API Gateway Configuration Guide

Version 9.12

October 2016

WEBMETHODS

This document applies to webMethods API Gateway Version 9.12 and to all subsequent releases.

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

Document ID: YAI-CG-912-20161221

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 storage locations for services on webMethods Integration Server, using the convention *folder.subfolder:service* .
UPPERCASE	Identifies keyboard keys. Keys you must press simultaneously are joined with a plus sign (+).
Italic	Identifies variables for which you must supply values specific to your own situation or environment. Identifies new terms the first time they occur in the text.
`Monospace font`	Identifies text you must type or messages displayed by the system.
{ }	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

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at http://documentation.softwareag.com. The site requires Empower credentials. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

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.

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.

1 API Gateway Introduction and Architecture

Overview

The APIs that an organization chooses to expose contain core assets the organization would naturally want to protect. As with the services they support, these APIs need to be managed and governed, and require mediation and security at run time. From an API provider's perspective, an API management tool is needed that enables the provider to do the following:

- Maintain an inventory of APIs and their associated resources.
- Publish and secure APIs according to defined service level agreements.
- Onboard API developers and give those developers the ability to publish APIs on behalf of the organization.
- Onboard API consumers who will use the published APIs in their own applications.
- Provide tiered access to APIs, for example according to authorization level.
- Track key performance indicators (KPIs) to help monitor and interpret API use.

webMethods API Gateway enables an organization to securely expose APIs to external developers, partners, and other consumers for use in building their own applications on their desired platforms. It provides a dedicated, web-based user interface to perform all the administration and API related tasks from the API creation, policy definition and activation, creation of consumer applications and API consumption. API Gateway gives you rich dashboarding capabilities for API Analytics. APIs created in API Gateway can also be published to API Portal for external facing developers consumption. API Gateway provides the following features:

Support for SOAP and REST APIs

API Gateway supports both SOAP-based APIs as well as REST-based APIs. This support enables organizations to leverage their current investments in SOAP-based APIs while they adopt REST for new APIs.

Secure APIs

API Gateway protects APIs from malicious attacks initiated by external client applications. Administrators can secure traffic between API consumer requests and the execution of services on API Gateway by filtering requests coming from particular IP addresses and blacklisting specified IP addresses, detecting and filtering requests coming from particular mobile devices, and avoiding additional inbound firewall holes through the use of reverse invoke.

Mediation

API Gateway provides complete run-time governance of APIs published to external destinations. API Gateway enforces access token and operational policies such as security policies for run-time requests between consumers and native services. API providers can enforce security, traffic management, monitoring, and SLA management policies, transform requests and responses into expected formats as necessary, perform

routing and load balancing of requests, and collect events metrics on API consumption and policy evaluation.

Easy discovery and testing of APIs

API Gateway provides text search capabilities that helps developers quickly find APIs of interest. API descriptions and additional documentation, usage examples, and information about policies enforced at the API level provide more details to the developers that helps them decide whether to adopt a particular API. Developers can use the provided code samples and expected error and return codes to try out APIs they are interested in, directly from within API Gateway, to see how the API works.

Clustering support

Multiple instances of API Gateway can be clustered together to provide scalability of mediation. A load balancer can be used to properly distribute the request messages.

Built-in usage analytics

API Gateway provides information about Gateway specific events and API-specific events, details about which APIs are more popular than others. This information is available by way of dashboards to API providers who have an API administrator role in API Portal. With this information, providers can understand how their APIs are being used, which in turn can help identify ways to improve their users' experience and increase API adoption.

Message transformation, pre-processing, and post-processing

API Gateway lets you configure an API and to transform the request and response messages to suit your requirements. To do this, you can specify an XSLT file to transform messages during the mediation process. You can also configure an API to invoke Integration Server services to pre-process or post-process the request or response messages.

