webMethods API Gateway Configuration Guide

Version 10.7

October 2020

WEBMETHODS

Document ID: YAI-CG-107-20240221

Table of Contents

About this Guide

This guide 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: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at http://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 TECHcommunity

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

- Access product documentation, if you have TECHcommunity 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 - Ports - License management - Load balancing - Keystore configuration	Yes	Yes
Administration - Extended settings	No	Yes
Alias management	No	Yes
Service management	No	Yes
Policy management - Threat protection rules	Yes	Yes
Policy management - Global policies - Policy templates	No	Yes
Export and Import	No	Yes

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

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.

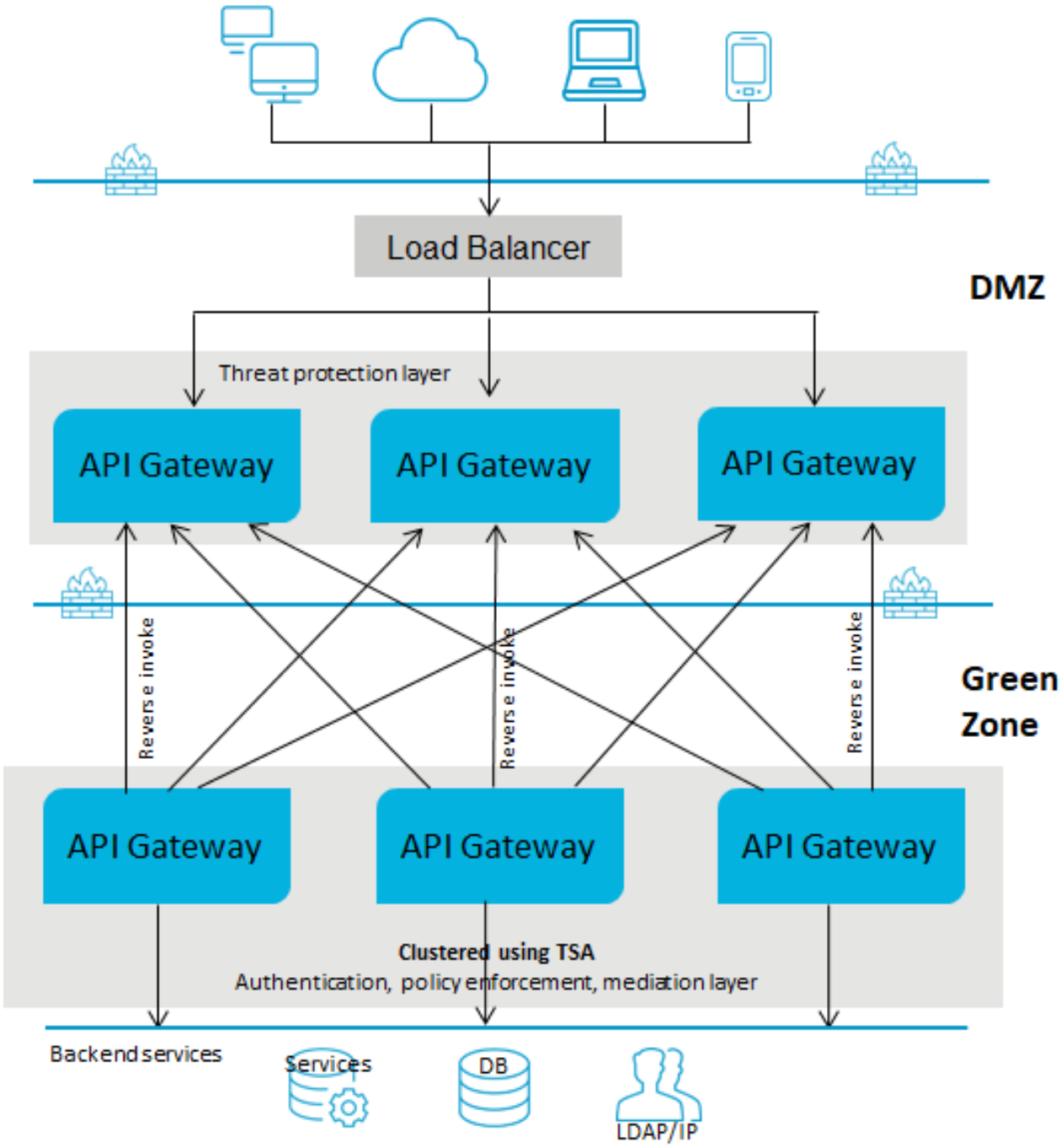
Deployment scenario 1: Paired gateway deployment

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. To learn more about reverse invoke, see "Reverse Invoke Configuration in API Gateway" on page 16.

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:



In a typical paired deployment scenario (2 API Gateways connected through Reverse Invoke), you have a standard edition API Gateway in DMZ and advanced edition API Gateway in green zone. But there are cases when you have both the DMZ API Gateway and green zone API Gateway are advanced editions. In such a setup, in most of the cases, the customer's APIs are deployed in the green zone and the requests to the API Gateway's internal APIs like pub/apigateway/oauth2/getAccessToken, /pub/apigateway/oauth2/authorize etc must be processed in the green zone API Gateway. Hence the property **forwardInternalAPIsRequest** must be set in the DMZ API Gateway as true so that the DMZ API Gateway simply forwards the requests to internal APIs to the API Gateway in the green zone. To learn more about the list of services to be

exposed for API communication, see the **API Gateway services to be exposed for API Portal and client communication** section of the *webMethods API Gateway User's Guide*.

Note:
Software AG recommends you to not cluster the Standard Edition API Gateways in a DMZ in a paired deployment setup.

To learn how to configure threat protection and invoke an API using REST API, read the **API Gateway standard edition in DMZ & API Gateway advanced edition in Green zone** section from the Threat protection in API Gateway article.

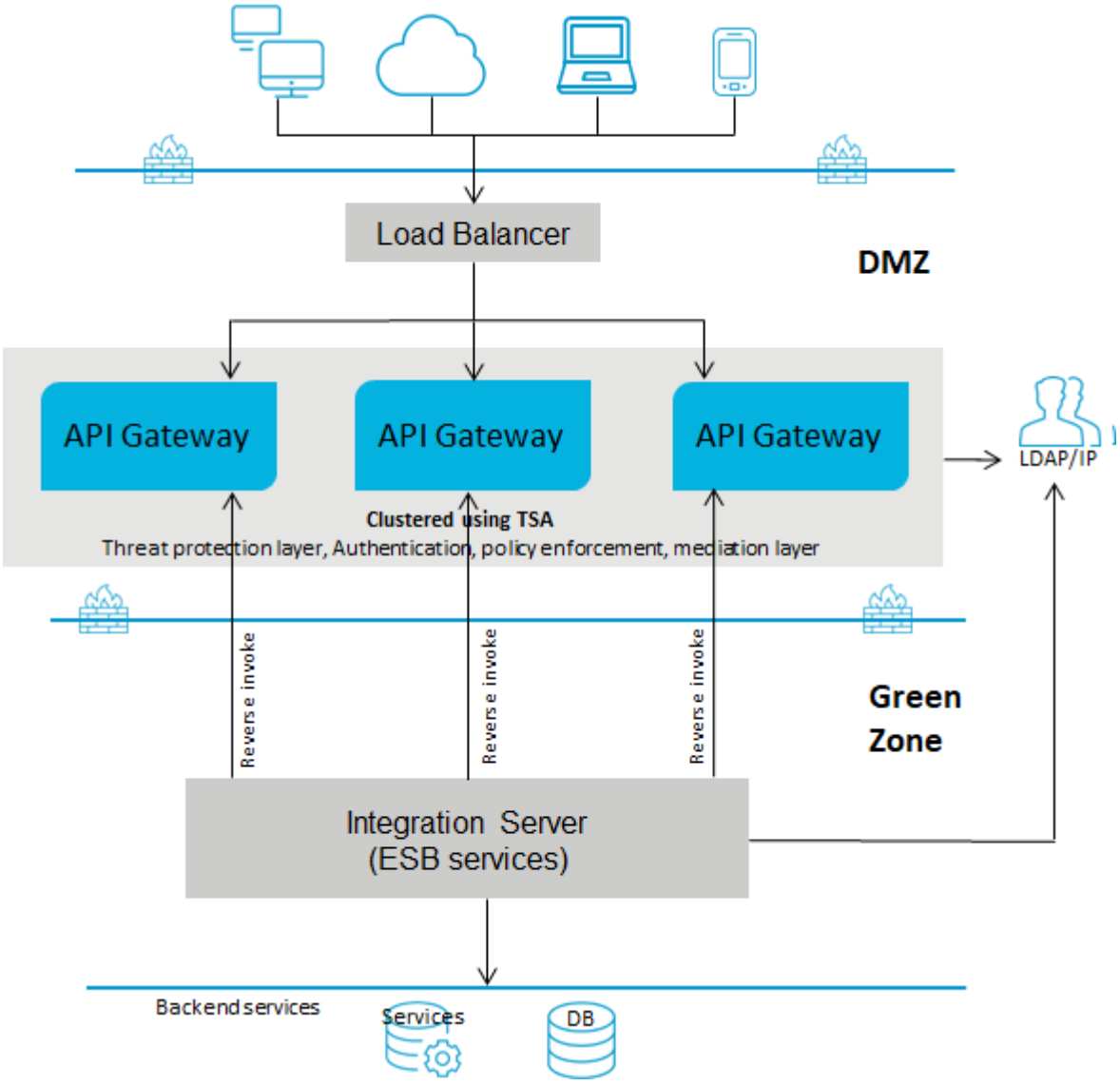
Deployment scenario 2: API Gateway in the DMZ with reverse invoke configuration

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 more information on ports and routing policies, see *webMethods API Gateway User's Guide* .

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

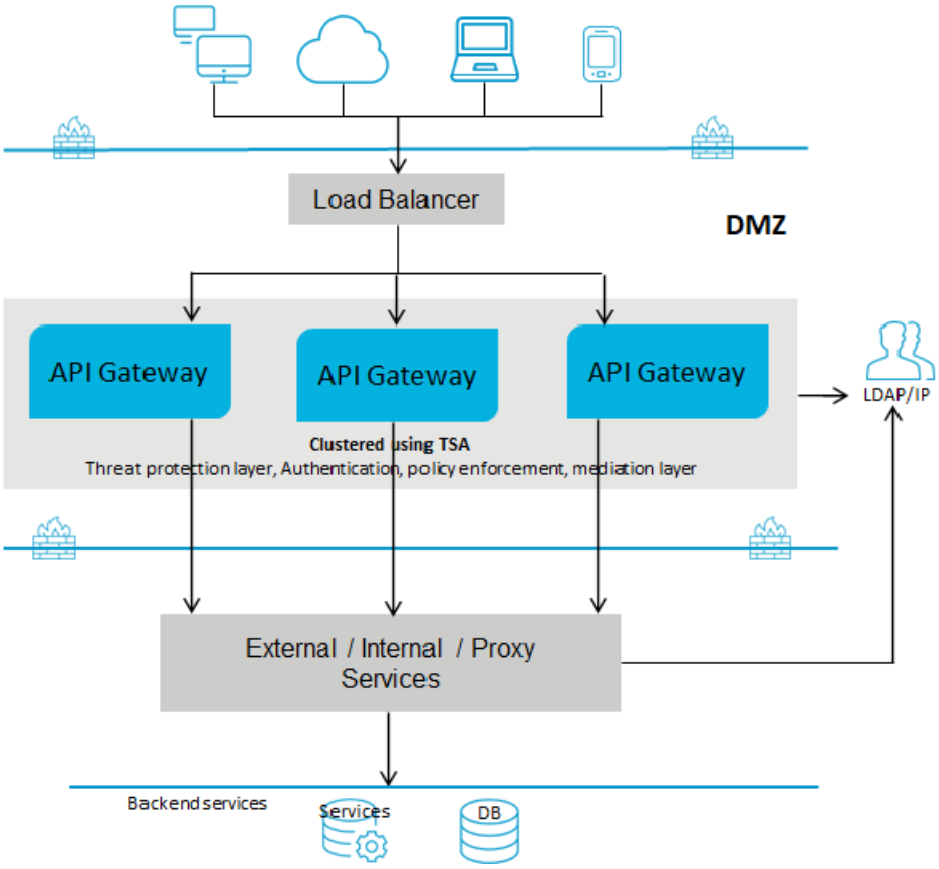


Deployment scenario 3: API Gateway with a Load Balancer in the DMZ

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 in the green zone for webMethods customers:

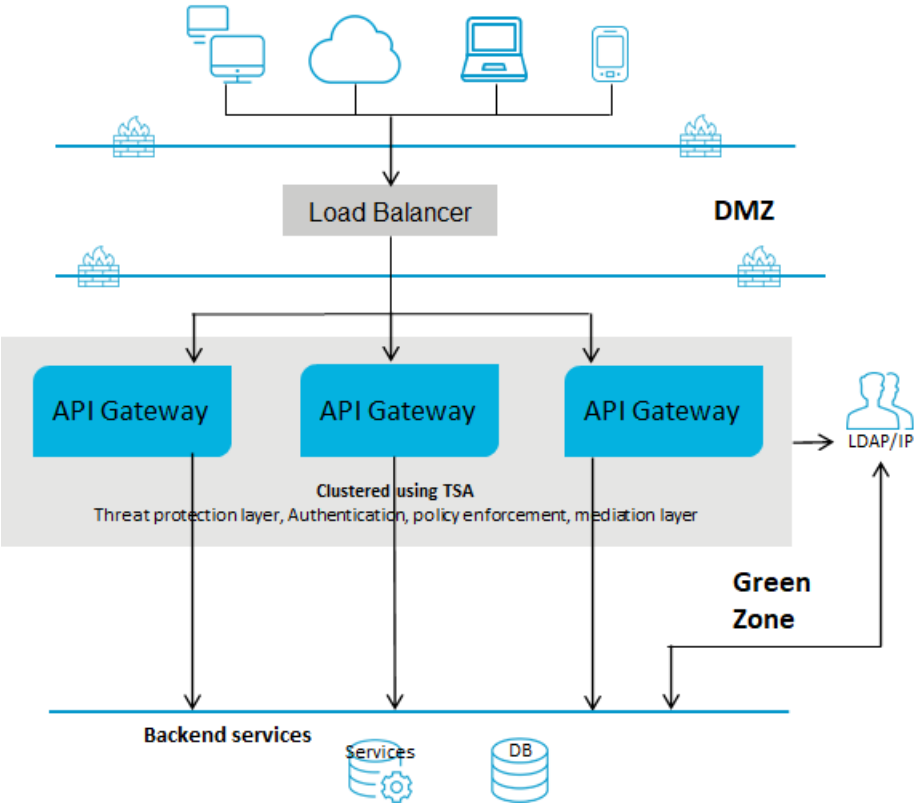


Deployment scenario 4: API Gateway in the green zone with a Load Balancer in the DMZ

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.

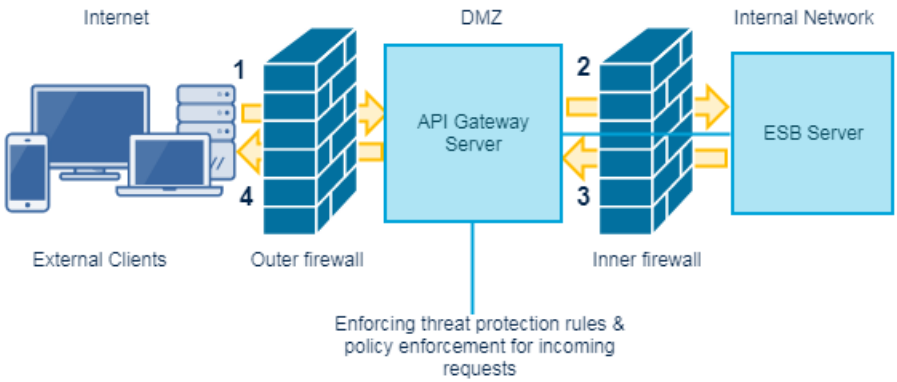
The following diagram provides an architectural overview of the API Gateway deployment for non webMethods customers:



Reverse Invoke Configuration in API Gateway

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

The following image shows how an internal server (API Gateway or webMethods Integration Server) in the green zone connects to an API Gateway in the DMZ zone.



In a reverse invoke scenario, you must have paired ports in the external server (external port used for external clients and a registration port). The external client's requests are routed to external

port which in turn routes the requests to its paired port that is the registration port. You must also create an internal port in the internal server, which responds to requests present in the registration port.

Note:

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

For more information on ports, see *webMethods API Gateway User's Guide*.

In a paired gateway deployment scenario, if your API Gateway is behind an internal firewall and is not allowed to accept communications from external clients through the DMZ, then you can configure the API Gateway instance in DMZ with an external port to listen to the requests from external clients. You can then use reverse invoke to route the requests to the internal servers. The API Gateway internal listener port or the WebSocket listener port receives the requests from the registration port of the API Gateway present in DMZ, thus eliminating the need to open a port in the internal firewall to let the traffic through. External clients send requests to the API Gateway present in DMZ. The DMZ API Gateway's external port listens to these requests and evaluates each request against the API Gateway rules that have been defined. The DMZ API Gateway then passes the requests that have not violated any rules, to the to the registration port from where the internal server pulls the requests of the internal network. These listener ports process the requests and send the responses to the DMZ API Gateway, which then passes the responses back to the client.

You can configure reverse invoke setup in one of the following ways:

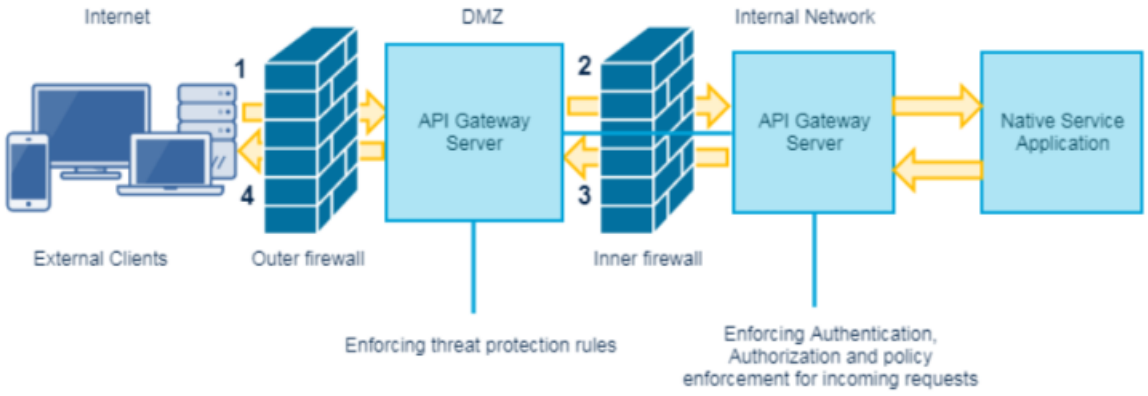
- "Connecting API Gateway in Green Zone to API Gateway in DMZ " on page 17.
- "Connecting Integration Server in Green Zone to API Gateway in DMZ" on page 20.

Connecting API Gateway in Green Zone to API Gateway in DMZ

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

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

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



To configure reverse invoke

1. Configure External and Registration ports on API Gateway in DMZ.

   a. Log on to API Gateway as an Administrator user.

   b. Expand the menu options icon, in the title bar, and select **Administration**.

   c. Navigate to **Security** > **Ports**.

   d. Click **Add ports**.

   e. Select **API Gateway external** option from the **Type** drop-down menu.

   f. Click **Add**.

   g. Provide the required information in the **API Gateway external listener configuration** to configure the External port.

      The important fields to be configured are **External port**, **Alias**, **Protocol**, **Backlog**, and **Keep alive timeout**. For more information on ports, see *webMethods API Gateway User's Guide* .

   h. To configure two-way SSL, select `HTTPS` in the **Protocol** field and select one of the following options in the **Client authentication** field in the in **API Gateway external listener configuration** section.

- **Request client certificate**. This option requests for a certificate from the client. However, even if the client does not provide a valid certificate, the connection is established.
- **Require client certificate**. This option requests for a certificate from the client. If the client does not provide a valid certificate, the connection is not established. If you select this option, you must also configure the **Truststore alias** field.

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

The important fields to be configured are **Registration port**, **Alias**, and **Protocol**. For more information on ports, see *webMethods API Gateway User's Guide* .

j. If you execute step h, perform the following steps to configure two-way SSL.

a. Select `HTTPS` in the **Protocol** field and select `Require client certificate` in the **Client authentication** field.

b. Configure the **Keystore alias** and **Truststore alias** fields.

2. Execute the following steps in the Green zone API Gateway.

a. Create an API Gateway internal port.

b. Select `HTTPS` in the **Protocol** field.

c. In the API Gateway external server section, enter the hostname of the DMZ API Gateway in the **Host** field.

d. Type the port number of the API Gateway registration port of DMZ API Gateway in the **Port** field.

e. In the Registration credentials section, provide the following information.

- **Keystore alias**. Select a Keystore.
- **Key alias(signing)**. Select a Key alias.
- **Truststore alias**. Select Truststore. If you select a Truststore, 2-way SSL is enabled.

3. Configure the internal port of the API Gateway in green zone with the Registration port of API Gateway in DMZ.

4. Configure Load Balancer URL in API Gateway present in the green zone.

a. Expand the menu options icon, in the title bar, and select **Administration**.

b. Navigate to **General** > **Load Balancer**.

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

For more information on load balancers, see *webMethods API Gateway User's Guide*.

5. Create an API in the internal API Gateway Server with routing protocol and endpoint as the Native API. For more information on how to create APIs, see *webMethods API Gateway User's Guide*.

6. You can now access the API by using the URL in the format `http://externalserver:externalport`/gateway/`api-name/resource-path`.

   If the API Gateway in green zone runs out of registration connections, it issues the following error message:

   number requests waiting for a registration connection.

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

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

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

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

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

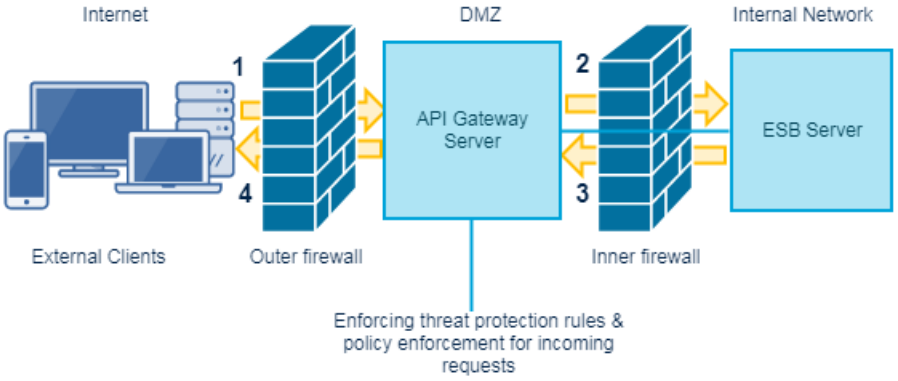
Connecting Integration Server in Green Zone to API Gateway in DMZ

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

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

Important:

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



To configure reverse invoke

1. Configure External and Registration ports on API Gateway in DMZ.

   a. Log on to API Gateway as an Administrator user.

   b. Expand the menu options icon, in the title bar, and select **Administration**.

   c. Navigate to **Security** > **Ports**.

   d. Click **Add ports**.

   e. Select **API Gateway external** option from the **Type** drop-down menu.

   f. Click **Add**.

   g. Provide the required information in the **API Gateway external listener configuration** to configure the External port.

      The important fields to be configured are **External port**, **Alias**, **Protocol**, **Backlog**, and **Keep alive timeout**. For more information on ports, see *webMethods API Gateway User's Guide*.

   h. To configure two-way SSL, select `HTTPS` in the **Protocol** field and select one of the following options in the **Client authentication** field in the in **API Gateway external listener configuration** section.

- **Request client certificate**. This option requests for a certificate from the client. However, even if the client does not provide a valid certificate, the connection is established.
- **Require client certificate**. This option requests for a certificate from the client. If the client does not provide a valid certificate, the connection is not established. If you select this option, you must also configure the **Truststore alias** field.

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

The important fields to be configured are **Registration port**, **Alias**, and **Protocol**. For more information on ports, see *webMethods API Gateway User's Guide*.

j. If you execute step h, perform the following steps to configure two-way SSL.

a. Select `HTTPS` in the **Protocol** field and select `Require client certificate` in the **Client authentication** field.

b. Configure the **Keystore alias** and **Truststore alias** fields.

2. Configure Load Balancer URL in API Gateway.

a. Expand the menu options icon, in the title bar, and select **Administration**.

b. Navigate to **General** > **Load Balancer**.

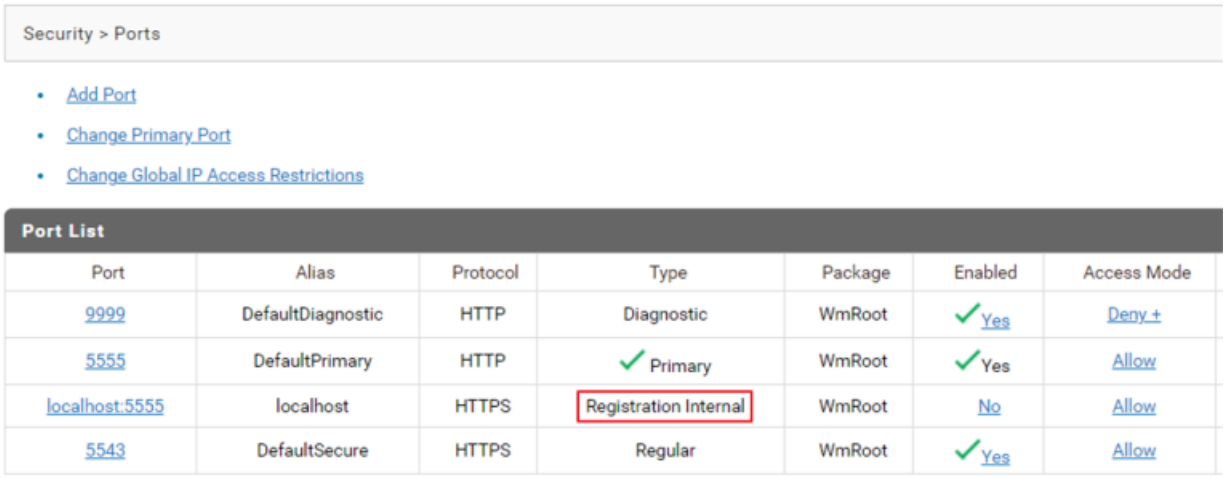
Provide the configured external port or an external Load Balancer URL. The API endpoints have this port for external consumers. If you have a Load Balancer, then the requests from the Load Balancer must be directed to API Gateway's External port.

For more information on load balancers, see *webMethods API Gateway User's Guide*.

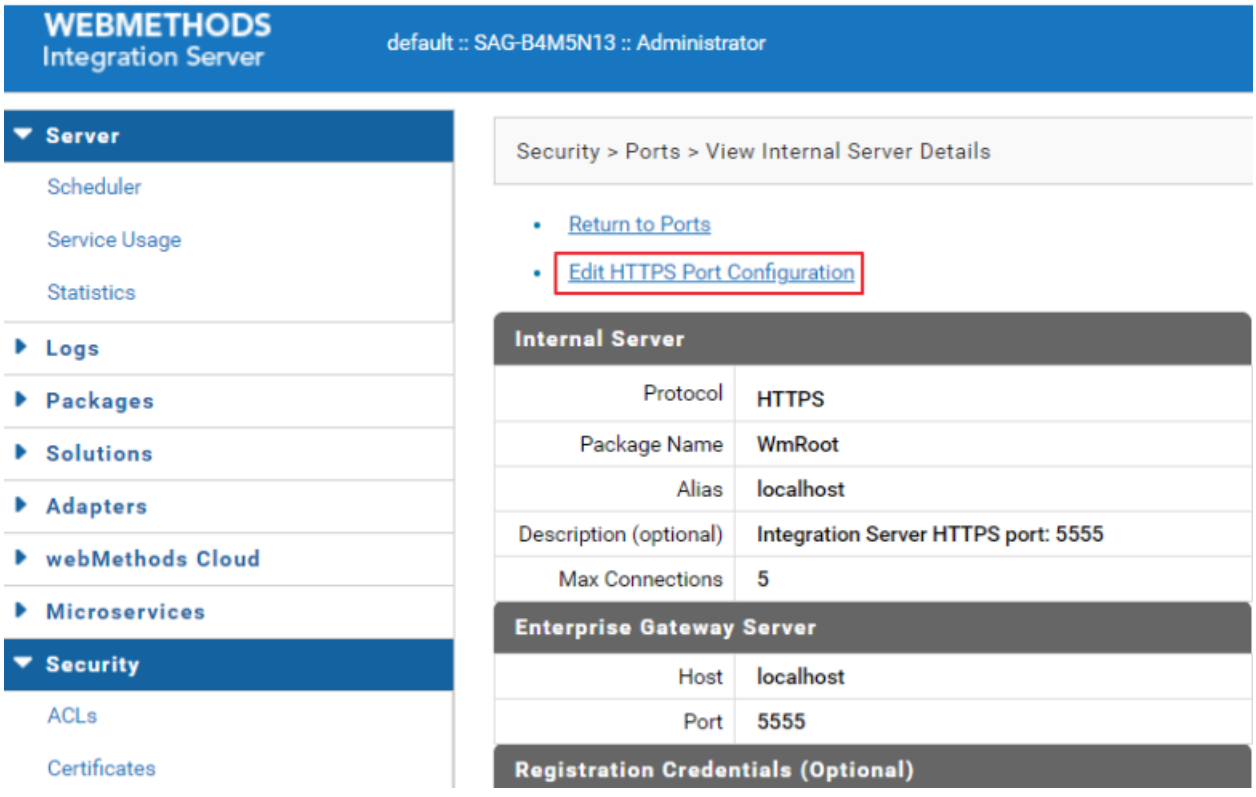
3. In the Green Zone Integration server, perform the following configurations to setup two-way SSL.

a. Navigate to `Security > Ports`.

b. Select the Registration Internal port of the API Gateway.



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



d. Select the `Yes` option in the **Enable** field.

e. Configure the fields, as required.

f. In the Registration credentials section, configure the **Keystore Alias** and **Truststore Alias** fields.

g. Select `Require Client Certificates` in the **Client Authentication** field.

h. Click **Save Changes**.

- Return to Ports

Internal Server	
Enable	(•) Yes ( ) No
Protocol	( ) HTTP (•) HTTPS
Package Name	WmRoot
Alias	localhost
Description (optional)	Integration Server HTTPS port: 5555
Max Connections	5
Threadpool	Enable
Enterprise Gateway Server	
Host	localhost
Port	5555
Registration Credentials (Optional)	
User Name	
Password	•••••
Use JSSE	(•) Yes ( ) No
Keystore Alias	DEFAULT_IS_KEYSTORE
Key Alias	ssos
Truststore Alias	DEFAULT_WMCLOUD_TRUSTSTORE
External Client Security	
Client Authentication	Require Client Certificates
	Must match Client Authentication setting on the Enterprise Gateway Server External Port.

Save Changes

4. Configure API Routing Endpoints with Registration Port Alias.

   a. Create an API in API Gateway.

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

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

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

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

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

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

5. Configure Internal Server Port on the ESB Server (internal network).

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

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

6. You can now access the API by using the URL in the format *`http://externalserver:externalport`*/gateway/*`api-name/resource-path`*.

   If the Internal Server runs out of registration connections, it issues the following error message:

   number requests waiting for a registration connection.

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

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

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

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

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

Note:
Since the Client Authentication in Integration server is set to Require client certificates, you must provide a certificate and map it to a user. To learn more about how to do this, see "Importing a Certificate and Mapping to User" on page 27.

Importing a Certificate and Mapping to User

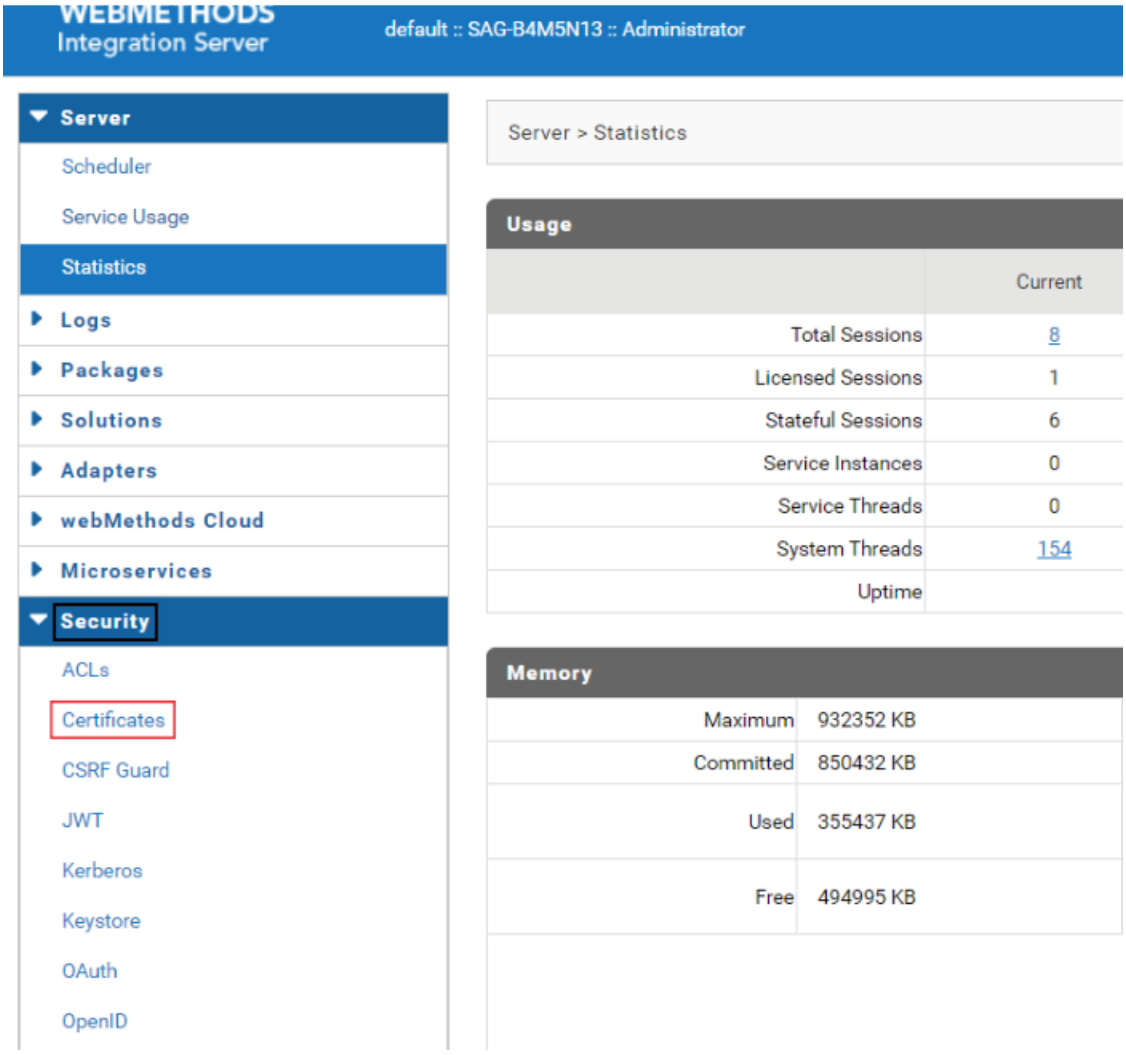
You can import client certificates and CA signing certificates through Integration Server Administrator to keep them on file, map them to particular user accounts, and specify how they are to be used. The user mapping to the certificate must be performed on the external server.

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

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

To import a client certificate and map it to a user

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



3. Click **Configure Client Certificates**.

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

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

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

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

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

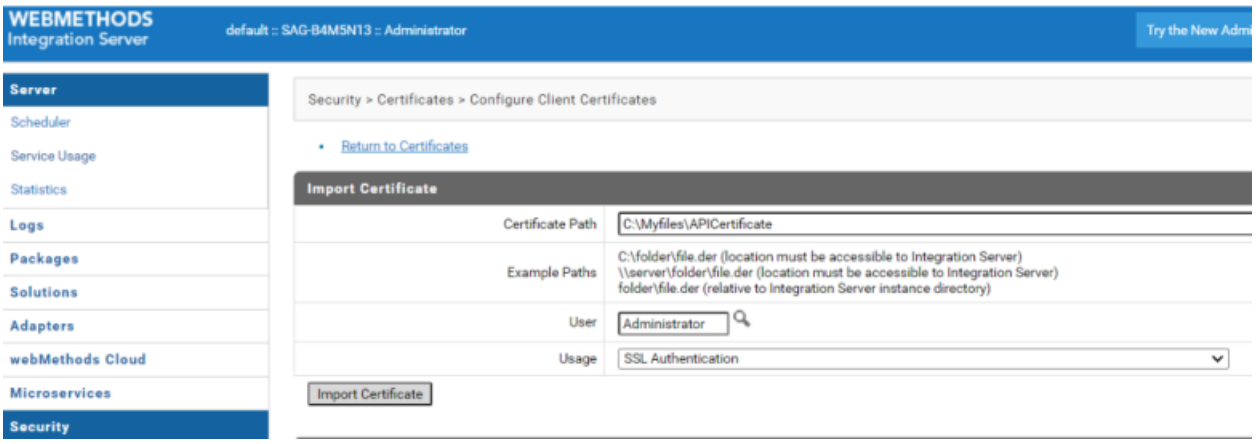
   - To select a local user, select `Local` in the **Provider list**. Select the local user to which you want to map the certificate. If you have not configured an external user directory, you cannot view the Provider list.
   - To select a user from an external directory (LDAP or a central user directory), select the user directory that you want to search, in the **Provider list**. In the **Search** field, enter the

criteria that you want to user to find a user and click **Go**. Select the user to which you want to map the certificate.

6. Select one of the following options from the **Usage** field.

   - **SSL Authentication**. Use the certificate to represent the client's authentication credentials when making an SSL connection with Integration Server.
   - **Verify**. Use the certificate's public key to verify the authenticity of documents, messages, or streams originating from the client and containing a digital signature.
   - **Encrypt**. Use the certificate's public key to encrypt outgoing documents, messages, or streams from Integration Server to the client.
   - **Verify and Encrypt**. Use the same certificate both to verify the authenticity of documents, messages, or streams originating from the client and containing a digital signature, and to encrypt outgoing documents, messages, or streams from Integration Server to the client.
   - **Message Authentication**. Use the certificate to represent the client's authentication credentials when making an SSL connection with Integration Server, when using message-level rather than transport-level authentication (for example, with web service messages whose SOAP message headers contain SSL certificate information).

