parameter works with **pg.snmp.communityTarget.sendTimeOut** to determine the delay in re-sending SNMP traps to malfunctioning SNMP servers (that is, it retries *sendTimeOut).

This means that if the retries parameter is set to 3, and the **sendTimeOut** parameter is set to 500 milliseconds, there is a 1.5 second delay before the thread sending the alert is available to send another event. Malfunctioning event destinations could delay the amount of time it takes API Gateway to report events, or it could cause discarded events when the queue reaches its maximum level.

pg.snmp.communityTarget.sendTimeOut

Specifies the time (in milliseconds) to wait before the SNMP trap times out because the server destination is not responding. This value schedules a timer that resends an SNMP event that has not yet completed when it expires. You must set a timeout value that ensures that the trap is sent to the SNMP server within the time frame. This parameter does not abort an event that is in progress. Set this parameter higher than the default when sending traps with large payloads.

The default value is 500.

This parameter works with **pg.snmp.communityTarget.retries** to determine the delay in re-sending SNMP traps to non-responsive SNMP servers (that is, it retries *sendTimeOut).

This means that if the retries parameter is set to 3, and the **sendTimeOut** parameter is set to 500 milliseconds, there is a 1.5 second delay before the thread sending the alert is available to send another event. Malfunctioning event destinations could delay the amount of time it takes API Gateway to report events, or it could cause discarded events when the queue reaches its maximum level.

pg.snmp.customTarget.connTimeout

Specifies the number of milliseconds before an inactive connection to the third-party SNMP server is closed. If set to 0, the socket remains open indefinitely.

pg.snmp.userTarget.maxRequestSize

Parameter and Description

Specifies the maximum size (in bytes) for SNMP traps.

The default value is 65535.

pg.snmp.userTarget.retries

Specifies the number of times to resend SNMP traps upon failure.

The default value is 1.

This parameter works with **pg.snmp.userTarget.sendTimeOut** to determine the delay in re-sending SNMP traps to malfunctioning SNMP servers (that is, it retries *sendTimeOut).

This means that if the retries parameter is set to 3, and the **sendTimeOut** parameter is set to 500 milliseconds, there is a 1.5 second delay before the thread sending the alert is available to send another event. Malfunctioning event destinations could delay the amount of time it takes API Gateway to report events, or it could cause discarded events when the queue reaches its maximum level.

pg.snmp.userTarget.sendTimeOut

Specifies the time (in milliseconds) to wait before the SNMP trap times out because the server destination is not responding. This value schedules a timer that resends an SNMP event that has not yet completed when it expires. You must set a timeout value that ensures that the trap is sent to the SNMP server within the time frame. This parameter does not abort an event that is in progress. Set this parameter higher than the default when sending traps with large payloads.

The default value is 500.

This parameter works with **pg.snmp.userTarget.retries** to determine the delay in resending SNMP traps to malfunctioning SNMP servers (that is, it retries *sendTimeOut).

This means that if the retries parameter is set to 3, and the **sendTimeOut** parameter is set to 500 milliseconds, there is a 1.5 second delay before the thread sending the alert is available to send another event. Malfunctioning event destinations could delay the amount of time it takes API Gateway to report events, or it could cause discarded events when the queue reaches its maximum level.

pg.soapToRest.typeConvertorEnabled

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. For example, if the XML is <number>10</number>, it is converted as "number" : 10.

Possible values:

- **true**. Values are converted to their primitive type. This is the default value.
 - **false**. Values are not converted to their primitive type.
-

Parameter and Description

pg.suppress.IS.lcm

Specifies whether to override IS lifecycle manager with API Gateway lifecycle manager.

Possible values:

- true. Set this to true to use the API Gateway lifecycle manager.
- false. Set this to false to use the IS lifecycle manager. By default, the value is false.

API Gateway lifecycle manager provides options to specify the interval for deleting of cached MTOM files using the **pg.MTOMStreaming.cachedFiles.delete.interval** setting.

pg.uddiClient.publish.maxThreads

Specifies the maximum allowed number of threads to publish the performance metrics data to CentraSite.

The default value is 2.

pg.uddiClient.uddiClientTimeout

Specifies the connection and read timeout, in milliseconds, for publishing performance metrics to CentraSite.

The default value is 5000.

pgmen.quotaSurvival.addLostIntervals

Specifies whether API Gateway must restore the time duration between the shutdown and restart when the Subscription counters are restored back into restarted instance.

Possible values are true or false.

pgmen.quotaSurvival.interval

Specifies the periodical time interval, in minutes, in which API Gateway should run to persist the Subscription counters into the database.

To keep the Subscription counters active during the server restart, API Gateway would periodically store the counters to the database.

Default value is 1.

portClusteringEnabled

Specifies whether the port configuration synchronization across API Gateway cluster nodes is enabled or disabled.

When as an API Gateway Administrator you create, update or delete a port definition, you might prefer to do it in one of the nodes in the cluster and have it instantly reflected on the other nodes without having to restart them. This setting controls whether you want to synchronize these port configuration changes across the API Gateway cluster nodes.

Parameter and Description

In this context clustered environment refers to clustering being enabled under **Administration > General > Clustering** section.

In a clustered environment, the possible values are:

- **true**. This enables port configuration synchronization across API Gateway cluster nodes. This is the default value
The ports that are not configured through the API Gateway UI (that is ports configured through Integration Server) are ignored by API Gateway UI and are removed on a restart.
- **false**. This disables port configuration synchronization across API Gateway cluster nodes. The ports that are not configured through the API Gateway UI (that is ports configured through Integration Server) are displayed in the API Gateway UI and they are not removed on a restart.

In a non-clustered environment, the possible values are:

- **true**. The ports that are not configured through the API Gateway UI (that is ports configured through Integration Server) are ignored by API Gateway UI and are removed on a restart.
- **false**. The ports that are not configured through the API Gateway UI (that is ports configured through Integration Server) are displayed in the API Gateway UI and they are not removed on a restart.

Note:

Restart all the cluster nodes, when you change this parameter, for the changes to take effect.

promotionListSortByTime

Specifies whether the list of promotions in the **Promotions** tab of the **Promotion management** page are sorted by the promotion time.

Possible values:

- **true**. Promotion list is sorted by promotion time with the latest on top.
- **false**. Promotion list is sorted by promotion name. This is the default value.

retainSOAPResponseStatus

Specifies whether the native SOAP API status has to be sent to the client.

Possible values:

- **true**. The native API responses retain the SOAP response status in the response sent to the API clients upon a failure. The default value is true.
- **false**. All response codes from the native SOAP API are converted to HTTP 500 (Internal Server Error) before sending the response to the API clients.

Parameter and Description

retainResponseStatus

Specifies whether the native service status has to be sent to the client.

Possible values:

- true. A 401 HTTP response received from the native API with outbound authentication is passed on to API clients.
- false. A 401 HTTP response received from the native API with outbound authentication is not exposed to the API clients. A HTTP response 500 is sent instead. This is the default value.

return408ForConnectionTimeout

Specifies the status code to be included in the response when a request to the native service times out.

Possible values:

- true. The response contains 408 error code uniformly when a request to the native service is timed out. This is the default value.
- false. The response does not contain 408 error code.

saveAuditlogsWithPayload

Specifies whether the audit logs have to be saved along with payload.

Possible values:

- true. The audit logs have to be saved along with payload. This is the default value.
- false. The audit logs are not saved along with payload.

sendClientRequestURI

Specifies whether the URI that is present in a request must be decoded before sending it to the native service.

Possible values:

- true. The URI is not decoded. The unicode characters in a request are encoded.
- false. The URI is decoded without change in the path of the URL. This is the default value.

setDefaultContentType

Specifies that default content type to be included in the GET and DELETE methods, if the content type is missing in request.

Possible values:

Parameter and Description

- true. The default content type **application/x-www-form-urlencoded** is added, if content type is missing in request. This is the default value.
 - false -The default content type is not added. The content-type is sent as is.
-

startDayOfTheWeek

Specifies the start day of a week for the unit of measurement of the **Alert Interval** configured, to monitor performance, before sending an alert.

By default, the start day of the week is set to Monday.

strictResourceMatching

Specifies whether API Gateway must perform a strict matching of the resource path from runtime invocation with the API definition of resource paths.

Possible values:

- true. API Gateway uses the strict resource matching criteria and the matching fails if it encounters any other characters than that are specified. This is the default value.
 - false. API Gateway matches the best resource instead of using the strict resource matching criteria.
-

tagsTypeAheadSearchResultSize

Specifies the number of existing tags to display during the type ahead search.

This is available while adding a tag to an API, where you can type a search term. A list of existing API tags appears depending on the search term. The number of API tags displayed in this list is restricted as per the value provided in the **tagsTypeAheadSearchResultSize** property.

The default value is 10.

The minimum value you can provide is 1. If you provide zero or an invalid value, the value of this property is set to the default value 10.

transferEncodingChunked_handleAsStream

Specifies whether the request and response payloads should be handled as stream. If you have applied the Validate API Specification policy enabled with schema validation, and a payload is sent as a stream then API Gateway does not validate the payload.

Possible values:

- true. API Gateway checks for the following conditions to handle the request and response payloads as stream:
 - Request payload is handled as stream, if the Transfer-encoding: chunked header is sent in the request.

Parameter and Description

- Response payload is handled as stream, if the `application/octet-stream` accept header is sent in the request.
- `false`. The payloads are not handled as streams (even though the conditions specified above are met). This is the default value.

Note:

The response from the native endpoint need not contain the `Transfer-encoding: chunked` header for the response payload to be handled as stream.

transformerPoolSize

Specifies the maximum size for transform pool that consists XSLT transformers. To reduce performance impacts, you can reuse the transformers from the pool instead of creating them for every request.

useTypeInIndexNameForESDestination

Specifies the index format used when sending data from API Gateway to a configured Elasticsearch destination.

The default value is `true`.

Elasticsearch verison 7.2 supports data with dedicated index for each of the event types. So, API Gateway sends data in different indexes for different event types if the value of this property is set to `true`. The event type is concatenated with the Elasticsearch destination index name. That is, the index name will be in the following format: `{IndexName}_{Event Type}`. For example, `database_transactionalevents`; where `default` is the index name and `transactional events` in the event type.

If the value is set to `false`, the type name is not concatenated with the index name. In this case, the index created for each event is sub-indexed under a main index when data is sent to configured Elasticsearch destination. The value of this setting must be set to `false`, if the version of the destination Elasticsearch is 5.x or earlier.

validateTransportProtocolAgainstEGPort

Specifies whether the EG protocol in the transport protocol policy of APIs must be validated during the reverse invoke of APIs.

Possible values:

- `true`. Validation is performed to ensure an EG protocol is specified in transport protocol policy of APIs that are invoked using the reverse invoke method.
- `false`. No validation is performed. The default value is `false`.

wsdlPortLayout

Specifies which ports entries are exposed in a WSDL.

Parameter and Description

Possible values:

- `service-port`. All the port entries are exposed. This is the default value.
- `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.
- `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.

`xmlToJSONConversion_keepString`

Specifies whether the data type of fields in a response payload, after XML to JSON conversion, must be *string* or a corresponding data type.

The default value is *true*.

Possible values:

- *true*. The data type of the fields are retained as *string*.
- *false*. The data type of the fields is changed to a corresponding data type.

For example:

Sample XML payload:

```
<?xml version="1.0" encoding="UTF-8"?>
<EmployeeDetail>
  <ID>12345</ID>
  <Name>John</Name>
  <Address>999 ABC Street</Address>
</EmployeeDetail>
```

- Sample response, if you set the extended setting to *true*:

```
{
  "EmployeeDetail": {
    "ID": "12345",
    "Name": "John",
    "Address": "999 ABC Street"
  }
}
```

All fields in the response payload are retained as *string*.

- Sample response, if you set the extended setting to *false*:

```
{
  "EmployeeDetail": {
    "ID": 12345,
    "Name": "John",
    "Address": "999 ABC Street"
  }
}
```

Parameter and Description

The data type of the field *ID* is *integer*.

5. You can configure the watt parameters in the Watt keys section by providing the required values.

The configured watt keys are listed under Watt settings below the Extended keys at the top of the page.

Only the following watt parameters get synchronized across nodes in a cluster setup:

- watt.security.ssl.client.ignoreEmptyAuthoritiesList
- watt.security.ssl.ignoreExpiredChains
- watt.server.url.alias.partialMatching
- watt.server.oauth.authServer.alias
- watt.server.oauth.requireHTTPS
- watt.net.http401.throwException
- watt.net.http501-599.throwException
- watt.server.SOAPMTOMStreaming.enable
- watt.server.http.Strict-Transport-Security
- watt.server.rest.removeInputVariablesFromResponse
- watt.server.coder.responseAsXML
- watt.security.ssl.cacheClientSessions
- watt.server.enterprisegateway.ignoreXForwardedForHeader

If you modify the other watt parameters in one API Gateway instance they do not get synchronized across other nodes in a cluster setup. You have to manually modify them in the other instances.

6. Click **Save**.

Configuring API Fault Settings

You must have the API Gateway's manage user administration functional privilege assigned to configure the API fault settings.

You can configure global error responses for all APIs to display the default error message.

➤ To configure the service fault settings

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > API fault**.
3. Select **Send native provider fault** to send the native API provider's fault content, if available. API Gateway ignores the default error message and sends whatever content it receives from the native API provider.

If you do not select this option then API Gateway sends the default error message.

4. Specify the error message text in the **Default error message** text box that is sent out when the **Send native provider fault** is not selected.
5. Click **Save**.

Approval Configuration

API Gateway allows you to configure an approval process to ensure that the requests are valid. If the requests are invalid, API Gateway enables approvers to reject the requests. API Gateway allows you to configure approvals for:

- Create application: To enforce approvals for creating an application in API Gateway.
- Register application: To enforce approvals for associating an application with APIs.
- Update application: To enforce approvals for modifying an application.
- Subscribe package: To enforce approvals in API Gateway to enable API Portal consumers for subscribing a package to a plan in API Portal.
- Update subscription: To enforce approvals for modifying a subscription.
- Delete subscription: To enforce approvals for deleting a subscription.
- Change owner/teams: To enforce approvals for ownership of assets and association of teams.

In API Gateway, you can create an application and associate (register) the application created with APIs. If you have the API Gateway's manage general administration configurations functional privilege assigned, you can configure approvals for creating or registering an application. If you have configured approvers, and if a user wants to create and register an application in API Gateway, then the application is created and registered with an API only if any one approver from the approvers group approves the create application and the register application requests. Similarly, you can configure approvals for updating an application or subscribing a package.

You can configure a set of different approvers for creating or registering applications. For example, if you have configured an approvers group, *group1* to approve or reject a create application request and an approvers group, *group2* to approve or reject a register application request, then when a user creates and registers an application in API Gateway, then the create application request must be approved by any one approver in *group1*. When the application is created, the register application request is sent to the approvers in *group2* and this register application request must be approved

by any one approver in *group2*. When the request is approved, the application created is registered to an API. However, if any one user from *group2* rejects the request, then the application gets created but is not registered to an API.

API Gateway approvals can also be used when API Gateway is integrated with API Portal and an API Portal API consumer uses the API get access token to register an application to an API in API Gateway. In this scenario, API Portal implicitly sends a request to API Gateway to create an application. When the approvers approve the create application request, an application is created in API Gateway and then API Gateway initiates a register application request and the approvers approve the register application request. The application is registered to an API which is published to API Portal. The consumer is now able to view and manage the API in API Portal.

Note:

You can customize the Approval page view by modifying the file OAuth_Approval.html located at *SAG_Root\IntegrationServer\instances\instance_name\packages\WmAPIGateway\templates*.

Similarly, you can configure approvals for subscribing a package in API Gateway, when an API consumer subscribes to a package in API Portal. API Gateway receives the package subscription request and the approvers in the approvers group approve or reject the request for subscribing a package. When the request is approved, the acknowledgement is sent to API Portal and the package is subscribed.

Configuring Approvals for Creating an Application

You have to configure the approval settings, to enforce approval for creating an application in API Gateway.

➤ To configure approvals for creating an application



1. Expand the menu options icon in the title bar, and select **Administration**.
2. Select **General > Approval configuration > Create application**.
3. Set the **Enable** toggle button to the on position to enable approval configuration to take effect.
4. Select the team of approvers from the **Approvers** drop-down list.
5. Select *Anyone* from the **Approved by** drop-down list.

This implies that, any one user of the selected team can approve or reject the requests. The requester need not wait for the approval of each approver in the approvers group.

Note:

If a user is associated with the selected team, then the user can approve or reject a pending request even if the user is not associated with the **Approvers** group.

6. Select the **Team approvers** option to allow the team approvers to approve the pending requests of applications that the team is associated with.

You can specify the users and user groups (Team approvers) in the **Approvers** section when creating or editing the team details. The Team approvers list will also include the users and user groups specified in the **Team Administrators** section, if the **Include team administrators as approvers** option is selected.

7. Select the **Disable auto-approval of the requestor** option to specify that applications cannot be automatically approved when they are created by users who have the privilege to approve application creation requests.

If you do not select this option, then the applications created by users who have the required privileges are automatically approved.

8. Select **Configure approval initiate request mail template to be sent to approver**.

This is to configure the email template to be sent to the approver, for approving the request of creating an application.

9. Provide the following information in the **Configure approval initiate request mail template to be sent to approver** section:

Field	Description
Send notification	To send an email notification to the approver to approve the request for creating an application.
Subject	The subject line of the email to be sent.
Content	By default, the template appears. You can customize the email content. Note: The @ character acts as a place holder and the values are automatically generated by the system. For example, Hello @approver.name appears as Hello Joe in the email sent, where Joe is the approvers login ID. For more infomration on supported variables see, " Email templates variable " on page 373.

Note:

The email notifications are sent only to the local API Gateway users.

10. Select **Configure request approved mail template to be sent to requester**.

This is to configure the email template to be sent to the requester to notify that the request for creating an application is approved.

11. Provide the following information in the **Configure request approved mail template to be sent to requester** section:

Field	Description
Send notification	To send an email notification to the requester that the request for creating an application is approved by the approver.
Subject	The subject line of the email to be sent.
Content	<p>By default, the template appears. You can customize the email content.</p> <p>Note: The @ character acts as a place holder and the values are automatically generated by the system. For example, Approval of @event.type appears as Approval of Create Application in the email sent, where Create Application is the event.type. For more information on supported variables see, "Email templates variable" on page 373.</p>

12. Select **Configure rejection mail template to be sent to requester**.

This is to configure the email template to be sent to the requester to notify that the request for creating an application is rejected.

13. Provide the following information in the **Configure rejection mail template to be sent to requester** section:

Field	Description
Send notification	To send an email notification to the requester that the request for creating an application is rejected by the approver.
Subject	The subject line of the email to be sent.
Content	<p>By default, the template appears. You can customize the email content.</p> <p>Note: The @ character acts as a place holder and the values are automatically generated by the system. For example, Approval of @event.type appears as Approval of Create Application in the email sent, where Create Application is the event.type. For more information on supported variables see, "Email templates variable" on page 373.</p>

14. Click **Cancel** to revert to the last saved changes or to abandon all the changes if the values are not saved.

15. Click **Save**.

Configuring Approvals for Registering Application

You have to configure the approval settings, to enforce approval for associating an application with APIs in API Gateway. The API Portal API get access token request is considered as a create and registering application event in API Gateway.

➤ To configure approvals for registering an application

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Approval configuration > Register application**.
3. Set the **Enable** toggle button to the on position to enable approval configuration to take effect.
4. Select the team of approvers from the **Approvers** drop-down list.
5. Select the *Anyone* from the **Approved by** drop-down list.

This implies that, any user of the selected team can approve or reject the requests. The requester need not wait for the approval of each approver in the approvers group.

Note:

If a user is associated with the selected team, then the user can approve or reject a pending request even if the user is not associated with the **Approvers** group.

6. Select the **Team approvers** option to allow the approvers of the team that owns the API being registered.

You can specify the users and user groups (Team approvers) in the **Approvers** section when creating or editing the team details. The Team approvers list will also include the users and user groups specified in the **Team Administrators** section, if the **Include team administrators as approvers** option is selected.

7. Select the **Disable auto-approval of the requestor** option to specify that applications cannot be automatically approved when they are created by users who have the privilege to approve application creation requests.

If you do not select this option, then the applications created by users who have the required privileges are automatically approved.

8. Select **Configure approval initiate request mail template to be sent to approver**.

This is to configure the email template to be sent to the approver for associating an application with APIs.

9. Provide the following information in the **Configure approval initiate request mail template to be sent to approver** section:

Field	Description
Send notification	To send an email notification to the approver to approve the request for associating an application with APIs.
Subject	The subject line of the email to be sent.
Content	<p>By default, the template appears. You can customize the email content.</p> <p>Note: The @ character acts as a place holder and the values are automatically generated by the system. For example, Hello @approver.name appears as Hello Joe in the email sent, where Joe is the approvers login ID.</p>

Note:

The email notifications are sent only to the local API Gateway users.

10. Select **Configure request approved mail template to be sent to requester**.

This is to configure the email template to be sent to the requester to notify that the request for associating an application with APIs is approved.

11. Provide the following information in the **Configure request approved mail template to be sent to requester** section:

Field	Description
Send notification	To send an email notification to the requester that the request for associating an application with APIs is approved by the approver.
Subject	The subject line of the email to be sent.
Content	<p>By default, the template appears. You can customize the email content.</p> <p>Note: The @ character acts as a place holder and the values are automatically generated by the system. For example, Approval of @event.type appears as Approval of Register Application in the email sent, where Register Application is the event.type.</p>

12. Select **Configure rejection mail template to be sent to requester**.

This is to configure the email template to be sent to the requester to notify that the request for registering an application is rejected.

13. Provide the following information in the **Configure rejection mail template to be sent to requester** section:

Field	Description
Send notification	To send an email notification to the requester that the request for associating an application with APIs is rejected by the approver.
Subject	The subject line of the email to be sent.
Content	By default, the template appears. You can customize the email content. Note: The @ character acts as a place holder and the values are automatically generated by the system. For example, Approval of @event.type appears as Approval of Register Application in the email sent, where Register Application is the event.type.

14. Click **Cancel** to revert to the last saved changes or to abandon all the changes if the values are not saved.
15. Click **Save**.

Configuring Approvals for Updating Application

You have to configure the approval settings, to enforce approval for modifying an existing application in API Gateway. The subscription usage is counted against the existing plan and package combination until an approver approves the change of plan or package request. Once an approver approves the change request, quota is reset.

➤ To configure approvals for updating an application

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Approval configuration > Update application**.
3. Set the **Enable** toggle button to the on position to enable approval configuration to take effect.
4. Select the team of approvers from the **Approvers** drop-down list.
5. Select **Anyone** from the **Approved by** drop-down list.

This implies that, any one user associated with the selected team can approve or reject the requests. The requester need not wait for the approval of each approver in the approvers group.

Note:

If a user is associated with the team, then the user can approve or reject a pending request.

- Select the **Team approvers** option to allow the team approvers to approve the pending requests of applications that the team is associated with.

You can specify the users and user groups (Team approvers) in the **Approvers** section when creating or editing the team details. The Team approvers list will also include the users and user groups specified in the **Team Administrators** section, if the **Include team administrators as approvers** option is selected.

- Select the **Disable auto-approval of the requestor** option to specify that applications cannot be automatically approved when they are created by users who have the privilege to approve application creation requests.

If you do not select this option, then the applications created by users who have the required privileges are automatically approved.

- Select **Configure approval initiate request mail template to be sent to approver**.

This is to configure the email template to be sent to the approver for approving the request of modifying an existing application.

- Provide the following information in the **Configure approval initiate request mail template to be sent to approver** section:

Field	Description
Send notification	To send an email notification to the approver to approve the request for modifying an existing application.
Subject	The subject line of the email to be sent.
Content	<p>By default, the template appears. You can customize the email content.</p> <p>Note: The @ character acts as a place holder and the values are automatically generated by the system. For example, Hello @approver.name appears as Hello Joe in the email sent, where Joe is the approvers login ID.</p>

Note:

The email notifications are sent only to the local API Gateway users.

- Select **Configure request approved mail template to be sent to requester**.

This is to configure the email template to be sent to the requester to notify that the request for modifying an existing application is approved.

- Provide the following information in the **Configure request approved mail template to be sent to requester** section:

Field	Description
Send notification	To send an email notification to the requester that the request for modifying an existing application is approved by the approver.
Subject	The subject line of the email to be sent.
Content	By default, the template appears. You can customize the email content. Note: The @ character acts as a place holder and the values are automatically generated by the system. For example, Approval of @event.type appears as Approval of Update Application in the email sent, where Update Application is the event.type.

12. Select **Configure rejection mail template to be sent to requester**.

This is to configure the email template to be sent to the requester to notify that the request for modifying an existing application is rejected.

13. Provide the following information in the **Configure rejection mail template to be sent to requester** section:

Field	Description
Send notification	To send an email notification to the requester that the request for modifying an existing application is rejected by the approver.
Subject	The subject line of the email to be sent.
Content	By default, the template appears. You can customize the email content. Note: The @ character acts as a place holder and the values are automatically generated by the system. For example, Approval of @event.type appears as Approval of Update Application in the email sent, where Update Application is the event.type.

14. Click **Cancel** to revert to the last saved changes or to abandon all the changes if the values are not saved.

15. Click **Save**.

Configuring Approvals for Subscribing Package

You have to configure the approval settings in API Gateway, to enforce approval for subscribing a package to a plan in API Portal.

➤ To configure approvals for subscribing a package

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Approval configuration > Subscribe package**.
3. Set the **Enable** toggle button to the on position to enable approval configuration to take effect.
4. Select the team of approvers from the **Approvers** drop-down list.
5. Select *Anyone* from the **Approved by** drop-down list.

This implies that, any one user associated with the team can approve or reject the requests. The requester need not wait for the approval of each approver in the approvers group.

Note:

If a user is associated with the selected team, then the user can approve or reject a pending request even if the user is not associated with the **Approvers** group.

6. Select the **Team approvers** option to allow the team approvers to approve the pending requests of applications that the team is associated with.

You can specify the users and user groups (Team approvers) in the **Approvers** section when creating or editing the team details. The Team approvers list will also include the users and user groups specified in the **Team Administrators** section, if the **Include team administrators as approvers** option is selected.

7. Select the **Disable auto-approval of the requestor** option to specify that applications cannot be automatically approved when they are created by users who have the privilege to approve application creation requests.

If you do not select this option, then the applications created by users who have the required privileges are automatically approved.

8. Select **Configure approval initiate request mail template to be sent to approver**.

This is to configure the email template to be sent to the approver for subscribing a package.

9. Provide the following information in the **Configure approval initiate request mail template to be sent to approver** section:

Field	Description
Send notification	To send an email notification to the approver to approve the request for subscribing a package.
Subject	The subject line of the email to be sent.

Field	Description
Content	<p>By default, the template appears. You can customize the email content.</p> <p>Note: The @ character acts as a place holder and the values are automatically generated by the system. For example, Hello @approver.name appears as Hello Joe in the email sent, where Joe is the approvers login ID.</p>

Note:

The email notifications are sent only to the local API Gateway users.

10. Select **Configure request approved mail template to be sent to requester**.

This is to configure the email template to be sent to the requester to notify that the request for subscribing a package is approved.

11. Provide the following information in the **Configure request approved mail template to be sent to requester** section:

Field	Description
Send notification	To send an email notification to the requester that the request for subscribing a package is approved by the approver.
Subject	The subject line of the email to be sent.
Content	<p>By default, the template appears. You can customize the email content.</p> <p>Note: The @ character acts as a place holder and the values are automatically generated by the system. For example, Approval of @event.type appears as Approval of Subscribe Package in the email sent, where Subscribe Package is the event.type.</p>

12. Select **Configure rejection mail template to be sent to requester**.

This is to configure the email template to be sent to the requester to notify that the request for subscribing a package is rejected.

13. Provide the following information in the **Configure rejection mail template to be sent to requester** section:

Field	Description
Send notification	To send an email notification to the requester that the request for registering an application is rejected by the approver.

Field	Description
Subject	The subject line of the email to be sent.
Content	By default, the template appears. You can customize the email content. Note: The @ character acts as a place holder and the values are automatically generated by the system. For example, Approval of @event.type appears as Approval of Subscribe Package in the email sent, where Subscribe Package is the event.type.

14. Click **Cancel** to revert to the last saved changes or to abandon all the changes if the values are not saved.
15. Click **Save**.

Configuring Approvals for Updating Subscription

You have to configure the approval settings, to enforce approval to modify an existing subscription in API Gateway. When you request an approval for subscription update, the existing package and plan are in use, until an approver approves your request. Once the approver approves the change of subscription request, the quota is reset.

➤ To configure approvals for updating a subscription

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Approval configuration > Update subscription**.
3. Set the **Enable** toggle button to the on position to enable approval configuration to take effect.
4. Select the team of approvers from the **Approvers** drop-down list.
5. Select **Anyone** from the **Approved by** drop-down list.

This implies that, any user associated with the selected team can approve or reject the requests. The requester need not wait for the approval of each approver in the approver's group.

Note:

If a user is associated with the team, then the user can approve or reject a pending request.

6. Select the **Team approvers** option to allow the team approvers to approve the pending requests of applications that the team is associated with.

You can specify the users and user groups (Team approvers) in the **Approvers** section when creating or editing the team details. The Team approvers list will also include the users and user groups specified in the **Team Administrators** section, if the **Include team administrators as approvers** option is selected.

7. Select the **Disable auto-approval of the requestor** option to specify that applications cannot be automatically approved when they are created by users who have the privilege to approve application creation requests.

If you do not select this option, then the applications created by users who have the required privileges are automatically approved.

8. Select **Configure approval initiate request mail template to be sent to approver**.

This is to configure the email template to be sent to the approver for approving the request of modifying an existing subscription.

9. Provide the following information in the **Configure approval initiate request mail template to be sent to approver** section:

Field	Description
Send notification	To send an email notification to the approver to approve the request for modifying an existing subscription.
Subject	The subject line of the email to be sent.
Content	By default, the template appears. You can customize the email content. Note: The @ character acts as a place holder and the values are automatically generated by the system. For example, Hello @approver.name appears as Hello Joe in the email sent, where Joe is the approver's login ID.

Note:

The email notifications are sent only to the local API Gateway users.

10. Select **Configure request approved mail template to be sent to requester**.

This is to configure the email template to be sent to the requester to notify that the request for modifying an existing subscription is approved.

11. Provide the following information in the **Configure request approved mail template to be sent to requester** section:

Field	Description
Send notification	To send an email notification to the requester that the request for modifying an existing subscription is approved by the approver.
Subject	The subject line of the email to be sent.
Content	<p>By default, the template appears. You can customize the email content.</p> <p>Note: The @ character acts as a place holder and the values are automatically generated by the system. For example, Approval of @event.type appears as Approval of Update subscription in the email sent, where Update subscription is the event.type.</p>

12. Select **Configure rejection mail template to be sent to requester**.

This is to configure the email template to be sent to the requester to notify that the request for modifying an existing subscription is rejected.

13. Provide the following information in the **Configure rejection mail template to be sent to requester** section:

Field	Description
Send notification	To send an email notification to the requester that the request for modifying an existing subscription is rejected by the approver.
Subject	The subject line of the email to be sent.
Content	<p>By default, the template appears. You can customize the email content.</p> <p>Note: The @ character acts as a place holder and the values are automatically generated by the system. For example, Approval of @event.type appears as Approval of Update subscription in the email sent, where Update subscription is the event.type.</p>

14. Click **Save**.

Configuring Approvals for Deleting Subscription

You have to configure the approval settings, to enforce approval to delete an existing subscription in API Gateway. When you raise a deletion request, the application is suspended automatically. If an approver approves the deletion request, the subscription is deleted. However, if the approver rejects the request, subscription is enabled. When the application is suspended, if a user invokes the application, the quota usage for that application is continued to be calculated. This is a known issue and would be resolved in subsequent releases.

➤ To configure approvals for updating a subscription

1. Expand the menu options icon , in the title bar, and select **Administration**.
2. Select **General > Approval configuration > Delete subscription**.
3. Set the **Enable** toggle button to the on position to enable approval configuration to take effect.
4. Select the team of approvers from the **Approvers** drop-down list.
5. Select **Anyone** from the **Approved by** drop-down list.

This implies that, any user associated with the selected team can approve or reject the requests. The requester need not wait for the approval of each approver in the approver's group.

Note:

If a user is associated with the team, then the user can approve or reject a pending request.

6. Select the **Team approvers** option to allow the team approvers to approve the pending requests of applications that the team is associated with.

You can specify the users and user groups (Team approvers) in the **Approvers** section when creating or editing the team details. The Team approvers list will also include the users and user groups specified in the **Team Administrators** section, if the **Include team administrators as approvers** option is selected.

7. Select the **Disable auto-approval of the requestor** option to specify that applications cannot be automatically approved when they are created by users who have the privilege to approve application creation requests.

If you do not select this option, then the applications created by users who have the required privileges are automatically approved.

8. Select **Configure approval initiate request mail template to be sent to approver**.

This is to configure the email template to be sent to the approver for approving the request of deleting an existing subscription.

9. Provide the following information in the **Configure approval initiate request mail template to be sent to approver** section:

Field	Description
Send notification	To send an email notification to the approver to approve the request for deleting an existing subscription.
Subject	The subject line of the email to be sent.

Field	Description
Content	<p>By default, the template appears. You can customize the email content.</p> <p>Note: The @ character acts as a place holder and the values are automatically generated by the system. For example, Hello @approver.name appears as Hello Joe in the email sent, where Joe is the approver's login ID.</p>

Note:

The email notifications are sent only to the local API Gateway users.

10. Select **Configure request approved mail template to be sent to requester**.

This is to configure the email template to be sent to the requester to notify that the request for deleting an existing subscription is approved.

11. Provide the following information in the **Configure request approved mail template to be sent to requester** section:

Field	Description
Send notification	To send an email notification to the requester that the request for deleting an existing subscription is approved by the approver.
Subject	The subject line of the email to be sent.
Content	<p>By default, the template appears. You can customize the email content.</p> <p>Note: The @ character acts as a place holder and the values are automatically generated by the system. For example, Approval of @event.type appears as Approval of Delete subscription in the email sent, where Delete subscription is the event.type.</p>

12. Select **Configure rejection mail template to be sent to requester**.

This is to configure the email template to be sent to the requester to notify that the request for deleting an existing subscription is rejected.

13. Provide the following information in the **Configure rejection mail template to be sent to requester** section:

Field	Description
Send notification	To send an email notification to the requester that the request for deleting an existing subscription is rejected by the approver.

Field	Description
Subject	The subject line of the email to be sent.
Content	<p>By default, the template appears. You can customize the email content.</p> <p>Note: The @ character acts as a place holder and the values are automatically generated by the system. For example, Approval of @event.type appears as Approval of Delete subscription in the email sent, where Delete subscription is the event.type.</p>

14. Click **Save**.

How Do I Configure the Approval Process for Ownership Change of Assets?

If you want to enforce an approval process, configure and enable the approval process for ownership change of assets. The approver can approve or reject the request.

Before you begin

Ensure that you have Administrator privileges.

➤ To configure the approval process for ownership changes of assets

1. On the title bar, expand the menu options icon  and select **Administration**.
2. Select **General > Change owner/teams**.
3. Set the **Enable** toggle button to the on position .
4. Select the team of approvers from the **Approvers** list.
5. Select **Anyone** in the **Approved by** list.

This specifies that any user associated with the approvers' access profile specified in the **Approvers** list can approve or reject the requests.



6. Select the **Team approvers** option to allow team approvers to approve the ownership change requests of assets that the team is associated with.

You can specify the users and user groups (Team approvers) in the **Approvers** section when creating or editing the team details. The Team approvers list will also include the users and

user groups specified in the **Team Administrators** section, if the **Include team administrators as approvers** option is selected.

7. Select the **Disable auto-approval of the requestor** option to specify that applications cannot be automatically approved when they are created by users who have the privilege to approve application creation requests.

If you do not select this option, then the applications created by users who have the required privileges are automatically approved.

8. In the Configure approval initiate request mail template to be sent to the approver section, provide the following information:

- Select **Send notification** to send an email notification to the approver for the pending approval.
- Provide the text to display in the subject line and the body of the email.

Configure approval initiate request mail template to be sent to approver

Send notification

Subject

Approval request pending

Content

Hello @approver.name,

A request by @requestor.name to @event.type needs your review and approval.

Best Regards,
API Gateway Team

*** This notification was sent automatically. Do not reply to this email.***

Note:

The at sign (@) character acts as a place holder and API Gateway automatically generates the values. For example, Hello @approver.name appears as Hello Joe in the email sent, where Joe is the approvers' login ID.

9. In the Configure request approved mail template to be sent to the requester section, provide the following information:

- Select **Send notification** to send an email notification to the approval requester.
- Provide the text to display in the subject line and the body of the email.

 [Configure request approved mail template to be sent to requester](#)

Send notification

Subject

Approval of @event.type

Content

Congratulations @requestor.name !

Your request for @event.type has been approved.

*** This notification was sent automatically. Do not reply to this email.***

Note:

The at sign (@) character acts as a place holder and API Gateway automatically generates the values. For example, Approval of @event.type appears as Approval of Change ownership in the email sent, where Change ownership is the event.type.

10. In the Configure rejection mail template to be sent to the requester section, provide the following information:

- Select **Send notification** to send an approval rejection notification to the requester.
- Provide the text to display in the subject line and the body of the email.

 [Configure rejection mail template to be sent to requester](#)

Send notification

Subject

Rejection of @event.type

Content

Hello @requestor.name,

Your @event.type request has been rejected.

Reasons:@rejectionReason.

Best Regards,
API Gateway Team

*** This notification was sent automatically. Do not reply to this email.***

Note:

The at sign (@) character acts as a place holder and API Gateway automatically generates the values. For example, Rejection of @event.type appears as Rejection of change of ownership of an asset in the email.

11. Click **Save**.

Email templates variable

Approval Email Variables

@application.name	@approver.name
@event.type	@application.name
@requestor.name	@event.type
@requestor.email	@requestor.name
@requestor.firstName	@requestor.email
@requestor.lastName	@requestor.firstName
	@requestor.lastName

Rejection Email Variables

@application.name	@rejectionReason
@event.type	@application.name
@requestor.name	@event.type
@requestor.email	@requestor.name
@requestor.firstName	@requestor.email
@requestor.lastName	@requestor.firstName
	@requestor.lastName

Approval Requests

You can manage your pending requests in the Pending Requests section. You can approve or reject a pending request or delete the requests approved by you.

The Pending Requests section displays the list of requests that you can approve. The following table lists the request types and the users who can view them:

Request types	Users who can view and approve requests
Create application	■ Team approvers. Approvers from the teams that the application is associated with. You can specify the team members who can approve the pending requests in the Approvers section of the Basic information tab when creating or editing team details.
Update application	

Request types	Users who can view and approve requests
Update subscription	<ul style="list-style-type: none"> ■ Team administrators. Administrators of the team that the application is associated with. This includes the list of the users and user groups specified in the Team Administrators section of the Basic information tab when creating or editing team details. The team administrators can approve the pending requests only when you have selected the Include team administrators as approvers option.
Delete subscription	
Register application	<ul style="list-style-type: none"> ■ Team approvers. Approvers from the teams that the API is associated with. You can specify the team members who can approve the pending requests in the Approvers section of the Basic information tab when creating or editing team details. ■ Team administrators. Administrators of the team that the API is associated with. This includes the list of the users and user groups specified in the Team Administrators section of the Basic information tab when creating or editing team details. The team administrators can approve the pending requests only when you have selected the Include team administrators as approvers option.
Subscribe package	<ul style="list-style-type: none"> ■ Team approvers. Approvers from the teams that the package is associated with. You can specify the team members who can approve the pending requests in the Approvers section of the Basic information tab when creating or editing team details. ■ Team administrators. Administrators of the team that the package is associated with. This includes the list of the users and user groups specified in the Team Administrators section of the Basic information tab when creating or editing team details. The team administrators can approve the pending requests only when you have selected the Include team administrators as approvers option.

➤ To approve or reject a pending request

1. Expand the menu options icon  in the title bar, and select **Pending Requests**.

The Pending requests page appears.

2. Select **Pending Requests**.

A list of pending requests appears with the following information:

Field	Description
Requested by	The name of the requestor.
Requestor comments	The additional information or comments added by the requestor.
Event	The type of event for which the request is pending.
Request details	The details of the type of event requested.

3. Click the **Approve** or **Reject** icon to approve or reject requests.

Alternatively, you can select multiple pending requests to be approved or rejected simultaneously by selecting the check boxes adjacent to the names of the requests.

The request gets removed from the Pending requests page.

Deleting Requests

When you perform a task, for example, you create an application in API Gateway, then an approval request is generated if an approval is configured in API Gateway and this request that is waiting for approvers approval is listed in the Pending Request section. If you want to delete this request, you can delete it from the Pending Request section.

➤ To delete a request

1. Expand the menu options icon  in the title bar, and select **Pending Requests**.
 2. Select **My requests**.
- A list of requests appears with the detailed information of the request.
3. Click the **Delete** icon for the request that has to be deleted.
 4. Click **Yes** in the confirmation dialog.

Outbound Proxy

When API Gateway executes a request against a remote server, it issues an HTTP or HTTPS request to the specified target server. If your API Gateway instance is behind a firewall, and must route these HTTP or HTTPS requests through a third party proxy server, you can configure proxy servers to which API Gateway routes these requests.

For API Gateway to use a proxy server, you must define a proxy server alias. The proxy server alias identifies a proxy server and a port on the server through which you want to route requests. You can configure API Gateway to route requests to one or more proxy server aliases for each type of outbound requests (HTTP, HTTPS).

All the configured and available proxy server aliases are listed in a table with the corresponding details of the port being used, the host on which it is, the protocol used, and so on. To use a proxy server you have to enable it.

Note:

Any modifications to the outbound proxy in Integration Server may not reflect in API Gateway UI. Hence, Software AG recommends that you do not configure or modify outbound proxy through Integration Server Administrator UI.

Configuring Proxy Server Alias

For API Gateway to use a proxy server, you must define a proxy server alias. Proxy server alias names must be unique across protocols, that is, you cannot have proxy server aliases of the same name but of different protocols.

➤ To configure proxy server alias

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Outbound proxies**.

This displays a list of available proxy server aliases and the corresponding details.

3. Click **Add proxy** and provide the following information:

Field	Description
Alias	The alias name of the proxy server.
Username	<i>Optional.</i> The username API must use when accessing this proxy server.
Password	<i>Optional.</i> A valid password associated with the username.
Hostname or IP address	The host name or IP address of the proxy server.
Port number	The port on which this proxy server listens for requests.
Protocol	The type of protocol (HTTP, HTTPS) to use for the host/port combination.
Set as default	Indicates whether this proxy server alias should be the default proxy server alias for its protocol type or not. Click Yes or No . Only one default proxy server alias can be set for each protocol type.

4. Click **Add**.

Modifying a Proxy Server Alias

➤ To modify a proxy server alias

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Outbound proxies**.
3. In the Outbound proxy list, select the proxy server alias that you want to edit.
4. Incorporate the required changes.
5. Click **Update**.

Deleting a Proxy Server Alias

➤ To delete a proxy server alias

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Outbound proxies**.
3. In the Outbound proxy list, click  in the action column of the proxy server alias to be deleted.
4. Click **Yes** in the confirmation dialog box.

The proxy server alias is deleted from the list.

Proxy Bypass

If you are using proxy servers for outbound HTTP or HTTPS requests, you can optionally route selected requests directly to their target servers, bypassing the proxy servers listed in the outbound proxies list. For information on outbound proxies, see “[Outbound Proxy](#)” on page 375.

To allow the outbound HTTP and HTTPS requests sent by API Gateway to bypass a particular proxy, you must add the proxy to the **Proxy Bypass** list.

Adding a Proxy Bypass

The HTTP and HTTPS requests sent from API Gateway to remote servers are not directed through the proxies added in the **Proxy Bypass** screen.

➤ To add a proxy bypass

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Proxy Bypass**.
3. In the **Addresses** field, provide the host name or IP of the proxy server that the HTTP and HTTPS requests must bypass.

You can type the fully qualified host and domain name of each server to which you want the API Gateway to issue requests directly. Type the host name or the domain name exactly as they appear in the URLs that the server uses. To enter multiple names, separate each with commas. You can use the asterisk (*) to identify several servers with similar names. The asterisk matches any number of characters. For example, if you want to bypass requests made to localhost, www.yahoo.com, home.microsoft.com, and all hosts whose names begin with NYC, you would type:

```
localhost, www.yahoo.com, home.microsoft.com, NYC*.*
```

4. Click **Add**.

The specified proxy server is displayed in the **Addresses** list.

5. Click **Save**.

The list is saved.

Modifying a Proxy Bypass

➤ To modify a proxy bypass

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Proxy Bypass**.
3. In the **Addresses** list, click  in the action column of the proxy server to be edited.
4. Incorporate the required changes.
5. Click .
6. Click **Save**.

The changes are saved.

Deleting a Proxy Bypass

➤ To delete a proxy bypass

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Proxy Bypass**.
3. In the **Addresses** list, click  in the action column of the proxy server to be deleted.
4. Click **Save**.

The selected proxy server is deleted.

URL Aliases

A URL alias is an alias that you create to replace a portion of the URL to an API on API Gateway. The URL alias typically replaces the path portion of the URL which identifies the name and invocation endpoint of the API. For example, if the URL is `http://test:2225/gateway/RESTCalcService/1.0`, you can create a URL alias, `calc` for an API, then the name and invocation endpoint of the API on API Gateway is replaced with `calc`, for example, `http://test:2225/calc`. If the client sends a request that contains the matching alias, then the corresponding URL path is invoked.

In addition to associating a URL alias with a single resource, you can also map different resources for each port, therefore, based on the incoming port, the corresponding resource path is invoked. This makes it possible to define a single URL alias that resolves to different destinations based on the incoming port of the HTTP request.

URL aliases have several benefits:

- A URL alias may be easier to type than full URL path names.
- A URL alias is more secure than URL path names.

The **URL aliases** page displays a list of available URL aliases with the corresponding details including the default URL path, port it is mapped to, and so on. You must have the Manage aliases functional privilege assigned to manage the alias information.

In the **URL aliases** page, with the **Global gateway endpoint** section, you can also define global gateway endpoint. For more information about global gateway endpoint, see *webMethods API Gateway User's Guide*.

Note:

Any modifications to the URL aliases in Integration Server do not reflect in API Gateway. Hence, Software AG recommends that you do not modify the aliases through Integration Server

Administrator. On migration from 10.0 to 10.1, the existing configuration in 10.0 is migrated to the API Gateway UI.

API Gateway only supports modification of URL aliases for the WmAPIGateway package. To modify URL aliases for any other packages, you must use Integration Server Administrator.

Creating URL Alias

When you create a URL alias, you create an association between an alias and an API on API Gateway.

Keep the following information in mind when creating a URL alias:

- An alias name must be unique across API Gateway.
- You can associate a single URL alias with multiple resources by specifying port mappings. A port mapping correlates the alias with a different URL alias based on the port on which the request was received.
- If you want to use URL alias with a REST resource you must enable partial matching of URL aliases.
- If you enabled or intend to enable partial matching of URL aliases, do not define an alias that begins with another alias.

➤ To create a URL alias

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > URL aliases**.

This displays a list of available URL aliases and the corresponding details.

3. Click **Add URL alias** and provide the following information:

Field	Description
Alias	<p>The alias name of the proxy server.</p> <p>An alias name must be unique across API Gateway.</p> <p>The alias name cannot include a space, nor can it include the following characters: # % ? ' " < \</p> <p>The alias name cannot begin with the string <code>http://</code> or <code>https://</code></p>
Port number	<p>The port number to use when accessing the API.</p> <p>When API Gateway receives a request that contains a URL alias, API Gateway resolves the request destination by first determining if there is</p>

Field	Description
	a URL path associated with the incoming port. If there is no URL path mapped to the port number, then API Gateway uses the default URL path for the alias as the request destination. The port mappings are always evaluated first.
	Because the URL path can be different for each port, using port mappings allows the use of a single URL alias with multiple destinations. Each port can have only one URL path mapping. You can add port mappings to multiple destinations by clicking the +Add button.
URL path	<p>The URL path for the URL alias and for port number mapped for the URL alias.</p> <p>The URL path cannot include a space or the following characters: # % ? ' " < \</p>
Default URL path	<p>The URL path to the API on API Gateway.</p> <p>You must specify the default URL path if you do not define any port mappings for the URL alias. If the URL alias includes port mappings, the Default URL Path field is optional.</p> <p>The URL path cannot include a space or the following characters: # % ? ' " < \</p>

4. Click **Save**.

Using Port Mappings with URL Alias

For a URL alias, you can create one or more port mappings which associates a port with a resource. Port mappings allow the use of a single URL alias with multiple resource paths, because each port can be mapped to a different URL path.

When API Gateway receives a request that contains a URL alias, API Gateway resolves the request destination by first determining if there is a URL path associated with the incoming port. If there is no URL path mapped to the port number, then API Gateway uses the default URL path for the alias as the request destination. The port mappings are always evaluated first.

Adding Port Mapping to URL Alias

You can create multiple port mappings for a URL alias. Incoming requests with the URL alias received on a port that has no URL path mapping, resolves to the path specified in the **Default URL Path** field. If the alias does not specify a **Default URL Path** value, API Gateway uses the alias as the URL path.

➤ To add a port mapping to a URL alias

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > URL aliases**.
3. In the URL alias list, select the URL alias for which you want to define port mappings.
4. In the **Port number** list, select the port for which you want to specify an alternate path.
5. In the **URL path** field, specify the path to API. The URL path cannot include a space or the following characters: # % ? " < \
6. Click **+Add** to add the port mapping to the alias.
7. Click **Save**.

Deleting Port Mapping for URL Alias

You can delete any of the port mappings added to a URL alias.

Note:

If you intend to delete all of the port mappings for a URL alias, make sure the URL alias specifies a **Default URL Path** value.

➤ To delete a port mapping for a URL alias

1. Expand the menu options icon , in the title bar, and select **Administration**.
2. Select **General > URL aliases**.
3. In the URL alias list, select the URL alias for which you want to delete a port mapping.
4. In the **Port number** list, click  in the action column of the port mapping that you want to delete.
5. Repeat the preceding step for each port mapping that you want to delete for an alias.
6. Click **Save**.

Enabling Partial Matching of URL Aliases

In some cases, URL requests include identifiers for a particular API. Because these identifiers vary for each instance of an API, URL requests might not exactly match any of the defined URL aliases for a particular API. To enable you to define URL aliases for such APIs, API Gateway can use

partial matching to process URL requests. A *partial match* occurs when a request URL matches or begins with only part of a URL alias.

When partial matching is enabled and API Gateway receives a request URL, an alias is considered a match if the entire alias matches all or the beginning of the request URL, starting with the first character of the request URL's path.

For example, if URL alias `calc` has a URL path of `gateway/RESTCalcService/1.0` and API Gateway receives the following request URL:

```
https://MyHost:9072/calc/deleteCalc
```

The request URL matches the `calc` alias and resolves to the path:

```
https://MyHost:9072/gateway/RESTCalcService/1.0/deleteCalc.
```

API Gateway retains the trailing characters of the request URL. In this case, API Gateway retains the `/deleteCalc`.

To enable API Gateway to use partial matching of URL requests, you must set the value of the parameter `watt.server.url.alias.partialMatching` to `true` in Integration Server (in the Integration Server Administrator, go to **Settings > Extended** link). The default is `false`. For more information on configuring partial matching of URL aliases using Integration Server, see *webMethods Integration Server Administrator's Guide*.

Important:

If you change the setting of this parameter, you must restart Integration Server for the changes to take effect.

Modifying a URL Alias

➤ To modify a URL alias

1. Expand the menu options icon  in the title bar, and select **Administration**.

2. Select **General > URL aliases**.

3. In the URL Alias list, select the URL alias that you want to edit.

Alternatively, in the URL Alias list, locate the row that contains the alias you want to modify, and click the **Edit** icon.

4. Incorporate the required changes.

5. Click **Save**.

Deleting a URL Alias

➤ To delete a URL alias

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > URL aliases**.
3. In the URL Alias list, locate the row that contains the alias you want to delete, and click .
4. Click **Yes** in the confirmation dialog box.

Example: Usage Scenarios of URL Aliases

This section explains how an API consumer can invoke an API for which a URL alias is created. This example uses the *RESTCalcService* API. Suppose, the *RESTCalcService* API is activated in API Gateway and following are the gateway endpoints for this API:

- <http://test:2225/gateway/RESTCalcService/1.0>
- <http://test:4000/gateway/RESTCalcService/1.0>
- <http://test:4010/gateway/RESTCalcService/1.0>
- <http://test:5555/gateway/RESTCalcService/1.0>

Also, the *RESTCalcService* consists of the *postCalc* resource path that adds two numbers.

If this API is published to the API consumer then the invocation endpoint for the consumer may appear as:

<https://test:5555/gateway/RESTCalcService/1.0/postCalc>

If you do not want to expose the API name and path information or if you want to shorten the invocation endpoint as it is complex, then you can create a custom URL alias. To create a URL alias:

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > URL aliases**.
3. Click **Add URL alias** and provide the following values:

Field	Value
Alias	calc
Default URL path	gateway/RESTCalcService/1.0/postCalc

4. Click **Save**.

Suppose, the URL alias name provided while creating a URL alias is `calc`. You can now expose the `https://test:5555/calc` invocation endpoint to the API consumer instead of `https://test:5555/gateway/RESTCalcService/1.0/postCalc`.

With the default URL path specified in alias configuration, the API consumer can use this URL alias for any ports of gateway endpoint shown in the API details page, for example,

- `http://test:2225/calc`
- `http://test:4000/calc`
- `http://test:4010/calc`
- `http://test:5555/calc`

Additionally, you can use port mapping, if you want to use the same alias for a different resource path. This can be done by providing a different resource path for each port, instead of the default URL path in alias configuration. To use the same alias for a different resource path:

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > URL aliases**.
3. Click **Add URL alias** and provide the following values:

Field	Value
Alias	<code>calc</code>
Port number	<code>5555</code>
URL path	<code>gateway/RESTCalcService/1.0/postCalc</code>

4. Click **+Add** to add port mappings to multiple destinations and provide the following values:

Field	Value
Port number	<code>4000</code>
URL path	<code>gateway/RESTCalcService/1.0/deleteCalc</code>

5. Click **Save**.

Note:

As the **Default URL path** is not provided, the incoming call for ports other than 4000 and 5555 fails. If the **Default URL path** is provided then the port mapping takes the first precedence. If the value of the port does not match, then the **Default URL path** is used.

For the alias `calc`, each port is mapped to a different resource, for the invocation endpoint:

- `https://test03:4000/calc`: RESTCalcService/1.0/deleteCalc resource is invoked.

- <https://test03:4010/calc>: error appears as the default URL is not provided and a path is not configured for the 4010 port.
- <https://test03:5555/calc>: RESTCalcService/1.0/postCalc resource is invoked.

Custom Content-types

An API Provider can configure custom content-types based on how the payloads in the incoming or outgoing requests have to be processed. If the native API consumes some custom content-type, the API Provider can configure a mapping between this custom content-type and a base content-type so that the schema validation, and identification in the policies are performed based on the base type.

For example, a native API consumes application/xyz content-type. The API provider creates an API for this native API and enforces the Validate API Specification policy and the API definition has schema mapping for application/json. When the request reaches API Gateway and as there is no content-type schema mapping for application/xyz, the schema validation is skipped. In such scenarios, if the API provider creates a custom content-type mapping in the API Gateway UI with the content type as application/xyz and base type as JSON, then the payload in the incoming request is validated against the JSON schema.

The following table explains the different identifiers and payload validation criteria that can be used for various content types that you can use to configure your custom content-types.

Content-type	Identifier	Payload validation
application/xml	XPath	XML schema
text/xml		
text/html		
multipart/form-data		
multipart/mixed		
application/json	JSONPath	JSON schema
application/json/badgerfish		
text/plain	Regex	Regex

Note:

The custom content-type feature is not supported for the SOAP to REST transformed APIs.

Configure Custom Content-types

When you configure a custom content-type, you specify what base-type the content type while processing the content.

You must have the API Gateway's manage general administration configurations functional privilege assigned to configure custom content-type.

➤ To configure custom content-type

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Custom Content-types**.
3. Provide the Content-type that has to be configured.
4. Select the **Base type** as one of the following:
 - **JSON**: Specifies that the base-type for the provided Content-type is set to JSON.
 - **XML**: Specifies that the base-type for the provided Content-type is set to XML.
 - **Text**: Specifies that the base-type for the provided Content-type is set to plain text.
5. Click .

You can configure multiple custom content-types by clicking Add.

Cache Configuration

In API Gateway version 10.1 or earlier, all assets are stored in memory and reside in memory indefinitely. Since the cache is loaded fully during startup, it takes a while to load all the elements, such as applications, APIs, policies, and so on in the cache. The Cache configuration functionality in API Gateway enables lazy loading of the assets and ensures that API Gateway startup is quick and is independent of the data size. If the memory size is not enough to contain all the entities API Gateway caches can be configured to load a certain percentage of the entities in memory.

API Gateway internally classifies cache based on the usage and size as follows:

- **size bounded**: In-memory cache that forgets the least recently used (LRU) entry when the maximum number of entries is reached. This can be configured as percentage of the total number of entries in the data store.
- **size unbounded**: When the cache configuration percentage is set to 100% manually or as computed by auto scale option, the cache is set as unbounded.

You can configure individual cache types in this section. You can set additional parameters that control the cache configuration in the **Administration > General > Extended Settings** section. For details see, “[Configuring Extended Settings](#)” on page 324

Note:

The cache configuration is synchronized across different nodes in a cluster.

Autoscaling mode

Depending on the load on the system the autoscaler thread computes the percentage of entities to be kept in memory. If the number of entities is less all the entities are stored in memory. If the number of entities does not fit into the memory then it computes the memory for each of the caches that are set to run in autoscale mode. The caches are reconfigured automatically based on the computed value. This can be controlled by setting the parameters `pg_Cache_autoScalerRunInterval` and `pg_Cache_minCachePercent`.

Percentage mode

Depending on the growth or the current size of the number of entities the cache size is computed and resized periodically. This is controlled by the setting the parameters `pg_Cache_boundedCacheResizeRunInterval` and `pg_Cache_minCacheSize`.

You can also collect the required statistics for the selected cache, which is displayed in the Global dashboard under **Analytics > Cache Statistics**. This statistics collection interval can be configured by setting the parameter `pg_Cache_statisticsProcessorRunInterval`.

Configuring Cache to Improve Performance

You must have the API Gateway's manage general administration configurations functional privilege assigned to configure cache.

➤ To configure cache to improve performance

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Cache configuration**.
3. Select the **Cache name**, from the drop-down list of available cache to be configured.
4. Select a value for the **Percentage of cache in memory**.

For example, you can set the value as 10%, 20%, and so on. If you do not select any value, the value is set to Auto scale by default.

5. Select **Collect statistics** to collect the required statistics for the selected cache.
6. Click .

The configured cache is listed in a table in the cache configuration section. If the configuration is set to Auto scale and Collect statistics is not selected then, by default, the cache is not listed.

You can edit or delete the cache configuration entries by clicking the icons in the action column.

7. Click **Update**.

This saves the cache configuration.

Note:

To enhance the performance of basic authentication scenarios at runtime, ACCESS_PROFILES cache type is always configured to 100% during restart of API Gateway. You cannot reconfigure the ACCESS_PROFILES cache from API Gateway user interface and Software AG recommends users not to modify the value of ACCESS_PROFILES cache type using REST APIs.

Application Log Configurations

API Gateway consists of different components such as API Gateway Server, API Data Store, Dashboard, and Platform (WSStack, OSGI etc). Each of these components generate separate logs and store them in different locations. This feature provides a central place where you can configure the logs across different components.

The **Administration > General > Application logs** tab provides a simplified solution where you can perform the following tasks:

- Set the log levels for different components
- Download logs
- Aggregate the logs from different components

Note:

The log aggregation is done across all the configured cluster nodes. The log level configuration is propagated across all the nodes in a cluster.

- View the logs from different components in a single place and perform data mining

Configuring Log Levels

You can configure the log level settings to specify the level of detail that you want to capture in the logs.

The log level settings that you can specify are described below. Each logging level includes the indicated type of message and all messages from the levels above it (for example, the **Warn** level includes **Fatal**, **Error**, and **Warn** messages).

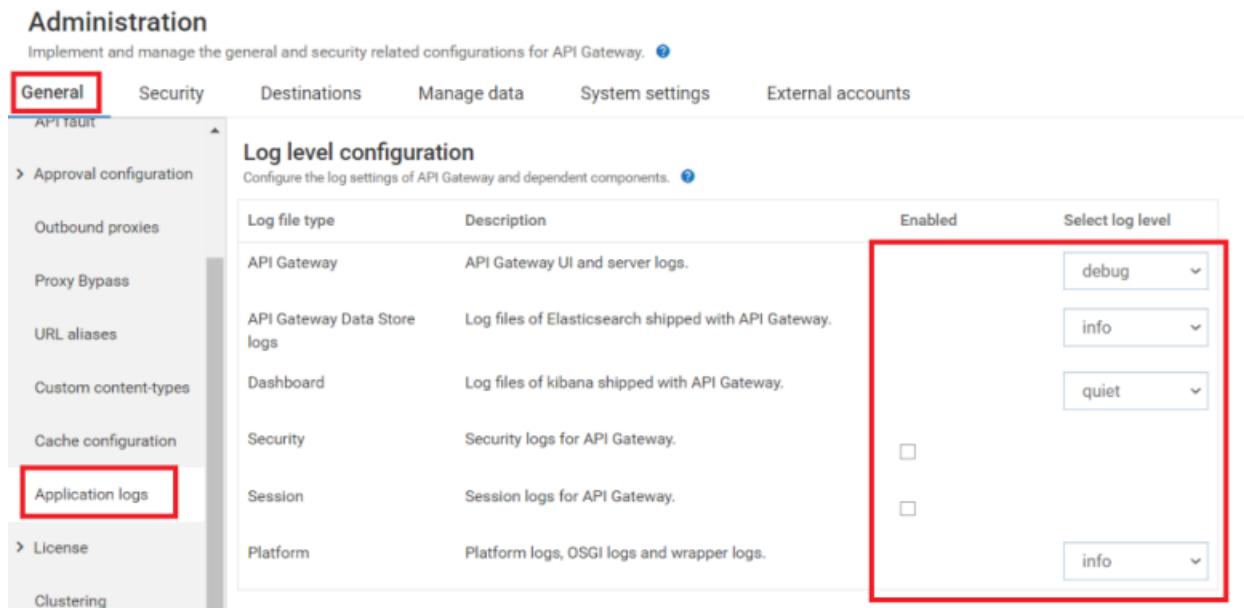
Log Level	API Gateway Server records these entries	Examples
Fatal	Failures that end operations in such a way that the operations cannot successfully continue without user intervention. Failure is very likely to affect other operations or products.	Product cannot read its configuration file and has no default settings.
Error	Same as Fatal, except that existing error handling renders the failure unlikely to affect other operations or products.	Business process step failed due to a service error caused by bad input data.

Log Level API Gateway Server records these entries		Examples
Warn	Problems that do not end operations, or unexpected or unusual conditions that might signal impending failure.	Multiple registered JMX servers were discovered where only one is needed.
Info	Success of an event that you need to know about. Package was loaded, or connection was pool initialized.	
Debug	Code-level statements recording unusual conditions or decisions that might lead to errors.	Expects an object to be initialized but it is not, or hash table is empty.
Trace	Code-level statements recording the state and flow of the program during normal execution.	Entry or exit method, or local and global object state.
Off	No information for the product or facility is written to the server log.	

Prerequisite: You must have the **Manage general administration configurations** functional privilege to configure log levels.

➤ To configure log levels

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Application logs**.



Log file type	Description	Enabled	Select log level
API Gateway	API Gateway UI and server logs.	<input type="checkbox"/>	debug
API Gateway Data Store logs	Log files of Elasticsearch shipped with API Gateway.	<input type="checkbox"/>	info
Dashboard	Log files of kibana shipped with API Gateway.	<input type="checkbox"/>	quiet
Security	Security logs for API Gateway.	<input type="checkbox"/>	
Session	Session logs for API Gateway.	<input type="checkbox"/>	
Platform	Platform logs, OSGI logs and wrapper logs.	<input type="checkbox"/>	info

3. In the Log level configuration section, configure the following parameters:

Log file type	Description
API Gateway	<p>This setting sets the API Gateway server log level. API Gateway server logs contain information on the API Gateway UI related activities. API Gateway server logs also contain information on the start-up activity of API Gateway server and its functioning, its state and activities related to connections to various components.</p> <p>Select a log level that you want to configure to collect required logs. The available log levels are: off, trace, debug, info, warn, error, fatal. By default, the log level for API Gateway server logs are set as info.</p>
<p>Note: Changing the log level of API Gateway server logs sets that value of log level for all the log components of API Gateway. If you want to set a different log level for a particular log components, then log into Integration Server user interface, click Logs > Logging Configuration > Sever Logger > API Gateway, then select the log level that you want to set for the corresponding log component.</p>	<p>Log file source: <i>Install_Dir/IntegrationServer/instances/default/logs/server.log</i></p>
API Gateway Data Store logs	<p>This setting sets the API Data Store log level. This contains log files of API Data Store. All API Gateway assets such as API, application, alias, and so on that are stored in API Data Store are logged based on this setting.</p> <p>Select a log level that you want to configure to collect required logs. The available log levels are: off, debug, info, warn, error, fatal. By default, the log level for API Data Store logs are set as info.</p>
<p>Log file source: <i>Install_Dir/InternalDataStore/logs/SAG_EventDataStore.log</i></p>	<p>This setting sets the log level for dashboard logs. These logs contain information regarding dashboard-related activities and the Kibana log files that are included in API Gateway.</p> <p>Select a log level that you want to configure to collect required logs. The available log levels are: silent, quiet, verbose.</p> <ul style="list-style-type: none"> ■ Select silent to suppress all logging output. ■ Select quite to suppress all logging output except for error messages.

Log file type	Description
	<ul style="list-style-type: none"> ■ Select verbose to log all events, including system usage information and all requests. <p>By default, the log level for dashboard logs are set as quiet.</p> <p>Log file source: <i>Install_Dir/profiles/IS_instance_name/apigateway/dashboard/startup.log</i></p>
Security	<p>API Gateway generates security logs when you enable this. These logs contain information on security-related administrative and operational events. For example, changes to authorization, authentication, port settings, password restrictions, attempts to log on to API Gateway and to access API Gateway services, and so on.</p> <p>Select the checkbox to enable the collection of security logs.</p> <p>Log file source: <i>Install_Dir/IntegrationServer/instances/default/logs/WMSECURITY*.log</i></p>
Session	<p>API Gateway generates session logs when you enable this. These logs contain information on sessions activated on API Gateway and provide data on when sessions start, their current status, their duration, and so on.</p> <p>Select the checkbox to enable the collection of session logs.</p> <p>Log file source: <i>Install_Dir/IntegrationServer/instances/default/logs/WMSESSION*.log</i></p>
Platform	<p>This setting sets the log level for API Gateway platform logs.</p> <p>Select a log level that you want to configure to collect required logs.</p> <p>The available log levels are: off, trace, debug, info, warn, error, fatal.</p> <p>By default, the log level for platform logs are set as info.</p> <p>Note: Whenever you change the log level settings for the API Gateway platform logs, make sure you restart API Gateway for the changes to take effect.</p> <p>Log file source: <i>Install_Dir/ profiles/IS_instance_name/logs/sag osgi.log</i></p>

4. Click **Save**.

This configures the log levels for the various log file types that can be downloaded as an archive.

Downloading the Log Files

You can download the logs as archived file. This contains all the logs collected from various API Gateway components, the thread dumps of all the currently active threads, and the server configurations, the details of which correspond to the log level configured. You can browse through the downloaded data to troubleshoot any issues, such as error or performance problems.

Prerequisite: You must have the **Manage general administration configurations** functional privilege to download log files.

➤ To download the log files

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Application logs**.
3. In the **Download diagnostics** section, click **Download diagnostics**.

The file is downloaded to your local machine at the predefined location in the zip format.

Configuring Log Aggregation

You can configure log aggregation such that the logs from different components can be stored at a single location in a common format. You can select any of the following destinations to store the aggregated data:

- API Data Store
- Elasticsearch, in case you have an external Elasticsearch configured with API Gateway

Log aggregation collects log from the different components and stores them either in API Data Store or external Elasticsearch based on the log aggregation configuration settings. If the API Data Store is configured as the destination the aggregated logs are displayed in the API Gateway dashboard under the **Analytics > Application logs** tab. You can filter the logs as required and search for a particular log in the dashboard.

Note:

If you configure to send the logs to the same API data store, which stores the API Gateway's core data (APIs & configurations), then it is very important to setup the data housekeeping in place since the log data grows over time in volume and will start to impact API Gateway's performance. For more information on log housekeeping practices, see "["Logs Housekeeping" on page 398](#)". Alternatively (also a better approach for production environments), logs can be sent to an External Elasticsearch.

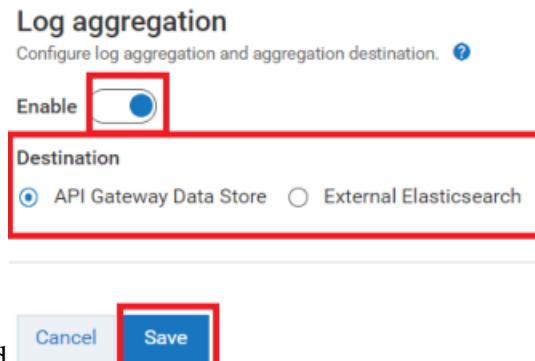
Prerequisite: You must have the **Manage general administration configurations** functional privilege to:

- Enable the log aggregation and

- Configure the destination for log aggregation

> To configure log aggregation

- Expand the menu options icon  in the title bar, and select **Administration**.
- Select **General > Application logs**.
- In the **Log aggregation** section, click the toggle button to change the status to  and enable the log aggregation.
- Select one of the following destination:
 - API Gateway Data Store:** Select this to set API Data Store as destination to store the aggregated logs and you can view them in the API Gateway



Log aggregation
Configure log aggregation and aggregation destination. 

Enable 

Destination

API Gateway Data Store External Elasticsearch

Save

dashboard

- External Elasticsearch.** Select this to set the external Elasticsearch as destination to store the aggregated log. This approach helps to separate the Elasticsearch that stores the API Gateway core data from the aggregated log . But, if you use the external Elasticsearch, you must create your own dashboard to view the logs.

Note:

Make sure you have configured an external Elasticsearch with API Gateway.



Log aggregation
Configure log aggregation and aggregation destination. 

Enable 

Destination

API Gateway Data Store External Elasticsearch

Protocol	Hostname*	Port*
HTTPS	ext_elasticsearch	9999
Indexname	Username	Password
app_log	ext_es_abc	*****

Save

Provide the following information:

Field Name	Description
Protocol	The type of protocol (HTTP, HTTPS) to use for the host and port combination.
Hostname	Specifies the host name or IP address of the machine on which Elasticsearch resides.
Port	Specifies the port where Elasticsearch server runs.
Indexname	Specifies the index of the collected logs.
Username	Specifies the Elasticsearch user ID for authenticating Elasticsearch when API Gateway communicates with it.
Password	Specifies the password of the Elasticsearch instance to be used for establishing communication between API Gateway and Elasticsearch.

5. Click **Save**.

The selected destination is now configured for log aggregation.

Filebeat

API Gateway uses Filebeat to monitor the log files and forwards them to either API Data Store or external ElasticSearch. Filebeat is shipped along with API Gateway and its configuration can be found in the following location:

`SAG_Install_Dir/profiles/IS_Instance_Name/apigateway/filebeat/filebeat_apigateway.yml`

When log aggregation is enabled, API Gateway initiates the Filebeat process in the background to monitor the log files and forwards the log files to either API Data Store or external Elasticsearch.

API Gateway + API Data Store + Terracotta Server

In case, if you also want to aggregate Terracotta server logs that are stored in different location, perform the following steps:

If Terracotta Server and API Gateway is installed in the same machine, perform the following steps:

1. In the **Administration > General > Application logs > Log aggregation** section, disable the log aggregation.

Note:

By default, the Terracotta logs are available in the `(user.home)/terracotta/server-logs` location. You can change this folder by specifying the custom log data location in the `tc-config.xml`.

2. Update the **filebeat_template.yml** file in the `SAG_Install_Dir\profiles\IS_Instance_Name\apigateway\filebeat` location with the following content. Make sure you place the following content below the **DashboardLogs** configuration:

```
- type: log
  # Change to true to enable this prospector configuration.
  enabled: true
  # Paths that should be crawled and fetched. Glob based paths.
  paths:
    - C:/Users/alice/terracotta/server-logs/terracotta-server.log
  fields:
    node: ${NODE}
    fileType: TerracottaServerLogs
    timezone: ${TIMEZONE}
  fields_under_root: true
  multiline.pattern: (([\s]+)20[0-9]{2}-)|20[0-9]{2}-
  multiline.negate: true
  multiline.match: after
```

3. In the **Administration > General > Application logs > Log aggregation** section, enable log aggregation.
4. In the **Analytics > Application Logs** section, check whether the Terracotta Server logs are aggregated.

If Terracotta Server and API Gateway are installed in different machine, perform the following steps:

1. Edit the **tc-config.xml** in Terracotta Server with the following content to store the logs in a network location and start the Terracotta Server:

```
<?xml version="1.0" encoding="UTF-8"?>
<tc:tc-config xmlns:tc="http://www.terracotta.org/config">

  <servers>
    <server host="localhost" name="%h">
      <data>C:/install/TerraCottaServer/Terracotta/server/bin/data</data>
      <logs>//worstation01/share/tcserverlog</logs>
      <offheap>
        <enabled>true</enabled>
        <maxDataSize>512m</maxDataSize>
      </offheap>
    </server>
    <restartable enabled="true"/>
  </servers>
</tc:tc-config>
```

2. In the **Administration > General > Application logs > Log aggregation** section, disable log aggregation.
3. Update the **filebeat_template.yml** file in the `SAG_Install_Dir\profiles\IS_Instance_Name\apigateway\filebeat` location with the following content. Make sure you place the following content below the **DashboardLogs** configuration:

```
- type: log
  # Change to true to enable this prospector configuration.
  enabled: true
```

```

# Paths that should be crawled and fetched. Glob based paths.
paths:
- //mckmut02/share/tc-server-logs/terracotta-server.log
fields:
  node: ${NODE}
  fileType: TerracottaServerLogs
  timezone: ${TIMEZONE}
  fields_under_root: true
multiline.pattern: (([\s]+)20[0-9]{2}-)|20[0-9]{2}-
multiline.negate: true
multiline.match: after

```

4. In the **Administration > General > Application logs > Log aggregation** section, enable the log aggregation.
5. In the **Analytics > Application Logs** section, check whether the Terracotta Server logs are aggregated.

Configuring Log for Elasticsearch Client in API Gateway

API Gateway uses Elasticsearch REST client to connect to API Data Store or External Elasticsearch. By Default, the logs that are created by these REST clients are ignored to avoid over logging. To troubleshoot or diagnose elasticsearch calls from API Gateway, you can manually configure the log configuration for Elasticsearch REST client using the following steps:

1. Open the **log4j2.properties** located at *SAG_Install_Dir\profiles\IS_Instance_Name\configuration\logging*.
2. Add the following configuration at the end of the file:

```

logger.10.name=org.elasticsearch.client
logger.10.additivity=false
logger.10.level=info
logger.10.appenderef.lar.ref=ESRestClient

logger.11.name=org.apache.http
logger.11.additivity=false
logger.11.level=info
logger.11.appenderef.lar.ref=ESRestClient

logger.12.name=org.apache.http.wire
logger.12.additivity=false
logger.12.level=info
logger.12.appenderef.lar.ref=ESRestClient

logger.13.name=org.apache.http.impl.conn
logger.13.additivity=false
logger.13.level=info
logger.13.appenderef.lar.ref=ESRestClient

appenderef.esrestclient.name=ESRestClient
appenderef.esrestclient.type=RollingFile
appenderef.esrestclient.fileName=<INSTALL_LOCATION>/IntegrationServer/instances/default/logs/ESRestClient.log
appenderef.esrestclient.filePattern=<INSTALL_LOCATION>/IntegrationServer/instances/default/logsESRestClient.log.%i
appenderef.esrestclient.layout.type=PatternLayout
appenderef.esrestclient.layout.pattern=%d [%t] %-5p %c %x - %m%n
appenderef.esrestclient.policies.type=Policies

```

```
appender.esrestclient.policies.size.type=SizeBasedTriggeringPolicy
appender.esrestclient.policies.size.size=10MB
appender.esrestclient.strategy.type=DefaultRolloverStrategy
appender.esrestclient.strategy.max=10
#This is a custom Platform provided filter which matches all log messages that
contain OSGi data in MDC (bundle.id, component.name, etc.)
appender.esrestclient.filter.osgi.type=LogServiceFilter
appender.esrestclient.filter.osgi.onMatch=DENY
appender.esrestclient.filter.osgi.onMismatch=NEUTRAL
```

3. Restart API Gateway.

You can see the ESRestClient.log at the `SAG_Install_Dir\IntegrationServer\instances\instance_name\logs` location. You can see the request sent to particular elasticsearch node and what is the response status code in the ESRestClient.log

Logs Housekeeping

The Log housekeeping is essential to free the space of old and unused log files. This can be achieved in the following ways:

- **Log rotation settings.** Setting log rotation for the individual log files that are collected from different components of API Gateway. For more information about log file rotation setting for each of the log files, see *webMethods API Gateway Administration*.
- **Archive and Purge.** You can archive and purge the aggregated logs that are stored in API Data Store.

For more information about how to archive and purge the application logs, see *webMethods API Gateway Administration*.

License Configuration

When you purchase webMethods API Gateway, your organization is granted a license to use it with certain features and functionality. The license expires after a time period specified by your purchase agreement.

Before you install API Gateway, you are provided with a license key file that you place in the file system of the machine on which API Gateway will run. This file contains the license key, which is a special code associated with your license. When you install API Gateway, the setup program asks you to provide the name and location of this file. The setup program then copies this file to the `SAGInstalldirectory\instances\instance_name\config` directory with the name `licenseKey.xml`. If this file is inadvertently deleted, API Gateway login fails.

Viewing Licensing Information

You must have the API Gateway's manage general administration configurations functional privilege assigned to configure licenses and view license information.

➤ To view license information

1. Expand the menu options icon  in the title bar, and select **Administration**.

2. Select **General > License > Details**.

This displays the current license information and an option to configure license

3. In the License information section you can view a list of features, with a check mark next to each feature you are licensed to use.
4. To view details of the licensing information, click **Details**.

API Gateway displays the following licensing information:

- **Sales information.** Displays the license details related to Sales such as Serial number of the license, the License key, the customer name and ID, number of contracts and details and so on.
- **Product information.** Displays the product details that comes with the current license such as license expiration date, operating system on which the product runs, product details such as code, name, version, and ID, the renewal date, and so on.
- **Integration Server.** Displays the Integration server details such as product code, version, concurrent sessions that can run, clustering and publishing capability, and so on.
- **License information.** Displays the license details such as Price unit, and quantity, installation type, license type, license version, and so on.
- **API Gateway.** Displays the features of API Gateway that are licensed.
- **Terracotta.** Displays the details of the License file, and expiration date of the license.

Configuring Licenses

You must have the API Gateway's manage general administration configurations functional privilege assigned to configure licenses and view license information.

➤ To configure the license

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > License > Configuration**.

This displays the current license information and an option to configure license

3. In the Configure license section, provide the following information:

- **API Gateway license file.** Provide the pathname for the API Gateway license file.

For example:

C:\Installations\IntegrationServer\instances\default\config\licenseKey.xml

- **Terracotta license file.** Provide the pathname for the Terracotta license key file if you have a Terracotta installed for clustering. An API Gateway requires Terracotta to be installed for a clustered API Gateway environment. The Terracotta license file contains the license information for all of your Terracotta components. You add this file to API Gateway by placing it into the SAGInstalldirectory\common\conf directory of the machine on which API Gateway runs. If the license file is located in a different directory than the specified directory for licenses you have to specify the pathname where the Terracotta license key file is located.

4. Click **Save**.

The license key file is saved in the location specified.

Configuring API Callback Processor Settings

API Gateway provides asynchronous form of API support for REST APIs with its capability of defining the callback component with the supported method parameters while creating a REST API.

You must have the *Manage general administration configurations* functional privileges to configure callback processor settings.

Configure the API callback processor setting **All API callback requests** so that API Gateway accepts all the requests from the client that contain the callback request URL and wraps the requests with its own URL before routing them to the native API. This lets API Gateway track the requests that the client sends to the native API and the callback messages that are sent by the native API to the client. In addition, you can use the settings **Allow HTTPS access only** and **Process only allowed IPs requests** to avoid any external threats in case an unauthorized user tries to access the protected resource. You can configure API Gateway to enforce any of the response processing policies that suits your needs on the immediate responses as well as the callback requests being sent from the native API to the client.

The callback requests-related event types can be distinguished by a new field with the value set to true and displayed in the dashboard in the transaction event type. For a normal request this field is set as false. The following are the field names that are displayed for various configured destinations:

- For API Gateway destination the field name is `callbackRequest`, which is set to true.
- For Elasticsearch destination the field name is `isCallbackRequest`, which is set to true.
- For all other destinations, API Portal, Audit Log, CentraSite, Email, JDBC, and Local log, the field name is `isCallbackRequest`, which is wrapped under the `customFields` column.

➤ **To configure API callback processor settings**

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Callback processor**.
3. Select **Process all API callback requests**.

This enables API Gateway to accept all the API callback requests coming from the client and wraps these requests with its own URL before it routes these requests to the native API. This option is selected by default.

When this setting is disabled, the request from the client reaches the native API, as is, without the API Gateway wrapping it with its own URL. So, when the native API sends out the callback request to the client it directly reaches the client and API Gateway is unable to track such events.

4. Select **Allow HTTPS access only**.

This allows API Gateway to receive only HTTPS callback requests from the native API and processes the requests before routing them to the client. If a HTTP callback request comes in, API Gateway sends out an Access denied message to the client. This option is selected by default.

If this option is not selected then API Gateway accepts the HTTP callback requests and processes the requests before routing them to the client.

5. Select **Process only allowed IPs requests**. This allows API Gateway to receive the callback requests only from the IP addresses specified in the Trusted IP addresses list.

API Gateway allows callback requests only from the allowed IPs configured in Trusted IP address list. You can configure your native APIs machine IPs or the native API outbound proxy server IPs here, so API Gateway allows a request coming from the native API and would then be routed to the client.

If there are no trusted IPs configured and this option is selected, then API Gateway does not allow any requests.

6. Type the IP address in the **Trusted IP address** and **Add**. You can add multiple IP addresses.

API Gateway allows only requests coming from these IP addresses when the option **Process only allowed IPs requests** is selected.

7. Click **Save**.

Messaging

API Gateway Messaging is an umbrella term that encompasses the exchanging of messages across multiple platforms in asynchronous style.

To configure API Gateway for JMS messaging, you need to:

- Create one or more JNDI provider aliases to specify where API Gateway can look up administered objects 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.

Creating a JNDI Provider Alias

Each JMS provider can store JMS administered objects in a standardized namespace called the Java Naming and Directory Interface (JNDI). JNDI is a Java API that provides naming and directory functionality to Java applications.

As the JMS client, API Gateway uses a JNDI provider alias to encapsulate the information needed to look up an administered object. When you create a JMS connection alias, you can specify the JNDI provider alias that API Gateway should use to look up administered objects (that is, Connection Factories and Destinations).

➤ To create a JNDI provider alias

1. Expand the menu options icon  in the title bar, and select **Administration**.

2. Select **General > Messaging**.

API Gateway displays a list of all the currently defined JNDI provider alias definitions.

3. Click **Add JNDI provider alias**.

4. Provide the following information for the JNDI provider alias

Field	Description
JNDI alias name	The alias name that you want to assign to the JNDI provider.
Description	A description for the JNDI alias.
Predefined JNDI Templates	Select the JNDI template that you want to use. The JNDI templates provide information that you can use to complete alias configuration for a specific provider. The available templates are: Broker, UM, filesystem, LDAP, JBoss, WebLogic, Qpid-AMQP (0-x)
Initial context factory	The class name of the JNDI provider. The JNDI provider uses the initial context as the starting point for resolving names for naming and directory operations. API Gateway displays the initial context

Field	Description
Provider URL	<p>factory for the provider depending on the predefined JNDI template selected.</p>
	<p>The primary URL of the initial context for sessions with the JNDI provider.</p> <p>The URL specifies the JNDI directory in which the JNDI provider stores JMS administered objects.</p> <p>If you are using Software AG Universal Messaging, this is the Universal Messaging realm server in the format nsp:// UM_host : UM_port (for example, nsp://127.0.0.1:9000).</p>
Provider URL failover list	<p>If you are using a cluster of Universal Messaging realm servers, supply a list of the URLs to each realm server in the cluster. Use a colon or semi-colon to separate each URL:</p> <ul style="list-style-type: none"> ■ Separate the URLs using a comma if API Gateway always attempts to connect to the first Universal Messaging server in the list, only trying the second Universal Messaging server in the list if the first server becomes unavailable. ■ Separate the URLs using a semicolon if API Gateway connects to a randomly chosen URL from the list. This may result in better distribution of clients across the servers in the cluster. <p>If you are using the filesystem you have to provide the filepath of the location of the file to be used.</p> <p>If you are using Qpid-AMQP option you have to provide the file path location where the amqp.properties file is saved.</p>
	<p>A list of the failover URLs of the initial context for sessions with the JNDI provider. If the connection to the primary JNDI provider becomes unavailable, API Gateway automatically attempts a connection to a JNDI provider specified in this list.</p> <p>Specify one URL per line.</p> <p>Keep the following points in mind when adding JNDI provider URLs to the failover list:</p> <ul style="list-style-type: none"> ■ The JNDI providers must be the same type as the primary provider. For example, if the primary

Field	Description
	<p>provider is a webMethods Broker, then the JNDI providers in the failover list must also be webMethods Brokers.</p>
	<ul style="list-style-type: none"> ■ The administered objects on the providers must be identical to each other. ■ Once Integration Server connects to a JNDI provider, it continues to use that JNDI provider until the connection is lost or interrupted. ■ When you start or restart a connection alias, Integration Server attempts to connect to the primary JNDI provider. If the connection fails, Integration Server immediately attempts to connect to the first JNDI provider in the failover list. If the connection fails, Integration Server attempts to connect to the next JNDI provider in the list. ■ When using webMethods Broker as a JNDI provider, you can keep objects in sync between webMethods Brokers using a webMethods Broker territory. That way objects can automatically forward from one webMethods Broker to another within the territory. ■ When using a cluster of Universal Messaging realm servers as the JNDI provider, Software AG recommends that you do not specify a Provider URL Failover List value. The realm URLs specified in Provider URL function as the failover list.
Security principal	<p>The principal name, or user name supplied by API Gateway to the JNDI provider, if the provider requires one for accessing the JNDI directory.</p>
	<p>For information about whether or not the JNDI provider requires security principal information, consult the product documentation for the JNDI provider.</p>
Security credentials	<p>The credentials, or password, that API Gateway provides to the JNDI provider, if the provider requires security credentials to access the JNDI directory.</p>
	<p>For information about whether or not the JNDI provider requires security credentials, consult the product documentation for the JNDI provider.</p>
Other properties	<p>Any additional properties the JNDI provider requires for configuration. For example, you might need to</p>

Field	Description
Additional Classpath	specify the classpath for any additional .jar or class files that the JNDI provider needs to connect to the JNDI.
Template	When you select a predefined JNDI template, API Gateway populates this field with any additional properties and placeholder information required by the JNDI provider.
Properties	For more information about additional properties or classes required by a JNDI provider and the location of those files, see the product documentation for the JNDI provider.

5. Click **Add**.

The JNDI provider alias created is listed in a table under JNDI provider alias definitions in the Messaging page.

You can edit, export the alias definition or delete the JNDI provider alias as required. You can also perform a test lookup for a JNDI provider to ascertain it is working as expected.

Considerations while using Software AG Universal Messaging as a JMS provider

Keep the following points in mind when using Universal Messaging as the JMS provider:

- When using Universal Messaging, if the version of Universal Messaging is equivalent to or higher than the version of API Gateway, you do not need to add any client libraries to the Integration Server classpath.
- When using Universal Messaging, if the version of Universal Messaging is lower than the version of API Gateway, you have to add the JMS provider's client libraries to the Integration Server classpath. You can do it in one of the following ways:
 - Place the libraries in the server's classpath by placing them in the *Install directory\instances\instance_name\lib\jars\custom* directory. For details on the procedure, see *webMethods Integration Server Administrator's Guide*.
 - Make the libraries available to all server instances by placing them in the *Install directory\IntegrationServer\instances\lib\jars* directory.
 - Isolate the jars within a package classloader by placing them in the following directory: *packageName\code\jars*. If you place the files in the package classloader, make sure to set the Class Loader property when configuring a JMS connection alias to this JMS provider.

Note:

If you have configured and are using Software AG Universal Messaging, on migration from an earlier version of API Gateway to API Gateway 10.5, for example, if you are migrating from API Gateway 10.3 to 10.5, then the JNDI provider alias created in the 10.3 version is found to point to the 10.3 Universal Messaging endpoint. You can resolve this by either copying the 10.3 Universal Messaging client jars to 10.5 Integration Server's classpath or by

manually changing the **Provider URL** of the **JNDI Provider Alias** in 10.5 Integration Server to point to the 10.5 Universal Messaging endpoint.

- Keep the Universal Messaging client libraries up to date. If you install a Universal Messaging fix that updates the client libraries, make sure to copy the updated Universal Messaging client library files to the Software AG_directory /common/lib directory used by API Gateway.

For details on other supported JMS providers, see *webMethods Integration Server Administrator's Guide*.

Creating a JMS Connection Alias

A JMS connection alias specifies the information that API Gateway needs to establish an active connection between API Gateway and the JMS/AMQP provider. API Gateway uses a JMS connection alias to send messages to and receive messages from the JMS/AMQP provider. When you create a JMS connection alias, keep the following points in mind:

- You can use JNDI to retrieve administered objects (Connection Factories and Destinations) and then use the Connection Factory to create a connection. If you intend to use a JNDI provider, you need to configure one or more JNDI provider aliases before creating a JMS connection alias
- Each JMS connection alias has an associated transaction type. Within API Gateway, certain functionality must be completed within a non-transacted session. For example, to use API Gateway to send or receive large message streams, you must specify a JMS connection alias whose transaction type is set to NO_TRANSACTION.

➤ To create a JMS connection alias

- Expand the menu options icon  in the title bar, and select **Administration**.
- Select **General > Messaging**.

API Gateway displays a list of all the currently defined JNDI provider alias definitions, JMS connection alias definitions, and individual SOAP JMS trigger controls.

- Click **Add JMS connection alias** under JMS connection alias definitions section.
- In the General Settings section, provide the following information:

Field	Description
Connection alias name	Name of the connection alias. This name should be unique as each connection alias represents a connection factory to a specific JMS provider.
Description	A description of the JMS connection alias.

Field	Description
Transaction type	<p>Specifies whether the sessions that use this JMS connection alias are transacted.</p> <p>Available values are:</p> <ul style="list-style-type: none"> ■ NO_TRANSACTION. Indicates that sessions that use this JMS connection alias are not transacted. ■ LOCAL_TRANSACTION. Indicates that sessions that use this JMS connection alias are part of a local transaction. ■ XA_TRANSACTION. Indicates that sessions that use this JMS connection alias are part of an XA transaction.
Connection client ID	<p>The JMS client identifier associated with the connections established by this JMS connection alias.</p>
Username	<p><i>Optional.</i> User name needed to acquire a connection from the connection factory.</p> <p>For more information about whether or not the JMS provider requires a user name and password to obtain a connection, refer to the product documentation for the JMS provider.</p>
Password	<p><i>Optional.</i> Password needed to acquire a connection from the connection factory.</p> <p>For more information about whether or not the JMS provider requires a user name and password to obtain a connection, refer to the product documentation for the JMS provider.</p>

5. In the Connection protocol settings section, provide the following information:

Field	Description
JNDI provider alias name	<p>The alias to the JNDI provider that you want this JMS connection alias to use to look up administered objects.</p>
Connection factory lookup name	<p>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.</p> <p>If the JMS connection alias connects to Universal Messaging, specify the Universal Messaging connection</p>

Field	Description
	<p>factory that you created when you set up your environment.</p> <p>If you are using SonicMQ as the JMS provider, specify the lookup name that refers to the serializable Java object file containing the SonicMQ object definitions. Include the .sjo extension as part of the lookup name.</p>
Create administered objects on demand (Universal Messaging)	<p>Specifies whether API Gateway creates administered objects on the JNDI provider if the object is not found when API Gateway looks up the object.</p> <p>Select the check box if you want API Gateway to create a destination or connection factory when an JNDI lookup for the object fails.</p>
Enable follow the master (Universal Messaging)	<p>Specifies whether connections created from this JMS connection alias always connect to the master realm server in the Universal Messaging cluster. This setting affects producer and consumer connections created using this JMS connection alias.</p> <p>You can do one of the following:</p> <ul style="list-style-type: none"> ■ Select the Enable follow the master check box to indicate that API Gateway connects to the master realm server in the Universal Messaging cluster when API Gateway uses this JMS connection alias to send or receive messages. ■ Clear the Enable follow the master check box to disable the follow the master behaviour for this JMS connection alias when API Gateway uses this JMS connection alias to send or receive messages.
	<p>Note: Enable follow the master option is available to JMS connection aliases that use Universal Messaging as the JMS provider.</p>
Monitor webMethods connection factory	<p>Specifies how API Gateway monitors the connection factory object for changes, if at all. This only applies if a JMS connection alias connects to the webMethods Broker using a webMethods Connection Factory object in a JNDI server.</p> <p>Select one of the following available options:</p>

Field	Description
	<ul style="list-style-type: none"> ■ No. Indicates that API Gateway does not monitor the connection factory. This is the default option. ■ Poll for changes (specify interval). Monitors the connection factory by polling the changes at an interval that you specify. ■ Poll for changes (interval defined by webMethods Connection Factory). Monitors the connection factory at an interval determined by the refresh interval specified for the webMethods Connection Factory object. ■ Register change listener. Monitors the connection factory by registering an event listener.

6. In the Advanced settings section, provide the following information:

Field	Description
Class loader	<p>The name of the class loader that you want to use with this JMS connection alias. API Gateway uses the specified class loader when performing certain activities with the JMS connection alias (send a message, receive a message, create a connection, create a destination, and so on.)</p> <p>By default, API Gateway uses the server class loader. However, you can specify the class loader for a package instead. This can help prevent conflicts between the jars required for different JMS providers.</p>
Maximum CSQ size (messages)	<p>The maximum number of messages that can exist in the client side queue for this JMS connection alias. API Gateway writes messages to the client side queue if the JMS provider is not available when messages are sent. Each JMS connection alias has its own client side queue.</p> <p>Specify -1 if you want the client side queue to be able to contain an unlimited number of messages. That is, specify -1 if you do not want to set a maximum limit.</p> <p>If you specify 0, API Gateway does not write messages to the client side queue for this JMS connection alias.</p>
Drain CSQ in order	Specifies whether API Gateway drains the client side queue by sending the messages to the JMS provider in the same order in which API Gateway placed the messages in the client side queue.

Field	Description
	<p>Select the check box if you want API Gateway to send messages from the client side queue in the same order in which API Gateway originally placed the messages in the client side queue.</p>
	<p>When the Drain CSQ in order check box is selected, after the connection to the JMS provider is re-established, API Gateway continues to write new messages to the client side queue until the client side queue is completely drained. If the Drain CSQ in order check box is not selected, after the connection to the JMS provider is re-established, API Gateway sends new messages directly to the JMS provider while it drains the client side queue.</p>
	<p>Note: You can also specify the number of messages API Gateway retrieves from the client side queue for delivery to the JMS provider at one time. By default, API Gateway sends 25 messages at a time.</p>
Create temporary queue	<p>Specifies whether API Gateway creates a temporary queue on the JMS provider to handle request-reply operations that do not specify a replyTo destination.</p> <p>Select the check box if you want API Gateway to create a temporary queue. Clear the check box if you do not want API Gateway to create a temporary queue.</p> <p>You must select the Create temporary queue check box if you want to select the Enable request-reply listener for temporary queue check box.</p>
Enable request-reply listener for temporary queue	<p>Specifies whether API Gateway creates a single dedicated MessageConsumer for receiving synchronous replies delivered to the temporary queue for this JMS connection alias.</p> <p>When this check box is selected, API Gateway creates a dedicated consumer for receiving replies to requests published using this JMS connection alias.</p> <p>When this check box is cleared, API Gateway creates a new JMS MessageConsumer for each reply message.</p>
Enable destination management with designer (Broker and Universal Messaging)	<p>Specifies whether users can use Designer to create, list, and modify destinations on the webMethods Broker or when working with JMS triggers.</p>

Field	Description
	<p>Select the check box if you want Designer users to be able to create, list, and modify destinations using a JMS trigger that uses this JMS connection alias.</p> <p>Note: Software AG recommends that you prevent Designer users from managing destinations in a production environment.</p>
Create new connection per trigger	<p>Specifies whether API Gateway creates a separate connection to the JMS provider for each JMS trigger.</p> <p>Select the check box if you want API Gateway to create a separate connection for each JMS trigger that uses this JMS connection alias.</p> <p>If you want a concurrent JMS trigger that uses this JMS connection alias to use multiple connections to the JMS provider, you must configure the alias to create a separate connection per trigger.</p>

7. In the Producer caching section, provide the following information to configure pools for caching of JMS Session and MessageProducer objects:

Field	Description
Caching Mode	<p>Specifies whether to enable caching of JMS Session and MessageProducer objects for this connection alias.</p> <p>Select one of the following:</p> <ul style="list-style-type: none"> ■ DISABLED. Indicates that API Gateway does not cache JMS Session or MessageProducer objects. ■ ENABLED PER DESTINATION. Enable caching of JMS Session and MessageProducer objects. <p>For a non-transacted JMS connection alias, API Gateway caches a Session object and a MessageProducer object. For a transacted JMS connection alias, API Gateway caches a Session object.</p>
Minimum pool size (unspecified)	<p>The minimum number of entries in the default session pool.</p>
	<p>The default is 1.</p>
Maximum pool size (unspecified)	<p>The maximum number of entries in the default session pool.</p>

Field	Description
	The default is 30.
Minimum poolsize per destination	The minimum number of entries in each destination-specific pool.
Maximum poolsize per destination	The maximum number of entries in each destination-specific pool. A value of 0 (or blank) indicates that API Gateway does not create separate pools for any of the destinations associated with the JMS connection alias.
Destination lookup names (semicolon delimited)	A semicolon delimited list of the lookup names for the destinations for which you want API Gateway to create separate pools. Note: This field appears only when the JMS connection alias specifies JNDI Lookup for creating the connection to the JMS provider.
Idle timeout (ms)	The number of milliseconds before API Gateway removes an inactive pool entry. The timeout value applies to the default session pool and the destination-specific pools. A value of 0 indicates that pool entries never expire. A value of -1 indicates that API Gateway uses the system default as defined by the <code>watt.server.jms.producer.pooledSession.timeout</code> parameter. The default value is 60000 milliseconds.

8. In the Producer retry section, provide the following information to configure automatic retry of pub.jms:send services that use this JMS connection alias to send a message to the JMS provider:

Field	Description
Maximum Retry attempts	The maximum number of times that API Gateway automatically retries a pub.jms:send service that fails because of a transient error. The default value is 0. A value of 0 indicates that automatic retry is disabled for this JMS connection alias.
Retry interval (ms)	The number of milliseconds that API Gateway waits between retry attempts.

Field	Description
	The default is 1000 milliseconds (1 second).

Note:

If the JMS connection alias is transacted or uses a connection factory to which the multi-send guaranteed policy is applied, API Gateway ignores the producer retry values.