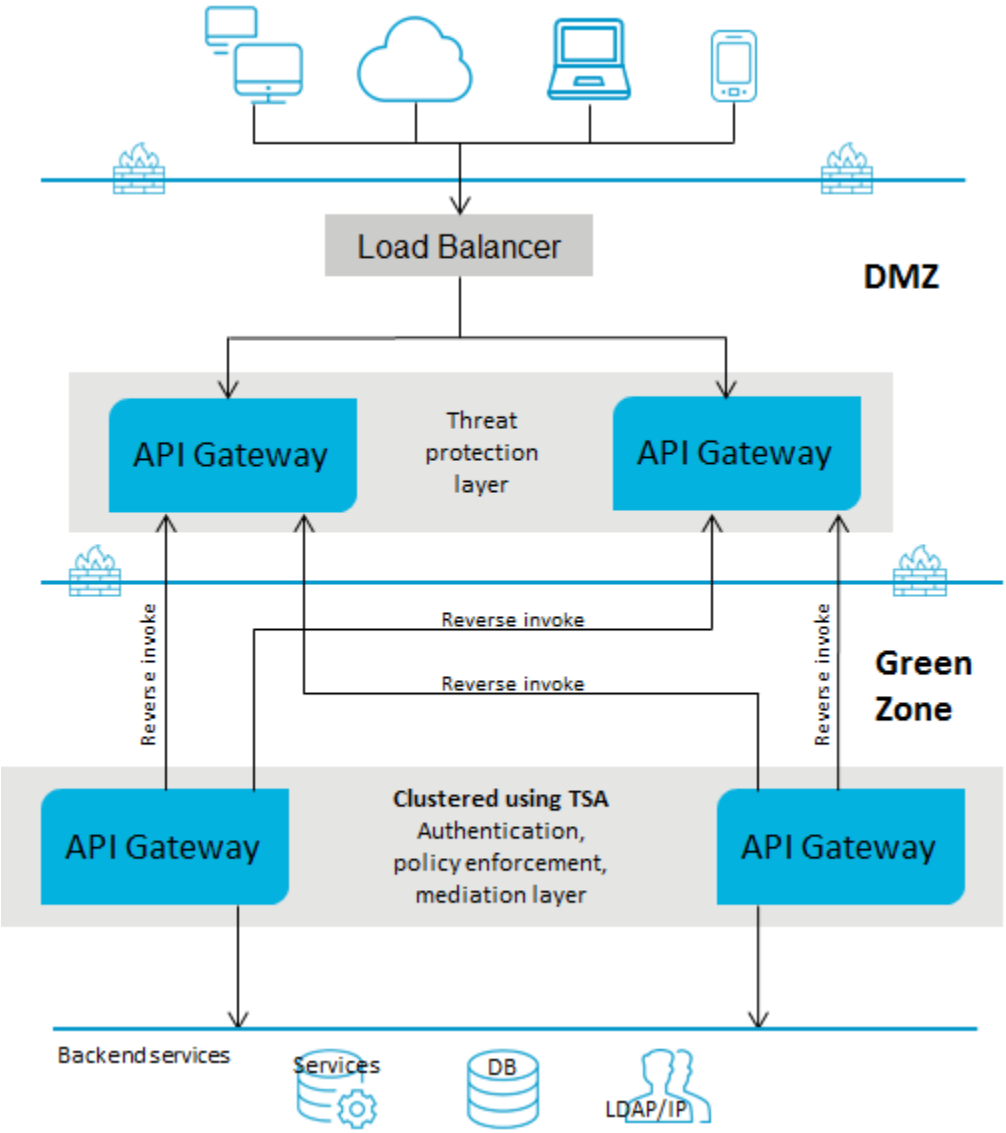
API Gateway Deployment Scenarios

API Gateway provides threat protection enforcement along with the policy enforcement, and mediation capabilities. This section describes high level API Gateway architecture for various deployment scenarios.

Deployment scenario 1: Paired Gateway Deployment Architecture

This deployment scenario has threat protection provided in the DMZ layer and authentication and policy enforcement in the green zone. The architecture consists of:

- One API Gateway for threat protection. This layer can have multiple instances connected using a load balancer.
- One API Gateway for authentication, policy, and mediation enforcement. This gateway infrastructure is completely protected using firewall and the communication for DMZ gateway happens through the reverse invoke approach. Various instances of API Gateways are clustered using Terracotta Server Array.

