# API Gateway Configuration Guide

Version 10.11

October 2021

WEBMETHODS

This document applies to webMethods API Gateway 10.11 and to all subsequent releases.

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

**Document ID: YAI-CG-1011-20211015**

# Table of Contents

# About this Documentation

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 *folder.subfolder.service*, APIs, Java classes, methods, properties. |
| *Italic* | Identifies:<br>Variables for which you must supply values specific to your own situation or environment.<br>New terms the first time they occur in the text.<br>References to other documentation sources. |
| `Monospace font` | Identifies:<br>Text you must type in.<br>Messages displayed by the system.<br>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

### Software AG Documentation Website

You can find documentation on the Software AG Documentation website at https://documentation.softwareag.com.

### Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at https://empower.softwareag.com/.

You can find product information on the Software AG Empower Product Support website at https://empower.softwareag.com.

To submit feature/enhancement requests, get information about product availability, and download products, go to Products.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the Knowledge Center.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at https://empower.softwareag.com/public_directory.aspx and give us a call.

### Software AG Tech Community

You can find documentation and other technical information on the Software AG Tech Community website at https://techcommunity.softwareag.com. You can:

- Access product documentation, if you have Tech Community credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

## 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 API Gateway Architecture

## API Gateway Deployment

You can deploy API Gateway in two editions based on the type of license used:

- **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.

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*

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<br>■ Ports<br>■ License management<br>■ Load balancing<br>■ Keystore configuration | Yes | Yes |
| Administration<br>■ Extended settings | No | Yes |
| Alias management | No | Yes |
| Service management | No | Yes |
| Policy management<br>■ Threat protection rules | Yes | Yes |
| Policy management<br>■ Global policies<br>■ Policy templates | No | Yes |
| Export and Import | No | Yes |

| Feature | Standard Edition | Advanced Edition |
|---|---|---|
| ■ APIs<br>■ Global policies | | |
| Application management | No | Yes |
| Plans and packages | No | Yes |
| Analytics<br>■ Threat protection rule violations | Yes | Yes |
| Analytics<br>■ Service<br>■ Applications<br>■ Consumers | No | Yes |
| Clustering and auto synchronization | No | Yes |

## API Gateway Components

A basic API Gateway setup consists of following three components:

- **API Gateway**
- **API Data Store (Elasticsearch)**. API Gateway installation bundles a default Elasticsearch, which is called as API Data Store and all the data by default is written to this Elasticsearch. API Gateway uses Elasticsearch as its primary data store for persisting different types of data like APIs, Policies, Applications, and other assets. API Gateway data is broadly classified into four types as follows:
  - Core configuration
  - Runtime transactions
  - Application logs
  - Audit logs

  You can store the core configuration data separately from the other types of data (such as runtime transactions, application logs, and audit logs). It is better to separate data when there is a large volume of runtime transactions data. In such cases you can use external Elasticsearch to store the runtime transactions that enables you to scale up or scale down the Elasticsearch independently. API Gateway provides an option to configure external Elasticsearch where API Gateway can store data such as runtime transactions, application logs, and audit logs.

- **Kibana**. Kibana is a data visualization dashboard software that works with Elasticsearch. It provides search and data visualization capabilities for the data indexed in Elasticsearch. API Gateway installation bundles a default Kibana. As Kibana and Elasticsearch have a one-to-one mapping, API Gateway provides an option to configure external Kibana.

By default, all the three components run in the same node, hence when you start the API Gateway instance, API Data Store and Kibana gets started. If API Data Store and Kibana fails to start on their own, you have to start it manually as follows:

- To start the API Data Store, run the startup.bat file located at `Install_Dir\instance_name\InternalDataStore\bin`.

  **Note:**
  To shutdown API Data Store run the shutdown.bat.

- To start the Kibana, run the kibana.bat (Windows) or kibana.sh (Linux) file located at`Install_Dir\profiles\IS_default\apigateway\dashboard\bin`.

If you want to scale Elasticsearch and Kibana independently, you can install and run them in two different nodes by configuring external Elasticsearch and external Kibana.

For details about how to configure and connect to external Elasticsearch and external Kibana, see " Connecting to an External Elasticsearch" on page 115 and " Connecting to an External Kibana" on page 120 respectively.

## API Gateway Deployment Scenarios

API Gateway enforces threat protection, policies and routing capabilities for APIs. This section describes high-level API Gateway architecture for various deployment scenarios.

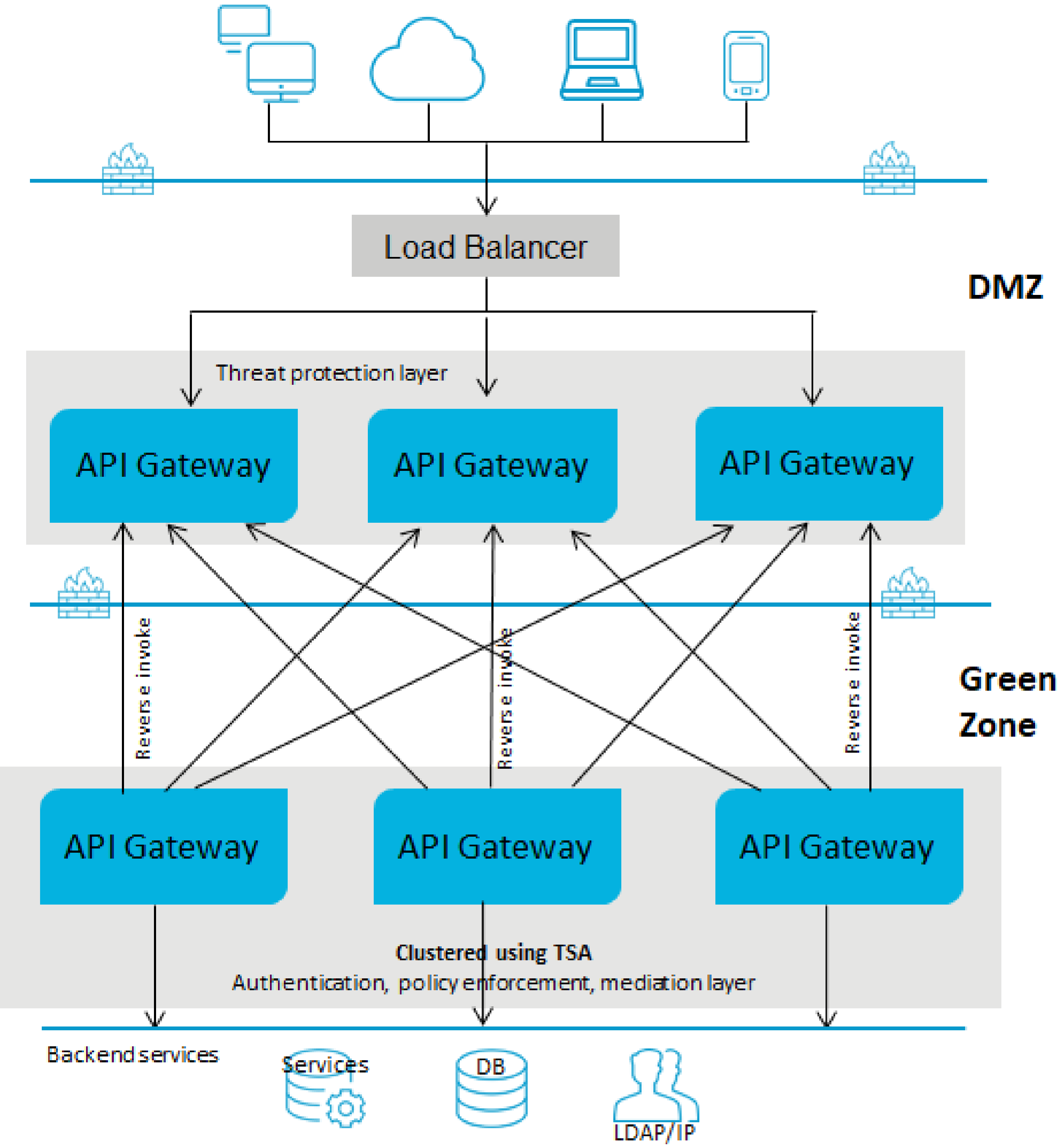
### Deployment scenario 1: Paired gateway deployment

This setup consists of:

- One or more standard edition API Gateways for threat protection and connected to a load balancer in DMZ.
- One or more advanced version API Gateways clustered in the green zone to enforce policies and provide routing capabilities. You can have multiple instances of API Gateways connected through a load balancer and clustered using Terracotta Server Array. You can add an extra layer of protection by using reverse invoke.

A firewall protects the API Gateway infrastructure in the paired deployment. You can add an extra layer of protection by using reverse invoke. The API Gateways communicate between the zones using the reverse invoke approach.

The following diagram provides an architectural overview of the paired gateway deployment:



**Note:**
If you have multiple instances of API Gateway connected using a load balancer for threat protection and you change the enforced rules on one of the API Gateway instances, you must restart the other instances to synchronize the rule enforcement across all the API Gateway instances.

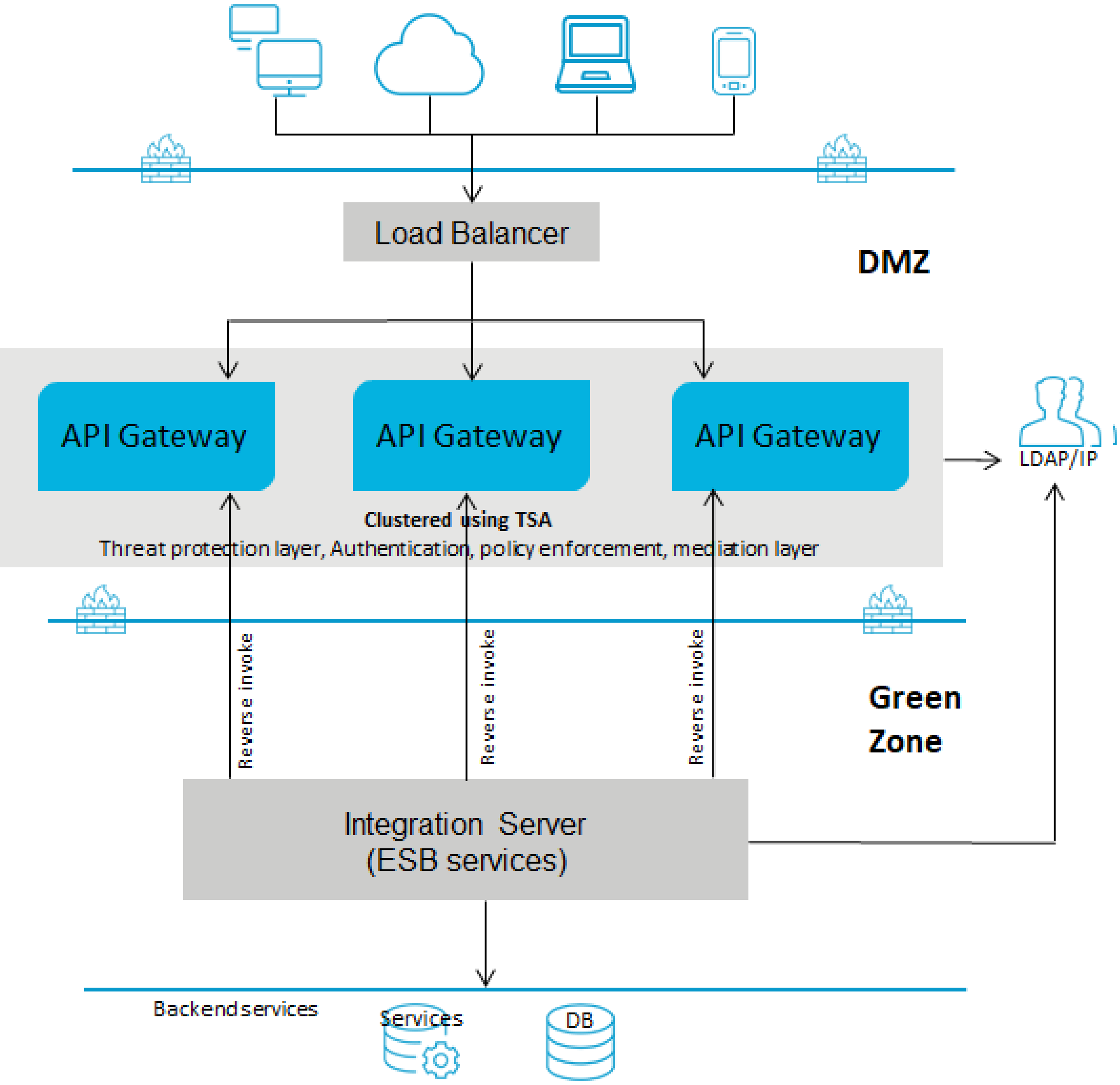
## Deployment scenario 2: Single gateway in the DMZ for webMethods customers

This setup consists of:

- One or more advanced edition API Gateways clustered and connected to a load balancer in DMZ. You can have multiple instances of API Gateways connected through a load balancer and clustered using Terracotta Server Array. A single API Gateway is used for enforcing authentication and routing capabilities.
- The ESB services in Integration Server reside in the green zone behind the firewall.

If you use reverse invoke for communication between API Gateway and the internal ESB, ensure that the endpoint in the routing policy applied is configured as apigateway://*registrationPort-aliasname*/*relative path* of the service. For details, see the Ports section and the Routing policies section in *webMethods API Gateway User's Guide*.

The following diagram provides an architectural overview of the API Gateway deployment in a DMZ for webMethods customers:

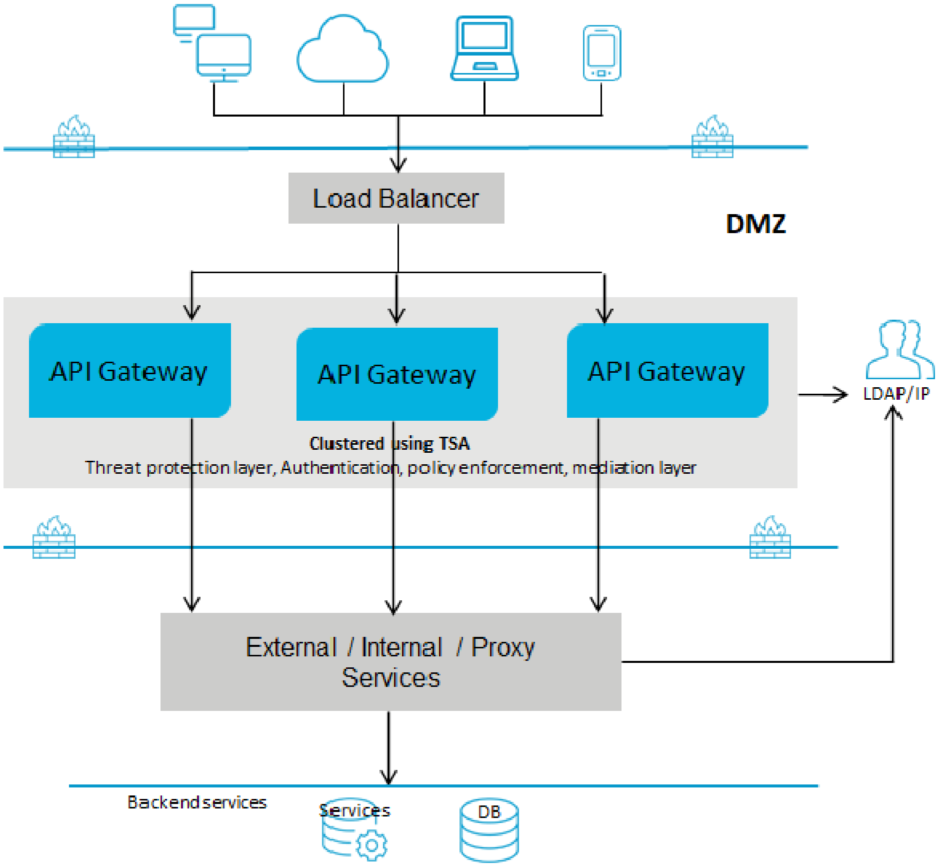


## Deployment scenario 3: Single gateway in the green zone for webMethods customers

This setup consists of:

- One or more advanced edition API Gateways clustered in the green zone and connected to a load balancer in DMZ. A single API Gateway is used for enforcing authentication and routing capabilities. This deployment does not require threat protection. However, you can configure and enforce threat protection, if required. You can have multiple instances of API Gateways connected through a load balancer and clustered using Terracotta Server Array.
- The ESB services in Integration Server reside in the green zone behind the firewall. Because the API Gateway and the ESB services reside in the green zone, the ESB services are directly invoked.

The following diagram provides an architectural overview of the API Gateway deployment in the green zone for webMethods customers:


Load Balancer
DMZ
API Gateway
API Gateway
API Gateway
LDAP/IP
Clustered using TSA
Threat protection layer, Authentication, policy enforcement, mediation layer
External / Internal / Proxy Services
Backend services
Services
DB

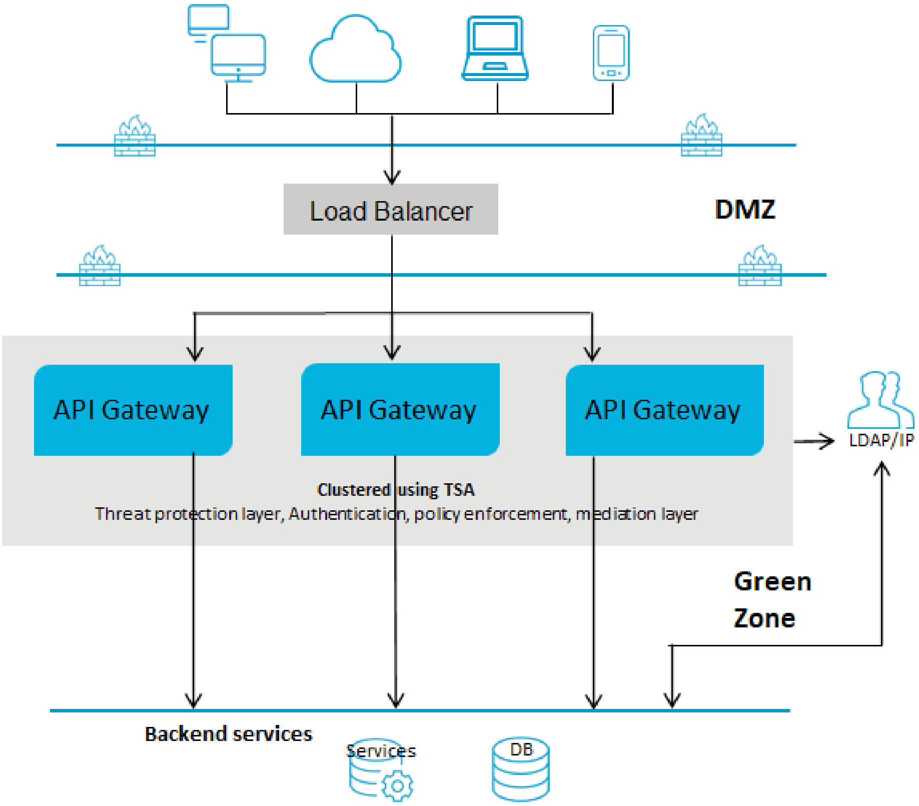
**Note:**
Because the API Gateway instance and the ESB service are in the same network, you can either directly invoke the ESB service or use the reverse invoke approach as required.

## Deployment scenario 4: Single gateway for non-webMethods customers

This setup consists of:

- One or more advanced edition API Gateways clustered and connected to a load balancer in DMZ. A single API Gateway is used for enforcing all policies or rules. You can have multiple instances of API Gateways connected through a load balancer and clustered using Terracotta Server Array.
- The native services reside in the green zone behind the firewall. As the native services are directly invoked, you must open the native service port to the gateway network.

The following diagram provides an architectural overview of the API Gateway deployment for non webMethods customers:



## API Gateway and DMZ connectivity properties

This section describes the properties that you can configure to maintain the connectivity between API Gateway and DMZ.

**watt.server.rg.internalregistration.timeout**

Specifies the time, in seconds, a connection on the internal server waits before closing the connection to the API Gateway server.

The default value of **watt.server.rg.internalregistration.timeout** setting is 0, so that the connection between internal server and API Gateway never times out. This is applicable when the network and connections are reliable.

If the network and connections are unreliable, and some times the connections breakdown, then you can set the **watt.server.rg.internalregistration.timeout** setting to a non-zero value, so that the connections times out.

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. If you enable **watt.server.rg.internalregistration.timeout** setting to XX seconds within which if API Gateway does not receive any requests from DMZ, then registration internal ports are auto refreshed.

Setting **watt.server.rg.internalregistration.timeout** to a value that is lower than **watt.net.socketpool.sweeperInterval** causes the internal server to close the connection to the API Gateway Server and re-establish a new connection regularly.

**watt.net.socketpool.sweeperInterval**

Specifies the frequency, in seconds, at which the socket pool sweeper executes. 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 executes 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.

# 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:

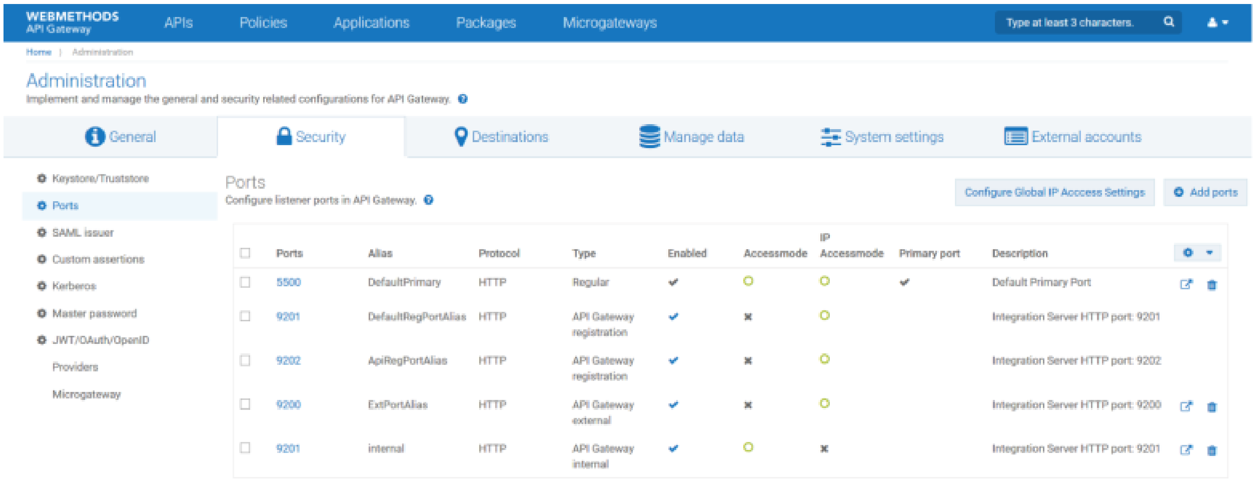
1. Navigate to **User menu** > **Administration** > **General** > **Extended settings**.
2. Click **Show and hide keys**.
3. Select the extended setting `forwardInternalAPIsRequest` and set it to `true`.

### 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)



Administration

Implement and manage the general and security related configurations for API Gateway.

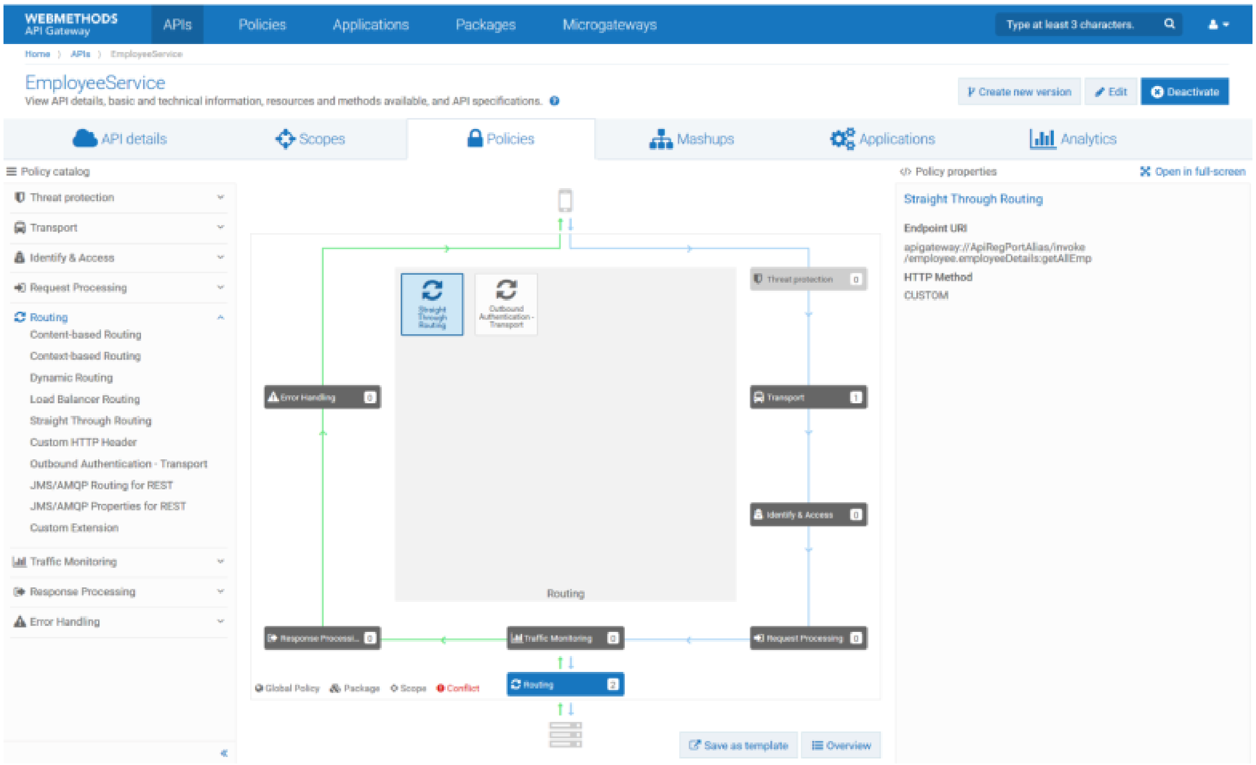
Ports

Configure listener ports in API Gateway.

| ☐ | 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.



EmployeeService

View API details, basic and technical information, resources and methods available, and API specifications.

Straight Through Routing

Endpoint URI

apigateway://ApiRegPortAlias/invoke
/employee.employeeDetails:getAllEmp

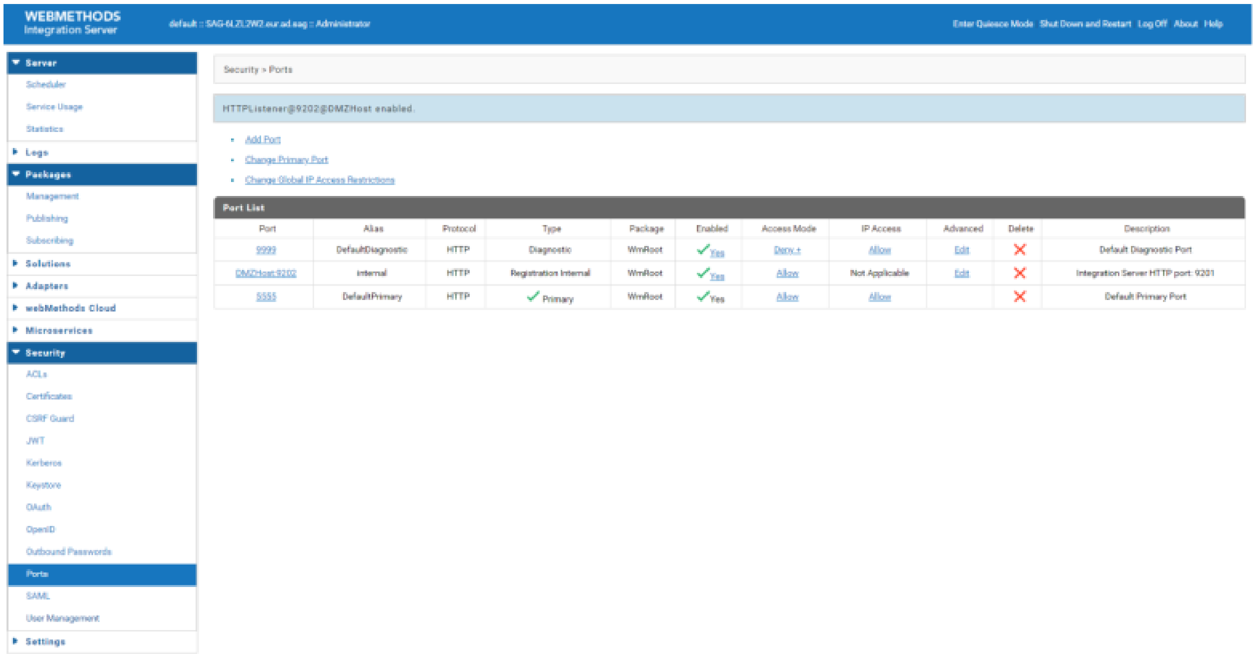
HTTP Method

CUSTOM

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.



HTTPListener@9202@DMZHost enabled.

- Add Port
- Change Primary Port
- Change Global IP Access Restrictions

Port List

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

# 2 API Gateway Data Store

## Overview of API Gateway Data Store

webMethods API Gateway Data Store is a data store for use only with webMethods API Gateway.

You can have only one API Gateway Data Store instance per Software AG installation. You can configure API Gateway Data Store as a single node storage, or you can combine multiple nodes to form a cluster.

You must install the following products to monitor and configure API Gateway Data Store:

- Software AG Command Central
- Software AG Platform Manager

## Administering API Data Store

This section describes the following administering tasks for API Data Store:

- “Starting, Stopping, and Restarting API Data Store” on page 28
- “Changing the API Data Store HTTP Port” on page 30
- “Changing the API Data Store TCP Port” on page 32
- “Configuring Custom API Data Store Properties” on page 35
- “Configuring Elasticsearch Properties” on page 36
- “Monitoring API Data Store” on page 37

### Starting, Stopping, and Restarting API Data Store

API Data store uses Elasticsearch 7.13.0. For details on the Elasticsearch versions that are compatible with different API Gateway versions, see “API Gateway, Elasticsearch, Kibana Compatibility Matrix” on page 119.

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 parametervm.max_map_count) by running the command `sysctl -a | fgrepvm.max_map_count`. If the value is less than 262144, log on as the rootuser and increase the value to at least 262144 by running `sysctl -wvm.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

1. In Command Central, navigate to **Environments > Instances > All > API Data Store > Configuration**.
2. Select **Ports** from the drop-down menu.
3. Click **http port** and specify the HTTP port number in the **Port Number** field.
4. Optionally, click **Test** to verify your configuration.
5. Save your changes.
6. Stop API Gateway instance, if it is running.
7. Update the Elasticsearch entry in the `config.properties` file located at *SAG_Installdir*/IntegrationServer/instances/*tenant_name*/packages/WmAPIGateway/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 88
   - Through Command Central. For details, see “Configuring Elasticsearch Connection Settings” on page 186.
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 32.

## 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

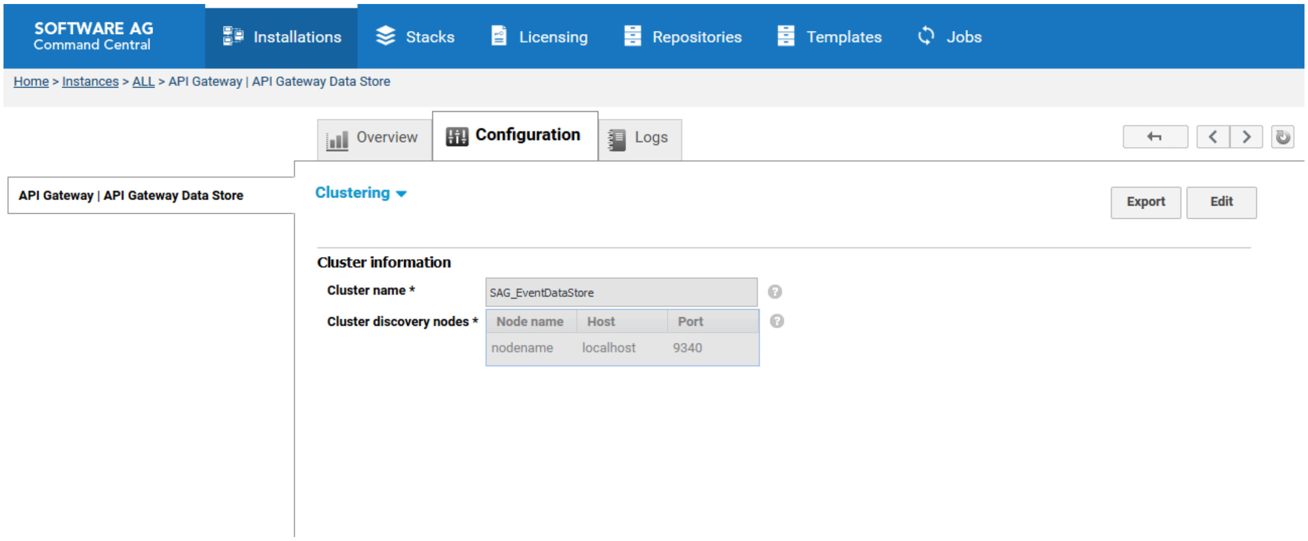
This section describes configuring an API Data Store cluster through Command Central. For details on configuring a cluster using the elasticsearch.yml file, see "API Data Store Cluster Configuration" on page 55.

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** | Required. The name of the cluster. All instances must have the same cluster name. |
| **Cluster Discovery Nodes** | Required. Click +, and then do the following to add host and port information for each API Data Store instance that is part of the cluster:<br>a. In the **Host** column, specify the host information for an API Data Store instance. The default host is `localhost`.<br>b. In the **Port** column, specify the port for an API Data Store instance. The default port is `9340`.<br>c. In the **Node name** column, specify the provide the node name details of the API Data Store instance. Ensure that this name matches with node.name property of the Elasticsearch instance. |



Home > Instances > ALL > API Gateway | API Gateway Data Store

API Gateway | API Gateway Data Store

Cluster information

Cluster name * SAG_EventDataStore

Cluster discovery nodes *

| Node name | Host | Port |
|---|---|---|
| nodename | 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}
```

## Monitoring API Data Store

Elasticsearch is an important component of the API Data Store. It is essential to monitor the health and various metrics of Elasticsearch to ensure smooth running of API Gateway. This section explains various metrics in Elasticsearch, which are essential and must be monitored.

To ensure optimal health and performance of Elasticsearch, Software AG recommends to perform the following:

- API Data Store Health
- System Health Check
- Application Health Check

### Elasticsearch Health Check

The liveliness and readiness probes monitor the health of Elasticsearch.

- **Readiness Probe**

  You can monitor the readiness of Elasticsearch by using the following REST API. This is a GET method.

  **HTTP://host:port/_cluster/health**

  If you get a 200 OK response, readiness test is successful. Software AG recommends that you perform the readiness test three times, with a time gap of 20 seconds between each attempt. If the result of the above endpoint does not return 200 OK even after a minute, the readiness probe fails. If the readiness probe fails, it implies that your cluster health is not good.

  If readiness probe fails, you can perform one of the following actions:

  - If you have installed API Gateway directly, check the Elasticsearch logs to find the status or exception.
  - If you have installed API Gateway through docker image or k8s, ensure that the existing pod is resolved or a new pod is created (automatically) and ready for serving the requests.

- **Liveliness Probe**

  As Elasticsearch works in a cluster-based environment, the liveliness probe's result is determined by the cluster health. To learn more about how to monitor cluster health, refer to the "Elasticsearch cluster health" on page 41 section.

### System Metrics

It is very important to measure the system health parameters such as CPU utilization, disk space, and memory utilization at a higher level. To ensure optimal health and performance of Elasticsearch, Software AG recommends to monitor the following parameters to get the basics of system health.

- Containers (you can see container metrics in
- CPU Health
- Disk
- Memory Utilization

## CPU Health

To ensure that CPU is not over utilized, Software AG recommends to monitor CPU health regularly. Software AG monitors CPU usage at two levels which are 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.

To retrieve the CPU utilization by the Elasticsearch pods, run the following command:

```
curl -X GET http://localhost:9240/_nodes/stats/process?pretty
```

To view the percentage CPU usage by an Elasticsearch pod, use the following JSON path expression:

`$.nodes.`*`nodeid`*`.process.cpu.percent`.

If a pod is using 80% of the CPU space for more than 15 minutes, check the process that has occupied highest amount of CPU, collect the thread dump and share it with the Software AG support team.

If a pod is using 90% of the CPU, look for Prometheus metrics **elasticsearch_os_cpu_percent** and **elasticsearch_process_cpu_percent**.

If **elasticsearch_os_cpu_percent** is more than 90%, restart the pod and check the readiness and liveliness of the pod.

If **elasticsearch_process_cpu_percent** is more than 90%, add a new node to the cluster. To learn more about how to add a new Elasticsearch node, see "Adding New Nodes to an Elasticsearch Cluster" on page 58.

> **Note:**
> The Prometheus metrics names can differ in your environment if you are using a different Prometheus exporter.

## Disk space

To ensure all nodes have enough disk space, Software AG recommends to monitor the disk space regularly.

To retrieve the disk space of the Elasticsearch nodes, run the following command:

```
curl -X GET http://localhost:9240/_nodes/stats/fs
```

This command lists the disk space available in all nodes.

For more information about Elasticsearch node statistics, see Elasticsearch documentation.

To view the disk space usage, use the following JSON path expression:

- **Total disk space**

  ```
  $.nodes..fs.total.total_in_bytes
  ```

- **Free disk space**

  ```
  $.nodes..fs.total.free_in_bytes
  ```

- **Available disk space**

  ```
  .nodes..fs.total.available_in_bytes
  ```

To know the configured disk-based shard allocations in Elasticsearch, run the following command. To learn more about disk-based shard allocations, see Elasticsearch documentation

```
curl -X GET http://localhost:9240/_cluster/settings?pretty
```

The shard allocation is based on the thresholds known as the Low, High, and Flood watermark.

To view different levels of watermark, use the following JSON path expression:

- **Low**

  ```
  $.persitent.cluster.routing.allocation.disk.watermark.low
  ```

  The default threshold for this level is 80%. Once the threshold is reached, Elasticsearch 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 ( average Disk Usage of the Elasticsearch cluster / standalone). If the result of this expression exceeds the defined threshold (80%), the disk is said to have reached the "Low" stage. If your disk usage has reached low, you can perform the following steps:

  - Query the transaction event index size and verify that the index size is above 525 GB (HA)/175 GB (single node). If its already breached, monitor the purge scripts are running and index size is decreasing.
  - Verify if the size of each transaction event index size is more or less equal to (range of 25 GB) the sum of used space. If this doesn't match, some other external items like increased logs size or heap dump are occupying a lot of space. Clear the logs and heap dump.
  - Repeat the above steps until the transaction event index size is less than 525 GB and the average disk usage of the cluster becomes less than 80%

- **High**

  ```
  $.persitent.cluster.routing.allocation.disk.watermark.high
  ```

  The default threshold for this level is 85%. Once the threshold is reached, Elasticsearch attempts to relocate shards away from a node whose disk usage is above 85%. You can calculate if the disk usage is low by using the expression ( average Disk Usage of the Elasticsearch cluster / standalone). If the result of this expression exceeds the defined threshold (85%), the disk is said to have reached the "High" stage. If your disk usage has reached high, you can perform the following steps:

  - Query the transaction event index size and verify that the index size is above 525 GB (HA)/175 GB (single node). If its already breached, monitor the purge scripts are running and index size is decreasing.
  - Verify if the size of each transaction event index size is more or less equal to (range of 25 GB) the sum of used space. If this doesn't match, some other external items like increased logs size or heap dump are occupying a lot of space. Clear the logs and heap dump.
  - Repeat the above steps until the transaction event index size is less than 525 GB and the average disk usage of the cluster becomes less than 85%

- **Flood**

  ```
  $.persitent.cluster.routing.allocation.disk.watermark.flood
  ```

The default threshold for this level is 90%. Once the threshold is reached, Elasticsearch enforces a read-only index block (index.blocks.read_only_allow_delete) on every index that has one or more shards allocated on the node that has at least one disk exceeding the flood stage. This is the last resort to prevent nodes from running out of disk space.

You can calculate if the disk usage is in flood stage, by using the expression ( average Disk Usage of the Elasticsearch cluster / standalone). If the result of this expression exceeds the defined threshold (90%), the disk is said to have reached the "Flood" stage. If your disk usage has reached the flood stage, you can perform the following steps:

- Monitor the purging of data and ensure the purging happens and the disk space occupancy gets reduced.
- If this situation is due to a peak in requests count & size, you can do follow-up actions like verifying with the customer about the reason for this sudden peak, ask the customer to compress payload data for transaction logging or not to store request or response payload, and so on.

To get information of specific metrics like fs, http, os, process, and so on, run the following command:

```
curl -X GET http://localhost:9240/_nodes/stats/metric
```

For more information about metrics, see Elasticsearch documentation.

### Memory Check

To retrieve the memory status utilized by the Elasticsearch pods, navigate to the following URL:

```
http://HOST:9240/_nodes/nodeid/stats/os
```

To get the nodeid in the above API, use the following:

```
http:URL/nodes?v&full_id=true&h=id,name,ip
```

This returns the node id, node name, and the node IP address.

You can now use the following JSON expression to get the percentage of memory that is free.