- In the Enhanced logging section, provide the following information to configure additional logging for the sending or receiving of JMS messages that use this messaging connection alias:

Field	Description
Logging type	<p>Specifies where API Gateway writes log messages when enhanced logging is enabled for the message producers or consumers that use this JMS connection alias to send or receive messages.</p> <p>You can select one of the following:</p> <ul style="list-style-type: none"> ■ SERVER LOG. Write enhanced logging messages to the server log. If you specify the server log as the destination, make sure to increase the logging level for the 0168 Messaging (Enhanced Logging) server log facility to at least Info. ■ MESSAGING AUDIT LOG. Write enhanced logging messages to the messaging audit log.
Enable producer message ID tracking	Select to indicate that API Gateway writes additional log messages when a message producer uses this JMS connection alias to send messages to a destination in Producer Message ID Tracking: Include Destinations.
Enable consumer message ID tracking	Select to indicate that API Gateway writes additional log messages for messaging consumers (triggers) that use this JMS connection alias to receive messages. API Gateway writes additional log message for the JMS triggers listed in Consumer Message ID Tracking: Include Triggers
Producer message ID tracking: include destinations (semicolon delimited)	<p>The destination names for which API Gateway performs additional logging when sending messages to the destination.</p> <p>Use a semicolon (;) to separate each destination name. Leave this field blank if you want API Gateway to perform enhanced logging for every destination to which this JMS connection alias sends messages.</p>

Field	Description
Consumer message ID tracking: include triggers (semicolon delimited)	The fully qualified name of the JMS triggers for which API Gateway performs additional logging during trigger processing. Use a semicolon (;) to separate each trigger name. Leave this field blank if you want API Gateway to perform enhanced logging for every JMS trigger that uses this JMS connection alias to receive messages.

10. Click **Add**.

The JMS connection alias created is listed in a table under JMS connection alias definitions in the Messaging page.

Web Services Endpoint Alias

A web service endpoint alias represents the network address and, optionally, any security credentials to be used with web services. You can use the network address properties to enable dynamic addressing for web services. The security credentials can be used to control both transport-level and message-level security for web services.

In web service descriptors, an endpoint alias is associated with a binder. API Gateway uses a binder to collect the definitions for addresses, communication protocols, and data formats for a particular port type in one container to collect the definitions for addresses, communication protocols, and data formats for a particular port type in one container.

For a provider web service descriptors, the endpoint alias is used to construct the `location=` attribute of the address element (which is contained within the port element) when WSDL is requested for the web service. The security credentials might be used when constructing a response to a web service request. When you create a provider web service descriptor, you can specify an existing endpoint alias, which is displayed (and can be changed) from the default binder of the web service descriptors.

For a consumer web service descriptor and its associated web service connectors (WSC), the alias information (including the addressing information and any security credentials), is used at run time to generate a request and invoke an operation of the web service.

API Gateway uses message addressing endpoint aliases to send responses to endpoints other than the one that initiated or sent the request. That is, when WSAddressing is enabled and the request SOAP message contains a non-anonymous ReplyTo or FaultTo endpoint, API Gateway uses the message addressing endpoint alias to determine the authentication details to be used to send a response to the ReplyTo and FaultTo endpoints.

An endpoint alias is usually created for one or more of the following reasons:

Dynamic endpoint addressing. As the actual value of the endpoint is looked up at run time, using an endpoint alias saves you from having to specify or change the server information each time you use the web service.

Security. An endpoint alias is required in order to configure WS-Security for web service providers and consumers.

When you create web service endpoint aliases, keep the following points in mind:

- Alias names must be unique within the specified usage (provider or consumer) and protocol. This can result in multiple endpoint aliases with the same name. For example, you can have a provider alias named aliasOne for the HTTP protocol. You could also have a consumer alias named aliasOne for the HTTP protocol and a provider alias named aliasOne for the HTTPS protocol.

API Gateway saves web service endpoint aliases at the location, *Integration Server_directory\instances\instance_name\config\endpoints*. The host name and port are required for a provider HTTP or HTTPS web service endpoint alias, but are optional for a consumer HTTP or HTTPS web service endpoint alias.

If API Gateway on which a consumer web service descriptors reside, is located behind a firewall and the web service request needs to be routed through a proxy server, you can assign a proxy alias to the consumer web service endpoint alias. You can identify default provider web service endpoint aliases for HTTP and HTTPS. If a provider web service descriptor contains a binder set to the default alias, API Gateway uses the information in the default alias when constructing the WSDL for the descriptor.

Creating an Endpoint Alias for a Provider Web Service Descriptor

If a provider web service descriptor binder specifies the JMS transport, you must assign a web service endpoint alias to the binder. For a web service descriptor that uses SOAP over JMS, the provider web service endpoint alias provides the following information: JMS message header information for the request message, such as delivery mode, time to live, and the destination for replies. Integration Server uses this information to populate the binding elements in the WSDL generated for the web service descriptor.

The SOAP-JMS trigger that listens for SOAP over JMS messages for the web service descriptor. The SOAP-JMS trigger also provides the JMS connection information needed to create a connection on the JMS provider. Integration Server uses the information provided by the SOAP-JMS trigger to construct most of the JMS URI (the web service descriptor determines the targetService). The JMS URI appears in the WSDL document as the value of the "location=" attribute for the address element within the port element. WS Security Properties that specify the information needed by the SOAP processor to decrypt and verify the inbound SOAP request and/or encrypt and sign the outbound SOAP response and the details for adding the timestamp information.

Keep the following information in mind when creating a web service endpoint alias for a JMS binder in a provider web service descriptor:

- You can associate the web service endpoint alias with
 - A SOAP-JMS trigger that already exists.
 - A WS endpoint trigger that you create at the same time you create the endpoint alias.
- If you use a SOAP-JMS trigger in the web service endpoint alias and subsequently assign the alias to a JMS binder in a provider web service descriptor, the web service descriptor has a

dependency on the SOAP-JMS trigger. Consequently, at start up or when reloading the package containing the web service descriptor, API Gateway must load the SOAP-JMS trigger before loading the web service descriptor. If the SOAP-JMS trigger and web service descriptor are not in the same package, you need to create a package dependency for the package that contains the web service descriptor on the package that contains the SOAP-JMS trigger.

- If you rename the SOAP-JMS trigger assigned to an alias, you need to update the alias to use the renamed trigger.

➤ To create a provider web service endpoint alias for use with JMS

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Web services**.

API Gateway displays a list of all the currently defined endpoint aliases.

3. Click **Add web service endpoint alias**.
4. In the Webservice endpoint alias properties section, provide the following information:

Field	Description
Alias	Name of the JMS provider web service endpoint alias. The alias name cannot include the following special characters: # ©\ & @ ^ ! % * : \$. / \ \ ` ; , ~ + =) (} {] [> < "
Description	A description for the endpoint alias.
Type	Specifies the type - Provider

5. In the JMS transport properties section, provide the following information:

Field	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 message that serves as the request message for the web service. You can select one of the following modes: <ul style="list-style-type: none">■ PERSISTENT. Specifies that the request message should be persistent. The message is not lost if the JMS provider fails.

Field	Description
	<ul style="list-style-type: none">■ NON_PERSISTENT. Specifies that the request message is not persistent. The message might be lost if JMS provider fails.
Time to live	<p>The number of milliseconds that can elapse before the request message expires on the JMS provider.</p> <p>If you specify a value 0, it indicates that the message does not expire.</p>
Priority	<p>Specifies the message priority.</p> <p>The JMS standard defines priority levels from 0 to 9, with 0 as the lowest priority and 9 as the highest.</p>
Reply to name	<p>Name or lookup name of the destination to which the web service sends a response (reply) message.</p> <p>Specify a name if the JMS connection alias used by the SOAP-JMS trigger connects to the webMethods Broker natively. Specify a lookup name if the JMS connection alias uses JNDI to retrieve a connection factory that is then used to connect to the JMS provider.</p>
Reply to type	<p>Type of destination to which the web service sends the response (reply) message.</p> <p>Specify the destination type if the following are true:</p> <ul style="list-style-type: none"> ■ The web service descriptor to which the endpoint alias is assigned use the In-Out message exchange pattern. ■ The JMS connection alias specified by the SOAP-JMS trigger connects to the webMethods Broker natively. On the webMethods Broker, a queue and topic can have the same name. You must specify Reply To Type to indicate to which destination the reply will be sent. <p>Select one of the following destination types:</p> <ul style="list-style-type: none"> ■ Queue. Specifies that the web service sends the response message to a particular queue. ■ Topic. Specifies that the web service sends the response message to a particular topic.

6. In the JMS WSDL options section, provide the following information:

Field	Description
Include connection factory name	When selected, includes the connection factory name in the JMS URI.
Include JNDI parameters	When selected, includes the JNDI parameters in the JMS URI.

Note:

The JMS URI appears in the WSDL document as the location attribute value for the address element contained within the port element.

7. In the WS security properties section, provide the following information:

Field	Description
Keystore alias	<p>Alias of the keystore containing the private key used to decrypt the inbound SOAP request or sign the outbound SOAP response.</p> <p>Note: The provider must have already given the consumer the corresponding public key.</p>
Key alias	<p>Alias of the private key used to decrypt the request or sign the response.</p> <p>The key must be in the keystore specified in Keystore alias.</p>
Truststore alias	The alias for the truststore that contains the list of CA certificates that API Gateway uses to validate the trust relationship.
Timestamp precision	<p>Specifies whether the timestamp is precise to the second or millisecond.</p> <p>If you set the precision to milliseconds, API Gateway uses the timestamp format yyyy-MM-dd'T'HH:mm:ss:SSS'Z'. If you set the precision to seconds, API Gateway uses the timestamp format yyyy-MM-dd'T'HH:mm:ss'Z'.</p> <p>If you do not select a precision value, API Gateway uses the value specified for the <code>watt.server.ws.security.timestampPrecisionInMilliseconds</code> parameter.</p>
Timestamp time to live (seconds)	Specifies the time-to-live value for the outbound message in seconds.

Field	Description
	<p>API Gateway uses the time-to-live value to set the expiry time in the Timestamp element of outbound messages. The Timestamp Time to Live value must be an integer greater than 0.</p> <p>If you do not specify a Timestamp time to live value, API Gateway uses the value specified for the <code>watt.server.ws.security.timestampTimeToLive</code> parameter.</p>
	<p>Note: The Timestamp time to live value should be greater than the Time to live value specified under JMS transport properties.</p> <p>Timestamp maximum skew (seconds) Specifies the maximum number of seconds that the web services client and host clocks can differ and still allow timestamp expiry validation to succeed. Specify a positive integer or zero.</p> <p>API Gateway uses the timestamp maximum skew value only when you implement WS-Security via a WS-Policy. API Gateway validates the inbound SOAP message only when the creation timestamp of the message is less than the sum of the timestamp maximum skew value and the current system clock time.</p> <p>If you do not specify a timestamp maximum skew value, API Gateway uses the value specified for the <code>watt.server.ws.security.timestampMaximumSkew</code> parameter.</p>

8. Click **Add**.

The web service endpoint alias created is listed in a table under Web service endpoints list.

Creating an Endpoint Alias for a Consumer Web Service Descriptor

A web service endpoint alias for use with a consumer web service descriptor that has a JMS binder specifies how and where API Gateway sends a request message when executing a web service descriptor.

When creating a consumer web service descriptor, API Gateway extracts the JMS information from the WSDL document and saves it with the binder information in the web service descriptor. However, as indicated in the SOAP over Java Message Service standard, the only JMS information required in the WSDL is the lookup variant and the destination name. Consequently, it is possible that some information necessary to connect to the JMS provider is absent from the WSDL. API

Gateway uses the information in a JMS consumer web service endpoint alias to replace or supplement the JMS information specified in the WSDL document.

When creating a consumer web service descriptor, the message addressing properties define the WS-addressing headers information that can be attached to the SOAP message.

Keep the following points in mind when creating a web service endpoint alias for use with a consumer web service descriptor with a SOAP over JMS binding:

- A JMS consumer web service endpoint alias can specify one of the following options to connect to a JMS provider:
 - JNDI provider alias and a connection factory
 - JMS connection alias

Only specify a JNDI provider alias and connection factory, or JMS connection alias, if information for connecting to the JMS provider was not included in the WSDL document used to create the consumer web service descriptor or if you want to overwrite the connection information included in the WSDL document.

- If you want to use the client side queue with the web service descriptor to which the alias is assigned, you must specify a JMS connection alias as the way to connect to the JMS provider.
- Information in the JMS consumer web service endpoint alias can supplement or replace the JMS URI information obtained from a WSDL.

➤ To create a consumer web service endpoint alias for use with JMS

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Web services**.

API Gateway displays a list of all the currently defined endpoint aliases.

3. Click **Add web service endpoint alias**.
4. In the Webservice endpoint alias properties section, provide the following information:

Field	Description
Alias	Name of the JMS provider web service endpoint alias. The alias name cannot include the following special characters: # ©\ & @ ^ ! % * : \$. / \ \ ` ; , ~ + =) (} {] [> < "
Description	A description for the endpoint alias.
Type	Specifies the type - Consumer

Field	Description
Execute ACL	<p>Specifies the ACL that governs which user groups on your server can use this web service endpoint alias.</p> <p>Select an ACL from the drop-down list. By default, only members of groups governed by the Internal ACL can use this alias.</p>

5. In the JMS transport properties section, provide the following information depending upon whether you want to connect to the JMS provider using a connection factory or a JMS connection alias:

Field	Description
JNDI properties	<p>Select this option if you want to connect to the JMS provider using a connection factory.</p> <p>Provide the following information:</p> <ul style="list-style-type: none"> ■ JNDI provider alias. The alias for the JNDI provider that API Gateway uses to look up administered objects. ■ Connection factory name. The lookup name for the connection factory to use to create a connection to the JMS provider. <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>Note: You have to specify a connection factory only if the WSDL document used to create the consumer web service descriptor does not specify a connection factory or you want to overwrite the connection factory.</p> </div>
JMC connection alias	<p>Select this option if you want to connect to the JMS provider using a JMS connection alias.</p> <p>Provide the following information:</p> <ul style="list-style-type: none"> ■ JMS connection alias. The name of the JMS connection alias that you want API Gateway to use to connect to the JMS provider.

6. In the WS security properties section, provide the following information:

Field	Description
Username	The user name to include with the UsernameToken.

Field	Description
Password	The password to include with the UsernameToken (must be plain text).
Retype password	Re-type the above password.
Keystore alias	<p>Alias to the keystore that contains the private key used to:</p> <ul style="list-style-type: none"> ■ Sign outbound SOAP requests ■ Include an X.509 authentication token for outbound SOAP requests ■ Decrypt inbound SOAP responses
	<p>Note: To verify messages from this consumer, the web services provider must have a copy of the corresponding public key.</p>
Key alias	<p>Alias to the private key used to sign or include X.509 authentication token for outbound SOAP messages or decrypt inbound SOAP responses.</p> <p>The key must be in the keystore specified in Keystore alias.</p>
Truststore alias	The alias for the truststore that contains the list of CA certificates that API Gateway uses to validate the trust relationship.
Timestamp precision	<p>Specifies whether the timestamp is precise to the second or millisecond.</p> <p>If you set the precision to milliseconds, API Gateway uses the timestamp format yyyy-MM-dd'T'HH:mm:ss:SSS'Z'. If you set the precision to seconds, API Gateway uses the timestamp format yyyy-MM-dd'T'HH:mm:ss'Z'.</p> <p>If you do not select a precision value, API Gateway uses the value specified for the <code>watt.server.ws.security.timestampPrecisionInMilliseconds</code> parameter.</p>
Timestamp time to live (seconds)	<p>Specifies the time-to-live value for the outbound message in seconds.</p> <p>API Gateway uses the time-to-live value to set the expiry time in the <code>Timestamp</code> element of outbound messages.</p>

Field	Description
	<p>The Timestamp Time to Live value must be an integer greater than 0.</p>
	<p>If you do not specify a Timestamp time to live value, API Gateway uses the value specified for the watt.server.ws.security.timestampTimeToLive parameter.</p>
	<p>Note: The Timestamp time to live value should be greater than the Time to live value specified under JMS transport properties.</p>
Timestamp maximum skew (seconds)	<p>Specifies the maximum number of seconds that the web services client and host clocks can differ and still allow timestamp expiry validation to succeed.</p>
	<p>Specify a positive integer or zero.</p>
	<p>API Gateway uses the timestamp maximum skew value only when you implement WS-Security via a WS-Policy. API Gateway validates the inbound SOAP message only when the creation timestamp of the message is less than the sum of the timestamp maximum skew value and the current system clock time.</p>
	<p>If you do not specify a timestamp maximum skew value, API Gateway uses the value specified for the watt.server.ws.security.timestampMaximumSkew parameter.</p>

7. Click **Add**.

The web service endpoint alias created is listed in a table under Web service endpoints list.

JMS Triggers

JMS Triggers are meant only for SOAP APIS. These triggers are created automatically when you create a Web Service Provider endpoint and act as Listener for that Web Service Provider endpoint.

API Gateway provides ways for managing JMS triggers and the resources used by JMS triggers. You can only update a trigger and cannot create a trigger. Specifically, you can use the controls provided by API Gateway to:

- Increase or decrease the maximum number of server threads used for JMS triggers
- Change the maximum execution threads for concurrent JMS triggers
- Change the destinations to which the trigger subscribes

- Change the JMS connection alias used by the trigger
- Delay the frequency with which a JMS trigger polls for more messages

The Individual SOAP JMS trigger controls section displays all the JMS triggers that exist on the API Gateway along with a summary of each trigger. The summary includes the current status, state, and thread usage of the trigger as well as configuration information such as the JMS connection alias used by the trigger, the destination to which the trigger subscribes, and the processing mode of the trigger.

JMS trigger status and state

The state of a JMS trigger indicates whether the trigger is enabled, disabled, or suspended. The status indicates whether trigger is running.

A JMS trigger can have one of the following states:

- **Enabled.** The JMS trigger is available. A JMS trigger must be enabled for it to receive and process messages. An enabled trigger can have a status of “Not Running” which means that it would not receive and process messages. Reasons that an enabled JMS trigger can be disabled include: a disabled JMS connection alias, an exception thrown by the trigger, and trigger failure at startup.
- **Disabled.** The JMS trigger is not available. Integration Server neither retrieves nor processes messages for the JMS trigger. The JMS trigger remains in this state until you enable the trigger.

A JMS trigger can have a status of “Running” or “Not Running (reason)” where reason identifies why the trigger is not running, such as “Not Running (trigger disabled)”.

You can enable a trigger by clicking the toggle button . The toggle button changes to  to depict that the trigger is enabled.

You can disable the trigger by clicking the toggle button . The toggle button changes to  to depict that the trigger is disabled.

Updating JMS triggers

When API Gateway creates a JMS trigger as part of creating a provider web service endpoint alias, API Gateway uses default values for some of the trigger properties and placeholders for other properties. You can modify the default and placeholder values in the SOAP JMS trigger controls.

➤ To update a JMS trigger

1. Expand the menu options icon , in the title bar, and select **Administration**.
2. Select **General > Messaging**.

The SOAP JMS trigger controls section displays a list of all the currently defined JMS triggers and the corresponding details.

3. Click a JMS trigger.

The Update JMS trigger screen appears.

4. In the JMS destinations and message selectors section you can modify the following information as required:

- a. **Destination name.** Type the name of the queue from which the JMS trigger receives messages.

5. In the Settings section you can modify the following information as required:

- a. **JMS connection alias.** Specify the JMS connection alias used by the trigger.

6. In the Properties section you can select one of the following Processing modes:

- **Concurrent.** Specifies that API Gateway processes multiple messages for this trigger at one time. You can modify the following information for this processing mode:
 - **Max execution threads.** Specify the maximum number of messages that API Gateway can process concurrently.
 - **Connection count.** Specify the number of connections you want the JMS trigger to make to the webMethods Broker.

Note:

The Connection count value must be less than or equal to the Max Execution Threads value.

- **Serial.** Specifies that API Gateway processes messages received by the trigger one after the other.

7. Click **Update**.

The trigger is saved with the modified values.

Users, Groups, and Teams

You can use API Gateway to define user information on the API Gateway server. The definition of user contains the login ID, password, and group membership.

Alternatively, you can set up API Gateway to access the information from a local user management system or you can use webMethods Integration Server to configure the Lightweight Directory Access Protocol (LDAP) external directory that your site uses for user information.

Note:

Central User Management is not supported by API Gateway.

webMethods Integration Server uses user information to authenticate clients and determine the server resources that a client is allowed to access. If the server is using basic authentication

(username and password) to authenticate a client, it uses the login ID and passwords defined in user accounts to validate the credentials a client supplies.

API Gateway enables you to define user and group information to the API Gateway server. The user definition contains the user login ID, password, and group membership. The group definition contains the group name and a list of users in the group. After creating users and groups, users can be given the required functional privileges based on the teams that they are part of. A user can have different set of functional privileges in the teams that they are part of. For example, a user can have administrative privileges in a team and view privileges in another.

You can add and manage user information from the User Management page. This page lists all the basic information for the following:

- Users. User personas who can access API Gateway and perform tasks. A predefined user is an Administrator who has administrator privileges.
- Groups. The group membership identifies the groups to which a user belongs. User can create a group, associate users to the group, and delete a group in API Gateway.
- Teams. Users who share a common role or responsibility can be grouped as teams. When the Team feature is enabled, the members of teams can access the API Gateway assets of their teams and they can perform actions on these assets based on the functional privileges assigned to their teams.
- Account settings. You can define the password restrictions, password expiry and the account lock settings here.
- LDAP configuration. You can configure API Gateway to use LDAP and manage LDAP directories here.

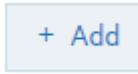
Adding a User

You must have the API Gateway's manage user administration functional privilege assigned to add a user to API Gateway.

➤ To add a user

1. Expand the menu options icon  in the title bar, and select **User management**.
2. Click **Users**.
3. Click **Add user**.
4. Provide the following information in the Basic information section:

Field	Description
Login ID	A unique ID using which the user can log on to the account.

Field	Description
First name	A first name that contains letters, numbers, or a combination of all. You can also use special characters: . (dot), _ (underscore), and @ (at). Other special characters and spaces are not allowed. The user name is case sensitive.
Last name	A last name that contains letters, numbers, or a combination of all. You can also use the special characters: . (dot), _ (underscore), and @ (at). Other special characters and spaces are not allowed. The user name is case-sensitive.
Password	A password that contains letters, numbers, special characters, or a combination all. Spaces are not allowed. The password is case-sensitive.
Confirm password	Retype your password to confirm.
Email addresses	A valid email address of the user. You can add multiple email addresses by clicking  + Add
Allow digest authentication	Allow Digest Access Authentication to authenticate the API as described in RFC2617.

5. Click **Continue to associate Groups >**.

Alternatively, you can click **Groups** to go to the Groups section and associate the user to groups. You can search for the group name in the **Name** field and associate the user to the group selected. You can associate a user to multiple groups by clicking +.

Click **Save** to save the user details at this stage and provide the group information for the user at a later time.

6. Provide the group name in the **Name** field to which the user is added.

7. Click **Save**.

Note:

After adding an API Gateway user, you must include the user in a group that is associated with a team.

Modifying User Details

You must have the API Gateway's manage user administration functional privilege assigned to modify user details.

You can modify the basic or the group information of a user. You can add the user to a different group or delete the user from an existing group.

➤ To modify the user details

1. Expand the menu options icon  in the title bar, and select **User management**.
2. Click **Users**.

A list of available users appears.

3. Select the login ID of the user to be modified.

The User details tab appears.

4. Click **Edit**.

This opens the basic information of the user.

5. Modify the basic information of the user.

Note:

Select **Active** if you want to make the user an active user.

6. Modify the group details of the user.

You can modify or delete the name of the existing group that appears.

7. Click **Save**.

Deleting a User

Deleting a user removes the user from all the associated groups.

➤ To delete a user

1. Expand the menu options icon  in the title bar, and select **User management**.
2. Click **Users**.

A list of available users appears.

3. Click the delete icon for the user that has to be deleted.
4. Click **Yes** in the confirmation dialog.

User Groups

API Gateway is shipped with the following predefined groups:

- Administrators
- API-Gateway-Administrators
- API-Gateway-Providers

By default, API Gateway's Administrator user, is part of Administrators and API-Gateway-Administrators group.

The table lists the privileges based on the user group.

Privileges	API Gateway Administrator	API Provider
Manage APIs	Y	Y
Manage aliases	Y	Y
Manage policy templates	Y	N
Activate/Deactivate APIs	Y	Y
Manage global policies	Y	N
Manage threat protection configurations	Y	N
Manage applications	Y	Y
Activate/Deactivate global policies	Y	N
Publish API to service registry	Y	Y
Manage packages and plans	Y	Y
Activate/Deactivate packages	Y	Y
Publish to API Portal	Y	Y
View administration configurations	Y	N
Execute service result cache APIs	Y	Y
Manage user administration	Y	N
Change ownership/teams	Y	N

Privileges	API Gateway Administrator	API Provider
Manage general administration configurations	Y	N
Manage destination configurations	Y	N
Manage promotions	Y	Y
Manage scope mapping	Y	N
Manage security configurations	Y	N
Manage system settings	Y	N
Manage service registries	Y	N
Import assets	Y	Y
Export assets	Y	Y
Manage purge and restore runtime events	Y	N
Manage microgateways	Y	N
Manage custom dashboards	Y	N

Authentication and Authorization

API Gateway is primarily accessed using API Gateway user interface, which supports Basic authentication and SAML SSO.

You can also use REST APIs to manage API Gateway. To invoke the APIs, you must have the required functional privileges.

Note:

You cannot delete predefined users, groups, and teams but you can delete the groups and access profiles that are created in API Gateway.

Adding a Group

You can add the required users to a group.

➤ **To add a group**

1. Expand the menu options icon , in the title bar, and select **User management**.
2. Select **Groups**.

3. Click **Add group**.

The Group details tab appears.

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

Field	Description
Name	Name of the user group to add.
Description	A description for the user group.

Click **Save** to save the group details at this stage and provide the group information for the user at a later time.

5. Click **Continue to associate Users >**.

Alternatively, you can click **Users** to go to the Users section.

6. Provide the user's login ID in the **Login ID** field.

You can search users based on the characters provided in user name and email id. Select the required user from the list displayed.

7. Click **Save**.

Modifying a Group

You can modify details for a selected group. You can add users to a group or remove them if required.

➤ To modify a group

1. Expand the menu options icon  in the title bar, and select **User management**.
2. Click **Groups**.

A list of groups appears.

3. Select the group to be modified.

The Group details tab appears.

4. Click **Edit**.

This opens the details of the selected group.

5. Modify the basic information of the user.

6. Modify the user details of the group.

Edit or delete (by clicking the delete icon) the name of the existing user that appears.

7. Click **Save**.

Deleting a Group

You can delete a group from the list that appears in the Groups section of the User Management page. Once a group is deleted, the user associated to the group is unable to perform the tasks associated to that group. Deleting a group does not delete the users associated with the group.

➤ To delete a group

1. Expand the menu options icon  in the title bar, and select **User management**.

2. Click **Groups**.

A list of groups appears.

3. Click the **Delete** icon for the group that has to be deleted.

Note:

You cannot delete predefined groups.

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

API Gateway Functional Privileges

The following table lists the functional privileges and their description:

Functional Privilege	Description
Select all	To select all the listed functional controls.
Manage APIs	To create and manage APIs.
Activate/ Deactivate APIs	To activate, deactivate, and manage APIs.
Manage applications	To create, manage applications, and register applications with the APIs.
Manage aliases	To create and manage aliases.
Manage global policies	To apply a global policy to all APIs or the selected set of APIs.
Activate/Deactivate global policies	To activate, deactivate, and manage global policies.

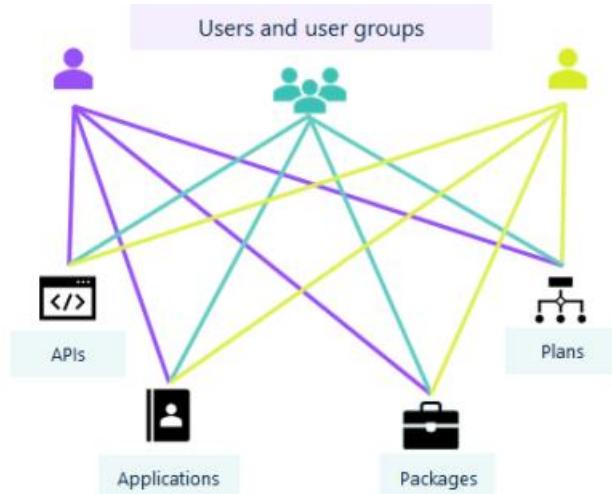
Functional Privilege	Description
Manage policy templates	To apply one or more policy templates to an API.
Manage threat protection configurations	To prevent malicious attacks on applications that typically involve large, recursive payloads, and SQL injections.
Publish API to service registry	To publish and unpublish APIs to service registry.
Manage packages and plans	To create packages and plans, associate a plan with a package, and associate APIs with a package. In addition, you can view the list of packages, package details, APIs, and plans associated with the package.
Activate/ Deactivate packages	To activate, deactivate, and manage packages.
Publish to API Portal	To publish and unpublish assets to API Gateway.
View administration configurations	To view administration configurations.
Manage general administration configurations	To create and manage administration configurations.
Manage security configurations	To create and manage security configurations.
Execute service result cache APIs	To execute service result cache API.
Manage destination configurations	To publish events and performance metrics data to the configured destinations.
Manage system settings	To create and manage system settings.
Manage user administration	To create and manage users.
Manage promotions	To create stages and manage promotions.
Manage service registries	To create and manage service registries.
Change ownership/ teams	To change ownership of an asset or teams.
Manage scope mapping	To manage OAuth and OpenID scopes.
Import assets	To import already exported APIs, application, policies, aliases, or other assets and configurations using the Import option in the Menu options (E).
Export assets	To export assets to your local system.
Manage purge and restore runtime event	To purge and restore events from the API Data Store by setting the required date or duration in the API Gateway.
Manage microgateways	To manage the Microgateways connected to the API Gateway instance.

Functional Privilege	Description
Manage custom dashboards	To manage custom dashboards in Global Analytics . You can not manage custom dashboards if you do not have this privilege.

Team Support

When users from multiple business units of an organization share the same API Gateway instance, by default, all users have access to all assets irrespective of their departments.

In a typical deployment scenario, all users have same level of access privileges to all API Gateway assets. However, there could be requirement for users from different business units to have different levels of access to specific assets; and they would not want interference from each other.

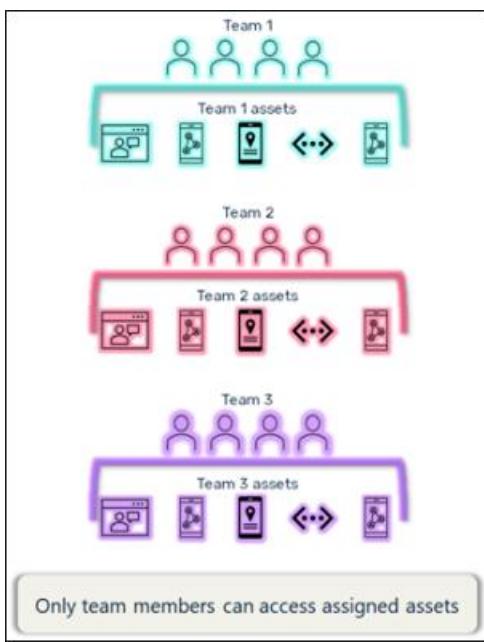


All users and user groups have access to all assets based on their access privileges

As a platform administrator, keeping in mind the various role-based access requirements, how do I

- group users of a business unit or a project with similar roles and assign certain assets to these teams?
- assign different access privilege to different set of users to specific assets?

This is where the **Team support** feature of API Gateway is useful. In a shared environment, this feature enables you to provide different level of access to different users to access specific assets.



The **Team support** is disabled by default; and you must enable the feature to use it. For information about enabling the feature, see “[Enabling Team Support](#)” on page 437.

When to use Team support?

Team support can be used to group the users of a business unit or users with similar roles in your organization. Using this feature, you can assign assets for each team and specify the access level of team members based on the team members' project requirements.

This feature is helpful for organizations that have multiple business units, who work on different projects. Users can access only the assets that are assigned to them. For example, consider an organization with different teams such as Development, Configuration Management, Product Analytics, and Quality Assurance. Each of these teams needs access to different assets at different levels. That is, developers would require APIs to develop applications and they require the necessary privileges to manage APIs and applications. Similarly, analysts would want the necessary privileges to view performance dashboards of assets. In such scenarios, you can group users based on their roles as a team and assign them the necessary privileges based on their responsibility.

Prior to the 10.5 version, users were given the necessary privileges using Access Profiles. Starting version 10.5, you can limit the access of your asset to the required team members and assign access privileges using the **Team support** feature.

What is a Team in API Gateway?

A team can be defined as a group of users with a set of defined responsibilities.

The assets supported by this feature are APIs, applications, packages, and plans.

This table lists the important points on API Gateway behavior with and without the **Team support** feature:

Without Team support	With Teams support
All users can view all details for all assets.	User can view only the assets that are assigned to the teams that they are a part of.
Users can add, modify, delete, activate and deactivate assets, publish and promote assets, export and import assets based on their functional privileges.	Privileges are assigned through teams. Users can perform actions based on their team's functional privileges.
Users can manage assets based on the functional privileges assigned to the teams they belong to. For details on asset visibility and accessibility, see " Team Support Considerations " on page 450.	

When **not** to use Team support?

You do not use the **Team support** feature when you require:

- **Tenant isolation.** If your requirement is to allow the access of assets by tenants, then you must have multiple tenants and isolate them from each other. The **Team support** feature does not address this requirement.
- **Full access management.** Users gain access based on their team privileges. There is no role-based access for users to an asset.

Teams management using API Gateway

You can create teams from the **User Management** section of the API Gateway UI by including the required user groups and assigning them the required functional privileges. You can also assign a Team administrator for each team, who can add or modify team members.

Users with the **Manage user administration** privilege can create teams. When creating a team, you can assign:

- **Team administrator.** You can assign a user or a user group as team administrator. Team administrators can add or remove users from a team. When you assign a user group as team administrator, all users of the groups can modify team members. When team administrators, who do not have the **Manage user administration** functional privilege log on to API Gateway, they can view only the teams assigned to them in the **Teams** tab of the **Administration** page.
- **Functional privileges for the team members.** The functional privileges assigned to a team determines the accessibility of assets to the respective team members. For example, if you assign all privileges under the APIs, Policies, and Applications sections, then the team members can manage APIs and applications assigned to their teams and perform operations related to policies.
- **Team members.** You can assign users and user groups to the team. Team members can access the assets assigned to their teams and perform operations on the assets based on the functional privileges assigned to the team.

Teams - Asset Association

After you have created teams, you can assign assets to teams in one of the following ways:

- **Assign team during asset creation.** When you create an asset, API Gateway provides an option to select the teams for the asset. You can select more than one team for an asset. You can modify the teams assigned by following the Change ownership process. For information about the process, see “[How do I Modify Teams Assigned to an API?](#)” on page 446 and “[How do I Change the Ownership of Multiple Teams?](#)” on page 448.
- **Using Global Team Assignment rule.** This is a preferred method to assign teams when you already have assets to which you want to assign teams to. You can create global assignment rules that are applied to assets and assign teams to them. You can specify one or more conditions in a rule. When an asset satisfies the conditions specified in a rule, the asset is assigned to the teams specified in the rule. When you create and activate a rule, the rule is applied to the existing assets and teams are assigned accordingly.

If you already have assets in your API Gateway instance and when you enable the **Teams support** feature, all assets are assigned to the **Default** team. Every user is automatically assigned to this team. That means, by default, every asset (APIs, applications, packages and plans) are still visible to all users. Access rights are restricted only if the asset is explicitly assigned to one or more teams during its creation.

You must manually remove the **Default** team from the respective asset details page.

Software AG recommends that you read the Team support considerations section to see the impact of Team support on other features. For more information, see “[Team Support Considerations](#)” on page 450.

Enabling Team Support

When you enable the **Team support** feature, you can manage teams from the **Teams** tab under the **User management** section of API Gateway.

If you do not enable this feature, you can still create teams, as explained in “[Creating Teams](#)” on page 438, and assign functional privileges to users. However, you cannot assign required assets to a set of users and restrict the access of assets to other users.

If you have enabled this feature, created teams, and assigned assets to teams, and then disable this feature, then the team assignments that you had performed earlier become invalid. That is, the assets are available to users based on their functional privileges and not based on the assigned teams.

➤ To enable teams

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Extended settings**.
3. Click **Show and hide keys**.

All configurable parameters appear.

4. Select the **enableTeamWork** from the parameter list.

The **enableTeamWork** field appears in the **Extended Settings** section.

5. Type *true* in the **enableTeamWork** field. By default, the value of the setting is set as *false*.

The feature is enabled.

Creating Teams

This use case explains how to create teams by assigning the required functional privileges and users to them.

This use case begins when you have identified the list of users who must given access to an asset or a particular set of assets and end when you have created a team including the identified users.

In this example, a team with developers called *DevTeam* is created with the *Dev* user group as the team members, *User1* as the Team administrator, and all privileges under Manage API, Policies, and Applications are assigned to the team. The user, *User1*, is also allowed to approve the pending requests of the applications that the team owns.

Before you begin

Ensure that you have:

- API Gateway Manage user administration privilege.
- The user group, *Dev* is created. For information on how to create a user group, see “[Adding a Group](#)” on page 430.

➤ To create teams

1. Expand the menu options icon  in the title bar, and select **User management**.

2. Click **Teams**.

3. Click **Add Team**.

The **Create Team** page appears.

4. Provide the name and description of the team in respective fields.

5. In the **Team Administrators** section, provide any or both of the following:

- Login Id of the user who want to assign as a team administrator in the **Login ID** field.
- Name of the API Gateway user group or the LDAP group that you want to assign as team administrator in the **Group name** field.

You can search users or user groups based on the characters provided in the above fields. Select the required user from the list displayed.

For the example considered in this use case, the team is named as *DevTeam* and the *User1* is specified as the team administrator.

The screenshot shows the 'Create Team' interface. In the 'Team details' section, the 'Name*' field is filled with 'DevTeam'. The 'Description' field contains 'All developers'. Under 'Team Administrator', there is a table with one row containing 'User1'. In the 'Approvers' section, there is another table with one row containing 'User1'. A checkbox labeled 'Include team administrators as approvers' is checked. At the bottom, a blue link reads 'Continue to assign functional privileges >'.

6. In the **Approvers** section, provide any or both of the following:
 - **Users.** Users who can view the pending requests of assets that the team is associated with. You can specify login Id of the required users.
 - **Group name.** User groups that can view pending requests of assets that the team is associated with. The users from the specified groups can approve the pending requests of applications that the team owns and approve them. You can specify names of the required API Gateway user groups or the LDAP groups.

You can search users or user groups based on the characters provided in the above fields. Select the required user from the list displayed.

For the example considered in this use case, the *User1* is specified as the approver.

7. Select the **Include team administrators as approvers** option.

If selected, the users and user groups specified in the **Team Administrators** section can view the pending requests of the applications that the team is associated with and approve them.

8. Click **Continue to assign functional privileges >**.

The **Functional privileges** list appears.

9. Select the functional privileges to be assigned to the team members. For information on the available functional privileges, see "[API Gateway Functional Privileges](#)" on page 432.

In this use case, you need to assign all privileges required to manage the APIs and applications assigned to the team. So, select all functional privileges under the **APIs, Policies, and Applications** section.

The screenshot shows the 'Functional privileges' section of a user interface. Under the 'APIs, Policies, and Applications' group, three checkboxes are selected: 'Manage APIs', 'Manage aliases', and 'Manage policy templates'. A red box highlights this group. Other groups listed include 'API Portal Management', 'Administration configurations', 'Export or Import assets and Purge and Archive events', and 'Microgateway register and de-register'. Each group contains several sub-options with checkboxes.

10. Click **Continue to assign groups >**.

The **Find groups** section appears.

11. In the **Name** field, provide the name of the user group that you want to add as team members.

For this use case, select the *Dev* user group that has all developers.

The screenshot shows the 'Create Team' page. In the 'Find groups' section, the 'Name' field contains 'de'. Below it, a table lists a single row with 'Name' 'Dev' and an 'Action' column containing a trash can icon. The 'Dev' entry is highlighted with a red box.

12. Click **Save**.

The team *DevTeam* is created and appears in the list of teams.

Name	Description	Actions
Administrators	Groups associated to this team are allowed to perform all the administration related tasks.	<input type="checkbox"/>
API-Gateway-Providers	Groups associated to this team are allowed to access an asset based on the functional privileges assigned to this team.	<input type="checkbox"/>
Default	All users in APIGateway will be part of this team.	<input type="checkbox"/>
DevTeam	All developers	<input type="checkbox"/>

You can now assign assets to the team.

How do I Assign Teams during Asset Creation?

This use case explains how to assign teams for an API Gateway asset.

The use case starts when you have an asset that you want to allow access only to a set of users in your organization and ends when you have assigned teams to an asset.

This example provides the steps to assign the asset, *DevAPI*, that is being created to a team that has all developers as team members.

Before you begin

Ensure that you have:

- API Gateway Manage user administration privilege.
- The team support feature enabled. For information on enabling the feature, see “[Enabling Team Support](#)” on page 437.
- The team, *DevTeam* is created. For information on creating team, see “[Creating Teams](#)” on page 438.

➤ To assign teams during asset creation

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

The **Manage APIs** page appears.

2. Click **Create API**. The **Create API** page appears.
3. Click **Import from an File**.
4. Click **Browse** to select the file using which you want to create the API.
5. Provide *DevAPI* in the **Name** field.
6. From the **Team** drop-down list, select the teams that you want to assign the asset to. For this use case select the *DevTeam*.

The screenshot shows the 'Create API' interface. In the 'Import API from file' section, a file named 'APIGatewayAdministration.json' is selected. The 'Name' field contains 'DevAPI', 'Type' is 'Swagger', and 'Version' is '1.0'. The 'Team' dropdown is set to 'Dev'. To the right, a 'Description' field says 'API meant for developers'. Below the team dropdown, a table lists 'Team' as 'DevTeam' and 'Action' as 'Add', with a small 'Edit' icon next to 'DevTeam'.

The assigned teams appear in the **Team** field of **Basic Information** section in the **API details** page. In this example, the asset, *DevAPI*, is assigned to *DevTeam*.

The screenshot shows the 'DevAPI' API details page. On the left, a sidebar lists 'API details', 'Scopes', and 'Policies'. The main area has tabs for 'Basic information' (selected), 'Technical information', 'Resources and methods', 'API mocking', 'Components', and 'Documentation'. In the 'Basic information' tab, the 'Team' field is highlighted with a red box and contains 'Administrators' and 'DevTeam'.

By default, all assets are assigned to the Administrators team in addition to the teams that you have assigned during asset creation.

How do I Assign Teams Using Team Assignment Rule?

This use case explains how to assign teams to API Gateway assets using Team assignment rules.

Team assignment rules are used to assign teams to existing assets and the ones you create. You can create a rule by specifying a set of required conditions. Assets are validated against the given conditions and assigned to the configured teams. If you do not provide any conditions for a rule, the rule is assigned to all assets in API Gateway when you activate the rule.

This section explains the steps to configure conditions and team names for creating a rule. Also, it lists the steps to activate rule to apply the rule to assets.

In this example, consider a team of users who must be enabled to view all assets. To achieve this, you can create a team called *ViewAllAssets*, and create a rule by selecting all assets and no conditions. When no conditions are specified, the rule is applied to all assets. When you activate this rule, all assets are assigned to the *ViewAllAssets* team.

The use case starts when you have a team and ends when you create a team assignment rule and activate the rule. All assets are assigned to the specified team and the team members can view all assets.

Before you begin

Ensure that you have:

- API Gateway administrator privileges.
- The team support feature enabled. For information on enabling the feature, see “[Enabling Team Support](#)” on page 437.
- Ensure that the team, *ViewAllAssets* is created. For information on creating team, see “[Creating Teams](#)” on page 438.

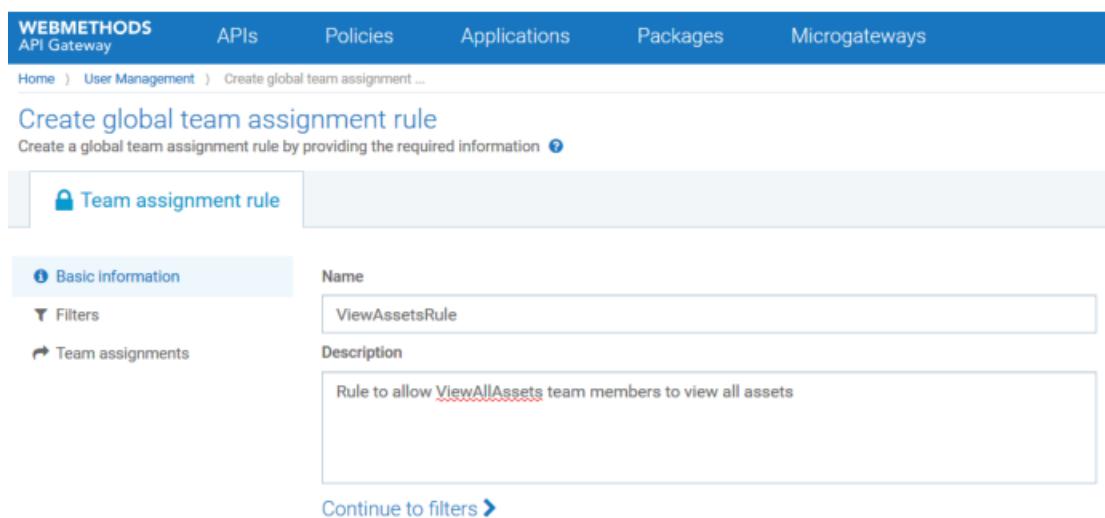
➤ To assign teams using team assignment rule

1. Expand the menu options icon  in the title bar, and select **User management**.
2. Click **Global team assignments**.
3. Click **Add global team assignment**.

The **Team assignment rule** page appears.

4. Provide the name and description of the rule in corresponding fields.

In this example, *ViewRule* is provided in the **Name** field.



Create global team assignment rule
Create a global team assignment rule by providing the required information 

 Team assignment rule	
Basic Information <input type="checkbox"/> Filters <input type="checkbox"/> Team assignments	Name ViewAssetsRule Description Rule to allow <u>ViewAllAssets</u> team members to view all assets

[Continue to filters >](#)

5. Click **Continue to filters >**.
6. Select any or all assets in the **Asset type** field to apply the rule to the selected asset types.

Available asset types are **API**, **Application**, **Plan**, and **Packages**.

7. Select any one of the following from the **Logical operator** field:

- **AND**. To apply the rule only if an asset satisfies all conditions.
- **OR**. To apply the rule when an asset satisfies any of the given condition.

8. To specify a condition based on the asset attributes, provide the following information, and click **Add**:

Field	Description
Attribute	<p>Specifies the asset attribute.</p> <p>Available attributes: Name, Description, Tags.</p> <p>The global team assignment rule supports only API level tags.</p> <p>If you select multiple asset types, the tag filter is applicable to the API type alone and is not applicable for the other asset types during filter evaluation.</p>
Operator	<p>Comparison operator to validate the attribute against the given value.</p> <p>Available operators:</p> <ul style="list-style-type: none"> ■ Equals. Checks if the specified asset attribute is equal to the given value. ■ Contains. Checks if asset contains the given value as a part of its name, description, or tag. ■ Start with. Checks if asset name, description, or tag starts with the given value. ■ Ends with. Checks if asset name, description, or tag ends with the given value.
Value	Value of the attribute.

For this use case, the rule has to be applied to all assets irrespective of their attributes. So, all asset types are selected and no condition is specified.

Create global team assignment rule
Create a global team assignment rule by providing the required information [?](#)

Team assignment rule

Basic information

Filters
Asset type
 API Application Plan Package

Filtering using asset attributes
Logical Operator
 AND OR

Attribute*	Operator*	Value*
Name	Equals	

[Continue to assign teams >](#)

9. Click **Continue to assign teams >**.

10. Provide the required team names in the **Name** field.

For this use case, select the *ViewAllAssets* team.

Create global team assignment rule
Create a global team assignment rule by providing the required information [?](#)

Team assignment rule

Basic information

Find Teams

Name	Action
ViewAllAssets	

11. Click **Save**.

The rule appears in the **Global team assignment** page.

12. Click the toggle button , adjacent to the rule.

User management
Manage users, groups and teams here. [?](#)

Global team assignments

Name	Description	Teams
RESTRule	Rule to assign REST API t...	DevTeam
ViewAssetsRule	Can view all assets across...	ViewAllAssets

The rule is activated and applied across all existing assets. As per the rule, all assets are assigned to the *ViewAllAssets* team.

13. Click **APIs** from the title bar and select *DevAPI*.

The **API details** page for the API appears.

Note that the API is assigned to the *ViewAllAssets* team as per the *ViewAssetsRule*.

The screenshot shows the 'API details' page for 'DevAPI'. In the 'Basic information' section, under the 'Team' field, the 'ViewAllAssets' team is selected and highlighted with a red box. Other teams listed are 'Administrators' and 'DevTeam'. The 'Basic information' section also displays the API's name ('DevAPI'), version ('1.0'), owner ('Administrator'), maturity state ('Beta'), creation date ('2019-09-13 13:51:15 GMT'), and description ('For developers'). The 'Technical information' section shows native endpoints at 'https://petstore.swagger.io/v2' and 'http://petstore.swagger.io/v2', and a service registry display name 'DevAPI_1.0'.

How do I Modify Teams Assigned to an API?

This use case explains how to modify the list of teams associated with an API. You can configure an approval process for the modification of teams assigned to an API, if required. You can only assign the API to the teams that you are part of. However, you cannot remove the Administrators team and the teams that are assigned to an API using global assignment rules.

The use case starts when you have an API that requires a modification in the list of teams assigned to it and ends when you successfully make the change.

In this example, an API *API* is assigned to the *Administrator* team (by default). The API has to be assigned to the *API-Gateway-Providers* team along with the existing team through an approval process.

Before you begin

- Ensure that you have the Change ownership or teams privilege.
- The change owner approval process configured and enabled if you want to enforce an approval process for ownership changes of assets.

➤ To modify the teams of an API

1. Log on to API Gateway as a user with the Change ownership/teams privilege.
2. Click **APIs** on the title navigation bar.
3. Click **API**.

The API details page appears. The asset you have considered for example is not assigned to any team. So, the default team **Administrators** is displayed in the **Team** field of the Basic information section.

Home > APIs > API

API

View API details, basic and technical information, resources and methods available, and API specifications. [?](#)

Basic information

Name	API
Version	1.0
Owner	Administrator edit
Team	Administrators edit
Active	No
Maturity state	Beta
Created	2019-08-25 03:32:57 GMT
Description	New API

4. Click **change**.
5. Select the team that you want to assign and click . In this example, select the team **API-Gateway-Providers**.

API

View API details, basic and technical information, resources and methods available, and API specifications. [?](#)

Basic information

Name	API
Version	1.0
Owner	Administrator edit
Team	<input type="text" value="Type a keyword"/> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px;"> <input type="checkbox"/> Administrators <input checked="" type="checkbox"/> API-Gateway-Providers </div>
Active	No
Maturity state	Beta
Created	2019-08-25 03:32:57 GMT
Description	New API

The change approval process is initiated.

Note:

If the approval flow is not configured, the *API-Gateway-Providers* team is added 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 team remains unchanged.

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 *API-Gateway-Providers* team is added.

The screenshot shows the 'API details' page for an API named 'API'. The 'Basic information' tab is selected. In the 'Team' field, the 'API-Gateway-Providers' team is listed, which is highlighted with a red box. Other fields shown include Name (API), Version (1.0), Owner (Administrator), Active (No), Maturity state (Beta), Created (2019-08-25 03:32:57 GMT), and Description (New API).

Field	Value
Name	API
Version	1.0
Owner	Administrator
Team	Administrators API-Gateway-Providers
Active	No
Maturity state	Beta
Created	2019-08-25 03:32:57 GMT
Description	New API

How do I Change the Ownership of Multiple Teams?

You can change the owners of multiple teams in a single step. This use case explains how to change the ownership of multiple teams 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 teams require change of owner and ends when you successfully change the ownership of the teams.

➤ To change the ownership of multiple teams

1. Use the following REST request to change the asset ownership to a new user.

```
POST http://host:port/rest/apigateway/assets/team
```

```
Content-Type: application/json
{
    "assetType": "*",
    "assetIds": ["*"],
    "currentTeams": "team1",
    "newTeams": ["team2"]
}
```

Provide the following information in the REST request:

- **assetType**. Specifies the asset type for which you want to assign new teams. Available values are API, APPLICATION, or the wildcard *. The wildcard * specifies all the assets, APIs and applications owned by the user specified in **currentTeams**.

- **assetIds**. Specifies the ID of the assets specified in **assetType**.

Note:

This is optional. **assetIds** is not required if you specify **currentOwner**.

- **currentTeams**. Specifies the current teams of the assets specified in the **assetType** field.

Note:

If both **currentTeams** 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 **currentTeams** field, then only *assetID1* ownership changes.

- **newTeams**. Specifies the teams to which you want to assign the specified assets.

Example: If all APIs assigned to the *DevTeam* must be assigned to two other teams, *Team2* and *Team3*, then send a REST request as follows:

```
POST http://localhost:5555/rest/apigateway/assets/team
Content-Type: application/json
{
    "assetType": "*",
    "currentTeams": "DevTeam",
    "newTeams": ["Team2", "Team3"]
}
```

This request assigns the ownership all APIs of *DevTeam* to *Team2* and *Team3*.

The change approval process is initiated.

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. All APIs that are assigned to the *DevTeam* are now assigned to *Team2* and *Team3*.

Team Support Considerations

This section lists the impact of the **Team support** feature on other API Gateway features.

As stated in previous sections, users can view only the assets that are assigned to their teams. Also, their accessibility depends on the functional privileges assigned to their team. Though this appears to be a simple behavior, there are some scenarios where it gets a little exceptional. Hence, as an administrator, read through the following scenarios carefully before you create team and assign functional privileges:

Visibility and Accessibility of Assets

You can use the **Teams support** feature to restrict the visibility and access of APIs, applications, packages, and plans. However, there are some scenarios where users from unassigned teams can view or access these assets. Also, note that the **Teams support** feature does not restrict the visibility and access of aliases, global policies, scopes, users, groups, and teams.

Scenarios that you must consider when using Team support

- When an API assigned to a team is associated with applications that are assigned to some other teams, then the team members, to whom the API is assigned can view the applications and remove them if required.

For example, consider two teams *Team_A* and *Team_B*. The API *API_1* and *Application_1* are assigned to *Team_A* and the API *API_2* and *Application_2* are assigned to *Team_B*. If you associate *Application_1* and *Application_2* with *API_1*, then the *Team_A* members can view *Application_2* and remove the application from *API_1*, if required.

- Users can view assets other than APIs, applications, packages, and plans based on their functional privileges. So, the **Team support** feature does not have an impact on assets like aliases, global policies, scopes, users, groups, and teams.

For example, consider two APIs, *API_1* and *API_2* assigned to two different teams; *API_1* assigned to *Team_A* and *API_2* assigned to *Team_B*. Consider a global policy applied to both APIs and an endpoint alias used in both APIs. In this case, users from both teams can view the global policy and the endpoint alias used by these APIs and there is no way to restrict this behavior.

The following table helps you to quickly recall the points discussed earlier in this topic:

Asset type	Visibility	Accessibility
APIs	Users can view the names, details, and versions of the APIs associated with their teams.	Users can access APIs based on functional privileges assigned to their teams.
	Users can view the names of applications that are associated with the APIs assigned to their teams.	Users can remove the applications that are visible (from the API).

Asset type	Visibility	Accessibility
Applications	Users can view the names, descriptions, versions of the applications associated with their teams.	Users can access applications based on functional privileges assigned to their teams.
	Users can view the names of APIs that are associated with the applications assigned to their teams.	Users can remove the APIs that are visible (from the application).
Packages and plans.	Users can view the names and details of the packages and plans assigned to their teams.	Users can access packages and plans based on functional privileges assigned to their teams.
Global search field	When users search for an asset using a keyword, the search returns only the assets that are assigned to your teams.	Users can access assets based on functional privileges assigned to their teams.
Aliases, Global Policies, Users, Groups, Teams, and Scopes	All users can view these assets, even if they have read-only access to API Gateway.	User can access the assets based on the privilege assigned to the individual users, irrespective of their teams.

Importing and Exporting Teams and Assets

Team members can import or export assets if they are assigned with the required functional privileges.

- When assets are exported, the respective team details are exported along with the assets; members of the teams are not exported.
- When assets are imported, the respective team is created if it is not present already; members of the teams cannot be imported.
- When team is exported, the users and groups that are part of team can be selected for export if required.
- When team is imported, the users and groups are imported along with the team.
- An export archive of an API will include the details of all associated applications (if this option was selected when exporting) - "visible" and "invisible"

An export archive of an Application will always include the details of all associated APIs - "visible" and "invisible"

Promotion management

Users with **Manage promotions** privilege can see all associated APIs/Applications (visible and invisible) on the Assets and dependencies page for the Applications/APIs selected on the Search page of the Promotion Management wizard

By switching between the Search page and the Assets and dependencies page using the Back and Next buttons, they can even drill down and recursively see all dependent assets (visible and invisible)

Members of a team can promote assets from one source stage to one or more target stages if they have the required functional privileges.

- When assets (all except teams) are promoted, they are promoted along with their team details; users are not promoted.
- When teams are promoted, the users and groups of teams can be promoted if required.

Scope mapping

Users with **Manage scope mapping** privilege can see all APIs (including APIs otherwise invisible to them) on the scope mapping configuration page. They can even add them to a scope mapping, but they cannot remove them again

Scope mappings are visible to all users

- API Gateway displays the names, descriptions, and versions for all associated APIs.
- The API entries are not linked with the respective API details pages

Note that there will be a scope mapping for every API if pg_oauth2_createDefaultScopes is set to true, so every API is listed in the list of scope mappings - visible to all users

Global policies for APIs

Global policies are visible to all users

- API Gateway displays names, descriptions, and versions for all associated APIs (by the policy's filter conditions)
- Only the entries for the visible APIs are linked with their API details pages

API mashup

Users can add APIs owned by you or assigned to your Team, but in an existing API Mashup, you can see the names, versions, selected resources and methods of all configured APIs - even those invisible to you.

API versioning

On the API details page (for a visible API version), the API Gateway displays all version numbers in the version drop-down list - for versions visible and invisible to the current user

When the user selects an invisible version, API Gateway displays an error message.

Scenarios where you cannot use Teams support

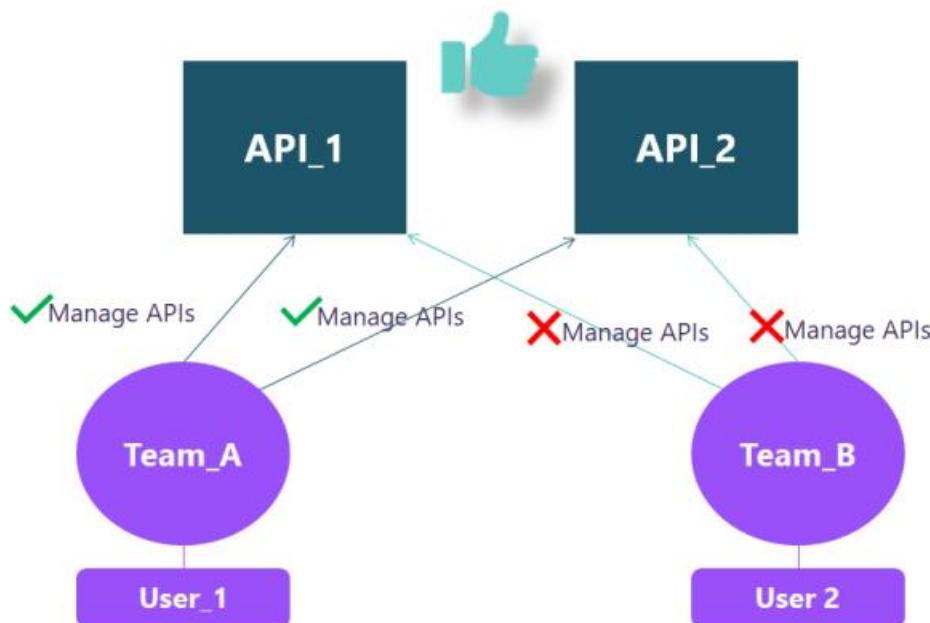
This section explains the use cases that cannot be achieved using the **Team support** feature.

Scenario 1

The functional privileges assigned to a team applies across all assets assigned to the team. You cannot assign different functional privileges to access different assets. That is, if you assign the **Manage APIs** functional privilege to a team, then the team members can perform all API-related transactions with the APIs assigned to the team. Similarly, if you assign the **Manage applications** privilege to a team, then all members of the team can manage the applications assigned to them.

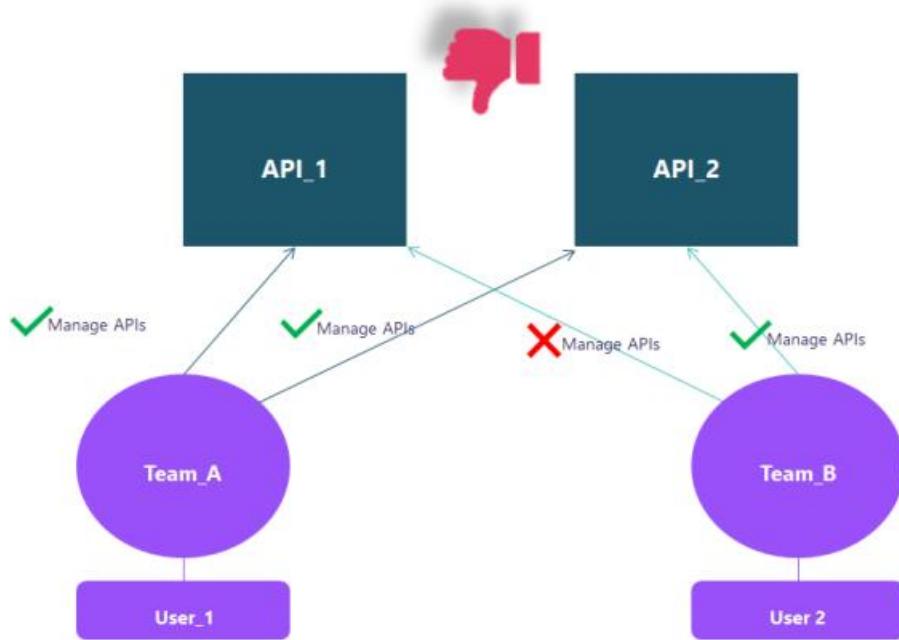
For example, if you assign the **Manage APIs** privilege to *Team_A* and assign two APIs *API_1* and *API_2*, then all users of the *Team_A* can manage both the APIs. If you do not assign the **Manage APIs** privilege to *Team_B* and assign two APIs *API_1* and *API_2*, then the *Team_B* members can only view the APIs. They cannot manage them.

This image describes the flow that you can achieve:



In the same example, you cannot allow *Team_B* members to view *API_1* and to manage *API_2*. You cannot assign different access level for different assets.

This image describes the flow that you cannot achieve:

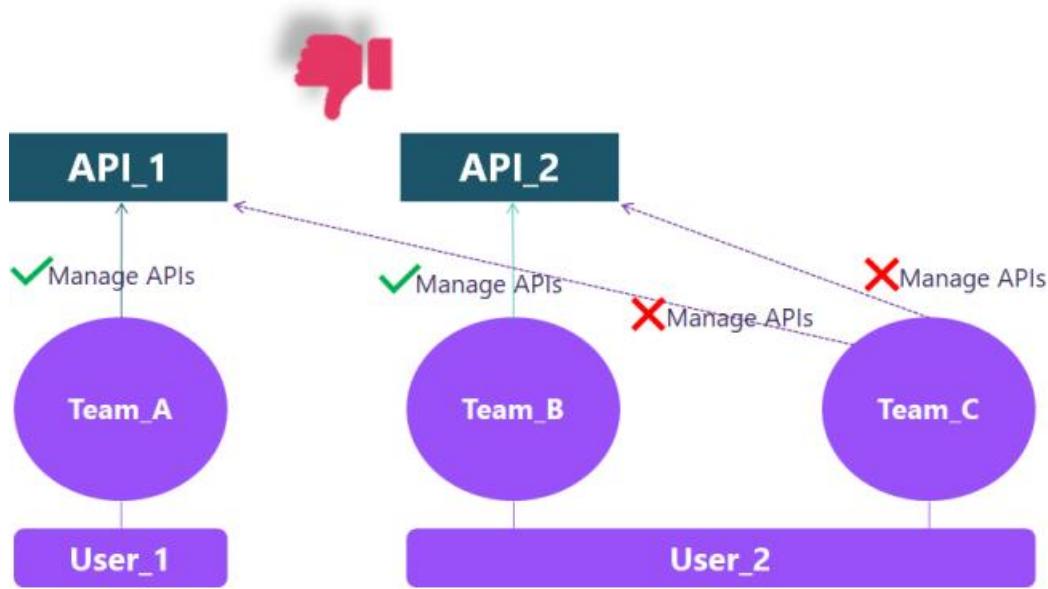


Scenario 2

The functional privileges assigned to a team applies across all users towards all assets assigned to the team. You cannot assign different privileges to different assets. That is, if a user has certain functional privilege through one of their teams, and when the user is assigned to another team that does not have the particular functional privilege, the user will still have the functional privilege assigned through the first team.

Consider *User_2* is a part of *Team_B* that has the **Manage APIs** privilege assigned with *API_2*. In this case, *User_2* can manage *API_2*. At the same time, if *User_2* is assigned to *Team_C* that does not have the **Manage APIs** privilege. If *API_2* is assigned to *Team_C*, then *User_2* can still manage *API_2*.

This image explains the flow that cannot be achieved:



Setting Password Restrictions

For security purposes, API Gateway places length and character restrictions on passwords for administrator and non-administrator users.

➤ To set password restrictions

1. Expand the menu options icon , in the title bar, and select **User management**.
2. Click **Account settings > Password restrictions**.
3. Provide the following information to set the required password restrictions.

Field	Description
Enable password change	Specifies whether users are allowed to change their passwords. This is selected by default.
Password enforcement mode	Specifies whether Administrator users are allowed to choose passwords that are not impacted by the password restriction settings. When this property is set to Strict , API Gateway enforces the password restrictions. When set to Lax , the password restrictions are not enforced.

Field	Description
Minimum password length	Specifies the minimum number of characters (alphabetic characters, digits, and special characters combined) the password must contain. The default value is 8.
Maximum password length	Specifies the maximum number of characters (alphabetic characters, digits, and special characters combined) the password must contain. Maximum number of characters that a password can have is 128. The default value is 64.
Minimum number of uppercase characters	Specifies the minimum number of uppercase alphabetic characters the password must contain. The default value is 0.
Minimum number of lowercase characters	Specifies the minimum number of lowercase alphabetic characters the password must contain. The default value is 0.
Minimum number of digits	Specifies the minimum number of digits the password must contain. The default value is 0.
Minimum number of special characters (neither alphabetic nor digits)	Specifies the minimum number of special characters, such as asterisk (*), period (.), and question mark (?) the password must contain. <p>Note: The use of special characters is regulated by the following restrictions:</p> <ul style="list-style-type: none"> ■ A password cannot begin with an asterisk (*)�. ■ Passwords cannot contain quotation marks ("), backslashes (\), ampersands (&), or less-than signs (<). Use the watt.server.illegalUserChars configuration property to restrict the use of additional characters. The default value is 0.
Maximum number of identical characters in a row	Specifies the maximum number of identical characters in a row a password can contain. The default value is 3.
Number of old passwords to remember (per user)	Specifies the maximum number of previously set passwords that API Gateway saves for a user (excluding the current password).

Field	Description
	You cannot choose a password that matches any of the stored passwords. Maximum number of saved passwords is 12.
	The default value is 0.

- Click **Save**.

Setting Password Expiry Restrictions

For security purposes, API Gateway allows administrators to set password expiration requirements on passwords for administrator and non-administrator users. An administrator user receives a reminder email to reset the password before certain number of days, as specified in the Password expiration settings page.

➤ To set password expiry restrictions

- Expand the menu options icon  in the title bar, and select **User management**.
- Click **Account settings > Password expiry settings**.
- Provide the following information to set the required password expiry restrictions.

Field	Description
Enabled	<p>Specifies whether to enable the password expiry settings.</p> <p>This option is disabled by default. Select Enabled to enable the password expiry settings.</p>
Expiration interval (days)	<p>Specifies the number of days after which a password expires, if not changed.</p> <p>The value should be a non-zero integer.</p> <p>The default value is 90.</p> <p>Note: Upon save, when this option is enabled, any password that is set before the expiration interval are considered expired and have to be reset. For example, if you changed your password 10 days ago and now, the Administrator changes the Expiration interval to 5 days, then your password has expired and needs to be reset.</p>

Field	Description
Days prior to password expiry for email reminders	<p>Specifies the number of days prior to password expiry that API Gateway starts sending the reminder emails for password reset. The emails are sent daily until the user either updates the password or changes the expiration interval.</p> <p>The default value is 3.</p> <p>Set the value to 0 to prevent API Gateway from sending the reminder emails for soon to expire passwords.</p> <p>Note: API Gateway uses the SMTP server and port details specified in Integration Server in the Email Notification section on Resource Settings screen (Settings > Resources).</p>
Expiration notice email addresses	<p>Specifies the list of email addresses to which API Gateway sends an email notification informing that the user password is about to expire or has already expired.</p> <p>You can add multiple email addresses by clicking +Add.</p>
Applies to users	<p>Specifies the users to whom these settings apply.</p> <p>You can add multiple users by clicking +Add.</p>

4. Click **Save**.

Configuring Account Locking Settings

For security purposes, it is important to lock an user account when the user fails to provide the correct password after a specified number of failed login attempts to API Gateway. A locked user account remains locked for a specific period of time, after which the account gets unlocked. API Gateway allows administrators to configure the account locking settings for administrator and non-administrator users. You can set the values for number of attempts by a user before locking the account and also the duration of the lock interval.

➤ To configure account locking settings

1. Expand the menu options icon  in the title bar, and select **User management**.
2. Click **Account settings > Account locking settings**.
3. Provide the following information to configure the required account locking settings.

Field	Description
Enabled	Specifies whether to enable the account locking settings.
	This option is disabled by default. Select Enabled to enable the account locking settings.
Maximum login attempts	Specifies the number of attempts in the specified time interval (minutes, hours, or days) to provide the correct password before locking the account.
	The default value is None.
Lockout duration	Specifies the duration (minutes, hours, or days) for which the account remains locked.
	The default value is None.
Apply account locking policy to	Specifies the list of users to whom the account locking settings apply.
	Specify one of the following:
	<ul style="list-style-type: none"> ■ All users. Indicates the account locking rules apply to all user accounts. ■ All users except predefined users. Indicates that account locking rules apply to all user accounts except the predefined user accounts (Administrator).

4. Click **Save**.

Unlocking User Accounts

API Gateway unlocks a user account after the specified locked duration. However, as an Administrator, you can manually unlock user accounts within the lockout duration configured.

➤ To unlock locked user accounts

1. Expand the menu options icon  in the title bar, and select **User management**.
 2. Click **Account settings > Account locking settings**.
- The Locked users section displays all the locked users.
3. From the list of locked user accounts, click  to unlock the user account.
 4. Click **Save**.

Restricting User Accounts

API Gateway provides an option to restrict the user accounts, who are part only of the *Default* team and not any other team. You can enable this option to restrict those user accounts from logging into API Gateway.

➤ To restrict user accounts who belongs only to the default team

1. Expand the menu options icon  in the title bar, and select **User management**.
2. Click **Account settings > Account locking settings**.
3. In the **Login Restrictions** section, select **Restrict login for users who are not member of any team other than Default** if you want to restrict the user accounts associated to the *Default* team and not any other team.
4. Click **Save**.

Configuring API Gateway to Use LDAP

If your site uses Lightweight Directory Access Protocol (LDAP) for user and group information, you can configure API Gateway to obtain user and group information from the external directory.

LDAP protocols are designed to facilitate sharing information about resources on a network. Typically, they are used to store profile information (login ID, password, and so on.). You can also use them to store additional information. API Gateway uses LDAP for performing external authentication.

Using your existing LDAP information allows you to take advantage of a central repository of user and group information. System administrators can add and remove users from the central location. Users do not need to remember a separate password for webMethods applications; they can use the same user names and passwords that they use for other applications. Remember to use your LDAP tools to administer users or groups stored in an external directory.

To configure the server to use LDAP, you must:

- Instruct API Gateway to use the LDAP protocol. API Gateway supports LDAP v3.
- Define one or more configured LDAP servers that API Gateway is to use for these users.
- Set the `watt.server.ssl.trustStoreAlias` property, if an LDAP provider is SSL-enabled, to point to the `trusstore` alias that contains the certificates required to establish a secure connection with the LDAP server.

➤ To specify LDAP as the external provider

1. Expand the menu options icon  in the title bar, and select **User management**.
2. Click **LDAP configuration**.
3. Under Provider select **LDAP**.
4. Provide the following information:

Field	Description
Cache size (number of users)	<p>Specifies the maximum number of LDAP users API Gateway can keep in memory in the user cache.</p> <p>The default value is 10.</p> <p>Once the limit is reached, API Gateway selects users for removal from the cache based on how long they have been idle. As a result, activity can extend the time a user remains in the cache.</p>
Credential time-to-live (minutes)	<p>Specifies the number of minutes an LDAP user's credentials (userid and password) can remain in the credential cache before being purged.</p> <p>The default is 60 minutes.</p> <p>When a user first attempts to log in, API Gateway creates a user object and checks the user's credentials against the LDAP directory. API Gateway stores the credentials so that subsequent requests to authenticate are made against the cached credentials, not the LDAP directory.</p>

5. Click **Save**.

Managing LDAP Directories

You can manage the LDAP directories in the LDAP directories section. You can view all the LDAP directories configured listed in a table here with their directory URL details. You can create, update, delete and prioritize the LDAP directories here.

➤ To add an LDAP directory

1. Expand the menu options icon  in the title bar, and select **User management**.
2. Click **LDAP configuration**.
3. In the LDAP directories section, click **Add LDAP directory**.

4. Provide the following information to add an LDAP directory.

Field	Description
Directory URL	<p>Specifies the complete URL of the LDAP server.</p> <p>The URL has the format <i>protocol://hostname:portnumber</i> where</p> <ul style="list-style-type: none"> ■ The <i>protocol</i> is LDAP for standard connections or LDAPS for secure connections ■ The <i>host</i> is the host name or IP address of the LDAP server. The <i>port</i> is the port on which the server is running. The port is optional. If omitted, the port defaults to 389 for LDAP, or 636 for LDAPS <p>For example, specifying the URL <code>ldaps:// ldaperv1:700</code> would create a secure connection to the LDAP server running on the non-standard port 700 on the host called <code>ldaperv1</code>.</p> <p>If you specify <code>ldaps</code>, API Gateway attempts to make a secure connection to the directory server using an SSL socket. If the directory server is configured to use SSL, it has a server certificate in place to identify itself to clients. This certificate must be signed by an authority to prove its validity that is, the server certificate is signed by a CA). By default, API Gateway only trusts certificates signed by a signing authority whose CA certificate is in the API Gateway's trusted CAs directory.</p>
Principal	<p>Specifies the user ID API Gateway should supply to connect to the LDAP server.</p> <p>For example, <code>o=webm.com</code> or <code>dc=webm,dc=com</code>.</p> <p>This user should not be the Administrator account, but a user that has permission to query groups and group membership. If your LDAP server allows anonymous access, leave this field blank.</p>
Credentials	<p>Specifies the password API Gateway should supply to connect to the LDAP server, that is, the Principal's password.</p>
Connection timeout (seconds)	<p>Specifies the number of seconds API Gateway waits while trying to connect to the LDAP server.</p> <p>After this time has passed, API Gateway tries for the next configured LDAP server on the list.</p> <p>The default is 5 seconds.</p>
Minimum connection pool size	<p>Specifies the minimum number of connections allowed in the pool that API Gateway maintains for connecting to the LDAP server.</p>

Field	Description
	When API Gateway starts, the connection pool initially contains this minimum number of connections. API Gateway adds connections to the pool as needed until it reaches the maximum allowed, which is specified in the Maximum Connection Pool field.
	The default value is 0.
Maximum connection pool size	Specifies the maximum number of connections allowed in the pool that API Gateway maintains for connecting to the LDAP server.
	When API Gateway starts, the connection pool initially contains the minimum number of connections as specified in the Minimum Connection Pool field. API Gateway adds connections to the pool as needed until it reaches the maximum allowed.
	The default value is 10.
	Distinguished Name (DN) method. Specifies the directory name to be built on selecting any of the following criteria.
Synthesize DN	Builds a distinguished name by adding a prefix and suffix to the user name. The Synthesize DN method can be faster than the Query DN method because it does not perform a query against the LDAP directory. However, if your LDAP system does not contain all users in a single flat structure, use the Query DN method instead.
	DN prefix
	A string that specifies the beginning of a DN you want to pass to the LDAP server.
	DN suffix
	A string that specifies the end of a DN you want to pass to the LDAP server.
	For example, if the prefix is <code>cn=</code> and the suffix is <code>,ou=Users</code> and a user logs in specifying <code>bob</code> , then API Gateway builds the DN <code>cn=bob,ou=Users</code> and sends it to the LDAP server for authentication.
	Note:
	Be sure to specify all the characters required to form a proper DN. For instance, if you omit the comma from the suffix above, that is, you specify <code>ou=Users</code> instead of <code>,ou=Users</code> , API Gateway builds an invalid DN <code>cn=bobou=Users</code> .

Field	Description
Query DN	<p>Builds a query that searches a specified root directory for the user. Use this method instead of the Synthesize DN method if your LDAP directory has a complex structure.</p> <p>UID property</p> <p>A property that identifies an LDAP userid, such as "cn" or "uid".</p> <p>User root DN</p> <p>Provide the full distinguished name. For example, if you specify ou=users,dc=webMethods,dc=com, API Gateway issues a query that starts searching in the root directory ou=users for a common name that matches the name the user has logged in with.</p>
User email attribute	<p>Specifies the name of the email attribute in the LDAP directory. The email ID of the API Gateway's user object is mapped to the value specified in this field .</p> <p>This value depends on the schema of the LDAP directory.</p>
Default group	<p>Specifies the API Gateway group with which the user is associated. The user is allowed to access APIs that members of this API Gateway group can access. This access is controlled by the ACLs with which the group is associated.</p> <p>If you also specify a value in the Group member attribute field, the user has the same access as members of the API Gateway group and members of LDAP groups that have been mapped to an ACL.</p>
Note:	<p>If you do not want to select a default group, you can select <None> from the options provided.</p>
Group member attribute	<p>Specifies the name of the attribute in a group's directory entry that identifies each member of the group. This value is usually <code>member</code> or <code>uniqueMember</code>, but can vary depending on the schema of the LDAP directory. API Gateway uses this information during ACL checking to see if the user attempting to log in belongs to an LDAP group that has been mapped to an ACL. If no value is specified here, API Gateway does not check for membership in an LDAP group. As a result, the user's ability to</p>

Field	Description
	access API Gateway services is controlled by the API Gateway group specified in the Default group field.
Group ID property	Specifies a property that identifies an LDAP group, such as CN.
Group root DN	Specifies the full distinguished name. For example, if you specify ou=groups,webMethods,dc=com, API Gateway issues a query that displays all the LDAP groups.

Note:

You must specify values in the **Group ID property** field and **Group root DN** fields.

- Click **Save**.

The LDAP directory is added and listed in a table under the LDAP directories section.

Note:

- If you define multiple LDAP servers, API Gateway searches the LDAP directories in the order in which they are displayed in the **User Management > LDAP directories** section. If API Gateway does not find the user in the first LDAP directory, it searches in order through the list.
- If the connection between API Gateway and the LDAP server drops intermittently, and you notice the following exception in the Trace logs, connect to the Global Catalog port (3268/3269) on the LDAP server, instead of using the standard LDAP port (389). For example,
ldap://hostname:3268

```
PartialResultException in the trace logs : [ISS.0002.0000T]
[LDAPv2] javax.naming.PartialResultException [Root exception is
javax.naming.CommunicationException:
[Root exception is java.net.SocketTimeoutException: connect timed out]]
```

- If the connection issues continue despite using the Global Catalog port (3268/3269), it may be due to the following errors:
 - Connection timeout error
 - Communication error
 - Resource shortage error
 - An orphaned domain acts as the Global Catalog

Set appropriate values for the `watt.server.ldap.retryCount` and `watt.server.ldap.retryWait` parameters to restore the connection in case of transient errors.

Next Steps:

You can perform the following operations in the LDAP directories section where the configured LDAP directories are listed.

- You can update an LDAP directory by clicking on the **LDAP directory URL** field in the table, modify the details as required and save the changes.

- You can prioritize the LDAP directory as required by clicking in the **Prioritize** column for the corresponding LDAP directory.
- You can delete an LDAP directory by clicking the  icon in the **Delete** column for the corresponding LDAP directory.

User Settings and Preferences

You can set the personal preferences and settings to control how you interact with API Gateway. You can specify custom values in the User settings page that are used instead of default values set by an Administrator for your user account.

The User settings page displays a summary of your current user preferences. In the User settings page, you can do the following:

- Change your account settings.
- Change your password.
- Change your display language on the user interface.
- View your roles and permissions.

Based on the logged in user, the User settings page is displayed in one of the following ways:

- As an LDAP user: Displays the roles and permissions that are assigned to your user account in the Roles and permissions section.
- As an API Gateway user: Displays the roles and permissions that are assigned to your user account in the Roles and permissions section.

Changing Your Account Settings

User account settings include the user name and email address. These settings are attributes of local users who are validated against credentials stored in API Gateway. If API Gateway uses an external authentication mechanism, such as an LDAP, you must change the equivalent settings in the external LDAP system.

When the administrator creates a user account and an email address for the user, that email address is set as the default email address for the account. If you have multiple email addresses, you can set up additional email addresses in the User settings page.

➤ To change your user name and email address

1. Expand the menu options icon , in the title bar, and select **Profile**.

The User settings page appears with your user preferences.

2. Provide the following information in the  Account settings section.

Field	Description
First name	Type your first name. First name is case sensitive. A first name can contain letters, numbers, or a combination of all. You can also use special characters: . (dot), _ (underscore), and @ (at). Other special characters and spaces are not allowed.
Last name	Type your last name. Last name is case sensitive. A last name can contain letters, numbers, or a combination of all. You can also use special characters: . (dot), _ (underscore), and @ (at). Other special characters and spaces are not allowed.
Email addresses	Type a valid email address. If the email address is invalid, API Gateway prompts you with an error message. You can add multiple email addresses by clicking  .

3. Click **Save**.

The changes are applied immediately.

Changing Your Password

You can change the password that you use to log on to API Gateway.

➤ To change your password

1. Expand the menu options icon  in the title bar, and select **Profile**.

The User settings page appears with your user preferences.

2. Provide the following information in the  Change password section.

Field	Description
Current password	Type the current password.
New password	Type a new password. Passwords are case sensitive. Your password must meet the complexity requirements configured in Integration Server

Field	Description
	(in the Integration Server Administrator, go to Security > User Management > Password Security Settings).
Confirm new password	Retype the new password to confirm.

3. Click **Save**.

The password is changed immediately.

Changing Your Display Language

You can select the language for the user interface of API Gateway.

API Gateway displays the user interface in English (en) by default. If you want API Gateway to use a different language other than English, you must install the intended language pack from the Software AG Installer.

➤ To change your display language

1. Expand the menu options icon  in the title bar, and select **Profile**.

The User settings page appears with your user preferences.

2. In the  Display settings section, select the language you would like to use for the user interface.

Note:

If the language you want to use is not available by default in the **Language** list box, you need to install the intended language pack from the Software AG Installer.

3. Click **Save**.
4. Log out and log on to API Gateway for the change to take effect.

Viewing Your Roles and Permissions

The User settings page displays a list of all the roles and permissions that are assigned to your user account. The roles and permissions assigned to your user account are defined through user groups and corresponding teams (see “[Users, Groups, and Teams](#)” on page 425 for more information).

➤ To view your roles and permissions

1. Expand the menu options icon  in the title bar.
2. Select **Profile** from the menu options.

In the  Roles and permissions section, you can view the list of roles and permissions that are assigned to you.

Destination Configuration

API Gateway can publish events and performance metrics data to the configured destinations. Event type data provides information about activities or conditions that occur on API Gateway. The performance data provides information on average response time, total request count, fault count, and so on, for the APIs that it hosts.

You must have the API Gateway's manage destination configurations functional privilege assigned to configure the following destinations to which the event types and performance metrics data is published:

- API Gateway
- API Portal
- Transaction logger
- CentraSite
- Database
- Digital Events
- Elasticsearch
- Email
- SNMP
- Custom destinations

Note:

In addition to this list of destinations, you can configure custom destination to publish data to different components. For details on custom destinations, see “Custom Destination” on page 496.

Configuring Events for API Gateway Destination

You have to configure the API Gateway destination so that the events and performance metrics data can be published to API Gateway. By default, error events, lifecycle events, policy violation event, and performance data are published to API Gateway.

➤ **To configure events for API Gateway destination**

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Destinations**.
3. Select **API Gateway** to configure the event types for this destination.
4. In **Event types**, select the type of events to publish to API Gateway.

The available event types are:

- **Error**. Occurs each time an API invocation results in an error.
 - **Lifecycle**. Occurs each time API Gateway is started or shut down.
 - **Policy violation**. Occurs each time an API invocation violates the policy enforcement that was set for the API.
5. Select **Report performance data** to publish performance metrics data.
 6. In the **Publish interval** box, enter a time interval (in minutes) how often API Gateway must publish performance metrics. Enter a value from 1 through 60. The default is 60 minutes.
 7. In the **Audit log data** section, select the required management areas for which the audit logs should be recorded in the API Gateway destination.

Audit logs provide a record of system transactions, events, and occurrences in API Gateway. You can configure audit logging to show the following events:

- **API management**
- **Application management**
- **Team management**
- **Group management**
- **Package management**
- **Promotion management**
- **Approval management**
- **Alias management**
- **Analytics management**
- **Policy management**
- **Plan management**
- **User management**

Note:

By default, audit logging is enabled for all of the above-listed management areas in the API Gateway destination.

8. Click **Save**.

Troubleshooting Tips: API Gateway Destination

I see that the API Gateway transactions are not logged in API Data Store when there is a medium to high load that is 70 to 400 Transactions per second (TPS)

If you have configured an external API Date Store destination, ensure that the configuration is right. This problem might occur when the API Data Store configurations are not set properly to handle the load.

Resolution:

Set the following configurations to the properties in config.properties file located at `SAG_Install_Directory\IntegrationServer\instances\IS_Instance_Name\packages\WmAPIGateway\config\resources\elasticsearch\config.properties` to meet the following throughput values: Up to 2000 TPS, Up to 500 GB storage utilization, and less than <500 ms native service latency:

```
pg.gateway.elasticsearch.hosts = Elasticsearch Service endpoint, that is localhost:9240
pg.gateway.elasticsearch.http.connectionTimeout = 10000
pg.gateway.elasticsearch.http.socketTimeout = 30000
pg.gateway.elasticsearch.sniff.enable = false
pg.gateway.elasticsearch.http.maxRetryTimeout = 100000
```

You can verify the transactional events count using
http://<hostname>:<es_port>/testindex/transactionalEvents/count

Note:

You can modify the configurations according to your business requirements.

For more information, see Hardware and Product Configurations in *webMethods API Gateway Administration*.

I see that I am not able to parameterize custom indices for transaction events in API Gateway

Custom indices can be configured for transaction events in API Gateway under **Destinations > Elasticsearch > Configuration page**. But parameterizing custom indices is not supported in API Gateway. For example, appending the current date to the index name. Consider an index name `txaData` and it must be appended with the current date, that is `txaData_12_08_2020` and rolled up daily.

Resolution:

You can achieve this use case using the Elasticsearch aliases and rollover APIs of Elasticsearch.

A sample use case is as follows:

Create an alias in the Elasticsearch server, for example txnData and an index rollover like txnData_12_08_2020, txnData_12_08_2020. The alias txnData should point to all the indices txnData_12_08_2020, txnData_12_08_2020. One of the indices in the list is write-enabled and that are the latest and rest of the indices is read-only.

API Gateway sends the events to txnData alias and that alias defines the index that it should write the data to, based on the date.

Configuring Developer Portal Destination

You have to configure Developer Portal as a destination to establish communication channel between API Gateway and Developer Portal to exchange data.

➤ To configure Developer Portal destination

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Destinations**.
3. Select **API Portal > Configuration** to configure Developer Portal as the destination.
4. Provide the following information in the Basic information section:

Field	Description
Name	Name of the Developer Portal being configured. This name should be unique as it is referenced in the list of providers within Developer Portal. This uniqueness is important when multiple API Gateways are publishing APIs to the same Developer Portal.
Version	Version of the portal being configured.

5. Provide the following information in the Portal configuration section for API Gateway to send data to Developer Portal:

Field	Description
Base URL	URL of the Developer Portal instance in the format <code>http://host:port</code>
Tenant	The tenant details of API Portal. By default, default tenant is considered if the tenant information is not provided.
Username	Username credential to access Developer Portal instance.
Password	Password credential to access Developer Portal instance.

6. Provide the following information in the Gateway configuration section for Developer Portal to create applications, request or revoke access tokens, subscriptions, and so on:

Field	Description
Base URL	URL of the API Gateway instance. This is pre-populated. Note: When a load balancer URL is updated in API Gateway, the API Gateway base URL is updated to same in this field.
Username	Username credential of the API Gateway Administrator to access API Gateway instance.
Password	Password credential to access API Gateway instance.
Stage name	<i>Optional.</i> The stage name of one of the stages in an API Gateway instance. The stage name included here is displayed in the API details, API tryout, Manage APIs, and Application details pages in the Developer Portal instance.

7. Click **Test** to ensure the connection with the given Developer Portal details is established.

The connection with the given server is tested and a success message appears.

8. Click **Publish**.

This establishes a communication channel between API Gateway and Developer Portal.

Troubleshooting Tips: Developer Portal Destination

I see that when API Gateway and Developer Portal are deployed on different machines, configuring Developer Portal as the destination fails.

When API Gateway and Developer Portal are deployed on different machines, configuring Developer Portal as the destination fails with the following error message when I click the **Test** button, "*The input url https://sawld01.cmcdev.be:18102/portal/rest/v1/providers?tenantId=default is not allowed as the host is in denied list. Contact administrator*".

Resolution:

Using the REST API, update the Developer Portal's IP address in the API Gateway's *whiteListedIPs* /rest/apigateway/configurations/whiteListingIPs

Configuring Events for API Portal Destination

Pre-requisites:

You have to configure API Portal to communicate with API Gateway before you select the events and metrics for publishing to API Portal.

You have to configure the API Portal destination so that the events and performance metrics data can be published to API Portal.

➤ To configure events for API Portal destination

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Destinations**.
3. Select **API Portal > Events** to configure the event types for this destination.
4. In **Event types**, select the type of events that you want API Gateway to publish to API Portal.

The available event types are:

- **Error**: Occurs each time an API invocation results in an error.
- **Lifecycle**: Occurs each time API Gateway is started or shut down.
- **Policy violation**: Occurs each time an API invocation violates the policy enforcement that was set for the API.

5. Select **Report performance data** to publish performance metrics data.
6. In the **Publish interval** box, enter a time interval (in minutes) how often API Gateway must publish performance metrics. Enter a value from 1 through 60. The default is 60 minutes.
7. Click **Save**.

Post-requisites:

After performing the event configurations, select **API Portal** as a **Destination** in the Policy Properties page for each policy, to receive the transaction event logs for the assigned policies.

Configuring API Control Plane

API Control Plane is a centralized control and monitoring system that oversees and manages the entire API management landscape deployed across multiple regions, including cloud and on-premise environments. Connecting API Gateway to API Control Plane enables you to control and manage the performance of API Gateway effectively through API Control Plane.

To connect API Gateway to API Control Plane, it is essential to configure API Control Plane agent within API Gateway to establish a communication channel with API Control Plane. The agent is responsible for establishing and maintaining communication with API Control Plane.

You can configure API Control Plane agent through:

- External configurations

To set up the API Control Plane agent using external configurations, ensure you are using API Gateway version *10.15.0 Fix 10* or later.

- API Gateway UI

To configure the API Control Plane agent through the API Gateway UI, ensure you are using API Gateway version *10.15.0 Fix 12* or later.

External configurations

Configure API Control Plane agent through external configurations in one of the following ways: Environment variables or JAVA properties or YAML config file. For more information about how to create external configurations and what configurations are required for agent, see **Deployment configuration** under Administering API Control Plane section in *webMethods API Control Plane documentation*.

API Gateway UI

This section details API Control Plane agent configuration using API Gateway UI.

Note:

If you have previously configured API Control Plane agent using external configurations and subsequently logged on to API Gateway, the API Control Plane communication screen displays as disabled and is not editable. In such case, if you want to re-configure API Control Plane agent using API Gateway UI, you must remove external configurations (Environment variables or Java properties or YAML config file) used for the previous connection and restart API Gateway. Upon restart, API Gateway establishes a connection with API Control Plane using previously specified external configurations from Elasticsearch since the configurations that are once specified through external configurations are retained in Elasticsearch to populate your desired values in the UI. This allows you to configure the agent through the UI.

Pre-requisite:

You must have API Gateway's manage destination configurations functional privilege assigned to configure API Control Plane.

➤ **To configure API Control Plane agent through API Gateway UI**

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Destinations**.
3. Select **API Control Plane > Configuration** to configure API Control Plane agent.
4. Provide the following information in the API Gateway Basic information section:

Field	Description
Runtime name*	<p>Name of the runtime (API Gateway instance). This field defines how you want to identify your API Gateway instance in API Control Plane. In the context of API Control Plane, a <i>runtime</i> refers to any product deployment within the API Management suite such as API Gateway, Developer Portal, and Microgateway.</p>
Description	<i>Optional.</i> Specify a description for the runtime.
Tags	<p><i>Optional.</i> Tag of the runtime. Tags are used to categorize the runtimes. Example: test, local, dev</p> <p>Note: A tag must not contain whitespaces.</p>

5. Provide the following information in the API Gateway deployment section:

Field	Description
Deployment type	<p><i>Optional.</i> API Gateway deployment type. Permitted values are as follows:</p> <ul style="list-style-type: none"> ■ ON_PREMISE ■ SAG_SAAS ■ PRIVATE_CLOUD
Region	<i>Optional.</i> Region name where API Gateway is deployed.
Location	<i>Optional.</i> Country name where API Gateway is deployed.
Host	<p><i>Optional.</i> Base URL of API Gateway in the format, <code>http://host:port</code> Example: <code>http://localhost:9072</code></p>
Capacity	<p><i>Optional.</i> The approximate estimate of the throughput that API Gateway can handle for the specified duration. You can configure the capacity value with any non-negative integer.</p>
Capacity units	<p><i>Optional.</i> Choose the unit of duration in which the capacity must be defined. Possible values are as follows:</p> <ul style="list-style-type: none"> ■ per second

Field	Description
	<ul style="list-style-type: none"> ■ per minute ■ per hour ■ per day ■ per week ■ per month ■ per year

6. Provide the following information in the API Control Plane configuration section:

Field	Description
Control Plane URL*	Base URL of API Control Plane. Example: https://localhost:8080
Username*	Username that is used to log on to API Control Plane. Note: Ensure that the user belongs to API platform provider user group.
Password*	Password of the corresponding user, which is used to log onto API Control Plane through basic authentication.

7. Provide the following information in the synchronization configuration section to synchronize the data from API Gateway to API Control Plane:

Field	Description
Heartbeat interval (in seconds)	The duration in seconds in which API Gateway must send its health status to API Control Plane. Default value: 60
Metrics synchronization interval (in seconds)	The duration in seconds in which API Gateway must send its metrics data to API Control Plane. Default value: 120
Asset synchronization interval (in seconds)	The duration in seconds in which API Gateway has to synchronize the changes made to the assets to API Control Plane. Default value: 180

Field	Description
Retry max attempts	The maximum number of retry attempts allowed for API Gateway to establish a connection with API Control Plane if it is lost. Default value: 5
Retry time interval (in seconds)	The duration in seconds in which API Gateway has to re-establish the connection with API Control Plane if it is lost. Default value: 60

8. Click **Test** to test the connection between API Gateway and API Control Plane.

The connection with API Control Plane is tested and a success message appears.

9. Enable the **Control Plane communication** toggle button to establish communication between API Gateway and API Control Plane.
10. Click **Save**.

Agent configuration is saved in Elasticsearch and a communication is established between API Gateway and API Control Plane.

Note:

When you re-configure and save the following configurations:

- Modifying runtime name field establishes a new communication channel, creating a new runtime in API Control Plane. The old runtime's status turns red, indicating that the connection is lost.
- Modifying any field except the runtime name stops the existing communication channel and re-establishes with the updated values. During this period, the runtime's status appears red, indicating that the connection is lost.

If the connection is lost, the Control Plane communication toggle button changes to a *disabled* state indicating that the connection is lost.

You can also manage API Control Plane agent configuration using REST APIs. For more details, see **API Gateway Administration** section in the *Developer's Guide*.

Configuring Transaction Logger Destination

You have to configure transaction logger as a destination to receive the API Gateway transaction event logs. The transaction log data is written to a file or a database based on the configurations. The events are sent to the log file for the Log Invocation policy.

Note:

Any modifications to the API Gateway transaction logger destination in Integration Server do not reflect in API Gateway UI. Hence, Software AG recommends that you do not configure or

modify API Gateway transaction logger destination through the Integration Server Administrator UI.

➤ To configure transaction logger destination

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Destinations**.
3. Select **Transaction logger**.

The default name of the transaction log, **API Gateway Transaction Logger** appears in the **Name** field.

4. Click the **Enable** activation toggle button to enable the logger to start writing the log entries to the database or the file.
5. Provide the following information:

Field	Description
Mode	<p>Specifies whether the logger is to write entries to the destination synchronously or asynchronously.</p> <ul style="list-style-type: none"> ■ Synchronous: In this mode, the logger writes entries directly to the destination. ■ Asynchronous: In this mode, the logger writes entries to a queue, then later writes the entries from the queue to the destination. Each logger has its own queue.
Guaranteed	<p>Provides data about guaranteed delivery transactions. You can use guaranteed delivery log entries to do the following:</p> <ul style="list-style-type: none"> ■ Track when transactions start and their current status. ■ See the names of guaranteed delivery processes that are running. ■ Track whether the processes completed successfully or failed. <p>The default value is <i>No</i>.</p>
Destination	<p>Specifies whether the logger is to write entries to a file or database.</p> <ul style="list-style-type: none"> ■ Database: If this option is selected, the logger writes entries to the database. <p>Note:</p>

Field	Description
	<p>You must ensure that in webMethods Integration Server, a pool alias is associated to the ISCoreAudit functional alias. For more information on pool alias, see <i>webMethods Integration Server Administrator's Guide</i>.</p>
	<ul style="list-style-type: none"> ■ File: If this option is selected, the logger writes entries to the file in <i>Integration Server_directory\instances\instance_name\logs\APIGateway</i> directory. The default value is file.
Maximum queue size	<p>Specifies the maximum number of entries the queue can hold. Specify numerals only (for example, do not include commas or periods).</p>
	<p>The default value is 100000 records.</p> <p>Choose a value that accommodates your system's average volume for log entries. If your logging volume has sudden spikes, the queue can usually catch up by writing the pending entries during lulls. If the queue size is reached, then any additional log entries would be lost. Ensure to check the availability of disk space or memory of API Gateway server before you change the default value.</p>
Maximum retries	<p>Specifies the maximum times the logger must retry writing the entry to the destination if the first attempt fails because of a transient error. A transient error is an error that arises from a temporary condition that might be resolved or corrected quickly, such as the unavailability of a resource due to network issues or failure to connect to a database. The default value is 3.</p>
Wait between retries	<p>Specifies the waiting time before the logger can reconnect and rewrite the entries to the destination in case of failure. The default value is 5 seconds.</p>

6. Click **Save** to save the specified transaction log configuration value.

Note:

For these modifications to take effect, you must restart the API Gateway server.