7. Click **Import Certificate**.

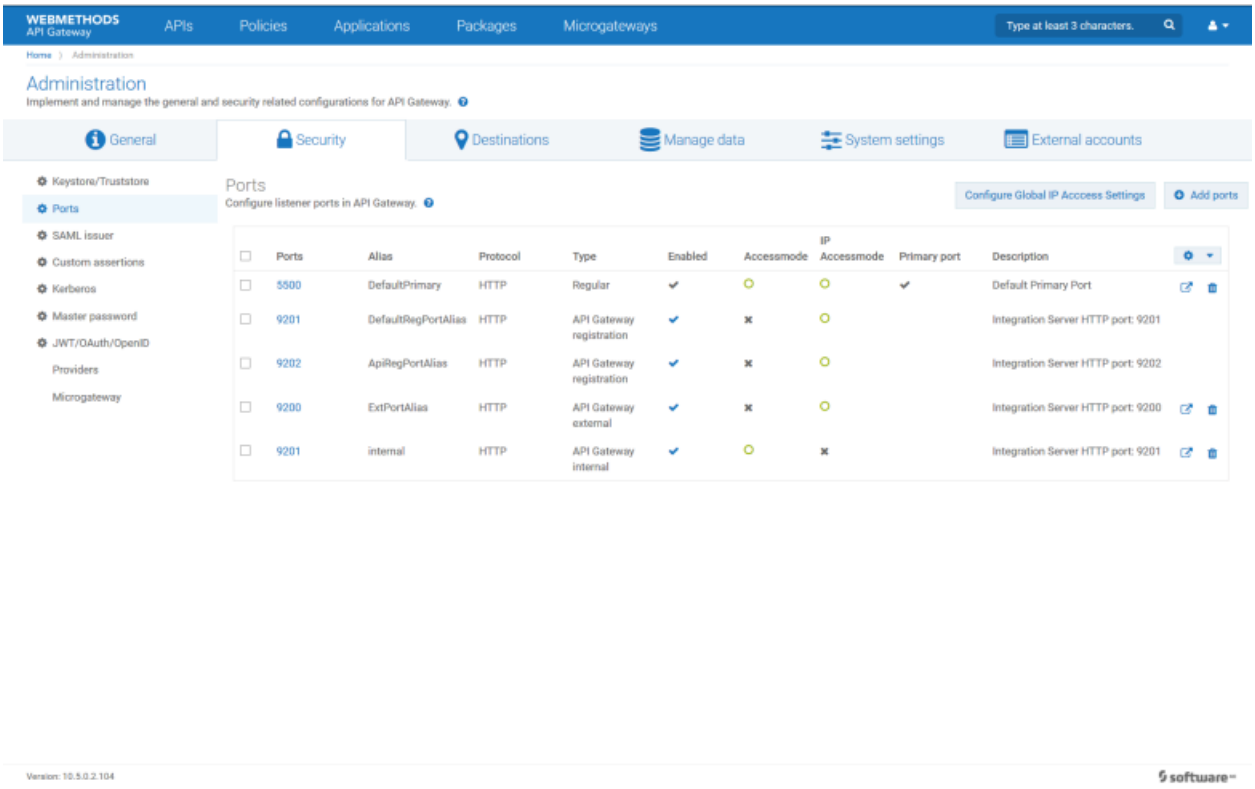


Blocking Incoming Requests to Green Zone

While configuring reverse invoke, you must ensure that you block any requests coming to either API Gateway or the Integration Server present in the green zone.

To block the requests to green zone

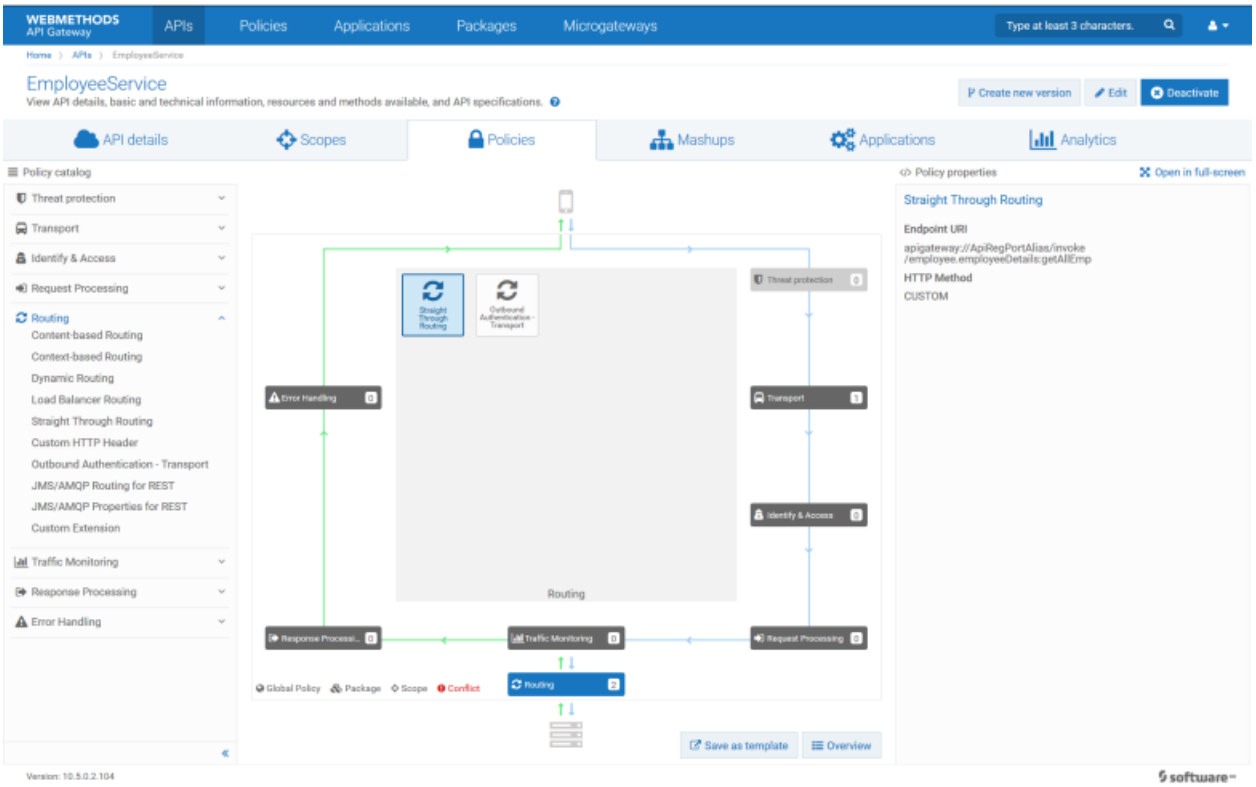
1. Configure the ports in API Gateway located in DMZ as follows.



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 DMZ API Gateway. This ensures that requests are not forwarded to the Integration Server or API Gateway in the green zone, but processed within the DMZ API Gateway. The default registration port is the first one defined for an external port. There is no specific naming required.

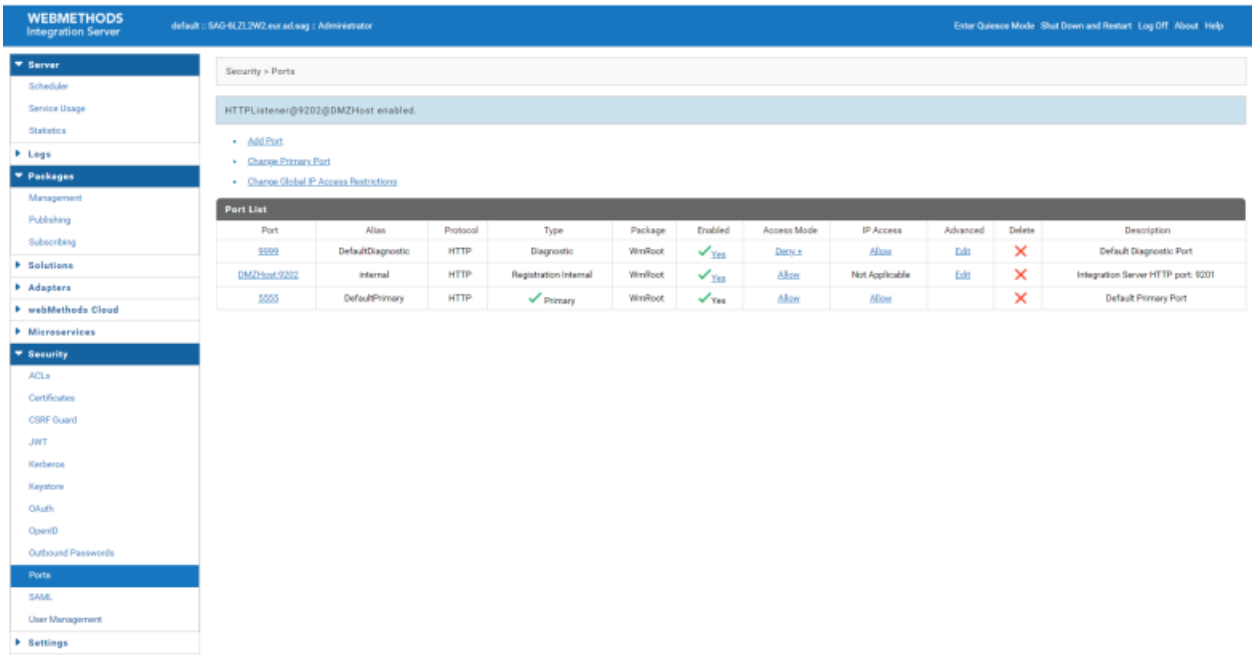
To forward API requests to the backend services, the API routing policies must point to the ApiRegPortAlias, the second registration port alias defined for the external port.

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



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 green zone Integration Server. As mentioned above, 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 DMZ API Gateway where the backend flow services are not defined.

3. Configure the internal port of the Integration Server as follows.



Reverse Invoke Connectivity Properties

This section describes the properties that you can configure to maintain the connectivity in a paired API Gateway deployment.

watt.server.rg.internalregistration.timeout

Specifies the time, in seconds, a connection on the internal server (API Gateway or Integration 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 API Gateway and the DMZ 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 internal API Gateway and DMZ API Gateway 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 the internal server does not receive any requests from DMZ API Gateway, 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 DMZ API Gateway 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.

To prevent a client request from waiting indefinitely, you can configure the **watt.server.rg.internalsocket.timeout** property. This property allows you to set a time-period until which the Integration Server waits to establish a connection to the internal server. If no connection is established in the set time-period, a HTTP 500-Internal Server error is returned.

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 Gateway Data Store

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

Starting, Stopping, and Restarting API Gateway Data Store

API Gateway Data store uses Elasticsearch 7.7.1. For details on the Elasticsearch versions that are compatible with different API Gateway versions, see "API Gateway, Elasticsearch, Kibana Compatibility Matrix" on page 93.

You can start, stop, and restart your API Gateway 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 Gateway Data Store instance.

Note:
Elasticsearch uses the OS temp directory when the **ES_TMPDIR** environment variable is not configured. If the OS temp directory does not have the executing permissions, the Elasticsearch does not start.

Starting and Stopping API Gateway Data Store in Command Central

Starting API Gateway Data Store in Command Central

Use the following procedure to start API Gateway Data Store in the Command Central web user interface.

To start API Gateway Data Store

1. In Command Central, navigate to **Environments > Instances > All > API Gateway Data Store**.
2. Click the status icon for API Gateway Data Store .
3. From the **Lifecycle Actions** drop-down menu, select **Start**.

Stopping API Gateway Data Store in Command Central

Use the following procedure to stop API Gateway Data Store in the Command Central web user interface.

To stop API Gateway Data Store

1. In Command Central, navigate to **Environments > Instances > All > API Gateway Data Store**.
2. Click the status icon for API Gateway Data Store .
3. From the **Lifecycle Actions** drop-down menu, select **Stop**.

Starting, Stopping, and Restarting API Gateway Data Store on Windows

When you install API Gateway Data Store on a Windows operating system, you can start and stop your API Gateway Data Store instance using the Windows Start menu or using scripts.

To start or stop API Gateway Data Store using the Windows Start menu, go to **Start** > ***product install folder***, select **Start API Gateway Data Store 10.7** or **Stop API Gateway Data Store 10.7** respectively.

To start, stop, or restart API Gateway Data Store using scripts, run:

- Start API Gateway Data Store -

  Software AG_directory \InternalDataStore\bin\config\startup.bat

- Stop API Gateway Data Store -

  Software AG_directory \InternalDataStore\config\bin\shutdown.bat

- Restart API Gateway Data Store -

  Software AG_directory \InternalDataStore\config\bin\restart.bat

Starting, Stopping, and Restarting API Gateway 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 Gateway Data Store by running the following commands on LINUX:

- Start API Gateway Data Store.

```
./startup.sh
```

- Stop API Gateway Data Store.

```
./shutdown.sh
```

- Restart API Gateway Data Store.

```
./restart.sh
```

Changing the API Gateway 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 Gateway Data Store HTTP port

1. In Command Central, navigate to **Environments > Instances > All > API Gateway 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 63.
   - Through Command Central. For details, see "Configuring Elasticsearch Connection Settings" on page 170.
8. Restart the API Gateway instance.

Changing the API Gateway 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 Gateway Data Store TCP Port

Java clients use the TCP port to make calls to API Gateway Data Store. In addition, the nodes in an API Gateway 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 Gateway Data Store TCP port

1. In Command Central, navigate to **Environments > Instances > All > API Gateway 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 Gateway 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 Gateway Data Store Cluster" on page 41.

Configuring an API Gateway Data Store Cluster

You can run an API Gateway Data Store instance as a single node, or you can configure multiple API Gateway Data Store instances to run as a cluster to provide high availability and redundancy.

You can configure API Gateway Data Store Cluster in one of the following ways:

- Through Command Central
- Through elasticsearch.yml file

This section describes configuring an API Gateway Data Store cluster through Command Central. For details on configuring a cluster using the elasticsearch.yml file, see "API Gateway Data Store Cluster Configuration" on page 51.

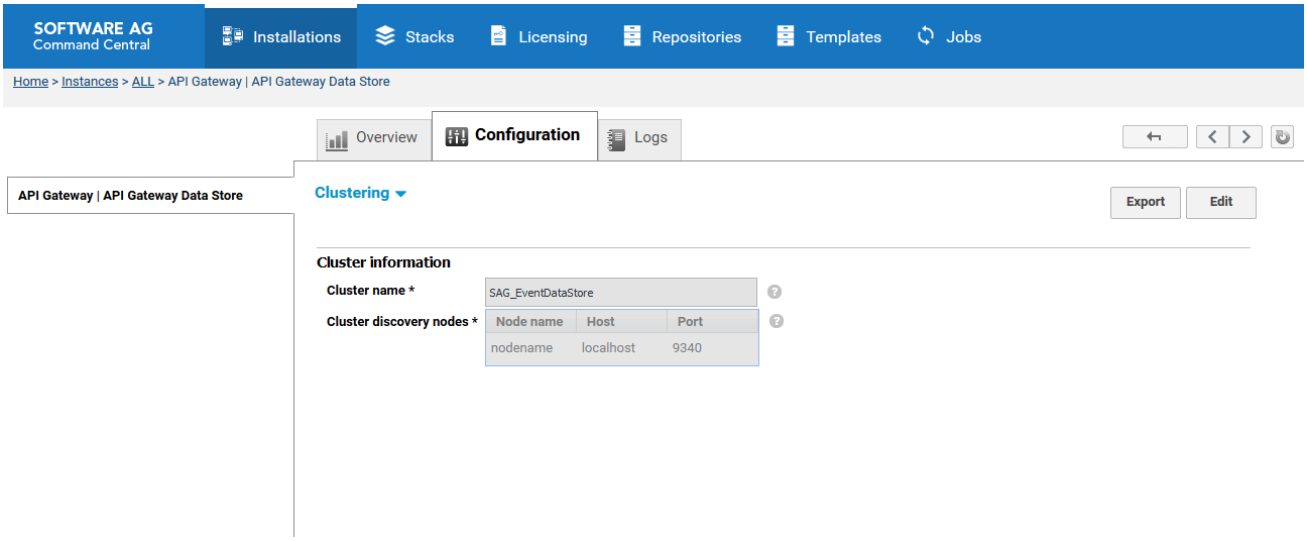
You must specify at least one host and port pair for your configuration in Command Central. API Gateway Data Store comes with a default host and port pair.

To configure an API Gateway Data Store cluster

1. In Command Central, for each API Gateway Data Store instance that is part of the cluster, navigate to **Environments > Instances > All > API Gateway 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 Gateway Data Store instance that is part of the cluster: a. In the **Host** column, specify the host information for an API Gateway Data Store instance. The default host is `localhost`. b. In the **Port** column, specify the port for an API Gateway Data Store instance. The default port is `9340`. c. In the **Node name** column, specify the provide the node name details of the API Gateway Data Store instance. Ensure that this

Field	Description
	name matches with node.name property of the Elasticsearch instance.



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 Gateway 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 Gateway Data Store Properties

You can specify custom properties for your Data Store configuration.

To specify custom properties for API Gateway Data Store

1. In Command Central, navigate to **Environments > Instances > All > API Gateway 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 Gateway Data Store instance.

Configuring Elasticsearch Properties

From Command Central, you can edit the properties of Elasticsearch that are used by API Gateway 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 Gateway Data Store instance that is part of the cluster, navigate to **Environments > Instances > All > API Gateway 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 Gateway 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}
```

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.

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.

For information on installing and configuring Search Guard, see " How do I Secure API Data Store Communication using HTTPS with Search Guard Plugin? " on page 123.

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 Gateway Data Store

You can manage API Gateway 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 Gateway Data Store Supports

API Gateway 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*.
`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 Gateway Data Store Supports

The following table lists the configuration types that the API Gateway Data Store run-time component supports, along with the description of each configuration type:

Configuration Type	Description
COMMON-CLUSTER	Settings for an API Gateway Data Store cluster. You can configure the name of the cluster and the host and port pairs of the server endpoints of the cluster. **Note:** The changes that you make to a cluster configuration take effect after you restart API Gateway Data Store.

Configuration Type	Description
COMMON-PORTS	Configuration instances for HTTP and TCP ports.
CUSTOM-PROPERTIES	Additional properties for the configuration of an API Gateway Data Store server.

Run-Time Monitoring Statuses for API Gateway Data Store

The following table lists the run-time statuses that the API Gateway 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 Gateway Data Store instance is running.
STOPPED	The API Gateway Data Store instance is stopped.

Lifecycle Actions for API Gateway Data Store

The following table lists the actions that API Gateway Data Store supports with the `sagcc exec lifecycle` command, along with the description of each action:

Action	Description
`start`	Starts the API Gateway Data Store instance.
`stop`	Stops the API Gateway Data Store instance.
`restart`	Restarts the API Gateway Data Store instance.

You can also perform these actions in the Command Central web user interface.

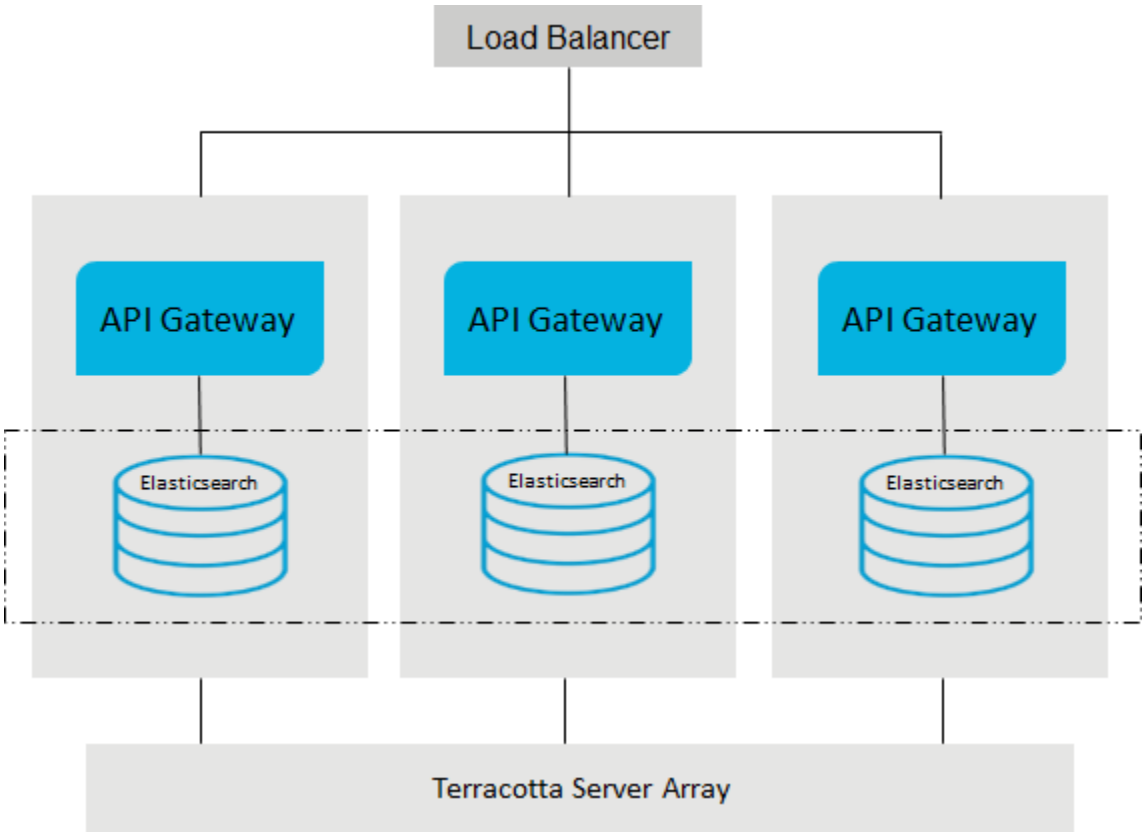
3 API Gateway Configuration

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 Gateway Data Store cluster
- Configuring Terracotta Server array
- Configuring load balancer
- Configuring ports

API Gateway Cluster Configuration

You can enable API Gateway clustering through the API Gateway user interface. For more information on enabling API Gateway clustering, see *webMethods API Gateway User's Guide*.

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

API Gateway Data Store Cluster Configuration

Each API Gateway cluster node consists of an API Gateway Data Store instance. For cluster configuration, the API Gateway Data Store instances should also be clustered using Elasticsearch clustering properties, by modifying the *SAG_root*/InternalDataStore/config/elasticsearch.yml file on each 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 will auto-bootstrap itself into a new cluster. You cannot change the configuration after bootstrap and thus, Elasticsearch will 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/7.7/index.html.

Configuring Elasticsearch Cluster

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

Note:
It is recommended that you learn the Elasticsearch best practices before proceeding on to the configuration.

To configure an Elasticsearch cluster

1. If you have started API Gateway before setting up the Elasticsearch cluster configuration, perform the following step before proceeding on to 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 the location where the Elasticsearch writes the snapshots to. Hence, it is important to have a location that is accessible to all the nodes.

`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.

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 will be like the example given here:

   ```
   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 Node 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. 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.

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 more information on API Gateway Availability, see *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`

  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`

  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.

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:
 ......
    uiConfig:
 ......
```

Elasticsearch Configuration

Note:

Install and run Elasticsearch, version 7.7.1, if you are configuring an external instance of 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 Gateway 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 external 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 Gateway data store. In this case, the hosts should point to API Gateway data store host and port. The default host for the API Gateway 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 7.7.1, 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

Note:
Install and run Terracotta server (a version that is compatible with API Gateway 10.7) for clustering API Gateway instances.

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

- `aware`, `name`, `tsaUrls`, `sessTimeout`, `actionOnStartupError`. All are Terracotta cluster settings that are set 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 would use this file for joining the Terracotta cluster.

A sample Cluster configuration is as follows:

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

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

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

API Gateway UI Configuration

The API Gateway UI configuration supports configuring the login page of API Gateway when the SSO configuration is enabled. The key configurations are as follows:

- `apigwAuthPriority`. Displays the login page based on the value.

- If the configuration is set as `apigwAuthPriority:form`, then API Gateway login page appears displaying the **Login with SSO option** option link.
- If the configuration is set as `apigwAuthPriority:"saml"`, then API Gateway redirects you to SSO login page.

A sample API Gateway UI configuration is as follows:

```
apigw:
  uiConfig:
    apigwAuthPriority: form
```

User Configuration

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

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

A sample user configuration is as follows:

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

```
      password: "oops"
      emailAddresses: [ bob@sag.com ]
      active: true
```

Master Password Configuration

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

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

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

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

A sample master password configuration is as follows:

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

UI Configuration

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

The key HTTP and HTTPS configurations are as follows:

Parameters	Description
`maxHttpHeaderSize` This parameter is applicable to HTTP and HTTPS.	Specifies the maximum size of the request and response HTTP header, specified in bytes. If not specified, this attribute is set to 8192 (8 KB).
`connectionTimeout` This parameter is applicable to HTTP.	This property specifies the time, in milliseconds, after which the connection times out.
`server` This parameter is applicable to HTTP and HTTPS.	This property overrides the server header for the HTTP response.

Parameters	Description
`maxSpareThreads` This parameter is applicable to HTTP and HTTPS.	Specifies the maximum number of request processing threads the connector creates, which determines the maximum number of simultaneous requests that API Gateway server can handle.
`disableUploadTimeout` This parameter is applicable to HTTP and HTTPS.	This flag allows the servlet container to use a different, usually longer connection timeout during data upload.
`minSpareThreads` This parameter is applicable to HTTP and HTTPS.	Specifies the minimum number of threads always kept running. This includes both active and idle threads.
`redirectPort` This parameter is applicable to HTTP and HTTPS.	When a SSL port is required by the client, the request is redirected to this port number.
`acceptCount` This parameter is applicable to HTTP and HTTPS.	Specifies the maximum queue length for incoming connection requests when all possible request processing threads are in use. Any requests received when the queue is full is refused. The default value is 100.
`port` This parameter is applicable to HTTP and HTTPS.	Specifies the TCP port number on which this connector creates a server socket and awaits incoming connections.
`enableLookups` This parameter is applicable to HTTP and HTTPS.	Set to `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` This parameter is applicable to HTTP and HTTPS.	Enable or disable API Gateway web-app port.
`alias` This parameter is applicable to HTTP and HTTPS.	Identifies a proxy server and a port on the server through which you want to route requests.
`sslProtocol` This parameter is applicable to HTTPS.	Specifies the SSL protocols to use. A single value may enable multiple protocols. For more information, see JVM documentation. If not specified, the default is `TLS`.
`sslEnabled`	Use this attribute to enable SSL traffic on a connector. The default value is `false`. To turn on SSL handshake or encryption or decryption

Parameters	Description
This parameter is applicable to HTTPS.	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` 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` 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` This parameter is applicable to HTTPS.	The password used to access the specified keystore file.
`scheme` 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` 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: "R3Vlc3NJdA=="
        domain: "EUR"
```

`isConfigurationAlias` configuration:

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

`authServerAlias` configuration:

```
apigw:
  aliases:
   testAuthServer:
```

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

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. ■ `connectionTimeout`. Time interval (in seconds) after which a connection attempt times out. ■ `readTimeout`. Time interval (in seconds) after which a socket read attempt times out. ■ `suspendDurationOnFailure`. Time to suspend the request upon a failure. ■ `optimizationTechnique`. Type of optimization technique used for SOAP messages. ■ `passSecurityHeaders`. Boolean value whether to pass security headers or not. ■ `keystoreAlias`. Keystore alias name that is used for the signing or encryption.

Type	Parameters
	■ `keyAlias`. Key alias in the particular keyStore . ■ `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.
	  ■ The authentication types supported are: `HTTP_BASIC`, `NTLM`, `OAUTH2`, `KERBEROS`, `JWT`, `ALIAS`, and `REMOVE_INCOMING_HTTP_HEADERS`.
	■ `authMode`. Authentication mode to use while communicating with the native API.
	  ■ 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`.
	■ `httpAuthCredentials`. Credentials to use for HTTP authentication. The authentication credentials supported are:
	  ■ `userName`. Specify a username to access the native API.
	  ■ `password`. Specify a password to access the native API.
	  ■ `domain`. Specify a domain to access the native API.
	■ `kerberosCredentials`. Credentials to use for Kerberos authentication. The authentication credentials supported are:
	  ■ `clientPrincipal`. A unique identity to which Kerberos can assign tickets.
	  ■ `clientPassword`. Password for the client principal.
	  ■ `servicePrincipal`. A unique identifier of a service instance.
	  ■ `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`.

Type	Parameters
	■ `requestDelegateToken`. Boolean value whether the token needs to be delegated or not. ■ `oauth2Token`. OAuth2 token to use for authentication.
`transformationAlias`	■ `fileName`. Name of the file. ■ `content`. Content of the file.
`serviceRegistryAlias`	■ `endpointURI`. Endpoint to use to communicate with the service registry. ■ `heartBeatInterval`. API Gateway pings the service registry on the configured interval for every API. ■ `username`. Username to use in the basic authentication when communicating with the service registry. ■ `password`. Password to use in the basic authentication when communicating with the service registry. ■ `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. ■ `keyAlias`. The key alias is the private key to use for signing when using SSL communication with the service registry. ■ `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. ■ `customHeaders`. Custom headers to send while communicating with the service registry. ■ `discoveryInfo`. Contains information like resource path and HTTP method to use while discovering a service in service registry. ■ `registrationInfo`. Contains information like resource path and HTTP method to use while registering a service in service registry. ■ `deRegistrationInfo`. Contains information like resource path and HTTP method to use while de-registering a service from service registry. ■ `serviceRegistryType`. Contains the information about the type of service registry.

Type	Parameters
	■ `connectionTimeout`. The time interval (in seconds) after which a connection attempt times out while communicating with service registry. ■ `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. ■ `accessKey`. The access key ID for the AWS instance. This is used to sign the requests. ■ `secretKey`. The secret access key for the AWS instance. This is used to sign the requests.
`samlIssuerAlias`	■ `issuerCommunicationMode`. Mode of communication to the STS. ■ `issuerPolicy`. The webMethods Integration Server service name. ■ `issuerAuthScheme`. The authentication type to use for communicating to STS. ■ `issuerAuthMode`. Mode of communication to STS. ■ `wssCredentials`. Credentials for the WSS Username token. ■ `kerberosCredentials`. Credentials for the Kerberos token. ■ `endpoint`. The endpoint URI of the STS. ■ `samlVersion`. SAML version used for authentication. ■ `wsTrustVersion`. WS-Trust version that API Gateway must use to send the RST to the SAML issuer. ■ `appliesTo`. Specify the scope for which this security token is required. ■ `extendedParameters`. Extensions to the `wst:RequestSecurityToken` element for requesting specific types of keys, algorithms, or key and algorithms, as specified by a given policy in the return tokens. ■ `signAndEncryptConfig`. Private and public keys to use signature and encryption.

Type	Parameters
`webmethodsAlias`	■ `serviceName`. The webMethods Integration Server service name. ■ `runAsUser`. The user name you want API Gateway to use to invoke the IS service. ■ `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. ■ `implNames`. Map of specification names to the implementation names of the service provider. ■ `extendedValuesV2`. List of headers that needs to be sent along with the client management request. ■ `generateCredentials`. Specifies whether API Gateway should generate `clientId` and `client secret`. ■ `supportedApplicationTypes`. List of `application_type` values supported by the authorization server provider.
`soapMessageSecurityAlias`	■ `authType`. Type of authentication. ■ `authMode`. Mode of authentication ■ `wssCredentials`. Credentials required for the WSS Username token. ■ `kerberosCredentials`. Credentials for the Kerberos token. ■ `samlIssuerConfig`. SAML issuer configuration name. ■ `signAndEncryptConfig`. Private and public keys to use for signature and encryption.
`isConfigurationAlias`	■ `url`. URL of the Integration Server. ■ `username`. User credentials required to access the Integration Server instance. ■ `password`. Password required to access the Integration Server instance. ■ `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.

Type	Parameters
	■ `keyAlias`. The alias for a specific key in the specified keystore. ■ `packageName`. Default package name where the alias is published. ■ `folderName`. Default folder name where the alias is published.
`authServerAlias`	■ `description`. Description of the auth server. ■ `tokenGeneratorConfig`. Specifies the token information that would be added as a bearer token in the HTTP request for client authentication. ■ `supportedGrantTypes`. Specifies the list of grant types that are supported by API Gateway. The grant types supported are: ■ `authorization_code`, `client_credentials`, and `implicit`. ■ `localIntrospectionConfig`. Specifies the introspection endpoint to check that access tokens used in client requests are currently active and are valid to invoke the protected resources. ■ `sslConfig`. Specifies the SSL configuration information.

Consolidating Externalized Configuration Files

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

A sample consolidated configuration file is as follows:

```
apigw:
 elasticsearch:
   tenantId: "apigateway"
   hosts: "localhost:9240"
   autostart: "true"
   http:
     username: ""
     password: "@secure.elasticsearch.http.password"
     keepAlive: "true"
     keepAliveMaxConnections: 10
     keepAliveMaxConnectionsPerRoute: 100
     connectionTimeout: 1000
     socketTimeout: 10000
     maxRetryTimeout: 100000
   https:
     enabled: "false"
```

```
    truststoreFilepath: "C:/softwares/elasticsearch-version/config/truststore-new.ks"

     keystoreAlias: "root-ca"
     truststorePassword: "@secure.elasticsearch.http.truststore.password"
     enforceHostnameVerification: "false"
   sniff:
     enable: "false"
     timeInterval: 1000
   outboundProxy:
     enabled: "false"
     alias: "esoutboundproxyalias"
   clientHttpResponseSize: 1001231
 kibana:
   dashboardInstance: "http://localhost:9405"
   autostart: "true"
  elasticsearch:
   sslCA: "C:/softwares/elasticsearch-version/config/SAG-B1HPWT2.pem"
  filebeat:
   output:
    elasticsearch:
     sslCA: "C:/softwares/elasticsearch-version/config/SAG-B1HPWT2.pem"
 cluster:
  aware: "true"
  name: "APIGatewayTSAcluster"
  tsaUrls: "VMYAI105BVT06:9510"
  terracottaLicenseFileName: "terracotta-license.key"
  sessTimeout: "20"
  actionOnStartupError: "standalone"
 uiConfig:
  apigwAuthPriority: form
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: "R3Vlc3NJdA=="
             domain: "EUR"
   myIsAlias:
          type: "isConfigurationAlias"
          url: "http://localhost:5555"
          username: "Administrator"
          password: "bXlwYXNz"
          keystoreAlias: "DEFAULT_IS_KEYSTORE"
          keyAlias: "ssos"
          packageName: "WmAPIGateway"
          folderName: "test2"
          importSwaggerBasedOnTags: "false"
```

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

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

```
apigw.elasticsearch.tenantId=apigateway
apigw.elasticsearch.autostart=true
apigw.elasticsearch.hosts=localhost:9240
apigw.elasticsearch.clientHttpResponseSize=1001231
apigw.elasticsearch.http.keepAlive=true
.
.
.
apigw.kibana.dashboardInstance=http://localhost:9405
apigw.kibana.elasticsearch.sslCert=/path/to/your/client.crt
apigw.kibana.elasticsearch.sslKey=/path/to/your/client.key
apigw.kibana.elasticsearch.sslCA=C:/softwares/elasticsearch-version/config/SAG-B1HPWT2.pem
.
.
.
.
apigw.filebeat.output.elasticsearch.sslCert=/path/to/your/client.crt
apigw.filebeat.output.elasticsearch.sslKey=/path/to/your/client.key
apigw.filebeat.output.elasticsearch.sslCA=C:/softwares/elasticsearch-version/config/SAG-B1HPWT2.pem
.
.
.
apigw.cluster.tsaUrls=VMYAI105BVT06:9510
```

```
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. ■ `YAML`. A YAML configuration file. ■ `PROPERTIES`. A properties configuration file. ■ `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.

Property	Detail
	Valid values are `true` and `false`. ■ If the value is set to `true`, it hides the clear text passwords. ■ 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. ■ `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.` **Important:** 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.

Setting	Password manager key replacement

`apigw:`
`  elasticsearch:`
`  http:`
`  username: elastic`
	@secure.elasticsearch.http.password

`apigw:`	@secure.elasticsearch.http.keystore.password

Setting	Password manager key replacement

`elasticsearch:`
`https:`
`keystorePassword: 6572b9b06156a0ff778c`	

`apigw:`
`elasticsearch:`
`https:`
`truststorePassword: 6572b9b06156a0ff778c`
	@secure.elasticsearch.http.truststore.password

Properties File Support for Externalized Configurations

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

A sample Properties file is as follows:

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

Environment Variables Support for Externalized configurations

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

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

A sample externalized configuration with environment variable is as follows:

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

Configuring Multiple Configuration Files and Its Effects

The master configuration file can have many entries (0 to N) for defining multiple configuration files as configuration sources. When such a file is used to start API Gateway, the configuration values from all the files would be merged into a single effective configuration. If the same

configuration value is present in two files, then the value in the file which has a higher preference is given priority. The order of preference is in the reverse order in which they are defined in the master configuration file, that is, the configuration values that are defined in the last configuration file entry would have the highest preference. A sample use case is explained below.

Assume `file1.yml` has the following configurations.

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

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

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

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

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

Then the combined configuration that becomes effective is as follows.

Effective `config.yml` configuration:

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

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.

Troubleshoot Tips for Externalizing Configurations

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 93.

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 64.

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 configurations 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 Gateway 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 the official Elasticsearch documentation (https://www.elastic.co/).

To connect to an external Elasticsearch

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

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

2. Configure the following properties:

Property and Description
pg.gateway.elasticsearch.autostart This property specifies whether the Elasticsearch starts automatically. If an external Elasticsearch is configured it has to be manually started. This property needs to be set to false to avoid API Gateway Data Store starting automatically. Default value: true
pg.gateway.elasticsearch.client.http.response.size This property specifies the response size, in MB, for API Gateway Elasticsearch client. Default value: 100
pg.gateway.elasticsearch.config.location This property specifies the location of the config file if you want to read port details from some other Elasticsearch config file
pg.gateway.elasticsearch.hosts *Mandatory* This property lists Elasticsearch hosts and ports. The values are comma separated. Default value: localhost:9240 **Note:** Once a host is added to this property, this is the value that is used to connect to Elasticsearch and the host configured in gateway-es-store.xml is not considered.
pg.gateway.elasticsearch.http.keepAlive *Mandatory* This property creates the persistent connection between client and server. Default value: true
pg.gateway.elasticsearch.http.connectionTimeout *Mandatory* This property specifies the time, in milliseconds, after which the connection times out. Default value: 10000
pg.gateway.elasticsearch.http.socketTimeout

Property and Description
Mandatory This property specifies the wait time, in milliseconds, for a reply once the connection to Elasticsearch is established after which it times out. Default value: 30000
pg.gateway.elasticsearch.http.maxRetryTimeout *Mandatory* This property specifies the wait time, in milliseconds, for retries after which it times out. Default value: 100000 It is advisable to set max retry time for a request to (number of nodes * socketTimeOut)+connectionTimeout
pg.gateway.elasticsearch.http.keepAlive.maxConnections *Mandatory* This property specifies the maximum number of persistent connections that can be established between an API Gateway and Elasticsearch cluster. Default value: 50
pg.gateway.elasticsearch.http.keepAlive.maxConnectionsPerRoute *Mandatory* This property specifies the maximum number of persistent connections that can be established per HTTP route to an Elasticsearch server. Default value: 15
pg.gateway.elasticsearch.http.username This property specifies the user name to connect to Elasticsearch using basic authentication.
pg.gateway.elasticsearch.http.password This property specifies the password to connect to Elasticsearch using basic authentication.
pg.gateway.elasticsearch.https.keystore.filepath This property specifies the Keystore file path for establishing HTTPS communication with Elasticsearch.
pg.gateway.elasticsearch.https.truststore.filepath This property specifies the truststore file path for establishing HTTPS communication with Elasticsearch.

Property and Description
pg.gateway.elasticsearch.https.keystore.password This property specifies the Keystore password for establishing HTTPS communication with Elasticsearch.
pg.gateway.elasticsearch.https.keystore.alias This property specifies the Keystore alias for establishing HTTPS communication with Elasticsearch.
pg.gateway.elasticsearch.https.truststore.password This property specifies the truststore password for establishing HTTPS communication with Elasticsearch.
pg.gateway.elasticsearch.https.enabled This property specifies whether you want to enable or disable the HTTPS communication with Elasticsearch. Default value: false If this property is set to false none of the above properties related to HTTPS are respected.
pg.gateway.elasticsearch.outbound.proxy.enabled This property specifies whether you want to enable or disable outbound proxy communication. Default value: true
pg.gateway.elasticsearch.outbound.proxy.alias This property specifies the outbound proxy alias name used to connect to Elasticsearch.
pg.gateway.elasticsearch.https.enforce.hostname.verification This property enforces the host name verification for SSL communication. Default value: false
pg.gateway.elasticsearch.sniff.enable *Mandatory* This property enables sniffers to add the other nodes in an Elasticsearch cluster to the client so that the client can talk to all nodes. Default value: true This configuration must be set to *false* if you are changing the network when API Gateway or Elasticsearch is running.
pg.gateway.elasticsearch.tenantId

Property and Description
This property allows you to specify a tenant name of your choice. This value must be same across all nodes. The default value of this property is the Integration Server instance name. So, ensure that you provide same name for all Integration Server nodes in a cluster. If you modify this value, you must edit the value in all nodes and restart the API Gateway server for the change to take effect.
pg.gateway.elasticsearch.sniff.timeInterval *Mandatory* This property enables adding the newly added Elasticsearch cluster nodes to existing REST client in a specified time interval in milliseconds. Default value: 60000

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

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

You can also externalize the Elasticsearch tenant ID and configuration by using a master configuration file. For details, see "Externalizing Configurations " on page 63.

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), and 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 (Core data level)	Compatible Elasticsearch versions (Transaction data level)	Compatible Kibana version
10.7	7.7.1	All Elasticsearch versions	7.7.1
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 Gateway ships the OSS versions of Elasticsearch, Kibana and Filebeat; and only the OSS versions of Kibana and Filebeat are compatible with the OSS version of Elasticsearch.

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 64. 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 Gateway Data Store.
- External Kibana connected to API Gateway Data Store.

  You can configure this setup as follows:

  For an external Kibana to connect to API Gateway 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.

Property	Description
	Example: `9405`
`server.host:` *server host IP address or host name*	Specifies the host to bind the server to. 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. 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. The server path you specify must not end with a /. Value: `"/apigatewayui/dashboardproxy"`
`elasticsearch.hosts:` http://*hostname:port*	Specifies the URLs of the Elasticsearch instance to use for all your queries. 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. 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 Gateway 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 64.

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
```

