# API Gateway

Version 7.6.2

14 July 2020

# Container Deployment Guide



This documentation describes the following Axway software:

Axway API Gateway 7.6.2

# Contents

# Preface

This guide describes how to deploy API Gateway, API Manager, and API Gateway Analytics in containers.

## Who should read this guide

This guide is intended for system engineers who are responsible for deploying and operating API Gateway in containers. Familiarity with containers is recommended.

Familiarity with API Gateway concepts and features is also recommended. For more details, see the *API Gateway Concepts Guide*.

## How to use this guide

This guide should be used in conjunction with the other guides in the API Gateway documentation set.

Before you begin deploying API Gateway in containers, review this guide thoroughly. The following is a brief description of the contents of each section:

- *Elastic deployment in the cloud* on page 11 – Provides an overview of deploying API Gateway in containers to achieve elasticity.
- *Get started with API Gateway in containers* on page 14 – Describes how existing users of API Gateway (version 7.5.3 and earlier) can get started with API Gateway in containers.
- *Deploy API Gateway in Docker containers* on page 19 – Describes the steps you must perform to deploy API Gateway in Docker containers.
- *Deploy API Manager or OAuth in Docker containers* on page 43 – Describes the steps you must perform to deploy API Gateway with optional API Manager or OAuth services in Docker containers.
- *Deploy API Gateway Analytics in Docker containers* on page 46 – Describes the steps you must perform to deploy the optional API Gateway Analytics monitoring and reporting tool in Docker containers.
- *Operate and monitor API Gateway containers* on page 51 – Describes how you can operate and monitor API Gateways running in a container environment.
- *Development and deployment with API Gateway containers* on page 55 – Describes how you can develop, promote, and deploy APIs and policies to API Gateways running in a container environment.

- *Migrate to container deployment* on page 58 – Describes how to migrate an API Gateway 7.6.2 classic deployment to an elastic container deployment.
- *Troubleshoot container deployments* on page 66 – Describes common problems and suggested solutions when running API Gateway in containers.

# Related documentation

The AMPLIFY API Management solution enables you to create, publish, promote, and manage Application Programming Interfaces (APIs) in a secure and scalable environment. For more information, see the *AMPLIFY API Management Getting Started Guide*.

The following reference documents are also available on the Axway Documentation portal at https://docs.axway.com:

- *Supported Platforms*

  Lists the different operating systems, databases, browsers, and thick client platforms supported by each Axway product.

- *Interoperability Matrix*

  Provides product version and interoperability information for Axway products.

# Support services

The Axway Global Support team provides worldwide 24 x 7 support for customers with active support agreements.

Email support@axway.com or visit Axway Support at https://support.axway.com.

See "Get help with API Gateway" in the *API Gateway Administrator Guide* for the information that you should be prepared to provide when you contact Axway Support.

# Training services

Axway offers training across the globe, including on-site instructor-led classes and self-paced online learning. For details, go to: http://www.axway.com/support-services/training

# Accessibility

Axway strives to create accessible products and documentation for users.

This documentation provides the following accessibility features:



## Screen reader support

- Alternative text is provided for images whenever necessary.
- The PDF documents are tagged to provide a logical reading order.

## Support for high contrast and accessible use of colors

- The documentation can be used in high-contrast mode.
- There is sufficient contrast between the text and the background color.
- The graphics have the right level of contrast and take into account the way color-blind people perceive colors.

# Updates and revisions

This guide includes the following documentation changes.

## Changes in version 7.6.2

- Added a new topic that describes how to get started with running API Gateway and API Manager in containers and highlights the main differences between container deployments and classic deployments. For more details, see *Get started with API Gateway in containers* on page 14.
- Added a new troubleshooting topic that describes common problems encountered in container deployments and provides suggested solutions. For more details, see *Troubleshoot container deployments* on page 66.
- Added information on the `EMT_DEPLOYMENT_ENABLED=true` environment variable that you can use when starting API Gateway Docker containers. This enables you to deploy configuration changes directly from Policy Studio to API Gateway containers in a development environment only. For more details, see *Step 8 – Start the API Gateway Docker container* on page 41 and *Development and deployment with API Gateway containers* on page 55.
- Renamed the `--merge-folder` option used when building API Gateway Docker images to `--merge-dir`, and renamed the `--default-anm-user` option to `--default-user`.
- Added a disclaimer to topics on creating base Docker images that you must ensure all base O/S images are up-to-date and apply any security patches. For more details, see *Step 4 – Create base Docker image* on page 26 and *Deploy API Gateway Analytics in Docker containers* on page 46
- Added a new topic on running API Gateway Analytics in a Docker container. For more details, see *Deploy API Gateway Analytics in Docker containers* on page 46.
- Updated the topic on building an Admin Node Manager Docker image to show the most common use case of building with a domain certificate and metrics enabled. For more details, see *Step 5 – Create an Admin Node Manager Docker image* on page 28.
- Updated the topic on starting the Admin Node Manager and API Gateway Docker containers to show the most common use cases of running with metrics and API Manager enabled. For more details, see *Step 6 – Start the Admin Node Manager Docker container* on page 33 and *Step 8 – Start the API Gateway Docker container* on page 41.
- Updated the topic on adding a docker network with an example API Gateway domain topology. For more details, see *Step 1 – Create a Docker network* on page 21.
- Removed demo-only details on starting Apache Cassandra in a container to refer instead to the Docker website for details on starting Apache Cassandra in a container in a production environment. For more details, see *Step 2 – Start external data stores* on page 22.

- Updated the topic on prerequisites to include details on installing the API Gateway Docker scripts. For more details, see *Before you start* on page 20.

# Changes in version 7.6.1

No changes.

# Changes in version 7.6.0

- This is a new guide that describes how to deploy and run API Gateway and API Manager in containers and elastically scale the capacity up or down as required.

# Elastic deployment in the cloud 1

You can deploy API Gateway and API Manager in Docker containers on a cloud platform hosted by infrastructure-as-a-service (IaaS) providers, and use the elastic capability that their automation tools and techniques provide. This enables you to manage the load on your system easily, adding and removing nodes as needed.

## Deployment process

To deploy your API Gateway or API Manager topology in containers, you must have an automated continuous integration (CI) pipeline spanning all your environments from development to production. The CI pipeline moves a Docker image from one environment to another. However, the image is always specific to a single environment, and you use the promoted image as the base image when creating the Docker image. You must update the configuration files to match the configuration of the environment (for example, certificates, ports, and back-end URLs often change between environments). If you are using API Manager, during promotion APIs are exported from the previous environment, environmentalized, and imported into the new environment.

You can download Docker scripts for API Gateway and Admin Node Manager from Axway Support at https://support.axway.com, and use these scripts to create the Docker images. You can then include your `.fed` and other configuration files in the API Gateway base image in your development environment to create a customized API Gateway Docker image.

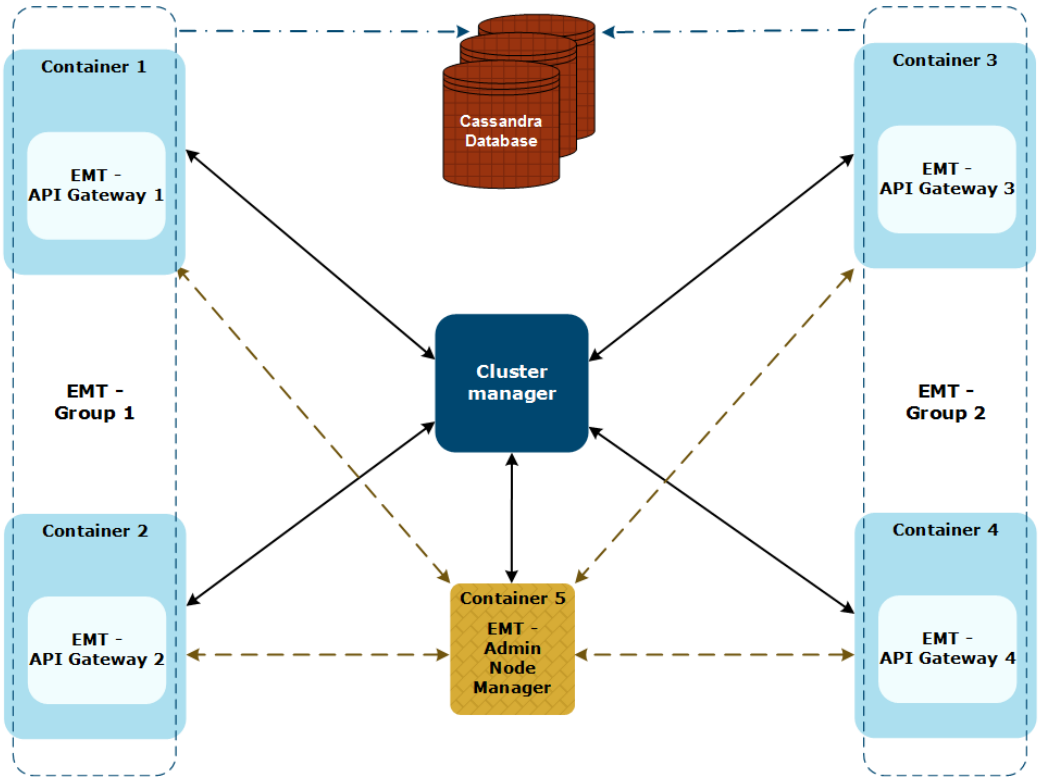
Because the `.fed` files are not separately deployed but included in the Docker image, after the image is created, it is immutable. However, you can create multiple Docker images with different `.fed` files as needed, and deploy them as required. Containers from each image (containing the same `.fed` file) form a group that scales independently from other groups, and you can use a load balancer to route traffic from one group to another.

### Example containerized topology

In the classic, non-containerized deployment, the topology is managed internally through an Admin Node Manager communicating with Node Managers in API Gateway nodes.

However, in a container deployment the topology is externally managed. Instead of the Admin Node Manager, a cluster manager, such as Docker Swarm or Kubernetes, manages the topology and communicates directly with each API Gateway. The API Gateway Docker images do not include a Node Manager.

The following diagram shows an example of the topology in container deployment:




## API Gateway containers

In this example, four API Gateway nodes run in Docker containers:

- The containers have been deployed from two images that use different `.fed` files (`file1.fed` and `file2.fed`), thus forming two groups, `EMT - Group 1` and `EMT - Group 2`.
- Because there are no Node Managers, each API Gateway node exposes the management interface directly on port `8085`.

## Admin Node Manager container

The topology includes an Admin Node Manager, but with a different role than in the classic deployment:

- Admin Node Manager *does not* manage the topology, but is managed by the same cluster manager as the API Gateway nodes.
- Admin Node Manager provides access to monitoring data and runtime settings.

- Admin Node Manager builds a dynamic picture of the topology:
    - New API Gateway nodes automatically start sending transaction event logs to Admin Node Manager when they are started.
    - Admin Node Manager listens to the API Gateway nodes, processes the received logs and writes them to the metrics database.
    - The Admin Node Manager container uses a shared volume to access the transaction event logs from API Gateway containers.

## *Cluster manager*

A cluster manager such as Docker Swarm or Kubernetes manages the topology and adds or removes nodes as the load on the system changes:

- When a node is added, the image for the new node is passed to the cluster manager that starts the required number of containers.
- When a node is removed, the cluster manager waits for the traffic to the node to stop before removing it.