You can click **Cancel** to revert to the last saved changes or to abandon all the changes if the values are not saved.

Post-requisites:

After performing the events configurations, you must select **Audit Log** as a **Destination** in the Policies properties page for each policy, to receive the transaction event logs for the assigned policies.

Configuring CentraSite Destination

You have to configure CentraSite as a destination to establish a communication channel between API Gateway and CentraSite to exchange data.

Once an API Gateway asset is published from CentraSite to API Gateway, the CentraSite communication and SNMP configuration details automatically appear in the **CentraSite** destination of API Gateway. If the same API Gateway asset is unpublished from the API Gateway instance at a later time, the CentraSite communication and SNMP configuration details are automatically removed from the **CentraSite** destination.

Note:

If you have already configured CentraSite as a destination in API Gateway, then publishing another API Gateway asset from any CentraSite instance to API Gateway fails and displays an error. In API Gateway, only one CentraSite instance can be configured as a destination at a time. To reconfigure another CentraSite instance, you need to use the **Force reset CentraSite communication and SNMP details** option.

➤ To configure CentraSite destination

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Destinations**.
3. Select **CentraSite > Configuration** to configure the CentraSite communication and SNMP details.

The Communication section displays the following information:

Field	Description
Protocol	Specifies the communication protocol used to establish communication between API Gateway and CentraSite.
Hostname	Specifies the host name or IP address of the machine on which CentraSite Application Server Tier (CAST) is running.
UDDI port	Specifies the port on which CAST is listening. The default port number for CAST is 53307.
Username	Specifies the CentraSite user ID for authenticating CentraSite when API Gateway communicates with CentraSite. This implies the user ID of a user who has the CentraSite Administrator role or the API Gateway Administrator role in CentraSite.
Target name	Specifies the name of the API Gateway asset as defined in CentraSite.

The SNMP section displays the following information:

Field	Description
Transport	Specifies the wire transport protocol that is used by the SNMP Listener. Supported values are: TCP and UDP.
Hostname	Specifies the CentraSite host name or IP address to which the SNMP listener binds.
Port	Specifies the port to which the SNMP listener binds. The default port number for CentraSite's SNMP server is 8181.
Username	Specifies the SecurityName that is used by the SNMP Listener.
Authorization protocol	Specifies the authorization protocol that is used by the SNMP Listener for decoding the incoming trap. Supported values are: MD5 and SHA.
Privacy protocol	Specifies the privacy protocol that is used by the SNMP Listener for decoding the incoming trap. Supported values are: AES128, AES, AES192, AES256, 3DES, and DESEDE.

4. Select **Send SNMP traps to CentraSite** to publish data about the runtime events and metrics to the CentraSite SNMP server.
5. Select **Force reset CentraSite communication and SNMP details**, and click **Reset** to delete the current configuration and restore the system configuration.
6. Click **Save**.

This establishes a communication channel between API Gateway and CentraSite.

Note:

The transaction events are published from API Gateway to CentraSite using SNMP. The runtime metrics are published from API Gateway to CentraSite using a UDDI client.

Configuring Events for CentraSite Destination

Pre-requisites:

You have to configure CentraSite to communicate with API Gateway before you select the events for publishing to CentraSite.

You have to configure the CentraSite destination so that events and performance metrics data of APIs in API Gateway can be published to CentraSite. However, you will be able to publish the data to CentraSite destination only for APIs published from CentraSite to API Gateway.

➤ To configure events for API Gateway destination

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Destinations**.
3. Select **CentraSite > Events** to configure the event types for this destination.
4. In **Event types**, select the type of events that you want API Gateway to publish to CentraSite.

The available event types are:

- **Error**: Occurs each time an API invocation results in an error.
 - **Lifecycle**: Occurs each time API Gateway is started or shut down.
 - **Policy violation**: Occurs each time an API invocation violates the policy enforcement that was set for the API.
5. Select **Report performance data** to publish performance metrics data.
 6. In the **Publish interval** box, enter a time interval (in minutes) to specify how often API Gateway must publish performance metrics. Enter a value from 1 through 60. The default is 60 minutes.
 7. Click **Save**.

Post-requisites:

After performing the event configurations, select **CentraSite** as a **Destination** in the Policy Properties page for each policy, to publish the transaction and monitoring event logs for the assigned policies. API Gateway sends these events to CentraSite through SNMP.

Important:

As a best practice, Software AG recommends you not to use the CentraSite destination for transaction events with large data payloads. This is because, the SNMP server using which the events are published from API Gateway to CentraSite does not handle transaction events with large data payloads.

Configuring Events for Database Destination

Before you begin

For preparing the database connection, ensure that you have:

- Installed Database Components Configurator with API Gateway database scripts. For details, see *Create Database Components Using Database Component Configurator* section in *Installing Software AG Products Guide*.

- Started Database Components Configurator and created tables for the component, *APIGatewayEvents*. For details, see *Create Database Components, Database User, and Storage* section in *Installing Software AG Products Guide*.
- Created database connection pools in the Integration Server UI. For details, see *Creating a connection pool* section in *webMethods Integration Server Administrator's Guide*.
- Configured API Gateway functional alias to the database connection pool in the Integration Server UI. For details, see *Pointing functions at Connection pools* section in *webMethods Integration Server Administrator's Guide*.

You have to configure the database destination so that the events, performance metrics, and audit log data can be published to the configured database.

API Gateway supports the following databases:

- DB2
- Oracle
- SQL Server

➤ To configure events for Database destination

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Destinations**.
3. Select **Database** to configure the event types for this destination.
4. In **Event types**, select the type of events that you want API Gateway to publish to the database.

The available event types are:

- **Error**: Occurs each time an API invocation results in an error.
 - **Lifecycle**: Occurs each time API Gateway is started or shut down.
 - **Policy violation**: Occurs each time an API invocation violates the policy enforcement that was set for the API.
5. Select **Report performance data** to publish performance metrics data.
 6. In the **Publish interval** box, provide a time interval (in minutes) to specify how often API Gateway must publish performance metrics. Provide a value from 1 through 60. The default value is 60 minutes.
 7. In the **Audit log data** section, select the required management area for which the audit logs should be recorded in the Database destination.

Audit logs provide a record of system transactions, events, and occurrences in API Gateway. You can configure audit logging to show the following events:

- **API management**
- **Approval management**
- **Application management**
- **Alias management**
- **Team management**
- **Analytics management**
- **Group management**
- **Policy management**
- **Package management**
- **Plan management**
- **Promotion management**
- **User management**

Note:

By default, audit logging is disabled for all the mentioned management areas in the Database destination.

8. Click **Save**.

Post-requisites:

After performing the database configurations, select **Database** as a **Destination** in the Policy Properties page for each policy, to receive the transaction event logs for the assigned policies.

Configuring Events for Digital Events Destination

Pre-requisites:

API Gateway communicates with the Digital Event Services (DES) component which runs within the API Gateway server.

Digital Event Services (DES) enables API Gateway to communicate through digital events. Digital events are typed and serialized data structures used to communicate application or system runtime information. The application information could be state of a business process step or any associated business data. The system information could be the amount of memory used by an application.

You have to configure the Digital Event Services destination so that the event types and performance metrics data can be published to Digital Event Services.

➤ To configure events for Digital Event Services destination

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Destinations**.
3. Select **Digital Event Services > Events** to configure the event types for this destination.
4. In **Event types**, select the type of events that you want API Gateway to publish to Digital Event Services.

The available event types are:

 - **Error**: Occurs each time an API invocation results in an error.
 - **Lifecycle**: Occurs each time API Gateway is started or shut down.
 - **Policy violation**: Occurs each time an API invocation violates the policy enforcement that was set for the API.
5. Select **Report performance data** to publish performance metrics data.
6. In the **Publish interval** box, enter a time interval (in minutes) how often API Gateway must publish performance metrics. Enter a value from 1 through 60. The default is 60 minutes.
7. In the **Audit log data** section, select the required management area for which the audit logs should be recorded in the Digital Event Services destination.

Audit logs provide a record of system transactions, events, and occurrences in API Gateway. You can configure audit logging to show the following events:

- **API management**
- **Approval management**
- **Application management**
- **Alias management**
- **Team management**
- **Analytics management**
- **Group management**
- **Policy management**
- **Package management**
- **Plan management**

- **Promotion management**
- **User management**

Note:

By default, audit logging is disabled for all of the above-listed management areas in the Digital Event Services destination.

8. Click **Save**.

Post-requisites:

After performing the server configurations, select **Digital Events** as a **Destination** in the Policy Properties page for each policy, to receive the transaction events logs for the assigned policies.

Configuring Elasticsearch Destination

You have to configure Elasticsearch as a destination to establish a communication channel between API Gateway and Elasticsearch to exchange data.

➤ To configure Elasticsearch destination

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Destinations**.
3. Select **Elasticsearch > Configuration** to configure Elasticsearch as the destination.
4. Provide the following information in the Basic information section:

Field	Description
Protocol	Specifies the communication protocol used to establish communication between API Gateway and Elasticsearch.
Hostname	Specifies the host name or IP address of the machine on which Elasticsearch is running.
Port	Specifies the port where Elasticsearch server runs. The default port number is 9240.
Index name	Specifies the index name for Elasticsearch, where the data is stored.
Username	Specifies the Elasticsearch user ID for authenticating Elasticsearch when API Gateway communicates with it.
Password	Specifies the password of the Elasticsearch instance to be used for establishing communication between API Gateway and Elasticsearch.

Note:

You can provide the username and password for the Elasticsearch instances having configured with Basic authentication. You can also provide HTTPS enabled Elasticsearch instance.

5. Click **Test**.

This tests the communication channel between API Gateway and the configured Elasticsearch.

6. Click **Save** to save the specified email configuration value.

You can click **Cancel** to revert to the last saved changes or to abandon all the changes if the values are not saved.

Troubleshooting Tips: Elasticsearch Destination

The transaction logs are not stored in the external Elasticsearch

When I view the Analytics tab of an API, some of the transaction events are not displayed. For example, when the default limit of the total fields of that index exceeds 50000, I see the following error message in the Server Log:

```
type": "illegal_argument_exception", "reason": "Limit of total fields [50000] in index [gateway_default_analytics] has been exceeded .
```

Resolution:

Increase the limit of total fields for that index using the following PUT REST calls, depending on the version of the API Gateway:

- For 10.3 and lower versions, increase the limit using the following PUT REST call and sample payload:

```
http://<Elasticsearch host:port>/gateway_default_analytics/<TYPE>/_settings
```

Sample Payload:

```
{  
  "index.mapping.total_fields.limit": 70000  
}
```

- For 10.5 and higher versions, increase the limit using the following PUT REST call and sample payload:

```
http://<Elasticsearch host:port>/gateway_default_analytics_<TYPE>/_settings
```

Sample Payload:

```
{  
  "index.mapping.total_fields.limit": 70000  
}
```

Configuring Events for Elasticsearch Destination

Pre-requisites:

You have to configure Elasticsearch to communicate with API Gateway before you select Elasticsearch as a destination.

You have to configure Elasticsearch as a destination so that the event types and performance metrics data can be published to Elasticsearch.

➤ To configure Elasticsearch destination



1. Expand the menu options icon in the title bar, and select **Administration**.
2. Select **Destinations**.
3. Select **Elasticsearch > Events** to configure the event types for this destination.
4. In **Event types**, select the type of events that you want API Gateway to publish to Elasticsearch.

The available event types are:

- **Error**: Occurs each time an API invocation results in an error.
- **Lifecycle**: Occurs each time API Gateway is started or shut down.
- **Policy violation**: Occurs each time an API invocation violates the policy enforcement that was set for the API.

5. Select **Report performance data** to publish performance metrics data.
6. In the **Publish interval** box, enter a time interval (in minutes) to specify how often API Gateway must publish performance metrics. Enter a value from 1 through 60. The default is 60 minutes.
7. In the **Audit log data** section, select the required management area for which the audit logs should be recorded in the Elasticsearch destination.

Audit logs provide a record of system transactions, events, and occurrences in API Gateway. You can configure audit logging to show the following events:

- **API management**
- **Application management**
- **Team management**
- **Group management**

- **Package management**
- **Promotion management**
- **Approval management**
- **Alias management**
- **Analytics management**
- **Policy management**
- **Plan management**
- **User management**

Note:

By default, audit logging is disabled for all of the above-listed management areas in the Elasticsearch destination.

8. Click **Save**.

Post-requisites:

After performing the event configurations, select **Elasticsearch** as a **Destination** in the Policy Properties page for each policy, to publish the transaction and monitoring event logs for the assigned policies.

Configuring Email Destination

You have to configure email as a destination to receive alert notifications. The alerts are sent to the email ID specified in the Log Invocation, Monitor Performance, Monitor SLA, and Traffic Optimization policies. Before configuring email as a destination, you must perform the following email server configurations to establish a connection between API Gateway and the email server.

➤ To configure Email destination

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Destinations**.
3. Select **Email > Configuration**.
4. Provide the following information:

Field	Description
SMTP server	The server name or the IP address of the SMTP server that API Gateway uses to send the messages.
Port	The port that API Gateway uses to connect to the specified SMTP server.
Transport layer security	<p>The SSL encryption type that API Gateway uses when communicating with an email server. Use one of the following transport layer security options:</p> <ul style="list-style-type: none"> <li data-bbox="572 540 1481 677">■ None. Default. Use an unsecure mode when API Gateway is communicating with an email server. When you specify None, the email server authenticates the email client using only the username and password. <li data-bbox="572 699 1481 857">■ Explicit. Use explicit security when API Gateway is communicating with an email server. With this type of security, API Gateway initially establishes an unsecure connection with the email server and then upgrades to the secure mode. <p data-bbox="621 878 1481 1121">With explicit transport layer security, API Gateway communicates with the email servers that support SSL encryption and also with those that do not support SSL encryption. If the email server does not support transport layer security, API Gateway disconnects the connection which was established earlier with the email server. You can then establish an unsecure connection by selecting None in the Transport layer security field and enable the port.</p> <ul style="list-style-type: none"> <li data-bbox="572 1142 1481 1279">■ Implicit. Use implicit security when API Gateway communicates with an email server. With this type of security, API Gateway always establishes an encrypted connection with the email server. Only clients that support SSL are allowed to access API Gateway.
Truststore alias	The truststore that API Gateway uses while sending the request to a native API. Truststore is a repository that stores all the trusted public certificates.
Default email charset	The character set that API Gateway uses to be recognized by the system. Type the character set in the Default email charset field. By default, API Gateway uses UTF-8 character set for the messages.
From email	The email address from which you want to send the alerts.
Test email	The email address to which you want to send the test email. This email address can be the same as the From email address.
Internal email	The email address to which you want to send critical log entry messages. Typically, you would specify the e-mail address for the API Gateway Administrator.

Field	Description
Service email	The email address to which you want to send service failure message. In a development environment, you might direct these messages to the developer. In a production environment, you might direct these messages to the API Gateway Administrator.

5. Click **Test**.

This tests the communication channel between API Gateway and the configured email server.

6. Click **Save** to save the specified email configuration value.

You can click **Cancel** to revert to the last saved changes or to abandon all the changes if the values are not saved.

Post-requisites:

After performing the server configurations, select **Email** as a **Destination** in the Policies properties page for each policy, to receive the email alerts for the assigned policies.

Configuring Email Templates

API Gateway provides a default email template to send email alerts. You can compose and save the subject line as well as the email content for reuse. You can also customize the template to suit your needs.

➤ To configure Email Templates

1. Expand the menu options icon , in the title bar, and select **Administration**.
2. Select **Destinations**.
3. Select **Email > Templates** to configure the event templates.
4. Specify the following for the Log Invocation, Monitor Service Level Agreement, Monitor Performance, and Traffic Optimization events:
 - **Subject:** The subject line of the email to be sent.
 - **Content:** By default, the template appears. You can customize the email content.

The template consists of the following default information for the Log Invocation event:

Note:

The @ character is a place holder and the values are automatically generated by the system. For example, Status: @status appears as Status: SUCCESS in the email. You can use the existing parameters multiple times, delete the parameter if the parameter is not required

from the available parameters, or use the corresponding *optional* parameters in the template. However, you cannot add new parameters.

The transaction event parameters from the API Gateway Metrics and Event Notification engine are:

```
Runtime_Policy: @policy_action_name
API: @api_name
Version : @version
Operation or Resource_Name: @operation_resource_name
Native endpoint: @native_endpoint
Event generation time: @description
Consumer_Name: @consumer_name
Consumer_ID: @consumer_ID
Status: @status
Coorelation_ID:@correlationID
Error origin: @errorOrigin
```

The *optional* parameters that you can include in the template for the **Log Invocation event** are:

```
Native payload : @nativeResponsePayload
nativeRequestHeaders:@nativeRequestHeaders
nativeRequestPayload:@nativeRequestPayload
nativeResponseHeaders:@nativeResponseHeaders
nativeResponsePayload:@nativeResponsePayload
nativeHttpMethod:@nativeHttpMethod
nativeURL:@nativeURL
externalCalls:@externalCalls
sourceGatewayNode:@sourceGatewayNode
```

The template consists of the following default information for the **Monitor Service Level Agreement, Monitor Performance** and **Traffic Optimization** events:

Note:

You can use the existing parameters multiple times or delete the parameter if the parameter is not required from the available parameters in the template. However, you cannot add new parameters.

The monitor event parameters from the API Gateway Metrics and Event Notification engine are:

```
Runtime_Policy: @policy_action_name
API: @api_name
Version : @version
Operation or Resource_Name: @operation_resource_name
Native endpoint: @native_endpoint
Action type: @actionType
Attribute: @attribute
Consumer_Name: @consumer_name
Consumer_ID: @consumer_ID
Alert Message: @alertMessage
```

Additionally, you can click  to abandon the changes and revert to the default template.

You can click  to review the changes before adding the changes to the email content.

5. Click **Save**.

Configuring SNMP Destination

You have to configure SNMP as a destination to send the events to a third-party SNMP server.

➤ To configure SNMP destination

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Destinations**.
3. Select **SNMP > Configuration**.
4. Provide the following information:

Field	Description
Send TRAPs to 3rd party SNMP server	Specifies API Gateway to use the SNMP traps to capture events that you can send to a third-party SNMP server.
SNMP target type	<p>Specifies target type of the SNMP server. The available options are:</p> <ul style="list-style-type: none"> ■ User: To send the events information to a SNMP user. ■ Community: To send the events information to a group of SNMP users that form a community. <p>By default, User is selected.</p>
Hostname	The host name or IP address of the machine where the CentraSite SNMP or the third-party SNMP server is installed and running.
<p>Note: The host name cannot be localhost.</p>	
Port	The SNMP trap receiver port that is listening for SNMP requests and packets.
Transport	<p>The protocol used by SNMP to send traps. The available options are: TCP and UDP. By default, TCP is selected.</p> <p>If you select UDP, ensure that the SNMP server is in the same subnet, or configure the routers to get packets across subnet boundaries. The maximum PDU size when running in UDP mode is 64 Kb which might be restrictive when sending large request and response payloads for Transaction Events.</p>
Username	If User is selected as the SNMP target type , then the Username field specifies the SNMPv3 user name to use when connecting to the receiver.

Field	Description
	If Community is selected as the SNMP target type , then the Community name field specifies the name to be used to interact with the SNMP receiver. This value must match the value that you set in the SNMP receiver.
Use authorization	Specifies whether an authorization key is required. You cannot edit the authorization fields unless Use authorization option is selected. The authorization fields are: <ul style="list-style-type: none"> ■ Password: The key to be used for authorization. ■ Protocol: The authorization protocol is MD5
Use privacy	Specifies whether a privacy (encryption) key is required. You cannot edit the privacy fields unless Use privacy option is selected. The privacy fields are: <ul style="list-style-type: none"> ■ Password: The key to be used for privacy. ■ Protocol: The privacy protocol is DES.

- Click **Save** to save the specified SNMP configuration value.

You can click **Cancel** to revert to the last saved changes or to abandon all the changes if the values are not saved.

Post-requisites:

After performing the server configurations, select **SNMP** as a **Destination** in the Policies properties page for each policy, to receive the transaction events logs for the assigned policies.

Note:

Metrics is not supported for third-party SNMP server.

Configuring Events for SNMP Destination

Pre-requisites:

You have to configure a SNMP server to communicate with API Gateway before you select SNMP as a destination for the events and metrics for publishing to a third-party SNMP server.

You have to configure SNMP as a destination so that the event types data can be published to a third-party SNMP server.

➤ To configure events for SNMP destination

- Expand the menu options icon  in the title bar, and select **Administration**.

2. Select **Destinations**.
3. Select **SNMP > Events** to configure the event types for this destination.
4. In **Event types**, select the type of events that you want API Gateway to publish to SNMP.

The available event types are:

- **Error**: Occurs each time an API invocation results in an error.
- **Lifecycle**: Occurs each time API Gateway is started or shut down.
- **Policy violation**: Occurs each time an API invocation violates the policy enforcement that was set for the API.

5. Click **Save**.

Post-requisites:

After configuring the events for SNMP destination, you need to select **SNMP** as a **Destination** in the Policy Properties page of the API to publish the transaction and monitoring events for the API.

Custom Destination

You can configure custom destinations to publish events, performance metrics, and audit logs to a required destination.

You can configure the following four types of destinations using the custom destinations feature:

- **External endpoint**

Use this type to configure an external REST endpoint URL as a destination.

- **webMethods IS service**

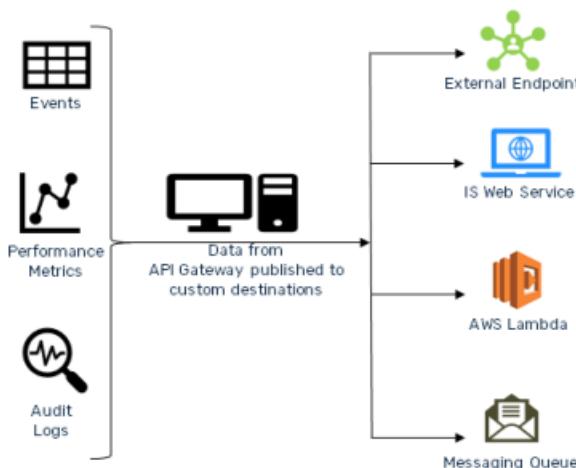
Use this type to configure a webMethods IS service as a destination.

- **AWS Lambda**

Use this type to configure an Amazon Web Services (AWS) Lambda function as a destination.

- **Messaging**

Use this type to configure a message queue as a destination. Messaging systems can read the message from the queue or topic and process it asynchronously.



Why Custom Destination?

API Gateway provides options to publish events and logs to preset destinations. But sometimes customer might require the data to be published to a custom destination for further data processing and for generating various reports as per their business requirements. The custom destination feature offers solution to this requirement.

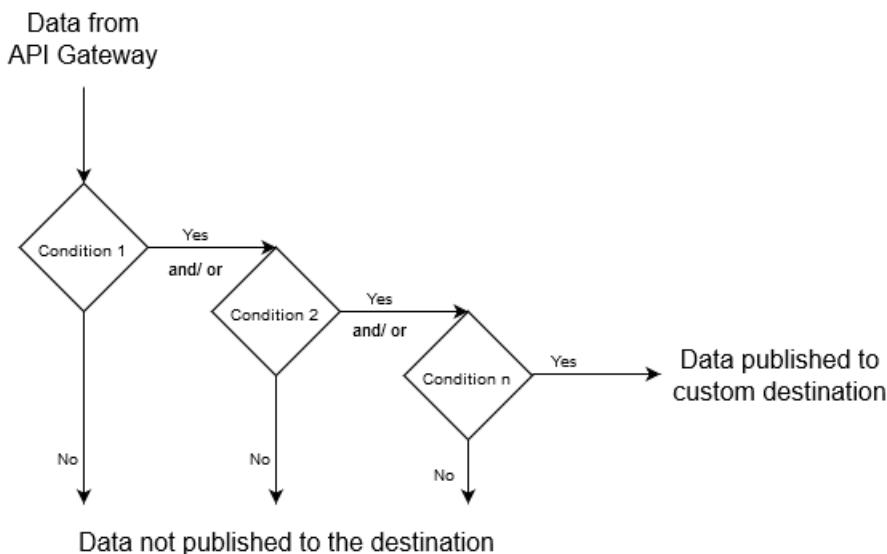
You can configure custom destinations to publish either or all of the following:

- Design time events such as audit logs of API Gateway modules.
- Error events and policy violation events of assets, and Performance metrics data.
- Traffic monitoring payloads and alerts of an API

You can use custom destination feature to perform:

Condition based publishing

You can configure conditions based on which API Gateway filters events to publish to a configured destination. That is, only the events that satisfy your conditions are published to the given destination. For example, you can configure a condition to publish the error events of an application, say *app1*, to a destination; and another condition to publish the error events of another application, *app2*, to a second destination and so on.



To configure a condition, you can use variables available in the variable framework, and specify a matching value based on which the condition must be validated. You can specify multiple conditions and configure whether the data to be published must satisfy all or any of the given conditions. The use cases in this section explain the process of configuring conditions.

Steps to configure a custom destination

The following diagram outlines the steps involved in configuring a custom destination:



You can configure any number of custom destinations.

How Do I Publish Data to an External Endpoint using Custom Destination?

This use case explains how to publish data to a REST endpoint using custom destination.

The use case starts when you have data to be published and ends when you have successfully configured a REST endpoint URL as a destination to publish the data.

Ensure you have the external endpoint URL to which you want to publish the data.

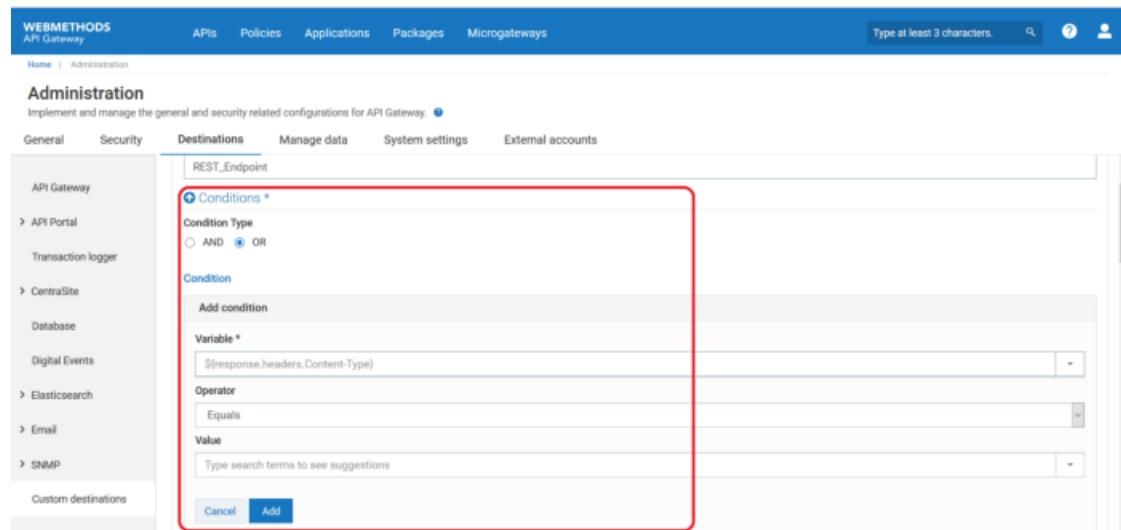
➤ To publish data to an external endpoint

1. Expand the menu options icon  in the title bar, and select **Administration**.

2. Click **Destinations**.
3. Select **Custom destinations** from the left navigation pane.
4. Click **+ Add custom destination**.
5. Provide the name of the custom destination in the **Name** field.

The name must be unique and must not be the name of any pre-defined API Gateway destinations such as Elasticsearch.

6. To configure conditions that determine the data to be published in the specified destination, perform the following steps in the **Conditions** section:
- Select one of the following options in the **Condition type** field:
 - **AND**. To publish data that satisfies all your conditions.
 - **OR**. To publish data that satisfies one of your conditions.
 - Click **+ Add condition**.
 - Provide the following details for your condition:
 - **Variable**. Name of the variable based on which you want to validate your condition. This field supports the variables that are available in the Variable framework. For details on the list of variables, see *webMethods API Gateway User's Guide*.
 - **Operator**. The operator to use to relate variable and the value.
 - **Value**. The value of the variable that must be matched to satisfy the condition.

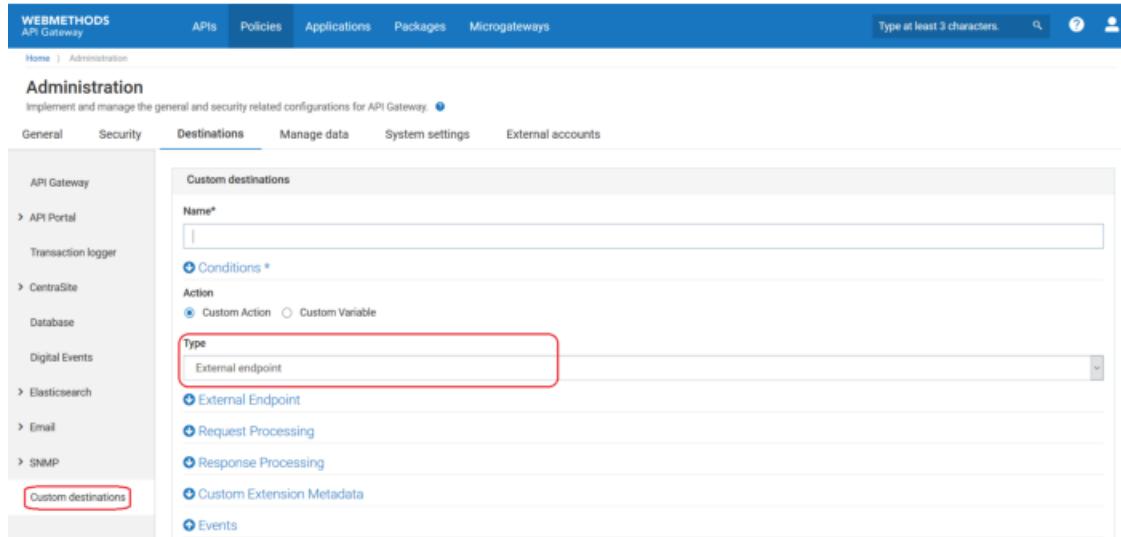


- Click **Add**.

The condition appears in the grid.

Repeat this process to add the required number of conditions. Click on a condition to edit it and click  next to a condition to delete it.

7. Select **External endpoint** in the **Type** field.



The screenshot shows the 'Administration' section of the webMethods API Gateway. On the left, there's a sidebar with various options like API Gateway, API Portal, Transaction logger, CentraSite, Database, Digital Events, Elasticsearch, Email, SNMP, and 'Custom destinations' (which is highlighted with a red box). The main area has tabs for General, Security, Destinations (which is selected), Manage data, System settings, and External accounts. Under Destinations, there's a 'Custom destinations' section with fields for Name*, Conditions*, Action (set to Custom Action), and Type (set to External endpoint, also highlighted with a red box). Below these are other options like External Endpoint, Request Processing, Response Processing, Custom Extension Metadata, and Events.

8. Provide the following information in the External Endpoint section, as required:

Property	Description
Endpoint URI	Provide the REST endpoint of the destination to publish the specified events. For example, <code>http://localhost:9292/rest_endpoint/</code> .
Method	Specify the method exposed by the API. Available values are: GET , POST , PUT , and DELETE .
SSL Configuration	Specifies the required SSL configuration details of the external endpoint. Provide the following information: <ul style="list-style-type: none"> ■ Keystore Alias. Specifies the keystore alias. For details on Keystore configuration, see “Keystore and Truststore” on page 548. ■ 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 “Keystore and Truststore” on page 548.

Property	Description
	<ul style="list-style-type: none"> ■ 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	<p>Specifies the path parameter you want to configure to your custom extension.</p> <p>Provide the following information:</p> <ul style="list-style-type: none"> ■ Path Parameter Name. Specifies 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 <code>http://host/authors/{id}/books</code> and provide <code>id</code> as path parameter name with the value you need to populate at runtime. ■ Path Parameter Value. Specifies the value for the path parameter specified.

The screenshot shows the 'Administration' page of the webMethods API Gateway. The 'Destinations' tab is active. On the left, a sidebar lists various destination types: API Gateway, API Portal, Transaction logger, CentraSite, Database, Digital Events, Elasticsearch, Email, SNMP, and Custom destinations. The 'Custom destinations' option is currently selected. In the main area, under 'External Endpoint', the 'Endpoint URI' field contains 'http://localhost:9240/_rest_endpoint/_doc' and the 'Method' dropdown is set to 'POST'. Other fields like 'SSL Configuration' and 'Keystore Alias' are also visible but not highlighted.

9. Configure the custom properties of the custom extension as required.

For details about the custom extension properties and their descriptions, see *webMethods API Gateway User's Guide*.

10. From the **Events** section, select the data that you want to publish to the configured destination. The options available are:

- **Event types**. Type of events to publish to the specified destination. The available event types are:

- **Error:** Occurs each time an API invocation results in an error.
- **Lifecycle:** Occurs each time API Gateway is started or shut down.
- **Policy violation:** Occurs each time an API invocation violates the policy enforcement that was set for the API.
- **Performance metrics data.** To publish to the specified destination.
In the **Publish interval of performance metrics data** field, type a time interval (in minutes) to specify how often API Gateway must publish performance metrics. Provide a value from 1 through 60. The default is 60 minutes.
- **Events.** API Gateway modules for which the audit logs to publish to the specified destination.

11. Click **Add**.

The custom destination is created successfully and appears in the **Custom destinations** page. The configured events are published in the specified destination.

Note:

To edit a custom destination, you can click the required custom destination, make changes and click Update. To delete a custom destination, click  next to required custom destination. You cannot delete a custom destination that is associated with an API.

How Do I Publish API-specific Traffic Monitoring Data to a Custom Destination?

This use case explains how to publish traffic monitoring policy alerts to a custom destination of the external endpoint type.

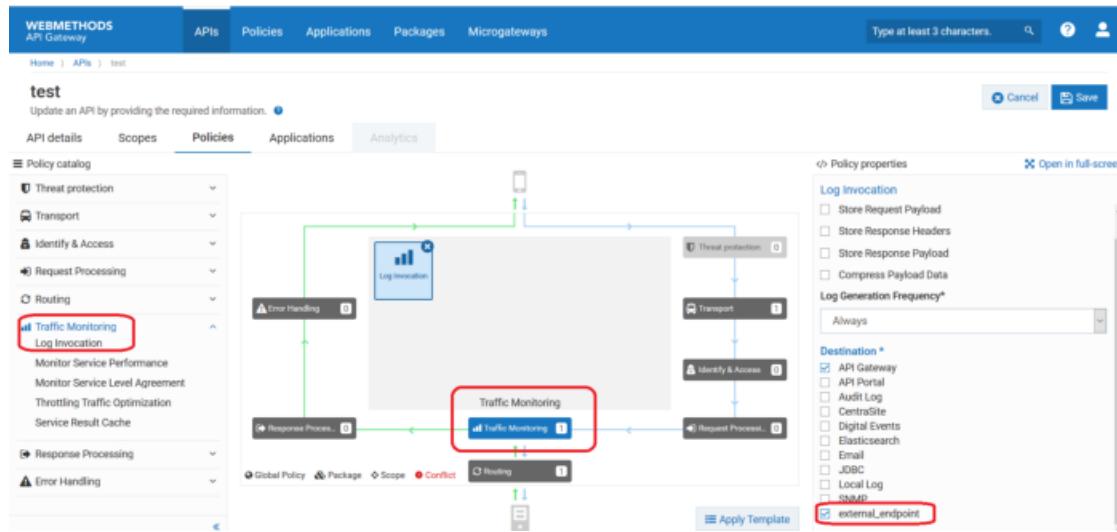
The use case starts when you have data to be an API for which you configure traffic monitoring policy and ends when you have successfully selected a custom destination to publish the logs.

Ensure you have an API and a custom destination of the type external endpoint (with required REST Endpoint URL configured).

➤ To publish traffic monitoring logs to a custom destination

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 **Traffic Monitoring** and select a required policy.
6. Select the custom destination from the **Destination** section in the **Policy properties** pane.

- Select any other required details such as Alert interval, Unit, Alert frequency, Alert message, and so on.



- Click **Save**.

The policy logs are published to the REST endpoint specified in the selected custom destination.

How Do I Publish Data to an Integration Server Service using Custom Destination?

This use case explains how to publish data to an IS service.

The use case starts when you have data to be published and ends when you have successfully configured an IS service as a destination to publish the data.

Ensure you have the IS service details to which the data has to be published.

➤ To publish data to a IS service

- Expand the menu options icon  in the title bar, and select **Administration**.

- Click **Destinations**.

- Select **Custom destinations** from the left navigation pane.

The Custom destination page appears.

- Provide the name of the custom destination in the **Name** field.

The name must be unique and must not be the name of any pre-defined API Gateway destinations such as `Elasticsearch`.

- To configure conditions that determine the data to be published in the specified destination, perform the following steps in the **Conditions** section:

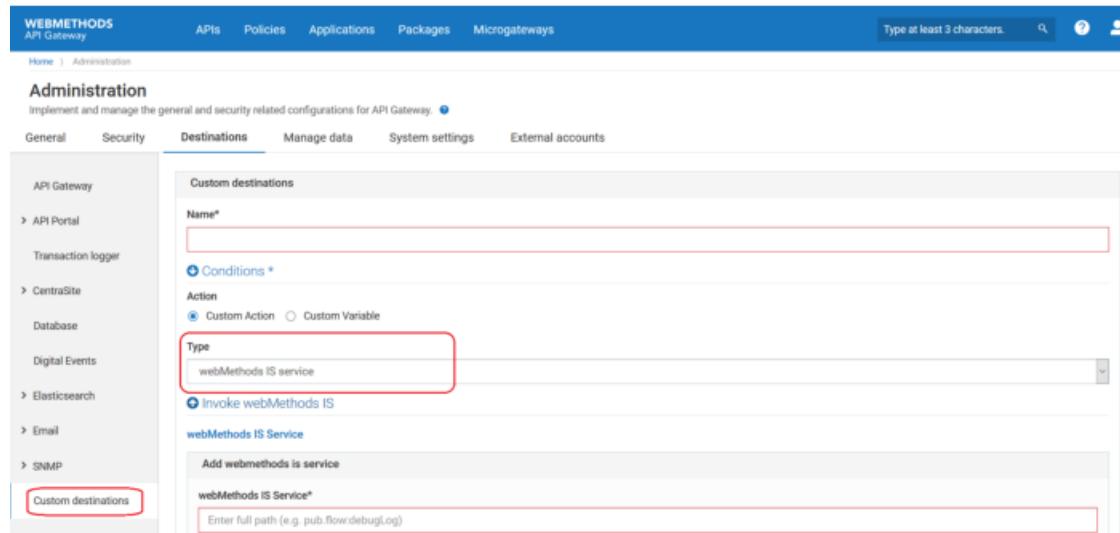
- Select one of the following options in the **Condition type** field:

- **AND.** To publish data that satisfies all your conditions.
- **OR.** To publish data that satisfies one of your conditions.
- Click **+ Add condition.**
- Provide the following details for your condition:
 - **Variable.** Name of the variable based on which you want to validate your condition. This field supports the variables that are available in the Variable framework. For details on the list of variables, see *webMethods API Gateway User's Guide*.
 - **Operator.** The operator to use to relate variable and the value.
 - **Value.** The value of the variable that must be matched to satisfy the condition.
- Click **Add.**

The condition appears in the grid.

Repeat this process to add the required number of conditions. Click on a condition to edit it and click  next to a condition to delete it.

6. Select **webMethods IS service** in the **Type** field.



The screenshot shows the 'Administration' screen of the webMethods API Gateway. The 'Destinations' tab is active. On the left, there's a sidebar with various options like API Gateway, API Portal, Transaction logger, etc. In the main area, under 'Custom destinations', a condition is being defined. The 'Type' dropdown is set to 'webMethods IS service'. A red box highlights both the 'Type' dropdown and the 'Custom destinations' section.

7. Provide the following information in the Invoke webMethods IS section, as required:

Property	Description
webMethods IS Service	Name of the webMethods IS service or IS server alias to which the data has to be published.

Note:
Only the local IS service is supported. You cannot specify a remote IS service.

Property	Description
Run As User	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.

Click **Add webMethods IS Service** to add another IS service. Repeat this step to add the required number of services.

The screenshot shows the 'Administration' section of the webMethods API Gateway. The 'Destinations' tab is selected. A table lists two 'webMethods IS Service' entries: 'webMethods IS Service' and 'customservice.changeresponse_post'. For the first entry, 'Run As User' is set to 'Administrator' and 'Comply to IS Spec' is set to 'True'. Below the table, there are sections for 'Events' and 'Performance metrics data'.

IS Service	Run As User	Comply to IS Spec (pub.apigateway.invokeISService.spec...)	Action
webMethods IS Service	Administrator	True	
customservice.changeresponse_post			

Events
Configure the event types to publish to the destination
 Error Lifecycle Policy violation

Performance metrics data
 Report performance data
Publish interval of performance metrics data

- From the **Events** section, select the data that you want to publish to the configured destination. The options available are:

- **Event types.** Type of events to publish to the specified destination. The available event types are:
 - **Error.** Occurs each time an API invocation results in an error.
 - **Lifecycle.** Occurs each time API Gateway is started or shut down.
 - **Policy violation.** Occurs each time an API invocation violates the policy enforcement that was set for the API.
- **Performance metrics data.** To publish to the specified destination.

In the **Publish interval of performance metrics data** field, enter a time interval (in minutes) to specify how often API Gateway must publish performance metrics. Provide a value from 1 through 60. The default is 60 minutes.

- **Events.** API Gateway modules for which the audit logs to publish to the specified destination.
9. Click **Add**.

The custom destination is created successfully and appears in the **Custom destinations** page. The configured events are published to the specified destination.

Note:

To edit a custom destination, you can click the required custom destination, make changes and click Update. To delete a custom destination, click  next to required custom destination. You cannot delete a custom destination that is associated with an API.

How Do I Publish Data to an AWS Lambda Function using Custom Destination?

This use case explains how to publish data to an AWS Lambda function.

The use case starts when you have data to be published and ends when you have successfully configured an AWS Lambda service as a destination to publish the data.

- Ensure you have an active Lambda function. For details on how to create an AWS Lambda function, see <https://docs.aws.amazon.com/lambda/latest/dg/getting-started.html>
- Ensure you have configured AWS alias. For details on how to configure an AWS alias, see “[Configuring an AWS Alias](#)” on page 538.

➤ To publish data to an AWS Lambda function

1. Click **APIs** on the title navigation bar.
2. Expand the menu options icon , in the title bar, and select **Administration**.
3. Click **Destinations**.
4. Select **Custom destinations** from the left navigation pane.

The Custom destination page appears.

5. Provide the name of the custom destination in the **Name** field.

The name must be unique and must not be the name of any pre-defined API Gateway destinations such as Elasticsearch.

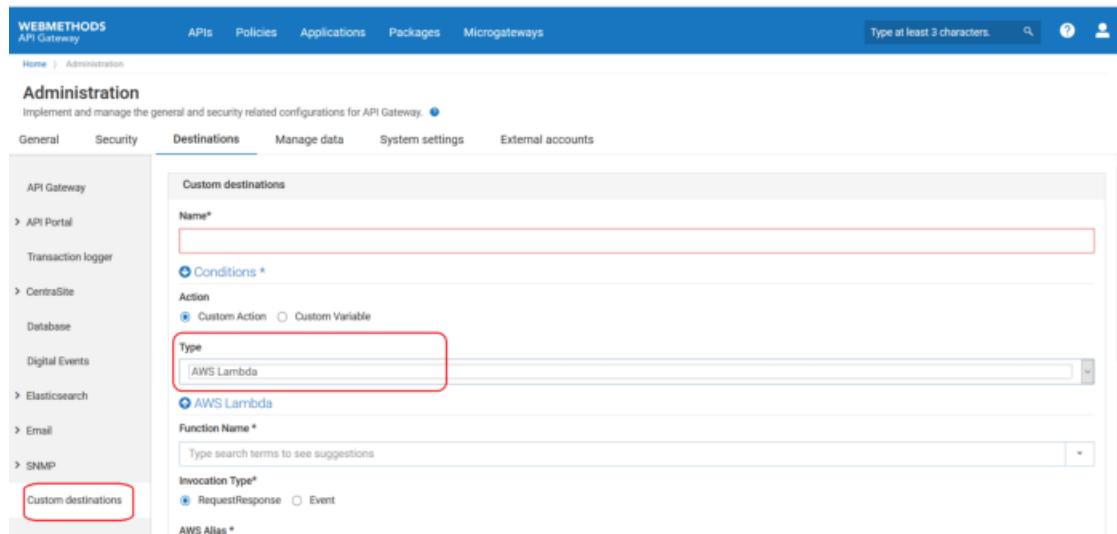
6. To configure conditions that determine the data to be published in the specified destination, perform the following steps in the **Conditions** section:
 - Select one of the following options in the **Condition type** field:
 - **AND.** To publish data that satisfies all your conditions.
 - **OR.** To publish data that satisfies one of your conditions.

- Click **+ Add condition**.
- Provide the following details for your condition:
 - **Variable**. Name of the variable based on which you want to validate your condition. This field supports the variables that are available in the Variable framework. For details on the list of variables, see *webMethods API Gateway User's Guide*.
 - **Operator**. The operator to use to relate variable and the value.
 - **Value**. The value of the variable that must be matched to satisfy the condition.
- Click **Add**.

The condition appears in the grid.

Repeat this process to add the required number of conditions. Click on a condition to edit it and click  next to a condition to delete it.

7. Select **AWS Lambda** in the **Type** field.



The screenshot shows the 'Administration' section of the webMethods API Gateway. On the left, there's a sidebar with various destination types like API Gateway, API Portal, Transaction logger, etc. A red box highlights the 'Custom destinations' option. The main panel shows a form for creating a new custom destination. The 'Type' dropdown is set to 'AWS Lambda', also highlighted with a red box. Other fields include 'Name*', 'Conditions*', 'Action' (set to 'Custom Action'), 'Function Name*', 'Invocation Type*' (set to 'RequestResponse'), and 'AWS Alias*'. The top navigation bar includes links for APIs, Policies, Applications, Packages, and Microgateways.

8. Provide the following information in the AWS Lambda section, as required:

Property	Description
Function Name	Provide the AWS Lambda function name to which you want to publish the configured data.
Invocation Type	<p>Specify the AWS invocation type, asynchronous or synchronous.</p> <p>Available options are:</p> <ul style="list-style-type: none"> ■ RequestResponse (synchronous type) ■ Event (asynchronous type)

Property	Description
AWS Alias	Provide the AWS alias configured for the AWS account.

The screenshot shows the 'Destinations' tab selected in the navigation bar. On the left, a sidebar lists various destination types. The 'AWS Lambda' configuration is highlighted with a red box. It includes fields for 'Function Name' (CustomDestinationAWS), 'Invocation Type' (RequestResponse selected), and 'AWS Alias' (\$AWSL).

9. Configure the custom properties of the custom extension as required.

For details about the custom extension properties and their descriptions, see *webMethods API Gateway User's Guide*.

10. From the **Events** section, select the data that you want to publish to the configured destination. The options available are:

- **Event types.** Type of events to publish to the specified destination. The available event types are:
 - **Error.** Occurs each time an API invocation results in an error.
 - **Lifecycle.** Occurs each time API Gateway is started or shut down.
 - **Policy violation.** Occurs each time an API invocation violates the policy enforcement that was set for the API.
- **Performance metrics data.** To publish to the specified destination.

In the **Publish interval of performance metrics data** field, type a time interval (in minutes) to specify how often API Gateway must publish performance metrics. Provide a value from 1 through 60. The default is 60 minutes.

- **Events.** API Gateway modules for which the audit logs to publish to the specified destination.

11. Click **Add**.

The custom destination is created successfully and appears in the **Custom destinations** page. The configured events are published to the specified destination.

Note:

To edit a custom destination, you can click the required custom destination, make changes and click Update. To delete a custom destination, click  next to required custom destination. You cannot delete a custom destination that is associated with an API.

How Do I Publish Data to a Message Queue using Custom Destination?

This use case explains how to publish data to a message queue.

The use case starts when you have data to be published and ends when you have successfully configured a message queue as a destination to publish the data.

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 “[Messaging](#)” on page 401.

To publish data to a message queue

1. Expand the menu options icon , in the title bar, and select **Administration**.
2. Click **Destinations**.
3. Select **Custom destinations** from the left navigation pane.

The Custom destination page appears.

4. Provide the name of the custom destination in the **Name** field.

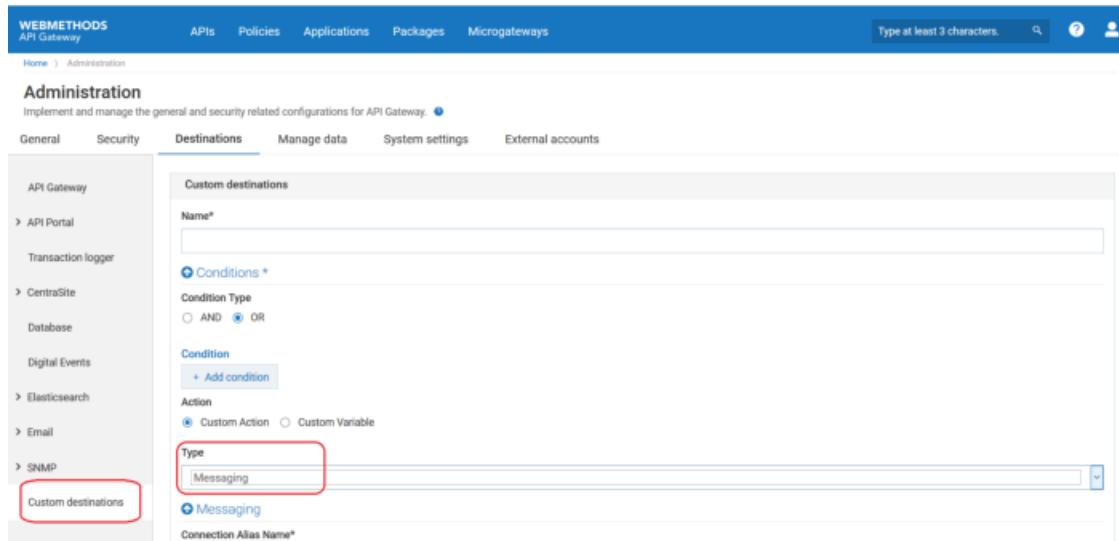
The name must be unique and must not be the name of any pre-defined API Gateway destinations such as Elasticsearch.

5. To configure conditions that determine the data to be published in the specified destination, perform the following steps in the **Conditions** section:
 - Select one of the following options in the **Condition type** field:
 - **AND**. To publish data that satisfies all your conditions.
 - **OR**. To publish data that satisfies one of your conditions.
 - Click **+ Add condition**.
 - Provide the following details for your condition:
 - **Variable**. Name of the variable based on which you want to validate your condition. This field supports the variables that are available in the Variable framework. For details on the list of variables, see *webMethods API Gateway User's Guide*.
 - **Operator**. The operator to use to relate variable and the value.
 - **Value**. The value of the variable that must be matched to satisfy the condition.
 - Click **Add**.

The condition appears in the grid.

Repeat this process to add the required number of conditions. Click on a condition to edit it and click  next to a condition to delete it.

6. Select **Mesaging** in the **Type** field.



The screenshot shows the 'Administration' section of the webMethods API Gateway. On the left, there's a sidebar with various options like API Gateway, API Portal, Transaction logger, etc. A red box highlights the 'Custom destinations' link under 'API Gateway'. The main panel shows a 'Custom destinations' form. It has fields for 'Name*', 'Conditions *', 'Condition Type' (set to OR), 'Action' (set to Custom Action), and a 'Type' dropdown which is also highlighted with a red box and set to 'Messaging'. There's also a 'Connection Alias Name*' field at the bottom.

7. 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 " "Messaging" on page 401 .
Destination Name	Specify the destination to which the request message is sent. A new destination is created with this name to publish events.
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	Specify the destination type to which the response message is sent. Select one of the following types: <ul style="list-style-type: none"> ■ QUEUE. Indicates that the response message is sent to a particular queue. ■ TOPIC. Indicates that the response message is sent to a particular topic.

Property	Description
Time to Live (ms)	<p>Provide a numeric value that specifies the expiration time (in milliseconds) of the JMS or AMQP message.</p> <p>If the time-to-live is specified as zero, expiration is set to zero, which indicates that the message does not expire.</p>
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	<p>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.</p> <p>Select one of the following modes:</p> <ul style="list-style-type: none"> ■ 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.

The screenshot shows the 'Administration' section of the webMethods API Gateway. On the left, there's a sidebar with links like 'General', 'Security', 'Destinations', 'Manage data', 'System settings', and 'External accounts'. Under 'Destinations', 'Custom destinations' is selected. The main panel shows a form for configuring a 'Messaging' destination. The fields include 'Connection Alias Name*' (set to 'AMQP.CON'), 'Destination Name*' (set to 'customdestinationqueue1'), 'Destination Type*' (radio button selected for 'QUEUE'), 'ReplyTo Name' (empty), 'ReplyTo Type*' (radio button selected for 'QUEUE'), 'Time to Live (ms)*' (set to '30000'), and 'Time to Wait (ms)*' (set to '30000'). There are also tabs for 'General', 'Security', 'Destinations', 'Manage data', 'System settings', and 'External accounts' at the top of the main panel.

8. Configure the custom properties of the custom extension as required.

For details on the custom extension properties and their description, see *webMethods API Gateway User's Guide*.

9. From the **Events** section, select the data that you want to publish to the configured destination. The options available are:

- **Event types.** Type of events to publish to the specified destination. The available event types are:

- **Error:** Occurs each time an API invocation results in an error.
- **Lifecycle:** Occurs each time API Gateway is started or shut down.
- **Policy violation:** Occurs each time an API invocation violates the policy enforcement that was set for the API.
- **Performance metrics data.** To publish to the specified destination.

In the **Publish interval of performance metrics data** field, type a time interval (in minutes) to specify how often API Gateway must publish performance metrics. Provide a value from 1 through 60. The default is 60 minutes.

- **Events.** API Gateway modules for which the audit logs to publish to the specified destination.

10. Click **Add**.

The custom destination is created successfully and appears in the **Custom destinations** page. The configured events are published to the specified destination.

Note:

To edit a custom destination, you can click the required custom destination, make changes and click Update. To delete a custom destination, click  next to required custom destination. You cannot delete a custom destination that is associated with an API.

Troubleshooting Tips: API Gateway - API Portal Integration

I see an obsolete API Portal destination reference in a published API

An old API Portal destination reference is associated to all APIs. This old reference is not available anymore, however, it is still associated to all APIs. Hence, all APIs have two associated API Portal destinations, one correct and the other obsolete.

Remove the obsolete reference as follows:

1. Retrieve API details using the following GET REST call:

```
http://localhost:5555/rest/apigateway/apis/apiId
```

2. Retrieve associated portal information using the following GET REST call:

```
http://localhost:5555/rest/apigateway/portalGateways
```

3. Retrieve the API Gateway to API Portal association using the following GET REST call:

```
http://localhost:9240/gateway_default/deploymentmap/_search
```

For every API, this displays two entries; one for the actual portal reference and another for the stale reference. You can delete the stale reference documents from the datastore.

You can delete the stale reference by using the REST request as follows:

```
DELETE http://localhost:9240/gateway_default/deploymentmap/<id>
```

The *id* you provide here is the id corresponding to the stale reference details you see in the response for Step 3.

I do not see a subscription object in API Portal as well as API Gateway when I subscribe for a plan in API Portal

If you do not see a subscription object in API Portal or API Gateway when you subscribe for a plan in API Portal, it might be due to one of the following reasons:

- Pending approval for the approval flow process.

Resolution: Go to **User menu > Pending Requests**. Accept any requests that are pending for approval.

- API Gateway not reachable from API Portal through the value configured in the destination.

Resolution: Go to **User menu > Administration > Destinations > API Portal > Configuration**. Verify the endpoint configurations. Provide correct credentials and ensure that the hosts are reachable using the **Test** option.

- General connection issues between API Gateway and API Portal.

Resolution: Perform the following general connection validation checks:

- Ensure that the API Gateway host is reachable from API Portal and API Portal is reachable from API Gateway host.
- Ensure that the ports are open and accessible.
- Ensure that in case of a Docker setup, the Gateway endpoint is configured with the port exposed from the host and not the actual Docker port.

I cannot publish an API to API Portal

If publishing APIs to API Portal fails, it might due to incorrect destination configuration for API Portal.

Resolution: Go to **User menu > Administration > Destinations > API Portal**. Ensure that all the required destination configuration details are provided correctly.

If you make any changes you must republish the configuration for the changes to take effect.

I see stale applications in API Portal

When a consumer subscribes to a plan or a package, an application is created both in API Gateway and API Portal. By mistake, when you delete an application from Gateway, the application in API Portal becomes stale.

To remove the stale applications from API Portal, unpublish the package associated with the application and republish it. This triggers the cleanup of the stale application in API Portal.

Audit Logging

The audit logging feature of API Gateway provides audit information for different categories of system transactions, events, and occurrences of specific events (for example, login attempts) over a period of time. You can use audit logs to view a detailed record of various auditable events that occurred on the API Gateway objects, user login and logout operations, and identify the users who are responsible for the changes. You can configure which audit events to log for a specific destination based on your auditing requirements.

You can configure API Gateway to log the auditable events for following destinations:

- API Gateway
- Database
- Digital Event Services (DES)
- Elasticsearch

The following auditable events can be configured to write to the API Gateway audit logs:

- **API management events**

API management consists of the following events:

- Creation, modification, and deletion of an API object.
- Activation and deactivation of an API.

- **Approval management events**

Approval management consists of the following events:

- Approval and rejection of a request to create, register, and modify an application.
- Approval and rejection of a request to subscribe a package in API Portal.

- **Application management events**

Application management consists of the following events:

- Creation, modification, and deletion of an Application object.

- **Alias management events**

Alias management consists of the following events:

- Creation, modification, and deletion of an Alias object.

- **Team management events**

Team management consists of the following events:

- Creation, modification, and deletion of teams.

- **Analytics management events**

Analytics management consists of the following events:

- Archiving, purging, and restoring of analytics data in the database.
- **Group management events**

Group management consists of the following events:

- Creation, modification, and deletion of a Group object.
- **Policy management**

Policy management consists of the following events:

- Creation, modification, and deletion of a global Policy object.
- Creation, modification, and deletion of an API level Policy object.
- Activation and deactivation of a global policy.
- Activation and deactivation of an API level policy.

■ **Package management events**

Package management consists of the following events:

- Creation, modification, and deletion of a Package object.

■ **Plan management events**

Plan management consists of the following events:

- Creation, modification, and deletion of a Plan object.

■ **Promotion management events**

Promotion management consists of the following events:

- Creation, modification, and deletion of a Stage object.
- Promotion of an API stage.
- Rollback operation of an API stage.

■ **User management events**

User management consists of the following events:

- A user logs in or fails to log in to API Gateway.
- A user logs out of API Gateway.
- Creation, modification, and deletion of a User object.

API Gateway writes the audit logging data to the **Audit logs** dashboard (in the API Gateway user interface, go to **Analytics > Audit logs**). You can view and download audit logs.

Best Practices for API Gateway Audit Logging

API Gateway's audit logging feature has been implemented on an event-driven approach. By default, the API Gateway destination is enabled to log the auditable events for all areas of management, such as APIs, policies, users, and so on. As a best practice, Software AG recommends you to enable audit logging for the required management areas in other supported destinations: Database, Digital Events, and Elasticsearch. This practice is especially important when you want to provide the audit log data to external sources for analytics and anomaly detections.

Configuring Audit Logs

You have to configure which events you need to audit for a destination so that API Gateway logs the auditable events data to the specific destination. You can configure API Gateway to log the auditable events data to the following destinations:

- API Gateway
- Database
- Digital Event Services (DES)
- Elasticsearch
- Custom Destination

Note:

You can configure custom destination to publish data to different components. For details on custom destinations, see “Custom Destination” on page 496.

The following events are available for audit log reports:

- API management
- Approval management
- Application management
- Alias management
- Team management
- Analytics management
- Group management
- Policy management
- Package management
- Plan management
- Promotion management
- User management

By default, all of the auditable events are logged into the API Gateway destination.

➤ **To configure audit logs**

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Destinations**.
3. Select the required destination to log the auditable events.
4. In the **Audit log data** section, select the required management areas to monitor, audit, and report the data.
5. Click **Save**.

Viewing Audit Logs

You can use the audit log reports to view the data of auditable events.

➤ **To view audit logs**

1. Expand the menu options icon  in the title bar, and select **Analytics**.

The dashboard displays the API Gateway-wide analytics based on the metrics monitored.

2. Select **Audit logs**.

3. In the drop-down list, choose the time interval in which you want to view the data of auditable events.

The available options are:

- Last 2 days
- Last 7 days
- Last 30 days
- Last 60 days
- Last 90 days
- Custom

4. If you select **Custom**, type the **From Date** and **To Date** to specify the time interval that best suits your needs.

5. Click **Apply filter** to filter the analytics based on the time interval chosen.

You can view logs for API Gateway auditable events in the **Audit logs** dashboard.

You can also download the API Gateway audit log in a text file and view the auditable events data.

Downloading Audit Logs

You can download the audit log reports to examine the data of auditable events.

➤ To download audit logs

1. Expand the menu options icon  in the title bar, and select **Analytics**.
The dashboard displays the API Gateway-wide analytics based on the metrics monitored.
2. Select **Audit logs**.
3. In the drop-down list, choose the time interval in which you want to view the data of auditable events.
4. If you select **Custom**, type the **From Date** and **To Date** to specify the time interval that best suits your needs.
5. Click **Download** to download and view the detailed report.

API Gateway generates a compressed file of the audit logs and downloads it to the default download folder configured in your browser..

The compressed file is named `auditlogs_N.zip`. The compressed file contains one or more simple text files, where each text file contains 10,000 audit log records.

API Gateway audit log are listed below.

Column	Detail
id	Unique identifier of the event that produced the audit record.
eventType	Type of event (audit log) that produced the record.
creationDate	Date and time the event entry was written to the log.
event_create_ts	Time in milliseconds when the event was generated in API Gateway.
objectType	Type of object (for example, User, API, Application, and so on) on which the event occurred.

Column	Detail
action	Type of action (for example, Create, Update, Delete, and so on) that was performed on the object.
object	Unique identifier of the API Gateway object on which the action was performed.
message	Message that describes the event that occurred.
user	Name of a user on the API Gateway instance that triggered the event.
sourceMachine	The host name of the machine on which the API Gateway instance is running.
clientIPAddress	IP address of the machine on which the API Gateway instance is running.
payload	The request payload defined for the event.
status	Current status of the event (Success or Failure).

System Settings

API Gateway user interface provides capability to configure system-level configuration parameters and communicate these changes across nodes in the cluster. You must have the API Gateway's manage system settings functional privilege assigned to configure the following settings:

- **Dashboard setting:** configure a port on which the dashboard runs.
- **System setting:** configure the API Gateway time out interval.
- **Logging setting:** modify the logging configuration.
- **SAML SSO:** configure the SAML settings for single sign-on.

Note:

For these configuration to take effect you have to restart API Gateway instance on every node after modifying any of these settings.

Modifying API Gateway Configuration Parameters

You can modify the port on which the dashboard runs, the API Gateway timeout interval, and the logging setting for API Gateway in this section.

Note:

For these modifications to take effect you must restart the API Gateway instance on each node.

➤ To modify API Gateway configuration parameters

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **System Settings > Configuration**.
3. To modify the dashboard details, provide the Kibana hostname and port number in the **Host** and **Port** fields in the Dashboard setting section.

You have to provide a port that is not in use in the system. If the provided port is already used by other system then the dashboard does not start.

4. To change the API Gateway timeout interval, type the required value in the **API Gateway timeout (minutes)** field in the System setting section.
5. To change the logging configuration, you can enable the logs based on the **Log level** by selecting the required log level.

The available log levels are **ERROR, WARN, INFO, DEBUG, TRACE**.

This changes the logging configuration for the UI logs.

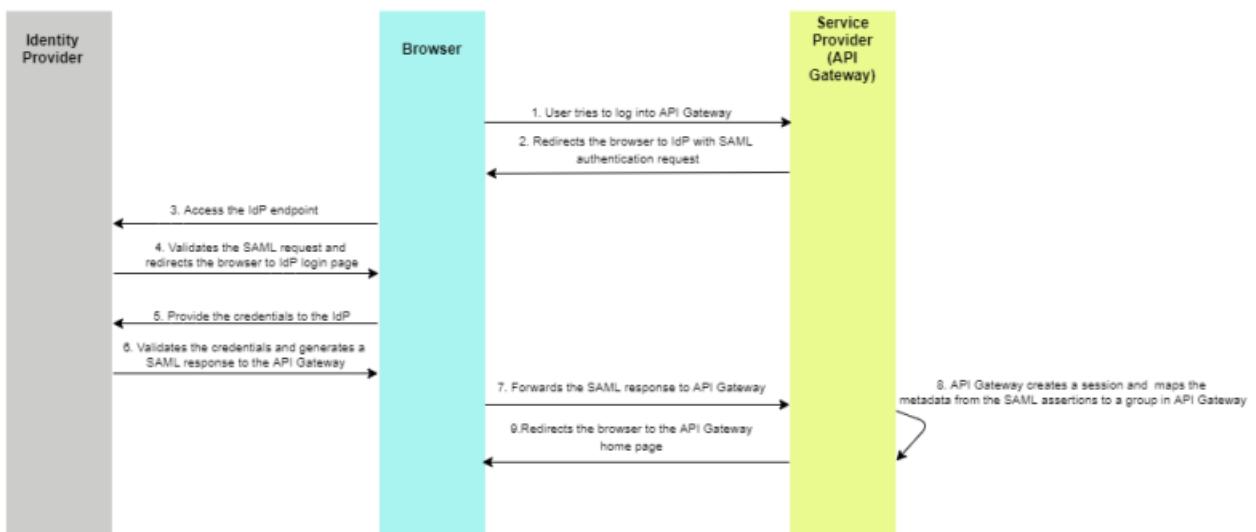
6. Select **Store log in server** to save the logs in the server. Select one of the following modes:
 - **Store at interval:** Provide the log storage interval. Stores the logs in the server at the specified time interval.
 - **Burst mode:** Store the logs for every event.
7. Click **Save**.

SAML SSO

Single sign-on (SSO) is a user authentication service that permits a user to use one set of login credentials to access multiple applications and service providers. In addition to the convenient factor, implementing SSO makes user logins more secure as it uses SAML protocol for communication.

Security Assertion Markup Language (SAML) is an open standard that allows identity providers to pass authorization credentials to service providers. SAML uses Extensible Markup Language (XML) for standardized communication between the identity providers and service providers. SAML provides a solution to allow your identity provider and service providers to exist separately from each other, which centralizes user management and provides access to services. In this case, API Gateway is the service provider.

- **Identity Provider (IdP)** - Performs authentication and passes the user's identity to the service provider for authorization.
- **Service Provider** - Trusts the identity provider and authorizes the given user to access the requested resource.



SAML Assertion

A SAML assertion is the XML document that the IdP sends to the service provider (that is API Gateway). It informs the API Gateway that a user has logged in. It also provides the necessary information for the API Gateway to confirm the user's identity and lists the groups to which the logged user belongs to.

In API Gateway, a user is created and a group gets associated to the created user based on the SAML assertion.

User Creation

In order to create a user you have to map the following attributes from the SAML assertion:

- Login ID
- First name
- Last name
- Email address

The attributes to be considered in the SAML assertion for the First Name, Last Name and Email address can be configured and value for the corresponding attributes are used in the creation of the user.

In the SAML assertion, the **NameID** element displays login ID of the user.

For example, as shown in the following sample, *alice* is the login ID.

```

<Subject>
  <NameID>alice</NameID>
  <SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
    <SubjectConfirmationData InResponseTo="a57d9j2i936ae5de2icdedg73jce390"
      NotOnOrAfter="2021-02-19T12:51:28.106Z"
    </SubjectConfirmationData>
  </SubjectConfirmation>
</Subject>
Recipient="https://localhost:9073/apigatewayui/saml/SSO"
  
```

```
        />
    </SubjectConfirmation>
</Subject>
```

The first name, last name, and email address attributes in the SAML assertion can be configured and their corresponding values are used in user creation.

Group Association

Once the user is created, the user needs to be assigned to a group in API Gateway.

In the SAML assertions, under the **AttributeStatement** element, if the **AttributeName** has any of the following values, then the **AttributeValue** element displays the group name to which the login

ID is associated in the IdP. This attribute value is used by API Gateway to map the user to the corresponding groups.

- <http://schemas.microsoft.com/ws/2008/06/identity/claims/role>
- <http://schemas.xmlsoap.org/claims/Group>

Example 1:

```
<AttributeStatement>
    <Attribute
        Name="http://schemas.microsoft.com/ws/2008/06/identity/claims/role">
        <AttributeValue>group1</AttributeValue>
    </Attribute>
</AttributeStatement>
```

In the example 1, based on the SAML assertion, the user is associated to the group called *group1* in the IdP. Later, API Gateway uses this value *group1* to map the user to the corresponding group.

Example 2 :

```
<AttributeStatement>
    <Attribute
        Name="http://schemas.microsoft.com/ws/2008/06/identity/claims/role">
        <AttributeValue>group2</AttributeValue>
    </Attribute>
</AttributeStatement>
```

In the example 2, based on the SAML assertion, the user is associated to the group called *group2* in the IdP. Later, API Gateway uses this value *group2* to map the user to the corresponding group.

Example 3:

```
<saml2:AttributeStatement xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
    <saml2:Attribute Name="http://schemas.xmlsoap.org/claims/Group"
        NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
        <saml2:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:type="xs:string">Everyone</saml2:AttributeValue>
        <saml2:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:type="xs:string">group1</saml2:AttributeValue>
```

```

<saml2:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xs:string">group2</saml2:AttributeValue>
</saml2:Attribute>
</saml2:AttributeStatement>

```

In the example 3, based on the SAML assertion, the user is associated to the groups called *Everyone*, *group1*, and *group2* in the IdP. Later, API Gateway uses these values *Everyone*, *group1*, and *group2* to map the user to the corresponding groups.

The SAML assertion is populated dynamically for each time when the user logs into API Gateway using SSO. If the user is mapped to a different group in the IdP or if the user is removed from the IdP during the subsequent login, then API Gateway maps the user to a group based on the SAML assertion of that subsequent session. This is to ensure that the mapping is always in synchronization between IdP and API Gateway.

How to enable SAML SSO in API Gateway?

This use case explains the steps involved in enabling SSO for API Gateway using the SAML protocol.

The use case starts when you configure the SAML settings for SSO in API Gateway and ends when you log into API Gateway using SSO.

Prerequisite

Ensure that you have:

- Manage user administration privilege.
- Provided the service provider SSO URL to the IdP administrator. Based on the service provider SSO URL, the IdP administrator generates the metadata.

Ensure the service provider SSO URL is in the given `http(s)://hostname:portnumber/apigatewayui/saml/SSO` format.

Note:

- For a standalone environment, replace the *hostname* and *portnumber* with the hostname and port number that is used to access the API Gateway.
- For a cluster environment, replace the *hostname* and *portnumber* with the hostname and port number of the load balancer.

- The metadata URL or file handy that was shared to you by the IdP administrator.
- Configured the keystore through which SSO communication is established between API Gateway and IdP.

➤ To configure SAML settings for single sign-on

1. Expand the menu options icon  in the title bar, and select **Administration**.

2. Select **System Settings > SAML SSO**.

The **Prerequisite** page appears.

3. Ensure that you have performed all the pre-requisite steps.

4. Click **Next**.

The **Connect** page appears.

5. Provide the SSO URL to which you want to redirect the browser in the **Service provider SSO URL** field.

Note:

By default, the **Service provider SSO URL** field is populated with the load balancer's web application URL that you have specified in the **Administration > General > Loadbalancer > Web application URLs** section. If you have not specified the load balancer's web

application URL, then the **Service provider SSO URL** field is populated with `http(s)://hostname : portnumber` through which you access the API Gateway instance.

6. Provide the API Gateway's entity ID in the **Service provider entity ID (URI)** field.
7. Provide the following information in the **Identity provider configuration URL** section:
 - Click **Import configuration from URL** to import the metadata file that the IdP administrator shared with you using the URL.
 - Provide the metadata file's URL in the **URL** field.
 - Click **Import configuration from file** to import the metadata file that the IdP administrator shared with you.
 - Provide the location where you have saved the metadata file in the **File** field.
8. Provide the following information in the **Keystore configuration** section:

Field	Description
Select a Keystore alias for signing/encryption	Select a keystore through which you can establish SSO communication between the IdP and API Gateway. Note: You can create the keystore from Administration > Security > Keystore/Truststore section. For details on how to configure keystore, see " Configuring Keystore Information " on page 549.
Use same key for signing and encryption	Click this if you use the same keystore alias for signing and encryption. Select the keystore that is used for both signing and encryption from the Key alias drop-down menu.
Use different keys for signing and encryption	Click this if you use two different key alias for signing and encryption. Select the keystore that is used for signing from the Sign key alias drop-down menu. Select the keystore that is used for encryption from the Encrypt key alias drop-down menu.

9. Provide the following information in the **Keystore configuration** section:
 - a. Select **Send signed SAML auth request**, if you want to send out the signed SAML authorization request to the Identity Provider (IdP).
 - b. Select **Require signed assertion from IDP** to receive a signed assertion from IdP.
10. Click **Save and try**.

The API Gateway Login page appears with **Log in with SSO** link.

The screenshot shows the webMethods API Gateway login interface. At the top, the "WEBMETHODS" logo is displayed above the text "API Gateway". Below this, a heading says "Log in to your account". There are two input fields: "Username" and "Password". A blue "Log in" button is positioned below the password field. Below the button, a horizontal line with the word "or" in the center separates it from a blue "Log in with SSO" link.

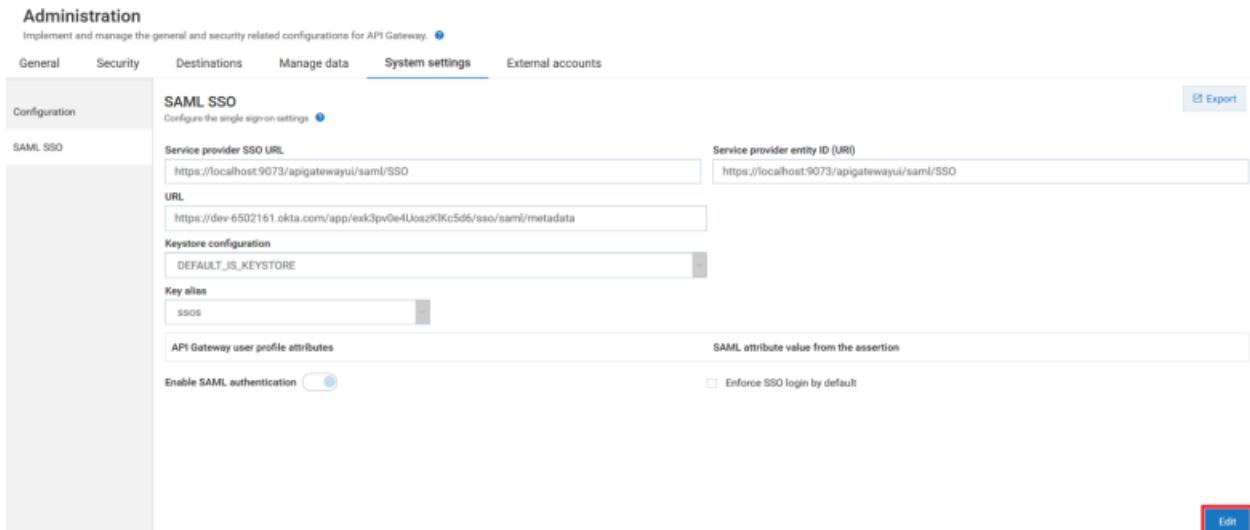
Click the **Log in with SSO** link to validate the SSO configuration. In this use case, as you have configured the OKTA as IdP, OKTA login page appears.

The screenshot shows the Okta sign-in page. The Okta logo is at the top. Below it, the "Sign In" heading is centered. There are two input fields: "Username" containing "testvsmeh@gmail.com" and "Password" containing masked text. A "Remember me" checkbox is followed by a "Sign In" button. At the bottom, there is a link "Need help signing in?".

Note:

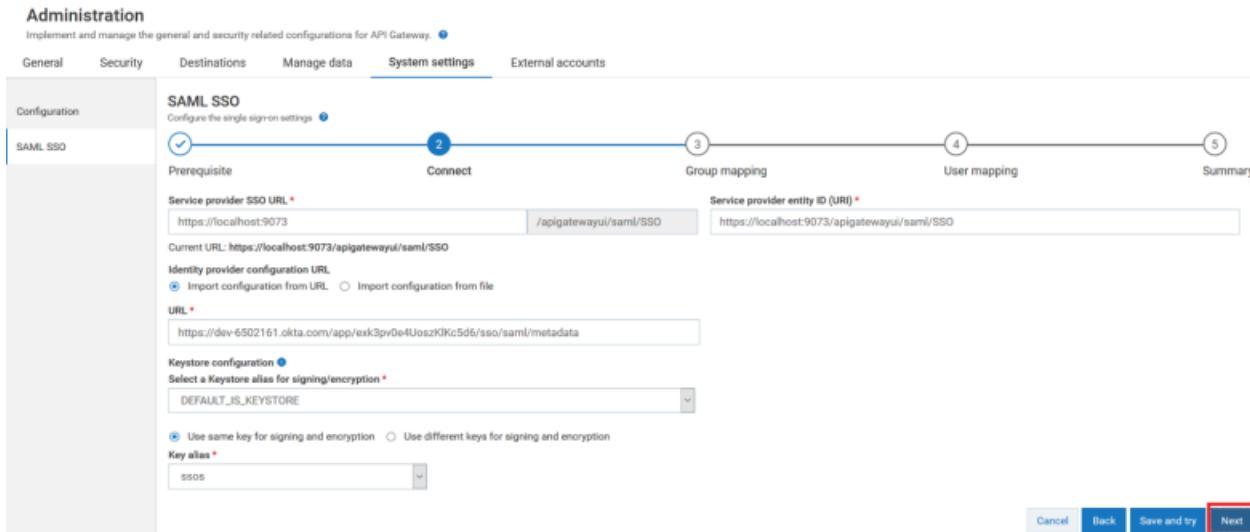
You can also skip this validation part and proceed with **Group mapping** page.

11. Once you successfully log into API Gateway using SSO, expand the menu options icon  in the title bar, and select **Administration > System Settings > SAML SSO**.
12. Click **Edit** to update the SAML SSO configuration with group mapping and user mapping details.



The screenshot shows the 'System settings' tab selected in the top navigation bar. Under the 'SAML SSO' section, there are fields for 'Service provider SSO URL' (set to https://localhost:9073/apigatewayui/saml/SSO) and 'Service provider entity ID (URI)' (set to https://localhost:9073/apigatewayui/saml/SSO). Below these are 'Keystore configuration' (set to DEFAULT_IS_KEYSTORE), 'Key alias' (set to ssos), and 'API Gateway user profile attributes'. A toggle switch for 'Enable SAML authentication' is turned on, and a checkbox for 'Enforce SSO login by default' is unchecked. At the bottom right, a blue 'Edit' button is highlighted with a red box.

The **Connect** page appears with the existing configuration.



The screenshot shows the 'System settings' tab selected. Under the 'SAML SSO' section, a progress bar indicates steps 1 through 5: 1. Prerequisite (checkmark), 2. Connect (highlighted with a red box), 3. Group mapping, 4. User mapping, and 5. Summary. The 'Prerequisite' step shows 'Service provider SSO URL' (https://localhost:9073) and 'Service provider entity ID (URI)' (https://localhost:9073/apigatewayui/saml/SSO). The 'Connect' step shows 'Current URL' (https://localhost:9073/apigatewayui/saml/SSO), 'Identity provider configuration URL' (radio button selected for 'Import configuration from URL' with URL https://dev-6502161.okta.com/app/exk3pv0e4UoszKIKc5d6/sso/saml/metadata), and 'Keystore configuration' (set to DEFAULT_IS_KEYSTORE). The 'Group mapping' and 'User mapping' steps are partially visible below. At the bottom right, buttons for 'Cancel', 'Back', 'Save and try', and 'Next' are shown, with 'Next' being highlighted with a red box.

13. Click **Next**.

The **Group mapping** page appears.

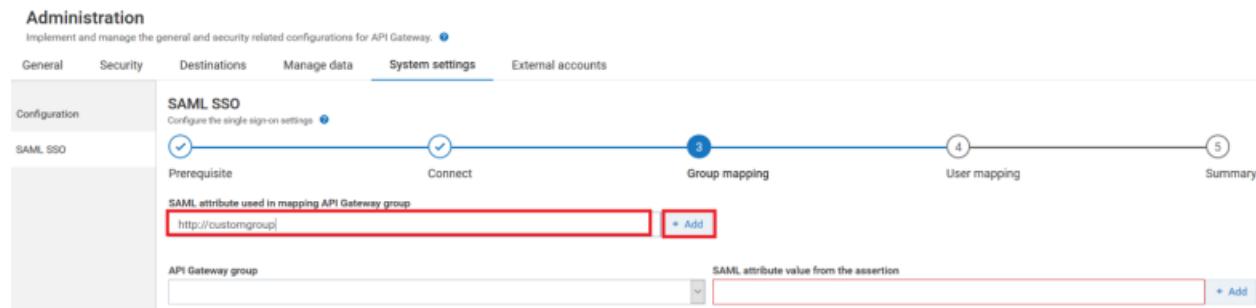
14. Provide the attribute name in the SAML assertion that you want to map to the API Gateway group in the **SAML attribute used in mapping API Gateway group** field.

By default, API Gateway supports the following two attribute names in the SAML assertion:

- <http://schemas.microsoft.com/ws/2008/06/identity/claims/role>
- <http://schemas.xmlsoap.org/claims/Group>

In addition, to the above two attribute names, if the SAML assertion has more attribute names, you can provide their attribute names in the **SAML attribute used in mapping API Gateway group** field and click **+ Add**.

For example: <http://customgroup> .

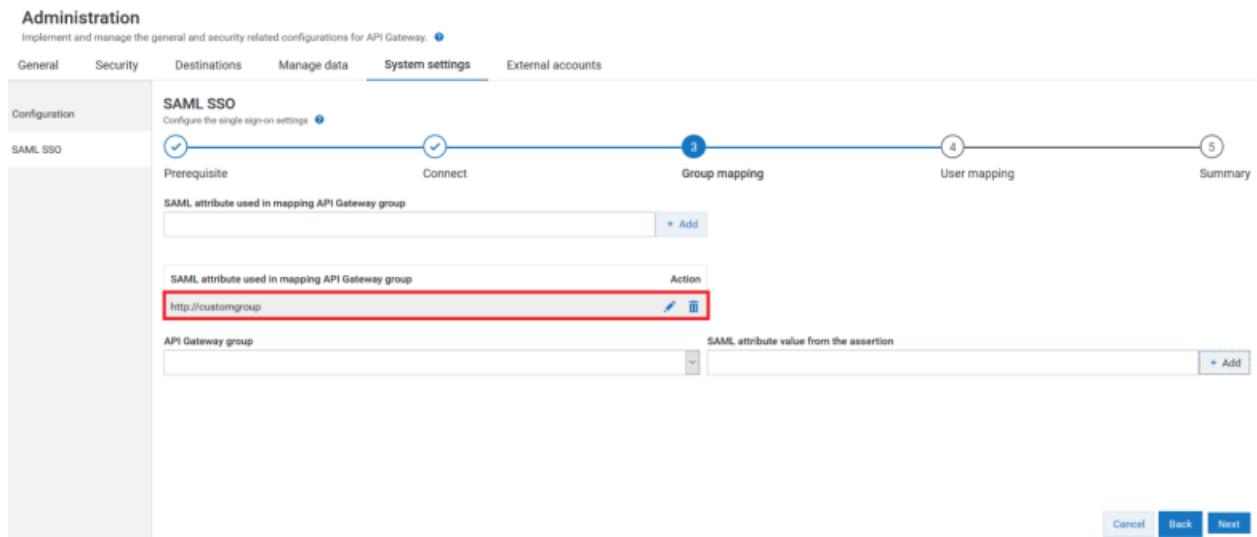


Sample SAML assertion for group mapping

```
<AttributeStatement>
    <saml2:Attribute Name="http://customgroup"
        NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
        <saml2:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:type="xs:string">ManageApplicationsGroup</saml2:AttributeValue>
    </saml2:Attribute>
</AttributeStatement>
```

In the sample SAML assertion, the user is associated to a new attribute name called `http://customgroup` and group called `ManageApplicationsGroup` in the IdP.

As shown in the image, the new attribute name `http://customgroup` specified in the SAML assertion is added to the API Gateway so that its corresponding attribute value `ManageApplicationsGroup` specified in the SAML assertion can be mapped to an API Gateway group.



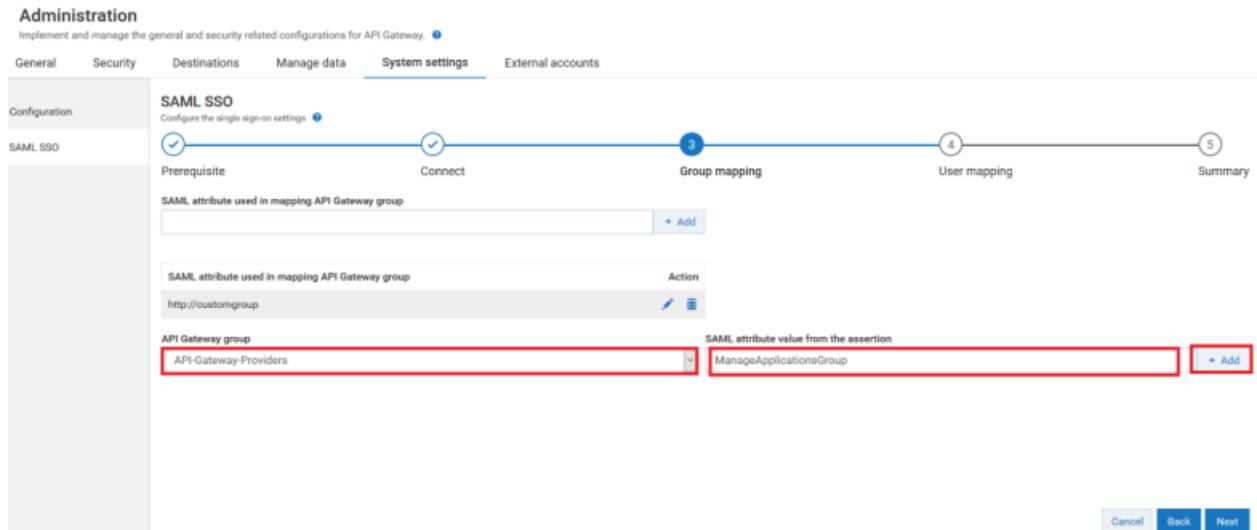
You can edit or delete the attribute name by clicking the or icons respectively. For details about what is SAML assertion, see "[SAML Assertion](#)" on page 521.

15. Select the API Gateway group to which you want to map the logged in user from the **API Gateway group** drop-down menu.

For example select API-Gateway-Providers.

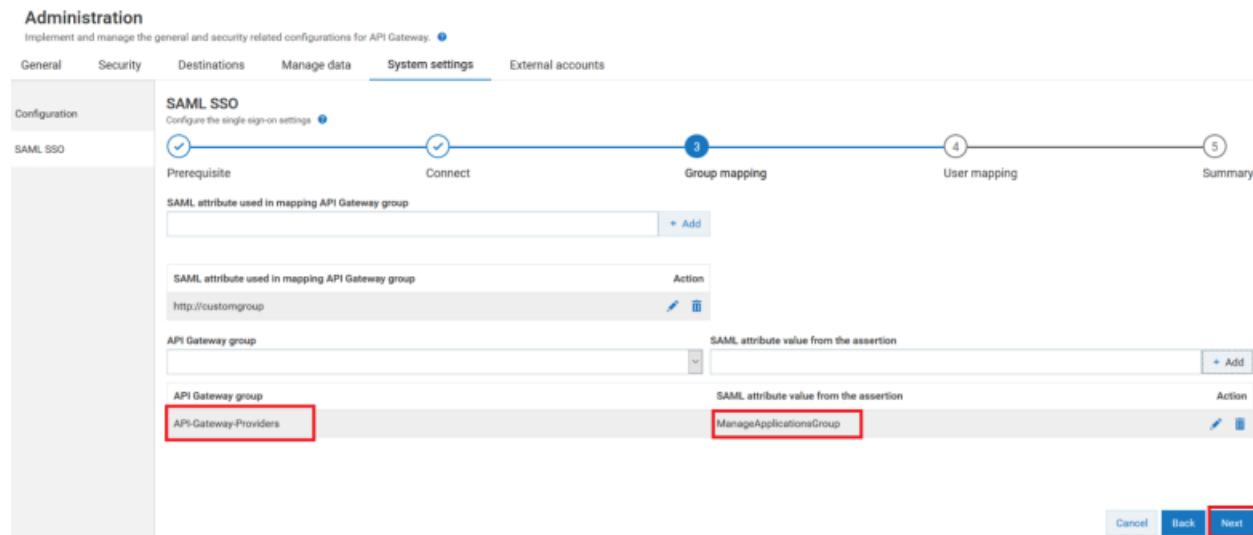
16. Provide the attribute value from the SAML assertion to which group the logged in user was mapped at the IdP in the **SAML attribute value from the assertion** field.

For example based on the sample SAML assertion, provide the user group as ManageApplicationsGroup



17. Click **+ Add** to map the IdP group of the logged in user specified in SAML assertion to the selected API Gateway group.

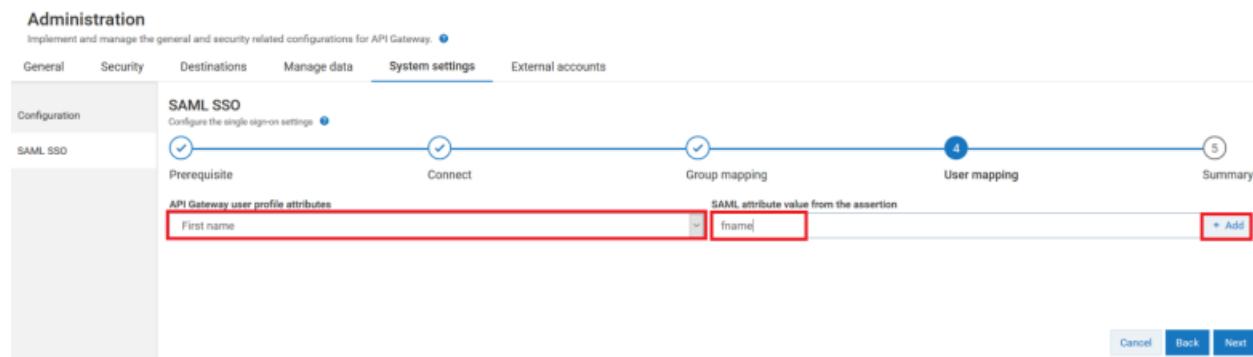
As shown in the image, the logged in user associated with the `ManageApplicationsGroup` in IdP is mapped to the `API-Gateway-Providers` group in API Gateway.



You can add multiple group mapping. You can edit or delete the group mapping by clicking the or icons respectively. For details on how to map the SAML assertion group to the API Gateway group, see "[Precedence in Group Mapping](#)" on page 532

18. Click **Next**.

The **User mapping** page appears.



19. Select the user profile attribute name in API Gateway from the **API Gateway user profile attributes** drop-down menu.

For example: Select the user profile attribute name as `First name`.

Sample SAML assertion for user mapping

```
<AttributeStatement>
    <saml2:Attribute Name=fname
        NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified"
    >
```

```

<saml2:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
                      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      xsi:type="xs:string"
                      >Joe</saml2:AttributeValue>
</saml2:Attribute>
</AttributeStatement>

```

In the sample SAML assertion, the user is associated to an attribute name called `fname` and attribute value as `Joe` in the IdP.

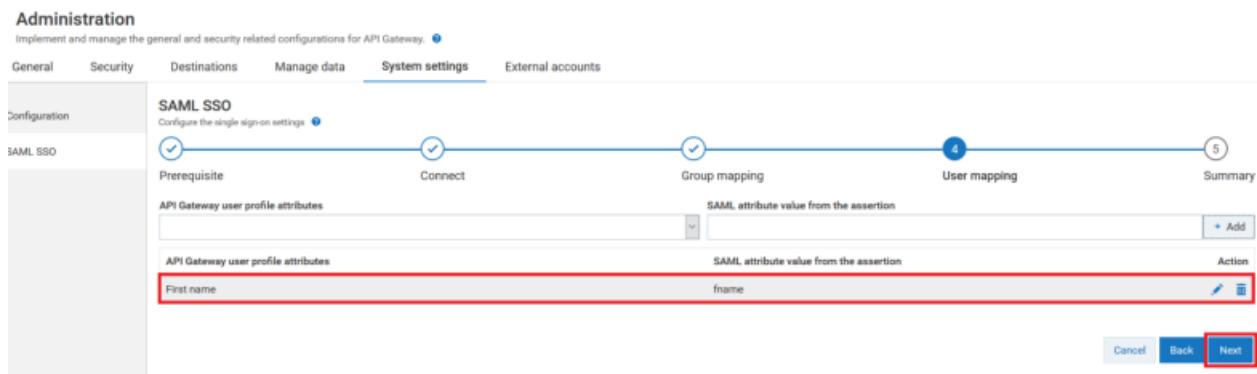
As shown in the image, the attribute name `fname` specified in the SAML assertion is mapped to user profile attribute called `First name` in API Gateway.

- Provide the user attribute name from the SAML assertion of the logged in user in the **SAML attribute value from the assertion** field.

For example: Provide the user profile attribute name as `fname`.

- Click **+ Add** to map the IdP user profile attribute value of the logged in user specified in SAML assertion to the selected API Gateway's user profile attribute.

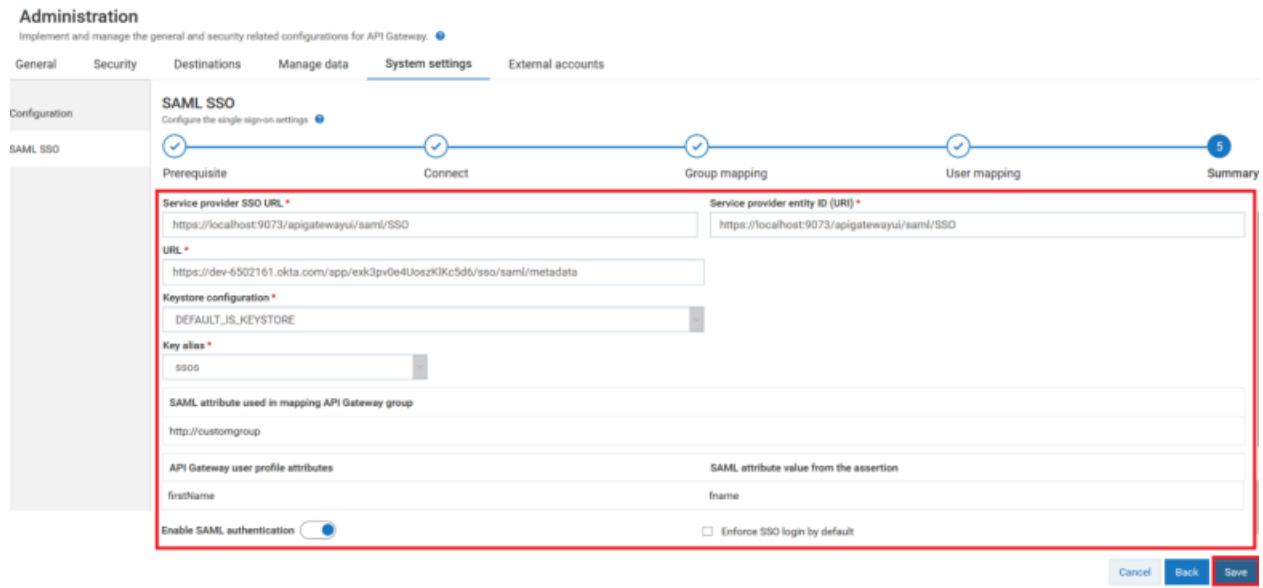
As shown in the image, the attribute name `fname` specified in the SAML assertion is mapped to user profile attribute called `First name` in API Gateway.



You can add multiple user profile attribute mapping. You can edit or delete the user mapping by clicking the or icons respectively.

- Click **Next**.

The **Summary** page appears.



23. Review the SSO configuration, group mapping, and user mapping information in the **Summary** page.

By default, the **Enable SAML authentication** toggle button is set on. If you want to disable the SAML authentication, set the toggle button off.

24. If you want to enable SSO login by default, select the **Enforce SSO login by default** check box.

If you have enabled SSO login by default, you will be directed to the SSO login page directly.

25. Click **Save** to save the group and user mapping details.

Precedence in Group Mapping

This use case explains the precedence involved in mapping the logged in SSO users to API Gateway groups based on the SAML assertion.

➤ Precedence order in mapping the IdP group in the SAML assertion to API Gateway group

1. API Gateway checks whether a group mapping exists in the SSO - Group Mapping configuration for the group in the SAML assertion. If the group mapping exists, then the user is automatically mapped to target group specified in the SSO.
2. If the group mapping does not exist in the SSO - Group Mapping configuration, then API Gateway checks whether the group exists in the API Gateway. If the group exists in the API Gateway, then the user is mapped to that group.
3. If there is no group specified in the SSO - Group Mapping configuration, and if the group does not exist in API Gateway, then the user is mapped to the default, *Everybody* group.

Troubleshooting Tips: SSO configuration

Issue	Symptom	Solution
org.opensaml.common.SAMLError: The audience URL in the Local entity is not the intended audience of the assertion in at least one AudienceRestriction.	The audience URL in the SAML assertion does not match with the Service provider identity in API Gateway.	Make sure the Service provider identity in API Gateway matches with the audience URL.
If you have enabled Enforce SSO login by default , and if you have provided incorrect information while configuring SAML SSO, you cannot update the SAML SSO configuration in API Gateway as you are redirected to the SSO Login page directly.		In such case, you can login into API Gateway using the <code>http(s)://hostname:portnumber/apigatewayui/login?usessouser=false</code> URL and update the SSO configuration with correct details.

Note:

If there is any other exception, check the `sag osgi.log` at `<SAGInstallDir>\profiles\IS_default\logs` directory to trouble shoot.

Limitation

When you log into API Gateway using SSO, both the IdP and API Gateway sessions are created. But when you log out from API Gateway, only the API Gateway session gets terminated, the IdP session gets terminated based on its session timeout configuration. API Gateway does not support Single Logout (SLO).

Configuring External Accounts

API Gateway provides capability to add and configure external accounts such as service registries, AWS accounts, Integration Server, and service mesh instances and communicate these changes across nodes in the cluster. An API Gateway administrator can add and remove the configured external accounts.

Service registry

The Service registry section allows you to configure service registries that API Gateway could use.

A service registry is essentially a catalog of services. Applications that expose services can register their services with one or more service registries. Applications that need to consume a service look up a service registry and obtain the address of an application server that provides the service. Using service registries with API Gateway adds resilience, scalability, and security to the application stack.

API Gateway uses service registries in the following ways:

- You can publish APIs defined in API Gateway to service registries. This enables other applications to use the service registry to dynamically locate an API Gateway instance that provides that API.
- You can set an API Gateway route to a service registry endpoint. This enables API Gateway to use the service registry to dynamically locate an application instance and route the request to it.