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. The default value is 256 MB.
`wrapper.java.maxmemory`	The maximum heap size. The default value is 1024 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=1024
   ```

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 Gateway 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 Gateway 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

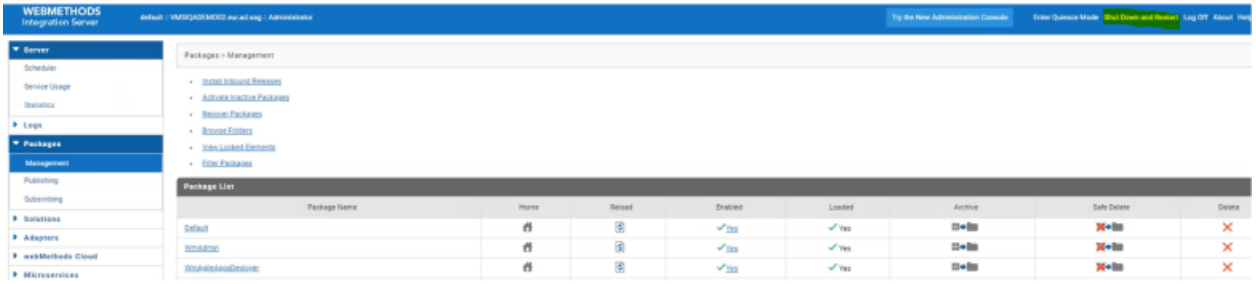
1. Open a browser and type `localhost:5555`.

   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.



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.

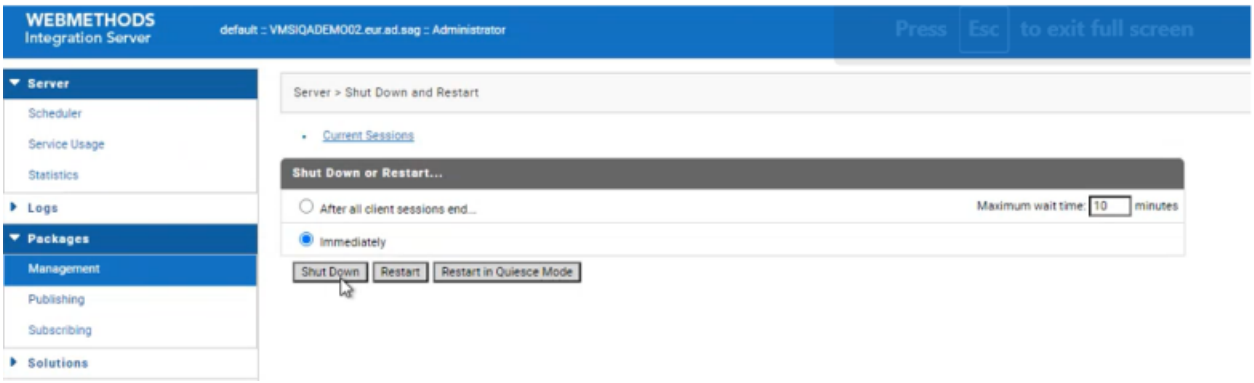
   Important:
   You must use the **Immediately** option only if your API Gateway has a clustered configuration. With clustered configuration, all the active sessions are transferred to another API Gateway node. If you select the **Immediately** option with a clustered configuration, all your active sessions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


WEBMETHODS
Integration Server
default :: VMSIQADEMO02.eur.ad.sag :: Administrator
Press Esc to exit full screen
Server
Scheduler
Service Usage
Statistics
Logs
Packages
Management
Publishing
Subscribing
Solutions
Server > Shut Down and Restart
Current Sessions
Shut Down or Restart...
After all client sessions end...
Maximum wait time: 10 minutes
Immediately
Shut Down
Restart
Restart in Quiesce Mode

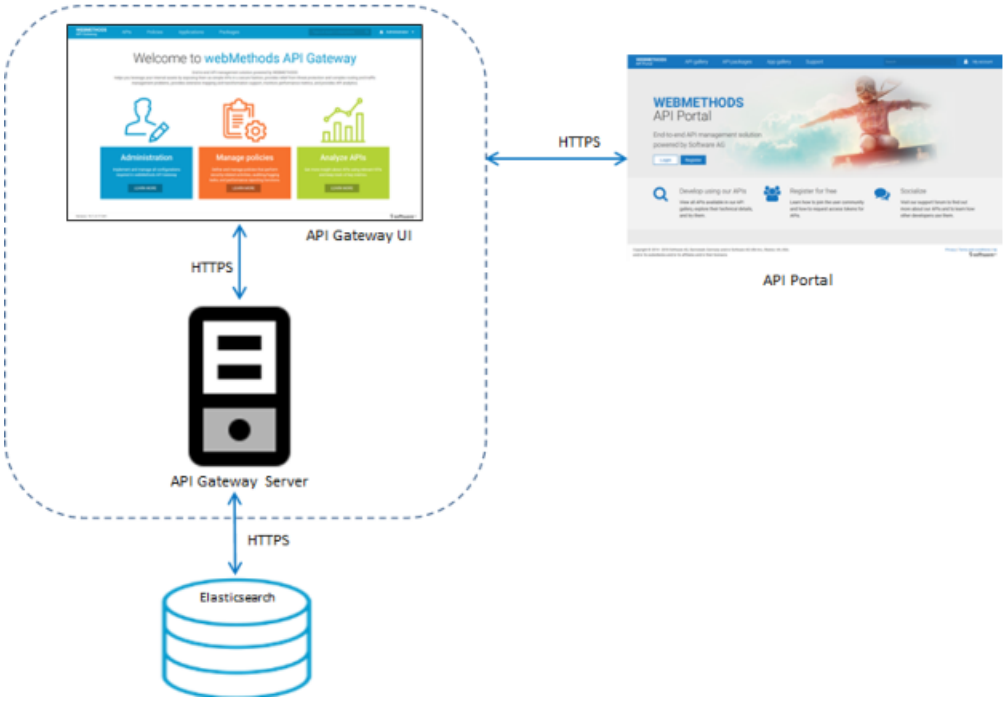
4 Securing API Gateway and its Components

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 Gateway 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 103
- API Gateway UI and Users. For details, see " How do I Secure API Gateway User Interface Communication? " on page 112
- API Gateway and API Portal. For details, see " How do I Configure a Secure Communication Channel between API Gateway and API Portal?" on page 114
- API Gateway and API Data Store. For details, see "How do I Secure API Data Store Communication using HTTPS?" on page 115

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 136 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.

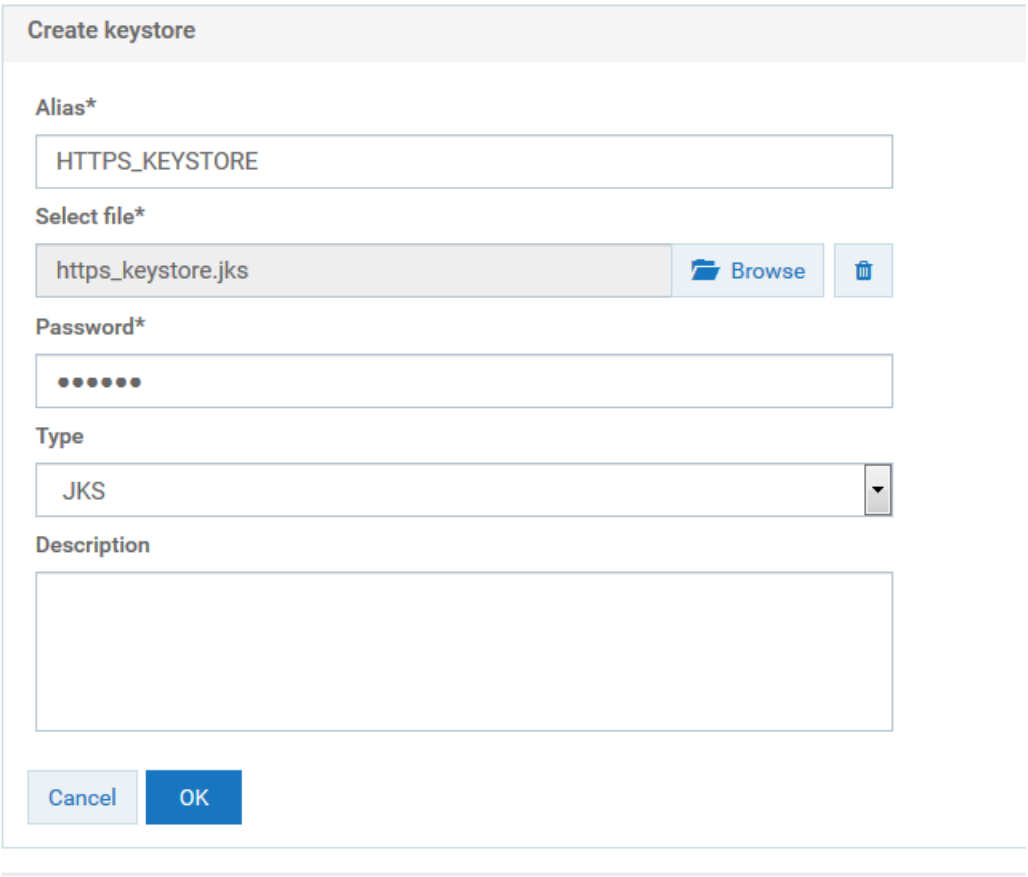
   a. Log on to API Gateway.

   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.



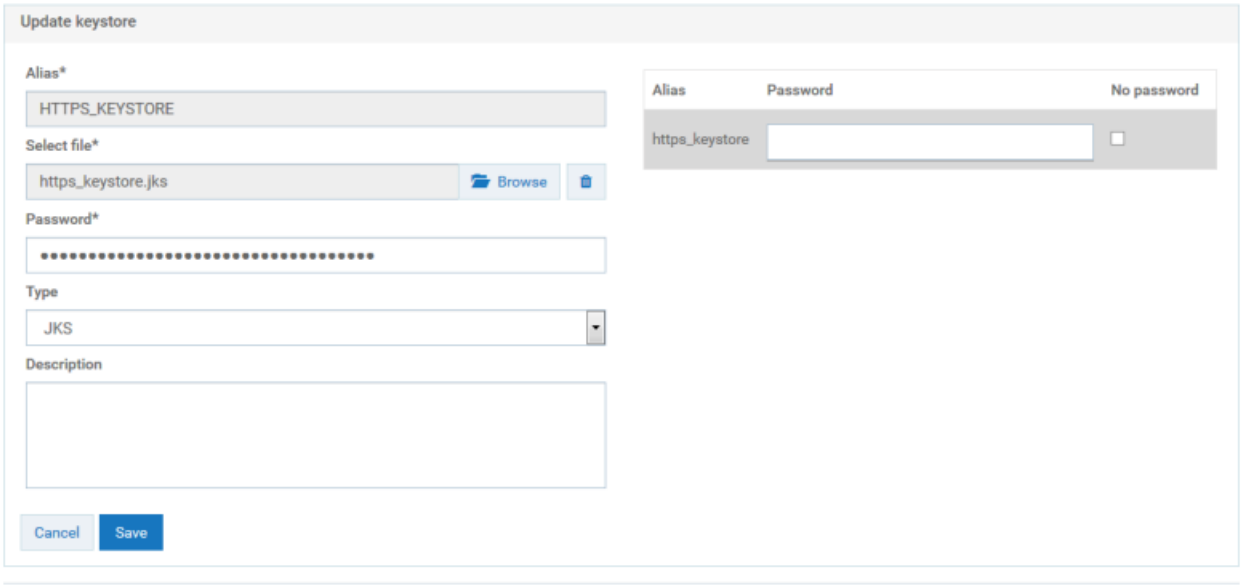
e. Click **OK**.

   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`.

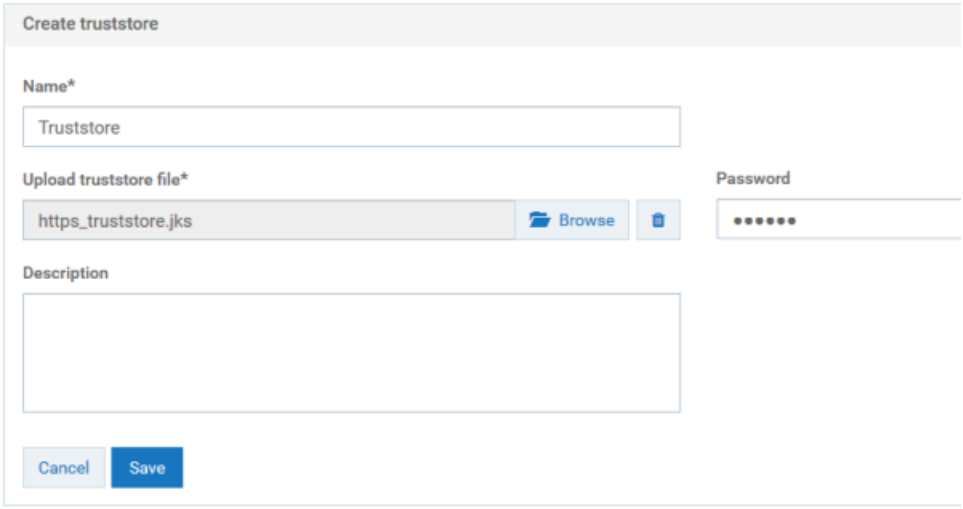


h. Click **Save**.

i. Click **Add truststore**.

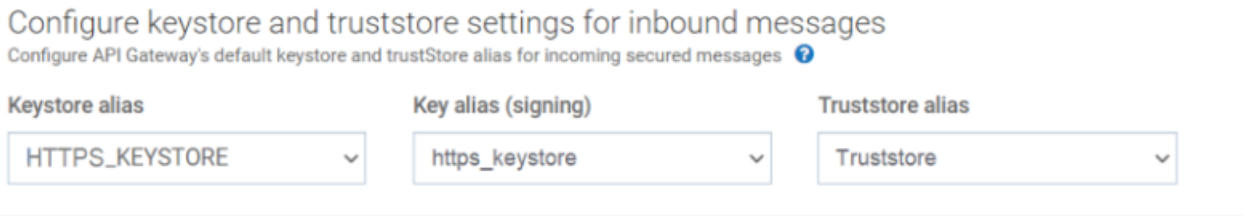
j. Provide the following details.

- **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.



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.



m. Click **Save**.

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

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

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

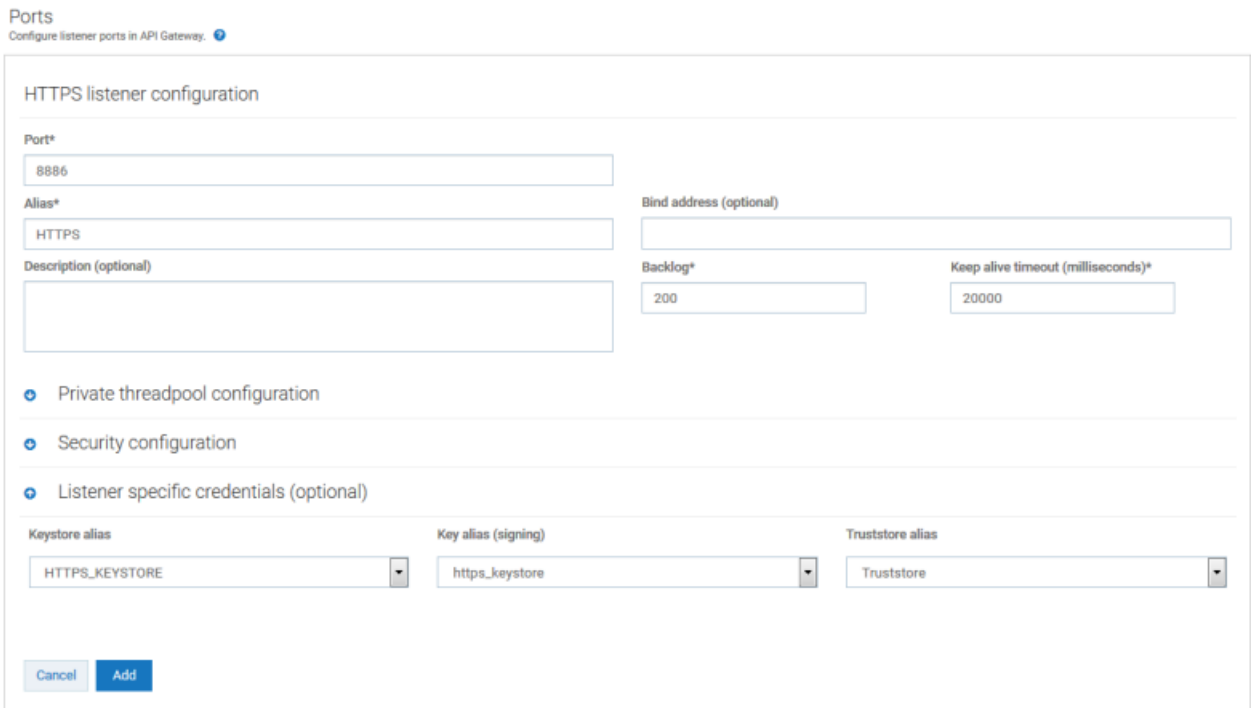
c. Click **Add**.

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.

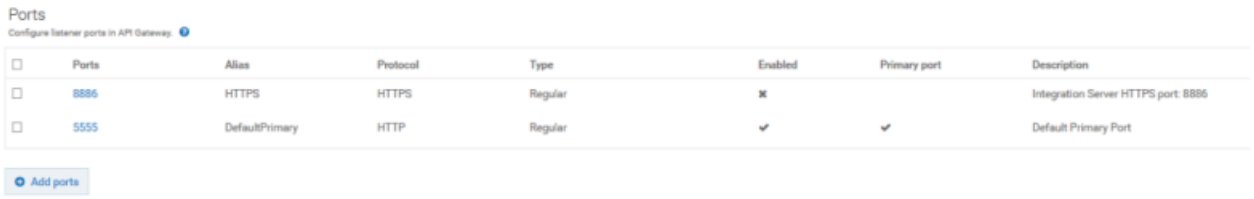
e. In the Listener-specific credentials section provide the following information:

- **Keystore alias**. Select HTTPS_KEYSTORE.
- **Key alias(signing)**. Select https_keystore.
- **Truststore alias**. Select Truststore.



f. Click **Add**.

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



g. Enable the new port 8886 by clicking the X mark in the port's **Enabled** column.

The port 8886 is now enabled and API Gateway server is now ready to accept requests over HTTPS port 8886.

4. Setup security configuration parameters for the HTTPS port, which is enabled for communication with API Clients, to determine how API Gateway server interacts with the clients and defines whether the connection is one-way or two-way SSL.

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

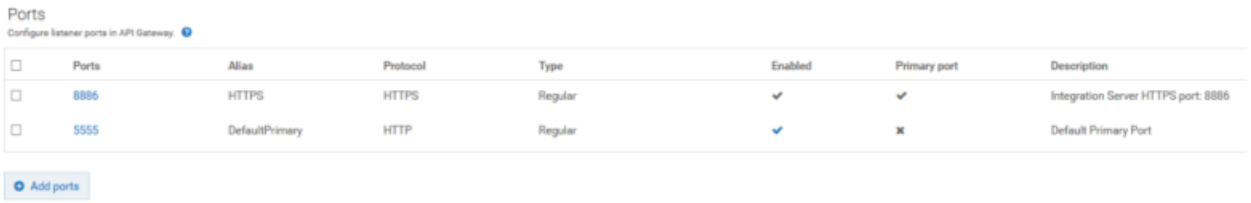
on file and is signed by a trusted authority. If so, the client is logged in as the user to which the certificate is mapped in API Gateway. If not, the client request fails, unless central user management is configured.

- **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.



   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
```

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 136 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.

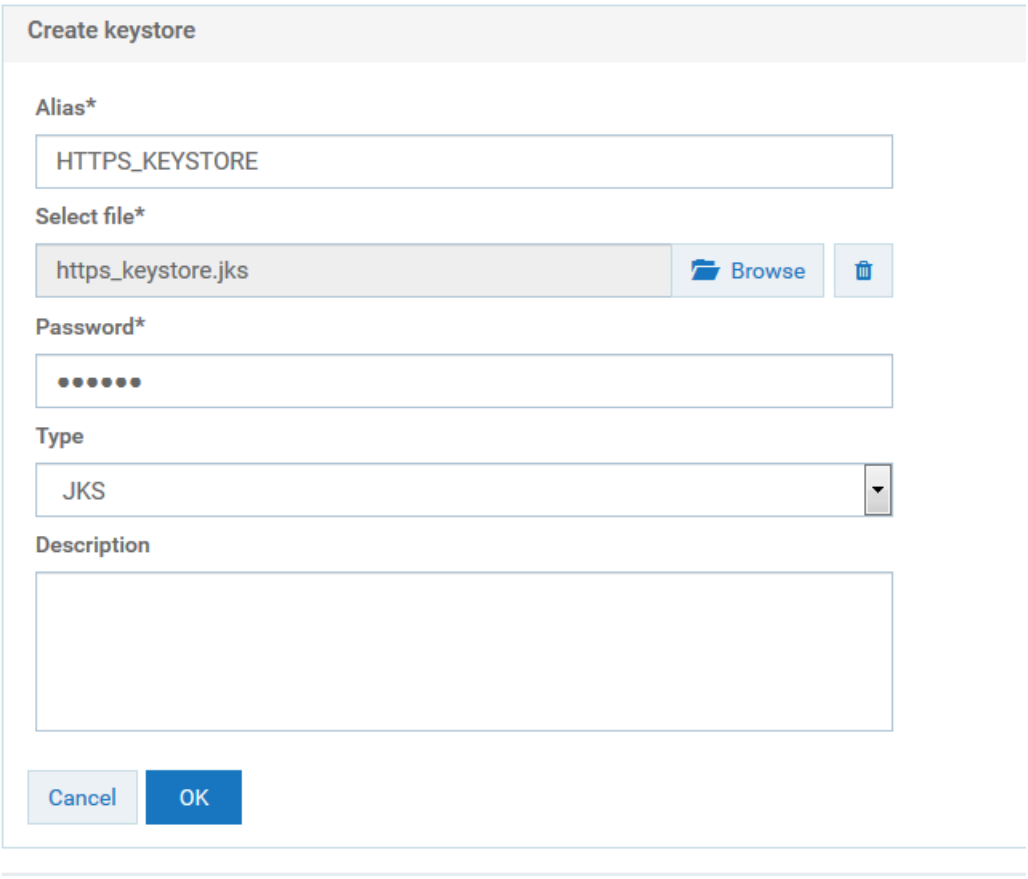
   a. Log on to API Gateway.

   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.



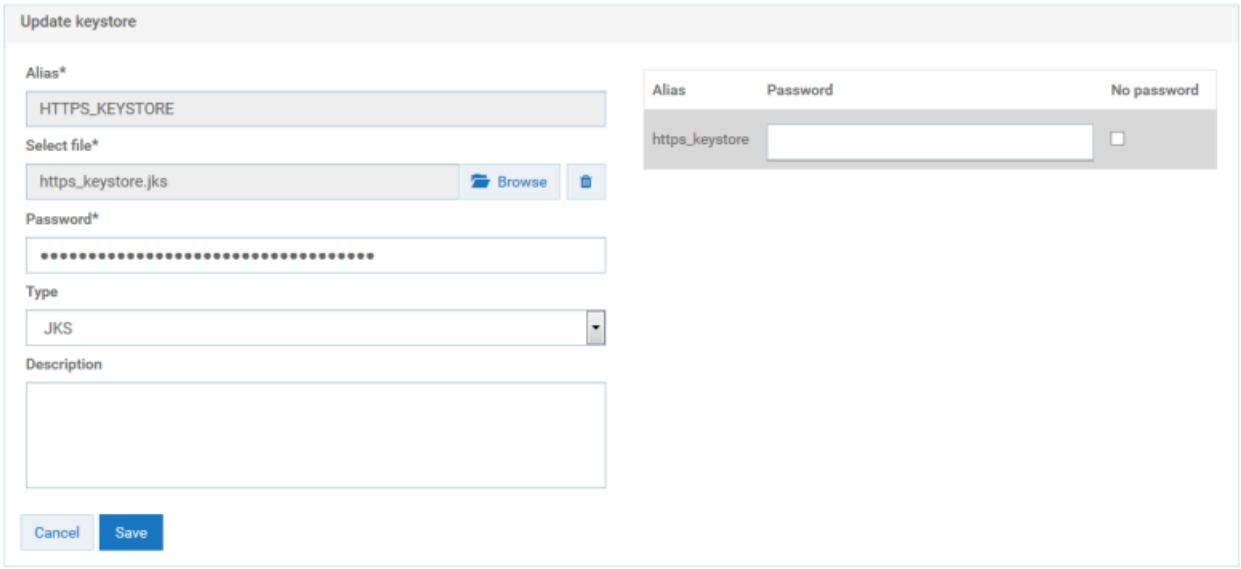
e. Click **OK**.

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`.

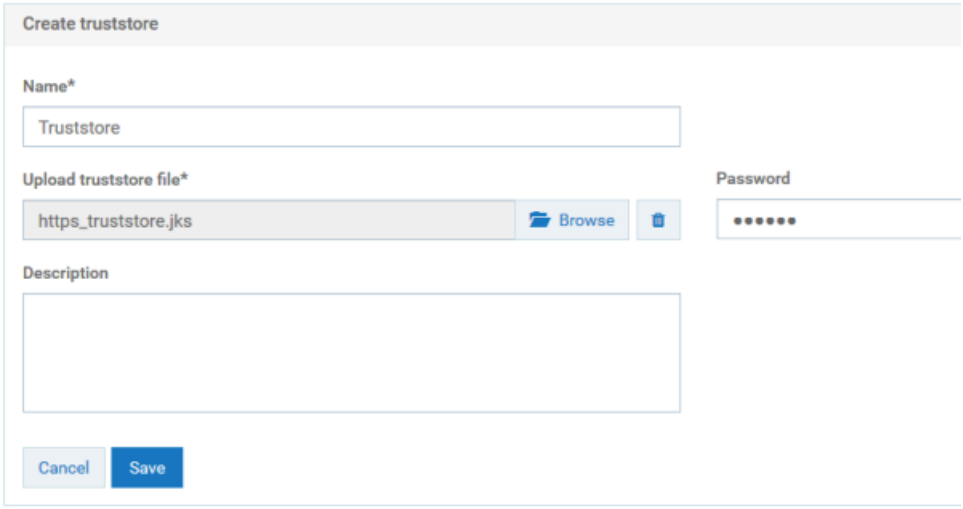


h. Click **Save**.

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.



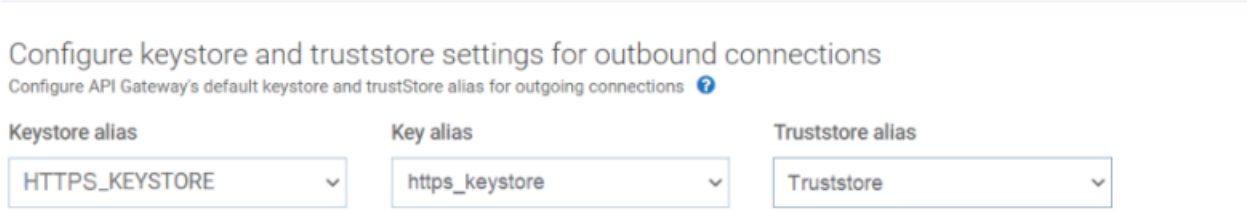
k. Click **Save**.

3. To communicate securely with the backend services you have to configure the keystore and truststore settings for outbound connections. This can be configured in one of the following ways:

   - Globally, you can configure the keystore and truststore settings for outbound connections in **Administration** > **Security configuration** section as follows:

     1. Navigate to **Administration > Security > Keystore/Truststore**.

     2. In the Configure keystore and truststore settings for outbound connections section, provide the keystore and truststore aliases for securing outgoing SSL connections. The keystore and key alias are required for outgoing two-way SSL connections.



   - At an API-level, you can configure the keystore and truststore in the following ways:

     - Through an endpoint alias configured in the routing policy:

       1. Create an endpoint alias where you specify the default URI, and the keystore and truststore for the backend service. For details about creating an endpoint alias, see Aliases section in the *webMethods API Gateway User's Guide*.

       2. Specify the endpoint alias in the **Endpoint URI** field in the routing policy properties section when you configure the policy. For details, see Routing Policies section in the *webMethods API Gateway User's Guide*.

     - Through a routing policy by specifying the URI of the backend service endpoint, and the keystore and truststore. For details, see Routing Policies section in the *webMethods API Gateway User's Guide*.

   Note:
   The global keystore and truststore configuration is the default configuration that applies for all APIs if there is no keystore or truststore configured through an endpoint alias or a routing policy at an API-level.

You now have a secure communication channel established between the API Gateway server and the backend services.

How do I Secure API Gateway User Interface Communication?

Secure API Gateway UI (web application), one of the API Gateway components in an API Management setup, to enable users to access the API Gateway UI securely over HTTPS. This section explains how to secure API Gateway communication using HTTPS protocol.

You must have API Gateway administrator privileges to perform this operation. Also, ensure that the required client and server certificates are available.

To configure API Gateway user interface for secure communication

1. Locate the keystore and truststore files in the file system.

   The default keystore and truststore files are available in the *Installation_Dir*\common\conf folder.

   Note:
   If you want to use a custom keystore with self-signed certificates, see "Creating a Custom Keystore with Self-Signed Certificates" on page 136 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.

To Harden TLS configuration of the API Gateway UI port

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.

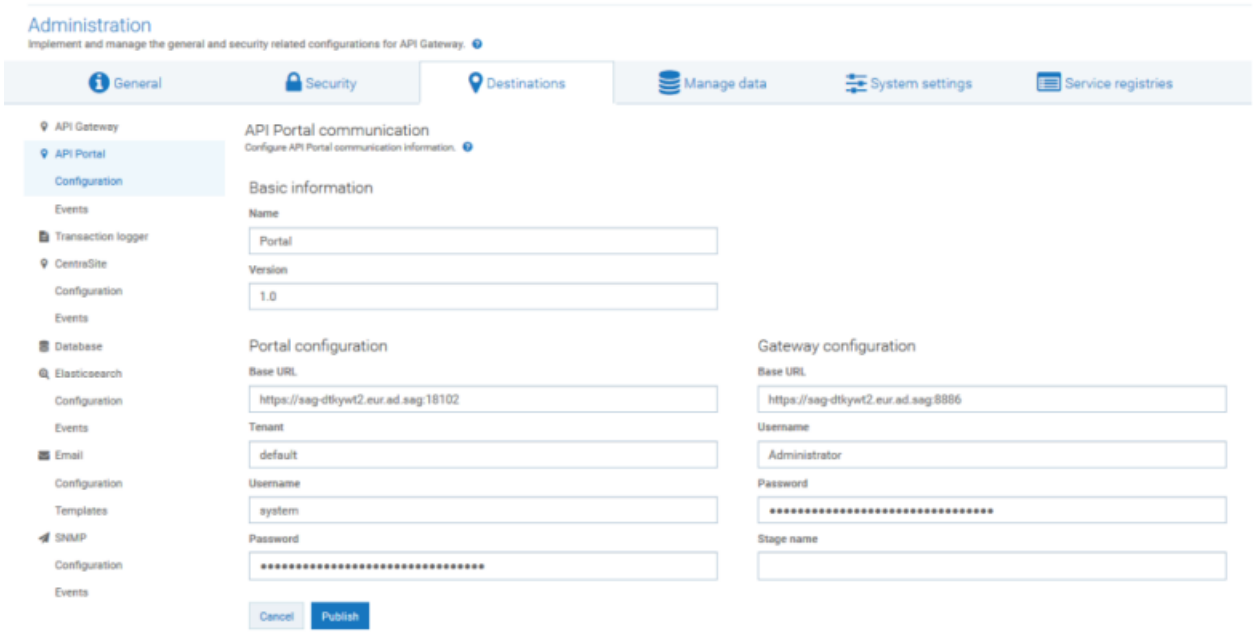
How do I Configure a Secure Communication Channel between API Gateway and API Portal?

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:



   - 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.

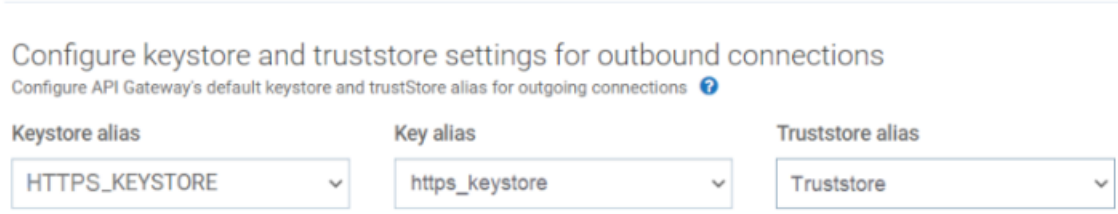
- In the Gateway configuration section, provide the following details:
  - **Base URL**. The API Gateway server URL, which API Portal uses to communicate to API Gateway using the HTTPS port. Specify the port 8886 that is configured for HTTPS communication.
  - **Username** and **Password** credentials 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.



3. You have to configure the API Portal truststore to trust the API Gateway outbound certificate. For details about how to configure API Portal truststore, see API Portal documentation.

You now have a secure communication channel between API Gateway and API Portal. You can now publish an API, which is enforced with Enable HTTPS/HTTPS policy with the HTTPS option configured, from API Gateway to API Portal and invoke the API from API Portal using the HTTPS endpoint that has been used to publish it to API Portal.

How do I Secure API Data Store Communication using HTTPS?

You can secure API Data Store (a simple Elasticsearch instance), one of the components in an API Management setup, to communicate securely over HTTPS. To secure API Data Store, you can use the following security plugins:

- ReadonlyREST. For details, see "How do I Secure API Data Store Communication using HTTPS with ReadonlyREST Plugin?" on page 116.

- Search Guard. For details, see " How do I Secure API Data Store Communication using HTTPS with Search Guard Plugin? " on page 123.

Both the plugins offer encryption, authentication, and authorization to protect data from attackers and other misuses. The plugins secure API Data Store by exposing it over HTTPS, and enables basic authentication by configuring users.

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.

How do I Secure API Data Store Communication using HTTPS with ReadonlyREST Plugin?

You can use the ReadonlyREST plugin to secure API Data Store (a simple Elasticsearch instance), one of the components in an API Management setup, to communicate securely over HTTPS.

Before you begin

Ensure that you have:

- Downloaded the ReadonlyREST plugin. You can download the ReadonlyREST plugin compatible with Elasticsearch 7.7.1 from https://readonlyrest.com/download and store it in your file system.
- Installed the API Gateway edition of version 10.7
- Installed an updated version of Java in your system and the path of the environment variable is set.

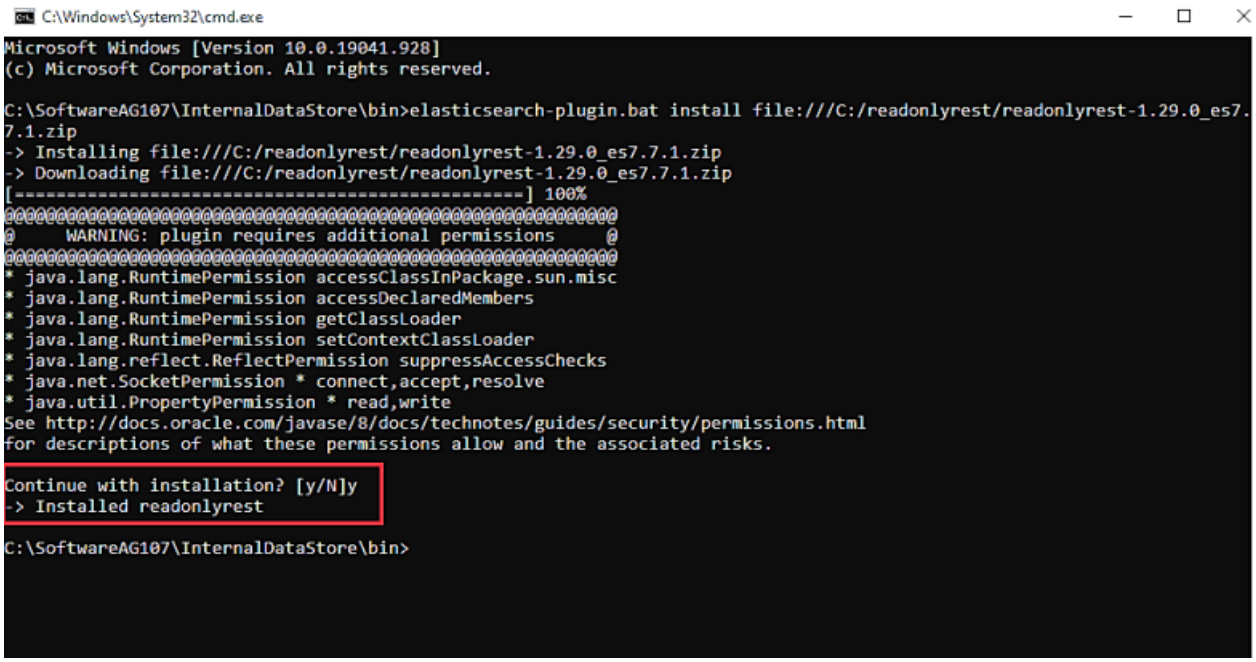
To secure API Data Store communication using HTTPS

1. Install and initialize the ReadonlyREST plugin.
   a. Shutdown API Gateway.
   b. Open the command prompt to the location `Installation_Dir/InternalDataStore/bin`
   c. Run the following command:

      ```
      elasticsearch-plugin.bat install
      ReadonlyREST_plugin_zip_file_location_in_the_file_system
      ```

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

      Installed readonlyrest message appears on successful installation.



e. Navigate to the *Installation_Dir*\InternalDataStore\config folder and create an empty file with the name, readonlyrest.yml (in the same folder where elasticsearch.yml is available).

f. Copy the folder **sagconfig** from *Installation_Dir*\IntegrationServer\instances*Instance_name*\packages\WmAPIGateway\config\resources\elasticsearch to *Installation_Dir*\InternalDataStore.

g. Copy the certificates **node-0-keystore.jks** and **truststore.jks** from *Installation_Dir*\InternalDataStore\sagconfig folder to *Installation_Dir*\InternalDataStore\config folder.