```
$.nodes.nodeid.os.mem.free_percent
```

If a pod is using 85% of the available memory, check the process in that pod that is consuming the highest amount of memory.

If a pod is using 90% of the available memory, check the process in that pod that is consuming the highest amount of memory, restart the pod and check the readiness and liveliness of the pod.

### Application Metrics

It is very important to check the information about all the processes that are running. To ensure optimal health and performance of Elasticsearch pod, Software AG recommends to monitor the following parameters to get the basics of application health.

- Index Size
- Cluster Health
- Number of Shards
- GC Monitoring

### Index Size

When data on a particular index exceeds a certain limit, it is essential to rollover and create a new index. In API Gateway, it is essential to monitor the indices 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. The naming convention of the new index is **gateway_default_analytics_transactionalEvents_YYYYMMDDHHMM**.

For information on creating a rollover, see "Creating Rollover of an Index" on page 251.

### Elasticsearch cluster health

To ensure optimal health and performance of API Data Store, Software AG recommends to monitor the Elasticsearch cluster health regularly.

To retrieve the Elasticsearch cluster health status, run the following command:

```
curl -X GET http://localhost:9240/_cluster/health?pretty
```

To check the cluster health status from the response, use the JSON path expression `$.status`.

To check the number of nodes in the cluster from the response, use the JSON path expression `$.number_of_nodes`.

The cluster health status is displayed based on the following color codes:

- **Green**. If the cluster health status is green, then the cluster is in good health. No action is needed to correct the cluster health. When the Elasticsearch is handling huge data, it takes some time to display the cluster health status.
- **Yellow**. If the cluster health status is yellow, identify the cause and rectify it. During this time Elasticsearch processes the requests for the index that is available. If there are unassigned

shards, then identify the unassigned shards, check the unallocation reason and resolve the issue.

  - Run the following command to retrieve the list of unassigned shards.

    ```
    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”
    ```

  - Run the following command to check the unallocated reason for specific shards.

    ```
    curl -X GET "http://localhost:9240/_cluster/allocation/
    explain" -d ‘{ "index" :"index name","primary" : "true|false","shard":
    "shardnumber"}’reason,help,s=index&v”
    ```

- **Red**. If the cluster health status is red, then the Elasticsearch nodes are down or not reachable or the Elasticsearch master is not discovered. If the number of nodes does not match the number of Elasticsearch nodes configured, identify the node that did not join the cluster and identify the root cause for the node to not join the cluster. Based on the root cause, you can perform one the following tasks to resolve the issue:

  - Identify if your Elasticsearch is down. If you Elasticsearch is down and not reachable, check the connectivity.

## Number of shards

To ensure proper allocation of shards to nodes, Software AG recommends to monitor the number of shards regularly.

To retrieve the number of shards on Elasticsearch, run the following command:

```
curl -X GET "http://localhost:9240/_cluster/health?pretty"
```

If the total number of active shards from the response exceeds the `heap space * nodes * 20` count then increase the heap space of Elasticsearch nodes or add a new Elasticsearch node. For more information on adding a new Elasticsearch node, see “Adding New Nodes to an Elasticsearch Cluster” on page 58.

Elasticsearch 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:

- Scale up the Elasticsearch node.
- If you are not able to scale up the Elasticsearch node, then increase the heap size as the last option. The heap space should not be more than half of system memory (RAM). Example, if the system memory is 16 GB you can allocate a maximum of 8 GB for Elasticsearch.

To increase the heap space, modify the parameters Xms2g and Xmx2g in the jvm.options file located at *SAG_Install_Directory*\InternalDataStore\config.

### Garbage Collection (GC) Monitoring

THE GC metric counts the number of seconds for which GC ran and the GC count. You must check GC run once every five minutes. The average GC run should not exceed one second. To check GC run time, you can view the quotient of (**elasticsearch_jvm_gc_collection_seconds_sum** / **elasticsearch_jvm_gc_collection_seconds_count**) metrics.

If the quotient of the above operation is more than 1 second, it implies that GC is taking longer time to run and this slows down the Elasticsearch request processing. You must collect the logs and get the mapping of API index and transaction index.

# Securing Communication with API Gateway Data Store

If you want to secure the API Gateway Data Store communications, you can use the ReadOnlyREST or Search Guard security plugins.

**Note:**
Starting version 10.7, the Search Guard security plugin is not shipped with API Gateway. Customers can download a security plugin and configure as per their requirement.

# 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 *Software AG Command Central Help*. |
| `sagcc update configuration data` | For general information about the command, see *Software AG Command Central Help*. |
| `sagcc get configuration instances` | For general information about the command, see *Software AG Command Central Help*. |
| `sagcc list configuration instances` | For general information about the command, see *Software AG Command Central Help*. |
| `sagcc get configuration types` | For general information about the command, see *Software AG Command Central Help*. |

| Commands | Additional Information |
|---|---|
| `sagcc list configuration types` | For general information about the command, see *Software AG Command Central Help*. |
| `sagcc exec configuration validation update` | For general information about the command, see *Software AG Command Central Help*. |
| `sagcc exec lifecycle` | For general information about the command, see *Software AG Command Central Help*. |
| `sagcc get monitoring` | For general information about the command, see *Software AG Command Central Help*. |

## 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.<br>**Note:**<br>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. |
| `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.

## Renaming Data Store Windows Service

You can rename API Gateway Data Store only if you have installed it as a windows service.

1. Stop the Data Store windows service.
2. Open command prompt.
3. To rename run the following commands in the *`SAG_Install_Directory\InternalDataStore\bin`* folder in the system where the API Gateway Data Store is installed:

   ```
   elasticsearch-service.bat remove current_service_name
   elasticsearch-service.bat install new_service_name
   ```

   For example,

   ```
   <SAG_Install_Directory>\InternalDataStore\bin>elasticsearch-service.bat remove
   datastore
   <SAG_Install_Directory>\InternalDataStore\bin>elasticsearch-service.bat install
   newstore
   ```

4. Restart the Data Store windows service.

# Troubleshooting Tips: API Data Store (Elasticsearch)

### Dangling Index issue in API Data Store cluster.

One of the reasons for the dangling index problem is that there are two or more API Gateway instances, which are used independently (standalone) for a while and then they are brought into a cluster. When this happens, both the API Data Store nodes try to update each other on the same indexes. As both the API Data Store nodes have the same indexes with different data, this leads to the dangling index error.

The following error message appears in any of the indexes:

[ WARN ] [o.e.g.DanglingIndicesState] [s1gp-igw01-gsb.sgi-idm.fednet.intra1566308686237] [[gateway_default_cache/rSspQXrrRKGdQCwS8KaYHg] ] can not be imported as a dangling index, as index with same name already exists in cluster metadata.

When this issue occurs, you can notice inconsistent data in the API Data Store nodes after every restart.

**Resolution**:

1. Create the API Data Store cluster first.
2. Start the API Gateway instances one after the other. That is, start an instance, wait for it to start, and start the next instance after the earlier one has started.

To resolve an existing dangling cluster, contact Software AG Global Support.

## 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 [BOf6SQe2SwyI93vi4RlBNQ] [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

## Unexpected lock file size - WARNINGs in the Data Store log lead to huge log file size (approximately 10 to 15 GBs).

The following error message appears:

Unexpected lock file size - Elasticsearch WARNINGs causing log file to grow to GB in size.

The elasticserch log files are growing very large, 10 to 15 Gig. These are the logs generated in the folder C:\SoftwareAG\EventDataStore\logs directory.

**Resolution**:

1. Open the `jvm.options` file located at `SAGInstallDir\internaldatastore\config\` and specify the following parameters to a higher heap size value. For example, `4 GB`. You can specify the size that you require.

   ```
   -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.

2. Go to the location, `SAGInstallDir\Eventdatastore\data\SAG_EventDataStore\nodes\0\indices` and check the size of the **write.lock** file. If the file size is not zero, delete the file. API Data Store recreates the file with zero as its size.

3. Restart API Data Store.

## 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`.

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. 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.

### API Gateway package is not accessible from Integration Server.

The following error message appears:

com.softwareag.apigateway.core.exceptions.DataStoreException:
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 this values are not 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`.

### When API Data Store is secured using Search Guard, the URL to access the API Data Store is generated with the http protocol.

The reason for this issue could be due to the certificates in nodes.

The following error message appears:

- sun.security.validator.ValidatorException: Extended key usage does not permit use for TLS client authentication.

**Resolution**:

- Ensure the following:
  - Node certificate contains both *serverauth* and *clientauth* in extended key usage.
  - Admin and client certificates contain only *clientauth* in extended key usage.

> **Note:**
> You can use a key explorer tool to export the certificates and view them.

### 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_]

I am getting the following error:

```
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.

I am getting the following issue

```
Exception: [WARN ][o.e.c.r.a.DiskThresholdMonitor] [localhost1568897216386] flood stage disk
watermark [95%] exceeded on
[BOf6SQe2SwyI93vi4RlBNQ][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**:

You can clean up the disk space by using the following CURL command:

```
curl -XPUT -H "Content-Type:
application/json"
http://localhost:9200/_all/_settings
 -d '{"index.blocks.read_only_allow_delete":
null}'
```

## My Elasticsearch server is not starting. I get a "bootstrap checks failed" error

I am getting the following error:

```
[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

You get the following error

```
[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.

## I see an error when Kibana fails to start

When I view the Analytics tab, Kibana does not start and displays the *Cannot connect to the Elasticsearch cluster currently configured for Kibana* error message.

**Resolution**:

Ensure that you have specified a valid elasticsearch hostname in the *elasticserach.host* property of the *kibana.yml* file located at `SAGInstallDir\profiles\IS_default\apigateway\dashboard\config`. Also ensure that the kibana and elasticsearch uses same version of OSS.

## 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
  }
  ```

### 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`SAGInstallDir\profiles\IS_default\configuration`:

- wrapper.java.initmemory
- wrapper.java.maxmemory

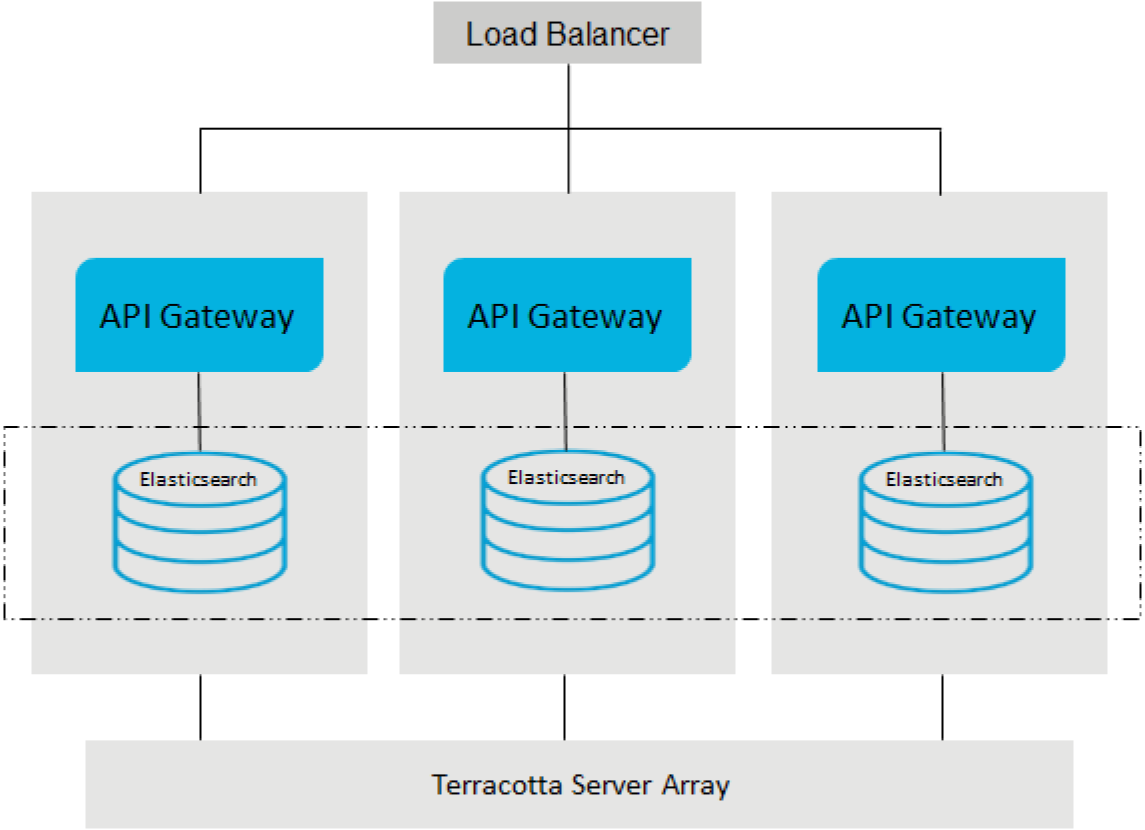
# 3 API Gateway Configuration

# API Gateway Cluster Configuration

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 software. For installation procedures for the product software, see *Installing webMethods Products On Premises*.

## Nodes and Clusters

API Gateway supports clustering to achieve horizontal scalability and reliability. The following figure illustrates an API Gateway cluster consisting of multiple API Gateway nodes.



Each API Gateway cluster node holds all the API Gateway components including UI, the API Gateway package running in webMethods Integration Server, and an API Gateway Data Store instance for storing assets. A load balancer distributes the incoming requests to the cluster nodes. The synchronization of the nodes is performed through a Terracotta server array and API Gateway Data Store clustering that is defined across the API Gateway Data Store instances.

**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.

The minimum requirements to achieve high availability in API Gateway are as follows:

- Two API Gateway instances.
- Three API Gateway Data Store instances.
- Two Terracotta Server instances (Active-Passive).

**Note:**

- Though only two API Gateway instances are sufficient, Software AG recommends the usage of three instances. If you use only two API Gateway instances, then you must have an additional API Gateway Data Store instance (Elasticsearch). The three Elasticsearch instances are required to form a proper Elasticsearch cluster to avoid split-brain scenario.
- If you have API Gateway Advanced Edition instances, clustering the API Gateway instances requires clustering of Elasticsearch and clustering of API Gateway nodes using Terracotta.
- If you have API Gateway Standard Edition instances, clustering the API Gateway instances does not require clustering using Terracotta. The API Gateway nodes just have to connect to Elasticsearch cluster.
- When you use one Terracotta server for multiple product clusters (for example, API Gateway cluster, Integration Server cluster) in parallel, provide unique names for each cluster in order to avoid conflicts.

## Configuring an API Gateway Cluster

Configuring an API Gateway cluster requires the following:

- Configuring API Gateway cluster
- Configuring API Data Store cluster
- Configuring Terracotta Server Array.

  **Note:**
  This configuration step is required if your API Gateway cluster uses TSA.
- Configuring load balancer
- Configuring ports

### 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 88.

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

### 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/7.13/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.tcp.port:9340
   path.repo:['SAG_root\InternalDataStore/archives']
   cluster.initial_master_nodes:["node1"]
   ```

   `discovery.seed_hosts`. You 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.

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/7.13/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.tcp.port:9340
   path.repo:['SAG_root\InternalDataStore/archives']
   ```

```
cluster.initial_master_nodes:["node1","node2","node3"]
```

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** from SAG_root/InternalDataStore/config/elasticsearch.yml from the system 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.tcp.port:9340
   path.repo:['SAG_root\InternalDataStore/archives']
   cluster.initial_master_nodes:["node4"]
   ```

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.tcp.port:9340
   path.repo:['SAG_root\InternalDataStore/archives']
   ```

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.

### 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.maxLogFileSize" 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>
```

### 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.

## API Gateway Availability and Health Status

You can monitor the availability and health status of API Gateway using the Availability REST API. The Availability API is used to report the overall health of the API Gateway.

The REST API is not deployed by default but can be defined by importing the Swagger file `APIGatewayAvailability.json` from the folder located at `SAG_Root/IntegrationServer/instances/default/packages/WmAPIGateway/resources/apigatewayservices`. For details, see the REST APIs section in *webMethods API Gateway User's Guide*.

You can check API Gateway health using the HTTP requests against `http://localhost:5555/gateway/availability`. This REST call also verifies the exposure and availability of the API Gateway REST API. You must have the **View administration configurations** privileges to invoke the Availability API to view the availability and health status of API Gateway.

Each health check request displays a `status` field as the first entry. The status can have the values `green`, `yellow` or `red` describing the overall status of the components to check. This means that when any of the components signals a problem, then the status is set to `red`. When the status is `green` and `yellow`, the request ends with HTTP 200, however when the status is `red`, then the request ends with HTTP 500.

The REST API provides the following resources and methods:

- **GET /gateway/availability/admin**

  The request retrieves the availability and health status of the API Gateway administration service (UI, Dashboards).

  Request: `GET http://localhost:5555/gateway/availability/admin`

  The overall admin status is assessed based on the UI ports (the port can be HTTP or HTTPS) status as follows:

  - When the HTTP and the HTTPS ports are accessible, the overall status is `green`.
  - When both ports are configured and they are inaccessible, the overall status is `red`.
  - When both ports are configured and one of the ports is inaccessible, the overall status is `yellow`.
  - When there is a SSL handshake failure while checking the HTTPS port, the overall status is `yellow`.

  The overall admin status is assessed based on the Kibana status as follows:

  - When Kibana's port is accessible, the overall status is `green`.
  - When Kibana's port is inaccessible, the overall status is `red`.
  - When Kibana's communication with Elasticsearch is not established, the overall status is `red`.

  A sample HTTP response looks as follows:

```
{
    "status": "green",
    "ui": {
        "https_port_9073": "ok",
        "http_port_9072" "ok",
        "status": "green",
        "response_time_ms": "727"
    },
    "kibana": {
        "status": {
            "overall": {
                "state": "green",
                "nickname": "Looking good"'
                "icon": "success",
                "uiColor": "secondary"
             }
         }
         "response_time_ms": "78"
    }
}
```

The sample HTTP response shows a green status as both the ports and Kibana are available.

- **GET /gateway/availability/engine**

  The request retrieves the availability and health status of the API Gateway to process API invocations and requests (Elasticsearch cluster, IS and Terracotta).

  The overall status is assessed based on the Elasticsearch status as follows:

  - When the internal status of Elasticsearch signals `green` or `yellow`, the overall status is `green`.
  - When the internal status of Elasticsearch signals `red`, the overall status is `red`.
  - When Elasticsearch port is inaccessible, the overall status is `red`.

  The overall status is assessed based on the IS status as follows:

  - When one of the resource types *memory*, *diskspace*, and *servicethread* reveals a resource problem, then the overall engine status is set to `yellow`.

  Request: `GET http://localhost:5555/gateway/availability/engine`

  A sample HTTP response looks as follows:

```
{
    "status": "green",
    "elasticsearch": {
        "cluster_name": "SAG_EventDataStore",
        "status": "green",
        "number_of_nodes": "3",
        "number_of_data_nodes": "3",
        "timed_out": "false",
        "active_shards": "236",
        "initializing_shards": "0",
        "unassigned_shards": "0",
        "task_max_waiting_in_queue_millis": "0",
        "port_9240": "ok",
        "response_time_ms": "29"
    },
    "is": {
        "status": "green",
        "diskspace": {
            "status": "up",
            "free": "8249233408",
            "inuse": "2476650496",
            "threshold": "1072588390",
            "total": "10725883904"
        },
        "memory": {
            "status": "up",
            "freemem": "252558496",
            "maxmem": "954728448",
            "threshold": "55679385",
            "totalmem": "556793856"
        },
        "servicethread": {
            "status": "up",
            "avail": "71",
            "inuse": "4",
            "max": "75",
            "threshold": "7"
        },
        "response_time_ms": "315"
    },
    "cluster": {
        "status": "green",
        "isClusterAware": "false",
        "message": "Non-Clustered node",
        "response_time_ms": "16"
    }
}
```

The overall status is `green` since all components work as expected.

- **GET /gateway/availability/externalServices**

The request retrieves the availability of external services accessed by API Gateway. The external services include destinations and external accounts. The checked external accounts include Service registries and Integration Servers.

The status field of externalServices displays the values `green` or `yellow`, if at least one of the destination resources is not available. In case of a problem, the error field displays the details of the problem encountered.

Request: `GET http://localhost:5555/gateway/availability/externalServices`

A sample HTTP response looks as follows:

```
{
    "status": "yellow",
    "destinations": [
        {
            "type": "centrasite",
            "name": "centrasite",
            "status": "yellow",
            "error": "Port 53307 not active",
            "response_time_ms": "1006"
        },
        {
            "type": "centrasite",
            "name": "centrasite_snmp",
            "status": "yellow",
            "error": "Port 8181 not active",
            "response_time_ms": "1005"
        },
        {
            "type": "api_portal",
            "name": "api_portal",
            "status": "not configured",
            "response_time_ms": "9"
        },
        {
            "type": "snmp",
            "name": "snmp",
            "status": "yellow",
            "error": "Port 8189 not active",
            "response_time_ms": "1004"
        },
        {
            "type": "email",
            "name": "email",
            "status": "green",
            "response_time_ms": "9"
        },
        {
            "type": "elasticsearch",
            "name": "elasticsearch",
            "status": "not configured",
            "response_time_ms": "0"
        }
    ],
    "external_accounts": [
        {
            "type": "service_registry",
            "name": "ServiceConsulDefault",
            "status": "green",
            "response_time_ms": "12"
        },
        {
            "type": "service_registry",
            "name": "EurekaDefault",
            "status": "yellow",
           "error": "Error: HttpResponse: 500 (Connect to http://daefermion3:9092
 failed): ",
            "response_time_ms": "1026"
        }
    ]
}
```

The sample response shows the status of all external services including those that are not configured. As the CentraSite destination is not properly configured, as shown in the sample response, this turns the overall status to yellow.

- **GET /gateway/availability/all**

  The request retrieves the availability of the administration service of the policy enforcement engine and of the external services accessed by API Gateway.

  Request: `GET http://localhost:5555/gateway/availability/all`

  A sample HTTP response looks as follows:

```
{
    "status": "green",
    "ui": {
        "https_port_9073": "ok",
        "http_port_9072" "ok",
        "status": "green",
        "response_time_ms": "727"
    },
    "kibana": {
        "status": {
            "overall": {
                "state": "green",
                "nickname": "Looking good"'
                "icon": "success",
                "uiColor": "secondary"
             }
         }
         "response_time_ms": "78"
    },
    "elasticsearch": {
        "cluster_name": "SAG_EventDataStore",
        "status": "green",
        "number_of_nodes": "3",
        "number_of_data_nodes": "3",
        "timed_out": "false",
        "active_shards": "236",
        "initializing_shards": "0",
        "unassigned_shards": "0",
        "task_max_waiting_in_queue_millis": "0",
        "port_9240": "ok",
        "response_time_ms": "7"
    },
    "is": {
        "status": "green",
        "diskspace": {
            "status": "up",
            "free": "8249327616",
            "inuse": "2476556288",
            "threshold": "1072588390",
            "total": "10725883904"
        },
        "memory": {
            "status": "up",
            "freemem": "232997664",
            "maxmem": "954728448",
            "threshold": "57094963",
            "totalmem": "570949632"
        },
        "servicethread": {
            "status": "up",
            "avail": "71",
            "inuse": "4",
            "max": "75",
            "threshold": "7"
        },
        "response_time_ms": "127"
    },
    "cluster": {
        "status": "green",
        "isClusterAware": "false",
        "message": "Non-Clustered node",
        "response_time_ms": "16"
    },
    "destinations": [
        {
            "type": "centrasite",
            "name": "centrasite",
            "status": "yellow",
            "error": "Port 53307 not active",
            "response_time_ms": "1006"
        },
        {
            "type": "centrasite",
            "name": "centrasite_snmp",
            "status": "yellow",
            "error": "Port 8181 not active",
            "response_time_ms": "1005"
        },
        {
            "type": "api_portal",
            "name": "api_portal",
            "status": "not configured",
            "response_time_ms": "9"
        },
        {
            "type": "snmp",
            "name": "snmp",
            "status": "yellow",
            "error": "Port 8189 not active",
            "response_time_ms": "1004"
        },
        {
            "type": "email",
            "name": "email",
            "status": "green",
            "response_time_ms": "9"
        },
        {
            "type": "elasticsearch",
            "name": "elasticsearch",
            "status": "not configured",
            "response_time_ms": "0"
        }
    ],
    "external_accounts": [
        {
            "type": "service_registry",
            "name": "ServiceConsulDefault",
            "status": "green",
            "response_time_ms": "12"
        },
        {
            "type": "service_registry",
            "name": "EurekaDefault",
            "status": "yellow",
           "error": "Error: HttpResponse: 500 (Connect to http://daefermion3:9092
 failed): ",
            "response_time_ms": "1026"
        }
    ]
}
```

**Note:**

- To perform the following API Gateway Availability REST calls you must have the *View Administration Configuration* privileges.
  - GET /gateway/availability/externalServices
  - GET /gateway/availability/all
- To perform the following API Gateway Availability REST calls you must be a valid API Gateway user.

- GET /gateway/availability/admin
- GET /gateway/availability/engine

You can use the existing health check request GET http://localhost:5555/rest/apigateway/health, without any authentication being set, to retrieve the health of API Gateway that monitors the availability and health status of Kubernetes and Docker containers . This returns a HTTP 200 response without additional data.

# Monitoring

Monitoring is an important part of maintaining the reliability, availability, and performance of API Gateway. You can monitor the resources of API Gateway, API Data Store, Terracotta, and Kibana with various metrics. In addition, you can identify and debug the problems easily at both system-level and application-level ensuring zero downtime of your application. A metric is a measurement related to health, capacity, or performance of a given resource such as CPU, Disk, and so on. You can easily check the consumption of your system resources and API Gateway resources using the metrics. This section provides information about the Software AG recommendations to monitor API Gateway and its various components.

## Monitoring API Gateway

To ensure optimal health and performance of API Gateway, Software AG recommends to perform a Health check regularly and monitor the following:

- System Metrics

- Application Metrics

## Health check

It is essential to monitor the health of API Gateway to check if the API Gateway server is healthy and ready to serve the requests. As part of Health check, Software AG recommends monitoring the health status of API Gateway by performing the Liveness Probe and Readiness Probe using the REST endpoints.

### Liveness Probe

To monitor the Liveness of API Gateway, that is to check the cluster health status of API Gateway to know if it is healthy and responding, use the following REST endpoint:

**GET /rest/apigateway/health/engine**

If you get a 200 OK response, Liveness check is successful. If the result of the above endpoint does not return 200 OK even after 8 minutes, the Liveness Probe fails. It implies that your cluster health is not good.

If the Liveness check fails, restart API Gateway.

**Note:**

Eight minutes failure threshold specified for the liveness check is not a standard value. It can vary according to your requirements. The rationale behind arriving at eight minutes as a failure threshold value is as follows:

The configuration proposed is based on the *Terracotta reconnect window*. When an active Terracotta server fails, the clients connected to the previous active server automatically switch to the new active server. However, these clients have a limited window of time called the client reconnect window to establish the connection with the new active server. Clients reconnecting after the reconnect window are rejected by the server and they must rejoin the cluster as new clients by establishing a new connection.

The Terracotta reconnect window proposed is six minutes, that is 360 seconds and two minutes is the additional time that Software AG recommends to ensure sufficient time to recoup the health of API Gateway. Hence, eight minutes is the failure threshold value for Terracotta server to perform a passive to active switchover within which the API Gateway clients must establish a connection with the active Terracotta server. Additionally, you must ensure the following when you change the recommended values based on your requirement:

- Terracotta failure threshold > reconnect window (360 seconds) - Terracotta failure threshold should be greater than the reconnect window time frame.
- API Gateway failure threshold >= Terracotta Failure threshold - API Gateway failure threshold value should not be less than the Terracotta failure threshold value. That is, API Gateway server should not be up before the Terracotta server.

Configure the reconnect window time frame in *client-reconnect-window* property in Terracotta config file, **tc-config.xml**. For more details about the Teracotta Server Array configuration, see "Terracotta Server Array Configuration" on page 59. For more information about Terracotta, see the Terracotta documentation located at `http://www.terracotta.org/`.

The Liveness probe has the following fields that controls the behavior of the liveness check. Following are the configurations based on 8 minute threshold value. You can change the configurations based on your requirement.

- initialDelaySeconds. *360*

  Specifies that you should wait for 360 seconds before the first probe is initiated.

- periodSeconds. *60*

  Specifies that liveness check is performed every 60 seconds.

- successThreshold. *1*

  Specifies the minimum consecutive successes for the probe to be considered successful and it is specified as 1. This shows that the API Gateway server is up and healthy.

- failureThreshold. *8*

  Specifies that you can mark that the server is unready when the probe fails for 8 times.

- timeoutInSeconds. *5*

Specifies that you can consider the probe result as failure when the probe does not return any response in 5 seconds.

### Readiness Probe

To monitor the Readiness of API Gateway, that is to check if API Gateway is ready to serve the requests, use the following REST endpoint:

#### GET /rest/apigateway/health

If you get a 200 OK response, Readiness check is successful. If the result of the above endpoint does not return 200 OK even after 3 minutes, the Readiness check fails.

If the Readiness check fails, you can consider that API Gateway is not ready to serve the requests.

The Readiness probe has the following fields that controls the behavior of the readiness check. Following are the recommended configurations. You can change the configurations based on your requirement.

- initialDelaySeconds. *130*

  Specifies that you should wait for 130 seconds before the first probe is initiated.

- periodSeconds. *60*

  Specifies that liveness check is performed every 60 seconds.

- successThreshold. *1*

  Specifies the minimum consecutive successes for the probe to be considered successful and it is specified as 1. This shows that the API Gateway server is up and healthy.

- failureThreshold. *3*

  Specifies that you can mark that the server is unready when the probe fails for 3 times.

- timeoutInSeconds. *5*

  Specifies that you can consider the probe result as failure when the probe does not return any response in 5 seconds.

## API Gateway Health endpoints

API Gateway provides the following key endpoints for monitoring the API Gateway health. Each health check request displays a status field as the first entry. The status can have the values green, yellow or red describing the overall status of the components to check. This means that when any of the components signals a problem, then the status is set to red. When the status is green and yellow, the request ends with HTTP 200, however when the status is red, then the request ends with HTTP 500. Following are the key endpoints for monitoring the health of API Gateway.

**GET /rest/apigateway/health/engine**

This endpoint retrieves the cluster health status of API Gateway to know if it is healthy and responding. Additionally, this endpoint provides details about the health of the resources of

API Gateway like CPU, diskspace and memory. When one of the resource types reveals a resource problem, then the overall engine status is set to yellow.

**GET /rest/apigateway/health/admin**
This endpoint retrieves the health status of the API Gateway administration service (UI, Dashboards). The overall admin status is assessed based on the UI ports (the port can be HTTP or HTTPS) and Kibana status.

The overall admin status is assessed based on the Kibana status as follows:

**GET /rest/apigateway/health/all**
This endpoint retrieves the availability of the administration service of the policy enforcement engine and of the external services accessed by API Gateway.

For details about the API, see the Rest APIs section in *webMethods API Gateway User's Guide* and the swagger file, APIGatewayAvailability.json, located at `SAGInstallDir/IntegrationServer/instances/default/packages/WmAPIGateway/resources/apigatewayservices`.

**Note:**

- To perform the following API Gateway REST calls, you must have the *View Administration Configuration* privileges.
  - GET /rest/apigateway/health/all
- To perform the following API Gateway REST calls you must be a valid API Gateway user.
  - GET rest/apigateway/health/admin
  - GET rest/apigateway/health/engine

## System Metrics

It is important to monitor the system health parameters such as CPU utilization, Disk utilization, and memory utilization. As part of Performance monitoring, Software AG recommends monitoring the following parameters for an optimal performance of API Gateway.

- CPU usage
- Disk usage
- Memory usage

To check the percentage of usage of the system parameters, use the following metrics endpoint. You can access the metrics using **GET /metrics** endpoint. When the endpoint is called, API Gateway gathers metrics and returns the data in a Prometheus format. If the metrics return an exceeded threshold value, you can 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. You can also use the following endpoint **GET /rest/apigateway/health/engine** to check the health of the resources of API Gateway like CPU, diskspace and memory. When one of the resource types reveals a resource problem, then the overall engine status is set to yellow in the response.

**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 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 take Thread Dump?" on page 78.

see "How Do I take Heap Dump?" on page 80.

## Monitor the CPU usage

To check the percentage of CPU usage of the Operating System, use the following metric:

### sag_is_server_proc_sys_percent

To check the percentage of CPU usage of the Integration Server JVM, use the following metric:

### sag_is_server_proc_cpu_percent

If the CPU usage is above the recommended threshold value, you can consider the severity as mentioned and perform the possible actions listed to identify the reason.

CPU usage: Above *80%* threshold, Severity: *ERROR*

CPU usage: Above *90%* threshold, Severity: *CRITICAL*

Following are the guidelines to identify the reason for higher CPU usage:

1. Identify the process that consumes the highest CPU.
2. Generate the thread dump.
3. Analyze the thread dump to identify the thread locks.
4. Analyze the logs of all the instances from the following locations:
    - `SAGInstallDirectory\IntegrationServer\instances\instance_name\logs`
    - `SAGInstallDirectory\profiles\IS_instance_name\logs`
    - `SAGInstallDirectory\InternalDataStore\logs`
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

## Monitor the Disk usage

To check the percentage of total available disk space in megabytes, use the following metric:

### sag_is_server_total_disk_mbytes

To check the percentage of used disk space in megabytes, use the following metric:

### sag_is_server_used_disk_mbytes

If the Disk usage is above the recommended threshold value, you can consider the severity as mentioned and perform the possible actions listed to identify the reason.

Disk usage: Above *80%* threshold, Severity: *ERROR*

Disk usage: Above *90%* threshold, Severity: *CRITICAL*

Following are the guidelines to identify the reason for higher disk usage:

1. The events that are archived in API Gateway are stored in the temp directory in the following location: `SAGInstallDirectory\profiles\IS_instance_name\workspace\temp`. 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 in the following locations and check the size of the log files that consumes more disk space to know if it is greater than the configured values.
   - `SAGInstallDirectory\IntegrationServer\instances\instance_name\logs`
   - `SAGInstallDirectory\profiles\IS_instance_name\logs`
   - `SAGInstallDirectory\InternalDataStore\logs`
3. Purge the events periodically to clean up the disk space for better performance of API Gateway.

   For more details about Purging, see "Archive and Purge using API" on page 272.

## Monitor the Memory usage

To check the percentage of total amount of physical memory available in megabytes, use the following metric:

### sag_is_server_total_memory_mbytes

To check the percentage of total amount of physical memory used in megabytes, use the following metric:

### sag_is_server_used_memory_mbytes

If the memory usage is above the recommended threshold value, you can consider the severity as mentioned and perform the possible actions listed to identify the reason.

Memory usage: Above *80%* threshold, Severity: *ERROR*

Memory usage: Above *90%* threshold, Severity: *CRITICAL*

Following are the guidelines to identify the reason for higher memory usage:

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 instances from the following locations and identify the file that consumes more memory:

  - `SAGInstallDirectory\IntegrationServer\instances\instance_name\logs`
  - `SAGInstallDirectory\profiles\IS_instance_name\logs`
  - `SAGInstallDirectory\InternalDataStore\logs`

5. Identify the server that has an issue and restart the server if required.
6. Perform the following actions after restarting the server:
   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.

## Application health check

To ensure optimal health and performance of API Gateway, Software AG recommends to monitor the following parameters to get the basics of application health.

- Threads statistics
- Service errors
- Memory usage of JVM
- HTTP or HTTPS requests
- Log monitoring

Software AG recommends monitoring the following metrics:

## Monitor the Threads statistics

To check the percentage of total number of threads used for service execution where the threads are obtained from the server thread pool, use the following metric:

### sag_is_service_threads

If the threads usage is above the recommended threshold value, you can consider the severity as mentioned and perform the possible actions listed to identify the reason.

Threads usage: Above *80%* threshold, Severity: *ERROR*

Threads usage: Above *90%* threshold, Severity: *CRITICAL*

Following are the guidelines to identify the reason for higher threads usage:

1. Identify the process that consumes the highest number of threads.

2. Generate the thread dump.
3. Analyze the thread dump to identify the thread locks.
4. Analyze the logs of all the instances from the following locations:
   - `SAGInstallDirectory\IntegrationServer\instances\instance_name\logs`
   - `SAGInstallDirectory\profiles\IS_instance_name\logs`
   - `SAGInstallDirectory\InternalDataStore\logs`

## Monitor the Service errors

To check the number of service errors encountered, which includes any service that gets an exception of any kind, use the following metric:

### sag_is_number_service_errors

If service errors are encountered, you can consider the severity as *ERROR*.

Following are the guidelines to identify the reason for service errors:

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 `SAGInstallDirectory\IntegrationServer\instances\instance_name\logs\server.log`.

## Monitor the Memory usage of JVM

To check the percentage of total used memory of JVM, use the following metric:

### sag_is_used_memory_bytes

If the memory usage is above the recommended threshold value, you can consider the severity as mentioned and perform the possible actions listed to identify the reason.

Memory usage: Above *80%* threshold, Severity: *ERROR*

Memory usage: Above *90%* threshold, Severity: *CRITICAL*

Following are the guidelines to identify the reason for higher memory usage of JVM:

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 instances from the following locations:
   - `SAGInstallDirectory\IntegrationServer\instances\instance_name\logs`
   - `SAGInstallDirectory\profiles\IS_instance_name\logs`

- `SAGInstallDirectory\InternalDataStore\logs`

4. Identify the server that has an issue and restart the server if required.
5. Perform the following actions after restarting the server:
    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 Elasticsearch connectivity with API Gateway server.
    e. Check the Terracotta client logs for errors in Terracotta communication for a cluster set-up.

## Monitor the HTTP or HTTPS requests

To check the percentage of total number of HTTP or HTTPS requests since the last statistics poll, use the following metric:

### sag_is_http_requests

The statistics poll interval is controlled by the watt.server.stats.pollTime server configuration parameter and the default interval is 60 seconds.

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, you can consider the severity as *ERROR*.

For more details about the API Gateway metrics , see *Developing Microservices with webMethods Microservices Runtime*.

## Log monitoring

It is essential to monitor the logs regularly. Perform the following actions to monitor the logs storage:

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.
4. The log files of all the instances are located at
    - `SAGInstallDirectory\IntegrationServer\instances\instance_name\logs`
    - `SAGInstallDirectory\profiles\IS_instance_name\logs`
    - `SAGInstallDirectory\InternalDataStore\logs`

To monitor the logs in different levels, check the availability of logs in FATAL level, ERROR or WARNING level.

### Container Monitoring Metrics

If you have installed API Gateway through Docker or Kubernetes, Software AG recommends monitoring the following **Kubernetes** metrics to check if the container is healthy.

When following events occur:

**PodNotReady**. If the status of the pod is not ready for more than 10 minutes,

**PodRestarting**. If the application inside the pod is not up in 1 minute,

**PodCrashLooping**. If the API Gateway pod is restarting continuously for 15 minutes,

you can consider the severity as *CRITICAL* and perform the following actions to identify the problem:

- Check the cluster status of API Gateway using the following REST endpoint: **GET /rest/apigateway/health/engine** to know if API Gateway and its components are healthy and are in a cluster mode.
- Check the possible cause for the pod restart, if it is due to the pod reallocation, node auto scaling and so on.
- Check the node pool resource availability.
- Check the previous logs of the pod for any exception.
- Check the pod events to find the status of the pod.
- Check the Terracotta client logs for errors in Terracotta communication to know if the tenant is in cluster mode.

**NodeNotReady**. If the status of the new node is *not* ready in Kubernetes cluster for more than 15 minutes, you can consider the severity as *CRITICAL*. Perform the following actions to identify the problem:

- Check the settings of Autoscale.
- Check the logs for the provisioning of the new node.
- Check if there is any issue with the provisioning of the new pod.
- Ensure that the status of the node is ready.
- Ensure that the pod reallocation is completed.

**DeploymentReplicasMismatch**. If there is any mismatch with the replicas, that is, If the pods replicas count does not match with the pods that are in a ready state for more than 10 minutes, you can consider the severity as *CRITICAL*. Perform the following actions to identify the problem:

- If replicas mismatch, Kubernetes spawns a new pod and checks if the pod is stuck in any state (init, crash loop back)

- If the pod is stuck in any state, Kubernetes deletes the pod and ensures that a new and a healthy pod is created.
- Check the pod events to find the status of the pod, for errors.
- Check the previous logs of the pod for any exception.
- Check the cluster status of API Gateway using the following REST endpoint: **GET /rest/apigateway/health/engine** to know if API Gateway and its components are healthy and are in a cluster mode.
- Check the node pool resource availability.
- Check the status of the new node if it is in a ready state.
- Check if there is any issue with the provisioning of the new pod.

## Pod restart procedure

For any reason, if the pod is restarted, you must check the following to verify the health of the new pod.

- Check for the readiness of the pod.
- Check the cluster status of API Gateway using the following REST endpoint: **GET /rest/apigateway/health/engine** to know if API Gateway and its components are healthy and are in a cluster mode.
- Check the possible cause for the pod restart, if it is due to the pod reallocation, node auto scaling and so on.
- Check the previous logs of the pod for any exception.
- Check the pod events to find the reason for the restart.
- Check the Terracotta client logs for errors in Terracotta communication, if the tenant is in cluster mode.

### How Do I take 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 take 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.

**Pre-requisites:**

- Java 7 version and above is considered for taking dumps.
- Check the Process ID (PID) of the JAVA application :

  ```
  Linux: Run the following command ps -ef | grep java
  ```

```
Windows: JAVA PID is available in the Task Manager
```

### Take Thread Dump using jstack

jstack is an effective command line tool to capture thread dumps. The jstack tool is located at *`JRE or JDK_HOME`*`\bin` folder.

Run the following command to capture thread dump:

```
jstack -l  pid > file_path_and_file_name
```

where:

**pid**. Process Id of the application, whose thread dump should be captured.

**file_path**. File path to where the thread dump has to be captured.

**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 can be used to take 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
> ```

### Take Thread Dump using jVisualVM

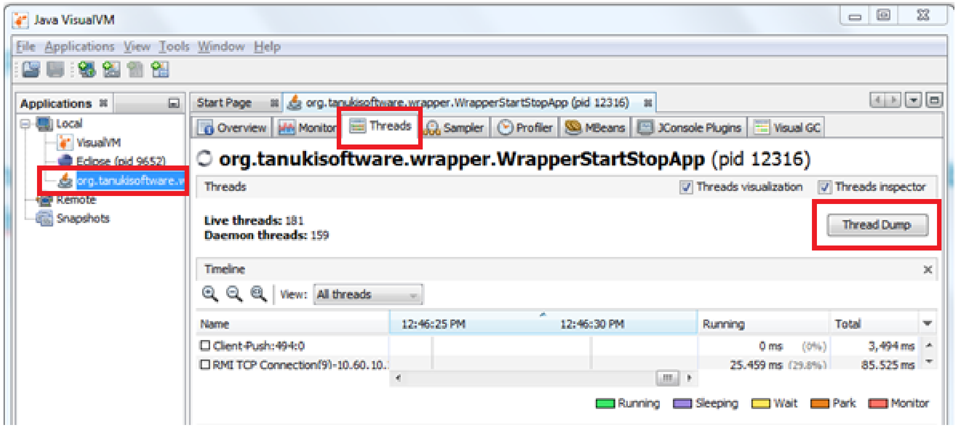
Java VisualVM is a graphical user interface tool that provides detailed information about the applications while they are running on a specified Java Virtual Machine (JVM). jVisualVm is located at *`JDK_HOME`*`\bin\jvisualvm.exe`

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 capture 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
```

## Take Thread Dump using JCMD

The jstack tool is located at *`JRE or JDK_HOME`*`\bin` folder. To take thread dump using JCMD:

Run the following JCMD command to generate thread dump:

```
jcmd <pid> Thread.print > file_path_and_file_name
```

where:

**pid**. Process Id of the application, whose thread dump should be captured.

**file_path**. File path to where the thread dump has to be captured.

**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 take 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 take a heap dump. This section explains few platforms from where you can take a heap dump. You can choose any platform according to your requirement.

**Pre-requisites:**

- Java 7 version and above is considered for taking dumps.
- Check the Process ID (PID) of the JAVA application :

  ```
  Linux: Run the following command ps -ef | grep java
  Windows: JAVA PID is available in the Task Manager
  ```

### Take Heap Dump using jmap

The jmap tool is located at `JRE or JDK_HOME\bin` folder. jmap tool generates heap dumps into a specified file location. To take a heap dump:

Run the following command:

```
jmap -dump:format=b,file=file_path_and_file_name.bin PID
```

where:

**pid**. Process Id of the application, whose heap dump should be captured.

**file_path**. File path to where the heap dump has to be captured.

**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.

### Take Heap Dump using jVisualVM

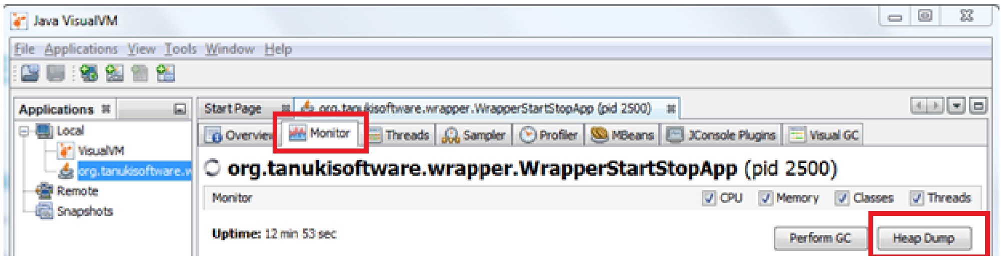
Java VisualVM is a graphical user interface tool that provides detailed information about the applications while they are running on a specified Java Virtual Machine (JVM). jVisualVm is located at `JDK_HOME\bin\jvisualvm.exe`

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:**



**Note:**
This tool also has the capability to capture 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
```

## HeapDumpOnOutOfMemoryError

When an application experiences java.lang.OutOfMemoryError, it is ideal to capture 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 take the heap dump whenever Out Of Memory (OOM) exception occurs in the application.

```
-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=file_path_and_file_name
```

where file_path_and_file_name is the file name and the location of the file to where the heap dump has to be captured.