Service Registries supported by API Gateway

API Gateway currently supports the following service registries.

- *Eureka*

Eureka is a REST-based service for locating services for the purpose of load balancing and failover of middle-tier servers. It has been primarily designed for applications in the AWS cloud.

- *Service Consul*

Service Consul is a tool for discovering and configuring services in IT infrastructure.

AWS configuration

The AWS configuration section allows you to configure AWS aliases that are used in custom policy extension and custom destination sections.

Integration Server

The Integration server section allows you to configure the details of Integration Server instances required for API first implementation.

Service mesh

The Service mesh section allows you to configure the details of 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. For information on configuring service mesh, see the section *Configure API Gateway to Connect to a Service Mesh Environment* in the *webMethods API Gateway User's Guide*.

Adding a Service Registry

You must have the **Manage service registries** functional privilege assigned to perform this task.

You have to add and configure a service registry in API Gateway before you reference it in an API. In cluster deployments of API Gateway, you can add a service registry on any API Gateway instance—other API Gateway instances are synced automatically.

➤ To add and configure a service registry

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **External accounts > Service Registry**.
3. Click **Add service registry**.

Field	Description
Name	Name used by API Gateway for the instance of service registry that you are adding.
Description	Description of the instance of service registry.
Type	Type of service registry. Available values are: SERVICE_CONSUL and EUREKA .
Endpoint configuration	
Endpoint URI	The base URI for the service registry. This should include the IP address or the FQDN and the port on which the service registry accepts requests.
Service discovery path	The relative path of the service registry discovery service. API Gateway appends this path (and the service ID) to the Endpoint URI to generate a service discovery request for the service registry.
<p>Note: A service registry discovery service returns the address (IP address or FQDN) of an application instance for the requested service ID.</p>	
Service registration path	The relative path of the service registry registration service. API Gateway appends this path (and the service ID) to the Endpoint URI to generate a service registration request for the service registry.
<p>Note: API Gateway uses this service to register (publish) a service with the service registry.</p>	
Service de-registration path	The relative path of the service registry de-registration service. API Gateway appends this path (and the service ID) to the Endpoint URI to generate a service de-registration request for the service registry.
<p>Note: API Gateway uses this service to de-register (unpublish) a service with the service registry.</p>	
Connection timeout (seconds)	Specifies the time (in seconds) after which a connection attempt times out while communicating with the service registry.

Field	Description
	<p>The precedence of the Connection Timeout configuration is as follows:</p> <ul style="list-style-type: none"> 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 <code>pg.endpoint.connectionTimeout</code> property. d. If you do not specify any value for <code>pg.endpoint.connectionTimeout</code>, then API Gateway uses the default value of 30 seconds.
Read timeout (seconds)	<p>Specifies the time (in seconds) after which a socket read attempt times out while communicating with the service registry.</p> <p>The precedence of the Read Timeout configuration is as follows:</p> <ul style="list-style-type: none"> 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 <code>pg.endpoint.readTimeout</code> property. d. If you do not specify any value for <code>pg.endpoint.readTimeout</code>, then API Gateway uses the default value of 30 seconds.
SSL configuration	

Field	Description
Keystore alias	List of keystores that are configured in API Gateway. This is used when the service registry is SSL enabled. Lists all available keystores. If you have not configured an Integration Server keystore the list is empty.
Key alias	Lists all the private keys that are present in the selected keystore alias. This is used when the service registry is SSL enabled.
Truststore alias	A truststore contains the certificates that are trusted by API Gateway. If the service registry is SSL enabled its certificate should be added to the selected truststore.
Basic authentication details	
Username	The basic authentication username.
Password	The basic authentication password.
Other configuration	
Heartbeat interval	(This setting is applicable only to Eureka service registries.) The interval at which the API Gateway sends a heartbeat message to the Eureka server to renew its leases. Eureka expects clients, such as API Gateway, to renew the lease by sending heartbeats as per the heartbeat interval configured in the Eureka server.
Headers	
Key	An HTTP header key that is included in all the HTTP requests sent to the service registry.
Value	A value for the given HTTP header key.

4. Click **Add**.

The service registry is added to API Gateway.

Removing a Service Registry

- You cannot remove a service registry if any API has been published to it.
- You can remove only the non-default service registries that are not used by any API.
- You must have the **Manage service registries** functional privilege assigned to perform this task.

In cluster deployments of API Gateway, you can remove the service registry on any API Gateway instance—other API Gateway instances are synced automatically.

➤ To remove a service registry

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **External accounts > Service Registry**.
3. Click  in the row that has the service registry that you want to remove.
4. Click **Yes** to confirm.

The service registry is removed from API Gateway.

Configuring an AWS Alias

You need to configure an AWS alias when you:

- Use a custom extension that calls an AWS Lambda function from within a runtime policy enforcement flow of API Gateway API.
- Publish data to an AWS Lambda function using custom destination.

➤ To configure an AWS alias

1. On the title bar, expand the menu options icon  and select **Administration**.
2. Click **External accounts > AWS configuration**.
3. Click **Add new AWS account**.
4. In the Add AWS configuration section, provide the following information
 - a. **Name**. Provide the AWS alias name.
 - b. **Description**. *Optional*. Provide a description for the alias.
 - c. **Region**. Provide the AWS account region where the Lambda function is running or deployed.
For details on how to create an AWS Lambda function, see <https://docs.aws.amazon.com/lambda/latest/dg/getting-started.html>
 - d. **Access key ID**. Provide the access key ID of the AWS account where the Lambda function is running.

- e. **Secret access key.** Provide the secret access key of the AWS account where the Lambda function is running.
5. Click **Add**.

This creates the AWS alias and lists in the AWS configuration page.

Configuring Integration Server Instance for API Implementation

To implement an API to Integration Server, you must provide the details of the Integration Server instances in API Gateway.

➤ To configure an Integration Server instance

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Click **External accounts**.
3. Click **Integration Servers**.

The list of configured Integration server instances appears.

4. Click **Add new Integration Server**.

The options to add new Integration Server details appear.

5. Provide the following details:

Field	Description
Name	Name for the Integration Server instance being added.
Description	Description for the configuration.
Integration Server URL	URL of the Integration Server.
User name	User credentials required to access the Integration Server instance.
Password	Password required to access the Integration Server instance.
Keystore alias	The text identifier for the Integration Server keystore file. The keystore contains the private keys and certificates (including the associated public keys) of Integration Server.
Key alias	The alias for a specific key in the specified keystore.

Field	Description
Package name	Package name of the Integration Server instance in which the API must be implemented.
Folder name	Folder name of the Integration Server instance in which the API must be implemented.

The screenshot shows the 'Administration' section of the webMethods API Gateway. The left sidebar has 'Integration Servers' selected. The main area is titled 'External accounts' and contains fields for 'Connection Details': Name*, Description, Integration Server URL*, Username*, Password*, Keystore alias, and Key alias. Below this is a section for 'Default API Publish Configurations' with fields for Package name* and Folder name*. At the bottom is a checkbox for 'Import Swagger Based on Tags'.

- Select the following based on the API type, to perform the corresponding action.

Task selection	Description
REST API	<p>Import Swagger Based on Tags. Select the Import Swagger Based on Tags check box to group the operations under a REST V2 resource based on the tags.</p> <p>If you do not select this option, the operations are grouped by path.</p> <p>By default, the check box is selected.</p>
SOAP API	<p>Content model compliance. Select one of the following to indicate how strictly Integration Server enforces content model compliance when creating Integration Server document types from the XML Schema definition in the WSDL document.</p>

Task selection	Description
	<ul style="list-style-type: none"> ■ Strict. Generate the Integration Server document type only if Integration Server can represent the content models defined in the XML Schema definition correctly. Document type generation fails if Integration Server cannot accurately represent the content models in the source XML Schema definition. ■ Lax. When you select lax compliance, Integration Server will generate the Integration Server document type even if the content models in the XML schema definition cannot be represented correctly. ■ None. Generate an Integration Server document type that does not necessarily represent or maintain the content models in the source XML Schema definition.
	<p>Enable MTOM streaming for elements of type base64Binary. Select the Enable MTOM streaming for elements of type base64Binary check box if you want elements declared to be of type base64Binary in the WSDL or schema to be enabled for streaming of MTOM attachments.</p>
	<p>Enforce WS-I Basic Profile 1.1 compliance. Select the Enforce WS-I Basic Profile 1.1 compliance check box, if you want to validate all the WSD objects and properties against the WS-I requirements before creating the WSD.</p>
	<p>Validate schema using Xerces. Select the Validate schema using Xerces check box, if you want Integration Server to use the Xerces Java parser to validate any schema elements in the WSDL document or any referenced XML Schema definitions.</p>
GRAPHQL API	<p>Skip custom scalar types. Select the Skip custom scalar types check box, if you want omit the GraphQL API's custom scalar data types.</p>

7. To validate the connectivity of the specified Integration Server instance, click **Test**.

The connection with the given server is tested and a success message appears.

8. Click **Add**.

The server details are saved.

Configuration Types and Properties

This section describes the configuration types and parameters that you must configure for API Gateway.

The configuration types are broadly classified as web-app, API Gateway package-level, and Elastic search configurations.

webApp Configuration Properties

These properties are not cluster-aware and, hence, you must manually copy them to all the nodes.

General properties

Location:

SAG_Install_Dir/profiles/IS_instance_name/apigateway/config/uiconfiguration.properties

apigw.auth.priority

API Gateway supports both Form-based and SAML-based authentication. If both are enabled, this property decides the login page to be displayed, by default, when a user visits the login page `http://host:port/apigatewayui`. A user can go to a specific login page using:

- Form: `http://host:port/apigatewayui/login`
- SAML: `http://host:port/apigatewayui/saml/sso/login`

Possible values: Form, SAML.

Default value is Form.

apigw.auth.form.enabled

This property enables or disables Form-based authentication. If both SAML and Form are disabled, the value Form is retained by default.

Possible values: true, false.

Default value is true.

apigw.auth.form.redirect

If a protected resource is accessed and the Form-based authentication is enabled, user is redirected to this page.

Default value is /login.

apigw.is.base.url

Host where the IS package is hosted. *localhost* is replaced by the hostname that is resolved through localhost.

Note:

The port changes to the default port of the Integration Server instance irrespective of HTTP or HTTPS.

Default value is `http://localhost:port`. Here, *port* denotes the port that is configured at the time of installation.

apigw.user.lang.default

This property denotes the language to be used in the API Gateway UI.

Default value is en (English).

apigw.is.timeout

This property denotes the user session timeout value in minutes.

Default value is 90.

Kibana

Location :

SAG_Install_Dir/profiles/IS_instance_name/apigateway/config/uiconfiguration.properties

apigw.kibana.autostart

Specifies whether Kibana should be started as part of web-app.

Possible values: true, false.

Default value is true.

apigw.kibana.url

Denotes the URL where Kibana is running. *localhost* is replaced by the hostname that is resolved through localhost. The port and other configurations of the Kibana can be changed from SAG_Install_Dir/profiles/IS_instance_name/apigateway/dashboard/config/kibana.yml

Default value is `http://localhost:9405`

apigw.es.url

Denotes the URL where API Data Store (HTTP) is running. *localhost* is replaced by the hostname that is resolved through localhost.

Default value is `http://localhost:port`

port denotes the API Data Store HTTP port configured during installation.

Note:

If the configured host resolves to the host name of the localhost, the port changes to the HTTP port configured in the SAG_Install_Dir/InternalDataStore/config/elasticsearch.yml file.

kibana.process.stop.signal.number

Specifies the signal number to be used when stopping the Kibana process.

The default signal number is *SIGINT(2)*.

SIGINT(2) stops the Kibana process without producing a core dump. This property is applicable only for Linux Operating System. For information about the signals, see <https://www.linux.org/threads/kill-commands-and-signals.8881/>.

API Gateway Package Configuration Properties

API Gateway uses API Data Store (Elasticsearch) as its data repository. API Gateway starts the API Data Store instance, if configured, using the default configuration shipped and located at `SAG_Install_Dir/InternalDataStore/config/elasticsearch.yml`

Note:

To run API Data Store instances in a cluster, the `elasticsearch.yml` file must be updated on each instance. For additional details, see <https://www.elastic.co/guide/en/elasticsearch/guide/current/important-configuration-changes.html#important-configuration-changes>.

Location : `SAG_Install_Dir/IntegrationServer/instances/IS_Instance_Name/packages/WmAPIGateway/config/resources/elasticsearch/config.properties`

pg.gateway.elasticsearch.autostart

Denotes the flag to manage (start or stop) API Data Store as part of API Gateway. Set it to false if the start or stop of API Data Store is managed from outside the API Gateway.

Possible values: `true, false`.

Default value is `true`.

pg.gateway.elasticsearch.http.connectionTimeout

Denotes maximum time in milliseconds API Gateway waits for API Data Store to start and stop if autostart is set to `true`.

Default value is `10000`.

pg.gateway.elasticsearch.config.location

Denotes the location of the config file. If you have to use a different config file, mention the location of the config file here.

Default value is `SAG_Install_Dir/InternalDataStore/config/elasticsearch.yml`

Note:

- If the API Data Store hostname is same as localhost, then the system automatically modifies the value of `<prop key=cluster.name>` in `SAG_Install_Dir/IntegrationServer/instances/IS_Instance_Name/packages/WmAPIGateway/config/resources/beans/gateway-datastore.xml` to `cluster.name` property in the `elasticsearch.yml` file.
- If the API Data Store hostname is same as localhost, then the system automatically modifies the port value of `localhost:9340` in `SAG_Install_Dir/IntegrationServer/instances/IS_Instance_Name/packages/WmAPIGateway/config/resources/beans/gateway-datastore.xml` to `transport.port` property in the `elasticsearch.yml` file.
- Ensure that the `cluster.name` and `transport.port` properties are in synchronization if you encounter any errors.

Troubleshooting Tips: wmAPIGateway Package

Error appears when WmAPIGateway is added as Integration Server package dependency

The following error message appears when you add **wmAPIGateway** as a package dependency for an Integration Server package using Designer, and compile the package.

error: Bad service configuration file, or exception thrown while constructing Processor object:
io/sundr/codegen/processor/JavaGeneratingProcessor

Resolution:

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **General > Extended settings**.
3. Click **Show and hide keys**. This displays all the configurable parameters.
4. Search and enable the **watt.server.compile** parameter checkbox.
5. Add `-proc:none` to the **watt.server.compile** parameter.

For example,

```
watt.server.compile=/opt/SwAG/API/jvm/jvm/bin/javac -proc:none  
-classpath {0} -d {1} {2}
```

Extended settings

Configure extended settings 

Extended settings

These are the settings provided by API Gateway. Changes to these settings are propagated across the cluster.

`pg_JWT_isHTTPS`

false

Watt settings

Watt properties are the settings provided by Integration Server.

`watt.server.compile`

C:\Installations\jvm\jvm\bin\javac **-proc:none -classpath {0} -d {1} {2}**

6. Click **Save**.

3 Security Configuration

■ Overview of Security Configuration in API Gateway	548
■ Keystore and Truststore	548
■ Ports	555
■ Global IP Access Settings For Ports	577
■ SAML Issuer	583
■ Custom Assertions	587
■ Kerberos Settings	596
■ Master Password Management	598
■ OAuth, JWT, and OpenID Configuration	604
■ Threat Protection Policies	622
■ Securing API Gateway Communication using TLS	633
■ Troubleshooting Tips: Securing API Data Store (Elasticsearch)	659

Overview of Security Configuration in API Gateway

You must have the API Gateway's manage security configurations functional privilege assigned to perform the following tasks in the security configuration section of API Gateway:

- Configure the keystores and truststores required for incoming and outgoing message-level and transport-level security.
- Configure ports of API Gateway.
- Configure the SAML issuer to use in API Gateway outbound authentication to fetch the SAML token from the STS (Security Token Service).
- Configure the custom assertions to use in inbound authentication of API Gateway.
- Configure Kerberos settings.
- Manage master password.
- Configure JSON web token(JWT), OAuth, and OpenID authorization servers and third-party providers.

Keystore and Truststore

Keystores and truststores are secure files with industry-standard file formats. The keystore file stores the private keys and SSL certificates and the truststore file stores the trusted roots for the certificates.

A keystore file contains one or more pairs of a private key and signed certificate for its corresponding public key. The keystore should be strongly protected with a password, and stored (either on the file system or elsewhere) so that it is accessible only to administrators.

The truststore file functions as a database containing all the public keys for CAs within a specified trusted directory.

To enable the two-way SSL for inbound connections, you must add a valid, authorized X.509 certificate along with the private key in a keystore file and the certificate of the client or partner in the API Gateway truststore file. To enable two-way SSL for outbound connections you have to add the certificate of the native API to the API Gateway truststore file. These keystore and truststore files have to be referred to in the HTTPs port that is used to access the API Gateway service.

API Gateway has a sample keystore that contains self-signed certificates, which are located in `InstallDir\IntegrationServer\instances\default\packages\WmAPIGateway\config\resources\security`. The sample self-signed certificates are specific to localhost and hence Software AG recommends not to use them for configuring SSL communication with API Gateway in a production environment.

Note:

Any modifications to the keystore and truststore aliases in Integration Server do not reflect in API Gateway. Hence, Software AG recommends that you do not modify the aliases through

the Integration Server Administrator. On migration from 10.0 to 10.1, the existing configuration in 10.0 is migrated to the API Gateway UI.

Configuring Keystore Information

➤ To configure Keystore information

1. Expand the menu options icon  in the title bar, and select **Administration**.

2. Select **Security > Keystore/Truststore**.

A list of existing keystores and truststores and corresponding details are displayed.

3. In the Keystores section, click **Add keystore**.

4. In the Create keystore section, provide the following information:

Field	Description
Alias	A text identifier for the keystore file. The alias name can contain only letters, numbers, and underscores. It can not include a space, hyphen, and special characters.
Select file	Select a keystore file of the specified type using Browse. Select the required file and upload it.
Password	Password for the saved keystore file associated with this alias.
Type	The certificate file format of the keystore file, which by default is JKS for keystores. You can also use PKCS12 format for a keystore.
Description	Optional. A text description for the keystore alias.

5. Click **OK**.

All the key aliases in the uploaded file are listed.

6. Type a password for the required key alias.

7. Click **Save**.

The keystore is configured and the alias listed in the keystore alias table.

Note:

If a wrong password has been provided for the keystore or one of its aliases, API Gateway saves the keystore but it is not loaded. The keystore alias is displayed as the loaded icon with an X mark in the keystore listing.

Modifying Keystore Information

➤ To modify keystore information

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > Keystore/Truststore**.

A list of keystores and truststores and corresponding details are displayed.

3. Click the keystore alias to be updated.

The update keystore section is displayed.

4. Modify the fields as required.
5. Click **Save**.

The set of available aliases is displayed.

Note:

If the keystore file is not updated during the edit, then clicking **Save** closes the form. If a different keystore file is uploaded, then the list of aliases in the file is loaded and you are prompted to configure the passwords for the aliases.

6. Type a password for the alias to be configured.
7. Click **Save**.

The keystore is updated.

Deleting Keystore Information

Be careful while deleting the keystore information. If the keystore and one of its key aliases is configured in the keystore settings and the keystore gets deleted, then the configuration would have issues.

➤ To delete keystore information

1. Expand the menu options icon  in the title bar, and select **Administration**.

2. Select **Security > Keystore/Truststore**.

A list of keystores and truststores and corresponding details are displayed.

3. Click  in the action column of the keystore to be deleted.

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

The keystore is deleted from the list.

Configuring Truststore Information

➤ To configure truststore information

1. Expand the menu options icon  in the title bar, and select **Administration**.

2. Select **Security > Keystore/Truststore**.

A list of existing keystores and truststores and corresponding details are displayed.

3. In the Truststores section, click **Add truststore**.

4. In the Create truststore section, provide the following information:

Field	Description
Name	<p>A name for the truststore file.</p> <p>The alias name can contain only letters, numbers, and underscores. It can not include a space, hyphen, and special characters.</p>
Upload truststore file	<p>Select a truststore file of the specified type using Browse. Select the required file and upload it.</p> <p>Note: Supports only JKS file format.</p>
Password	Password that is used to protect the contents of the truststore.

Field	Description
	This password must have been defined at truststore creation time using a keystore utility.
	Make sure you have the truststore password available when managing its corresponding truststore alias.
Description	Optional. A text description for the truststore alias.

5. Click **Save**.

The truststore is configured and the alias listed in the truststore alias table.

Note:

If a wrong password has been entered for the truststore, API Gateway saves the truststore but it is not loaded. The truststore alias is displayed as the loaded icon with an X mark in the truststore listing.

Modifying Truststore Information

➤ To modify truststore information

1. Expand the menu options icon  in the title bar, and select **Administration**.

2. Select **Security > Keystore/Truststore**.

A list of keystores and truststores and corresponding details are displayed.

3. Click the truststore alias to be updated.

The update truststore section is displayed.

4. Modify the fields as required.

5. Click **Save**.

The truststore is updated.

Deleting Truststore Information

Be careful while deleting the truststore information. If the truststore settings and the truststore gets deleted, then the configuration would have issues.

➤ To delete keystore information

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > Keystore/Truststore**.

A list of keystores and truststores and corresponding details are displayed.

3. Click  in the action column of the truststore to be deleted.
4. Click **Yes** in the confirmation dialog.

The truststore is deleted from the list.

Configuring Keystore and Truststore Information for Inbound Messages

You might want to configure API Gateway to refer to a default keystore, truststore, or both, before deploying any SOAP message flows that require signature, encryption, X.509 authentication, and so on, as configured in the Inbound Auth - Message policy. The default keystore and truststore are that you want API Gateway to use for the incoming secured messages.

To configure keystore and truststore settings for inbound messages

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > Keystore/Truststore**.

A list of existing keystores and truststores loaded during startup, and those created in API Gateway and the corresponding details appears.

3. To configure API Gateway's default keystore and truststore alias for incoming secured messages, provide the following information in the Configure keystore and truststore settings for inbound messages section:

Field	Description
Keystore alias	Select a keystore that API Gateway uses for incoming message-level security. Lists all available keystores. If you have not configured any keystore, the list is empty.
Key alias (signing)	Select the alias for the private key to sign the outgoing response from API Gateway to the original client.

Field	Description
	This alias value validates the inbound requests to API Gateway and signs the outgoing response from API Gateway to the original client. This field is auto-populated based on the selected keystore alias. It lists all the aliases available in the chosen keystore. If there are no configured keystores, this field is empty.
Truststore alias	The alias for the truststore that contains the list of CA certificates that API Gateway uses to validate the trust relationship with the client.

- Click **Save**.

Post-requisites

While securing the SOAP APIs using WS-Security policies, perform the following:

- Restart the server after configuring keystore and truststore information for the configuration to take effect.
- Deactivate the APIs that have Inbound Auth - Message policy enforced.
- Update the keystore and truststore configuration.
- Activate the APIs that were deactivated.

Configuring Keystore and Truststore Information for Outbound Connections

You might want to configure API Gateway to refer to a default truststore that you want API Gateway to use for securing outgoing SSL connections. The keystore and key alias can be configured for outgoing two-way SSL connections. During the SSL handshake between API Gateway and the native API, the server certificate, which is sent by the native API, has to be validated against a truststore in API Gateway.

➤ To configure keystore and truststore settings for outbound secured connections

- Expand the menu options icon  in the title bar, and select **Administration**.
- Select **Security > Keystore/Truststore**.

A list of existing keystores and truststores loaded during startup, and those created in API Gateway and the corresponding details appears.

- To configure API Gateway's default keystore and truststore alias for outgoing secured connections, provide the following information in the Configure keystore and truststore settings for outbound connections section:

Field	Description
Keystore alias	Select a keystore that API Gateway uses for outgoing secured connections. Lists all available keystores. If you have not configured any keystore, the list is empty.
Key alias	Select the alias for the private key for an outbound connection from API Gateway to the native API. This field is auto-populated based on the selected keystore alias. It lists all the aliases available in the chosen keystore. If there are no configured keystores, this field is empty.
Truststore alias	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.

4. Click **Save**.

Ports

API Gateway listens for requests on ports that you specify. Each port is associated with a specific type of protocol, HTTP or HTTPS. In addition to these port types, API Gateway also provides three ports; API Gateway external port, API Gateway internal listener port, and the WebSocket listener port.

You can specify one or more HTTP or HTTPS ports on which API Gateway and the deployed APIs are available for consumption. API Gateway, by default, is available on the primary HTTP port. The primary HTTP port is the port specified on the Integration Server's **Security > Ports** page.

If your API Gateway is behind an internal firewall and is not allowed to accept communications from external clients through the DMZ, then you can configure the API Gateway instance in DMZ with an external port to listen to requests from external clients and using reverse invoke route them to the internal servers. The API Gateway internal listener port or the WebSocket listener port pulls the requests from the registration port of API Gateway in DMZ thus safeguarding from any malicious attacks.

External clients send requests to API Gateway. API Gateway external port listens to this client information from each request and evaluates the request against any API Gateway rules that have been defined. It then passes requests that have not violated a rule to the API Gateway internal port or the WebSocket listener port . These listener ports process the requests and send the responses to API Gateway, which then passes the responses back to the client.

The page displays the following information about the configured ports:

Field	Description
Ports	<p>Specifies the port number.</p> <p>Click on the port number to edit the port configuration.</p> <p>API Gateway does not allow you to update an active port as long as the port is enabled. If you want to update an active port, API Gateway first automatically disables the port, then updates the configured details. On successful updation, API Gateway enables the port back. In case, when updating the port, if an error occurs, API Gateway displays the error message that stops from updating the active port.</p>
Alias	Specifies the port alias that can be used across to refer to the corresponding port.
Protocol	Specifies the protocol used by the port to communicate.
Type	Specifies the type of port.
Enabled	Specifies the status of the port; whether it is enabled or disabled.
Accessmode	Allows to configure whether the port must allow or deny access to ESB services and folders, by default. For information on how to configure the access mode for a port, see “ Configuring Access Mode for a Port ” on page 575.
IP Accessmode	Allows to configure whether the port must allow or deny access to the external hosts, or must follow global configuration for allowing or denying external hosts. For information on how to configure the IP access mode of a port, see “ Configuring IP Access Mode for a Port ” on page 572.
Primary port	<p>Specifies whether the port is used as a primary port.</p> <p> specifies that it is set as a primary port and depicts it is not set as a primary port. Only the HTTP and HTTPS ports can be set as primary ports once they are enabled.</p> <p>You can click for a port to set it as a primary port.</p> <div style="background-color: #f0f0f0; padding: 5px;"> <p>Note: You can not disable or delete the primary port. Also, you can not modify the primary port details as long as the port is set as primary port.</p> </div>
Description	Provides a short description of the port.
Action	Specifies the actions that can be performed on the port.

Note:

Any modifications to the ports in Integration Server may not reflect in API Gateway UI. Hence, Software AG recommends that you do not configure or modify the ports through Integration Server Administrator UI.

Adding an HTTP Port

The HTTP port enables API Gateway to authenticate the client and server securely and exchange the data. In addition, you can configure the type of client authentication that you want the server to perform. Client authentication allows you to verify the identity of the client.

➤ To add an HTTP port

1. Expand the menu options icon  in the title bar, and select **Administration**.

2. Select **Security > Ports**.

The ports page lists all the ports configured with API Gateway, if any.

3. Click **Add ports**.

4. Select the type of port as **HTTP** and click **Add**.

5. Provide the following information:

Field	Description
HTTP listener configuration. Provide the following details to configure the HTTP listener set up.	
Port	<p>Specify the number you want to use for the port.</p> <p>Select a number that is not already in use on this host machine.</p>
Alias	<p>Specifies an alias for the port that is unique for this API Gateway.</p> <p>An alias must be between 1 and 255 characters in length and include one or more of the following: letters (a -z, A-Z), numbers (0-9), underscore (_), period (.), and hyphen (-).</p>
Description (optional) Provide a description of the port.	
Bind address (optional)	<p>Specifies the IP address to which to bind this port.</p> <p>Specify a bind address if your machine has multiple IP addresses and you want the port to use this specific address. If you do not specify a bind address, API Gateway picks one for you.</p>
Backlog	<p>Specifies the number of requests that can remain in the queue for an enabled port before API Gateway begins rejecting requests.</p> <p>The default is 200. The maximum value is 65535.</p>

Field	Description
Keep alive timeout	Specifies when to close the connection if the server has not received a request from the client within this timeout value (in milliseconds) or when to close the connection if the client has explicitly placed a close request with the server.
Private threadpool configuration.	Specifies whether to create a private thread pool for this port or use the common thread pool.
Enable	Select to enable the private threadpool configuration for this port.
Threadpool min	Specifies the minimum number of threads for this private threadpool. The default value is 1.
Threadpool max	Specifies the maximum number of threads for this private thread pool. The default value is 5.
Thread priority	Specifies the Java thread priority. The default value is 5.
Security configuration	Provide the following details to configure security parameters.
Client authentication	<p>Specifies the type of client authentication you want API Gateway to perform for requests that arrive on this HTTPS port.</p> <p>Select one of the following:</p> <ul style="list-style-type: none"> ■ Username/Password. API Gateway does not request client certificates. The server looks for user and password information in the header of requests coming from an external client. ■ Digest. API Gateway uses password digest to authenticate all requests. If the client does not provide the authentication information, API Gateway returns an HTTP WWW-Authenticate header with digest scheme to the client requesting for authentication information. If the client provides the required authentication information, API Gateway verifies and validates the request. ■ Request Kerberos Ticket. API Gateway looks for a Kerberos ticket in the HTTP Authorization header using the Negotiate authentication scheme. If it does not find the ticket, API Gateway uses username and password for basic authentication. If the client does not provide any authentication information, API Gateway returns an HTTP WWW-Authenticate header with negotiate scheme to the client requesting for authentication information. If the client provides the required authentication information, API Gateway verifies and validates the request. ■ Require Kerberos Ticket. API Gateway looks for a Kerberos ticket in the HTTP Authorization header using the Negotiate authentication scheme. If it does not find the ticket, API Gateway

Field	Description
	fails the authentication. If the client does not provide any authentication information, API Gateway returns an HTTP WWW-Authenticate header with negotiate scheme to the client requesting for authentication information. If the client provides the required authentication information, API Gateway verifies and validates the request.
	You have to enable Kerberos by providing the following Kerberos properties with details that are used for handling service requests that come with a Kerberos ticket:

■ **JAAS context.** Specify the custom JAAS context used for Kerberos authentication.

■ **Principal.** Specify the name of the principal to use for Kerberos authentication.

■ **Principal password.** Specify the password for the principal to use to authenticate the principal to the KDC.

■ **Retype principal password.** Retype the principal password.

■ **Service principal name.** Specify the name of the principal used with the service that the Kerberos client wants to access.

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.

6. Click **Add**.

The port is created and is listed in the ports table.

Important:

The default access mode of the port is set to *Allow by default*. This implies that the port allows connections to all ESB services and folders, which in turn, increases the risk of exposing all enterprise assets hosted in internal Integration Server. Also, the risk is higher when the IS assets are secured by Anonymous Access Control Lists (ACL) or if the installation is exposed to the public internet. Hence, to avoid any potential security risk, you can set the access mode of the port to *Deny by default* before enabling it. When you change the access mode, you add the required services and folders to the *Allow* list. For more information, see “[Configuring Access Mode for a Port](#)” on page 575.

Also, the global IP access mode will be applied to the newly created HTTP ports. You can modify the IP access mode as per your requirement. For information on modifying IP access mode of ports, see “[Configuring IP Access Mode for a Port](#)” on page 572.

7. Click the ✘ icon in the **Enabled** column next to the port to enable the port.

The port is enabled and a success message appears.

Adding an HTTPS Port

The HTTPS port enables API Gateway to authenticate the client and server securely and encrypt the data exchanged. By default, the HTTPS listener uses the certificates for the default SSL key. In addition, you can configure the type of client authentication that you want the server to perform. Client authentication allows you to verify the identity of the client.

➤ To add an HTTPS port

1. Expand the menu options icon  in the title bar, and select **Administration**.
 2. Select **Security > Ports**.
- The ports page lists all the ports configured with API Gateway, if any.
3. Click **Add Ports**.
 4. Select the type of port as **HTTPS** and click **Add**.
 5. Provide the following information:

Field	Description
HTTP listener configuration. Provide the following details to configure the HTTP listener set up.	
Port	Specify the number you want to use for the port. Select a number that is not already in use on this host machine.
Alias	Specifies an alias for the port that is unique for this API Gateway. An alias must be between 1 and 255 characters in length and include one or more of the following: letters (a -z, A-Z), numbers (0-9), underscore (_), period (.), and hyphen (-).
Description (optional)	Provide a description of the port.
Bind address (optional)	Specifies the IP address to which to bind this port. Specify a bind address if your machine has multiple IP addresses and you want the port to use this specific address. If you do not specify a bind address, API Gateway picks one for you.

Field	Description
Backlog	Specifies the number of requests that can remain in the queue for an enabled port before API Gateway begins rejecting requests. The default is 200. The maximum value is 65535.
Keep alive timeout	Specifies when to close the connection if the server has not received a request from the client within this timeout value (in milliseconds) or when to close the connection if the client has explicitly placed a close request with the server.
Private threadpool configuration.	Specifies whether to create a private thread pool for this port or use the common thread pool.
Enable	Select to enable the private threadpool configuration for this port.
Threadpool min	Specifies the minimum number of threads for this private threadpool. The default is 1.
Threadpool max	Specifies the maximum number of threads for this private thread pool. The default is 5.
Thread priority	Specifies the Java thread priority. The default is 5.
Security configuration	. Provide the following details to configure security parameters.
Client authentication	<p>Specifies the type of client authentication you want API Gateway to perform for requests that arrive on this HTTPS port.</p> <p>Select one of the following:</p> <ul style="list-style-type: none"> ■ Username/Password. API Gateway does not request client certificates. The server looks for user and password information in the header of requests coming from an external client. ■ Digest. API Gateway uses password digest to authenticate all requests. If the client does not provide the authentication information, API Gateway returns an HTTP WWW-Authenticate header with digest scheme to the client requesting for authentication information. If the client provides the required authentication information, API Gateway verifies and validates the request. ■ Request Kerberos Ticket. API Gateway looks for a Kerberos ticket in the HTTPS Authorization header using the Negotiate authentication scheme. If it does not find the ticket, API Gateway uses username and password for basic authentication. If the client does not provide any authentication information, API Gateway returns an HTTP WWW-Authenticate header with negotiate scheme to the client requesting for authentication information.

Field	Description
	<p>If the client provides the required authentication information, API Gateway verifies and validates the request.</p> <ul style="list-style-type: none"> ■ Require Kerberos Ticket. API Gateway looks for a Kerberos ticket in the HTTPS Authorization header using the Negotiate authentication scheme. If it does not find the ticket, API Gateway fails the authentication. If the client does not provide any authentication information, API Gateway returns an HTTP WWW-Authenticate header with negotiate scheme to the client requesting for authentication information. If the client provides the required authentication information, API Gateway verifies and validates the request.
	<p>You have to enable Kerberos by providing the following Kerberos properties with details that are used for handling service requests that come with a Kerberos ticket:</p> <ul style="list-style-type: none"> ■ JAAS context. Specify the custom JAAS context used for Kerberos authentication. ■ Principal. Specify the name of the principal to use for Kerberos authentication. ■ Principal password. Specify the password for the principal to use to authenticate the principal to the KDC. ■ Retype principal password. Retype the principal password. ■ Service principal name. Specify the name of the principal used with the service that the Kerberos client wants to access. API Gateway supports the <i>username</i> 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. ■ Request Client Certificate. API Gateway requests client certificates for all requests. If the client does not provide a certificate, the server prompts the client for a userid and password. The server checks whether the certificate exactly matches a client certificate on file and is signed by a trusted authority. If so, the client is logged in as the user to which the certificate is mapped in API Gateway. If not, the client request fails, unless central user management is configured. ■ Require Client Certificate. API Gateway requires client certificates for all requests. The server behaves as described for Request Client Certificates, except that the client must always provide a certificate.

Field	Description
	<ul style="list-style-type: none"> ■ Use Identity Provider. API Gateway uses an OpenID Provider to authenticate requests. API Gateway redirects all requests sent to this port to the OpenID Provider specified in Identity Provider.
Listener specific credentials. Provide the following details to configure listener specific credentials.	
Keystore alias	<p>Specifies a user-specified, text identifier for an API Gateway keystore.</p> <p>The alias points to a repository of private keys and their associated certificates. Although each listener points to one keystore, there can be multiple keys and their certificates in the same keystore, and more than one listener can use the same keystore alias.</p>
Key alias (signing)	Specifies the private key of keystore.
Truststore alias	<p>Specifies the public certificates of truststore.</p> <p>The alias points to a repository of public certificates.</p>

6. Click **Add**.

The port is created and is listed in the ports table.

Important:

The default access mode of the port is set to *Allow by default*. This implies that the port allows connections to all ESB services and folders. Users must note that this setting allows access to all enterprise assets hosted in internal Integration Server. There is a potential security risk for the IS assets that are secured by Anonymous Access Control Lists (ACL) or if the installation is exposed to the public internet. Hence, you can set the access mode of the port to *Deny by default* before enabling it. When you change the access mode, you add the required services and folders to the *Allow* list. For more information, see “[Configuring Access Mode for a Port](#)” on page 575.

Also, the global IP access mode will be applied to the newly created HTTPS ports. You can modify the IP access mode as per your requirement. For information on modifying IP access mode of ports, see “[Configuring IP Access Mode for a Port](#)” on page 572.

7. Click the  icon in the **Enabled** column next to the port to enable the port.

The port is enabled and a success message appears.

Adding an API Gateway External Port

The API Gateway external and registration ports work as a pair. One port is not functional without the other.

➤ To add an API Gateway external port

1. Expand the menu options icon  in the title bar, and select **Administration**.

2. Select **Security > Ports**.

The ports page lists all the ports configured with API Gateway, if any.

3. Click **Add Ports**.

4. Select the type of port as **API Gateway external** and click **Add**.

5. Provide the following information:

Field	Description
API Gateway external listener configuration. Provide the following details to configure the HTTP listener set up.	
External port	<p>Specifies the port number you want to use for the external port.</p> <p>Use a number that is not already in use. This is the port that clients connect to through your outer firewall.</p>
Alias	<p>Specifies an alias for the port.</p> <p>An alias must be between 1 and 255 characters in length and include one or more of the following: letters (a -z, A-Z), numbers (0-9), underscore (_), period (.), and hyphen (-).</p>
Description (optional) A description of the port.	
Protocol	<p>Specifies the protocol to use for this port: HTTP or HTTPS.</p> <p>If you select HTTPS, additional security and credential boxes appear for which you have to provide the required values.</p>
Bind address (optional)	<p>Specifies the IP address to which to bind this port.</p> <p>Specify a bind address if your machine has multiple IP addresses and you want the port to use this specific address. If you do not specify a bind address, API Gateway picks one for you.</p>
Backlog	<p>Specifies the number of requests that can remain in the queue for an enabled port before API Gateway begins rejecting requests.</p> <p>The default is 200. The maximum value is 65535.</p>
Keep alive timeout	Specifies when to close the connection if the server has not received a request from the client within this timeout value (in milliseconds)

Field	Description
	or when to close the connection if the client has explicitly placed a close request with the server. The default value is 20000ms.
Private threadpool configuration.	Specifies whether to create a private thread pool for this port or use the common thread pool.
Enable	Select to enable the private threadpool configuration for this port.
Threadpool min	Specifies the minimum number of threads for this private threadpool. The default value is 1.
Threadpool max	Specifies the maximum number of threads for this private thread pool. The default value is 5.
Thread priority	Specifies the Java thread priority. The default value is 5.

Security configuration. Provide the following details to configure security parameters.

Client authentication For the external port, specify the type of client authentication required.

Select one of the following:

- **Username/Password**. API Gateway does not request client certificates. The server looks for user and password information in the header of requests coming from an external client.
- **Digest**. API Gateway uses password digest authentication. API Gateway looks for password digest information in the header of requests coming from an external client.
- **Request Kerberos ticket**. API Gateway looks for a Kerberos ticket in the HTTP Authorization header using the Negotiate authentication scheme. If it does not find the ticket, API Gateway uses username and password for basic authentication. If the client does not provide any authentication information, API Gateway returns an HTTP WWW-Authenticate header with negotiate scheme to the client requesting for authentication information. If the client provides the required authentication information, API Gateway verifies and validates the request.
- **Require Kerberos ticket**. API Gateway looks for a Kerberos ticket in the HTTP Authorization header using the Negotiate authentication scheme. If it does not find the ticket, API Gateway fails the authentication. If the client does not provide any authentication information, API Gateway returns an HTTP WWW-Authenticate header with negotiate scheme to the client requesting for authentication information. If the client provides

Field	Description
	<p>the required authentication information, API Gateway verifies and validates the request.</p> <ul style="list-style-type: none"> ■ Request client certificate. <i>This option appears only if you select the HTTPS option in the Protocol field of the API Gateway external listener configuration section.</i> API Gateway requests client certificates for all requests. If the client does not provide a certificate, the server prompts the client for a userid and password. The server checks whether the certificate exactly matches a client certificate on file and is signed by a trusted authority. If so, the client is logged in as the user to which the certificate is mapped in API Gateway. If not, the client request fails, unless central user management is configured. ■ Require client certificate. <i>This option appears only if you select the HTTPS option in the Protocol field of the API Gateway external listener configuration section.</i> API Gateway requires client certificates for all requests. The server behaves as described for Request Client Certificates, except that the client must always provide a certificate.
	<p>You have to enable Kerberos by providing the following Kerberos properties with details that are used for handling service requests that come with a Kerberos ticket:</p> <ul style="list-style-type: none"> ■ JAAS context. Specify the custom JAAS context used for Kerberos authentication. ■ Principal. Specify the name of the principal to use for Kerberos authentication. ■ Principal password. Specify the password for the principal that is used to authenticate the principal to the KDC. ■ Retype principal password. Retype the principal password. ■ Service principal name. Specify the name of the principal used with the service that the Kerberos client wants to access. API Gateway supports the <i>username</i> format for Service Principal Names (SPNs). This format represents the principal name as a named user defined in the LDAP or central user directory used for authentication to the KDC.
	<p>Listener specific credentials (optional). <i>This section appears only if you select the HTTPS option in the Protocol field of the API Gateway external listener configuration section.</i> Provide the following details to configure listener specific credentials.</p>

Keystore alias Specifies a user-specified, text identifier for an API Gateway keystore.

Field	Description
	The alias points to a repository of private keys and their associated certificates. Although each listener points to one keystore, there can be multiple keys and their certificates in the same keystore, and more than one listener can use the same keystore alias.
Key alias (signing)	Specifies the private key of keystore.
Truststore alias	Specifies the public certificates of truststore. The alias points to a repository of public certificates. Provide the client or server truststore alias based on the client authentication.
API Gateway registration listener configuration . Provide the following details to configure listener specific credentials.	
Registration port	Specifies the number you want to use for the registration port. Use a number that is not already in use. It is best not to use a standard port such as 80 (the standard port for HTTP) or 443 (the standard port for HTTPS) because the external firewall allows access to those ports from the outside world. You can add multiple registration ports by clicking +Add .
Alias	Specifies an alias for the port. An alias must be between 1 and 255 characters in length and include one or more of the following: letters (a -z, A-Z), numbers (0-9), underscore (_), period (.), and hyphen (-).
Description (optional)	A description of the port.
Protocol	Specifies the protocol to use for this port: HTTP or HTTPS. If you select HTTPS, additional security and credential boxes appear for which you have to provide the required values.
Bind address (optional)	Specifies the IP address to which to bind this port. Specify a bind address if your machine has multiple IP addresses and you want the port to use this specific address. If you do not specify a bind address, API Gateway picks one for you.

Security configuration. Provide the following details to configure security parameters.

Client authentication For the external port, specify the type of client authentication required.

Select one of the following:

- **Username/Password** . API Gateway does not request client certificates. The server looks for user and password information in the header of requests coming from an external client.

Field	Description
	<ul style="list-style-type: none"> ■ Request client certificate. This option appears only if you select the HTTPS option in the Protocol field of the API Gateway registration listener configuration section. API Gateway requests client certificates for all requests. If the client does not provide a certificate, the server prompts the client for a userid and password. The server checks whether the certificate exactly matches a client certificate on file and is signed by a trusted authority. If so, the client is logged in as the user to which the certificate is mapped in API Gateway. If not, the client request fails, unless central user management is configured. ■ Require client certificate. This option appears only if you select the HTTPS option in the Protocol field of the API Gateway registration listener configuration section. API Gateway requires client certificates for all requests. The server behaves as described for Request Client Certificates, except that the client must always provide a certificate.

Listener specific credentials (optional). This section appears only if you select HTTPS in the Protocol field of the API Gateway registration listener configuration section.. Provide the following details to configure listener specific credentials.

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. Although each listener points to one keystore, there can be multiple keys and their certificates in the same keystore, and more than one listener can use the same keystore alias.
Key alias (signing)	Specifies the private key of keystore.
Truststore alias	Specifies the public certificates of truststore. The alias points to a repository of public certificates. Provide the client or server truststore alias based on the client authentication.

6. Click **Add**.

The port is created and is listed in the ports table.

Important:

The global IP access mode will be applied to the newly created external and registration listener ports. You can modify the IP access mode as per your requirement. For information on modifying IP access mode of ports, see “[Configuring IP Access Mode for a Port](#)” on page 572.

7. Click the ✘ icon in the **Enabled** column next to the external and registration ports to enable them.

The port is enabled and a success message appears.

Configuring the API Gateway Internal listener

The API Gateway Internal listener processes the requests received from the API Gateway External port and sends responses to API Gateway. This procedure describes how to connect the Internal listener to API Gateway.

➤ To configure the API Gateway Internal listener

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > Ports**.
3. Click **Add Ports**.
4. Select the type of port as **API Gateway internal** and click **Add**.
5. Provide the following information:

Field	Description
Protocol	Specifies the protocol to use for this port (HTTP or HTTPS). If you select HTTPS, additional security and credential boxes appear for which you have to provide the required values.
Description (optional)	A description of the port.
Alias	Specifies an alias for the port. An alias must be between 1 and 255 characters in length and include one or more of the following: letters (a -z, A-Z), numbers (0-9), underscore (_), period (.), and hyphen (-).
Max connections	Specifies the number of connections maintained between API Gateway Internal port and API Gateway. The default value is 5.
Private threadpool configuration	Specifies whether to create a private thread pool for this port or use the common thread pool.
Enable	Select to enable the private threadpool configuration for this port.

Field	Description
Threadpool min	Specifies the minimum number of threads for this private threadpool. The default value is 1.
Threadpool max	Specifies the maximum number of threads for this private thread pool. The default value is 5.
Thread priority	Specifies the Java thread priority. The default value is 5.
API Gateway external server. Provide the following details to configure API Gateway external server.	
Host	Specifies the host name or IP address of the machine on which the server is running.
Port	Specifies the port number of the registration port on the Server.
Registration credentials (optional)	
User name	Specifies the name of the user on API Gateway that the internal server should connect as.
Password	Specifies the password of the user on API Gateway that the internal server should connect as.
External client security.	
Client authentication	Specifies the type of client authentication the internal server performs against external clients. External clients pass their authentication information to API Gateway, which in turn passes it to the internal server. Select one of the following:
<ul style="list-style-type: none"> ■ Username/Password. API Gateway does not request client certificates. Instead it looks for user and password information in the request header. ■ Digest. The Internal Server looks for password digest information in the request header. ■ Request Client Certificate. API Gateway requests client certificates for requests from external clients. If the client does not provide a certificate, the server prompts the client for a userid and password. The server checks whether the certificate exactly matches a client certificate on file and is signed by a trusted authority. If so, the client is logged in as the user to which the certificate is mapped in API Gateway. If not, the client request fails, unless central user management is configured. ■ Require Client Certificate. API Gateway requires client certificates for requests from external clients. If the external client does not supply a certificate, the request fails. 	

6. Click **Add**.

The port is created and is listed in the ports table.

Important:

The default access mode of the port is set to *Allow by default*. This implies that the port allows connections to all ESB services and folders, which in turn, increases the risk of exposing all enterprise assets hosted in internal Integration Server. Also, the risk is higher when the IS assets are secured by Anonymous Access Control Lists (ACL) or if the installation is exposed to the public internet. Hence, to avoid any potential security risk, you can set the access mode of the port to *Deny by default* before enabling it. When you change the access mode, you add the required services and folders to the *Allow* list. For more information, see [“Configuring Access Mode for a Port” on page 575](#).

7. Click the  icon in the **Enabled** column next to the port to enable the port.

The port is enabled and a success message appears.

Configuring the WebSocket Listener

The API Gateway WebSocket listener processes the requests from the clients. This procedure describes how to create the WebSocket listener to API Gateway.

Note:

WebSocket Secure port is not supported.

➤ To configure the WebSocket listener

1. Expand the menu options icon , in the title bar, and select **Administration**.
2. Select **Security > Ports**.
3. Click **Add Ports**.
4. Select the type of port as **WS** and click **Add**.
5. Provide the following information:

Field	Description
Port	Specify the number you want to use for the port. Select a number that is not already in use on this host machine.

Field	Description
Alias	Specifies an alias for the port. An alias must be between 1 and 255 characters in length and include one or more of the following: letters (a-z, A-Z), numbers (0-9), underscore (_), period (.), and hyphen (-).
Description (optional)	A description of the port.
Bind address (optional)	Specifies the IP address to which to bind this port. Specify a bind address if your machine has multiple IP addresses and you want the port to use this specific address. If you do not specify a bind address, API Gateway selects one for you.
Backlog	Specifies the number of requests that can remain in the queue for an enabled port before API Gateway begins rejecting requests. The default is 200. The maximum value is 65535.
Keep alive timeout	Specifies when to close the connection if the server has not received a request from the client within this timeout value (in milliseconds) or when to close the connection if the client has explicitly placed a close request with the server.

- Click **Add**.

The port is created and is listed in the ports table.

Important:

The default access mode of the port is set to *Allow by default*. This implies that the port allows connection to all ESB services and folders. To avoid any potential security risk, you can set the access mode of the port to *Deny by default* before enabling it. For information on changing the access mode, see “[Configuring Access Mode for a Port](#)” on page 575.

Also, the global IP access mode will be applied to the newly created WebSocket listener ports. You can modify the IP access mode as per your requirement. For information on modifying IP access mode of ports, see “[Configuring IP Access Mode for a Port](#)” on page 572.

- Click the  icon in the **Enabled** column next to the port to enable the port.

The port is enabled and a success message appears.

Configuring IP Access Mode for a Port

You can configure the access of a port by internal and external IP hosts or apply the global IP access settings.

This section explains how to apply global IP setting for a port.

➤ To apply global IP settings for a port

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > Ports**.
The ports page lists all ports configured with API Gateway, if any.
3. Click the **IP Accessmode** button for the port that you want to configure the IP access mode.
The options to configure the port access mode are displayed.
4. Select **Global**. The global IP settings are applied to the selected port.
5. Click **Save**.
The changes are saved.

Allowing Access to All IP Hosts

The following procedure describes how to change the IP access settings for an individual port to Allow by Default and deny some hosts.

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > Ports**.
The ports page lists all ports configured with API Gateway, if any.
3. Click the **IP Accessmode** button for the port that you want to configure the IP access mode.
The options to configure the port access mode are displayed.
4. Select **Allow by default**.
5. In the Deny List field, provide the names of hosts for which you want to deny access to the port and click + Add.

You can specify the host names (for example, `workstation5.webmethods.com`) or IP addresses (for example, `132.906.19.22` or `2001:db8:85a3:8d3:1319:8a2e:370:7348`) in the above field. Repeat this step to add the required host names and IP addresses to the list. Also, you can also edit or delete the entered values by clicking the respective action next to the required value.

The host names or IP addresses can include upper and lower case alphabetic characters, digits (0-9), hyphens (-), and periods (.) but cannot include spaces. For IPv6, IP addresses can also include colons (:) and brackets ([]).

You can use the following pattern-matching characters to identify several clients with similar host names or IP addresses.

Char	Description	Example
*	Matches any number of characters	r*.webmethods.com
?	Matches any single character	workstation?.webmethods.com

6. Click **Save**.

The changes are saved.

Denying Access to All IP Hosts

The following procedure describes how to change the IP access settings for an individual port to Deny by Default and allow some hosts.

1. Expand the menu options icon  in the title bar, and select **Administration**.
 2. Select **Security > Ports**.
- The ports page lists all ports configured with API Gateway, if any.
3. Click the **IP Accessmode** button for the port that you want to configure the IP access mode.
- The options to configure the port access mode are displayed.
4. Select **Deny by default**.
 5. In the Allow List field, provide the names of hosts for which you want to allow access to the port and click + Add.

You can specify the host names (for example, `workstation5.webmethods.com`) or IP addresses (for example, `132.906.19.22` or `2001:db8:85a3:8d3:1319:8a2e:370:7348`) in the above field. Repeat this step to add the required host names and IP addresses to the list. Also, you can also edit or delete the entered values by clicking the respective action next to the required value.

The host names or IP addresses can include upper and lower case alphabetic characters, digits (0-9), hyphens (-), and periods (.) but cannot include spaces. For IPv6, IP addresses can also include colons (:) and brackets ([]).

You can use the following pattern-matching characters to identify several clients with similar host names or IP addresses.

Char	Description	Example
*	Matches any number of characters	r*.webmethods.com
?	Matches any single character	workstation?.webmethods.com

6. Click **Save**.

The changes are saved.

Configuring Access Mode for a Port

The access mode of a port determines whether APIs can be invoked through the port or not. This section explains the steps required to configure the access mode of ports.

You can either allow or deny the access of all APIs through a port. When you allow access of APIs using a port by default, you can specify a list of APIs that must be denied access over the port. Also, if you deny the access of APIs using a port, you can specify a list of APIs that can be allowed to access using the port. This configuration is applicable for REST, SOAP, and OData APIs.

To enable the individual access modes ensure that you have set the following parameters accordingly:

- `watt.server.portAccess.axis2 = true` for SOAP APIs
- `pg.security.honourPortAccessModeSettings = true` for REST and OData APIs

Important:

When performing the following procedure, do not log into the server through the port you want to change, if you are selecting Deny by default. The procedure involves temporarily denying access to all APIs through the port. If you log on through the port you want to change and then deny access to all APIs through it, you will be locked out of the server. Instead, log on through a different existing port or create a new port to log on through.

➤ **To configure access mode for a port**

1. Expand the menu options icon  in the title bar, and select **Administration**.

2. Select **Security > Ports**.

The ports page lists all the ports configured with API Gateway.

3. Click the **Accessmode** button for the port that you want to configure the access mode.

The options to configure the port access mode are displayed.

4. Select one of the following options:

- **Allow by default.** To allow access of the port, by default.
- **Deny by default.** To deny access of the port, by default.

The port is enabled or denied for access by all APIs.

5. Optional. Perform one of the following:

- If you have selected **Allow by default**, provide the APIs for which you want to deny access through the port in the **Add APIs to Deny List** field and click **+ Add**. Repeat this step to add the required APIs to the list. You can also edit or delete the entered values by clicking the respective action next to the required value.
- If you have selected **Deny by default**, provide the APIs for which you want to allow access through the port in the **Add APIs to Allow List** field and click **+ Add**. Repeat this step to add the required APIs to the list. You can also edit or delete the entered values by clicking the respective action next to the required value.

To allow or deny access of all versions of an API through a port, specify the API in the following format:

- `apiName` or `apiName/` - For REST and OData APIs
- Not possible - For SOAP APIs

To allow or deny access of a particular version of an API through a port, specify the API in the following format:

- `apiName/version` or `apiName/version/` - For REST and OData APIs
- `/ws/apiName/version` - For SOAP APIs

For example, if you select **Allow by default** and provide `calc/1.0/` in **Add APIs and services to Deny List**, then the `1.0` version of the API `Calc` is denied to access through the port.

If you specify an API in the `apiName/version/` or `/ws/apiname/version/` format, then you must invoke the API using the protocol:`//host:port/ws/apiname/version/` format.

The API names along with their version numbers specified in the **Allow** and **Deny** lists must exactly match the required API names.

Note:

Even if an API has custom endpoints, you must provide the `apiName` and not the custom endpoint paths in the **Allow** and **Deny** lists.

6. Click **Save**.

The changes are saved.

Note:

To enforce this configuration on REST and OData APIs, when accessed through the HTTP and HTTPS ports, set the value of the **pg.security.honourPortAccessModeSettings** extended setting as `true`. To enforce this configuration on SOAP APIs, set the

watt.server.portAccess.axis2 setting as *true*. For information on configuring an extended setting, see “[Configuring Extended Settings](#)” on page 324.

API Gateway services to be exposed for API Portal and client communication

If you have configured port access restrictions to allow access only to the APIs hosted on the API Gateway (say with /gateway/, /ws/, and so on), then ensure that you also provide access to the following APIs in case the APIs are protected by security policies such as OAuth, OpenId or JWT. Allowing access to these endpoints is important for API Portal and API consumers to access API Gateway to retrieve the tokens.

- pub.apigateway.oauth2:getAccessToken
- secure.apigateway.oauth2:approve
- pub.apigateway.oauth2:authorize
- pub.apigateway.oauth2:authorize
- pub.apigateway.openid:getOpenIDToken
- pub.apigateway.openid:openIDCallback
- pub.apigateway.jwt:getJsonWebToken
- pub.apigateway.jwt:certs
- pub.apigateway.jwt:configuration
- pub.apigateway.jwt:thirdPartyConfiguration

Additionally, the following REST API endpoints are exposed by API Gateway, which are required from the API Portal to access API Gateway. This is to ensure that while you only allow required REST API endpoints, API Portal functionalities continue to work without any impact.

API Portal invokes the following two internal APIs of API Gateway:

- Token request endpoint (apigateway.accesstokens)
- JWT request endpoint (apigateway.jwt:getJsonWebToken)

Global IP Access Settings For Ports

This section describes how to specify the global IP access setting for ports. The server uses this setting to determine IP access for ports that do not have a custom IP access setting. The default global setting is Allow by Default.

When you create a port, you can customize IP access for it, or you can specify that it use the global IP access setting for the server. If you use the global IP access setting and later change it, the server will use the new global setting for the port. For example, as shipped, the server uses Allow by Default as the global IP access setting (with no hosts explicitly denied). If you create a new port 6666 and do not customize IP access for it, the server uses Allow by Default for port 6666. If you later change the global IP access to Deny by Default, the server will then use Deny by Default for

port 6666. If you later customize IP access to port 6666, subsequent changes to the global setting will have no effect on port 6666.

For any given port, you can specify IP access one of two ways:

- **Deny by Default.** Set up the port to deny requests from all hosts except for ones you explicitly allow. Use this approach if you want to deny most hosts and allow a few.
- **Allow by Default.** Set up the port to allow requests from all hosts except for ones you explicitly deny. Use this approach if you want to allow most hosts and deny a few.

You can specify access settings globally (for all ports) or individually (for one port). For more information, refer to respective sections listed in the following table:

Type of access	Section to refer
Configuring Global IP Access	
Deny by Default	"Allowing Connections from Specified Hosts" on page 578
Allow by Default	"Denying Connections from Specified Hosts" on page 579
Configuring IP Access for Individual Ports	
Allow by Default	"Allowing Access to All IP Hosts" on page 573
Deny by Default	"Denying Access to All IP Hosts" on page 574

To customize IP access for individual ports, see ["Allowing Access to All IP Hosts" on page 573](#) and ["Denying Access to All IP Hosts" on page 574](#).

Allowing Connections from Specified Hosts

The following procedure describes how to change the global IP access setting to Deny by Default and specify some hosts to allow.

Important:

If you inadvertently lock all hosts out of the server, you can correct the problem by following the steps given in the ["If You Inadvertently Deny IP Access to All Hosts" on page 580](#) section.

➤ To allow connection from specified IP hosts

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > Global IP Access Settings**.
3. Click **Port restrictions - Allow/Deny by IP address**.

The Global IP Access Settings section appears.

4. Select **Deny by default**.

Note:

If you select *Deny by default*, ensure that you configure the IP that must be allowed to access ports. Otherwise, ports cannot be accessed by external hosts.

5. In the Allow List field, provide the names of hosts for which you want to allow access to the port and click + Add.

You can specify the host names (for example, `workstation5.webmethods.com`) or IP addresses (for example, `132.906.19.22` or `2001:db8:85a3:8d3:1319:8a2e:370:7348`) in the above fields. Repeat this step to add the required host names and IP addresses to the list.

The host names or IP addresses can include upper and lower case alphabetic characters, digits (0-9), hyphens (-), and periods (.) but cannot include spaces. For IPv6, IP addresses can also include colons (:) and brackets ([]).

Note:

IP addresses are harder to spoof, and therefore more secure.

You can use the following pattern-matching characters to identify several clients with similar host names or IP addresses.

Char	Description	Example
*	Matches any number of characters	r*.webmethods.com
?	Matches any single character	workstation?.webmethods.com

6. Click **Save**.

The changes are saved.

Denying Connections from Specified Hosts

The following procedure describes how to change the global IP access setting to Allow by Default and specify some hosts to deny.

➤ To deny connection from specified IP hosts

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > Global IP Access Settings**.
3. Click **Port restrictions - Allow/Deny by IP address** section.

The Global IP Access Settings section appears.

4. Select **Allow by default**.
5. In the Deny List field, provide the names of hosts for which you want to allow access to the port and click + Add.

You can specify the host names (for example, `workstation5.webmethods.com`) or IP addresses (for example, `132.906.19.22` or `2001:db8:85a3:8d3:1319:8a2e:370:7348`) in the above fields. Repeat this step to add the required host names and IP addresses to the list.

The host names or IP addresses can include upper and lower case alphabetic characters, digits (0-9), hyphens (-), and periods (.) but cannot include spaces. For IPv6, IP addresses can also include colons (:) and brackets ([]).

You can use the following pattern-matching characters to identify several clients with similar host names or IP addresses.

Char	Description	Example
*	Matches any number of characters	r*.webmethods.com
?	Matches any single character	workstation?.webmethods.com

6. Click **Save**.

The changes are saved.

If You Inadvertently Deny IP Access to All Hosts

If you select *Deny by Default* in **Port restrictions - Allow/Deny by IP address** section from **Port Configure Global IP Access Settings** tab, all hosts are restricted from accessing the server ports. Hence, API Gateway cannot communicate with Integration Server.

For example, if you have defined five ports and those ports are configured to use the global IP access settings. In such case, if you change the global IP access setting to **Deny By Default**, API Gateway cannot communicate with hosts through any ports.

This section explains the steps to reset your setting and allow API Gateway to communicate with Integration Server.

➤ To modify the global IP access settings

1. Log off from API Gateway.
2. Open the diagnostic port of the Integration Server by entering `http://hostname:diagnostic_port` in your browser's address bar.

For example, `10.2.100.112:9999`. By default, the diagnostic port is 9999.

3. Navigate to the **Security > Ports**.

The Port List appears.

4. Perform any of the following:

- Click **Change Global IP Access Restrictions** and make required changes. For more information, see “[Allowing Connections from Specified Hosts](#)” on page 578.
- Click **Allow** in the **IP Access** column of the required port.

5. Restart Integration Server.

6. Refresh your browser and log on to API Gateway and modify global IP access settings.

Configuring Restriction to IP Address based on Authentication

You must have the *Manage Security Configuration* functional privilege to configure this restriction.

You can configure the restriction to client IP address based on authentication failure in API Gateway to prevent malicious attack that occurs when a client floods a server with many requests in an attempt to interfere with server processing. This restriction prevents the malicious attack by blocking or denying the unauthenticated client from accessing the APIs, when API Gateway fails to authenticate the client. Using API Gateway, you can limit the number of times a client fails to authenticate the API in a specified time interval. The reason for authentication failure can be either of the following:

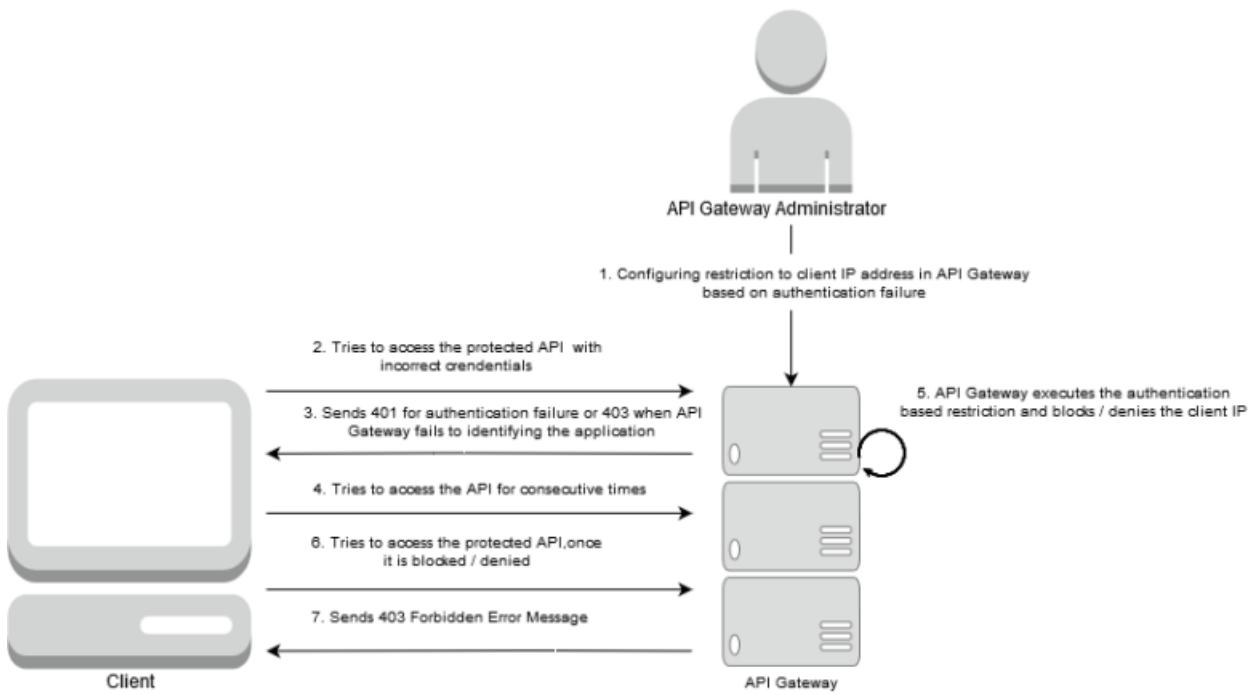
- when API Gateway fails to authenticate the client (**or**)
- when API Gateway fails to identify the client and its respective application.

When the above mentioned authentication failure occurs, API Gateway sends the *401* or *403* error message to the client.

When API Gateway detects that the failed authentication limit has been exceeded, it blocks or denies that particular client IP address from accessing any of the APIs and sends the *403 Forbidden* error to the client.

Note:

- If an API is enforced with Identify and Access Application policy, and if the invocation fails due to non-preemptive authentication failure. API Gateway does not take such non-preemptive authentication failure into count and increase the failed authentication count.
- When you use Load Balancer for configuring high availability between the API Gateway instances, API Gateway honours the X-Forwarded-For (XFF) header from the client. As the XFF header has the actual client IP address, API Gateway can block or deny the problematic client from accessing the protected API based on your configuration.



➤ To configure restriction to IP address based on authentication

1. Expand the menu options icon in the title bar, and select **Administration**.
2. Select **Security > Global IP Access Settings**.
3. Click **Authentication based restrictions - Block/Deny by IP address** section and provide the following information.

Field	Description
Enable	Specifies whether restriction to IP address based on authentication is enabled. Click the toggle button to change the state to to enable IP address restriction. By default this option is disabled.
Maximum failed authentication	Specifies the maximum number of failed authentication that API Gateway can accept from a specific IP address in a given time interval.
In (seconds)	Specifies the time interval, in seconds, in which maximum authentication failure can be permitted.
Action when limit exceeds	Specifies the action to be performed when the number of failed authentication from an IP address exceeds the specified limits.

Field	Description
	<p>Select one of the following:</p> <ul style="list-style-type: none"> ■ Add IP address to deny list. Permanently denies the IP address from accessing any APIs. ■ Block the IP address. Temporarily blocks the IP address from accessing any APIs for specified time interval. ■ In (seconds). Specify the time interval for which you want to block the IP address.
Denied IP list	<p>Specifies the list of IP addresses that are denied from access.</p> <p>Click  in the Action column to remove an IP address from the denied list.</p>

4. Click **Save**.

The configuration is saved.

SAML Issuer

If a native API is enforced with the SAML policy, API Gateway uses this configuration to communicate to STS (Security Token Service) to retrieve the SAML token.

➤ To add a SAML issuer

1. Expand the menu options icon , in the title bar, and select **Administration**.
2. Select **Security > SAML issuer**.

The SAML issuer page lists all the issuers configured along with the Endpoint URI corresponding to each SAML issuer, if any.

3. Click **Add SAML issuer**.
4. In the Add SAML issuer section, provide the following information:

Field	Description
Name	<p>Name of a SAML token issuer used by API Gateway.</p> <p>This value must match the value of the Issuer field in the SAML assertion.</p>

Field	Description
Normal client	Selecting this sets the client that requests the SAML token.
Act as delegation	Selecting this delegates the SAML request to another user (delegator). The delegator uses a signature element to authenticate the SAML request.
Issuer policy	Specifies the name of an issuer policy to be used to communicate with SAML issuer. <ul style="list-style-type: none"> ■ If a value is specified for the Issuer policy field, then the selected issuer policy is applied to all APIs that are using the SAML authentication. ■ If a value is NOT specified for this field, then a default issuer policy based on the WSS Username or Kerberos communication mode is applied to all APIs.
Communicate using. Specifies the mode of communication.	
WSS Username	Specifies that WSS Username mode is used to obtain the SAML assertion to access the API. The WSS username token supplied in the header of the SOAP request that the consumer application submits to the API.
Kerberos	Specifies that Kerberos mode is used to obtain the SAML token and assertion to access the API. Transports the Kerberos token over the Transport Layer Security (TLS) protocol to provide additional security features.
Authenticate using. Specify the type of authentication you want to use while communicating with the SAML issuer.	
For the Authentication type WSS Username , authenticate using the following:	
Custom credentials	Specifies the values provided in the policy required to communicate the SAML issuer. Provide the following information: <ul style="list-style-type: none"> ■ Username. Specify a username. ■ Password. Specify a password.
For the Authentication type Kerberos , authenticate using any of the following:	
Custom credentials	Specifies the values provided in the policy required to communicate the SAML issuer. Provide the following information:

Field	Description
	<ul style="list-style-type: none"> ■ 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: <ul style="list-style-type: none"> ■ 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	<p>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.</p>
	<p>Provide the following information:</p> <ul style="list-style-type: none"> ■ 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: <ul style="list-style-type: none"> ■ 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	<p>Specifies the incoming HTTP basic authentication credentials in the transport header of the incoming request for client principal and client password.</p>
	<p>Provide the following information:</p>

Field	Description
	<ul style="list-style-type: none"> ■ 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: <ul style="list-style-type: none"> ■ 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.
Endpoint URI	Provide the endpoint URI of the STS.
SAML version	Specify the SAML version to be used for authentication. Available values are: SAML 1.1, SAML 2.0
WS-Trust version	Specify the WS-Trust version that API Gateway must use to send the RST to the SAML issuer. Available values are: WS-Trust 1.0, WS-Trust 1.3
Applies to	Specify the scope for which this security token is required. For example, the APIs to which this token is applied.
Signing configurations	
Keystore alias	Specify the keystore to be used by API Gateway while sending the request to the STS. A keystore is a repository of private keys and corresponding public certificates.
Key alias (signing)	Specify the key alias, a private key used to sign the request sent to STS.
Encryption configurations	
Truststore alias	Select the truststore that should be used by API Gateway while sending the STS request. Truststore is a repository that holds all the trusted public certificates.
Certificate alias (Encryption)	Select the certificate from the truststore used to encrypt the request that is sent to the STS.

Field	Description
Request security token template parameters.	Defines extensions to the <wst:RequestSecurityToken> element for requesting specific types of keys, algorithms, or key and algorithms, as specified by a given policy in the return token(s).
Key	Specifies the key type of the security token template.
Value	Specifies a value for the request token.
	You can add multiple key and values by clicking  .

5. Click **Add**.

This adds the SAML issuer and it is listed in the SAML issuers list.

Custom Assertions

API Gateway uses WS-Security (WSS) to provide message-level security and protection for SOAP message requests from a client to an API, and SOAP message responses from an API to a client. By default, API Gateway supports the WSS policies like Username, X.509 certificate, Security Assertion Markup Language (SAML), Kerberos, Encryption, and so on, for the request or response SOAP messages, or both.

API Gateway also provides an extension to define and use custom policy assertions. Custom assertions allow the API providers to extend and provide additional security policies that are not available by default in API Gateway.

In WS-Security, custom assertions are used for expressing individual security requirements, constraints, or both. The individual policy assertions can be combined to create security policies that ensure secure and reliable exchanges of SOAP messages between a client and a SOAP API.

API Gateway supports the following assertion types for enforcing a custom security policy:

Binding Assertions

These assertions specify the security mechanism that is to be used by the client or API such as the keys being used, algorithms, and so on. Common properties used by other assertions are also defined in the security binding assertion.

API Gateway supports the following WS-SecurityPolicy binding assertions:

Binding Assertion	Description
Transport Binding	This assertion is used when the message is protected at the transport level. In this binding, messages are exchanged only through a defined medium, for example, HTTPS.
Note:	

Binding Assertion	Description
	By default, API Gateway uses the transport binding for Kerberos authentication.
Asymmetric Binding	This assertion is used when both the initiator and the recipient possess security tokens. In this binding, initiator uses its private key to sign and the recipient's public key to encrypt. Recipient uses its private key to decrypt and initiator's public key to verify the signature.
	<p>Note: By default, API Gateway uses the asymmetric binding for the security policies.</p>
Symmetric Binding	This assertion is used when only the initiator or recipient has a security token. In this binding, both the signing and encrypting of messages is done using a single security token.

Token Assertions

These assertions specify the types of tokens to be used to authenticate and secure SOAP messages.

API Gateway supports the following WS-SecurityPolicy token assertions:

Token Assertion	Description
Username Token	When using this assertion, the message-level security is implemented using a WSS username token. The assertion authenticates a client using the username and password in the SOAP request. If validation of the username token succeeds, then API Gateway passes the message to the API. If validation fails, then API Gateway returns a SOAP fault.
X509 Token	When using this assertion, the message-level security is implemented using an X.509v3 certificate. The assertion authenticates a client using the X.509v3 certificate in the SOAP request. If validation of the X.509v3 certificate succeeds, then API Gateway passes the message to the API. If validation fails, then API Gateway returns a SOAP fault.
Kerberos Token	When using this assertion, the message-level security is implemented using a Kerberos token. The assertion authenticates a client using the Kerberos token in the SOAP request. If validation of the Kerberos token succeeds, then API Gateway passes the message to the API. If validation fails, then API Gateway returns a SOAP fault.
SAML Token	When using this assertion, the message-level security is implemented using a SAML (Security Assertions Markup Language) token. SAML is a standard data format for exchanging authentication and authorization data between the client and the SOAP API. If validation of the SAML

Token Assertion	Description
	token succeeds, then API Gateway passes the message to the API. If validation fails, then API Gateway returns a SOAP fault.
	<p>Note: API Gateway supports both the SAML 1.1 and 2.0 standards.</p>

Policy Assertions

API Gateway allows you to even define a complete custom policy assertion. For example, a policy assertion might specify a symmetric binding and the security token types that are used to digitally sign or encrypt SOAP messages between the client and API.

Creating a Custom Assertion

Pre-requisites:

You must have the API Gateway's manage security configurations functional privilege assigned to add a custom assertion.

You might want to create a custom assertion when you want to:

- Enforce symmetric binding with an authentication mechanism that is not available by default in API Gateway.
- Support signing and encryption at the desired level.
- Modify the predefined encryption algorithm and security layout properties.
- Enforce custom authentication tokens that are not available by default in API Gateway.

Important:

When creating a custom assertion, make sure that both the syntax and the semantics of the assertion element are valid and in compliance with the Web Services Security Policy specification.

➤ To create a custom assertion

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > Custom assertions**.
API Gateway displays a list of all the currently defined policy assertions.
3. Click **Add assertion**.
4. Select the assertion type. The available options are:

- **Binding**
- **Token**
- **Policy**

5. Provide the following information:

Field	Description
Name	<p>Name of the custom assertion.</p> <p>For a binding or token assertion type, this is the display name of the assertion in the Binding Assertion field or Custom Token Assertion of the Inbound Auth - Message policy.</p> <p>For a policy assertion type, this is the display name of the assertion in the Issuer Policy field of the Add SAML Issuer configuration page.</p>
Select file	<p>Click Browse and select the policy assertion file to be uploaded.</p> <p>The Assertion element text box displays the data from the assertion file.</p> <p>If you have uploaded the policy assertion file and want to replace it, click the Delete icon.</p>
Assertion element	If you have not uploaded the policy assertion file, provide the XML representation of assertion.

6. Click **Add**.

The custom assertion is added. You can create as many custom assertions you require.

Post-requisites:

To enforce the custom binding or token assertion in an API, select the assertion in the appropriate fields of the **Inbound Auth - Message** policy:

- **Binding Assertion**
- **Custom Token Assertion**

To enforce the custom policy assertion in an API, select the assertion and the corresponding SAML issuer in the appropriate fields:

- **Issuer Policy** field of the **Add SAML Issuer** configuration page.
- **Authentication scheme** field of the **Outbound Auth - Message** policy.

Viewing Custom Assertion List and Assertion Configuration

You can view the list of configured custom assertions in API Gateway. In addition, you can view and modify the configuration in the individual custom assertion details page.

➤ To view a list of custom assertions and assertion configuration

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > Custom assertions**.

A list of all available custom assertions appears.

You can delete a custom assertion by clicking its **Delete** icon.

3. Click any custom assertion to view the configuration details.

The custom assertion details page displays the XML element.

Modifying Custom Assertion

Pre-requisites:

You must have the API Gateway's manage security configurations functional privilege assigned to modify a policy assertion.

You might want to modify a custom assertion to change the currently defined security settings, such as, authentication scheme, signing and encryption, algorithms and supporting tokens, of SOAP messages.

➤ To modify a custom assertion

1. Expand the menu options icon  in the title bar, and select **Administration**.
 2. Select **Security > Custom assertions**.
- A list of all available custom assertions appears.
3. Select the required custom assertion that you want to modify.
 4. Modify the fields as required.
 5. Click **Save**.

The custom assertion is updated.

Post-requisites:

When you are ready to put the policy assertion into effect in an API, select it in the appropriate field of **Inbound Auth - Message** policy.

Deleting Custom Assertion

Pre-requisites:

You must have the API Gateway's manage security configurations functional privilege assigned to delete a policy assertion.

You delete a policy assertion to remove it from API Gateway permanently.

➤ To delete a policy assertion

1. Expand the menu options icon  in the title bar, and select **Administration**.
 2. Select **Security > Custom assertions**.
- A list of all available custom assertions appears.
3. Click  in the action column of the custom assertion to be deleted.
 4. Click **Yes** in the confirmation dialog.

The custom assertion is deleted from API Gateway.

Example: Custom Assertions

API Gateway, by default, uses the asymmetric binding assertion with X.509v3 token for implementing SOAP message protection. If you would like to enforce any authentication (other than the predefined authentications shipped with API Gateway), include additional WSS custom assertions, sign and encrypt SOAP messages, and define custom properties, such as the algorithms and layout of security header, you can create custom assertions that would construct the custom policy file to suit your specific security requirements.

Following is a policy file that API Gateway generates when a WSS username token is enforced by the Inbound Authentication Message policy for an API.

```
<wsp:Policy wsu:Id="9dbda2fb-9cef-4ff9-bc70-115c942a3b76"
    xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
        oasis-200401-wss-wssecurity-utility-1.0.xsd">
    <wsp:ExactlyOne>
        <wsp:All>
            (L01) <sp:AsymmetricBinding xmlns:sp=
                "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
                xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
```

```

<wsp:Policy>
  <sp:InitiatorToken>
    <wsp:Policy>
      <sp:X509Token sp:IncludeToken=
        "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702
         /IncludeToken/Never">
        <wsp:Policy>
          <sp:WssX509V3Token10 />
        </wsp:Policy>
      </sp:X509Token>
    </wsp:Policy>
  </sp:InitiatorToken>
  <sp:RecipientToken>
    <wsp:Policy>
      <sp:X509Token sp:IncludeToken=
        "http://docs.oasis-open.org/ws-sx/ws-securitypolicy
         /200702/IncludeToken/Never">
        <wsp:Policy>
          <sp:WssX509V3Token10 />
        </wsp:Policy>
      </sp:X509Token>
    </wsp:Policy>
  </sp:RecipientToken>
  <sp:AlgorithmSuite>
    <wsp:Policy>
      <sp:TripleDesRsa15 />
    </wsp:Policy>
  </sp:AlgorithmSuite>
  <sp:Layout>
    <wsp:Policy>
      <sp:Strict />
    </wsp:Policy>
  </sp:Layout>
  <sp:ProtectTokens/>
</wsp:Policy>
</sp:AsymmetricBinding>
<sp:SupportingTokens xmlns:sp=
  "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
  <wsp:Policy>
    <sp:UsernameToken sp:IncludeToken=
      "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/
       IncludeToken/AlwaysToRecipient"/>
  </wsp:Policy>
</sp:SupportingTokens>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>

```

You might have a requirement to change the policy assertion that is available by default in API Gateway. For example, you might want to generate the above security policy using a symmetric binding instead of the default asymmetric binding, and modify the username token that is defined by default as a supporting token to a signed supporting token. You could then create custom policy assertions to achieve these specific requirements.

Important:

When adding a custom policy assertion, make sure that both the syntax and the semantics of the assertion are valid and in compliance with the Web Services Security Policy specification.

Symmetric Binding Assertion

You might want to use a symmetric binding (instead of the default asymmetric binding) when only API Gateway possess the X.509v3 token for authentication. You might also want to sign and encrypt the SOAP messages, modify the encryption algorithm, and include timestamp on the SOAP messages. You would then create a custom binding assertion with the specific property lines:

```
<sp:SymmetricBinding
    xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
    xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
<wsp:Policy>
<sp:ProtectionToken>
<wsp:Policy>
<sp:X509Token sp:IncludeToken=
    "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/
     IncludeToken/AlwaysToRecipient">
<wsp:Policy>
<sp:WssX509V3Token10/>
<sp:WssX509PkiPathV1Token10/>
</wsp:Policy>
</sp:X509Token>
</wsp:Policy>
</sp:ProtectionToken>
<sp:AlgorithmSuite>
<wsp:Policy>
<sp:TripleDesRsa15/>
</wsp:Policy>
</sp:AlgorithmSuite>
<sp:Layout>
<wsp:Policy>
<sp:Strict/>
</wsp:Policy>
</sp:Layout>
<sp:IncludeTimestamp/>
<sp:ProtectTokens/>
<sp:OnlySignEntireHeadersAndBody/>
<sp:SignBeforeEncrypting/>
</wsp:Policy>
</sp:SymmetricBinding>
```

You could create custom assertions to include one or more of the following security requirements:

Supporting Token Assertions

You might want to sign the supporting token for example, WSS username token, and use SignedSupportingTokens assertion. You might also want to specify that the signed username token must always be included in the messages sent to the recipient. You would then create a custom token assertion with the specific property lines:

```
<sp:SignedSupportingTokens xmlns:sp=
    "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
<wsp:Policy>
<sp:UsernameToken sp:IncludeToken=
    "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/
     IncludeToken/AlwaysToRecipient"/>
</wsp:Policy>
</sp:SignedSupportingTokens>
```

WSS Token Assertions

You might want to include WSS10 and WSS11 assertions to provide additional SOAP message security. You would then create two separate custom token assertions with the specific property lines:

Wss10 assertion:

```
<sp:Wss10
  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"

  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
<wsp:Policy>
  <sp:MustSupportRefIssuerSerial/>
</wsp:Policy>
</sp:Wss10>
```

Wss11 assertion:

```
<sp:Wss11
  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
<wsp:Policy>
  <sp:MustSupportRefIssuerSerial/>
  <sp:MustSupportRefThumbprint/>
  <sp:RequireSignatureConfirmation/>
</wsp:Policy>
</sp:Wss11>
```

After you have defined these custom assertions in API Gateway, execution of a policy that is configured with all of these custom assertions in the Inbound Auth - Message policy, would construct the custom security policy file as follows:

```
<wsp:Policy wsu:Id="1e747a18-b55d-4e99-ac67-80a8eaf76b3"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
    oasis-200401-wss-wssecurity-utility-1.0.xsd">
<wsp:ExactlyOne>
  <wsp:All>
    <sp:SymmetricBinding xmlns:sp=
      "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
        <wsp:Policy>
          <sp:ProtectionToken>
            <wsp:Policy>
              <sp:X509Token sp:IncludeToken=
                "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/
                  IncludeToken/AlwaysToRecipient">
                <wsp:Policy>
                  <sp:WssX509PkPathV1Token10/>
                </wsp:Policy>
              </sp:X509Token>
            </wsp:Policy>
          </sp:ProtectionToken>
        <sp:AlgorithmSuite>
          <wsp:Policy>
            <sp:TripleDesRsa15/>
          </wsp:Policy>
        </sp:AlgorithmSuite>
      <sp:Layout>
```

```
<wsp:Policy>
  <sp:Strict/>
</wsp:Policy>
</sp:Layout>
<sp:OnlySignEntireHeadersAndBody/>
</wsp:Policy>
</sp:SymmetricBinding>
<sp:SignedSupportingTokens xmlns:sp=
  "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
<wsp:Policy>
  <sp:UsernameToken sp:IncludeToken=
    "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/
     IncludeToken/AlwaysToRecipient"/>
</wsp:Policy>
</sp:SignedSupportingTokens>
<sp:Wss11 xmlns:sp=
  "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
<wsp:Policy>
  <sp:MustSupportRefIssuerSerial/>
  <sp:MustSupportRefThumbprint/>
  <sp:RequireSignatureConfirmation/>
</wsp:Policy>
</sp:Wss11>
<sp:Wss10 xmlns:sp=
  "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
<wsp:Policy>
  <sp:MustSupportRefIssuerSerial/>
</wsp:Policy>
</sp:Wss10>
<sp:EncryptedParts xmlns:sp=
  "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
<sp:Body/>
</sp:EncryptedParts>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
```

Kerberos Settings

Kerberos is an authentication protocol that uses symmetric encryption and a trusted third party system to validate the identity of clients. The Kerberos protocol provides authentication over open and insecure networks in which communication between the hosts can be intercepted.

You can use API Gateway to configure Kerberos authentication for API requests. API Gateway provides support for using Kerberos authentication for inbound and outbound HTTP and HTTPS requests at the transport and the message level.

Kerberos authentication system consists of a Kerberos client that needs to access and use Kerberos services, a trusted third-party system, specifically a Key Distribution Center (KDC), and a server that hosts APIs that are accessible using Kerberos authentication.

Note:

You can configure the kerberos settings through API Gateway and Integration Server UI. But, Software AG recommends to use API Gateway UI to configure or modify introspection endpoint.

Configuring API Gateway to Use Kerberos

Before you configure API Gateway to use Kerberos authentication, ensure that:

- A working Key Distribution Center (KDC) is set up.
- The KDC is configured as an LDAP directory, for authenticating incoming requests with Kerberos tickets.
- The Kerberos client is registered with the principal database of the KDC.
- The API that you want to access is registered with the KDC.
- A valid Kerberos configuration file is available.

➤ To configure API Gateway to use Kerberos

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > Kerberos**.
3. Provide or modify the following information as required:

Field	Description
Realm	Optional. The domain name of the Kerberos server, in uppercase letters. Note: A value specified for Realm overwrites the realm set in the KDC configuration file specified in Kerberos configuration file .
Key distribution center	Optional. The host name of the machine on which the KDC resides. A value specified for Key distribution center overwrites the default key distribution center set in the KDC configuration file specified in Configuration file .
Configuration file	The location of the Kerberos configuration file that contains the Kerberos configuration information, including the locations of KDCs, defaults for the realm and for Kerberos applications, and the host names and Kerberos realms mappings.
Use subject credentials	Specifies whether API Gateway requires a Kerberos V5 Generic Security Services (GSS) mechanism to obtain the necessary credentials from an existing subject set up by the JAAS authentication module. Here, subject represents the user or service being authenticated in the JAAS login context.

4. Click **Save**.

Master Password Management

In the normal course of its operations API Gateway may connect to native APIs, applications, databases, and other systems such as, API Portal, CentraSite, or external entities such as Email servers and databases. API Gateway is required to provide a password to each of these systems before connecting to them. API Gateway uses this password to identify itself or authenticate to the other systems.

When you configure API Gateway to connect to an application or subsystem, for example a database, you specify the password that API Gateway must send to the database server in order to connect to it. Later, when an API Gateway user makes a request that requires the database, API Gateway sends the configured password to the database server and connects to it. In API Gateway, you would be using passwords while enforcing security related policies, while connecting to various destinations such as, API Portal, CentraSite, Email, and SNMP, while configuring the security-related aliases, configuring outbound proxy servers, and so on.

To protect these passwords API Gateway encrypts them. By default, it encrypts them using Password-Based Encryption (PBE) standard, also known as PKCS5. This encryption method requires the use of an encryption key or master password that you specify. The encrypted passwords are stored in a file. The master password is also encrypted, and by default, is stored in a file. For greater security, you can change the master password in API Gateway at regular intervals or you can configure API Gateway to prompt for the master password at server startup instead.

Points to remember regarding master password:

- When the master password is updated in one node, it is not synchronized across other nodes in the cluster. The master password has to be updated manually in all the nodes.
- During export or import of assets, ensure that the master password is identical across stages and on different instances of API Gateway.

Backing up the Password and Master Password Files

You should regularly back up the files API Gateway uses to maintain the passwords and the master password. In API Gateway instance's home directory (*APIGateway_directory\instances\instance_name*), these files are:

- config/txnPassStore.dat: Stores encrypted passwords.
- config/empw.dat: Stores encrypted master password.
- config/configPassman.cnf: Specifies password configuration settings.
- config/passman.cnf: Non-editable version of configPassman.cnf.

Always back up and restore these files together. If you change the name or location of the password store or the master password store, make sure your backup procedure backs up the correct files.

Updating the Master Password

When you first install API Gateway, the master password is *manage*. For security purposes, you should change the master password immediately after installation and again on a regular basis. You should also change it when there are personnel changes.

The default expiry interval for a master password is 90 days. Once the master password expires, the status of the password changes to Inactive. As the expiration date nears, API Gateway displays the password expiration status on the API Gateway user interface and sends warning messages to the console stating that it is time to change the master password. If API Gateway is configured for e-mail notification, API Gateway also sends e-mail messages with this information to the configured addresses.

You must have the Manage security configurations functional privilege assigned to update the master password.

➤ To update the master password

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > Master password**.
3. Click **Update master password**.
4. Type the current password in the **Current password** field.
5. Type the new password in the **New password** field.
6. Re-type the new password to confirm the new password in the **Confirm new password** field.
7. Click **Update**.

This updates the master password.

Managing Master Password Expiry Interval

When you first install API Gateway, it is configured to use PBE to encrypt passwords, and has a master password of *manage* with an expiry interval of 90 days. You can see the current expiry date by looking at the **Administration > Security > Master password** screen.

The expiry interval is the time between password changes. If you do not change the master password by the expiry date, API Gateway continues to operate using the existing password indefinitely. If you specify an interval value 0, the password does not expire and no warnings are sent to API Gateway or the server log.

You must have the Manage security configurations functional privilege assigned to update the master password expiry interval.

➤ To update the master password expiry interval

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > Master password**.
3. Provide a value for the expiry interval.
4. Click **Update**.

This updates the expiry interval.

Advanced Configuration to Manage Master Password

The configPassman.cnf file contains additional configuration settings for password encryption. The file consists of a number of properties, some of which are commented out in the default configuration.

Note:

The configPassman.cnf file has a companion file, passman.cnf. If you make changes to configPassman.cnf file, API Gateway automatically updates passman.cnf to reflect these changes when you initialize API Gateway. Never update passman.cnf directly.

As shipped, the configPassman.cnf file specifies that passwords are stored in the config/txnPassStore.dat file and encrypted using Password-Based Encryption (PBE). In addition, it specifies that the master password is stored in the config/empw.dat file. Properties that can be used to specify other settings are commented out.

If you want to change these optional settings, you must edit the configPassman.cnf file. The file must always specify the following:

- Encryption method for passwords.
- Location of the file that contains the passwords.
- Method API Gateway uses to obtain the master password.

The following sections describe the configPassman.cnf file in detail and how to change password and master password settings.

Working with Password Settings

This section describes how to use the configPassman.cnf file to change settings for passwords.

Controlling Name and Location of Password File

The default file name and location for the password file is in the server instance's home directory under config/txnPassStore.dat. To change it, locate and modify the following property:

```
outbound.password.field.fileName=config/txnPassStore.dat
```

This property must always be present and uncommented. If you want to change the file name or location, change the right hand side only. You can specify an absolute or relative path. In the path name, use the forward slash (/) only; the backward slash (\) is not supported.

Controlling Encryption of Outbound Password File

The default encryption method for the password file is Password-Based Encryption (PBE). To change it, locate the following properties and uncomment a different method. One and only one of these properties must always be uncommented.

Property	Description	Security
default.encryptor=EntrustPbePlus	This denotes PBE encryption. Most secure	
#default.encryptor=Base64	This denotes Base64 encoding. Not secure	
#default.encryptor=None	This denotes Clear text.	Not secure

Working with Master Password Settings

By default, the master password is stored in the file config/empw.dat under the server instance's home directory, but if you prefer, you can configure API Gateway to prompt for the master password at server initialization. The following sections describe how to tell API Gateway which method to use.

Storing the Master Password in a File

To store the master password in a file, use the following properties:

Property	Description
master.password.storeInFile=true	This controls whether API Gateway stores the masterpassword in a file (true) or prompts for it at server initialization (false). If this value is set to true, make sure the master.password.field.attemptsLimit properties are commented out.
master.password.field.fileName=config/empw.dat	This indicates the location of the master password store. Use the forward slash (/) only; the backward slash (\) is not supported.
master.password.field.repeatLimit=3	This indicates the number of password changes required before you can reuse a password.

Prompting for the Master Password at Server Initialization

To prompt for the master password at server initialization, use the following properties. Use these properties only if you want API Gateway to prompt for the password at server initialization (that is, you specify `false` for `master.password.storeInFile`). If you do not want API Gateway to prompt for the password at server initialization, make sure these two properties are commented out.

Property	Description
<code>#master.password.field.useGUI=true</code>	Specify <code>true</code> to prompt for the password in a pop-up window. If you select this method, you can start the server from the Windows start menu. This is default if <code>master.password.storeInFile</code> is set to <code>false</code> .
<code>#master.password.field.attemptsLimit=3</code>	This indicates the number of unsuccessful login attempts permitted before API Gateway rejects the request.

You cannot configure API Gateway to prompt for the master password at server initialization if:

- API Gateway runs as a Windows service.
- API Gateway runs as a background application on UNIX.

Restoring the Password and Master Password Files

If your API Gateway is configured to encrypt passwords using Password-Based Encryption (PBE), your API Gateway will have a master password, which is the key used to encrypt the other passwords. You have to provide the master password whenever you want to change to a new encryption key. In addition, some installations are configured so that API Gateway prompts for the master password when API Gateway initializes; without the password, API Gateway starts up in safe mode. Therefore, if you lose or forget your master password, you have to restore it or reset it, depending on the circumstances.

You can restore passwords if either of the following is true:

- Your master password and other passwords are stored in files and you have recent backups of both and the `passman.cnf` file.
- API Gateway is configured to prompt for the master password, you have a recent backup of the password file and the `passman.cnf` file, and you know the master password for that backup.

➤ To restore the master password and other password files

1. Determine which files you need to restore.

If your master password is not stored in a file, that is, your API Gateway prompts you for a master password at server startup, then you can restore just the password file and the

passman.cnf file. Otherwise, you must restore the master password file, the password file, and the passman.cnf file from backups.

2. Determine the name and location of the files.

The passman.cnf file is always config/passman.cnf located under the server instance's home directory (APIGateway_directory\instances\instance_name). By default, the master password file is config/empw.dat and the password file is in config/txnPassStore.dat. If you are not sure of the location of these files on your system, look at the file config/configPassman.cnf.

3. Shut down API Gateway.
4. Copy the replacement files to appropriate directory.
5. Restart API Gateway.

Note:

Always back up and restore the master password file (if you use one), the password file, and the passman.cnf file together.

Resetting the Master Password

You can use the API Gateway Administration > Security > Master password section to reset the master password and all the stored passwords in the unlikely event the master password or the other passwords are lost or corrupted.

The reset procedure clears the stored passwords and resets the master password to *manage*.

You must have the Manage security configurations functional privilege assigned to reset the master password.

You must reset the passwords if any of the following is true:

- Your master password and passwords are stored in files and you do not have recent backups of the master password file, the password file, and the passman.cnf file.
- API Gateway is configured to prompt for the master password and you do not have recent backups of the password file and the passman.cnf file.
- API Gateway is configured to prompt for the master password and you have lost or forgotten the master password.

➤ To reset the master password

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Master password**.

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

This clears the stored passwords and resets the master password to *manage*.

OAuth, JWT, and OpenID Configuration

This section describes the Open Authorization (OAuth), JSON Web Token (JWT), and OpenID Connect (OpenID) authentication protocols that you can use to identify and authorize a client application. The application is first identified based on the criteria provided in the strategy configured. A strategy is a way to authenticate the incoming request and provides multiple authentication mechanisms or multiple authorization servers for a single authentication scheme. API Gateway identifies the application and validates the token submitted through the strategy configured in the application.

Configuring the Internal Authorization Server

Pre-requisites:

You must have the API Gateway's manage security configurations functional privilege assigned to add an authorization server.

You have to 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.

➤ To configure an internal authorization server

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > OAuth/JWT/OpenID**.
3. In the Internal Authorization servers section, click **local**.

This is the internal authorization server available that you can configure with required information to act as an internal authorization server for OAuth, JWT or OpenID authentication protocols.

4. The name field is pre-populated with the name of the internal authorization server, local, which is non-editable.
5. The description for the internal authorization server is pre-populated with the description available. You can modify the description as required.

-
6. Click **JWT configuration** to configure API Gateway as a JWT issuer.

Alternatively you can expand or collapse a section, using the down arrow () and the up arrow () and that appear next to the section name.

7. Provide the following information as required:

Field	Description
Token issuer	<p>Name of the JWT token issuer used by API Gateway.</p> <p>Note: The Token issuer value is case-sensitive.</p>
Algorithm	<p>The cryptographic algorithm to sign JSON Web Tokens (JWTs). Supported values are: RS256, RS384, and RS512.</p>
Expiry duration	<p>The duration (in minutes) for which the token is valid. For example, the value 60 denotes that the access token will expire in one hour from the time the token was generated.</p>
Audience	<p><i>Optional.</i> The intended recipient of the token. The application that receives the token must verify that the audience value is correct and reject any tokens intended for a different audience.</p>
Keystore alias	<p>Alias of the keystore containing the private key that is used to sign JWTs. The Keystore alias field contains a list of the available keystore aliases in API Gateway. If there are no configured keystore aliases, this field displays the <code>DEFAULT_IS_KEYSTORE</code>.</p>
Key alias	<p>Alias of the private key used to sign JWTs. The Key alias field contains a list of the available aliases in the selected keystore. If there are no configured keystores, this field is empty.</p>

8. Click **OAuth configuration** to configure API Gateway as an OAuth authorization server.

Alternatively you can expand or collapse a section, using the down arrow () and the up arrow () and that appear next to the section name.

9. Provide the following information as required:

- **Authorization code expiration interval.** Specifies the time (in seconds) during which the authorization code issued by the authorization server is valid. Valid values are between 1 and 2147483647. The default value is 600.