2. Protect API DataStore with two-way SSL and basic authorization.

The ReadonlyREST plugin supports plain-text credentials and hashed credentials for basic authorization. You can choose to add plain-text credentials or hashed credentials in the readonlyrest.yml file.

Plain-text credentials

Perform the following steps to include plain-text credentials for basic authorization in the readonlyrest.yml file.

a. Open the **readonlyrest.yml** file from the *Installation_Dir*\InternalDataStore\config folder and add the following text:

```
readonlyrest:
    access_control_rules:

    - name: "Require HTTP Basic Auth"
      type: allow
      auth_key: <plain_text_credentials> For example:
auth_key:Administrator:manage
```

```
 ssl:
   enable: true
   keystore_file: "node-0-keystore.jks"
   keystore_pass: a362fbcce236eb098973
   key_pass: a362fbcce236eb098973
   client_authentication: true
```

where access control rule name: *Require HTTP Basic Auth* is basic auth authentication, auth_key is the credentials (username and password) in the plain text.

Note:
Ensure that the content is indented properly as shown above, so that the YAML parser can parse them correctly.

Hashed credentials

The ReadonlyREST plugin supports obfuscating the credentials by hashing the credentials using *SHA256* algorithm. Use hashed credentials to keep the application secure.

Perform the following steps to include hashed credentials for basic authorization in the readonlyrest.yml file.

To generate the hash code for your password:

a. Use the tool https://xorbin.com/tools/sha256-hash-calculator to hash your credentials. This generates the hash code.

b. Replace the `auth_key` property in the **readonlyrest.yml** file with `auth_key_sha256` property.

c. Add the hashed credentials to the property `auth_key_sha256` in **readonlyrest.yml** file located at *`Installation_Dir`*`\InternalDataStore\config`.

```
readonlyrest:
    access_control_rules:

    - name: "Require HTTP Basic Auth"
      type: allow
      auth_key_sha256:
927d5619ff87227be6ca8a2cc9ee68c11dd7a08d64d1e20bdc8d86254850b418

    ssl:
      enable: true
      keystore_file: "node-0-keystore.jks"
      keystore_pass: a362fbcce236eb098973
      key_pass: a362fbcce236eb098973
      client_authentication: true
```

where access control rule name: *Require HTTP Basic Auth* is basic auth authentication, auth_key_sha256 is the credentials (username and password) in the hashed format and client_authentication is the two way SSL authentication. By default, this property is disabled. You can remove this property or set the value to `false` if you don't want the server or API Gateway Data Store to validate the client authentication.

d. Save the configuration file **readonlyrest.yml**.

Note:
The keystore file and its passwords keystore_pass & key_pass are shipped out with API Gateway product by default. This may not be safe for production environment. For production setup, you can generate your own certificates (keystore and trustore) and configure in the **readonlyrest.yml** file. To generate your own certificates, see "ReadonlyREST" on page 121.

b. After adding plain-text credentials or hashed credentials in readonlyrest.yml file, open the **elasticsearch.yml** file from the *Installation_Dir*\InternalDataStore\config folder and add the following text to the existing content and save the file.

```
http.type: ssl_netty4
```

This enables HTTPS connection for API Data Store.

3. Secure the inter-node communication.

In a clustered setup, to establish a secure communication between the API Gateway Data Store instances, perform the following steps:

a. Open the **readonlyrest.yml** file from the *Installation_Dir*\InternalDataStore\config folder and add the following text towards the end of the existing content and save the file.

```
  ssl_internode:
  keystore_file: "node-0-keystore.jks"
  keystore_pass: a362fbcce236eb098973
  key_pass: a362fbcce236eb098973
```

The consolidated content of the **readonlyrest.yml** is as follows:

```
    readonlyrest:
    access_control_rules:

    - name: "Require HTTP Basic Auth"
      type: allow
      auth_key_sha256:
927d5619ff87227be6ca8a2cc9ee68c11dd7a08d64d1e20bdc8d86254850b418

    ssl:
      enable: true
      keystore_file: "node-0-keystore.jks"
      keystore_pass: a362fbcce236eb098973
      key_pass: a362fbcce236eb098973

    ssl_internode:
      keystore_file: "node-0-keystore.jks"
      keystore_pass: a362fbcce236eb098973
      key_pass: a362fbcce236eb098973
      client_authentication: true
```

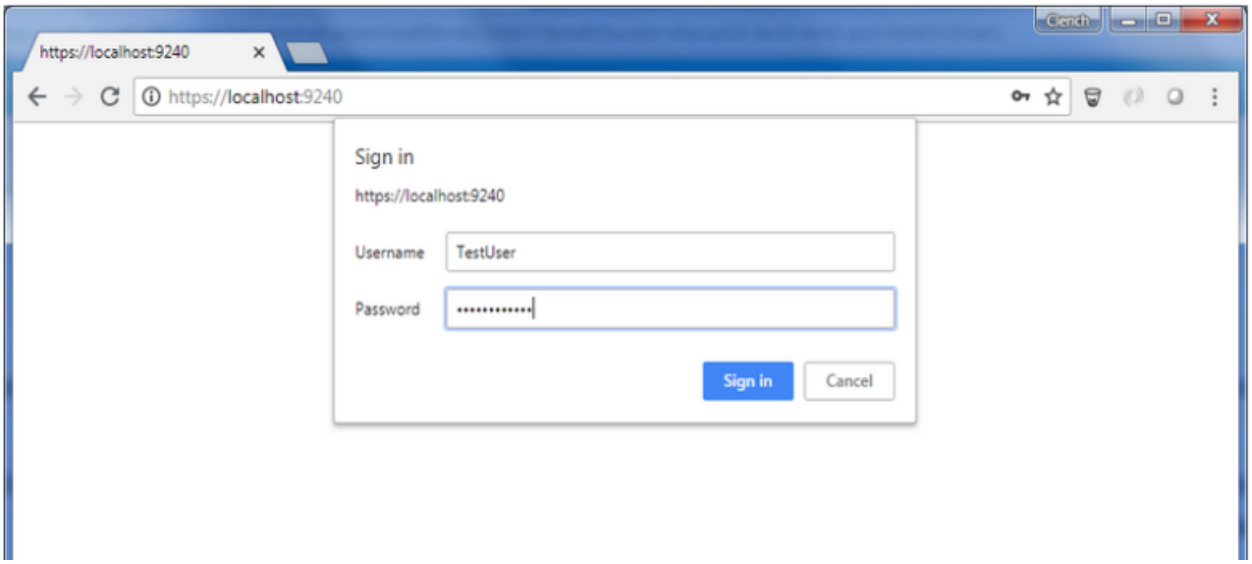
b. Open the **elasticsearch.yml** file from the *Installation_Dir*\InternalDataStore\config folder and add text as follows to the existing content and save the file.

```
transport.type: ror_ssl_internode
```

c. Shutdown and restart the API Data Store node to verify if it is protected by accessing the node in HTTPS URL with the given username and password.

```
https://<host>:9240
```

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



4. Change the Kibana configuration to connect to API Data Store.
   a. Open the **kibana.yml** file from the *Installation_Dir*\profiles\IS_*instance_name*\apigateway\dashboard\config folder.
   b. Remove the # symbol from the following properties and provide the corresponding values for the properties and save the file.
      - elasticsearch.username: `Administrator`
      - elasticsearch.password: `manage`
      - elasticsearch.ssl.verificationMode: `certificate`
      - elasticsearch.ssl.certificateAuthorities: *<file path of your root-ca.pem certificate>*
      - elasticsearch.hosts: https://*hostname*:`9240`

      Sample kibana.yml file is as follows:

```
server.port: 9405
server.host: "0.0.0.0"
server.basePath: "/apigatewayui/dashboardproxy"
elasticsearch.hosts: "https://localhost:9240"
kibana.index: "gateway_default_dashboard"
elasticsearch.username: "Administrator"
elasticsearch.password: "manage"
elasticsearch.ssl.verificationMode: certificate
elasticsearch.ssl.ca: C:/SoftwareAG107/InternalDataStore/sagconfig/root-ca.pem
elasticsearch.requestTimeout: 90000
pid.file: kibana.pid
console.enabled: false
```

c. Open the **uiconfiguration.properties** file from the *Installation_Dir*\profiles\IS_*instance_name*\apigateway\config folder and set the property **apigw.kibana.autostart** to false.

5. Change the API Gateway configurations to connect to API Data Store.

a. Open the **config.properties** file from the *Installation_Dir*\IntegrationServer\instances*instance_name*\packages\WmAPIGateway\config\resources\elasticsearch folder.

b. Remove the # symbol from the following properties and provide the corresponding values for the properties and save the file.

- pg.gateway.elasticsearch.http.username=Administrator
- pg.gateway.elasticsearch.http.password=manage
- pg.gateway.elasticsearch.https.truststore.filepath=Installation_Dir /InternalDataStore/sagconfig/truststore.jks
- pg.gateway.elasticsearch.https.truststore.password=2c0820e69e7dd5356576
- pg.gateway.elasticsearch.https.enabled=true

c. Start API Gateway Data Store.

d. When API Gateway Data Store is up and running, start the Kibana server 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 ReadonlyREST plugin with Self-generated certificates

As an API Provider, if you want to generate your own certificates to use with the ReadonlyREST plugin instead of the default certificates that are shipped with API Gateway, you can configure ReadonlyREST with user generated certificates. You have to perform this procedure if your organization does not have policies and procedures in place regarding the generation and use of digital certificates and certificate chains, including the use of certificates signed by a CA and you want to generate a self-signed certificate and import them into the keystore and truststore. Use **Java Keytool** to generate the required certificates for running ReadonlyREST in a production environment.

1. Shut down API Gateway, API Gateway DataStore, and Kibana.

2. Generate **keystore.jks** file.

a. Run the following command from <*Java_Install_Directory*>/bin to generate keystore certificate in the corresponding file location using Java Keytool:

```
keytool -genkey -alias alias_name -keyalg RSA -keystore
file_location\keystore.jks -storetype JKS
```

The keystore certificate is generated in the file location that you specify.

Example:

```
C:\Program Files\Java\jdk1.8.0_181\bin>keytool -genkey -alias test -keyalg RSA -keystore c:\work\apigateway\keystore.jks -storetype JKS
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:
What is the name of your organizational unit?
  [Unknown]:
What is the name of your organization?
  [Unknown]:
What is the name of your City or Locality?
  [Unknown]:
What is the name of your State or Province?
  [Unknown]:
What is the two-letter country code for this unit?
  [Unknown]:
Is CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown correct?
  [no]:  yes

Enter key password for <test>
       (RETURN if same as keystore password):
```

b. Replace the generated keystore.jks file in the *Installation_Dir*\InternalDataStore\config folder.

c. Update the properties **keystore_file**, **keystore_pass** and **key_pass** in the **readonlyrest.yml** file located at *Installation_Dir*\InternalDataStore\config.

3. Export the generated certificate from the keystore to configure in Kibana.yml file:

a. Run the following command to export the certificate from the keystore and place it in the required location. :

```
keytool -export -alias alias_name -keystore keystore_file_location -rfc -file
<filename>.cert
```

```
C:\Program Files\Java\jdk1.8.0_181\bin>keytool -export -alias test -keystore c:\work\apigateway\keystore.jks -rfc -file c:\work\apigateway\test.cert
Enter keystore password:
Certificate stored in file <c:\work\apigateway\test.cert>
```

The exported certificate is saved in the specified location.

b. After exporting, update the exported certificate name and the file location of the exported certificate in **elasticsearch.ssl.certificateAuthorities** property in **kibana.yml** file located at *Installation_Dir*\profiles\IS_default\apigateway\dashboard\config.

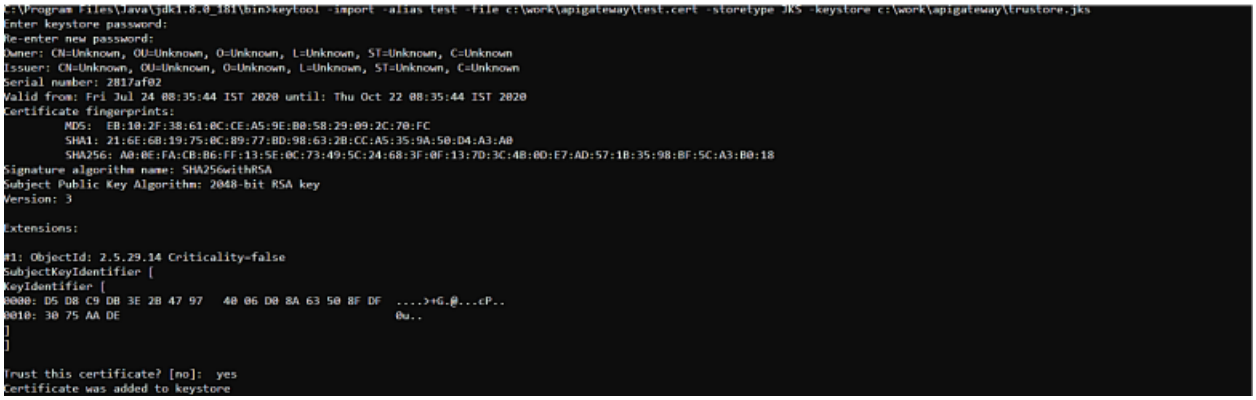
Example:

```
elasticsearch.ssl.certificateAuthorities: C:\work\apigateway\test.cert
```

4. Generate truststore and import the generated certificate.

a. Run the following command to generate the truststore file and import the generated certificate into the truststore file.

```
keytool -import -alias alias_name -file Certificate_file_location
 -storetype JKS -keystore file_location_and_file name_for_trustore.jks
```



The trust store certificate is stored in the specified location. For example: *C:\work\apigateway\trustore.jks.*

b. Update the details of the generated trustore file in the following properties

- `pg.gateway.elasticsearch.https.truststore.filepath`
- `pg.gateway.elasticsearch.https.truststore.password` in the **config.properties** file located at *`Installation_Dir`*`\IntegrationServer\instances\default\packages\WmAPIGateway\config\resources\elasticsearch`.

Example:

```
pg.gateway.elasticsearch.https.truststore.filepath=C:/work/apigateway/trustore.jks
pg.gateway.elasticsearch.https.truststore.password=your_trustore_password
```

c. Start API Data Store.

d. When API Data Store is up and running, start the Kibana server by running the **kibana.bat** file located at *`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.

How do I Secure API Data Store Communication using HTTPS with Search Guard Plugin?

You can use the Search Guard plugin to secure API Data Store (a simple Elasticsearch instance), one of the components in an API Management setup, to communicate securely over HTTPS.

Note:
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.

Before you begin

Ensure that you have:

- You have downloaded the Search Guard plug in. You can download the Search Guard plugin version 42.1.0 compatible for Elasticsearch 7.7.1 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"
```

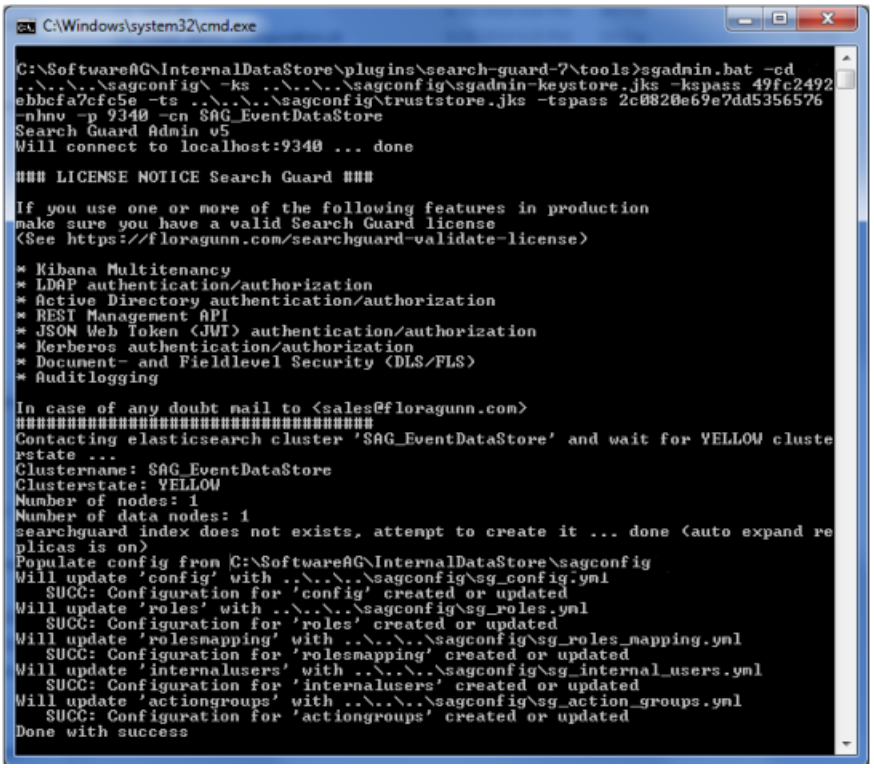
For details about all the Search Guard properties, see "Search Guard Properties" on page 132.

i. Save and close the file.

j. Shutdown and restart the API Data Store.

k. Navigate to *Installation_Dir*\InternalDataStore\plugins\search-guard-7\tools and run the following command to initialize the API Gateway 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
```



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.

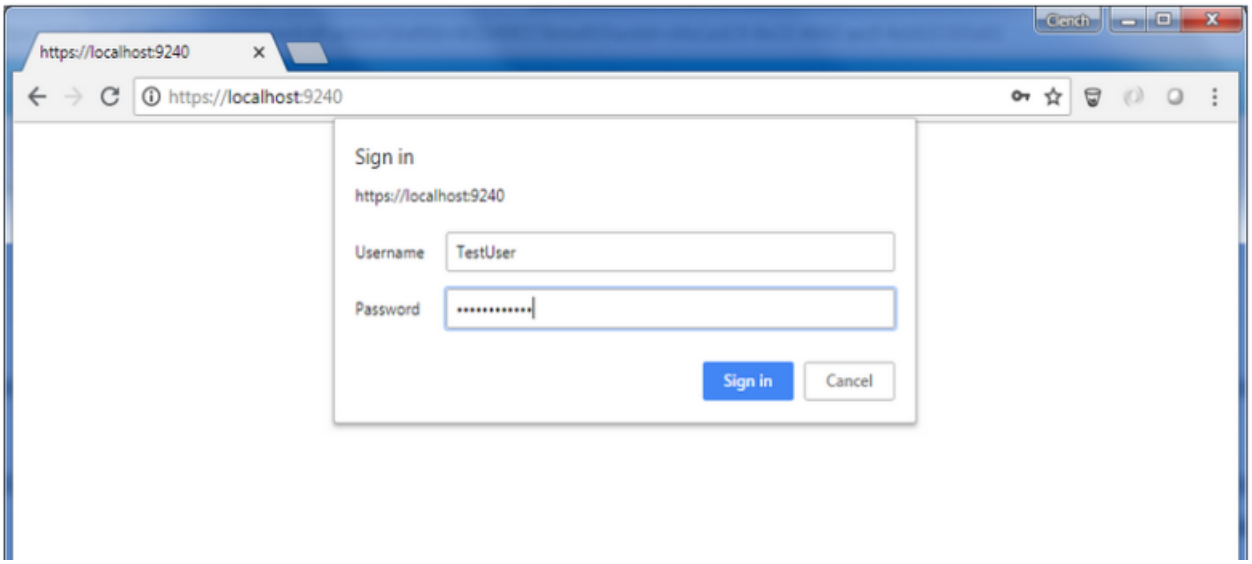
   d. Navigate to *Installation_Dir* \InternalDataStore\sagconfig and open the file sg_internal_users.yml.

   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.



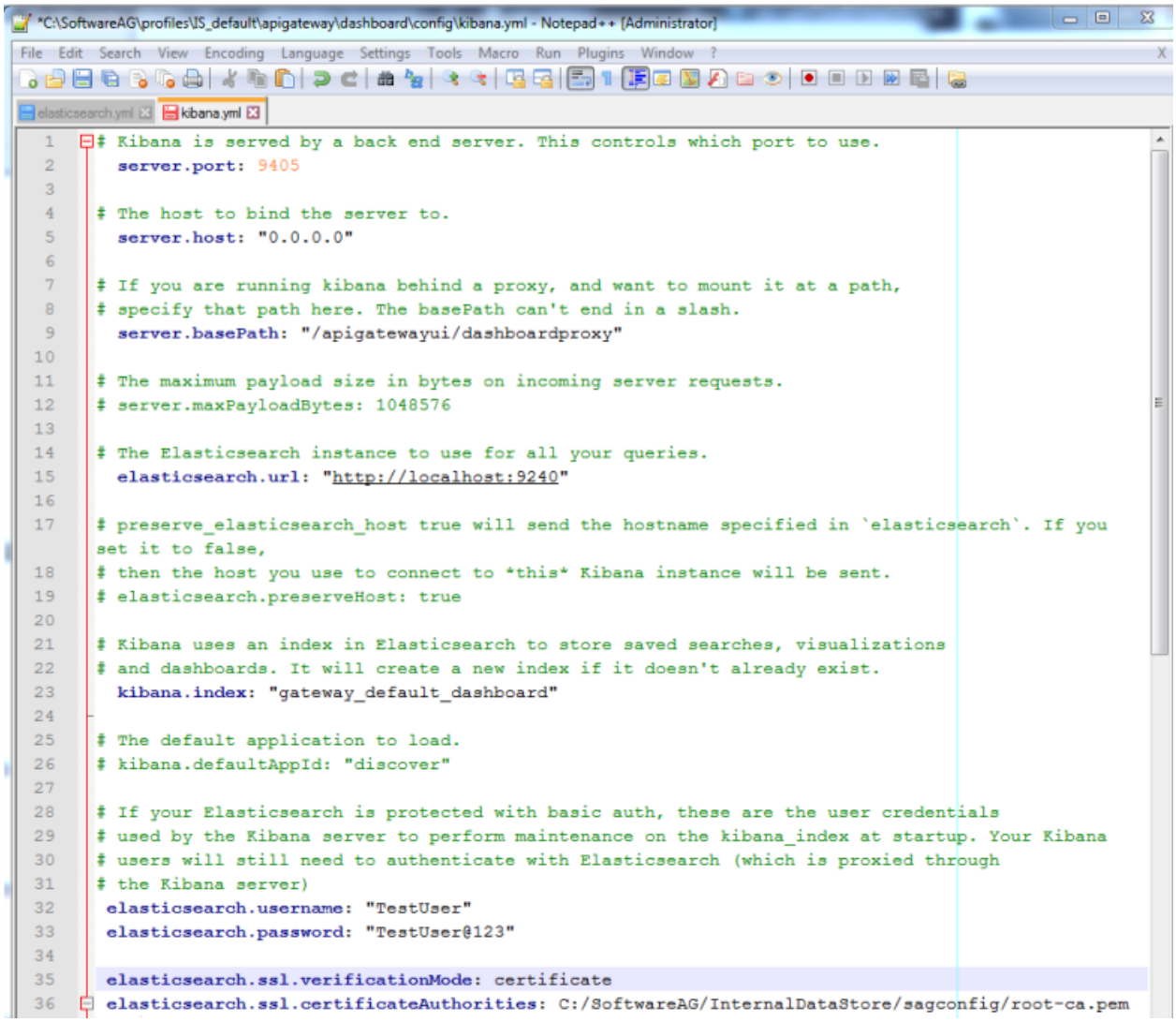
3. Change the Kibana configuration to connect to Elasticsearch.

   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. 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