## Monitoring Terracotta

To ensure optimal health and performance of Terracotta server, Software AG recommends to perform a Health check regularly and monitor the System Metrics.

## Health check

It is essential to monitor the health of Terracotta server to check if the Terracottaserver is healthy and ready to serve the requests. As part of Health check, Software AG recommends monitoring the health status of Terracotta by performing the Liveness Probe and Readiness Probe using the following script. Software AG recommends the following configurations for the Liveness and Readiness Probes.

### Liveness Probe

To monitor the Liveness of Terracotta server, that is to check the cluster health status of Terracotta to know if it is healthy and responding, 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, 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
```

If you get a 200 OK response, Liveness check is successful. If the result of the above endpoint does not return 200 OK even after 8 minutes, it implies that the Liveness check failed.

If the Liveness check fails, restart Terracotta server.

**Note:**
The configuration proposed is based on the Terracotta reconnect window. The Terracotta client reconnect window is six minutes and two minutes is the time frame for the Terracotta passive server to take an active role. Hence, eight minutes is the failure threshold value for Terracotta to perform a passive to active switchover within which the API Gateway clients must establish a connection with the active Terracotta server. This time window ensures sufficient time to recoup the health of API Gateway. For more details, see Liveness Probe section under "Health check" on page 68 in API Gateway Monitoring section.

This reconnect window time frame can be configured in the *client-reconnect-window* property in Terracotta config file, **tc-config.xml**. For more details about the Teracotta Server Array configuration, see "Terracotta Server Array Configuration" on page 59. For more information about Terracotta, see the Terracotta documentation located at `http://www.terracotta.org/`.

The Liveness probe has the following fields that controls the behavior of the liveness check. Following are the configurations for a 8 minute threshold value. You can change the configurations based on your requirement.

- initialDelaySeconds. *30*

  Specifies that you should wait for 30 seconds before the first probe is initiated.

- periodSeconds. *60*

  Specifies that liveness check is performed every 60 seconds.

- successThreshold. *1*

Specifies the minimum consecutive successes for the probe to be considered successful and it is specified as 1. This shows that the Terracotta server is up and healthy.

- failureThreshold. *8*

  Specifies that you can mark that the server is unready when the probe fails for 8 times.

- timeoutInSeconds. *5*

  Specifies that you can consider the probe result as failure when the probe does not return any response in 5 seconds.

### Readiness Probe

To monitor the Readiness of Terracotta server, that is to check if Terracotta is ready to serve the requests, use the same script that is mentioned for the liveness check and monitor the readiness with the same condition.

If you get a 200 OK response, Readiness check is successful. If the result of the above endpoint does not return 200 OK even after 2 minutes, the Readiness check fails.

If the Readiness check fails, you can consider that the Terracotta server is not ready to serve the requests. The proposed values for the fields in the readiness check are the same as the liveness check except for the failure threshold to be 2 minutes.

## System Metrics

It is important to monitor the system health parameters such as CPU utilization, Disk utilization, and memory utilization. As part of Performance monitoring, Software AG recommends monitoring the following parameters for an optimal performance of Terracotta server:

- CPU usage
- Disk usage
- Memory usage

You can monitor the system parameters using the following metrics. If the metrics return an exceeded threshold value, you can 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 Terracotta server. 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 “How Do I take Thread Dump?” on page 78.

see “How Do I take Heap Dump?” on page 80.

### Monitor the CPU usage

If the CPU usage of the system is above the recommended threshold value, you can consider the severity as mentioned and perform the possible actions listed to identify the reason.

CPU usage: Above *80%* threshold for 15 minutes continuously, Severity: *WARNING*

CPU usage: Above *90%* threshold for 15 minutes continuously, Severity: *CRITICAL*

Following are the guidelines to identify the reason for higher CPU usage:

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 `SAGInstallDirectory/Terracotta/server/bin/server-stat.sh`
6. Check if the API Gateway clients can establish the connection to Terracotta cluster using the following REST endpoint **GET /rest/apigateway/health/engine**.

### Monitor the Disk usage

If the Disk usage of the system shows a higher value, rotate logs based on a fixed size and fix the number of rotated files to be persisted

### Monitor the Memory usage

If the Memory usage is above the recommended threshold value, you can consider the severity as mentioned and perform the possible actions listed to identify the reason.

Memory usage: Above *80%* threshold, Severity: *WARNING*

Memory usage: Above *90%* threshold, Severity: *CRITICAL*

Following are the guidelines to identify the reason for higher memory usage:

- 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 `SAGInstallDirectory/Terracotta/server/bin/server-stat.sh`

- Check if the API Gateway clients can establish the connection to Terracotta cluster using the following REST endpoint **GET /rest/apigateway/health/engine**

### Container Monitoring Metrics

If you have installed Terracotta through Docker or Kubernetes, Software AG recommends monitoring the following **Kubernetes** metrics to check if the container is healthy.

When following events occur:

- **PodNotReady**. If the status of the pod is *not* ready for more then 10 minutes.
- **StatefulSetReplicasMismatch**. If the Statefulset replicas mismatch is longer than 5 minutes.

you can consider the severity as *CRITICAL* and perform the following actions to identify the problem:

- Check the console logs of the pod to find the status for any exception.
- 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 *not* up in 1 minute, you can consider the severity as *CRITICAL*. Kubernetes creates a new pod to maintain the availability.

perform the following actions to identify the problem:

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, you must 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 liveliness checks returns an unhealthy status and an appropriate action that is defined for the unhealthy pod is performed.

## Troubleshooting Tips: API Gateway Cluster Environment

### I see API Gateway UI in a cluster responding slowly.

One of the reasons for this issue could be firewall protection in one or more cluster nodes. Requests coming into a cluster are processed in a round-robin fashion. When a request comes in, it is directed to the first node as per your configuration. If the node is firewall-protected, the request is forwarded to the next node only after the specified wait time lapses. For example, consider that the wait time is 30 seconds, the request that is not processed by node 1 is forwarded to node 2 only after 30 seconds of waiting for node 1 to respond. Hence, when there are one or more firewall-protected nodes in a cluster, then there could be a delay in processing the requests.

This could occur when one of the nodes gives slow response or it is not accessible by the other nodes.

**Resolution**

- Ensure that there is no firewall in the nodes that blocks the communication between the nodes.

  **Note:**
  If there is any other reason for this issue, contact Software AG Global Support.

### I see that i can not bind API Gateway Ports to a specific IP address in a cluster setup

I am unable to bind API Gateway port and Kibana port to a specific IP address in a cluster setup.

**Resolution**:

To bind API Gateway ports to a specific IP address, set the values of the corresponding properties of the following components:

**Kibana**

Set the IP address in server.host property in kibana.yml file located at `SAGInstallDirectory\profiles\IS_instance_name\apigateway\dashboard\config`. or `Kibana_InstallDirectory\config`.

For API Gateway to connect to Kibana, go to **API Gateway UI > Administration > System settings > Dashboard settings** and set the same host details.

**API Data Store**

Set the IP address in pg.gateway.elasticsearch.hosts property in config.propertes file located at `SAGInstallDirectory\IntegrationServer\instances\instance_name\Packages\WmAPIGateway\config\resources\elasticsearch`.

**UI WebApp**

Set the desired port value in port property in com.softwareag.catalina.connector.https.pid-apigateway.properties file located at `SAGInstallDirectory\profiles\IS_instance_name\configuration`.

### I see that API Gateway fails to start in a cluster setup

The following error message appears when API Gateway fails to start in a cluster setup:

Elasticsearch is not ready to serve the requests.

The error message appears only when the API Data Store cluster health status is not green or yellow. This means that one of the API Data Store node is not started properly or errors occurred in cluster. The package initialization might fail due to this.

**Resolution**:

Check the cluster health status of API Gateway using the following REST endpoint **GET /rest/apigateway/health/engine** to identify the problem. If the cluster health status is red, it means that errors occurred in cluster or one of the API Data store node is not started properly. Errors might occur in API Data Store when there is a problem with its configurations. For more information about the cluster health status and its configurations, see "Monitoring API Gateway " on page 68, and "API Gateway Cluster Configuration" on page 55.

## 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:
    .....
  kibana:
    .....
  filebeat:
   ......
  cluster:
   ......
  users:
   ......
  masterpassword:
   ......
  ui:
   ......
  alias:
   ......
```

## Elasticsearch Configuration

**Note:**
Install and run Elasticsearch, version 7.13.0, 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.
- `clientHttpResponseSize`. Maximum Response payload size in MB.

A sample Elasticsearch configuration is as follows:

```
apigw:
  elasticsearch:
    tenantId: apigateway
    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
    clientHttpResponseSize: 1001231
```

## Kibana Configuration

**Note:**
Install Kibana, version 7.13.0, 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 .

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.
- `ignite.communicationPort`. The port that is used to communicate between distributed caches. Each server opens this port to receive information and distribute distribute cache entries to other cluster members.
- `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`.

  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 as well as Elasticsearch 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: daeirnd33974:9510
    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`<br>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 property specifies the time, in milliseconds, after which the connection times out. |

| Parameters | Description |
|---|---|
| This parameter is applicable to HTTP. | |
| `server`<br>This parameter is applicable to HTTP and HTTPS. | This property overrides the server header for the HTTP response. |
| `maxSpareThreads`<br>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`<br>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`<br>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`<br>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. |
| `acceptCount`<br>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`<br>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`<br>This parameter is applicable to HTTP and HTTPS. | Set to `true` if you want calls to `request.getRemoteHost()` to perform DNS lookup in order to return the actual host name of the remote client. Set to `false` to skip the DNS lookup and return the IP address in string form instead (thereby improving performance). |
| `enabled`<br>This parameter is applicable to HTTP and HTTPS. | Enable or disable API Gateway web-app port. |
| `alias`<br>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. |

| **Parameters** | **Description** |
|---|---|
| `sslProtocol`<br>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 `TLS`. |
| `sslEnabled`<br>This parameter is applicable to HTTPS. | Use this attribute to enable SSL traffic on a connector. The default value is `false`. To turn on SSL handshake or encryption or decryption on a connector set this value to `true`.When turning this value to `true`, set the scheme and the secure attributes to pass the correct `request.getScheme()` and `request.isSecure()` values to the servlets. |
| `keystoreType`<br>This parameter is applicable to HTTPS. | The keystore file type to use for the server certificate. If not specified, the default value is `JKS`. |
| `keystoreFile`<br>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 `.keystore` in the operating system home directory of the user that is running Tomcat. If the `keystoreType` doesn't need a file use " " (empty string) for this parameter. |
| `keystorePass`<br>This parameter is applicable to HTTPS. | The password used to access the specified keystore file. |
| `scheme`<br>This parameter is applicable to HTTPS. | Set this attribute to the name of the protocol you wish to have returned by calls to `request.getScheme()`. |
| `secure`<br>This parameter is applicable to HTTPS. | Set this attribute to `true` if you wish to have calls to `request.isSecure()` to return `true` 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: "xxy"
      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: "GuessIt"
        domain: "EUR"
```

**`isConfigurationAlias` configuration:**

```
apigw:
  aliases:
    myIsAlias:
      type: "isConfigurationAlias"
      url: "http://localhost:5555"
      username: "Administrator"
      password: "mypass"
      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"
        cientSecret: "secet"
        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"
```

The properties of the supported aliases are as follows:

| Type | Parameters |
|---|---|
| `simple` | `value`. Value of the simple alias. |
| `endpoint` | ■ `endPointURI`. The default URI or components of the URI such as service name.<br>■ `connectionTimeout`. Time interval (in seconds) after which a connection attempt times out.<br>■ `readTimeout`. Time interval (in seconds) after which a socket read attempt times out.<br>■ `suspendDurationOnFailure`. Time to suspend the request upon a failure.<br>■ `optimizationTechnique`. Type of optimization technique used for SOAP messages. |

| Type | Parameters |
|---|---|
| | ■ `passSecurityHeaders`. Boolean value whether to pass security headers or not.<br>■ `keystoreAlias`. Keystore alias name that is used for the signing or encryption.<br>■ `keyAlias`. Key alias in the particular keyStore .<br>■ `truststoreAlias`. Truststore alias name to validate the server certificate. |
| `httpTransportSecurityAlias` | ■ `authType`. The type of authentication you want to use while communicating with the native API.<br>■ The authentication types supported are: `HTTP_BASIC`, `NTLM`, `OAUTH2`, `KERBEROS`, `JWT`, `ALIAS`, and `REMOVE_INCOMING_HTTP_HEADERS`.<br>■ `authMode`. Authentication mode to use while communicating with the native API.<br>■ The authentication modes supported are: `NEW`, `INCOMING_HTTP_BASIC_AUTH`, `INCOMING_WSS_USER`, `INCOMING_X509`, `DELEGATE_INCOMING`, `INCOMING_OAUTH_TOKEN`, `INCOMING_JWT`, `TRANSPARENT`, and `INCOMING_KERBEROS`.<br>■ `httpAuthCredentials`. Credentials to use for HTTP authentication. The authentication credentials supported are:<br>■ `userName`. Specify a username to access the native API.<br>■ `password`. Specify a password to access the native API.<br>■ `domain`. Specify a domain to access the native API.<br>■ `kerberosCredentials`. Credentials to use for Kerberos authentication. The authentication credentials supported are:<br>■ `clientPrincipal`. A unique identity to which Kerberos can assign tickets.<br>■ `clientPassword`. Password for the client principal.<br>■ `servicePrincipal`. A unique identifier of a service instance. |

| Type | Parameters |
|---|---|
| | ■ `servicePrincipalNameForm`. The format in which you want to specify the principal name of the service that is registered with the principal database. `servicePrincipalNameForm` value can be `hostbased` or `username`.<br>■ `requestDelegateToken`. Boolean value whether the token needs to be delegated or not.<br>■ `oauth2Token`. OAuth2 token to use for authentication. |
| `transformationAlias` | ■ `fileName`. Name of the file.<br>■ `content`. Content of the file. |
| `serviceRegistryAlias` | ■ `endpointURI`. Endpoint to use to communicate with the service registry.<br>■ `heartBeatInterval`. API Gateway pings the service registry on the configured interval for every API.<br>■ `username`. Username to use in the basic authentication when communicating with the service registry.<br>■ `password`. Password to use in the basic authentication when communicating with the service registry.<br>■ `keystoreAlias`. A keystore is a repository of private key. This keystore contains the private key to use for the SSL communication with the service registry.<br>■ `keyAlias`. The key alias is the private key to use for signing when using SSL communication with the service registry.<br>■ `trustStoreAlias`. 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.<br>■ `customHeaders`. Custom headers to send while communicating with the service registry.<br>■ `discoveryInfo`. Contains information like resource path and HTTP method to use while discovering a service in service registry.<br>■ `registrationInfo`. Contains information like resource path and HTTP method to use while registering a service in service registry. |

| Type | Parameters |
|---|---|
| | - `deRegistrationInfo`. Contains information like resource path and HTTP method to use while de-registering a service from service registry.<br>- `serviceRegistryType`. Contains the information about the type of service registry.<br>- `connectionTimeout`. The time interval (in seconds) after which a connection attempt times out while communicating with service registry.<br>- `readTimeout`. The time interval (in seconds) after which a socket read attempt times out while communicating with service registry. |
| `awsConfigurationAlias` | - `region`. The configured AWS instance region detail.<br>- `accessKey`. The access key ID for the AWS instance. This is used to sign the requests.<br>- `secretKey`. The secret access key for the AWS instance. This is used to sign the requests. |
| `samlIssuerAlias` | - `issuerCommunicationMode`. Mode of communication to the STS.<br>- `issuerPolicy`. The webMethods Integration Server service name.<br>- `issuerAuthScheme`. The authentication type to use for communicating to STS.<br>- `issuerAuthMode`. Mode of communication to STS.<br>- `wssCredentials`. Credentials for the WSS Username token.<br>- `kerberosCredentials`. Credentials for the Kerberos token.<br>- `endpoint`. The endpoint URI of the STS.<br>- `samlVersion`. SAML version used for authentication.<br>- `wsTrustVersion`. WS-Trust version that API Gateway must use to send the RST to the SAML issuer.<br>- `appliesTo`. Specify the scope for which this security token is required.<br>- `extendedParameters`. Extensions to the `wst:RequestSecurityToken` element for requesting |

| Type | Parameters |
|---|---|
| | specific types of keys, algorithms, or key and algorithms, as specified by a given policy in the return tokens.<br>■ `signAndEncryptConfig`. Private and public keys to use signature and encryption. |
| `webmethodsAlias` | ■ `serviceName`. The webMethods Integration Server service name.<br>■ `runAsUser`. The user name you want API Gateway to use to invoke the IS service.<br>■ `complyToISSpec`. Set to true, if you want the input and the output parameters to comply to the IS Spec specified. |
| `clientMetadataMapping` | ■ `providerName`. Name of the provider.<br>■ `implNames`. Map of specification names to the implementation names of the service provider.<br>■ `extendedValuesV2`. List of headers that needs to be sent along with the client management request.<br>■ `generateCredentials`. Specifies whether API Gateway should generate `clientId` and `client secret`.<br>■ `supportedApplicationTypes`. List of `application_type` values supported by the authorization server provider. |
| `soapMessageSecurityAlias` | ■ `authType`. Type of authentication.<br>■ `authMode`. Mode of authentication<br>■ `wssCredentials`. Credentials required for the WSS Username token.<br>■ `kerberosCredentials`. Credentials for the Kerberos token.<br>■ `samlIssuerConfig`. SAML issuer configuration name.<br>■ `signAndEncryptConfig`. Private and public keys to use for signature and encryption. |
| `isConfigurationAlias` | ■ `url`. URL of the Integration Server.<br>■ `username`. User credentials required to access the Integration Server instance. |

| Type | Parameters |
|---|---|
| | `password`. Password required to access the Integration Server instance.<br>`keystoreAlias`. 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.<br>`keyAlias`. The alias for a specific key in the specified keystore.<br>`packageName`. Default package name where the alias is published.<br>`folderName`. Default folder name where the alias is published. |
| `authServerAlias` | `description`. Description of the auth server.<br>`tokenGeneratorConfig`. Specifies the token information that would be added as a bearer token in the HTTP request for client authentication.<br>`supportedGrantTypes`. Specifies the list of grant types that are supported by API Gateway. The grant types supported are:<br>`authorization_code`, `client_credentials`, and `implicit`.<br>`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.<br>`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"
cluster:
  aware: "true"
  name: "APIGatewayTSAcluster"
  tsaUrls: "VMYAI105BVT06:9510"
  terracottaLicenseFileName: "terracotta-license.key"
  sessTimeout: "20"
  actionOnStartupError: "standalone"
  esClusterName: "SAG_EventDataStore"
  discoveryZenPingUnicastHosts: "daeirnd33974:9340,daeirnd33562:9300,daeirnd33974:8300"
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: "xxy"
      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: "GuessIt"
             domain: "EUR"
   myIsAlias:
          type: "isConfigurationAlias"
          url: "http://localhost:5555"
          username: "Administrator"
          password: "mypass"
          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"
            cientSecret: "secet"
            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.tsaUrls=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:

```
######################## Master configuration ############################
# 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` | Indicates the type of the configuration source. The applicable types are YAML, PROPERTIES and CC_YAML.<br>- `YAML`. A YAML configuration file. |

| Property | Detail |
|---|---|
| | ■ `PROPERTIES`. A properties configuration file.<br>■ `CC_YAML`. A YAML configuration file, which is reserved for Command Central updates. |
| `allowEdit` | Indicates whether this file can be updated from API Gateway and is useful for hiding passwords.<br>Valid values are `true` and `false`.<br>■ If the value is set to `true`, it hides the clear text passwords.<br>■ If the value is set to `false`, it displays the clear text passwords. |
| `properties` | Properties that enable API Gateway to connect to the defined configuration source. For the 10.5 release only the `location` property is supported.<br>■ `location`. 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 *SAGInstallDir*`\IntegrationServer\instances\instance_name\packages\WmAPIGateway\resources\configuration`.<br>**Important:**<br>For the CC_YAML file type, the location is fixed as `cc-config.yml`. 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. |

**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.

<table>
<tr><th>Setting</th><th>Password manager key replacement</th></tr>
<tr><td><pre>apigw:
   elasticsearch:
   http:
   username: elastic</pre></td><td>@secure.elasticsearch.http.password</td></tr>
<tr><td><pre>apigw:
   elasticsearch:
   https:
   keystorePassword: 6572b9b06156a0ff778c</pre></td><td>@secure.elasticsearch.http.keystore.password</td></tr>
<tr><td><pre>apigw:
   elasticsearch:
   https:
   truststorePassword: 6572b9b06156a0ff778c</pre></td><td>@secure.elasticsearch.http.truststore.password</td></tr>
</table>

### 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
```

## Limitations

- 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-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}

[291]2019-08-16 11:19:01 IST [YAI.0013.8887D] [default][SAG-G43RXF2] 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-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}

[289]2019-08-16 11:19:01 IST [YAI.0013.8887D] [default][SAG-G43RXF2] 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-G43RXF2] 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

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.

## Connecting to an External Elasticsearch

API Gateway uses Elasticsearch as its primary data store to persist different types of assets such as APIs, Policies, and Applications apart from runtime events and metrics. By default, all assets are stored in the default Elasticsearch. But, you can configure API Gateway to use an external Elasticsearch to store the API Gateway assets. For information about the Elasticsearch version that is compatible with your API Gateway version, see "API Gateway, Elasticsearch, Kibana Compatibility Matrix" on page 119.

When you configure external Elasticsearch you can have one of the following configurations:

- External Elasticsearch to store only the analytics.

  This is achieved by configuring the external Elasticsearch as a destination store the analytics data in the configured destination. For details about the supported destinations and their configuration, see *webMethods API Gateway User's Guide*. In this case the core configurations (such as APIs, Applications, Policies, Plans, Packages, Administration Settings, Security Configurations (Keystores/Trustores) and Tokens (OAuth/API Keys)) are stored in the internal default Elasticsearch.

- External Elasticsearch to store all API Gateway assets.

  You can configure this in one of the following ways:

  - Specifying the appropriate properties in the configurations config.properties file, which is explained in this section.
  - Using externalized configuration files. For details, see " Using the Externalized Configuration Files" on page 88.

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

The configuration specified in the config.properties file override the values that are configured in gateway-es-store.xml during runtime and the values in gateway-es-store.xml are not changed. During the first start-up of API Gateway, default values from gateway-es-store.xml are automatically copied to config.properties. From the next start-up of API Gateway, values from config.properties are used. Once the host is specified in config.properties the value is not over-written from gateway-es-store.xml.

**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 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**<br>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.<br>Default value: true |
| **pg.gateway.elasticsearch.client.http.response.size**<br>This property specifies the response size, in MB, for API Gateway Elasticsearch client.<br>Default value: 100 |
| **pg.gateway.elasticsearch.config.location**<br>This property specifies the location of the config file if you want to read port details from some other Elasticsearch config file |
| **pg.gateway.elasticsearch.hosts**<br>*Mandatory*<br>This property lists Elasticsearch hosts and ports. The values are comma separated.<br>Default value: localhost:9240<br>**Note:**<br>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. |

| Property and Description |
|---|
| **pg.gateway.elasticsearch.http.keepAlive**<br>*Mandatory*<br>This property creates the persistent connection between client and server.<br>Default value: true |
| **pg.gateway.elasticsearch.http.connectionTimeout**<br>*Mandatory*<br>This property specifies the time, in milliseconds, after which the connection times out.<br>Default value: 10000 |
| **pg.gateway.elasticsearch.http.socketTimeout**<br>*Mandatory*<br>This property specifies the wait time, in milliseconds, for a reply once the connection to Elasticsearch is established after which it times out.<br>Default value: 30000 |
| **pg.gateway.elasticsearch.http.maxRetryTimeout**<br>*Mandatory*<br>This property specifies the wait time, in milliseconds, for retries after which it times out.<br>Default value: 100000<br>It is advisable to set max retry time for a request to (number of nodes * socketTimeOut )+connectionTimeout |
| **pg.gateway.elasticsearch.http.keepAlive.maxConnections**<br>*Mandatory*<br>This property specifies the maximum number of persistent connections that can be established between an API Gateway and Elasticsearch cluster.<br>Default value: 50 |
| **pg.gateway.elasticsearch.http.keepAlive.maxConnectionsPerRoute**<br>*Mandatory*<br>This property specifies the maximum number of persistent connections that can be established per HTTP route to an Elasticsearch server.<br>Default value: 15 |

| Property and Description |
|---|
| **pg.gateway.elasticsearch.http.username**<br>This property specifies the user name to connect to Elasticsearch using basic authentication. |
| **pg.gateway.elasticsearch.http.password**<br>This property specifies the password to connect to Elasticsearch using basic authentication. |
| **pg.gateway.elasticsearch.https.keystore.filepath**<br>This property specifies the Keystore file path for establishing HTTPS communication with Elasticsearch. |
| **pg.gateway.elasticsearch.https.truststore.filepath**<br>This property specifies the truststore file path for establishing HTTPS communication with Elasticsearch. |
| **pg.gateway.elasticsearch.https.keystore.password**<br>This property specifies the Keystore password for establishing HTTPS communication with Elasticsearch. |
| **pg.gateway.elasticsearch.https.keystore.alias**<br>This property specifies the Keystore alias for establishing HTTPS communication with Elasticsearch. |
| **pg.gateway.elasticsearch.https.truststore.password**<br>This property specifies the truststore password for establishing HTTPS communication with Elasticsearch. |
| **pg.gateway.elasticsearch.https.enabled**<br>This property specifies whether you want to enable or disable the HTTPS communication with Elasticsearch.<br>Default value: false<br>If this property is set to false none of the above properties related to HTTPS are respected. |
| **pg.gateway.elasticsearch.outbound.proxy.enabled**<br>This property specifies whether you want to enable or disable outbound proxy communication.<br>Default value: true |
| **pg.gateway.elasticsearch.outbound.proxy.alias**<br>This property specifies the outbound proxy alias name used to connect to Elasticsearch. |
| **pg.gateway.elasticsearch.https.enforce.hostname.verification** |

| Property and Description |
|---|
| This property enforces the host name verification for SSL communication.<br>Default value: false |
| **pg.gateway.elasticsearch.sniff.enable**<br>*Mandatory*<br>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.<br>Default value: true<br>This configuration must be set to *false* if you are changing the network when API Gateway or Elasticsearch is running. |
| **pg.gateway.elasticsearch.tenantId**<br>This property allows you to specify a tenant name of your choice.<br>Default value: API Gateway instance name |
| **pg.gateway.elasticsearch.sniff.timeInterval**<br>*Mandatory*<br>This property enables adding the newly added Elasticsearch cluster nodes to existing REST client in a specified time interval in milliseconds.<br>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 88.

## API Gateway, Elasticsearch, Kibana Compatibility Matrix

As stated earlier, API Gateway uses Elasticseach 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 versions that support the storage of core data and transaction data of the available API Gateway versions:

| API Gateway version | Compatible Elasticsearch versions<br>(Core data level) | Compatible Elasticsearch versions<br>(Transaction data level) | Compatible Kibana version |
|---|---|---|---|
| 10.11 | 7.13.0 | All Elasticsearch versions | 7.13.0 |
| 10.7 | 7.7 | All Elasticsearch versions | 7.7 |
| 10.5 | 7.2.0 | All Elasticsearch versions | 7.2.0 |
| 10.4 | 5.6.4, 2.3.2 | All Elasticsearch versions | 5.6.x, 4.5.x |
| 10.3 | 5.6.4, 2.3.2 | All Elasticsearch versions | 5.6.x, 4.5.x |
| 10.2 | 5.6.4, 2.3.2 | All Elasticsearch versions | 5.6.x, 4.5.x |
| 10.1 | 2.3.2 | All Elasticsearch versions | 4.5.x |
| 9.12 | 2.3.2 | All Elasticsearch versions | 4.5.x |

API Gateway10.11 ships 7.13.0 version of Elasticsearch and Kibana, and 7.13.0 OSS version of Filebeat.

## 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 88. 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.

| Property | Description |
|---|---|
| `server.port:` *port number* | Specifies which server port to use.<br>Example: `9405` |
| `server.host:` *server host IP address or host name* | Specifies the host to bind the server to.<br>The default value is `localhost`, 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.<br>Example: `"0.0.0.0"` |
| `server.basePath:` *server path of the proxy* | Specifies the proxy setting to render the charts from the external Kibana in API Gateway UI.<br>The server path you specify must not end with a /.<br>Value: `"/apigatewayui/dashboardproxy"` |
| `elasticsearch.hosts:` http://*hostname*:*port* | Specifies the URLs of the Elasticsearch instance to use for all your queries.<br>Example: `"http://localhost:9240"` |
| `kibana.index:` `gateway_`*tenant_name*`_dashboard` | Specifies the index in Elasticsearch, which Kibana uses to store saved searches, visualizations, and dashboards. It creates a new index if it does not exist.<br>Example: `"gateway_default_dashboard"` |

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.

- Default Kibana connected to External Elasticsearch.

- If the external Elasticsearch is used 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"
  ```

- If the external Elasticsearch is used to store only the analytics and the core configuration is stored in the API Data Store, then configure the following:

  Copy the kibana.index (gateway_*tenant-name*_dashboard) from the Elasticsearch that stores the core configurations to the Elasticsearch that stores the analytics data. This can be achieved by using the reindex API. Reindex supports reindexing from a remote Elasticsearch cluster. The sample payload is as follows:

  ```
  POST _reindex
  {
    "source": {
      "remote": {
        "host": "https://host:port",
        "username": "username",
        "password": "password"
      },
      "index": "gateway_tenant-name_dashboard",
      },
    "dest": {
      "index": "gateway_target-tenant-name_dashboard"
    }
  }
  ```

  The host parameter must contain a scheme, host, and port. The username and password parameters are optional, and when they are present _reindex connects to the remote Elasticsearch node using basic auth.

  For details about the reindex API, see https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-reindex.html#reindex-from-remote.

  Remote hosts have to be explicitly allowed in elasticsearch.yml using the reindex.remote.whitelist property. It can be set to a comma delimited list of allowed remotehost and port combinations. Scheme is ignored, only the host and port are used. The list of allowed hosts must be configured on the target node where the index is being copied.

- External Kibana connected to External Elasticsearch.

  You can configure this setup by using externalized configuration files. For details, see “ Using the Externalized Configuration Files” on page 88.

**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.

## 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 port using `primary.port option`.

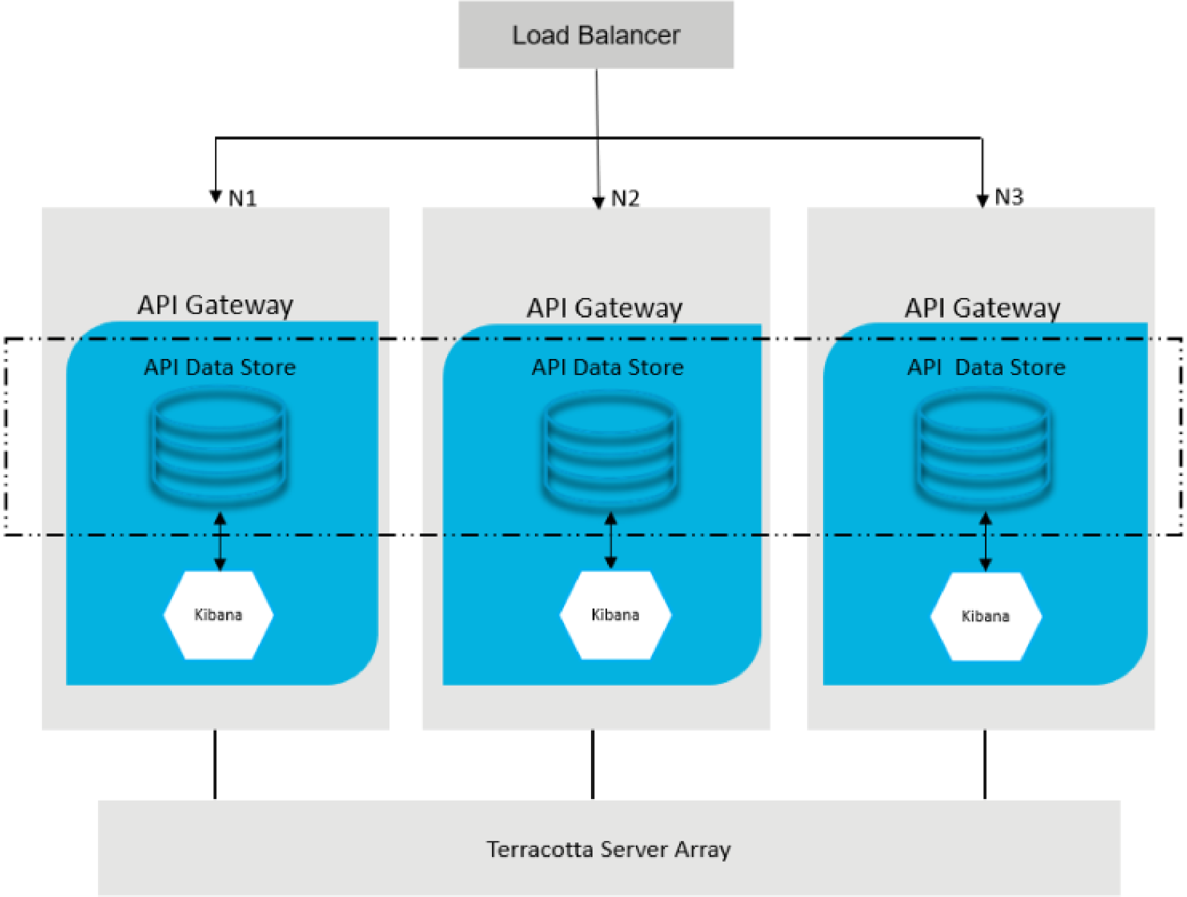
To create a new instance, run the following command:

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

## 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.



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 .

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.<br>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

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/Housekeeping" on page 234,"Monitoring" on page 68. 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.

**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. 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<br>Minimum: *4* GB | *2* cores | *30* GB | Minimum Heap size: *2048* MB<br>Maximum Heap size: *3584* MB<br>Parameters to define the Heap size:<br>`Minimum - Xms`<br>`Maximum - Xmx` |
| Terracotta | Maximum: *8* GB<br>Minimum: *8* GB | *2* cores | *30* GB | Heap usage: *2* GB<br>Off Heap usage: *2* GB |
| Elasticsearch | Maximum: *6* GB<br>Minimum: *4* GB | *2* cores | *300* GB | Minimum Heap size: *3584* MB<br>Maximum Heap size: *3584* MB |

| Component | RAM | CPU | HDD | Heap |
|---|---|---|---|---|
| | | | | Parameters to define the Heap size:<br>Minimum - `Xms`<br>Maximum - `Xmx` |
| Kibana | Maximum: *6* GB<br>Minimum: *4* GB | *2* cores | *30* GB | Maximum Heap size: *4096* MB<br>Parameter to define the Maximum Heap size:<br>`--max-old-space-size=4096 MB` |

### Standalone setup

| Component | RAM | CPU | HDD | Heap |
|---|---|---|---|---|
| API Gateway | Maximum: *6* GB<br>Minimum: *4* GB | *2* cores | *30* GB | Minimum Heap size: *2048* MB<br>Maximum Heap size: *3584* MB<br>Parameters to define the Heap size:<br>`Minimum – Xms`<br>`Maximum – Xmx` |
| Elasticsearch | Maximum: *6* GB<br>Minimum: *4* GB | *2* cores | *300* GB | Minimum Heap size: *3584* MB<br>Maximum Heap size: *3584* MB<br>Parameters to define the Heap size:<br>Minimum – `Xms`<br>Maximum - `Xmx` |
| Kibana | Maximum: *6* GB<br>Minimum: *4* GB | *2* cores | *30* GB | Maximum Heap size: *4096* MB<br>Parameter to define the Maximum Heap size:<br>`--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:

- `SAGInstallDirectory\IntegrationServer\instances\instance_name\logs`.
- `SAGInstallDirectory\profiles\IS_instance_name\logs`.

API Data Store logs:

- `SAGInstallDirectory\InternalDataStore\logs`.

Terracotta logs:

- Client log: `SAGInstallDirectory\IntegrationServer\instances\instance_name\logs`.
- Server log: `SAGInstallDirectory\tsa`.

Kibana logs:

- `SAGInstallDirectory\profiles\IS_instance_name\apigateway\dashboard\config` or `Kibana_InstallDirectory\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 `SAGInstallDirectory\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 `SAGInstallDirectory\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 `SAGInstallDirectory\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 `SAGInstallDirectory\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 `SAGInstallDirectory\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 `SAGInstallDirectory\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*\IntegrationServer\instances\*instance_name*\Packages\WmAPIGateway\config\resources\elasticsearch\config.properties`.

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
```

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 `SAGInstallDirectory\profiles\IS_instance_name\apigateway\dashboard\config` or `Kibana_InstallDirectory\config`.

To configure the total size of the logs as **300** MB, add the following properties and its values in kibana.yml log file and save it.