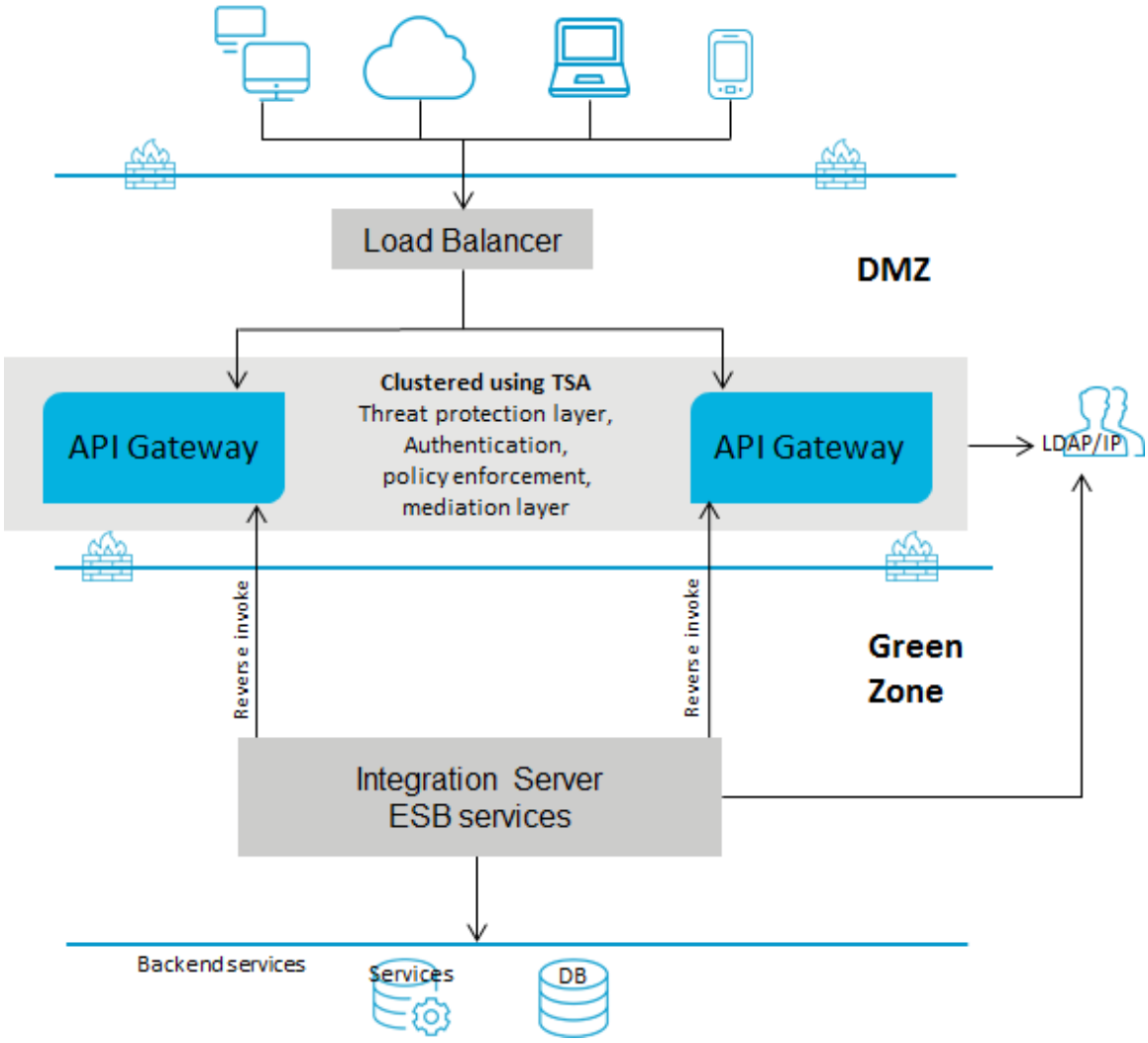


Note: If you have multiple instances of API Gateway connected using a load balancer for threat protection, when you change the rules enforced rules on one of the API Gateway instances you have to restart the other instances to synchronize the rule enforcement across all the API Gateway instances.

Deployment scenario 2: Single Gateway within DMZ for webMethods Customers

This deployment scenario has a single API Gateway and the load balancer in DMZ and IS ESB services in the green zone.

The single API Gateway is used for enforcing all policies and rules. There can be multiple instances of API Gateway connected through a load balancer and clustered using Terracotta Server array. The IS ESB service is protected using firewall.

