apiman -
Installation Guide

Chapter 1. Installation

This guide provides detailed information about how to install and configure apiman.

1.1. Installing in WildFly 10

The apiman primarily targets WildFly 10 as a runtime environment. In order to install apiman you will need to download both WildFly 10 and the apiman overlay distribution. Once both are downloaded, it's a simple matter of unpacking both into the same location.

1.1.1. Download

First you will need to download both WildFly 10 and apiman:

- *Download WildFly 10* [http://download.jboss.org/wildfly/10.0.0.Final/wildfly-10.0.0.Final.zip]
- *Download apiman 1.2.6.Final* [http://downloads.jboss.org/apiman/1.2.6.Final/apiman-distro-wildfly10-1.2.6.Final-overlay.zip]

```
curl http://download.jboss.org/wildfly/10.0.0.Final/wildfly-10.0.0.Final.zip
 -o wildfly-10.0.0.Final.zip
curl http://downloads.jboss.org/apiman/1.2.6.Final/apiman-distro-
wildfly10-1.2.6.Final-overlay.zip -o apiman-distro-wildfly10-1.2.6.Final-
overlay.zip
```

1.1.2. Unpack

Once both files have been downloaded, simply unpack both in the same location.

```
unzip wildfly-10.0.0.Final.zip
unzip -o apiman-distro-wildfly10-1.2.6.Final-overlay.zip -d
 wildfly-10.0.0.Final
```

1.1.3. Run WildFly 10

The apiman overlay contains everything needed to run apiman, including:

- apiman binaries (several WAR files)
- apiman-specific WildFly 10 configuration (**standalone-apiman.xml**)
- apiman rdbms datasource (h2)
- pre-configured **admin** user with password **admin123!**
- pre-configured h2 database for the API Manager (populated with default values)

- embedded Elasticsearch instance for metrics and gateway storage

For this reason, there is no additional configuration required to run apiman. Simply start up WildFly using the apiman configuration file:

```
cd wildfly-10.0.0.Final
./bin/standalone.sh -c standalone-apiman.xml
```

1.2. Installing in WildFly 9

The apiman project can target WildFly 9 as a runtime environment. In order to install apiman you will need to download both WildFly 9 and the apiman overlay distribution. Once both are downloaded, it's a simple matter of unpacking both into the same location.

1.2.1. Download

First you will need to download both WildFly 9 and apiman:

- *Download WildFly 9* [http://download.jboss.org/wildfly/9.0.2.Final/wildfly-9.0.2.Final.zip]
- *Download apiman 1.2.6.Final* [http://downloads.jboss.org/apiman/1.2.6.Final/apiman-distro-wildfly9-1.2.6.Final-overlay.zip]

```
curl http://download.jboss.org/wildfly/9.0.2.Final/wildfly-9.0.2.Final.zip -
o wildfly-9.0.2.Final.zip
curl http://downloads.jboss.org/apiman/1.2.6.Final/apiman-distro-
wildfly9-1.2.6.Final-overlay.zip -o apiman-distro-wildfly9-1.2.6.Final-
overlay.zip
```

1.2.2. Unpack

Once both files have been downloaded, simply unpack both in the same location.

```
unzip wildfly-9.0.2.Final.zip
unzip -o apiman-distro-wildfly9-1.2.6.Final-overlay.zip -d
 wildfly-9.0.2.Final
```

1.2.3. Run WildFly 9

The apiman overlay contains everything needed to run apiman, including:

- apiman binaries (several WAR files)
- apiman-specific WildFly 9 configuration (**standalone-apiman.xml**)
- apiman rdbms datasource (h2)

- pre-configured **admin** user with password **admin123!**
- pre-configured h2 database for the API Manager (populated with default values)
- embedded Elasticsearch instance for metrics and gateway storage

For this reason, there is no additional configuration required to run apiman. Simply start up WildFly using the apiman configuration file:

```
cd wildfly-9.0.2.Final
./bin/standalone.sh -c standalone-apiman.xml
```

1.3. Installing in JBoss EAP 7

If you prefer to run apiman on a Red Hat supported application server, you can install it into JBoss EAP 7. In order to install apiman you will need to download both EAP and the apiman overlay distribution. Once both are downloaded, it's a simple matter of unpacking both into the same location.