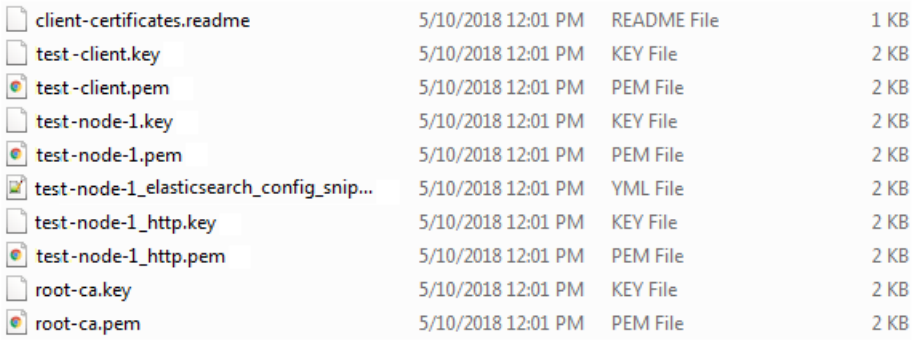
```
ca:
   root:
      # The distinguished name of this CA. You must specify a distinguished name.
      dn: CN=MCDIJA01,OU=eur,O=ad.sag Com\, Inc.,DC=Chennai,DC=IN
      # The size of the generated key in bits
      keysize: 2048
      # The validity of the generated certificate in days from now
      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.



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.seed_hosts: ["localhost:9340"]
transport.tcp.port: 9340
path.repo: ['C:\SoftwareAG\InternalDataStore/archives']

cluster.initial_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` Type of keystore. Possible values: JKS, PKCS12 Default value: JKS
`searchguard.ssl.transport.keystore_filepath` Location of the keystore.
`searchguard.ssl.transport.keystore_alias` Keystore entry name if there are more than one entries.
`searchguard.ssl.transport.keystore_password` Password to access keystore.
`searchguard.ssl.transport.truststore_type` Type of truststore. Possible values: JKS, PKCS12 Default value: JKS
`searchguard.ssl.transport.truststore_filepath` Location of the truststore.
`searchguard.ssl.transport.truststore_alias` Truststore entry name if there are more than one entries.
`searchguard.ssl.transport.truststore_password` Password to access truststore.
`searchguard.ssl.transport.enforce_hostname_verification` If `true`, the hostname mentioned in certificate is validated. Set this as `false` if you are using the general purpose self signed certificates. Possible values: true, false Default value: true
`searchguard.ssl.transport.resolve_hostname`

Property and description

If `true`, the hostname is resolved against the DNS server. Set this as `false` if you are using general purpose self signed certificates.

Note:
This is applicable only if the property searchguard.ssl.transport.enforce_hostname_verification is `true`.

Possible values: true, false

Default value: true

`searchguard.ssl.transport.enable_openssl_if_available`

Use if OpenSSL is available instead of JDK SSL.

Possible values: true, false

Default value: true

HTTP

`searchguard.ssl.http.enabled`

Set this to `true` to enable SSL for a REST interface (HTTP).

Possible values: true, false

Default value: true

`searchguard.ssl.http.keystore_type`

Type of keystore.

Possible values: JKS, PKCS12

Default value: JKS

`searchguard.ssl.http.keystore_filepath`

Location of the keystore.

`searchguard.ssl.http.keystore_alias`

Keystore entry name if there are more than one entries.

`searchguard.ssl.http.keystore_password`

Password to access keystore.

`searchguard.ssl.http.truststore_type`

Type of truststore.

Possible values: JKS, PKCS12

Property and description
Default value: JKS
`searchguard.ssl.http.truststore_filepath` Location of the truststore.
`searchguard.ssl.http.truststore_alias` Truststore entry name if there are more than one entries.
`searchguard.ssl.http.truststore_password` Password to access truststore.
`searchguard.ssl.http.clientauth_mode` Option to enable two-way authentication. Possible values: ■ REQUIRE : Requests for the client certificate. ■ OPTIONAL : Used if client certificate is available. ■ NONE : Ignores client certificate even if it is available. Default value: OPTIONAL
Search Guard Admin
`searchguard.authcz.admin_dn` 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` 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. Default value: '1.2.3.4.5.5'
`searchguard.config_index_name` Index where all the security configuration is stored. Currently, non-configurable. Default value: searchguard
`searchguard.enable_snapshot_restore_privilege`, `searchguard.check_snapshot_restore_write_privileges`

Property and description
Enables user privileges, which a user requires to perform snapshot and restore operations.
`searchguard.restapi.roles_enabled`
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 Gateway 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 Gateway Data Store are installed. API Gateway communicates with this API Gateway 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 Gateway 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*.

Installing API Gateway using Command Central

When you install API Gateway using Command Central, API Gateway, and API Gateway Data Store are installed. API Gateway communicates with this API Gateway Data Store by default.

You can install API Gateway from Command Central 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 User Interface

Before you begin, it is important to understand the following terms.

- **Host Node**: The primary node on which Command Central is installed. You can install API Gateway on other nodes by using the Command Central instance present in the host node,
- **Remote Nodes**: The nodes on which API Gateway must be installed from the Command Central instance, present on the host node. You can install API Gateway either on a single remote node or multiple remote nodes. You can also install API Gateway on the host node.

Prerequisites

- Command Central must be installed on the host node.
- Host node and remote nodes must be associated to the same domain network.

To install API Gateway using Command Central user interface, perform the following tasks.

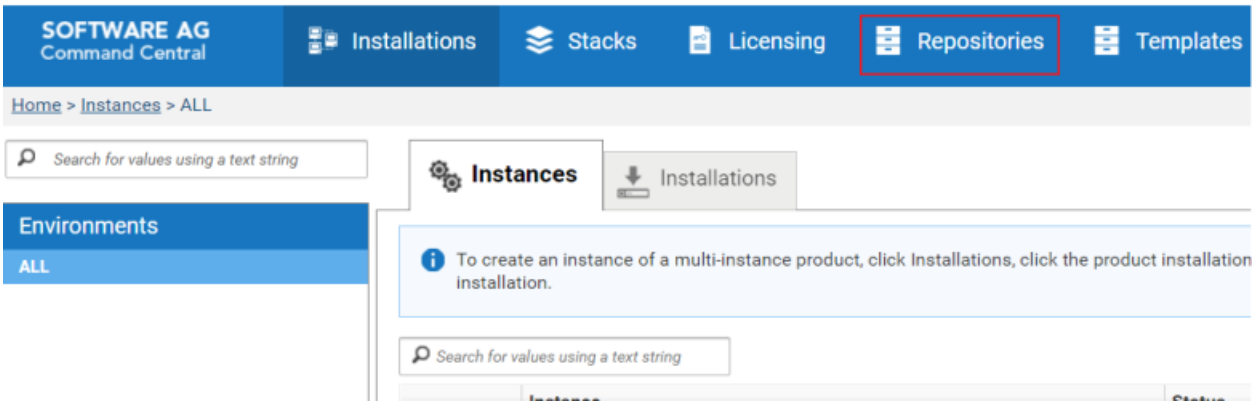
1. “Connect to Repository” on page 141.
2. “ Configure Platform Manager on remote nodes” on page 142.
3. “Install API Gateway” on page 150.
4. “Create API Gateway Instance” on page 153.

Connecting to Repository

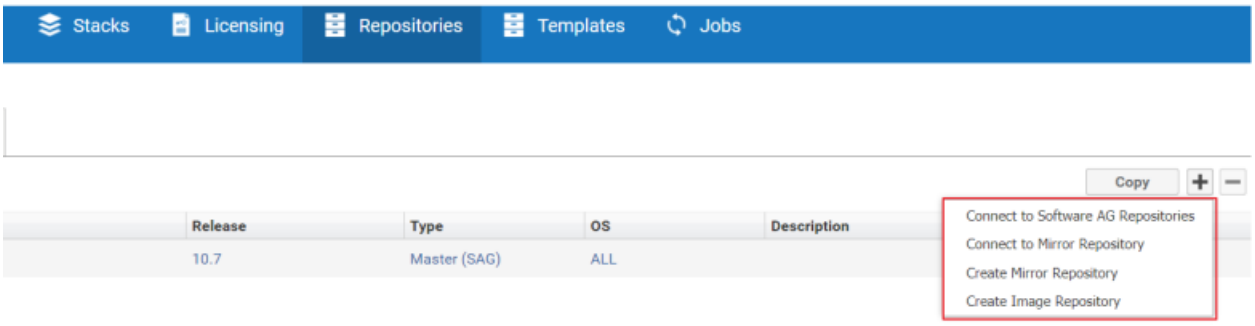
The Software AG Download Center has repositories, which contain Software AG products. You must connect your Command Central instance with a Software AG repository.

To connect to a Repository

1. Log on to Command Central.
2. Click **Repositories**.

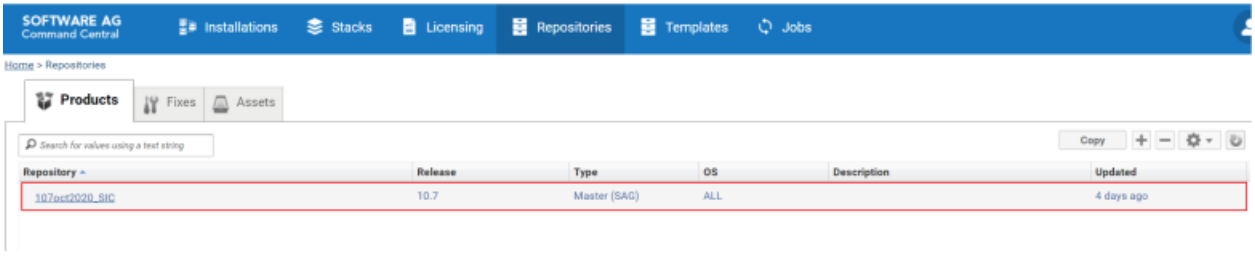


3. Click **+** and select a method to connect to a repository.



To learn more about the fields present in each method of connecting to Repository, see Create, Refresh, or Change Source for Mirror Repository article from Command Central help.

Once you connect to a repository, you can see it in the list of Repositories page.

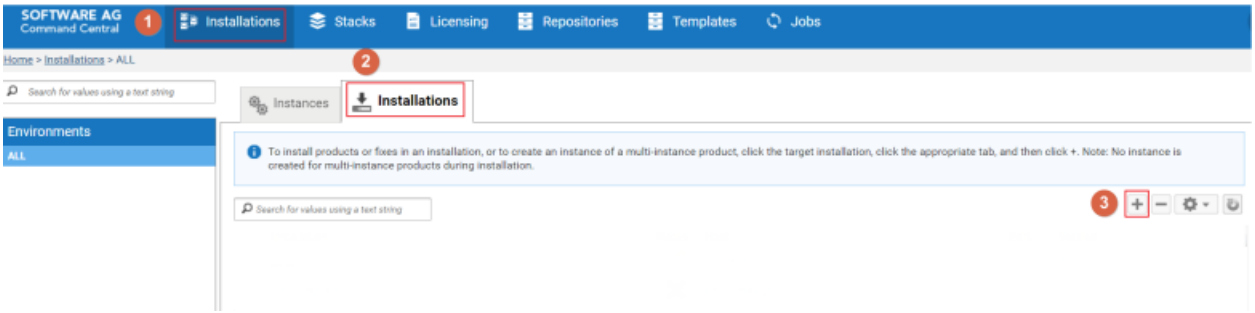


Configuring Platform Manager on Remote Nodes

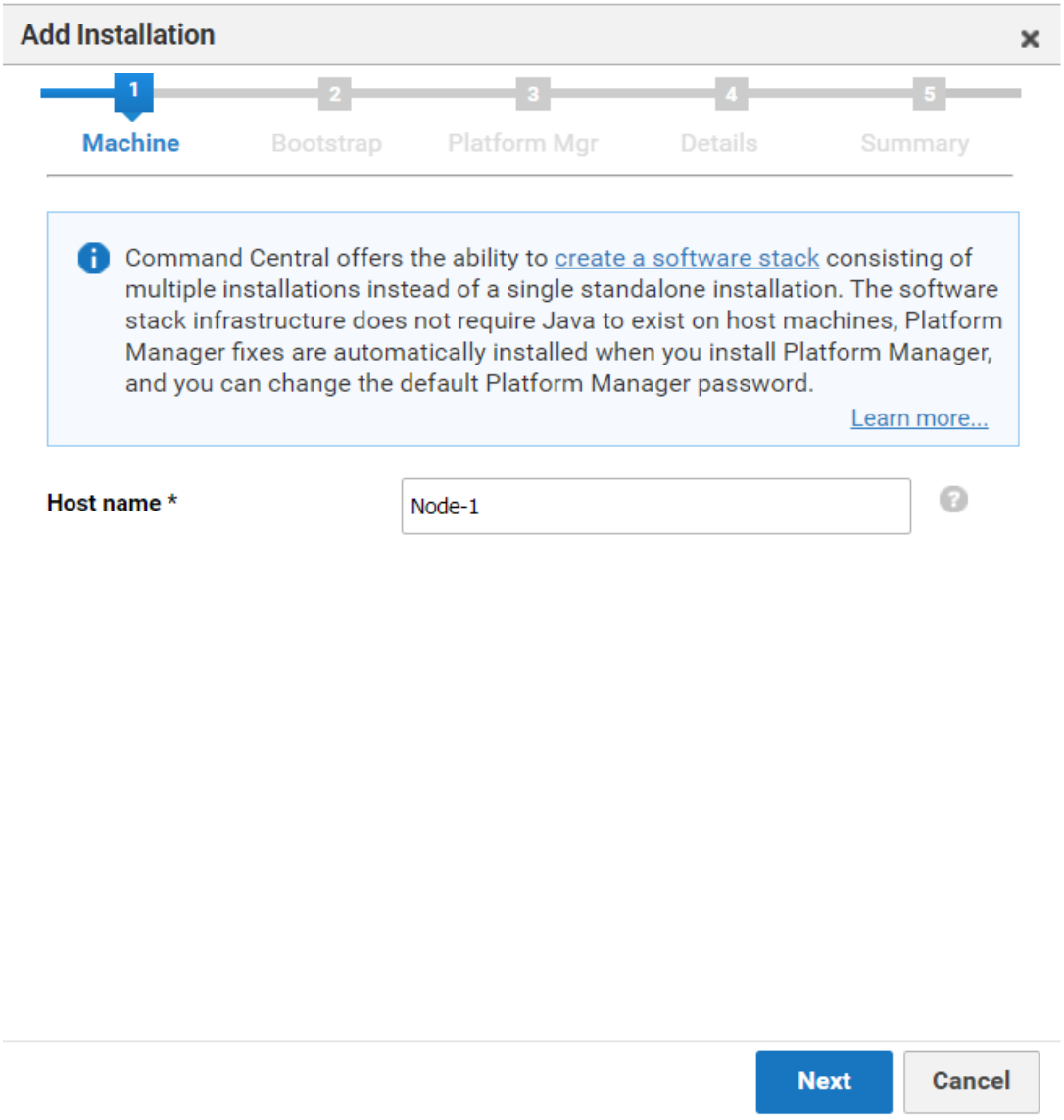
A Platform manager is a Software AG tool that assists you in installation and upgradation of Software AG products. To perform API Gateway installation using Command Central, you must install Platform manager on remote nodes. If Platform Manager is not installed on any remote node, you must install it as well through Command Central.

To configure Platform Manager

1. Click the **Installations** menu.
2. Click the **Installations** tab.
3. Click **+**. The **Add Installation** window is displayed.



4. Type the host name or IP address of the remote node in the **Host name** field.
5. Click **Next**.



6. Select one of the following options on the Bootstrap window.

   - **Platform Manager is already installed**: Select this option if platform manager is already installed on the remote node.
   - **Install Platform Manager remotely**: Select this option if Platform Manager is not installed on the remote node

7. *This step is applicable only if you have selected option b in step 6*. Configure the following fields.

   a. **Operating System**. Select the operating system of the remote node.

   b. **Installation directory**. Select the directory in which Platform Manager must be installed. If you are installing API Gateway on the host node, do not select the same directory in which Command Central is installed.

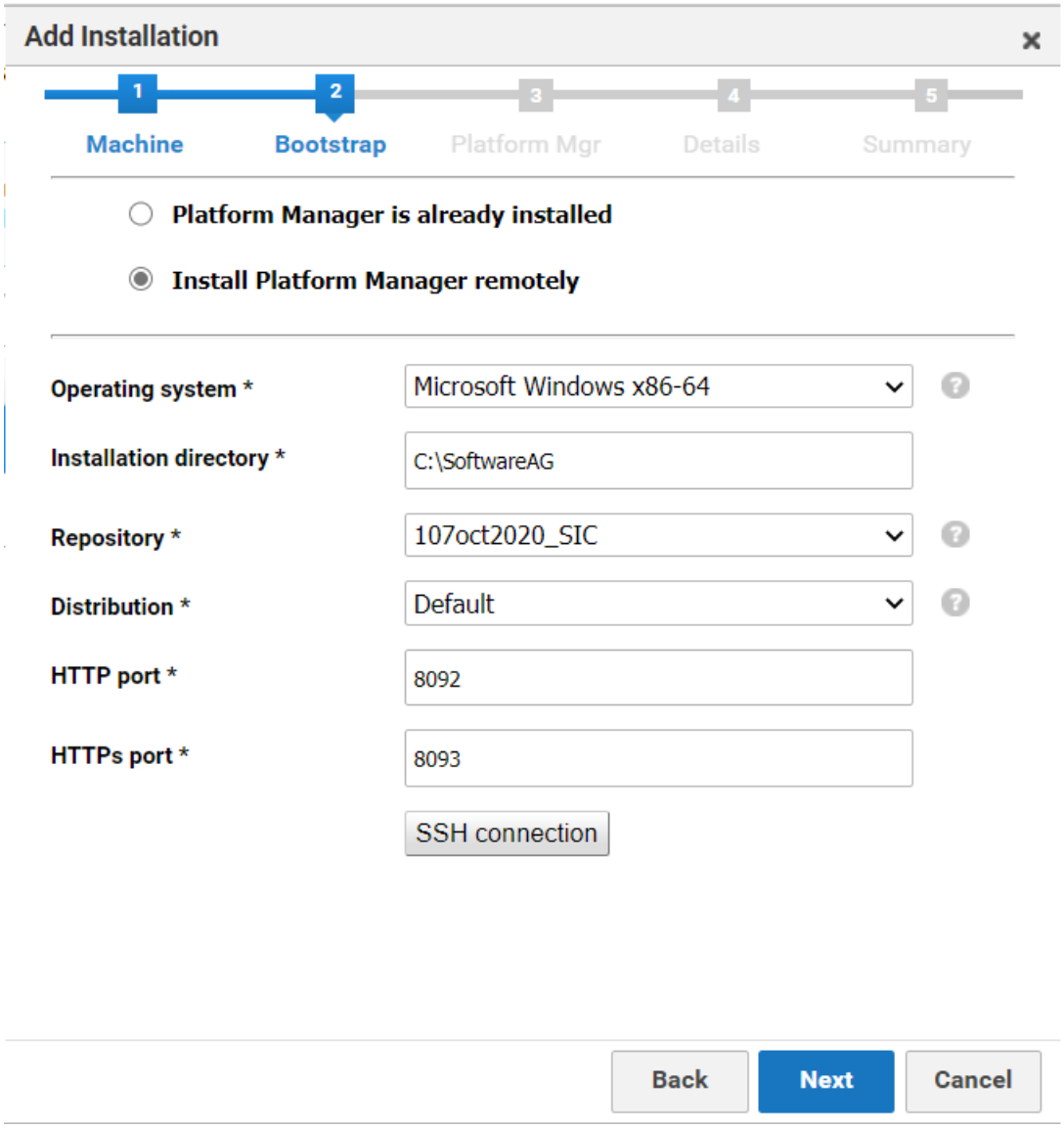
c. **Repository**. Select a repository to install Platform Manager, if you have configured multiple repositories. If a single repository is connected, it is selected by default.

d. **Distribution**. Select one of the following options.

- **Default**. Select this option if you do not want to configure plugins.
- **Complete**. Select this option to configure plugins.

e. **HTTP port**. Type the HTTP port number of Platform manager. By default, this value is set to 8092.

f. **HTTPs port**. Type the HTTPS port number of Platform manager. By default, this value is set to 8093.



8. *This step is applicable only if you selected option b in step 6*. Configure SSH connection by executing the following steps.

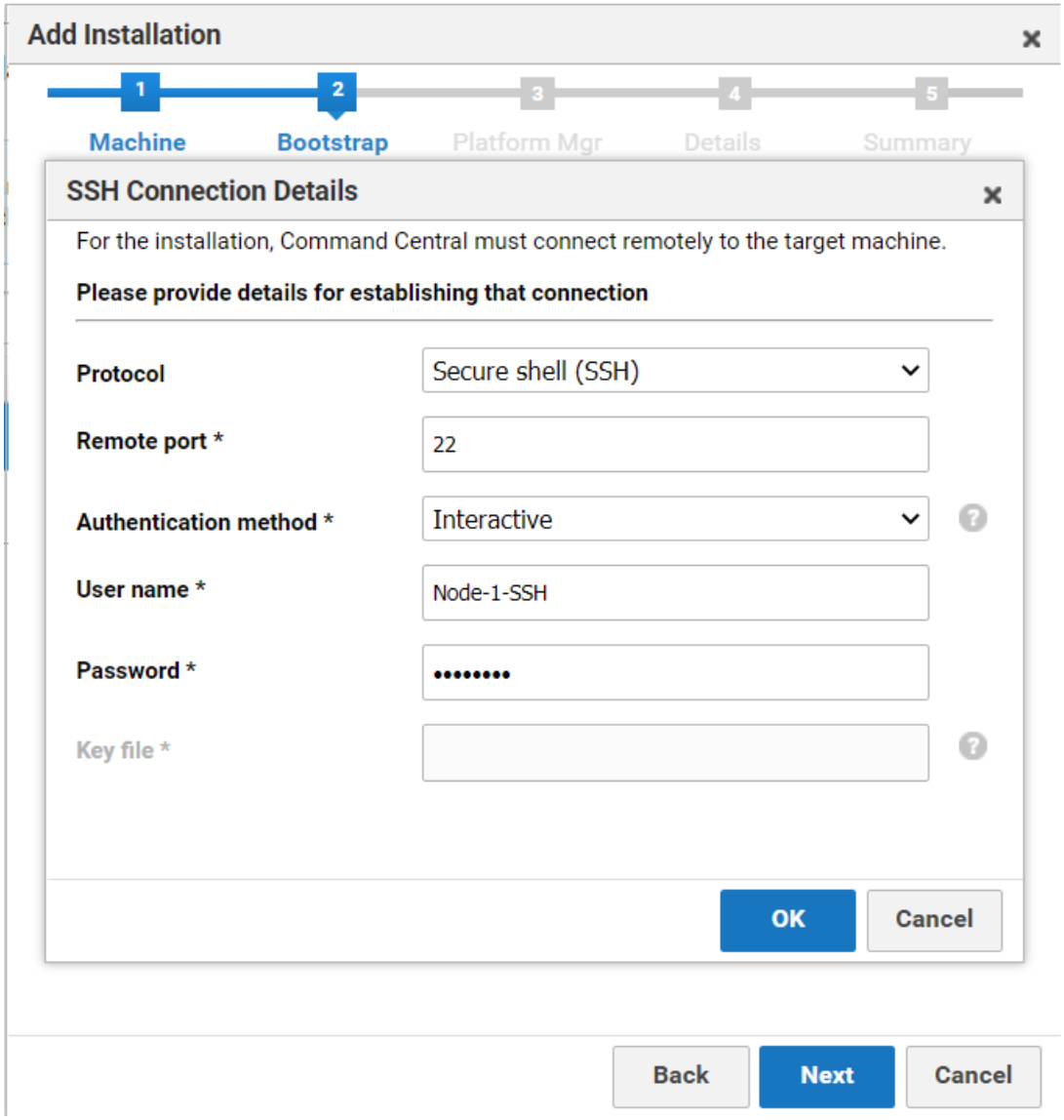
   Important:
   Before configuring SSH from host node, you must set up SSH on the remote node using the Cygwin tool. To learn more about how to perform this, see the Using Cygwin to Configure Open SSH tech community article.

   a. Click **SSH connection**. The SSH Connection Details window displays.

   b. Configure the following fields.

   - **Protocol**. Select Secure Shell (SSH).
   - **Remote port**. Type the port number of the remote node, on which SSH is running.
   - **Authentication method**: Select one of the following options.
   - **Password**. The credentials are sent for verification through the SSH tunnel by the client.
   - **Interactive**. The server initiates a password request session.
   - **Certificate**. The client sends a signature based file created from the user's private key.
   - **User name**. Type the user name of the remote node used while configuring SSH.
   - **Password**. Type the password of the remote node used while configuring SSH.

   c. Click **OK**.



9. Click **Next**.

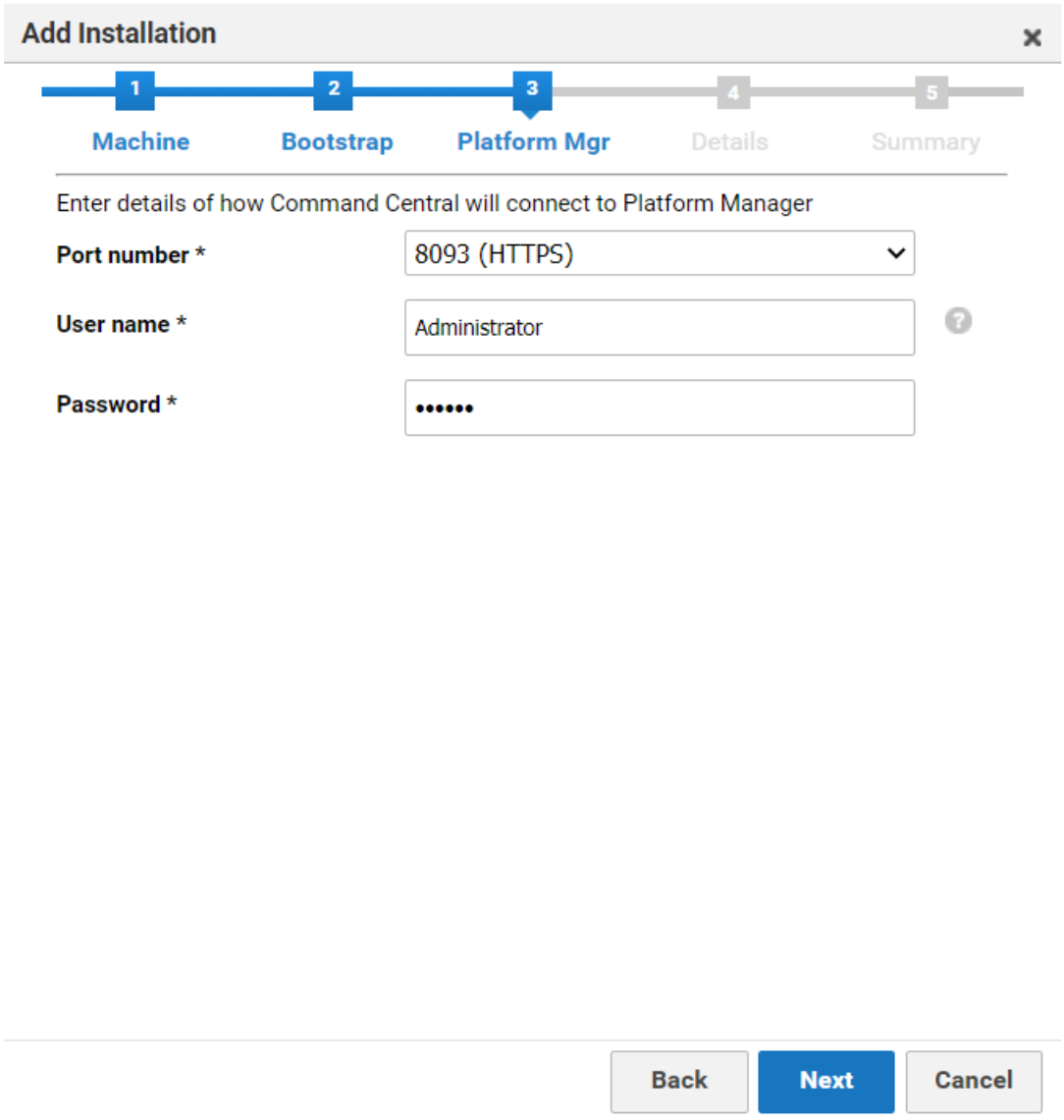
10. Configure the following fields on the Platform Manager window.

    a. **Port number**. Select the port (HTTP or HTTPS) through which Platform manager must be accessed.

    b. **User name**. Type the user name of the Platform manager instance, installed or to be installed on the remote node.

    c. **Password**. Type the password of the Platform manager instance, installed or to be installed on the remote node.

    d. Click **Next**.



11. Configure the following fields on Details window.

    a. **Display name**. Modify the display name of the remote host, if required. By default, the name typed in the Machine window is used

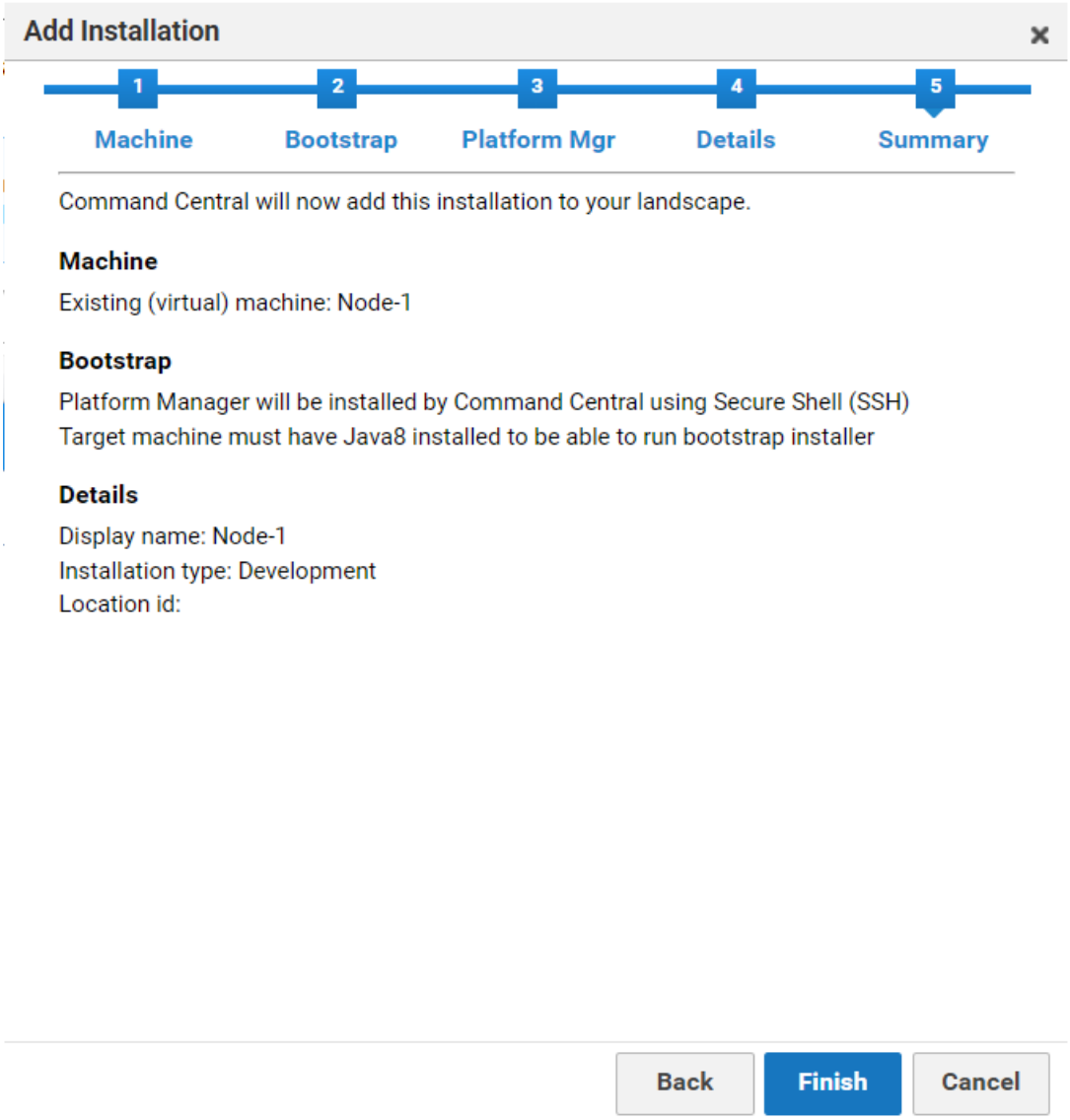
    b. **Alias**. Modify the alias name of the remote host, if required. By default, the name typed in the Machine window is used.

    c. **Description**. Type description of the remote node.

    d. **Installation type**. Select if the remote node is part of Production, Development or Test environment.

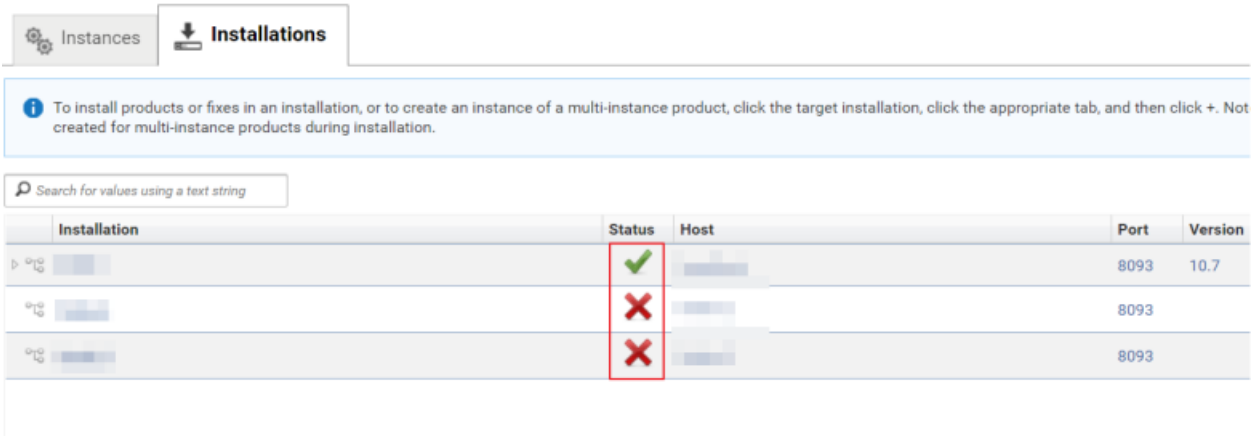
    e. Click **Next**.

12. Verify the details on the Summary window and click **Finish**.



To add more remote nodes, repeat steps 1-12 of the "Configuring Platform Manager" on page 142 section.

If Command Central is able to access Platform Manager, installed on the remote node, the **Status** column displays the **Online** symbol (right arrow). If not, the **Status** column shows the **Offline** symbol (cross mark).



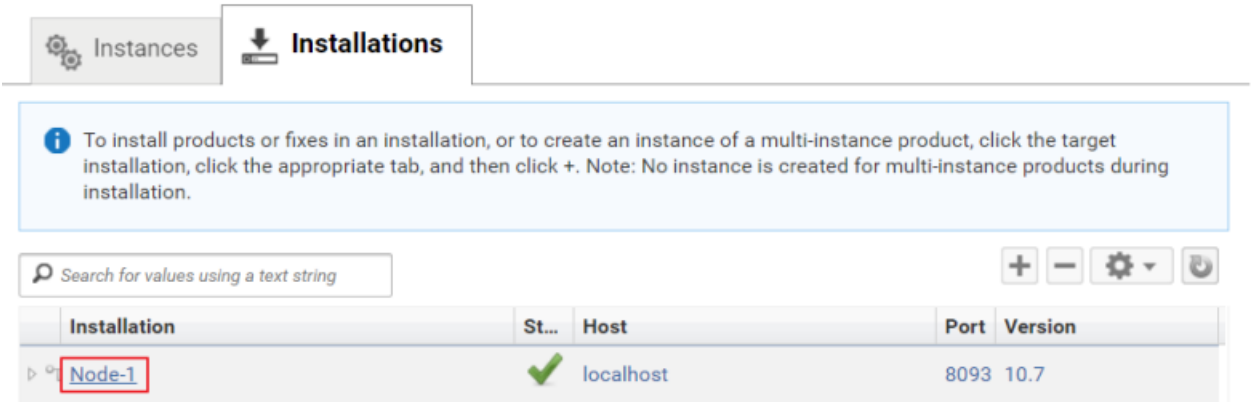
If Command Central is unable to connect to Platform Manager on the remote node, ensure that Platform Manager is started on the remote node. Also, ensure that there is no firewall that may be blocking the host node.

Installing API Gateway

After Command Central establishes a connection with Platform Manager (present on the remote node), you can install API Gateway on the remote node.

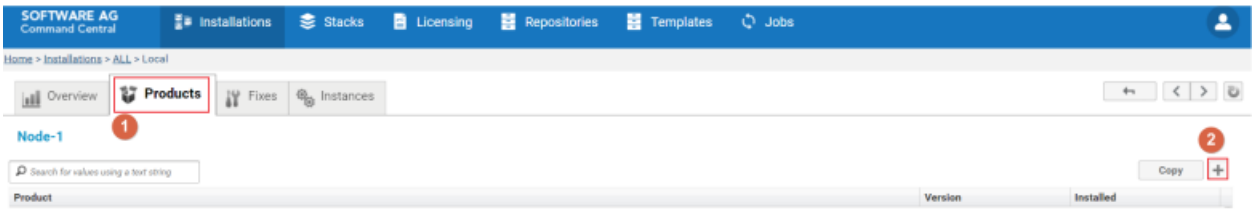
To install API Gateway

1. From the **Installations** tab, select the required remote node (the remote node on which API Gateway must be installed).



2. Click the **Products** tab.

3. Click **+**. The **Install Products** window displays.

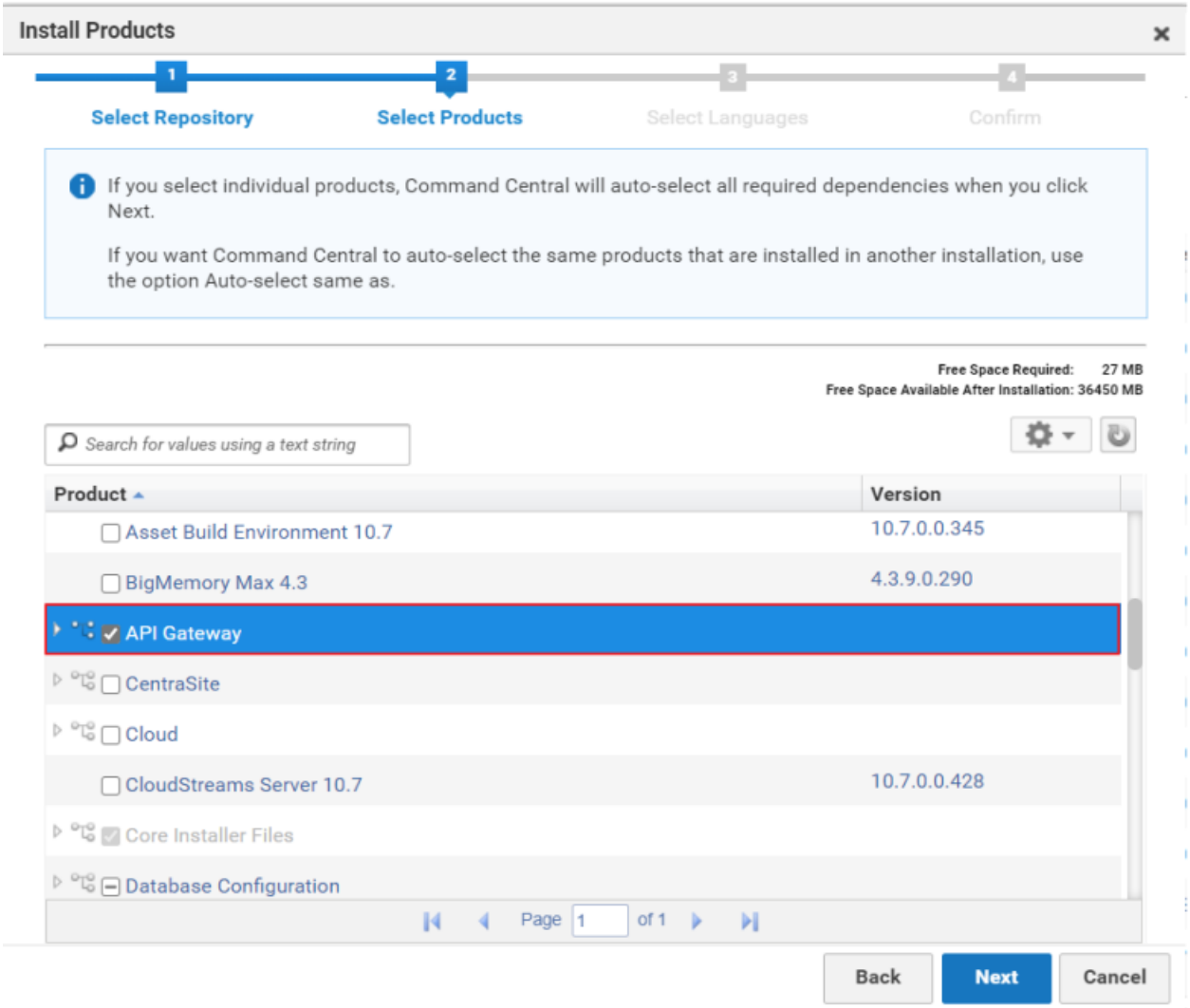


4. Select a repository to install API Gateway, if you are connected to multiple repositories. If a single repository is connected, it is selected by default.

5. Click **Next**.

6. Select the **API Gateway** check box. This selects API Gateway for installation.

7. Click **Next**.



8. Select any additional languages of installation.

9. Click **Next**.

10. Verify the details and click **Finish**.

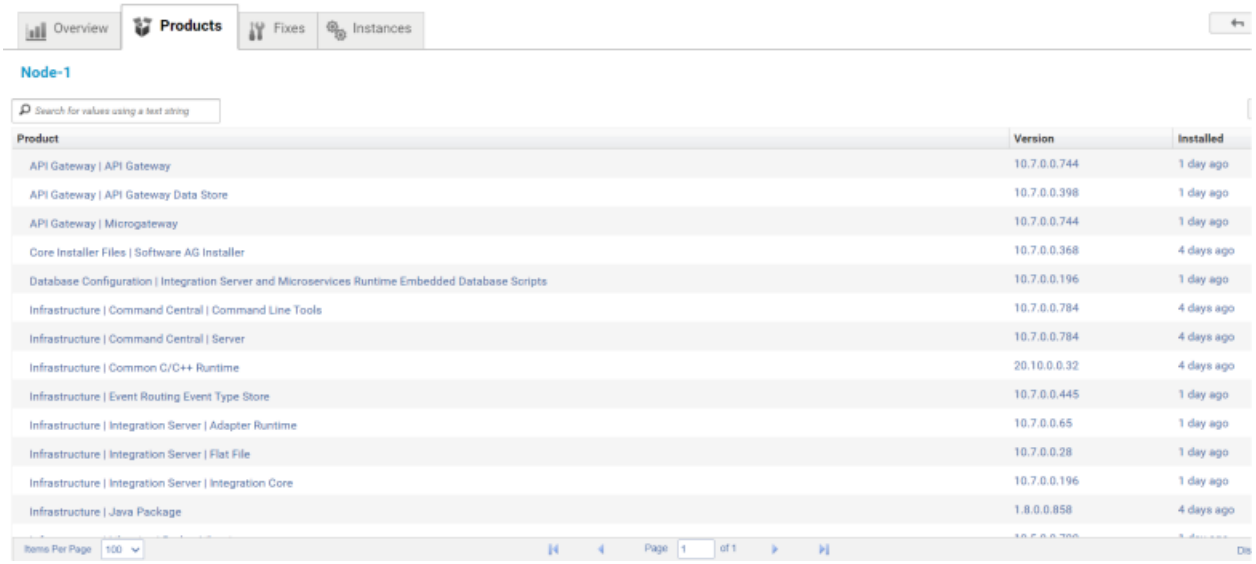
    This starts the installation process of API Gateway on the remote node. A job is created in the **Jobs** menu. You can check the status of installation from the **Jobs** menu. To install API Gateway on multiple nodes, repeat the steps in Installing API Gateway section for each remote node.

    After API Gateway is installed on a remote node, the job status is updated as successful. You can view the list of products that were installed on the remote node by execute the following steps.

11. Navigate to the **Installations** tab.

12. Click the required remote node (the remote node whose installation needs to be checked).

13. Click the **Products** tab. You can view the list of installed products.



Creating API Gateway Instance

After you install API Gateway, the installed API Gateway is plain and not usable. To make the API Gateway usable, you must create an instance of it in Command Central. To create an instance, you must add license file, port information, and so on. You cannot use API Gateway without creating an instance.

To create an instance

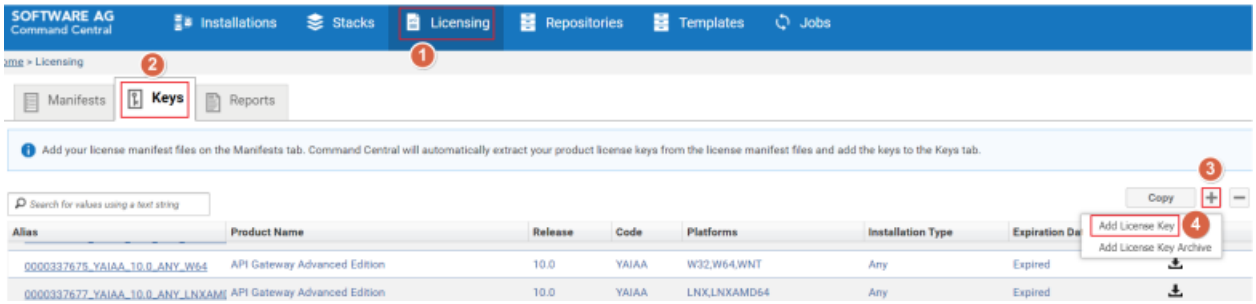
1. Perform the following steps to add an API Gateway license file.

    a. Click **Licensing**.

    b. Click the **Keys** tab.

    c. Click **+**.

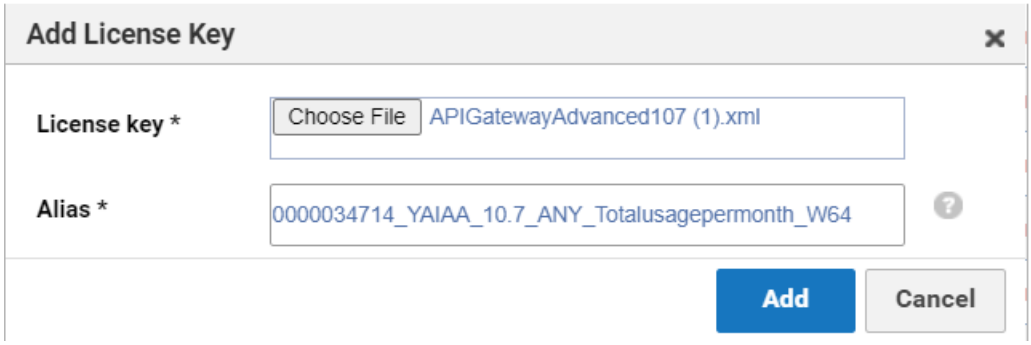
    d. Select **Add License Key**. The Add License Key pop-up window displays.



e. Click **Choose File**.

f. Navigate to the location where your API Gateway license file is located and select it. The **Alias** field value is auto populated after you select the API Gateway license file.

g. Click **Add**.

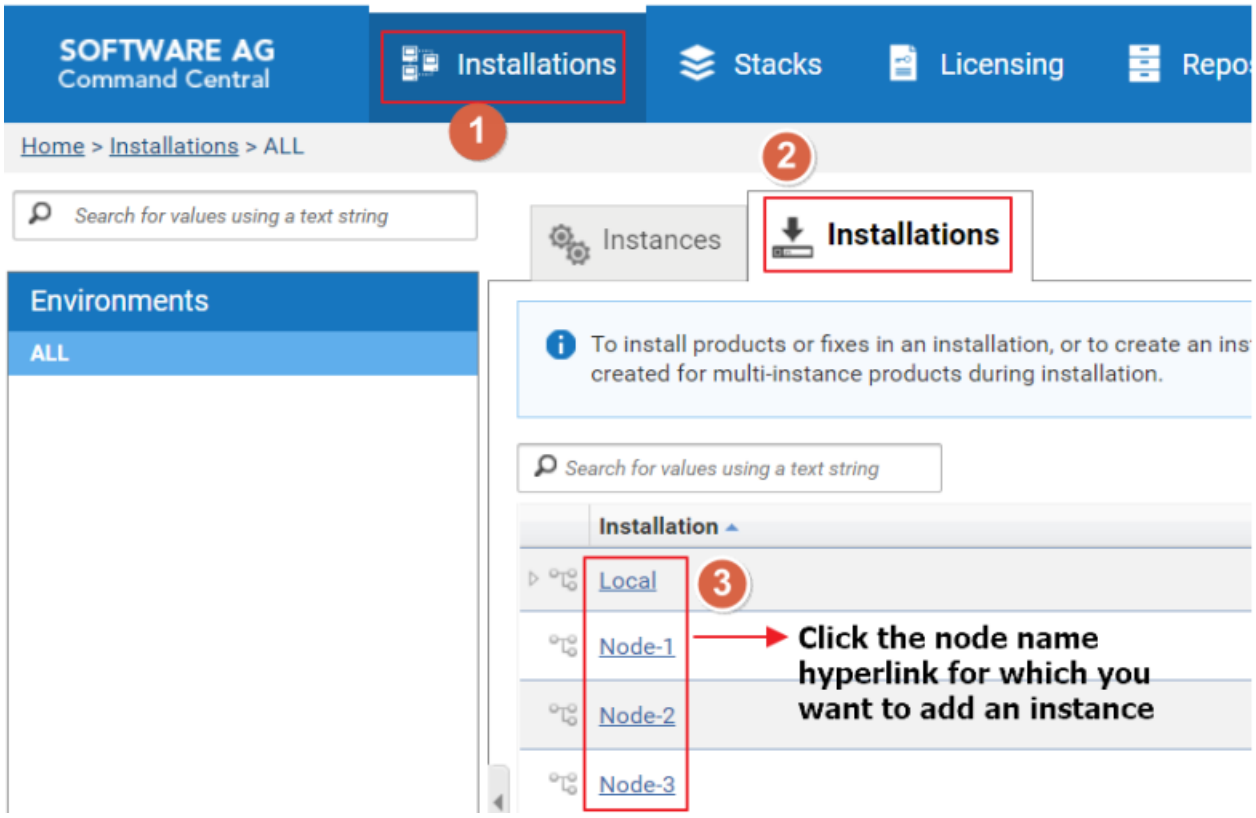


2. Perform the following steps to create an API Gateway instance.

a. Click the **Installations** menu.

b. Click the **Installations** tab.

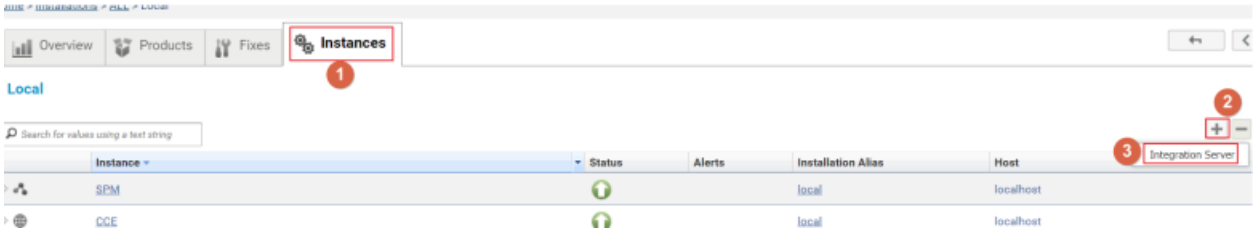
c. Click the required node name (the node for which an instance needs to be added).



d. Click the **Instances** tab.

e. Click **+**.

f. Select **Integration Server**.

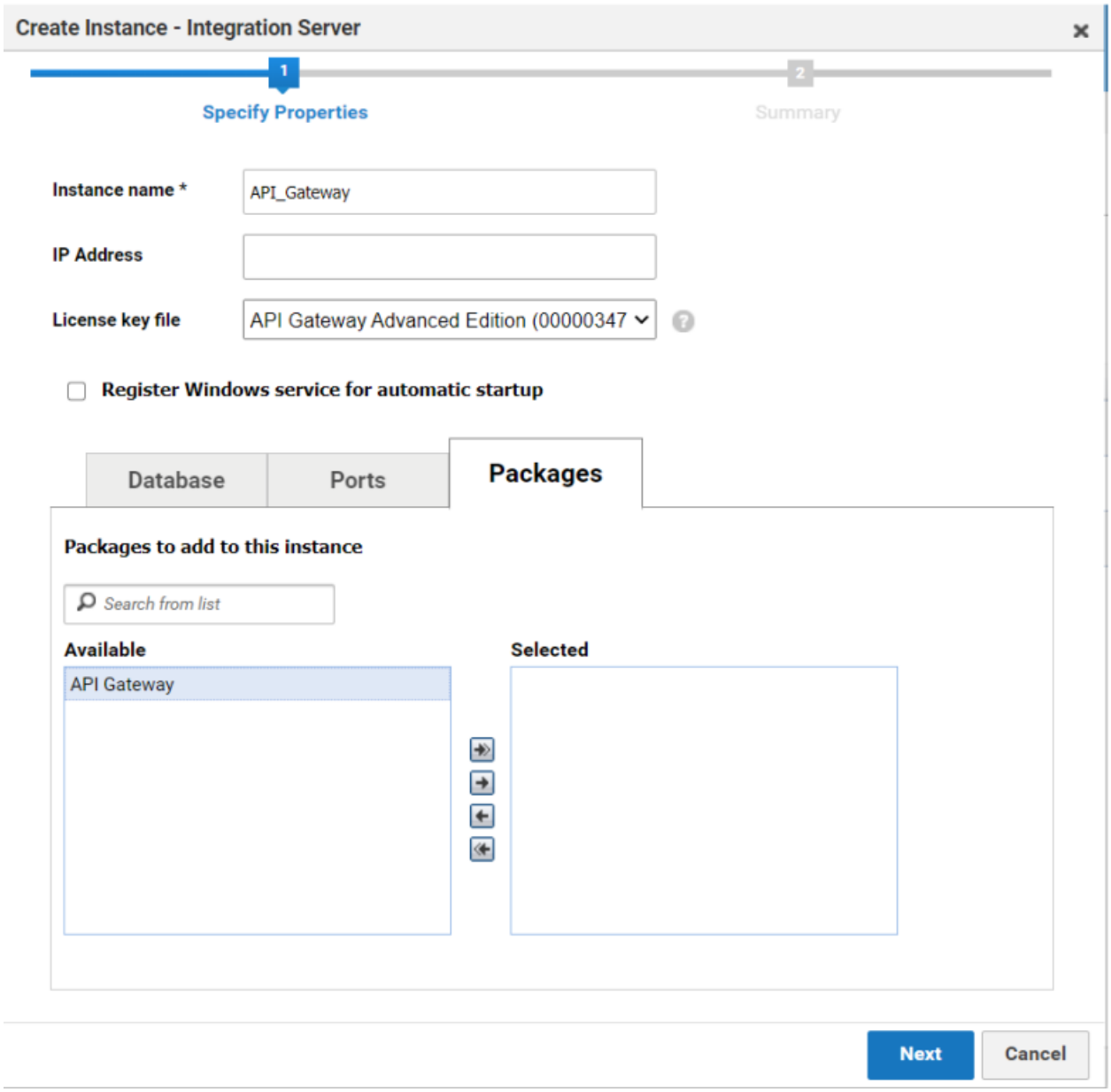


g. Type a name for the instance in the **Instance name** field.

h. Select the API Gateway license key file from the **License key file** drop-down menu.

i. Configure the **Database**, **Ports**, and **Packages** for API Gateway.

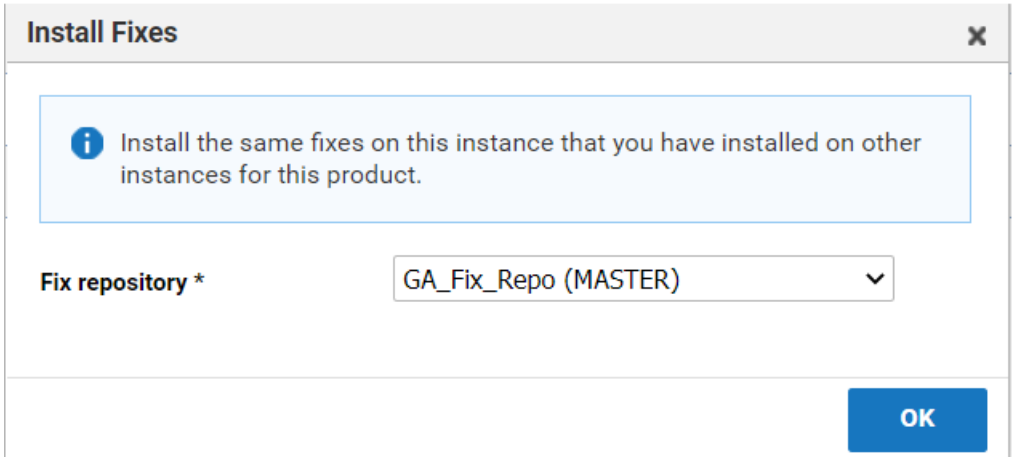
j. Click **Next**.



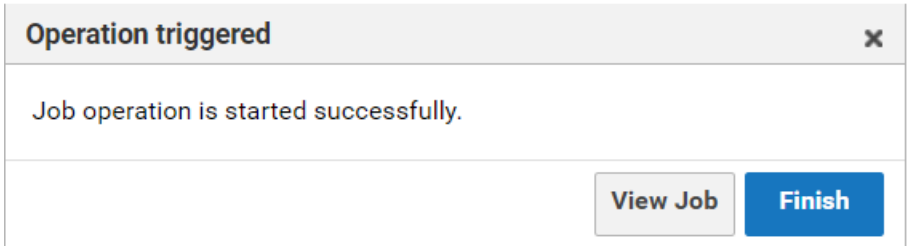
k. Verify the details and click **Finish**. The Install Fixes pop-up window is displayed.

l. Select a repository to install API Gateway fixes.

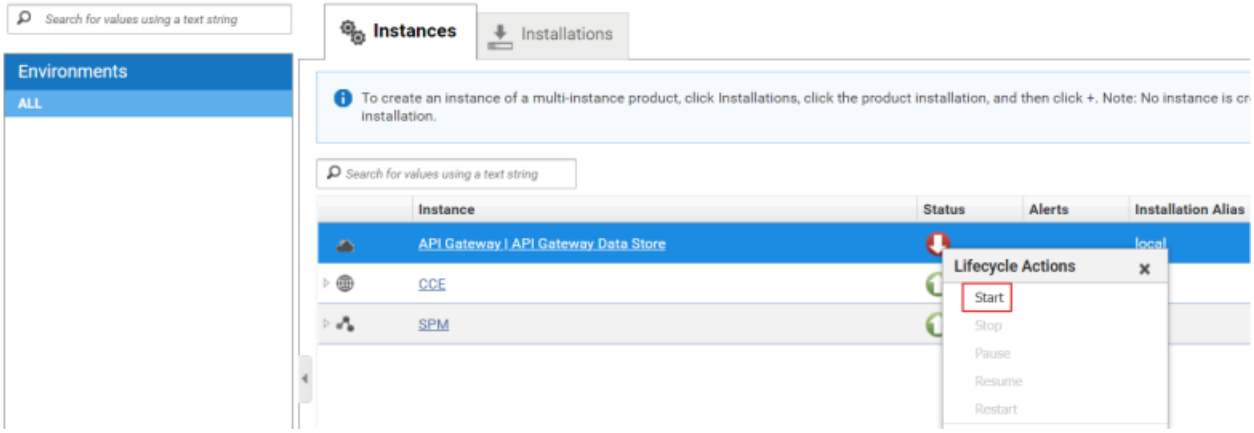
m. Click **OK**.



A job is created to create an API Gateway instance. The **Operation triggered** pop-up window displays**.** Click **View Job** to view the details of the job in the **Jobs** menu. If you do not want to view the job details, click **Finish**.



After the job is completed, you can view the API Gateway instance. If the **Status** column of your instance shows **Stopped** status, click the **Stop** icon and select **Start**.



Installing API Gateway Using Command Central templates

This section lists the steps that you need to execute to install API Gateway using Command Central templates. A Command Central template is a YAML file (.yaml extension file) which contains all the installation information.

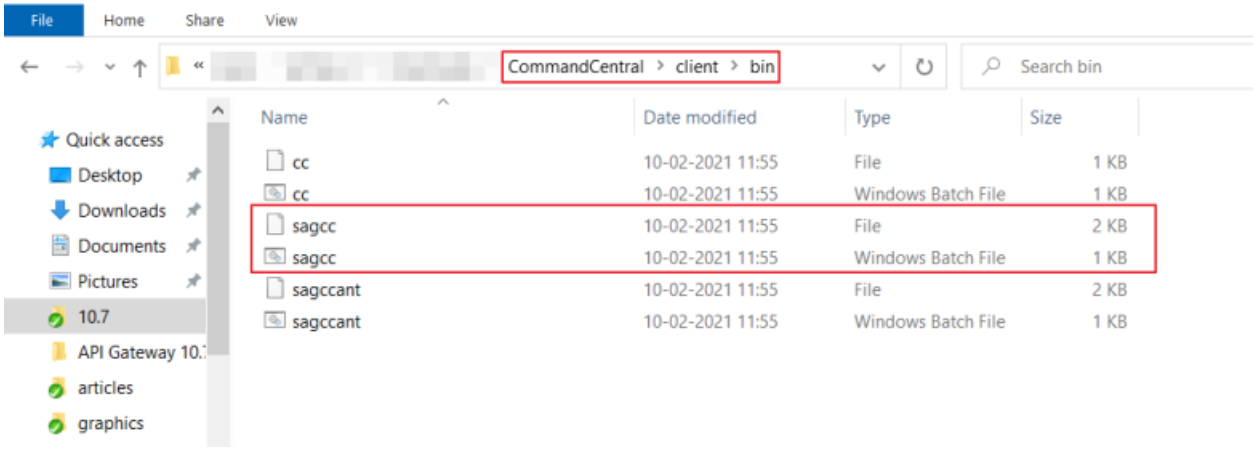
To install API Gateway on multiple nodes, all the information about host node, remote nodes must be provided in the Command Central template. Also, other important information like environment details (in the form of key-value pairs), API Gateway Data Store details, product details (instance details), API Gateway ports information, must be provided in the template. For more details on how to use Command Central templates, see the *Software AG Command Central Help*.

Prerequisites

- Ensure that your Command Central template is configured as per your requirements.

You can install API Gateway by using Command Central templates by executing commands. You must execute the commands from the host node. If the host node is a windows machine, you can execute the commands from either Command Prompt or Windows Powershell. If the host node is on a Linux machine, you can execute commands from the Linux Terminal.

When you install Command Central, the installation directory has a batch file and a code file. Both of their names are **sagcc**. All Command Central commands start with **sagcc**. To execute the commands, your command prompt or terminal must point to the directory in which sagcc files are located. The sagcc files are located in the `<installation directory>/CommandCentral/client/bin` folder. On a Windows machine, the image looks as follows.



To install API Gateway using templates

1. Navigate your Command Prompt/Powershell/Terminal to point to the folder in which sagcc files are located.

2. Run the following command to add the credentials to connect 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
```