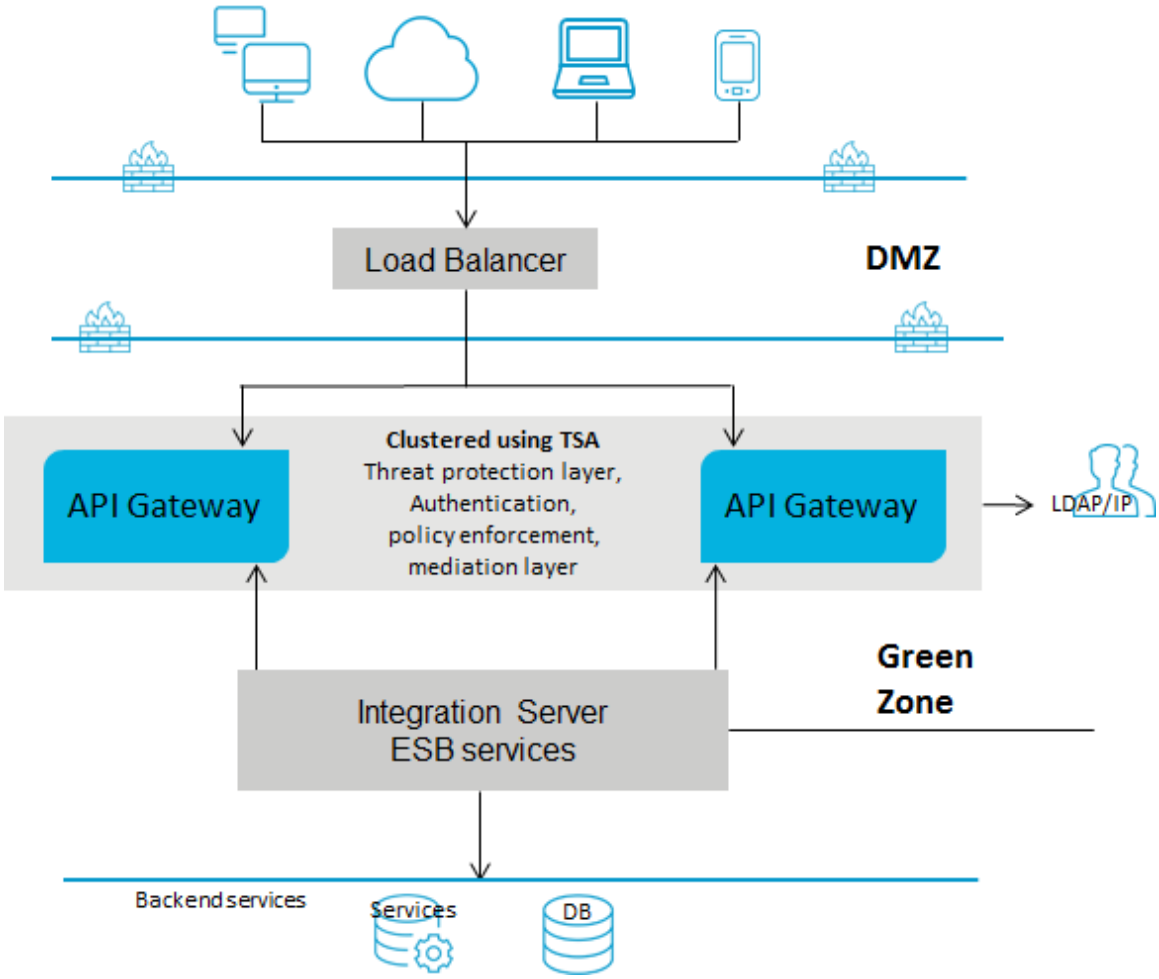


In this case, since the internal server is secured by the firewall and the mediator cannot invoke it directly, the endpoint in the routing policy that is applied should be configured as apigateway://*<registrationPort-aliasname>*/*<relative path of the service>*. For details, see the Ports section and the Routing policies section in the *webMethods API Gateway User's Guide*.

Deployment scenario 3: Single Gateway in Green zone for webMethods Customers

This deployment scenario has a single API Gateway and native ESB service within the Green Zone and the load balancer in DMZ.

A single API Gateway is used for enforcing authentication, and mediation. Threat protection is not required in this deployment structure, but if required the treat protection rules can be configured for enforcement. You can have multiple instances of API Gateways connected through a load balancer and clustered using Terracotta Server Array. There is a direct invocation of ESB services.