## *Apache Cassandra*

If the deployment includes API Manager, Apache Cassandra is configured for high availability (HA) to store API Manager data and Key Property Store (KPS) tables. For more details, see "Configure a Cassandra HA cluster" in the *API Gateway Apache Cassandra Administrator Guide*.

# Get started with API Gateway in containers 2

This topic is aimed at users of earlier versions of API Gateway who are deploying API Gateway in containers for the first time. It highlights the differences between container deployments (API Gateway 7.6.0 and later) and classic deployments (API Gateway 7.5.3 and earlier) and provides answers to frequently asked questions (FAQ) about deploying API Gateway in containers. It includes the following sections:



## How to deploy API Gateway in Docker containers

In API Gateway version 7.6.0 and later, you can deploy API Gateway in Docker containers and run it in externally managed topology (EMT) mode. This involves creating Docker images for the Admin Node Manager and API Gateway, and starting Docker containers from those images. For more details, see *Deploy API Gateway in Docker containers* on page 19.

## How container deployments differ from classic deployments

There are significant differences when deploying in containers and running in EMT mode, compared to deploying and running in non-containerized classic mode.

| Container deployment | Classic deployment |
|---|---|
| Docker manages API Gateway topology. | Admin Node Manager manages API Gateway topology. |
| An API Gateway domain comprises one Admin Node Manager and one or more API Gateways. | An API Gateway domain comprises an Admin Node Manager, one or more Node Managers, and one or more API Gateways. |

| Container deployment | Classic deployment |
| --- | --- |
| API Gateway configuration ( `.fed` file) is baked into the API Gateway Docker image. To deploy changes in API Gateway configuration you must export a `.fed` file from Policy Studio, rebuild the API Gateway Docker image, and restart the API Gateway Docker container.<br>**Note** In a development environment, you can use the `EMT_DEPLOYMENT_ENABLED` environment variable when starting an API Gateway container to enable deploying changes directly from Policy Studio to the running container. | To deploy changes in API Gateway configuration you can deploy directly from Policy Studio. |
| You can easily scale a domain up or down by starting or stopping Docker containers. | You cannot easily scale a domain up or down. The Admin Node Manager manages starting and stopping API Gateways. |

# Frequently asked questions about container deployments

## How do you deploy changes in configuration from Policy Studio?

In a development environment only, you can set the `EMT_DEPLOYMENT_ENABLED` environment variable to `true` when starting an API Gateway Docker container. This enables you to deploy changes directly from Policy Studio to the container and to create Policy Studio projects from the running API Gateway container.

In a production environment, API Gateway configuration ( `.fed` file) is baked into the API Gateway Docker image. To deploy changes in API Gateway configuration you must export a `.fed` file from Policy Studio, rebuild the API Gateway Docker image, and restart the API Gateway Docker container.

For more information, see *Development and deployment with API Gateway containers* on page 55.

# How do you promote configuration through environments?

The promotion flow in a container deployment is very similar to a classic deployment, however, because API Gateway configuration ( `.fed` file) is baked into the API Gateway Docker image, you must export the policy package ( `.pol` file) and environment package (`.env` file) from Policy Studio, rebuild the API Gateway Docker image, and restart the API Gateway Docker container, at each stage in the promotion flow.

For more information, see *Development and deployment with API Gateway containers* on page 55.

# How do you promote APIs through environments?

In a container deployment, you can promote APIs registered in API Manager in the same way as in a classic environment. See "Promote managed APIs between environments" in the *API Manager User Guide*.

# What API Gateway Manager functionality is not available?

In a container deployment, you cannot perform the following in API Gateway Manager:

- You cannot create or delete API Gateway groups and instances
- You cannot start or stop API Gateways
- You cannot deploy configuration packages to a group of API Gateway instances

For more information, see *Operate and monitor API Gateway containers* on page 51.

# How do you persist logs?

To persist API Gateway logs to a directory on your host machine, you can use the `docker run -v` option to mount volumes for logs when running the API Gateway Docker container. For an example, see *Mount volumes to persist logs outside the API Gateway container* on page 42.

For more information on using volumes to persist data, see the Use volumes Docker documentation.

# What license do you need to run?

You must have an appropriate license to run API Gateway or API Manager in a Docker container. For more information, see *Before you start* on page 20.

# How do you upgrade from 7.5.3 to 7.6.2?

To upgrade from API Gateway 7.5.3 (classic deployment) to API Gateway 7.6.2 (container deployment), you must first upgrade to a 7.6.2 classic deployment, and then migrate to a 7.6.2 container deployment.

For information on upgrading to 7.6.2, see the *API Gateway Upgrade Guide*, and for more details on migrating to a container deployment, see *Migrate to container deployment* on page 58.

# How do you create API Gateway Docker images with customized configuration?

When building your API Gateway or Admin Node Manager Docker images, you can specify a merge directory containing custom configuration, JAR files, and so on, to add to the Docker image. For detailed examples, see:

- *Create an Admin Node Manager image using existing fed and customized configuration* on page 30
- *Create an API Gateway image using existing fed and customized configuration* on page 37

# How do you manage API Gateway topology?

In a classic deployment, topology is managed internally through an Admin Node Manager communicating with Node Managers in API Gateway nodes. In a container deployment, the topology is externally managed by a cluster manager, such as Docker Swarm or Kubernetes, and it manages the topology and communicates directly with each API Gateway.

This means that you cannot use the `managedomain` script or the API Gateway Manager web UI to manage your topology in a container deployment. Instead, you must use an orchestration tool. For more information on Docker Swarm, see the Getting started with swarm mode Docker documentation, or for more information on Kubernetes go to the Kubernetes website.

# How do you run API Gateway administration tools or scripts?

In a container deployment, you must connect to the running Admin Node Manager Docker container to run API Gateway administration tools, such as `kpsadmin`. For an example, see *Manage KPS with kpsadmin* on page 66.

# How do you set up API Manager metrics?

To set up API Manager metrics you must first create an Admin Node Manager Docker image with metrics processing enabled, and then run the Admin Node Manager and API Gateway Docker containers using the `docker run -v` option to mount a volume for the API Gateway events directory.

When starting the containers, you must also specify the connection details for the metrics database using environment variables. For an example, see *Create a metrics-enabled Admin Node Manager image* on page 29.

# Deploy API Gateway in Docker containers

# 3

This topic describes how to use scripts to build API Gateway and Admin Node Manager Docker images, and how to deploy those images in Docker containers. It also describes how to build and deploy API Gateway Analytics in a Docker container. For more information on Docker, see the Docker user documentation.

The following sections describe the steps you must follow to deploy API Gateway in Docker containers:

## *Set up Docker environment*



## *Create base Docker image*



## *Create and start Admin Node Manager Docker container*



## *Create and start API Gateway Docker container*

# Before you start

Your system must meet the following prerequisites before you can run the scripts to build and deploy API Gateway in Docker containers.

## Set up your Docker environment

You must have the following installed on your local system:

- Docker (see *Docker requirements* on page 20 for supported versions)
- Python version 2.7.x
- OpenSSL version 1.1 or later

### *Docker requirements*

The following versions of Docker are supported:

- Docker CE version 17.09 or later on CentOS 7
- Docker EE version 17.06 or later on RHEL 7

**Note** Axway supports Red Hat Enterprise Linux 7 and CentOS Linux version 7 as the base image for Docker containers. Axway supports deployment on any host operating system, cloud provider, or container orchestration system supported by your Docker version.

For more details on Docker system requirements, see the Docker documentation.

## Set up API Gateway Docker scripts

You must download the following from Axway Support at https://support.axway.com:

- API Gateway Linux installer
- Docker scripts package

### *API Gateway licenses*

You must have specific API Gateway licenses to run the following:

- API Gateway container
- Admin Node Manager or API Gateway container in Federal Information Processing Standard (FIPS) mode
- API Manager-enabled API Gateway container
- API Gateway Analytics container

### *Unzip and install the Docker scripts*

Unzip the Docker scripts package that you downloaded from Axway Support at https://support.axway.com. For example:

```
$ unzip APIGateway_7.6.2-<n>_ScriptsPackageDocker_linux-x86-64_BN<bn>.zip
```

The unzipped package includes the following:

- Python scripts (`*.py`)
- Docker files
- Quickstart demo (see `readme.md`)

# Step 1 - Create a Docker network

You must run the `docker network` command to create a Docker network for the API Gateway domain. This enables all of the containers in the domain to communicate with each another easily (for example, the API Gateway container and Admin Node Manager container). A containerized API Gateway domain must include one Admin Node Manager container and one or more API Gateway containers.

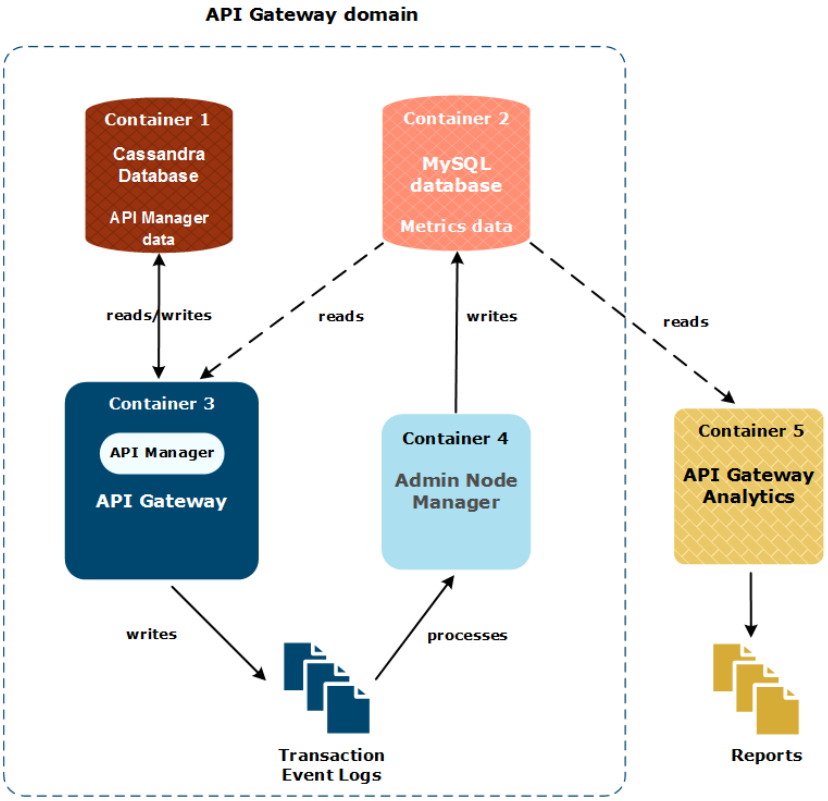
Run the `docker network` command as follows:

```
$ docker network create api-gateway-domain
```

This example creates a Docker network called `api-gateway-domain`. For more details on the `docker network` command, see the Docker user documentation.