```
#Enables you to specify a file location where Kibana should store the log output.
logging.dest = <kibana_logfile_location>/kibana.log
#Enables the rotation of the logs
```

```
logging.rotate.enabled = true
logging.rotate.everyBytes = 10485760
logging.rotate.keepFiles = 30
logging.rotate.usePolling = true
```

A sample configuration is as follows:

```
server.name: apigw-kibana-{{ .Values.tenantName }}-107-0
server.host: "0.0.0.0"
server.port: 9405
elasticsearch.hosts: [ "http://apigw-{{ .Values.tenantName }}-es-107-svc:80" ]
console.enabled: false
server.basePath: "/apigatewayui/dashboardproxy"
kibana.index: "gateway_{{ .Values.tenantName }}_dashboard"
logging.dest: "kibana.log"
logging.rotate.enabled: true
logging.rotate.everyBytes: 10485760
logging.rotate.keepFiles: 30
logging.rotate.usePolling: true
elasticsearch.requestTimeout: 90
telemetry.enabled: false
```

## 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* section in *API Gateway Administration* chapter in *webMethods API Gateway User's Guide*.

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 Serverbecause 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.

**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 available under **Username > Administration >General > Extended settings** in API Gateway to false.

Recommended value: false

For more details about Ports Configuration, see “Ports Configuration” on page 61.

**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.

For changing the refresh interval, go to **API Gateway UI -> Administration -> Extended settings -> eventRefreshInterval** and change it.

Recommended value: 10s

## Terracotta Configurations

See "Terracotta Server Array Configuration" on page 59 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 88.

## Kibana Configurations

This section explains the configuration changes for Kibana in kibana.yml file located at `SAGInstallDirectory\profiles\IS_instance_name\apigateway\dashboard\config`or `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*

**telemetry.enabled**

Disables the telemetry data being sent to elastic server.

Recommended value: false

For more details about Kibana configurations, see " Connecting to an External Kibana" on page 120.

## 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 |
|---|---|
| `wrapper.java.initmemory` | The minimum heap size.<br>The default value is 256 MB. |
| `wrapper.java.maxmemory` | The maximum heap size.<br>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.

## Accessing the API Gateway User Interface

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.

## Restarting API Gateway Using Scripts

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

**To restart 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 restart API Gateway.

## Restarting API Gateway Using User Interface

You can restart API Gateway through the API Gateway user interface. This lets you restart API Gateway without shutting it down. 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 User Interface

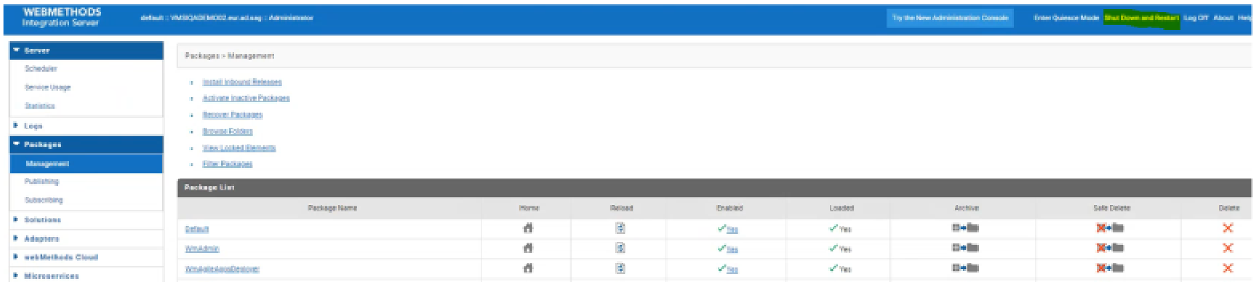
1. Open a browser and type `localhost:5555`.

   **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 **Shut Down and Restart**.

   This opens the Shut Down and Restart page as shown below.



| Package Name | Home | Reload | Enabled | Loaded | Archive | Safe Delete | Delete |
|---|---|---|---|---|---|---|---|
| Default | | | ✓ Yes | ✓ Yes | | | ✕ |
| WmAdmin | | | ✓ Yes | ✓ Yes | | | ✕ |
| WmAgileAppsDeployer | | | ✓ Yes | ✓ Yes | | | ✕ |

3. In the Shut Down or Restart menu, select one of the following options:

   - **After all sessions end**. Select this option to shut down API Gateway after all the active sessions are completed.
   - **Immediately**: Select this option to shut down API Gateway immediately.

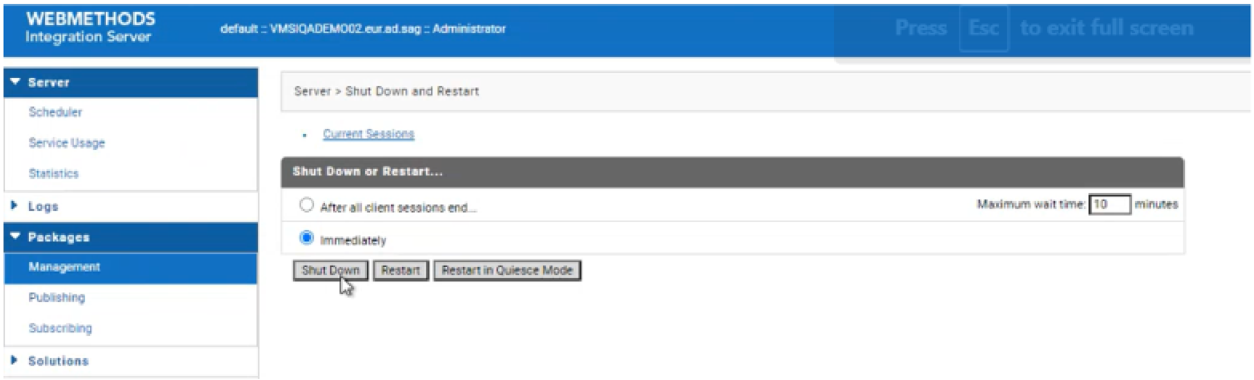
   **Important:**
   You must use the **Immediately** option only if your API Gateway has a clustered configuration. With clustered configuration, all the active sessions are transferred to another API Gateway node. If you select the **Immediately** option with a clustered configuration, all your active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

4. Click one of the following buttons to restart API Gateway:

   - **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.

   To shut down API Gateway, you can use the **Shut Down** button.



Server > Shut Down and Restart

○ After all client sessions end...

Maximum wait time: 10 minutes

● Immediately

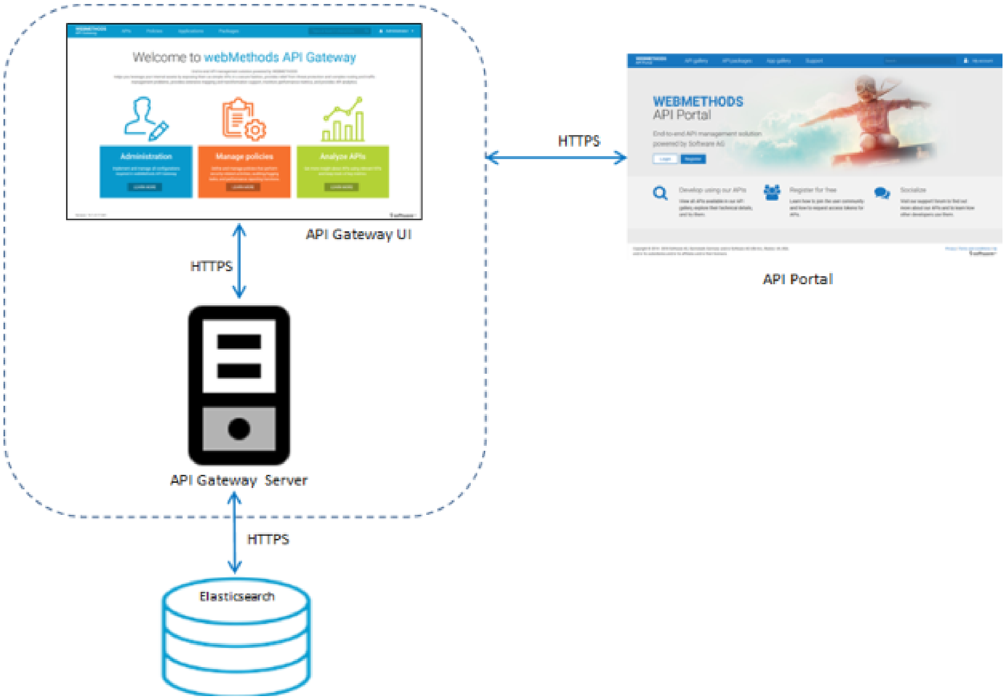
# 4 Securing API Gateway and its Components

## Overview

The basic API Management setup comprises of API Gateway, the API Clients, Users, Backend services, and API Portal. 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 API Gateway setup comprises various components, such as, API Gateway server, API Gateway UI, and API Data Store. This section also describes how to secure the communication between the components of API Gateway.

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 143
- API Gateway UI and Users. For details, see “ How do I Secure API Gateway User Interface Communication? ” on page 153
- API Gateway and API Portal. For details, see “ How do I Configure a Secure Communication Channel between API Gateway and API Portal?” on page 155
- API Gateway and API Data Store. For details, see

# 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 169 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.

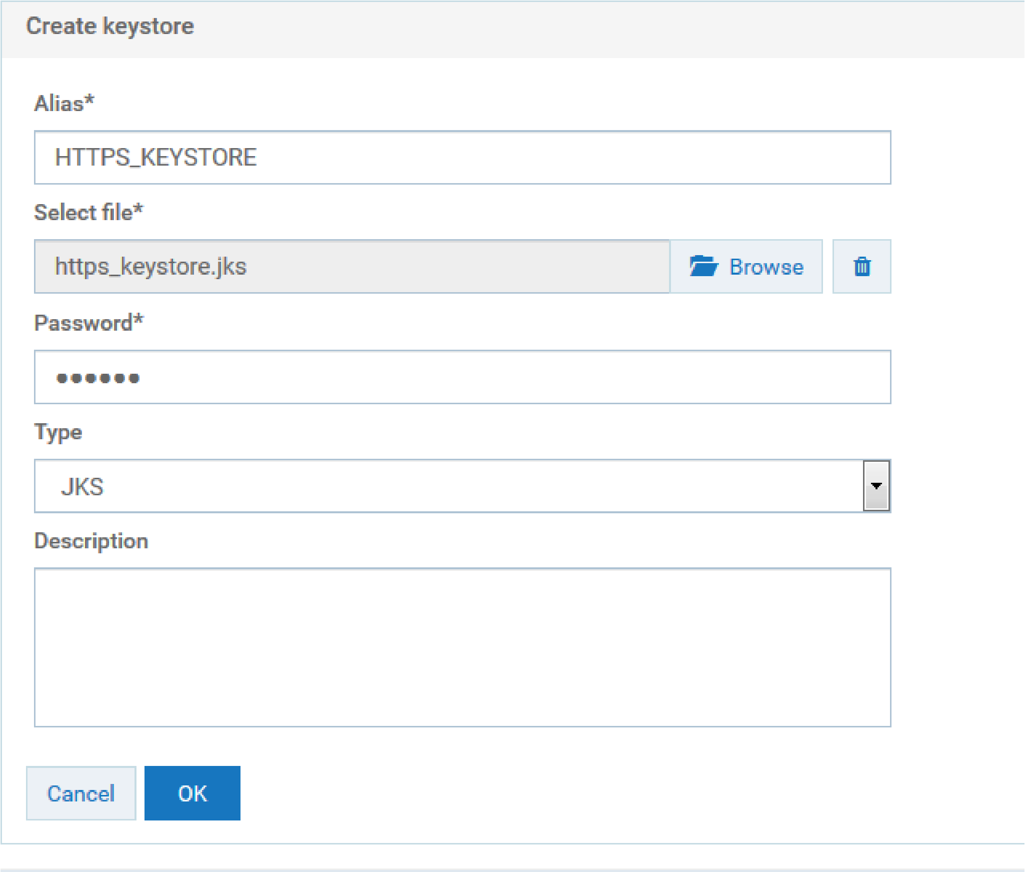
   a. Log on to API Gateway.

   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

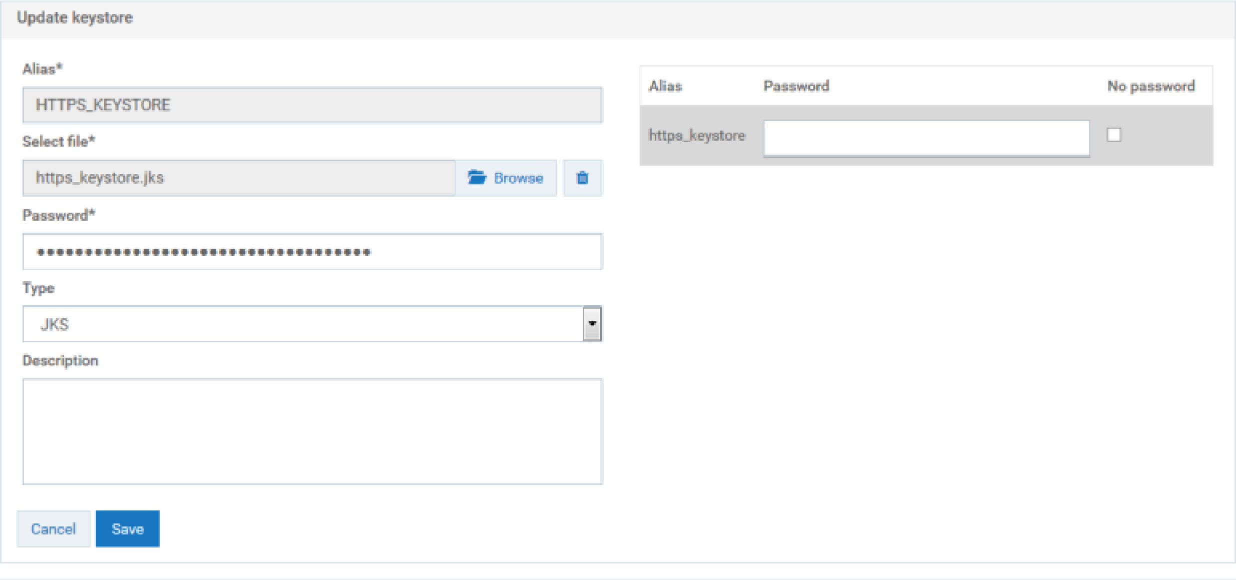
e. Click **OK**.

   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:

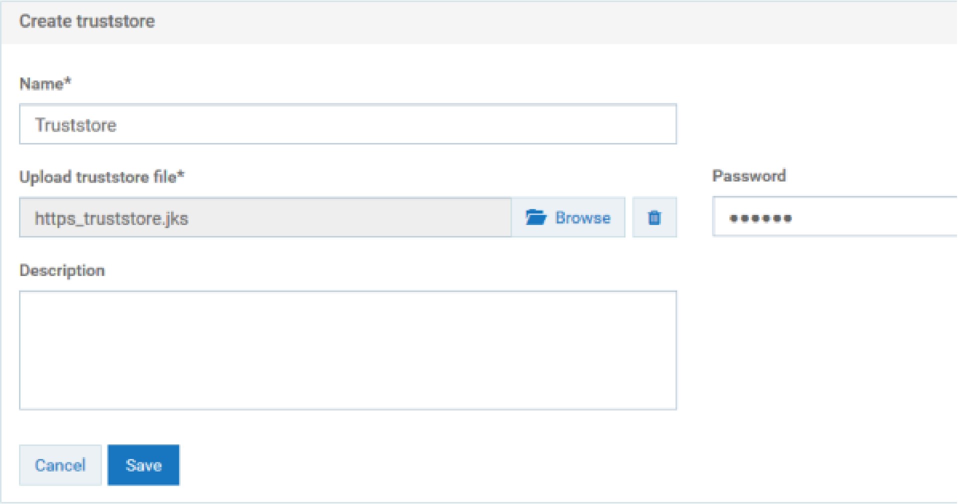
| Alias | Password | No password |
|---|---|---|
| https_keystore | | ☐ |

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

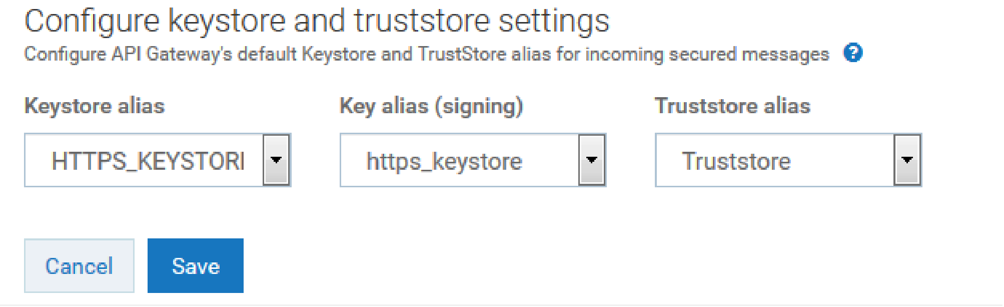
Upload truststore file*: https_truststore.jks

Password: ••••••

Description:

k. Click **Save**.

l. 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: HTTPS_KEYSTORI

Key alias (signing): https_keystore

Truststore alias: Truststore

m. Click **Save**.

3. Create an HTTPS port in API Gateway and associate the keystore and truststore aliases.

   a. Navigate to **Administration > Security > Ports**.

   b. Click **Add ports**, and select **HTTPS** as the port type.

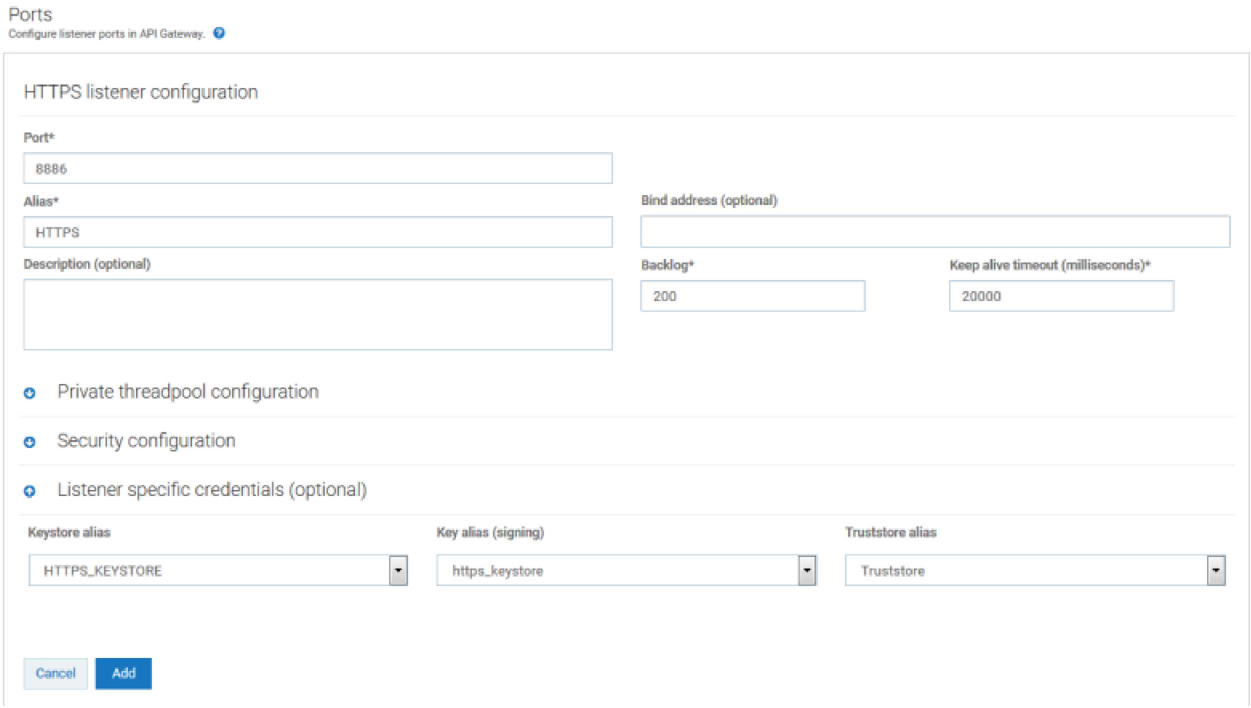
   c. Click **Add**.

   d. 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.

   e. 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

Bind address (optional):

Description (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

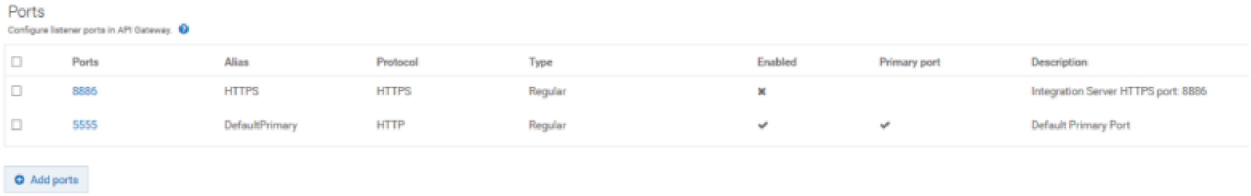
Truststore alias: Truststore

f. In the Security configuration section, provide the following details:

   **Use JSSE**. Select **Yes** to create the port using the Java Secure Socket Extension (JSSE) socket factory for the port to support TLS 1.1 or TLS 1.2. The default value is **Yes**. If you set this value to **No** , the port supports only SSL 3.0 and TLS 1.0.

g. Click **Add**.

   The HTTPS port 8886 is added and displayed in the list of ports.



Ports

Configure listener ports in API Gateway.

| ☐ | Ports | Alias | Protocol | Type | Enabled | Primary port | Description |
|---|---|---|---|---|---|---|---|
| ☐ | 8886 | HTTPS | HTTPS | Regular | ✖ |  | Integration Server HTTPS port: 8886 |
| ☐ | 5555 | DefaultPrimary | HTTP | Regular | ✓ | ✓ | Default Primary Port |

h. 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.

   a. Navigate to **Administration > Security > Ports**. This displays the list of ports.

   b. Click the port 8886.

c. In the **Security configuration** > **Client authentication** section, select one of the following values:

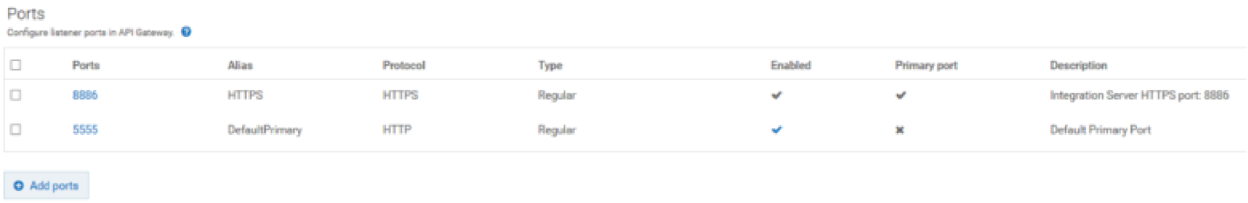
   - **Request client certificate**. API Gateway requests client certificates for all requests. If the client does not provide a certificate, the server prompts the client for a userid and password. The server checks whether the certificate exactly matches a client certificate on file and is signed by a 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

Configure listener ports in API Gateway.

| ☐ | Ports | Alias | Protocol | Type | Enabled | Primary port | Description |
|---|---|---|---|---|---|---|---|
| ☐ | 8886 | HTTPS | HTTPS | Regular | ✓ | ✓ | Integration Server HTTPS port: 8886 |
| ☐ | 5555 | DefaultPrimary | HTTP | Regular | ✓ | ✖ | Default Primary Port |

   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. Restrict the TLS version by adding the following setting:

   ```
   watt.net.jsse.server.enabledProtocols=TLSv1.2
   ```

   This specifies the SSL protocol versions that API Gateway supports when acting as a server handling inbound requests. Java Secure Socket Extensions (JSSE) is required to support TLS 1.1 or 1.2.

   For more information about configuring portsnto use JSSE, see https://documentation.softwareag.com/webmethods/integration_server/pie10-5/10-5_Integration_Server_Administrators_Guide.pdf

2. 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
   ```

3. 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

4. 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 169 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 serivice certificates.

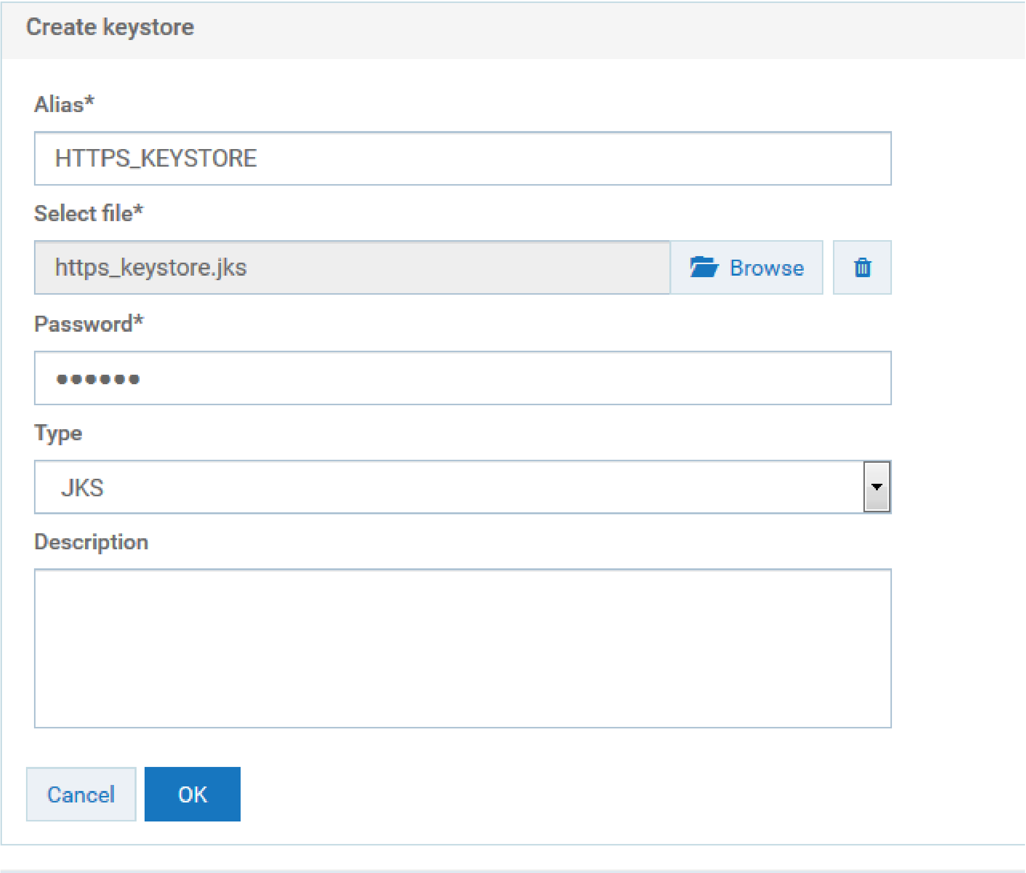
   a. Log on to API Gateway.

   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

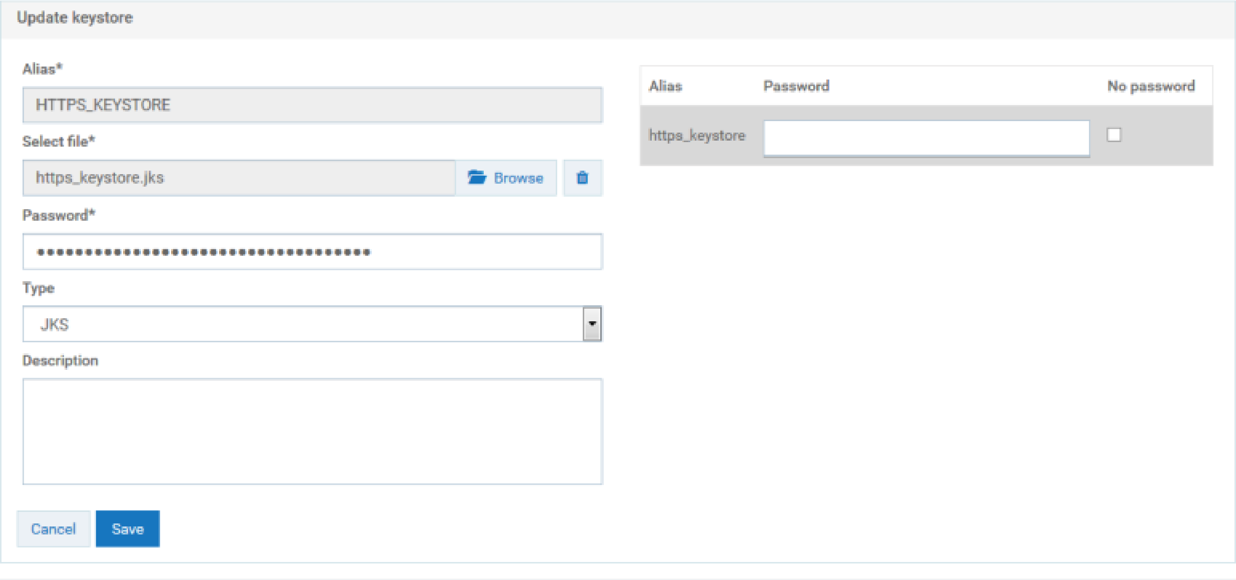
e. Click **OK**.

   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

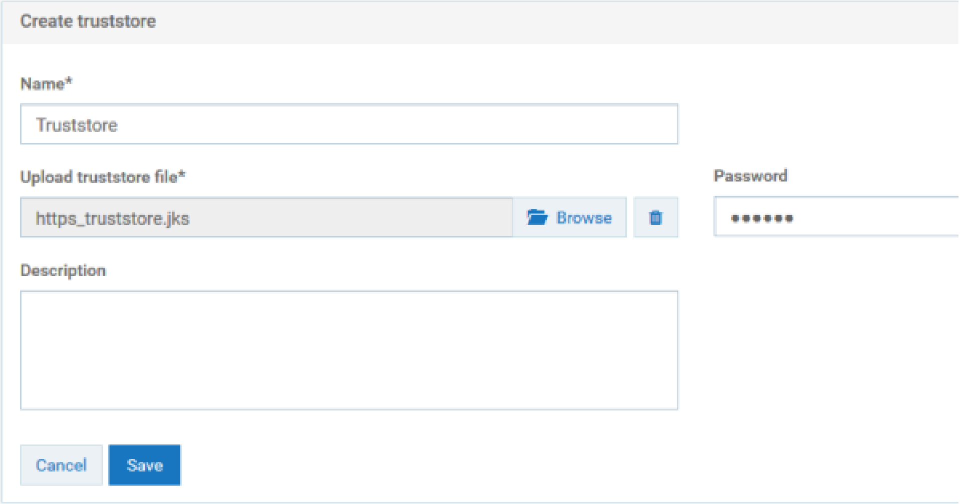
| Alias | Password | No password |
|---|---|---|
| https_keystore | | ☐ |

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

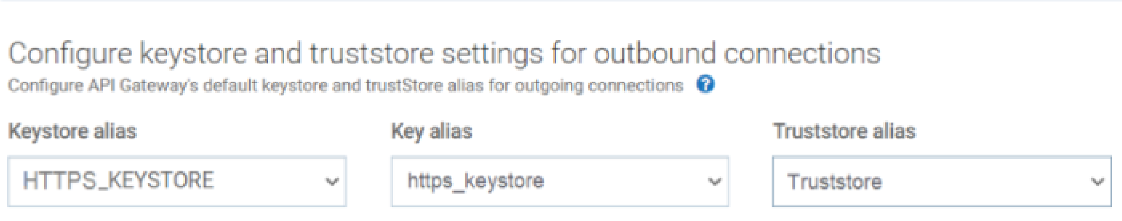
k. Click **Save**.

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.



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

   - 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 169 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 open the property file `com.softwareag.catalina.connector.https.pid-apigateway.properties`.

   b. Modify the following properties by providing the keystore, passsword, 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 https://documentation.softwareag.com/webmethods/wmsuites/wmsuite10-5/Cross_Product/10-5_Software_AG_Infrastructure_Administrators_Guide.pdf 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 by adding the following line to the properties file com.softwareag.catalina.connector.https.pid-apigateway.properties located in the directorySAG_root
   /profiles/IS_default/configuration/com.softwareag.platform.config.propsloader.

   ```
   sslEnabledProtocols=TLSv1.2
   ```

2. Specify a list of secure cipher suites by adding the following line to the properties file com.softwareag.catalina.connector.https.pid-apigateway.properties located in the directorySAG_root
   /profiles/IS_default/configuration/com.softwareag.platform.config.propsloader.

   ```
   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 the https://documentation.softwareag.com/webmethods/integration_server/pie10-5/10-5_Integration_Server_Administrators_Guide.pdf

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 version10.2 or later. Also ensure that the required certificates for API Gateway and API Portal are available.

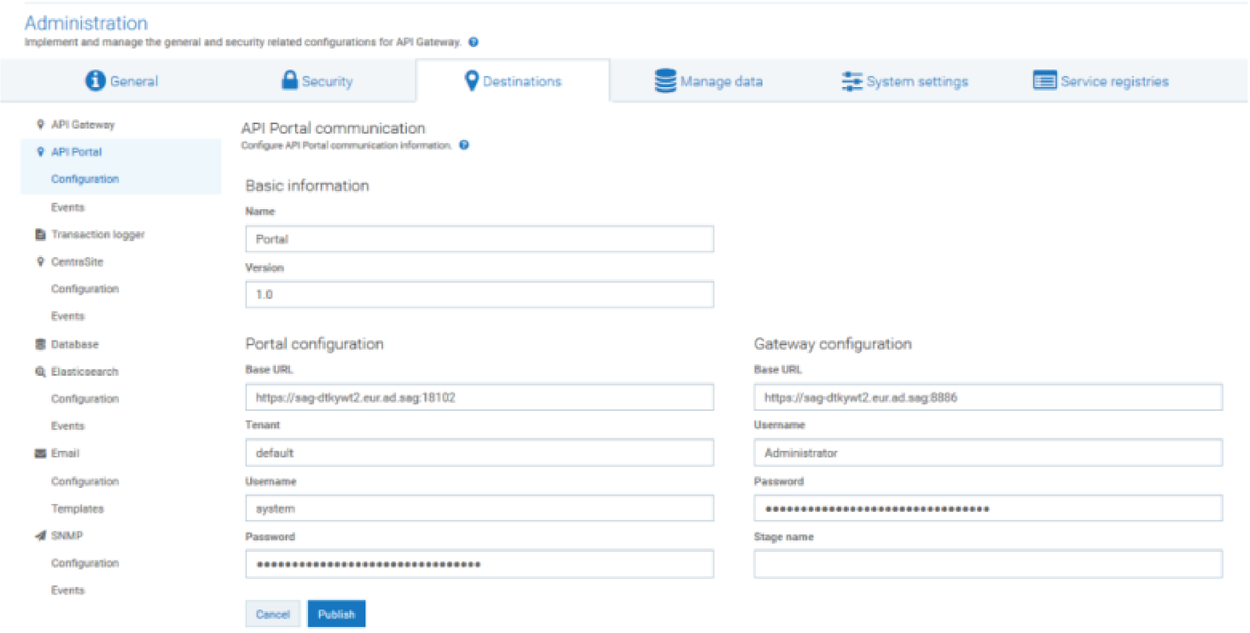
### To configure a secure communication channel between API Gateway and API Portal

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:



Name: Portal
Version: 1.0

Portal configuration
Base URL: https://sag-dtkywt2.eur.ad.sag:18102
Tenant: default
Username: system
Password: ********************************

Gateway configuration
Base URL: https://sag-dtkywt2.eur.ad.sag:8886
Username: Administrator
Password: ********************************
Stage name:

   - In the Portal configuration section, provide the following details:

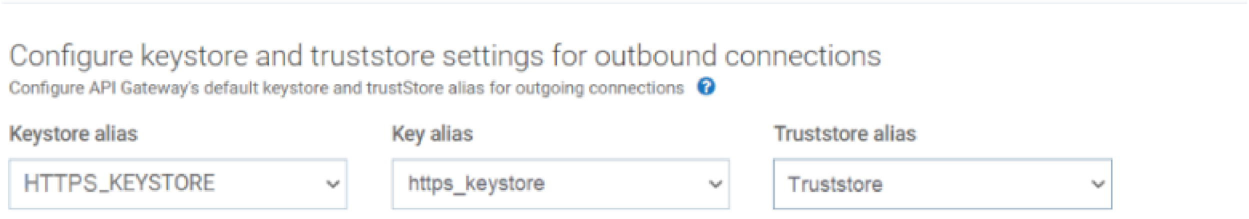
    - **Base URL**. The API Portal base URL which API Gateway uses to communicate to API Portal using the HTTPS port. By default, API Portal uses port 18102 for HTTPS communication.
    - **Username** and **Password** credentials to access API Portal.
  - In the Gateway configuration section, provide the 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. This section explains how to secure Elasticsearch using Search Guard, an Elasticsearch plugin, that offers encryption, authentication, and authorization to protect data from attackers and other misuses. Search Guard secures Elasticsearch by exposing it over HTTPS, and enables basic authentication by configuring users.

## Before you begin

Ensure that you have:

- A basic understanding of API Gateway and its communication with API Data Store for storing data.
- A basic understanding of Kibana and its communication with API Data Store for rendering the dashboards in API Gateway.
- You have downloaded the Search Guard plug in. You can download the Search Guard plugin version 51.0.0 compatible for Elasticsearch 7.13.0 from https://maven.search-guard.com/artifactory/webapp/#/home and store it in your file system.

## To secure API Data Store communication using HTTPS

1. Install and initialize Search Guard plugin.
    a. Shutdown API Gateway.

    b. Open the command prompt to the location `Installation_Dir/InternalDataStore/bin`

    c. Run the following command:

    ```
    elasticsearch-plugin.bat install Search_Guard_plugin
    file_location_in_the_file_system
    ```

    d. Type `y` when the installation procedure prompts for additional required permissions it requires.

    You should see a procedure completion message *Installed search-guard-7* on successful installation.

    e. Copy the folder sagconfig from `Installation_Dir\IntegrationServer/instances\Instance_name\packages\WmAPIGateway\config\resources\elasticsearch` to `Installation_Dir\InternalDataStore`.

    f. Copy the certificates node-0-keystore.jks and truststore.jks from `Installation_Dir\InternalDataStore\sagconfig` to `Installation_Dir\InternalDataStore\config`.

    g. Navigate to `Installation_Dir\InternalDataStore\config\` and open the file elasticsearch.yml.

    h. Delete all the properties that start with *searchguard*, if present, and add the Search Guard properties as follows:

    ```
    searchguard.ssl.transport.keystore_type: JKS
    searchguard.ssl.transport.keystore_filepath: node-0-keystore.jks
    searchguard.ssl.transport.keystore_alias: cn=node-0
    searchguard.ssl.transport.keystore_password: a362fbcce236eb098973
    searchguard.ssl.transport.truststore_type: JKS
    searchguard.ssl.transport.truststore_filepath: truststore.jks
    searchguard.ssl.transport.truststore_alias: root-ca-chain
    searchguard.ssl.transport.truststore_password: 2c0820e69e7dd5356576
    ```

```
searchguard.ssl.transport.enforce_hostname_verification: false
searchguard.ssl.transport.resolve_hostname: false
searchguard.ssl.transport.enable_openssl_if_available: true

searchguard.ssl.http.enabled: true
searchguard.ssl.http.keystore_type: JKS
searchguard.ssl.http.keystore_filepath: node-0-keystore.jks
searchguard.ssl.http.keystore_alias: cn=node-0
searchguard.ssl.http.keystore_password: a362fbcce236eb098973
searchguard.ssl.http.truststore_type: JKS
searchguard.ssl.http.truststore_filepath: truststore.jks
searchguard.ssl.http.truststore_alias: root-ca-chain
searchguard.ssl.http.truststore_password: 2c0820e69e7dd5356576
searchguard.ssl.http.clientauth_mode: OPTIONAL
searchguard.enable_snapshot_restore_privilege: true
searchguard.check_snapshot_restore_write_privileges: true
searchguard.restapi.roles_enabled: ["SGS_ALL_ACCESS"]
searchguard.authcz.admin_dn:
  - "CN=sgadmin"
```

For details about all the Search Guard properties, see "Search Guard Properties" on page 166.

i. *Optional*. If you are using trial version of Search Guard, add the following entry.

```
searchguard.enterprise_modules_enabled: false
```

j. Save and close the file.

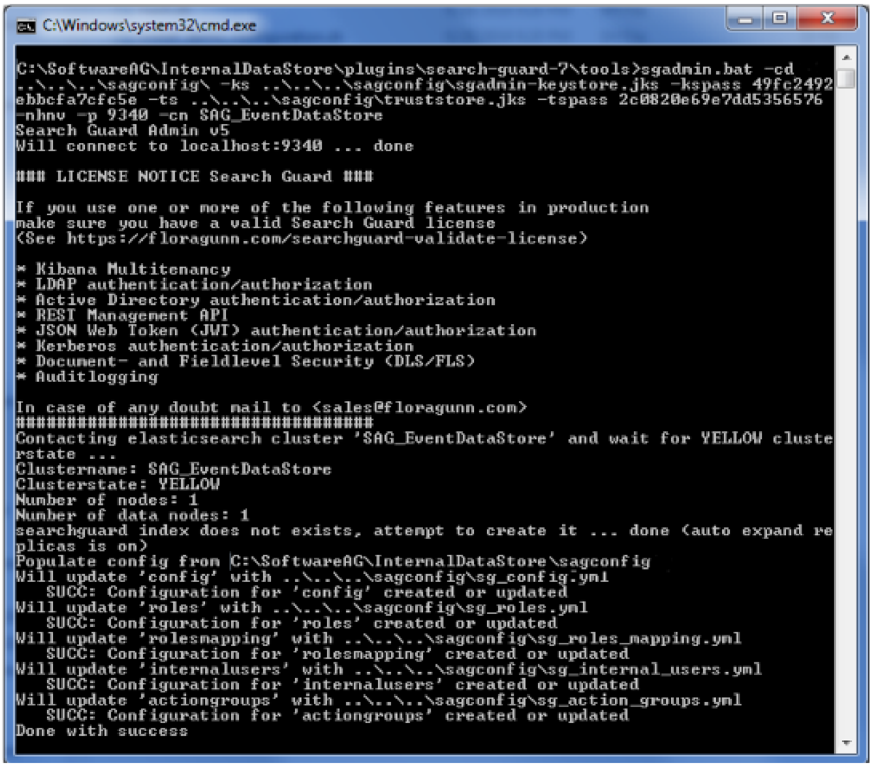
k. Run *Installation_Dir*\InternalInternalDataStore\bin\enable_ssl.bat,

This installs the Search Guard plugin and starts the API Data Store.

l. Shutdown and restart the API Data Store.

m. Navigate to `Installation_Dir\InternalDataStore\plugins\search-guard-7\tools` and run the following command to initialize the API Data Store.