- **Access token expiration interval.** Specifies the time (in seconds) for which the access tokens issued by the authorization server are valid. The default value is 3600. Value of -1 specifies that the access token does not expire.
- **PKCE configuration.** Select **Enforce PKCE** check box to secure the get access token calls with PKCE mechanism.

10. Click **OAuth tokens**.

This lists the available OAuth tokens with the following details:

- **Client ID.** Specifies the ID of the client application that requested the access token.
- **Owner ID.** Specifies the ID of the owner who issues the access token.
- **Access token.** Specifies the access token
- **Refresh token.** You can use to generate a new access token if the existing access token is expired.
- **Remaining refresh limit.** Displays the remaining attempts for refreshing the access token.
- **Action.** Revokes the access tokens, which means those tokens cannot be used to invoke the protected resource.

Note:

By default, API Gateway lists only 5 records and provides pagination to explore more tokens. You can also use the search and filter options to find the OAuth tokens.

11. Click **OAuth scopes**.

OAuth 2.0 scopes provide a way to limit the amount of access that is granted to an access token. For example, an access token issued to a client application may be granted READ and WRITE access to the protected resources, or just the READ access. You can implement your APIs to enforce any scope or a combination of scopes as required. So, if a client receives a token that has READ scope, and it tries to invoke an API endpoint that requires WRITE access, the invocation fails.

You can provide the meaning to the scope in OAuth/OpenID scopes management section.

12. Type the scope that is registered in the authorization server and click **+Add**.

You can include multiple scopes.

13. Click **Update**.

This updates the internal authorization server details with the required information and is listed in the table of Internal authorization server.

Adding a Provider

Pre-requisites:

You must have the API Gateway's manage security configurations functional privilege assigned to add a provider.

The OAuth 2.0 configuration in API Gateway is split into two sections - Providers and Authorization servers.

You have to add a provider and configure the authorization provider metadata information in this section for API Gateway to communicate with this provider during dynamic client registration only. If there is any deviation from the actual OAuth specification then the provider has to be configured for these deviations.

➤ To add a provider

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > Providers**.
3. Click **Add provider** and provide the following information:

Field	Description
Name	<p>Name of a third-party provider. For example, Amazon.</p> <p>You can also use one of the following pre-configured third-party providers that is shipped with the API Gateway installation:</p> <ul style="list-style-type: none"> ■ OKTA ■ PingFederate <p>Note: Considerations while using the PingFederate providers:</p> <ul style="list-style-type: none"> ■ If you want to use the pre-configured PingFederate provider, you have to use the Admin APIs for dynamic client registration for registering clients. ■ If you want to use the DCR API, you can create a provider to use DCR API. But, you cannot update or delete the clients created using the DCR API.

Client metadata field mapping. Specifies the mapping of dynamic client registration specification to that of the client implementation of the provider.

The **Client metadata field mapping** fields are required when you are adding a third-party provider that is not shipped with API Gateway.

Field	Description
Specification name	<p>The client metadata attributes in accordance with the dynamic client registration specification as defined in RFC 7591.</p>
	<p>The available values are:</p>
	<ul style="list-style-type: none"> ■ redirect_uris. Redirection URL that the authorization server uses to redirect the authorization code once the authorization request is approved by end user.
	<p>Note: If you do not specify this attribute, API Gateway automatically generates the URL.</p>
	<ul style="list-style-type: none"> ■ token_endpoint_auth_method. The client authentication method at the token endpoint. ■ grant_types. The grant type of authorization flow to obtain authorization codes, ID tokens, and refresh tokens. ■ application_type ■ response_types. The type of response that the client application uses at the authorization endpoint. ■ client_name. Name of the client to use to represent the client application to the end user during authorization. ■ client_uri. URL of the client application. ■ logo_uri. URL of an image to use to represent the client application to the end user during authorization.
	<p>Note: The logo_uri is currently not supported in API Gateway.</p>
	<ul style="list-style-type: none"> ■ scope. List of user-authorized scopes that the client uses for requesting access tokens.
	<p>Note: If you do not specify this attribute, the authorization server registers the client with a default set of scopes.</p>
	<ul style="list-style-type: none"> ■ contacts. The means (for example, Email address) by which end users can contact the client for support requests. ■ tos_uri. URL of the service document for the client that describes a contractual relationship between the end-user and the client that the end-user accepts when authorizing the client.
	<p>Note:</p>

Field	Description
	<p>The tos_uri is currently not supported in API Gateway.</p> <ul style="list-style-type: none"> ■ jwks_uri. URL of the JSON Web Key (JWK) Set document containing the client's public keys. <p>Note: The jwks_uri is currently not supported in API Gateway.</p> <ul style="list-style-type: none"> ■ client_id. Identifier that is unique to the client application. ■ client_secret. The password or phrase for the client application to use to authorize communication with the end user.
Implementation name	<p>The client metadata attributes that are used by the authorization server, but are not in accordance with the dynamic client registration specification.</p> <p>Example:</p> <ul style="list-style-type: none"> ■ For the redirect_uris field, provide the value <i>redirectUris</i>. ■ For the grant_types field, provide the value <i>grantTypes</i>. ■ For the client_name field, provide the value <i>name</i>. ■ For the logo_uri field, provide the value <i>logoUrl</i>. ■ For the client_id field, provide the value <i>clientId</i>. ■ For the client_secret field, provide the value <i>secret</i>.
Extended request parameters	<p>Specifies the additional client metadata attributes that are specific to the authorization server, and are not specified in the dynamic client registration specification.</p> <p>In PingFederate (For example):</p> <pre>forceSecretChange = true</pre>
Type	<p>Specifies the client metadata attribute type.</p> <p>The available values are: Client read, Client registration, Client update, Client delete.</p>
Key	<p>The client metadata attribute key that is specific to the authorization server.</p>
Value	<p>A value for the client metadata attribute key. When sending requests to the authorization server, this value is appended to all requests.</p> <p>You can add multiple request parameters by clicking + Add.</p>

Field	Description
Application profile.	Specifies the application profile that is specific to the authorization server.
Type	<p>Specifies custom application type other than web and native. By default, the web and native application is added. You can add multiple application type by clicking + Add. You can also modify and delete the added application type by clicking the respective Edit or Delete icon.</p>

- Click **Save**.

The provider is added and displayed in the list of providers.

Adding an External Authorization Server

Pre-requisites:

You must have the API Gateway's manage security configurations functional privilege assigned to add 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. To use an external authorization server, you must configure your third-party authorization server.

➤ To add an external authorization server

- Expand the menu options icon  in the title bar, and select **Administration**.
- Select **Security > OAuth/JWT/OpenID**.

The available and configured internal authorization servers and external authorization servers are listed in the respective sections.

- Click **Add authorization server** in the External authorization servers section.
- Type a name for the external authorization server.
- Type a description for the external authorization server that is being configured.
- Provide the discovery endpoint in the **Discovery URL** field and click **Discover**.

When you specify the discovery URL, API Gateway fetches data from the URL response and auto-populates the fields with the fetched data. The discovery URL of the external authorization server is persisted after clicking the **Add** button.

- Click **Introspection** and provide the local or remote introspection details to validate the incoming tokens.

Field	Description
Local introspection. Provide the following information to validate the tokens locally by API Gateway.	
Issuer	Name of the token issuer.
JWKS URI	<p>Specifies JSON Web Key Signature endpoint to retrieve the corresponding public certificates for performing local introspection.</p> <p>API Gateway's cache has a key as <i>kid</i> claim and its value is the certificate corresponding to the <i>kid</i> claim. The cache is populated on every restart of API Gateway by invoking the JWKS URI.</p> <p>In the runtime, while validating the token using the local introspection, the <i>kid</i> value from the incoming JWT is fetched and the corresponding certificate is retrieved from the cache and the signature validation happens.</p>
Truststore alias	<p>Specify the alias of the truststore on API Gateway that holds the Certificate Authority (CA) certificate of third-party authorization server.</p> <p>This is required if the JWKS URI is not available for the authorization server and you want to configure this certificate directly.</p>
Certificate alias	<p>Alias of the certificate used to validate the token.</p> <p>The Certificate alias field contains a list of the available aliases in the selected truststore. If there are no configured truststores, this field is empty.</p>
Remote introspection. Provide the following information to validate the tokens remotely if local introspection cannot be done.	
Introspection endpoint	URL of the token introspection endpoint of a third-party OAuth 2.0 authorization server. API Gateway uses the introspection endpoint to check that access tokens used in client requests are currently active and are valid to invoke the protected resources.
Gateway user	The name of the Gateway user that API Gateway uses to invoke the token introspection endpoint.
Client ID	ID of the introspection client on the authorization server that API Gateway uses to introspect the access tokens.
<p>Note: Introspection client is any OAuth2 confidential client in API Gateway.</p>	

Field	Description
Client secret	Password of the introspection client that API Gateway uses to introspect the access tokens.

8. In the Dynamic client registration section, provide the following information if you want to dynamically create a client from API Gateway when required.

Note:

The dynamic client registration is not supported for external authorization servers when you publish an application from CentraSite to API Gateway.

You would use this configuration only if you do not intend to use any of the existing clients.

Field	Description
Enabled	<p>Specifies whether dynamic client registration is enabled.</p> <p>Click the toggle button to change the state to  to enable dynamic client registration.</p> <p>By default this option is disabled.</p>
Provider name	Select the name of the third-party provider.
Client registration URL	Specifies the corresponding REST endpoint URLs for the client configuration of REST APIs.
Authentication type	<p>Specifies the type of authentication scheme that API Gateway would use to communicate with the external authorization server for client management.</p> <p>Select one of the following authentication type:</p> <ul style="list-style-type: none"> ■ Basic. Specifies the username and password information that would be passed in the authorization header of HTTP request for client authentication. <ul style="list-style-type: none"> ■ Username. The username to access the protected resources of REST APIs. ■ Password. A valid password associated with the username. ■ Token. Specifies the token information that would be added as a bearer token in the HTTP request for client authentication. <ul style="list-style-type: none"> ■ Token type. The type of token that would be contained in the HTTP request. ■ Token. The token that would be contained in the HTTP requests.

Field	Description
	<ul style="list-style-type: none"> ■ Refresh token. Specifies the refresh token information that would be added as a bearer token in the HTTP request for client authentication. ■ Refresh token. The refresh token that you would get from the external authorization server for the registered client ID and client secret. ■ Client ID. The client ID that you want to specify from the external authorization server. ■ Client secret. A valid client secret associated with the client ID. <ul style="list-style-type: none"> ■ Client credentials. Specifies the client information for which the application is created in the external authorization server. ■ Scope. The scope of the client application that you want to specify from the external authorization server. ■ Client ID. The client ID that you want to specify from the external authorization server. ■ Client secret. A valid client secret associated with the client ID. <ul style="list-style-type: none"> ■ None . Specifies that you could create the client dynamically in the external authorization server without using any type of authorization.
Supported grant types	<p>Specifies the list of grant types that are supported by API Gateway. Basically, grant types are the ways to get an access token from the external authorization server.</p> <p>Provide the grant type, in the Supported grant types field and click +Add. You can add more than one grant by clicking +Add.</p>

9. In the SSL Configuration section, provide the following information for SSL configuration, if the authorization server wants the 2-way SSL for the requests.

Field	Description
Keystore alias	<p>Alias of the keystore containing the private key that is used for a secured communication between API Gateway and the authorization server.</p> <p>You can view all the keystore aliases available in API Gateway. If there are no configured keystore aliases, the list box contains only the default keystore, <i>DEFAULT_IS_KEYSTORE</i>.</p>
Key alias	Alias for the private key to use to validate the HTTP requests from the client.

Field	Description
	You can view all the aliases available in the selected keystore. If there are no configured keystores, this list box is empty.
Truststore alias	<p>Alias of the truststore on API Gateway that holds the Certificate Authority (CA) certificate of third-party authorization server.</p> <p>Note: You need to select a truststore alias only when all of the following are true:</p> <ul style="list-style-type: none"> ■ The client account on the third-party authorization server is configured to use mutual (two-way) SSL, and ■ The authorization server's Certificate Authority certificate is not in the set of well-known authorities trusted by the JVM in which API Gateway runs.

10. In the Metadata section, provide the following information for the authorization server metadata, which is used for the communication to portal.

Field	Description
Access token URL	The endpoint URL on the authorization server through which the client application exchanges the authorization code, client ID, and client secret, for an access token.
Authorize URL	The endpoint URL on the authorization server through which the end user authenticates and grants authorization to the client application.
Refresh token URL	The endpoint URL on the authorization server through which the client application refreshes an expired access token.

11. Click **Scopes**.

OAuth 2.0 scopes provide a way to limit the amount of access that is granted to an access token. For example, an access token issued to a client application may be granted READ and WRITE access to the protected resources, or just the READ access. You can implement your APIs to enforce any scope or a combination of scopes as required. So, if a client receives a token that has READ scope, and it tries to invoke an API endpoint that requires WRITE access, the invocation fails.

You can provide the meaning to the scope in OAuth/OpenID scopes management section.

12. Provide the scope, in the Scope field that is registered in the authorization server and click **+Add**.

You can add more than one scope by clicking **+Add**.

13. Click **Add**.

The external authorization server is added. You can add as many authorization servers as required, but only one is the default at any given time.

OAuth or OpenID Scopes

You must have the API Gateway's manage security configurations functional privilege assigned to manage scopes.

You have to map the scope that you have 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.

For example, if there is a scope you have defined for an external authorization server, such as readonly, then the access tokens which contain readonly as their scope, should access only the GET resources. So, you can create an API Scope for the GET resources in an API or for multiple APIs and then map this readonly scope to all those API Scopes. Now this access token can invoke only the GET resources. If it tries to invoke any POST or PUT resource it fails. As another example you can consider mapping a business scope such as, inventory, that you have defined in the authorization server; you can map all the resources required for the inventory business to this scope.

➤ To map a scope

1. Expand the menu options icon  in the title bar, and select **OAuth/OpenID scopes**.
2. Click **Map scope**.
3. Provide the following information in the Authorization server scope section:

Field	Description
Select authorization server scope	Specifies the scope linked to the authorization server. Type a search word and select the required scope from the search list populated.
Name	Displays the name of the authorization server scope selected. This is populated by default and is non-editable.
Description	A brief description for the scope being mapped.
Audience	Provide a value or URL, 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.

4. Click **API scopes**.
5. Specify an API scope that is to be linked to the authorization server.

Alternatively, you can type a search word and select the required API scope from the search list populated.

The API scopes added are listed in the Selected API scopes table. You can click the delete icon  , in the corresponding column, to delete an API scope from the list.

6. Click **Save**.

This maps the authorization server scope to the selected API scopes and lists the authorization scope in the scopes list.

Viewing Scope Mapping Details

You must have the API Gateway's manage security configurations functional privilege assigned to manage scopes.

You can view the scope details and modify the scope details as required from the OAuth/OpenID scopes page.

➤ To view scope mapping details

1. Expand the menu options icon  , in the title bar, and select **OAuth/OpenID scopes**.

A list of available scopes appears. Use the **Show** drop-down list at the bottom of the page to set the maximum number of scopes you want to display in a page. The details, such as, name and description of the scope is displayed in the form of a table. You can delete a scope by clicking the delete icon  .
2. Click a scope.

The scope details page appears. This page displays the details such as the authorization server name, the server scope, the API scopes that are linked to the server scope and the API scope details such as the API to which the scope is associated, the description of the API and API version number.

You can modify the scope by clicking the **Edit** button and modifying the required values.

Note:

You can edit or delete the APIs from the scope mapping, only if the APIs are assigned to your team(s).

Viewing Provider List and Provider Configuration

You can view the list of integrated third-party providers and their configuration details, modify the provider configuration, and delete a provider in the Providers section.

➤ To view a list of providers and provider configuration

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > Providers**.

The Providers section displays a list of all the defined third-party providers in API Gateway.

You can also perform the following operations in the Authorization servers section..

- You can view the configuration details of the provider by clicking the required provider. The provider details page displays the client metadata field mapping and extended request parameter configuration information of the selected provider.
- You can edit the provider configuration by clicking the required authorization server and modifying the details as required.
- You can reset the configuration to system default value by clicking  in the Action column for the respective provider.
- You can delete the required authorization server by clicking  in the Action column for the respective provider.

Modifying the Provider Configuration

Pre-requisites:

You must have the API Gateway's manage security configurations functional privilege assigned to modify a provider.

You might want to modify a provider to change the currently defined configuration settings.

➤ To modify a provider configuration

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > Providers**.

The Providers section displays a list of all the available providers in API Gateway.

3. Click the name of the partner provider you want to modify.
4. Modify the fields as required.
5. Click **Save**.

The provider is updated.

Viewing Authorization Server List and Server Configuration

You can view the list of authorization servers and their configurations details, modify the server configuration, and delete an authorization server in the Authorization servers section.

➤ To view a list of authorization servers and server configuration

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > JWT/OAuth/OpenID**.

The Authorization servers section displays a list of all the internal and external authorization servers in API Gateway under respective sections.

You can also perform the following operations in the Authorization servers section..

- You can view the configuration details of the authorization server by clicking the required authorization server. The authorization server details page displays the client information, scope information, and token information of the selected authorization server.
- You can edit the server configuration by clicking the required authorization server and modifying the details as required.
- You can delete the required authorization server by clicking  in the Action column for the respective authorization server.

Modifying Authorization Server Configuration

Pre-requisites:

You must have the API Gateway's manage security configurations functional privilege assigned to modify an authorization server.

You might want to modify an OAuth 2.0 authorization server to change the currently defined configuration settings.

➤ To modify the authorization server configuration

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Security > JWT/OAuth/OpenID**.

The Authorization servers section displays a list of all the internal and external authorization servers in API Gateway.

3. Click the name of the authorization server you want to modify.
4. Modify the fields as required.
5. Click **Save**.

The authorization server is updated.

Deleting an Authorization Server

Pre-requisites:

You must have the API Gateway's manage security configurations functional privilege assigned to delete an authorization server.

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

➤ To delete an authorization server

1. Expand the menu options 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 in API Gateway.

3. Click  in the action column of the authorization server to be deleted.
4. Click **Yes** in the confirmation dialog.

The authorization server is deleted from API Gateway.

Deleting a Provider

Pre-requisites:

You must have the API Gateway's manage security configurations functional privilege assigned to delete a provider.

You delete a provider to remove it from API Gateway permanently.

Important:

You must not delete a provider if it is being used by an authorization server.

➤ **To delete a provider**

1. Expand the menu options icon , in the title bar, and select **Administration**.
2. Select **Security > Providers**.

The Providers section displays a list of available providers in API Gateway.

3. Click  in the Action column of the provider to be deleted.
4. Click **Yes** in the confirmation dialog.

The provider is deleted from API Gateway.

Configuring Communication Details for Microgateway

When you want Microgateway to use API Gateway as an OAuth2 authorization server, the communication channel between Microgateway and API Gateway has to be set up. The access token is then introspected in the Microgateway using remote introspection. To enable this you have to configure communication details, such as the introspection endpoint, client ID and client secret, in API Gateway, which are then used by Microgateway to introspect the tokens in API Gateway.

➤ **To configure the communication details for Microgateway**

1. Expand the menu options icon , in the title bar, and select **Administration**.
2. Select **Security > Microgateway**.
3. In the **Introspection endpoint** field, provide the URL of the introspection endpoint.

Microgateway uses the introspection endpoint to check that access tokens used in client requests are currently active and are valid to invoke the protected resources.

The endpoint must use the https protocol. Considering the default https port is being used, the Integration Server introspection endpoint would be
`https://localhost:5543/invoke/pub.oauth/introspectToken`

4. In the **Client ID** field, provide an ID, which specifies the ID of the introspection client on the authorization server that Microgateway uses to introspect the access tokens.

Note:

Introspection client is any OAuth2 confidential client in API Gateway.

5. In the **Client secret** field, provide the Client secret, which specifies the password of the introspection client that Microgateway uses to introspect the access tokens.
6. In the **JWKS URI** field, provide the JSON Web Key Signature endpoint to retrieve the corresponding public certificates for performing local introspection.

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.

7. Click **Save**.

The information provided here is stored in the configuration properties file and provisioned as part of the asset provisioning during Microgateway startup.

Removing Expired OAuth Tokens

You can remove all the expired OAuth access tokens using the given API.

You can also schedule the cleanup of the expired OAuth access tokens as required.

➤ To remove expired OAuth tokens

1. Make a REST call to the following endpoint:

```
GET hostname:port/invoke/pub.oauth/removeExpiredAccessTokens
```

Revoking OAuth Tokens

You can revoke the OAuth access token from an application using the given API.

➤ To revoke OAuth token from an application

1. Make a REST call to the following endpoint with the corresponding client Id and secret of the application in Basic authentication header:

```
POST hostname:port/invoke/pub.oauth/revokeToken
```

Sample request

```
POST http(s)://hostname:port/invoke/pub.oauth/revokeToken
{
  "token": "3d77988d5020493c8edde78b12c347e2046ac8438a91405597e669ed714ba96a",
  "token_type_hint": "accessToken"
}
```

Threat Protection Policies

Threat protection policies prevent malicious attacks from client applications that typically involve large, recursive payloads, and SQL injections. You can limit the size of things, such as maximum message size, maximum number of requests, and maximum node depth and text node length, in the XML document. You can configure the global threat protection policies and rules for all the incoming requests that comes through the external port of API Gateway. These policies and rules are enforced by API Gateway based on your configuration.

You must have the API Gateway's manage threat protection functional privilege to configure the following policies and rules.

- Global Denial of Service
- Denial of Service by IP
- Rules

In addition, the API Gateway administrator can configure the necessary mobile devices and applications for which you want to deny the access, configure and customize the deny and alert rules, and manage the denied IPs.

Note:

- If the API Gateway instances used for Threat protection are clustered using TSA, and if you apply threat protection policy configuration in one of the API Gateway instances, the other API Gateway instances are updated automatically.
- If the API Gateway instances used for Threat Protection are not clustered using TSA, then you need to apply the required threat protection policy configurations in each of the API Gateway instance.

Basically, when you configure the threat protection policy in a clustered setup, you specify the limitations (such as number of requests and concurrent request) that an API Gateway instance in the cluster can handle during a specified time interval. Hence, if you add X number of API Gateway instances, the limitations set in the configuration also increases by X times.

For example, if you have two API Gateway instances and set the limitations as 100 requests per minute, then the API Gateway instances should be able to handle 200 requests per minute. When you add one more API Gateway instance, the processing capacity also increases to 300 requests per minute. Here, the API Gateway cluster used for Threat Protection does not act as a single unit.

Note:

When you have configured a load balancer, the load balancer exposes the actual client IP address using the X-Forwarded-For (XFF) headers. The `watt.server.enterprisegateway.ignoreXForwardedForHeader` property specifies whether API Gateway uses or ignores the IP address in the XFF headers. By default, API Gateway ignores the client IP address and so the `watt.server.enterprisegateway.ignoreXForwardedForHeader` property is set to true. If you want API Gateway to use the actual client IP address present in the XFF, then set the `watt.server.enterprisegateway.ignoreXForwardedForHeader` property to false.

Configuring Global Denial of Service Policy

You can configure this policy in API Gateway to prevent Denial of Service (DoS) attacks. One form of DoS attack occurs when a client floods a server with many requests in an attempt to interfere with server processing. Using API Gateway, you can limit the number of requests that API Gateway accepts within a specified time interval and the number of requests that it can process concurrently. By specifying these limits, you can protect API Gateway from DoS attacks.

You can configure API Gateway to limit the total number of incoming requests from the external ports. For example, you might want to limit the total number of requests received to 1000 requests in 10 seconds, and limit the number of concurrent requests to 100 requests in 10 seconds. When API Gateway detects that a limit has been exceeded, it blocks the exceeding requests for a specific time interval and displays an error message to the client based on your configuration. You can also configure a list of trusted IP addresses so that the requests from these IP addresses are always allowed and not blocked.

➤ To configure global denial of service policy

1. Click **Policies** in the title navigation bar.
2. Select **Threat protection > Global denial of service**.
3. Set the **Enable** button to the **On** position to enable the policy.
4. Type the maximum number of requests, in the **Maximum requests** field, that API Gateway can accept from any IP address in a given time interval.
5. Specify time in seconds, in the **In (seconds)** field, in which the maximum requests have to be processed.
6. Type the maximum number of concurrent requests, in the **Maximum requests in progress** field, that API Gateway can process concurrently.
7. Specify the time in minutes, in the **Block intervals (minutes)** field, for which you want requests to be blocked.
8. Type the alert message text, in the **Error message** field, to be displayed when the policy is breached.
9. Add IP addresses, in the **Trusted IP addresses** field, that can be trusted and are always allowed.
 - API Gateway supports IPv4 and IPv6 addresses in the trusted IP addresses lists.
 - You can specify a range of IP addresses using the classless inter-domain routing (CIDR) notation. To specify an IP address range, type the first IP address in the range followed by a forward slash (/) and a CIDR suffix.

Example IPv4 address range:

- 192.168.100.0/22 represents the IPv4 addresses from 192.168.100.0 to 192.168.103.255
- 148.20.57.0/30 represents the IPv4 addresses from 148.20.57.0 to 148.20.57.3

Example IPv6 address range:

- f000::/1 represents the IPv6 addresses from f000:: to ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff.
- 2001:db8::/48 represents the IPv6 addresses from 2001:db8:0:0:0:0:0 to 2001:db8:0:ffff:ffff:ffff:ffff:ffff.

Click  to add more than one IP address.

10. Click **Save**.

Configuring Denial of Service by IP Policy

You can configure this policy in API Gateway to prevent Denial of Service (DoS) attacks. One form of DoS attack occurs when a particular client floods a server with many requests in an attempt to interfere with server processing and not letting other clients in accessing the server. Using Denial of Service (DoS) by IP policy, you can limit the number of requests that API Gateway accepts from a particular IP address within a specified time interval and the number of requests that it can process concurrently from any IP address. By specifying these limits, you can protect API Gateway from DoS attacks by a particular IP address. When API Gateway detects that a limit has been exceeded, it blocks or denies the requests from that particular IP address and displays an error message to the client based on your configuration. You can also configure a list of trusted IP addresses so that the requests from these IP addresses are always allowed and not denied.

Note:

When you configure a load balancer, you need to insert the XFF headers on the load balancer to track the actual client IP address. When you use Load Balancer for high availability between the API Gateway instances, by default for all the incoming request, the source IP address will be the load balancer's IP address instead of the actual client IP address. In such scenario, when the Denial of Service by IP policy is enforced, all incoming requests will be denied irrespective of the problematic client. So, to prevent DoS attack from a problematic client, you need to consider the XFF headers that are inserted on the load balancer. This is achieved by setting `watt.server.enterprisegateway.ignoreXForwardedForHeader` property to false. When this setting is configured, the incoming request header will have the XFF header and tracks actual client IP address, which in turn allows you to configure DoS by IP.

➤ To configure the denial of service by IP policy

1. Click **Policies** in the title navigation bar.
2. Select **Threat protection > Denial of service by IP**.

3. Set the **Enable** button to the **On** position to enable the policy.
4. Type the maximum number of requests, in the **Maximum requests** field, that API Gateway can accept from a specific IP address in a given time interval.
5. Specify time in seconds, in the **In (seconds)** field, in which the maximum requests have to be processed.
6. Type the maximum number of requests, in the **Maximum requests in progress** field, that API Gateway can process concurrently from any single IP address.
7. Select one of the following actions to be taken when the number of requests from a non-trusted IP address exceeds the specified limits:
 - **Add to deny list** to permanently deny future requests from the IP address.
 - **Block** temporarily block requests from this IP address.
8. Type the alert message text, in the **Error message** field, to be displayed when the policy is breached.
9. Add IP addresses, in the **Trusted IP Addresses** field, that can be trusted and not blocked.
 - API Gateway supports IPv4 and IPv6 addresses in the trusted IP addresses lists.
 - You can specify a range of IP addresses using the classless inter-domain routing (CIDR) notation. To specify an IP address range, type the first IP address in the range followed by a forward slash (/) and a CIDR suffix

Example IPv4 address range:

 - 192.168.100.0/22 represents the IPv4 addresses from 192.168.100.0 to 192.168.103.255
 - 148.20.57.0/30 represents the IPv4 addresses from 148.20.57.0 to 148.20.57.3

Example IPv6 address range:

 - f000::/1 represents the IPv6 addresses from f000:: to ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff
 - 2001:db8::/48 represents the IPv6 addresses from 2001:db8:0:0:0:0:0 to 2001:db8:0:ffff:ffff:ffff:ffff:ffff

Click  to add more than one IP address.

 10. Click **Save**.

Managing Denied IP List

The Denied IPs section has a list of client IPs that were denied access due to breach of denial of service by IP policy. You can delete the IP from the denial list and make it available on client's request.

➤ To manage the denied IP list

1. Click **Policies** in the title navigation bar.
2. Select **Threat protection > Denied IPs.**

This displays a list of IP address that are denied from access.

3. Click  in the **Action** column so that the specified IP can be made available.

Configuring Rules

You can configure rules to filter malicious requests based on message size, authentication requests, requests from specific mobile devices and applications that could be harmful, requests that could cause an SQL injection attack, requests on anti-virus scan, XML / JSON requests, or use custom filters to avoid malicious attacks.

When a rule is established without any filters, with the Action set to **Deny Request and Alert** and the Request Type configured as **All**, all requests entering through the external port are rejected

API Gateway applies rules in the order in which they are displayed on the **Threat Protection > Rules** screen. Because a violation of a denial rule stops API Gateway from processing a request, hence it is important to prioritize the rules based on the order in, which you want them to be executed. The API Gateway processes denial rules followed by the alert rules.

➤ To configure rules

1. Click **Policies** in the title navigation bar.
 2. Select **Threat protection > Rules.**
- This displays a list of rules that are already configured.
3. Click **Add rule.**
 4. In the Rule properties section provide the following information:
 - a. Type a name for the rule in the **Rule name** field.

Valid rule names:

- Must be unique.
 - Must not be empty.
 - Must not contain spaces.
 - Must not contain the special characters - ? ~ ` ! @ # \$ % ^ & * () - + = { } | [] \\ : \ " ; ' < , /
- b. Type a description for the rule in the **Description** field.
- c. Select an action to be followed when the policy is violated:
- **Deny request and alert** to deny the access and send an alert when the policy is violated.
 - **Alert** to allow the request and send an alert when the policy is violated.
- d. Type the alert message text, in the **Error message** field, to be displayed when the policy is violated.
- e. Select the required **Request type** to which you want to apply the rule and provide the additional information required.
- The available values are:
- **ALL**. Applies the rule to all requests.
 - **REST**. Applies the rule to all REST requests.
 - **SOAP**. Applies the rule to all SOAP requests.
 - **INVOKE**. Applies the rule to all INVOKE requests.
 - **CUSTOM**. Applies the rule to all requests specified by the custom directives. You can use this option if you want a single rule applied for multiple request types and custom directives.
- f. Provide the following information to filter the requests depending on the **Request type** selected:
- **Resource path**. Provide the **Resource path** for the REST, SOAP, INVOKE, or CUSTOM Request type selected to filter the requests based on the resource being requested. The format for the REST, SOAP, and INVOKE request types is `folder_name/service_name` and the format for a CUSTOM request type is `given_directive/service_name`. You can add multiple resource paths using the **Add** button.
 - **Custom directives**. Provide the custom directives for the CUSTOM Request type to filter the incoming requests. For example, if you provide `gateway` as the directive, the rule applies to all these requests that are received in API Gateway with the directive `gateway`. You can add multiple directives using the **Add** button.
5. Configure the required filters as follows:

- **Alert settings.** Select one of the following options:
 - **Default.** Sets the default alert settings to be used.
 - **Custom.** You can specify this option to use the custom alert settings and provide the required information.
 - **Alert destination.** Specify the alert destination. Values are **Email** and **Flow service**.

If you select **Email**, provide the email ids to which the alert notification has to be sent.

If you select **Flow service**, a flow service is invoked. Specify the name of the flow service. You can create a flow service using the pub.security.enterpriseGateway:alertSpec as the signature of the service and or use the pre-defined flow service, pub.apigateway.threatProtection:violationListener. When you use the pre-defined service, the alerts are saved in API Data Store and displayed in the API Gateway Dashboard. For more information about the pub.security.enterpriseGateway:alertSpec specification, see the Integration Server Built-In Services Reference Guide.

- Provide the user, who has permissions to execute the service, as the user type. For example, Administrator.

Send alert: Select a condition depending on when you want the alert to be sent. Available values are **On rule violation** which sends an alert every time a request violates a rule or **Every** and specify the time interval (in minutes), which send alerts at specified intervals.

■ **Message size filter**

- Set the **Enable** button to the **On** position to enable the filter.
- Type the maximum size allowed for HTTP and HTTPS requests in the **Maximum message size (MB)** field.

If the request is larger than the size specified in this limit, the request violates the rule and API Gateway performs the configured action.

■ **OAuth filter**

- Set the **Enable** button to the **On** position to enable the filter.
- Set the **Require OAuth credentials** toggle button to the **On** position. This implies the request should contain the OAuth credentials else the request would be denied.

■ **Mobile application protection filter**

You can configure this filter to disable access for certain mobile application versions on a predefined set of mobile platforms. By disabling access to these versions, you are ensuring that all users are using the latest versions of the applications and taking advantage of the latest security and functional updates.

- Set the **Enable** button to the **On** position to enable the filter.
- Select the device type.

- Select the mobile application.
- Select the operator condition **=, >, <, >=, <= or <>**.
- Type the mobile application version.

You can add multiple entries by clicking .

■ **SQL injection protection filter**

You can use the SQL injection protection filter to block requests that could possibly cause an SQL injection attack. When this filter is enabled, API Gateway checks each request message for specific patterns of characters or keywords that are associated with potential SQL injection attacks. If a match is found in the request parameters or payload, API Gateway blocks the request from further processing.

- Set the **Enable** button to the **On** position to enable the selected filter.
- Select the required filters as follows:
 - Select **Database-specific SQL injection protection** and select a database against which specific parameters needs to be checked.

When enabled, API Gateway checks the incoming payload based on the specified database and GET or POST request parameters. If no parameter is specified, all input parameters are checked for possible SQL injection attack.

- Select **Standard SQL injection protection** and specify one or more GET or POST request parameters that could be present in the incoming requests. Parameters can contain only alphanumeric characters, dollar sign (\$), and underscore (_).

You can add multiple entries by clicking .

■ **Anti virus scan filter**

You can use the antivirus scan filter to configure API Gateway to interact with an Internet Content Adaptation Protocol (ICAP)-compliant server. An ICAP server is capable of hosting multiple services that you can use to implement features such as virus scanning or content filtering. Using the antivirus scan filter, API Gateway can leverage the ICAP protocol to scan all incoming HTTP requests and payloads for viruses.

- Set the **Enable** button to the **On** position to enable the filter.
- Type the antivirus ICAP engine name in the **ICAP name** field.
- Type the host name or IP address of the machine on which the ICAP server is running in the **ICAP host name or IP address** field.
- Type the port number on which the ICAP server is listening in the **ICAP port number** field.

- Type the name of the service exposed by the ICAP server that you can use to scan your payload for viruses in the **ICAP service name** field.
- **JSON threat protection filter**

You can use this filter to block attacks through JSON payload that have infinitely long strings or deeply nested payloads. API Gateway recommends that this protection should be combined with message size filter to identify infinite payloads.

Set the **Enable** button to the **On** position to enable the filter.

You can specify any of these parameters as filter criteria. If you do not specify a value, the system applies a default value of -1, which means an unlimited value.

Field	Description
Container depth	Specifies the maximum allowed containment depth, where the containers are objects or arrays. For example, an array containing an object which contains an object would result in a containment depth of 3.
Object entry count	Specifies the maximum number of entries allowed in an object.
Object entry name length field	Specifies the maximum string length allowed for a property name within an object.
Array element count	Specifies the maximum number of elements allowed in an array.
String value length	Specifies the maximum length allowed for a string value.
Applicable content type	Specify any other content types to be included in the filter.

You can add more entries by clicking .

- **XML threat protection filter**

You can use this filter to block attacks through XML payload that have infinitely long strings or deeply nested payloads. API Gateway recommends that this protection should be combined with message size filter to identify infinite payloads.

Set the **Enable** button to the **On** position to enable the filter.

You can specify any of these parameters as filter criteria. If you do not specify a value, the system applies a default value of -1, which the system equates to no limit.

Field	Description
Namespace prefix length	Specifies a limit on the maximum number of characters permitted in the namespace prefix in the XML document.

Field	Description
Namespace URI length	Specifies a character limit for any namespace URIs present in the XML document.
Namespace count per element	Specifies the maximum number of namespace definition allowed for any element.
Child count	Specifies the maximum number of child elements allowed for any element.
Attribute name length	Specifies a limit on the maximum number of characters permitted in any attribute name in the XML document.
Attribute value length	Specifies a limit on the maximum number of characters permitted in any attribute value in the XML document.
Attribute count per element	Specifies the maximum number of attributes allowed for any element.
Element name length	Specifies a limit on the maximum number of characters permitted in any element name in the XML document.
Text length	Specifies a character limit for any text node present in the XML document.
Comment length	Specifies a character limit for any comments present in the XML document.
Processing instruction target length	Specifies a limit on the maximum number of characters permitted in the target of any processing instructions in the XML document.
Processing instruction data length	Specifies a limit on the maximum number of characters permitted in the data value of any processing instructions in the XML document.
Node depth	Specifies the maximum node depth allowed in the XML.
Applicable content types	Specify any other content types to be included in the filter. You can add multiple values by clicking  .

■ Custom filter

You can use the custom filter to invoke a service that is available on API Gateway to perform actions such as custom authentication of external clients in the DMZ, logging or auditing in the DMZ, or implementation of custom rules for processing various payloads.

- Set the **Enable** button to the **On** position to enable the filter.
- Click **Browse** and select a service to invoke it.

- Select the user name of a user you want API Gateway to run the service. The default value is Administrator.
6. Click **Save**.

The new rule is created and appears in the list of rules in the Rules page.

The rule is applied to requests only if the rule is enabled. You can enable the rule in the Rules page by selecting the enable icon for the required rule.

Registering a Mobile Device or Application

You can use API Gateway to disable access for certain mobile application versions on a predefined set of mobile platforms. By registering the required devices and applications and disabling access to these versions, you ensure that all users use the latest versions of the applications and take advantage of the latest security and functional updates.

➤ To register a mobile device or application

1. Click **Policies** in the title navigation bar.
2. Select **Global Policies > Mobile devices and apps**.

3. Provide the mobile device type name and click .

Click  to add more entries. Click  to delete the added entries.

4. Provide the mobile application name and click .

Click  to add more entries. Click  to delete the added entries.

5. Click **Save**.

Configuring Alert Settings

You can configure the alert settings to control the following aspects of alerts that API Gateway sends when a request violates a rule:

- Whether API Gateway issues an alert for a rule violation.
- How often API Gateway issues the alert.
- The method API Gateway uses to send the alert.
- Whether a rule uses the default alert options or its own customized alert options.

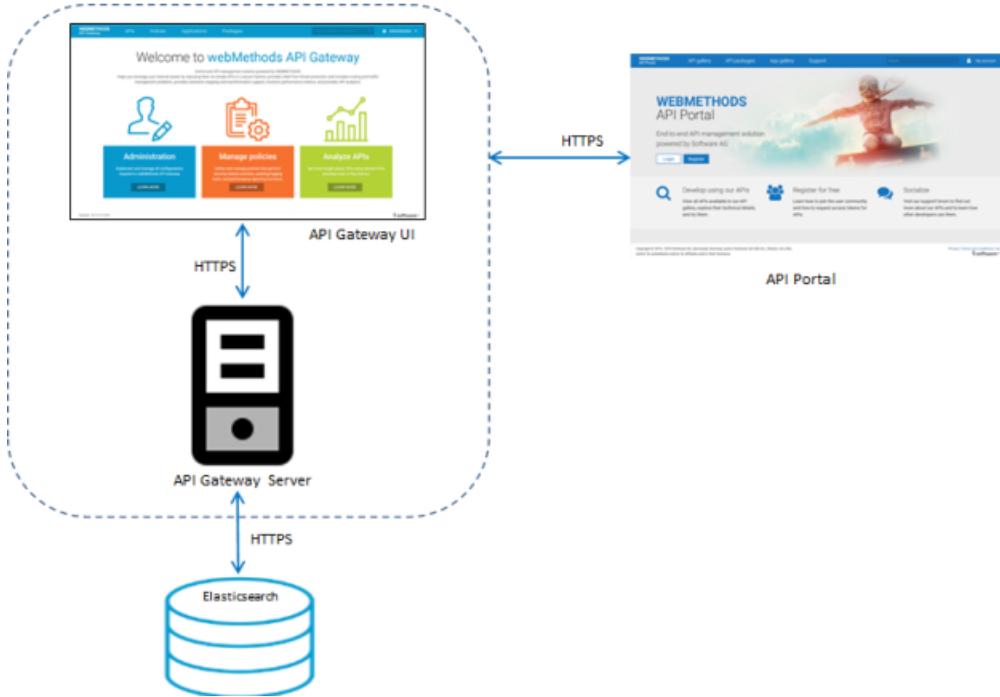
➤ To configure alert settings

1. Click **Policies** in the title navigation bar.
2. Select **Global Policies > Alert settings**.
3. Select one or both the alert destination types:
 - **Email**. This sends email alerts.
 - Type the email ids to which the email has to be sent.
 - **Flow Service**. This invokes a flow service to alert you of a rule violation. Specify the name of the flow service. You can create a flow service using the pub.security.enterpriseGateway:alertSpec specification as the signature of the service or use the pre-defined flow service, pub.apigateway.threatProtection:violationListener. When you use the predefined service, the alerts are saved in API Data Store and the displayed API Gateway Dashboard. For more information about the pub.security.enterpriseGateway:alertSpec specification, see the Integration Server Built-In Services Reference Guide.
 - Provide the user, who has permissions to execute the service, as the user type. For example, Administrator.
4. Select one of the following conditions depending on when you want the alert to be sent.
 - **On rule violation** to send an alert every time a request violates a rule,
 - **Every** and specify the time interval (in minutes) to send to send alerts at specified intervals.
5. Click **Save**.

Securing API Gateway Communication using TLS

This section describes how to secure communication, by leveraging SSL/TLS, between API Gateway and the API Clients, Users, Backend services, and API Portal.

The following figure illustrates how API Gateway communicates securely using HTTPS in the basic API Management setup.



For ensuring the security of the data being transferred between two components, you can implement one-way or two-way SSL/TLS. In an API Management setup you can configure a secure communication between the following:

- API Gateway and API clients. For details, see “[How Do I Secure API Gateway Server Communication with API Clients?](#)” on page 634
- API Gateway UI and Users. For details, see “[How do I Secure API Gateway User Interface Communication?](#)” on page 644
- API Gateway and API Portal. For details, see “[How do I Configure a Secure Communication Channel between API Gateway and API Portal?](#)” on page 646
- API Gateway and API Data Store. For details, see “[How do I Secure API Data Store Communication using HTTPS?](#)” on page 648

How Do I Secure API Gateway Server Communication with API Clients?

Secure API Gateway server to enable API clients to communicate with the API Gateway server over HTTPS. This section explains how to secure API Gateway server communication using HTTPS protocol by using the existing server and client certificates.

You must have API Gateway administrator privileges to perform this operation. Also, ensure that the required client and server certificates are available.

To configure API Gateway server for secure communication with API Clients

1. Locate the keystore and truststore files in the file system.

The default keystore and truststore files are available in the *Installation_Dir\common\conf* folder.

Note:

If you want to use a custom keystore with self-signed certificates, see “[Creating a Custom Keystore with Self-Signed Certificates](#)” on page 657 for details on how to create a keystore and generate the required self-signed certificate.

2. Configure keystore and truststore in the API Gateway UI.

You require a keystore alias for configuring an HTTPS port in API Gateway. You require the truststore alias for validating client certificates.

- a. Log on to API Gateway.
- b. Navigate to **Administration > Security > Keystore/Truststore**.
- c. Click **Add keystore**.
- d. Provide the following details:
 - **Alias.** A text identifier for the keystore file. The alias name can contain only alphabets, numbers and underscores. It cannot include a space, hyphen, and special characters.
 - **Select file.** Browse and select the file https_keystore.jks file located at *Installation_Dir\common\conf*.
 - **Password.** Specify the password for the saved keystore file associated with this alias.
 - **Type.** Specify the certificate file format of the keystore file, which, by default, is JKS for keystores.

Create keystore

Alias*
HTTPS_KEYSTORE

Select file*
https_keystore.jks

Password*

Type
JKS

Description

- e. Click **OK**.

A warning appears, prompting you to create a password for the key alias.

- f. Close the warning dialog box.

The Update keystore dialog box appears.

- g. Provide the password for the https_keystore file, for example, `manage`.

Update keystore

Alias*
HTTPS_KEYSTORE

Select file*
https_keystore.jks

Password*

Type
JKS

Description

- h. Click **Save**.
- i. Click **Add truststore**.
- j. Provide the following details.
 - **Name**. A name for the truststore file.
 - **Upload truststore file**. Browse and select the https_truststore.jks file located at *Installation_Dir\common\conf*.
 - **Password**. Specify the password that is used to protect the contents of the truststore, for example, manage.

Create truststore

Name*
Truststore

Upload truststore file*
https_truststore.jks

Password

Description

- k. Click **Save**.

- In the Configure keystore and truststore settings for inbound messages section, provide the keystore and truststore aliases for deploying any SOAP message flows that require signature, encryption, X.509 authentication, and so on, as configured in the Inbound Authentication - Message policy.

Configure keystore and truststore settings

Configure API Gateway's default Keystore and TrustStore alias for incoming secured messages [?](#)

Keystore alias	Key alias (signing)	Truststore alias
HTTPS_KEYSTORI	https_keystore	Truststore

[Cancel](#) [Save](#)

- Click **Save**.
- Create an HTTPS port in API Gateway and associate the keystore and truststore aliases.
 - Navigate to **Administration > Security > Ports**.
 - Click **Add ports**, and select **HTTPS** as the port type.
 - Click **Add**.
 - Provide the following details
 - **Port**. Specify the port number you want to use for the HTTPS communication.
 - **Alias**. Specify an alias for the port that is unique for this API Gateway instance. The alias must be between 1 and 255 characters in length and include one or more of the following: alphabets (a -z, A-Z), numbers (0-9), underscore (_), period (.), and hyphen (-).
 - **Backlog**. Specify the number of requests that can remain in the queue for an enabled port before API Gateway begins rejecting requests. The default is 200. The maximum value is 65535.
 - **Keep alive timeout**. Specify when to close the connection if the server has not received a request from the client within this timeout value (in milliseconds) or when to close the connection if the client has explicitly placed a close request with the server.
 - In the Listener-specific credentials section provide the following information:
 - **Keystore alias**. Select **HTTPS_KEYSTORE**.
 - **Key alias(signing)**. Select **https_keystore**.
 - **Truststore alias**. Select **Truststore**.

Ports
Configure listener ports in API Gateway.

HTTPS listener configuration

Port*: 8886

Alias*: HTTPS

Description (optional)

Bind address (optional)

Backlog*: 200

Keep alive timeout (milliseconds)*: 20000

Private threadpool configuration

Security configuration

Listener specific credentials (optional)

Keystore alias: HTTPS_KEYSTORE

Key alias (signing): https_keystore

Truststore alias: Truststore

Add

f. Click **Add**.

The HTTPS port 8886 is added and displayed in the list of ports.

Ports	Configure listener ports in API Gateway.						
<input type="checkbox"/>	Ports	Alias	Protocol	Type	Enabled	Primary port	Description
<input type="checkbox"/>	8886	HTTPS	HTTPS	Regular	X		Integration Server HTTPS port: 8886
<input type="checkbox"/>	5555	DefaultPrimary	HTTP	Regular	✓	✓	Default Primary Port

Add ports

g. Enable the new port 8886 by clicking the X mark in the port's **Enabled** column.

The port 8886 is now enabled and API Gateway server is now ready to accept requests over HTTPS port 8886.

4. Setup security configuration parameters for the HTTPS port, which is enabled for communication with API Clients, to determine how API Gateway server interacts with the clients and defines whether the connection is one-way or two-way SSL.
- Navigate to **Administration > Security > Ports**. This displays the list of ports.
 - Click the port 8886.
 - In the **Security configuration > Client authentication** section, select one of the following values:
 - Request client certificate**. API Gateway requests client certificates for all requests. If the client does not provide a certificate, the server prompts the client for a userid and password. The server checks whether the certificate exactly matches a client certificate

on file and is signed by a trusted authority. If so, the client is logged in as the user to which the certificate is mapped in API Gateway. If not, the client request fails, unless central user management is configured.

- **Require client certificate.** API Gateway requires client certificates for all requests. The server checks whether the certificate exactly matches a client certificate on file and is signed by a trusted authority.

- d. Click **Update**. The security configuration updates are saved.
5. Set port 8886 as primary port. *This is an optional step only if you want to change the primary port.*
 - a. Set the port 8886 as primary port by clicking in the port's **Primary port** column. The port 8886 is now enabled and API Gateway server is now ready to accept requests over HTTPS port 8886.

Ports	Alias	Protocol	Type	Enabled	Primary port	Description
8886	HTTPS	HTTPS	Regular	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Integration Server HTTPS port: 8886
5555	DefaultPrimary	HTTP	Regular	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default Primary Port

Add ports

- b. Disable the port 5555 by clicking the tick mark in the port's **Enabled** column.
The default primary port 5555 that accepts requests on HTTP is now disabled.
6. Configure the API Gateway UI to access the API Gateway server securely.

This step is required only when the primary port is set to HTTPS.

- a. Open the file uiconfiguration.properties located in the folder `Installation_Dir\profiles\IS_default\apigateway\config\`.
- b. Modify the following properties:

```
#IS properties
apigw.is.base.url = https://localhost:8886
apigw.is.rest.directive = /rest
apigw.user.lang.default = en
```

Here we configure the HTTPS port 8886 in the base URL property to point the API Gateway to communicate to the server URL.

Restart API Gateway server for the changes to take effect. You now have a secure communication channel established between the API Gateway server and the client.

Harden TLS configuration of the API Gateway server ports

To harden the TLS configuration of the API Gateway server ports, perform the following:

1. Disable support for unsafe protocols using the following setting:

```
watt.net.jsse.server.disabledProtocols=SSLv2Hello,SSLv3,TLSv1,TLSv1.1
```

2. *Optional.* You can restrict the supported protocols to exclusively use TLSv1.3 protocol by disabling TLSv1.2 using the following setting:

```
watt.net.jsse.server.disabledProtocols=SSLv2Hello,SSLv3,TLSv1,TLSv1.1,TLSv1.2
```

3. Reject the client initiated renegotiation by adding the following line to the custom_wrapper.conf file located in the directory SAG_root /profiles/IS_default/configuration.

```
wrapper.java.additional.402=-Djdk.tls.rejectClientInitiatedRenegotiation=TRUE
```

4. Specify a list of secure cipher suites.

For details about the recommended cipher suites, see the cipher suite recommendation by IANA organization (<https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml>) or the https://documentation.softwareag.com/webmethods/integration_server/pie10-5/10-5_Integration_Server_Administrators_Guide.pdf

5. Set the size of Ephemeral Diffie-Hellman Keys to 2048 depending on the configured cipher suites. You can do this by adding the following line to the custom_wrapper.conf file located in the directory SAG_root /profiles/IS_default/configuration:

```
wrapper.java.additional.401=-Djdk.tls.ephemeralDHKeySize=2048
```

You can verify the resulting TLS configuration using tools such as testTLS.sh that checks for vulnerable TLS configurations.

How Do I Secure API Gateway Server Communication with Backend Services?

Secure API Gateway server to enable secure communication with the backend services over HTTPS.

You must have API Gateway administrator privileges to perform this operation.

To configure API Gateway server for secure communication with Backend Services

1. Locate the keystore and truststore files in the file system.

The default keystore and truststore files are available in the *Installation_Dir\common\conf* folder.

Note:

If you want to use a custom keystore with self-signed certificates, see “[Creating a Custom Keystore with Self-Signed Certificates](#)” on page 657 for details on how to create a keystore and generate the required self-signed certificate.

2. Configure keystore and truststore in the API Gateway UI.

You require a keystore alias for configuring an HTTPS port in API Gateway. You require the truststore alias for validating backend service certificates.

- a. Log on to API Gateway.

- b. Navigate to **Administration > Security > Keystore/Truststore**.
- c. Click **Add keystore**.
- d. Provide the following details:
 - **Alias**. A text identifier for the keystore file. The alias name can contain only alphabets, numbers and underscores. It cannot include a space, hyphen, and special characters.
 - **Select file**. Browse and select the file https_keystore.jks file located at *Installation_Dir\common\conf*.
 - **Password**. Specify the password for the saved keystore file associated with this alias.
 - **Type**. Specify the certificate file format of the keystore file, which, by default, is JKS for keystores.

Create keystore

Alias*	HTTPS_KEYSTORE
Select file*	https_keystore.jks <input type="button" value="Browse"/> <input type="button" value="Delete"/>
Password*	*****
Type	JKS <input type="button" value="▼"/>
Description	
<input type="button" value="Cancel"/>	<input type="button" value="OK"/>

- e. Click **OK**.

A warning appears, prompting you to create a password for the key alias.

- f. Close the warning dialog box.

The Update keystore dialog box appears.

- g. Provide the password for the https_keystore file, for example, manage.

Update keystore

Alias*
HTTPS_KEYSTORE

Select file*
https_keystore.jks

Password*

Type
JKS

Description

- h. Click **Save**.
- i. Click **Add truststore**.
- j. Provide the following details.
 - **Name**. A name for the truststore file.
 - **Upload truststore file**. Browse and select the https_truststore.jks file located at *Installation_Dir\common\conf*.
 - **Password**. Specify the password that is used to protect the contents of the truststore, for example, manage.

Create truststore

Name*
Truststore

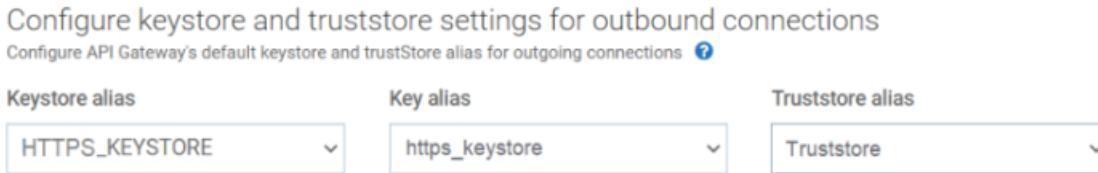
Upload truststore file*
https_truststore.jks

Password

Description

- k. Click **Save**.

3. To communicate securely with the backend services you have to configure the keystore and truststore settings for outbound connections. This can be configured in one of the following ways:
- Globally, you can configure the keystore and truststore settings for outbound connections in **Administration > Security configuration** section as follows:
 1. Navigate to **Administration > Security > Keystore/Truststore**.
 2. In the Configure keystore and truststore settings for outbound connections section, provide the keystore and truststore aliases for securing outgoing SSL connections. The keystore and key alias are required for outgoing two-way SSL connections.



- At an API-level, you can configure the keystore and truststore in the following ways:
 - Through an endpoint alias configured in the routing policy:
 1. Create an endpoint alias where you specify the default URI, and the keystore and truststore for the backend service. For details about creating an endpoint alias, see Aliases section in the *webMethods API Gateway User's Guide*.
 2. Specify the endpoint alias in the **Endpoint URI** field in the routing policy properties section when you configure the policy. For details, see Routing Policies section in the *webMethods API Gateway User's Guide*.
 - Through a routing policy by specifying the URI of the backend service endpoint, and the keystore and truststore. For details, see Routing Policies section in the *webMethods API Gateway User's Guide*.

Note:

The global keystore and truststore configuration is the default configuration that applies for all APIs if there is no keystore or truststore configured through an endpoint alias or a routing policy at an API-level.

You now have a secure communication channel established between the API Gateway server and the backend services.

How do I Secure API Gateway User Interface Communication?

Secure API Gateway UI (web application), one of the API Gateway components in an API Management setup, to enable users to access the API Gateway UI securely over HTTPS. This section explains how to secure API Gateway communication using HTTPS protocol.

You must have API Gateway administrator privileges to perform this operation. Also, ensure that the required client and server certificates are available.

To configure API Gateway user interface for secure communication

1. Locate the keystore and truststore files in the file system.

The default keystore and truststore files are available in the *Installation_Dir\common\conf* folder.

Note:

If you want to use a custom keystore with self-signed certificates, see “[Creating a Custom Keystore with Self-Signed Certificates](#)” on page 657 for details on how to create a keystore and generate the required self-signed certificate.

2. Configure the keystore and the HTTPS port on which you want to expose API Gateway UI.
 - a. Navigate to *Installation_Dir\profiles\IS_default\configuration\com.softwareag.platform.config.propsloader* and search for files that begin with the following file name pattern *com.softwareag.catalina.connector.https.pid-*.properties*. If there are more than one file in the selected folder that match this pattern, then the first file that matches the pattern is selected.

Note:

The file name by default is *com.softwareag.catalina.connector.https.pid-apigateway.properties*.

- b. Open the file and modify the following properties by providing the keystore, password, and port details.

```
keystoreFile=generated_keystore_file_path/https_keystore.jks
port=9073 (https port in which you want to expose webApp)
@secure.keystorePass=password (password used while creating the keystore file)
```

For details about the configurations, see *Software AG Infrastructure Administrator’s Guide* and <https://tomcat.apache.org/tomcat-7.0-doc/config/http.html>.

Harden TLS configuration of the API Gateway UI port

To harden the TLS configuration of the API Gateway UI port, perform the following:

1. Enable TLSv1.2 or TLSv1.3 protocol as follows:

- a. Navigate to *Installation_Dir\profiles\IS_default\configuration\com.softwareag.platform.config.propsloader* and search for files that begin with the following file name pattern *com.softwareag.catalina.connector.https.pid-*.properties*. If there are more than one file in the selected folder that match this pattern, then the first file that matches the pattern is selected.

Note:

The file name by default is *com.softwareag.catalina.connector.https.pid-apigateway.properties*.

- b. Open the file and add the following line to the properties file.

```
sslEnabledProtocols=TLSvversion number
```

For example, if you want to enable the TLSv1.3 protocol, the sample code is as follows:

```
sslEnabledProtocols=TLSv1.3
```

2. Specify a list of secure cipher suites by adding the following line to the above properties file

```
ciphers="List of Secure Cipher_Suites"
```

For details about the recommended cipher suites, see the cipher suite recommendation by IANA organization (<https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml>) or *webMethods Integration Server Administrator's Guide*

3. Set the size of Ephemeral Diffie-Hellman Keys to 2048 depending on the configured cipher suites. You can do this by adding the following line to the custom_wrapper.conf file located in the directory SAG_root /profiles/IS_default/configuration:

```
wrapper.java.additional.401=-Djdk.tls.ephemeralDHKeySize=2048
```

You can verify the resulting TLS configuration using tools such as testTLS.sh that checks for vulnerable TLS configurations.

How do I Configure a Secure Communication Channel between API Gateway and API Portal?

This section explains the steps required for API Gateway to securely communicate with API Portal for sending the runtime events and metrics and API Portal to communicate with API Gateway securely for key requests.

The described SSL configuration procedure applies only to API Portal version 10.2 or later. Also ensure that the required certificates for API Gateway and API Portal are available.

To configure a secure communication channel between API Gateway and API Portal

1. Configure API Portal HTTPS port.
 - a. Navigate to **Administration > Destinations** in the API Gateway user interface.
 - b. Click **API Portal > Configuration**.
 - c. Provide the following information:

The screenshot shows the 'Administration' section of the API Gateway interface. On the left, a sidebar lists various configuration categories: API Gateway, API Portal (which is selected), Configuration, Events, Transaction logger, CentralSite, Configuration, Events, Database, Elasticsearch, Configuration, Events, Email, Configuration, Templates, SNMP, Configuration, and Events. The main panel is titled 'API Portal communication' and contains sections for 'Basic information' and 'Portal configuration'. In 'Basic information', the 'Name' field is set to 'Portal' and the 'Version' field is set to '1.0'. In 'Portal configuration', the 'Base URL' field is filled with 'https://sag-dtkywt2.eur.ad.sag:18102'. Below it, the 'Tenant' field is set to 'default', 'Username' to 'system', and 'Password' to a masked value. To the right, under 'Gateway configuration', the 'Base URL' field is set to 'https://sag-dtkywt2.eur.ad.sag:8886', and the 'Username' and 'Password' fields are also masked. At the bottom of the panel are 'Cancel' and 'Publish' buttons.

- In the Portal configuration section, provide the following details:
 - **Base URL.** The API Portal base URL which API Gateway uses to communicate to API Portal using the HTTPS port. By default, API Portal uses port 18102 for HTTPS communication.
 - **Username** and **Password** credentials to access API Portal.
 - In the Gateway configuration section, provide the following details:
 - **Base URL.** The API Gateway server URL, which API Portal uses to communicate to API Gateway using the HTTPS port. Specify the port 8886 that is configured for HTTPS communication.
 - **Username** and **Password** credentials to access API Gateway.
- d. Click **Publish**.
- This configures API Portal as a destination and creates a communication channel between API Gateway and API Portal over the HTTPS port.
2. Ensure that outbound truststore is configured correctly to trust the certificate exposed by API Portal.
- You can achieve this by configuring keystore and truststore settings for outbound connections in API Gateway. In the Configure keystore and truststore settings for outbound connections section, provide the keystore and truststore aliases for securing outgoing SSL connections. The keystore and key alias is required for outgoing two-way SSL connections.

Configure keystore and truststore settings for outbound connections

Configure API Gateway's default keystore and trustStore alias for outgoing connections 

Keystore alias

HTTPS_KEYSTORE

Key alias

https_keystore

Truststore alias

Truststore

3. You have to configure the API Portal truststore to trust the API Gateway outbound certificate. For details about how to configure API Portal truststore, see API Portal documentation.

You now have a secure communication channel between API Gateway and API Portal. You can now publish an API, which is enforced with Enable HTTPS/HTTPS policy with the HTTPS option configured, from API Gateway to API Portal and invoke the API from API Portal using the HTTPS endpoint that has been used to publish it to API Portal.

How do I Secure API Data Store Communication using HTTPS?

You can secure API Data Store (a simple Elasticsearch instance), one of the components in an API Management setup, to communicate securely over HTTPS. To secure API Data Store, you can use the following security plugins:

- X-Pack. For details, see <https://www.elastic.co/guide/en/elasticsearch/reference/current/security-minimal-setup.html>
- ReadonlyREST. For details, see “[How do I Secure API Data Store Communication using HTTPS with ReadonlyREST Plugin?](#)” on page 648.

Both the plugins offer encryption, authentication, and authorization to protect data from attackers and other misuses. The plugins secure API Data Store by exposing it over HTTPS, and enables basic authentication by configuring users.

How do I Secure API Data Store Communication using HTTPS with ReadonlyREST Plugin?

You can use the ReadonlyREST plugin to secure API Data Store (a simple Elasticsearch instance), one of the components in an API Management setup, to communicate securely over HTTPS.

Before you begin

Ensure that you have:

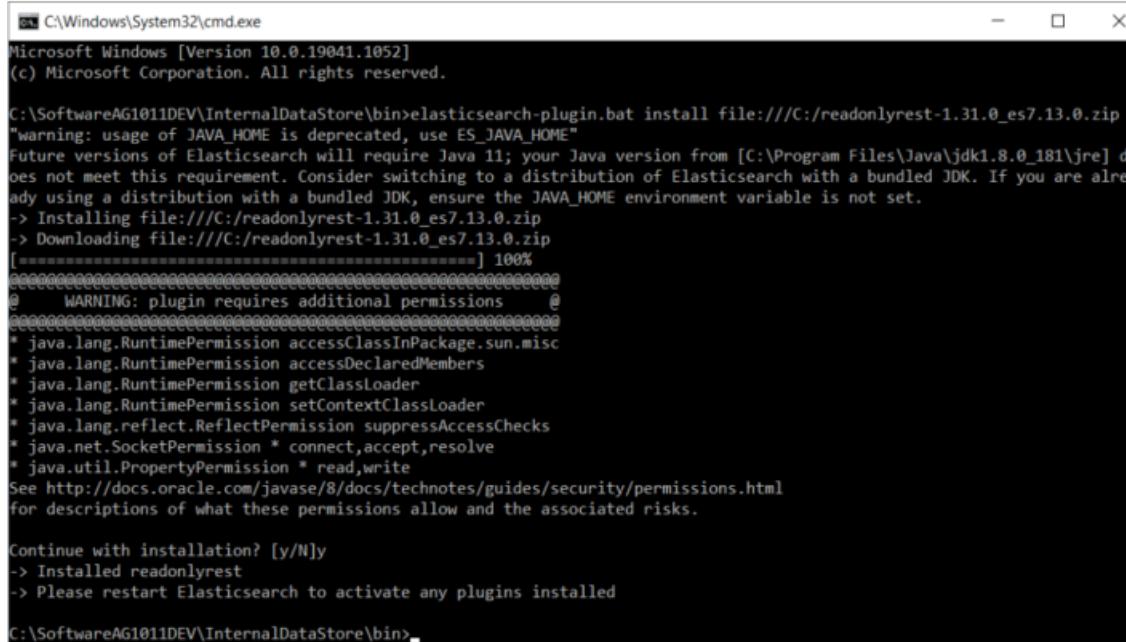
- Downloaded the ReadonlyREST plugin. You can download the ReadonlyREST plugin compatible with Elasticsearch 8.2.3 from <https://readonlyrest.com/download> and store it in your file system.
- Installed an updated version of Java in your system and the path of the environment variable is set.

To secure API Data Store communication using HTTPS

1. Install and initialize the ReadonlyREST plugin.
 - a. Shutdown API Gateway.
 - b. Open the command prompt from the location `SAG_Install_Dir/InternalDataStore/bin`
 - c. Run the following command:

```
elasticsearch-plugin.bat install  
file:///ReadonlyREST_plugin_zip_file_location_in_the_file_system
```
 - d. Type `y` when the installation procedure prompts for additional required permissions.

Installed readonlyrest message appears on successful installation.



The screenshot shows a Windows Command Prompt window with the title 'C:\Windows\System32\cmd.exe'. The window displays the output of the command 'elasticsearch-plugin.bat install file:///C:/readonlyrest-1.31.0_es7.13.0.zip'. The output includes a warning about the deprecation of JAVA_HOME, a progress bar indicating the download is at 100%, and a list of Java runtime permissions required by the plugin. It also provides a link for more information on permissions and asks if the user wants to continue with the installation. The command prompt ends with the path 'C:\SoftwareAG1011DEV\InternalDataStore\bin>'.

- e. Navigate to the `<SAGInstallDir>\InternalDataStore\config` folder and create an empty file with the name, `readonlyrest.yml` (in the same folder where `elasticsearch.yml` is available).
- f. Copy the folder `sagconfig` from `<SAGInstallDir>\IntegrationServer\instances\Instance_name\packages\WmAPIGateway\config\resources\elasticsearch` to `<SAGInstallDir>\InternalDataStore`.
- g. Copy the certificates **node-0-keystore.jks** and **truststore.jks** from `<SAGInstallDir>\InternalDataStore\sagconfig` folder to `<SAGInstallDir>\InternalDataStore\config` folder. Because, the keystore should be stored in the same directory where `elasticsearch.yml` and `readonlyrest.yml` is available.
- h. Patch the Readonly REST plugin by running the command -

```
java -jar <SAGInstallDir>/InternalDataStore/plugins/readonlyrest/ror-tools.jar  
patch --es-path <SAGInstallDir>/InternalDataStore
```

- i. Once the patching is successful, you can verify it by running the command -

```
java -jar <SAGInstallDir>/InternalDataStore/plugins/readonlyrest/ror-tools.jar  
verify --es-path <SAGInstallDir>/InternalDataStore
```

2. Protect API DataStore with two-way SSL and basic authorization.

The ReadonlyREST plugin supports plain-text credentials and hashed credentials for basic authorization. You can choose to add plain-text credentials or hashed credentials in the `readonlyrest.yml` file.

Plain-text credentials

Perform the following steps to include plain-text credentials for basic authorization in the `readonlyrest.yml` file.

- a. Open the `readonlyrest.yml` file from the `SAG_Install_Dir\InternalDataStore\config` folder and add the following text:

```
readonlyrest:  
  access_control_rules:  
  
    - name: "Require HTTP Basic Auth"  
      type: allow  
      auth_key: <plain_text_credentials>  
        (#For example: auth_key:Administrator:manage)  
  
  ssl:  
    enable: true  
    keystore_file: "node-0-keystore.jks"  
    keystore_pass: a362fbcce236eb098973  
    key_pass: a362fbcce236eb098973  
    client_authentication: true  
    truststore_file: "truststore.jks"  
    truststore_pass: 2c0820e69e7dd5356576
```

where access control rule name: *Require HTTP Basic Auth* is basic auth authentication, `auth_key` is the credentials (username and password) in the plain text.

Note:

Ensure that the content is indented properly as shown above, so that the YAML parser can parse them correctly.

Hashed credentials

The ReadonlyREST plugin supports obfuscating the credentials by hashing the credentials using *SHA256* algorithm. Use hashed credentials to keep the application secure.

Perform the following steps to include hashed credentials for basic authorization in the `readonlyrest.yml` file.

To generate the hash code for your password:

- a. Use the tool <https://xorbin.com/tools/sha256-hash-calculator> to hash your credentials. This generates the hash code.
- b. Replace the auth_key property in the **readonlyrest.yml** file with auth_key_sha256 property.
- c. Add the hashed credentials to the property auth_key_sha256 in **readonlyrest.yml** file located at *SAG_Install_Dir\InternalDataStore\config*.

```

readonlyrest:
  access_control_rules:
    - name: "Require HTTP Basic Auth"
      type: allow
      auth_key_sha256:
        927d5619ff87227be6ca8a2cc9ee68c11dd7a08d64d1e20bdc8d86254850b418

  ssl:
    enable: true
    keystore_file: "node-0-keystore.jks"
    keystore_pass: a362fbcce236eb098973
    key_pass: a362fbcce236eb098973
    client_authentication: true
    truststore_file: "truststore.jks"
    truststore_pass: 2c0820e69e7dd5356576

```

where access control rule name: *Require HTTP Basic Auth* is basic auth authentication, auth_key_sha256 is the credentials (username and password) in the hashed format and client_authentication is the two way SSL authentication. By default, this property is disabled. You can remove this property or set the value to false if you do not want the server or API Gateway Data Store to validate the client authentication.

- d. Save the configuration file **readonlyrest.yml**.

Note:

The keystore file and its passwords keystore_pass & key_pass are shipped out with API Gateway product by default. This may not be safe for production environment. For production setup, you can generate your own certificates (keystore and trustore) and configure in the **readonlyrest.yml** file. To generate your own certificates, see [" ReadonlyREST" on page 655](#).

- b. After adding plain-text credentials or hashed credentials in **readonlyrest.yml** file, open the **elasticsearch.yml** file from the *SAG_Install_Dir\InternalDataStore\config* folder and add the following configuration to the existing content and save the file.

```

http.type: ssl.netty4
xpack.security.enabled: false

```

This enables HTTPS connection for API Data Store.

3. Secure the inter-node communication.

In a clustered setup, to establish a secure communication between the API Gateway Data Store instances, perform the following steps:

- a. Open the **readonlyrest.yml** file from the *SAG_Install_Dir\InternalDataStore\config* folder and add the following configurations towards the end of the existing content and save the file. This configuration must be added to all the nodes which requires encrypted communication within cluster.

```
ssl_internode:  
  keystore_file: "node-0-keystore.jks"  
  keystore_pass: a362fbcce236eb098973  
  key_pass: a362fbcce236eb098973  
  client_authentication: true
```

The consolidated content of the **readonlyrest.yml** is as follows:

```
readonlyrest:  
  access_control_rules:  
  
    - name: "Require HTTP Basic Auth"  
      type: allow  
      auth_key_sha256:  
        927d5619ff87227be6ca8a2cc9ee68c11dd7a08d64d1e20bdc8d86254850b418  
  
      ssl:  
        enable: true  
        keystore_file: "node-0-keystore.jks"  
        keystore_pass: a362fbcce236eb098973  
        key_pass: a362fbcce236eb098973  
        client_authentication: true  
        truststore_file: "truststore.jks"  
        truststore_pass: 2c0820e69e7dd5356576  
  
      ssl_internode:  
        keystore_file: "node-0-keystore.jks"  
        keystore_pass: a362fbcce236eb098973  
        key_pass: a362fbcce236eb098973  
        client_authentication: true
```

- b. Open the **elasticsearch.yml** file from the *SAG_Install_Dir\InternalDataStore\config* folder and add the following configuration to the existing content and save the file.

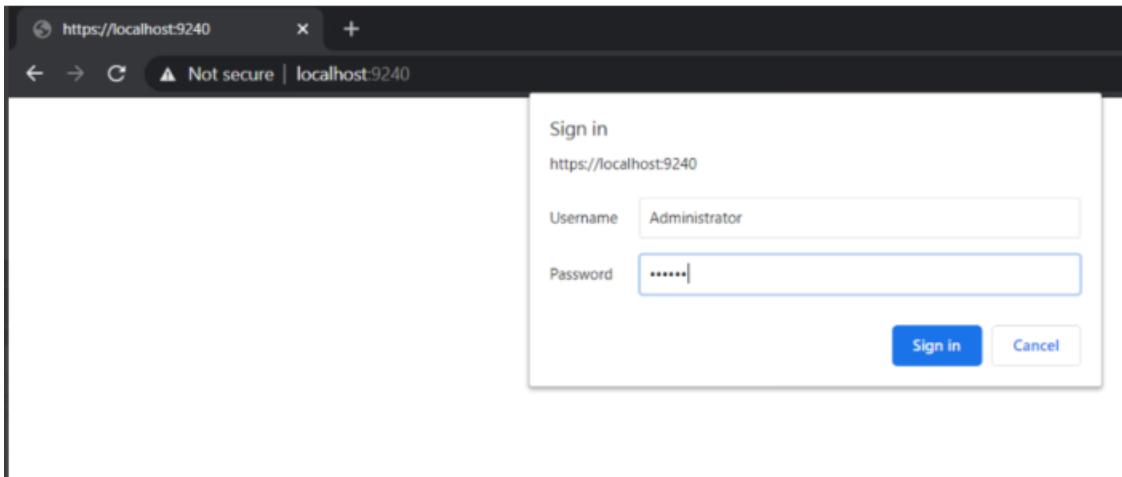
```
transport.type: ror_ssl_internode
```

This configuration must be added to all the nodes which requires encrypted communication within cluster.

- c. Shutdown and restart the API Data Store node and verify if it is protected by accessing the node in HTTPS URL with the given username and password.

```
https://<host>:<port>
```

API Data Store now runs on a secure channel on the HTTPS port and requests the basic authentication details.



4. Change the Kibana configuration to connect to API Data Store.
 - a. Open the **kibana.yml** file from the `SAG_Install_Dir\profiles\IS_instance_name\apigateway\dashboard\config` folder.
 - b. Remove the # symbol from the following properties and provide the corresponding values for the properties and save the file.
 - `elasticsearch.customHeaders.Authorization: "Basic QWRtaW5pc3RyYXRvcjptYW5hZ2U=`
 - `elasticsearch.ssl.verifyMode: certificate`
 - `elasticsearch.ssl.certificateAuthorities: <file path of your root-ca.pem certificate>`
 - `elasticsearch.ssl.keystore.path: "<SAGInstallDir>/InternalDataStore/config/node-0-keystore.p12"`
 - `elasticsearch.ssl.keystore.password: "password"`
 - `elasticsearch.ssl.truststore.path: "<SAGInstallDir>/InternalDataStore/config/truststore.p12"`
 - `elasticsearch.ssl.truststore.password: "password"`
 - `elasticsearch.ssl.alwaysPresentCertificate: true`
 - `elasticsearch.hosts: [https://<host>:<port>]`

Here, `QWRtaW5pc3RyYXRvcjptYW5hZ2U=` is **Base64** encoded format of username:password - **Administrator:manage**. See <https://forum.readonlyrest.com/t/kibana-error-401/1878> for more information.

Sample kibana.yml file is as follows:

```
server.port: 9405
server.host: "0.0.0.0"
server.basePath: "/apigatewayui/dashboardproxy"
elasticsearch.hosts: ["http://localhost:9240"]
```

```
elasticsearch.customHeaders.Authorization: "Basic  
QWRtaW5pcRyYXRvcjptYW5hZ2U="  
elasticsearch.ssl.verificationMode: certificate  
elasticsearch.ssl.certificateAuthorities:  
<SAG_Root>/InternalDataStore/sagconfig/root-ca.pem  
pid.file: kibana.pid  
console.enabled: false  
elasticsearch.ssl.keystore.path:  
"<SAGInstallDir>/InternalDataStore/config/node-0-keystore.p12"  
elasticsearch.ssl.keystore.password: "password"  
elasticsearch.ssl.truststore.path:  
"<SAGInstallRoot>/InternalDataStore/config/truststore.p12"  
elasticsearch.ssl.truststore.password:"password"  
elasticsearch.ssl.alwaysPresentCertificate:true
```

- c. Open the **uiconfiguration.properties** file from the *SAG_Install_Dir\profiles\IS_instance_name\apigateway\config* folder and set the property **apigw.kibana.autostart** to false.
- d. Generate a PKCS12 file from jks files for both node-0-keystore.jks and truststore.jks files. To generate the PKCS12 files run the following command -

```
keytool -importkeystore -srckeystore [MY_KEYSTORE.jks] -destkeystore [MY_FILE.p12] -srcstoretype JKS -deststoretype PKCS12 -deststorepass [PASSWORD_PKCS12]  
Example :  
keytool -importkeystore -srckeystore <SAG_Root>/InternalDataStore/config/node-0-keystore.jks -destkeystore node-0-keystore.p12 -srcstoretype JKS -deststoretype PKCS12 -deststorepass password  
keytool -importkeystore -srckeystore <SAG_Root>/InternalDataStore/config/truststore.jks -destkeystore truststore.p12 -srcstoretype JKS -deststoretype PKCS12 -deststorepass password
```

Once the files are generated, copy the files to *<SAGInstallDir>/InternalDataStore/config*.

5. Change the API Gateway configurations to connect to API Data Store.
 - a. Open the **config.properties** file from the *SAG_Install_Dir\IntegrationServer\instances\instance_name\packages\WmAPIGateway\config\resources\elasticsearch* folder.
 - b. Remove the # symbol from the following properties and provide the corresponding values for the properties and save the file.
 - pg.gateway.elasticsearch.http.username=Administrator
 - pg.gateway.elasticsearch.http.password=manage
 - pg.gateway.elasticsearch.https.truststore.filepath=<SAGInstallDir>/InternalDataStore/sagconfig/truststore.jks
 - pg.gateway.elasticsearch.https.truststore.password=2c0820e69e7dd5356576
 - pg.gateway.elasticsearch.https.enabled=true
 - pg.gateway.elasticsearch.https.keystore.filepath=<SAG_Root>/InternalDataStore/sagconfig/demouser-keystore.jks
 - pg.gateway.elasticsearch.https.keystore.password=6572b9b06156a0ff778c

- c. Start API Data Store.
- d. When API Data Store is up and running, start the Kibana server by running the **kibana.bat** file located at *SAG_Install_Dir\profiles\IS_default\apigateway\dashboard\bin*.
- e. Start API Gateway.

You can now log on, create APIs, and access the Analytics page without any challenge window for user credentials.

Configuring ReadonlyREST plugin with Self-generated certificates

As an API Provider, if you want to generate your own certificates to use with the ReadonlyREST plugin instead of the default certificates that are shipped with API Gateway, you can configure ReadonlyREST with user generated certificates. You have to perform this procedure if your organization does not have policies and procedures in place regarding the generation and use of digital certificates and certificate chains, including the use of certificates signed by a CA and you want to generate a self-signed certificate and import them into the keystore and truststore. Use **Java Keytool** to generate the required certificates for running ReadonlyREST in a production environment.

1. Shut down API Gateway, API DataStore, and Kibana.
2. Generate **keystore.jks** file.
 - a. Run the following command from *<Java_Install_Directory>/bin* to generate keystore certificate in the corresponding file location using Java Keytool:

```
keytool -genkey -alias alias_name -keyalg RSA -keystore
file_location\keystore.jks -storetype JKS
```

The keystore certificate is generated in the file location that you specify.

Example:

```
C:\Program Files\Java\jdk1.8.0_181\bin>keytool -genkey -alias test -keyalg RSA -keystore c:\work\apigateway\keystore.jks -storetype JKS
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]:
What is the name of your organizational unit?
[Unknown]:
What is the name of your organization?
[Unknown]:
What is the name of your City or Locality?
[Unknown]:
What is the name of your State or Province?
[Unknown]:
What is the two-letter country code for this unit?
[Unknown]:
Is CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown correct?
[no]: yes

Enter key password for <test>
      (RETURN if same as keystore password):
```

- b. Replace the generated keystore.jks file in the *SAG_Install_Dir\InternalDataStore\config* folder.
- c. Update the properties **keystore_file**, **keystore_pass** and **key_pass** in the **readonlyrest.yml** file located at *SAG_Install_Dir\InternalDataStore\config*.

Sample **readonlyrest.yml** file is as follows:

```
readonlyrest:
  access_control_rules:
    - name: "Require HTTP Basic Auth"
      type: allow
      auth_key: Administrator:manage

  ssl:
    enable: true
    keystore_file: "keystore.jks"
    keystore_pass: <Keystore pass>
    key_pass: <Key pass>
    client_authentication: true
    truststore_file: <<Truststore pass>>
    truststore_pass: <<Truststore pass>>
```

3. Export certificate from the keystore to configure in Kibana.yml file:

- a. Run the following command to export the certificate from the keystore and place it in the required location. :

```
keytool -export -alias alias_name -keystore keystore_file_location -rfc -file <filename>.cert
```

```
C:\Program Files\Java\jdk1.8.0_181\bin>keytool -export -alias test -keystore c:\work\apigateway\keystore.jks -rfc -file c:\work\apigateway\test.cert
Enter keystore password:
Certificate stored in file <c:\work\apigateway\test.cert>
```

The exported certificate is saved in the specified location.

- b. After exporting, update the exported certificate name and the file location of the exported certificate in **elasticsearch.ssl.certificateAuthorities** property in **kibana.yml** file located at *SAGInstallDir\profiles\IS_default\apigateway\dashboard\config*.

Example:

```
elasticsearch.ssl.certificateAuthorities: C:\work\apigateway\test.cert
```

4. Generate truststore and import the generated certificate.

- a. Run the following command to generate the truststore file and import the generated certificate into the truststore file.

```
keytool -import -alias alias_name -file Certificate_file_location
        -storetype JKS -keystore file_location_and_file_name_for_trustore.jks
```

```

E:\Program Files\Java\jdk1.8.0_181\bin>keytool -import -alias test -file c:\work\apigateway\test.cert -storetype JKS -keystore c:\work\apigateway\trustore.jks
Enter keystore password:
Re-enter new password:
Owner: CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
Issuer: CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
Serial number: 2817af92
Valid from: Fri Jul 24 08:35:44 IST 2020 until: Thu Oct 22 08:35:44 IST 2020
Certificate fingerprints:
        MD5: EB:10:2F:38:63:0C:CE:A5:9E:B0:5B:29:09:2C:70:FC
        SHA1: 21:6E:6B:19:75:0C:B9:89:77:BD:98:63:2B:CC:A5:35:9A:50:D4:A3:AB
        SHA256: A8:BE:FA:CB:B6:FF:13:5E:0C:73:49:5C:24:6B:3F:0F:13:7D:3C:4B:00:E7:AD:57:1B:35:9B:BF:5C:A3:B0:18
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: D5 DB C9 DB 3E 2B 47 97 48 06 D8 8A 63 50 8F DF ....>+G.8...cP..
0010: 30 75 AA DE 0u..
]
]

Trust this certificate? [no]: yes
Certificate was added to keystore

```

The trust store certificate is stored in the specified location. For example:

C:\work\apigateway\trustore.jks.

- Update the details of the generated truststore file in the following properties

- pg.gateway.elasticsearch.https.truststore.filepath
- pg.gateway.elasticsearch.https.truststore.password in the **config.properties** file located at *SAG_Install_Dir\IntegrationServer\instances\default\packages\WmAPIGateway\config\resources\elasticsearch*.

Example:

```

pg.gateway.elasticsearch.https.truststore.filepath=C:/work/apigateway/trustore.jks
pg.gateway.elasticsearch.https.truststore.password=your_trustore_password

```

- Start API Data Store.
- When API Data Store is up and running, start the Kibana server by running the **kibana.bat** file located at *SAG_Install_Dir\profiles\IS_default\apigateway\dashboard\bin*.
- Start API Gateway.

You can now log on, create APIs, and access the Analytics page without any challenge window for user credentials.

Creating a Custom Keystore with Self-Signed Certificates

You have to perform this procedure if your organization does not have policies and procedures in place regarding the generation and use of digital certificates and certificate chains, including the use of certificates signed by a CA but want to generate a self-signed certificate and import them into the keystore and truststore.

- Create a new keystore with a self-signed certificate.

- Run the following command, and provide the keystore password (for example, `manage`) and the other required details to generate a new key and store it in the specified keystore `https_keystore.jks`.

```
keytool -genkey -v -keystore https_keystore.jks
```

```
-alias HTTPS_KEYSTORE -keyalg RSA -keysize 2048 -validity 10000
```

Example:

```
C:\SoftwareAG\common\conf>keytool -genkey -v -keystore https_keystore.jks -alias HTTPS_KEYSTORE -keyalg RSA -keysize 2048 -validity 10000
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: user1
What is the name of your organizational unit?
[Unknown]: Software AG
What is the name of your organization?
[Unknown]: Software AG
What is the name of your City or Locality?
[Unknown]: Bangalore
What is the name of your State or Province?
[Unknown]: Karnataka
What is the two-letter country code for this unit?
[Unknown]: IN
Is CN=user1, OU=Software AG, O=Software AG, L=Bangalore, ST=Karnataka, C=IN correct?
[no]: y

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 10,000 days
        for: CN=user1, OU=Software AG, O=Software AG, L=Bangalore, ST=Karnataka,
        C=IN
Enter key password for <HTTPS_KEYSTORE>
      <RETURN if same as keystore password>:
[Storing https_keystore.jks]

Warning:
The JKS keystore uses a proprietary format. It is recommended to migrate to PKCS12 which is an industry standard format using "keytool -importkeystore -srckeystore https_keystore.jks -destkeystore https_keystore.jks -deststoretype pkcs12".
```

- b. Run the following command and provide the keystore password (for example, `manage`) to export the certificate from the keystore `https_keystore`, and place it in a specified location.

```
keytool -exportcert -v -alias HTTPS_KEYSTORE -file
Installation_Dir\common\conf\https_gateway.cer -keystore
Installation_Dir\common\conf\https_keystore.jks
```

Example:

```
C:\SoftwareAG\common\conf>keytool -exportcert -v -alias HTTPS_KEYSTORE -file C:\SoftwareAG\common\conf\https_gateway.cer -keystore C:\SoftwareAG\common\conf\https_keystore.jks
Enter keystore password:
Certificate stored in file <C:\SoftwareAG\common\conf\https_gateway.cer>

Warning:
The JKS keystore uses a proprietary format. It is recommended to migrate to PKCS12 which is an industry standard format using "keytool -importkeystore -srckeystore C:\SoftwareAG\common\conf\https_keystore.jks -destkeystore C:\SoftwareAG\common\conf\https_keystore.jks -deststoretype pkcs12".
```

The certificate `https_gateway.cer` is exported from the keystore `https_keystore` and placed in the location `Installation_Dir\common\conf\`.

2. Create a truststore and import the generated certificate.

- a. Run the following command to create a truststore file and import the generated certificate into the truststore file.

```
keytool -importcert -alias HTTPS_TRUSTSTORE -file
Installation_Dir\common\conf\https_gateway.cer -keystore
Installation_Dir\common\conf\https_truststore.jks
```

Example:

```
C:\SoftwareAG\common\conf>keytool -importcert -alias HTTPS_TRUSTSTORE -file C:\SoftwareAG\common\conf\https_gateway.cer -keystore C:\SoftwareAG\common\conf\https_truststore.jks
Enter keystore password:
Re-enter new password:
Owner: CN=user1, OU=Software AG, O=Software AG, L=Bangalore, ST=Karnataka, C=IN
Issuer: CN=user1, OU=Software AG, O=Software AG, L=Bangalore, ST=Karnataka, C=IN

Serial number: 413fa3dd
Valid from: Wed Apr 17 10:29:59 IST 2019 until: Sun Sep 02 10:29:59 IST 2046
Certificate fingerprints:
MD5: B7:CB:9C:49:AE:51:39:7B:DB:0A:27:19:1A:2C:D3:13
SHA1: 21:B9:36:AF:67:43:BE:11:22:D1:4B:DC:F7:AD:5D:63:E6:F6:E4:DC
SHA256: 91:86:8D:6F:BB:31:BB:F3:C1:51:FB:8D:D2:4D:92:22:C6:8F:C7:6C:DD:
1D:45:D2:10:A6:11:36:05:5F:0F:92
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: E6 2E D8 29 80 78 F2 C4    FB 90 C6 32 EC C8 24 DD  ...>.x.....2..$.
0010: 60 F6 41 BE               .A.
]
]

Trust this certificate? [no]: y
Certificate was added to keystore
```

A truststore file https_truststore.jks is created with the imported certificate.

You can now view the keystore and truststore files created and located at *Installation_Dir\common\conf*.

Troubleshooting Tips: Securing API Data Store (Elasticsearch)

During authentication a pop-up window appears when I try to view any dashboards in the Analytics tab

At times, when I view the **Analytics** tab, the Authentication pop-up window appears. This might be because Kibana is secured with SSL and API Gateway is unable to connect to Kibana. As of now, API Gateway does not support enabling SSL for Kibana. Kibana ports can be blocked from external access through firewall configuration.

Resolution:

Remove the following properties from *kibana.yml* file located at *SAGInstallDir\profiles\IS_default\apigateway\dashboard\config*:

- server.ssl.enabled: true
- server.ssl.cert: "/eip/apps/sag/InternalDataStore/config/PVWSLDWM001_pem.cer"

- server.ssl.key: "/eip/apps/sag/InternalDataStore/config/PVWSLDWM001_pem.key"

4 Container-based Provisioning

■ Docker Configuration	662
■ Kubernetes Support	687

Docker Configuration

Docker is an open-source technology that allows users to deploy applications to software containers. A Docker container is an instance of a Docker image, where the Docker image is the application, including the file system and runtime parameters.

You can create a Docker image from an installed and configured API Gateway instance and then run the Docker image as a Docker container. To facilitate running API Gateway in a Docker container, API Gateway provides a script to build a Docker image and then load or push the resulting Docker image to a Docker registry.

Support for API Gateway with Docker 18 and later is available on Linux and UNIX systems for which Docker provides native support.

For details on Docker and container technology, see [Docker documentation](#).

Docker security

Docker, by default, has introduced a number of security updates and features, which have made Docker easier to use in an enterprise. There are certain guidelines or best practices that apply to the following layers of the Docker technology stack, that an organization can look at:

- Docker image and registry configuration
- Docker container runtime configuration
- Host configuration

For detailed guidelines on security best practices, see the official Docker Security documentation at <https://docs.docker.com/engine/security/security/>.

Docker has also developed Docker Bench, a script that can test containers and their hosts' security configurations against a set of best practices provided by the Center for Internet Security. For details, see <https://github.com/docker/docker-bench-security>.

For details on how to establish a secure configuration baseline for the Docker Engine, see [Center for Information Security \(CIS\) Docker Benchmark](#) (Docker CE 17.06).

For information on the potential security concerns associated with the use of containers and recommendations for addressing these concerns, see [NIST SP 800](#) publication (Application Container Security Guide)

Prerequisites for Building a Docker Image

Prior to building a Docker image for API Gateway, you must complete the following:

- Install Docker client on the machine on which you are going to install API Gateway and start Docker as a daemon. The Docker client should have connectivity to Docker server to create images.
- Install API Gateway, packages, and fixes on a Linux or UNIX system using the instructions in [Installing Software AG Products](#), and then configure API Gateway and the hosted products.

Building the Docker Image for an API Gateway Instance

The API Gateway Docker image provides an API Gateway installation. Depending on the existing installation, the API Gateway Docker image provides a standard API Gateway or an advanced API Gateway instance. When running the image, the API Gateway is started. The API Gateway image is created on top of an Integration Server image.

➤ To build a Docker image for an API Gateway instance

1. Create a docker file for the Integration Server (IS) instance by running the following command:

```
./is_container.sh createDockerfile [optional arguments]
```

Argument	Description
-Dimage.name	<i>Optional.</i> Name of base image upon which the new image is built. Default: centos:7
-Dinstance.name	<i>Optional.</i> IS instance name to include in the image. Default: default
-Dport.list	<i>Optional.</i> Comma-separated list of the ports on the instance to expose in the image. Default: 5555,9999
-Dpackage.list	<i>Optional.</i> Comma-separated list of Wm packages on the instance to include in the image. Default: all (this includes all the Wm packages and the Default package)
-Dinclude.jdk	<i>Optional.</i> Whether to include the Integration Server JDK (true) or JRE (false) in the image. Default: true
-Dfile.name	<i>Optional.</i> File name for the generated docker file. Default: Dockerfile_IS

2. Build the IS Docker image using the Docker file Dockerfile_IS by running the following command:

```
./is_container.sh build [optional arguments]
```

Argument	Description
<code>-Dfile.name</code>	<i>Optional.</i> File name of the Docker file to use to build the Docker image. Default: Dockerfile_IS
<code>-Dimage.name</code>	<i>Optional.</i> Name of the generated Docker image. Default: is:micro

3. Create a Docker file for the API Gateway instance from the IS image is:micro by running the following command:

```
./apigw_container.sh createDockerfile [optional arguments]
```

Argument	Description
<code>--instance.name</code>	<i>Optional.</i> API Gateway instance to include in the image. Default: default
<code>--port.list</code>	Comma-separated list of the ports on the instance to expose in the image. Default: 9072
<code>--base.image</code>	Name of the base Integration Server image upon which this image should be built. Default: is:micro
<code>--file.name</code>	<i>Optional.</i> File name for the generated Docker file. Default: Dockerfile_IS_APIGW
<code>--target.configuration</code>	<i>Optional.</i> Target configuration for which Dockerfile is created. Not specifying any value builds a Dockerfile for the Docker and Kubernetes environments. Specifying the value OpenShift builds a Dockerfile for an OpenShift environment.
Note: If you specify the --target.configuration option, the Integration Server image specified by the --base.image option should be available before you create the API Gateway Dockerfile. The Integration Server Docker image is analyzed with docker inspect in order to extract some information necessary for the API Gateway Dockerfile.	

Argument	Description
--os.image	<p><i>Optional.</i> Name of the base operating system image upon which this image is built if the --target.configuration is set to OpenShift.</p> <p>Default: centos:7</p>
	<p>Note: The value of this parameter has to be aligned with the one specified for -Dimage.name in Step 1.</p>

The Docker file is created under the packages directory of the specified Integration Server instance. In a default installation, the Docker file is created in the folder `SAG_Root/IntegrationServer/instances/default/packages/Dockerfile_IS_APIGW`.

- Build the API Gateway Docker image using the core Docker file `Dockerfile_IS_APIGW` by running the following command:

```
./apigw_container.sh build [optional arguments]
```

Argument	Description
<code>instance.name</code>	<p>Optional. API Gateway instance to include in the image.</p> <p>Default: default</p>
<code>file.name</code>	<p>File name of the Docker file to use to build the Docker image.</p> <p>Default: <code>Dockerfile_IS_APIGW</code></p>
<code>image.name</code>	<p>Optional. Name for the generated Docker image that contains the custom packages.</p> <p>Default: <code>is:apigw</code></p>

The image is stored in the local registry of the Docker host. To check the image, run the following command:

```
$ docker images
```

Example

A sample shell script for creating an API Gateway Docker image looks as follows:

```
echo "is createDockerfile ====="
./is_container.sh createDockerfile
status=$?

if [ $status -ne 0 ]
then
    echo "Failed! status: $status"
```

```

    exit $status
fi

echo "is build ====="
./is_container.sh build
status=$?

if [ $status -ne 0 ]
then
    echo "Failed! status: $status"
    exit $status
fi

echo "apigw createDockerfile ====="
./apigw_container.sh createDockerfile
status=$?

if [ $status -ne 0 ]
then
    echo "Failed! status: $status"
    exit $status
fi

echo "apigw build ====="
./apigw_container.sh build
status=$?

if [ $status -ne 0 ]
then
    echo "Failed! status: $status"
    exit $status
fi

```

After running the steps, the created images can be listed using the command `docker images`. The following sample result shows the base image `centos:7`, the Integration Server image `is:micro`, and the API Gateway image `is:apigw`.

REPOSITORY	TAG	IMAGEID	CREATED	SIZE
is	apigw	af29373fc98a	15 hours ago	1.3GB
is	micro	06e7c0de4807	15 hours ago	1.1GB
centos	7	36540f359ca3	12 days ago	193MB

Note:

The `is:micro` and therefore, the `is:apigw` images are based on the `centos:7` image, which is available from the official CentOS repository

The Docker images resulting from Docker files created using the `createDockerFile` command feature the following:

- **Docker logging.**

API Gateway Docker containers log to `stdout` and `stderr`. The API Gateway logs can be fetched with Docker logs.

- **Docker health check.**

API Gateway Docker containers perform health checks. You can use `wget` request against the API Gateway REST API to check the health status of API Gateway.

The following wget request shows a curl invocation sending a request against the HTTP port. If API Gateway exposes an HTTPS port, only the wget is created accordingly. The option --no-check-certificate is used to avoid any failure due to certificate problems.

```
HEALTHCHECK CMD curl --no-check-certificate
http://localhost:5555/rest/apigateway/health
```

The wget checks the API Gateway availability by sending requests to the API Gateway REST health resource. If the wget is successful, API Gateway is considered healthy.

- **Graceful shutdown.**

Docker stop issues a SIGTERM to the running API Gateway.

Retrieving Port Information of the API Gateway Image

- To retrieve the port information of the API Gateway image (is:apigw), run the following command :

```
docker inspect --format='{{range $p,
$conf := .Config.ExposedPorts}}
{{$p}} {{end}}' is:apigw
```

A sample output looks as follows:

```
5555/tcp 9072/tcp 9999/tcp
```

Running the API Gateway Container

Before starting API Gateway, ensure that the main memory and the kernel settings of your docker host are correctly configured. The docker host should provide at least 4 GB of main memory. Since API Gateway comes with an Elasticsearch, the `vm.max_map_count` kernel setting needs to be set to at least 262144. You can change the setting on your docker host by running the following command:

```
sysctl -w vm.max_map_count=262144
```

For further details about the important system settings to be considered, see the *Elasticsearch documentation*.

- Start the API Gateway image using the docker run command:

```
docker run -d -p 5555:5555 -p 9072:9072 -name apigw is:apigw
```

The docker run is parameterized with the IS and the webApp port exposed by the Docker container. If you have configured different ports for IS and UI, the call has to be adapted accordingly. The name of the container is set to apigw.

The status of the Docker container can be determined by running the docker ps command:

```
docker ps
```

A sample output looks as follows:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	->

```
5b95c9badd59  is:apigw  "/bin/sh -c 'cd /s..."  15 hours ago  Up 15 hours
->
PORTS          NAMES
0.0.0.0:5555->5555/tcp, 0.0.0.0:9072->9072/tcp, 9999/tcp  apigw
```

Load Balancer Configuration with the Docker Host

A port mapping is specified when you run the Docker container. For example, to map the IS port to the port 5858 on the Docker host, run the Docker image with the following command:

```
docker run -d -p 5858:5555 -p 9073:9072 --name apigw is:apigw
```

The host and the port within the Docker container are different from the host running the Docker container and the port exposed on the host. As a result, the gateway endpoints exposed by API Gateway are set incorrectly. To set this right you have to set up a load balancer configuration with the Docker host and the mapped ports.