# Example API Gateway domain

The example Quick Start API Gateway domain topology provided with the API Gateway Docker scripts includes the following Docker containers:




This simple API Gateway domain topology is described as follows:

- Container 1 runs an Apache Cassandra database that stores API Manager data.
- Container 2 runs a MySQL database that stores API Gateway metrics data.
- Container 3 runs an API Gateway that writes transaction event logs, and includes API Manager is an optional component. For more details, see *Deploy API Manager or OAuth in Docker containers* on page 43.
- Container 4 runs an Admin Node Manager that generates metrics from transaction event logs, which are then read from the metrics database by API Manager and API Gateway Analytics.
- Container 5 runs API Gateway Analytics, which is an optional standalone client of the metrics database. For more details, see *Deploy API Gateway Analytics in Docker containers* on page 46.

**Note** The example Quick Start domain topology is suitable for a development environment only. For more details, see the readme file provided with the API Gateway Docker scripts.

# Step 2 - Start external data stores

If you are using any external data stores, such as Apache Cassandra for API Manager, or a metrics database for API Manager or API Gateway Analytics, you must start these data stores.

## Start Apache Cassandra

**Note** Deploying a Cassandra container is only recommended for development environments. In a production environment, you must configure Cassandra for high availability (HA) as detailed in "Configure a Cassandra HA cluster" in the *API Gateway Apache Cassandra Administrator Guide*.

For details on starting Apache Cassandra in a Docker container, see https://hub.docker.com/_/cassandra.

## Start the metrics database

If you are using a metrics database, you can use the `docker run` command to start a database container. For example:

```
$ cd emt_containers-<version>
$ cp quickstart/mysql-analytics.sql /tmp/sql
$ docker run -d --name metricsdb --network=api-gateway-domain
-v /tmp/sql:/docker-entrypoint-initdb.d
-e MYSQL_ROOT_PASSWORD=root01 -e MYSQL_DATABASE=metrics
mysql:5.7
```

This example performs the following:

- Downloads a MySQL 5.7 Docker image from the public Docker registry.
- Mounts the host directory `/tmp/sql` (containing a MySQL metrics database creation script) inside the container.
- Uses environment variables `MYSQL_ROOT_PASSWORD` and `MYSQL_DATABASE` to specify the database root password and the database name.
- Starts a MySQL container named `metricsdb`.

# Step 3 – Generate domain SSL certificates

To secure the communications between the Admin Node Manager and API Gateways, you can use the `gen_domain_cert.py` script to generate a self-signed CA certificate for a domain, or a Certificate Signing Request (CSR) to be signed by an external Certificate Authority (CA).

**Note** If you already have a domain certificate (for example, from another API Gateway installation) you can skip this section. You can specify your existing domain certificate (certificate and private key in .pem format) to the `build_anm_image.py` and `build_gw_image.py` scripts. You cannot use one domain certificate for both a container deployment and a classic deployment if they are running in parallel.

# Generate domain certificates script options

You must specify the following as options when using the `gen_domain_cert.py` script:

- Domain identifier
- Passphrase for the domain private key

This script also supports additional options when generating certificates. For example:

- Specify a signing algorithm (SHA256, SHA384, or SHA512)
- Generate a CSR
- Specify custom values for the fields in the domain certificate (for example, organization)

For the latest script usage and options, run the script with no options, or with the `-h` option. For example:

```
$ cd emt_containers-<version>
$ ./gen_domain_cert.py -h
```

The following examples show how you can use the script to generate domain certificates:

- *Generate a default certificate and key* on page 24
- *Generate a self-signed certificate and key* on page 25
- *Generate a CSR* on page 25

# Generate a default certificate and key

The following example creates a certificate and private key using default values.

## *Usage guidelines*

- Do not use default options on production systems. The `--default-cert` option is provided only as a convenience for development environments.

## *Example command*

```
$ cd emt_containers-<version>
$ ./gen_domain_cert.py --default-cert
```

This example creates a default certificate and private key:

- The certificate uses a domain identifier of `DefaultDomain`
- The certificate and key are stored in the `certs/DefaultDomain` directory

- The certificate uses a default passphrase

# Generate a self-signed certificate and key

The following example creates a self-signed certificate and private key.

## *Example command*

```
$ cd emt_containers-<version>
$ ./gen_domain_cert.py --domain-id=mydomain --pass-file=/tmp/pass.txt
```

This example creates a self-signed certificate and private key:

- The certificate uses a domain identifier of `mydomain`.
- The certificate and key are stored in the `certs/mydomain` directory
- The certificate uses a specified passphrase

# Generate a CSR

The following example creates a certificate signing request (CSR).

## *Usage guidelines*

- You must send the generated CSR to a CA for signing.
- When running the scripts to build Admin Node Manager or API Gateway images, specify the certificate and private key returned from the CA, and not the CSR.

## *Example command*

```
$ cd emt_containers-<version>
$ ./gen_domain_cert.py --domain-id=mydomain --pass-file=/tmp/pass.txt --out=csr --
O=MyOrg
```

This example creates a CSR that:

- Uses a domain identifier of `mydomain`
- Is stored in the `certs/mydomain` directory
- Uses a specified passphrase and organization

# Step 4 - Create base Docker image

To create a base Docker image containing an operating system and an API Gateway installation, use the `build_base_image.py` script. This script builds a base API Gateway Docker image using an API Gateway7.6.2 Linux installer and a Docker image based on a standard or custom CentOS7 or RHEL7 operating system image.

**Caution** Docker automatically downloads the latest CentOS or RHEL7 image, which may potentially contain security vulnerabilities. Axway is not responsible for any third-party base O/S images. You must ensure that all base O/S images are up-to-date and apply any security patches if necessary.

## Base image script options

You must specify the following as options when using the `build_base_image.py` script:

- API Gateway 7.6.2 Linux installer
- Operating system based on one of the following:
  - Standard CentOS7 Docker image downloaded from the public Docker registry
  - Standard RHEL7 Docker image downloaded from the Red Hat Docker registry
  - If you specify a custom CentOS7 or RHEL7-based OS Docker image, Docker first tries to find the custom image in the local registry, and then tries to download it from a remote registry

**Note** You must have an RHEL7 license to build a base API Gateway image based on an RHEL7 OS.

This script also supports additional options when generating a base image. For the latest script usage and options, run the script with no options, or with the `-h` option. For example:

```
$ cd emt_containers-<version>
$ ./build_base_image.py -h
```

The following examples show how you can use the script to build base Docker images:


- *Create a base image based on standard CentOS7* on page 26
- *Create a base image based on standard RHEL7* on page 27
- *Create a base image based on custom CentOS7/RHEL7* on page 27


## Create a base image based on standard CentOS7

The following example creates a base Docker image using a standard CentOS7 image.

### Example command

```
$ cd emt_containers-<version>
$ ./build_base_image.py
--installer=apigw-installer.run
--os=centos7
```

This example creates a base Docker image named `apigw-base` with a tag of `latest`. The image is based on a standard CentOS7 Docker image.

## Create a base image based on standard RHEL7

The following example creates a base Docker image using a standard RHEL7 image.

### Usage guidelines

- You must have an RHEL7 license to build a base API Gateway image based on an RHEL7 OS.

### Example command

```
$ cd emt_containers-<version>
$ ./build_base_image.py
--installer=apigw-installer.run
--os=rhel7 --out-image=my-gw-base:1.0
```

This example creates a base Docker image named `my-gw-base` with a tag of `1.0`. The image is based on a standard RHEL7 Docker image.

## Create a base image based on custom CentOS7/RHEL7

The following example creates a base Docker image using a custom RHEL7 image.

### Usage guidelines

- You must have an RHEL7 license to build a base API Gateway image based on an RHEL7 OS.

### *Example command*

```
$ cd emt_containers-<version>
$ ./build_base_image.py
--installer=apigw-installer.run
--parent-image=my-custom-rhel:2.0
--out-image=my-custom-gw-base:1.0
```

This example creates a base Docker image named `my-custom-gw-base` with a tag of `1.0`. The image is based on the specified `my-custom-rhel:2.0` Docker image.

# Step 5 - Create an Admin Node Manager Docker image

To create an Admin Node Manager Docker image, use the `build_anm_image.py` script. This script builds an Admin Node Manager Docker image using the base image you created in *Step 4 – Create base Docker image* on page 26.

## Admin Node Manager image script options

You must specify the following as options when using the `build_anm_image.py` script:

- Domain certificate, private key, and password.
- User name and password for the administrator user. You can use this user name and password to log in to the API Gateway Manager web console.

This script also supports additional options when generating an Admin Node Manager image. For example:

- Enable API metrics processing in the Admin Node Manager Docker image. This enables you to monitor APIs and applications in API Manager.
- Reuse the same configuration in multiple domains by specifying a `.fed` file containing Admin Node Manager configuration to include in the Admin Node Manager Docker image.
- Specify a merge directory to add to the Admin Node Manager Docker image. This directory can include custom configuration, JAR files, and so on.
- Enable FIPS mode for the Admin Node Manager Docker image.

For the latest script usage and options, run the script with no options, or with the `-h` option. For example:

```
$ cd emt_containers-<version>
$ ./build_anm_image.py -h
```

# Create a metrics-enabled Admin Node Manager image

The following example creates an Admin Node Manager Docker image with a specified domain certificate that runs with metrics processing enabled. The Admin Node Manager container processes event logs from API Gateway containers and writes them to a specified metrics database. This is the recommended option and is suitable for a production environment.

## *Usage guidelines*

- Use the `--merge-dir` option to specify the `apigateway` directory containing the JDBC driver JAR file for the metrics database in the `ext/lib` directory:
    - The merge directory must be called `apigateway` and must have the same directory structure as in an API Gateway installation.
    - Copy the JAR file to a new directory `/tmp/apigateway/ext/lib/` and specify `/tmp/apigateway` to the `--merge-dir` option.
- When running the Admin Node Manager and API Gateway Docker containers, use the `docker run -v` option to mount a volume for the API Gateway events directory:
    - Run the API Gateway container with a volume mounted for the events directory (for example, `-v /tmp/events:/opt/Axway/apigateway/events` writes API Gateway event logs to `/tmp/events` on the host machine).
    - Run the Admin Node Manager container with the same volume mounted (for example, `-v /tmp/events:/opt/Axway/apigateway/events` enables the Admin Node Manager to read API Gateway event logs from `/tmp/events` on the host machine). For details, see *Start a metrics-enabled Admin Node Manager container* on page 33.
- Use the metrics options to specify the URL, user name, and password for your metrics database. If not specified, the metrics options have the following default values:
    - `--metrics-db-url`: Defaults to `${environment.METRICS_DB_URL}`
    - `--metrics-db-username`: Defaults to `${environment.METRICS_DB_USERNAME}`
    - `--metrics-db-pass-file`: Default value for password if password file not specified is `${environment.METRICS_DB_PASS}`

**Note** When running in a multi-node system, you must mount a shared network volume that is accessible from the Admin Node Manager and from all API Gateways.