```
sgadmin.bat -cd ..\..\..\sagconfig\
-ks ..\..\..\sagconfig\sgadmin-keystore.jks
-kspass 49fc2492ebbcfa7cfc5e -ts ..\..\..\sagconfig\truststore.jks
-tspass 2c0820e69e7dd5356576 -nhnv -p 9340 -cn SAG_EventDataStore
```



```
C:\SoftwareAG\InternalDataStore\plugins\search-guard-7\tools>sgadmin.bat -cd
..\..\..\sagconfig\ -ks ..\..\..\sagconfig\sgadmin-keystore.jks -kspass 49fc2492
ebbcfa7cfc5e -ts ..\..\..\sagconfig\truststore.jks -tspass 2c0820e69e7dd5356576
-nhnv -p 9340 -cn SAG_EventDataStore
Search Guard Admin v5
Will connect to localhost:9340 ... done

### LICENSE NOTICE Search Guard ###

If you use one or more of the following features in production
make sure you have a valid Search Guard license
(See https://floragunn.com/searchguard-validate-license)

* Kibana Multitenancy
* LDAP authentication/authorization
* Active Directory authentication/authorization
* REST Management API
* JSON Web Token (JWT) authentication/authorization
* Kerberos authentication/authorization
* Document- and Fieldlevel Security (DLS/FLS)
* Auditlogging

In case of any doubt mail to <sales@floragunn.com>
###################################
Contacting elasticsearch cluster 'SAG_EventDataStore' and wait for YELLOW cluste
rstate ...
Clustername: SAG_EventDataStore
Clusterstate: YELLOW
Number of nodes: 1
Number of data nodes: 1
searchguard index does not exists, attempt to create it ... done (auto expand re
plicas is on)
Populate config from C:\SoftwareAG\InternalDataStore\sagconfig
Will update 'config' with ..\..\..\sagconfig\sg_config.yml
   SUCC: Configuration for 'config' created or updated
Will update 'roles' with ..\..\..\sagconfig\sg_roles.yml
   SUCC: Configuration for 'roles' created or updated
Will update 'rolesmapping' with ..\..\..\sagconfig\sg_roles_mapping.yml
   SUCC: Configuration for 'rolesmapping' created or updated
Will update 'internalusers' with ..\..\..\sagconfig\sg_internal_users.yml
   SUCC: Configuration for 'internalusers' created or updated
Will update 'actiongroups' with ..\..\..\sagconfig\sg_action_groups.yml
   SUCC: Configuration for 'actiongroups' created or updated
Done with success
```

If you are using the Linux command it would be as follows:

```
sgadmin.sh -cd ../../../sagconfig\
-ks ../../../sagconfig/sgadmin-keystore.jks
-kspass 49fc2492ebbcfa7cfc5e -ts ../../../sagconfig\truststore.jks
-tspass 2c0820e69e7dd5356576 -nhnv -p 9340 -cn SAG_EventDataStore
```

2. Add users for basic authentication.

   a. Navigate to `Installation_Dir\InternalDataStore\sagconfig` and open the sg_roles_mapping.yml file.

   b. Add the username (for example, TestUser) in the users list as follows:

   ```
   sg_all_access:
   reserved: true
   users:
      "TestUser"
   backend_roles:
      "admin"
   ```

   c. Generate the hash code for your password.

      a. Run the command as follows:

      ```
      Installation_Dir\InternalDataStore\
      plugins\search-guard-7\tools>hash.bat.
      ```

      b. Type the password.

      c. Press **Enter**.

This generates the hash code.

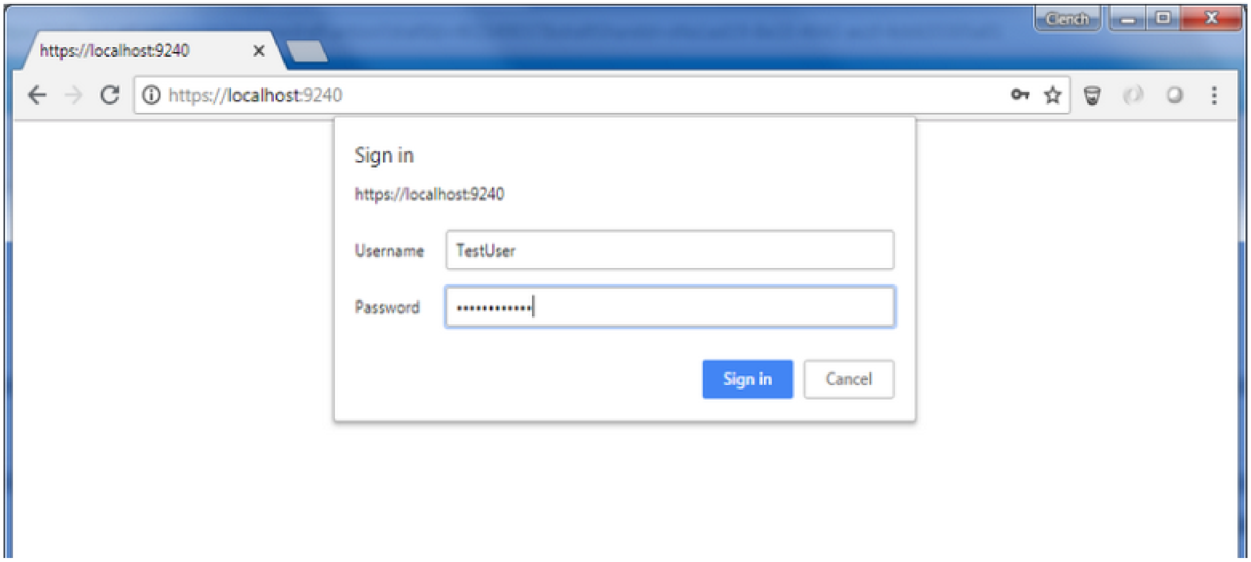
d. Navigate to `Installation_Dir \InternalDataStore\sagconfig` and open the file sg_internal_users.yml.

e. Add the username and password as follows:

```
#keys cannot contain dots
#if you have a username with dots then specify it with username: xxx
Administrator:
  hash: "$2a$12$sm2AEpQx6QNq6YRSYHGCnetiRWKMWrQY/udSSI0dDFZ1r3qo51bzK"
TestUser:
  hash: "$2a$12$Ua1gUiWaW5/b8ohgDqTfg.ruEDNOCsuV9RexlTigNf65TvSn6/Loy"
```

f. Shutdown and restart the API Data Store.

API Data Store now runs on a secure channel on the HTTPS port and requests the basic authentication details.



Username: TestUser

Password: ••••••••••••

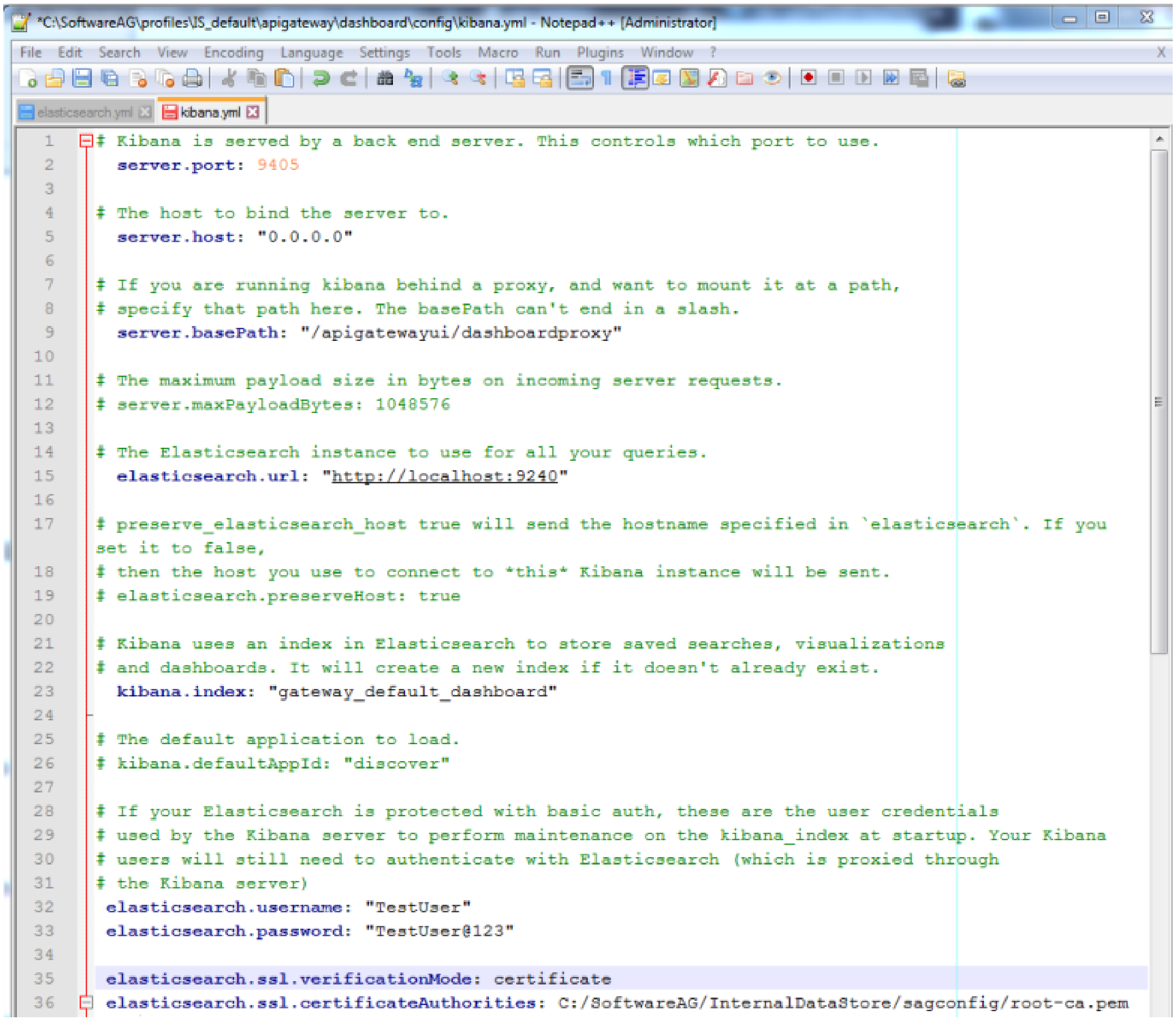
3. Change the Kibana configuration to connect to Elasticsearch.

a. Navigate to `Installation_Dir\profiles\IS_default\apigateway\dashboard\config\` and open the file, kibana.yml.

b. Uncomment the following properties and update them as follows:

- elasticsearch.username: `TestUser`
- elasticsearch.password: `TestUser@123`
- elasticsearch.ssl.verificationMode: `certificate`
- elasticsearch.ssl.certificateAuthorities: *file path of your root-ca.pem certificate*
- elasticsearch.url: https://*hostname*: `9240`

**Sample kibana.yml file**


*C:\SoftwareAG\profiles\IS_default\apigateway\dashboard\config\kibana.yml - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
elasticsearch.yml
kibana.yml

```
# Kibana is served by a back end server. This controls which port to use.
  server.port: 9405

# The host to bind the server to.
  server.host: "0.0.0.0"

# If you are running kibana behind a proxy, and want to mount it at a path,
# specify that path here. The basePath can't end in a slash.
  server.basePath: "/apigatewayui/dashboardproxy"

# The maximum payload size in bytes on incoming server requests.
# server.maxPayloadBytes: 1048576

# The Elasticsearch instance to use for all your queries.
  elasticsearch.url: "http://localhost:9240"

# preserve_elasticsearch_host true will send the hostname specified in `elasticsearch`. If you
set it to false,
# then the host you use to connect to *this* Kibana instance will be sent.
# elasticsearch.preserveHost: true

# Kibana uses an index in Elasticsearch to store saved searches, visualizations
# and dashboards. It will create a new index if it doesn't already exist.
  kibana.index: "gateway_default_dashboard"

# The default application to load.
# kibana.defaultAppId: "discover"

# If your Elasticsearch is protected with basic auth, these are the user credentials
# used by the Kibana server to perform maintenance on the kibana_index at startup. Your Kibana
# users will still need to authenticate with Elasticsearch (which is proxied through
# the Kibana server)
 elasticsearch.username: "TestUser"
 elasticsearch.password: "TestUser@123"

 elasticsearch.ssl.verificationMode: certificate
 elasticsearch.ssl.certificateAuthorities: C:/SoftwareAG/InternalDataStore/sagconfig/root-ca.pem
```

c. Open the uiconfiguration.properties file located at *`Installation_Dir`*`\profiles\IS_default\apigateway\config` and set `apigw.kibana.autostart` to `false`.

4. Change the API Gateway configuration to connect to Elasticsearch.

   a. Navigate to *`Installation_Dir`*`\IntegrationServer\instances\default\packages\WmAPIGateway\config\resources\elasticsearch` and open config.properties file.

   b. Uncomment the following properties and update them as follows:

   ```
   pg.gateway.elasticsearch.http.username=TestUser
   pg.gateway.elasticsearch.http.password=TestUser@123
   pg.gateway.elasticsearch.https.truststore.filepath=Installation_Dir/InternalDataStore
   /sagconfig/truststore.jks
   pg.gateway.elasticsearch.https.truststore.password=2c0820e69e7dd5356576
   pg.gateway.elasticsearch.https.enabled=true
   ```

   c. Start the API Data Store manually.

   d. When API Data Store is up and running, start the Kibana server manually by running the kibana.bat file located at *Installation_Dir*`\profiles\IS_default\apigateway\dashboard\bin`.

   e. Start API Gateway.

      You can now log on, create APIs, and access the Analytics page with the user credentials.

## Configuring Search Guard with self-generated certificates

As an API Provider, if you want to generate your own certificates to use with Search Guard instead of the default certificates that are shipped with API Gateway, you can configure Search Guard with user generated certificates as Step 5. Search Guard provides an offline TLS tool. Use the tool to generate the required certificates for running Search Guard in a production environment.

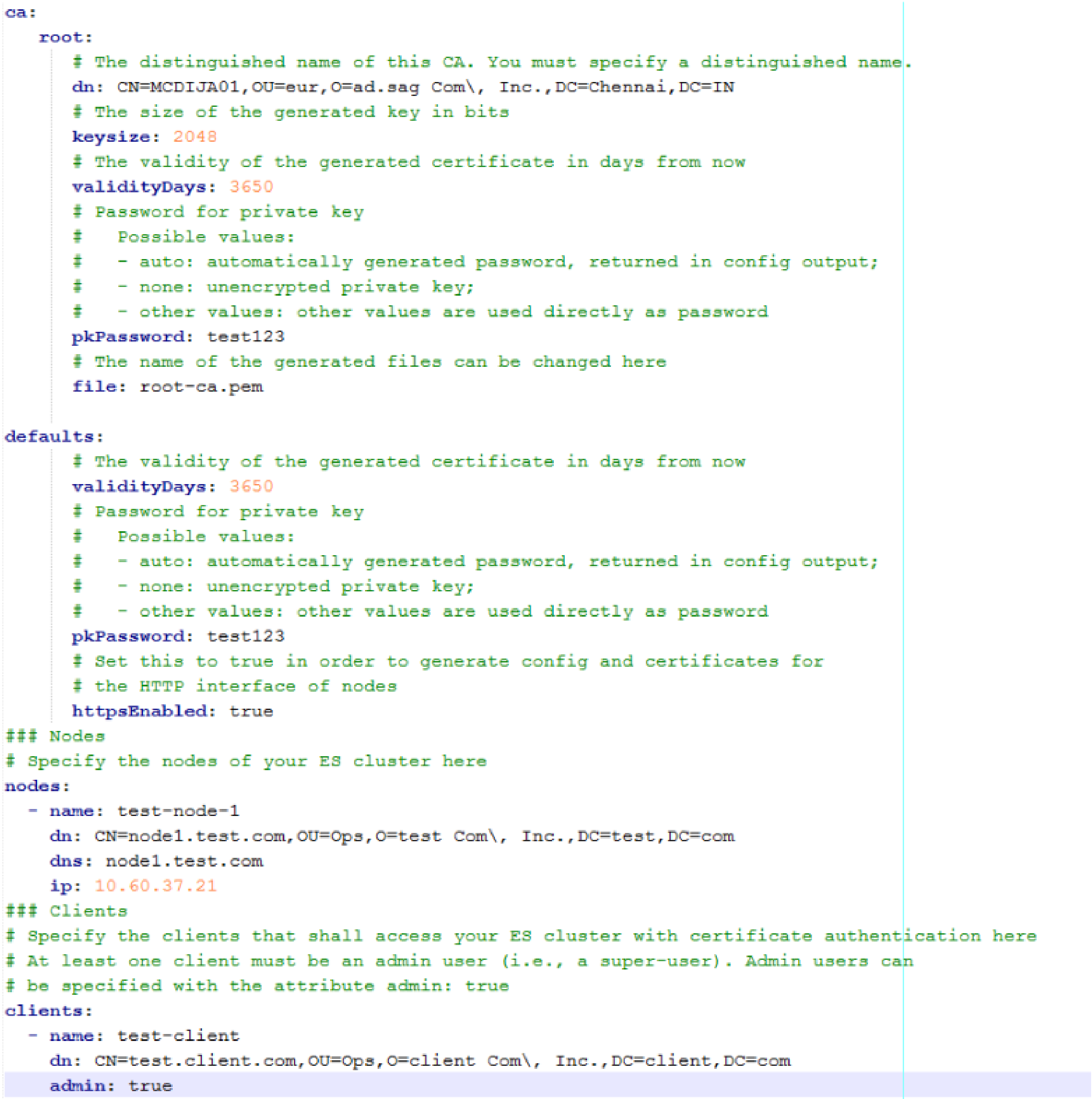
1. Configure Search Guard with user generated certificates.

   a. Download the tool zip file from https://search.maven.org/search?q=a:search-guard-tlstool

   b. Create a YAML file at *Tool Installation Directory*`\config`

      When you run the TLS tool command, it reads the node and certificate configuration settings from this YAML file, and places the generated files in a configured directory.

      **Sample YAML file**

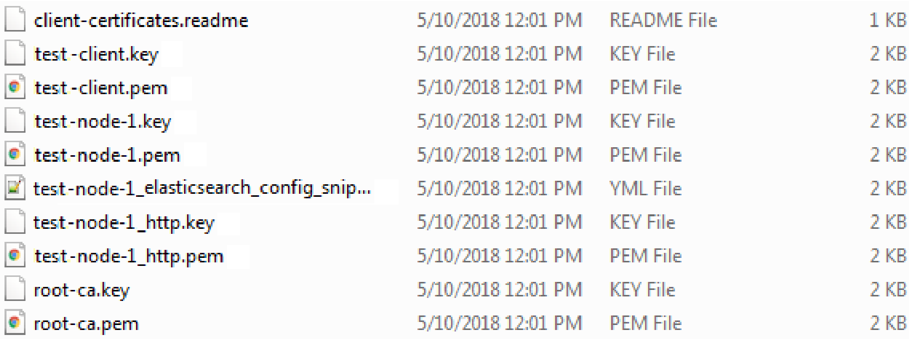


```
ca:
   root:
      # The distinguished name of this CA. You must specify a distinguished name.
      dn: CN=MCDIJA01,OU=eur,O=ad.sag Com\, Inc.,DC=Chennai,DC=IN
      # The size of the generated key in bits
      keysize: 2048
      # The validity of the generated certificate in days from now
      validityDays: 3650
      # Password for private key
      #   Possible values:
      #   - auto: automatically generated password, returned in config output;
      #   - none: unencrypted private key;
      #   - other values: other values are used directly as password
      pkPassword: test123
      # The name of the generated files can be changed here
      file: root-ca.pem

defaults:
      # The validity of the generated certificate in days from now
      validityDays: 3650
      # Password for private key
      #   Possible values:
      #   - auto: automatically generated password, returned in config output;
      #   - none: unencrypted private key;
      #   - other values: other values are used directly as password
      pkPassword: test123
      # Set this to true in order to generate config and certificates for
      # the HTTP interface of nodes
      httpsEnabled: true
### Nodes
# Specify the nodes of your ES cluster here
nodes:
  - name: test-node-1
    dn: CN=node1.test.com,OU=Ops,O=test Com\, Inc.,DC=test,DC=com
    dns: node1.test.com
    ip: 10.60.37.21
### Clients
# Specify the clients that shall access your ES cluster with certificate authentication here
# At least one client must be an admin user (i.e., a super-user). Admin users can
# be specified with the attribute admin: true
clients:
  - name: test-client
    dn: CN=test.client.com,OU=Ops,O=client Com\, Inc.,DC=client,DC=com
    admin: true
```

c. Run the following command to generate the required certificates.

```
Tool Installation Directory/tools/sgtlstool.bat
-c ../config/Demo.yml -ca -crt
```

The generated certificates are placed in the *Tool Installation Directory*`/tools/out` folder.

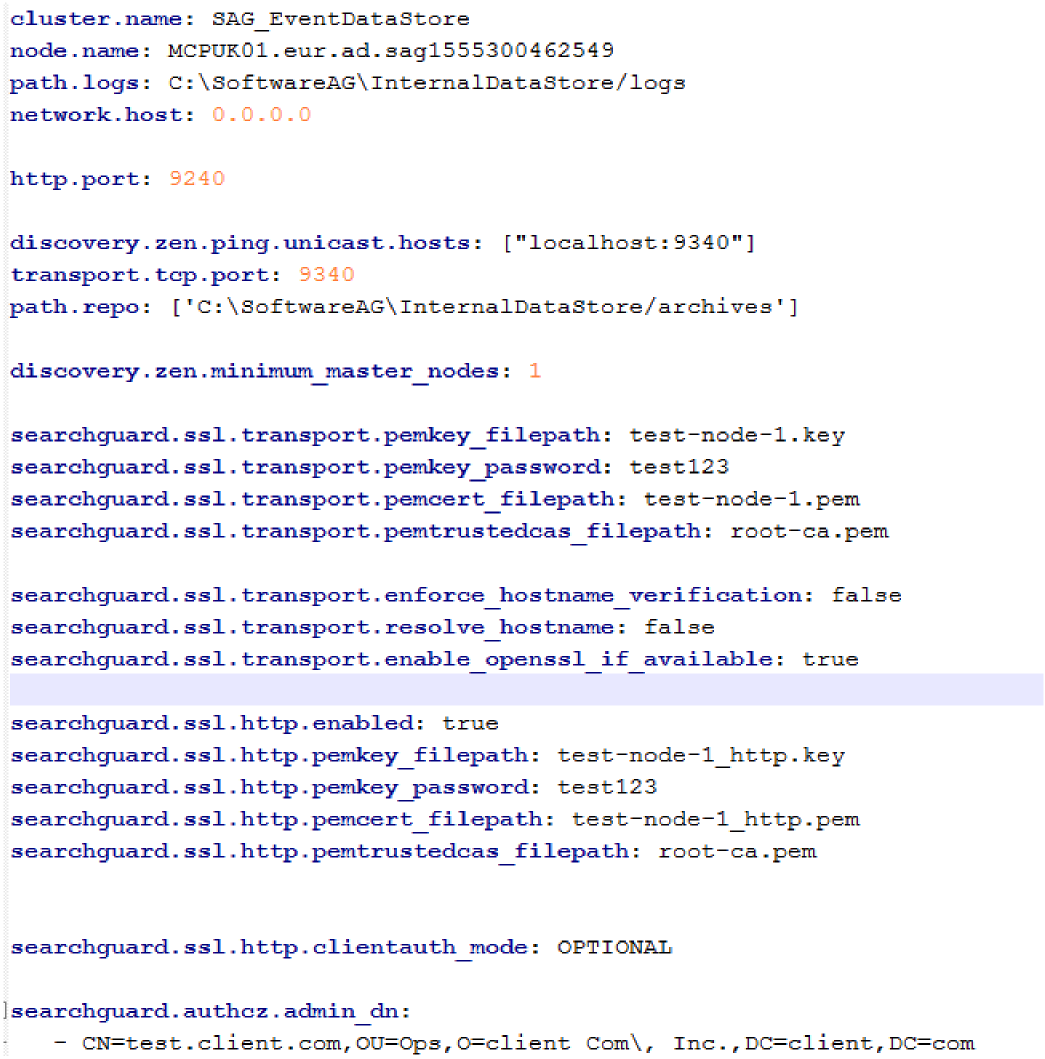


| | | | |
|---|---|---|---|
| client-certificates.readme | 5/10/2018 12:01 PM | README File | 1 KB |
| test-client.key | 5/10/2018 12:01 PM | KEY File | 2 KB |
| test-client.pem | 5/10/2018 12:01 PM | PEM File | 2 KB |
| test-node-1.key | 5/10/2018 12:01 PM | KEY File | 2 KB |
| test-node-1.pem | 5/10/2018 12:01 PM | PEM File | 2 KB |
| test-node-1_elasticsearch_config_snip... | 5/10/2018 12:01 PM | YML File | 2 KB |
| test-node-1_http.key | 5/10/2018 12:01 PM | KEY File | 2 KB |
| test-node-1_http.pem | 5/10/2018 12:01 PM | PEM File | 2 KB |
| root-ca.key | 5/10/2018 12:01 PM | KEY File | 2 KB |
| root-ca.pem | 5/10/2018 12:01 PM | PEM File | 2 KB |

d. Copy the certificates listed below from the folder *`Tool Installation Directory`*`/tools/out` to the *`Installation_Dir`*`/EventDataStore/config` folder.

- test-node-1.key
- test-node-1.pem
- test-node-1_http.pem
- test-node-1_http.key
- test-client.pem
- test-client.key
- root-ca.pem
- root-ca.key

e. Configure the generated certificates in the API Data Store elasticsearch.yml file.



```
cluster.name: SAG_EventDataStore
node.name: MCPUK01.eur.ad.sag1555300462549
path.logs: C:\SoftwareAG\InternalDataStore/logs
network.host: 0.0.0.0

http.port: 9240

discovery.zen.ping.unicast.hosts: ["localhost:9340"]
transport.tcp.port: 9340
path.repo: ['C:\SoftwareAG\InternalDataStore/archives']

discovery.zen.minimum_master_nodes: 1

searchguard.ssl.transport.pemkey_filepath: test-node-1.key
searchguard.ssl.transport.pemkey_password: test123
searchguard.ssl.transport.pemcert_filepath: test-node-1.pem
searchguard.ssl.transport.pemtrustedcas_filepath: root-ca.pem

searchguard.ssl.transport.enforce_hostname_verification: false
searchguard.ssl.transport.resolve_hostname: false
searchguard.ssl.transport.enable_openssl_if_available: true

searchguard.ssl.http.enabled: true
searchguard.ssl.http.pemkey_filepath: test-node-1_http.key
searchguard.ssl.http.pemkey_password: test123
searchguard.ssl.http.pemcert_filepath: test-node-1_http.pem
searchguard.ssl.http.pemtrustedcas_filepath: root-ca.pem


searchguard.ssl.http.clientauth_mode: OPTIONAL

searchguard.authcz.admin_dn:
  - CN=test.client.com,OU=Ops,O=client Com\, Inc.,DC=client,DC=com
```

f. Start API Data Store manually.

   A log message warns that the Search Guard is not initialized after API Data Store is up because the Search Guard is not initialized with the latest certificates.

g. Open a command prompt and change the directory to `Installation_Dir\EventDataStore\plugins\search-guard-7\tools`

h. Run the command

```
sgadmin.bat -cd ..\sagconfig -nhnv -icl -cacert
..\..\..\config\root-ca.pem -cert ..\..\..\config\test-client.pem
-key ..\..\..\config\test-client.key
-keypass your certificate password -p 9340
```

   Done with success log message appears.

i. Shut down and restart API Data Store.

   API Data Store now uses the generated certificates for SSL communication.

## Search Guard Properties

| Property and description |
|---|
| **TRANSPORT ( 2-way authentication is enabled by default)** |
| `searchguard.ssl.transport.keystore_type`<br>Type of keystore.<br>Possible values: JKS, PKCS12<br>Default value: JKS |
| `searchguard.ssl.transport.keystore_filepath`<br>Location of the keystore. |
| `searchguard.ssl.transport.keystore_alias`<br>Keystore entry name if there are more than one entries. |
| `searchguard.ssl.transport.keystore_password`<br>Password to access keystore. |
| `searchguard.ssl.transport.truststore_type`<br>Type of truststore.<br>Possible values: JKS, PKCS12<br>Default value: JKS |
| `searchguard.ssl.transport.truststore_filepath`<br>Location of the truststore. |
| `searchguard.ssl.transport.truststore_alias`<br>Truststore entry name if there are more than one entries. |
| `searchguard.ssl.transport.truststore_password`<br>Password to access truststore. |
| `searchguard.ssl.transport.enforce_hostname_verification`<br>If `true`, the hostname mentioned in certificate is validated. Set this as `false` if you are using the general purpose self signed certificates.<br>Possible values: true, false<br>Default value: true |

| Property and description |
|---|
| `searchguard.ssl.transport.resolve_hostname`<br>If `true`, the hostname is resolved against the DNS server. Set this as `false` if you are using general purpose self signed certificates.<br>**Note:**<br>This is applicable only if the property searchguard.ssl.transport.enforce_hostname_verification is `true`.<br>Possible values: true, false<br>Default value: true |
| `searchguard.ssl.transport.enable_openssl_if_available`<br>Use if OpenSSL is available instead of JDK SSL.<br>Possible values: true, false<br>Default value: true |
| **HTTP** |
| `searchguard.ssl.http.enabled`<br>Set this to `true` to enable SSL for a REST interface ( HTTP).<br>Possible values: true, false<br>Default value: true |
| `searchguard.ssl.http.keystore_type`<br>Type of keystore.<br>Possible values: JKS, PKCS12<br>Default value: JKS |
| `searchguard.ssl.http.keystore_filepath`<br>Location of the keystore. |
| `searchguard.ssl.http.keystore_alias`<br>Keystore entry name if there are more than one entries. |
| `searchguard.ssl.http.keystore_password`<br>Password to access keystore. |
| `searchguard.ssl.http.truststore_type`<br>Type of truststore. |

| Property and description |
|---|
| Possible values: JKS, PKCS12<br>Default value: JKS |
| `searchguard.ssl.http.truststore_filepath`<br>Location of the truststore. |
| `searchguard.ssl.http.truststore_alias`<br>Truststore entry name if there are more than one entries. |
| `searchguard.ssl.http.truststore_password`<br>Password to access truststore. |
| `searchguard.ssl.http.clientauth_mode`<br>Option to enable two-way authentication.<br>Possible values:<br>- REQUIRE : Requests for the client certificate.<br>- OPTIONAL : Used if client certificate is available.<br>- NONE : Ignores client certificate even if it is available.<br>Default value: OPTIONAL |
| **Search Guard Admin** |
| `searchguard.authcz.admin_dn`<br>Search Guard maintains all the data in the index `searchguard`. This is accessible to only users ( client certificate passed in `sdadmin` command) configured here. |
| **Miscellaneous** |
| `searchguard.cert.oid`<br>All certificates used by the nodes at the transport level need to have the **oid** field set to a specific value. Search Guard checks this oid value to identify if an incoming request comes from a trusted node in the cluster or not. In the former case, all actions are allowed. In the latter case, privilege checks apply. Additionally, the oid is also checked whenever a node wants to join the cluster.<br>Default value: '1.2.3.4.5.5' |
| `searchguard.config_index_name`<br>Index where all the security configuration is stored. Currently, non-configurable.<br>Default value: searchguard |

| Property and description |
|---|
| `searchguard.enable_snapshot_restore_privilege`,<br>`searchguard.check_snapshot_restore_write_privileges`<br>Enables user privileges, which a user requires to perform snapshot and restore operations. |
| `searchguard.enterprise_modules_enabled`<br>Specifies the license type. If you are using a trail version, set the property to `false`. |
| `searchguard.restapi.roles_enabled`<br>Specifies which Search Guard roles can access the REST Management API to perform changes to the configuration. |

## Creating a Custom Keystore with Self-Signed Certificates

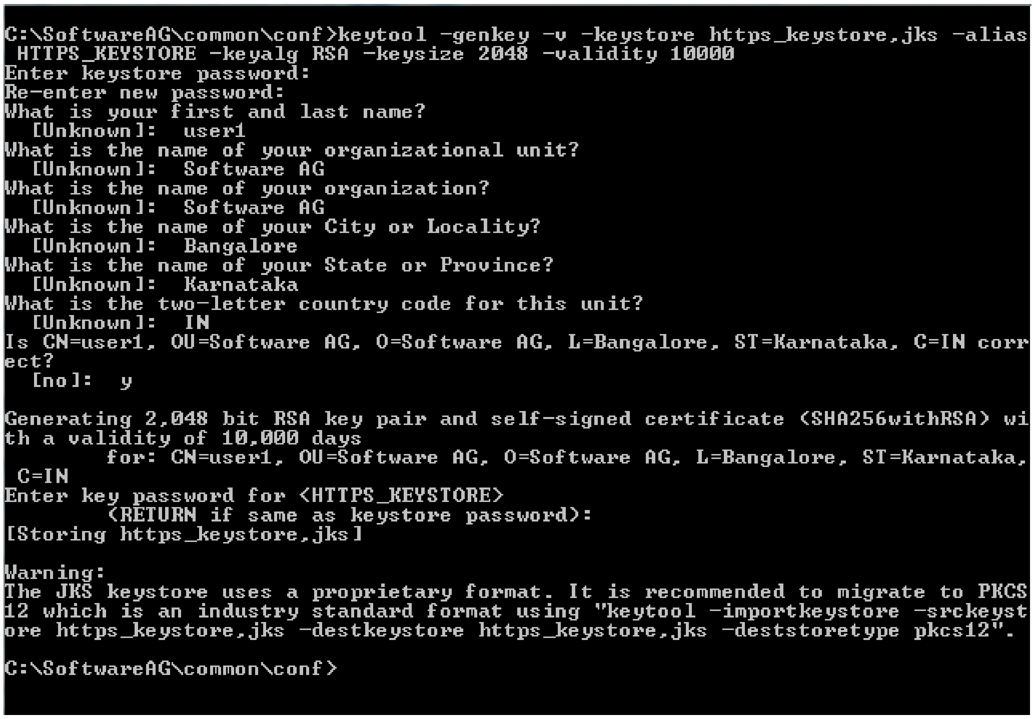
You have to perform this procedure if your organization does not have policies and procedures in place regarding the generation and use of digital certificates and certificate chains, including the use of certificates signed by a CA but want to generate a self-signed certificate and import them into the keystore and truststore.

1. Create a new keystore with a self-signed certificate.

    a. 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:**

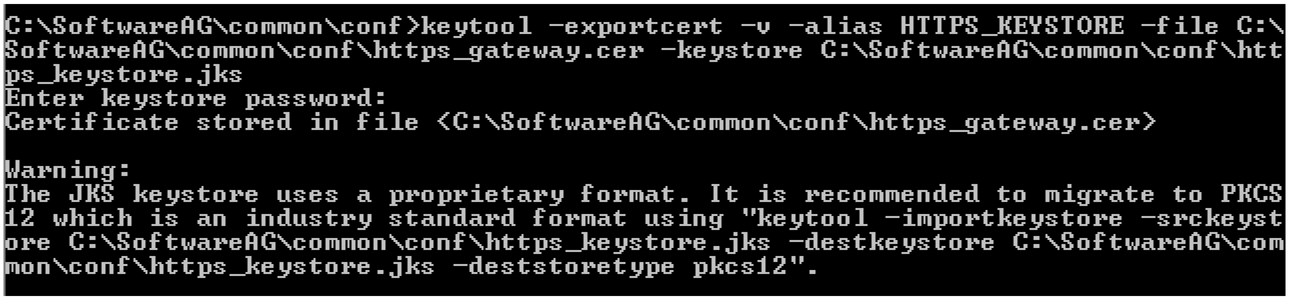


```
C:\SoftwareAG\common\conf>keytool -genkey -v -keystore https_keystore.jks -alias
 HTTPS_KEYSTORE -keyalg RSA -keysize 2048 -validity 10000
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  user1
What is the name of your organizational unit?
  [Unknown]:  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 corr
ect?
  [no]:  y

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) wi
th 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 PKCS
12 which is an industry standard format using "keytool -importkeystore -srckeyst
ore https_keystore.jks -destkeystore https_keystore.jks -deststoretype pkcs12".

C:\SoftwareAG\common\conf>
```

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\htt
ps_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 PKCS
12 which is an industry standard format using "keytool -importkeystore -srckeyst
ore C:\SoftwareAG\common\conf\https_keystore.jks -destkeystore C:\SoftwareAG\com
mon\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\`.

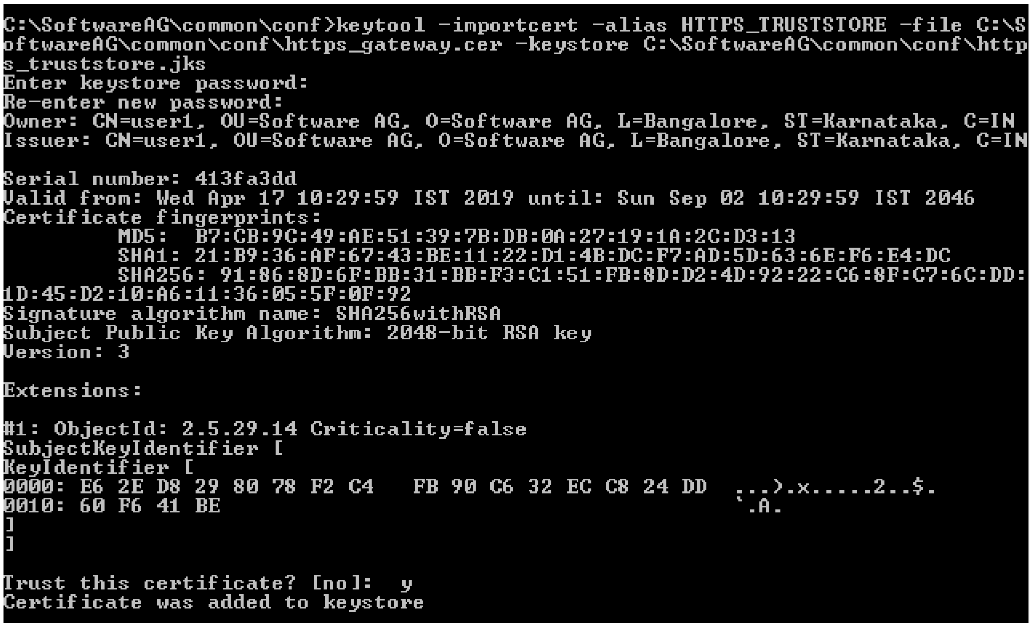
2. Create a truststore and import the generated certificate.

   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:\S
oftwareAG\common\conf\https_gateway.cer -keystore C:\SoftwareAG\common\conf\http
s_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:63:6E: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\`.

# 5 API Gateway Configuration with Command Central

## Overview

Command Central allows users who have administration privileges to administer API Gateway and API Data Store.

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.
   ```

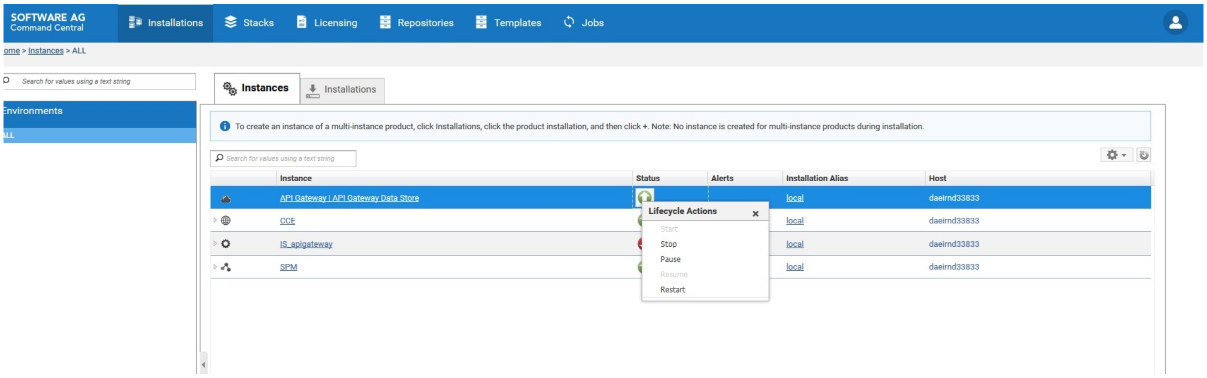
   d. Apply the template.

   ```
   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,

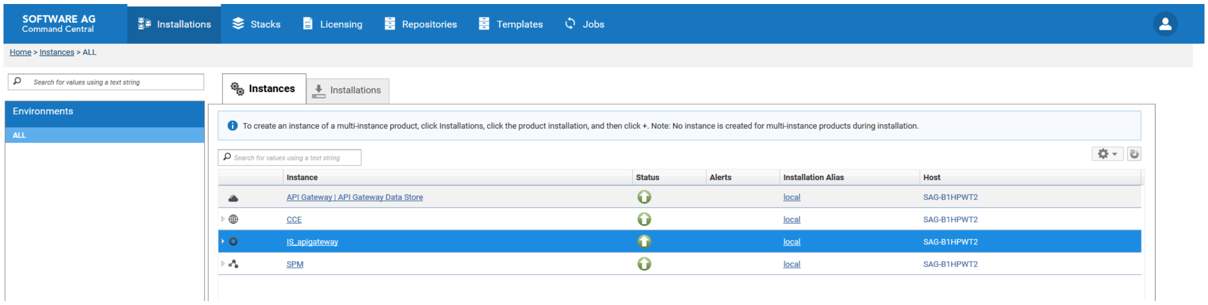
- API Gateway > API Data Store contains details about default Elasticsearch shipped with API Gateway.



To create an instance of a multi-instance product, click Installations, click the product installation, and then click +. Note: No instance is created for multi-instance products during installation.

| Instance | Status | Alerts | Installation Alias | Host |
|---|---|---|---|---|
| API Gateway \| API Gateway Data Store | | | local | daeirnd33833 |
| CCE | | | local | daeirnd33833 |
| IS_apigateway | | | local | daeirnd33833 |
| SPM | | | local | daeirnd33833 |

- IS_<profile> contains details about API Gateway, Digital Event Services, Event Routing, and Integration Server.



To create an instance of a multi-instance product, click Installations, click the product installation, and then click +. Note: No instance is created for multi-instance products during installation.

| Instance | Status | Alerts | Installation Alias | Host |
|---|---|---|---|---|
| API Gateway \| API Gateway Data Store | | | local | SAG-B1HPWT2 |
| CCE | | | local | SAG-B1HPWT2 |
| IS_apigateway | | | local | SAG-B1HPWT2 |
| SPM | | | local | SAG-B1HPWT2 |

## Manage API Data Store Configurations in Command Central

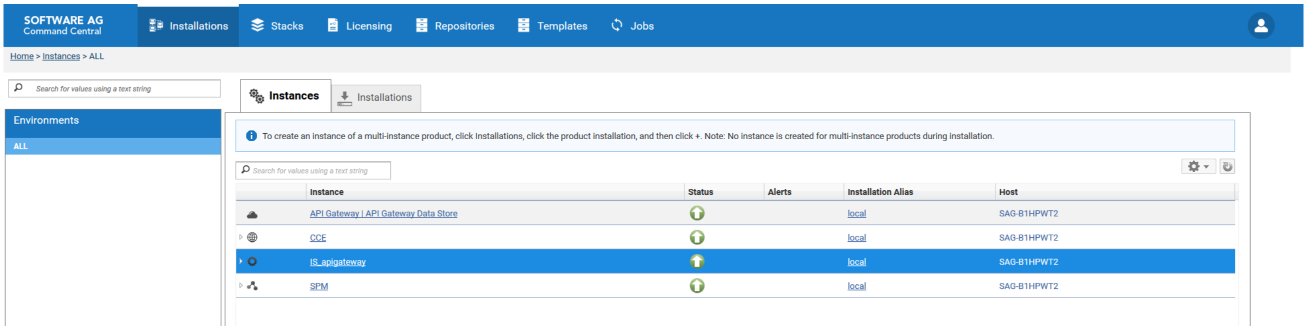
Command Central lists API Gateway and API Data Store shipped with API Gateway. API Gateway stores all its core and analytics data in this Data Store by default. You can start, stop, and restart API Data Store from Command Central. You can also manage Clustering details, Keystores, Ports, Properties, and Truststores.

This section describes the following administering tasks for API Data Store:

- "Starting and Stopping API Data Store in Command Central" on page 28
- "Changing the API Data Store HTTP Port" on page 30
- "Changing the API Data Store TCP Port" on page 32
- "Configuring an API Data Store Cluster" on page 32
- "Configuring Elasticsearch Properties" on page 36

## 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.



To create an instance of a multi-instance product, click Installations, click the product installation, and then click +. Note: No instance is created for multi-instance products during installation.

| Instance | Status | Alerts | Installation Alias | Host |
|---|---|---|---|---|
| API Gateway \| API Gateway Data Store | | | local | SAG-B1HPWT2 |
| CCE | | | local | SAG-B1HPWT2 |
| IS_apigateway | | | local | SAG-B1HPWT2 |
| SPM | | | 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:

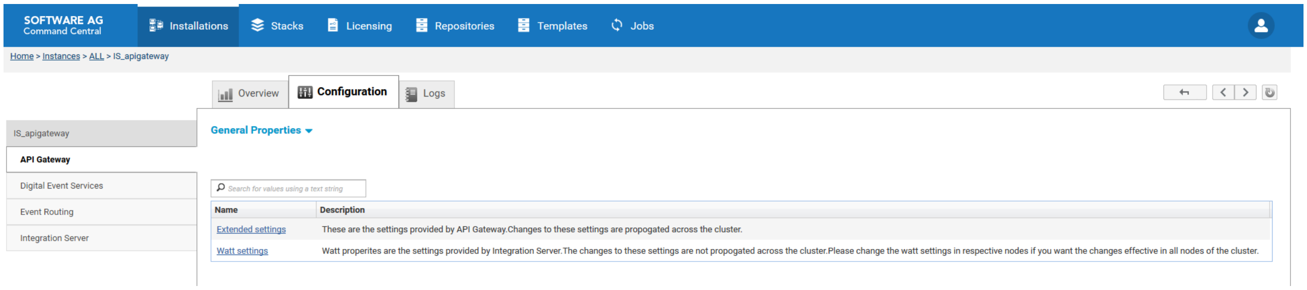
- Clusters
- Elasticsearch instances
- General and extended properties
- Keystores
- Kibana instances
- License keys
- Loggers
- Ports
- Truststores

## 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**.



| Name | Description |
|---|---|
| Extended settings | These are the settings provided by API Gateway.Changes to these settings are propogated across the cluster. |
| Watt settings | Watt properites are the settings provided by Integration Server.The changes to these settings are not propogated across the cluster.Please change the watt settings in respective nodes if you want the changes effective in all nodes of the cluster. |

6. Click **Watt Settings**. The properties are listed as key value pairs.
7. Make the required changes.

8. Save your changes.

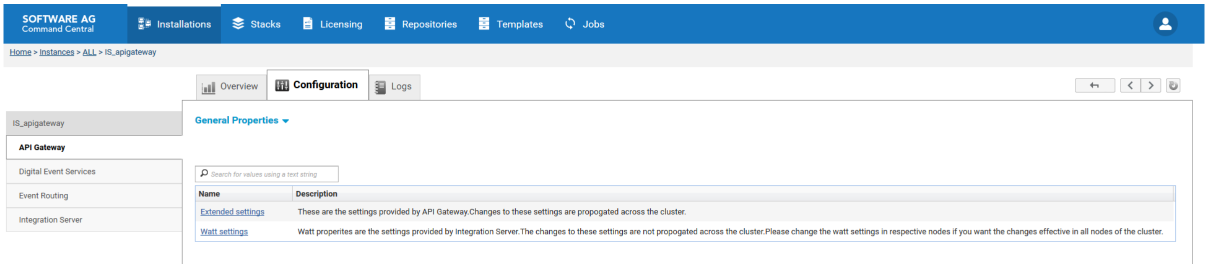
## Configuring Keystores

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.



| Name | Description |
|---|---|
| Extended settings | These are the settings provided by API Gateway.Changes to these settings are propogated across the cluster. |
| Watt settings | Watt properites are the settings provided by Integration Server.The changes to these settings are not propogated across the cluster.Please change the watt settings in respective nodes if you want the changes effective in all nodes of the cluster. |

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=youekeystorepath
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+GWlrw==}uSFvFjWALKWdMOAjuwGpVA=='
                    Key:
                    - '@alias': partner1
                      Password: '{AES/CBC/PKCS5Padding}
                      {VPQ5ojZEZgzUR7x0WfO317ROK+bxvMyjSCSigoBiAEo=}
                      {+96qyCFXAiXg2gX3CzdIWA==}7kAeXaZcieuJuRefScC0Ig=='
                    - '@alias': partner2
                      Password: '{AES/CBC/PKCS5Padding}
                      {4cu7D8zZ+Bng2CvoeX71tlb1TSv5yKwqNAXjDN1yLKI=}
                      {wOE8hwyO2s5BlSZV1tKtNA==}mIVtB9dVL8TCVb35zQGJaA=='
                    - '@alias': policygateway
                      Password: '{AES/CBC/PKCS5Padding}
                      {PWBrBO5D5w6KSdloz8q8yTcrVThiZEbyPhre1u7gXb4=}
                      {FuESDHiSW1rXqmBIfL7P7g==}/hMP4Bzp0hmCF2Jlrsy00w=='
                  ExtendedProperties:
                    Property:
                    - '@name': fileContent
                      $:
```

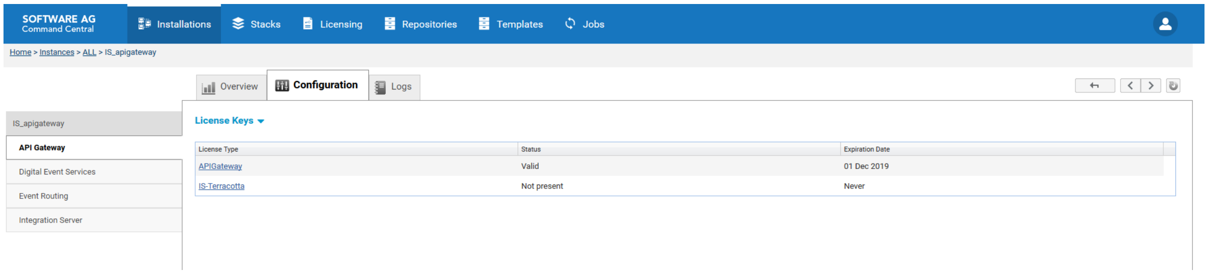
## Configuring Licenses

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.



| 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**.