For the above example, the following load balancer URLs are required:

- **Load balancer URL (HTTP):** http://dockerhost:5858
- **Load balancer URL (WS):** ws://dockerhost:5858
- **Web application load balancer URL:** http://dockerhost:9073

Note:

If the API Gateway UI port is mapped to a different port on the Docker host, the API Gateway solution link in the IS Administration UI does not work.

Stopping the API Gateway Container

- Stop the API Gateway container using the docker stop command:

```
docker stop -t90 apigw
```

The docker stop is parameterized with the number of seconds required for a graceful shutdown of the API Gateway and the API Gateway Docker container name.

Note:

The docker stop does not destroy the state of the API Gateway. On restarting the Docker container all assets that have been created or configured are available again.

Managing API Gateway Images

You can manage the API Gateway images using the is_container.sh script

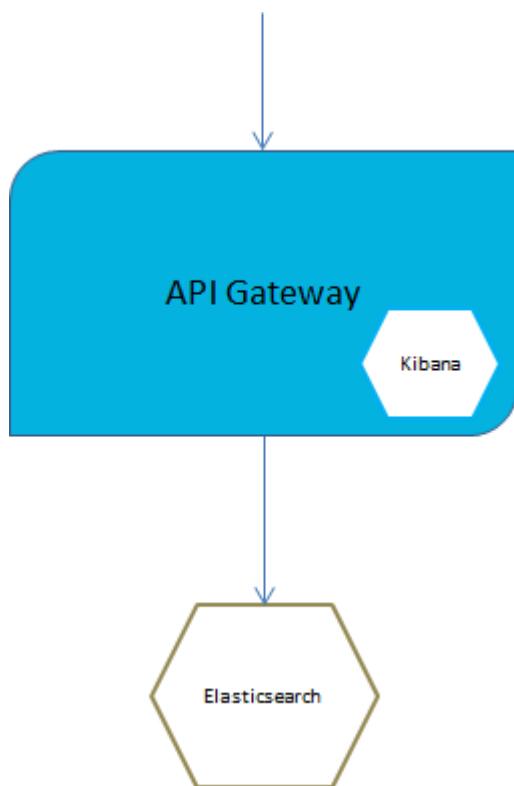
- **saveImage:** To save an API Gateway image to a file (creating a tar ball from an image)
- **loadImage:** To load an image to a Docker registry (loading an image into a Docker registry from tar ball)

API Gateway Docker Container with Externalized Elasticsearch and Kibana

The best practices for Docker container specify having a single process per container. This allows to control the components of an API Gateway container and enables horizontal scaling. A full split results into three separate containers, one each for API Gateway, Elasticsearch and Kibana. Since Kibana is not scaled independently it can be included into the API Gateway container.

API Gateway Container with an Externalized Elasticsearch

The following figure depicts an API Gateway container with an externalized Elasticsearch where Kibana is included in the API Gateway container.



Do the following to set up API Gateway container with an external Elasticsearch:

1. Run the external Elasticsearch.

You can start Elasticsearch container by using the Elasticsearch Docker image available on docker hub. The Elasticsearch version should be the same as used in API Gateway.

```

docker run -p 9200:9240 -p 9300:9340 -e "xpack.security.enabled=false"
-v es-data:/usr/share/elasticsearch/data
docker.elastic.co/elasticsearch/elasticsearch:8.2.3
  
```

Use the option `-e xpack.security.enabled=false` to disable basic authentication for Elasticsearch. This is the default option available in API Gateway.

Use the volume mapping `-v es-data:/usr/share/elasticsearch/data` to persist the Elasticsearch data outside the Docker container.

2. Run API Gateway Docker container.

To create a Docker file or image for an API Gateway that does not contain Elasticsearch the `./apigw_container.sh createDockerFile` and build command offer the following option:

```
--extern_ES
```

Setting the flag ensures that the InternalDataStore is not added to the Docker image created by the generated Docker file.

Elasticsearch configuration can be injected into an existing API Gateway image. Assuming an existing API Gateway image `sag:apigw`:

```
docker run -d -p 5555:5555 -p 9072:9072 --env-file apigw-env.list  
--hostname apigw --name apigw sag:apigw
```

The `apigw-env.list` contains the environment variables required for configuring an external Elasticsearch and External Kibana:

```
apigw_elasticsearch_hosts=host:port  
apigw_elasticsearch_https_enabled=("true" or "false")  
apigw_elasticsearch_http_username=user  
apigw_elasticsearch_http_password=password
```

An example looks as follows:

```
apigw_elasticsearch_hosts=testhost1:9200  
apigw_elasticsearch_https_enabled=false  
apigw_elasticsearch_http_username=  
apigw_elasticsearch_http_password=
```

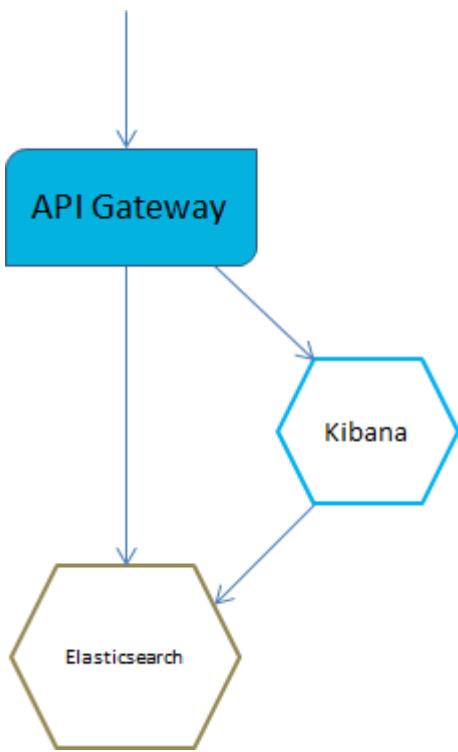
You can specify the Elasticsearch properties to modify the property files on the container startup.

Instead of using the env file to change the environment variables, you can set them using `-e` options in the Docker run. For setting the Elasticsearch host the Docker run command looks as follows:

```
docker run -d -p 5555:5555 -p 9072:9072 \  
-e apigw_elasticsearch_hosts=testhost1:9200 \  
--hostname apigw \  
--name apigw sag:apigw
```

API Gateway Container with an External Elasticsearch and External Kibana

The following figure depicts an API Gateway container with external Elasticsearch and external Kibana containers.



Do the following to set up API Gateway container with an external Elasticsearch and external Kibana:

1. Run the external Elasticsearch.

You can start Elasticsearch by using the default Elasticsearch Docker image available on docker hub. The Elasticsearch version should be the same as used in API Gateway.

```
docker run -p 9200:9240 -p 9300:9340 -e "xpack.security.enabled=false"
-v es-data:/usr/share/elasticsearch/data
docker.elastic.co/elasticsearch/elasticsearch:8.2.3
```

Use the option `-e xpack.security.enabled=false` to disable basic authentication for Elasticsearch. This is the default option available in API Gateway.

Use the volume mapping `-v es-data:/usr/share/elasticsearch/data` to persist the Elasticsearch data outside the Docker container.

2. Run the external Kibana

If you have modified the original Kibana, for example by adding a style sheet file, or modified the `kibana.yml` file, as per your requirements, then this customization of Kibana is bundled with API Gateway. This customized Kibana is provided under the directory: `profiles/IS_default/apigateway/dashboard`. To achieve this, create and run a Docker image based on the customization. This can be achieved by a Docker file as follows:

```
FROM centos:7
COPY /opt/softwareag/profiles/IS_default/apigateway/dashboard /opt/softwareag/kibana
EXPOSE 9405
RUN chmod 777 /opt/softwareag/kibana/bin/kibana
CMD /opt/softwareag/kibana/bin/kibana
```

Build and run the Docker file as follows:

```
docker build -t sagkibana .
docker run -p 9405:9405 sagkibana
```

3. Run API Gateway Docker container

To run a Docker image for an API Gateway running against an external Kibana the Docker run can be called with the following environment variable:

```
apigw_kibana_dashboardInstance=instance
```

The environment variable can be added to an env file. The env file for running a Docker container with external Elasticsearch and external Kibana looks as follows:

```
apigw_elasticsearch_hosts=testhost1:9200
apigw_elasticsearch_http_username=
apigw_elasticsearch_http_password=
apigw_kibana_dashboardInstance=http://testhost1:9405
```

Note:

All the configurations supported through externalized API Gateway configuration can be configured through environment variables.

API Gateway Container Cluster Configuration

You can combine API Gateway Docker containers to form a cluster.

To configure an API Gateway Docker container cluster:

1. Configure loadbalancer on the Docker host.

The custom loadbalancer is installed on the Docker host. For more details on setting up the load balancer, see “[API Gateway Cluster Configuration](#)” on page 26.

2. Configure Terracotta Server Array.

Note:

This configuration step is required only if your API Gateway cluster uses TSA.

API Gateway requires a Terracotta Server Array installation. For details, see *webMethods Integration Server Clustering Guide* and Terracotta documentation (<https://www.terracotta.org/>). The Terracotta Server Array on its own can be deployed as a Docker container.

3. Create the basic API Gateway Docker image.

For details on creating the API Gateway Docker image, see “[Building the Docker Image for an API Gateway Instance](#)” on page 663.

4. Create cluster API Gateway Docker image and enhance it with the cluster configuration in one of the following ways:

- Clustered all-in-one containers that consist of API Gateway, API Data Store (Elasticsearch), and Kibana.

- Clustered API Gateway containers with externalized Elasticsearch and Kibana containers.

Clustered all-in-one containers that consist of API Gateway, Kibana and API Data Store

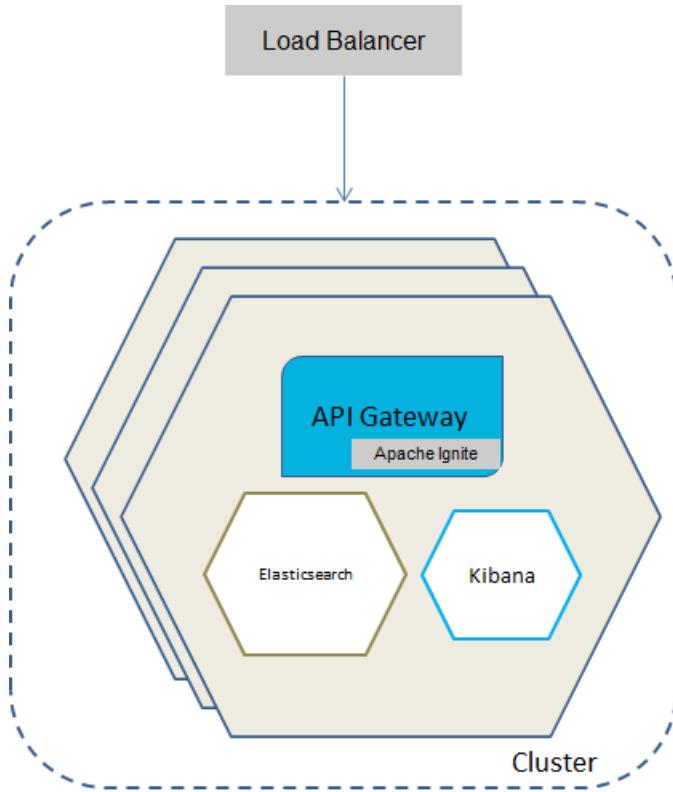
Although API Gateway clusters with externalized Elasticsearch is the preferred approach, you can also cluster API Gateway all-in-one containers. The clustering can be configured using Apache Ignite or Terracotta Server Array.

Note:

Having external Kibana is an optional variation.

Peer-to-peer clustering using Apache Ignite

The following diagram depicts peer-to-peer clustering using Apache Ignite.



The all-in-one containers hold API Gateway, Kibana, and Elasticsearch. The clustering is done using Apache Ignite and the cluster capabilities of the embedded Elasticsearch instances.

Inject the required settings for the cluster configuration during Docker run through an environment file.

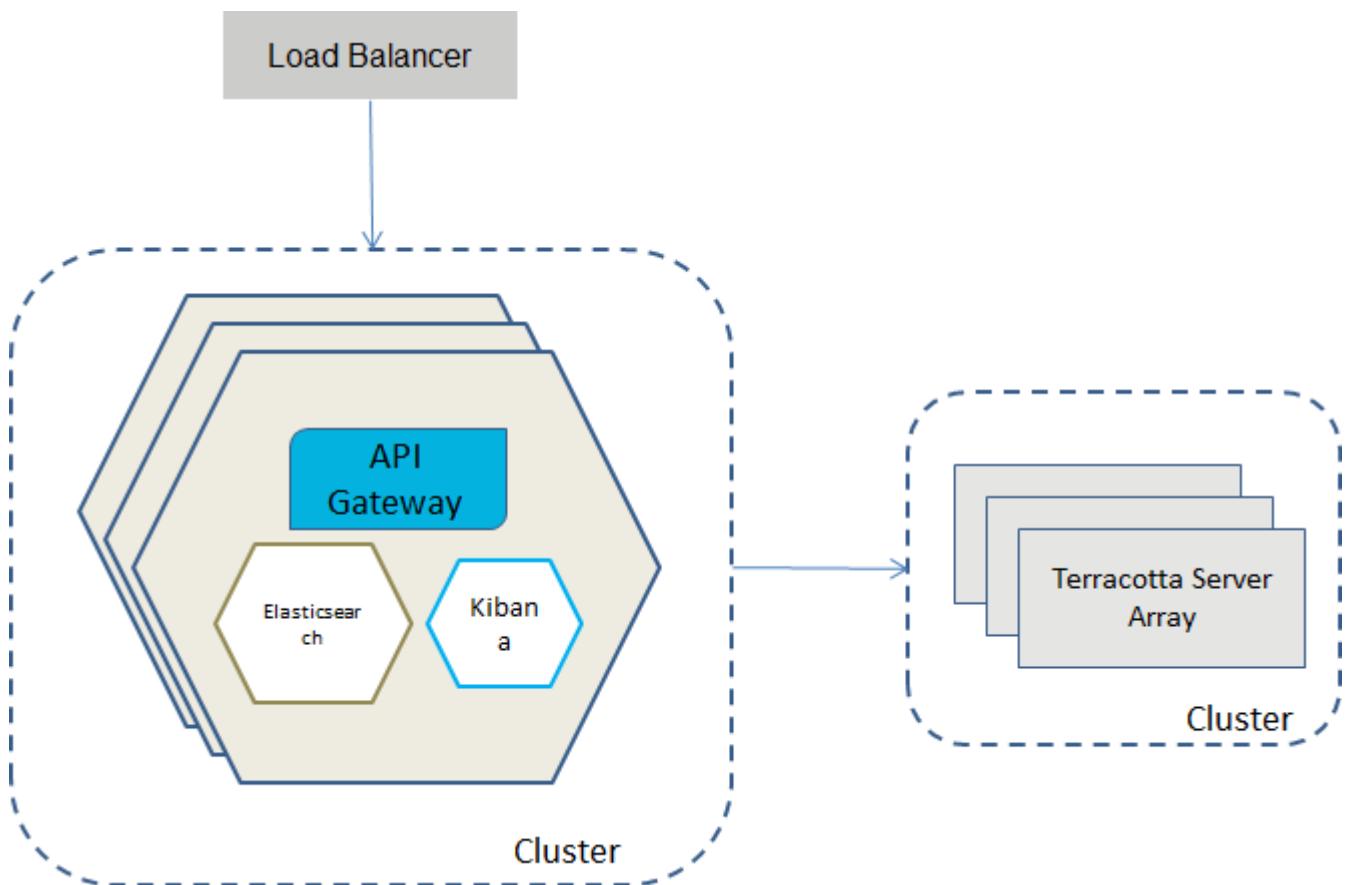
A sample environment file when clustering is done through Apache Ignite looks as follows.

```
apigw_cluster_aware=true
apigw_cluster_name=APIGatewayCluster
apigw_cluster_ignite_hostnames=apigw1,apigw2,apigw3
apigw_cluster_ignite_discoveryPort=10100
```

```
apigw_cluster_ignite_communicationPort=10400
```

Clustering using Terracotta Server Array

The following diagram depicts clustering using TSA.



The all-in-one containers hold API Gateway, Kibana, and Elasticsearch. The clustering is done using a TSA and the cluster capabilities of the embedded Elasticsearch instances.

Inject the required settings for the cluster configuration during Docker run through an environment file.

A sample environment file when clustering is done through Terracotta server array looks as follows.

```
apigw_cluster_tsaUrls=tc:9510
apigw_terracotta_license_filename=terracotta-license.key
apigw_cluster_discoverySeedHosts=apigw1:9340,apigw2:9340,apigw3:9340
apigw_cluster_initialMasterNodes=apigw1_master
```

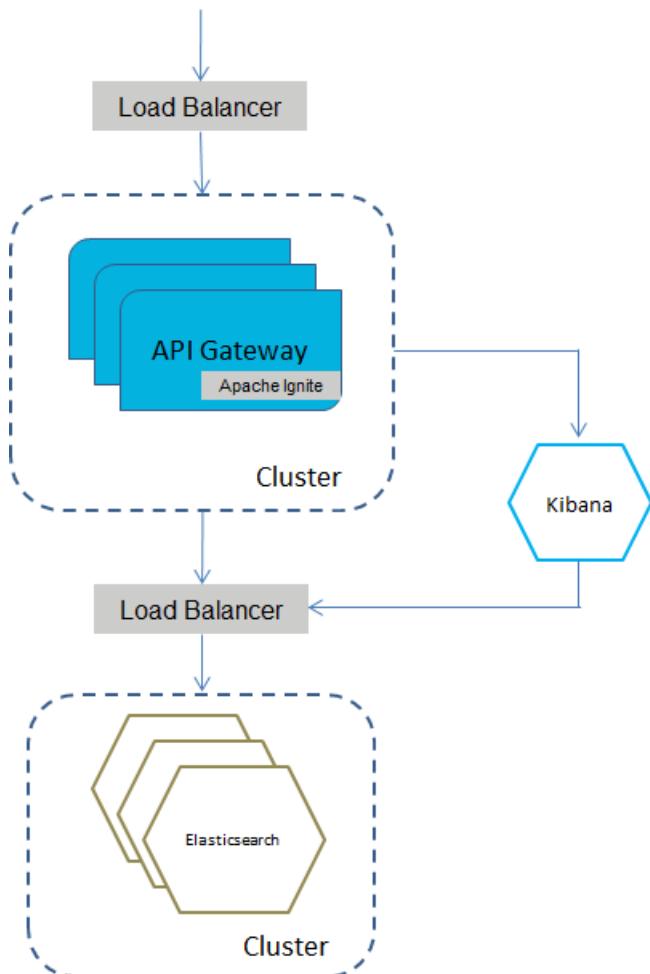
Clustered API Gateway containers with externalized Elasticsearch and Kibana containers

The API Gateway cluster can be peer-to-peer, or based on TSA, and it communicates to a cluster of Elasticsearch containers through a load balancer. The Elasticsearch load balancer also provides the Elasticsearch endpoint for the Kibana containers.

Note:

The externalized Kibana is optional. You can still run Kibana within the API Gateway container.

Peer-to-peer clustering using Apache Ignite



To cluster API Gateway with external containers for Elasticsearch, Kibana, and Apache Ignite, you can inject the settings into an API Gateway Docker image when starting, by providing an environment file. The environment file has to define the following environment variables.

```

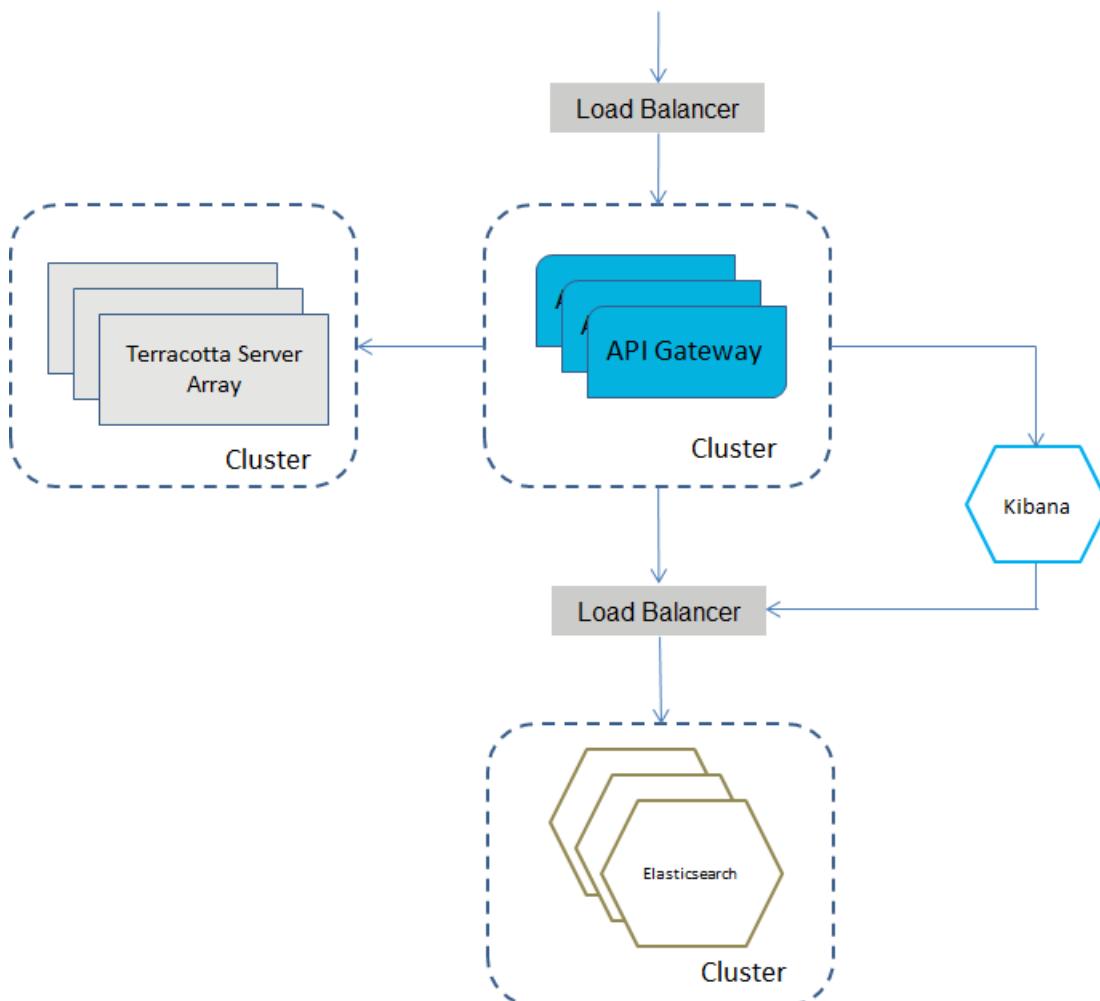
apigw_cluster_aware=true
apigw_cluster_name=name
apigw_cluster_ignite_hostnames=comma-separated list of host names
apigw_cluster_ignite_discoveryPort=port
apigw_cluster_ignite_communicationPort=port
apigw_elasticsearch_hosts=host:port
  
```

```
apigw_elasticsearch_http_username=user
apigw_elasticsearch_http_password=password
apigw_kibana_dashboardInstance=instance
```

A sample assignment of environment variables looks as follows:

```
apigw_cluster_aware=true
apigw_cluster_name=APIGatewayCluster
apigw_cluster_ignite_hostnames=apigw1,apigw2,apigw3
apigw_cluster_ignite_discoveryPort=10100
apigw_cluster_ignite_communicationPort=10400
apigw_elasticsearch_hosts=testhost1:9200
apigw_elasticsearch_http_username=
apigw_elasticsearch_http_password=
apigw_kibana_dashboardInstance=http://testhost1:9405
```

Clustering using Terracotta Server Array



To cluster the API Gateway with external containers for Elasticsearch, Kibana, and TSA, you can inject the settings into an API Gateway Docker image when starting by providing an environment file. The environment file has to define the following environment variables.

```
apigw_cluster_tsaUrls=host:port
```

```
apigw_terracotta_license_filename=license-key-filename
apigw_elasticsearch_hosts=host:port
apigw_elasticsearch_http_username=user
apigw_elasticsearch_http_password=password
apigw_kibana_dashboardInstance=instance
```

A sample assignment of the environment variables looks as follows.

```
apigw_cluster_tsaUrls=tc:9510
apigw_terracotta_license_filename=terracotta-license.key
apigw_elasticsearch_hosts=testhost1:9200
apigw_elasticsearch_http_username=
apigw_elasticsearch_http_password=
apigw_kibana_dashboardInstance=http://testhost1:9405
```

Running API Gateway Docker Containers with Docker Compose

You can run API Gateway Docker containers and use Docker Compose's ability to allow you to define and run multi-container Docker applications in your deployment environment.

The API Gateway installation provides sample Docker Compose files in the folder located at `SAG_Root/IntegrationServer/instances/default/packages/WmAPIGateway/resources/samples/docker-compose`. The API Gateway installation provides the following three sample Docker Compose files:

- **apigw-elasticsearch-no-cluster.yml** : An API Gateway instance with an Elasticsearch container.
- **apigw-elasticsearch-cluster.yml** : An API Gateway cluster with three API Gateway containers, three clustered Elasticsearch containers and a Terracotta container.
- **apigw-elasticsearch-cluster-kibana.yml** : Containers of an API Gateway cluster and a Kibana container.

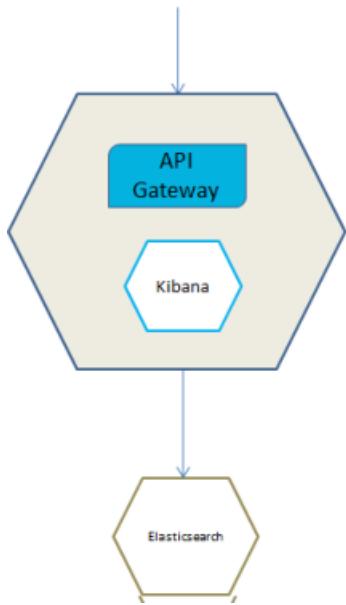
The Docker Compose files can be parameterized through environment variables.

For more Docker configuration samples, see <https://github.com/SoftwareAG/webmethods-api-gateway/tree/master/samples/docker>.

Running a Single API Gateway and an Elasticsearch Container

You can run a single API Gateway and an Elasticsearch container using Docker Compose. In this deployment scenario you can use the sample Docker Compose file `apigw-elasticsearch-no-cluster.yml`.

The following figure depicts an API Gateway container with an externalized Elasticsearch where Kibana is included in the API Gateway container.



➤ To deploy a single API Gateway and an Elasticsearch container

1. Set the environment variables to define the image for the API Gateway container as follows:

```
export APIGW_DOCKER_IMAGE_NAME=image name or filepath location of an existing image
```

The composite file requires an API Gateway Docker image. You can create the referenced image through API Gateway scripting. For details on creating a Docker image, see “[Building the Docker Image for an API Gateway Instance](#)” on page 663. The Docker Compose file references the standard Elasticsearch 8.2.3 image:
<https://www.docker.elastic.co/r/elasticsearch/elasticsearch:8.2.3>.

Specify the API Gateway image by changing the .env file. API Gateway uses the .env file when the working directory is .../samples/docker-compose, else you must specify the environment variables.

2. Run the following command to start the API Gateway Docker container and the Elasticsearch container using the Docker Compose sample file:

```
cd SAG-Root/IntegrationServer/instances/default/packages/WmAPIGateway/resources/
samples/docker-compose
docker-compose -f apigw-elasticsearch-no-cluster.yml up
```

In the Docker Compose sample file apigw-elasticsearch-no-cluster.yml ensure that you have specified the required information such as image name, name and port of the Elasticsearch host, server port, and UI port. This creates and starts the containers. Run the docker ps command to view the details of the containers created.

To run it in the detached mode, append -d in the docker-compose command.

Note:

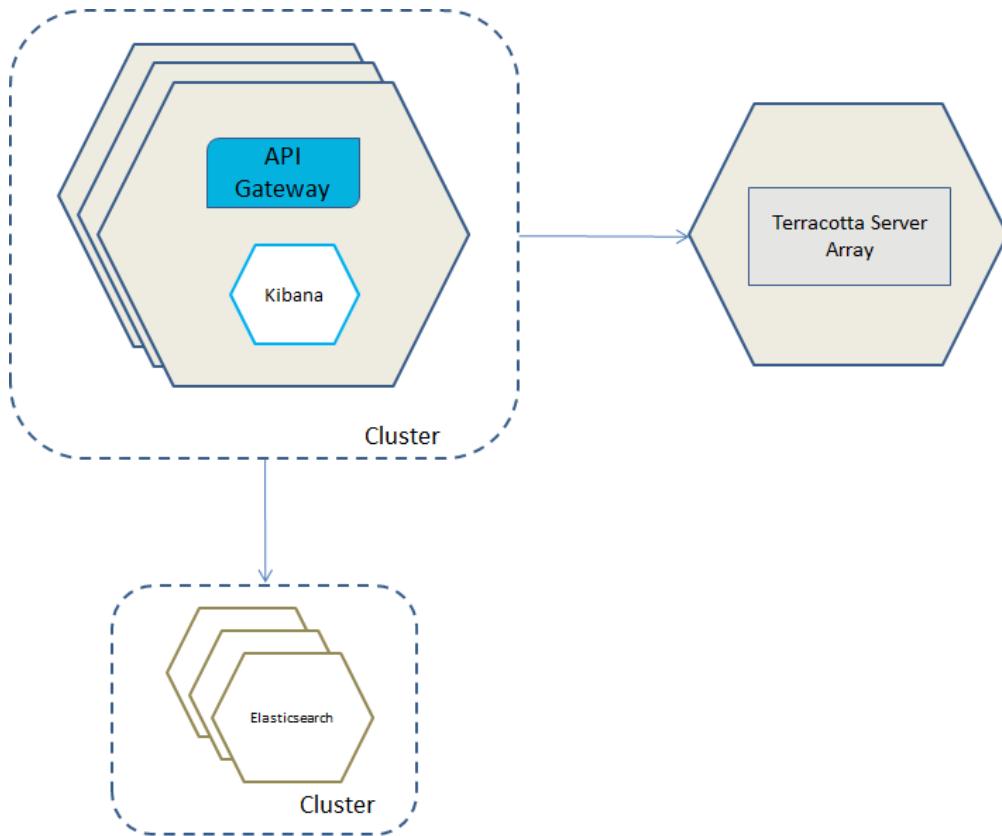
You can stop the API Gateway Docker container and the Elasticsearch container using the Docker Compose sample file with the following command:

```
docker-compose -f apigw-elasticsearch-no-cluster.yml down
```

Running Clustered API Gateway Containers and Elasticsearch Containers

In this deployment scenario you can use the sample Docker Compose file apigw-elasticsearch-cluster.yml.

The following diagram depicts a set-up that has clustered API Gateway containers and Elasticsearch containers.



➤ To run clustered API Gateway containers and Elasticsearch containers

1. Set the environment variables to define image for the API Gateway Docker container and Terracotta as follows:

```
export APIGW_DOCKER_IMAGE_NAME=image name or filepath location of an existing image
export TERRACOTTA_DOCKER_IMAGE_NAME=terracotta image name
```

The composite file requires Terracotta and the API Gateway Docker image. You can create the API Gateway image through API Gateway scripting. For details on creating a Docker image, see “[Building the Docker Image for an API Gateway Instance](#)” on page 663.

You can create the Terracotta image as follows:

```
cd /opt/softwareag  
docker build --file Terracotta/docker/images/server/Dockerfile -tag is:tc
```

Specify the API Gateway image by changing the .env file. API Gateway uses the .env file when the working directory is .../samples/docker-compose, else you must specify the environment variables.

2. Run the following command to start Terracotta, clustered API Gateway, and Elasticsearch containers using the Docker Compose sample file:

```
cd SAG-Root/IntegrationServer/instances/default/packages/WmAPIGateway  
/resources/samples/docker-compose  
docker-compose -f apigw-elasticsearch-cluster.yml up
```

In the Docker Compose sample file apigw-elasticsearch-cluster.yml ensure that you have specified the required information such as image name, name and port of the Elasticsearch host, server port, and UI port. This creates and starts the containers. Run the docker ps command to view the details of the containers created.

To run it in the detached mode, append -d in the docker-compose command.

Note:

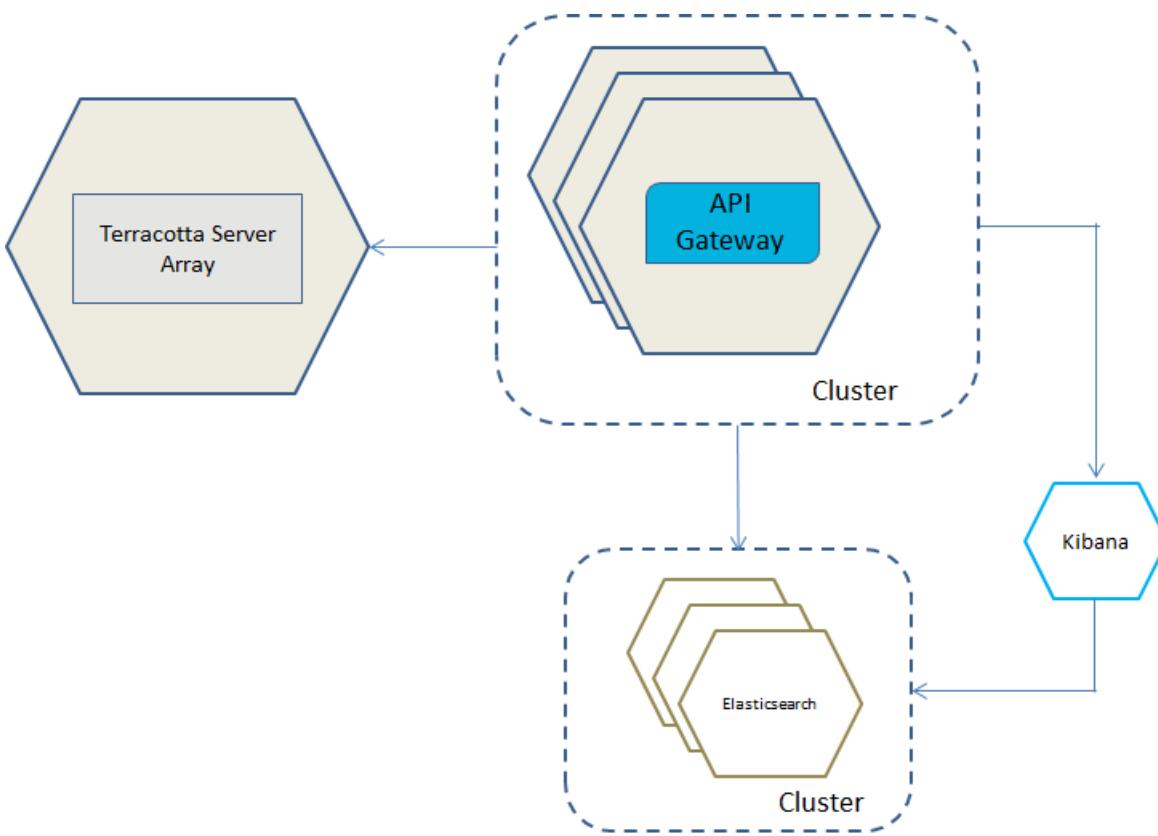
You can stop the API Gateway Docker container and the Elasticsearch container using the Docker Compose sample file with the following command:

```
docker-compose -f apigw-elasticsearch-cluster.yml down
```

Running Clustered API Gateway and Elasticsearch Containers and a Kibana Container

In this deployment scenario you can use the sample Docker Compose file apigw-elasticsearch-cluster-kibana.yml.

The figure depicts clustered API Gateway containers. They are talking to a clustered Terracotta Server Array container, a cluster of Elasticsearch container and an external Kibana.



➤ To run clustered API Gateway and Elasticsearch containers, and a Kibana container

1. Set the environment variables to define the API Gateway, Terracotta, and the Kibana image as follows:

```
export APIGW_DOCKER_IMAGE_NAME=image name or filepath location of an existing image
export TERRACOTTA_DOCKER_IMAGE_NAME=terracotta image name
export KIBANA_DOCKER_IMAGE_NAME=kibana image name
```

You can create the required API Gateway Docker image through API Gateway scripting. For details on creating a Docker image, see [“Building the Docker Image for an API Gateway Instance” on page 663](#).

Create the Terracotta image as follows:

```
cd /opt/softwareag
docker build --file Terracotta/docker/images/server/Dockerfile -tag is:tc
```

Specify the API Gateway image by changing the .env file. API Gateway uses the .env file when the working directory is .../samples/docker-compose, else you must specify the environment variables..

API Gateway requires a customized Kibana image. The Docker file for creating the Kibana image is as follows:

```
FROM centos:7
COPY /opt/softwareag/profiles/IS_default/apigateway/dashboard /opt/softwareag/kibana
```

```
EXPOSE 9405
RUN chmod 777 /opt/softwareag/kibana/bin/kibana
CMD /opt/softwareag/kibana/bin/kibana
```

- Run the following command to start the API Gateway Docker container and the Elasticsearch container using the Docker Compose sample file:

```
cd SAG-Root/IntegrationServer/instances/default/packages/WmAPIGateway/resources/
samples/docker-compose
docker-compose -f apigw-elasticsearch-cluster-kibana.yml up
```

In the Docker Compose sample file apigw-elasticsearch-cluster-kibana.yml ensure that you have specified the required information such as image name, name and port of the Elasticsearch host, server port, UI port, and Kibana dashboard instance details. This creates and starts the containers. Run the docker ps command to view the details of the containers created.

To run it in the detached mode, append -d in the docker-compose command.

Note:

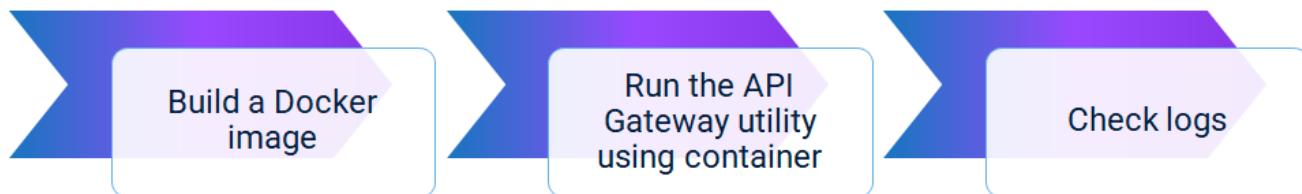
You can stop the API Gateway Docker container and the Elasticsearch container using the docker-compose sample file with the following command:

```
docker-compose -f apigw-elasticsearch-cluster-kibana.yml down
```

Data Backup using Docker Container

You can perform API Data Store backup using Docker container by running the **apigatewayutil** utility container. You can run the container with the required command to perform a backup operation.

The process of backing up data using Docker container involves the following steps:



Building Docker Image on top of the public API Gateway Image

You must first build a Docker image to run the API Gateway utility. You can perform any backup operation inside the container.

You can either build a image on top of the public API Gateway image or on top of your local API Gateway installation.

➤ To build a Docker image:

1. Check out the following folder to your local machine, `webmethods-api-gateway/samples/datamanagement/APIGatewayUtility/`.
2. Run the following command:

```
docker build . -t <tag_name>
```

For example,

```
docker build . -t apigwimage
```

3. Run the following command to verify that the image is listed with the name provided in the previous step:

```
docker image ls
```

The list of available images, including the new image, appears.

Building Docker Image on top of the local API Gateway installation

Before you begin

- Create a Docker image by following the steps provided in the article, <https://github.com/SoftwareAG/webmethods-api-gateway/tree/master/samples/docker#building-an-api-gateway-docker-image>.

➤ To build a Docker image:

1. Check out the following folder to your local machine, `webmethods-api-gateway/samples/datamanagement/APIGatewayUtility/`.
2. Edit the Docker file.
3. Replace the image name on the first line with the name of the API Gateway image that you have created.

```
FROM [API_GATEWAY_IMAGE_NAME]
```

4. Run the following command:

```
docker build . -t <tag_name>
```

For example,

```
docker build . -t apigwimage
```

The list of available images, including the new image, appears.

Running the API Gateway Utility Container

After you have built a Docker image, you can run the utility and perform a backup operation.

You can either run a command and exit the container or you can keep the container alive to perform more than one backup operation.

Exiting the container after performing an operation

If you want to exit the container after performing the operation then follow below steps.

➤ To perform a backup operation by running the API Gateway Utility container:

1. Run the following command:

```
docker run -e apigw_elasticsearch_hosts=[host:port] [IMAGE] -c [COMMAND]
```

where,

- **-e**. Variable to specify the API Data Store or Elasticsearch host and port.
- **[IMAGE]**. API Gateway Image that you created following the steps in the section, “[Running the API Gateway Utility Container](#)” on page 683.
- **-c**. Variable to specify the backup command.
- **[COMMAND]**. The required backup command. For information on possible backup operations and the commands to be used with the **apigatewayutil** utility, see “[Backup Operation](#)” on page 119.

For example,

```
docker run -e apigw_elasticsearch_hosts=daeyaix1bvt01.eur.ad.sag:9240 apigwutil -c  
list backup
```

Running the API Gateway Utility Container

After you have built a Docker image, you can run the utility and perform a backup operation.

You can either run a command and exit the container or you can keep the container alive to perform more than one backup operation.

Exiting the container after performing an operation

If you want to exit the container after performing the operation then follow below steps.

➤ To perform a backup operation by running the API Gateway Utility container

1. Run the following command:

```
docker run -e apigw_elasticsearch_hosts=[host:port] [IMAGE] -c [COMMAND]
```

where,

- **-e**. Variable to specify the API Data Store or Elasticsearch host and port.
- **[IMAGE]**. API Gateway Image that you created following the steps in the section, “[Running the API Gateway Utility Container](#)” on page 683.

- **-c.** Variable to specify the backup command.
- **[COMMAND].** The required backup command. For information on possible backup operations and the commands to be used with the **apigatewayutil** utility, see “[Backup Operation](#)” on page 119.

For example,

```
docker run -e apigw_elasticsearch_hosts=daeyaix1bvt01.eur.ad.sag:9240 apigwutil -c
list backup
```

Keeping the container alive to perform multiple backup operations

You need not exit the container, if you want to perform more than one backup operation.

➤ **To perform a backup operation by running the API Gateway Utility container**

1. Run the following command:

```
docker run -e apigw_elasticsearch_hosts=[host:port] [IMAGE] -i [COMMAND]
```

where,

- **-e.** Variable to specify the API Data Store or Elasticsearch host and port.
- **[IMAGE].** API Gateway Image that you created following the steps in the section, “[Running the API Gateway Utility Container](#)” on page 683.
- **-i.** Variable to specify the backup command.
- **[COMMAND].** The required backup command. For information on possible backup operations and the commands to be used with the **apigatewayutil** utility, see “[Backup Operation](#)” on page 119.

For example,

```
docker run -e apigw_elasticsearch_hosts=daeyaix1bvt01.eur.ad.sag:9240 apigwutil -i
list backup
```

Note:

Ensure that the Elasticsearch port is accessible to run the backup command.

Entering into a container to perform a backup operation

You can follow the steps in this section to enter in to an alive container to perform the required backup operations.

➤ **To enter into a container:**

1. Run the following command:

```
docker exec -it [CONTAINER_ID] /bin/bash
```

2. Run the following command to open the **apigatewayutil** folder:

```
cd/opt/softwareag/IntegrationServer/instances/default/packages/WmAPIGateway/cli/bin
```

3. Perform the required backup operation.

4. *Optional.* If the Elasticsearch instance is secured, run the following command with your credentials:

```
apigw_elasticsearch_hosts=host:port  
apigw_elasticsearch_https_enabled=("true" or "false")  
apigw_elasticsearch_http_username=user  
apigw_elasticsearch_http_password=password
```

Checking the API Gateway Utility logs

After performing the backup operations, you can check the backup logs to monitor the backup events.

➤ To check the API Gateway utility logs:

1. Run the following command to enter into the container:

```
docker exec -it [CONTAINER_ID] /bin/bash
```

2. Run the following command to navigate to the log file location:

```
cd /opt/softwareag/IntegrationServer/instances/default/packages/WmAPIGateway/cli/logs
```

3. Run the following command to view the log:

```
vi APIGWimage.log
```

The log file is generated.

Since the log file is present inside the container, the log file will be removed if the container is removed. To have a local copy of the log, include the option **-logFileLocation** when executing the command and mount the Docker volume.

For example,

```
docker run -e apigw_elasticsearch_hosts=daeyaix1bvt01.eur.ad.sag:9240 apigwimage  
-c list backup -logFileLocation \utils\logs
```

Viewing Console Logs

Message that is printed in the console while executing the command, will be saved under the Docker logs.

➤ To view console logs:

- Run the following command:

```
docker logs [CONTAINER_ID]
```

Viewing the help text or the list of existing commands

- Run the following command:

```
docker run -e apigw_elasticsearch_hosts=[host:port] [IMAGE] -help
```

Checking the exit code of the command

- Run the following command:

Windows

```
echo %errorlevel%
```

Linux

```
$?
```

Troubleshooting Tips: Docker Configuration

Docker run with environment variables is not able to replace UI properties

When I use docker run with environment variables as shown below, the Elasticsearch values in **uiconfiguration.properties** are not replaced. Hence the analytics is broken.

```
docker run -d --name gateway_externales --hostname gateway_externales -p 7072:9072 -p 3555:5555 -e apigw_elasticsearch_hosts=elastic:9200
```

However, the values in `WmAPIGateway/config/resources/elasticsearch/config.properties` are correctly replaced.

Resolution:

Set the kibana autostart to false and try.

Kubernetes Support

API Gateway can be run within a Kubernetes (k8s) environment. Kubernetes provides a platform for automating deployment, scaling and operations of services. The basic scheduling unit in Kubernetes is a *pod*. It adds a higher level of abstraction by grouping containerized components. A pod consists of one or more containers that are co-located on the host machine and can share resources. A Kubernetes service is a set of pods that work together, such as one tier of a multi-tier application.

The API Gateway Kubernetes support provides the following:

- Liveliness check to support Kubernetes pod lifecycle.

This helps in verifying that the API Gateway container is up and responding.

- Readiness check to support Kubernetes pod lifecycle.

This helps in verifying that the API Gateway container is ready to serve requests. For details on pod lifecycle, see *Kubernetes documentation*.

- Prometheus metrics to support the monitoring of API Gateway pods.

API Gateway support is based on the Microservices Runtime Prometheus support. You use the IS metrics endpoint /metrics to gather the required metrics. When the metrics endpoint is called, Microservices Runtime gathers metrics and returns the data in a Prometheus format. Prometheus is an open source monitoring and alerting toolkit, which is frequently used for monitoring containers. For details on the prometheus metrics, see *Developing Microservices with webMethods Microservices Runtime*.

The following sections describe in detail different deployment models for API Gateway as a Kubernetes service. Each of the deployment models described require an existing Kubernetes environment. For details on setting up of a Kubernetes environment, see Kubernetes documentation.

With the API Gateway Kubernetes support, you can deploy API Gateway in one of the following ways:

- A pod with API Gateway container and an Elasticsearch container
- A pod with API Gateway container connected to an Elasticsearch Kubernetes service

API Gateway also supports Red Hat OpenShift containerized platform that you can use for building and scaling containerized applications. For details and special considerations, see the following sections:

- “[Building the Docker Image for an API Gateway Instance](#)” on page 663, in particular the --target.configuration and --os.image parameters
- “[OpenShift Support](#)” on page 695

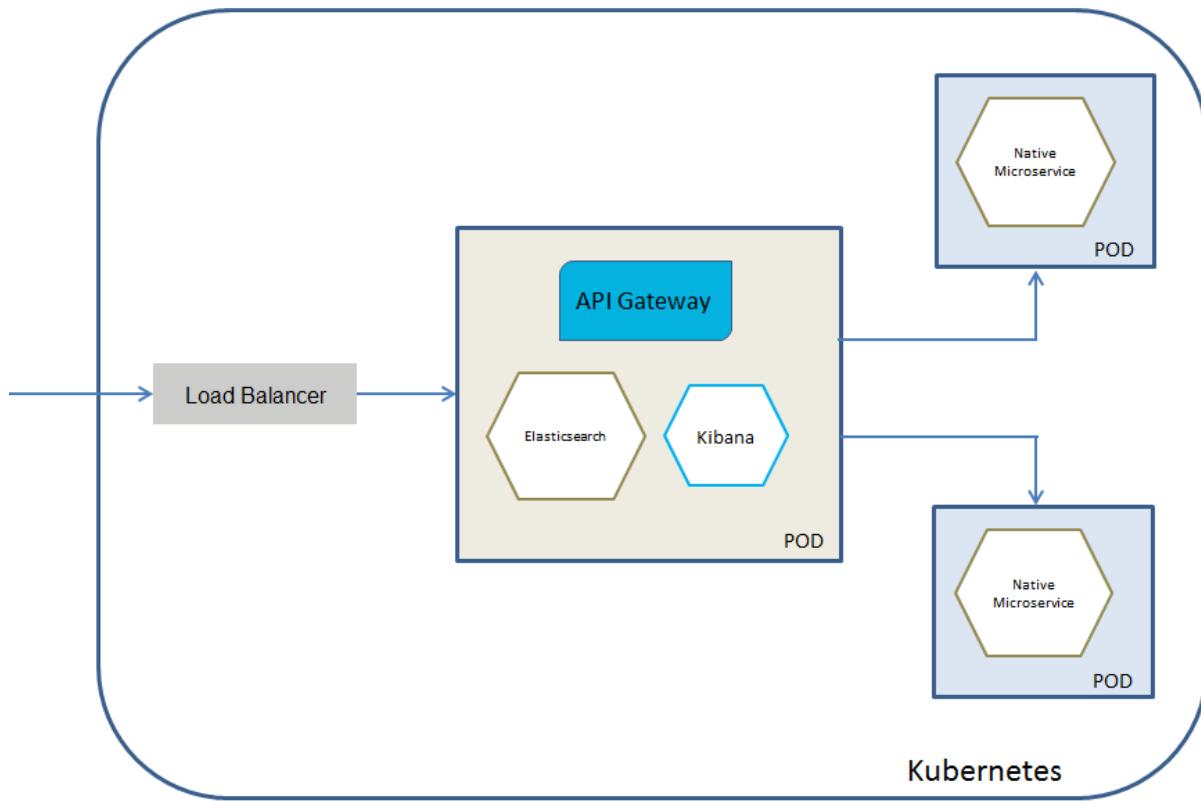
For details about OpenShift in general, see OpenShift documentation.

Deploying API Gateway Pod with API Gateway and Elasticsearch Containers

Select this deployment model if you want API Gateway as a Kubernetes service protecting the native services deployed to Kubernetes. Here, API Gateway runs in dedicated pods, and each pod has Elasticsearch and Kibana containers. API Gateway routes the incoming API requests to the native services. The invocation of the native services by the consumers happens through APIs provisioned by API Gateway.

The figure depicts the API Gateway Kubernetes service deployment model where you have a single API Gateway pod that contains API Gateway and Elasticsearch containers. The Kibana can

either be embedded in the API Gateway container or can reside as a separate container within the pod.



➤ To deploy API Gateway Kubernetes pod that contains an Elasticsearch container

1. Ensure that **vm.max_map_count** is set to a value of at least 262144 to run an Elasticsearch container within a pod. This is done in an init container as follows:

```
initContainers:
- command:
  - sysctl
  - -w
  - vm.max_map_count=262144
image: busybox
imagePullPolicy: IfNotPresent
name: init-sysctl
resources: {}
securityContext:
  privileged: true
```

2. Ensure that you have an API Gateway Docker image and an Elasticsearch image for this deployment. For the API Gateway container, you have to set the following environment:

```
apigw_elasticsearch_hosts=localhost:9200
```

This assumes that Elasticsearch runs on the standard port 9200 and the xpack.security is disabled. You can disable the xpack.security by setting the environment variable xpack.security.enabled to `false`.

The following YAML snippet displays how the environment variable `apigw_elasticsearch_hosts` is set.

```
spec:  
  containers:  
    - env:  
      - name: apigw_elasticsearch_hosts  
        value: localhost:9200
```

You can disable the xpack.security by setting the environment variable `xpack.security.enabled` to `false` for the Elasticsearch container.

3. Run the following command to deploy API Gateway in the Kubernetes setup:

```
kubectl create -f api-gateway-deployment-embedded-elasticsearch.yaml
```

Ensure that you have specified the required information such as image name, default ports in the Kubernetes sample file `api-gateway-deployment-embedded-elasticsearch.yaml` located at `SAG_Root/IntegrationServer/instances/default/packages/WmAPIGateway/resources/samples/K8s`. For details on Kubernetes YAML files, see Kubernetes documentation.

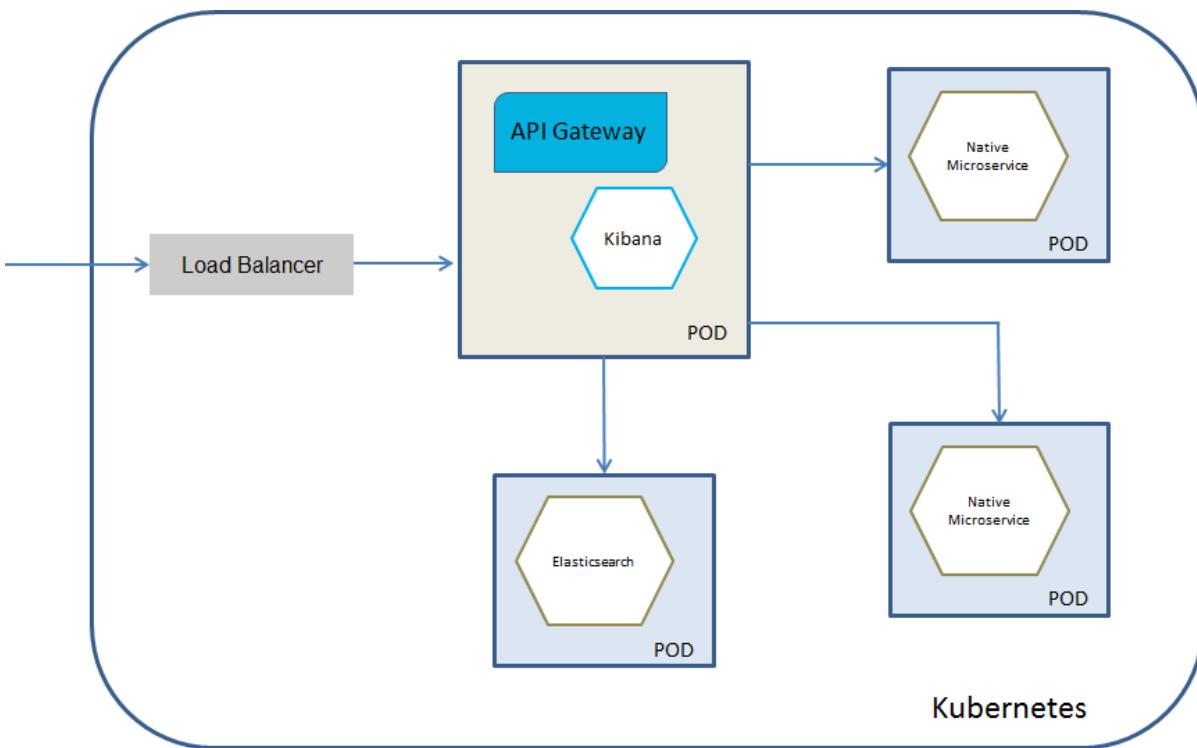
This now pulls the image specified and creates the API Gateway pod with API Gateway and Elasticsearch containers.

Run the command `kubectl get pods` to view the pods created.

Deploying API Gateway Pod with API Gateway Container connected to an Elasticsearch Kubernetes Service

Select this deployment model if you want to have a separate Elasticsearch service. This deployment allows you to scale Elasticsearch independently or to use an already existing Elasticsearch service. Ensure that you have an Elasticsearch Kubernetes service for Elasticsearch 8.2.3.

The diagram depicts the API Gateway Kubernetes service deployment model where you have a separate API Gateway pod that constitutes an API Gateway container connected to an Elasticsearch service. Kibana can run as a separate container within the API Gateway pod or can be embedded in the API Gateway container.



➤ To deploy an API Gateway Kubernetes pod that communicates with an Elasticsearch Kubernetes service

1. Ensure that you have an Elasticsearch Kubernetes service for Elasticsearch 8.2.3.

For more details on deploying Elasticsearch on Kubernetes, see *Elasticsearch and Kubernetes documentation*.

2. Ensure that you have an API Gateway Docker image for this deployment. For the API Gateway container, you have to set the following environment variable:

```
apigw_elasticsearch_hosts=elasticsearch-host:elasticsearch-port
```

3. Run the following command to deploy API Gateway in the Kubernetes setup:

```
kubectl create -f api-gateway-deployment-external-elasticsearch.yaml
```

Ensure that you have specified the required information such as image name, default ports, details of the external elastic search and how to access it in the Kubernetes sample file `api-gateway-deployment-external-elasticsearch.yaml` located at `SAG_Root/IntegrationServer/instances/default/packages/WmAPIGateway/resources/samples/K8s`. For details on Kubernetes YAML files, see Kubernetes documentation.

This now pulls the image specified and creates the API Gateway pod with API Gateway container connected to an Elasticsearch Kubernetes service.

Run the command `kubectl get pods` to view the pods created.

API Gateway Clustering on Kubernetes

When deploying API Gateway on Kubernetes, the intention is to create a highly available and scalable setup that can dynamically scale up and down according to the current load. Hence, always configure API Gateway as a cluster. You can provide the cluster configurations as environment variables in the Kubernetes deployment YAML file for API Gateway. The environment variables are the same as described in the Docker configuration section. For details about Docker configuration, see “[API Gateway Container Cluster Configuration](#)” on page 672.

Alternatively, you can also provide the cluster configurations in the externalized configuration files as described in the “[Using the Externalized Configuration Files](#)” on page 62 section. For Kubernetes, the configuration files are implemented as ConfigMaps, which are then injected into the pods through volume mapping.

Peer-to-peer clustering on Kubernetes

If you have configured API Gateway with peer-to-peer clustering you must consider that in a Kubernetes deployment the clustering is not configured with a list of host names. Instead, the namespace and service name of the API Gateway deployment are used. To detect other cluster members, each API Gateway server talks to the Kubernetes API server in order to analyze the endpoints attached to the service. This lookup operation requires specific Kubernetes permissions, which are not available out of the box. It is necessary to create a role with the appropriate permissions, create a role binding that assigns the role to a service account, and finally start the API Gateway deployment with the service account, instead of the default one.

The Kubernetes YAML file to create a service account

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: cluster-discovery-sa
```

The Kubernetes YAML file to create a role

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: cluster-discovery-role
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - endpoints
  verbs:
  - get
  - list
  - watch
```

The Kubernetes YAML file to assign the role

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
```

```

metadata:
  name: cluster-discovery-rolebinding
roleRef:
  kind: Role
  name: cluster-discovery-role
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: cluster-discovery-sa

```

The Kubernetes YAML file to use the service account in the API Gateway deployment YAML file

```

apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      serviceAccountName: cluster-discovery-sa

```

Kubernetes Sample Files

The API Gateway installation provides Kubernetes deployment samples. For details about these sample files, see <https://github.com/SoftwareAG/webmethods-api-gateway/tree/master/samples/kubernetes>.

To use the samples to deploy API Gateway in the Kubernetes setup, you must adapt the samples to configure the required specifications. Depending upon the Kubernetes deployment model, use the respective Kubernetes sample deployment files. API Gateway provides the following three sample deployment files:

- [api-gateway-deployment-embedded-elasticsearch.yaml](#)

This file shows how to deploy an API Gateway with an embedded Elasticsearch to a Kubernetes cluster. Required information you have to specify before you use this file are: container name, the path to your API Gateway image stored in a docker registry and container port.

- [api-gateway-deployment-external-elasticsearch.yaml](#)

This file shows how to deploy an API Gateway without Elasticsearch to a Kubernetes cluster. You must have an external Elasticsearch to be up and running. Required information you have to specify before you use this file are: container name, the path to your API Gateway image stored in a docker registry, container port, and information to access your external Elasticsearch.

- [api-gateway-deployment-sidecar-elasticsearch.yaml](#)

This file shows how to deploy an API Gateway with an Elasticsearch as a sidecar container (side car means the Elasticsearch container is deployed within the pod of the API Gateway) to a Kubernetes cluster. Required information you have to specify before you use this file are: API Gateway container name, the path to your API Gateway image stored in a docker registry, Elasticsearch container name, and the path to the Elasticsearch image.

The sample file also deploys an application service for the selected deployment. You can specify the configuration details for the service to be deployed. You can create and start all the services from your configuration with a single command.

Helm Chart

The API Gateway installation provides a sample helm chart. API Gateway uses Helm to streamline the Kubernetes installation and management. Helm allows you to easily templatize the Kubernetes deployments and provides a set of configuration parameters that you can use to customize the deployment. Helm chart combines the Kubernetes deployments and provides a service to manage them.

The Helm chart covers the following Kubernetes deployments:

- A pod with containers for API Gateway, Elasticsearch, and Kibana
- A pod with containers for API Gateway and Kibana
- A pod with containers for API Gateway and Kibana that supports clustering

The Helm chart supports a values.yaml file for the following Elasticsearch configurations:

- Embedded Elasticsearch
- External Elasticsearch
- Elasticsearch in a sidecar deployment

The values.yaml file passes the configuration parameters into the Helm chart. A sample values.yaml file is available at *SAG_Install_Directory/IntegrationServer/instances/default/packages/WmAPIGateway/resources/samples/helm/sag-apigateway*. Provide the required parameters in this file to customize the deployment.

Using Helm to Start the API Gateway Service

To use Helm chart to start the API Gateway service

1. Install and initialize Helm and then create a Helm chart.

For details, see <https://github.com/helm/helm/blob/master/docs/quickstart.md#install-helm?>.

This creates a standard layout with some basic templates and examples. Use the templates to easily templatize your Kubernetes manifests. Use the set of configuration parameters that the templates provide to customize your deployment.

2. Update the values.yaml file with the required information, such as the URL pointing to your repository, the port and service details, and the deployment type for which you want to create a service. The values.yaml file passes the configuration parameters into the helm chart.
3. Navigate to the working folder where the charts are stored, and run the following command.

```
helm install sag-api-gateway-10.5
```

Where, `sag-api-gateway-10.5` is the Helm chart name.

The Kubernetes cluster starts API Gateway and the service.

OpenShift Support

RedHat OpenShift is a container platform built upon and extends the Kubernetes functionality. In addition to Kubernetes' ability of orchestrating containerized applications, OpenShift provides support for the complete CI/CD life cycle of applications, called Source-To-Image.

The API Gateway OpenShift support provides the following, in the same way as the Kubernetes support does:

- Liveliness check. This helps in verifying that the API Gateway container is up and running.
- Readiness check. This helps in verifying that the API Gateway container is ready to server requests.
- Prometheus metrics to support the monitoring of API Gateway pods.
- Kubernetes-specific logging.
- Architectural patterns for running Elasticsearch as embedded, sidecar, or external.
- Auto scaling.

OpenShift extends Kubernetes and introduces new objects. For example, Kubernetes deployment is called `DeploymentConfig` and has the version id `apps.openshift.io/v1`. In order to make services accessible from outside the cluster, OpenShift provides `Route` objects. The images required to start containers are not necessarily referenced directly inside the container specification, rather they can be managed by `ImageStream` objects.

OpenShift has a specific way for running ElasticSearch containers. ElasticSearch needs an increased virtual memory `mmap count: vm.max_map_count >= 262144`. In a plain Kubernetes environment you can solve this by adding an `initContainer` that has to run in the privileged mode. OpenShift offers a much simpler solution. If a pod carries a specific label then OpenShift applies the necessary system changes behind the scenes when starting the pod's containers.

For details on how these OpenShift specific topics are reflected in YAML configuration files for API Gateway, see “[OpenShift Sample Files](#)” on page 698.

When starting a new container, by default, OpenShift ignores the built-in user of the Docker image and injects a new user. This user is a member of the root group, and hence the files, scripts, and programs inside the container have to be readable, writable, and executable by the root group. To understand how to work with this OpenShift behavior, see the following sections:

- “[Building a Docker Image for an API Gateway Instance in OpenShift Environment](#)” on page 695
- “[Running the Docker Image With the sagadmin user](#)” on page 697

Building a Docker Image for an API Gateway Instance in OpenShift Environment

When starting the API Gateway container, OpenShift ignores the built-in user of the Docker image and injects a new user. This user is a member of the root group, and hence the files, scripts, and programs inside the API Gateway container have to be readable, writable, and executable by the root group. To build a Docker image that fulfills these requirements, perform the procedure outlined.

➤ **To build a docker image for an API Gateway instance in an OpenShift environment**

1. Follow the steps outlined in “[Building the Docker Image for an API Gateway Instance](#)” on [page 663](#).

Ensure that you have set the parameters `--target.configuration` and `--os.image` specific to the OpenShift environment.

A sample shell script for creating an API Gateway Docker image for an Openshift environment looks as follows:

```
echo "is createDockerfile ====="
./is_container.sh createDockerfile
status=$?

if [ $status -ne 0 ]
then
    echo "Failed! status: $status"
    exit $status
fi

echo "is build ====="
./is_container.sh build
status=$?

if [ $status -ne 0 ]
then
    echo "Failed! status: $status"
    exit $status
fi

echo "apigw createDockerfile ====="
./apigw_container.sh createDockerfile --target.configuration openshift
status=$?

if [ $status -ne 0 ]
then
    echo "Failed! status: $status"
    exit $status
fi

echo "apigw build ====="
./apigw_container.sh build
status=$?

if [ $status -ne 0 ]
then
    echo "Failed! status: $status"
    exit $status
fi
```

The resulting Docker file uses `chgrp` and `chmod` commands to assign proper permissions to the root group. Running these commands almost doubles the Docker image size, hence the Docker file is organized as a multi-stage build where the first stage prepares the file system with root group permissions, and the second stage copies this into the final image. For the second stage, it is necessary to specify the base operating system image using the `--os.image` parameter, unless the default value, `centos:7`, is sufficient. As the API Gateway Docker image builds upon a previously created Integration Server Docker image, the value of the `--os.image` parameter is same as the value of the `-Dimage.name` parameter that is used in the creation of the Integration Server image.

The resulting API Gateway image has the built-in `sagadmin` user, but due to the adapted root group permissions, the image can be deployed to an OpenShift cluster.

Note:

The resulting API Gateway image can also be deployed to Docker or Kubernetes systems where it is deployed under the control of the `sagadmin` user.

Running the API Gateway Docker Image with the `sagadmin` User

If you do not want to use the default OpenShift behavior of starting the API Gateway container with an arbitrary root group user, you have to create a special service account with corresponding permissions using the `oc` command line tool of OpenShift.

➤ To run the API Gateway Docker image with the built-in `sagadmin` user

1. Switch to the API Gateway project where you intend to deploy API Gateway.

```
oc project API Gateway project name
```

2. Create a service account `runassagadmin`.

```
oc create serviceaccount runassagadmin
```

3. Assign the permission to the service account `runassagadmin` to use the built-in user of the Docker image.

```
oc adm policy add-scc-to-user anyuid -z runassagadmin
```

Note:

You must have OpenShift administrator privileges to perform this step.

4. In the `DeploymentConfig.yaml` file for API Gateway, set the field `spec.template.spec.serviceAccountName` to the name of the newly created service account.

```
apiVersion: apps.openshift.io/v1
kind: DeploymentConfig
metadata:
  name: api-gateway-deployment

spec:
  template:
```

```
spec:  
  serviceAccountName: runassagadmin
```

In the API Gateway sample YAML file, described in “[OpenShift Sample Files](#)” on page 698 section, the `serviceAccountName` field is pre-populated with the default service account `default` for OpenShift.

5. Apply the modified DeploymentConfig YAML file.

```
oc apply -f modified deploymentconfig for API Gateway
```

Note:

The API Gateway Docker image referenced in the DeploymentConfig YAML file can be any API Gateway Docker image. It is not necessary to build it using the `--target.configuration` parameter as described in “[Building a Docker Image for an API Gateway Instance in OpenShift Environment](#)” on page 695.

OpenShift Sample Files

API Gateway installation provides OpenShift deployment samples. For details about these sample files, see <https://github.com/SoftwareAG/webmethods-api-gateway/tree/master/samples/openshift>. To use the samples to deploy API Gateway to an OpenShift cluster, you must adapt the samples to configure the required specifications.

The OpenShift samples are conceptually identical to the ones described in the “[Kubernetes Sample Files](#)” on page 693 section and support the same architectural patterns for ElasticSearch. This section highlights the parts that are specific to OpenShift environment.

OpenShift uses a DeploymentConfig object with API version `apps.openshift.io/v1` to describe a deployment. The section in the sample file is as follows:

```
apiVersion: apps.openshift.io/v1  
kind: DeploymentConfig
```

If you have a pod labeled as `tuned.openshift.io/elasticsearch`, then OpenShift automatically changes the required system settings on the machine where the pod with the ElasticSearch container is started. The section in the sample file is as follows:

```
template:  
  metadata:  
    labels:  
      deploymentconfig: api-gateway-deployment  
      tuned.openshift.io/elasticsearch: ""
```

In OpenShift, use the `ImageStream` and `ImageStreamTag` objects to reference the image to be used for a container instead of specifying the image name directly in the `spec.template.spec.containers` section. The section in the sample file is as follows:

```
triggers:  
  - type: ConfigChange  
  - type: ImageChange  
    imageChangeParams:  
      automatic: true  
    containerNames:
```

```
- api-gateway-deployment
from:
  kind: ImageStreamTag
  name: api-gateway-deployment:10.11
---
apiVersion: image.openshift.io/v1
kind: ImageStream
metadata:
  name: api-gateway-deployment
spec:
  lookupPolicy:
    local: false
  tags:
    - from:
        kind: DockerImage
        # Please fill in the path to your api gateway image stored in a docker registry.
        name: <yourDockerRegistry>:<RegistryPort>/<PathToApiGateway>:10.11
      importPolicy: {}
  name: "10.11"
  referencePolicy:
    type: Source
```

Use the Route objects that OpenShift provides to make a service visible outside the cluster. Note that the URL specified in the spec.host parameter is unique across the whole OpenShift cluster. The section in the sample file is as follows:

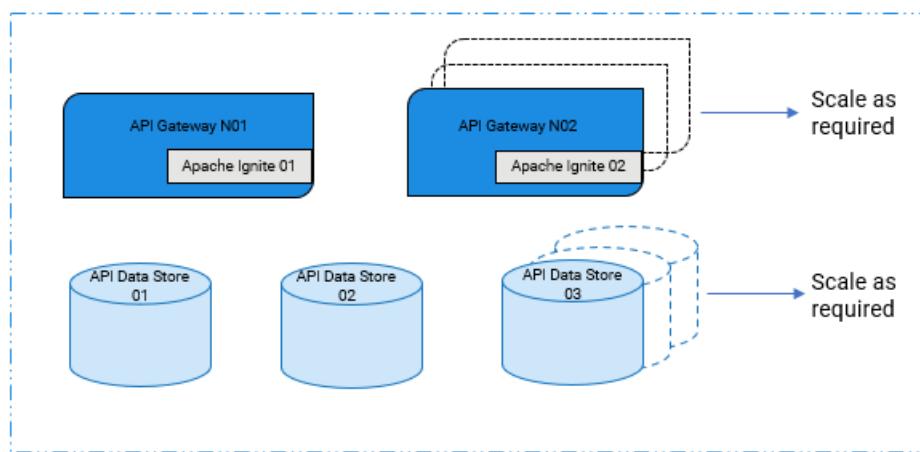
```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: api-gateway-ui
spec:
  # Provide a URL that will be visible outside of the OpenShift cluster
  host: api-gateway-ui.apps.<yourClusterBaseUrl>
  port:
    targetPort: 9072-tcp
  subdomain: ""
  to:
    kind: Service
    name: api-gateway-service
    weight: 100
  wildcardPolicy: None
```


5 High Availability, Disaster Recovery, and Fault Tolerance

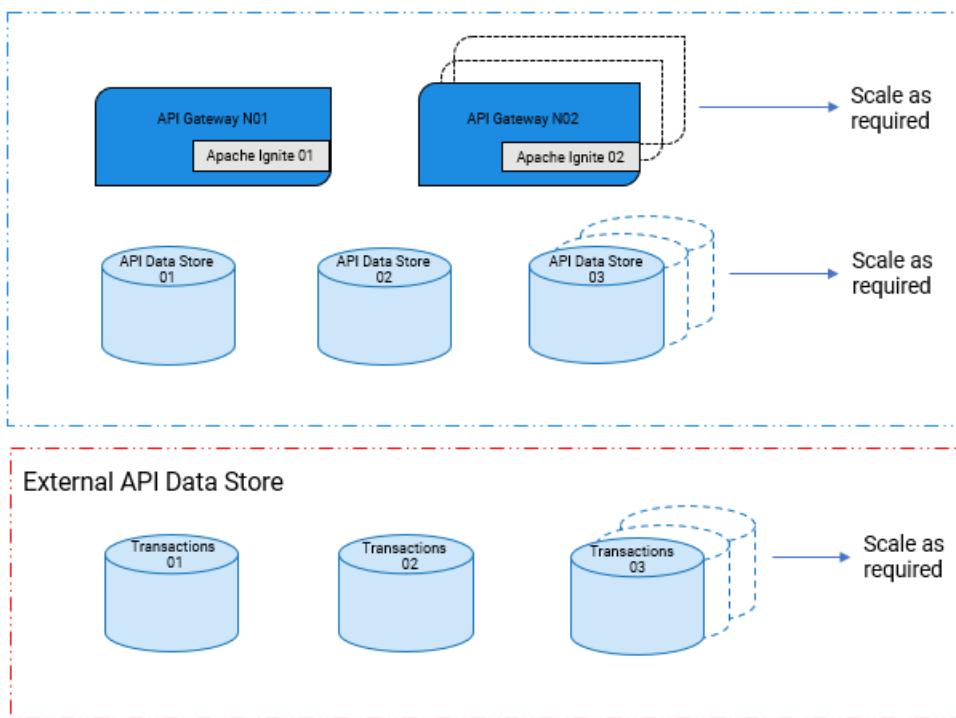
■ High Availability	702
■ High Availability and Disaster Recovery	706
■ High Availability and Fault Tolerance	714

High Availability

To achieve high availability, you can cluster your API Gateway, API Data Store and Apache Ignite instances within a data center or a region. Clustering ensures that there is no single point of failure in the system in that data center. As an alternative for Apache Ignite, Terracotta Server Array can be used to set up a cluster. For more information about the cluster deployment, see “[Cluster Deployment](#)” on page 24. The typical HA architecture is as follows.



There is another variant to the HA architecture. You can store the Transactional Events (TE) and logs in an External API Data Store too. For information about the external API Data Store, see “[Connecting to an External Elasticsearch](#)” on page 56. The HA architecture with the External API Data Store is as follows.

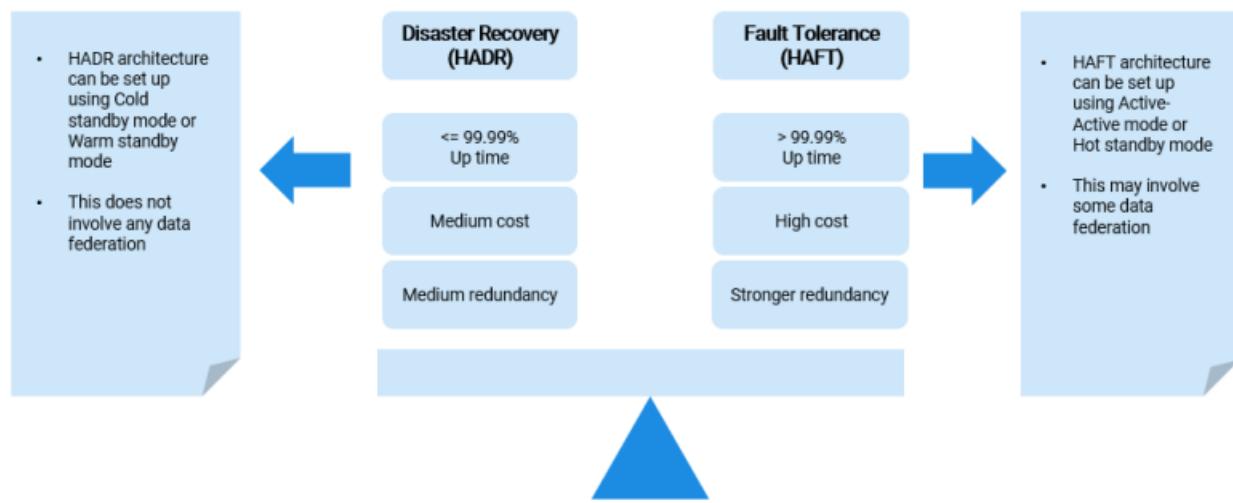


While HA architecture offers protection against single point of failure within a single data center, it does not protect you from the failure of an entire data center due to events like a natural disaster, cyber attack and so on. For protection against data center failures and to ensure business continuity, you must choose one business continuity solution from the following:

- High Availability and Disaster Recovery (HADR) solution.
- High Availability and Fault Tolerance (HAFT) solution.

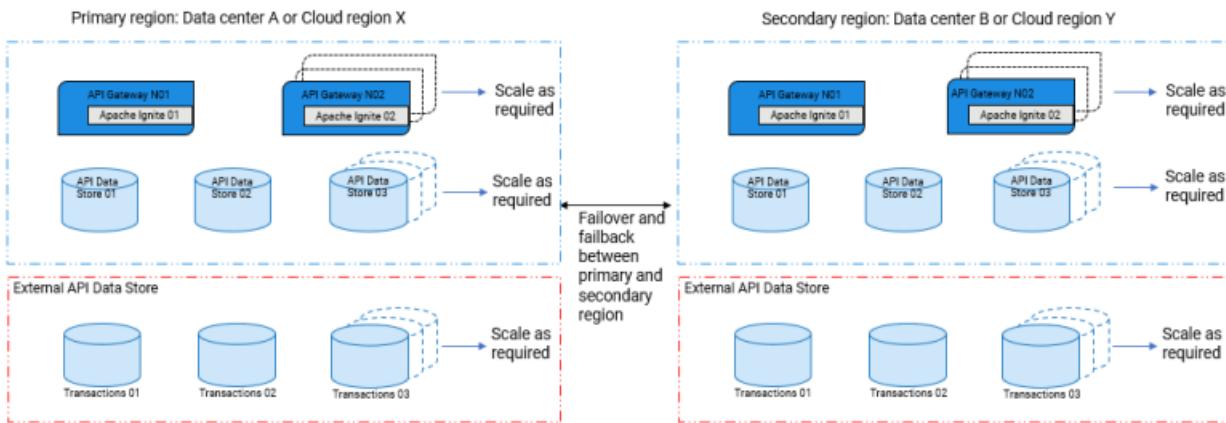
Disaster Recovery vs Fault Tolerance

Both **High Availability and Disaster Recovery (HADR)** and **High Availability and Fault Tolerance (HAFT)** architectures ensure that the application runs without any system degradation. However, the unique attributes that differentiate them are cost, design, redundancy level, and behavior on component faults or failures.



Keypoints about High Availability and Disaster Recovery (HADR) solution

The architecture of HADR is as follows:



The keypoints about HADR solution is as follows:

- Each data center or the cloud region hosts an independent cluster of its own.
- Primary data center serves the traffic and is exposed to the client and you must turn on the secondary data center only when the primary data center goes down due to a disaster.
- The recovery process during disaster recovery is categorized into two stages:
 - **Failover.** The failover operation is the process of switching data from a primary data center to a secondary data center.
 - **Fallback.** The fallback operation is the process of returning data from a secondary data center to a primary data center.
- HADR solution can be set up using **Cold standby mode** or **Warm standby mode**. The following table explains the difference between both the modes.

Failover Mode	RTO (approximately)	RPO (customizable)	Description
Cold standby mode	30 minutes to 1 hour	>=15 minutes	<ul style="list-style-type: none"> ■ Secondary data center remains switched off until failover operation. Hence, this approach saves cost. ■ Backup scheduler is set to meet the RPO definition.
Warm standby mode	15 minutes	>=15 minutes	<ul style="list-style-type: none"> ■ API Data Store is operational in the secondary data center. ■ Backup scheduler is set to meet the RPO definition. ■ As API Data Store is operational and data is restored from the latest backup snapshots in secondary data center, this approach reduces

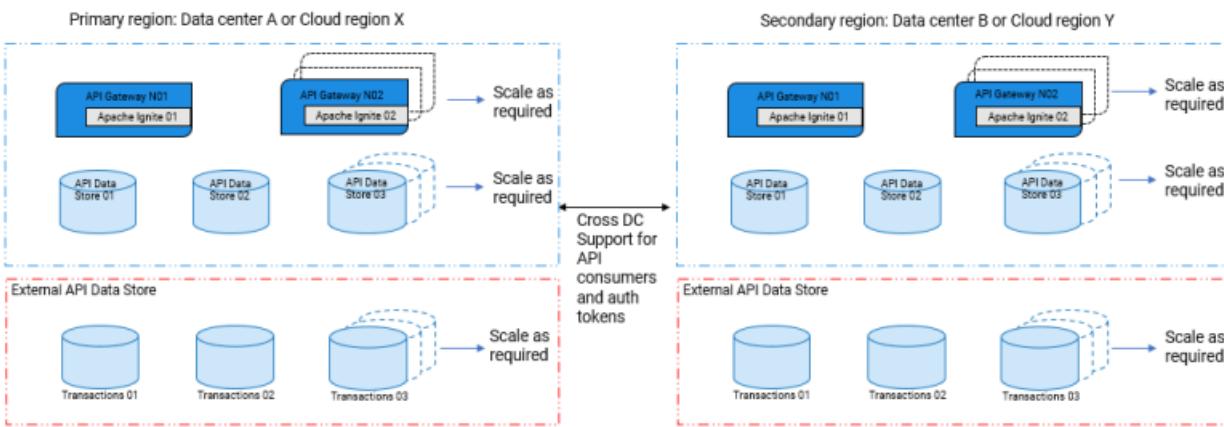
Failover Mode	RTO (approximately)	RPO (customizable)	Description
			RTO, but the cost is higher compared to Cold standby mode.

- Before performing the failover process:
 - In cold standby mode:
 - Install API Gateway in the secondary data center and ensure that the fix upgrades of the primary data center is applied to the secondary data center too.
 - Run the backup scripts periodically to take a snapshot of data from the primary data store and store the snapshots in a suitable externalized storage like AWS S3, Azure Blobs.
 - In warm standby mode:
 - Install API Gateway in the secondary data center and ensure that the fix upgrades of the primary data center is applied to the secondary data center too.
 - Run the backup scripts periodically to take a snapshot of data from the primary data store and store the snapshots in a suitable externalized storage system.
 - Ensure that API Data Store is turned on and running.
 - Run the restore scripts periodically to restore the backedup snapshots from the externalized storage in the secondary data center.
- During the failover process:
 - In cold standby mode, start API Data Store, restore the data from the snapshots in API Data Store, and then start API Gateway in the secondary data center.
 - In warm standby mode, as API Data Store is up and running, and the data is already restored from the snapshots in API Data Store, you must just restart API Gateway alone.
- After completing the disaster recovery process, when you want to switch back the traffic from the secondary data center to primary data center, follow the failback process.

For more details, see ["High Availability and Disaster Recovery "](#) on page 706.

Keypoints about High Availability and Fault Tolerance (HAFT) solution

The architecture of HAFT is as follows:



The keypoints about HAFT solution is as follows:

- Use this set up, if the RTO is highly demanding, which ranges from few seconds or minutes.
- Each data center or the cloud region hosts an independent or isolated cluster of its own.
- HAFT solution can be set up using **Hot standby mode** or **Active-Active mode**. The following table explains the difference between both the modes.

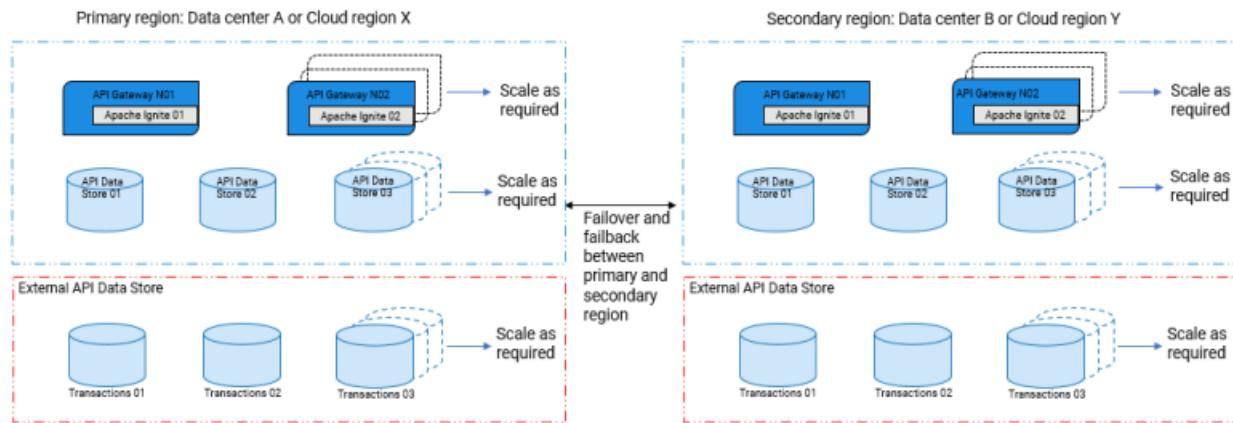
Failover Mode	RTO	RPO	Description
Hot standby mode	few seconds or minutes (time taken to switch the load balancer).	less than 5 seconds.	<ul style="list-style-type: none"> ■ You can have only two data centers. Out of which only the primary data center serves the client request.
Active-Active no down time mode	no down time.	less than 5 seconds.	<ul style="list-style-type: none"> ■ You can have N (Min 2) number of data centers and all the data centers serve the client request.

- API assets should be synchronized across the data center through CI-CD deployments.
- Cross-DC federation support offered by API Gateway is limited to API consumers and auth tokens.
- API transactions are local to the cluster where the traffic is served and not aggregated. For aggregated transactions, use a centralized API Data Store.

For more details, see [“High Availability and Fault Tolerance” on page 714](#).

High Availability and Disaster Recovery

The architecture of high availability and disaster recovery (HADR) is as follows:

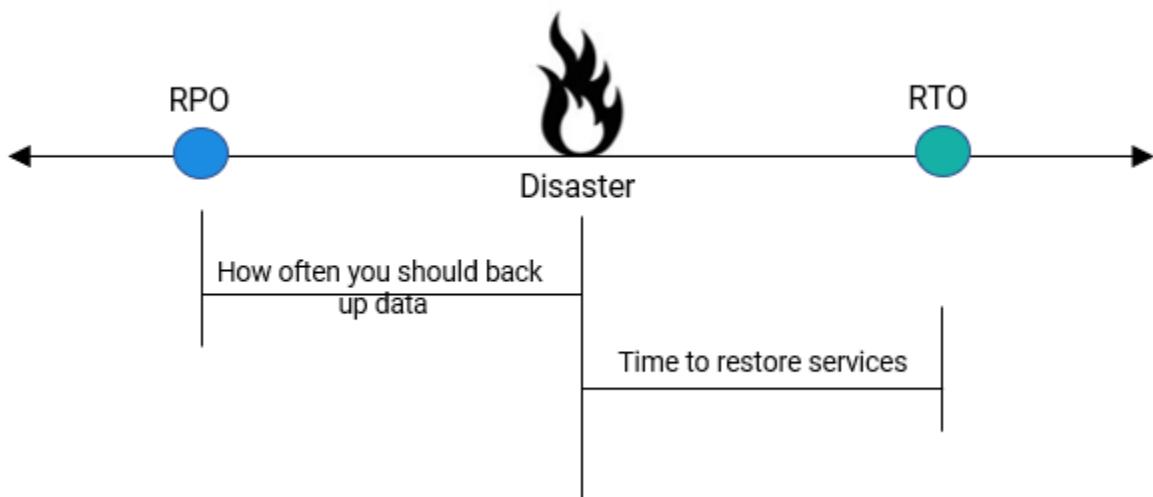


When an entire data center goes down due to natural disaster, equipment failure, or cyber attack, a business has to recover lost data from where the data is backed up. The disaster recovery relies upon the replication of the backed up data in a safe network or a cloud location that is not affected by the disaster.

Disaster recovery architecture can be setup using **Cold standby mode** or **Warm standby mode**.

The two most important parameters for a disaster recovery plan are:

- **Recovery Point Objective (RPO)**. Describes the age of files that must be recovered from backup for a business operation to resume after a disaster. It also specifies how often you should back up data. For example, if your RPO value is 15 minutes, then the data before 15 minutes of a disaster must be restored for operations to resume.
- **Recovery Time Objective (RTO)**. Describes the duration and service level within which you must restore the most critical IT services after a disaster. For example, if your RTO value is 60 minutes, the data in the required systems must be restored within 60 minutes of a disaster event.



You can have an effective disaster recovery management in place by configuring a reliable repository and by taking data backup at regular intervals.

The recovery process during disaster recovery is categorized into two stages:

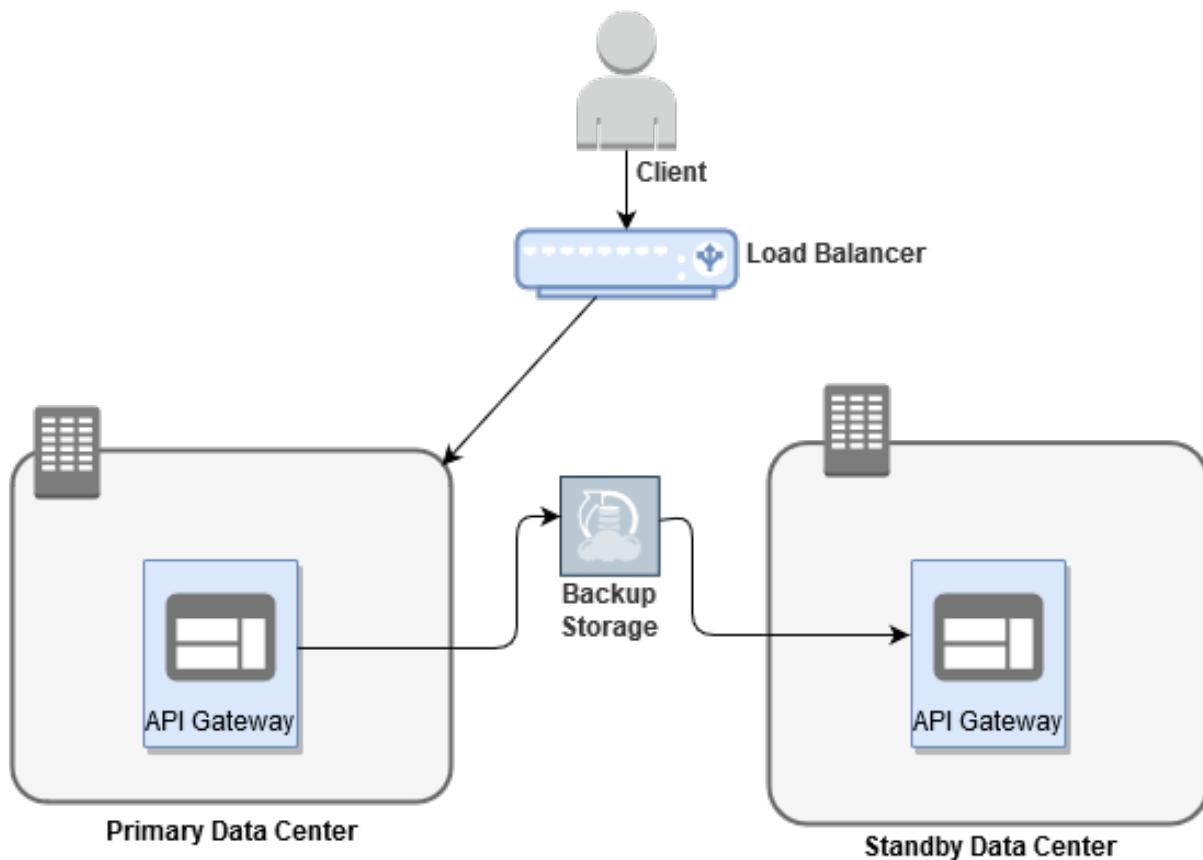
- **Failover.** The failover operation is the process of switching data from a primary data center to a backup facility.
- **Failback.** The failback operation is the process of returning data from the backup facility to the primary data center.

What is Cold Standby Mode?

In the *Cold Standby* mode, there are only two data centers. The primary data center is up and running, whereas the secondary data center is turned on only when the primary data center goes down. On failure of the primary data center, the secondary data center replaces the primary data center. As part of disaster recovery procedure, perform the following steps in the secondary data center:

- Bring up API Data Store.
- Run the scripts for restore.
- Bring up API Gateway post restore of data.
- Reconfigure the load balancer to redirect the traffic to the secondary data center.

Cold standby mode is cost-effective in terms of data center operations. However, there is a downtime if the primary data center goes down. The RPO and RTO for cold standby mode is high as compared to rest of modes.



How Do I Set Up Data Center in Cold Standby Mode?

This use case explains how to set up data centers in cold standby mode for achieving disaster recovery.

For example, assume that you have two data centers DC 1 and DC 2 in the following landscape:

Data Center Name	Host Name	Region
DC 1	uk.myhost.com	United Kingdom
DC 2	us.myhost.com	United States

Here, the DC 1 is in active mode and DC 2 is passive. You want to bring up DC 2, when DC 1 goes down. Assume that DC 2 is in cold standby mode, then there would be no data in DC 2.

Pre-requisites

- Install API Gateway in DC2 and ensure that the fix upgrades of DC1 is applied to DC2 too.
- Run the following backup script periodically to take a snapshot of data from DC1 and store the snapshots in a suitable externalized storage system based on the RPO.

```
apigatewayUtil.sh create backup -name backup_file_name
```

For more information about back up and restore, see “[Data Backup](#)” on page 117.

Note:

This command takes a complete backup. This impacts your RTO. To take a backup of the assets data alone, see “[How Do I Set Up Data Center in Cold Stand By Mode With Backup of Assets Data Only?](#)” on page 710.

➤ Failover and Failback process

1. During the failover process:

- Start API Data Store in DC2.
- Run the following command in DC 2 to restore the backed up data in API Data Store:

```
apigatewayUtil.sh restore backup -name backup_file_name
```
- Start API Gateway in DC 2.
- Reconfigure the load balancer by exposing DC 2 to the client.

2. During the failback process:

- Run the following command in DC 2 to back up the data:

```
apigatewayUtil.sh create backup -name backup_file_name
```
- Run the following command in DC 1 to restore the backed up data:

```
apigatewayUtil.sh restore backup -name backup_file_name
```
- Start API Gateway in DC 1.
- Reconfigure the load balancer by exposing DC 1 to the client.

To optimize your RTO, perform the following:

- Configure an external API Data Store to store the Transactional Events (TE) and logs. For more information, see “[Connecting to an External Elasticsearch](#)” on page 56.
- Instead of taking a complete backup, take a backup of the assets data alone from the primary data center (DC1) and restore it in the secondary data center (DC2), which will in turn reduce your RTO. To take a backup of assets data, see “[How Do I Set Up Data Center in Cold Stand By Mode With Backup of Assets Data Only?](#)” on page 710.

How Do I Set Up Data Center in Cold Stand By Mode With Backup of Assets Data Only?

This use case explains how to backup, restore assets data during failover, failback operations, and how to subsequently merge analytics data in both the data centers during failback operation.

For example, assume that you have two data centers DC 1 and DC 2 in the following landscape:

Data Center Name	Host Name	Region
DC 1	uk.myhost.com	United Kingdom
DC 2	us.myhost.com	United States

Here, the DC 1 is in active mode and DC 2 is passive. You want to bring up DC 2, when DC 1 goes down. Assume that DC 2 is in cold standby mode, then there would be no data in DC 2.

Pre-requisites

- Install API Gateway in DC2 and ensure that the fix upgrades of DC1 is applied to DC2 too.
- Run the following backup script periodically to take a snapshot of assets data from DC1 and store the snapshots in a suitable externalized storage system based on the RPO.

```
apigatewayUtil.sh create backup -name backup_file_name -include assets
```

This command takes a backup of asset data only. For more information about back up and restore, see “[Data Backup](#)” on page 117.

➤ Failover and Fallback process

1. During the failover process:
 - Start API Data Store in DC2.
 - Run the following command in DC 2 to restore the backed up data in API Data Store:


```
apigatewayUtil.sh restore -name backup_file_name -include assets
```
 - Start API Gateway in DC 2.
 - Reconfigure the load balancer by exposing DC 2 to the client.
2. During the fallback process, perform the restore (overwrite) operation in two parts. Firstly, restore the assets data, and dashboard data from DC2 to DC1. Secondly, merge the analytics data, and logging data from DC2 to DC1.
 - Run the following command to restore the assets data, and dashboard data from DC2 to DC1:


```
apigatewayUtil.sh restore backup -name backup_file_name -include assets,dashboard
```
 - Run the following command to restore other indices such as analytics data, logging data from DC2 to DC1 to ensure that the data is merged between both the data centers:


```
apigatewayUtil.sh restore backup -name backup_file_name -include analytics,license,audit,cache,log -aggregate true
```
 - Start API Gateway in DC 1.

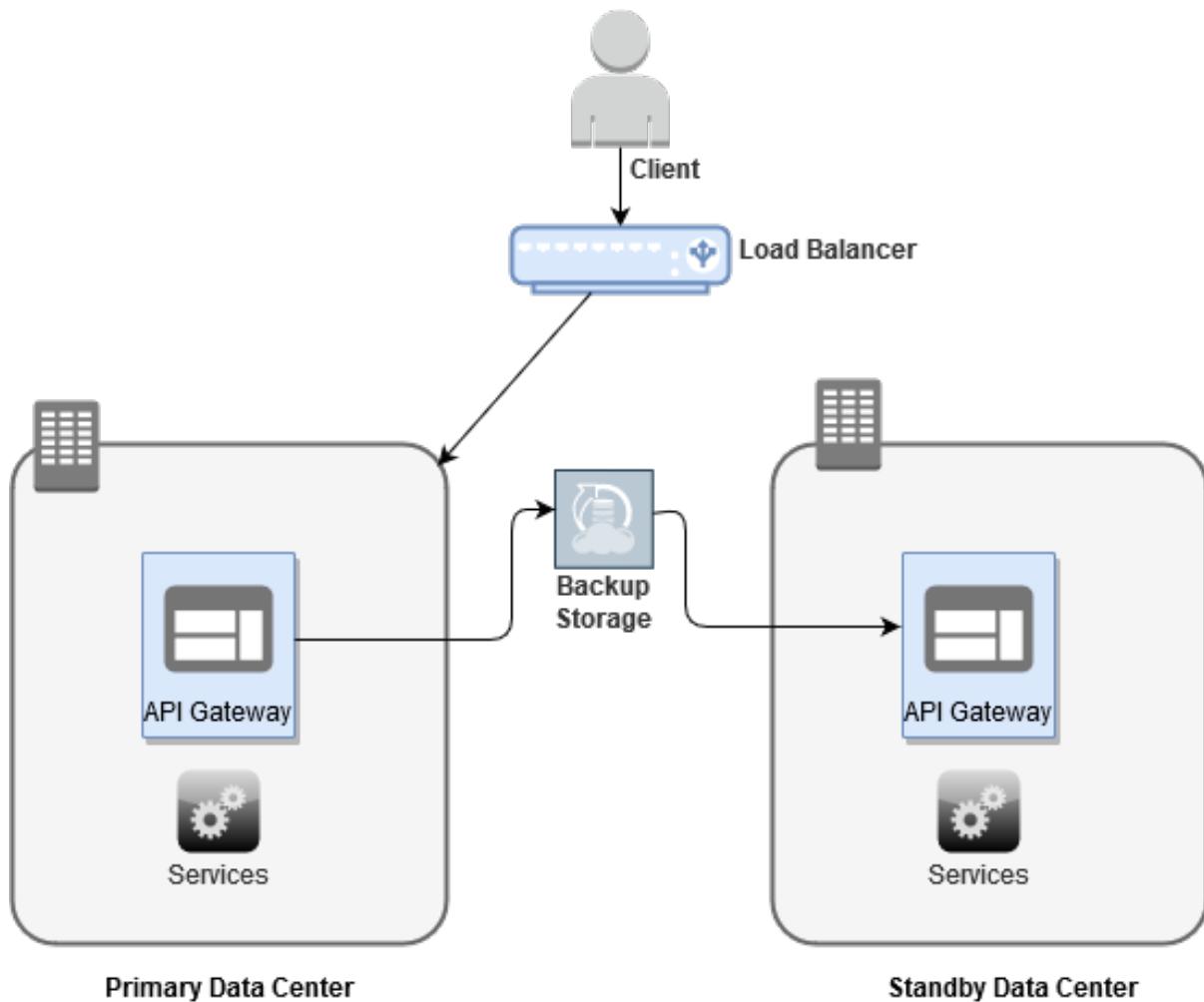
- Reconfigure the load balancer by exposing DC 1 to the client.

What is Warm Standby Mode?

In the *Warm Standby* mode, there are only two data centers. The primary data center is up and running, and in the secondary data center, only API Data Store is up and running, whereas API Gateway is in a shutdown state and is turned on only during disaster recovery. The secondary data center is regularly backed up with primary data center's data. On failure of the primary data center, the secondary data center replaces the primary data center. As part of disaster recovery procedure, perform the following steps in the secondary data center:

- Bring up API Gateway.
- Reconfigure the load balancer to redirect the traffic to the standby data center.

The RPO and RTO for warm standby mode is less when compared to cold standby mode.



How Do I Set Up Data Center in Warm Standby Mode?

This use case explains how to set up the data centers in warm standby mode to achieve disaster recovery.

For example, assume that you have two data centers DC 1 and DC 2 in the following landscape:

Data Center Name	Host Name	Region
DC 1	uk.myhost.com	United Kingdom
DC 2	us.myhost.com	United States

Here, DC 1 serves the traffic, and in DC2, only API Data Store is up and running, whereas API Gateway is in a shutdown state.

Pre-requisites

- Install API Gateway in DC2 and ensure that the fix upgrades of DC1 is applied to DC2 too.
- Run the following backup script periodically to take a snapshot of data from DC1 and store the snapshots in a suitable externalized storage system based on the RPO:


```
apigatewayUtil.sh create backup -name backup_file_name
```
- Run the following restore script periodically to restore the backup snapshots from the externalized storage in DC2.