### *Example command*

```
$ cd emt_containers-<version>
$ ./build_anm_image.py
--domain-cert=certs/mydomain/mydomain-cert.pem
--domain-key=certs/mydomain/mydomain-key.pem
--domain-key-pass-file=/tmp/pass.txt
--anm-username=gwadmin --anm-pass-file=/tmp/gwadminpass.txt
--parent-image=my-gw-base:1.0 --out-image=my-metrics-admin-node-manager:1.0
--metrics --merge-dir=/tmp/apigateway
```

This example creates an Admin Node Manager Docker image named `my-metrics-admin-node-manager` with a tag of `1.0`. This image has the following characteristics:

- Based on the `my-gw-base:1.0` image
- Uses a specified domain certificate, key, and password
- Uses a specified user name of `gwadmin` and a specified password for the administrator user
- Runs with metrics processing enabled
- Uses a specified merge directory (containing the JDBC driver JAR file for the metrics database) that is merged into the API Gateway image

## Other options for creating Admin Node Manager Docker images

The following are additional examples of using the `build_anm_image.py` script to build Admin Node Manager Docker images:

- *Create an Admin Node Manager image using existing fed and customized configuration* on page 30
- *Create a FIPS-enabled Admin Node Manager image* on page 32
- *Create an Admin Node Manager image for a development environment* on page 32

### *Create an Admin Node Manager image using existing fed and customized configuration*

The following example creates an Admin Node Manager Docker image using an existing Admin Node Manager deployment package (`.fed` file) and customized configuration from an existing API Gateway installation.

## Usage guidelines

- Ensure that your `.fed` contains the following:
  - Admin Node Manager configuration. You can open the `.fed` in Policy Studio and verify that it is identified as a Node Manager configuration in the navigation pane.
  - Only IP addresses that are accessible at runtime. For example, the `.fed` cannot contain IP addresses of container-based Admin Node Managers and API Gateways, because IP addresses are usually dynamically assigned in a Docker network.
- Use the `--merge-dir` option to add more files and folders to the `apigateway` directory inside the image:
  - The merge directory must be called `apigateway` and must have the same directory structure as in an API Gateway installation.

    For example, to add an optional custom `envSettings.props` file to your image, copy `envSettings.props` to a new directory named `/tmp/apigateway/conf/`, and specify `/tmp/apigateway` to the `--merge-dir` option.
  - To add custom JAR files to your image, copy the JAR files to a new directory named `/tmp/apigateway/ext/lib/`, and specify `/tmp/apigateway` to the `--merge-dir` option.

    **Note** `envSettings.props` specifies settings such as the port the Admin Node Manager listens on (default of `8090`), and the session timeout for API Gateway Manager (default of 12 hours). `envSettings.props` must contain only IP addresses and host names that are accessible at runtime. It cannot contain IP addresses of container-based Admin Node Managers and API Gateways because these are usually dynamically assigned in a Docker network.

## Example command

```
$ cd emt_containers-<version>
$ ./build_anm_image.py
--domain-cert=certs/mydomain/mydomain-cert.pem
--domain-key=certs/mydomain/mydomain-key.pem
--domain-key-pass-file=/tmp/pass.txt
--anm-username=gwadmin --anm-pass-file=/tmp/gwadminpass.txt
--parent-image=my-gw-base:1.0 --out-image=my-fed-admin-node-manager:1.0
--fed=my-anm-fed.fed --fed-pass-file=/tmp/anmfedpass.txt
--merge-dir=/tmp/apigateway
```

This example creates an Admin Node Manager Docker image named `my-fed-admin-node-manager` with a tag of `1.0`. This image has the following characteristics:

- Based on the `my-gw-base:1.0` image
- Uses a specified certificate and key
- Uses a specified user name of `gwadmin` and a specified password for the administrator user
- Uses a specified `.fed` that contains Admin Node Manager configuration
- Uses a specified merge directory that is merged into the Admin Node Manager image

## Create a FIPS-enabled Admin Node Manager image

The following example creates an Admin Node Manager Docker image that runs in FIPS-compliant mode.

## Usage guidelines

- You must have a valid FIPS-compliant mode API Gateway license file to create an image that can run in FIPS-compliant mode.

## Example command

```
$ cd emt_containers-<version>
$ ./build_anm_image.py
--domain-cert=certs/mydomain/mydomain-cert.pem
--domain-key=certs/mydomain/mydomain-key.pem
--domain-key-pass-file=/tmp/pass.txt
--anm-username=gwadmin --anm-pass-file=/tmp/gwadminpass.txt
--parent-image=my-gw-base:1.0 --out-image=my-fips-admin-node-manager:1.0
--fips --license=/tmp/api_gw_fips.lic
```

This example creates an Admin Node Manager Docker image named `my-fips-admin-node-manager` with a tag of `1.0`. This image has the following characteristics:

- Based on the `my-gw-base:1.0` image
- Uses a specified domain certificate, key, and password
- Uses a specified user name of `gwadmin` and a specified password for the administrator user
- Runs in FIPS-compliant mode

## Create an Admin Node Manager image for a development environment

The following example creates a simple Admin Node Manager Docker image suitable for a development environment only using default certificates and a default administrator user.

### Usage guidelines

- Do not use default options on production systems. The `--default-cert` and `--default-user` options are provided only as a convenience for development environments.

### Example command

```
$ cd emt_containers-<version>
$ ./build_anm_image.py
--default-cert --default-user
```

This example creates an Admin Node Manager Docker image named `admin-node-manager` with a tag of `latest`. This image has the following characteristics:

- Based on the `apigw-base:latest` image
- Uses a default domain certificate and key (generated from running `./gen_domain_cert.py --default-cert`)
- Uses a default user name of `admin` and a default password for the administrator user

# Step 6 – Start the Admin Node Manager Docker container

Use the `docker run` command to start the Admin Node Manager container. This topic explains the required command options for starting a metrics-enabled Admin Node Manager container.

## Start a metrics-enabled Admin Node Manager container

The following example shows how to run a metrics-enabled Admin Node Manager container in the background on a specific port:

```
$ docker run -d -p 8090:8090 --name=anm --network=api-gateway-domain
-v /tmp/events:/opt/Axway/apigateway/events
-e METRICS_DB_URL=jdbc:mysql://metricsdb:3306/metrics?useSSL=false
-e METRICS_DB_USERNAME=db_user1 -e METRICS_DB_PASS=my_db_pwd admin-node-manager:1.0
```

This example performs the following:

- Starts an Admin Node Manager container named `anm` from an image named `admin-node-manager:1.0`. You must specify the name of the image that you created in *Step 5 – Create an Admin Node Manager Docker image* on page 28.

- Binds the default management port `8090` of the container to port `8090` on the host machine. This enables you to access the API Gateway Manager web console on port `8090` of your host machine.
- Runs the container in the background using the `-d` option.
- Mounts the `/tmp/events` host directory in the container, which contains API Gateway transaction event logs.
- Uses environment variables to specify connection details for the metrics database.

### *Further information*

For more details on the `docker run` command, see the Docker user documentation.

# Step 7 - Create an API Gateway Docker image

To create an API Gateway Docker image, use the `build_gw_image.py` script. This script builds an API Gateway Docker image using the base image you created in *Step 4 – Create base Docker image* on page 26.

## Build API Gateway image script options

You must specify the following as options when using the `build_gw_image.py` script:

- Domain certificate, private key, and password.
- API Gateway license. Your license must also include any optional licensed features that you are using (for example, API Manager, FIPS mode).

This script also supports additional options when generating an API Gateway image. For example, you can:

- Specify a group ID for the API Gateway group. All containers started from this image are part of this API Gateway group.
- Build an image from existing API Gateway configuration by specifying an existing `fed` file (or existing `pol` and `env` files). If OAuth or API Manager are enabled in the `fed`, they are enabled the API Gateway Docker image.
- Specify a merge directory to add to the API Gateway Docker image. This merge directory can include custom configuration, JAR files, and so on.
- Enable FIPS mode for the API Gateway Docker image.

For the latest script usage and options, run the script with no options, or with the `-h` option. For example:

```
$ cd emt_containers-<version>
$ ./build_gw_image.py -h
```

The following examples show how you can use the script to build API Gateway Docker images:



# Create an API Gateway image using defaults

The following example creates an API Gateway Docker image using default certificates and a default factory `fed`.

## Usage guidelines

- Do not use default options on production systems. The `--default-cert` option is provided only as a convenience for development environments.

## Example command

```
$ cd emt_containers-<version>
$ ./build_gw_image.py
--license=/tmp/api_gw_license_complete.lic
--default-cert
--factory-fed
```

This example creates an API Gateway Docker image named `api-gateway-defaultgroup` with a tag of `latest`. This image has the following characteristics:

- Uses a default certificate and key (generated from running `./gen_domain_cert.py --default-cert`)
- Uses a default factory `fed`

# Create an API Manager image using defaults

The following example creates an API Manager Docker image using default certificates and a default factory `fed` with samples.

## Usage guidelines

- Do not use default options on production systems. The `--default-cert` and `--api-manager` options are provided only as a convenience for development environments.
- When using the `--api-manager` default option:
  - You must have an Apache Cassandra server running at the host name specified by `${environment.CASS_HOST}`.
  - You must have a metrics database running at `${environment.METRICS_DB_URL}`, with credentials of `${environment.METRICS_DB_USERNAME}` and `${environment.METRICS_DB_PASS}`.
  - You can log in to the API Manager web console using a default user name of `apiadmin` and the default password.
- You must have a valid API Manager license file to create an API Manager image.
- Use the `--merge-dir` option to specify the `apigateway` directory containing the JDBC driver JAR file for the metrics database in the `ext/lib` directory:
  - The merge directory must be called `apigateway` and must have the same directory structure as in an API Gateway installation.
  - Copy the JAR file to a new directory `/tmp/apigateway/ext/lib/` and specify `/tmp/apigateway` to the `--merge-dir` option.

## Example command

```
$ cd emt_containers-<version>
$ ./build_gw_image.py
--license=/tmp/api_gw_api_mgr.lic
--merge-dir /tmp/apigateway
--default-cert --api-manager
```

This example creates an API Gateway Docker image named `api-gateway-defaultgroup` with a tag of `latest`. This image has the following characteristics:

- Uses a default certificate and key (generated from running `./gen_domain_cert.py --default-cert`)
- Uses a default factory `fed` with samples and with API Manager configured
- Uses a specified merge directory (containing the JDBC driver JAR file for the metrics database) that is merged into the API Gateway image

# Create an API Gateway image using domain certificate

The following example creates an API Gateway Docker image using a specified domain certificate and a default factory `fed`.

## *Example command*

```
$ cd emt_containers-<version>
$ ./build_gw_image.py
--license=/tmp/api_gw_license_complete.lic
--domain-cert=certs/mydomain/mydomain-cert.pem
--domain-key=certs/mydomain/mydomain-key.pem
--domain-key-pass-file=/tmp/pass.txt
--factory-fed
--parent-image=my-gw-base:1.0 --out-image=my-api-gateway:1.0
```

This example creates an API Gateway Docker image named `my-api-gateway` with a tag of `1.0`. This image has the following characteristics:

- Based on the `my-gw-base:1.0` image
- Uses a specified certificate and key
- Uses a default factory `fed`

# Create an API Gateway image using existing fed and customized configuration

The following example creates an API Gateway Docker image using an existing API Gateway deployment package (`fed` file) and customized configuration from an existing API Gateway installation.

## *Usage guidelines*

- Ensure that your `fed` contains the following:
    - API Gateway version 7.6.2 configuration.

      You can upgrade existing projects (from version 7.5.1 or later) using `projupgrade`, see "Upgrade an API Gateway project" in the *API Gateway DevOps Deployment Guide*.