3. 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.

4. 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.

5. Run the following command to import the API Gateway installation template. This command imports the Command Central template from the location given in the command and places it in the `<installation directory>\profiles\CCE\data\templates\composite` folder. The sample installation template, *template.yaml* is used in the following command. `sag/apigateway/server/trunk` is the location of this file.

```
sagcc exec templates composite import -i <template path>/template.yaml
```

This imports the template required for installing API Gateway.

6. Execute one of the following command to run the template.

- If you have not configured any parameters in your template, execute the following command. All the parameters are configured in this command.

```
sagcc exec templates composite apply template_alias
nodes=local is.instance.type=integrationServer agw.memory.max=512
repo.product=webMethods-10.5 os.platform=W64
agw.key.license=apigateway_license
```

- If you have configured all the parameters in your template, execute the following command.

```
sagcc exec templates composite apply template_alias
```

- If you have configured the parameters in the environment.properties file, execute the following command.

```
sagcc exec templates composite apply template_alias -i <path to the
environmt.properties file>
```

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.

7. 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.
```

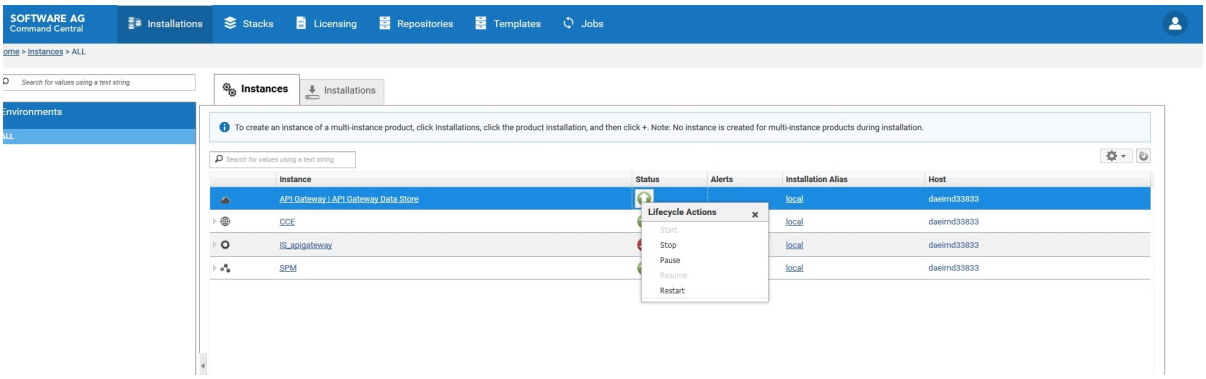
d. Apply the template.

```
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
```

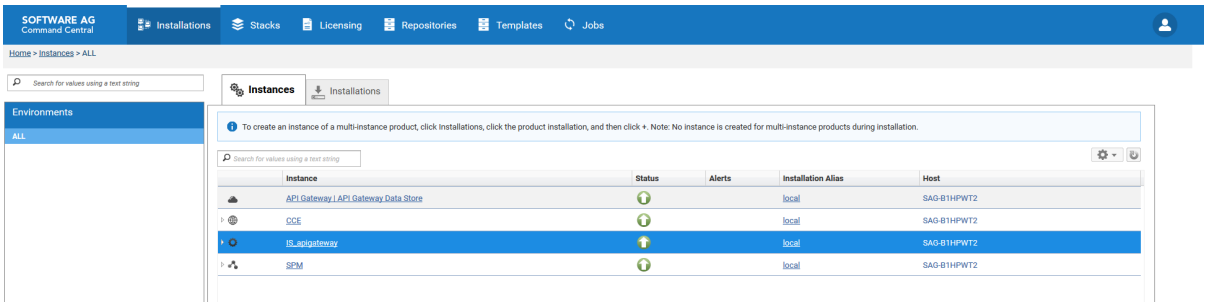
This procedure completes API Gateway installation and you can see API Gateway and API Gateway Data Store in **Jobs** menu of the Command Central UI.

In Command Central,

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



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



Manage API Gateway Data Store Configurations in Command Central

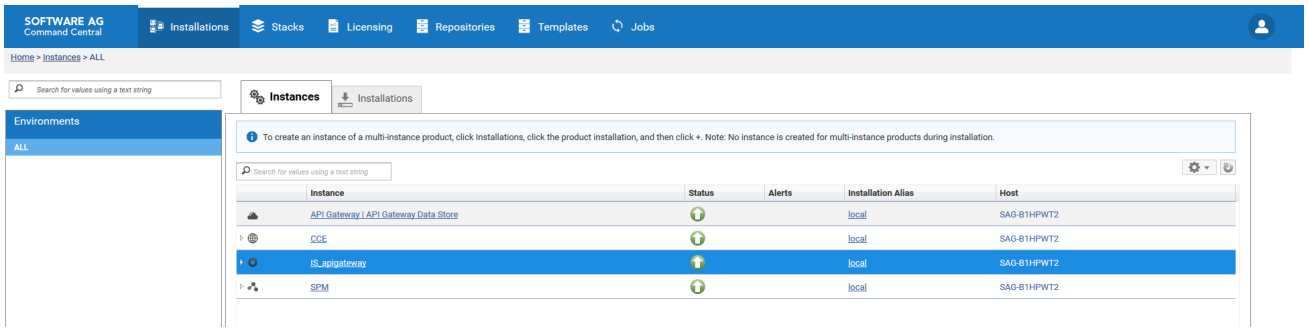
Command Central lists API Gateway and API Gateway Data Store shipped with API Gateway. API Gateway stores all its core and analytics data in this Data Store by default. You can start, stop, and restart API Gateway Data Store from Command Central. You can also manage Clustering details, Keystores, Ports,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

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

- "Starting and Stopping API Gateway Data Store in Command Central" on page 36
- "Changing the API Gateway Data Store HTTP Port" on page 39
- "Changing the API Gateway Data Store TCP Port" on page 40
- "Configuring an API Gateway Data Store Cluster" on page 41
- "Configuring Elasticsearch Properties" on page 44

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.



The image shows the IS instance apigateway with the name IS_apigateway. Under IS_apigateway, users can configure the following assets and components of API Gateway instances:

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

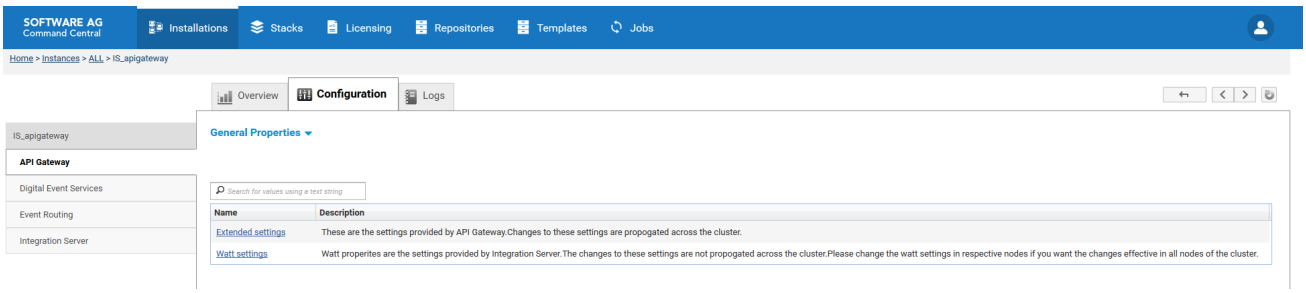
- Loggers
- Ports
- Truststores

Configuring Properties

This section provides information about configuring Extended and Watt settings of API Gateway.

To configure the properties

1. In Command Central, navigate to **Environments** > **Instances** > **All** > **API Gateway** > **Configuration**.
2. Click **General Properties**. The **General Properties** page appears.
3. Click **Extended Settings**. The properties are listed as key value pairs.
4. Make the required changes.
5. Click **Save**.



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

Configuring Keystores

This section provides information about adding keystores for API Gateway from Command Central.

To configure the Keystores

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

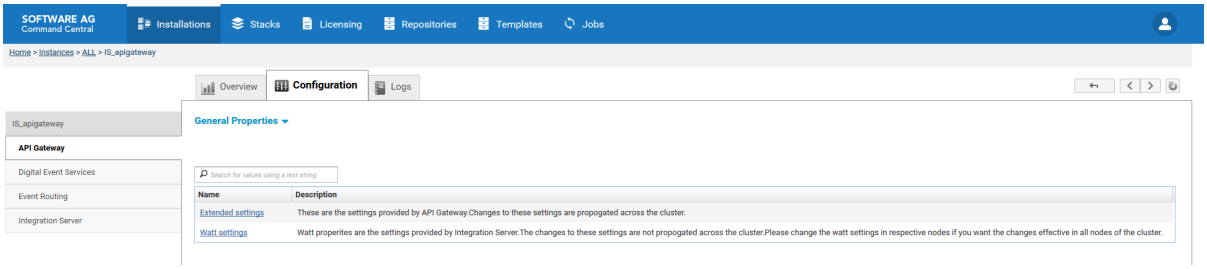
2. Select **Keystores** from the drop-down menu.

   The Keystores list appears.

3. Click  to add a new keystore.

4. Provide an **Alias** for the keystore.

5. Provide **Type**, **Provider**, and **Location** of the keystore in the **Keystore Configuration** section.



6. Click **Save** .

   The keystore is added to the list.

Configuring Keystores using Template

You can configure Keystores using the following Command Central template:

```
sagcc exec templates composite import -i keystore.yaml
sagcc exec templates composite apply keyStoreAlias nodes=local
keystore.path=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

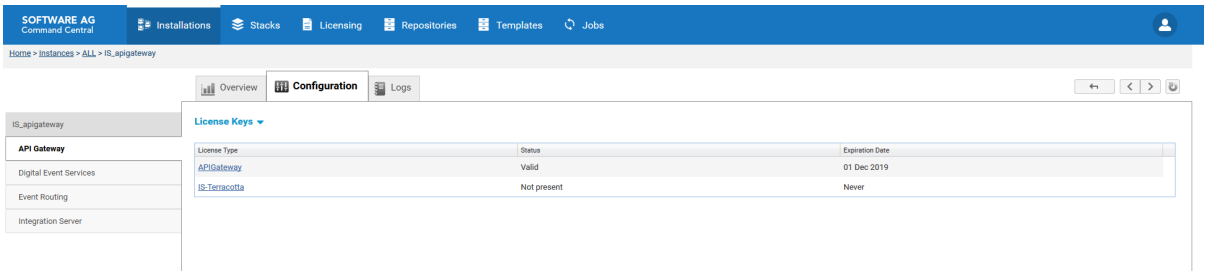
This section provides information about adding API Gateway licenses using Command Central.

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.



3. Click  to add a new license and provide the required license.

Configuring Loggers

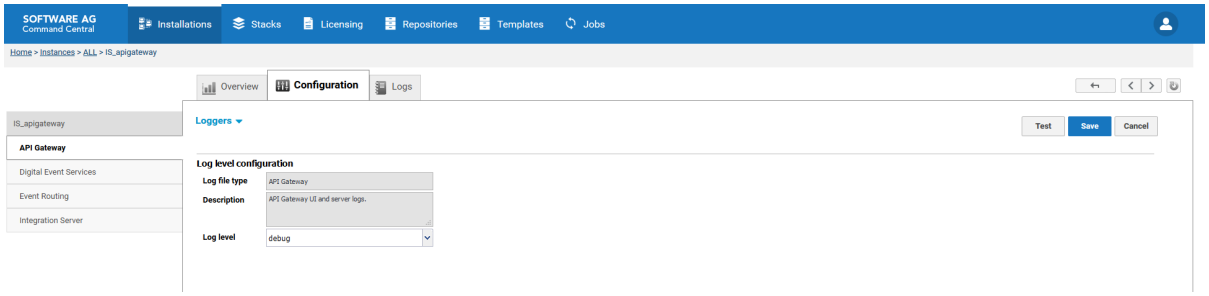
To configure Loggers

1. In Command Central, navigate to **Environments** > **Instances** > **All** > **API Gateway** > **Configuration**.
2. Select **Loggers** from the drop-down menu.

   This section displays components and their corresponding log levels.
3. Follow these steps to change the log level of a component:

   a. Click the required log file type from the list.

   b. Select the required **Log Level** from the drop-down list.



   c. Click **Save**.

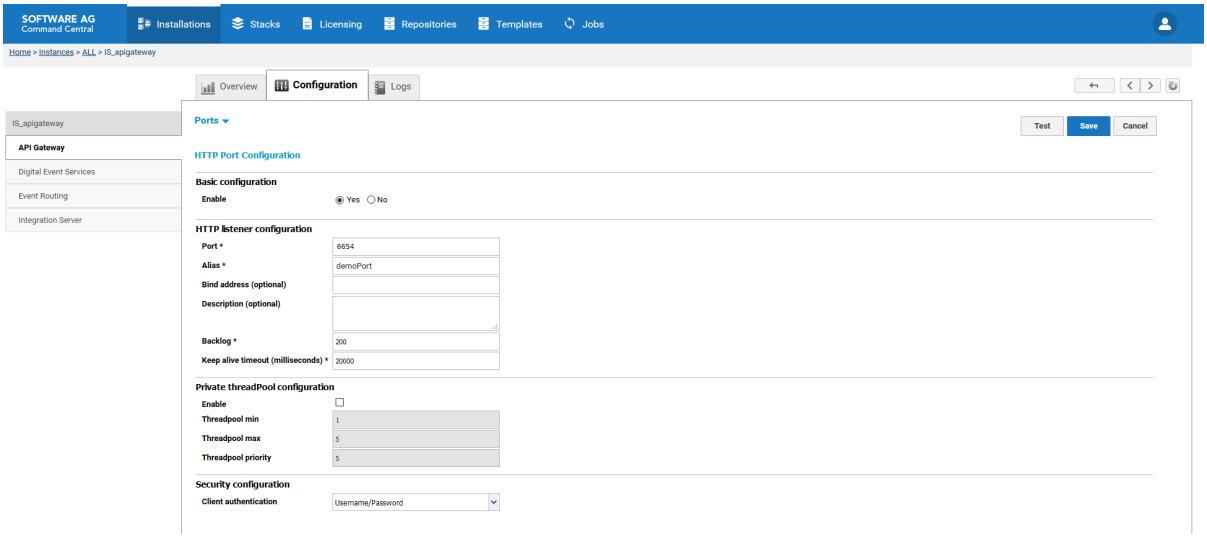
Configuring HTTP Port

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

To configure the HTTP port

1. In Command Central, navigate to **Environments** > **Instances** > **All** > **API Gateway** > **Configuration**.
2. Select **Ports** from the drop-down menu.
3. Click **HTTP Port Configuration**.
4. Select `Yes` in the **Enable** field in the **Basic configuration** section.

5. Provide valid port numbers in the **Port** and **Alias** field of the **HTTP listener configuration** section.



6. Optionally, click **Test** to verify your configuration.

7. Save your changes.

8. Restart the API Gateway instance.

   The port is created and enabled.

Configuring HTTPS Port

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

To configure the HTTPS port

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

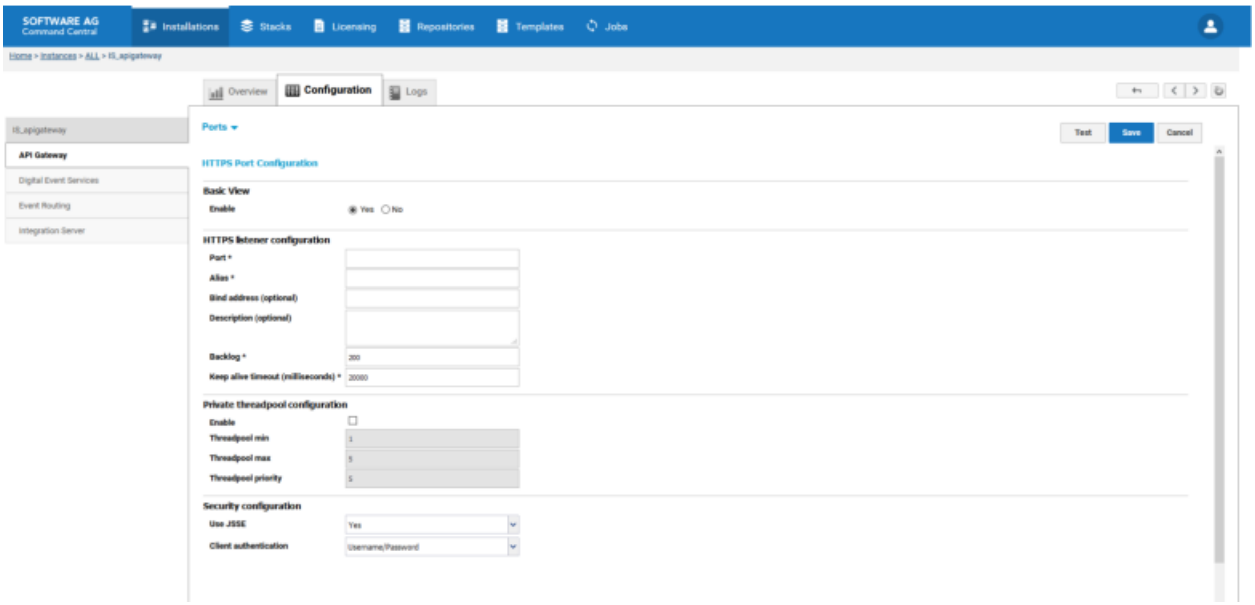
2. Select **Ports** from the drop-down menu.

3. Click **HTTPS Port Configuration**.

4. Select `Yes` in the **Enable** field in the **Basic configuration** section.

5. Provide valid port numbers in the **Port** and **Alias** field of the **HTTPS listener configuration** section.

6. Select the required Keystore and Truststore from the available list of options.



7. Optionally, click **Test** to verify your configuration.
8. Save your changes.
9. Restart the API Gateway instance.

   The port is created and enabled.

Configuring HTTPS Port using Template

You can configure port by using the following Command Central template:

```
sagcc exec templates composite import -i httpPort.yaml
sagcc exec templates composite apply httpPortAlias
```

Sample ports configuration template

```
alias: httpsPortAlias
description: API Gateway https port creation
layers:
  runtime:
    templates: httpsPort-Template
templates:
  httpsPort-Template:
    products:
      integrationServer:
        apigateway:
          configuration:
            OSGI-IS_apigateway-WmAPIGateway:
              COMMON-PORTS:
                COMMON_PORTS_HTTPS:
                  Port:
                    '@primary': 'false'
                    '@alias': HTTPS
```

```
                    Enabled: 'true'
                    CustomType: HTTPSListener@5558
                    Number: '5558'
                    Protocol: HTTPS
                    Backlog: '200'
                    KeepAliveTimeout: '20000'
                    ThreadPool:
                    SSL:
                      KeystoreAlias: pgkey
                      KeyAlias: partner2
                      TruststoreAlias: trust
                    ExtendedProperties:
                      Property:
                      - '@name': DIS_PORT
                        $: '5558'
                      - '@name': DIS_PORT_ALIAS
                        $: HTTPS
                      - '@name': DIS_PROTOCOL
                        $: HTTPS
                      - '@name': DIS_ENABLE
                        $: 'true'
                      - '@name': DIS_PRIMARY
                        $: 'false'
                      - '@name': 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,
comodoaaaca,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

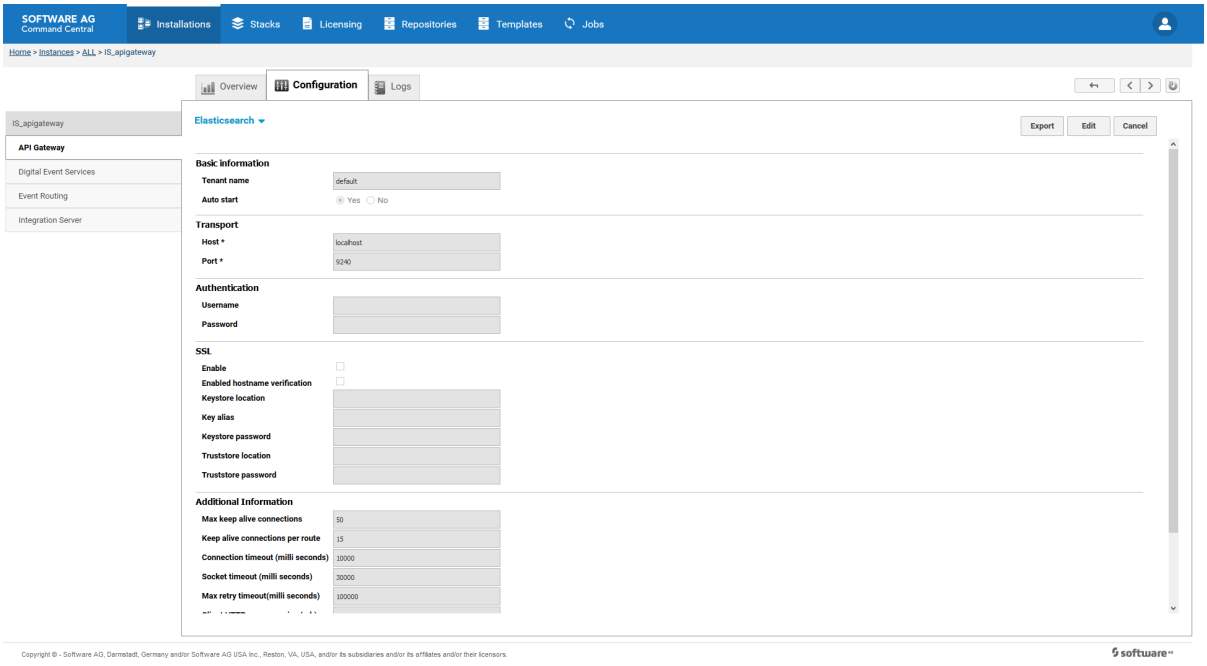
1. In Command Central, navigate to **Environments** > **Instances** > **All** > **API Gateway** > **Configuration**.
2. Click **Elasticsearch** from the drop-down menu. The Elasticsearch section appears.
3. Provide **Tenant name**.
4. Select one of the following values in the **Auto start** field:
   - `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.



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}
```

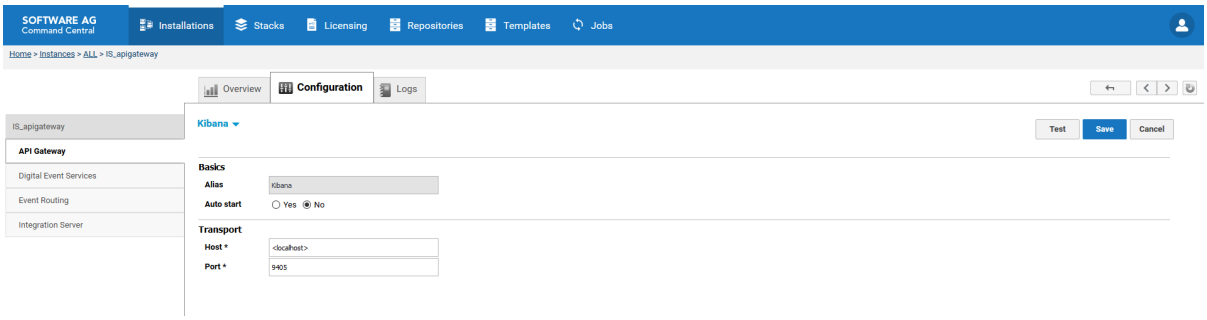
Configuring Kibana Connection Settings

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 provide any values in those fields.



6. Save your changes.

   The **Pending restart** value is changed to *true* and **Status** is *Enabled*.
7. Restart the API Gateway instance.

   The Kibana details are updated in API Gateway.

Configuring Kibana using Template

You can configure Kibana using the following Command Central template:

```
sagcc exec templates composite import -i cc-kibana.yaml
sagcc exec templates composite apply cc-kibana nodes=local host=hostname port=portnumber
```

Sample Kibana configuration template

```
alias: cc-kibana-alias
description: HTTPS elastic search template
layers:
  runtime:
    templates:
     - cc-kibana
templates:
  cc-kibana:
    products:
      integrationServer:
        default:
          configuration:
            OSGI-IS_apigateway-WmAPIGateway:
              APIGATEWAY-KIBANA:
                APIGATEWAY-KIBANA:
                  '@alias': Kibana
                  autostart: 'false'
                  Transport:
                    Host: ${host}
                    Port: ${port}


provision:
  default:
    runtime: ${nodes}
```

Configuring API Gateway Cluster

This section provides information about configuring cluster details for API Gateway in the API Gateway section.

Note:
Ensure that the Terracotta server is running when configuring cluster.

To configure API Gateway Clustering

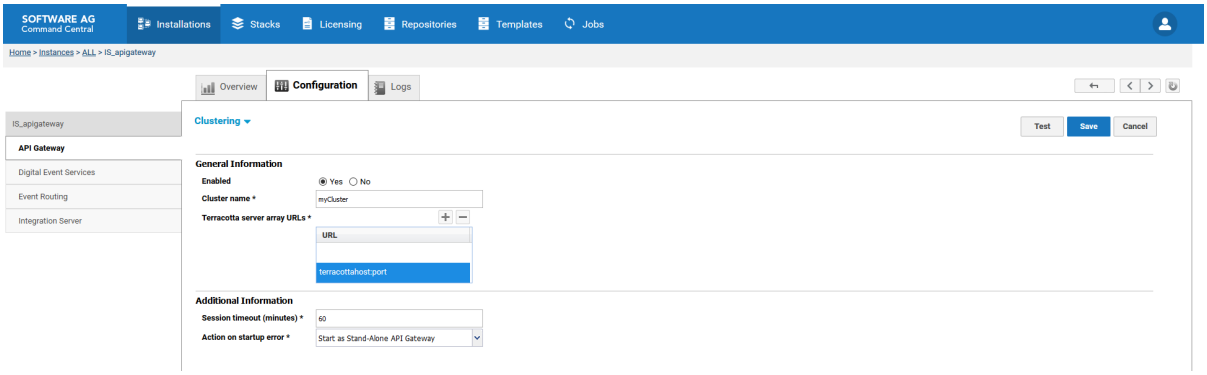
1. In Command Central, navigate to **Environments** > **Instances** > **All** > **API Gateway** > **Configuration**.

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

   The initial clustering status appears as *Disabled*.

3. Click **Disabled**. The **General Information** section appears.

4. Click **Edit** to provide the cluster details.



5. Select *Yes* in the **Enable** field.

6. Provide **Cluster name**.

7. Provide the host name and port of the server where Terracotta is running, in the **Terracotta server array URLs** field.

8. Optionally, click **Test** to verify your configuration.

9. Save your changes.

   The **Pending restart** value is changed to *true* and **Status** is *Enabled*.

10. Restart the API Gateway instance.

    The clustering details are updated in API Gateway.

Configuring Cluster using Template

You can configure Cluster using the following Command Central template:

```
sagcc exec templates composite import -i cc-clustering.yaml
sagcc exec templates composite apply commandcentral-clustering-alias nodes=local
tchost=terracotta_host tcport=terracotta_port
```

Sample clustering configuration template

```
alias: cc-clustering-alias
description: cluster config
layers:
  runtime:
    templates:
     - cc-clustering
templates:
  cc-clustering:
    products:
      integrationServer:
        default:
          configuration:
            OSGI-IS_apigateway-WmAPIGateway:
```

```
              COMMON-CLUSTER:
                COMMON-CLUSTER:
                  Enabled: 'true'
                  Name: APIGatewayTSAcluster
                  Servers:
                    Server:
                      URL: daeirnd33974:9510
                  ExtendedProperties:
                    Property:
                    - '@name': SessionTimeout
                      $: '60'
                    - '@name': ActionOnStartupError
                      $: standalone
provision:
  default:
    runtime: ${nodes}
```

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. Default: centos:7
`-Dinstance.name`	*Optional*. IS instance name to include in the image. Default: default
`-Dport.list`	*Optional*. Comma-separated list of the ports on the instance to expose in the image. Default: 5555,9999
`-Dpackage.list`	*Optional*. Comma-separated list of Wm packages on the instance to include in the image. Default: all (this includes all the Wm packages and the Default package)
`-Dinclude.jdk`	*Optional*. Whether to include the Integration Server JDK (true) or JRE (false) in the image. Default: true
`-Dfile.name`	*Optional*. File name for the generated docker file. Default: Dockerfile_IS

2. Build the IS Docker image using the Docker file Dockerfile_IS by running the following command:

```
./is_container.sh build [optional arguments]
```

Argument	Description
`-Dfile.name`	*Optional*. File name of the Docker file to use to build the Docker image. Default: Dockerfile_IS
`-Dimage.name`	*Optional*. Name of the generated Docker image. Default: is:micro

3. Create a Docker file for the API Gateway instance from the IS image is:micro by running the following command:

```
./apigw_container.sh createDockerfile [optional arguments]
```

Argument	Description
`--instance.name`	*Optional*. API Gateway instance to include in the image. Default: default
`--port.list`	Comma-separated list of the ports on the instance to expose in the image. Default: 9072
`--base.image`	Name of the base Integration Server image upon which this image should be built. Default: is:micro
`--file.name`	*Optional*. File name for the generated Docker file. Default: Dockerfile_IS_APIGW
`--target.configuration`	*Optional*. Target configuration for which Dockerfile is created. Not specifying any value builds a Dockerfile for the Docker and Kubernetes environments. Specifying the value `OpenShift` builds a Dockerfile for an OpenShift environment. **Note:** If you specify the --target.configuration option, the Integration Server image specified by the --base.image option should be available before you create the API Gateway Dockerfile. The Integration Server Docker image is analyzed with docker inspect in order to extract some information necessary for the API Gateway Dockerfile.

Argument	Description
`--os.image`	*Optional*. Name of the base operating system image upon which this image is built if the `--target.configuration` is set to `OpenShift`. Default: centos:7 **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. Default: default
`file.name`	File name of the Docker file to use to build the Docker image. Default: Dockerfile_IS_APIGW
`image.name`	Optional. Name for the generated Docker image that contains the custom packages. 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

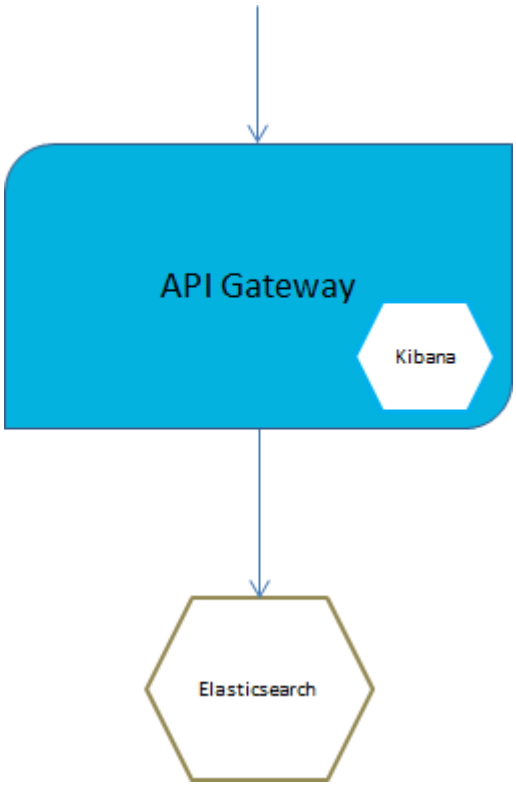
- `saveImage`: To save an API Gateway image to a file (creating a tar ball from an image)
- `loadImage`: To load an image to a Docker registry (loading an image into a Docker registry from tar ball)

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:7.7.1
   ```

   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=
   ```

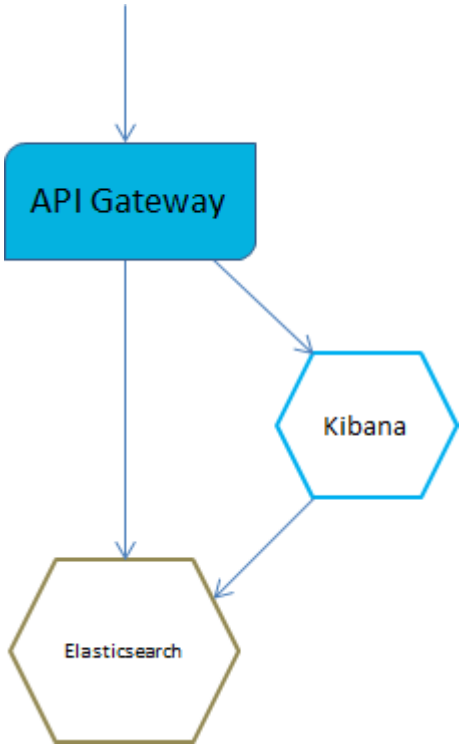
   You can specify the Elasticsearch properties to modify the property files on the container startup.