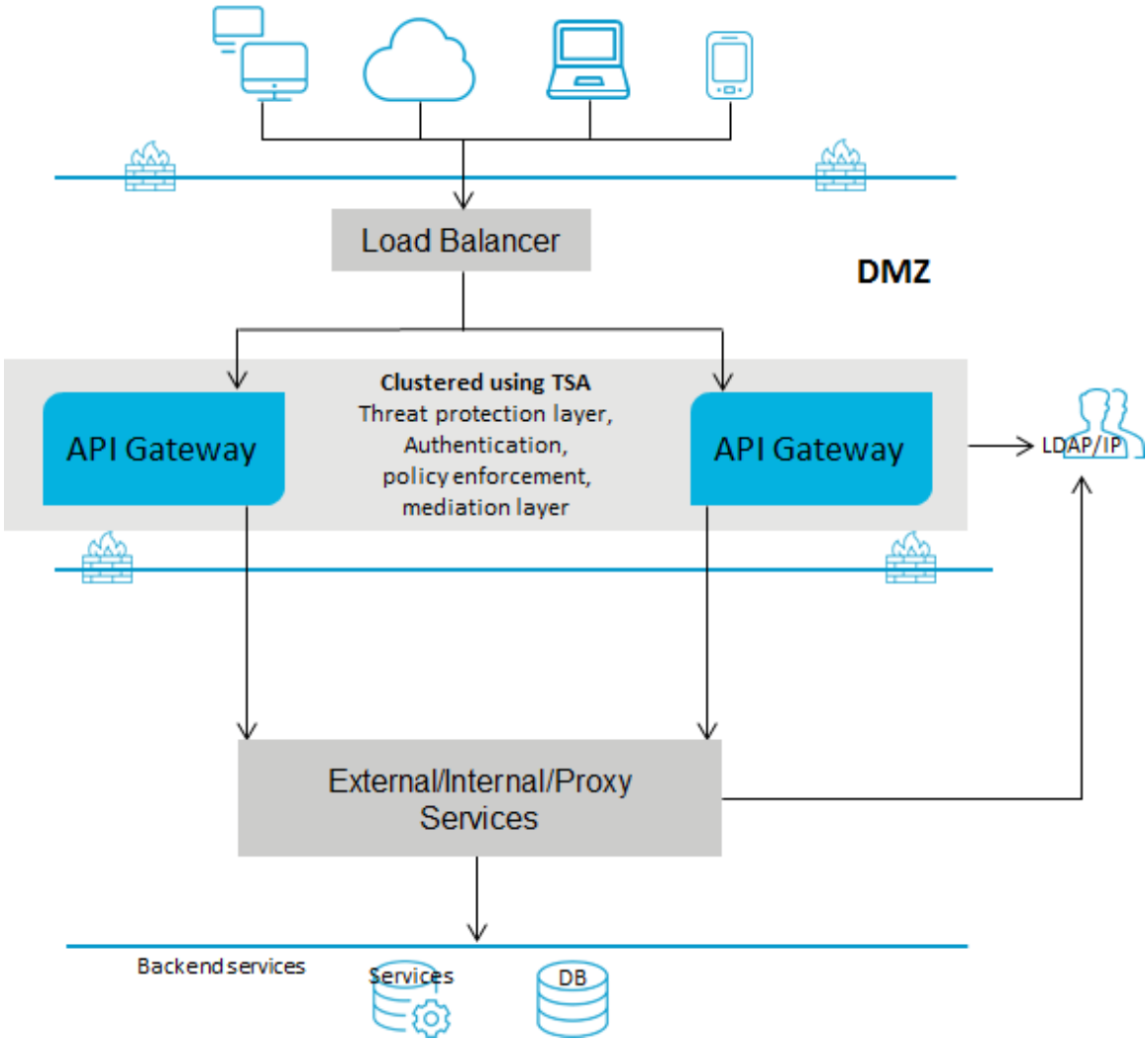


Note: Since the API Gateway instance and the ESB service are in the same network you can either have a direct invocation of ESB service or you can use the reverse invoke approach as required.

Deployment scenario 4: Single Gateway for non webMethods Customers

This deployment scenario has a single API Gateway and native services within the Green Zone and the 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. There is a direct invocation of native services. You have to open the native service port to the mediator network.



API Gateway Editions

API Gateway is available in the following two editions based on the type of license used:

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

The type of license (Standard or Advanced) imported into API Gateway determines whether the customer has just the threat protection of internal ESB servers in DMZ or complete capabilities. Customers can change the licensing at any time from Standard Edition to the Advanced Edition.

This table lists the capabilities provided 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
Service management	No	Yes
Policy management ■ Threat protection rules	Yes	Yes
Policy management ■ Mediation policies	No	Yes
Application management	No	Yes
Analytics ■ Threat protection rules violations	Yes	Yes
Analytics ■ Service ■ Applications ■ Consumers	No	Yes
Clustering and auto synchronization	No	Yes
HA and failover	Yes, if you have a terracotta server	Yes, if you have a terracotta server

Feature	Standard Edition	Advanced Edition
	installed in the DMZ zone.	installed in the DMZ zone.