```
apigatewayUtil.sh restore backup -name backup_file_name
```

For more information about backup and restore, see “[Data Backup](#)” on page 117

➤ Failover and Failback process

1. During the failover process:

- Restart API Gateway in DC2.

As API Data Store is up and running and the data is already restored from the snapshots in API Data Store, you just have to restart API Gateway in DC 2. The downtime for this mode would be only the time until API Gateway is up.

- Reconfigure the load balancer by exposing DC 2 to the client.

2. During the failback process:

- Run the following command in DC 2 to back up the data:


```
apigatewayUtil.sh create backup -name backup_file_name
```

- Run the following command in DC 1 to restore the backed up data:

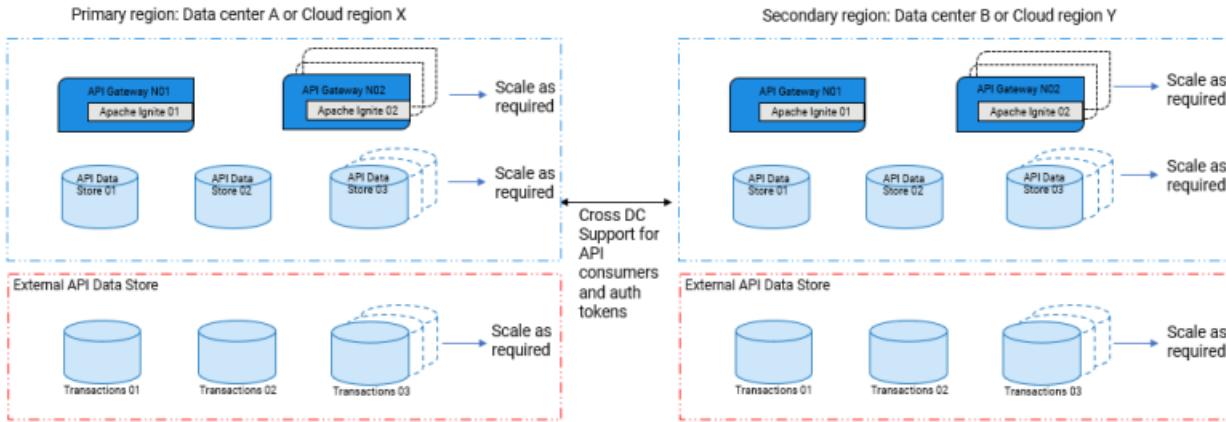

```
apigatewayUtil.sh restore backup -name backup_file_name
```

- Restart API Gateway in DC 1.

- Reconfigure the load balancer by exposing DC 1 to the client.

High Availability and Fault Tolerance

The architecture of high availability through fault tolerance is as follows:



High availability and fault tolerance solution ensures availability of services with very minimal downtime in case of a failure. Data centers are set up in different geographical regions to ensure business continuity and operational flexibility. This architecture provides greater resiliency compared to the HADR solution. However, the cost of this solution is high due to the availability of services in different data centers. For achieving high availability through fault tolerance, API Gateway provides **Cross DC** support through which volatile data like API consumers, OAuth tokens, and so on are federated across the data centers.

The Cross-Data Center (DC) support provides protection against data center failures by setting up API Gateway across different data centers using: **Hot standby mode** or **Active-Active mode**.

What is Hot Standby Mode?

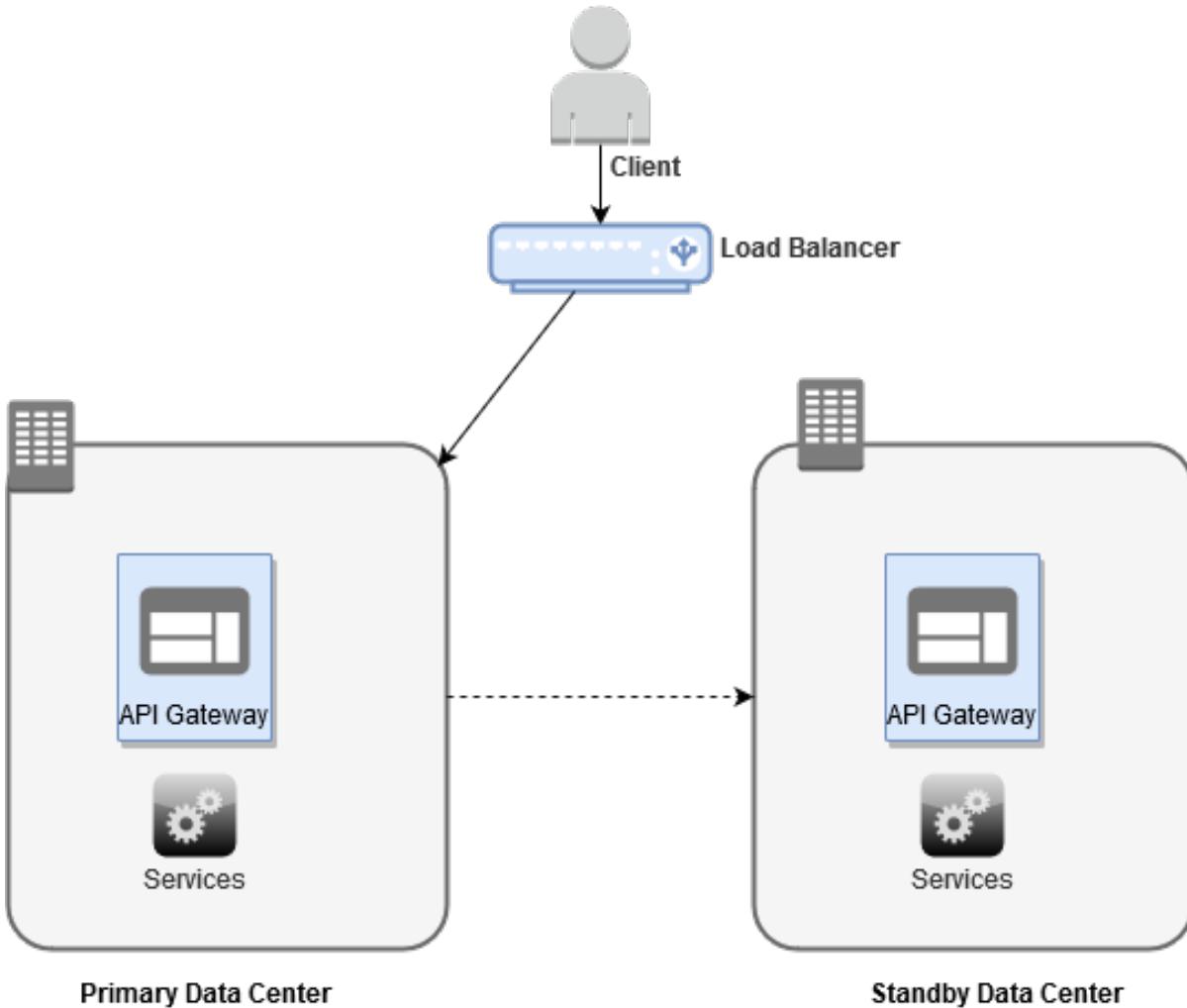
In the *Hot Standby* mode, there are only two data centers. Both the data centers are up, running, and symmetric. But only one of the data center is exposed to the clients to handle and process their requests. In other words, the load balancer directs traffic only to the primary data center. The secondary data center shadow-writes all the client requests, hence the data is mirrored in real time and both the data centers have identical data. When the primary data center goes down, you can expose the secondary data center to the clients in no time with minimal intervention. This is done by reconfiguring the load balancer to redirect the traffic to the secondary data center. The RPO and RTO for hot standby mode is less compared to warm and cold standby modes.

Note:

API Gateway automatically federates the following volatile data only:

1. Applications (API consumers)
2. Oauth tokens and Auth code
3. Oauth or OpenID scopes (scope mapping)
4. Application registration to the API

Other data such as APIs, and so on should be promoted to the secondary data center using CI/CD pipelines.



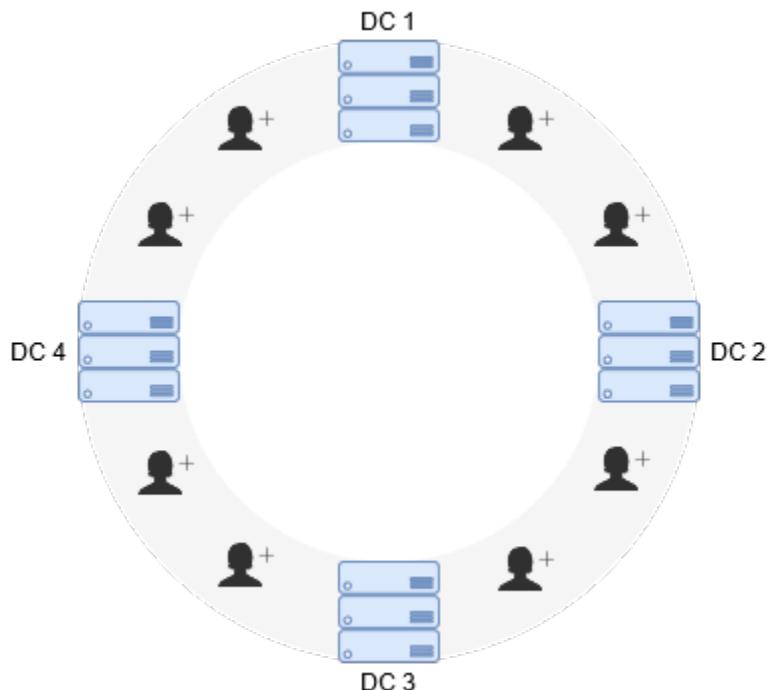
Set up the Cross-DC support in API Gateway in the hot standby mode using one of the following methods:

- Method 1: "[Setting Up the Data Centers in Hot Standby Using Basic Operation](#)" on page 719.
You can configure the data centers individually using a basic operation, where each data center is considered as a unit. This set up requires finer details like node name to configure the data centers at unit level. In case the configuration procedure encounters an error, it is easier to troubleshoot, because the configuration is done at unit level. You can reconfigure that data center, which causes the problem, in no time. Choose this method, if you want to configure both the data centers in your environment at a unit level.
- Method 2: "[Setting Up the Data Centers in Hot Standby Using Composite Operation](#)" on page 724.
You can configure the data centers simultaneously using a composite operation. This composite operation includes setting up the data center and establishing connection between data centers.

Both the data centers are configured simultaneously, and configuring the data centers takes less time. This method requires basic details such as host name, port, and so on to configure the data center. In case the configuration procedure encounters an error, you must reconfigure both the data centers. Choose this method, if you want to configure both the data centers in your environment simultaneously.

What is Active-Active Mode?

In the *Active - Active* mode, you can accommodate as many data centers as you want. In this mode, all data centers are up and running. Each data center can handle and process the requests from the client. Here the data centers are located in ring topology and with consistent hashing technique, the load gets balanced with average $1/N$ uniform load across the data centers. With consistent hashing technique and replication factor 2, each data center maintains up-to-date copies of data of the immediate data center in clockwise direction. In the event of a catastrophe, if any one of the data centers goes down then all the requests handled by that data center are handled by the next near-by data center in clockwise direction.



For example, if DC 1 in the above depicted figure goes down, then the two requests that were handled by DC 1 are handled by DC 2.

Note:

API Gateway automatically federates the following volatile data only:

1. Applications (API consumers)
2. Oauth tokens and Auth code
3. Oauth or OpenID scopes (scope mapping)
4. Application registration to the API

Other data such as APIs, and so on should be promoted to the secondary data center using CI/CD pipelines.

Set up the Cross-DC support in API Gateway in the active-active mode using one of the following methods:

- Method 1: ["Setting Up the Data Centers in Active-Active Using Basic Operation" on page 731.](#)

You can configure the data centers individually using a basic operation, where each data center is considered as a unit. This set up requires finer details like node name to configure the data centers at unit level. In case the configuration procedure encounters an error, it is easier to troubleshoot, because the configuration is done at unit level. You can reconfigure that data center, which causes the problem, in no time. Choose this method, if you want to configure less number of data centers in your environment at a unit level.

- Method 2: ["Setting Up the Data Centers in Active-Active Using Composite Operation" on page 736.](#)

You can configure the data centers simultaneously using a composite operation. This composite operation includes setting up the data center and establishing connection between data centers. All the data centers are configured simultaneously, and configuring the data centers takes less time. This method requires basic details such as host name, port, and so on to configure the data center. In case the configuration procedure encounters an error, you must reconfigure all the data centers. Choose this method, if you want to configure more number of data centers in your environment simultaneously.

Data Synchronization in Hot Standby and Active-Active Modes

In both hot standby and active-active modes, the data centers are inter-connected with fully connected mesh topology in a ring configuration. The data synchronization happens at the application level and not through API Data Store. Hence, the data is symmetrical at any point of time.

As part of listener configuration you set up the port for gRPC channel through which each data center sends and receives the gossip. Later, with the ring configuration you establish connection with the associated data centers.

The underlying technology is same for both hot standby and active-active modes. The only difference is that, in the active-active mode you can have multiple data centers in a ring configuration and each data center can handle the client request. On the other hand, the hot standby data center can have only two data centers in a ring configuration and only one data center handles the client request.

Currently, the Cross-DC support in API Gateway synchronizes the following data across the data centers:

- Application assets such as API key, identifiers, strategy, and client registrations that are associated with the strategy
- OAuth tokens, auth code, refresh tokens, and so on
- OAuth and OpenID scope mapping

- Application that are registered to API

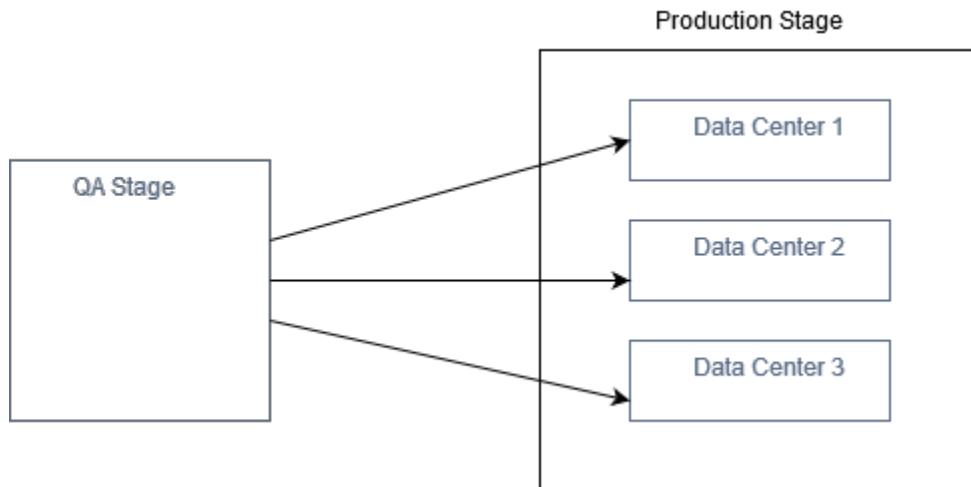
The below table suggests the possible workaround that could be employed for the different classes of data:

Data Type	Possible Workaround
APIs	Handle using Promotion Management.
Policies	Handle using Promotion Management.
Aliases	Handle using Promotion Management.
Packages	Handle using Promotion Management.
Plans	Handle using Promotion Management.
Subscriptions	Handle using Promotion Management.
Teams	Handle using Promotion Management.
Approval configurations	Handle using Promotion Management.
Keystore and Truststore configurations	Handle using Promotion Management.
Group	Handle using Promotion Management.
Email destination	Handle using Promotion Management.
JMS connection alias	Handle using Promotion Management.
Web service endpoint alias	Handle using Promotion Management.
Custom destination	Handle using Promotion Management.
Port	Handle using Promotion Management.
Service Registry	Handle using Promotion Management.
LDAP configuration	Handle using Promotion Management.
Password expiry settings	Handle using Promotion Management.
Analytics and Transaction Logs	Handle using external destinations.
Server Logs	Handle by pushing the logs to a centralized server.

Promotion Management in Cross-DC Support

Promotion refers to moving API Gateway assets from the source stage to one or more target stages. For example, you might want to promote assets you have developed on servers in a QA stage (the source API Gateway instance) to data centers in a Production stage (the target API Gateway).

instance). If you have three data centers in a Production stage, you have to explicitly promote the API Gateway assets to each data center.



When you promote an asset from one stage to another, the asset's metadata is copied from the source instance to the target instance.

How Do I Set Up the Data Centers in Hot Standby Mode Using Basic Operation?

Before you start setting up the data centers for Cross-DC support, ensure that you have:

- *Manage general administration configurations* functional privilege.
- API Gateway installed in all the data centers.

This use case explains how to set up data centers in hot standby mode. When you want to set up the data centers at a unit level, you can use this method.

The data centers are set up in hot standby mode using the REST APIs. You can find the REST API in the swagger file `APIGatewayDataManagement.json` located at `SAG_Root/IntegrationServer/instances/default/packages/WmAPIGateway/resources/apigatewayservices`.

For example, assume that you have two data centers DC 1 and DC 2 in the following landscape:

Data Center Name	Host Name	Region
DC 1	uk.myhost.com	United Kingdom
DC 2	us.myhost.com	United States

➤ To set up the data centers in hot standby mode

1. Configuring listener.

Configure the listener in both the data centers DC 1 and DC 2 using the **PUT/rest/apigateway/dataspace/listener** REST API.

Request: PUT <http://uk.myhost.com:5555/rest/apigateway/dataspace/listener>.

Sample payload for the DC 1 is as follows:

```
{  
    "listener": {  
        "nodeName": "ecb1308f-22ac-4877-aba9-471a31a834e6",  
        "host": "uk.myhost.com",  
        "port": 4440  
    },  
}
```

The system assigns unique node name for each data center. You must know the node name to configure the data centers as listener and to establish a ring. If you are unaware of the node names, invoke the **GET/rest/apigateway/dataspace** REST API on that data center whose node name you want to know. If you have multiple API Gateway instances clustered in a data center and when you use load balancer for high availability between the API Gateway instances, you have to provide the load balancer URL as host in the payload.

HTTP response appears as follows:

```
{  
    "listener": {  
        "nodeName": "ecb1308f-22ac-4877-aba9-471a31a834e6",  
        "host": "uk.myhost.com",  
        "port": 4440  
    },  
}
```

Note:

Similarly, you can configure the listener on DC 2 by invoking the **PUT/rest/apigateway/dataspace/listener** REST API with the respective payload.

On successful configuration, the response status code displays as 200 and you can see the corresponding log entry in the **Server Logs**.

2. Establishing ring.

Establish a fully connected network, where both the data centers are inter-connected and forms a ring using the **PUT/rest/apigateway/dataspace/ring** REST API. You must invoke this REST API on both the data centers DC 1 and DC 2.

Request: PUT <http://uk.myhost.com:5555/rest/apigateway/dataspace/ring>.

Sample payload for DC 1 is as follows:

```
{  
    "ring": [  
        {  
            "nodeName": "a04609a0-ca13-44db-98e1-f988ba18fbb4",  
            "host": "us.myhost.com",  
            "port": 4440  
        }  
    ]  
}
```

```

        }
    ]
}

```

Note:

When you establish the ring configuration on DC 1, you have to specify the DC 2 details in the payload. Similarly, when you establish the ring configuration on DC 2, you have to specify the DC 1 details in the payload.

HTTP response appears as follows:

```
{
  "ring": [
    {
      "nodeName": "a04609a0-ca13-44db-98e1-f988ba18fbb4",
      "host": "us.myhost.com",
      "port": 4440
    }
  ]
}
```

Note:

Similarly, you can establish the ring on DC 2 by invoking the **PUT/rest/apigateway/dataspace/ring** REST API with the respective payload.

On successful configuration, the response status code displays as 200 and you can see the corresponding log entry in the **Server Logs**.

3. Securing the Remote Procedure Call (gRPC) channel.

This is optional. You update the configuration only when you want to secure the gRPC channel. In Cross-DC support, the communication between data centers happens through gRPC channel. Securing the gRPC channel prevents data leaks and cyber attacks. You can secure the gRPC channel of all data centers by updating the configuration with keystore and truststore information. The gRPC channel is secured by configuring keystore and truststore with self-signed or CA signed certificates. Make sure that you have configured keystore and truststore in the API Gateway instance running on the data center for which you want to secure the gRPC channel. For information about configuring keystore and truststore, see ["Keystore and Truststore" on page 548](#). You have to update the listener configuration using the **PUT/rest/apigateway/dataspace/listener** REST API with keystore and truststore details on both the data centers DC 1 and DC 2 to secure the gRPC channel.

Request: `PUT http://uk.myhost.com:5555/rest/apigateway/dataspace/listener`

Sample payload for DC 1 that uses SSL certificate is as follows:

```
{
  "keyStoreAlias": "UK_Key",
  "keyAlias": "Key_Alias_UK",
  "trustStoreAlias": "Trustpackage"
  "listener": {
    "nodeName": "ecb1308f-22ac-4877-aba9-471a31a834e6",
    "host": "uk.myhost.com",
    "port": 4440
  }
}
```

```
    },
    "insecureTrustManager": false
}
```

HTTP response appears as follows:

```
{
  "keyStoreAlias": "UK_Key",
  "keyAlias": "Key_Alias_UK",
  "trustStoreAlias": "Trustpackage"
  "listener": {
    "nodeName": "ecb1308f-22ac-4877-aba9-471a31a834e6",
    "host": "uk.myhost.com",
    "port": 4440
  },
  "insecureTrustManager": false,
  "$resourceID": "listener"
}
```

Note:

- If you have configured the truststore using CA signed certificate, then in the payload, set "insecureTrustManager": false.
- Invoke the **PUT/rest/apigateway/dataspace/listener** REST API on DC 2 and provide a similar payload for DC 2.

On successful configuration, the response status code displays as 200 and you can see the corresponding log entry in the **Server Logs**.

Important:

Whenever you update the listener configuration, make sure you update the ring configuration in all the associated data centers using the **PUT/rest/apigateway/dataspace/ring** REST API. For example, if you update the listener configuration on DC 1, you have to update the ring configuration on DC 2.

4. Activating data centers in hot standby mode.

Data centers can be activated in two different ways. You can activate each data center separately by invoking the **PUT/rest/apigateway/dataspace/activate** REST API from each data center or activate all the data centers in this mode at a time by invoking the **PUT/rest/apigateway/dataspace/activateAll?mode=STANDBY** REST API once from any one of the data centers.

- Activating individual data centers.

Activate DC 1 and DC 2 separately using the **PUT/rest/apigateway/dataspace/activate** REST API.

Request: `PUT http://uk.myhost.com:5555/rest/apigateway/dataspace/activate.`

Sample payload for DC 1 is as follows:

```
{
  "mode": "STANDBY"
}
```

HTTP response appears as follows:

```
{
  "mode": "STANDBY"
}
```

Note:

Similarly, you can activate DC 2 by invoking the **PUT/rest/apigateway/dataspace/activate** REST API with the respective payloads.

On successful activation, the response status code displays as 200 and you can see the corresponding log entry in the **Server Logs**.

- Activating multiple data centers.

Activate both DC 1 and DC 2 data centers in a single step using the **PUT/rest/apigateway/dataspace/activateAll?mode= STANDBY** REST API from any one of the data centers.

Request: PUT `http://uk.myhost.com:5555/rest/apigateway/dataspace/activateAll?mode=STANDBY`.

Sample payload for DC 1 is as follows:

```
{
  "local": {
    "host": "uk.myhost.com",
    "syncPort": 4440,
    "keyStoreAlias": "UK_Key",
    "keyAlias": "Key_Alias_UK",
    "trustStoreAlias": "Trustpackage",
    "insecureTrustManager": true
  },
  "remotes": [
    {
      "host": "us.myhost.com",
      "syncPort": 4440,
      "userName": "Administrator",
      "password": "manage",
      "keyStoreAlias": "US_Key",
      "keyAlias": "Key_Alias_US",
      "trustStoreAlias": "Trustpackage",
      "insecureTrustManager": true
    }
  ]
}
```

HTTP response appears as follows:

```
{
  "mode": "STANDBY",
  "local": {
    "host": "uk.myhost.com",
    "syncPort": 4440,
    "keyStoreAlias": "UK_Key",
    "keyAlias": "Key_Alias_UK",
```

```
        "trustStoreAlias": "Trustpackage",
        "insecureTrustManager": true
    },
    "remotes": [
        {
            "host": "us.myhost.com",
            "syncPort": 4440,
            "userName": "Administrator",
            "password": "manage",
            "keyStoreAlias": "US_Key",
            "keyAlias": "Key_Alias_US",
            "trustStoreAlias": "Trustpackage",
            "insecureTrustManager": true
        }
    ],
    "acknowledged": true
}
```

On successful activation, the response status code displays as 200 and you can see the corresponding log entry in the **Server Logs**.

You can validate whether the data center is activated in the respective mode by reading the current configuration of the data center using the **GET/rest/apigateway/dataspace** REST API. For more information, see [“How Do I Read the Current Configuration of the Data Center?” on page 753](#).

Note:

When DC 1 fails, you have to reconfigure the load balancer with DC 2 details, so that DC 2 handles the client request. When DC 1 is restored back, it gets added to the ring automatically as a standby data center. DC 2 continues to handle the client requests.

If you want to replace any one of the data centers (DC 1 or DC 2) with a new data center (for example, DC 3) in hot standby mode, you have to bring down either DC 1 or DC 2 to standalone mode. For example, if you bring down DC 2 to standalone mode, then you can reconfigure the setup with DC 1 and DC 3 in hot standby mode.

How Do I Set Up the Data Centers in Hot Standby Mode Using Composite Operation?

This use case explains how to set up data centers in hot standby mode. When you want to set up the data centers simultaneously, you can use this method.

The data centers are set up in hot standby mode using the REST APIs. You can find the REST API in the swagger file `APIGatewayDataManagement.json` located at `SAG_Root/IntegrationServer/instances/default/packages/WmAPIGateway/resources/apigatewayservices`.

For example, assume that you have two data centers DC 1 and DC 2 in the following landscape:

Data Center Name	Host Name	Region
DC 1	uk.myhost.com	United Kingdom
DC 2	us.myhost.com	United States

➤ To set up the data centers in hot standby mode

1. Configuring data centers.

Configure and establish connection between DC 1 and DC 2 data centers in a single step rather than configuring the listener and ring separately using the **PUT/rest/apigateway/dataspace/configure** REST API. You can invoke this REST API on any one of the data centers (DC 1 or DC 2).

Request: PUT `http://uk.myhost.com:5555/rest/apigateway/dataspace/configure`.

Sample payload for DC 1 is as follows:

```
{
  "local": [
    {
      "host": "uk.myhost.com",
      "syncPort": 4440
    },
    "remotes": [
      [
        {
          "host": "us.myhost.com",
          "port": 5555,
          "syncPort": 4440,
          "userName": "Administrator",
          "password": "manage"
        }
      ]
    ]
}
```

Ensure that the local section in the payload contains the details of the data center on which you invoke the REST API. You must have the *Manage general administration configurations* functional privilege for the API Gateway instance running on the data center to authenticate the unit level operations that are performed simultaneously. If you have multiple API Gateway instances clustered in a data center and when you use load balancer for high availability between the API Gateway instances, then you have to provide the load balancer URL as host in the payload.

HTTP response appears as follows:

```
{
  "local": [
    {
      "host": "uk.myhost.com",
      "syncPort": 4440
    },
  ]}
```

```
"remotes":  
[  
{  
"host": "us.myhost.com",  
"port": 5555,  
"syncPort": 4440,  
"userName": "Administrator",  
"password": "manage"  
}  
]  
}
```

On successful configuration, the response status code displays as 200 and you can see the corresponding log entry in the **Server Logs**.

2. Securing the Remote Procedure Call (gRPC) channel.

This is optional. You update the configuration only when you want to secure the gRPC channel. In Cross-DC support, the communication between data centers happens through gRPC channel. Securing the gRPC channel prevents data leaks and cyber attacks. You can secure the gRPC channel of all the data centers by updating the configuration with keystore and truststore information. The gRPC channel is secured by configuring keystore and truststore with self-signed or CA signed certificates. Make sure that you have configured keystore and truststore in the API Gateway instance running on the data center for which you want to secure the gRPC channel. For information about configuring keystore and truststore, see [“Keystore and Truststore” on page 548](#). This configuration can be updated on any one of the data centers (DC 1 or DC 2) by invoking the **PUT/rest/apigateway/dataspace/configure** REST API with keystore and truststore details to secure the gRPC channel.

Request: PUT <http://uk.myhost.com:5555/rest/apigateway/dataspace/configure>.

Sample payload for DC 1 that uses SSL certificate is as follows:

```
{  
"local": {  
"host": "uk.myhost.com",  
"syncPort": 4440,  
"keyStoreAlias": "UK_Key",  
"keyAlias": "Key_Alias_UK",  
"trustStoreAlias": "Trustpackage",  
"insecureTrustManager": true  
},  
"remotes": [  
{  
"host": "us.myhost.com",  
"port": 5555,  
"syncPort": 4440,  
"userName": "Administrator",  
"password": "manage",  
"keyStoreAlias": "US_Key",  
"keyAlias": "Key_Alias_US",  
"trustStoreAlias": "Trustpackage",  
"insecureTrustManager": true  
}  
]  
}
```

HTTP response appears as follows:

```
{
  "local": {
    "host": "uk.myhost.com",
    "syncPort": 4440,
    "keyStoreAlias": "UK_Key",
    "keyAlias": "Key_Alias_UK",
    "trustStoreAlias": "Trustpackage",
    "insecureTrustManager": true
  },
  "remotes": [
    {
      "host": "us.myhost.com",
      "port": 5555,
      "syncPort": 4440,
      "userName": "Administrator",
      "password": "manage",
      "keyStoreAlias": "US_Key",
      "keyAlias": "Key_Alias_US",
      "trustStoreAlias": "Trustpackage",
      "insecureTrustManager": true
    }
  ]
}
```

Note:

If you have configured the truststore using CA signed certificate, then in the payload, set "insecureTrustManager": false.

On successful configuration, the response status code displays as 200 and you can see the corresponding log entry in the **Server Logs**.

3. Configuring data centers to use HTTPS port.

This is optional. You update the configuration, if the API Gateway instances running on the data center use HTTPS port. By default, API Gateway is available on a HTTP port. You can also make API Gateway available on an external HTTPS port to establish a secure connection. If you make API Gateway available on a HTTPS port, then you must update the configuration with the HTTPS port details. Make sure you have added and enabled the HTTPS port in the API Gateway instance running on the data center. You must also make sure that you have configured the listener specific credentials to the added port. For information about adding HTTPS port, see ["Adding an HTTPS Port" on page 560](#). This configuration can be updated on any one of the data centers (DC 1 or DC 2) by invoking the **PUT/rest/apigateway/dataspace/configure** REST API with HTTPS port details to secure the ports.

Request: `PUT https://uk.myhost.com:2503/rest/apigateway/dataspace/configure`.

Sample payload for DC 1 is as follows:

```
{
  "local": {
    "host": "uk.myhost.com",
    "port": 2503,
```

```
"isHttps": true,
"syncPort": 4440,
"keyStoreAlias":"UK_Key",
"keyAlias":"Key_Alias_UK",
"trustStoreAlias":"Trustpackage",
"insecureTrustManager": true
},
"remotes":
[
{
"host": "us.myhost.com",
"port": 2505,
"isHttps": true,
"syncPort": 4440,
"userName": "Administrator",
"password": "manage",
"keyStoreAlias":"US_Key",
"keyAlias":"Key_Alias_US",
"trustStoreAlias":"Trustpackage",
"insecureTrustManager": true
}
]
```

HTTP response appears as follows:

```
{
  "local": {
    "host": "uk.myhost.com",
    "port": 2503,
    "isHttps": true,
    "syncPort": 4440,
    "keyStoreAlias": "UK_Key",
    "keyAlias": "Key_Alias_UK",
    "trustStoreAlias": "Trustpackage",
    "insecureTrustManager": true
  },
  "remotes": [
    {
      "host": "us.myhost.com",
      "port": 2505,
      "isHttps": true,
      "syncPort": 4440,
      "userName": "Administrator",
      "password": "manage",
      "keyStoreAlias": "US_Key",
      "keyAlias": "Key_Alias_US",
      "trustStoreAlias": "Trustpackage",
      "insecureTrustManager": true
    }
  ]
}
```

On successful configuration, the response status code appears as 200 and you can see the corresponding log entry in the **Server Logs**.

4. Activating data centers in hot standby mode.

Data centers can be activated in two different ways. You can activate each data center separately by invoking the **PUT/rest/apigateway/dataspace/activate** REST API from each data center or activate all the data centers in this mode at a time by invoking the **PUT/rest/apigateway/dataspace/activateAll?mode= STANDBY** REST API once on any one of the data centers.

- Activating individual data centers.

Activate DC 1 and DC 2 separately using the **PUT/rest/apigateway/dataspace/activate** REST API.

Request: PUT <https://uk.myhost.com:2503/rest/apigateway/dataspace/activate>.

Sample payload for DC1 is as follows:

```
{
  "mode": "STANDBY"
}
```

HTTP response appears as follows:

```
{
  "mode": "STANDBY"
}
```

Note:

Similarly, you can activate DC 2 data center by invoking the **PUT/rest/apigateway/dataspace/activate** REST API with the respective payloads.

On successful activation, the response status code appears as 200 and you can see the corresponding log entry in the **Server Logs**.

- Activating multiple data centers.

Activate DC 1 and DC 2 data centers in a single step using the **PUT/rest/apigateway/dataspace/activateAll?mode= STANDBY** REST API on any one of the data centers (DC 1 or DC 2)

Request: PUT <https://uk.myhost.com:2503/rest/apigateway/dataspace/activateAll?mode=STANDBY>.

Sample payload is as follows:

```
{
  "local": {
    "host": "uk.myhost.com",
    "port": 2503,
    "isHttps": true,
    "syncPort": 4440,
    "keyStoreAlias": "UK_Key",
    "keyAlias": "Key_Alias_UK",
    "trustStoreAlias": "Trustpackage",
    "insecureTrustManager": true
  },
  "remotes": [
    {
      "host": "us.myhost.com",
      "port": 2503,
      "isHttps": true,
      "syncPort": 4440,
      "keyStoreAlias": "US_Key",
      "keyAlias": "Key_Alias_US",
      "trustStoreAlias": "Trustpackage",
      "insecureTrustManager": true
    }
  ]
}
```

```
{  
  "host": "us.myhost.com",  
  "port": 2505,  
  "isHttps": true,  
  "syncPort": 4440,  
  "userName": "Administrator",  
  "password": "manage",  
  "keyStoreAlias": "US_Key",  
  "keyAlias": "Key_Alias_US",  
  "trustStoreAlias": "Trustpackage",  
  "insecureTrustManager": true  
}  
]  
}
```

HTTP response appears as follows:

```
{  
  "local": {  
    "host": "uk.myhost.com",  
    "port": 2503,  
    "isHttps": true,  
    "syncPort": 4440,  
    "keyStoreAlias": "UK_Key",  
    "keyAlias": "Key_Alias_UK",  
    "trustStoreAlias": "Trustpackage",  
    "insecureTrustManager": true  
  },  
  "remotes": [  
    {  
      "host": "us.myhost.com",  
      "port": 2505,  
      "isHttps": true,  
      "syncPort": 4440,  
      "userName": "Administrator",  
      "password": "manage",  
      "keyStoreAlias": "US_Key",  
      "keyAlias": "Key_Alias_US",  
      "trustStoreAlias": "Trustpackage",  
      "insecureTrustManager": true  
    }  
  ]  
}
```

On successful activation, the response status code displays as *200* and you can see the corresponding log entry in the **Server Logs**.

You can validate whether the data center is activated in the respective mode by reading the current configuration of the data center using the **GET/rest/apigateway/dataspace** REST API. For more information, see “[How Do I Read the Current Configuration of the Data Center?](#)” on page 753.

Note:

When DC 1 fails, you have to reconfigure the load balancer with DC 2 details, so that DC 2 handles the client request. When DC 1 is restored back, it gets added to the ring automatically as a standby data center. DC 2 continues to handle the client requests.

If you want to replace any one of the data centers (DC 1 or DC 2) with a new data center (for example, DC 3) in hot standby mode, you have to bring down either DC 1 or DC 2 to standalone mode. For example, if you bring down DC 2 to standalone mode, then you can reconfigure the setup with DC 1 and DC 3 in hot standby mode.

How Do I Set Up the Data Centers in Active-Active Mode Using Basic Operation?

This use case explains how to set up the data centers in the active-active mode. When you want to set up the data centers at a unit level, you can use this method.

The data centers are set up in active-active mode using the REST APIs. You can find the REST API in the swagger file `APIGatewayDataManagement.json` located at `SAG_Root/IntegrationServer/instances/default/packages/WmAPIGateway/resources/apigatewayservices`.

For example, assume that you have three data centers DC 1, DC 2, and DC 3 in the following landscape:

Data Center Name	Host Name	Region
DC 1	uk.myhost.com	United Kingdom
DC 2	us.myhost.com	United States
DC 3	in.myhost.com	India

In general, the active-active mode can accommodate any number of data centers.

➤ To set up the data centers in active-active mode

1. Configuring listener.

Configure the listener on all the data centers (DC 1, DC 2, and DC 3) using the **PUT/rest/apigateway/dataspace/listener** REST API.

Request: `PUT http://uk.myhost.com:5555/rest/apigateway/dataspace/listener`.

Sample payload for the DC 1 is as follows:

```
{
  "listener": {
    "nodeName": "ecb1308f-22ac-4877-aba9-471a31a834e6",
    "host": "uk.myhost.com",
    "port": 4440
  },
}
```

The system assigns unique node name for each data center. You must know the node name to configure the data centers as listener and to establish a ring. If you are unaware of the node names, invoke the **GET/rest/apigateway/dataspace** REST API on that data center whose

node name you want to know. If you have multiple API Gateway instances clustered in a data center and when you use load balancer for high availability between the API Gateway instances, you have to provide the load balancer URL as host in the payload.

HTTP response appears as follows:

```
{  
  "listener": {  
    "nodeName": "ecb1308f-22ac-4877-aba9-471a31a834e6",  
    "host": "uk.myhost.com",  
    "port": 4440  
  },  
}
```

Note:

Similarly, you can configure the listener on DC 2 and DC 3 by invoking the `PUT/rest/apigateway/dataspace/listener` REST API with the respective payload.

On successful configuration, the response status code displays as 200 and you can see the corresponding log entry in the **Server Logs**.

2. Establishing ring.

Establish a fully connected network, where all the data centers are inter-connected and forms a ring using the `PUT/rest/apigateway/dataspace/ring` REST API. You must invoke this REST API on all the data centers (DC 1, DC 2, and DC 3).

Request: `PUT http://uk.myhost.com:5555/rest/apigateway/dataspace/ring`.

Sample payload for DC 1 is as follows:

```
{  
  "ring": [  
    {  
      "nodeName": "a04609a0-ca13-44db-98e1-f988ba18fbb4",  
      "host": "us.myhost.com",  
      "port": 4440  
    },  
    {  
      "nodeName": "4820681b-f2fd-42d7-bccd-cf580ea8bf1c",  
      "host": "in.myhost.com",  
      "port": 4440  
    }  
  ]  
}
```

Note:

When you configure the ring from one data center, you have to provide the details of other associated data centers with which you want to establish the ring configuration in the payload. For example, when you establish the ring configuration from DC 1, you have to specify the DC 2 and DC 3 details in the payload. Similarly, when you establish the ring configuration from DC 2, you have to specify the DC 3 and DC 1 details in the payload.

Likewise, when you establish the ring configuration from DC 3, you have to specify the DC 2 and DC 1 details in the payload.

HTTP response appears as follows:

```
{
  "ring": [
    {
      "nodeName": "a04609a0-ca13-44db-98e1-f988ba18fbb4",
      "host": "us.myhost.com",
      "port": 4440
    },
    {
      "nodeName": "4820681b-f2fd-42d7-bccd-cf580ea8bf1c",
      "host": "in.myhost.com",
      "port": 4440
    }
  ]
}
```

Note:

Similarly, you can configure the ring on DC 2 and DC 3 by invoking the **PUT/rest/apigateway/dataspace/ring** REST API with the respective payload.

On successful configuration, the response status code displays as **200** and you can see the corresponding log entry in the **Server Logs**.

3. Securing the Remote Procedure Call (gRPC) channel.

This is optional. You update the configuration only when you want to secure the gRPC channel. In Cross-DC support, the communication between data centers happens through gRPC channel. Securing the gRPC channel prevents data leaks and cyber attacks. You can secure the gRPC channel of all the data centers by updating the configuration with keystore and truststore information. The gRPC channel is secured by configuring keystore and truststore with self-signed or CA signed certificates. Make sure that you have configured keystore and truststore in the API Gateway instance running on the data center for which you want to secure the gRPC channel. For information about configuring keystore and truststore, see "[Keystore and Truststore](#)" on page 548. You have to update the listener configuration with keystore and truststore details using this **PUT/rest/apigateway/dataspace/listener** REST API with keystore and truststore details on all the data centers DC 1, DC 2, and DC 3 to secure the gRPC channel.

Request: `PUT http://uk.myhost.com:5555/rest/apigateway/dataspace/listener.`

Sample payload for DC 1 that uses SSL certificate is as follows:

```
{
  "keyStoreAlias": "UK_Key",
  "keyAlias": "Key_Alias_UK",
  "trustStoreAlias": "Trustpackage"
  "listener": {
    "nodeName": "ecb1308f-22ac-4877-aba9-471a31a834e6",
    "host": "uk.myhost.com",
    "port": 4440
  },
  "insecureTrustManager": false
}
```

```
}
```

HTTP response appears as follows:

```
{
  "keyStoreAlias": "UK_Key",
  "keyAlias": "Key_Alias_UK",
  "trustStoreAlias": "Trustpackage"
  "listener": {
    "nodeName": "ecb1308f-22ac-4877-aba9-471a31a834e6",
    "host": "uk.myhost.com",
    "port": 4440
  },
  "insecureTrustManager": false,
  "$resourceID": "listener"
}
```

Note:

- If you have configured the truststore using CA signed certificate, then in the payload, set "insecureTrustManager": false.
- Invoke the **PUT/rest/apigateway/dataspace/listener** REST API on DC 2 and DC 3. Provide a similar payload for DC 2 and DC 3.

On successful configuration, the response status code displays as 200 and you can see the corresponding log entry in the **Server Logs**.

Important:

Whenever you update the listener configuration, make sure you update the ring configuration in all the associated data centers using the **PUT/rest/apigateway/dataspace/ring** REST API. For example, if you update the listener configuration on DC 1, you have to update the ring configuration on DC 2 and DC 3.

4. Activating data centers in active-active mode.

Data centers can be activated in two different ways. You can activate each data center separately by invoking the **PUT/rest/apigateway/dataspace/activate** REST API from each data center or activate all the data centers in this mode at a time by invoking the **PUT/rest/apigateway/dataspace/activateAll?mode= ACTIVE_RING** REST API once on any one of the data centers.

- Activating individual data centers.

Activate DC 1, DC 2, and DC 3 separately using the **PUT/rest/apigateway/dataspace/activate** REST API.

Request: `PUT http://uk.myhost.com:5555/rest/apigateway/dataspace/activate.`

Sample payload for DC 1 is as follows:

```
{
  "mode": "ACTIVE_RING"
}
```

HTTP response appears as follows:

```
{
  "mode": "ACTIVE_RING"
}
```

Note:

Similarly, you can activate DC 2 and DC 3 data centers by invoking the **PUT/rest/apigateway/dataspace/activate** REST API with the respective payloads.

On successful activation, the response status code displays as 200 and you can see the corresponding log entry in the **Server Logs**.

- Activating multiple data centers.

Activate DC 1, DC 2, and DC 3 data centers in a single step using the **PUT/rest/apigateway/dataspace/activateAll?mode= ACTIVE_RING** REST API in any one of the data centers.

Request: `PUT http://uk.myhost.com:5555/rest/apigateway/dataspace/activateAll?mode=ACTIVE_RING`.

Sample payload for DC 1 is as follows:

```
{
  "local": {
    "host": "uk.myhost.com",
    "syncPort": 4440,
    "keyStoreAlias": "UK_Key",
    "keyAlias": "Key_Alias_UK",
    "trustStoreAlias": "Trustpackage",
    "insecureTrustManager": true
  },
  "remotes": [
    {
      "host": "us.myhost.com",
      "syncPort": 4440,
      "userName": "Administrator",
      "password": "manage",
      "keyStoreAlias": "US_Key",
      "keyAlias": "Key_Alias_US",
      "trustStoreAlias": "Trustpackage",
      "insecureTrustManager": true
    },
    {
      "host": "in.myhost.com",
      "syncPort": 4440,
      "userName": "Administrator",
      "password": "manage",
      "keyStoreAlias": "IN_Key",
      "keyAlias": "Key_Alias_IN",
      "trustStoreAlias": "Trustpackage",
      "insecureTrustManager": true
    }
  ]
}
```

HTTP response appears as follows:

```
{  
    "mode": "ACTIVE_RING",  
    "local": {  
        "host": "uk.myhost.com",  
        "syncPort": 4440,  
        "keyStoreAlias": "UK_Key",  
        "keyAlias": "Key_Alias_inchn",  
        "trustStoreAlias": "Trustpackage",  
        "insecureTrustManager": true  
    },  
    "remotes": [  
        {  
            "host": "us.myhost.com",  
            "syncPort": 4440,  
            "userName": "Administrator",  
            "password": "manage",  
            "keyStoreAlias": "US_Key",  
            "keyAlias": "Key_Alias_US",  
            "trustStoreAlias": "Trustpackage",  
            "insecureTrustManager": true  
        },  
        {  
            "host": "in.myhost.com",  
            "syncPort": 4440,  
            "userName": "Administrator",  
            "password": "manage",  
            "keyStoreAlias": "IN_Key",  
            "keyAlias": "Key_Alias_IN",  
            "trustStoreAlias": "Trustpackage",  
            "insecureTrustManager": true  
        }  
    ],  
    "acknowledged": true  
}
```

On successful activation, the response status code displays as 200 and you can see the corresponding log entry in the **Server Logs**.

You can validate whether the data center is activated in the respective mode by reading the current configuration of the data center using the **GET/rest/apigateway/dataspace** REST API. For more information, see “[How Do I Read the Current Configuration of the Data Center?](#)” on page 753.

Note:

In active-active, if any one of the data center (DC 1 or DC 2 or DC 3) goes down, then that data center is removed from the ring. When the same data center is restored back, then that data center gets added to the ring automatically. If you want to add one more new data center (DC 4) to the ring, then you have to update the configuration with DC 4.

How Do I Set Up the Data Centers in Active-Active Mode Using Composite Operation?

This use case explains how to set up the data centers in the active-active mode. When you want to set up the data centers simultaneously, you can use this method.

The data centers are set up in active-active mode using the REST APIs. You can find the REST API in the swagger file `APIGatewayDataManagement.json` located at `SAG_Root/IntegrationServer/instances/default/packages/WmAPIGateway/resources/apigatewayservices`.

For example, assume that you have three data centers DC 1, DC 2, and DC 3 in the following landscape:

Data Center Name	Host Name	Region
DC 1	uk.myhost.com	United Kingdom
DC 2	us.myhost.com	United States
DC 3	in.myhost.com	India

In general, the active-active mode can accommodate any number of data centers.

➤ To set up the data centers in active-active mode

1. Configuring multiple data centers.

Configure and establish connection between multiple data centers in a single step rather than configuring the listener and ring separately using the **PUT/rest/apigateway/dataspace/configure** REST API. You can invoke this REST API on any one of the data centers (DC 1 or DC 2 or DC 3).

Request: `PUT http://uk.myhost.com:5555/rest/apigateway/dataspace/configure`.

Sample payload for DC 1 is as follows:

```
{
  "local": [
    {
      "host": "uk.myhost.com",
      "syncPort": 4440
    },
    "remotes": [
      [
        {
          "host": "us.myhost.com",
          "port": 5555,
          "syncPort": 4440,
          "userName": "Administrator",
          "password": "manage"
        },
        {
          "host": "in.myhost.com",
          "port": 5555,
          "syncPort": 4440,
          "userName": "Administrator",
          "password": "manage"
        }
      ]
    ]
}
```

```
}
```

Ensure that the local section in the payload contains the details of the data center on which you invoke the REST API. You must have the *Manage general administration configurations* functional privilege for the API Gateway instance running on the data center to authenticate the unit level operations that are performed simultaneously. If you have multiple API Gateway instances clustered in a data center and when you use load balancer for high availability between the API Gateway instances, then you have to provide the load balancer URL as host in the payload.

HTTP response appears as follows:

```
{
  "local": [
    {
      "host": "uk.myhost.com",
      "syncPort": 4440
    },
    "remotes": [
      {
        "host": "us.myhost.com",
        "port": 5555,
        "syncPort": 4440,
        "userName": "Administrator",
        "password": "manage"
      },
      {
        "host": "in.myhost.com",
        "port": 5555,
        "syncPort": 4440,
        "userName": "Administrator",
        "password": "manage"
      }
    ]
}
```

On successful configuration, the response status code displays as 200 and you can see the corresponding log entry in the **Server Logs**.

2. Securing the Remote Procedure Call (gRPC) channel.

This is optional. You update the configuration only when you want to secure the gRPC channel. In Cross-DC support, the communication between data centers happens through gRPC channel. Securing the gRPC channel prevents data leaks and cyber attacks. You can secure the gRPC channel of all the data centers by updating the configuration with keystore and truststore information. The gRPC channel is secured by configuring keystore and truststore with self-signed or CA signed certificates. Make sure that you have configured keystore and truststore in the API Gateway instance running on the data center for which you want to secure the gRPC channel. For information about configuring keystore and truststore, see “[Keystore and Truststore](#)” on page 548. This configuration can be updated on anyone of the data centers (DC 1 or DC 2 or DC 3) by invoking the **PUT/rest/apigateway/dataspace/configure** REST API with keystore and truststore details to secure the gRPC channel.

Request: `PUT http://uk.myhost.com:5555/rest/apigateway/dataspace/configure.`

Sample payload for DC 1 that uses SSL certificate is as follows:

```
{
  "local": {
    "host": "uk.myhost.com",
    "syncPort": 4440,
    "keyStoreAlias": "UK_Key",
    "keyAlias": "Key_Alias_UK",
    "trustStoreAlias": "Trustpackage",
    "insecureTrustManager": true
  },
  "remotes": [
    {
      "host": "us.myhost.com",
      "port": 5555,
      "syncPort": 4440,
      "userName": "Administrator",
      "password": "manage",
      "keyStoreAlias": "US_Key",
      "keyAlias": "Key_Alias_US",
      "trustStoreAlias": "Trustpackage",
      "insecureTrustManager": true
    },
    {
      "host": "in.myhost.com",
      "port": 5555,
      "syncPort": 4440,
      "userName": "Administrator",
      "password": "manage",
      "keyStoreAlias": "IN_Key",
      "keyAlias": "Key_Alias_IN",
      "trustStoreAlias": "Trustpackage",
      "insecureTrustManager": true
    }
  ]
}
```

HTTP response appears as follows:

```
{
  "local": {
    "host": "uk.myhost.com",
    "syncPort": 4440,
    "keyStoreAlias": "UK_Key",
    "keyAlias": "Key_Alias_UK",
    "trustStoreAlias": "Trustpackage",
    "insecureTrustManager": true
  },
  "remotes": [
    {
      "host": "us.myhost.com",
      "port": 5555,
      "syncPort": 4440,
      "userName": "Administrator",
      "password": "manage",
      "keyStoreAlias": "US_Key",
      "keyAlias": "Key_Alias_US",
      "trustStoreAlias": "Trustpackage",
      "insecureTrustManager": true
    },
  ]
}
```

```
{  
  "host": "in.myhost.com",  
  "port": 5555,  
  "syncPort": 4440,  
  "userName": "Administrator",  
  "password": "manage",  
  "keyStoreAlias": "IN_Key",  
  "keyAlias": "Key_Alias_IN",  
  "trustStoreAlias": "Trustpackage",  
  "insecureTrustManager": true  
}  
]  
}
```

Note:

If you have configured the truststore using CA signed certificate, then in the payload, set "insecureTrustManager": false.

On successful configuration, the response status code displays as 200 and you can see the corresponding log entry in the **Server Logs**.

3. Configuring data centers to use HTTPS port.

This is optional. You update the configuration, if the API Gateway instances running on the data center use HTTPS port. By default, API Gateway is available on a HTTP port. You can also make API Gateway available on an external HTTPS port to establish a secure connection. If you make API Gateway available on a HTTPS port, then you must update the configuration with the HTTPS port details. Make sure you have added and enabled the HTTPS port in the API Gateway instance running on the data center. You must also make sure that you have configured the listener specific credentials to the added port. For information about adding HTTPS port, see ["Adding an HTTPS Port" on page 560](#). This configuration can be updated on any one of the data centers (DC 1 or DC 2 or DC 3) by invoking the

PUT/rest/apigateway/dataspace/configure REST API with HTTPS port details to secure the ports.

Request: PUT <https://uk.myhost.com:2503/rest/apigateway/dataspace/configure>.

Sample payload for using secure port is as follows:

```
{  
  "local": {  
    "host": "uk.myhost.com",  
    "port": 2503,  
    "isHttps": true,  
    "syncPort": 4440,  
    "keyStoreAlias": "UK_Key",  
    "keyAlias": "Key_Alias_UK",  
    "trustStoreAlias": "Trustpackage",  
    "insecureTrustManager": true  
  },  
  "remotes": [  
    {  
      "host": "us.myhost.com",  
      "port": 2505,  
      "isHttps": true,  
      "syncPort": 4441,  
      "keyStoreAlias": "US_Key",  
      "keyAlias": "Key_Alias_US",  
      "trustStoreAlias": "Trustpackage",  
      "insecureTrustManager": true  
    }  
  ]  
}
```

```

    "isHttps": true,
    "syncPort": 4440,
    "userName": "Administrator",
    "password": "manage",
    "keyStoreAlias":"US_Key",
    "keyAlias":"Key_Alias_US",
    "trustStoreAlias":"Trustpackage",
    "insecureTrustManager": true
},
{
"host": "in.myhost.com",
"port": 2504,
"isHttps": true,
"syncPort": 4440,
"userName": "Administrator",
"password": "manage",
"keyStoreAlias":"IN_Key",
"keyAlias":"Key_Alias_IN",
"trustStoreAlias":"Trustpackage",
"insecureTrustManager": true
}
]
}

```

HTTP response appears as follows:

```

{
  "local": {
    "host": "uk.myhost.com",
    "port": 2503,
    "isHttps": true,
    "syncPort": 4440,
    "keyStoreAlias":"UK_Key",
    "keyAlias":"Key_Alias_UK",
    "trustStoreAlias": "Trustpackage",
    "insecureTrustManager": true
  },
  "remotes": [
    {
      "host": "us.myhost.com",
      "port": 2505,
      "isHttps": true,
      "syncPort": 4440,
      "userName": "Administrator",
      "password": "manage",
      "keyStoreAlias": "US_Key",
      "keyAlias": "Key_Alias_US",
      "trustStoreAlias": "Trustpackage",
      "insecureTrustManager": true
    },
    {
      "host": "in.myhost.com",
      "port": 2504,
      "isHttps": true,
      "syncPort": 4440,
      "userName": "Administrator",
      "password": "manage",
      "keyStoreAlias":"IN_Key",
      "keyAlias":"Key_Alias_IN",
      "trustStoreAlias":"Trustpackage",
      "insecureTrustManager": true
    }
  ]
}

```

```
        "insecureTrustManager": true
    }
]
}
```

On successful configuration, the response status code displays as 200 and you can see the corresponding log entry in the **Server Logs**.

4. Activating data centers.

Data centers can be activated in two different ways. You can activate each data center separately by invoking the **PUT/rest/apigateway/dataspace/activate** REST API from each data center or activate all the data centers in this mode at a time by invoking the **PUT/rest/apigateway/dataspace/activateAll?mode= ACTIVE_RING** REST API once on any one of the data centers.

- Activating individual data centers.

You can activate DC 1, DC 2, and DC 3 separately using the **PUT/rest/apigateway/dataspace/activate** REST API.

Request: PUT <https://uk.myhost.com:2503/rest/apigateway/dataspace/activate>.

Sample payload for DC 1 is as follows:

```
{
  "mode": "ACTIVE_RING"
}
```

HTTP response appears as follows:

```
{
  "mode": "ACTIVE_RING"
}
```

Note:

Similarly, you can activate DC 2 and DC 3 data centers by invoking the **PUT/rest/apigateway/dataspace/activate** REST API with the respective payloads.

On successful activation, the response status code displays as 200 and you can see the corresponding log entry in the **Server Logs**.

- Activating multiple data centers.

You can activate DC 1, DC 2, and DC 3 data centers in a single step using the **PUT/rest/apigateway/dataspace/activateAll?mode= ACTIVE_RING** REST API on any one of the data centers (DC 1 or DC 2 or DC 3).

Request: PUT https://uk.myhost.com:2503/rest/apigateway/dataspace/activateAll?mode= ACTIVE_RING.

Sample payload for DC 1 is as follows:

```
{
  "local": {
```

```

"host": "uk.myhost.com",
"port":2503,
"isHttps": true,
"syncPort": 4440,
"keyStoreAlias":"UK_Key",
"keyAlias":"Key_Alias_UK",
"trustStoreAlias":"Trustpackage",
"insecureTrustManager": true
},
"remotes":
[
{
"host": "us.myhost.com",
"port": 2505,
"isHttps": true,
"syncPort": 4440,
"userName": "Administrator",
"password": "manage",
"keyStoreAlias":"US_Key",
"keyAlias":"Key_Alias_US",
"trustStoreAlias":"Trustpackage",
"insecureTrustManager": true
},
{
"host": "in.myhost.com",
"port": 2504,
"isHttps": true,
"syncPort": 4440,
"userName": "Administrator",
"password": "manage",
"keyStoreAlias":"IN_Key",
"keyAlias":"Key_Alias_IN",
"trustStoreAlias":"Trustpackage",
"insecureTrustManager": true
}
]
}

```

HTTP response appears as follows:

```

{
"mode": "ACTIVE_RING",
"local": {
    "host": "uk.myhost.com",
    "port": 2503,
    "isHttps": true,
    "syncPort": 4440,
    "keyStoreAlias":"UK_Key",
    "keyAlias":"Key_Alias_UK",
    "trustStoreAlias": "Trustpackage",
    "insecureTrustManager": true
},
"remotes": [
{
    "host": "us.myhost.com",
    "port": 2505,
    "isHttps": true,
    "syncPort": 4440,
    "userName": "Administrator",
    "password": "manage",
}
]
}

```

```
        "keyStoreAlias": "US_Key",
        "keyAlias": "Key_Alias_US",
        "trustStoreAlias": "Trustpackage",
        "insecureTrustManager": true
    },
    {
        "host": "in.myhost.com",
        "port": 2504,
        "isHttps": true,
        "syncPort": 4440,
        "userName": "Administrator",
        "password": "manage",
        "keyStoreAlias": "IN_Key",
        "keyAlias": "Key_Alias_IN",
        "trustStoreAlias": "Trustpackage",
        "insecureTrustManager": true
    }
],
"acknowledged": true
}
```

On successful activation, the response status code displays as 200 and you can see the corresponding log entry in the **Server Logs**.

You can validate whether the data center is activated in the respective mode by reading the current configuration of the data center using the **GET/rest/apigateway/dataspace** REST API. For more information, see “[How Do I Read the Current Configuration of the Data Center?](#)” on page 753.

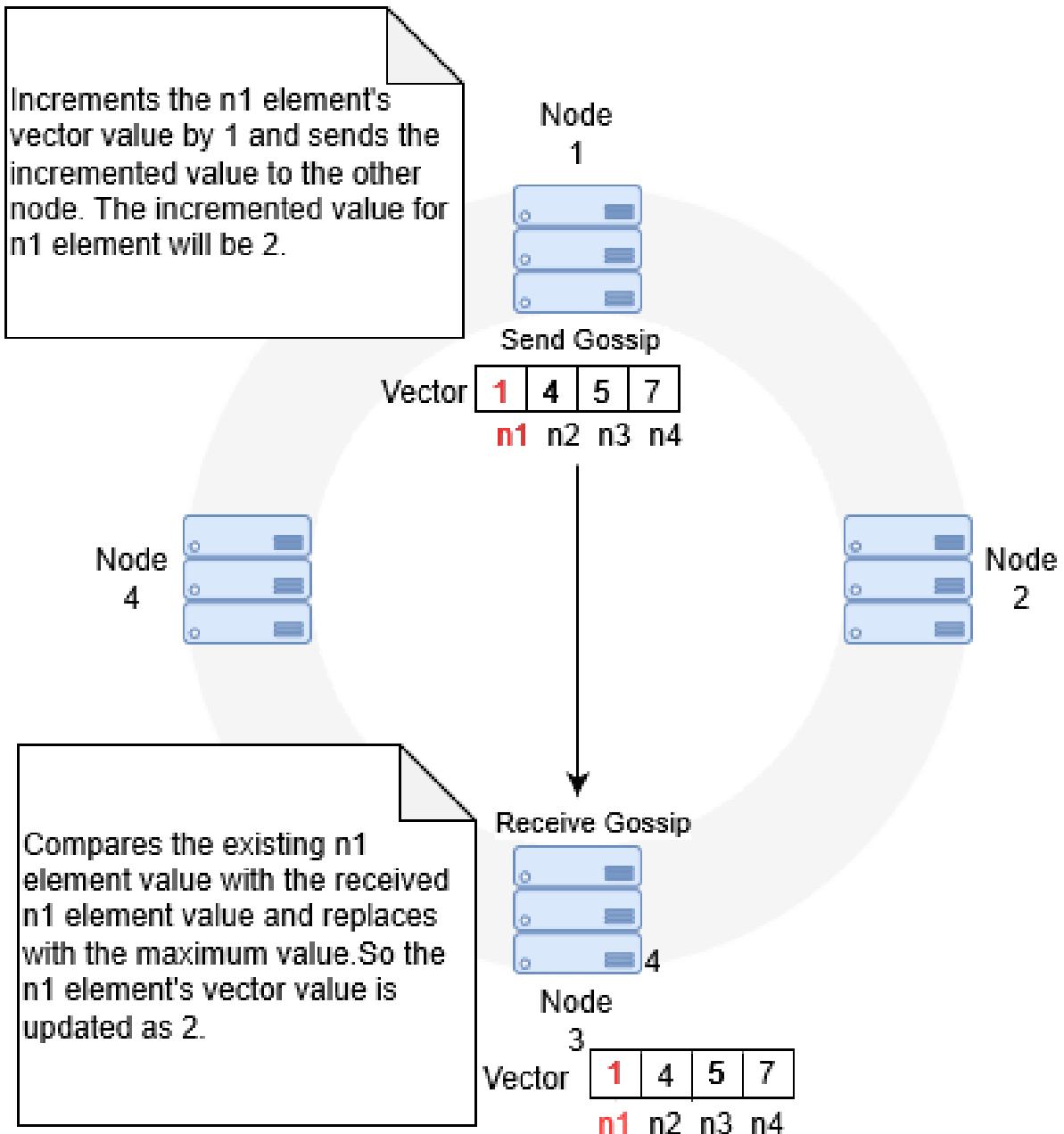
Note:

In active-active, if any one of the data center (DC 1 or DC 2 or DC 3) goes down, then that data center is removed from the ring. When the same data center is restored back, then that data center gets added to the ring automatically. If you want to add one more new data center (DC 4) to the ring, then you have to update the configuration with DC 4.

How does Cross-DC Support Detect Data Center Failures?

All the data centers are connected together to form a consistent hash ring using gRPC channel. The consistent hash ring is maintained properly with the help of *Gossiping protocol*. Using *Gossiping protocol*, API Gateway detects the failure of nodes. Here nodes represent the data centers.

This is how the *Gossiping protocol* works in Cross-DC support in active-active mode. Each node sends and receives gossip between one another to ensure that they are up and running. Each node has a vector of elements. For example, if there are n nodes in a ring then each node has a vector with n elements.



Each node sends and receives gossips with one another. When a node receives a gossip, it compares the received vector element value with the existing vector element value and replaces the vector element of the received node with the maximum value. When the vector element value is replaced, then that particular time frame gets captured. If the last updated time interval happens to be greater than permissible time interval between two consecutive gossips (that you define in the **pg_Dataspace_TimeToFail** extended settings), then that particular node is marked as dead. If the last updated time interval is twice as that of permissible time interval, then that particular node is removed from the ring. When the dead nodes are up, the *Gossiping protocol* detects them and rehashes the nodes.

As in active-active mode, the hot standby mode also uses consistent hash ring and *Gossiping protocol* to detect the failure of nodes. But in the hot standby mode there are only two nodes in the ring.

In hot standby mode, if the primary node goes down, the API Gateway administrator receives a notification and reconfigures the load balancer with the secondary node details. Hence, the secondary node handles the client request. When the primary node is restored back, the node gets added to the ring automatically. If the secondary node goes down, when the primary is active, then the secondary node is removed from the ring. When the secondary is restored it gets added to ring automatically.

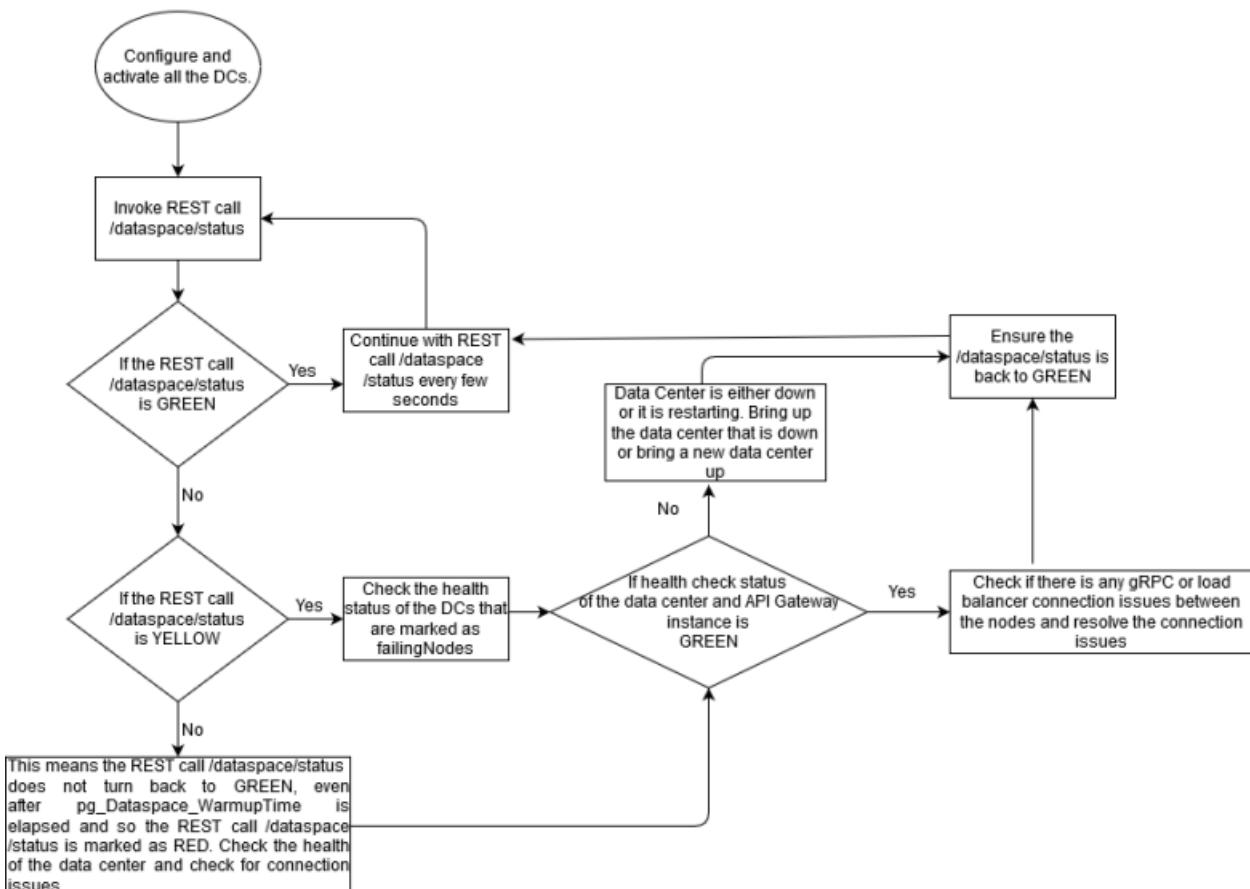
How Do I Monitor the Health Status of the Data Center?

This use case explains how to track and monitor the health status of the data centers in the consistent hash ring using the REST API **GET /rest/apigateway/dataspace/status**. The status can be GREEN, YELLOW, or RED.

If the health status is GREEN, then all the data centers are able to communicate with one another without any problem.

If the health status is YELLOW, then it indicates that this data center is unable to communicate with one or more of the data centers in the ring. This might be because one or more data centers in the ring is down temporarily as it is getting restarted. It could also be permanently down because of connection issue or the API Gateway instance is down. The API Gateway waits for a certain period of time (as configured in the pg_Dataspace_TimeToFail) before marking the data center as down. Once the data center is marked as down, it will be marked as failed data center and health status will be marked as YELLOW. The node gets marked as YELLOW until the time configured in the pg_Dataspace_WarmupTime property gets elapsed. If the data center is not up even after the time specified in the pg_Dataspace_WarmupTime property, then the health status is marked as RED.

The following flow chart explains how the health of the data center is monitored using the GET /rest/apigateway/dataspace/status REST API:



For example, assume that you have three data centers DC 1, DC 2, and DC 3 in the following landscape:

Data Center Name	Host Name	Region
DC 1	uk.myhost.com	United Kingdom
DC 2	us.myhost.com	United States
DC 3	in.myhost.com	India

In general, the active-active mode can accommodate any number of data centers in the consistent hash ring. But in the hot standby mode there are only two nodes in the ring.

➤ To track the health status of the data centers

1. Invoke the REST API to track the health status.

Track the health status of the data center in the consistent hash ring using the REST API **GET /rest/apigateway/dataspace/status**

Note:

To invoke the REST API, you must provide the basic authentication.

For example:

Request: GET `http://uk.myhost.com:5555/rest/apigateway/dataspace/status`.

HTTP response appears as follows:

```
{  
    "detectedNodes": [  
        {  
            "nodeName": "a04609a0-ca13-44db-98e1-f988ba18fbb4",  
            "host": "us.myhost.com",  
            "port": 4440  
        },  
        {  
            "nodeName": "4820681b-f2fd-42d7-bccd-cf580ea8bf1c"  
            "host": "in.myhost.com",  
            "port": 4440  
        }  
    ],  
    "liveNodes":  
    "[a04609a0-ca13-44db-98e1-f988ba18fbb4],[4820681b-f2fd-42d7-bccd-cf580ea8bf1c]"  
    "health": "GREEN"  
}
```

When all the data center is able to communicate, the response status code displays as 200 and health status displays as GREEN.

2. Invoke the REST API to detect the failing nodes .

Detect the failing nodes in the consistent hash ring when the health status is in YELLOW using the REST API **GET /rest/apigateway/dataspace/status?watchFailingNodes=true**. For example:

Request: GET

`http://uk.myhost.com:5555/rest/apigateway/dataspace/status?watchFailingNodes=true`.

HTTP response appears as follows:

```
{  
    "detectedNodes": [  
        {  
            "nodeName": "a04609a0-ca13-44db-98e1-f988ba18fbb4",  
            "host": "us.myhost.com",  
            "port": 4440  
        },  
        {  
            "nodeName": "4820681b-f2fd-42d7-bccd-cf580ea8bf1c"  
            "host": "in.myhost.com",  
            "port": 4440  
        }  
    ],  
    "liveNodes":  
    "[a04609a0-ca13-44db-98e1-f988ba18fbb4],[4820681b-f2fd-42d7-bccd-cf580ea8bf1c]"  
    "failingNodes": "[4820681b-f2fd-42d7-bccd-cf580ea8bf1c]"  
    "health": "YELLOW"  
}
```

```
}
```

Though the DC 3 is detected as failing node, the response status code displays as 200 and health status displays as YELLOW.

Note:

If the DC 3 does not come back even after the time specified in the pg_Dataspace_TimeToFail and pg_Dataspace_WarmupTime properties, then the health status is marked as RED. In that case, the number of detected nodes and lives nodes differs in the response payload.

HTTP response appears as follows:

```
{
  "detectedNodes": [
    {
      "nodeName": "a04609a0-ca13-44db-98e1-f988ba18fbb4",
      "host": "us.myhost.com",
      "port": 4440
    },
    {
      "nodeName": "4820681b-f2fd-42d7-bccd-cf580ea8bf1c",
      "host": "in.myhost.com",
      "port": 4440
    }
  ],
  "liveNodes": "[a04609a0-ca13-44db-98e1-f988ba18fbb4]"
  "failingNodes": "[4820681b-f2fd-42d7-bccd-cf580ea8bf1c]"
  "health": "RED"
}
```

The response status code displays as 500 and health status displays as RED.

3. Invoke the REST API to monitor the gossip data.

Monitor the gossip vector element value in the nodes, when the health status is in YELLOW using the REST API **GET /rest/apigateway/dataspace/status?watchFailingNodes=true &fetchGossipData=true** to ensure if the *Gossiping protocol* works well. For example:

Request: GET

<http://uk.myhost.com:5555/rest/apigateway/dataspace/status?watchFailingNodes=true&fetchGossipData=true>.

HTTP response appears as follows:

```
{
  "detectedNodes": [
    {
      "nodeName": "a04609a0-ca13-44db-98e1-f988ba18fbb4",
      "host": "us.myhost.com",
      "port": 4440
    },
    {
      "nodeName": "4820681b-f2fd-42d7-bccd-cf580ea8bf1c",
      "host": "in.myhost.com",
      "port": 4440
    }
  ],
  "liveNodes": "[a04609a0-ca13-44db-98e1-f988ba18fbb4]"
  "failingNodes": "[4820681b-f2fd-42d7-bccd-cf580ea8bf1c]"
  "health": "RED"
}
```

```
    "liveNodes":  
    "[a04609a0-ca13-44db-98e1-f988ba18fbb4],[4820681b-f2fd-42d7-bccd-cf580ea8bf1c]"  
    "failingNodes":"[4820681b-f2fd-42d7-bccd-cf580ea8bf1c]"  
    "gossipData": "[a04609a0-ca13-44db-98e1-f988ba18fbb4,"  
121],[4820681b-f2fd-42d7-bccd-cf580ea8bf1c, 78]"  
    "health": "YELLOW"  
}
```

The DC 3 is detected as failing node and the gossip vector element in the DC 3 does not get incremented, the response status code displays as 200 and health status displays as YELLOW. API Gateway waits until the time specified in the pg_Dataspace_TimeToFail property, once the time elapses, then DC 3 is removed from the ring and marked as down.

How Do I Bring Down a Single Data Center from Active-Active or Hot Standby Mode to Standalone Mode?

This use case explains how to bring down a single data center from active-active or hot standby mode to standalone mode. You must bring down a data center to standalone mode in the following scenarios:

- When a data center is scheduled for maintenance.
- When you want to shut down a data center to relocate it permanently from one location to another.

By default, all the data centers are in standalone mode until you activate any other modes.

➤ To bring down a data center to standalone mode

1. Invoke the REST API.

You can bring down a single data center using the REST API

PUT/rest/apigateway/dataspace/activate on the data center that you want to bring down to standalone mode. For example:

Request: PUT <http://uk.myhost.com:5555/rest/apigateway/dataspace/activate>.

Sample payload:

```
{  
  "mode": "STANDALONE"  
}
```

HTTP response appears as follows:

```
{  
  "mode": "STANDALONE",  
}
```

When the data center is activated to standalone mode, the response status code displays as 200 and you can see the corresponding log entry in the **Server Logs**.

Note:

If you want to revert a data center that you have brought down, you have to update the configuration accordingly. For example, if you have brought down DC 1 (from active-active or hot standby to standalone mode) for maintenance activity, you can revert DC 1 to active-active or hot standby mode by updating the configuration with the details of DC 1.

You can validate whether the data center is brought down to standalone mode by reading the current configuration of the data center using the **GET/rest/apigateway/dataspace** REST API. For more information, see “[How Do I Read the Current Configuration of the Data Center?](#)” on page 753.

How Do I Bring Down Multiple Data Centers from Active-Active or Hot Standby Mode to Standalone Mode?

This use case explains how to bring down multiple data centers from active-active or hot standby mode to standalone mode. You must bring down multiple data centers to standalone mode in the following scenarios:

- When multiple data centers are scheduled for maintenance.
- When you want to shut down multiple data centers to relocate them permanently from one location to another.

By default, all the data centers are in standalone mode until you activate any other modes.

➤ To bring down multiple data centers to standalone mode

1. Invoke the REST API.

You can bring down multiple data centers using the REST API

PUT/rest/apigateway/dataspace/activateAll?mode=STANDALONE on any one of the data centers that you want to bring down. For example:

Request: PUT

`http://uk.myhost.com:5555/rest/apigateway/dataspace/activateAll?mode=STANDALONE.`

Consider DC 1 (uk.myhost.com), DC 2 (us.myhost.com), and DC 3 (in.myhost.com) are in active-active mode, and if you want to bring down DC 1 and DC 2, here is the sample payload:

```
{
  "local": {
    "host": "uk.myhost.com",
    "port": 5555,
    "syncPort": 4440,
    "keyStoreAlias": "UK_Key",
    "keyAlias": "Key_Alias_UK",
    "trustStoreAlias": "Trustpackage",
    "insecureTrustManager": true
  },
  "remotes": [
    {
      "host": "us.myhost.com",
      "port": 5555,
      "syncPort": 4440
    }
  ]
}
```

```
"port": 5555,
"syncPort": 4440,
"userName": "Administrator",
"password": "manage",
"keyStoreAlias": "US_Key",
"keyAlias": "Key_Alias_US",
"trustStoreAlias": "Trustpackage",
"insecureTrustManager": true
}
]
}
```

HTTP response appears as follows:

```
{
  "mode": "STANDALONE",
  "local": {
    "host": "uk.myhost.com",
    "port": 5555,
    "syncPort": 4440,
    "keyStoreAlias": "UK_Key",
    "keyAlias": "Key_Alias_UK",
    "trustStoreAlias": "Trustpackage",
    "insecureTrustManager": true
  },
  "remotes": [
    {
      "host": "us.myhost.com",
      "port": 5555,
      "syncPort": 4440,
      "userName": "Administrator",
      "password": "manage",
      "keyStoreAlias": "US_Key",
      "keyAlias": "Key_Alias_US",
      "trustStoreAlias": "Trustpackage",
      "insecureTrustManager": true
    }
  ],
  "acknowledged": true
}
```

When the data centers are activated to standalone mode, the response status code displays as 200 and you can see the corresponding log entry in the **Server Logs**.

Note:

If you want to revert the data centers that you have brought down, you have to update the configuration accordingly. For example, if you have brought down the data centers (DC 1 and DC 2) from active-active mode, you can revert the data centers to active-active mode by updating the configuration with the details of DC 1 and DC 2.

You can validate whether the data center is brought down to standalone mode by reading the current configuration of the data center using the **GET/rest/apigateway/dataspace** REST API. For more information, see “[How Do I Read the Current Configuration of the Data Center?](#)” on page 753.

How Do I Read the Current Configuration of the Data Center?

This use case explains how to read the current configuration of the data center using a REST API. You can validate your configuration by reading the current configuration of the data center.

➤ To read the current configuration of the data center

1. Read the current configuration of the data center using the **GET/rest/apigateway/dataspace** REST API.

Request: GET `http://uk.myhost.com:5555/rest/apigateway/dataspace`.

HTTP response appears as follows:

```
{
  "listener": {
    "listener": {
      "nodeName": "ecb1308f-22ac-4877-aba9-471a31a834e6",
      "host": "uk.myhost.com",
      "port": 4440
    },
    "insecureTrustManager": false
  },
  "listener.active": [
    "listener": {
      "nodeName": "ecb1308f-22ac-4877-aba9-471a31a834e6",
      "host": "uk.myhost.com",
      "port": 4440
    },
    "insecureTrustManager": false
  ],
  "ring": [
    {
      "nodeName": "a04609a0-ca13-44db-98e1-f988ba18fbb4",
      "host": "us.myhost.com",
      "port": 4440
    },
    {
      "nodeName": "4820681b-f2fd-42d7-bccd-cf580ea8bf1c",
      "host": "in.myhost.com",
      "port": 4440
    }
  ],
  "ring.active": [
    {
      "nodeName": "a04609a0-ca13-44db-98e1-f988ba18fbb4",
      "host": "us.myhost.com",
      "port": 4440
    },
    {
      "nodeName": "4820681b-f2fd-42d7-bccd-cf580ea8bf1c",
      "host": "in.myhost.com",
      "port": 4440
    }
  ],
  "mode": "ACTIVE_RING"
}
```

```
}
```

On successful configuration, the response status code displays as *200* and the first entry in the response displays `mode` field with the current configuration mode of the data center.

Cross-DC Extended Settings

The following table lists the extended settings that help you to specify the Cross-DC support:

Extended Setting	Description
pg_Dataspace_GossipInterval	Specifies how frequently each node should gossip with one another. By default, the value is set to 3 seconds.
pg_Dataspace_TimeToFail	Specifies the maximum permissible interval between two consecutive gossips. By default, the value is set to 30 seconds.
pg_Dataspace_WarmupTime	Specifies the maximum permissible rehashing interval from start-up or shut down of the server. By default, the value is set to 300 seconds.

Make sure you configure the extended settings in each of the API Gateway instances that are installed across the data centers. For information about configuring extended settings, see “Configuring Extended Settings” on page 324.

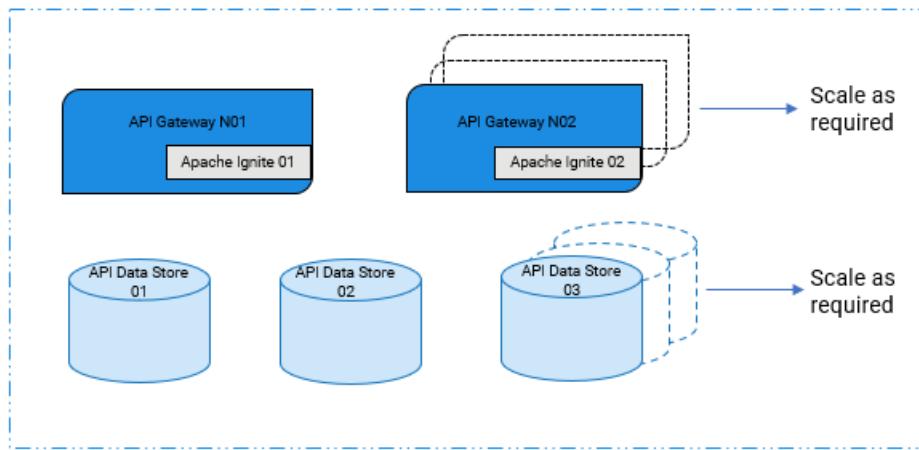
6 Performance Tuning and Scaling

■ Hardware and Product Configurations	756
■ Changing the JVM Heap Size to Tune API Gateway Performance	770
■ Data Separation	771
■ Scaling	779

Hardware and Product Configurations

Installing Software AG Products On Premises document provides the minimum system requirements to run API Gateway. These configurations change based on the production needs.

This section provides the hardware and product configuration guidelines that are required to setup API Gateway to run at an optimal scale. The hardware and product configuration guidelines are proposed for the following deployment architecture.



As an alternative for Apache Ignite, Terracotta Server Array can be used to set up a cluster. For more information about the cluster deployment, see “[Cluster Deployment](#)” on page 24. This is one of the architectures that is driven by availability and throughput factors. For information about the other variants of deployment architectures that is influenced by security, see “[Paired Deployment](#)” on page 36.

Typically, the parameters that influence the deployment architecture and the configurations are high availability, transactions per second (TPS), data volume, security requirements and so on. The hardware and product configurations that are recommended in this chapter are tested for the following throughput values. Consider the recommended hardware and product configurations as an outcome of a case study and not an official bench marking guide.

Parameters	Value
Transactions Per Second (TPS)	Up to 2000 Transactions Per Second.
Data volume	Up to 500 GB Storage utilization. Purge the data if the storage utilization is above 500 GB
Native service Latency	< 500ms

It is important to have the right sizing for the following components of API Gateway to meet the desired throughput requirements.

- API Gateway server

- API Data Store (Elasticsearch)
- Kibana
- Terracotta

As Apache Ignite component is embedded into API Gateway, it does not require any dedicated sizing or configuration.

Apart from the sizing and configurations, to ensure high availability and optimal performance of API Gateway, it is also important to employ good data housekeeping, monitoring and other operational best practices. For details about Data housekeeping and Monitoring, see “[Data Management](#)” on page 98, “[Monitoring API Gateway](#)” on page 176. Additionally, it is important to consider the scaling options when there is an additional load on the system. Scaling is primarily influenced by two factors, load or TPS, and data volume. Hence, the components that are impacted for scaling are API Gateway and API Data Store respectively. For details about scaling, see “[Scaling](#)” on page 779.

Note:

These recommendations should be considered as a guideline for the specified architecture to meet the specified throughput values. You can modify the configurations according to your business requirements. These recommendations apply to both Ignite based API Gateway cluster and Terracotta Server Array based API Gateway cluster. The sizing guidelines are specific to the components of API Gateway and does not include the resource allocations for the operating system and the other tools that you require to co-host while running API Gateway.

Resource Sizing Guidelines

This section provides recommendations on sizing of different components of API Gateway to meet the desired throughput requirements mentioned in the table for a cluster and standalone setup.

System Resource Allocation

Cluster setup

Component	RAM	CPU	HDD	Heap
API Gateway	Maximum: 6 GB Minimum: 4 GB	2 cores	30 GB	Minimum Heap size: 2048 MB Maximum Heap size: 3584 MB Parameters to define the Heap size: Minimum – Xms Maximum – Xmx
Terracotta	Maximum: 8 GB Minimum: 8 GB	2 cores	30 GB	Heap usage: 2 GB Off Heap usage: 2 GB

Component	RAM	CPU	HDD	Heap
Elasticsearch	Maximum: 6 GB Minimum: 4 GB	2 cores	300 GB	Minimum Heap size: 3584 MB Maximum Heap size: 3584 MB Parameters to define the Heap size: Minimum - Xms Maximum - Xmx
Kibana	Maximum: 6 GB Minimum: 4 GB	2 cores	30 GB	Maximum Heap size: 4096 MB Parameter to define the Maximum Heap size: --max-old-space-size=4096 MB

Standalone setup

Component	RAM	CPU	HDD	Heap
API Gateway	Maximum: 6 GB Minimum: 4 GB	2 cores	30 GB	Minimum Heap size: 2048 MB Maximum Heap size: 3584 MB Parameters to define the Heap size: Minimum - Xms Maximum - Xmx
Elasticsearch	Maximum: 6 GB Minimum: 4 GB	2 cores	300 GB	Minimum Heap size: 3584 MB Maximum Heap size: 3584 MB Parameters to define the Heap size: Minimum - Xms Maximum - Xmx
Kibana	Maximum: 6 GB Minimum: 4 GB	2 cores	30 GB	Maximum Heap size: 4096 MB Parameter to define the Maximum Heap size: --max-old-space-size=4096 MB

Logging Configurations

To troubleshoot any operational issues efficiently, it is crucial to manage the log files. It enables you to track and analyze the activity, usage, problems, and security like, user access and critical configuration changes. It helps you to identify unexpected anomalies in logs. Additionally, it is also important to add the configurations related to log rotation and retention settings.

This section provides recommendations on configuring the log levels of every component of API Gateway to enable automatic log rotation. By default, the log files are stored in the following locations:

API Gateway logs:

- `SAG_Install_Directory\IntegrationServer\instances\instance_name\logs.`
- `SAG_Install_Directory\profiles\IS_instance_name\logs.`

API Data Store logs:

- `SAG_Install_Directory\InternalDataStore\logs.`

Terracotta logs:

- Client log: `SAG_Install_Directory\IntegrationServer\instances\instance_name\logs.`
- Server log: `SAG_Install_Directory\tsa.`

Kibana logs:

- `SAG_Install_Directory\profiles\IS_instance_name\apigateway\dashboard\config` or
`Kibana_Install_Directory\config.`

Note:

By default, the log files are in *INFO* mode.

Log File Rotation Settings

API Gateway

Software AG recommends the following logging guidelines for API Gateway server to enable automatic log rotation. You must have the API Gateway's manage user administration functional privilege assigned to configure the watt parameters in API Gateway UI for server log and audit log. You can configure the watt parameters in the Watt keys section under **API Gateway UI > Administration > General > Extended settings > Show and hide keys** by providing the recommended values.

Server.log

Log level of the API Gateway server.

Server log file server.cnf is located at `SAG_Install_Dir\IntegrationServer\instances\instance_name\config.`

To configure the total size of the logs as **1 GB**, set the following values to the corresponding properties in the Watt keys section under **API Gateway UI > Administration > General > Extended settings > Show and hide keys** .

```
watt.server.serverlogFilesToKeep=100  
watt.server.serverlogRotateSize=10MB
```

Audit.log

Software AG logs the audit information for different categories of system transactions and events.

Audit log file server.cnf is located at *SAG_Install_Dir\IntegrationServer\instances\instance_name\config*.

To configure the total size of the logs as **1 GB**, set the following values to the corresponding properties in the Watt keys section under **API Gateway UI > Administration > General > Extended settings > Show and hide keys** .

```
watt.server.audit.logFilesToKeep=100  
watt.server.audit.logRotateSize=10MB
```

Osgi.log

Log level of the Osgi file type.

Osgi log file log4j2.properties is located at *SAG_Install_Dir\profiles\IS_instance_name\configuration\logging*.

To configure the total size of the logs as **300 MB**, set the following values in log4j2.properties log file and save the file.

```
appender.rolling.policies.size=10MB  
appender.rolling.strategy.max=30
```

Wrapper.log

Log level of the Wrapper file type.

Wrapper log file custom_wrapper.conf is located at *SAG_Install_Dir\profiles\IS_instance_name\configuration*.

To configure the total size of the logs as **300 MB**, add the following properties and its values as suggested in custom_wrapper.conf log file and save it.

```
wrapper.logfile.maxfiles=30  
wrapper.logfile.maxsize=10MB
```

Terracotta

Software AG recommends the following logging guidelines for Terracotta server and client to enable automatic log rotation.

Server log

Log level of the Terracotta server.

Logs at the server side server-logs are located at *SAG_Install_Dir\tsa*.

To configure the total size of the logs as **10 GB**, set the following value in server-logs file and save the file.

```
<property name="reconnect.maxLogFileSize" value="512"/>
```

Client log

Log level information at client side about the client and server interaction.

Logs at the client side are located at `SAG_Install_Dir\IntegrationServer\instances\instance_name\logs`.

To configure the total size of the logs as **10 GB**, set the following value in the client log file:

```
<property name="logging.maxBackups" value="20"/>
```

API Data Store (Elasticsearch)

Software AG recommends the following logging guidelines for Elasticsearch to enable automatic log rotation. For more information about Elasticsearch, see [Elasticsearch documentation](#).

elasticsearch.log

Log level of Elasticsearch.

Elasticsearch log file `log4j2.properties` is located at `SAGInstallDirectory\InternalDataStore\config`.

Software AG recommends you to set the following properties for the rolling file on `log4j2.properties`.

```
#Condition and Action to apply when handling roll overs
appender.rolling.strategy.action.condition.nested_condition.type = IfAny
#Perform the actions only if you have accumulated too many logs
appender.rolling.strategy.action.condition.nested_condition.type =
IfAccumulatedFileSize
# The size condition on the compressed logs is 512 MB
appender.rolling.strategy.action.condition.nested_condition.exceeds = 512MB
# A nested condition to apply to files matching the glob
appender.rolling.strategy.action.condition.nested_condition.lastMod.type =
IfLastModified
# Retains logs for seven days
appender.rolling.strategy.action.condition.nested_condition.lastMod.age = 7D
```

The properties for the old style pattern appenders is as follows. If the `log4j2.properties` in your system uses the old style layout of appenders, set the configurations for the following properties. Note that these should be considered as deprecated and can be removed in the future.

```
appender.rolling_old.strategy.action.condition.nested_condition.type = IfAny
appender.rolling_old.strategy.action.condition.nested_condition.type =
IfAccumulatedFileSize
appender.rolling_old.strategy.action.condition.nested_condition.exceeds = 512MB
appender.rolling_old.strategy.action.condition.nested_condition.lastMod.type =
IfLastModified
appender.rolling_old.strategy.action.condition.nested_condition.lastMod.age = 7D
```

If the file size is **512MB** or last modified date is **7D**, log rotation is enabled.

A sample configuration is as follows:

```
status = error
# log action execution errors for easier debugging
logger.action.name = org.elasticsearch.action
logger.action.level = debug
appender.rolling.type = Console
appender.rolling.name = rolling
appender.rolling.layout.type = ESJsonLayout
```

```

appender.rolling.layout.type_name = server
appender.rolling.strategy.action.condition.nested_condition.type = IfAny
appender.rolling.strategy.action.condition.nested_condition.type =
IfAccumulatedFileSize
appender.rolling.strategy.action.condition.nested_condition.exceeds = 512MB
appender.rolling.strategy.action.condition.nested_condition.lastMod.type =
IfLastModified
appender.rolling.strategy.action.condition.nested_condition.lastMod.age = 7D
appender.rolling_old.strategy.action.condition.nested_condition.type = IfAny
appender.rolling_old.strategy.action.condition.nested_condition.type =
IfAccumulatedFileSize
appender.rolling_old.strategy.action.condition.nested_condition.exceeds = 512MB
appender.rolling_old.strategy.action.condition.nested_condition.lastMod.type =
IfLastModified
appender.rolling_old.strategy.action.condition.nested_condition.lastMod.age = 7D
rootLogger.level = info
rootLogger.appenderRef.rolling.ref = rolling
appender.deprecation_rolling.type = Console
appender.deprecation_rolling.name = deprecation_rolling
appender.deprecation_rolling.layout.type = ESJsonLayout
appender.deprecation_rolling.layout.type_name = deprecation
logger.deprecation.name = org.elasticsearch.deprecation
logger.deprecation.level = warn
logger.deprecation.appenderRef.deprecation_rolling.ref = deprecation_rolling
logger.deprecation.additivity = false
appender.index_search_slowlog_rolling.type = Console
appender.index_search_slowlog_rolling.name = index_search_slowlog_rolling
appender.index_search_slowlog_rolling.layout.type = ESJsonLayout
appender.index_search_slowlog_rolling.layout.type_name = index_search_slowlog
logger.index_search_slowlog_rolling.name = index.search.slowlog
logger.index_search_slowlog_rolling.level = trace
logger.index_search_slowlog_rolling.appenderRef.index_search_slowlog_rolling.ref =
index_search_slowlog_rolling
logger.index_search_slowlog_rolling.additivity = false
appender.index_indexing_slowlog_rolling.type = Console
appender.index_indexing_slowlog_rolling.name = index_indexing_slowlog_rolling
appender.index_indexing_slowlog_rolling.layout.type = ESJsonLayout
appender.index_indexing_slowlog_rolling.layout.type_name = index_indexing_slowlog
logger.index_indexing_slowlog.name = index.indexing.slowlog.index
logger.index_indexing_slowlog.level = trace
logger.index_indexing_slowlog.appenderRef.index_indexing_slowlog_rolling.ref =
index_indexing_slowlog_rolling
logger.index_indexing_slowlog.additivity = false

```

Note:

Log4j's configuration parsing does not recognize extraneous whitespaces. Ensure to trim any leading and trailing whitespace when you copy the configurations.

Kibana

Software AG recommends the following logging guidelines for Kibana to enable automatic log rotation. For more information about Kibana, see <https://www.elastic.co/guide/en/kibana/current/introduction.html>.

kibana.log

Log level of Kibana.

Kibana log file kibana.yml is located at `SAG_Install_Dir\profiles\IS_instance_name\apigateway\dashboard\config` or `Kibana_InstallDirectory\config`.

Provide values for the following properties based on your logging requirements in kibana.yml log file and save it.