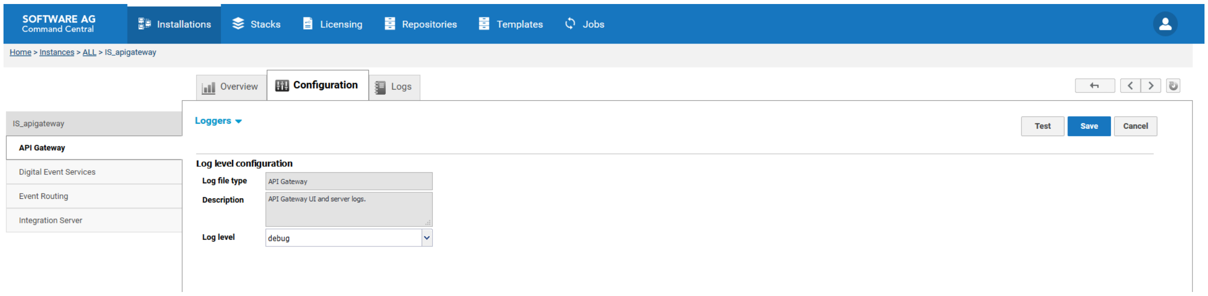
2. Select **Loggers** from the drop-down menu.

   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.



Log level configuration

Log file type: API Gateway
Description: API Gateway UI and server logs.
Log level: debug

   c. Click **Save**.

## Configuring HTTP Port

This section provides information about configuring HTTP ports available in API Gateway.

### To configure the HTTP port

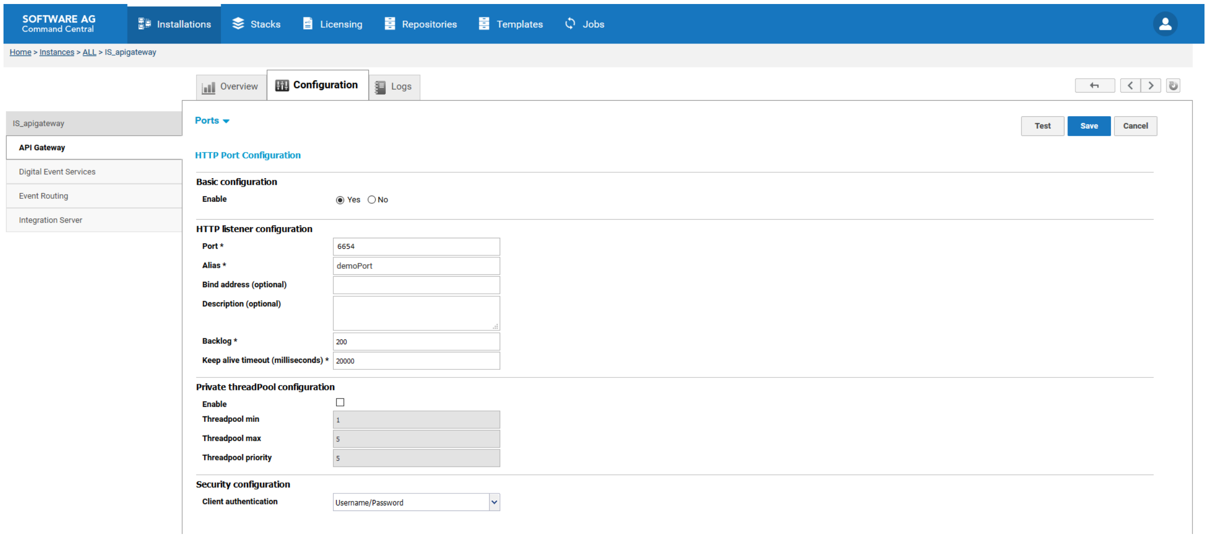
1. In Command Central, navigate to **Environments** > **Instances** > **All** > **API Gateway** > **Configuration**.

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.



HTTP Port Configuration

Basic configuration

Enable ● Yes ○ No

HTTP listener configuration

Port * 6654

Alias * demoPort

Bind address (optional)

Description (optional)

Backlog * 200

Keep alive timeout (milliseconds) * 20000

Private threadPool configuration

Enable ☐

Threadpool min 1

Threadpool max 5

Threadpool priority 5

Security configuration

Client authentication Username/Password

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**.

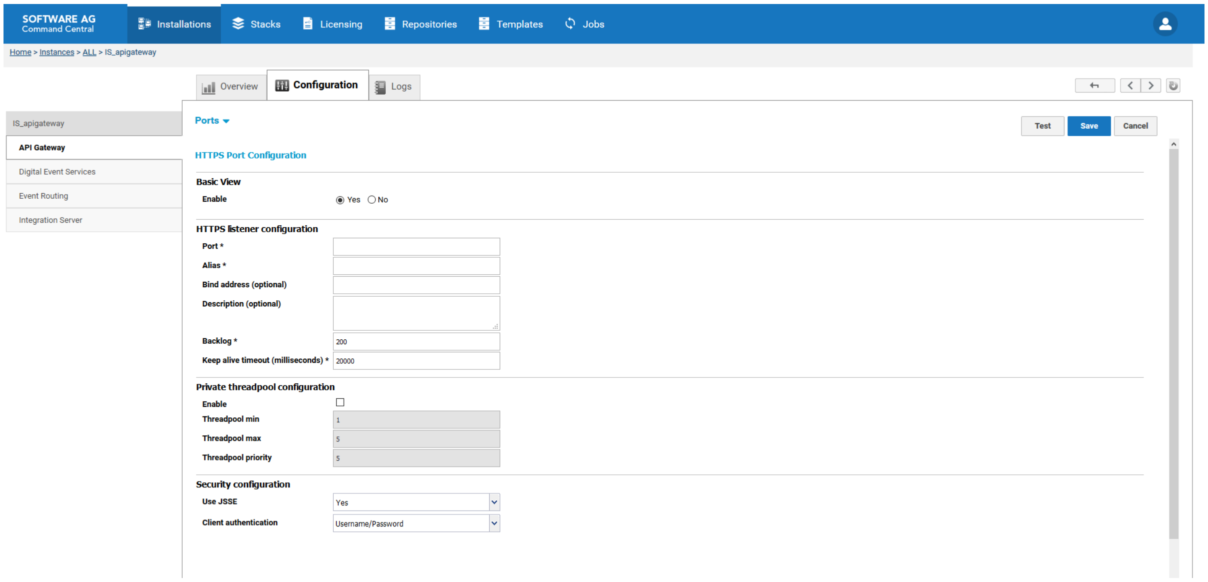
2. Select **Ports** from the drop-down menu.

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.



HTTPS Port Configuration

Basic View

Enable ● Yes ○ No

HTTPS listener configuration

Port *

Alias *

Bind address (optional)

Description (optional)

Backlog * 200

Keep alive timeout (milliseconds) * 20000

Private threadpool configuration

Enable ☐

Threadpool min 1

Threadpool max 5

Threadpool priority 5

Security configuration

Use JSSE Yes

Client authentication Username/Password

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': certficateAliases
                        $:
addtrustclass1ca,addtrustexternalca,addtrustqualifiedca,baltimorecodesigningca,baltimorecybertrustca,
comodobaaaca,entrust2048ca,entrustclientca,entrustglobalclientca,entrustgsslca,entrustsslca,equifaxsecureca,equifaxsecureebusinessca1,
equifaxsecureebusinessca2,equifaxsecureglobalebusinessca1,geotrustglobalca,godaddyclass2ca,gtecybertrust5ca,gtecybertrustca,
gtecybertrustglobalca,lhca,partner1,partner2,policygateway,soneraclass1ca,soneraclass2ca,starfieldclass2ca,synapse,
thawtepersonalbasicca,thawtepersonalfreemailca,thawtepersonalpremiumca,thawtepremiumserverca,thawteserverca,
utndatacorpsgcca,utnuserfirstclientauthemailca,utnuserfirsthardwareca,utnuserfirstobjectca,valicertclass2ca,
verisignclass1ca,verisignclass1g2ca,verisignclass1g3ca,verisignclass2ca,verisignclass2g2ca,verisignclass2g3ca,
verisignclass3ca,verisignclass3g2ca,verisignclass3g3ca,verisignserverca,webm test ca
                      - '@name': isLoaded
                        $: 'true'
                      - '@name': fileContent
                        $:
/u3+7QAAAAIAAAAxAAAAAgAMd2VibSB0ZXN0IGNhAAABSLIi/poABVguNTA5AAADazCCA2cwggJPo
                        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.

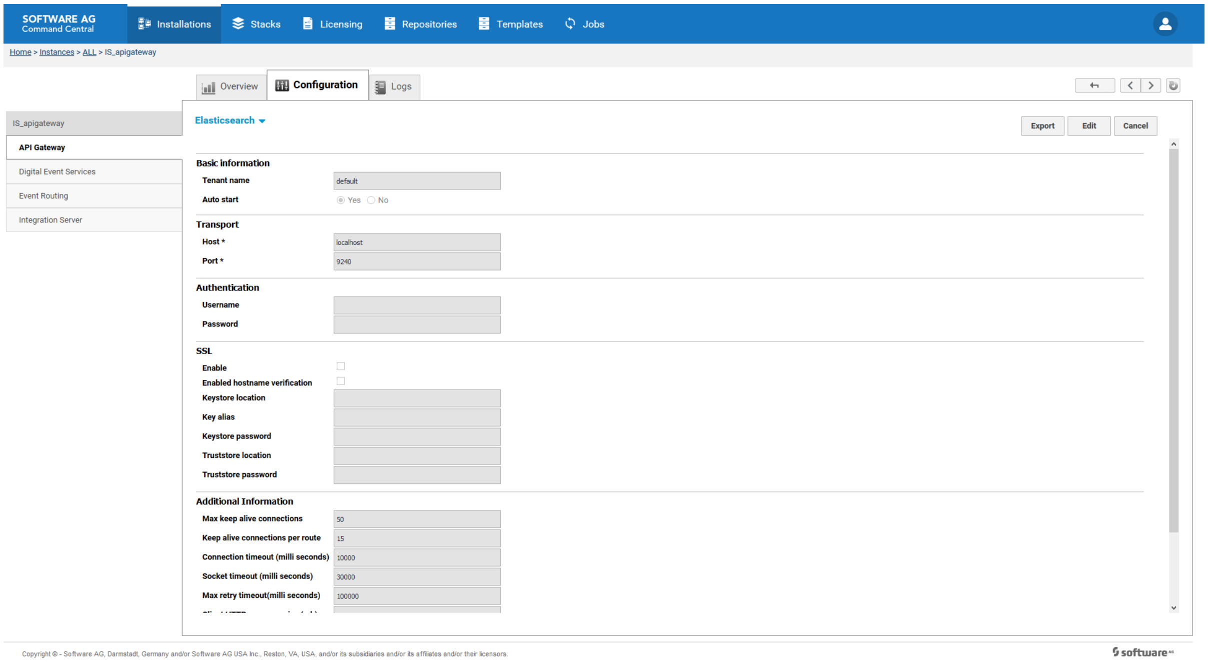
6. If the Elasticsearch is protected with basic authorization, provide the user name and password in the **Authentication** section.

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.



**Basic information**

| | |
|---|---|
| Tenant name | default |
| Auto start | ● Yes ○ No |

**Transport**

| | |
|---|---|
| Host * | localhost |
| Port * | 9240 |

**Authentication**

| | |
|---|---|
| Username | |
| Password | |

**SSL**

| | |
|---|---|
| Enable | ☐ |
| Enabled hostname verification | ☐ |
| Keystore location | |
| Key alias | |
| Keystore password | |
| Truststore location | |
| Truststore password | |

**Additional Information**

| | |
|---|---|
| Max keep alive connections | 50 |
| Keep alive connections per route | 15 |
| Connection timeout (milli seconds) | 10000 |
| Socket timeout (milli seconds) | 30000 |
| Max retry timeout(milli seconds) | 100000 |

Copyright © - 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.

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.

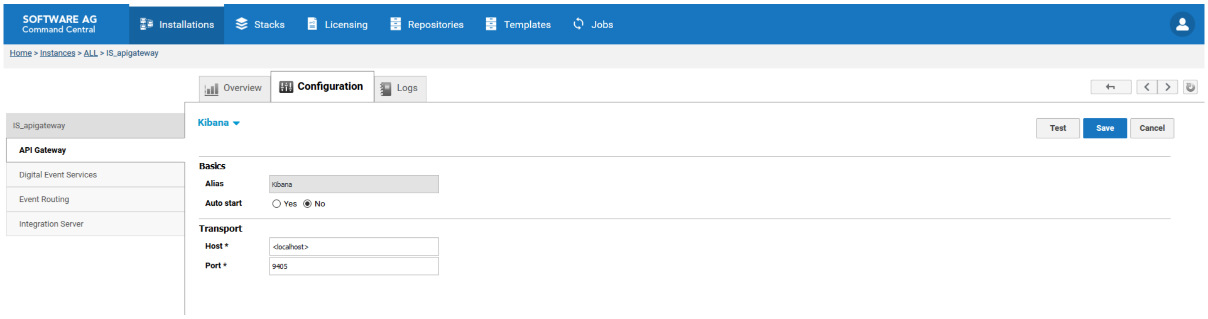
   The Kibana instances list appears.

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.



Alias: Kibana
Auto start: ○ Yes ● No
Host *: <localhost>
Port *: 9405

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**

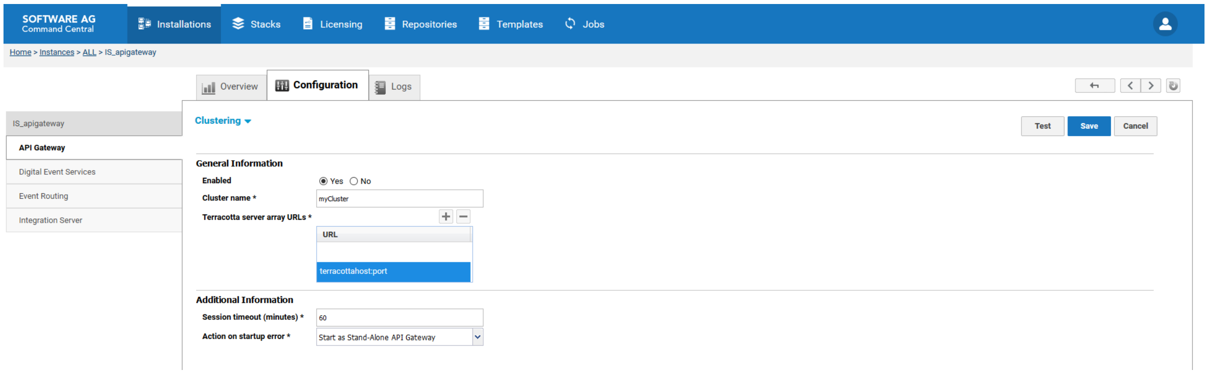
1. In Command Central, navigate to **Environments** > **Instances** > **All** > **API Gateway** > **Configuration**.

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.



General Information

Enabled ● Yes ○ No

Cluster name * myCluster

Terracotta server array URLs *

| URL |
|---|
| terracottahost:port |

Additional Information

Session timeout (minutes) * 60

Action on startup error * Start as Stand-Alone API Gateway

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}
```

# 6 Docker Configuration

## Overview

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` | *Optional*. Name of base image upon which the new image is built.<br>Default: centos:7 |
| `-Dinstance.name` | *Optional*. IS instance name to include in the image.<br>Default: default |
| `-Dport.list` | *Optional*. Comma-separated list of the ports on the instance to expose in the image.<br>Default: 5555,9999 |
| `-Dpackage.list` | *Optional*. Comma-separated list of Wm packages on the instance to include in the image.<br>Default: all (this includes all the Wm packages and the Default package) |
| `-Dinclude.jdk` | *Optional*. Whether to include the Integration Server JDK (true) or JRE (false) in the image.<br>Default: true |
| `-Dfile.name` | *Optional*. File name for the generated docker file.<br>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 |
|---|---|
| `-Dfile.name` | *Optional*. File name of the Docker file to use to build the Docker image.<br>Default: Dockerfile_IS |
| `-Dimage.name` | *Optional*. Name of the generated Docker image.<br>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 |
|---|---|
| `--instance.name` | *Optional*. API Gateway instance to include in the image.<br>Default: default |
| `--port.list` | Comma-separated list of the ports on the instance to expose in the image.<br>Default: 9072 |
| `--base.image` | Name of the base Integration Server image upon which this image should be built.<br>Default: is:micro |
| `--file.name` | *Optional*. File name for the generated Docker file.<br>Default: Dockerfile_IS_APIGW |
| `--target.configuration` | *Optional*. Target configuration for which Dockerfile is created.<br>Not specifying any value builds a Dockerfile for the Docker and Kubernetes environments.<br>Specifying the value `OpenShift` builds a Dockerfile for an OpenShift environment.<br>**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` | *Optional*. Name of the base operating system image upon which this image is built if the `--target.configuration` is set to `OpenShift`.<br>Default: centos:7<br>**Note:** The value of this parameter has to be aligned with the one specified for `-Dimage.name` in Step 1. |

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`.

4. 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 |
|---|---|
| `instance.name` | Optional. API Gateway instance to include in the image.<br>Default: default |
| `file.name` | File name of the Docker file to use to build the Docker image.<br>Default: Dockerfile_IS_APIGW |
| `image.name` | Optional. Name for the generated Docker image that contains the custom packages.<br>Default: is:apigw |

The image is stored in the local registry of the Docker host. To check the image, run the 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

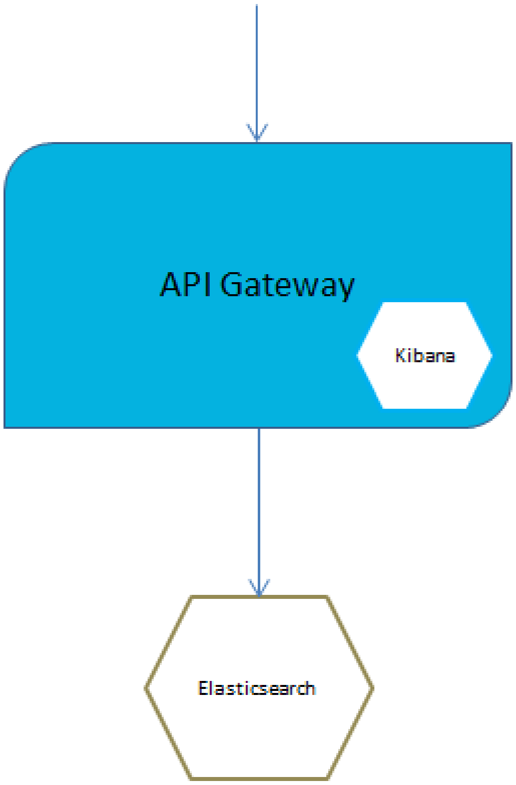
- `saveImage`: To save an API Gateway image to a file (creating a tar ball from an image)
- `loadImage`: To load an image to a Docker registry (loading an image into a Docker registry from tar ball)

# 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:7.13.0
   ```

   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=
   ```

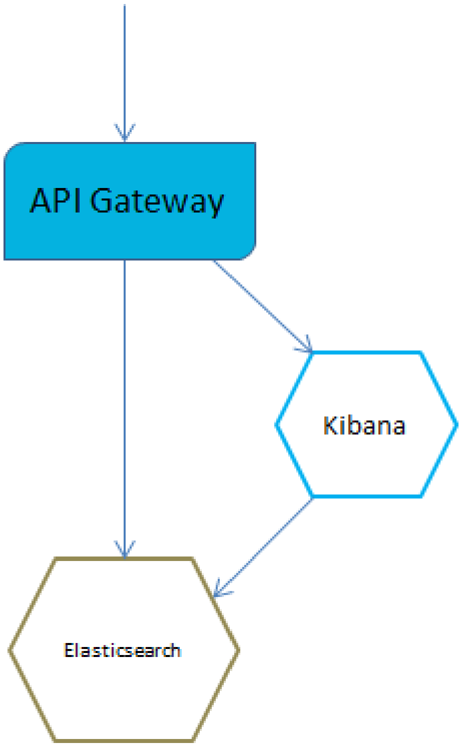
   You can specify the Elasticsearch properties to modify the property files on the container startup.

   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:7.13.0
   ```

   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 "Configuring an API Gateway Cluster" on page 55.

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 195.

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.

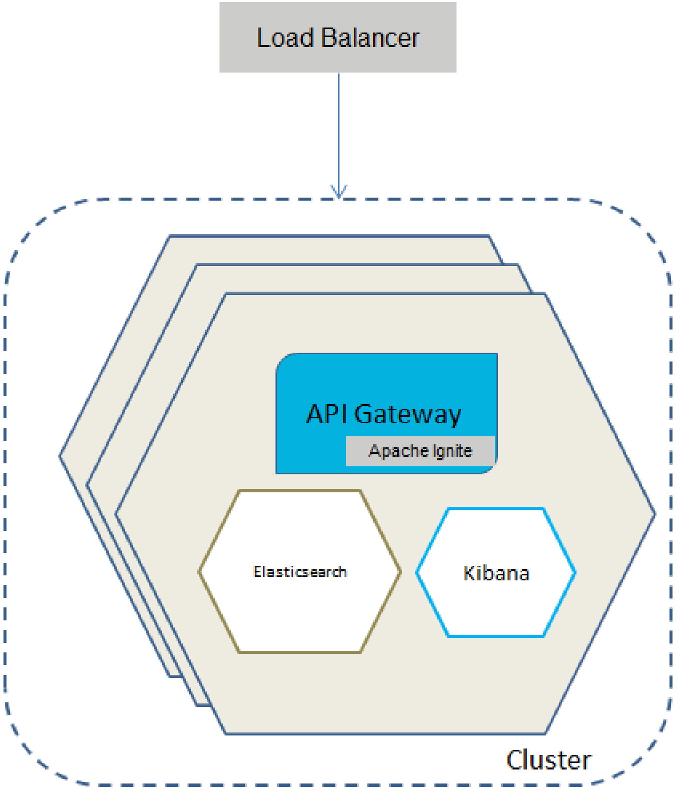
## Clustered all-in-one containers that consist of API Gateway, Kibana and API Data Store

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.

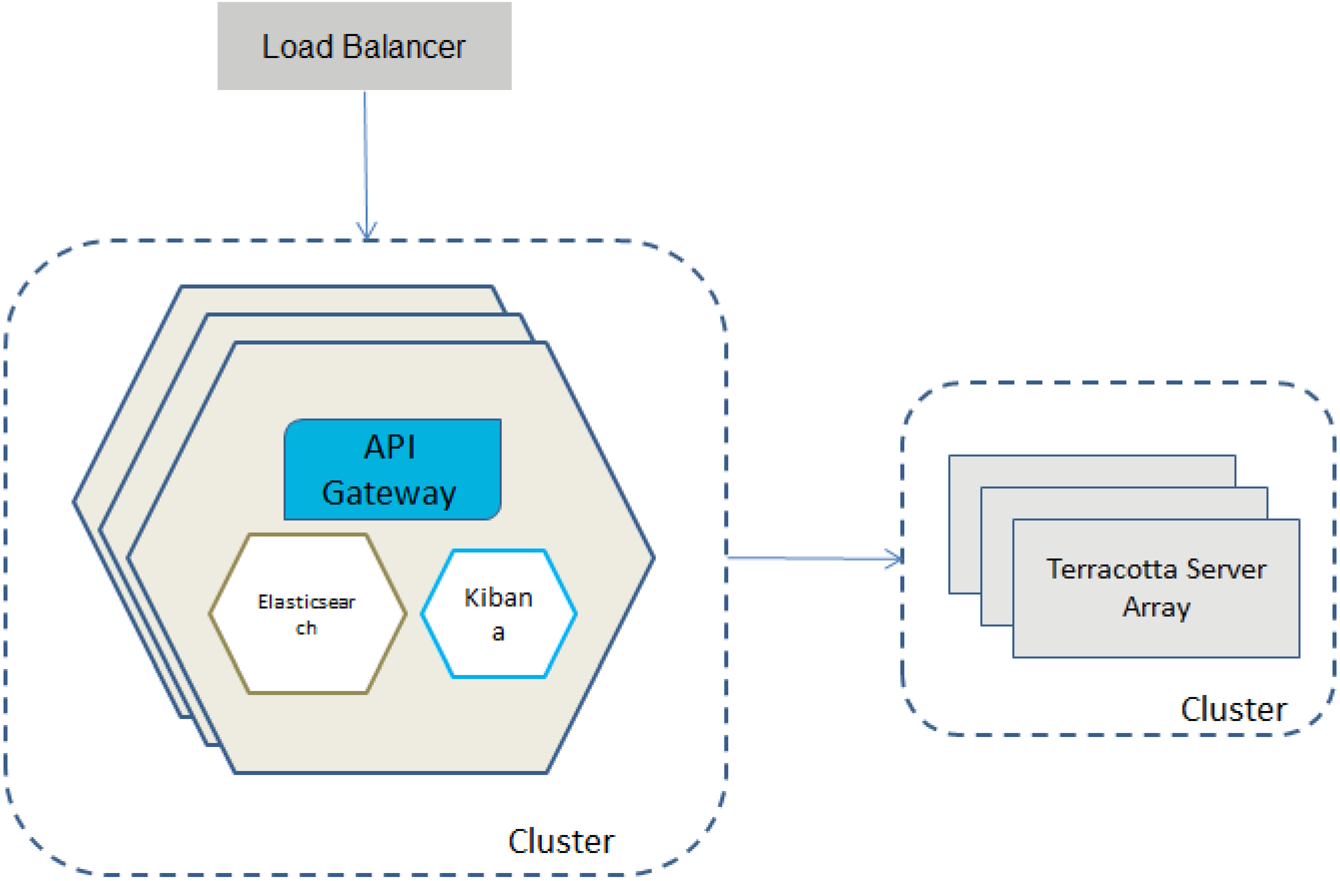
A sample environment file when clustering is done through Apache Ignite looks as follows.

```
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.

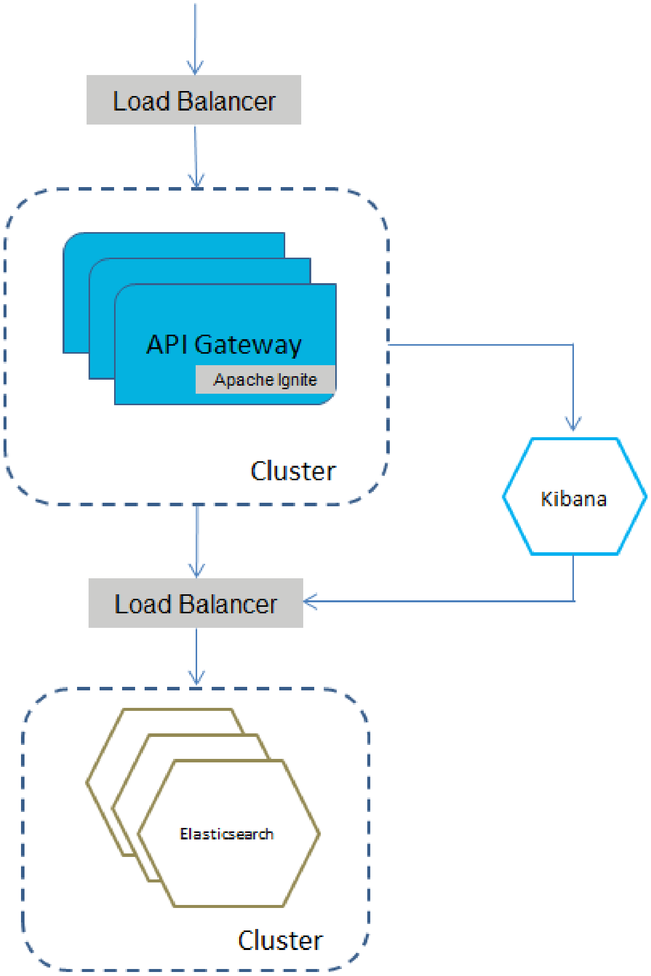
```
apigw_cluster_tsaUrls=tc:9510
apigw_terracotta_license_filename=terracotta-license.key
apigw_cluster_discoverySeedHosts=apigw1:9340,apigw2:9340,apigw3:9340
apigw_cluster_initialMasterNodes=apigw1_master
```

## 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.

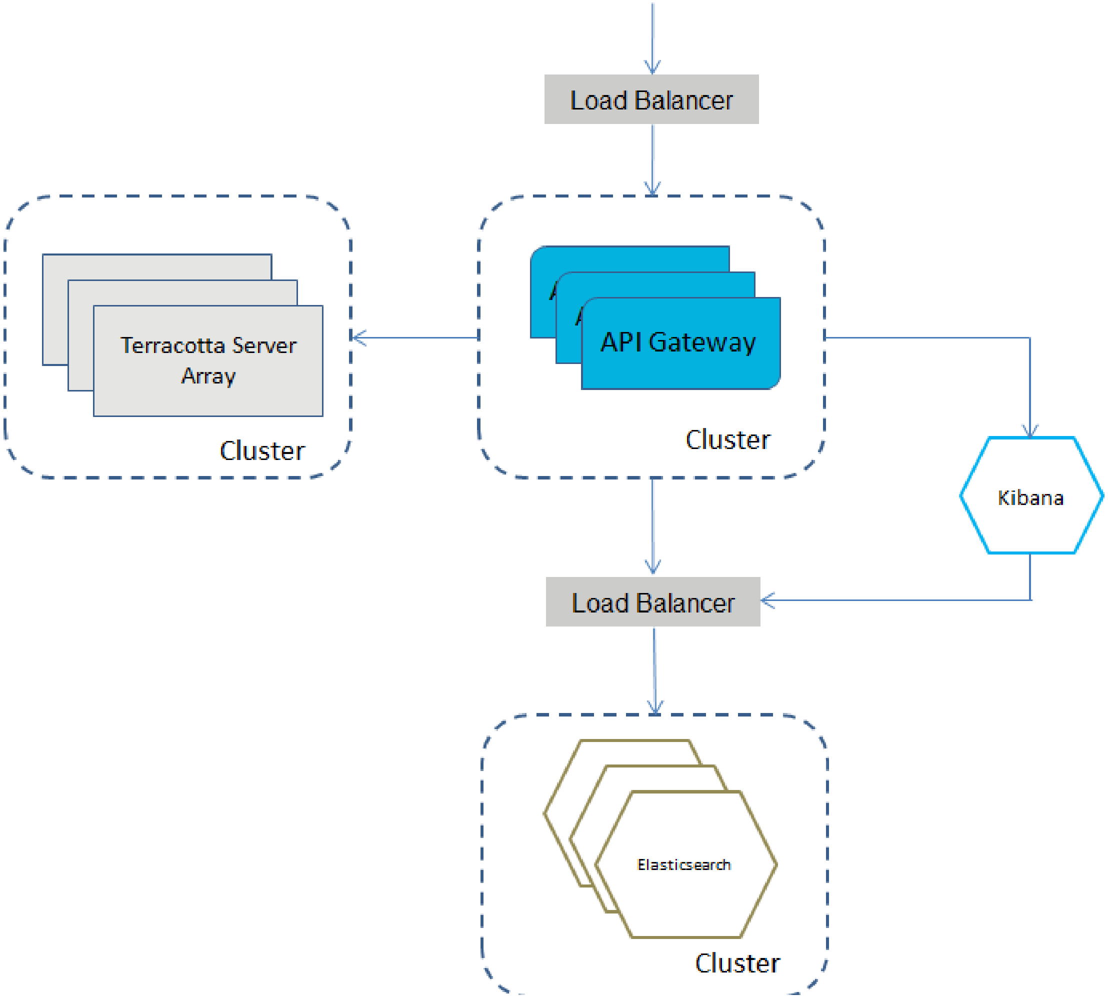
```
apigw_cluster_aware=true
apigw_cluster_name=name
apigw_cluster_ignite_hostnames=comma-separated list of host names
apigw_cluster_ignite_discoveryPort=port
apigw_cluster_ignite_communicationPort=port
apigw_elasticsearch_hosts=host:port
```

```
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.

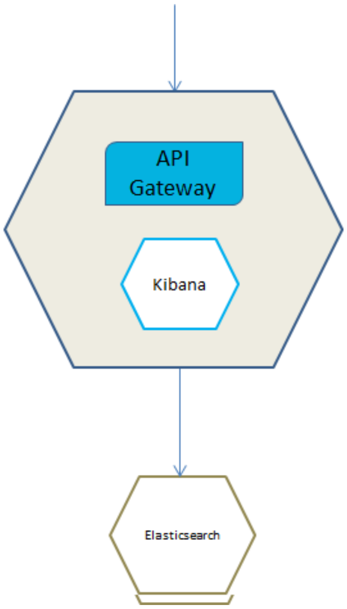
The Docker Compose files can be parameterized through environment variables.

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 195. The Docker Compose file references the standard Elasticsearch 7.13.0 image: docker.elastic.co/elasticsearch/elasticsearch:7.13.0.

   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:**

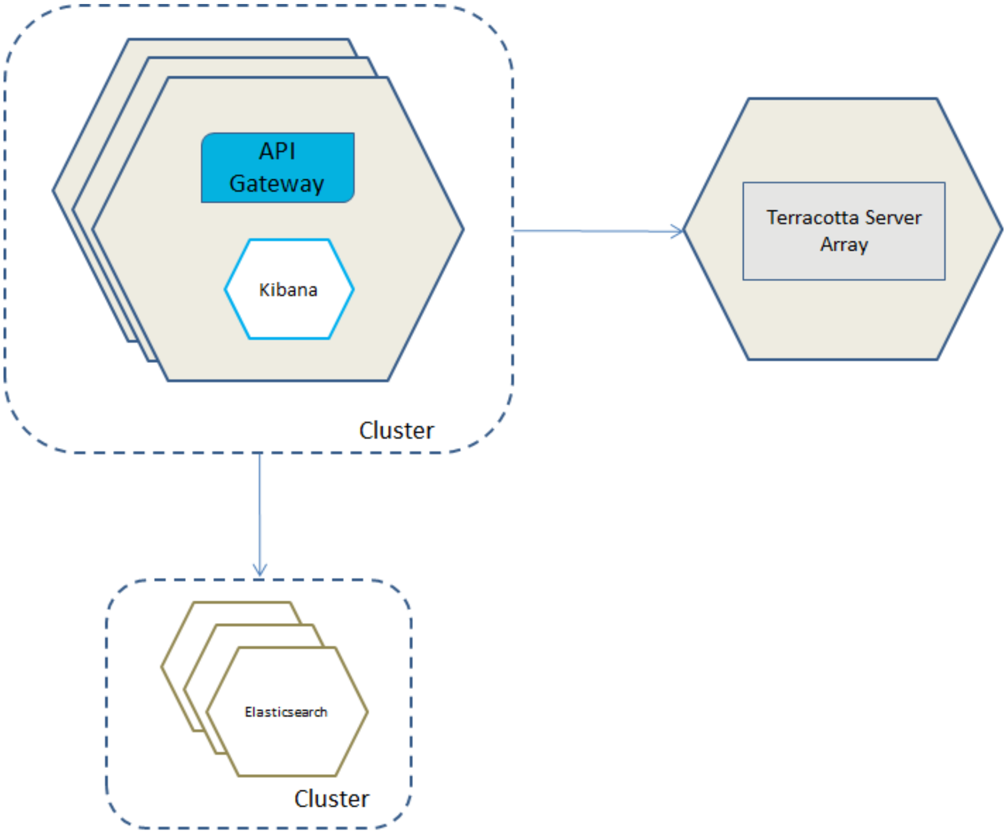
You can stop the API Gateway Docker container and the Elasticsearch container using the Docker Compose sample file with the following command:

```
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 195.

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.

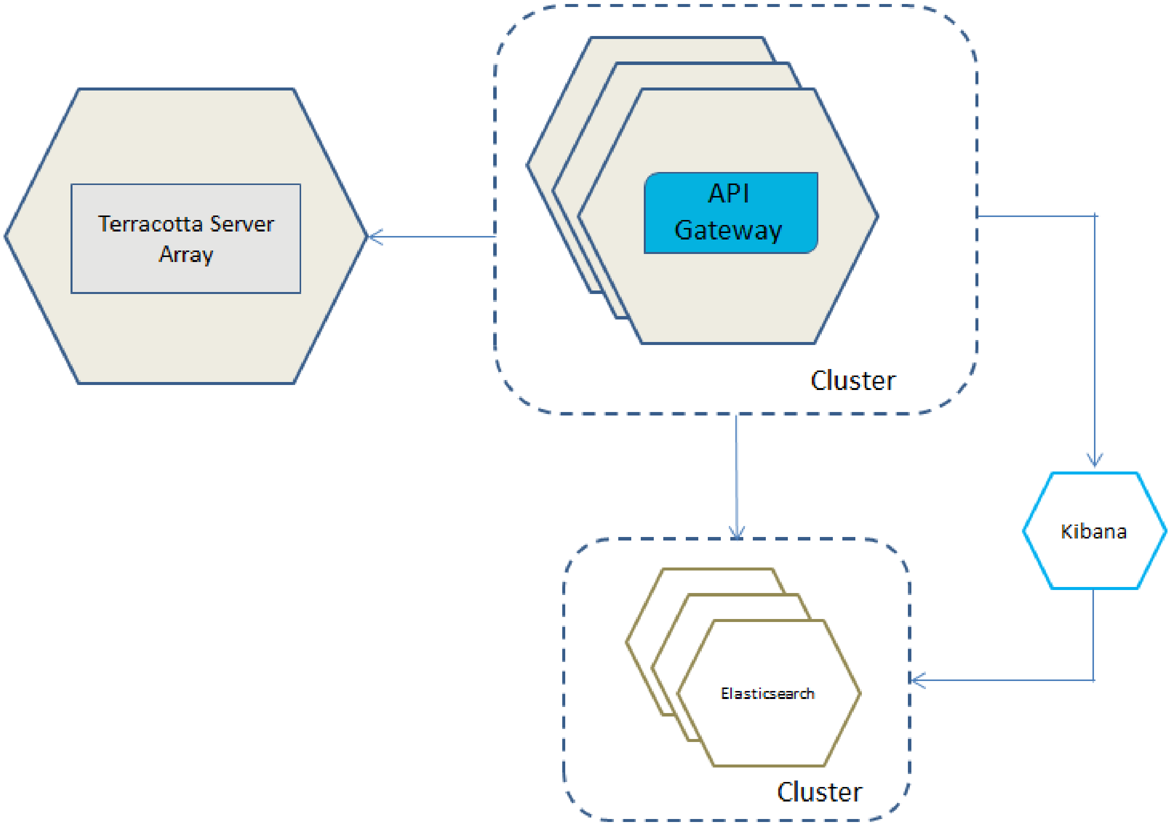
**Note:**
You can stop the API Gateway Docker container and the Elasticsearch container using the Docker Compose sample file with the following command:

```
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 195.

   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
```

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-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
```

# 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.

## My data store migration fails when I provide hostname instead of localhost in the elasticsearch.yml file.

My data store migration fails when I provide hostname instead of localhost in elasticsearch.yml.

Basically, I am adding the following line in the file `<installDir>\InternalDataStore\config\elasticsearch.yml` before migration.

```
reindex.remote.whitelist:hostName:9240
```

It works fine with localhost, but with actual hostName it fails with the following error, Backup process started for the configured elasticsearch host [localhost:8240] Creating repository [migration_default] 2019-06-13 17:00:41 ERROR AbstractElasticsearchClient:98 - APIGW Migration Exception:Error while creating repository - migration_default. Message - listener timeout after waiting for [100000] ms java.io.IOException: listener timeout after waiting for [100000] ms at org.elasticsearch.client.RestClient$SyncResponseListener.get(RestClient.java:660) ~ [elasticsearch-rest-client-5.6.4.jar:5.6.4]

**Resolution**:

You can use regular expressions instead of using the exact hostname. To learn more about how to set regex values for the reindex.remote.whitelist in the elasticsearch.yml file, see Elasticsearch documentation.