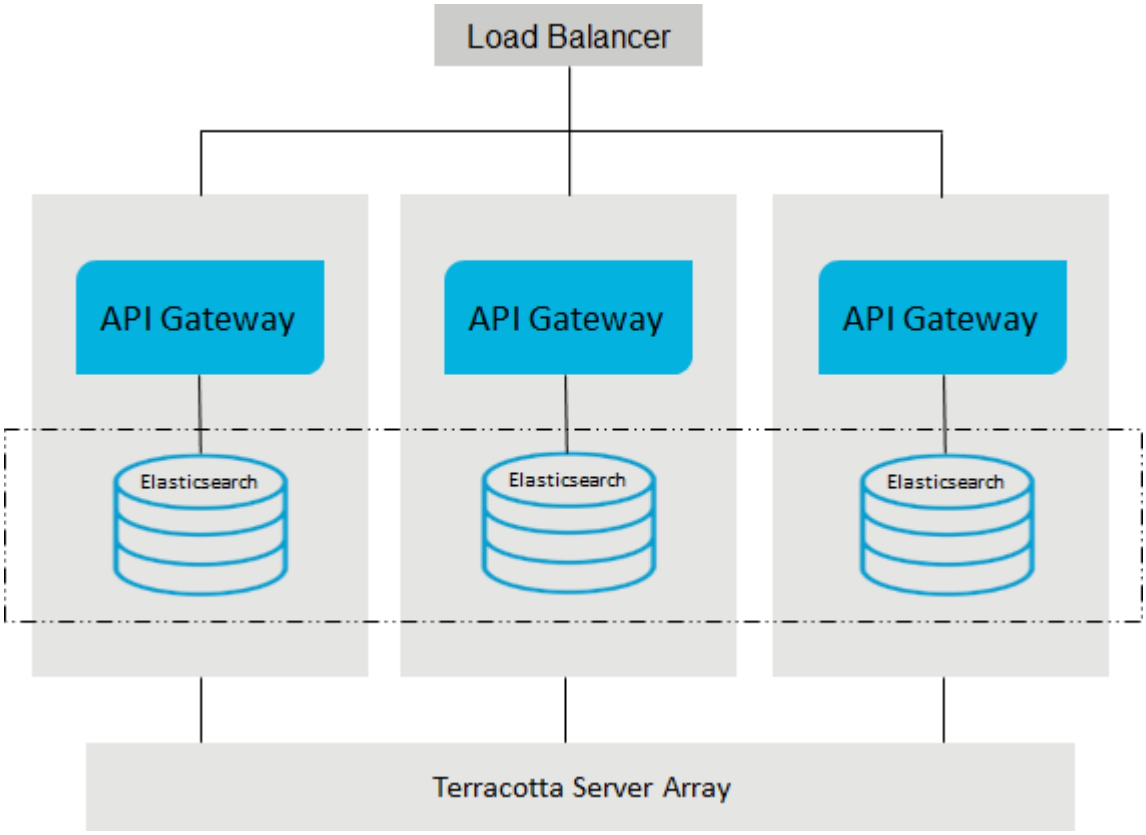
2 API Gateway Configuration

API Gateway Cluster Configuration

This section covers aspects of setting up and using API Gateway after you have installed the product software. For installation procedures for the product software, see *Installing webMethods Products*

Nodes and Clusters

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



EachAPI Gateway cluster node holds all the API Gateway components including UI, the API Gateway package running in webMethods Integration Server and an Event 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 Event Data Store clustering that has to be defined across the Event Data Store instances.

Since 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 covers configuration items, and runtime

assets like APIs, policies and applications. The synchronized runtime assets become active automatically.

Configuring an API Gateway Cluster

Configuring an API Gateway cluster requires the following:

- Configuring Terracotta Server array
- Configuring Integration Server cluster
- Configuring Event Data Store cluster
- Configuring load balancer

Integration Server Configuration

API Gateway's cluster implementation is built upon the Integration Server's cluster support. In contrast to the Integration Server clustering (see *webMethods Integration Server Clustering Guide*) API Gateway does not require any database that is shared across the cluster nodes.

The first step of the Integration Server Configuration is to add the following entries to the file <INSTALL-DIR>/IntegrationServer/instances/default/config/server.cnf:

- watt.server.cluster.aware=true
- watt.server.cluster.name=APIGatewayTSAcluster
- watt.server.cluster.tsaURLs=<TSA host>:<TSA port>
- watt.server.terracotta.license.path=<path to TSA license file>

The second step is to extend the wrapper.conf with an additional java parameter. For this the the file <INSTALL-DIR>/profiles/IS_default/configuration/custom_wrapper.conf has to be extended.

- wrapper.java.additional.xx=-Dtest.cluster.withDerby=true

Here xx means any free additional java parameter number.

Note: Ensure that you configure all the cluster nodes with the same Integration Server configuration changes.

For additional details, see *webMethods Integration Server Clustering Guide*

Event Data Store Configuration

Each API Gateway cluster node comes with an Event Data Store instance for storing runtime assets and configuration items. An Event Data Store instance is a non clustered Elasticsearch node. For a cluster configuration, the Event Data Store instances should also be clustered by modifying the <SAG_root>/EventDataStore/config/elasticsearch.yml file on each instance using standard Elasticsearch clustering properties (for more information see, https://www.elastic.co/guide/en/elasticsearch/guide/current/important-configuration-changes.html and https://www.elastic.co/guide/en/elasticsearch/

reference/2.3/index.html). The cluster name has to be specified and the cluster nodes have to be configured.

A sample configuration looks like follows:

```
cluster.name:"elasticsearch"
network.host:0.0.0.0
http.port:9415
transport.tcp.port:9425
node.master:true
discovery.zen.ping.multicast.enabled:false
discovery.zen.ping.unicast.hosts: ["apigateway1:9425","apigateway2:9425","apigateway3:9425"]
```

Cluster Health

The health of the Elastic Search cluster can be checked using the following URL:

http://daefermion4:9415/_cluster/health?pretty=true

The cluster health response looks as follows:

```
{
  "cluster_name" : "elasticsearch",
  "status" : "green",
  "timed_out" : false,
  "number_of_nodes" : 3,
  "number_of_data_nodes" : 3,
  "active_primary_shards" : 11,
  "active_shards" : 22,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 0,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 100.0
}
```

The response shows the status of the cluster and the number of its nodes. A sample response showing an unhealthy cluster status looks as follows:

```
{
  "cluster_name" : "elasticsearch",
  "status" : "yellow",
  "timed_out" : false,
  "number_of_nodes" : 2,
  "number_of_data_nodes" : 2,
  "active_primary_shards" : 11,
  "active_shards" : 15,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 7,
  "delayed_unassigned_shards" : 7,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 68.18181818181817
}
```

Here the Event Data Store cluster state is yellow and the number of nodes indicate that a cluster node is missing. An unhealthy cluster state can be caused by communication problems between the cluster nodes. To recover from the unhealthy state the Integration

Server running the API Gateway with the missing node has to be restarted. A restart forces the Event Data Store instance to rejoin the cluster.