```
# ===== System: Logging =====
#Set the value of this setting to off to suppress all logging output, or to debug
# to log everything. Defaults to 'info'
# =====
logging.root.level: debug
# Enables you to specify a file where Kibana stores log output.
# =====
logging.appenders.default:
  type: file
  fileName: /var/logs/kibana.log
  layout:
    type: json
```

For example, to configure the total size of the logs as **300 MB**, provide **300mb** in the property `logging.appenders[].<appender-name>.policy.size`:

```
logging:
  appenders:
    json-file-appender:
      type: rolling-file
      fileName: ../logs/startup.log
      policy:
        type: size-limit
        size: 10mb
      strategy:
        type: numeric
        pattern: '-%i'
        max: 30
      layout:
        type: json
  root:
    appenders: [json-file-appender]
    level: info
```

Product Configurations Guidelines

This section provides Software AG guidelines for configuring the following components of API Gateway: API Gateway server, API Data Store (Elasticsearch), Kibana, and Terracotta. These recommendations should be considered as a guideline for setting the configurations to meet the throughput values specified in the table. You can modify the configurations according to your business requirements.

API Gateway Configurations

You must have the API Gateway's manage user administration functional privilege assigned to configure the watt parameters in API Gateway UI. You can configure the watt parameters in the Watt keys section under **API Gateway UI > Administration > General > Extended settings > Show and hide keys** by providing the recommended values. For more information about the extended settings, see “[Configuring Extended Settings](#)” on page 324.

Following is the list of WATT properties that you can alter by changing the default value with the recommended value that Software AG suggests for an optimal performance of API Gateway:

watt.server.threadPool

Specifies the maximum number of threads that the server maintains in the thread pool that it uses to run services. If this maximum number is reached, the server waits until services complete and return threads to the pool before running more services.

Recommended value: 600

watt.server.threadPoolMin

Specifies the minimum number of threads that the server maintains in the thread pool that it uses to run services. When the server starts, the thread pool initially contains this minimum number of threads. The server adds threads to the pool as needed until it reaches the maximum allowed, which is specified by the watt.server.threadPool setting.

Recommended value: 200

watt.server.control.serverThreadThreshold

Specifies the threshold at which API Gateway starts to warn of insufficient available threads. When the percentage of available server threads goes below the value of this property, API Gateway generates a journal log message indicating the current available thread percentage stating "Available Thread Warning Threshold Exceeded." When you receive this message in the journal log, you can adjust the thread usage to make server threads available.

Recommended value: 20%

watt.server.clientTimeout

Specifies the amount of time in minutes after which an idle user session times out.

Recommended value: 75

watt.server.serverlogFilesToKeep

Specifies the number of server log files that API Gateway keeps on the file system, including the current server log file. When API Gateway reaches the limit for the number of server log files, API Gateway deletes the oldest archived server log file each time API Gateway rotates the server log. If you set watt.server.log.filesToKeep to 1, API Gateway keeps the current server.log file and no previous server.log files. When API Gateway rotates the server.log, API Gateway does not create an archive file for the previous server log. If you set watt.server.log.filesToKeep to 0, or any value less than 1, API Gateway keeps an unlimited number of server log files.

Recommended value: 100

Important:

If you change the setting of this parameter, you must restart API Gateway for the changes to take effect.

watt.server.serverlogRotateSize

Specifies the file size at which API Gateway rolls over the server.log file. Set this property to N[KB|MB|GB], where N is any valid integer. The minimum size at which API Gateway rotates the server.log file is 33KB. If you use KB as the unit of measure, you must set N to a value greater than or equal to 33. If you do not specify a unit of measure, API Gateway treats the supplied N value as bytes. In this case, N must be greater than or equal to 32768 to take effect. Do not include any spaces between the integer and the unit of measure.

Recommended value: 10 MB

watt.server.audit.logFilesToKeep

Specifies the number of audit log files, including the current log file for the audit logger, that API Gateway keeps on the file system for an audit logger that writes to a file. When API Gateway reaches the limit for the number of log files for the audit logger, each time API Gateway rotates the audit log, API Gateway deletes the oldest archived audit log file. If you set watt.server.audit.log.filesToKeep to 1, API Gateway keeps the current audit log file and no previous audit log files for each file-system based audit logger. That is, when API Gateway rotates the audit log for a logger, API Gateway does not create an archive file for the previous audit log. If you set watt.server.audit.logFilesToKeep to 0, or any value less than 1, API Gateway keeps an unlimited number of audit log files.

Recommended value: 100

The watt.server.audit.logFilesToKeep parameter affects only the audit loggers configured to write to a file. The parameter does not affect audit loggers configured to write to a database nor does it affect the FailedAuditLog.

If you reduce the number of logs that API Gateway keeps for file-based audit logs and then restart API Gateway, the existing audit logs will not be pruned until API Gateway writes to the audit log. For example, if the error logger writes to a file and you reduce the number of log files to keep from 10 to 6, API Gateway does not delete the 4 oldest error audit log files immediately after start up. API Gateway deletes the 4 oldest error audit logs after the error logger writes to the error audit log.

Important:

If you change the setting of this parameter, you must restart API Gateway for the changes to take effect.

watt.server.audit.logRotateSize

Specifies the file size at which API Gateway rolls over the audit log for a logger that writes to a file. Set this property to N[KB|MB|GB], where N is any valid integer. The minimum size at which API Gateway rotates an audit log is 33KB. If you use KB as the unit of measure, you must set N to a value greater than or equal to 33. If you do not specify a unit of measure, API Gateway treats the supplied N value as bytes. In this case, N must be greater than or equal to 32768 to take effect. Do not include any spaces between the integer and the unit of measure.

Recommended value: 10MB

The watt.server.audit.logRotateSize parameter affects only the audit loggers configured to write to a file. The parameter does not affect audit loggers configured to write to a database.

Important:

If you change the setting of this parameter, you must restart API Gateway for the changes to take effect.

watt.net.maxClientKeepaliveConns

Sets the default number of client keep alive connections to retain for a given target endpoint.

The default is 0, which indicates that API Gateway does not retain client keep aliveconnections for a target endpoint. API Gateway creates a new socket for each request.

Recommended value: 500

This benefits in situations where the frequency and number of concurrent requests to a given target endpoint are high. In situations where this is not the case, idle sockets will become stale and inoperable, resulting in unexpected exceptions such as the following:

```
[ISC.0077.9998E] Exception --> org.apache.axis2.AxisFault: Broken pipe
```

watt.server.revInvoke.proxyMapUserCerts

Specifies whether an API Gateway server is to perform client authentication itself in addition to passing authentication information to the Internal Server for processing.

Recommended value: true

If it is set to true, API Gateway rejects all anonymous requests (no certificate and no username or password supplied), even if the request is for an unprotected service on the Internal Server.

watt.security.ssl.cacheClientSessions

Controls whether API Gateway reuses previous SSL session information (for example, client certificates) for connections to the same client.

Recommended value: true

When this property is set to true, API Gateway caches and reuses SSL session information. For example, set this property to true when there are repeated HTTPS requests from the same client.

watt.security.ssl.client.ignoreEmptyAuthoritiesList

Specifies whether an API Gateway acting as a client sends its certificate chain after a remote SSL server returns an empty list of trusted authorities. When set to true, API Gateway disregards the empty trusted authorities list and sends its chain anyway. When set to false, before sending out its certificate chain, API Gateway requires the presentation of trusted authorities list that proves itself trusted.

Recommended value: true

watt.server.url.alias.partialMatching

Specifies whether API Gateway enables partial matching on URL aliases. If you set this server configuration parameter to true and define a URL alias in API Gateway Administrator, API Gateway enables partial matching on URL aliases.

Recommended value: true

When partial matching is enabled, API Gateway considers an alias a match if the entire alias matches all or part of the request URL, starting with the first character of the request URL path.

Important:

If you change the setting of this parameter, you must restart API Gateway for the changes to take effect.

watt.net.clientKeepaliveUsageLimit

Specifies the maximum number of usages for a socket in a client connection pool. Before returning a socket to the pool, API Gateway compares the number of times the socket has been used to send a request to the watt.net.clientKeepaliveUsageLimit value. If the socket usage count is greater than the watt.net.clientKeepaliveUsageLimit value, then Integration Server does not return the socket to the pool. Instead, API Gateway closes the socket. If a new socket is needed in the pool, API Gateway creates one.

Recommended value: 10000000 uses

Note:

Even if the number of connection usages is less than the `watt.net.clientKeepaliveUsageLimit` parameter, API Gateway closes the connection if the connection has exceeded the age limit set by the `watt.net.clientKeepaliveAgingLimit` server configuration parameter.

The `watt.net.clientKeepaliveUsageLimit` parameter applies only if `watt.net.maxClientKeepaliveConns` is set to a value greater than 0.

Important:

If you change the setting of this parameter, you must restart API Gateway for the changes to take effect.

watt.server.rg.internalsocket.timeout

Specifies the length of time, in milliseconds, that API Gateway server allows a client request to wait for a connection to the Internal Server before terminating the request with an HTTP 500 Internal Server Error. If a connection to the Internal Server becomes available within the specified timeout period, Enterprise Gateway Server forwards the request to the Internal Server. If a connection does not become available before the timeout elapses, API Gateway returns a HTTP 500-Internal Server Error to the requesting client and writes the following message to the error log: *Enterprise Gateway port {port_number} is unable to forward the request to Internal Server because there are no Internal Server connections available*. This is applicable for paired deployment with reverse invoke setup.

Recommended value: 300 ms

watt.server.enterprisegateway.ignoreXForwardedForHeader

Specifies whether API Gateway must ignore the X-Forwarded-For request header while processing the rules in API Gateway server. If this property is set to true, then API Gateway ignores the X-Forwarded-For request header and considers the proxy server's IP address as the host IP address. If the property is set to false, then API Gateway obtains the actual host IP address from the X-Forwarded-For request header. This is applicable for paired deployment with reverse invoke setup.

Recommended value: false

API Gateway Extended Settings

Software AG recommends the following configurations for the Extended settings. You can configure the extended settings under **API Gateway UI > Administration > General > Extended settings > Show and hide keys** by providing the recommended values. For more information about the extended settings, see “[Configuring Extended Settings](#)” on page 324.

portClusteringEnabled

By default, API Gateway provides synchronization of the port configuration across API Gateway cluster nodes. If you do not want the ports to be synchronized across API Gateway cluster nodes, set the `portClusteringEnabled` parameter to false.

Recommended value: false

For more details about Ports Configuration, see “[Ports Configuration](#)” on page 36.

eventsRefreshInterval

Specifies the frequency at which Elasticsearch should refresh its own indices. In Elasticsearch, an operation that updates the data, which is visible in search is called a refresh. Any document

that is modified or inserted appears in search operations only after the index is refreshed. Specifying a lesser value to this parameter overloads Elasticsearch with frequent indexing when there is a large volume of data. Henceforth, it is recommended to specify a higher value to this parameter.

Note:

In API Gateway, this property is only for the analytics indices and core data changes are refreshed every 1 second by default.

Recommended value: 10s

events.collectionPool.maxThreads

Specifies the maximum number of threads to be used for the event data collection pool.

Recommended value: 40

events.collectionPool.minThreads

Specifies the minimum number of threads to be used for the event data collection pool.

Recommended value: 2

events.collectionQueue.size

Specifies the size of the collection queue to be used during event data collection. When events like transaction, error events, and performance metrics are generated during API invocations, they are put in the collection queue for further processing. Each thread in the collection pool is assigned a task of collecting these events and processing them for sending to the desired destinations such as API Gateway, Elasticsearch, API Portal, and so on.

If the queue capacity is reached, then any additional event data would be lost. To avoid the loss of data, it is recommended to increase this size when there is an increase in incoming traffic. You should choose the value based on the payload size of the transactions as queue size and payload size determine the memory occupied by the queue.

events.reportingPool.maxThreads

Specifies the maximum number of threads to be used for the event data reporting pool.

Recommended value: 24

events.reportingPool.minThreads

Specifies the minimum number of threads to be used for the event data reporting pool.

Recommended value: 4

events.reportingQueue.size

Specifies the size of the reporting queue to be used during event data reporting. When events like transaction, error events, and performance metrics are generated during API invocations, they are put in the collection queue for further processing. Each thread in the collection pool is assigned a task of collecting these events, processing them and put in the reporting queue for sending to the desired destinations such as API Gateway, Elasticsearch, API Portal, and so on.

If the reporting queue capacity is reached, then any additional event data would be lost. To avoid the loss of data, it is recommended to increase this size when there is an increase in incoming traffic. You should choose the value based on the payload size of the transactions as queue size and payload size determine the memory occupied by the queue.

Terracotta Configurations

See “[Terracotta Server Array Configuration](#)” on page 30 and set the configurations accordingly.

API Data Store (Elasticsearch) Configurations

This section explains the API Data Store configurations. As part of API Data Store configurations, this section covers the connection properties:

Connection properties

This section explains the configurations that are required to make API Gateway connect to desired Elasticsearch cluster located at `SAGInstallDirectory\IntegrationServer\instances\instance_name\Packages\WmAPIGateway\config\resources\elasticsearch\config.properties`. You must change the configurations of the following properties and tune it as per the following guidelines. You can modify the configurations according to your business requirements.

```
pg.gateway.elasticsearch.hosts = Elasticsearch Service endpoint, that is localhost:9240
pg.gateway.elasticsearch.http.connectionTimeout = 10000
pg.gateway.elasticsearch.http.socketTimeout = 30000
pg.gateway.elasticsearch.sniff.enable = false
pg.gateway.elasticsearch.http.maxRetryTimeout = 100000
```

Note:

This can be done through externalized configurations also. A default template is located in `SAGInstallDirectory\IntegrationServer\instances\packages\WmAPIGateway\resources\configuration` folder. Ensure to add the desired settings in `system-settings.yml` file. After adding the desired settings, you have to enable the file `config-sources.yml` by uncommenting the appropriate lines in the file for API Gateway to know that this is the configuration file. For more information about externalized configurations, see “[Externalizing Configurations](#)” on page 62.

Kibana Configurations

This section explains the configuration changes for Kibana in `kibana.yml` file located at `SAGInstallDirectory\profiles\IS_instance_name\apigateway\dashboard\config\Kibana_InstallDirectory\config`. You must change the configurations of the following properties and tune it as per the following guidelines. You can modify the configurations according to your business requirements.

`elasticsearch.requestTimeout` property

This property specifies Kibana’s wait time to receive a response from Elastic Search, in seconds, for retries after which it times out.

Recommended value: 90 seconds

`elasticsearch.hosts`

Specifies the URLs of the Elasticsearch instance to use for all your queries.

[`http://hostname:port`]

For example,

```
elasticsearch.hosts: ["http://localhost:9240"]
```

telemetry.enabled

Disables the telemetry data being sent to Elasticsearch server.

Recommended value: `false`

For more details about Kibana configurations, see “[Connecting to an External Kibana](#)” on [page 60](#).

Changing the JVM Heap Size to Tune API Gateway Performance

The JVM heap or on-heap size indicates how much memory is allotted for server processes. At some point, you might want to increase the minimum and maximum heap size to ensure that the JVM that API Gateway uses does not run out of memory. In other words, for example, if you notice `OutOfMemoryError: Java heap space for Integration Server process`, then you have to increase the minimum and maximum heap size to overcome the out of memory error.

The heap size is controlled by the following Java properties specified in the `custom_wrapper.conf` file.

Property	Description
<code>wrapper.java.initmemory</code>	The minimum heap size. The default value is 256 MB.
<code>wrapper.java.maxmemory</code>	The maximum heap size. The default value is 3584 MB.

Your capacity planning and performance analysis should indicate whether you need to set higher maximum and minimum heap size values.

➤ To change the heap size

1. Open the `custom_wrapper.conf` file in a text editor.

You can find the `custom_wrapper.conf` file in the following location: `Software AG_directory\profiles\IS_instance_name\configuration\`.

2. Set the `wrapper.java.initmemory` and `wrapper.java.maxmemory` parameters so that they specify the minimum and maximum heap size required by API Gateway.

For example:

```
wrapper.java.initmemory=256  
wrapper.java.maxmemory=3584
```

3. Save and close the file.

4. Restart API Gateway.

If you notice an out of memory issue for Elasticsearch, then you have to tune the Elasticsearch performance. For example, if you notice OutOfMemoryError: Java heap space for API Data Store process (that is, Elasticsearch), then you have to increase the following minimum and maximum heap size to overcome the out of memory error. Open the `jvm.options` file located at `Software AG_directory\InternalDataStore\config` and set the following parameters to configure the heap size as 4GB:

```
-Xms4g
-Xmx4g
```

where, Xms represents the initial size of total heap and Xmx represents the maximum size of total heap space. You have to restart the API Data Store for the changes to take effect.

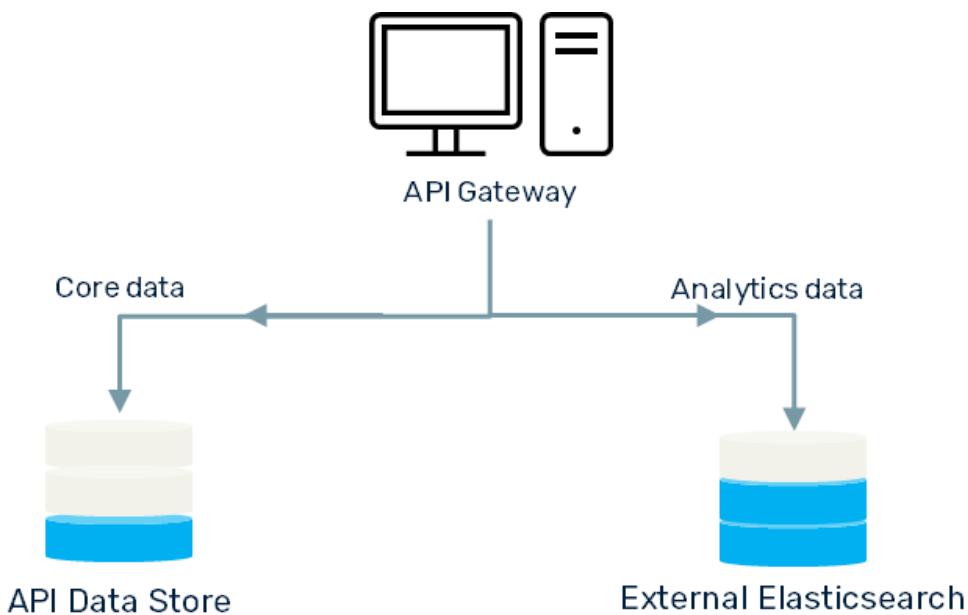
Data Separation

API Gateway stores the data in API Data Store by default. It includes the core data and analytics data. These two data types are different in the following ways:

Core data	Analytics data
Includes APIs and assets that are critical to keep business running.	Includes the API analytics data
Requires frequent backup. For example, 15 minutes.	Can be backed up in lesser frequent intervals. For example, 24 hours.
Requires lesser storage space than the analytics data.	Requires huge storage space.

Hence, if you are managing large transactions volume, Software AG recommends that you save the core data and analytics data separately.

To separate the core data from analytics data, you can configure an external Elasticsearch and save the analytics data as illustrated in the below fig.



When you save the analytics data in an external Elasticsearch instance, the data is not reflected in the API Gateway Dashboard. You can configure a separate Kibana dashboard for your external Elasticsearch destination.

Data Separation Benefits

The main benefit of data separation is that the API Gateway core data is lesser in size when compared to the analytics data. The data stored in API Data Store remains light and

- Performance of the application improves.
- Data housekeeping and data backup processes become faster and simpler.

Storing Analytics Data in external Elasticsearch

➤ To store analytics data in external Elasticsearch

1. Configure an external Elasticsearch. For information on configuring and connecting to an external Elasticsearch, see “[Connecting to an External Elasticsearch](#)” on page 56.
2. Apply the Log Invocation policy globally with Elasticsearch as destination.

For information on the Log Invocation policy, see the section Log Invocation in the *webMethods API Gateway User's Guide*.

The analytics data is saved to the configured external Elasticsearch destination.

Configuring Analytics Data Store using UI

You can configure analytics data store to store analytics data separately from the core data.

To configure analytics data store

1. Expand the menu options icon  in the title bar, and select **Administration**.
2. Select **Destinations**.
3. Select **API Gateway > Analytics** to configure analytics data store.

Provide the following information:

Field	Description
Protocol	Specifies the communication protocol used to establish communication between API Gateway and the analytics data store. The available protocol types are: <ul style="list-style-type: none"> ■ HTTP ■ HTTPS
Hostname	Specifies the host name or IP address of the machine on which analytics data store is running.
Port	Specifies the port which analytics data store uses to communicate with API Gateway.
Username	Specifies the analytics data store user ID for authenticating analytics data store when API Gateway communicates with it.
Password	Specifies the password of the analytics data store instance to be used for establishing communication between API Gateway and analytics data store.

4. Click **Test**.

This tests the communication channel between API Gateway and the configured analytics data store.

5. Click **Save**.

You can click **Cancel** to revert to the last saved changes or to abandon all the changes if the values are not saved.

Clicking **Save**

- updates the configurations in the core data store.
- updates the configurations in the analyticsds.properties file
 - in the current API Gateway node.

- in other API Gateway nodes in a cluster.
6. Restart API Gateway for the configuration to take effect.

Analytics data store is configured and a communication channel is established between API Gateway and analytics data store to exchange data.

Configuring Analytics Data Store manually

You can configure analytics data store to store analytics data separately from the core data. You can configure the analytics data store manually if you want API Gateway to automatically connect to analytics data store when you start API Gateway.

To configure analytics data store

1. Navigate to `SAGInstallDirectory\IntegrationServer\instances\instance_name\packages\WmAPIGateway\config\resources\elasticsearch\analyticssds.properties`.

The analyticssds.properties file contains all the properties and external Elasticsearch configurations.

2. Configure the following properties:

Property and Description

apigw.analytics.ds.hosts

Mandatory

This property lists analytics data store hosts and ports. The values are comma separated.

Default value: changeOnInstall

If you modify this value, for example, `localhost:port`, you must edit the value in all nodes and restart the API Gateway server for the configuration to take effect.

Note:

Once a host is added to this property, this is the value that is used to connect to the analytics data store and the host configured in `SAGInstallDirectory\IntegrationServer\instances\default\packages\WmAPIGateway\config\resources\beans\gateway-analytics-ds.xml` file is not considered.

apigw.analytics.ds.http.keepAlive

Mandatory

This property creates the persistent connection between client and server.

Default value: true

Property and Description

apigw.analytics.ds.http.connectionTimeout

Mandatory

This property specifies the time, in milliseconds, after which the connection times out.

Default value: 10000

apigw.analytics.ds.http.socketTimeout

Mandatory

This property specifies the wait time, in milliseconds, for a reply once the connection to analytics data store is established after which it times out.

Default value: 30000

apigw.analytics.ds.http.maxRetryTimeout

Mandatory

This property specifies the wait time, in milliseconds, for retries after which it times out.

Default value: 100000

Software AG recommends that you set the maximum retry time for a request to (number of nodes * socketTimeOut) + connectionTimeout.

apigw.analytics.ds.http.keepAlive.maxConnections

Mandatory

This property specifies the maximum number of persistent connections that can be established between an API Gateway and analytics data store cluster.

Default value: 50

apigw.analytics.ds.http.keepAlive.maxConnectionsPerRoute

Mandatory

This property specifies the maximum number of persistent connections that can be established per HTTP route to an analytics data store server.

Default value: 15

apigw.analytics.ds.http.username

This property specifies the user name to connect to analytics data store using basic authentication.

apigw.analytics.ds.http.password

Property and Description

This property specifies the password to connect to analytics data store using basic authentication.

apigw.analytics.ds.https.keystore.filepath

This property specifies the keystore file path for establishing HTTPS communication with analytics data store.

apigw.analytics.ds.https.truststore.filepath

This property specifies the truststore file path for establishing HTTPS communication with analytics data store.

apigw.analytics.ds.https.keystore.password

This property specifies the keystore password for establishing HTTPS communication with analytics data store.

apigw.analytics.ds.https.keystore.alias

This property specifies the keystore alias for establishing HTTPS communication with analytics data store.

apigw.analytics.ds.https.truststore.password

This property specifies the truststore password for establishing HTTPS communication with analytics data store.

apigw.analytics.ds.https.enabled

This property specifies whether you want to enable or disable the HTTPS communication with analytics data store.

Default value: false

If this property is set to false none of the above properties related to HTTPS are respected.

apigw.analytics.ds.outbound.proxy.enabled

This property specifies whether you want to enable or disable outbound proxy communication.

Default value: true

apigw.analytics.ds.outbound.proxy.alias

This property specifies the outbound proxy alias name used to connect to analytics data store.

apigw.analytics.ds.https.enforce.hostname.verification

This property enforces the host name verification for SSL communication.

Default value: false

Property and Description

apigw.analytics.ds.sniff.enable

Mandatory

This property enables sniffers to add the other nodes in an analytics data store cluster to the client so that the client can talk to all nodes.

Default value: true

This configuration must be set to *false* if you are changing the network when API Gateway or analytics data store is running.

apigw.analytics.ds.tenantId

This property allows you to specify a tenant name, which must be same as what is defined for API Data Store and across all nodes.

The default value of this property is the Integration Server instance name. So, ensure that you provide same name for all Integration Server nodes in a cluster.

If you modify this value, you must edit the value in all nodes and restart the API Gateway server for the change to take effect.

apigw.analytics.ds.sniff.timeInterval

Mandatory

This property enables adding the newly added analytics data store cluster nodes to existing REST client in a specified time interval in milliseconds.

Default value: 60000

apigw.analytics.ds.client.http.response.size

Mandatory

This property specifies the response size, in MB, for API Gateway analytics data store client.

Default value: 100

-
3. Start API Gateway for the HTTP client to take effect.

Connecting to Analytics Data Store

You need to establish a communication channel between API Gateway and analytics data store to exchange data. This section explains the steps to enable API Gateway to connect to analytics data store.

You can also connect to an analytics data store through externalized configurations. For more details, see “[Externalizing Configurations](#)” on page 62.

A sample analytics data store externalized configuration is as follows:

```
---
apigw:
  analyticsDataStore:
    tenantId: default
    hosts: localhost:9241
    http:
      username: analyticsdatastore
      password: changeme
      keepAlive: true
      keepAliveMaxConnections: 60
      keepAliveMaxConnectionsPerRoute: 20
      connectionTimeout: 2000
      socketTimeout: 40000
      maxRetryTimeout: 200000
    https:
      enabled: false
      keystoreFilepath: C:/softwares/analyticsdatastore/config/demouser-keystore.jks
      truststoreFilepath: C:/softwares/analyticsdatastore/config/truststore.jks
      keystoreAlias: cn=demouser
      keystorePassword: 6572b9b06156a0ff778c
      truststorePassword: 2c0820e69e7dd5356576
      enforceHostnameVerification: false
    sniff:
      enable: false
      timeInterval: 70000
    outboundProxy:
      enabled: false
      alias: somealias
    clientHttpResponseSize: 200
```

To connect API Gateway to analytics data store

1. Configure an external Elasticsearch as analytics data store. For information about configuring and connecting to an external Elasticsearch, see “[Connecting to an External Elasticsearch](#)” on page 56.
2. Configure an external Kibana connecting to analytics data store for rendering dashboards in API Gateway. For information about how to connect to external Kibana, see “[Connecting to an External Kibana](#)” on page 60.

Note:

Once analytics data store is used or connected on startup of API Gateway, the default internal Kibana is not used anymore. For rendering dashboards, you require external Kibana.

3. Configure the **apigw.analytics.ds.hosts** property in the analyticsds.properties file located at `SAGInstallDirectory\IntegrationServer\instances\instance_name\packages\WmAPIGateway\config\resources\elasticsearch`. For example, `apigw.analytics.ds.hosts=localhost:9241`.

Note:

When you reset the **apigw.analytics.ds.hosts** with an invalid host value, analytics data store is not used to store the analytics data. Set the **apigw.kibana.autostart** property to

true in the **uiconfiguration.properties** file located at *SAGInstallDirectory\profiles\IS_default\apigateway\config* to use internal Kibana again.

API Gateway connects to the analytics data store.

Start up sequence of components when using analytics data store for the first time:

1. Start external analytics data store.
2. Start API Gateway.
3. After API Gateway is started and available, start external Kibana.

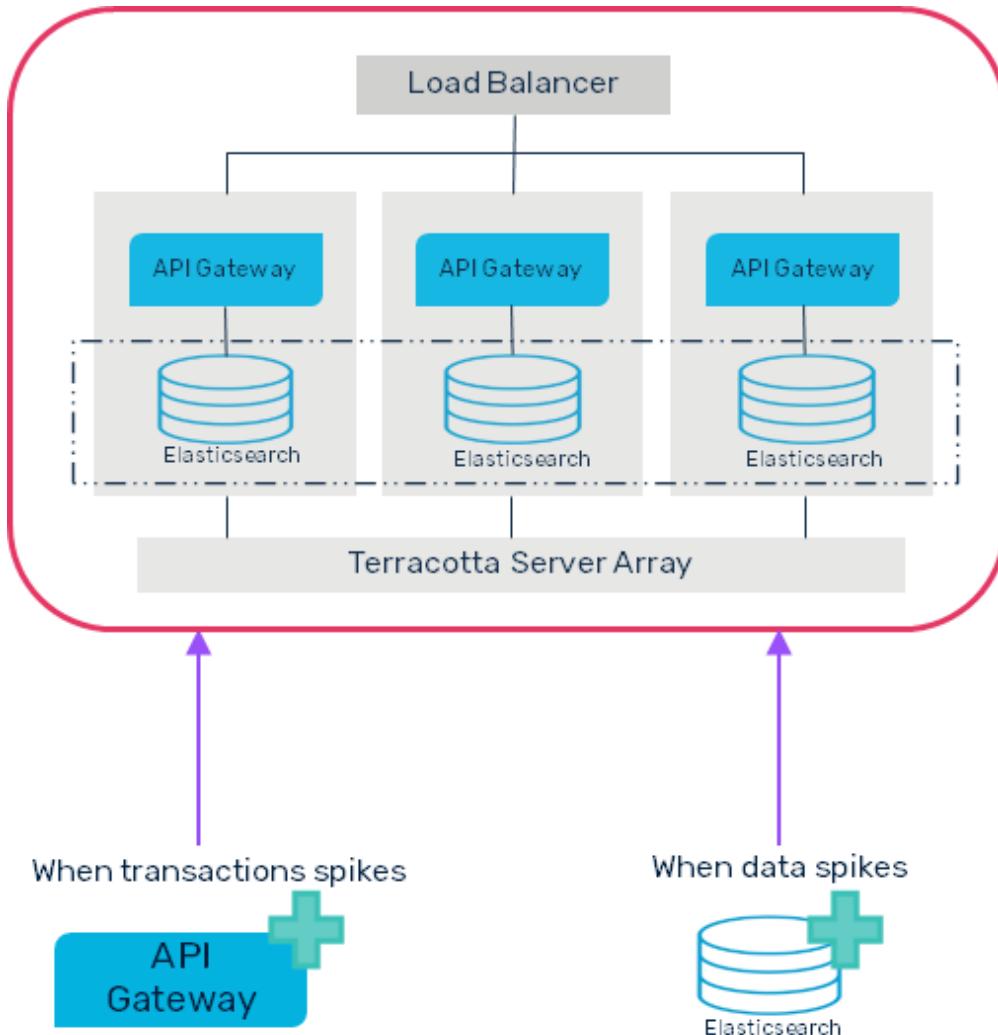
Scaling

As a critical step in your API Gateway deployment, you perform the capacity planning for API Gateway and its components that can live up to the estimated transactions (TPS) demands and data volume storage needs in compliance with your data and analytics retention SLAs. Though it is recommended to have the right sizing in place, it is important to consider scaling up or scaling down API Gateway to address the spikes that are not factored in the initial capacity planning.

You can do horizontal scale up or scale down of API Gateway to meet the spikes in the transactions or data.

You can scale up:

- **API Gateway.** When the number of transactions exceeds the estimated capacity. For example, if the existing deployment handles 200 transactions per second, you have to scale up API Gateway if the number exceeds this limit.
- **API Data Store (Elasticsearch).** When the volume of data to be stored exceeds the estimated size. For example, if the API Data Store capacity is 500 GB, and if the data to be stored exceeds the estimated size, you have to scale up the API Data Store.

**Note:**

You can minimize the need for the scaling of API Data Store with the right sizing (capacity planning), monitoring, and Data housekeeping procedures that are in compliance with your data and analytics retention SLAs.

Scaling requirements analysis

You can monitor the key metrics of system and application to decide whether they must be scaled up or not.

- For information on monitoring API Gateway, see “[Monitoring API Gateway](#)” on page 189.
- For information on monitoring API Data Store, see “[Monitoring API Data Store](#)” on page 303 or “[System Metrics](#)” on page 204.

Software AG recommends that you

- Set up Data housekeeping procedures.
- Separate the lifecycle of API Gateway and Data store in a clustered environment.

- Scale up only the required component based on the need (API Gateway for transactions and Elasticsearch for Data volume).

Scaling up API Gateway

You must add a new API Gateway node to scale up API Gateway and handle the spike.

To scale up API Gateway,

1. You can scale up API Gateway by simply adding a new API Gateway node to the existing cluster.

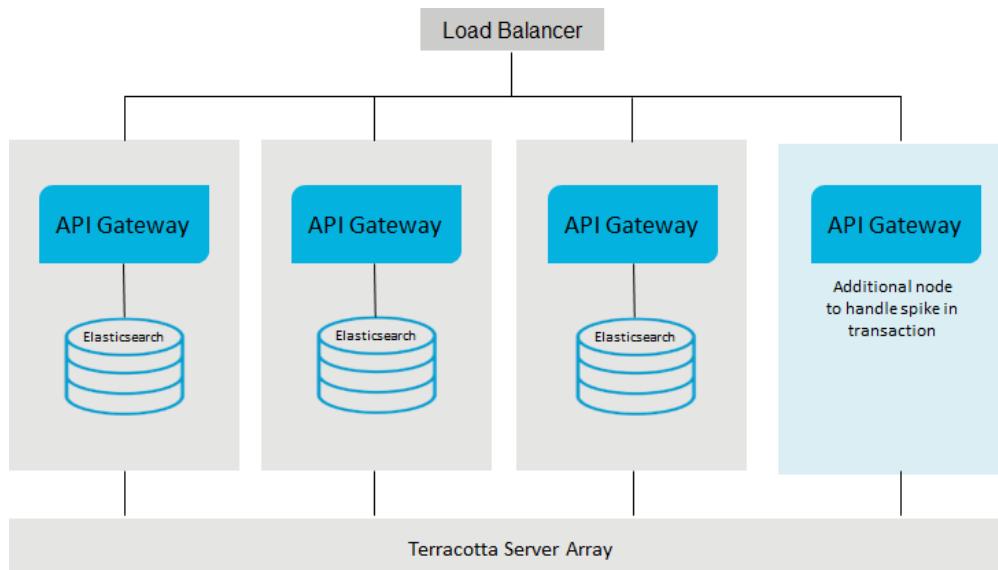
For information on adding nodes to a cluster, see “[API Gateway Cluster Configuration](#)” on [page 26](#).

2. Add the node to Load balancer for the traffic to be distributed across the nodes.

Note:

Ensure that the new API Gateway node has same configurations as other nodes in the cluster.

API Gateway is scaled up, and the transactions are now distributed among the nodes.



Scaling up API Gateway in a Reverse Invoke setup

The procedure is same as the one described above.

To scale up API Gateway in a Reverse Invoke setup, you can perform any of the following:

- Add a new node in the Green zone.

Note:

If the **portClusteringEnabled** setting is disabled, then you need to create internal ports in the new node to connect to the registration ports of all the API Gateway nodes in the DMZ.

- Add a new node in the DMZ zone. After bringing up the new node to the DMZ, ensure that you
 - Explicitly establish connections from all nodes in the green zone to the new node that you added to the DMZ zone. This can be done by adding Internal port in the servers in the green zone to connect to the newly created API Gateway in the DMZ zone.
 - Specify same configurations in the new node as other nodes in the DMZ zone.

You can use the **Port Configuration** REST API to automate the port settings and allow the Load balancer to add the new node.

Note:

API Gateway instances running in the DMZ do not need to be clustered.

Scaling up API Data Store

➤ To scale up API Data Store

1. Install API Data Store or Elasticsearch in a new node.
2. Configure the required heap size. For information on increasing heap size, see “[Changing the JVM Heap Size to Tune API Gateway Performance](#)” on page 770.
3. In the `Elasticsearch.yml` file of the new node, modify the following:
 - Set the value of the `node.roles` to `[data]` if the node is not a master node; and set it to `[master, data]`, if the node is a master node.
 - Specify the value of the `path.repo` variable same as the value in other nodes.
 - Specify the names of the nodes in the `discovery.seed.hosts` variable, in the following format:

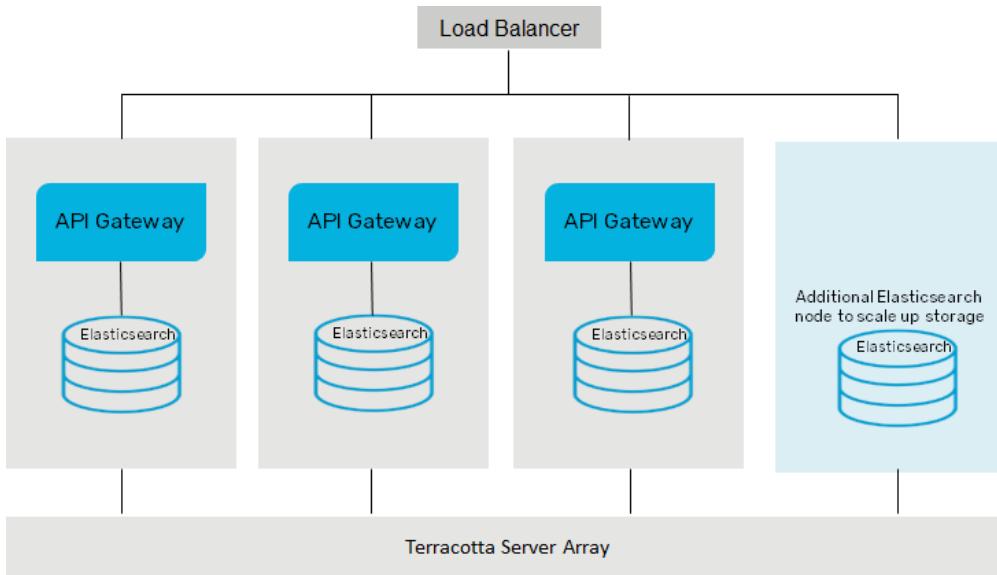
```
host_name:port
```

For example,

```
discovery.seed_hosts: ["node1:9340", "node2:9340", "node3:9340"]
```

The node names must be same as the list of nodes provided in the `cluster.initial_master_nodes` variable.

- Save the configurations.
- Start the API Data Store or Elasticsearch.



The API Data Store node is added and scaled up.

Scaling down API Gateway

➤ To scale down API Gateway

1. Remove a node from the cluster by performing the following steps:
 - a. Enable the *Quiesce* mode in the node that you want to remove. The node will stop accepting transactions requests, and the Load balancer routes the requests to other nodes.
 - b. Shutdown the node after a grace period to allow ongoing requests to complete. The period depends on the responsiveness of the native services and the timeout setting for outbound requests. The timeout for outbound requests is defined via **watt.net.timeout** configuration parameter
 - c. Remove the node from the Load balancer.

The node is removed.

Scaling down API Gateway in a Reverse Invoke setup - DMZ Zone

1. Remove a node from the DMZ zone by removing the node from the Load balancer.
The node is removed.
2. Shut down the node.

3. After the grace period for the inflight transactions to complete, remove the internal port from the nodes in green zone, that is configured to connect to the removed API Gateway node from the DMZ zone.

Scaling down API Gateway in a Reverse Invoke setup - Green Zone

1. Remove a node from the Green zone by performing the following steps:

- a. Disable the internal ports of the node.
- b. Shut down the instance in the node.

In-flight transactions would fail as the communication channel is closed.

The node is removed.

Scaling down API Data Store

➤ To scale down API Data Store

1. Remove a API Data Store node from the cluster by stopping the API Data Store.

If you are removing master node, ensure that the node is not specified in the **cluster.initial_master_nodes** property. Else, the cluster formation becomes impossible.

The API Data Store is scaled down.

7 Remove User Data from API Gateway

- Removing User Data 786

Removing User Data

Data protection laws and regulations, such as the General Data Protection Regulation (GDPR) might require specific handling of user data, even after a user profile is removed. Additionally, employees or other clients with user accounts on API Gateway may request that any user identifying information such as user name, email addresses, or client IP addresses be removed from API Gateway. To comply with data protection requirements and user requests, in addition to deleting the user account, you may need to complete activities such as deleting or masking the user data.

Note:

API Gateway can optionally capture the runtime transaction logs, which contain the API request and response data (customer-defined) that flow through to the API Gateway. Though there are options to purge or clean up these data, this section does not define procedure for the same as the customer-defined data is out of the scope of this functionality.

Types of Data and their stores in API Gateway

- **Core data**

This consists of APIs, policies, applications, aliases, packages, plans, administration configurations, users, and groups. This is stored in API Data Store and Integration Server store.

- **Runtime transactions**

This consists of the transaction events, monitoring events, error events, policy violation events, threat protection events, lifecycle events and performance metrics. This information can be stored in API Data Store or external destinations.

- **Application logs**

This consists of UI, server, Elasticsearch, Kibana, filebeat, and Platform logs . This is stored in filesystem and API Data Store.

- **Audit logs**

This is stored in API Data Store.

- **Tracer logs**

This consists of the runtime transactions that are captured when you trace the API. This information can be stored in API Data Store.

Handling Core Data

API Gateway strongly recommends the use of internal or technical user for policy configurations like Authorize user, Invoke IS Service and Outbound Authentication that has user credentials. This avoids the life cycle of actual or real user object from impacting the API Gateway configurations.

In case, real users are used in policy configurations, then the API Provider or API Administrators should take the responsibility of changing the configurations once the user is deleted from the system.

Handling Application Logs

In API Gateway, as you increase the log level to Debug or Trace, there can be messages that include the userid. So when a user has to be deleted from the system, Administrators have to clean up this data. The application logs are stored in file system till API Gateway version 10.2. From API Gateway 10.3, API Administrators have an option to persist the logs additionally in API Data Store.

The following is a sample query run to mask the user data in the application log stored either in the API Data Store or an external data store.

```
curl -X POST -H 'Accept: application/json' -H 'Content-Type: application/json'
http://hostname:port/gateway_default_log/doc/_update_by_query -d '{
"script": { "id": "findAndReplace", "params": {
"find": "user123", "replace": "*****" } } }
```

- *hostname:port* refers to the host name and port of the system where the API Data Store or the external data store that contains the data resides.
- The **Find** field contains the data or user information, such as username, id, that is to be masked.
- The field **Replace** contains the string that is used to mask the data mentioned in the **Find** field and replaces the data with the string provided in the logs.

Logs are always persisted in a file system. You can perform a search and replace using text editing tools. For example, you could search all server.log files for the id of the user to be deleted and replace it with anonymous or blank string. Following logs need to be cleaned up.

- *Install Directory\Integration Server\instances\Instance_Name\logs\server.log*. For details to cleanup other Integration Server related logs, see *webMethods Integration Server Administrator's Guide*.
- *Install Directory\Integration Server\instances\Instance_Name\logs\APIGateway.log*.
- *Install Directory\InternalDataStore\logs*.

Handling Audit Logs

If enabled, audit logs would have information about user actions. This contains user reference and has to be cleaned up after user deletion. You can achieve the cleanup in the following ways:

- As an Administrator, you can clean up the audit logs persisted in API Data Store by running the following curl query to replace or mask the desired data.

The following is a sample query run to mask the user data in the Audit logs stored in the API Data Store.

```
curl -X POST -H 'Accept: application/json' -H 'Content-Type: application/json'
http://hostname:port/gateway_default_audit_auditlogs/_update_by_query -d '{
"script": { "id": "findAndReplace", "params": {
"find": "user123", "replace": "*****" } } } '
```

- *hostname:port* refers to the host name of the system where the API Data Store resides and the corresponding port.
- The **Find** field contains the data or user information, such as `username`, `id`, that is to be masked.
- The field **Replace** contains the string that is used to mask the data mentioned in the **Find** field and replaces the data with the string provided in the logs.
- As an Administrator, you can use the extended setting `saveAuditlogsWithPayload` to not store the request payloads in the audit logs. Though this does not completely eliminate the user information in audit logs it certainly minimizes the occurrences. Software AG recommends you to use this property with caution as turning on this option might lead to less audit data being captured.

Handling Tracer Logs

If you enable the tracer, the data that API Gateway captures might have user-specific information. You can either mask or remove the user data from the server log trace span, mediator trace span, and request response trace span indices. You can either mask full or partial text in the user data.

The following sample query is used to mask the user data in the server log trace span:

```
curl -X POST http(s)://elasticsearch_hostname:elasticsearch_port/
gateway_{tenant}_serverlogtracespans/_doc/_update_by_query?pretty -H 'Content-Type:
application/json' -d'
{
  "script": {
    "id": "findAndReplaceInTracerData",
    "params": {
      "find": "textToFind",
      "replace": "textToReplace"
    }
  }
}'
```

The following sample query is used to mask the user data in the mediator trace span:

```
curl -X POST http(s)://elasticsearch_hostname:elasticsearch_port/
gateway_{tenant}_mediatortracespan/_doc/_update_by_query?pretty -H 'Content-Type:
application/json' -d'
{
  "script": {
    "id": "findAndReplaceInTracerData",
    "params": {
      "find": "textToFind",
      "replace": "textToReplace"
    }
  }
}'
```

The following sample query is used to mask the user data in the request response trace span:

```
curl -X POST http(s)://elasticsearch_hostname:elasticsearch_port/
```

```
gateway_{tenant}_requestresponsetracespans/_doc/_update_by_query?pretty -H  
'Content-Type: application/json' -d'  
{  
  "script": {  
    "id": "findAndReplaceInTracerData",  
    "params": {  
      "find": "textToFind",  
      "replace": "textToReplace"  
    }  
  }  
}
```

- The **Find** field contains the data or user information, such as username, id, that is to be masked.
- The field **Replace** contains the string that is used to mask the data mentioned in the **Find** field and replaces the data with the string provided in the server log trace span, mediator trace span, and request response trace span indices.

8 API Gateway Configuration with Command Central

■ Overview	792
■ Install API Gateway using Command Central	793
■ Manage API Data Store Configurations in Command Central	795
■ Manage API Gateway Product Configurations in Command Central	804
■ Manage Inter-component and Cluster configurations	813
■ Command Line to Manage API Data Store	819
■ Troubleshooting Tips: API Gateway Configuration with Command Central	821

Overview

Command Central allows users who have administration privileges to administer API Gateway and API Data Store.

Note:

Command Central support for API Gateway is deprecated and will be removed in future releases.

Command Central is a centralized application using which administrators can configure multiple Software AG products at a time. When you install API Gateway using Command Central, API Gateway and API Data Store are installed. API Gateway communicates with this API Data store by default. This feature helps administrators to make API Gateway to use an external data store (Elasticsearch) to store its core data and analytics, configure external Kibana, in addition to managing the product configurations such as Ports, Keystores, Truststores, Loggers, License Keys, General Properties, and Clustering.

You can perform the following common functions available in Command Central for API Gateway:

- Install API Gateway using Command Central
- Update fixes using Command Central
- Manage configurations and life cycle of API Data Store
- Product configurations of API Gateway
 - General Properties
 - License Keys
 - Loggers
 - Ports
 - Keystores
 - Truststores
- Inter-component and Cluster configurations
 - Elasticsearch Connection Settings
 - Kibana Connection Settings
 - API Gateway Clustering

Since Command Central supports configuring through its UI and using templates, users can pick their choice for configuring the above seen components. In a typical scenario, administrators prefer configuring through the UI when it is a first time setup and for subsequent configurations, they use templates.

This section describes the operations that are specific to API Gateway. For all common operations, see the *Software AG Command Central Help*.

Install API Gateway using Command Central

You can install API Gateway in either of the following ways:

- Using Command Central UI. See the *Software AG Command Central Help*.
- Using Command Central templates.

Before you begin, ensure that:

- You are familiar with Command Central as a product.
- You are familiar with Command Central templates.
- You have a basic understanding of API Gateway as a product.
- You have a basic understanding of API Gateway administrator configurations.

Installing API Gateway using Command Central Commands

This section lists the steps that you need to run in the Command Central command-line interface for installing API Gateway. For more details on how to use templates and the command-line interface in Command Central, see the *Software AG Command Central Help*.

When you install API Gateway using Command Central, API Gateway, and API Data Store are installed. API Gateway communicates with this API Data Store by default.

1. Run the following command to add the credentials for connecting to the Software AG server. The credentials are maintained in an XML file, *credentials_installer.xml*.

```
sagcc create configuration data local OSGI-CCE-ENGINE COMMON-CREDENTIALS -i  
credentials_installer.xml
```

2. Run the following command to add the repository where the products are available.

```
sagcc add repository products master name=webMethods-10.5 location=<repository url>  
credentials=SAGCONNECT  
description="10.5 repository"
```

credentials=SAGCONNECT. This is the alias for the credentials created in Step 1. The alias is saved in the *credentials_installer.xml* file.

3. Run the following command to add the required license key to install API Gateway.

```
sagcc add license-tools keys apigateway_license -i license_apigateway.xml
```

apigateway_license is the license name that Command Central refers to *license_apigateway.xml* file.

4. Run the following command to add the API Gateway installation template. The sample installation template, *template.yaml* is used in the following command.

```
sagcc exec templates composite import -i sag-apigateway-server-trunk/template.yaml
```

This imports the template required for installing API Gateway.

5. Run the following template to apply the template.

```
sagcc exec templates composite apply sag-apigateway-server  
nodes=local is.instance.type=integrationServer agw.memory.max=512  
repo.product=webMethods-10.5 os.platform=W64  
agw.key.license=apigateway_license
```

This installs API Gateway on the specified node. In this case, it's the local machine. You can specify the required node name in the above command to install in the corresponding node.

6. Run the commands in the given order for applying the fixes:

- a. Add SUM related credentials.

```
sagcc create configuration data local OSGI-CCE-ENGINE COMMON-CREDENTIALS -i  
credentials_fixes.xml.
```

- b. Add the fix repository.

```
sagcc add repository fixes master name=GA_Fix_Repo location=<Fix repo location>  
credentials=EMPOWER  
description="105 GA fix repo"
```

- c. Add the fix template similar to installation template.

```
sagcc exec templates composite import -i  
sag-apigateway-server-qa-fix/template.yaml.
```

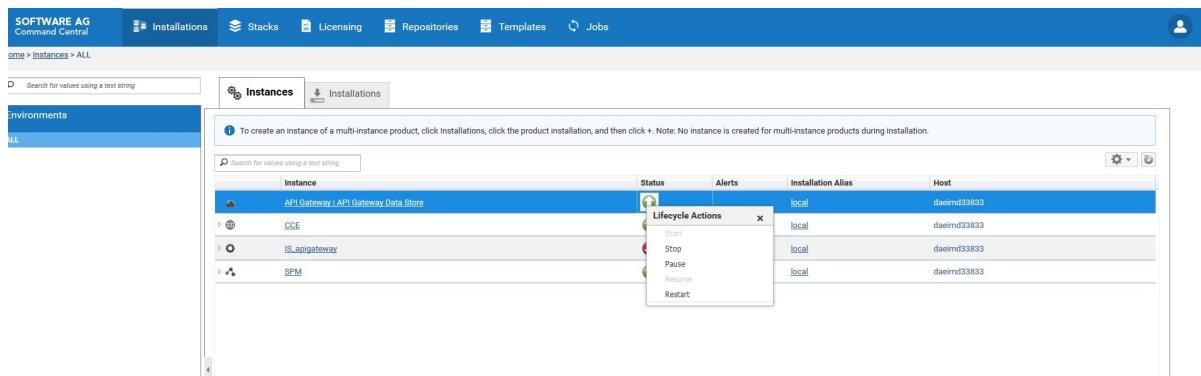
- d. Apply the template.

```
sagcc exec templates composite apply sag-apigateway-server-fix nodes=local  
is.instance.type=integrationServer agw.memory.max=512  
repo.product=webMethods-10.5 os.platform=W64  
agw.key.license=apigateway_license  
is.instance.type=integrationServer repo.fix=GA_Fix_Repo
```

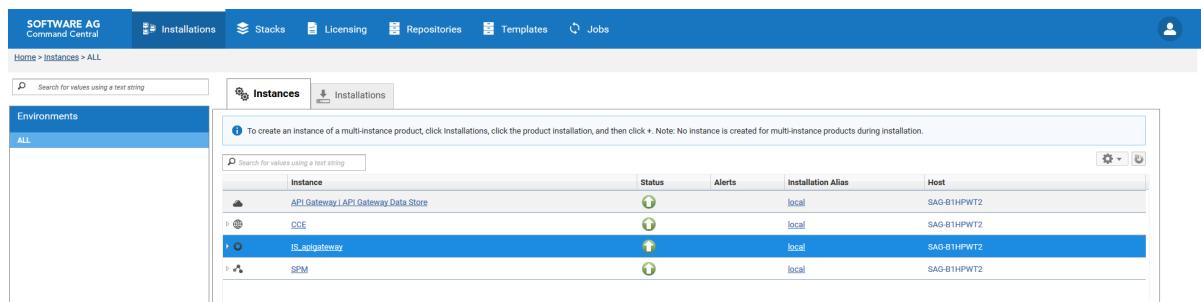
This procedure completes API Gateway installation and you can see API Gateway and API Data Store in Command Central UI.

In Command Central,

- API Gateway > API Data Store contains details about default Elasticsearch shipped with API Gateway.



- IS_<profile> contains details about API Gateway, Digital Event Services, Event Routing, and Integration Server.



Manage API Data Store Configurations in Command Central

Command Central lists API Gateway and API Data Store shipped with API Gateway. API Gateway stores all its core and analytics data in this Data Store by default. You can start, stop, and restart API Data Store from Command Central. You can also manage Clustering details, Keystores, Ports, Properties, and Truststores.

This section describes the following administering tasks for API Data Store:

- “[Starting and Stopping API Data Store in Command Central](#)” on page 796
- “[Changing the API Data Store HTTP Port](#)” on page 798
- “[Changing the API Data Store TCP Port](#)” on page 799
- “[Configuring an API Data Store Cluster](#)” on page 800
- “[Configuring Elasticsearch Properties](#)” on page 803

Administering API Data Store

This section describes the following administering tasks for API Data Store:

- “[Starting, Stopping, and Restarting API Data Store](#)” on page 796
- “[Changing the API Data Store HTTP Port](#)” on page 798

- “[Changing the API Data Store TCP Port](#)” on page 799
- “[Configuring Custom API Data Store Properties](#)” on page 802
- “[Configuring Elasticsearch Properties](#)” on page 803
- “[Monitoring API Data Store](#)” on page 303

Starting, Stopping, and Restarting API Data Store

API Data store uses Elasticsearch 8.2.3. For details on the Elasticsearch versions that are compatible with different API Gateway versions, see “[API Gateway, Elasticsearch, Kibana, and TSA Compatibility Matrix](#)” on page 92.

You can start, stop, and restart your API Data Store instance using the Command Central web user interface and command line interface. Additionally, you can use scripts on Unix and Windows, and the Windows Start menu on Windows to manage the runtime status of your API Data Store instance.

Note:

You must create a temporary directory temp in the *Installation Location/InternalDataStore* and set the “-Djava.io.tmpdir=temp” in the *Installation Location/InternalDataStore/config/jvm.options*.

Starting and Stopping API Data Store in Command Central

Starting API Data Store in Command Central

Use the following procedure to start API Data Store in the Command Central web user interface.

➤ **To start API Data Store**

1. In Command Central, navigate to **Environments > Instances > All > API Data Store**.
2. Click the status icon for API Data Store .
3. From the **Lifecycle Actions** drop-down menu, select **Start**.

Stopping API Data Store in Command Central

Use the following procedure to stop API Data Store in the Command Central web user interface.

➤ **To stop API Data Store**

1. In Command Central, navigate to **Environments > Instances > All > API Data Store**.
2. Click the status icon for API Data Store .
3. From the **Lifecycle Actions** drop-down menu, select **Stop**.

Starting, Stopping, and Restarting API Data Store on Windows

When you install API Data Store on a Windows operating system, you can start and stop your API Data Store instance using the Windows Start menu or using scripts.

To start or stop API Data Store using the Windows Start menu, go to **Start > product install folder**, select **Start API Data Store 10.11** or **Stop API Data Store 10.11** respectively.

To start, stop, or restart API Data Store using scripts, run:

- Start API Data Store -

Software AG_directory \InternalDataStore\bin\config\startup.bat

- Stop API Data Store -

Software AG_directory \InternalDataStore\config\bin\shutdown.bat

- Restart API Data Store -

Software AG_directory \InternalDataStore\config\bin\restart.bat

Starting, Stopping, and Restarting API Data Store on LINUX

Elasticsearch cannot be run as the root user on a Linux system, so you must create a data store user and install and run the data store as that user.

Elasticsearch does several checks before starting up. Software AG recommends that you review the bootstrap checks and important system configuration settings before starting the data store. In particular, you may need to adjust these settings:

- Check the settings for the system-wide maximum number of file descriptors (kernel parameter `fs.file-max`) by running the command `sysctl -a | fgrep fs.file-max`. If the value is less than 65536, log on as the root user and increase the value by running `sysctl -w fs.file-max=200000` or `echo "fs.file-max=65536" >> /etc/sysctl.conf`, then activate the new value by running `sysctl -p`.
- Check the data store user settings for the maximum number of open file descriptors by running the commands `ulimit -Hn` and `ulimit -Sn`, where `-Hn` is the hard limit and `-Sn` is the soft limit. If the value is less than 65536, log on as the data store user and increase the value to at least 65536 by running `ulimit -n 65536`. To permanently save this setting for the user, run the following:

```
echo "user_name soft nofile 65536" >> /etc/security/limits.conf
echo "user_name hard nofile 65536" >> /etc/security/limits.conf
```

- Check the setting for the system-wide maximum map count (kernel parameter `vm.max_map_count`) by running the command `sysctl -a | fgrep vm.max_map_count`. If the value is less than 262144, log on as the root user and increase the value to at least 262144 by running `sysctl -w vm.max_map_count=262144` or `echo "vm.max_map_count=262144" >> /etc/sysctl.conf`, then activate the new value by running `sysctl -p`.

- Check the data store user settings for the maximum number of processes by running the command `ulimit -u`. If the value is less than 4096, log on as the data store user and increase the value to at least 4096 by running `ulimit -n 4096`. To permanently save this setting for the user, run the following:

```
echo "user_name soft nproc 4096" >> /etc/security/limits.conf  
echo "user_name hard nproc 4096" >> /etc/security/limits.conf
```

You can start, stop, and restart API Data Store by running the following commands on LINUX:

- Start API Data Store.

```
./startup.sh
```

- Stop API Data Store.

```
./shutdown.sh
```

- Restart API Data Store.

```
./restart.sh
```

Changing the API Data Store HTTP Port

The default HTTP port that clients use to make calls to API Gateway Data Store is 9240. Use the following procedure to change the HTTP port number.

Note:

You cannot add a new port from this section. You can only edit existing port details.

➤ To change the API Data Store HTTP port

- In Command Central, navigate to **Environments > Instances > All > API Data Store > Configuration**.
- Select **Ports** from the drop-down menu.
- Click **http port** and specify the HTTP port number in the **Port Number** field.
- Optionally, click **Test** to verify your configuration.
- Save your changes.
- Stop API Gateway instance, if it is running.
- Update the Elasticsearch entry in the `config.properties` file located at `SAG_InstallDir/IntegrationServer/instances/tenant_name/packages/WnAPIGateway/config/resources/elasticsearch/`.

Instead of changing the entries manually you can include these changes in one of the following ways:

- Through the externalization of configurations feature. For details, see “[Externalizing Configurations](#)” on page 62
- Through Command Central. For details, see “[Configuring Elasticsearch Connection Settings](#)” on page 813.

8. Restart the API Gateway instance.

Changing the API Data Store HTTP Port using Template

You can change the HTTP Port details using the following Command Central template:

```
sagcc exec templates composite import -i ports.yaml
sagcc exec templates composite apply sag-apigw-datastore-port nodes=local
port.alias=port_alias port.number=port_number
```

Sample ports configuration file:

```
alias: sag-apigw-datastore-port
description: API Gateway Data Store Port configuration
layers:
  runtime:
    templates:
      - apigw-datastore-port
templates:
  apigw-datastore-port:
    products:
      CEL:
        default:
          configuration:
            CEL:
              COMMON-PORTS:
                COMMON-PORTS-defaultHttp:
                  Port:
                    '@alias': ${port.alias}
                    Number: ${port.number}
                    Protocol: HTTP

provision:
  default:
    runtime: ${nodes}
```

Changing the API Data Store TCP Port

Java clients use the TCP port to make calls to API Data Store. In addition, the nodes in an API Data Store cluster use the TCP port to communicate with one another. The default TCP port is 9340.

Important:

If you change the default TCP port, you must change the respective TCP port value in the **Clustering** configuration.

➤ To change the API Data Store TCP port

1. In Command Central, navigate to **Environments > Instances > All > API Data Store > Configuration**.
2. Select **Ports** from the drop-down menu.
3. Click **tcp port** and specify the TCP port number in the **Port Number** field.
4. Optionally, click **Test** to verify your configuration.
5. Save your changes.
6. Restart the API Data Store instance.

In a cluster setup, if you change the TCP port in one node, then you have to change the respective cluster configuration in other nodes. You can change the cluster configuration through Command Central. For details, see “[Configuring an API Data Store Cluster](#)” on page 800.

Configuring an API Data Store Cluster

You can run an API Data Store instance as a single node, or you can configure multiple API Data Store instances to run as a cluster to provide high availability and redundancy.

You can configure API Data Store Cluster in one of the following ways:

- Through Command Central
- Through elasticsearch.yml file

This section describes configuring an API Data Store cluster through Command Central. For details on configuring a cluster using the elasticsearch.yml file, see “[API Data Store Cluster Configuration](#)” on page 31.

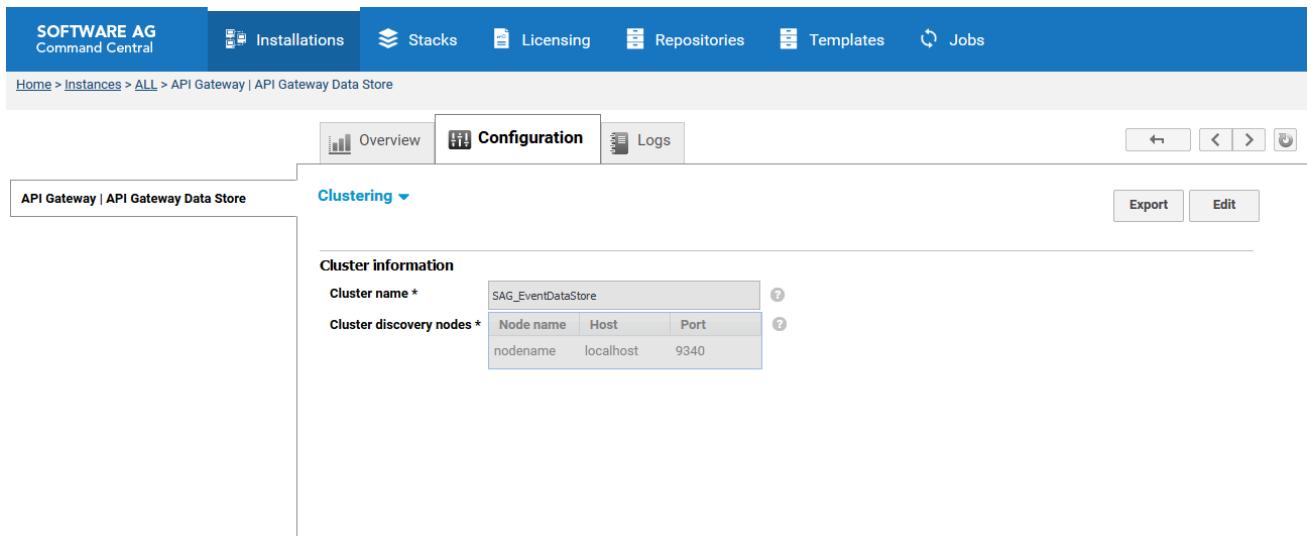
You must specify at least one host and port pair for your configuration in Command Central. API Data Store comes with a default host and port pair.

➤ To configure an API Data Store cluster

1. In Command Central, for each API Data Store instance that is part of the cluster, navigate to **Environments > Instances > All > API Data Store > Configuration**.
2. Select **Clustering** from the drop-down menu, and then click **Edit**.
3. Specify values for each field in the table as outlined in the description column:

Field	Description
Cluster Name	Mandatory. The name of the cluster. All instances must have the same cluster name.

Field	Description
Cluster Discovery Nodes	Mandatory. Click  +, and then do the following to add host and port information for each API Data Store instance that is part of the cluster: <ol style="list-style-type: none"> In the Host column, specify the host information for an API Data Store instance. The default host is <code>localhost</code>. In the Port column, specify the port for an API Data Store instance. The default port is <code>9340</code>. In the Node name column, specify the provide the node name details of the API Data Store instance. Ensure that this name matches with <code>node.name</code> property of the Elasticsearch instance.



The screenshot shows the Software AG Command Central interface. The top navigation bar includes links for Installations, Stacks, Licensing, Repositories, Templates, and Jobs. Below the navigation is a breadcrumb trail: Home > Instances > ALL > API Gateway | API Gateway Data Store. The main content area has tabs for Overview, Configuration (which is selected), and Logs. A sub-menu titled 'Clustering' is open, showing 'Cluster information' and 'Cluster discovery nodes'. The 'Cluster discovery nodes' table contains one row:

Node name	Host	Port
nodenode	localhost	9340

4. Optionally, click **Test** to verify that your configuration is valid.
5. Save your changes.
6. Select **Properties** from the drop-down menu, and then click **Edit**.
7. Specify the Elasticsearch configuration property details. When you want to form a cluster with nodes on other hosts, you must use the `discovery.seed_hosts` setting to provide a list of other nodes in the cluster that are master-eligible and likely to be live and can be contacted in order to seed the discovery process. This setting should normally contain the addresses of all the master-eligible nodes in the cluster as follows:

```
discovery.seed_hosts:
- "<HostName>:<TCPPort>"
- "<HostName>:<TCPPort>"
```

Example:

```
discovery.seed_hosts:  
- "Host1:9340"  
- "Host2:9340"
```

8. Click **Apply** to save your changes.
9. Restart the API Data Store instance.

Configuring Data Store Cluster using Template

You can configure the Data Store cluster using the following Command Central template:

```
sagcc exec templates composite import -i clustering.yaml  
sagcc exec templates composite apply sag-apigw-datastore-clustering nodes=local  
node.name=node_name node.host=node_host node.port=node_port
```

Sample clustering configuration template:

```
alias: sag-apigw-datastore-clustering  
description: API Gateway Data Store Clustering Configuration  
layers:  
  runtime:  
    templates:  
      - apigw-datastore-clustering  
templates:  
  apigw-datastore-clustering:  
    products:  
      CEL:  
        default:  
          configuration:  
            CEL:  
              COMMON-CLUSTER:  
                COMMON-CLUSTER-default:  
                  Enabled: 'true'  
                  Name: SAG_EventDataStore  
                  Servers:  
                    Server:  
                      ExtendedProperties:  
                        Property:  
                          - '@name': node  
                            $: ${node.name}  
                          - '@name': host  
                            $: ${node.host}  
                          - '@name': port  
                            $: ${node.port}  
  
provision:  
  default:  
    runtime: ${nodes}
```

Configuring Custom API Data Store Properties

You can specify custom properties for your Data Store configuration.

➤ To specify custom properties for API Data Store

1. In Command Central, navigate to **Environments > Instances > All > API Data Store > Configuration**.
2. Select **Properties** from the drop-down menu and click **Edit**.
3. In the **Content** field, specify custom parameters. Use YAML syntax and the *property_name : value* format.
4. Restart the API Data Store instance.

Configuring Elasticsearch Properties

From Command Central, you can edit the properties of Elasticsearch that are used by API Data Store. The changes made to the properties are saved in the `elasticsearch.yml` file.

➤ To configure Elasticsearch properties

1. In Command Central, for each API Data Store instance that is part of the cluster, navigate to **Environments > Instances > All > API Data Store > Configuration**.
2. Select **Properties** from the drop-down menu, and then click **Edit**.

This section lists properties maintained in `elasticsearch.yml` file.

3. Make the required your changes.
4. Restart the API Data Store instance.

Configuring Elasticsearch Properties using Template

You can configure the Elasticsearch properties using the following Command Central template:

```
sagcc exec templates composite import -i properties.yaml (properties.yaml)
sagcc exec templates composite apply sag-apigw-datastore-properties nodes=local
```

Sample template:

```
alias: sag-apigw-datastore-properties
description: API Gateway Data Store Properties
layers:
  runtime:
    templates:
      - apigw-datastore-properties
templates:
  apigw-datastore-properties:
    products:
      CEL:
```

```

default:
  configuration:
    CEL:
      CUSTOM-PROPERTIES:
        CUSTOM-PROPERTIES-default: |
        ---
        path.logs: "C:\\\\sag\\\\cc\\\\InternalDataStore/newlogs"
        path.repo:
          - "C:\\\\sag\\\\cc\\\\InternalDataStore/archives"
        cluster.initial_master_nodes:
          - "nodename"
provision:
  default:
    runtime: ${nodes}

```

Manage API Gateway Product Configurations in Command Central

Starting API Gateway 10.5, you can use external Elasticsearch and configure API Gateway to communicate with that Elasticsearch. Once API Gateway is installed using Command Central, it lists installed Integration Server instances as shown in the image below.

Instance	Status	Alerts	Installation Alias	Host
API Gateway / API Gateway Data Store	green	0	local	SAG-B1HPWT2
CCE	green	0	local	SAG-B1HPWT2
IS_apigateway	green	0	local	SAG-B1HPWT2
SPM	green	0	local	SAG-B1HPWT2

The image shows the IS instance apigateway with the name IS_apigateway. Under IS_apigateway, users can configure the following assets and components of API Gateway instances:

- Clusters
- Elasticsearch instances
- General and extended properties
- Keystores
- Kibana instances
- License keys
- Loggers
- Ports
- Truststores

Configuring Properties

This section provides information about configuring Extended and Watt settings of API Gateway.

➤ To configure the Properties

1. In Command Central, navigate to **Environments > Instances > All > API Gateway > Configuration**.
2. Click **General Properties**. The **General Properties** page appears.
3. Click **Extended Settings**. The properties are listed as key value pairs.
4. Make the required changes.
5. Click **Save**.

6. Click **Watt Settings**. The properties are listed as key value pairs.
7. Make the required changes.
8. Save your changes.

Configuring Keystores

This section provides information about adding keystores for API Gateway from Command Central.

➤ To configure the Keystores

1. In Command Central, navigate to **Environments > Instances > All > API Gateway > Configuration**.
2. Select **Keystores** from the drop-down menu.

The Keystores list appears.

3. Click  to add a new keystore.
4. Provide an **Alias** for the keystore.
5. Provide **Type, Provider, and Location** of the keystore in the **Keystore Configuration** section.

6. Click **Save**.

The keystore is added to the list.

Configuring Keystores using Template

You can configure Keystores using the following Command Central template:

```
sagcc exec templates composite import -i keystore.yaml
sagcc exec templates composite apply keyStoreAlias nodes=local
keystore.path=youkeystorepath
keystore.password=keystorepassword key.alias=keyAlias
key.password=keyPassword
```

Sample keystore configuration template

```
alias: keyStoreAlias
description: API Gateway keystore creation
layers:
  runtime:
    templates: keyStore-Template
templates:
  keyStore-Template:
    products:
      integrationServer:
        apigateway:
          configuration:
            OSGI-IS_apigateway-WmAPIGateway:
              COMMON-KEYSTORES:
                COMMON-KEYSTORES_pgkey:
                  Keystore:
                    '@alias': pgkey
                    Description: pgkey
                    Type: JKS
                    Provider: SUN
                    Location: ${keystore.path}
                    Password: '{AES/CBC/PKCS5Padding}
{7BhetRrOVU+AVsox8WKkwQwMVemomS3dpCgNJj5ByYA=}
{JSQ88/tEzqkDGq8D+GWrw==}uSFvFjWALKWdMOAjuwGpVA=='
```

```

Key:
- '@alias': partner1
  Password: '{AES/CBC/PKCS5Padding}
  {VPQ5ojZEZgzUR7x0WF0317R0K+bxvMyjSCSigoBiAEo=}'
  {+96qyCFXAiXg2gX3CzdIWA==}7kAeXaZcieuJuRefScC0Ig=='
- '@alias': partner2
  Password: '{AES/CBC/PKCS5Padding}
  {4cu7D8zz+Bng2CvoeX71tlb1TSv5yKwqNAXjDN1yLKI=}'
  {wOE8hwy02s5BLSZV1tKtNA==}mIVtB9dVL8TCVb35zQGJaA=='
- '@alias': policygateway
  Password: '{AES/CBC/PKCS5Padding}
  {PWBrB05D5w6KSdloz8q8yTcrVThizebyPhre1u7gXb4=}'
  {FuESDHISW1rXqmBIfL7P7g==}/hMP4Bzp0hmCF2Jlrsy00w=='
ExtendedProperties:
  Property:
    - '@name': fileContent
      $:

```

Configuring Licenses

This section provides information about adding API Gateway licenses using Command Central.

➤ To configure Licenses

1. In Command Central, navigate to **Environments > Instances > All > API Gateway > Configuration**.
2. Select **License Keys** from the drop-down menu.

The License Keys list appears.

The screenshot shows the Command Central navigation bar at the top with links for Installations, Stacks, Licensing, Repositories, Templates, and Jobs. Below the navigation bar, the path Home > Instances > ALL > IS_apigateway is displayed. On the left, there is a sidebar with sections for API Gateway, Digital Event Services, Event Routing, and Integration Server. The main content area has tabs for Overview, Configuration, and Logs, with the Configuration tab selected. Under the Configuration tab, there is a sub-tab labeled 'License Keys'. A table titled 'License Keys' is shown with two entries:

License Type	Status	Expiration Date
APIGateway	Valid	01 Dec 2019
IS-Terracotta	Not present	Never

3. Click to add a new license and provide the required license.

Configuring Loggers

➤ To configure Loggers

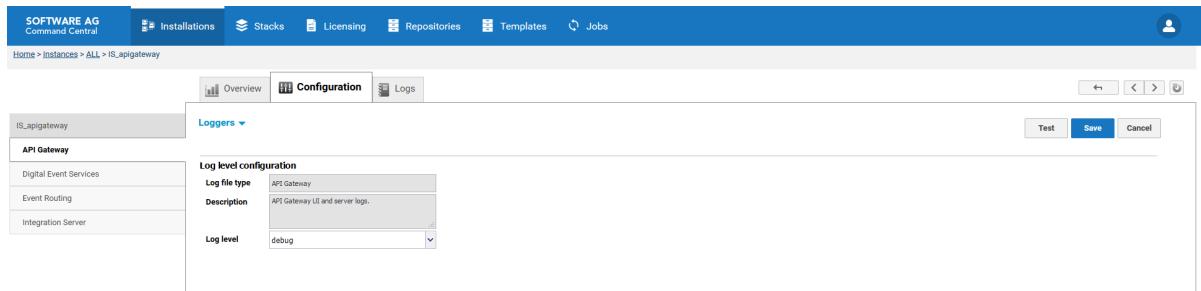
1. In Command Central, navigate to **Environments > Instances > All > API Gateway > Configuration**.

2. Select **Loggers** from the drop-down menu.

This section displays components and their corresponding log levels.

3. Follow these steps to change the log level of a component:

- a. Click the required log file type from the list.
- b. Select the required **Log Level** from the drop-down list.



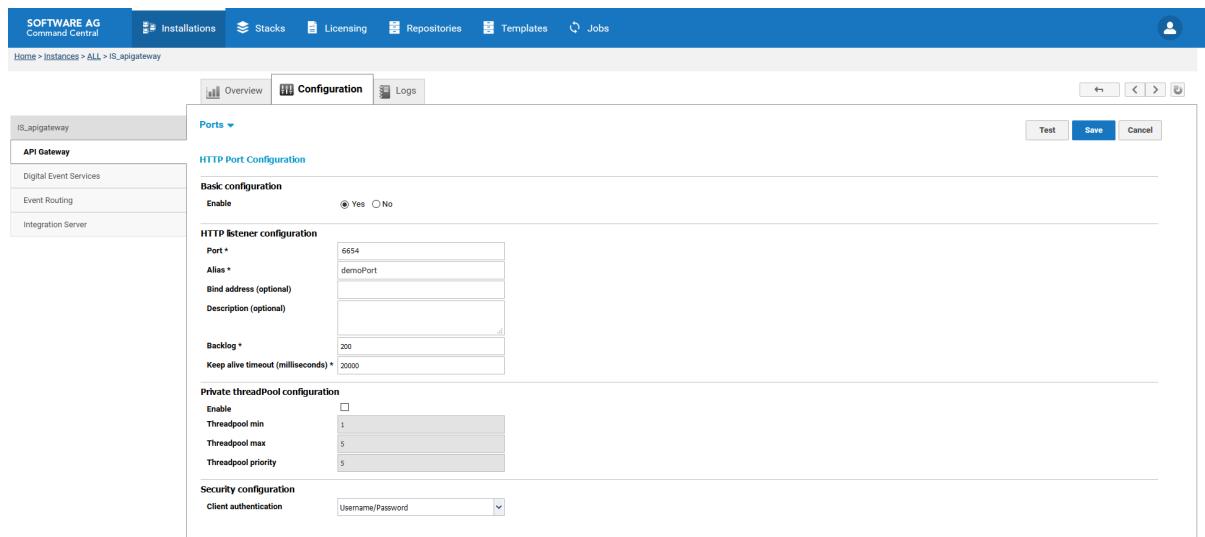
- c. Click **Save**.

Configuring HTTP Port

This section provides information about configuring HTTP ports available in API Gateway.

➤ To configure the HTTP port

1. In Command Central, navigate to **Environments > Instances > All > API Gateway > Configuration**.
2. Select **Ports** from the drop-down menu.
3. Click **HTTP Port Configuration**.
4. Select **Yes** in the **Enable** field in the **Basic configuration** section.
5. Provide valid port numbers in the **Port** and **Alias** field of the **HTTP listener configuration** section.



6. Optionally, click **Test** to verify your configuration.
7. Save your changes.
8. Restart the API Gateway instance.

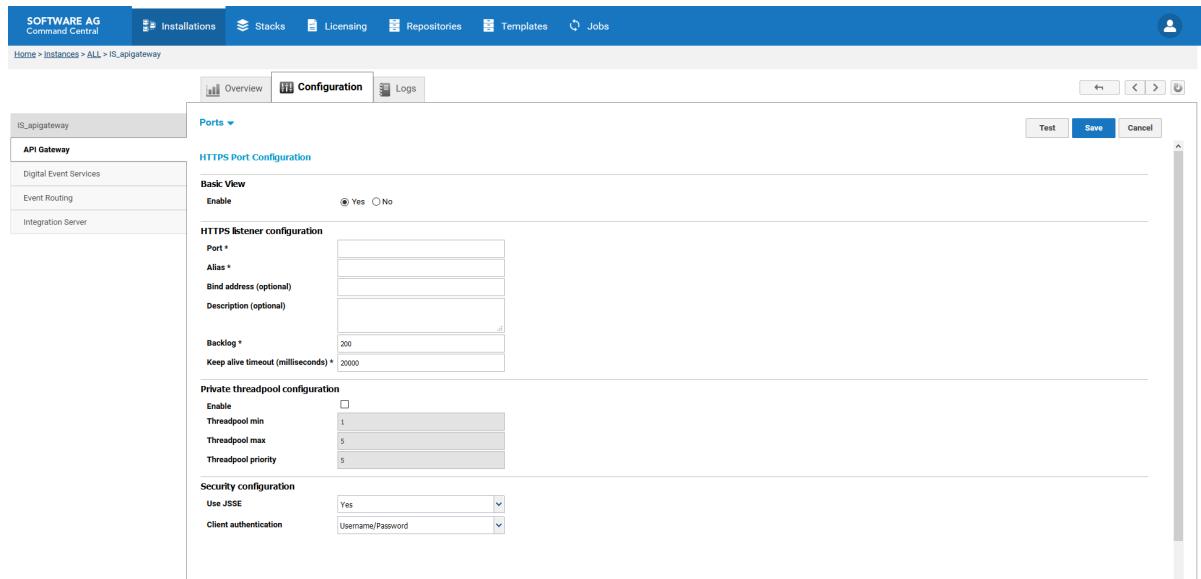
The port is created and enabled.

Configuring HTTPS Port

This section provides information about configuring HTTPS ports available in API Gateway.

➤ To configure the HTTPS port

1. In Command Central, navigate to **Environments > Instances > All > API Gateway > Configuration**.
2. Select **Ports** from the drop-down menu.
3. Click **HTTPS Port Configuration**.
4. Select **Yes** in the **Enable** field in the **Basic configuration** section.
5. Provide valid port numbers in the **Port** and **Alias** field of the **HTTPS listener configuration** section.
6. Select the required Keystore and Truststore from the available list of options.



7. Optionally, click **Test** to verify your configuration.
8. Save your changes.
9. Restart the API Gateway instance.

The port is created and enabled.

Configuring HTTPS Port using Template

You can configure port by using the following Command Central template:

```
sagcc exec templates composite import -i httpPort.yaml
sagcc exec templates composite apply httpPortAlias
```

Sample ports configuration template

```
alias: httpsPortAlias
description: API Gateway https port creation
layers:
  runtime:
    templates: httpsPort-Template
templates:
  httpsPort-Template:
    products:
      integrationServer:
        apigateway:
          configuration:
            OSGI-IS_apigateway-WmAPIGateway:
              COMMON-PORTS:
                COMMON_PORTS_HTTPS:
                  Port:
                    '@primary': 'false'
                    '@alias': HTTPS
                    Enabled: 'true'
```

```

CustomType: HTTPSListener@5558
Number: '5558'
Protocol: HTTPS
Backlog: '200'
KeepAliveTimeout: '20000'
ThreadPool:
SSL:
  KeystoreAlias: pgkey
  KeyAlias: partner2
  TruststoreAlias: trust
ExtendedProperties:
  Property:
    - '@name': DIS_PORT
      $: '5558'
    - '@name': DIS_PORT_ALIAS
      $: HTTPS
    - '@name': DIS_PROTOCOL
      $: HTTPS
    - '@name': DIS_ENABLE
      $: 'true'
    - '@name': DIS_PRIMARY
      $: 'false'
    - '@name': UseJSSE
      $: 'true'
    - '@name': listenerType
      $: Regular
    - '@name': Type
      $: Regular
    - '@name': DIS_TYPE
      $: Regular
    - '@name': PortType
      $: HTTPS
    - '@name': PortDescription
      $: https ports
    - '@name': ClientAuth
      $: require
    - '@name': IdleTimeout
    - '@name': MaxConnections
    - '@name': ProxyHost
    - '@name': Username
    - '@name': Password
provision:
  default:
    runtime: ${nodes}

```

Configuring Truststores

This section provides information about adding Truststores for API Gateway from Command Central.

➤ To configure the Truststores

1. In Command Central, navigate to **Environments > Instances > All > API Gateway > Configuration**.

2. Select **Truststores** from the drop-down menu.

The Truststores list appears.

3. Click  to add a new Truststore.
4. Provide an **Alias** for the Truststore.
5. Provide **Type**, **Provider**, and **Location** of the Truststore in the **Truststore Configuration** section.
6. Click **Save**.

The Truststore is added to list.

Configuring Truststores using Template

You can configure Truststores using the following Command Central template:

```
sagcc exec templates composite import -i truststore.yaml
sagcc exec templates composite apply trustStoreAlias nodes=local
truststore.location=trustStoreLocation
truststore.password=trustStorePassword
```

Sample Truststores configuration template

```
alias: trustStoreAlias
description: API Gateway trust store creation
layers:
  runtime:
    templates: trustStore-Template
templates:
  trustStore-Template:
    products:
      integrationServer:
        default:
          configuration:
            OSGI-IS_apigateway-WmAPIGateway:
              COMMON-TRUSTSTORES:
                COMMON-TRUSTSTORES_testTrustStore:
                  Truststore:
                    '@alias': testTrustStore
                    Description: Test truststore for command central
                    Type: JKS
                    Provider: SUN
                    Location: ${truststore.location}
                    Password: ${truststore.password}
                  ExtendedProperties:
                    Property:
                      - '@name': certificateAliases
                      $:
addtrustclass1ca,addtrustexternalca,addtrustqualifiedca,baltimorecodesigningca,baltimorecybertrustca,
comodocaaca,entrust2048ca,entrustclientca,entrustglobalclientca,entrustgsslca,entrustsslca,eqifaxsecureca,eqifaxsecurebusinessca,
eqifaxsecurebusinessca2,eqifaxsecureglobalbusinessca1,geotrustglobalca,godaddyclass2ca,gtecybertrust5ca,gtecybertrustca,
```

```

gtecybertrustglobalca,lhca,partner1,partner2,policygateway,soneraclass1ca,soneraclass2ca,starfieldclass2ca,synapse,
thawtepersonalbasicca,thawtepersonalfreemailca,thawtepersonalpremiumca,thawtepremiumserverca,thawteserverca,
utndatacorpsgccca,uthuserfirstclientauthemailca,uthuserfirsthardwareca,uthuserfirstobjectca,valicertclass2ca,
verisignclass1ca,verisignclass1g2ca,verisignclass1g3ca,verisignclass2ca,verisignclass2g2ca,verisignclass2g3ca,
verisignclass3ca,verisignclass3g2ca,verisignclass3g3ca,verisignserverca,webm test ca
    - '@name': isLoaded
      $: 'true'
    - '@name': fileContent
      $:
/u3+7QAAAAIAAAAxAAAAAgAMd2VibSB0ZXN0IGNhAAABSLIi/poABVguNTA5AADazCCA2cwgJPo
AMCAQICBFQih6gwDQYJKoZIhvcNAQELBQAwazELMAkGA1UEBhM

JoAMCAQICBDdwz7UwDQYJKoZIhvcNAQEFBQAwTjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDkVxdWlmYXggU2VjdXJlMSYwJAYD
    - '@name': fileName
      $: cacerts
provision:
  default:
    runtime: ${nodes}

```

Manage Inter-component and Cluster configurations

This section describes the administering tasks for the following API Gateway components:

- Elasticsearch Connection Settings
- Kibana Connection Settings
- API Gateway Clustering

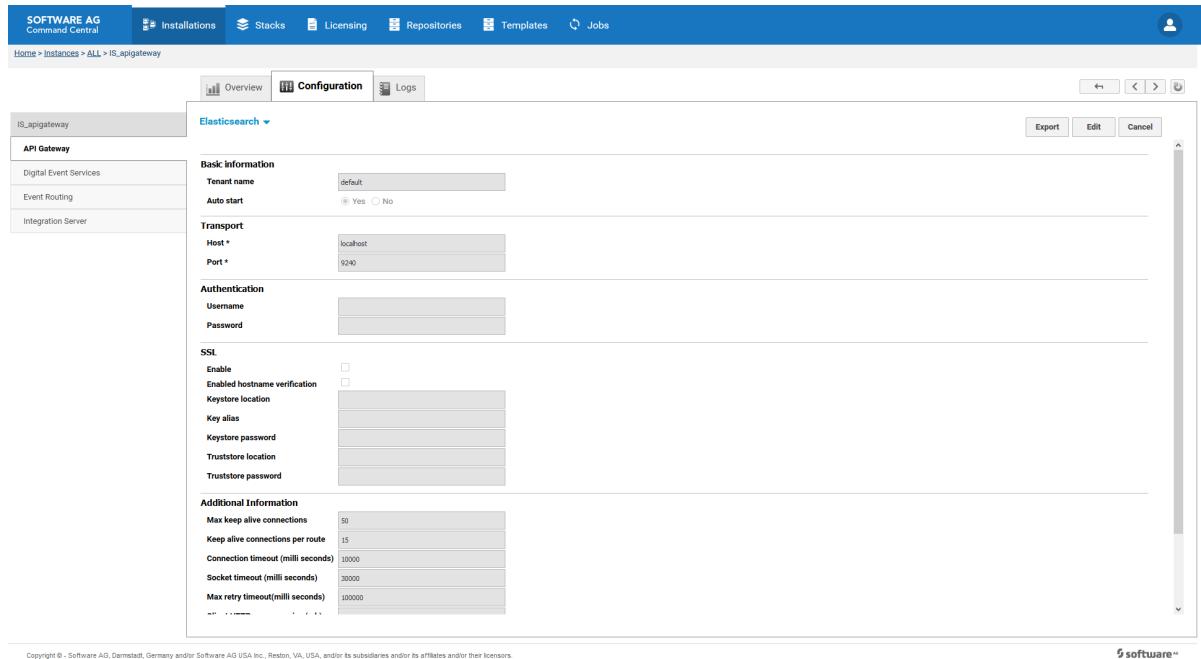
Configuring Elasticsearch Connection Settings

This section provides information about configuring internal or external Elasticsearch for API Gateway.

➤ To configure Elasticsearch

1. In Command Central, navigate to **Environments > Instances > All > API Gateway > Configuration**.
2. Click **Elasticsearch** from the drop-down menu. The Elasticsearch section appears.
3. Provide **Tenant name**.
4. Select one of the following values in the **Auto start** field:
 - Yes - if you are using internal Elasticsearch.
 - No - if you are using external Elasticsearch.
5. Provide the **Host** and **Port** of the server where the Elasticsearch (external or internal) is running, in the **Transport** section.

6. If the Elasticsearch is protected with basic authorization, provide the user name and password in the **Authentication** section.
7. If the Elasticsearch is protected with HTTPS, perform the following in the **SSL** section:
 - a. Select the **Enable** check box.
 - b. Provide valid **Keystore** and **Truststore** details.
8. Provide additional configurations that defines the API Gateway's connectivity to Elasticsearch in the **Additional Information** section.



9. Save your changes.

The **Pending restart** value is changed to *true* and **Status** is *Enabled*.

10. Restart the API Gateway instance.

The Elasticsearch details are updated in API Gateway.

Configuring External Elasticsearch using Template

You can configure external Elasticsearch using the following Command Central template:

```
sagcc exec templates composite import -i cc-minimal-es.yaml
sagcc exec templates composite apply cc-minimal-es nodes=local ssl_username=username
  ssl_password=password
eshost=eshost esport=esport keystore_location=your_keystore_location
keystore_alias=alias_of_keystore
truststore_location=your_truststore_location truststorealias=your_truststore_alias
```

```
truststore_password=truststorepassword
```

Sample external Elasticsearch configuration template

```

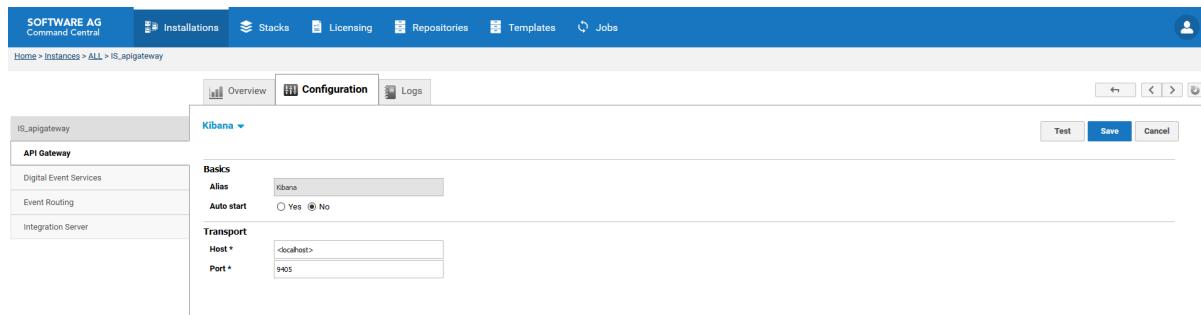
alias: elasticsearch-alias
description: Elastic search configuration
layers:
  runtime:
    templates:
      - cc-minimal-es
templates:
  cc-minimal-es:
    products:
      integrationServer:
        default:
          configuration:
            OSGI-IS_apigateway-WmAPIGateway:
              APIGATEWAY-ELASTICSEARCH:
                APIGATEWAY-ELASTICSEARCH:
                  '@alias': Elasticsearch
                  autostart: 'false'
                  tenantId: apigateway
                  Auth:
                    '@type': SSL
                    User: ${ssl_username}
                    Password: ${ssl_password}
                  Transport:
                    Host: ${eshost}
                    Port: ${esport}
                  SSL:
                    Enable: 'true'
                    HostnameVerification: 'false'
                    KeystoreLocation: ${keystore_location}
                    KeystoreAlias: ${keystore_alias}
                    TruststoreLocation: ${truststore_location}
                    TruststoreAlias: ${truststore_alias}
                    TruststorePassword: ${truststore_password}
                  ExtendedProperties:
                    Property:
                      - '@name': clientHttpResponseSize
                        $: '1024'
                      - '@name': connectionTimeout
                        $: '10000'
                      - '@name': keepalive
                        $: '10'
                      - '@name': keepAliveConnectionsPerRoute
                        $: '1000'
                      - '@name': maxRetry
                        $: '10000'
                      - '@name': socketTimeout
                        $: '10000'
                      - '@name': sniffEnabled
                        $: 'true'
                      - '@name': sniffTimeInterval
                        $: '5000'
provision:
  default:
    runtime: ${nodes}
```

Configuring Kibana Connection Settings

This section provides information about configuring internal or external Kibana for API Gateway from Command Central.

➤ To configure Kibana

1. In Command Central, navigate to **Environments > Instances > All > API Gateway > Configuration**.
2. Select **Kibana** from the drop-down menu.
The Kibana instances list appears.
3. Click the instance that you want to configure.
4. Select one of the following values in the **Auto start** field:
 - Yes - if you are using internal Kibana.
 - No - if you are using external Kibana.
5. If you are using external Kibana, provide the **Host** and **Port** of the server where the Kibana is running in the **Transport** section. Else, do not enter any values in those fields.



6. Save your changes.

The **Pending restart** value is changed to *true* and **Status** is *Enabled*.

7. Restart the API Gateway instance.

The Kibana details are updated in API Gateway.

Configuring Kibana using Template

You can configure Kibana using the following Command Central template:

```
sagcc exec templates composite import -i cc-kibana.yaml
sagcc exec templates composite apply cc-kibana nodes=local host=hostname port=portnumber
```

Sample Kibana configuration template

```
alias: cc-kibana-alias
description: HTTPS elastic search template
layers:
  runtime:
    templates:
      - cc-kibana
templates:
  cc-kibana:
    products:
      integrationServer:
        default:
          configuration:
            OSGI-IS_apigateway-WmAPIGateway:
              APIGATEWAY-KIBANA:
                APIGATEWAY-KIBANA:
                  '@alias': Kibana
                  autostart: 'false'
                  Transport:
                    Host: ${host}
                    Port: ${port}

provision:
  default:
    runtime: ${nodes}
```

Configuring API Gateway Cluster

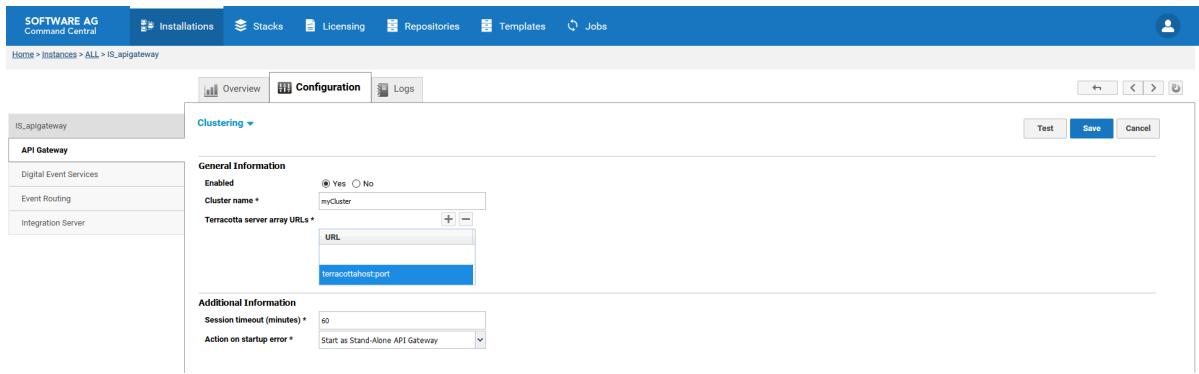
This section provides information about configuring cluster details for API Gateway in the API Gateway section.

Note:

Ensure that the Terracotta server is running when configuring cluster.

➤ To configure API Gateway Clustering

1. In Command Central, navigate to **Environments > Instances > All > API Gateway > Configuration**.
2. Select **Clustering** from the drop-down menu.
The initial clustering status appears as *Disabled*.
3. Click **Disabled**. The **General Information** section appears.
4. Click **Edit** to provide the cluster details.



5. Select **Yes** in the **Enable** field.
6. Provide **Cluster name**.
7. Provide the host name and port of the server where Terracotta is running, in the **Terracotta server array URLs** field.
8. Optionally, click **Test** to verify your configuration.
9. Save your changes.

The **Pending restart** value is changed to *true* and **Status** is *Enabled*.

10. Restart the API Gateway instance.

The clustering details are updated in API Gateway.

Configuring Cluster using Template

You can configure Cluster using the following Command Central template:

```
sagcc exec templates composite import -i cc-clustering.yaml
sagcc exec templates composite apply commandcentral-clustering-alias nodes=local
tchost=terracotta_host tcport=terracotta_port
```

Sample clustering configuration template

```
alias: cc-clustering-alias
description: cluster config
layers:
  runtime:
    templates:
      - cc-clustering
templates:
  cc-clustering:
    products:
      integrationServer:
        default:
          configuration:
            OSGI-IS_apigateway-WmAPIGateway:
```

```

COMMON-CLUSTER:
  COMMON-CLUSTER:
    Enabled: 'true'
    Name: APIGatewayTSAcluster
    Servers:
      Server:
        URL: daeirnd33974:9510
    ExtendedProperties:
      Property:
        - '@name': SessionTimeout
          $: '60'
        - '@name': ActionOnStartupError
          $: standalone
provision:
  default:
    runtime: ${nodes}

```

Command Line to Manage API Data Store

You can manage Data Store using command line. This section provides details about the various commands and configuration types that the Data Store supports, the run-time monitoring statuses and the lifecycle actions for the Data Store.

Commands that API Data Store Supports

API Data Store supports the Platform Manager commands listed in the following table. The table also lists where you can find information about each command.

Commands	Additional Information
sagcc get configuration data	For general information about the command, see <i>Software AG Command Central Help</i> .
sagcc update configuration data	For general information about the command, see <i>Software AG Command Central Help</i> .
sagcc get configuration instances	For general information about the command, see <i>Software AG Command Central Help</i> .
sagcc list configuration instances	For general information about the command, see <i>Software AG Command Central Help</i> .
sagcc get configuration types	For general information about the command, see <i>Software AG Command Central Help</i> .
sagcc list configuration types	For general information about the command, see <i>Software AG Command Central Help</i> .
sagcc exec configuration validation update	For general information about the command, see <i>Software AG Command Central Help</i> .
sagcc exec lifecycle	For general information about the command, see <i>Software AG Command Central Help</i> .

Commands	Additional Information
sagcc get monitoring	For general information about the command, see <i>Software AG Command Central Help</i> .

Configuration Types that API Data Store Supports

The following table lists the configuration types that the API Data Store run-time component supports, along with the description of each configuration type:

Configuration type	Description
COMMON-CLUSTER	Settings for an API Data Store cluster. You can configure the name of the cluster and the host and port pairs of the server endpoints of the cluster. Note: The changes that you make to a cluster configuration take effect after you restart API Data Store.
COMMON-PORTS	Configuration instances for HTTP and TCP ports.
CUSTOM-PROPERTIES	Additional properties for the configuration of an API Data Store server.

Run-Time Monitoring Statuses for API Data Store

The following table lists the run-time statuses that the API Data Store run-time component can return in response to the `sagcc get monitoring state` command, along with the meaning of each run-time status.

Run-time status	Meaning
ONLINE	The API Data Store instance is running.
STOPPED	The API Data Store instance is stopped.

Lifecycle Actions for API Data Store

The following table lists the actions that API Data Store supports with the `sagcc exec lifecycle` command, along with the description of each action:

Action	Description
start	Starts the API Data Store instance.

Action	Description
stop	Stops the API Data Store instance.
restart	Restarts the API Data Store instance.

You can also perform these actions in the Command Central web user interface.

Troubleshooting Tips: API Gateway Configuration with Command Central

I see that the HTTPS port for API Gateway UI is not working

When API Gateway is installed using Command Central, API Gateway Administration page is not accessible with default ports.

Resolution:

Check if com.softwareag.catalina.connector.https.pid-apigateway.properties file is available at `SAGInstallDirectory\profiles\IS_instance_name\configuration\com.softwareag.platform.config.propsloader`.

If the file is missing in the specified location, contact SoftwareAG Support to fix the problem.

Note:

Do not copy the file from a different location. If the file is copied from a different location, the HTTPS port might not work.