      You can also upgrade existing `fed` files using Policy Studio or `upgradeconfig`, see the *API Gateway Upgrade Guide*.

    - Only IP addresses that are accessible at runtime. For example, the `fed` cannot contain IP addresses of container-based Admin Node Managers and API Gateways, as IP addresses are usually dynamically assigned in a Docker network.
- Use the `--merge-dir` option to add more files and folders to the `apigateway` directory inside the image:
    - The merge directory must be called `apigateway` and must have the same directory structure as in an API Gateway installation.

      For example, to add an optional custom `envSettings.props` file to your image, copy `envSettings.props` to a new directory named `/tmp/apigateway/groups/emt-group/emt-service/conf/`, and specify `/tmp/apigateway` to the `--merge-dir` option.
    - To add custom JAR files to your image, copy the JAR files to a new directory named `/tmp/apigateway/ext/lib/`, and specify `/tmp/apigateway` to the `--merge-dir` option.

      **Note** `envSettings.props` specifies settings such as the port the Admin Node Manager listens on (default of `8090`), and the session timeout for API Gateway Manager (default of 12 hours). `envSettings.props` must contain only IP addresses and host names that are accessible at runtime. It cannot contain IP addresses of container-based Admin Node Managers and API Gateways because these are usually dynamically assigned in a Docker network.

## Example command

```
$ cd emt_containers-<version>
$ ./build_gw_image.py
--license=/tmp/api_gw.lic
--domain-cert=certs/mydomain/mydomain-cert.pem
--domain-key=certs/mydomain/mydomain-key.pem
--domain-key-pass-file=/tmp/pass.txt
--parent-image=my-gw-base:1.0
--fed=my-group-fed.fed --fed-pass-file=/tmp/my-group-fedpass.txt
--group-id=my-group
--merge-dir=/tmp/apigateway
```

This example creates an API Gateway Docker image named `api-gateway-my-group` with a tag of `latest`. This image has the following characteristics:

- Based on the `my-gw-base:1.0` image.
- Uses a specified certificate and key.
- Uses a specified `fed` that contains API Gateway 7.6.2 configuration.
- Belongs to the API Gateway group `my-group`. All containers started from this image belong to

this group.

- Uses a specified merge directory that is merged into the API Gateway image.

# Create a FIPS-enabled API Gateway image

The following example creates an API Gateway Docker image that runs in FIPS-compliant mode.

## *Usage guidelines*

- You must have a valid FIPS-compliant mode API Gateway license file to create an image that can run in FIPS-compliant mode.

## *Example command*

```
$ cd emt_containers-<version>
$ ./build_gw_image.py
--license=/tmp/api_gw_fips.lic
--domain-cert=certs/mydomain/mydomain-cert.pem
--domain-key=certs/mydomain/mydomain-key.pem
--domain-key-pass-file=/tmp/pass.txt
--parent-image=my-gw-base:1.0 --out-image=my-fips-api-gateway:1.0
--fips
```

This example creates an API Gateway Docker image named `my-fips-api-gateway` with a tag of `1.0`. This image has the following characteristics:

- Based on the `my-gw-base:1.0` image.
- Uses a specified certificate and key.
- Runs in FIPS-compliant mode.

# Create an API Manager or OAuth enabled API Gateway image

The following example creates an API Manager enabled API Gateway Docker image using a deployment package exported from Policy Studio that has API Manager configured.

You can create an OAuth-enabled API Gateway Docker image in the same way (using a deployment package exported from Policy Studio that has OAuth configured).

## *Usage guidelines*

To create an API Manager enabled image:

- You must have a valid API Manager license file to create an API Manager image.
- Use the `--merge-dir` option to specify the `apigateway` directory containing the JDBC driver JAR file for the metrics database in the `ext/lib` directory:
    - The merge directory must be called `apigateway` and must have the same directory structure as in an API Gateway installation.
    - Copy the JAR file to a new directory `/tmp/apigateway/ext/lib/` and specify `/tmp/apigateway` to the `--merge-dir` option.
- Before running the `build_gw_image.py` script you must first create a project in Policy Studio, configure API Manager in that project, and export the configuration from Policy Studio as a `fed` file (or `pol` and `env` files). For more information, see *Step 1 – Configure API Manager in Policy Studio* on page 43.
- You must specify the configuration exported from Policy Studio to the `build_gw_image.py` script when building the API Gateway Docker image.

To create an OAuth-enabled image:

- Before running the `build_gw_image.py` script you must first create a project in Policy Studio, configure OAuth in that project, and export the configuration from Policy Studio as a `fed` file (or `pol` and `env` files). For more information, see *Step 1 – Configure OAuth in Policy Studio* on page 44.
- You must specify the configuration exported from Policy Studio to the `build_gw_image.py` script when building the API Gateway Docker image.

## Example command

```
$ cd emt_containers-<version>
$ ./build_gw_image.py
--license=/tmp/api_gw_api_mgr.lic
--domain-cert=certs/mydomain/mydomain-cert.pem
--domain-key=certs/mydomain/mydomain-key.pem
--domain-key-pass-file=/tmp/pass.txt
--parent-image=my-gw-base:1.0
--fed=api-mgr-group-fed.fed --fed-pass-file=/tmp/api-mgr-group-fedpass.txt
--group-id=api-mgr-group
--merge-dir=/tmp/apigateway
```

This example creates an API Gateway Docker image named `api-gateway-api-mgr-group` with a tag of `latest`. This image has the following characteristics:

- Based on the `my-gw-base:1.0` image.
- Uses a specified certificate and key.
- Uses a specified `fed` that contains API Manager configuration that was exported from Policy Studio.

- Belongs to the API Gateway group `api-mgr-group`. All containers started from this image belong to this group.
- Uses a specified merge directory (containing the JDBC driver JAR file for the metrics database) that is merged into the API Gateway image

# Step 8 - Start the API Gateway Docker container

Use the `docker run` command to start the API Gateway container. For example:

```
$ docker run -d --name=apimgr --network=api-gateway-domain
  -p 8075:8075 -p 8065:8065 -p 8080:8080
  -v /tmp/events:/opt/Axway/apigateway/events
  -e EMT_ANM_HOSTS=anm:8090 -e CASS_HOST=casshost1
  -e METRICS_DB_URL=jdbc:mysql://metricsdb:3306/metrics?useSSL=false
  -e METRICS_DB_USERNAME=root -e METRICS_DB_PASS=my_db_pwd
   api-gateway-my-group:1.0
```

This example performs the following:

- Starts an API Manager-enabled container from an image named `api-gateway-my-group:1.0`. You must specify the name of the API Gateway Docker image that you created in *Step 7 – Create an API Gateway Docker image* on page 34.
- Runs the container in the background using the `-d` option.
- Binds the default traffic port `8080` of the container to port `8080` on the host machine, which enables you to test the API Gateway on your host machine.
- Mounts the `/tmp/events` host directory in the container using the `-v` option. This directory contains API Gateway transaction event logs. For more details, see *Mount volumes to persist logs outside the API Gateway container* on page 42.
- Uses environment variables to specify connection details for the metrics database and the Apache Cassandra host used to store the API Manager data.
- Sets the `EMT_ANM_HOSTS` environment variable to `anm:8090` in the container. This enables the API Gateway to communicate with the Admin Node Manager container on port `8090`. The API Gateway is now visible in the API Gateway Manager topology view. For example:

# Mount volumes to persist logs outside the API Gateway container

You can persist API Gateway trace and event logs to a directory on your host machine. For example, run the following `docker run` command to start an API Gateway container from an image named `api-gateway-my-group:1.0` and mount volumes for trace and event logs:

```
$ docker run -it -v /tmp/events:/opt/Axway/apigateway/events
-v /tmp/trace:/opt/Axway/apigateway/groups/emt-group/emt-service/trace
-e EMT_ANM_HOSTS=anm:8090 -p 8080:8080 --network=api-gateway-domain api-gateway-my-
group:1.0
```

This example starts the API Gateway container and writes the trace and log files to `/tmp/events` and `/tmp/trace` on your host machine. The trace and log files contain the container ID of the API Gateway container in the file names.

**Note** To enable an Admin Node Manager container to process the event logs from API Gateway containers, you must run the Admin Node Manager container with the same volume mounted. For more details, see *Create a metrics-enabled Admin Node Manager image* on page 29 and *Start a metrics-enabled Admin Node Manager container* on page 33.

# Start a deployment-enabled API Gateway container in a development environment

The following simple example sets the `EMT_DEPLOYMENT_ENABLED` environment variable to `true` to enable you to deploy configuration directly from Policy Studio to the running API Gateway container:

```
$ docker run -d -e EMT_DEPLOYMENT_ENABLED=true -e EMT_ANM_HOSTS=anm:8090
-p 8080:8080 --network=api-gateway-domain api-gateway-my-group:1.0
```

**Caution** The `EMT_DEPLOYMENT_ENABLED` environment variable is provided as a convenience for development environments only:

- Do not set `EMT_DEPLOYMENT_ENABLED=true` on production systems. In production environments, to deploy changes in API Gateway configuration, you must export a `.fed` file from Policy Studio, rebuild the API Gateway Docker image, and restart the API Gateway Docker container.
- The `EMT_DEPLOYMENT_ENABLED=true` setting only enables you to deploy changes to a running container from Policy Studio. You cannot deploy changes using the API Gateway Manager, `managedomain`, or `projdeploy` tools.

# Deploy API Manager or OAuth in Docker containers 4

This topic describes how to deploy API Manager or OAuth services in your API Gateway containers. These steps are optional and only for users who wish to use API Manager or OAuth in their environment.

**Tip** For details on how to deploy API Manager or OAuth services in a classic deployment (non-containerized), see the following:

- "Enable API Manager" in the *API Manager User Guide*
- "Deploy OAuth configuration" in the *API Gateway OAuth User Guide*

## Deploy API Manager

To deploy API Manager in a Docker container, follow these steps:


- *Step 1 – Configure API Manager in Policy Studio* on page 43
- *Step 2 – Deploy API Manager enabled API Gateway container* on page 44


### Step 1 - Configure API Manager in Policy Studio

Follow these steps:

1. Open Policy Studio and open or create a new project.
2. Select **File > Configure API Manager**.
3. If you do not have any Cassandra hosts configured, you must add a Cassandra host before you can continue:
    - Enter a name for the Cassandra server (for example, `container_cassandra`).
    - Enter the name of the Cassandra container as the host name (for example, `cassandra228`).
    - Enter the port of the Cassandra container (for example, `9042`).
4. Click **Next**.
5. Enter the appropriate API Manager settings. For full details, see "Enable API Manager" in the *API Manager User Guide*.

**Note** The default API administrator user name and password set in Policy Studio are used only when creating the administrator account in Apache Cassandra. After the account has been created in Cassandra, you cannot change the credentials in Policy Studio. You must use API Manager to change the administrator credentials. You can also reset the administrator password by running the `setup-apimanager` script with the option `--resetPassword` inside the Admin Node Manager container. For details, see *Reset the default API administrator password* on page 66.

6. Click **Finish**.
7. Configure additional API Manager settings under **Server Settings > API Manager**. For example, you can specify custom policies that are called as traffic passes through API Manager.
8. Select **File > Export** and select a package to export the configuration as a package (`fed`, `pol`, or `env`).