For details on cluster health see, https://www.elastic.co/guide/en/elasticsearch/guide/current/_cluster_health.html.

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/

Load Balancer Configuration

A custom load balancer can be used for API Gateway cluster. Here we are using 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 daefermion4:5555;
        server daefermion4:5556;
        server daefermion4:5557;
    }
    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 may be used for greater debugging. The load needs to be exposed through the firewall that is protecting the host the firewall is running on.

Limitations of the API-Gateway Clustering

OAuth tokens - OAuth tokens are not synchronized across cluster nodes. Currently there is no workaround for this limitation.

Ports configuration - Port configuration are not synchronized across cluster nodes. Currently there is no workaround for this limitation.

3 Configuration Properties

Configuration Types and Properties

This section describes the configuration types and parameters that have to configured for API Gateway.

The configuration types are broadly classified as web-app and API Gateway package level configurations.

Web-app Configuration Properties

These properties are not cluster aware and hence have to be manually copied to all nodes.

General properties

Location: <*SAG_Root*>`/profiles/IS`<*IS_Instance_Name*>`/apigateway/config/uiconfiguration.properties`

apigw.auth.priority

API Gateway supports both Form and SAML based authentication. If both are enabled, this property decides the which login page should be shown by default when user visits login page http://host:9072/apigatewayui. Of course user can go to specific login page using:

- Form : http://host:9072/apigatewayui/login
- SAML : http://host:9072/apigatewayui/saml/sso/login

Default value is form. Possible values are form and saml

apigw.auth.form.enabled

This property enables or disables form based authentication. If both SAML and Form are disabled, Form remains as default.

Default value is true. Possible values are true and false.

apigw.auth.form.redirect

If a protected resource is accessed and if 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 does not change by default. If IS is run on some other default port, this property should be corrected.

Default value is http://<localhost>:5555

apigw.is.timeout

This property denotes the user session timeout value.

Default value is 90.

SAML SSO properties

API Gateway user interface supports the following SAML values:

- **Profile**: Web Browser SSO
- **Protocol**: SAML Auth Request
- **Binding**: HTTP Post

The value sent in NameID is used as logged in user Id. In the absence of NameID, the attribute value with http://schemas.xmlsoap.org/ws/2005/05/identity/claims/upn namespace is used for user id. Attribute value with http://schemas.microsoft.com/ws/2008/06/identity/claims/role or http://schemas.xmlsoap.org/claims/Group can be used to pass the roles. The roles can be one or more roles supported by API Gateway. The other option is to send any roles and map those roles to one of the API Gateway roles. The mapping should be done in `<SAG_Root>/IntegrationServer/instances/<IS_Instance_Name>/packages/WmAPIGateway/config/resources/security/saml_groups_mapping.xml`. The format is <group source="<Some Role>" target="<API Gateway Role>" />. Example <group source="API Managers" target="API-Gateway-Providers" /> .

The next step is to configure <SAG_Root>/IntegrationServer/instances/<IS_Instance_Name>/config/is_jaas.cnf. Replace `com.wm.app.b2b.server.auth.jaas.SamlOSGiLoginModule requisite;` with `com.softwareag.apigateway.auth.saml.APIGatewaySamlLoginModule requisite;` and save the changes. Configure the properties mentioned below to enable the SAML based SSO.

Location: *<SAG_Root>*`/profiles/IS`*<IS_Instance_Name>*`/apigateway/config/uiconfiguration.properties`

apigw.auth.saml.enabled

This property enables or disables SAML based authentication.

Default value is false. Possible values are true and false.

apigw.auth.saml.redirect

Denotes the location of keystore. The keystore with self-signed certificates are shipped at <SAG_Root>/profiles/IS_ <IS_Instance_Name> /apigateway/config/keystore/saml_sso.jks. The keystore has 3 certificates with the following alias:

- sign: the password is signapigw
- encrypt: the password is encryptapigw
- default: the password is defaultapigw

The password for keystore is apigwstore.

Default value is none.

apigw.auth.saml.keystore.type

Denotes the keystore type.

Default value is JKS. Possible values are JKS and PKCS12.

apigw.auth.saml.keystore.pwd

Denotes the keystore password. On web-app start, this password is moved to passman secure store (<SAG_Root>/profiles/IS_<IS_Instance_Name>/apigateway/config/passman) and the handle is maintained here. Tampering with the handle results in exceptions.