# 7 Kubernetes Support

# Overview

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:

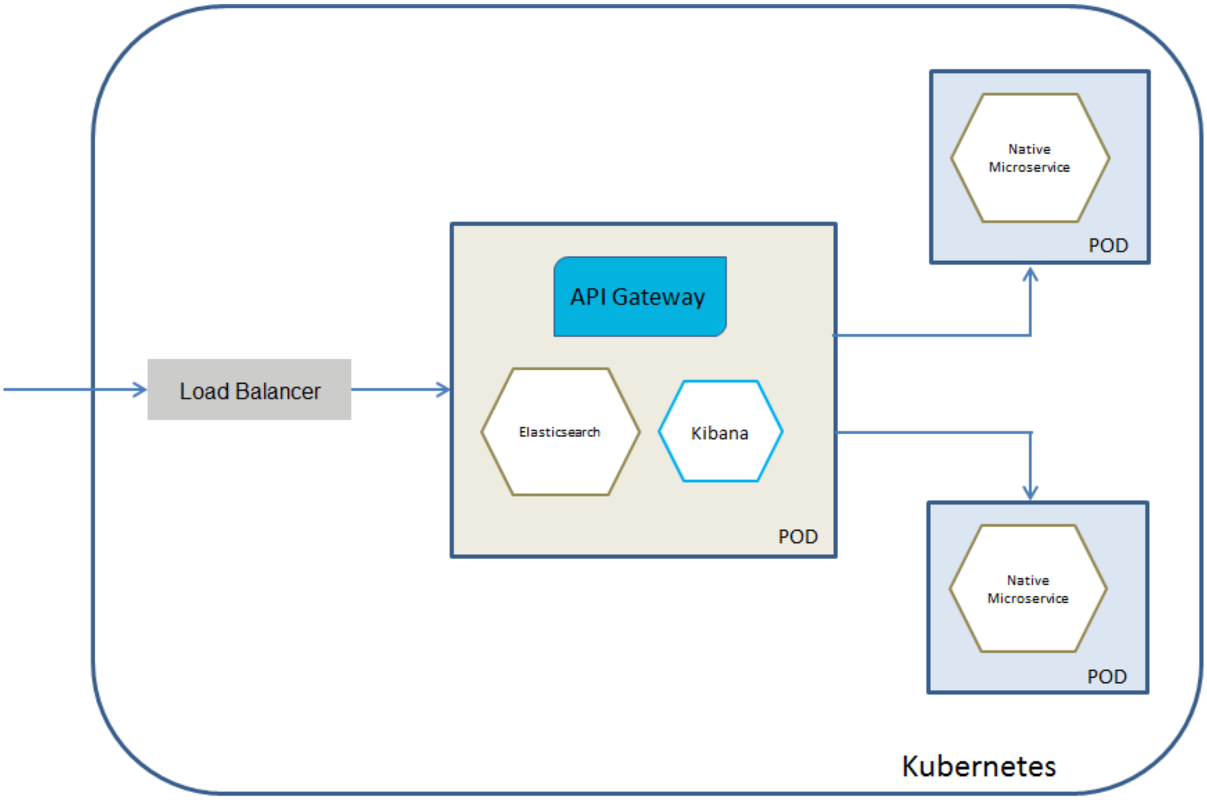
- "Building the Docker Image for an API Gateway Instance " on page 195, in particular the --target.configuration and --os.image parameters
- "OpenShift Support" on page 223

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.

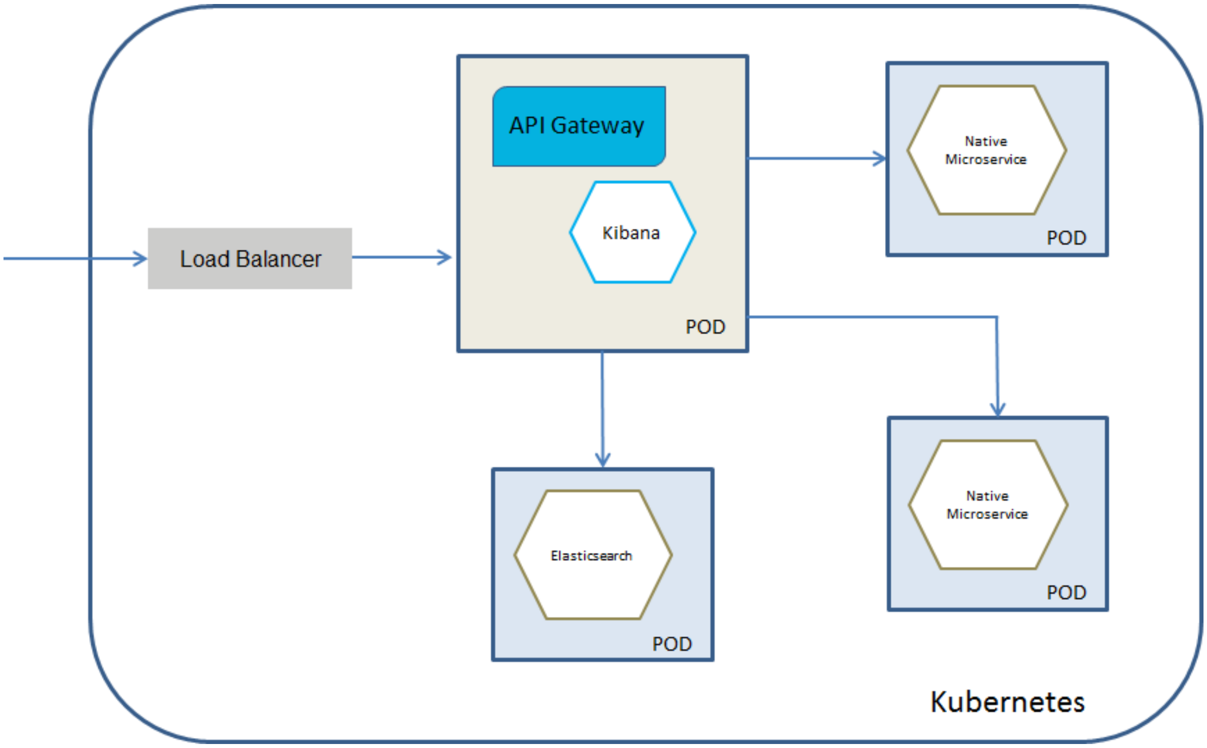
   This now pulls the image specified and creates the API Gateway pod with API Gateway and Elasticsearch containers.

   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 7.13.0.

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 7.13.0.

   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.

## 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.yml file passes the configuration parameters into the Helm chart. A sample values.yaml file is available at *SAG_Root*/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.yml 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 Kubenetes' 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 226.

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 224
- "Running the Docker Image With the sagadmin user" on page 225

## 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 195.

   Ensure that you have set the parameters `--target.configuration` and `--os.image` specific to the OpenShift environment.

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 226 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 224.

## 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 222 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
```

# 8 Configuration Properties

# 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_root`*`/profiles/`*`IS`*`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_root/profiles/IS_IS_Instance_Name/apigateway/config/uiconfiguration.properties`

**apigw.kibana.autostart**

Decides 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_root/profiles/IS_IS_Instance_Name/apigateway/kibana-4.5.1/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_root/InternalDataStore/config/elasticsearch.yml` file.

## 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_root*`/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_root*/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_root`*`/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_root*/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_root*/IntegrationServer/instances/*IS_Instance_Name*/packages/WmAPIGateway/config/resources/beans/gateway-datastore.xml to `transport.tcp.port` property in the elasticsearch.yml file.
> - Ensure that the `cluster.name` and `transport.tcp.port` properties are in synchronization if you encounter any errors.

# 9 API Gateway Data Management and Housekeeping

## Data Management/Housekeeping

Data is an integral part of API Gateway and contains all the information about your APIs and API Gateway itself. It is pretty essential to manage data efficiently. Data Management or Housekeeping involves two important functions:

- **Backup and restore**. A backup is a replica of actual data stored in a known location. If the original data is lost or damaged, you can retrieve the data from Backup thus ensuring that no business-critical data is lost. The process of retrieving original data from Backup is known as restoration of data.
- **Archive and purge**. In API Gateway, the size of the database grows rapidly as API transactions and events continue to occur on a day-by-day basis. An unexpected database growth can eventually degrade the system performance over a period of time and generate unwanted errors in the API Gateway database.

## Backup and Restore

API Gateway application data such as APIs, configurations, applications are stored in Elasticsearch (API Data Store). This data needs to be periodically backed up to protect against accidental loss of data. The backup frequency and backup retention policies must be driven by the Recovery Point Objective (RPO) obligations.

You can create periodical backup schedules of API Gateway data as it would be helpful if you want to revert to a desirable state of API Gateway in case of disaster or a failover.

## Backup Operation

For creating a complete API Gateway data backup, you must include

- **Data stored in API Data Store**. The API Data Store is used to save the primary data such as APIs, Policies, Applications, and so on. It includes the data that is accessible from API Gateway UI. You can take a backup of this data using the **create backup** command with the **apigatewayUtil** script. For information on taking a backup of API Data Store, see "Data Store Backup" on page 235.
- **API Gateway Platform Data**. The files containing the API Gateway configuration data. You can take a backup of the platform data using the **export platformConfiguration** command with the **apigatewayUtil** script or manually take a backup of the configuration files. For more information on creating backup of the platform data, see .

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" on page 271.

### Data backup considerations

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

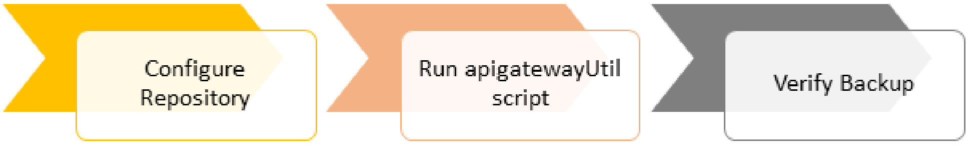
1. Always store your backup files in a fail-safe location.

2. Schedule your backup periodically.
3. For an effective Data Housekeeping practice, estimate the data you might store, depending on your data retention strategy, to help manage the data. Based on the data retention period, you will have two options: Purge old data or Archive and Purge the data. API Gateway provides options to archive the transactions, audit, or application logs data from the UI.

## 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.

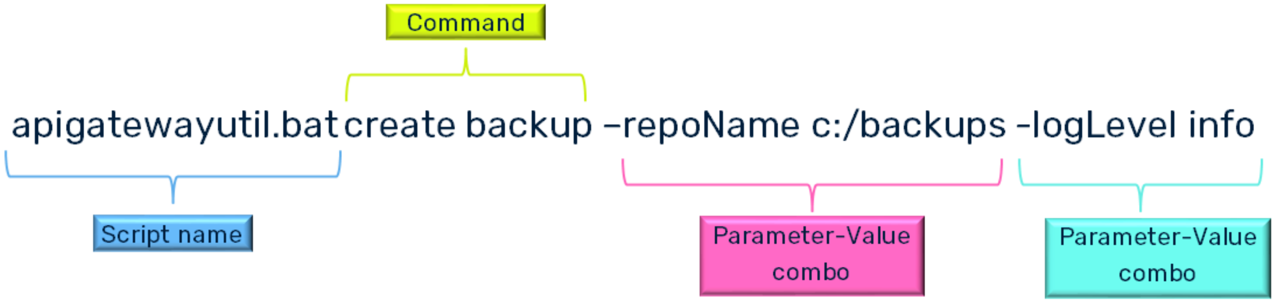
This process of incrementing the data with snapshots only applies within a single repository because no data is shared between repositories.

### 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,



```
apigatewayutil.bat create backup -repoName c:/backups -logLevel info
```

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

Linux

```
./apigatewayUtil.bat -help
```

Windows

```
apigatewayUtil.bat -help
```

## Configuring a 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 240.

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
   ```

   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 258

### 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 258

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 240.

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
   ```

   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 258

### 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
   ```

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

8. 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
   ```

9. 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
```

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 258.

10. 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 240.

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
    ```

    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 258.

## 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 "Configuring a Backup Repository" on page 236.

### 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.<br>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, *default-2021-may-17-22-23-2-266000000*.<br>**Note:** Do not use uppercase in the name. |
| tenant | Name of the tenant in which the data exists.<br>If you do not provide this parameter, the value is *default*. |
| repoName | Name of the repository where the backup must be saved. If you do not specify this value, then the backup is stored in the default location. |
| include | 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.<br>Available options for this parameter are:<br>■ **Assets**. To include all API Gateway assets in the backup being created.<br>■ **Audit**. To include the audit data in the backup.<br>■ **Analytics**.To include analytics data in the backup.<br>■ **License**. To include the license data in the backup. This data provides the transaction usage analytics across all stages of the application. |

| Parameter | Description |
|---|---|
| | - **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.<br>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 *include assets audit* parameter in your backup command, and if your backup name is *backup01*, then the backup file name is created as *backup01 -assets audit*.<br>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:<br>`apigatewayUtil.bat create backup`<br>`-tenant mytenant`<br>`-name mytenant_2021June031230`<br>`-include assets -repoName myrepo`<br>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 277. |
| debug | Option to specify whether you want to activate debugging when creating a backup. Available levels are:<br>- `true`. 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.<br>- `false`. Does not display detailed information. The value is considered as false, when this parameter is not provided. |
| logLevel | Level of log that you want to create. Log levels indicate the severity of logs. Available levels are:<br>- `Info`. Provides the list of regular events that occur during the process. These events are informative.<br>- `Debug`. Provides the events that could be useful, if you have to debug the process.<br>- `Warning`. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process.<br>- `Error`. Indicates the events that stop the functionality from working as designed. |

| Parameter | Description |
|---|---|
| | ■ `Trace`. Provides the list of events in a much detailed manner that could be useful for debugging.<br>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:<br>`apigatewayUtil.bat list backup -logLevel warning`<br>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.<br>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.<br>For example, to save the log file in `C:/apiglogs/backups`, you can provide the location as seen in the following example:<br>`apigatewayUtil.bat create backup -name samplebackup -logFileLocation C:/apiglogs/backups`<br>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 246 and for information on viewing the list of backup files, see "Viewing Backup Files List" on page 249.

> **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.

## 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 277.

### 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 280.

   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: !, @, $, % , ( , ).<br>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, *default-2021-may-17-22-23-2-266000000*.<br>**Note:** Do not use uppercase in the name. |

| Parameter | Description |
|---|---|
| tenant | Name of the tenant in which the data exists.<br>If you do not provide this parameter, the value is *default*. |
| repoName | Name of the repository where the backup must be saved. If you do not specify this value, then the backup is stored in the default location. |
| debug | Option to specify whether you want to activate debugging when creating a backup. Available levels are:<br>- `true`. 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.<br>- `false`. Does not display detailed information. The value is considered as false, when this parameter is not provided. |
| logLevel | Level of log that you want to create. Log levels indicate the severity of logs. Available levels are:<br>- `Info`. Provides the list of regular events that occur during the process. These events are informative.<br>- `Debug`. Provides the events that could be useful, if you have to debug the process.<br>- `Warning`. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process.<br>- `Error`. Indicates the events that stop the functionality from working as designed.<br>- `Trace`. Provides the list of events in a much detailed manner that could be useful for debugging.<br>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:<br>`apigatewayUtil.bat list backup -logLevel warning`<br>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.<br>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. |

| Parameter | Description |
|---|---|
| | For example, to save the log file in `C:/apiglogs/backups`, you can provide the location as seen in the following example:<br>`apigatewayUtil.bat create backup -name samplebackup -logFileLocation C:/apiglogs/backups`<br>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` |

## 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 `-debug true` parameter along with the `create backup` command or by checking the logs. For more information, see "Troubleshooting a Failed Backup" on page 260. |
| 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 260. |
| Unavailable | The specified backup file does not exist. Verify the file name and repository. |

**To verify the status of a 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:

   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. |
| repoName | Name of the repository where the backup can be found. |
| verbose | Option to display detailed status of the backup files in a given repository. Possible values are:<br>- `true`. The backup files appear with the following details:<br>  - **snapshot**. Name of the backup file.<br>  - **status**. Status of the backup process.<br>  - **startTime**. Time when the backup process was initiated.<br>  - **endTime**. Time when the backup process was completed.<br>  - **Duration**. Time when for the backup creation.<br>  - **Indices**. Name of the indexes included in the backup.<br>  - **Successful shards**. Number of successful shards in backup.<br>  - **Failed shards**. Number of failed shards in backup.<br>  - **Total shards**. Number of successful shards in backup.<br>- `false`. The backup files appear without the details listed above.<br>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. |
| format | Option to specify the format in which the details must appear. Works in combination with the **verbose** parameter. Available options are:<br>- JSON |

| Parameter | Description |
|---|---|
| | ■ Text<br>For example, if you run the following command, the backup status details are displayed in plain text format:<br>`apigatewayUtil.bat status backup -name samplebackup -verbose true -format text` |
| logLevel | Level of log that you want to create. Log levels indicate the severity of logs. Available levels are:<br>■ `Info`. Provides the list of regular events that occur during the process. These events are informative.<br>■ `Debug`. Provides the events that could be useful, if you have to debug the process.<br>■ `Warning`. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process.<br>■ `Error`. Indicates the events that stop the functionality from working as designed.<br>■ `Trace`. Provides the list of events in a much detailed manner that could be useful for debugging.<br>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:<br>`apigatewayUtil.bat list backup -logLevel warning`<br>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.<br>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.<br>For example, to save the log file in `C:/apiglogs/backups`, you can provide the location as seen in the following example:<br>`apigatewayUtil.bat create backup -name samplebackup -logFileLocation C:/apiglogs/backups`<br>This parameter is optional. If you do not specify the parameter, the logs are saved in the following location *`SAGInstallDir`*`/` |

| Parameter | Description |
|---|---|
| | `IntegrationServer/instances/instance_name/packages/WmAPIGateway/cli/logs/APIGWUtility.log` |

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.

## 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

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:

   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:

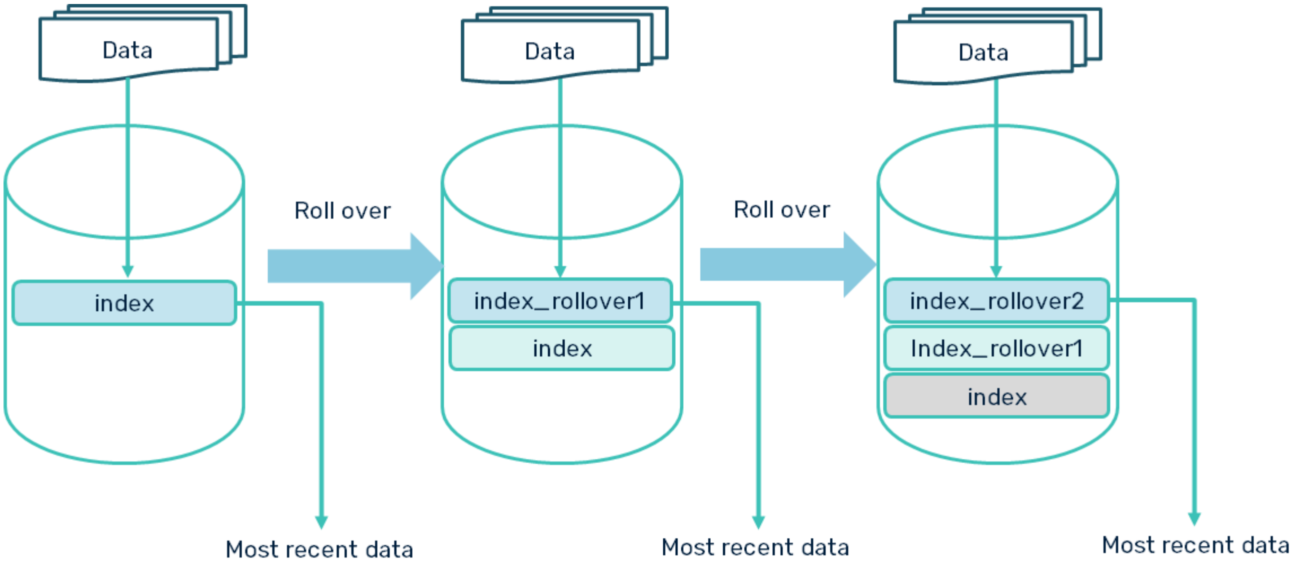
   | Parameter | Description |
   |---|---|
   | tenant | Name of the tenant for which you want to view the list of backup files. |
   | repoName | Name of the repository for which you want to view the list of backup files. If you do not provide this value when running the *list* command, then the list of backup files in the default repository appears. |

| Parameter | Description |
|---|---|
| status | Option to specify the status of backup files and filter backup files based on their status. For example, if you want to view the list of backup files whose status is `Partial`, then you can run the command as seen here:<br>`apigatewayUtil.bat list backup -status partial`<br>For information on possible states of backup files, see "Verifying Backup Status" on page 246. |
| verbose | Option to display detailed status of the backup files in a given repository. Possible values are:<br>- `true`. The backup files appear with the following details:<br>  - **snapshot**. Name of the backup file.<br>  - **status**. Status of the backup process.<br>  - **startTime**. Time when the backup process was initiated.<br>  - **endTime**. Time when the backup process was completed.<br>  - **Duration**. Time taken for the backup creation.<br>  - **Indices**. Name of the indexes included in the backup.<br>  - **Successful shards**. Number of successful shards in backup.<br>  - **Failed shards**. Number of failed shards in backup.<br>  - **Total shards**. Total number of shards in backup.<br>- `false`. The backup files appear without the details listed above.<br>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. |
| format | Option to specify the format in which the details must appear. Works in combination with the **verbose** parameter. Available options are:<br>- JSON<br>- Text<br>For example, if you run the following command, the backup status details are displayed in plain text format:<br>`apigatewayUtil.bat status backup -name samplebackup -verbose true -format text` |
| logLevel | Level of log that you want to create. Log levels indicate the severity of logs. Available levels are: |

| Parameter | Description |
|---|---|
| | - `Info`. Provides the list of regular events that occur during the process. These events are informative.<br>- `Debug`. Provides the events that could be useful, if you have to debug the process.<br>- `Warning`. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process.<br>- `Error`. Indicates the events that stop the functionality from working as designed.<br>- `Trace`. Provides the list of events in a much detailed manner that could be useful for debugging.<br>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:<br>`apigatewayUtil.bat list backup -logLevel warning`<br>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.<br>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.<br>For example, to save the log file in `C:/apiglogs/backups`, you can provide the location as seen in the following example:<br>`apigatewayUtil.bat create backup -name samplebackup -logFileLocation C:/apiglogs/backups`<br>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` |

## 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.<br>If you do not provide this parameter, the value is *default*. |
| 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.<br>For example, if you provide the following command:<br>`apigatewayutil.bat rollover index`<br>`-type performancemetrics`<br>`-targetIndexSuffix 15aug2021`<br>the suffix is added to the rolled over index<br>`Rollover is completed successfully`<br>`Old Index:`<br>`gateway_default_analytics_performancemetrics_14aug2021`<br>`New Index:`<br>`gateway_default_analytics_performancemetrics_15aug2021`<br>`Rolled over: true`<br>`Dry Run: false`<br>`Conditions: [{}]` |
| dryRun | Option to check whether the current index matches with any of the specified conditions. |

| Parameter | Description |
|---|---|
|  | Possible values are:<br>■ *true*. Performs check on the current index against the specified conditions.<br>■ *false*. Does not check the current index. This is the default value and the value of the parameter is considered as `false`, if do not specify this parameter.<br>You can specify one of the conditions listed in the "Conditions that can be given with rollover command" on page 255 section.<br>For example, to check if the size of the current log index is *6 GB*, you can run the command as seen here:<br>`apigatewayutil.bat rollover index -type log -dryRun true -maxSize 6GB`<br>Sample result:<br>`Dry run executed successfully.`<br>`Old Index:`<br>`gateway_default_log_1628582905777-000001`<br>`New Index:`<br>`gateway_default_log_202108271018`<br>`Rolled over: false`<br>`Dry Run: true`<br>`Conditions: [{[max_size: 6gb]=false}]`<br>**Note:**<br>This parameter does not perform rollover. It displays results based on the check performed. |
| logLevel | Level of log that you want to create. Log levels indicate the severity of logs. Available levels are:<br>■ `Info`. Provides the list of regular events that occur during the process. These events are informative.<br>■ `Debug`. Provides the events that could be useful, if you have to debug the process.<br>■ `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 |
|---|---|
| | ■ `Error`. Indicates the events that stop the functionality from working as designed.<br>■ `Trace`. Provides the list of events in a much detailed manner that could be useful for debugging.<br>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:<br>`apigatewayUtil.bat list backup -logLevel warning`<br>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.<br>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.<br>For example, to save the log file in `C:/apiglogs/backups`, you can provide the location as seen in the following example:<br>`apigatewayUtil.bat create backup -name samplebackup`<br>`-logFileLocation C:/apiglogs/backups`<br>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 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 | Maximum age of the indexes.<br>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.<br>For example, to roll over the **monitorevents** index if it is older than 2 minutes, you can provide<br>`apigatewayutil.bat rollover index`<br>`-type monitorevents -maxAge 2m` |
| maxDocs | Maximum number of the documents in the indexes.<br>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.<br>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<br>`apigatewayutil.bat rollover index`<br>`-type monitorevents -maxDocs 100` |
| maxSize | Maximum size of the indexes.<br>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.<br>For example, to roll over the **monitorevents** index if its size is more than 1 GB, you can provide<br>`apigatewayutil.bat rollover index`<br>`-type monitorevents -maxAge 1gb` |
| maxPrimaryShardSize | Maximum size of the primary shard of the indexes.<br>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.<br>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<br>`apigatewayutil.bat rollover index`<br>`-type monitorevents -maxPrimaryShardSize 1GB` |

## 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 performing a file system delete.

### To delete a backup file

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 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 249.

   You can provide the following parameters based on your requirements:

| Parameter | Description |
|---|---|
| tenant | Name of the tenant in which the backup file is located. |
| repoName | Name of the repository from which you want to delete the backup file. If you do not provide this value when running the *delete* command, then the script looks for the specified backup file in the default repository. |
| olderThan | 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: |

```
apigatewayUtil.bat delete backups -olderThan 10
```

| Parameter | Description |
|---|---|
| | This command deletes the backup files that were created earlier than the past 10 days. |
| logLevel | Level of log that you want to create. Log levels indicate the severity of logs. Available levels are:<br>- `Info`. Provides the list of regular events that occur during the process. These events are informative.<br>- `Debug`. Provides the events that could be useful, if you have to debug the process.<br>- `Warning`. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process.<br>- `Error`. Indicates the events that stop the functionality from working as designed.<br>- `Trace`. Provides the list of events in a much detailed manner that could be useful for debugging.<br>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:<br>`apigatewayUtil.bat list backup -logLevel warning`<br>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.<br>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.<br>For example, to save the log file in `C:/apiglogs/backups`, you can provide the location as seen in the following example:<br>`apigatewayUtil.bat create backup -name samplebackup -logFileLocation C:/apiglogs/backups`<br>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` |

## 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 | Allows you to specify the required level of log that you want to create. Log levels indicate the severity of logs. Available levels are:<br>- **Info**. Provides the list of regular events that occur during the process. These events are informative.<br>- **Debug**. Provides the events that could be useful, if you have to debug the process.<br>- **Warning**. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process.<br>- **Error**. Indicates the events that stop the functionality from working as designed.<br>- **Trace**. Provides a much detailed events that could be useful for debugging.<br>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:<br>`apigatewayUtil.bat list backup -logLevel warning`<br>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.<br>This parameter is optional. If you do not specify the parameter, then the **Info** level logs are saved by default. |
| logFileLocation | Allows you to provide the location where you want to save the log file.<br>For example, to save the log file in `C:/apiglogs/backups`, you can provide the location as seen in the following example:<br>`apigatewayUtil.bat create backup -name samplebackup`<br>`-logFileLocation C:/apiglogs/backups` |

| Parameter | Description |
|---|---|
| | 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` |

## 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:**
While taking a Backup in sync mode, if you encounter a socket time out error, you can take a backup in async mode.

## 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. |

| Exit codes | Description |
|---|---|
| 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
   ```

## Creating Platform Data Backup using File System Backup

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:

| Configuration | File name | File location |
|---|---|---|
| UI configurations | uiconfiguration.properties | `SAGInstallDir/profiles/instance_name/apigateway/config/` |
| WebApp settings | com.softwareag.catalina.connector.http.pid-apigateway.properties<br>com.softwareag.catalina.connector.https.pid-apigateway.properties | `SAGInstallDir/profiles/instance_name/configuration/com.softwareag.platform.config.propsloader/` |
| Custom ESB packages | File name specified by users. | Location used by users to save the file.<br>Generally, the customized packages are saved in the following location: `SAGInstallDir\IntegrationServer\tenant\packages\` |

## Creating Platform Data Backup using Script

**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 | Allows you to specify the required level of log that you want to create. Log levels indicate the severity of logs. Available levels are:<br>- **Info**. Provides the list of regular events that occur during the process. These events are informative.<br>- **Debug**. Provides the events that could be useful, if you have to debug the process.<br>- **Warning**. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process.<br>- **Error**. Indicates the events that stop the functionality from working as designed.<br>- **Trace**. Provides a much detailed events that could be useful for debugging. |

| Parameter | Description |
|---|---|
| | 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:<br>`apigatewayUtil.bat list backup -logLevel warning`<br>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.<br>This parameter is optional. If you do not specify the parameter, then the **Info** level logs are saved by default. |
| logFileLocation | Allows you to provide the location where you want to save the log file.<br>For example, to save the log file in `C:/apiglogs/backups`, you can provide the location as seen in the following example:<br>`apigatewayUtil.bat create backup -name samplebackup -logFileLocation C:/apiglogs/backups`<br>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` |

## 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.

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 263.

**File System restore** - to restore the API Gateway configuration data. For more information, see “Restoring Platform Data” on page 271.

### Restoring Data Store Backup

You can restore the API Data Store data backed up 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 270.

### 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 script looks for the specified backup file in the default repository. For example,

   ```
   C:\SoftwareAG\IntegrationServer\instances\default\packages\WmAPIGateway\cli\bin>apigatewayutil.bat

   restore backup -name sample
   The tenant name is default
   Data types that will overwrite the existing data:
   analytics,assets,license,audit,log,cache
   ```

   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 the `tenant` parameter, then the script reads the tenant value from the **tenantId** property of the `gateway-core.xml` file. |
| repoName | Name of the repository from which you want to restore the backup file. If you do not specify this value, then the script looks for the specified backup file in the default repository. 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. |
| sync | Option to specify whether the restore process is performed in a synchronously or asynchronously. The possible values are:<br>- *true*. 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.<br>- *false*. 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 |

| Parameter | Description |
|---|---|
| | 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 268. |
| srcTenant | 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 `tenant` parameter that specifies the tenant name to which the data backup is being restored. That is,<br>`apigatewayUtil.sh restore backup`<br>`-name backup_file_name`<br>`-srcTenant backup_tenant_name`<br>`-tenant tenant_name`<br>If you do not provide the tenant parameter, then the script assumes the tenant name as the value for srcTenant.<br>**Note:**<br>The srcTenant, aggregate, and restoreClusterState 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 | Option to include any of the following based on your input:<br>■ `analytics`. Restores the analytics data, logs, and events data.<br>■ `assets`. Restores assets.<br>■ `license`. Restores license metrics.<br>■ `audit`. Restores audit logs.<br>■ `log`. Restores log data.<br>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<br>`./apigatewayUtil.sh restore`<br>`backup -name backup_file_name`<br>`-repoName repository_name`<br>`-include analytics,assets` |
| aggregate | Option to specify whether the existing license, logs, audit, and analytics data should be merged with the restored data. The possible values are:<br>■ *true*. 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 `-aggregate true`, 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 |

| Parameter | Description |
|---|---|
| | analytics data as well the analytics data from the restored backup file in your instance.<br>■ *false*. The existing data is replaced with the restored data. This is the default value.<br>**Note:**<br>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. |
| restoreClusterState | 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:<br>■ *true*. The global state settings are restored.<br>■ *false*. The global state settings are not restored. This is the default value.<br>Do not use this parameter for normal restore operations. Use only when restoring backup from a different Elasticsearch or when the -srcTenant parameter is specified.<br>**Note:**<br>If you have secured Data Store using Search Guard, you cannot restore the global cluster state of the Data Store. The global cluster state includes information such as persistent cluster settings, index templates, pipelines, and so on. To restore the global cluster state, you must either perform the restore with the cluster state first, then secure the Data Store, or disable Search Guard, perform the restore with cluster state, and then enable the security plugin. However, you can still perform a restore without the global cluster state with the Search Guard plugin enabled. For information on Restoring data and the usage of restore parameters, refer the Restore using script section. |
| logLevel | Level of log that you want to create. Log levels indicate the severity of logs. Available levels are:<br>■ `Info`. Provides the list of regular events that occur during the process. These events are informative.<br>■ `Debug`. Provides the events that could be useful, if you have to debug the process.<br>■ `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 |
|---|---|
| | - `Error`. Indicates the events that stop the functionality from working as designed.<br>- `Trace`. Provides the list of events in a much detailed manner that could be useful for debugging.<br>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:<br>`apigatewayUtil.bat list backup -logLevel warning`<br>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.<br>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.<br>For example, to save the log file in `C:/apiglogs/backups`, you can provide the location as seen in the following example:<br>`apigatewayUtil.bat create backup -name samplebackup`<br>`-logFileLocation C:/apiglogs/backups`<br>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 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 270. |
| 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. |
| 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 `SAGInstallDir\InternalDataStore\logs` 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. |
| repoName | Name of the repository that you want to verify. |
| logLevel | Allows you to specify the required level of log that you want to create. Log levels indicate the severity of logs. Available levels are:<br>- **Info**. Provides the list of regular events that occur during the process. These events are informative.<br>- **Debug**. Provides the events that could be useful, if you have to debug the process.<br>- **Warning**. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process.<br>- **Error**. Indicates the events that stop the functionality from working as designed.<br>- **Trace**. Provides a much detailed events that could be useful for debugging.<br>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:<br>`apigatewayUtil.bat list backup -logLevel warning`<br>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.<br>This parameter is optional. If you do not specify the parameter, then the **Info** level logs are saved by default. |
| logFileLocation | Allows you to provide the location where you want to save the log file.<br>For example, to save the log file in `C:/apiglogs/backups`, you can provide the location as seen in the following example:<br>`apigatewayUtil.bat create backup -name samplebackup -logFileLocation C:/apiglogs/backups`<br>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` |

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.

## 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 |
|---|---|
| logLevel | Allows you to specify the required level of log that you want to create. Log levels indicate the severity of logs. Available levels are:<br>- **Info**. Provides the list of regular events that occur during the process. These events are informative.<br>- **Debug**. Provides the events that could be useful, if you have to debug the process.<br>- **Warning**. Indicates unexpected events that occurred during the process. Usually, these events do not interrupt or have an immediate effect on the process.<br>- **Error**. Indicates the events that stop the functionality from working as designed. |

| Parameter | Description |
|---|---|
| | - **Trace**. Provides a much detailed events that could be useful for debugging.<br>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:<br>`apigatewayUtil.bat list backup -logLevel warning`<br>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.<br>This parameter is optional. If you do not specify the parameter, then the **Info** level logs are saved by default. |
| logFileLocation | Allows you to provide the location where you want to save the log file.<br>For example, to save the log file in `C:/apiglogs/backups`, you can provide the location as seen in the following example:<br>`apigatewayUtil.bat create backup -name samplebackup`<br>`-logFileLocation C:/apiglogs/backups`<br>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` |

## Restoring Platform Data

The platform data backed up using the **export platformConfiguration** command and the file system method must be restored using the file system restore method.

### To restore the platform data

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, "Backup Operation" on page 234.

2. Restart API Gateway.

In cluster setups, restart all nodes in the cluster.

# Archive and Purge using API

API Gateway provides a comprehensive data management solution for archiving, purging, and restoring events data. It allows you to manage and protect critically important data, and also conserve database disk space, thereby improving the performance of API Gateway.

This section focuses on Housekeeping measures and explains how you can archive transaction and audit data, and purge data.

## Archive Transaction and Audit Data

Transaction and audit data is important. You can archive outdated data (for example, data that is older than an year) for forensic analysis and decision making. Archived data is not the same as Backup data. Software AG recommends that you use proper naming convention for every archive that you create. You can perform an archive for a specific time period (like from 1 June to 1 July)

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 from 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=2021-06-12%2000:00:00&until=2021-06-13%2000:00:00&eventType=ALL"
```

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

You can monitor the status of an archive job by 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/9c0eefde-dc26-4cb7-b0eb-dfe8f3a8a545"
```

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

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

If the archive job fails, the status field in the above output, shows Failed. You must configure Alerts for failures of the purge job. Common reasons for failure include health of Elasticsearch cluster health, load on the system, and so on. You can look into server logs and analyze the failure reasons.

## Purge 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 section. You can purge the following data using commands.

- Analytics data
- Backup Snapshots
- Obsolete or Empty indices
- Expired OAuth tokens
- Archive data

## Purge Analytics Data

You can purge analytics data based on either 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 general API to purge analytics data based on time:

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

In the above API, you can use only the required events in the **eventType** field. The available event types are **transactionalEvents**, **lifecycleEvents**, **performanceMetrics**, **monitorEvents**, **threatProtectionEvents**, **policyViolationEvents**, **errorEvents**, **auditlogs**, **applicationlogs**.

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 |
| 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 |

### Purge Based on Size

Size Based purging is relatively complex. You can follow one of the following methods to purge data based on size:

You must first identify the data range that needs to be purged, to get your system back to the desired limit. You must query the data for a given date range, determine the size, compare with the requirement and then take the next action (purge or to increase/decrease the date range).

You must define the size limit for indices that are expected to grow. You must employ the roll-over procedure once the index size exceeds the defined limit. After a roll-over a new index is created. 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 indices and the size of each index is 25 GB. You can now delete the oldest index so that you get 25 GB of free space and you stay within the defined limits for the analytics data.

Perform the following steps to rollover an index:

1. Find the oldest index by 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 index in descending order of index name. API Gateway follows the pattern gateway_default_transactionalevents_YYYYMMDDHHMM and hence this API returns the index such that the oldest index appears last.

2. Delete the index returned in the previous step by using the following API:

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

3. You can monitor the deletion of 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 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": "default-2017-08-31-1504177207377"
}
```

## Purge Backup Snapshots

It is essential to backup snapshots that are older than the data retention period. You can delete a backup snapshot using the following command:

```
apigatewayUtil.bat/sh  delete backup -name name of the backup to delete
-tenant default or configured tenant name -repoName repo_name
```

To delete backup snapshots that are older than a specific period, you can use the **olderthan** argument, as shown below:

```
apigatewayUtil.bat/sh  delete backup -olderthan <numberofdays>
-tenant <tenantname> -repoName <repoName>
```

You must schedule the execution of these commands through cron jobs or other scheduling options. You can schedule the above purging commands of Backup to execute once, everyday. You can use the following command to monitor the status of the backup:

```
apigatewayUtil.sh status backup -tenant default or configured tenant name -name
name of the backup to delete -repoName repo_name
```

The above command returns success, if the specified name typed in the **name** field does not exist. If the above command fails,

### Purge Obsolete or Empty Indices

API Gateway may have empty indices due to roll-over and purge operations. It is essential to cleanup the empty indices. You can delete an index if there are multiple indices and the index to be deleted is not a write index. Software AG recommends that you schedule or perform a Backup or archive and then purge the indices.

You can use the following API to check the documents stored by the indices.

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

The above 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 the indices that 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 indices using a cron job or some other scheduling method. You can schedule index purging on a daily basis. You can monitor the status of a delete index API 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.

### Purge Expired OAuth Tokens

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

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

You can schedule the purge operation of indices 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 max retention period. There is no API to clear the archive data. You must delete archives manually. You can delete archives on a daily basis.

## Disaster Recovery Management

When there is a technical problem or the system 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 strategies can be broadly categorized into four approaches:

- Cold Standby Mode
- Warm Standby Mode
- Hot Standby Mode
- Active-Active 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.

# 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 |

| Index type | Index name |
|---|---|
| | gateway_default_truststores |
| | gateway_default_registeredapplications |
| | gateway_default_policyactions |
| | 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 |

| Index type | Index name |
|---|---|
| | gateway_default_egpolicyindexlists |
| | gateway_default_groups |
| | gateway_default_oauth2scopes |
| | 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 |

| Index type | Index name |
|---|---|
| | 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
- jmstrigger
- 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 243.

## Cross-Data Center Support

The Cross-Data Center (DC) support provides protection against data center failure by setting up API Gateway across different data centers. It is important to have this support primarily for the following reasons:

- **Disaster Recovery**. When the primary data center goes down due to natural disaster, equipment failure, or cyber attack, a business has to recover lost data from a secondary data center where the data is backed up. The disaster recovery relies upon the replication of data and computer processing of the secondary data center in an off-premises location that is not affected by the disaster. Ideally, an organization transfers its computer processing to that secondary data center in order to continue operations.
- **Load Distribution**. Huge voluminous data can be managed by distributing the data across multiple data centers. Load distribution across data centers provides high performance data access for globally distributed, mission critical applications.

### Deploying Cross-DC Support in API Gateway

The Cross-DC support is deployed in API Gateway in the following modes:

- Hot Standby
- Active - Active

Before you start setting up the data centers for a Cross-DC support, ensure that you have:

- *Manage general administration configurations* functional privilege.
- API Gateway installed in all the data centers.