## Step 2 - Deploy API Manager enabled API Gateway container

Follow the steps in *Deploy API Gateway in Docker containers* on page 19. When creating the API Gateway Docker image using `build_gw_image.py`, specify the deployment package you exported from Policy Studio. For an example, see *Create an API Manager or OAuth enabled API Gateway image* on page 39.

# Deploy OAuth services

To deploy OAuth services in a Docker container, follow these steps:



## Step 1 - Configure OAuth in Policy Studio

Follow these steps:

1. Open Policy Studio and open or create a new project.
2. Select **File > Configure OAuth**.
3. If you do not have any Cassandra hosts configured, you must add a Cassandra host before you can continue:
   - Enter a name for the Cassandra server (for example, `container_cassandra`).
   - Enter the name of the Cassandra container as the host name (for example, `cassandra228`).
   - Enter the port of the Cassandra container (for example, `9042`).

4. Click **Next**.
5. Select the OAuth deployment type. For full details, see "Deploy OAuth configuration" in the *API Gateway OAuth User Guide*.
6. Click **Finish**.
7. Select **File > Export** and select a package to export the configuration as a package (`fed`, `pol`, or `env`).

**Note** When you configure OAuth in Policy Studio, this does not register the sample client applications in the Client Application Registry. You must import the sample client applications manually, as detailed in "Import sample client applications" in the *API Gateway OAuth User Guide*.

# Step 2 – Deploy OAuth-enabled API Gateway container

Follow the steps in *Deploy API Gateway in Docker containers* on page 19. When creating the API Gateway Docker image using `build_gw_image.py`, specify the deployment package you exported from Policy Studio. For an example, see *Create an API Manager or OAuth enabled API Gateway image* on page 39.

# Deploy API Gateway Analytics in Docker containers 5

This topic describes the steps to create an API Gateway Analytics Docker image and how to start an API Gateway Analytics Docker container. These steps are optional and only for users who wish to use API Gateway Analytics monitoring and reporting in their environment. In a containerized deployment, API Gateway Analytics runs as a standalone client of the metrics database.

**Tip** For details on how to deploy API Gateway Analytics in a classic deployment (non-containerized), see the *API Gateway Analytics User Guide*.

## Create an API Gateway Analytics Docker image

To create an API Gateway Analytics Docker image, use the `build_aga_image.py` script. This script builds an API Gateway Analytics Docker image using an API Gateway 7.6.2 Linux installer and a Docker image based on a standard or custom CentOS7 or RHEL7 operating system image.

**Caution** Docker automatically downloads the latest CentOS or RHEL7 image, which may potentially contain security vulnerabilities. Axway is not responsible for any third-party base O/S images. You must ensure that all base O/S images are up-to-date and apply any security patches if necessary.

### API Gateway Analytics image script options

You must specify the following as options when using the `build_aga_image.py` script:

- API Gateway Analytics license
- API Gateway 7.6.2 Linux installer
- Operating system based on one of the following:
  - Standard CentOS7 Docker image downloaded from the public Docker registry
  - Standard RHEL7 Docker image downloaded from the Red Hat Docker registry
  - If you specify a custom CentOS7 or RHEL7-based OS Docker image, Docker first tries to find the custom image in the local registry, and then tries to download it from a remote registry

**Note** You must have an RHEL7 license to build a base API Gateway image based on an RHEL7 OS.

This script also supports additional options when generating an API Gateway Analytics image. For example, you can:

- Build an image from existing API Gateway Analytics configuration by specifying an existing `.fed` file.
- Specify a merge directory to add to the API Gateway Analytics Docker image. This directory can include custom configuration, JAR files, and so on.

For the latest script usage and options, run the script with no options, or with the `-h` option. For example:

```
$ cd emt_containers-<version>
$ ./build_aga_image.py -h
```

The following examples show how you can use this script to build API Gateway Analytics Docker images:

- *Create an API Gateway Analytics image for a development environment* on page 47
- *Create an API Gateway Analytics image for a production environment* on page 48

# Create an API Gateway Analytics image for a development environment

The following example creates a simple API Gateway Analytics Docker image using a default administrator user and default connection for the metrics database. This approach is recommended in a development environment only.

## *Usage guidelines*

- Do not use default options on production systems. The `--default-user` option is provided only as a convenience for development environments.
- Use the `--merge-dir` option to specify the `analytics` directory containing the JDBC driver JAR file for the specified metrics database in the `ext/lib` directory.
    - The merge directory must be called `analytics` and must have the same directory structure as the `analytics` directory of an API Gateway Analytics installation.
    - Copy the JAR file to a new directory `/tmp/analytics/ext/lib/` and specify `/tmp/analytics` to the `--merge-dir` option.
- Use the metrics options to specify the URL, user name, and password for your metrics database. If not specified, the metrics options have the following default values:
    - `--metrics-db-url`: Defaults to `${environment.METRICS_DB_URL}`
    - `--metrics-db-username`: Defaults to `${environment.METRICS_DB_USERNAME}`

- `--metrics-db-pass-file`: Default value for password if password file not specified is `${environment.METRICS_DB_PASS}`

### *Example command*

```
$ cd emt_containers-<version>
$ ./build_aga_image.py
--installer=apigw-installer.run
--os=centos7
--license=/tmp/analytics_license.lic
--default-user
--merge-dir=/tmp/analytics
--out-image=apigw-analytics:1.0
```

This example creates an API Gateway Analytics Docker image named `apigw-analytics` with a tag of `1.0`. This image has the following characteristics:

- Based on a standard CentOS7 Docker image
- Uses a default user name of `admin` and a default password for the API Gateway Analytics administrator user
- Uses default values for the metrics database
- Uses a specified merge directory (containing the JDBC driver JAR file for the metrics database) that is merged into the API Gateway Analytics image

## Create an API Gateway Analytics image for a production environment

The following example creates an API Gateway Analytics Docker image using a specified API Gateway Analytics administrator user and specified connection details for the metrics database. This is the recommended approach in a production environment.

### *Usage guidelines*

- Use the `--merge-dir` option to specify the `analytics` directory containing the JDBC driver JAR file for the specified metrics database in the `ext/lib` directory.
  - The merge directory must be called `analytics` and must have the same directory structure as the `analytics` directory of an API Gateway Analytics installation.
  - Copy the JAR file to a new directory `/tmp/analytics/ext/lib/` and specify `/tmp/analytics` to the `--merge-dir` option.
- Use the metrics options to specify the URL, user name, and password for your metrics database. If not specified, the metrics options have the following default values:

- `--metrics-db-url`: Defaults to `${environment.METRICS_DB_URL}`
- `--metrics-db-username`: Defaults to `${environment.METRICS_DB_USERNAME}`
- `--metrics-db-pass-file`: Default value for password if password file not specified is `${environment.METRICS_DB_PASS}`

### *Example command*

```
$ cd emt_containers-<version>
$ ./build_aga_image.py
--installer=apigw-installer.run
--os=centos7
--license=/tmp/analytics_license.lic
--analytics-username=user1 --analytics-pass-file=/tmp/pass.txt
--metrics-db-url=jdbc:mysql://metricsdb:3306/metrics
--metrics-db-username=db_user1 --metrics-db-pass-file=/tmp/dbpass.txt
--merge-dir=/tmp/analytics
--out-image=apigw-analytics:1.0
```

This example creates an API Gateway Analytics Docker image named `apigw-analytics` with a tag of `1.0`. This image has the following characteristics:

- Based on a standard CentOS7 Docker image
- Uses a user name of `user1` and a specified password for the API Gateway Analytics administrator user
- Uses a MySQL metrics database with specified connection details
- Uses a specified merge directory (containing the JDBC driver JAR file for the metrics database) that is merged into the API Gateway Analytics image

## Start the API Gateway Analytics Docker container

Use the `docker run` command to start the API Gateway Analytics container. For example:

```
$ docker run -it --name=analytics -p 8040:8040 --network=api-gateway-domain
-v /tmp/reports:/tmp/reports
-e METRICS_DB_URL=jdbc:mysql://metricsdb:3306/metrics?useSSL=false
-e METRICS_DB_USERNAME=db_user1 -e METRICS_DB_PASS=my_db_pwd
apigw-analytics:1.0
```

This example performs the following:

- Starts an API Gateway Analytics container named `analytics` from an image named `apigw-analytics:1.0`. You must specify the name of the API Gateway Analytics Docker image that you created in *Create an API Gateway Analytics Docker image* on page 46.
- Binds the port 8040 of the container to port `8040` on the host machine. This enables you to access the API Gateway Analytics web UI on port `8040` of your host machine.
- Mounts the host directory `/tmp/reports` inside the container to store API Gateway Analytics reports.
- Uses environment variables to specify connection details for the metrics database. The metrics database must be running as detailed in *Step 2 – Start external data stores* on page 22.

To run the container in the background, use the `-d` option, for example:

```
$ docker run -d --name=analytics -p 8040:8040 --network=api-gateway-domain
-v /tmp/reports:/tmp/reports
-e METRICS_DB_URL=jdbc:mysql://metricsdb:3306/metrics?useSSL=false
-e METRICS_DB_USERNAME=db_user1 -e METRICS_DB_PASS=my_db_pwd
apigw-analytics:1.0
```

# Operate and monitor API Gateway containers 6

You can use the API Gateway Manager web UI to operate and monitor API Gateways running in containers:



## Connect to API Gateway Manager

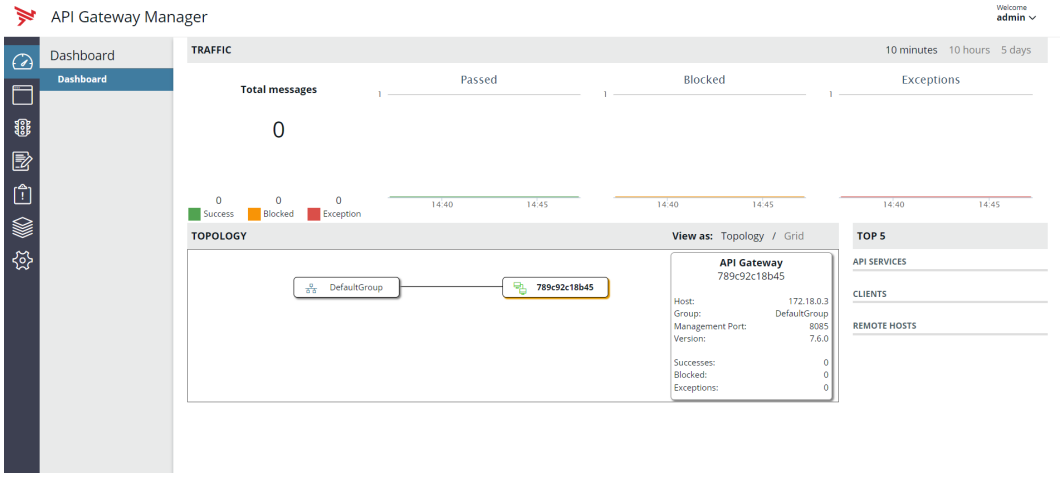
To access the web-based API Gateway Manager UI in a container environment, enter the following URL in a web browser:

```
https://HOST:8090/
```

`HOST` refers to the host name or IP address of the host machine (for example, `https://localhost:8090/`).