Default value is None.

apigw.auth.saml.signkey.alias

This is the certificate alias used for signing the SAML authentication request.

Default value is None.

apigw.auth.saml.signkey.pwd

Denotes the password for the above certificate alias. On web-app start, this password is moved to passman secure store and the handle is maintained here. Tampering with the handle results in exceptions.

Default value is None.

apigw.auth.saml.encrypkey.alias

Denotes the certificate alias to be used for encrypting the SAML authentication request.

Default value is None.

apigw.auth.saml.encrypkey.pwd

Password for above. On web-app start, this will password will be moved to passman secure store and handle will be maintained here. Tampering handle will result in exceptions.

Default value is None.

apigw.auth.saml.defaultkey.alias

This alias is used for signing and encryption if sign and encryption related alias are missed.

Default value is None.

apigw.auth.saml.defaultkey.pwd

Denotes password for above default key alias. On web-app start, this password is moved to passman secure store and the handle is maintained here. Tampering with the handle results in exceptions.

Default value is None.

apigw.auth.saml.authreq.signed

Denotes whether to send the signed SAML authentication request.

Default value is true. Possible values are true and false.

apigw.auth.saml.assertion.signed

Denotes whether we expect the signed assertion to be sent by the IDP.

Default value is true. Possible values are true and false.

apigw.auth.saml.sp.id

Denotes service provider identity which is sent as part of SAML authentication request.

Default value is Host name of localhost.

apigw.auth.saml.ldp.metadata.url

Denotes the file URL of IDP metadata. Consult your IDP documentation on how to generate one.

Default value is none.

apigw.auth.saml.sp.metadata.url

Denotes the file URL of Gateway metadata. For now, you can get the content from http://host:9072/apigatewayui/saml/sso/metadata

Default value is none.

Kibana

Location : <*SAG_Root*>`/profiles/IS_`<*IS_Instance_Name*>`/apigateway/config/uiconfiguration.properties`

apigw.kibana.autostart

Decides whether kibana should be started as part of web-app.

Default value is true. Possible values are true and false.

apigw.kibana.url

Denotes the URL where Kibana is running. <*localhost*> is replaced by the hostname that is resolved through localhost. The port and other configurations of the kibana can be changed from <*SAG_Root*>`/profiles/IS_`<*IS_Instance_Name*>`/apigateway/kibana-4.5.1\config/kibana.yml`

Default value is http://<localhost>:9405

apigw.es.url

Denotes the URL where EventDataStore (HTTP) is running. <*localhost*> is replaced by the hostname that is resolved through localhost.

Default value is http://<localhost>:9240

API Gateway Package Configuration Properties

API Gateway uses Software AG's Event Data Store (Elasticsearch) as its data repository. API Gateway starts the Elasticsearch instance, if configured, using the default configuration shipped at <*SAG_Root*>`\EventDataStore\config\elasticsearch.yml`

Note: To run Event Data Store instances in a cluster, the elasticsearch.yml file must be updated on each instance. See https://www.elastic.co/guide/en/elasticsearch/guide/current/important-configuration-changes.html#important-configuration-changes for details.

Location : <*SAG_Root*>`/IntegrationServer/instances/`<*IS_Instance_Name*>`/packages/WmAPIGateway/config/resources/elasticsearch/config.properties`

pg.gateway.elasticsearch.autostart

Flag to manage (start or stop) Elasticsearch as part of API Gateway. Set it to false if the start or stop of Elasticsearch is managed from outside the purview of API Gateway.

Default value is true. Possible values are true and false.

pg.gateway.elasticsearch.start.maxwait

Denotes maximum time in seconds API Gateway waits for Elasticsearch to start and stop if autostart is set to true.

Default value is 300.

pg.gateway.elasticsearch.config.location

If a different config file has to be used for some reason, the location of the config file has to mentioned here.

Default value is <*SAG_Root*>`\EventDataStore\config\elasticsearch.yml`

Note:

- If the cluster.name property in the above mentioned elasticsearch.yml has to be modified, the same name should be used for <prop key=cluster.name> in <*SAG_Root*>`/IntegrationServer/instances/`<*IS_Instance_Name*>`/packages/WmAPIGateway/config/resources/beans/gateway-datastore.xml`
- If the transport.tcp.port in the above mentioned elasticsearch.yml has to be modified, the same port should be used for <value>localhost:9340</value> in <*SAG_Root*>`/IntegrationServer/instances/`<*IS_Instance_Name*>`/packages/WmAPIGateway/config/resources/beans/gateway-datastore.xml`

- If the http.port in the above mentioned elasticsearch.yml has to be modified, the same port should be used for apigw.es .url in <*SAG_Root*>`/profiles/IS_ <IS_Instance_Name>/apigateway/config/uiconfiguration.properties`