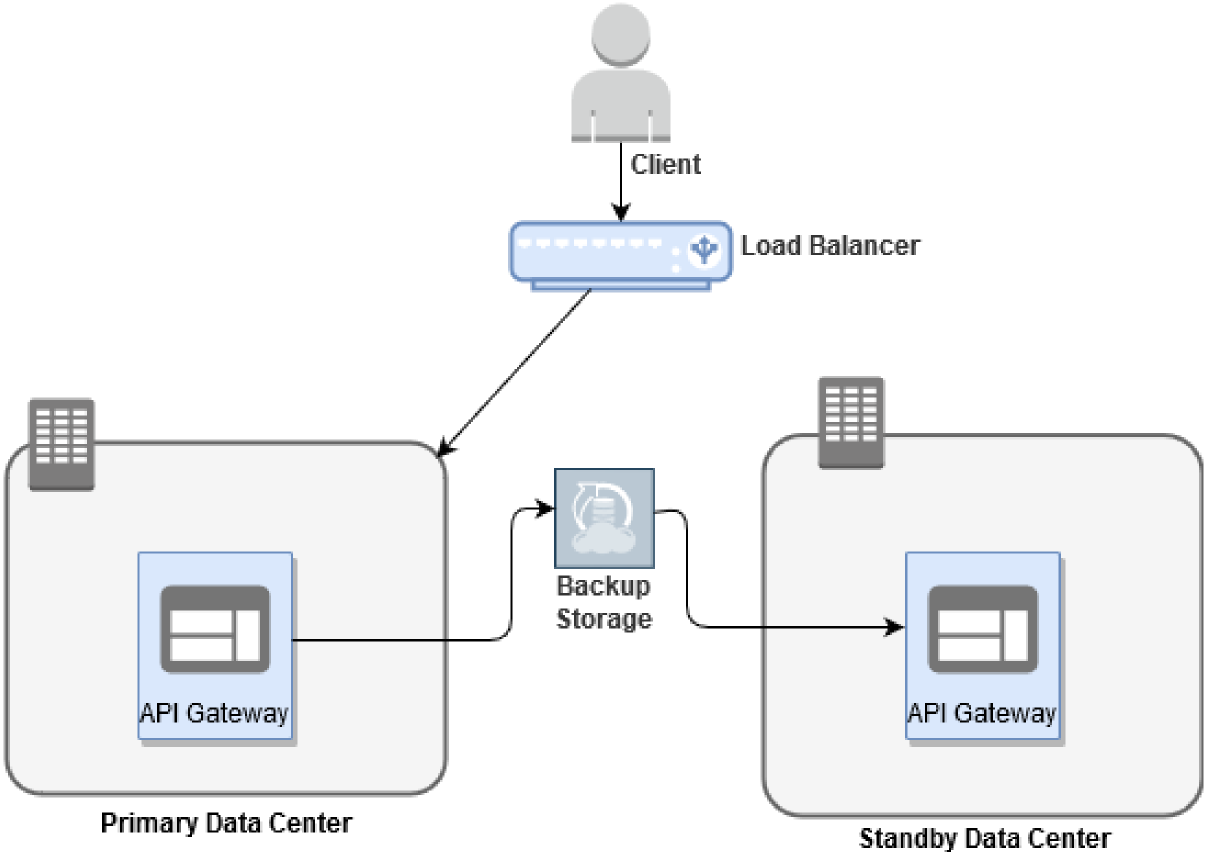
## What is Cold Standby Mode?

In the *Cold Standby* mode, there are only two data centers. The primary data center is up and running, whereas the standby data center is turned on only when the primary data center goes down. The standby data center has a very basic setup. On failure of the primary data center, the standby data center replaces the primary data center. As part of disaster recovery procedure:

- Bring up the services.

- Run the scripts for backup and restore.
- Reconfigure the load balancer to redirect the traffic to the standby 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.



**Note:**
Generally, the standby data center in the cold standby mode is a standalone system. In case of disaster, the standby data center, in the cold standby mode, replaces the primary data center. After the problem is resolved, you must bring the primary data center back into action.

## How Do I Set Up the Data Center in Cold Standby Mode?

This use case explains how to set up the data center in cold standby mode.

The use case starts when you set up the data center in cold standby mode and ends when you successfully replace the primary data center.

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 or API Gateway installed on DC 2. All the DC 1 data is backed up to some external data store based on the RPO. After you back up DC 1, you have to install the API Gateway instance in DC 2, and restore the backed up data from DC 1 to DC 2.

**To set up the data center in cold standby mode**

1. Run the following command in DC 1 to back up the data:

   ```
   apigatewayUtil.sh create backup -name backup_file_name
   ```

   For more information about back up and restore, see "Backup and Restore" on page 234.

2. Install API Gateway on DC 2 after DC 1 goes down.

3. Run the following command in DC 2 to restore the backed up data:

   ```
   apigatewayUtil.sh restore backup -name backup_file_name
   ```

   **Note:**
   In cold standby mode, the data is restored only once.

4. Start the API Gateway instance in DC 2.

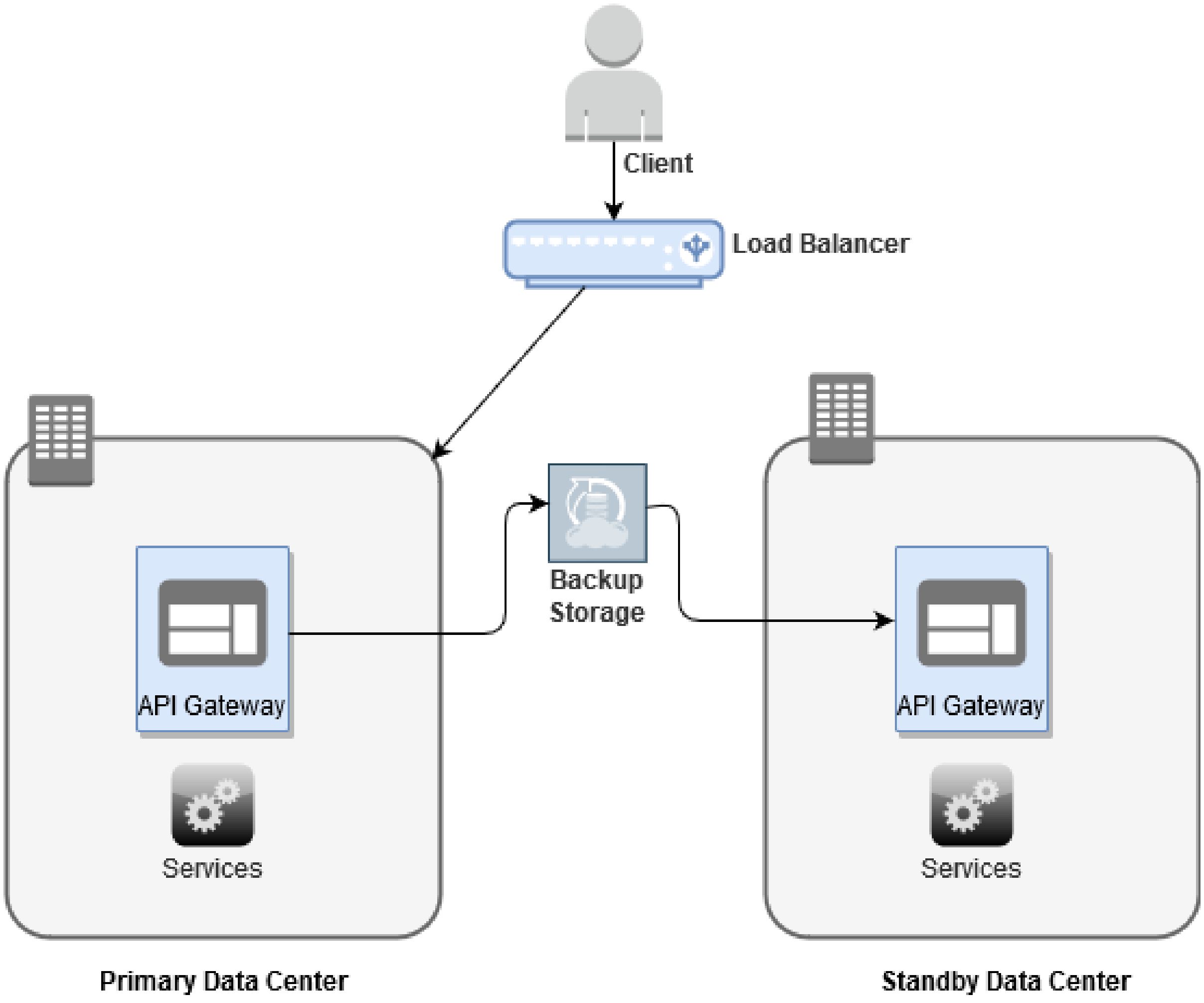
5. Reconfigure the load balancer by exposing DC 2 to the client.

## What is Warm Standby Mode?

In the *Warm Standby* mode, there are only two data centers. The primary data center is up and running, whereas the standby data center is turned on only when the primary data center goes down. The standby data center is regularly backed up with primary data center's data. At times, the data is not mirrored or identical between the two data centers. On failure of the primary data center, the standby data center replaces the primary data center. As part of disaster recovery procedure:

- Run the scripts for backup and restore.
- Reconfigure the load balancer to redirect the traffic to the standby data center.

Warm standby mode is cost-effective in terms of data center operations. However, the application availability might defer depending on how quickly the standby data center replaces the primary data center. The RPO and RTO for warm standby mode is less when compared to cold standby mode.



**Note:**
Generally, the standby data center in the warm standby mode is a standalone system. In case of disaster, the standby data center, in the warm standby mode, replaces the primary data center. After the problem is resolved, you must bring the primary data center back into action.

### How Do I Set Up the Data Center in Warm Standby Mode?

This use case explains how to set up the data center in warm standby mode.

The use case starts when you set up the data center in warm standby mode and ends when you successfully replace the primary data center.

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 warm standby mode, the data store is up and running in DC 2, hence the data is already present in DC 2. As API Gateway is already installed on DC 2, you just have to start API Gateway on DC 2. The downtime for this mode would be the time when you start the API Gateway in DC 2 soon after DC 1 failure.

**To set up the data center in warm standby mode**

1. Run the backup command in DC 1.

   You can schedule the backup frequency based on your RPO. For example, you can use `crontab` command scheduler to back up the data for every 15 minutes using this command:

   ```
   */15 * * * * /APIM/SAG/apigatewayUtil.sh
   ```

   For more information about backup and restore, see "Backup and Restore" on page 234.

   **Note:**
   You can have different backup scheduled at different frequency for core asset data and analytics data.

2. Run the following command in DC 2 to restore the data:

   ```
   apigatewayUtil.sh restore backup -name backup_file_name
   ```

3. Start the API Gateway instance in DC 2, once DC 1 goes down.

4. Reconfigure the load balancer by exposing DC 2 to the client.

## 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.

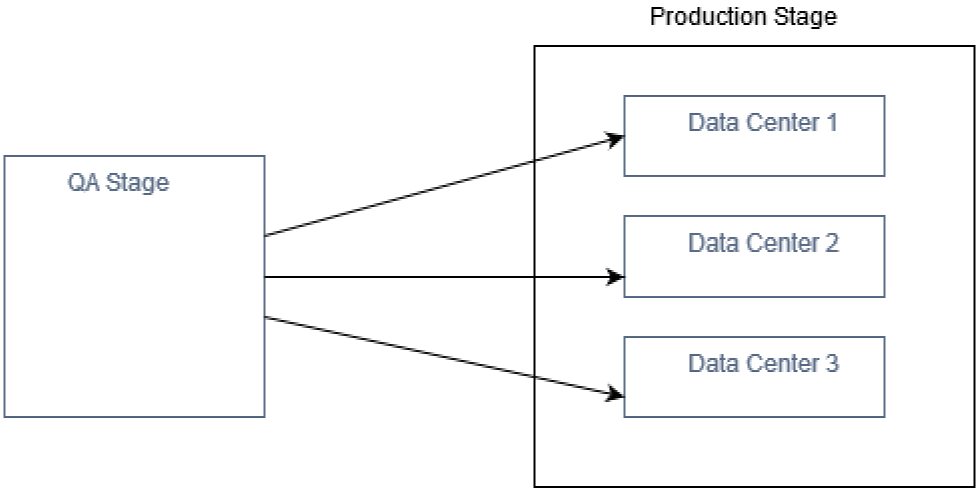
Currently, the Cross-DC support in API Gateway synchronizes application asset across the data centers.

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**

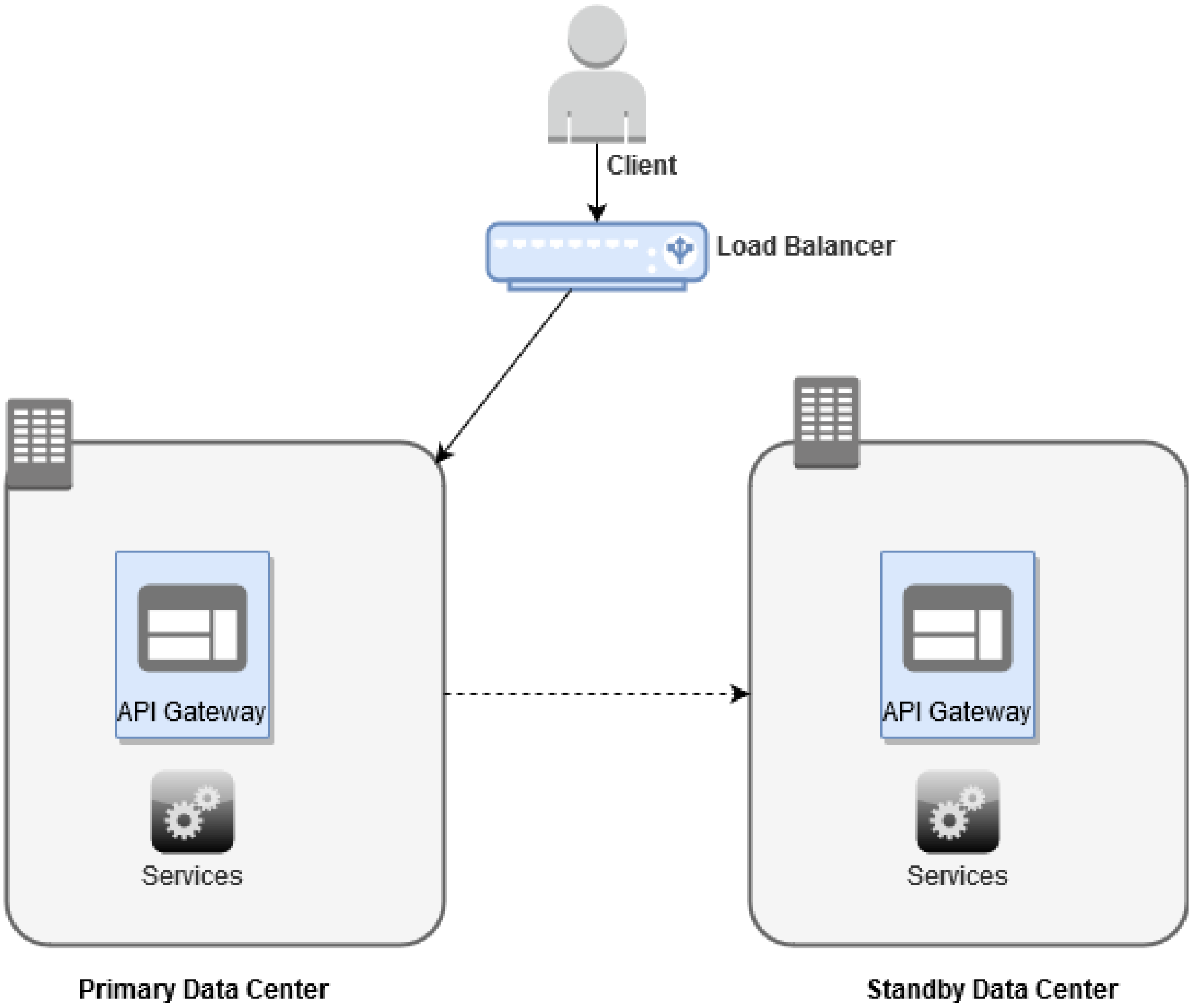
Promotion refers to moving API Gateway assets from the source stage to one or more target stages. For example, you might want to promote assets you have developed on servers in a QA stage (the source API Gateway instance) to data centers in a Production stage (the target API Gateway instance). If you 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.

## 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 standby 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 standby 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 standby data center. The RPO and RTO for hot standby mode is less compared to warm and cold standby modes.



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 291.

  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 296.

  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.

### How Do I Set Up the Data Centers in Hot Standby Mode Using Basic Operation?

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 section in *webMethods API Gateway User's Guide*. 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 activationw, 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 321.

> **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 section in *webMethods API Gateway User's Guide*. 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 section in *webMethods API Gateway User's Guide* . 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": 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 321.

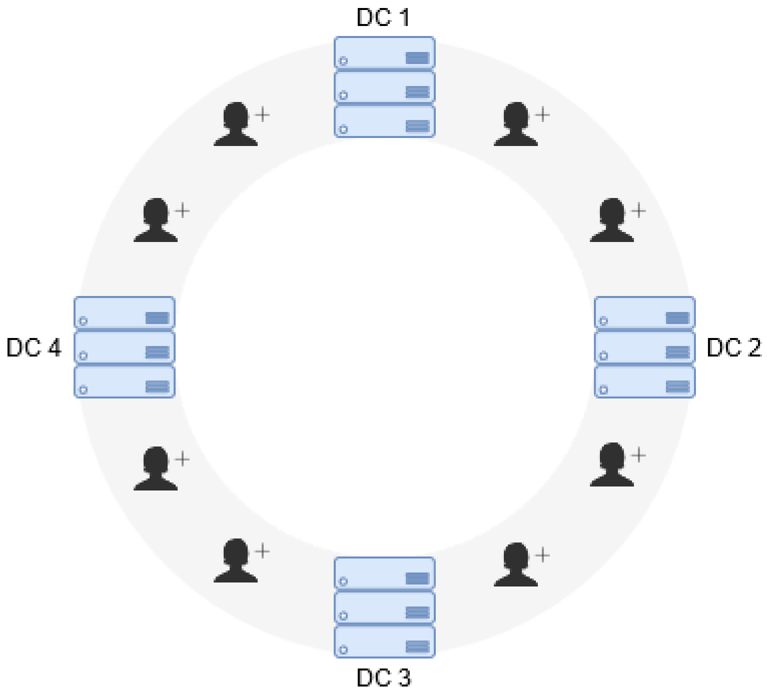
**Note:**

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.

## 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.

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 303.

  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 309.

  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.

## 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": "b04609v0-ef13-44vu-98x1-f988mn18max4",
            "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": "b04609v0-ef13-44vu-98x1-f988mn18max4",
            "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 section in *webMethods API Gateway User's Guide*. 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 321.

**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 section in *webMethods API Gateway User's Guide*. 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 section in *webMethods API Gateway User's Guide* . 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": 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
       }
    ]
}
```

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**.

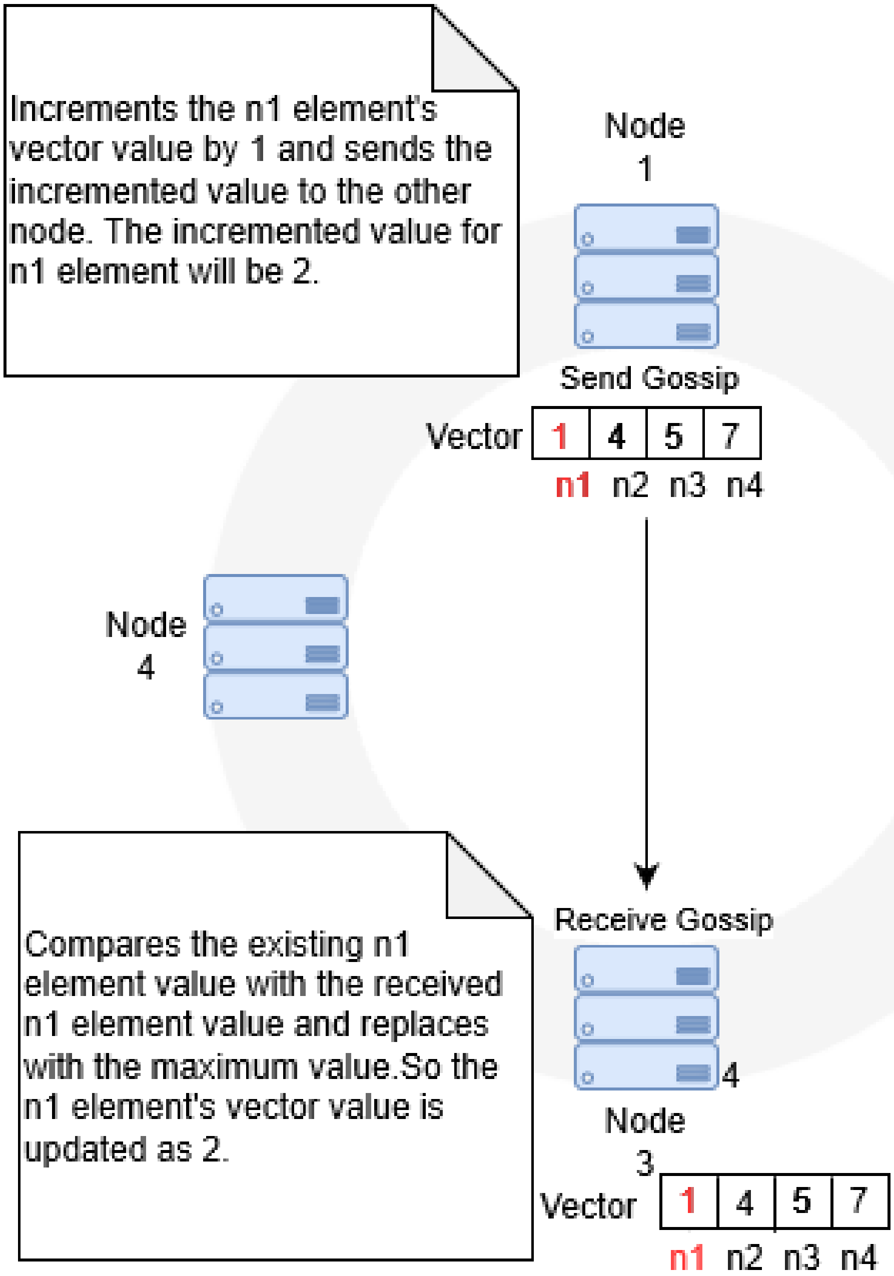
You can validate whether the data center is activated in the respective mode by reading the current configuration of the data center using the **GET/rest/apigateway/dataspace** REST API. For more information, see “How Do I Read the Current Configuration of the Data Center?” on page 321.

**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.


Increments the n1 element's vector value by 1 and sends the incremented value to the other node. The incremented value for n1 element will be 2.
Node 1
Send Gossip
Vector
1
4
5
7
n1 n2 n3 n4
Node 4
Node 2
Compares the existing n1 element value with the received n1 element value and replaces with the maximum value.So the n1 element's vector value is updated as 2.
Receive Gossip
4
Node 3
Vector
1
4
5
7
n1 n2 n3 n4

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 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 321.

## 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,
 "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 321.

## 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": "b04609v0-ef13-44vu-98x1-f988mn18max4",
            "host": "in.myhost.com",
            "port": 4440
        }
    ],
    "ring.active": [
        {
            "nodeName": "a04609a0-ca13-44db-98e1-f988ba18fbb4",
            "host": "us.myhost.com",
            "port": 4440
        },
        {
            "nodeName": "b04609v0-ef13-44vu-98x1-f988mn18max4",
            "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:

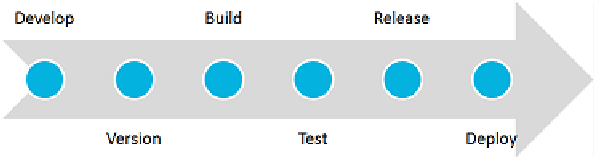
| Extended Setting | Description |
|---|---|
| **pg_Dataspace_GossipInterval** | Specifies how frequently each node should gossip with one another.<br>By default, the value is set to 3 seconds. |
| **pg_Dataspace_TimeToFail** | Specifies the maximum permissible interval between two consecutive gossips.<br>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.<br>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 Extended Setting section in *webMethods API Gateway User's Guide*.

# 10 API Gateway Staging and Promotion

## Overview of Staging, Promotion, and Rollbacks

API Gateway supports staging and promotion of assets. In a typical enterprise-level, solutions are separated according to the different stages of Software Development Lifecycle (SDLC) such as development, quality assurance (QA), and production stages. As each organization builds APIs for easy consumption and monetization, continuous integration (CI) and continuous delivery (CD) is an integral part of the solution. CI is a development practice that requires developers to integrate code into a shared repository several times a day and CD is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time. Development of assets starts at the development stage and once the assets are developed, they are promoted to the QA stage for testing, after testing of the assets is complete, the assets are promoted to the deployment stage.



API Gateway provides tools and features to automate your CI and CD practices. Modifications made to the APIs, policies, and other assets can be efficiently delivered to the application developers with speed and agility. For example, When you publish new applications, the API definitions change. These changes are to be propagated to application developers. The API provider has to update the associated documentation for the API or application. In most cases this process is a tedious manual exercise. You can use API Gateway staging and promotion to address such cases to automate API and policy management that makes deployment faster, introduces continuous innovation with speed and agility. This ensures that new updates and capabilities are automatically, efficiently, and securely delivered to their developers and partners, in a timely fashion and without manual intervention.

Staging and promotion allows you to:

- Promote all the run time assets such as API Gateway APIs, aliases, applications, policies, or admin configurations across different stages.
- Select and promote a subset of assets from one stage to another stage. For example, you can promote a single API and its dependencies from one stage to another.
- Optionally select the asset's dependencies during the promotion.
- Create stage-specific aliases.

  When promoting an alias from source stage to target stage, API Gateway checks if the target stage already has an alias. If so, then API Gateway replaces the existing alias in target stage with the alias that is promoted from source stage.
- Roll back assets in case of incomplete information.
- Repromote assets, if there are any recent changes, to the already promoted stages.

**Note:**

Software AG recommends you to have API Gateway instances across stages to be completely independent. For example, the API Gateway instances from the development stage and the API Gateway instances from the QA stage must not share any resources in common such as databases.

## Promotions

Promotion refers to moving API Gateway assets from one stage to another.

API Gateway staging and promotion allows you to:

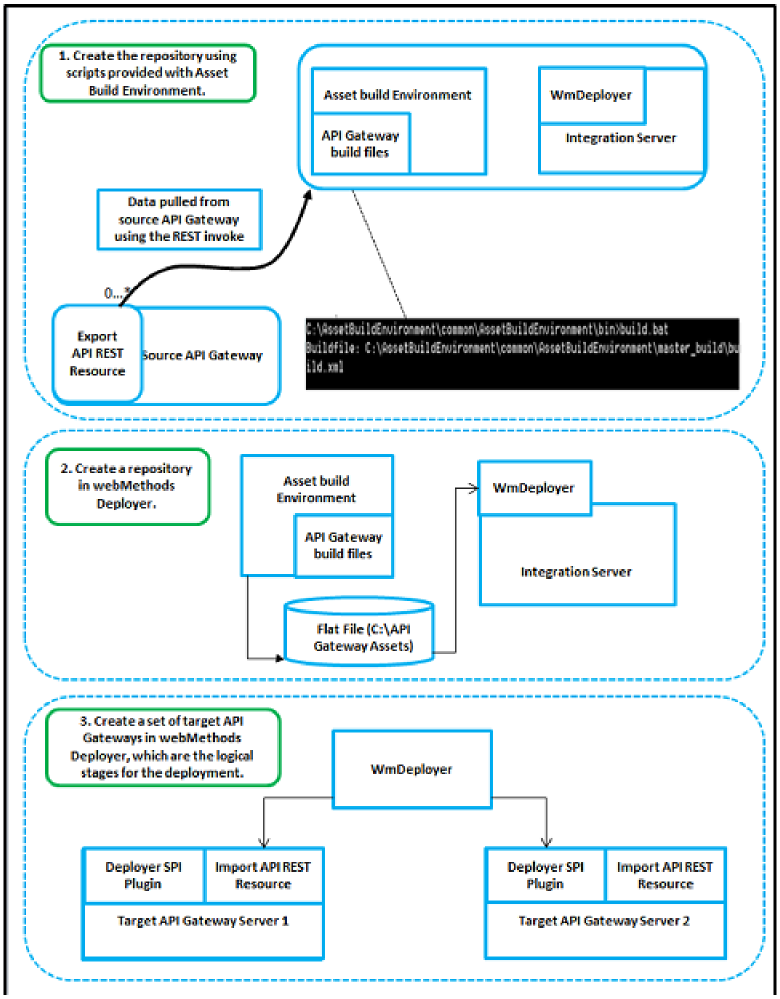
- promote all the run time assets such as API Gateway APIs, aliases, applications, policies, or admin configurations across different stages.
- select and promote a subset of assets from one stage to another stage. For example, you can promote a single API and its policy dependencies from one stage to another.
- select dependencies involved while promoting an asset. For example, while selecting a service for promotion, you must also select the dependent policies, applications, and so on.
- modify values of attributes of selected aliases during promotion.
- roll back assets in case of failures.

**Note:**
During the promotion process ensure that both the source and the target system have the same master password. For more information on promoting assets using the webMethods API Gateway, see *webMethods API Gateway User's Guide*.

## Promoting Assets Using Deployer

You can promote API Gateway assets from one stage to the other using webMethods Deployer. webMethods Deployer is a tool you use to deploy user-created assets that reside on source webMethods runtimes or repositories to target webMethods runtime components (runtimes). For example, you might want to deploy assets you have developed on servers in a development environment (the source) to servers in a test or production environment (the target).

The high level steps involved are as follows:


1. Create the repository using scripts provided with Asset Build Environment.
Asset build Environment
API Gateway build files
WmDeployer
Integration Server
Data pulled from source API Gateway using the REST invoke
0...*
Export API REST Resource
Source API Gateway
2. Create a repository in webMethods Deployer.
Asset build Environment
API Gateway build files
Flat File (C:\API Gateway Assets)
WmDeployer
Integration Server
3. Create a set of target API Gateways in webMethods Deployer, which are the logical stages for the deployment.
WmDeployer
Deployer SPI Plugin
Import API REST Resource
Target API Gateway Server 1
Deployer SPI Plugin
Import API REST Resource
Target API Gateway Server 2

```
C:\AssetBuildEnvironment\common\AssetBuildEnvironment\bin>build.bat
Buildfile: C:\AssetBuildEnvironment\common\AssetBuildEnvironment\master_build\bu
ild.xml
```

For more information on promoting assets using webMethods Deployer , see *webMethods Deployer User's Guide*.

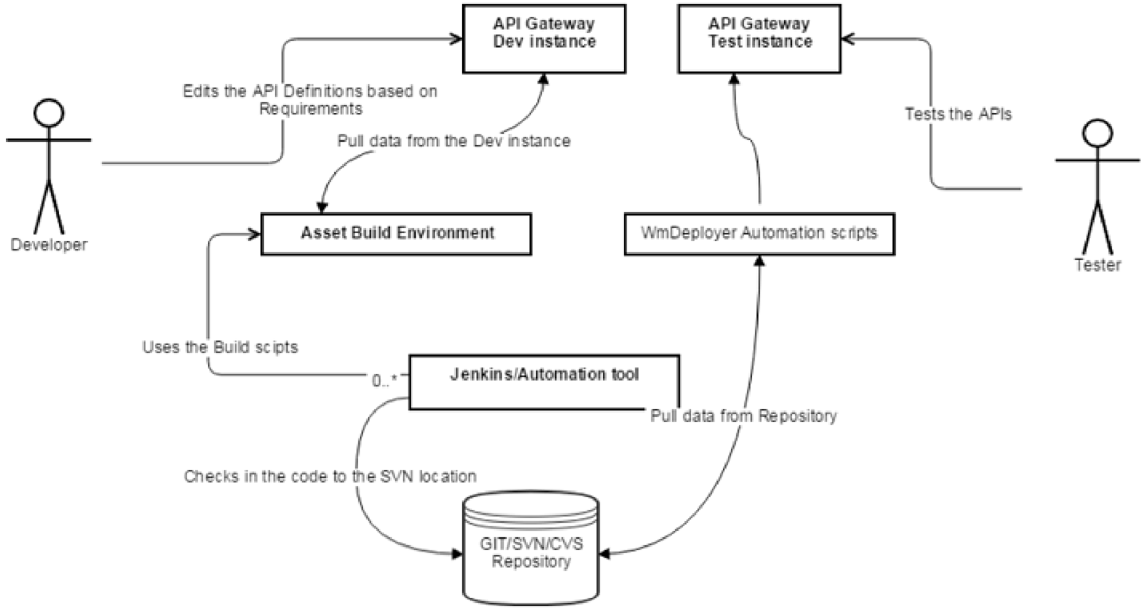
For details about the automation scripts provided by ABE and Deployer and their usage to promote assets from one stage to another, see http://techcommunity.softwareag.com/pwiki/-/wiki/Main/Staging%2C%20Promotion%20and%20DevOps%20of%20API%20Gateway%20assets

## DevOps Use Case using Asset Build Environment and webMethods Deployer

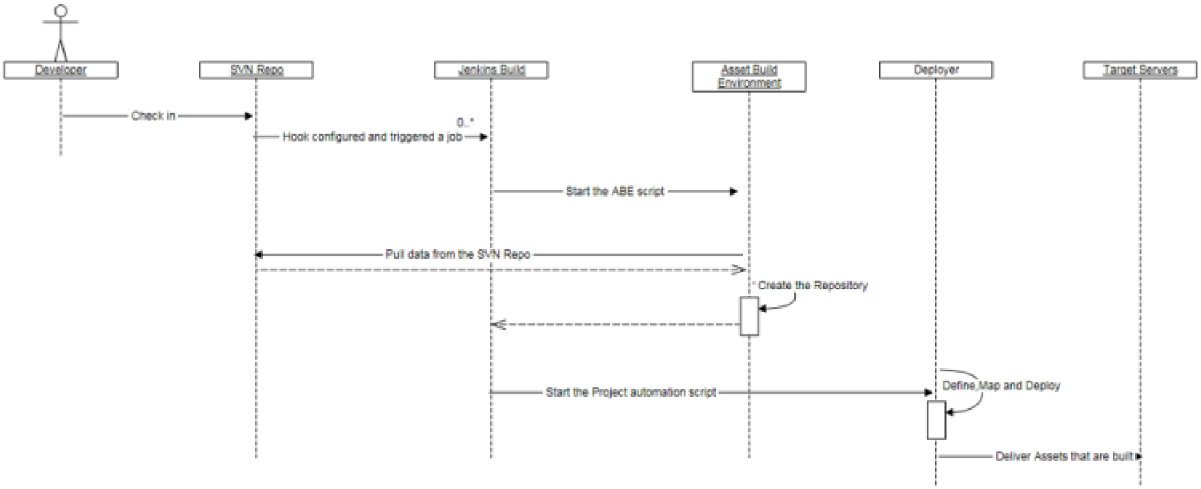
The API Gateway specific scripts that are provided as part of the Asset Build Environment and webMethods Deployer can be used by continuous integration tools like Jenkins. The sample flow is as follows:

1. The developer makes changes to a development API Gateway instance.
2. A Jenkins job then uses the build script to pull data from this development instance and push it to a version control system such as GIT.
3. Another job is used to pull it from a version control system and then use the webMethods Deployer scripts to directly push it to the test instance. In this way, the test instance always have the APIs.

**Sample: Staging workflow**


API Gateway Dev instance
API Gateway Test instance
Edits the API Definitions based on Requirements
Pull data from the Dev instance
Tests the APIs
Developer
Tester
Asset Build Environment
WmDeployer Automation scripts
Uses the Build scipts
0..*
Jenkins/Automation tool
Pull data from Repository
Checks in the code to the SVN location
GIT/SVN/CVS Repository

**Sample: Staging call flow**



For detailed information about promoting assets using webMethods Deployer , see *webMethods Deployer User's Guide*.

## Promoting Assets Using Promotion Management API

The promotion management capabilities allows for moving assets from lower to higher environments. For details, see

API Gateway enables continuous integration (CI) and continuous delivery (CD) practices to be used for development, deployment, and promotion of the APIs, applications, other related assets, and for supporting the use of DevOps tooling. There are different ways in which API Gateway enables continuous integration (CI) and continuous delivery (CD).

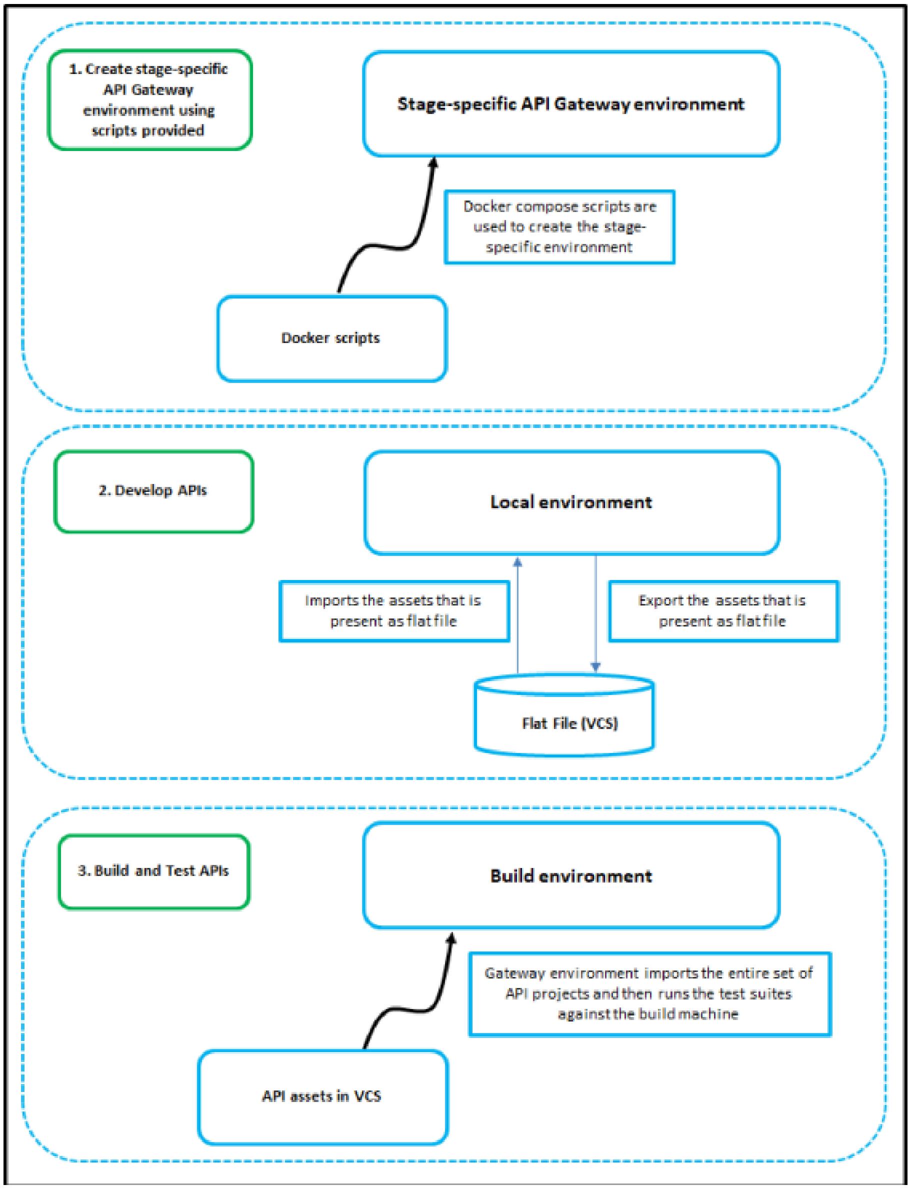
The promotion management REST APIs allow for automation for CI/CD. For detail about the promotion management API, see *webMethods API Gateway User's Guide*.

## DevOps Use Case using Promotion Management APIs

This example explains a sample DevOps use case using the promotion management APIs. You can promote API Gateway assets from one stage to the other using API Gateway specific scripts provided in GitHub. You can use the continuous integration tools like Jenkins and Azure to deploy user-created assets that reside on source API Gateway instance or repositories to a target API Gateway instance. For example, you might want to deploy assets you have developed on an API Gateway instance in a development environment (the source) to an API Gateway instance in a test or production environment (the target).

The high level steps to achieve this are as follows and are depicted in the illustration:

1. Create a stage-specific API Gateway environment.
2. Develop APIs.
3. Test the APIs.


1. Create stage-specific API Gateway environment using scripts provided
Stage-specific API Gateway environment
Docker compose scripts are used to create the stage-specific environment
Docker scripts
2. Develop APIs
Local environment
Imports the assets that is present as flat file
Export the assets that is present as flat file
Flat File (VCS)
3. Build and Test APIs
Build environment
Gateway environment imports the entire set of API projects and then runs the test suites against the build machine
API assets in VCS

For details about various API Gateway-specific scripts and their usage, see https://github.com/SoftwareAG/webmethods-api-gateway-devops.

# 11 Mediator Migration to API Gateway

# Migrating from Mediator to API Gateway

API Gateway supports the migration of Mediator 9.7 and later; the earlier versions of Mediator should be migrated first.

## Migrating Mediator Deployments to API Gateway

Existing Mediator deployments can be migrated to API Gateway by publishing the virtual service, applications, and runtime aliases to API Gateway. This lets you build an API Gateway runtime enforcement landscape in parallel to the existing Mediator landscape.

To migrate the existing Mediator deployments, perform the following procedure:

- For all installed Mediators:
  1. Stop Mediator.
  2. Install corresponding API Gateway.
  3. Migrate Mediator configuration to API Gateway.
- For all Mediator targets configured in CentraSite:
  1. Configure a corresponding API Gateway in CentraSite.
  2. Deploy all virtual services from the Mediator target to the corresponding API Gateway.
  3. (Optional) Undeploy all virtual services from the Mediator target.

**Note:**
The procedure assumes that the Mediators and the corresponding API Gateway provide the same endpoints. Therefore either the Mediator or its corresponding API Gateway can be up and running. If the endpoint compatibility is not required it is not necessary to stop the Mediators. Also undeploying the Mediator deployments is optional. This means Mediator and API Gateway can be driven by CentraSite in parallel.

## Migrating Mediator Configurations to API Gateway

As the publishing of virtual services, applications, and runtime aliases to API Gateway is done through CentraSite, CentraSite is required for migrating Mediator to API Gateway.

To migrate existing Mediator configurations to API Gateway, perform the following procedure:

1. Run IS migration using the IS migration tool.

   For details of the IS migration tool, see *Upgrading Software AG Products On Premises*.

2. Run Mediator migration using the API Gateway migration tool.

   The API Gateway migration tool is available within the IS instance running the API Gateway. If API Gateway is running in the default IS instance the tool is available in the folder: `Install_Dir/IntegrationServer/instances/default/packages/WmAPIGateway/bin/migrate`.

The script migrateFromMediator.sh has two parameters:

- Full path to Integration service installation running the Mediator to be migrated. (for example, E:/SoftwareAG/IntegrationServer)
- Name of the instance that is running the Mediator (for example, default)

On Unix the script can be invoked as follows:

```
./migrateFromMediator.sh /opt/softwareag/IntegrationServer default
```

On Windows the script can be invoked as follows:

```
migrateFromMediator.bat C:\SoftwareAG\IntegrationServer default
```

3. Start API Gateway.

The Mediator configuration migration covers the following configuration items:

- Elasticsearch
- SNMP
- Email
- HTTP Configuration
- Keystore Configuration
- Ports Configuration
- Service Fault
- Extended Settings

The following configuration items are not automatically migrated. The configuration of these items have to be done manually in API Gateway.

- Security Token Service (STS) Configuration
- `apig_rest_service_redirect` parameter: When you set this to `true`, the apig_rest_service_redirect in the extended Administration setting in API Gateway REST requests against the /mediator directive will be redirected to the /gateway directive. This means that REST requests can be sent to /mediator and to /gateway.

**Note:**

- The Mediator configuration migration can only be applied to a fresh API Gateway installation once.
- On migrating from Mediator to API Gateway, API Gateway does not modify or change anything that is part of the incoming request. The incoming request along with the query parameters or headers is forwarded to the native service as it is without any modification. If you require API Gateway to remove any invalid query parameters, in API Gateway UI, add webMethods IS service under **Request transformation policy** > **Advanced Transformation**, configure any flow service and select **Comply to IS spec**.