**Note** To access API Gateway Manager from your host machine you must have mapped the Admin Node Manager container port 8090 to port 8090 on your host machine, as detailed in *Step 6 – Start the Admin Node Manager Docker container* on page 33.

# Monitor API Gateway containers in API Gateway Manager




API Gateway Manager includes the following features for operating and monitoring API Gateway running in containers:

**Dashboard**

- View traffic summary.
- Monitor network topology (API Gateway containers and groups).

  **Note** You cannot create or delete groups, start or stop API Gateways, or deploy new configuration from API Gateway Manager when running API Gateway in containers.

**Monitoring**

- Monitor traffic processed in real-time.

**Traffic**

- View message log and performance statistics on the traffic processed.

**Logs**

- View domain audit logs, transaction audit logs, transaction access logs, and trace information.

**Events**

- View transaction audit events, alerts, and SLA alerts.

**Settings**

- Configure dynamic settings.

# Where to go next

For more information on monitoring traffic, see "Monitor services in API Gateway Manager" in the *API Gateway Administrator Guide*.

For more information on monitoring topology, see "Manage domain topology in API Gateway Manager" in the *API Gateway Administrator Guide*.

For more information on viewing logs and events, and configuring dynamic settings, see "Configure API Gateway logging and events" in the *API Gateway Administrator Guide*.

# Redirect logs to stdout

You can configure the API Gateway logging system to redirect the trace and traffic logs to `stdout` instead of to separate files. This allows the logs to be read directly from each container by an external logging service (for example, Elastic Stack or Splunk).

## Trace logs

The trace log behavior can be modified through the `trace.xml` file or the system environment variables.

**Note** The environment variables override the `trace.xml` file settings. This enables the logging behavior of individual containers to be defined at runtime.

### *Redirect trace logs using the trace.xml file*

Update the `INSTALL_DIR/apigateway/system/conf/trace.xml` file as follows:

1. Disable the trace file on disk by commenting out the following line:

```
<!--<FileRolloverTrace maxfiles="500"
filename="%s_%Y%m%d%H%M%S.trc"/>-->
```

2. Enable JSON output by changing the `jsonOutput` property value to `true`:

```
<FileTrace filename="@stdout" jsonOutput="true"/>
```

### *Redirect trace logs using system environment variables*

Ensure the following environment variables are passed to the container:

```
APIGW_LOG_TRACE_TO_FILE=false
```

```
APIGW_LOG_TRACE_JSON_TO_STDOUT=true
```

# Open traffic logs

The open traffic log behavior can be modified through the Policy Studio settings or system environment variables.

**Note** The environment variables override the Policy Studio settings. This enables the logging behavior of individual containers to be defined at runtime.

## *Redirect open traffic logs using Policy Studio*

Select the **Server Settings** node in the Policy Studio tree, click **Logging > Open Traffic Event Log**, and select **Use console/traces** from the **Event Log Output** field.

## *Redirect open traffic logs using system environment variables*

Ensure the following environment variable is passed to the container:

```
APIGW_LOG_OPENTRAFFIC_OUTPUT=stdout
```

# Further information

For more information on trace logging and open traffic logging, see the *API Gateway Administrator Guide*. This guide also describes the open logging JSON schema.

# Development and deployment with API Gateway containers 7

This topic describes how you can develop and test APIs and policies in a development environment, and how you can promote and deploy them in other environments (for example, preproduction and production).

In a typical API Gateway container deployment:

- You create Docker images in your build or continuous integration (CI) environment from various artifacts checked out of your source control system. The resulting images are versioned and immutable.
- You start Docker containers from the immutable images in a specific runtime environment (for example, development, preproduction, or production).

Alternatively, you can create images and start containers in a single CI/CD pipeline, where artifact updates trigger a build that results in the automatic creation or refresh of an execution environment.

For detailed information on promotion and deployment, see the *API Gateway DevOps Deployment Guide*.

## Test in development environment

In a development environment, the policy developer works in a continuous cycle of iterative development, deployment, and testing. In this environment, it makes sense to keep all API Gateway configuration in a single deployment package ( `.fed` file), which you can deploy or export from Policy Studio.

As a developer, you have three different options when testing in a development environment:

- You can use an API Gateway that is running in classic (non-containerized) mode. In this case, to test changes you can deploy configuration changes directly from Policy Studio. For more information, see the *API Gateway Policy Developer Guide*.
- You can use a deployment-enabled API Gateway container. In this case, to test changes you can deploy directly from Policy Studio to the running container. Follow these steps:
    1. Follow the steps in *Deploy API Gateway in Docker containers* on page 19. When starting the API Gateway Docker container, specify `EMT_DEPLOYMENT_ENABLED=true`. For an example, see *Start a deployment-enabled API Gateway container in a development environment* on page 42.
    2. Make the configuration changes to be tested in Policy Studio and click **Deploy** in the toolbar to deploy the updated configuration to the running API Gateway container.

- You can use an API Gateway container that does not have deployment enabled. In this case, to test changes you must export the configuration from Policy Studio and use it to generate a new API Gateway image and start a new container. Follow these steps:
    1. Make the configuration changes to be tested in Policy Studio and select **File > Export > Deployment Package** to export the configuration as a deployment package (`.fed`).
    2. Follow the steps in *Deploy API Gateway in Docker containers* on page 19. When creating the API Gateway Docker image using `build_gw_image.py`, specify the deployment package you exported from Policy Studio.

       For an example, see *Create an API Gateway image using existing fed and customized configuration* on page 37.

# Promote to preproduction environment

After the configuration is deployed and tested in the development environment, you can promote it to the preproduction environment. This involves environmentalizing the configuration that will require environment-specific settings in upstream environments and exporting it as a policy package (`.pol` file) from Policy Studio, and creating an environment package (`.env` file) that is specific to the preproduction environment in Configuration Studio.

To promote a configuration to a preproduction environment, follow these steps:

1. Environmentalize the configuration in Policy Studio and select **File > Export > Policy Package** to export the configuration as a policy package (`.pol`).
2. Create the environment package for the preproduction environment in Configuration Studio and select **File > Save > Environment Package** to export the environment package (`.env`).
3. Follow the steps in *Deploy API Gateway in Docker containers* on page 19. When creating the API Gateway Docker image using `build_gw_image.py`, specify the policy package and preproduction environment package you exported from Policy Studio and Configuration Studio.

   For reference, see the example *Create an API Gateway image using existing fed and customized configuration* on page 37, however, in this case you will need to specify a `.pol` and `.env` instead of a `.fed`.

For a detailed example of promoting configuration through environments, see "Example: Promote from development to testing environment" in the *API Gateway DevOps Deployment Guide*.

# Promote to production environment

After the configuration is deployed and tested in the preproduction environment, you can promote it to the production environment. This involves creating an environment package (`.env` file) that is specific to the production environment in Configuration Studio.

To promote a configuration to a production environment, follow these steps:

1. Create the environment package for the production environment in Configuration Studio and select **File > Save > Environment Package** to export the environment package (`.env`).
2. Follow the steps in *Deploy API Gateway in Docker containers* on page 19. When creating the API Gateway Docker image using `build_gw_image.py`, specify the policy package and production environment package you exported from Policy Studio and Configuration Studio.

   For reference, see the example *Create an API Gateway image using existing fed and customized configuration* on page 37, however, in this case you will need to specify a `pol` and `env` instead of a `.fed`.

For a detailed example of promoting configuration through environments, see "Example: Promote from development to testing environment" in the *API Gateway DevOps Deployment Guide*.

# Continuous policy deployment

You can continuously deploy policy updates in a container environment, as part of your CI/CD process by scripting the steps detailed in *Deploy API Gateway in Docker containers* on page 19 and specifying the latest policy package (`.pol`) exported from the development environment and the environment package (`.env`) for the production environment to the `build_gw_image.py` script.

# Promote APIs registered in API Manager

You can promote APIs registered in API Manager using the approaches described in "Promote managed APIs between environments" in the *API Manager User Guide*.

# Migrate to container deployment

# 8

This topic describes how to migrate an API Gateway or API Manager 7.6.2 classic deployment to an elastic container deployment.

## Prerequisites

- Your 7.6.2 classic deployment must be configured correctly and working as expected. The 7.6.2 deployment can be a system you upgraded from an earlier version or it can be a new installation.
- Apache Cassandra and any relational databases your system uses must be in the same network as your Docker host and accessible to your Docker host.
- You must have an appropriate license to run API Gateway or API Manager in a Docker container.

## Sample topology

The following is a sample topology used to demonstrate the steps you must perform to migrate from a classic deployment to a container deployment. Your own topology might be significantly different and you must adapt the steps appropriately.

In this example, the API Gateway 7.6.2 classic deployment being migrated has the following topology:

- One Admin Node Manager named ANM1
- API Gateway Group1 contains an API Gateway named GW1
    - API Manager is enabled
    - Apache Cassandra is configured for GW1
    - Metrics processing is enabled and a relational database is configured
    - GW1 is registered with ANM1
- API Gateway Group2 contains an API Gateway named GW2
    - API Manager is enabled
    - GW2 is registered with ANM1

# Migration steps

The steps are as follows:



## Step 1 - Generate domain certificates

Generate a new domain certificate and key as detailed in *Step 3 – Generate domain SSL certificates* on page 23.

Alternatively, you can reuse your existing system domain certificate and key. Domain certificates for a classic deployment are located in the `/INSTALL_DIR/apigateway/groups/certs/` directory. This can be useful if you have domain certificates signed by an external CA and you want to reuse them.

**Note** You must not use the same domain certificates for classic and container deployments that are running in parallel. Reusing the same certificates in this case can cause problems as the API Gateway containers might try to register with both the elastic container deployment and classic deployment Node Managers.

## Step 2 - Export Admin Node Manager configuration

To export the Admin Node Manager configuration, follow these steps:

1. In Policy Studio, create a project from existing configuration. Select the Node Manager configuration from your classic deployment (for example, `/INSTALL_DIR/apigateway/conf/fed`).
2. Export the configuration to a `fed` file. Select **File > Save Package** and enter a file name (for example, `ANM.fed`). Use the same passphrase as you used in the classic deployment.

## Step 3 - Export API Gateway configuration

To export the API Gateway configuration, follow these steps:

1. In Policy Studio, create a project from a running API Gateway. Select the Group1 GW1 from your classic deployment. API Manager is already enabled. Leave the password blank for the API administrator user as this is already stored in Cassandra.
2. Export the configuration to a `fed` file. Select **File > Save Package** and enter a file name (for example, `Group1.fed`). Use the same passphrase as you used for Group1 in the classic deployment.
3. Repeat steps 1 to 3 for the Group2 GW2 and export the configuration to a `fed` file (for example, `Group2.fed`) using the same passphrase as you used for Group2 in the classic deployment.

# Step 4 - Export API Manager and KPS data

You can set up your container deployment to either use the existing Apache Cassandra keyspace from your classic deployment or you can create a new keyspace.

To point your API Gateway container to an existing Cassandra keyspace used by your classic deployment, follow these steps:

1. In **Server Settings > Cassandra > Keyspace** change the keyspace name from `x${DOMAINID}_${GROUPID}` to the actual keyspace name of this group in Cassandra (for example, `x5dca69a3_443d_4527_9893_3b4652c9ff6b_group_2`).