   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:7.7.1
   ```

   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 51.

2. Configure Terracotta Server Array.

   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 179.

4. Create cluster API Gateway Docker image and enhance it with the cluster configuration in one of the following ways:

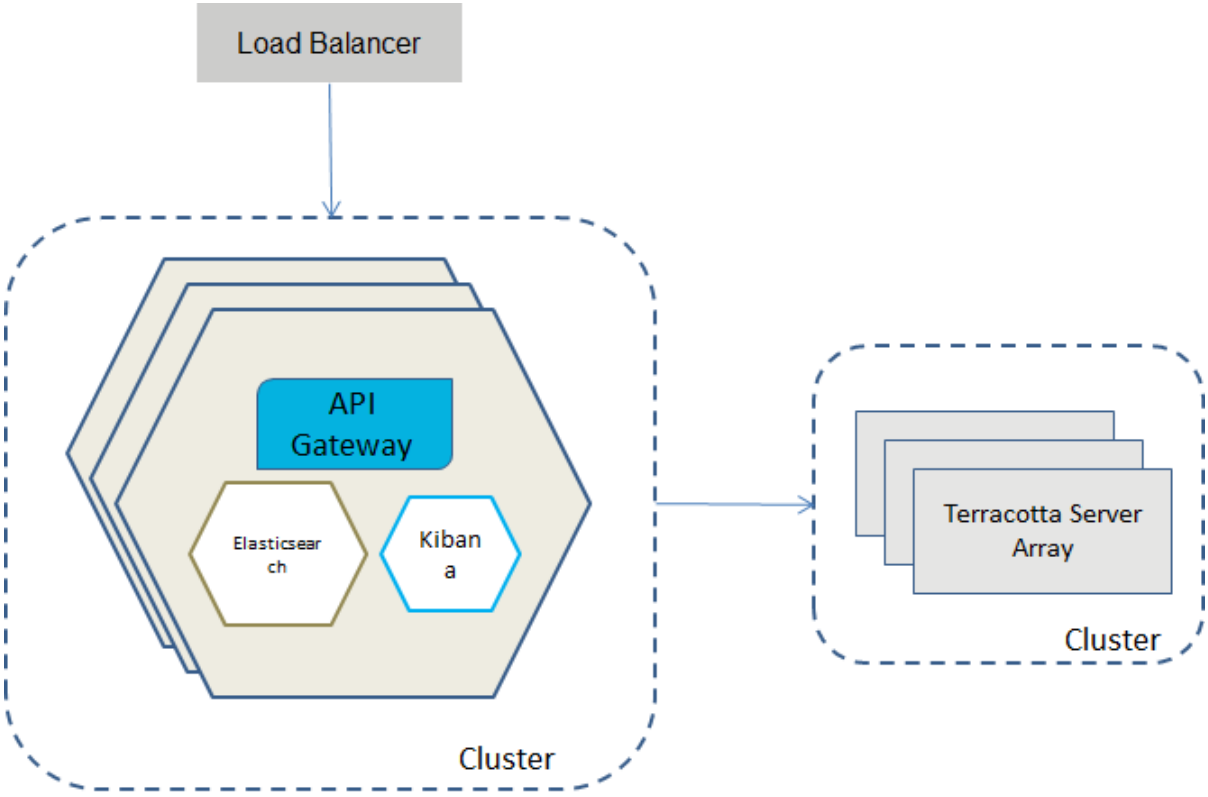
   - Clustered all-in-one containers that consist of API Gateway, Elasticsearch, and Kibana.
   - Clustered API Gateway containers with externalized Elasticsearch and Kibana containers.

Clustered all-in-one Containers that consist of API Gateway, Kibana and Elasticsearch

Although API Gateway clusters with externalized Elasticsearch is the preferred approach API Gateway all-in-one containers can also be clustered.

Note:
Having external Kibana is an optional variation.

The following diagram depicts clustering based on all-in-one containers.



The all-in-one containers hold API Gateway, Kibana and Elasticsearch. The clustering is done through a Terracotta Server Array and the cluster capabilities of the embedded Elasticsearch instances.

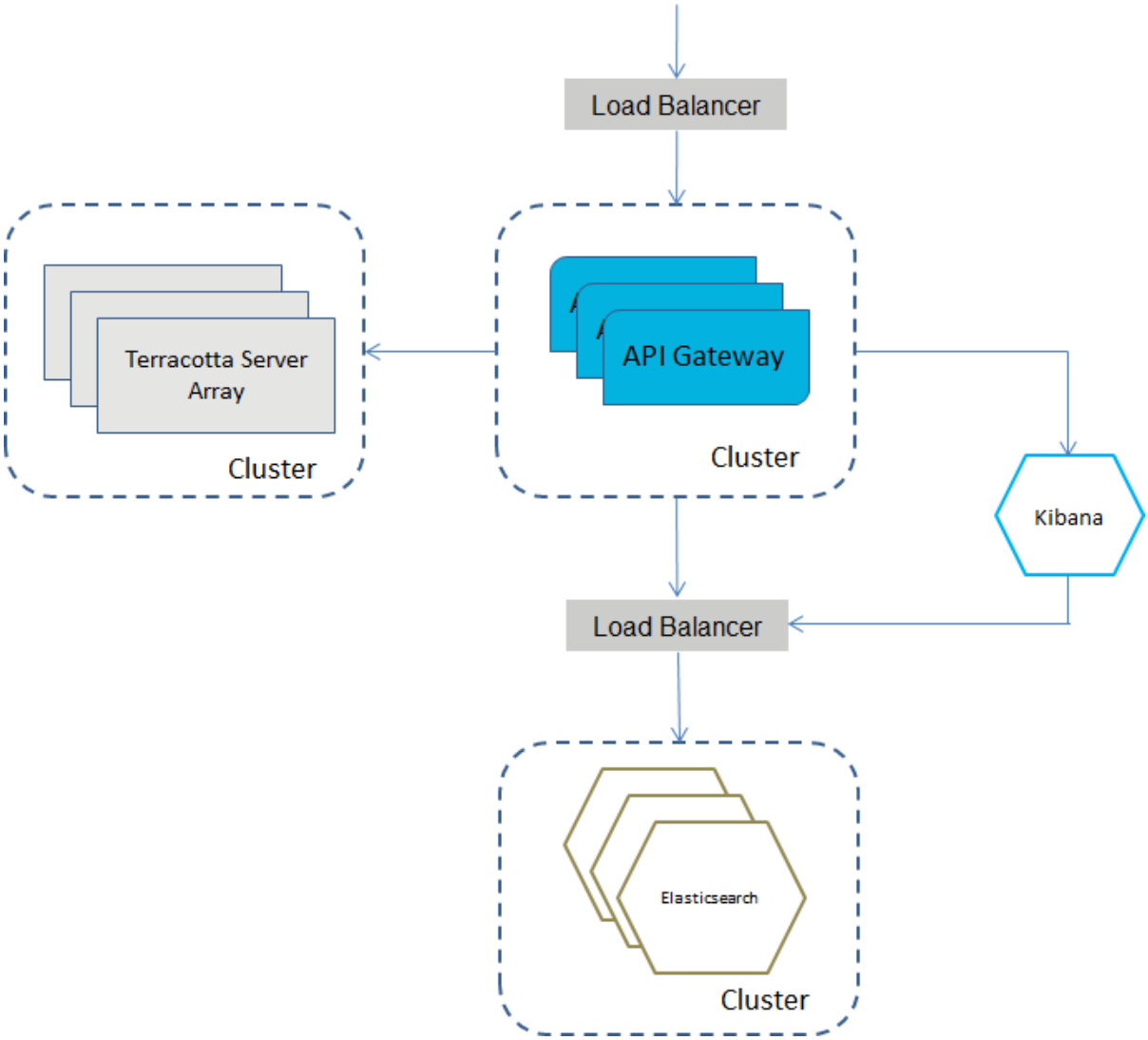
The required settings for the cluster configuration can be injected during Docker run through an environment file. A sample environment file looks as follows.

```
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 containers are clustered. They are talking to a clustered Terracotta Server Array container and to a cluster of Elasticsearch container through a loadbalancer. The Elasticsearch loadbalancer is also providing the Elasticsearch endpoint for the Kibana containers.

Note:
The externalized Kibana is optional. You can still run Kibana within the API Gateway container.



To cluster the API Gateway with external containers for Elasticsearch, Kibana, and Terracotta Server Array, the settings can be injected into an API Gateway Docker image when starting by proving an environment file. The environment file needs 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:

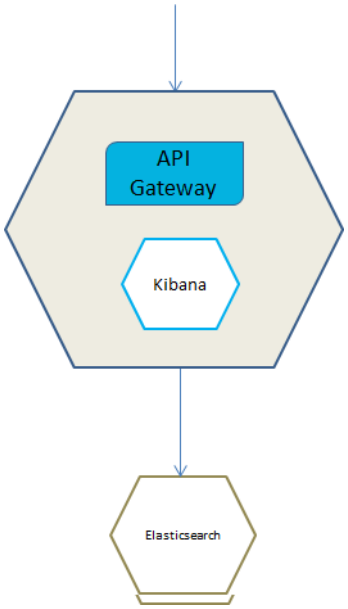
- **apigw-elasticsearch-no-cluster.yml** : An API Gateway instance with an Elasticsearch container.
- **apigw-elasticsearch-cluster.yml** : An API Gateway cluster with three API Gateway containers, three clustered Elasticsearch containers and a Terracotta container.
- **apigw-elasticsearch-cluster-kibana.yml** : Containers of an API Gateway cluster and a Kibana container.

The Docker Compose files can be parameterized through environment variables.

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 179. The Docker Compose file references the standard Elasticsearch 7.7.1image: docker.elastic.co/elasticsearch/elasticsearch:7.7.1

   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:

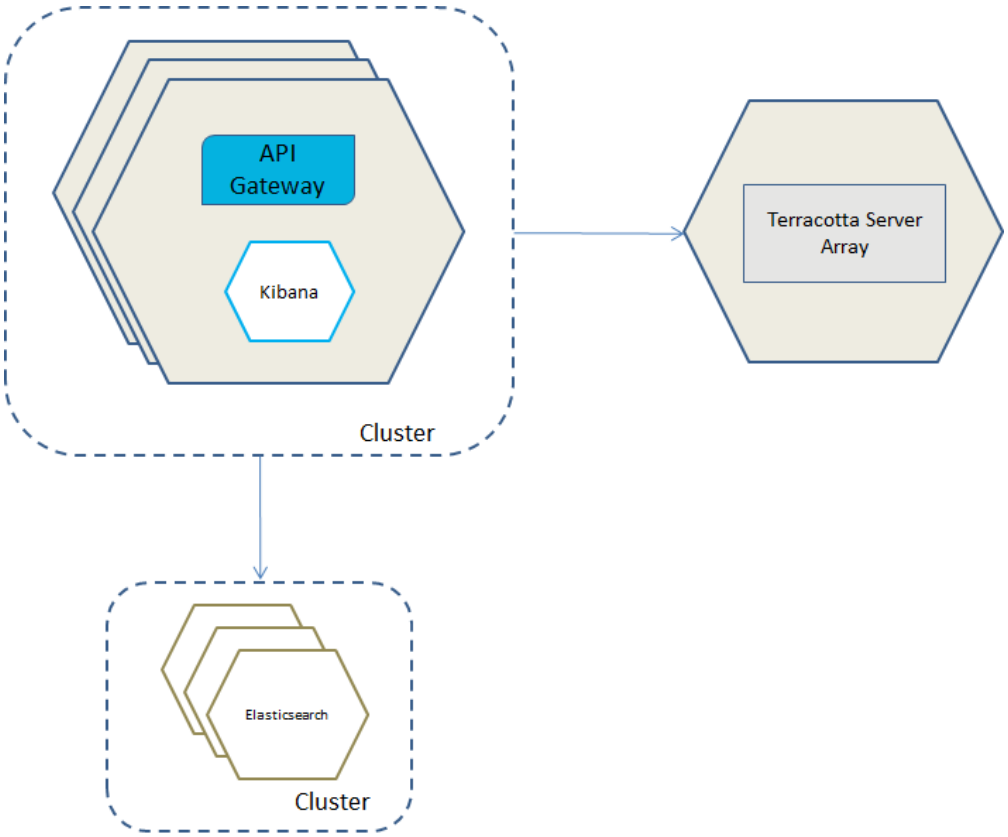
You can stop the API Gateway Docker container and the Elasticsearch container using the Docker Compose sample file with the following command:

```
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 179.

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.

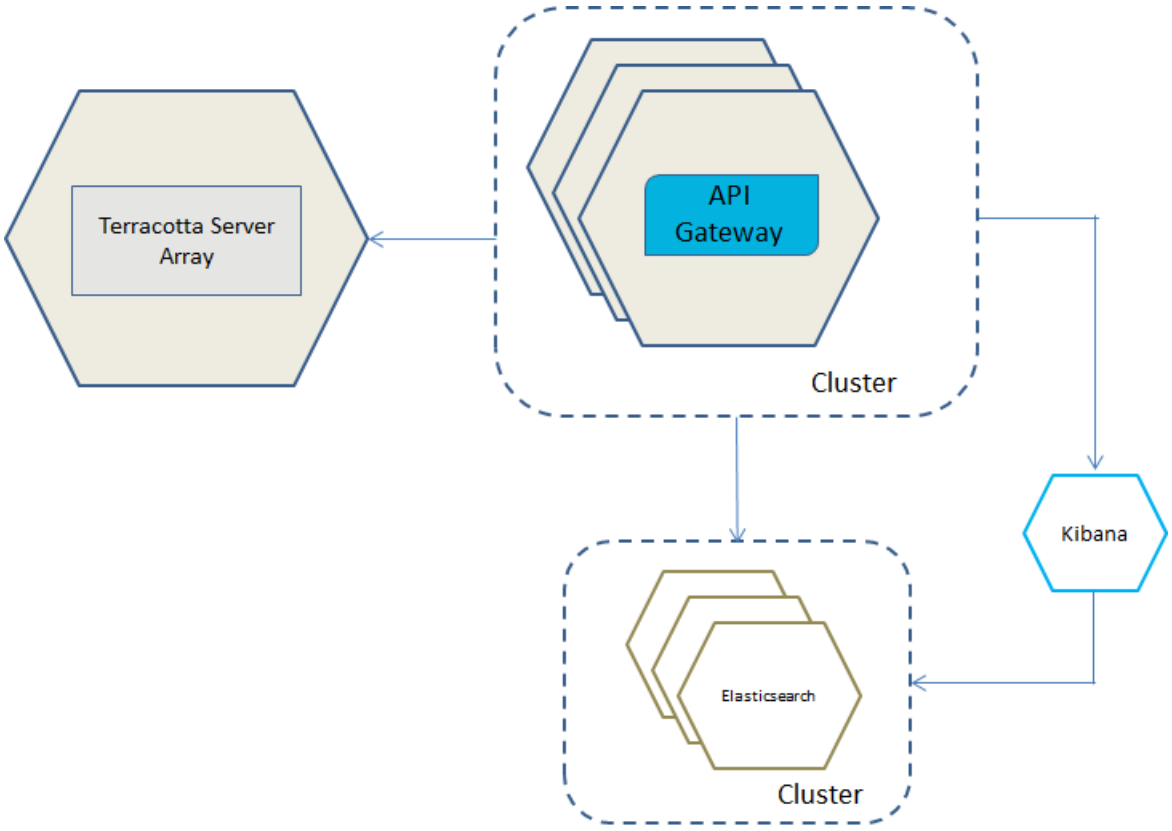
Note:
You can stop the API Gateway Docker container and the Elasticsearch container using the Docker Compose sample file with the following command:

```
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 179.

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
```

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:

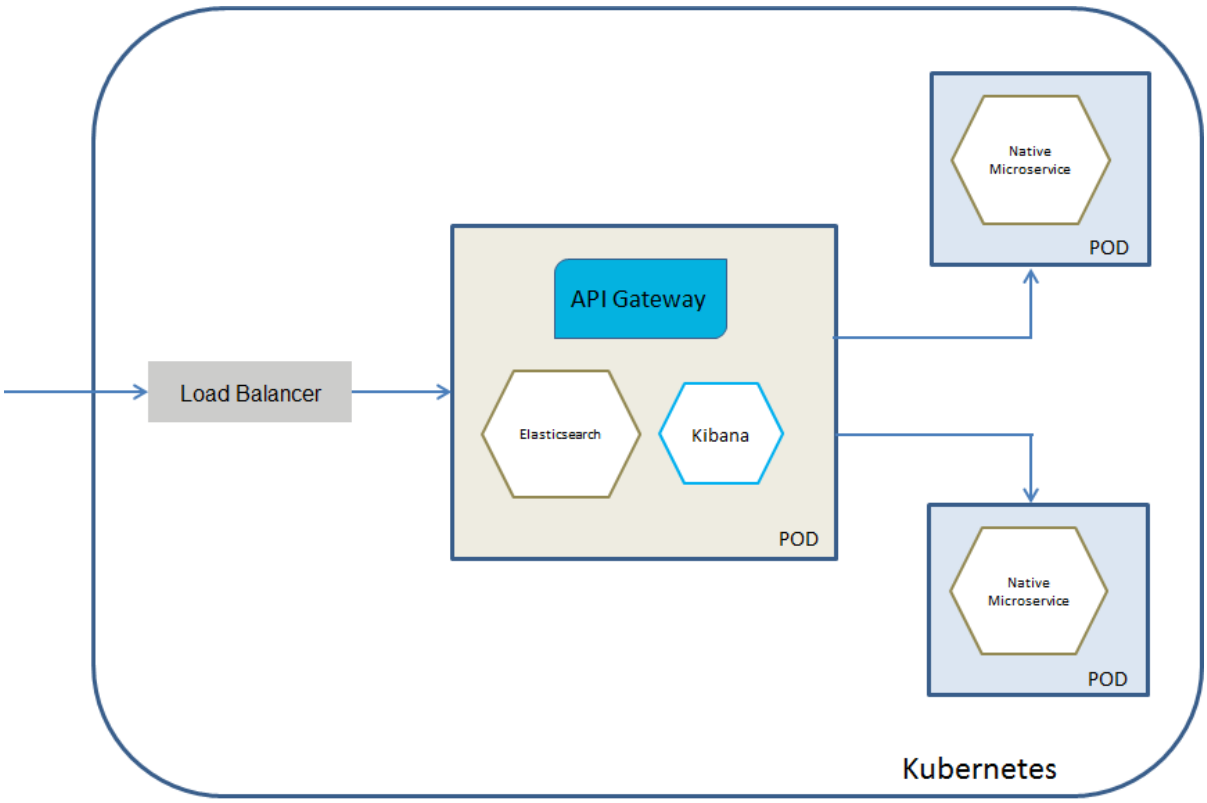
- "Building the Docker Image for an API Gateway Instance " on page 179, in particular the --target.configuration and --os.image parameters
- "OpenShift Support" on page 203

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.

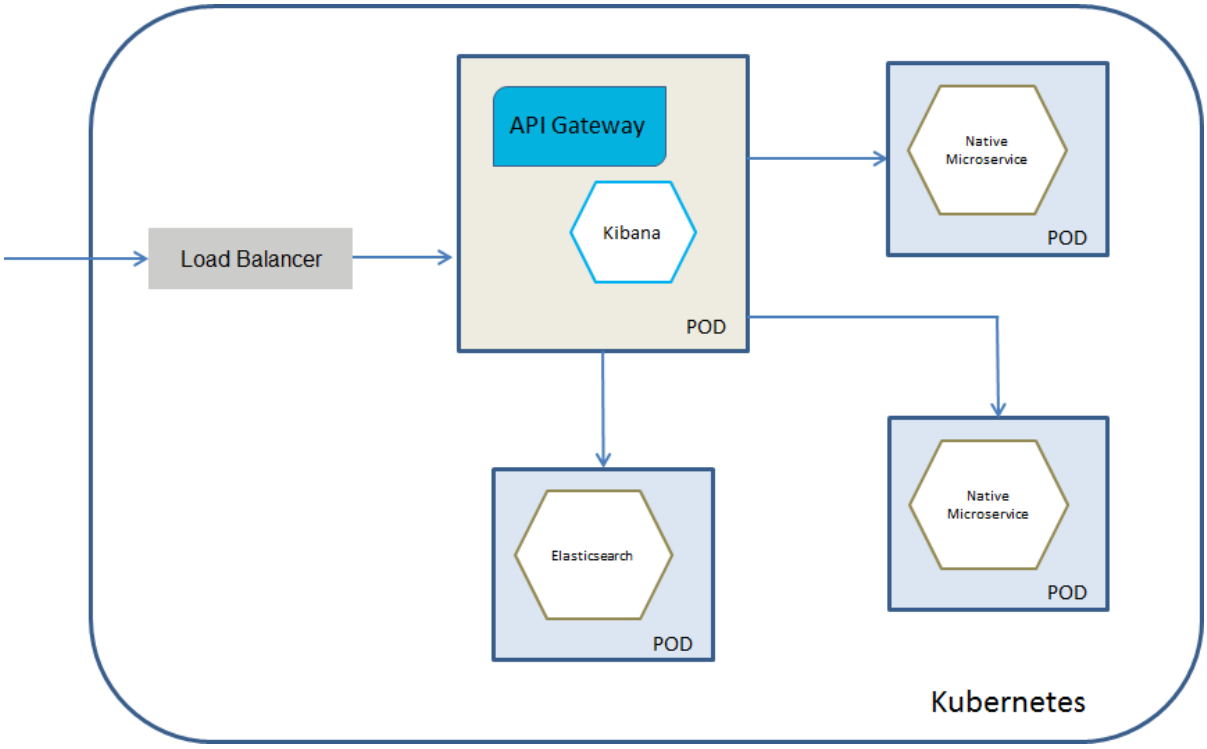
   This now pulls the image specified and creates the API Gateway pod with API Gateway and Elasticsearch containers.

   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 7.7.1.

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 7.7.1.

   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 located at *SAG_Root*/IntegrationServer/instances/default/packages/WmAPIGateway/resources/samples/K8s.

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 206.

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 204
- "Running the Docker Image With the sagadmin user " on page 205

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 179.

   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 206 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 204.

OpenShift Sample Files

API Gateway installation provides OpenShift deployment samples located at *SAG_Root*/`IntegrationServer/instances/default/packages/WmAPIGateway/resources/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 202 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:10.7
---
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>:10.7
    importPolicy: {}
    name: "10.7"
    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_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_instance_name/apigateway/config/uiconfiguration.properties`

apigw.kibana.autostart

Specifies whether Kibana should be started as part of web-app.

Possible values: `true`, `false`.

Default value is `true`.

apigw.kibana.url

Denotes the URL where Kibana is running. *localhost* is replaced by the hostname that is resolved through localhost. The port and other configurations of the Kibana can be changed from `SAG_root/profiles/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 Gateway 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 Gateway 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.

kibana.process.stop.signal.number

Specifies the signal number to be used when stopping the Kibana process.

The default signal number is *SIGINT*(2).

SIGINT(2) stops the Kibana process without producing a core dump. This property is applicable only for Linux Operating System. For information about the signals, see https://www.linux.org/threads/kill-commands-and-signals.8881/.

API Gateway Package Configuration Properties

API Gateway uses API Gateway Data Store (Elasticsearch) as its data repository. API Gateway starts the API Gateway Data Store instance, if configured, using the default configuration shipped and located at *SAG_root*`/InternalDataStore/config/elasticsearch.yml`

Note:
To run API Gateway 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 Gateway Data Store as part of API Gateway. Set it to false if the start or stop of API Gateway 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 Gateway 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 Gateway 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 Gateway 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.

Troubleshooting Tips: wmAPIGateway Package

Error appears when WmAPIGateway is added as Integration Server package dependency

The following error message appears when you add **wmAPIGateway** as a package dependency for an Integration Server package using Designer, and compile the package.

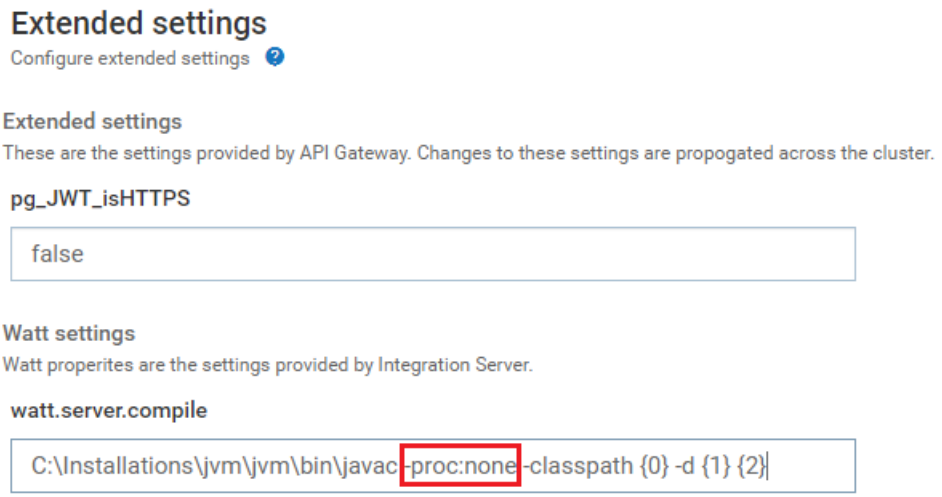
error: Bad service configuration file, or exception thrown while constructing Processor object: io/sundr/codegen/processor/JavaGeneratingProcessor

Resolution:

1. Expand the menu options icon  , in the title bar, and select **Administration**.
2. Select **General > Extended settings**.
3. Click **Show and hide keys**. This displays all the configurable parameters.
4. Search and enable the **watt.server.compile** parameter checkbox.
5. Add `-proc:none` to the **watt.server.compile** parameter.

   For example,

   ```
   watt.server.compile=/opt/SwAG/API/jvm/jvm/bin/javac -proc:none
       -classpath {0} -d {1} {2}
   ```

6. Click **Save**.

9 API Gateway Data Management

Data Backup and Restore

You can take regular backups of the internal database where API Gateway data is stored to protect against accidental data loss. You can take a backup of complete API Gateway data that includes analytics data and assets data or you can take a partial backup that includes the backup of assets data or backup of analytics data. When you take a backup, you copy the contents of the repository to a file or to a cloud storage. At a later stage, you can retrieve the contents of the backup and restore them to API Gateway.

Note:

- API Gateway supports incremental backup. For example, if you have taken a backup of 50 GB and there is an increase in backup to 52 GB, API Gateway takes a backup of the new 2 GB data added.
- While performing a backup, the database experiences additional load, therefore, Software AG recommends taking a backup when the usage is low so as to avoid performance degradation.
- While restoring the backup from the repository, API Gateway replaces the existing data in API Gateway.
- API Gateway is not accessible when database restore is in progress.

To take a complete or partial backup of the API Gateway data and restore it to API Gateway, you can use the API Gateway command line utility. To back up and restore the database in command line, use the `apigatewayUtil.bat` and the `apigatewayUtil.sh` files available in the *`Integration Server_directory`*`/instances/`*`instance_id`*`/packages/WmAPIGateway/cli/bin` folder for Windows and UNIX systems respectively.

API Gateway supports the following storage platforms:

- Network File System (NFS)
- Amazon Simple Storage Service (S3)

  Note:
  You must install the Amazon Web Services (AWS) cloud plugin if you want to use the Amazon S3 storage platform. To install the AWS cloud plugin, run the following command and restart Integration Server: *Integration Server_directory* `InternalDataStore/bin/elasticsearch-plugin install repository-s3`

Using NFS Storage Platform

API Gateway uses NFS as the default repository in which the backup is stored. You can configure the repositories in which the backup is stored either in NFS or Amazon S3 cloud. However, you can create a single repository and place all the backup files in that repository.

- Taking a backup:

  By default, API Gateway stores the backup in the *`Integration Server_directory`*`/InternalDataStore/archives/` directory. For example, if you run the command, `apigatewayUtil.sh create backup -name` *`backup_file_name`*, to take a backup, the backup is saved in the *`Integration Server_directory`*`/InternalDataStore/archives/default` directory.

- Restoring a backup:

  To restore the data taken as backup to API Gateway, run the following command:
  `apigatewayUtil.sh restore backup -name backup_file_name`

  Note:
  Once the backup is restored, you must restart the API Gateway instance.

- Restoring backup to a new instance:

  1. Copy the data from *Integration Server_directory*`/InternalDataStore/archives/` directory where the backup data is available.
  2. Go to the *Integration Server_directory*`/InternalDataStore/archives/` directory where the backup data is to be restored and ensure that you delete any existing data in this directory.
  3. Paste the data in the *Integration Server_directory*`/InternalDataStore/archives/` directory.
  4. Run the following command to restore the data: `apigatewayUtil.sh restore backup -name backup_file_name`

     Note:
     Once the backup is restored, you must restart the API Gateway instance.

- Specifying NFS Directory path:

  For API Gateways in a clustered environment, you must specify a NFS directory path. This directory path is a shared file location, which must be accessible to all the API Gateway nodes in the cluster to take a backup and restore the backup files.

  1. Configure the NFS directory path before creating the NFS repository in Elasticsearch by running the following command: `apigatewayUtil.sh configure fs_path -path c://sample//APIGATEWAY`
  2. Restart Integration Server to make the new NFS directory path available to store the backup, else the backup is stored in the default location.

Using Amazon S3 Storage Platform

You can save your backups to Amazon S3 cloud.

- Creating a repository:

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

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

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

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

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

   Example:

   ```
   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:

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

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

   Example:

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

   Important:
   You must restart Elasticsearch once you provide the secret key.

7. Run the following command:

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

   where *`file_path`* is the path where the Amazon S3 cloud details are specified.

   For example, `apigatewayUtil.sh configure manageRepo -file` Integration Server_directory/instances/instance_id/packages/WmAPIGateway/cli/bin/conf/gateway-s3-repo.cnf.

8. Go to *`Integration Server_directory`*`/instances/`*`instance_id`*`/packages/WmAPIGateway/cli/bin/conf`.

9. Open the `gateway-s3-repo.cnf` file.

10. Configure the Amazon S3 details in the `gateway-s3-repo.cnf` file.

    After modifying the `gateway-s3-repo.cnf` file, run the following command:
    `apigatewayUtil.sh configure manageRepo -file <file_path>`

    For example, `apigatewayUtil.sh configure manageRepo -file` *`Integration Server_directory`*`/instances/`*`instance_id`*`/packages/WmAPIGateway/cli/bin/conf/gateway-s3-repo.cnf`

11. To enable encryption of the backup data, set the `server_side_encryption` property to *true*. The backup files saved in the backup files in the S3 repository are encrypted using the AES256 algorithm.

12. To complete the repository creation process, restart your Elasticsearch and Integration Service instances.

- Taking a backup:

  To take a backup of the data, run the following command: `apigatewayUtil.sh create backup -name` *`backup_file_name`*

Note:
The *backup_file_name* must be specified in lowercase.

- Restoring a backup:

  To restore the data taken as backup to API Gateway, run the following command:
  `apigatewayUtil.sh restore backup -name backup_file_name`

  Note:
  Once the backup is restored, you must restart the API Gateway instance.

- Restoring backup to a new instance:

  1. Create a repository using Amazon S3 if not already created.

     Note:
     The Amazon S3 details which you provide in the `gateway-s3-repo.cnf` should point to the location where you have the backup files which were taken earlier.

  2. In case of multiple backups, run the following command to retrieve a list of backups:
     `apigatewayUtil.sh list backup`

  3. Run the following command to restore the data using the required backup file:
     `apigatewayUtil.sh restore backup -name backup_file_name`

  4. Restart API Gateway.

     Note:
     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.

API Gateway Backup and Restore Commands

You can use a command-line interface (CLI) script to back up data that is stored on API Gateway Data Store. You can use the CLI script to restore database after a data failure or hardware failure on the API Gateway instance.

In a command line, go to `<Integration Server_directory>\instances\default\packages\WmAPIGateway\cli\bin` and run the following commands to take a database backup or restore the database from a backup:

If you want to...	Command
Backup data	`apigatewayUtil.sh create backup -name <backupFilename>`

If you want to...	Command
Backup custom data	`apigatewayUtil.sh create backup -name <backupFile_name> -include <reference name>` Possible values for the parameter `reference name`: ■ `analytics` - to back up analytical data. ■ `assets` - to back up asset data.
Delete the backed up data	`apigatewayUtil.sh delete backup -name <backupFile_name>`
Restore the backed up data	`apigatewayUtil.sh restore backup -name <backupFile_name>`
To retrieve all available backup files in the repository	`apigatewayUtil.sh list backup`
Delete a repository from API Gateway	`apigatewayUtil.sh delete manageRepo`
To retrieve all available repositories	`apigatewayUtil.sh list manageRepo`
To configure a repository in S3	`apigatewayUtil.sh configure manageRepo -file <file_path>`

Pre-requisites for Backing up and Restoring Data

The following points are to be considered in the API Gateway instances used for backup and restore:

- The Software AG root installation directory must be the same.
- The Integration Server instance name must be the same.
- The ports defined for the API Gateway webApp, Integration Server, and the API Gateway Data Store must be the same.

Backing up API Gateway Configuration Data

You can back up the API Gateway configuration information and data. At a later stage, you can restore the API Gateway Data Store from the backup archive.

The configuration backups are performed using the following commands:

- `apigw-backup-tenant.bat` - Windows.
- `apigw-backup-tenant.sh` - Linux.

This command creates a backup archive of the API Gateway configuration information and data. It is typically used in the disaster recovery scenarios to backup the data periodically and restore the data in event of any disaster.

The default location of the backed up data is the *Integration Server_directory*/InternalDataStore/archives directory. You can write the backed up data to the InternalDataStore/archives folder mount from an external NFS or S3 service. API Gateway uses NFS as the default repository in which the backup is stored.

Note:
The following scripts will not be available from release above 10.7; and you must use the apigatewayUtil.bat (Windows) or apigatewayUtil.sh (Linux) scripts instead of these scripts:

Operating system	Scripts
Windows	apigw-backup-packages.bat, apigw-backup-tenant.bat, and apigw-upgrade-backup.bat
Linux	apigw-backup-packages.sh, apigw-backup-tenant.sh, and apigw-upgrade-backup.sh

Pre-requisites for Backing up in a Distributed Environment

The following points are to be considered if API Gateway is installed in a clustered high availability setup:

- Configure a path to backup the API Gateway Data Store.
- Restart the API Gateway Data Store.

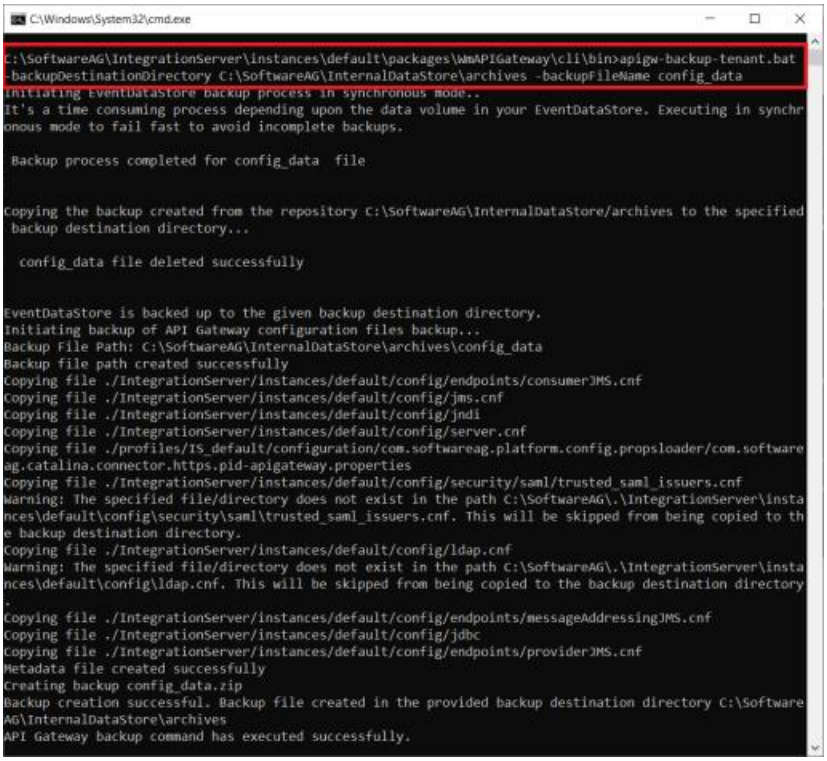
To backup configurations and data

- Run one of the following commands depending on your operating system:
  - **Windows** -

```
C:/SoftwareAG/IntegrationServer/instances/default/packages/WmAPIGateway/cli/bin>
apigw-backup-tenant.bat -backupDestinationDirectory
directory_path_to_store_backup_file
-backupFileName backup_file_name_without_spaces
-backupTemplate file_path_to_backup_template
-packagesTemplate file_path_to_packages_template
-help
```

  - **Linux** -

```
C:/SoftwareAG/IntegrationServer/instances/default/packages/WmAPIGateway/cli/bin>
apigw-backup-tenant.sh -backupDestinationDirectory
directory_path_to_store_backup_file
-backupFileName backup_file_name_without_spaces
-backupTemplate file_path_to_backup_template
-packagesTemplate file_path_to_packages_template
-help
```



In the example, the backup command is run with the *-backupDestinationDirectory* and *-backupFileName* parameters.

The input parameters are:

Parameter	Description
`-backupDestinationDirectory`	*Mandatory.* Path to the destination folder where you want to create the backup. Example: `C:/SoftwareAG/AnotherBackupLocation` Specify this option only if you want to create the backup in a local directory or NFS itself.
`-backupFileName`	*Optional.* Name of the file for the data backup. **Note:** The file name must not contain any spaces. Default value is `apigw_disaster_recovery_backup` If a file name is not specified, the default value is automatically set for this parameter.
`-backupTemplate`	*Optional.* Name of the backup template along with its location.

Parameter	Description
	Example: `C:/SoftwareAG/tenant-backup-template.txt` **Note:** The paths specified in the backup file should be relative to the Software AG root installation folder `<SAGInstallDir>` Default value is C:/SoftwareAG/IntegrationServer/instances/*instance_name*/packages/WmAPIGateway/cli/bin/conf/tenant-backup-template.txt. The backup file (`tenant-backup-template.txt`) contains a list of configuration files and folders that need to be backed up, with one file or folder name in each line. The backup file can also include custom configuration files, that are defined specifically for a particular API Gateway instance.
`-packagesTemplate`	*Optional.* Name of the backup packages template along with its location. This file contains a list of custom packages to be backed up and includes one package name in each line. Default value is `conf/tenant-backup-packages.txt`
`-help`	*Optional.* Prints the help text summarizing the input parameters of this command.

The `apigw-backup-tenant` command creates the following entries in `-backupDestinationDirectory`:

- A ZIP file with the name specified for the parameter `-backupFileName`. This ZIP file contains the backup of API Gateway configurations.

  If a file name is not specified in this parameter, then the command creates a ZIP file named `apigw_disaster_recovery_backup.zip`.

- A folder named, `default`, within the zip file. This folder contains the backup of API Gateway configuration data.

The backup command also creates a backup log file named `backup-tenant.log` in the *SAGInstallDir*`/IntegrationServer/instances/`*instance_name*`/packages/WmAPIGateway/cli/bin` directory.

Restoring API Gateway Configuration Data

You use the command tool `apigw-restore-tenant.[bat|sh]` to restore previously archived configuration files and data on an API Gateway instance.

Restoring overwrites the existing content in your API Gateway instance.

Note:
The `apigw-restore-tenant.bat` (Windows) and `apigw-restore-tenant.sh` scripts will not be available in releases above 10.7. You must use the `apigatewayUtil.bat` (Windows) or `apigatewayUtil.sh` (Linux) scripts instead of the apigw-restore-tenant script.