This enables the API Gateway container to use the same keyspace as the classic API Gateway.

Alternatively, to create a new keyspace for the API Gateway container, follow these steps:

1. Use `kpsadmin backup` to export the KPS data.
2. Copy the backup directory to the API Gateway container and run `kpsadmin restore` from the Admin Node Manager container.

For more details, see the `kpsadmin` section in the *API Gateway Key Property Store User Guide*.

**Note** You can also export your API Manager APIs as an API collection if you require a backup. For details see the *API Manager User Guide*.

# Step 5 - Prepare merge directory

You can use a merge directory to add custom configuration when building your Admin Node Manager and API Gateway Docker images. This directory is only for specifying custom configuration settings that are outside the `fed` (for example, `envSettings.props`, custom JARs, and so on).

To create a merge directory, follow these steps:

1. Create a new directory named `apigateway` for each API Gateway group. The structure of this directory must be similar to the `apigateway` directory in an API Gateway installation (with the exception of the `apigateway/groups` directory which has a different structure in a container deployment).

The following is an example merge directory for Group1 and the Admin Node Manager:

```
/tmp/mergedir/group1/apigateway
```

The following is an example merge directory for Group2:

```
/tmp/mergedir/group2/apigateway
```

2. Copy Admin Node Manager and API Gateway GW1 custom configuration files from your classic deployment to the `group1` directory (for example, copy `apigateway/conf/envSettings.props` and custom JARs from `apigateway/ext/lib` and copy `apigateway/groups/group-X/instance-X/conf/envSettings.props` to `apigateway/groups/emt-group/emt-service/conf/envSettings.props`).
3. Copy API Gateway GW2 custom configuration files from your classic deployment to the `group2` directory (for example, copy `apigateway/groups/group-X/instance-X/conf/envSettings.props` to `apigateway/groups/emt-group/emt-service/conf/envSettings.props`).

# Step 6 - Build Admin Node Manager and API Gateway Docker images

To build images, follow these steps:

1. Build a base image as detailed in *Step 4 – Create base Docker image* on page 26. For example:

```
./build_base_image.py
--installer=apigw-installer.run
--os=rhel7
--out-image= apigw-base:1.0
```

2. Build an Admin Node Manager image as detailed in *Step 5 – Create an Admin Node Manager Docker image* on page 28. For example:

```
./build_anm_image.py
--parent-image=apigw-base:1.0
--out-image my-domain-anm:1.0
--merge-dir /tmp/mergedir/group1/apigateway
--fed ANM.fed
--domain-cert ClassicDomainCert.pem
--domain-key ClassicDomainKey.pem
--domain-key-pass-file /tmp/domainpass.txt
```

3. Build the Group1 API Gateway image as detailed in *Step 7 – Create an API Gateway Docker*

*image* on page 34. For example:

```
./build_gw_image.py
--parent-image=apigw-base:1.0
--out-image my-gw-group1:1.0
--license LicenseWithoutHostname.lic
--domain-cert ClassicDomainCert.pem
--domain-key ClassicDomainKey.pem
--domain-key-pass-file /tmp/domainpass.txt
--fed Group1.fed
--fed-pass-file /tmp/fedpass.txt
--merge-dir /tmp/mergedir/group1/apigateway
--group-id=Group1
```

4. Build the Group2 API Gateway image. For example:

```
./build_gw_image.py
--parent-image=apigw-base:1.0
--out-image my-gw-group2:1.0
--license LicenseWithoutHostname.lic
--domain-cert ClassicDomainCert.pem
--domain-key ClassicDomainKey.pem
--domain-key-pass-file /tmp/domainpass.txt
--fed Group2.fed
--fed-pass-file /tmp/fedpass.txt
--merge-dir /tmp/mergedir/group2/apigateway
--group-id=Group2
```

# Step 7 - Start Docker containers from the Admin Node Manager and API Gateway images

To start the Admin Node Manager and API Gateway Docker containers, follow these steps:

1. Start the Admin Node Manager container as detailed in *Step 6 – Start the Admin Node Manager Docker container* on page 33. For example:

```
docker run -it -p 8090:8090 --name=ANM1 --network=my_network my-domain-anm:1.0
```

Alternatively, if metrics are enabled:

```
docker run -it -p 8090:8090 --name=ANM1 --network=my_network -v /tmp/gw-events:/opt/Axway/apigateway/events my-domain-anm:1.0
```

You can now log in to API Gateway Manager at `https://docker_host:8090` with the same administrator credentials as you used for your classic deployment.

2. Start the Group1 API Gateway container as detailed in *Step 8 – Start the API Gateway Docker container* on page 41. For example:

```
docker run -it -p 8075:8075 -e EMT_ANM_HOSTS= ANM1:8090 --network=my_
network my-gw-group1:1.0
```

3. Start the Group2 API Gateway container. For example:

```
docker run -it -e EMT_ANM_HOSTS=ANM1:8090 --network=my_network my-gw-
group2:1.0
```

Alternatively, if metrics are enabled start GW1 and GW2 as follows:

```
docker run -it -p 8075:8075 -p 8065:8065 -e EMT_ANM_HOSTS=ANM1:8090 --
network=my_network
-v /tmp/gw-events:/opt/Axway/apigateway/events my-gw-group1:1.0

docker run -it -e EMT_ANM_HOSTS=ANM1:8090 --network=my_network
-v /tmp/gw-events:/opt/Axway/apigateway/events my-gw-group2:1.0
```

You can now log in to API Manager UI of Group1 at `https://docker_host:8075` with the same API administrator credentials as you used for your classic deployment.

# Apply a patch or service pack 9

This topic describes how to apply a patch or a service pack (SP) to an API Gateway or API Manager container deployment. In a container deployment, a patch or service pack is rolled out using an orchestration tool (for example, Kubernetes or OpenShift) after new Docker images containing the patch or service pack are pushed to the Docker registry. This enables you to perform a rolling zero downtime update of services.

## Install a patch

To install a patch, follow these steps:

1. Download the patch from Axway Support at https://support.axway.com.
2. Create a merge directory to contain the patch files and any custom configuration (for example, `/tmp/apigateway`). The merge directory must be called `apigateway` and must have the same directory structure as the `apigateway` directory of an API Gateway installation.
3. Unzip and extract the patch into the merge directory.
4. Add any custom configuration to the merge directory. For example, to add a custom `envSettings.props` file to your image, copy `envSettings.props` to `/tmp/apigateway/conf/`.
5. Create new Admin Node Manager and API Gateway images using the `--merge-dir` option to specify the merge directory containing the patch files and custom configuration. For example:

```
$ cd emt_containers-<version>

$ ./build_anm_image.py
--domain-cert=certs/mydomain/mydomain-cert.pem
--domain-key=certs/mydomain/mydomain-key.pem
--domain-key-pass-file=/tmp/pass.txt
--anm-username=gwadmin --anm-pass-file=/tmp/gwadminpass.txt
--merge-dir=/tmp/apigateway

$ ./build_gw_image.py
--license=/tmp/api_gw.lic
--domain-cert=certs/mydomain/mydomain-cert.pem
--domain-key=certs/mydomain/mydomain-key.pem
--domain-key-pass-file=/tmp/pass.txt
--merge-dir=/tmp/apigateway
```

# Install a service pack

To install a service pack, follow these steps:

1. Download the latest API Gateway 7.6.2 Linux installer (which includes the service pack) from Axway Support at https://support.axway.com.
2. Create a new base image using the `--installer` option to build the image from the downloaded API Gateway installer. For example:

```
$ cd emt_containers-<version>
$ ./build_base_image.py
--installer=apigw-new-installer.run
--os=centos7
```

3. Create a merge directory to contain any custom configuration (for example, `/tmp/apigateway`). The merge directory must be called `apigateway` and must have the same directory structure as the `apigateway` directory of an API Gateway installation.
4. Add any custom configuration to the merge directory. For example, to add a custom `envSettings.props` file to your image, copy `envSettings.props` to `/tmp/apigateway/conf/`.
5. Create new Admin Node Manager and API Gateway images using the `--merge-dir` option to specify the merge directory containing the custom configuration. For example:

```
$ cd emt_containers-<version>

$ ./build_anm_image.py
--domain-cert=certs/mydomain/mydomain-cert.pem
--domain-key=certs/mydomain/mydomain-key.pem
--domain-key-pass-file=/tmp/pass.txt
--anm-username=gwadmin --anm-pass-file=/tmp/gwadminpass.txt
--merge-dir=/tmp/apigateway

$ ./build_gw_image.py
--license=/tmp/api_gw.lic
--domain-cert=certs/mydomain/mydomain-cert.pem
--domain-key=certs/mydomain/mydomain-key.pem
--domain-key-pass-file=/tmp/pass.txt
--merge-dir=/tmp/apigateway
```

### Related topics

*Step 4 – Create base Docker image* on page 26

*Step 5 – Create an Admin Node Manager Docker image* on page 28

*Step 7 – Create an API Gateway Docker image* on page 34

# Troubleshoot container deployments 10

This topic describes problems you might encounter when running API Gateway and API Manager in Docker containers, and provides possible solutions.

## Reset the default API administrator password

**Problem**: You have forgotten the password for the default API administrator user in API Manager and you cannot log in to the API Manager UI.

**Solution**: After the default API administrator account has been created in Apache Cassandra, you cannot change the password in Policy Studio. If you cannot log in to the API Manager UI to change the password, you must use the `setup-apimanager` script with the `--resetPassword` option to reset it. Follow these steps:

1. Use the `docker exec` command to connect to the running Admin Node Manager Docker container.

```
docker exec -it <anm-container-id> bash
```

2. Change to the `bin` directory.

```
cd /opt/Axway/apigateway/posix/bin
```

3. Run the `setup-apimanager` script with the `--resetPassword` option.

```
./setup-apimanager --resetPassword
```

## Manage KPS with kpsadmin

**Problem**: You need to perform administrative or management of a key property store (KPS) in a container deployment.

**Solution**: To run the `kpsadmin` tool in a container deployment, you must connect to the Admin Node Manager container and run the tool from there. Follow these steps:

1. Use the `docker exec` command to connect to the running Admin Node Manager Docker container.

```
docker exec -it <anm-container-id> bash
```

2. Change to the `bin` directory.

```
cd /opt/Axway/apigateway/posix/bin
```

3. Run the `kpsadmin` tool.

```
./kpsadmin
```

For more information on using `kpsadmin`, see the *API Gateway Key Property Store User Guide*.

# Logs do not persist when container stops

**Problem**: Traffic monitor data and trace logs do not persist when a container stops.

**Solution**: You can redirect the trace and traffic logs to `stdout` instead of to separate files, which allows the logs to be read directly from each container by an external logging service. For more information, see *Redirect logs to stdout* on page 53.

# Use Apache Cassandra as a distributed data store

**Problem**: Distributed Ehcache is not supported in a container deployment.

**Solution**: You can use Apache Cassandra as a distributed data store. This involves using the KPS scripting API, which enables you to perform CRUD operations and interact directly with a KPS. For details, see "Use the KPS scripting API" in the *API Gateway Key Property Store User Guide*.