Pre-requisites for Restoring in a Distributed Environment

The following points are to be considered if API Gateway is installed in a clustered high availability setup:

- The API Gateway Data Store must be active in only a single node in the cluster.
- The API Gateway instance should be up and running.

To restore configurations and data

- Run the command `apigw-restore-tenant.sh`

  The syntax is of the format: `C:/SoftwareAG/IntegrationServer/instances/default/packages/WmAPIGateway/cli/bin>apigw-restore-tenant./sh -backupFileName <backup_file_name_without_spaces> -backupDestinationDirectory <directory_path_to_store_backup_file> -filesToSkip <file_path_to_files_to_skip> -skipDataRestore -help`

  The input parameters are:

Parameter	Description
`-backupDestinationDirectory`	*Mandatory.* Path to the destination folder where the backup is available. Example: `C:\SoftwareAG\AnotherBackupLocation` Specify this option only if you want the backup is available in a local directory or NFS itself.
`-backupFileName`	*Mandatory.* Name of the file for the data backup. **Note:** The file name must not contain any spaces.
`-filesToSkip`	*Optional.* Path to the data restore file.

Parameter	Description

	Example:
	`C:\SoftwareAG\skip-files.txt`
	Note:
	The paths specified in the restore file should be relative to the Software AG root installation folder `<SAGInstallDir>`
	Default value is
	`C:\SoftwareAG\IntegrationServer\instances\{instance_name}\packages\WmAPIGateway\cli\bin\conf\skip-files.txt`
	The restore file (`skip-files.txt`) contains a list of configuration files and folders that need to be restored in the API Gateway instance, with one file or folder name in each line. If you do not want to restore a specific configuration file, you can remove it from this restore file.

`-skipDataRestore`	*Optional.* Skips restoring of the API Gateway Data Store (Elasticsearch) data.
`-help`	*Optional.* Prints the help text summarizing the input parameters of this command.

The `apigw-restore-tenant` command creates a restore log file named `restore-tenant.log` in the `<SAGInstallDir>\IntegrationServer\instances\{instance_name}\packages\WmAPIGateway\cli\bin` directory.

Note:
The `apigw-restore-tenant` command automatically restarts the API Gateway instance.

Post-requisites in a Distributed Environment

- If the API Gateway Data Store in active node does not start automatically, manually restart the API Gateway Data Store.
- Start the API Gateway Data Store in all other nodes in the cluster. This is important to synchronize the data on a restored API Gateway instance with all other nodes in the cluster.
- The API Gateway configuration files should be restored separately for each individual API Gateway instance in the clustered environment.

  Run the command `apigw-restore-tenant` with the parameter `-skipDataRestore`. This restores the API Gateway configuration files without restoring the API Gateway Data Store data in all other nodes in the cluster.

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.

  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 business operation to resume after a disaster. It also specifies how often you should back up data.
  - **Recovery Time Objective (RTO)**. Describes the duration and service level within which you must restore the most critical IT services after a disaster.
- **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:

- Cold Standby
- Warm Standby
- 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.

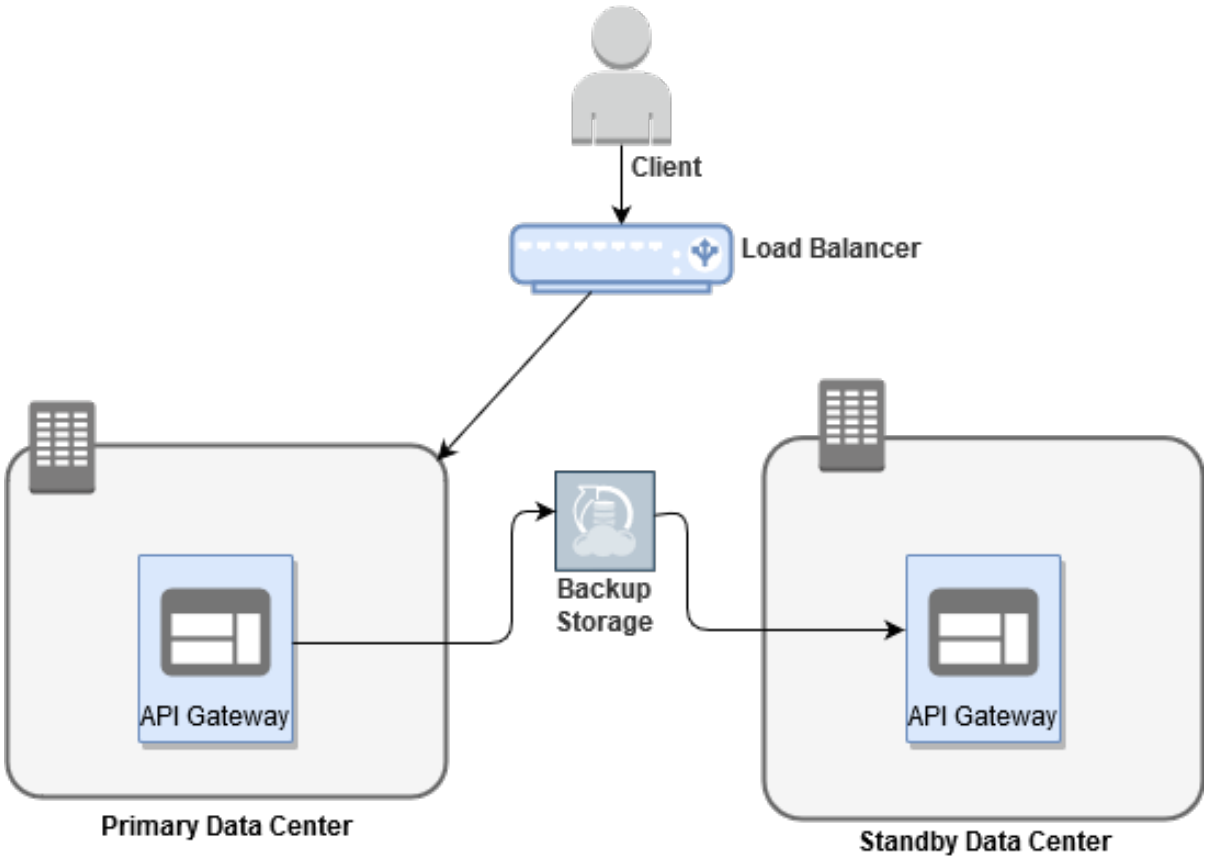
What is Cold Standby Mode?

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 "Data Backup and Restore" on page 216.

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.

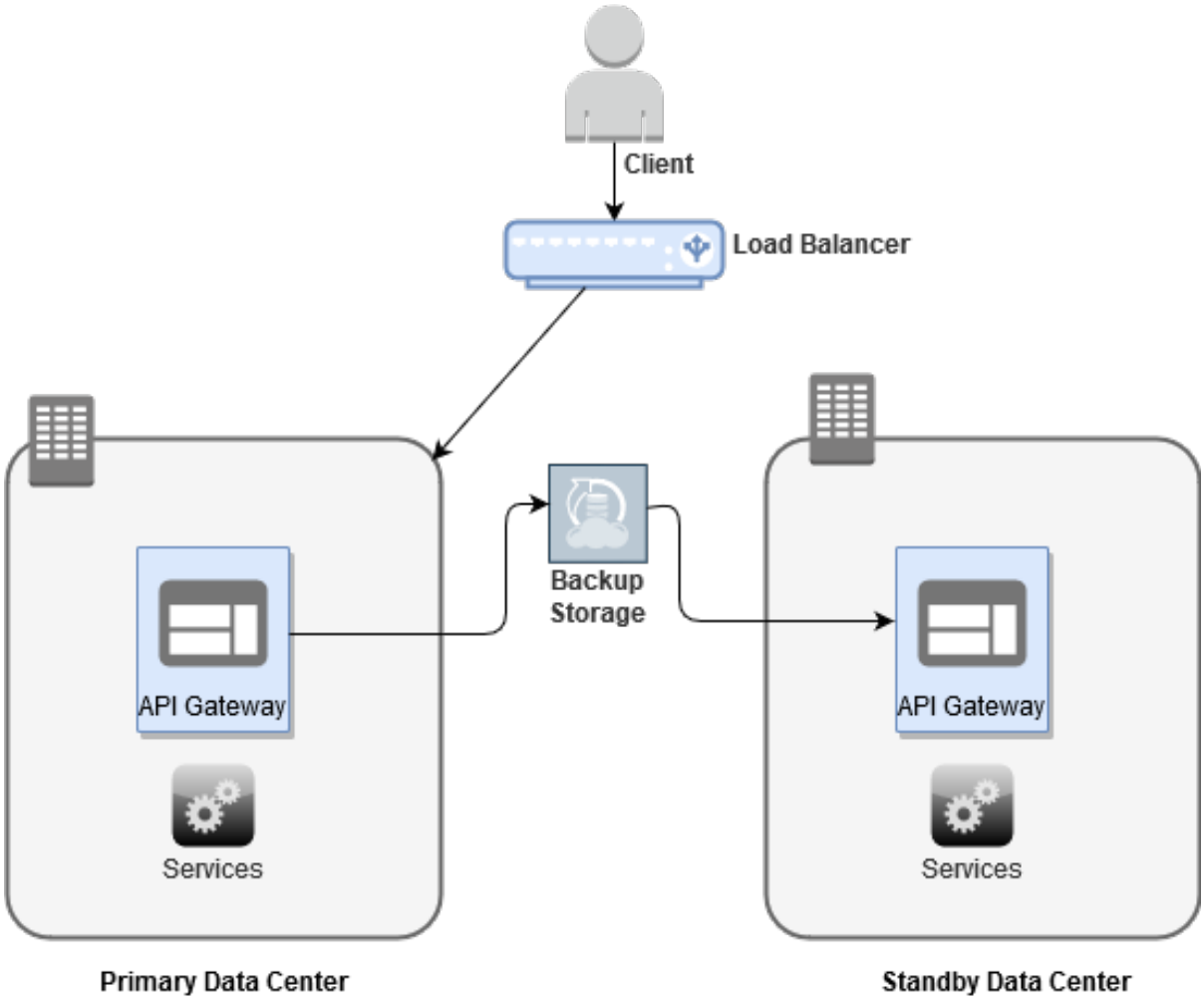
What is Warm Standby Mode?

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 "Data Backup and Restore" on page 216.

   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 Gateway Data Store. Hence, the data is symmetrical at any point of time.

As part of listener configuration you set up the port for gRPC channel through which each data center sends and receives the gossip. Later, with the ring configuration you establish connection with the associated data centers.

The underlying technology is same for both hot standby and active-active modes. The only difference is that, in the active-active mode you can have multiple data centers in a ring

configuration and each data center can handle the client request. On the other hand, the hot standby data center can have only two data centers in a ring configuration and only one data center handles the client request.

Currently, the Cross-DC support in API Gateway synchronizes the following data across the data centers:

- Application assets such as API key, identifiers, strategy, and client registrations that are associated with the strategy
- OAuth tokens, auth code, refresh tokens, and so on
- OAuth and OpenID scope mapping
- Application that are registered to API

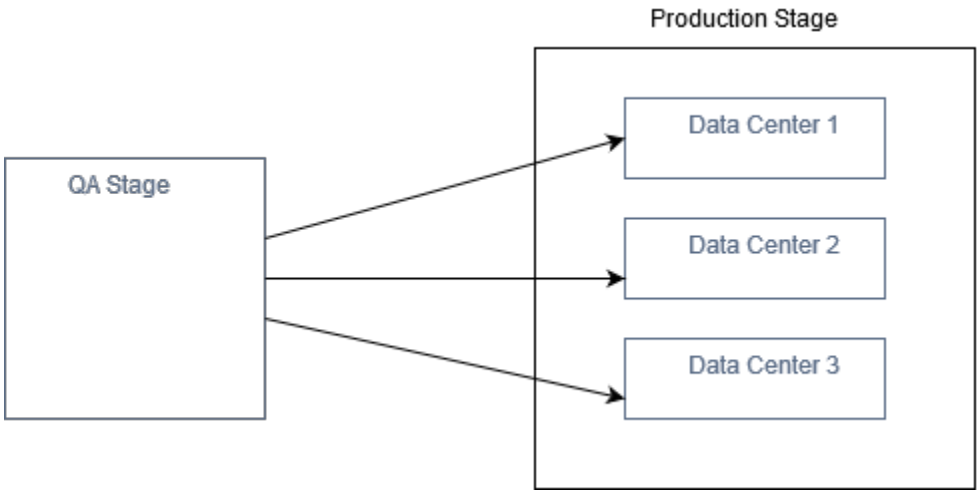
The below table suggests the possible workaround that could be employed for the different classes of data:

Data Type	Possible Workaround
APIs	Handle using Promotion Management.
Policies	Handle using Promotion Management.
Aliases	Handle using Promotion Management.
Packages	Handle using Promotion Management.
Plans	Handle using Promotion Management.
Subscriptions	Handle using Promotion Management.
Teams	Handle using Promotion Management.
Approval configurations	Handle using Promotion Management.
Keystore and Truststore configurations	Handle using Promotion Management.
Group	Handle using Promotion Management.
Email destination	Handle using Promotion Management.
JMS connection alias	Handle using Promotion Management.
Web service endpoint alias	Handle using Promotion Management.
Custom destination	Handle using Promotion Management.
Port	Handle using Promotion Management.
Service Registry	Handle using Promotion Management.
LDAP configuration	Handle using Promotion Management.

Data Type	Possible Workaround
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

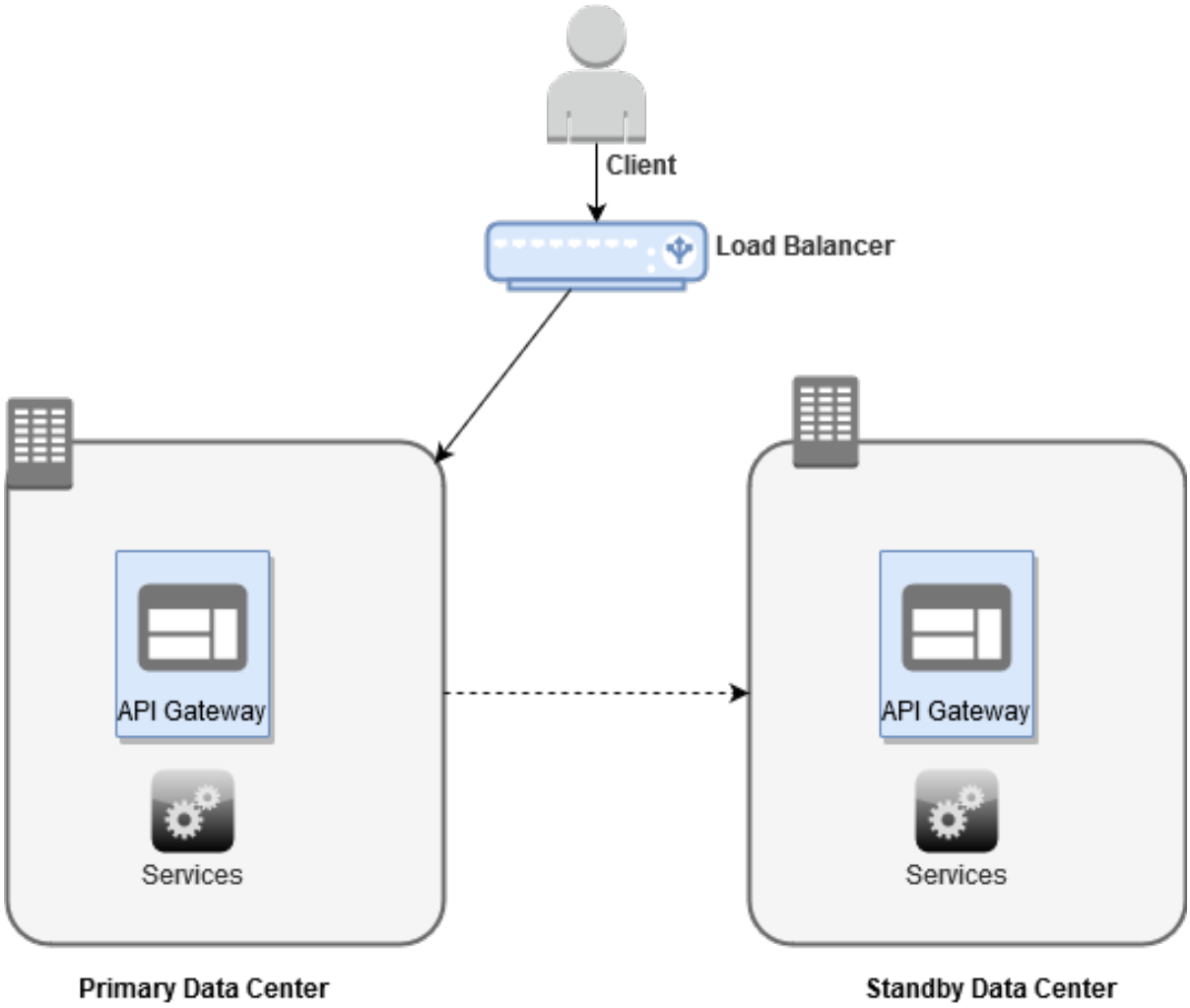
Promotion refers to moving API Gateway assets from the source stage to one or more target stages. For example, you might want to promote assets you have developed on servers in a QA stage (the source API Gateway instance) to data centers in a Production stage (the target API Gateway instance). If you 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 234.

  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 239.

  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 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 more information about configuring keystore and truststore, see *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 264.

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 more information about configuring keystore and truststore, see *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 more information on adding HTTPS port, see *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 264.

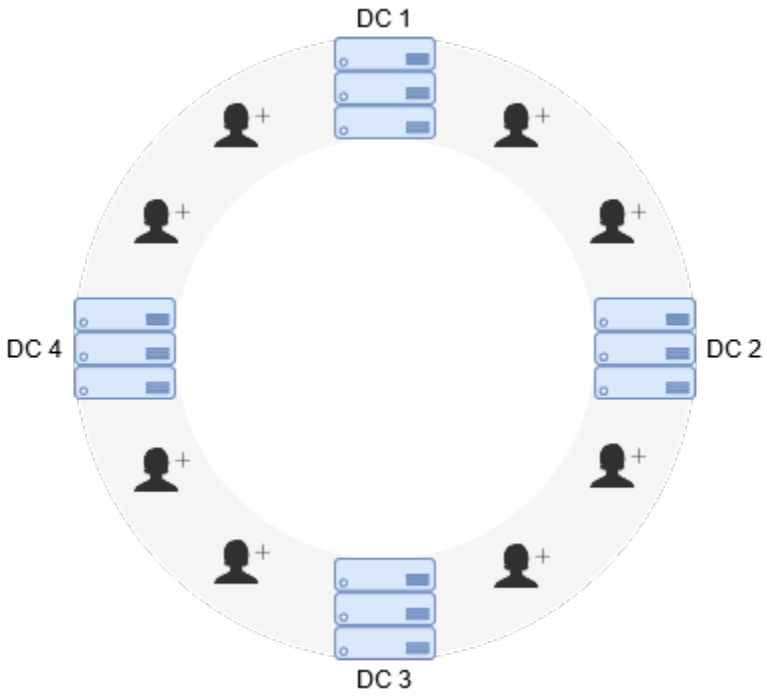
Note:

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 246.

  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 252.

  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 more information about configuring keystore and truststore, see *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 264.

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 more information about configuring keystore and truststore, see *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 more information on adding HTTPS port, see *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**.

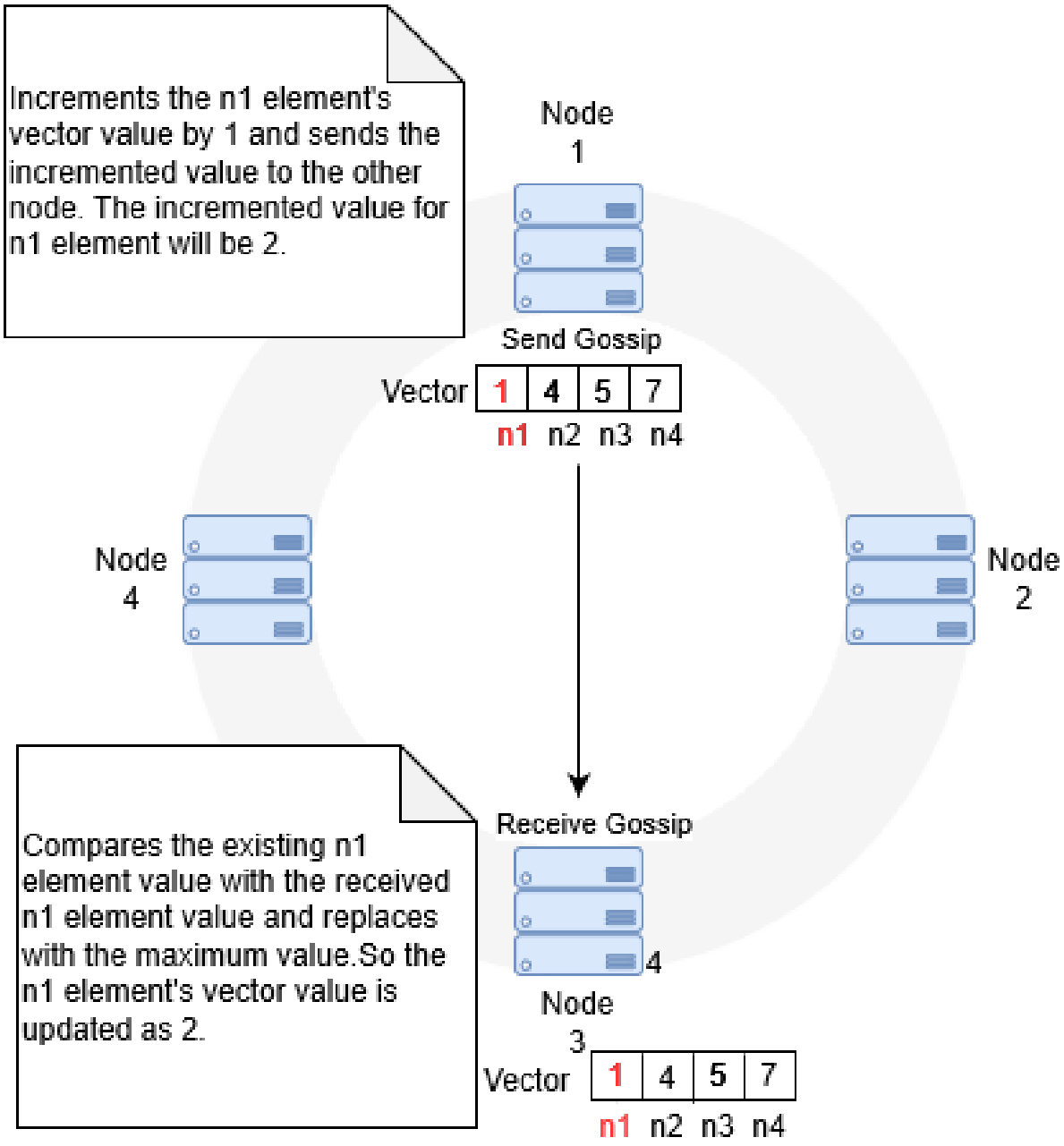
You can validate whether the data center is activated in the respective mode by reading the current configuration of the data center using the **GET/rest/apigateway/dataspace** REST API. For more information, see "How Do I Read the Current Configuration of the Data Center?" on page 264.

Note:
In active-active, if any one of the data center (DC 1 or DC 2 or DC 3) goes down, then that data center is removed from the ring. When the same data center is restored back, then that data center gets added to the ring automatically. If you want to add one more new data center (DC 4) to the ring, then you have to update the configuration with DC 4.

How does Cross-DC Support Detect Data Center Failures?

All the data centers are connected together to form a consistent hash ring using gRPC channel. The consistent hash ring is maintained properly with the help of *Gossiping protocol*. Using *Gossiping protocol*, API Gateway detects the failure of nodes. Here nodes represent the data centers.

This is how the *Gossiping protocol* works in Cross-DC support in active-active mode. Each node sends and receives gossip between one another to ensure that they are up and running. Each node has a vector of elements. For example, if there are *n* nodes in a ring then each node has a vector with *n* elements.



Each node sends and receives gossips with one another. When a node receives a gossip, it compares the received vector element value with the existing vector element value and replaces the vector element of the received node with the maximum value. When the vector element value is replaced, then that particular time frame gets captured. If the last updated time interval happens to be greater than permissible time interval between two consecutive gossips (that you define in the **pg_Dataspace_TimeToFail** extended settings), then that particular node is marked as dead. If the last updated time interval is twice as that of permissible time interval, then that particular node is removed from the ring. When the dead nodes are up, the *Gossiping protocol* detects them and rehashes the nodes.

As in active-active mode, the hot standby mode also uses consistent hash ring and *Gossiping protocol* to detect the failure of nodes. But in the hot standby mode there are only two nodes in the ring.

In hot standby mode, if the primary node goes down, the API Gateway administrator receives a notification and reconfigures the load balancer with the secondary node details. Hence, the secondary node handles the client request. When the primary node is restored back, the node gets added to the ring automatically. If the secondary node goes down, when the primary is active, then the secondary node is removed from the ring. When the secondary is restored it gets added to ring automatically.

How Do I 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 264.

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 264.

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:

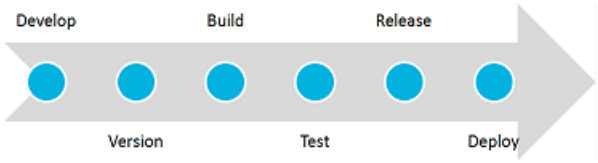
Extended Setting	Description
pg_Dataspace_GossipInterval	Specifies how frequently each node should gossip with one another. By default, the value is set to 3 seconds.
pg_Dataspace_TimeToFail	Specifies the maximum permissible interval between two consecutive gossips. By default, the value is set to 30 seconds.
pg_Dataspace_WarmupTime	Specifies the maximum permissible rehashing interval from start-up or shut down of the server. By default, the value is set to 300 seconds.

Make sure you configure the extended settings in each of the API Gateway instances that are installed across the data centers. For more information on configuring extended settings, see *webMethods API Gateway User's Guide*.

10 API Gateway Staging and Promotion

Staging and Promotion

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. Where, 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.

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.

Asset Promotion in API Gateway

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

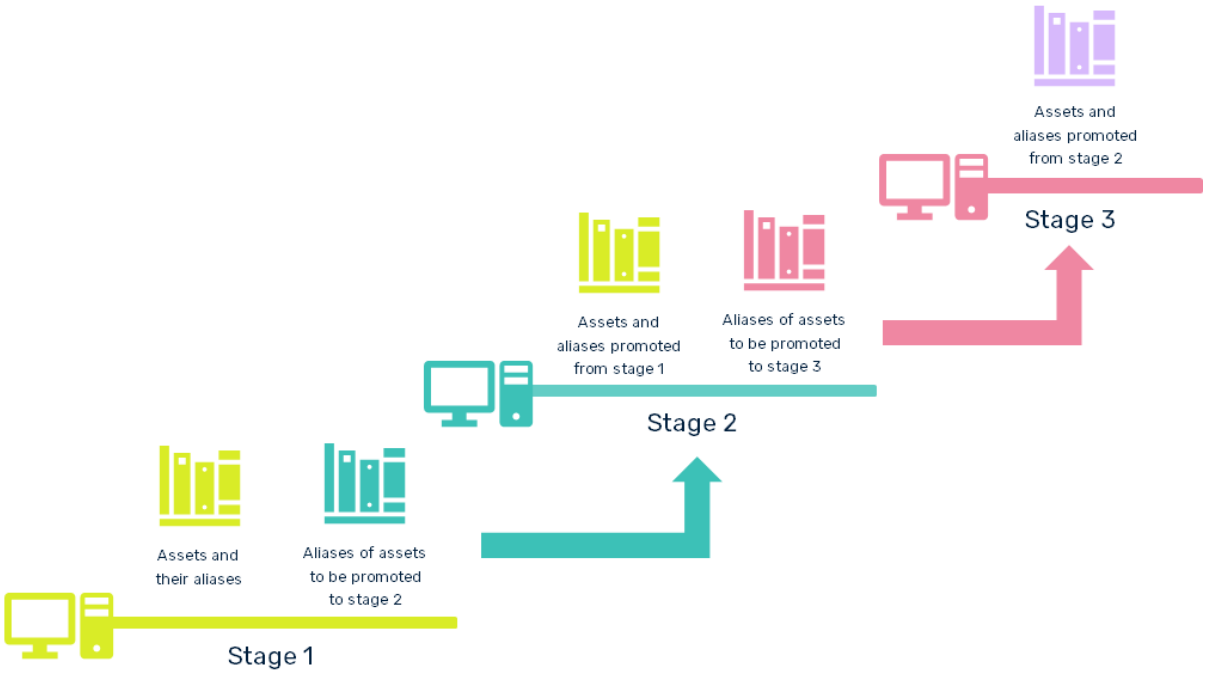
API Gateway staging and promotion allows you to:

- 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.

When you have three stages namely *Dev*, *QA*, and *Prod*, you can promote assets in the given sequence. That is, you can promote assets from stage 1 to stage 2, and from there to stage 3.

To promote assets from first stage, you create aliases of those assets that you want to promote to the second stage and promote them. Similarly, from the second stage you create aliases of assets that you want to promote to the third stage and promote them. The following illustration shows the flow:



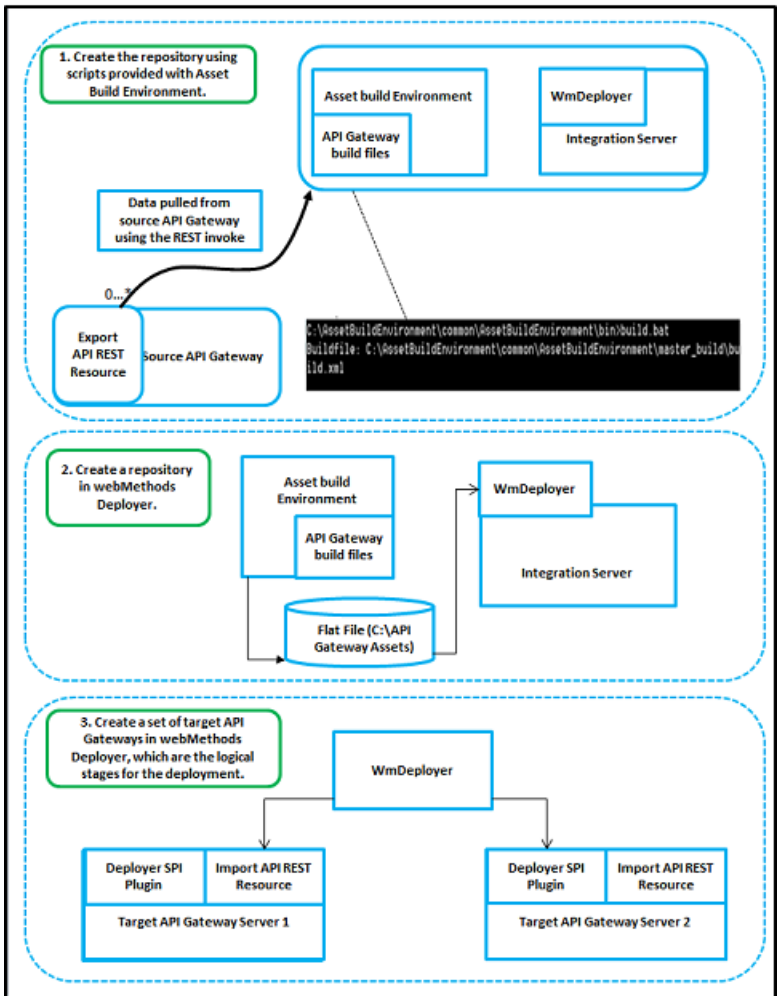
Assets can have aliases that are associated to a target stage and aliases without a target stage. When you promote to a target stage, the aliases associated to the particular target stage are promoted to the target stage. If an asset has no alias that is created for the target stage, then aliases that are not associated are included in the promotion.

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, see *webMethods API Gateway User's Guide*.

Promoting Assets Using webMethods 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:



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.

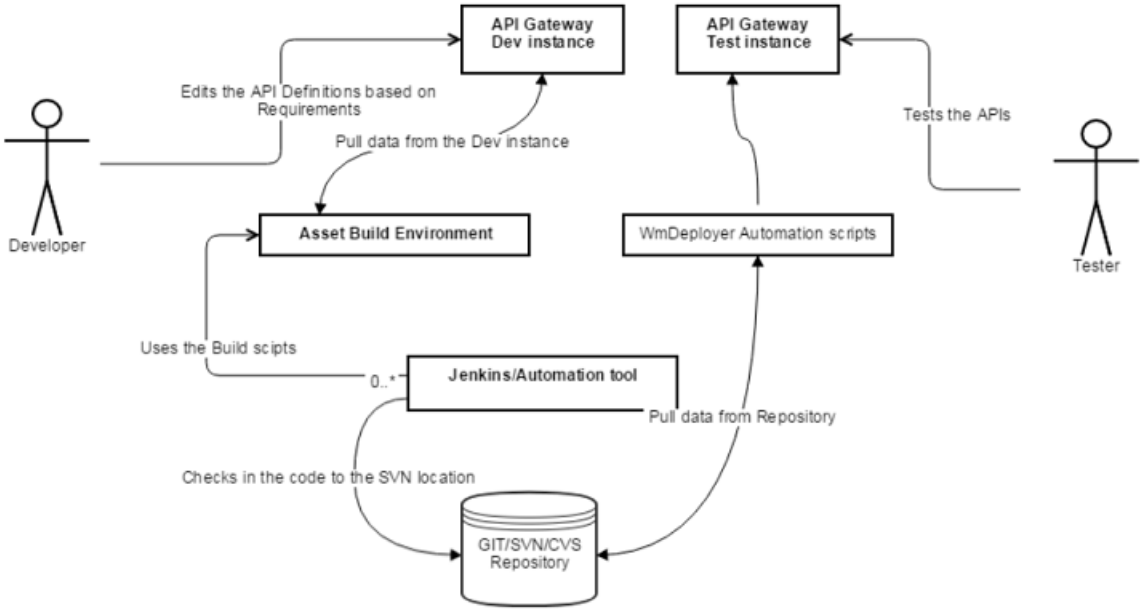
DevOps Use Case using Asset Build Environment and webMethods Deployer

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:

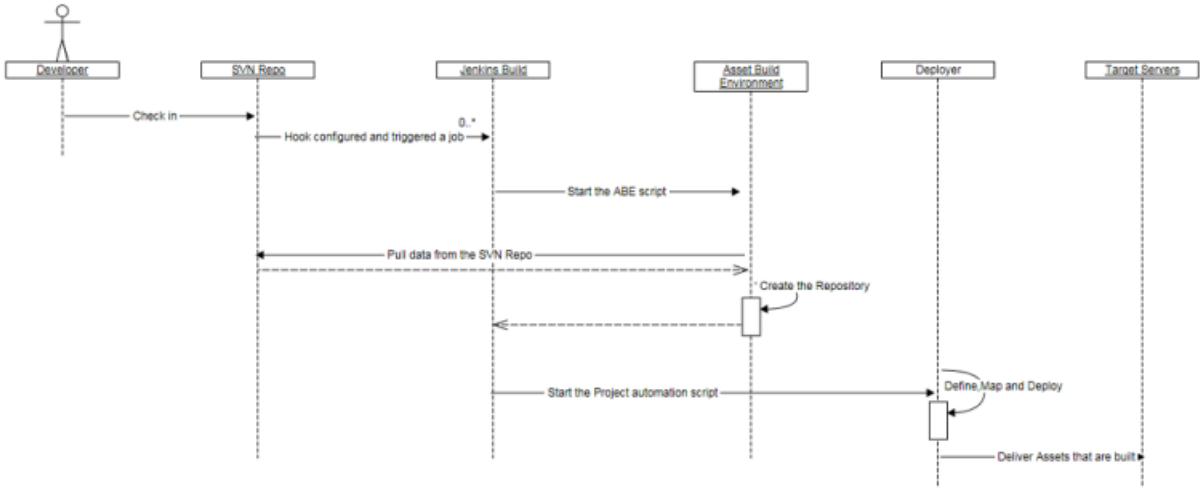
1. The developer makes changes to a development API Gateway instance.
2. A Jenkins job then uses the build script to pull data from this development instance and push it to a version control system such as GIT.

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



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 "Staging and Promotion" on page 268

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).

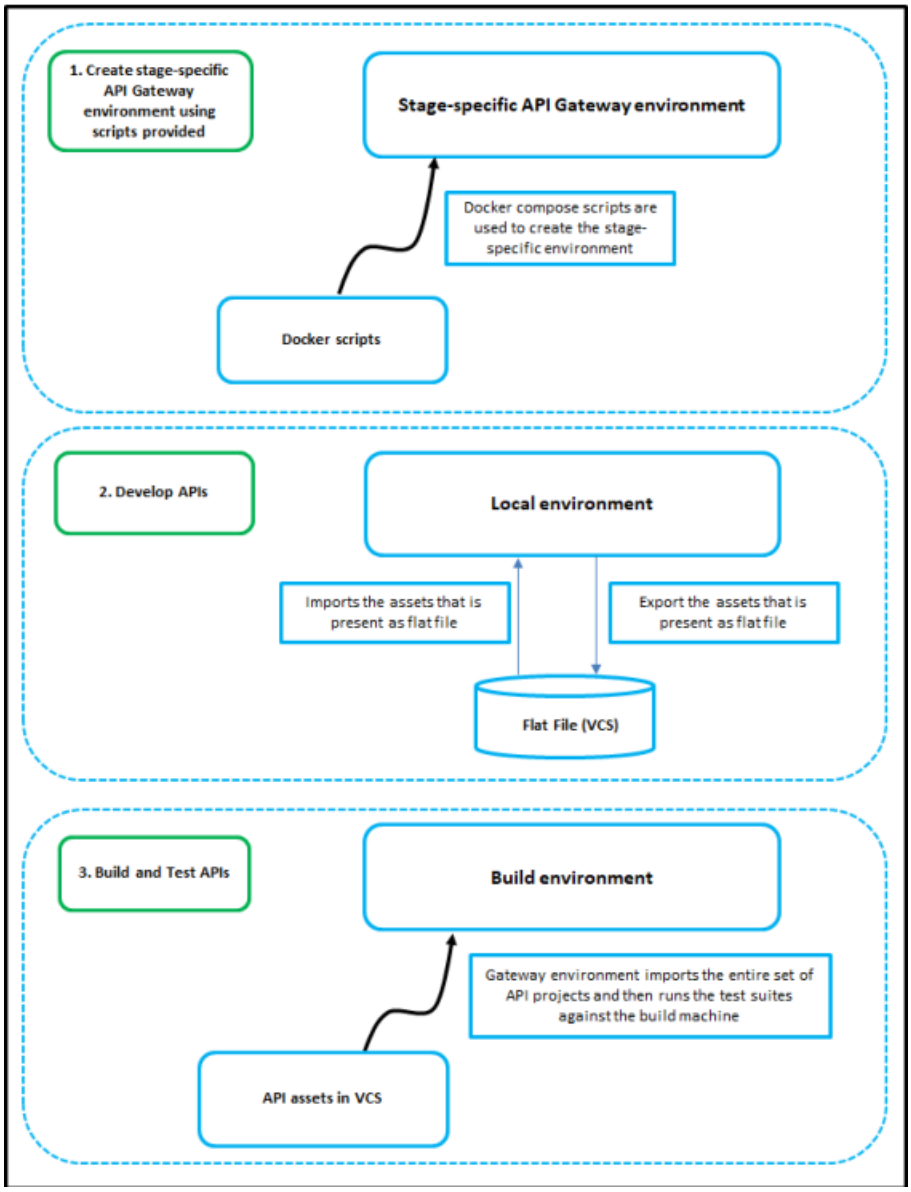
The promotion management REST APIs allow for automation for CI/CD. For more information about the promotion management API, see *webMethods API Gateway User's Guide*.

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.



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 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**.