1.3.1. Download

First you will need to download both EAP 7 and apiman:

- *Download EAP 7* [http://www.jboss.org/products/eap/download/]
- *Download apiman 1.2.6.Final* [http://downloads.jboss.org/apiman/1.2.6.Final/apiman-distro-eap7-1.2.6.Final-overlay.zip]

1.3.2. Unpack

Once both files have been downloaded, simply unpack both in the same location (see the instructions for Wildfly above).

1.3.3. Run EAP 7

The apiman overlay contains everything needed to run apiman, including:

- apiman binaries (several WAR files)
- apiman-specific EAP configuration (**standalone-apiman.xml**)
- apiman rdbms datasource (h2)
- pre-configured **admin** user with password **admin123!**
- pre-configured h2 database for the API Manager (populated with default values)
- embedded Elasticsearch instance for metrics and gateway storage

For this reason, there is no additional configuration required to run apiman. Simply start up EAP using the apiman configuration file:

```
cd jboss-eap*
./bin/standalone.sh -c standalone-apiman.xml
```

1.4. Installing using Docker

Another option when installing apiman is to use our docker image. You're probably pretty familiar with docker if you're going that route, but here is an example of how to start up the apiman docker image:

```
docker pull jboss/apiman-wildfly
docker run -it -p 8080:8080 -p 8443:8443 jboss/apiman-wildfly
```

Note

You can find apiman on *docker hub* [https://registry.hub.docker.com/repos/apiman/].

Chapter 2. Logging In

Once apiman is running, you should be able to log in to the API Manager by pointing your browser at the following URL:

```
http://localhost:8080/apimanui/
```

You may log in with credentials **admin/admin123!**.

Note

We strongly advise that you immediately change the Keycloak admin user password, as well as the "admin" user found in the "apiman" realm!! (you can do that by navigating to *http://localhost:8080/auth/admin/*)

Chapter 3. General Configuration

Of course apiman is made up of a number of different components, many of which can be configured to use various implementations and/or providers. When downloading and installing apiman, the default distribution includes reasonable default values for all options. This section details these options and explains the default values.

3.1. Configuration Properties

All of the apiman WARs share a common configuration file called **apiman.properties**, which can be found in **standalone/configuration**. This file therefore contains configuration settings for all three applications (API Manager, API Manager UI, API Gateway).

Please refer to the apiman.properties file itself, as well as this document, for more information on each property's purpose and possible values.

3.2. API Manager Database

The API Manager, by default, is a typical CDI application and uses JPA/Hibernate to persist its data. The JPA layer requires a data source to connect to a supported database. When running in WildFly this datasource is made available by deploying the following file:

```
standalone/deployments/apiman-ds.xml
```

Out of the box this data source is usually a simple H2 configuration, but you can (of course) change it to support whatever database you desire.

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
        <datasource    jndi-name="jdbc/ApiManDT"    pool-name="apiman-manager-
api" enabled="true"
    use-java-context="true">
      <connection-url>jdbc:h2:${jboss.server.data.dir}${/}h2${/}apiman-manager-
api;MVCC=true</connection-url>
    <driver>h2</driver>
    <security>
      <user-name>sa</user-name>
    </security>
  </datasource>
</datasources>
```

The project comes with DDLs for MySQL and PostgreSQL, to hopefully make it easy to switch away from H2. Note that switching databases also requires a change to the apiman.properties file. The following should be changed to appropriate values for your database:

```
apiman.hibernate.dialect=io.apiman.manager.api.jpa.ApimanMySQL5Dialect
apiman.hibernate.hbm2ddl.auto=validate
```

Note that the following dialects are available:

- io.apiman.manager.api.jpa.ApimanH2Dialect
- io.apiman.manager.api.jpa.ApimanMySQL5Dialect
- io.apiman.manager.api.jpa.ApimanOracle12Dialect
- io.apiman.manager.api.jpa.ApimanPostgreSQLDialect

You can, of course, set the hbm2ddl property to "update" so that hibernate automatically creates the database structure when it starts up. Additional DDLs for various databases can be found in **apiman/ddls/**.

3.3. API Gateway Registry

The API Gateway includes a registry that stores the published API and Client App information. This registry is updated whenever a user publishes an API (or registers a Client App) from within the API Manager UI. The registry contains just the configuration information necessary for the API Gateway to properly apply the appropriate policies to all inbound requests.

Out of the box, the API Gateway is configured to use Elasticsearch to store the published/registered data.

The configuration of the Registry can be found in the **apiman.properties** file.

See section 4.2 for more information on modifying Elasticsearch settings.

3.4. API Gateway Rate Limiter

Part of the running apiman system is a "Rate Limiter" component. This component is used by apiman policies to enforce rate limits and uses Elasticsearch to store data.

The configuration of the Rate Limiter component can be found in the **apiman.properties** file.

See section 4.2 for more information on modifying Elasticsearch settings.

3.5. API Gateway Shared State

Part of the running apiman system is a "Shared State" component. This component is used by apiman policies to share interesting state information across multiple requests. The shared state component uses Elasticsearch to store data.

The configuration of the Shared State component can be found in the **apiman.properties** file.

See section 4.2 for more information on modifying Elasticsearch settings.

3.6. Gateway API Authentication

The Gateway's REST API is what the API Manager invokes when publishing APIs and Client Apps to the Gateway. This REST API should be protected, often using BASIC authentication. By default, the Gateway REST API requires BASIC authentication credentials, as well as a role of **apipublisher**. In other words, the Gateway REST API can only be invoked by a valid user, and that user must have the **apipublisher** role.

3.7. Data Export/Import

Apiman has a feature that allows admin users to export all of the configuration data from the Manager into a single export file (JSON formatted). This exported file can then be edited (if necessary) and then imported into another instance of apiman. This feature attempts to address the following use-cases:

- Data backups
- Migrating between environments
- Upgrading apiman to a new version

Using the feature is simple - you must log into the apiman UI as an admin user, then navigate to the "Export/Import" UI page by clicking the "Export/Import Data" link on the API Manager Dashboard. From there you can export or import data.

3.7.1. Backing Up Your Data

There are multiple strategies for backing up your apiman data, depending on the configuration you have chosen (e.g. whether you are using a Database or elasticserach to store your data). However, once approach to data backups that is consitent across all configurations is to use the Data Export feature of apiman to create a JSON file containing all of the apiman configuration data.

This can be done via the UI or via the following API Manager REST endpoint:

```
https://HOST:PORT/apiman/system/export
```

3.7.2. Migrating Data Between Environments

Often times you may have a Test version of apiman deployed, as well as a Production version. Depending on your workflow, you may wish to configure your APIs in the Test environment and then migrate that configuration into Production (rather than having to re-create the same configuration in Production manually). This can be accomplished by Exporting data from Test and then importing it into Production.

When doing this, note that the Export feature will export the entire set of configuration from apiman. This may be precisely what you want, but many times only a subset of the data is desired. If this

is the case, then you will need to edit the resulting JSON file to only include the data you wish to migrate. In the future, we hope to build tools that will make this editing of the exported file easier.

Once you have edited the exported file, you can simply log into your production apiman instance and use the Export/Import UI page to import the data.

3.7.3. Upgrading to a New Apiman Version

Whenever you wish to upgrade from an old to a newer version of apiman, you will likely want to preserve all of the Plan, API, and Client App configurations you have created. To do this, you can follow these steps:

1. Export all data from OLD VERSION of apiman
2. Shut down OLD VERSION of apiman
3. Install NEW VERSION of apiman
4. Start up NEW VERSION of apiman
5. Import data into NEW VERSION of apiman (data exported in step #1)

Once these steps are complete, you should have a new version of apiman running with all of your existing data.

Chapter 4. HowTos

This section contains specific instructions for how to configure apiman for specific scenarios. For example, it is possible to use Elasticsearch instead of Infinispan for certain API Gateway components. This section details how to make these sorts of changes.

4.1. How To: Use Elasticsearch instead of an RDBMS in the API Manager

The API Manager is configured (by default) to use JPA as the persistence technology for storing all of its data. But this isn't the only persistence technology supported. Another option is to use Elasticsearch. This section details how to set up apiman to use Elasticsearch instead of an RDBMS to store your API Manager data.

4.1.1. High Level Overview

1. Download and install *Elasticsearch* [https://www.elastic.co/downloads/elasticsearch]
2. Make changes to "apiman.properties" to switch from JPA to Elasticsearch
3. (Re)start apiman!
4. Perform standard admin configuration of apiman (the database will of course be empty!)

4.1.2. Download and install Elasticsearch

This part is pretty easy - download the Elasticsearch software and get it running. A very good resource for this can be found here:

http://www.elastic.co/guide/en/elasticsearch/guide/master/getting-started.html

Tip

As of apiman 1.1.4.Final, an instance of Elasticsearch is included in the default apiman distribution. You may use it as your API Manager persistence store. It is running on port 19200.

4.1.3. Make changes to "apiman.properties"

Once Elasticsearch is running smoothly, you must make some changes to the **apiman.properties** file in order to tell apiman to use ES instead of a database. You should modify the apiman.properties file to have the following properties set:

```
apiman.es.protocol=http
```

```
apiman.es.host=localhost
apiman.es.port=19200
apiman.es.username=
apiman.es.password=

apiman-manager.storage.type=es
apiman-manager.storage.es.protocol=${apiman.es.protocol}
apiman-manager.storage.es.host=${apiman.es.host}
apiman-manager.storage.es.port=${apiman.es.port}
apiman-manager.storage.es.username=${apiman.es.username}
apiman-manager.storage.es.password=${apiman.es.password}
apiman-manager.storage.es.initialize=true
```

Make sure you enter appropriate values for the apiman.es.protocol, apiman.es.host, and apiman.es.port properties. These values should reflect the settings of your Elasticsearch instance.

Tip

You can optionally also set the username and password - this is only useful if you are using something like Elasticsearch Shield to enable basic authentiation.

4.1.4. (Re)start apiman

If apiman was running, you should stop it now. Once everything is shutdown, and the changes to apiman.properties have been made, go ahead and start apiman up again. It will pick up the new settings in apiman.properties and attempt to use Elasticsearch instead of the database!

Perform standard admin configuration. Note that the apiman quickstart overlay ZIP comes pre-configured with a number of settings, including:

- Installed policy definitions
- Default configured roles (Organization Owner, API Developer, Appliation Developer)
- A default configured Gateway

This built-in configuration will be lost when you switch from JPA to Elasticsearch. You will need to use the apiman admin UI to reconfigure these settings. Refer to the "System Administration" section of the User Guide for more information on this.

4.2. How To: Use a standalone Elasticsearch instance/cluster instead of the quickstart instance

The apiman quickstart overlay ZIP ships by default with an embedded instance of Elasticsearch. This is suitable for getting up and running quickly, but is not a good long term solution. Instead, users are encouraged to install a standalone instance of Elasticsearch and point apiman to it.

4.2.1. High Level Overview

1. Download and install *Elasticsearch* [https://www.elastic.co/downloads/elasticsearch]
2. Make changes to "apiman.properties" to point to your standalone Elasticsearch instance
3. (Re)start apiman!

4.2.2. Download and install Elasticsearch

This part is pretty easy - download the Elasticsearch software and get it running. A very good resource for this can be found here:

http://www.elastic.co/guide/en/elasticsearch/guide/master/getting-started.html

4.2.3. Make changes to "apiman.properties"

Once Elasticsearch is running smoothly, you must make some minor changes to the **apiman.properties** file in order to tell apiman where your Elasticsearch instance is located.

There are a set of global properties used for all apiman components that use Elasticsearch to load data. These properties are:

```
apiman.es.protocol=http
apiman.es.host=localhost
apiman.es.port=19200
apiman.es.username=
apiman.es.password=
```

Make sure you enter appropriate values for this properties - they should reflect the settings of your Elasticsearch installation.

4.2.4. (Re)start apiman

If apiman was running, you should stop it now. Once everything is shutdown, and the changes to apiman.properties have been made, go ahead and start apiman up again. It will pick up the new settings in apiman.properties and attempt to use Elasticsearch instead of Infinispan.

4.3. How To: Enable MTLS (Mutual SSL) Support for Endpoint Security

If you wish to use mutual SSL to ensure endpoint security between the apiman API Gateway and your back-end API(s), you must update some settings in the apiman.properties file.

4.3.1. High Level Overview

1. Create Trust and Key Stores

2. Make changes to "apiman.properties" to switch from JPA to Elasticsearch

3. (Re)start apiman!

4. Configure one or more API to use MTLS

4.3.2. Create Trust and Key Stores

Please refer to *official JDK documentation* [https://docs.oracle.com/javase/7/docs/technotes/tools/solaris/keytool.html] to learn how to create and managed your SSL Trust and Key stores. Minimally a Keystore is required in order to successfully utilise MTLS, and in many cases also a Truststore.

A **keystore** contains a given node's private key material, and must be kept safe. Each node should have a unique key entry. For instance, a gateway should have its own keystore, and each API likewise. In a production system, these keys should be issued by a trusted certificate authority.

A **truststore** typically contains a set of certificate authorities which are trusted issuers. Therefore, any certificate signed by the trusted CA would be trusted by the gateway. If no truststore is explicitly provided to apiman the *default trusted certificates* [https://docs.oracle.com/javase/7/docs/technotes/tools/solaris/keytool.html#cacerts] provided by the JVM will be used. A typical use-case would be that an organization's internal signing authority is marked as trusted within in the truststore, and as the authority has been used to sign the certificate material in the keystores, they will mutually trust each other by virtue of the issuer.

It is also possible to directly insert the **public/self-signed certificate** corresponding to a given private key pair into a truststore, which works well at small scales and for development, but will quickly cause the accumulation of a huge number of certificates in larger systems as it requires a 1:1 mapping of certificates and private keys (rather than 1:N by using a trusted authority).

Your back-end APIs must be SSL enabled and **require authenticated client SSL connections**. This means you must have server SSL certificates generated (and appropriate certificates and/or CAs stored in your Trust Store).

4.3.3. Example Scenarios

There are many potential configuration permutations, but we'll outline a few simple ones here to get you started.

4.3.3.1. Development Setup

In our hypothetical development setup, let's imagine we have two APIs and a single gateway.

Table 4.1. Simple Development MTLS Setup

Component	Key Alias	Truststore's Trusted Certificates
Apiman Gateway	gateway	api_a.cer, api_b.cer

Component	Key Alias	Truststore's Trusted Certificates
API A	api_a	gateway.cer
API B	api_b	gateway.cer

Walkthrough

- Generate a keystore and export a certificate for each component:
  - Gateway:

```
keytool -genkey -keyalg RSA -keystore gateway_ks.jks -alias gateway
keytool -export -alias gateway -file gateway.cer -keystore
 gateway_ks.jks
```

  - API A:

```
keytool -genkey -keyalg RSA -keystore api_a_ks.jks -alias api_a
keytool -export -alias api_a -file api_a.cer -keystore api_a_ks.jks
```

  - API B:

```
keytool -genkey -keyalg RSA -keystore api_b_ks.jks -alias api_b
keytool -export -alias api_b -file api_b.cer -keystore api_b_ks.jks
```

- Import certificates into appropriate trust stores:
  - Gateway:

```
keytool -import -file api_a.cer -alias api_a -keystore gateway_ts.jks
keytool -import -file api_b.cer -alias api_b -keystore gateway_ts.jks
```

  - API A:

```
keytool -import -file gateway.cer -alias gateway -keystore api_a_ts.jks
```

  - API B:

```
keytool -import -file gateway.cer -alias gateway -keystore api_b_ts.jks
```

Now simply set the appropriate paths to the keystore and truststore in `apiman.properties` for the gateway, and set up your APIs with their respective truststores and keystores (the specifics of how to do this will depend on your API's implementation).

We will also set the following in `apiman.properties` to make our development easier:

```
apiman-gateway.connector-factory.tls.allowAnyHost=true
```

When you add your MTLS protected APIs into apiman, you should set the `API Security` field to `MTLS/Two-Way-SSL`.

4.3.3.2. MTLS via Custom Certificate Authority

The previous approach works for development, but doesn't scale well, is harder to manage and doesn't gracefully handle revocations, expiry, expansion, etc. Instead, let's summarise a scenario where an organisation has an internal CA which they use to sign APIs' certificates. The process for generating a CA and signing certificates is out of scope for this guide, but is trivial to accomplish using OpenSSL, LibreSSL, or similar.

Let's imagine we have a CA called `apimanCA`, and have **signed** the certificates for each node.

Table 4.2. CA-based MTLS Setup

Component	Signed Key Alias	Truststore Contents
Apiman Gateway	gateway (signed by apiman-CA)	apimanCA.cer
API A	api_a (signed by apimanCA)	apimanCA.cer
API N	api_n (signed by apimanCA)	apimanCA.cer

Despite the initial administrative work setting up the CA and signing the certificates, this process is drastically less effort to maintain in large deployments. Only the trusted CA needs to be in the truststore, and any certificates signed by it are trusted by virtue of this.

4.3.4. Make changes to "apiman.properties"

Once you have your Trust Store and Key Store properly configured, you must configure your apiman.properties file. Here is a summary of the properties:

Omit any properties which are not relevant to you, with the exception of `trustStore`, which is mandatory for MTLS.

```
# ---------------------------------------------------------------------
# SSL/TLS settings for the gateway connector(s).
# ---------------------------------------------------------------------

# Trust store contains certificate(s) trusted by gateway.
apiman-gateway.connector-factory.tls.trustStore=<PATH_TO_TRUST_STORE>
apiman-gateway.connector-factory.tls.trustStorePassword=<PASSWORD_IF_ANY>

# Key store contains gateway's keys (including private components: keep it
 safe).
apiman-gateway.connector-factory.tls.keyStore=<PATH_TO_KEY_STORE>
```

```
apiman-gateway.connector-factory.tls.keyStorePassword=<PASSWORD_IF_ANY> #
 Password on key store as a whole
apiman-gateway.connector-factory.tls.keyPassword=<PASSWORD_IF_ANY> #
 Password on specific key(s)
# By default all keys can be used (will try all). If alias list provided,
 will only attempt to use listed keys.
apiman-gateway.connector-factory.tls.keyAliases=<COMMA_SEPARATED_LIST>

# Allowed TLS/SSL protocols and ciphers suites as CSV. Availability will
 vary depending on your JVM impl.
# Uses JVM defaults depending if not explicitly provided.
# See: https://docs.oracle.com/javase/7/docs/technotes/guides/security/
SunProviders.html
apiman-gateway.connector-factory.tls.allowedProtocols=TLSv1.2,TLSv1.1
apiman-gateway.connector-
factory.tls.allowedCiphers=TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_C

# Whether certificate host checks should be bypassed. *Use with great care.*
apiman-gateway.connector-factory.tls.allowAnyHost=false

# Whether self-signed certificates should be automatically trusted. *Use
 with great care.*
apiman-gateway.connector-factory.tls.allowSelfSigned=false
```

Caution

The settings chosen here have significant security implications. Best practice guides are *available at OWASP* [https://www.owasp.org/].

4.3.5. (Re)start apiman

If apiman was running, you should stop it now. Once everything is shutdown, and the changes to apiman.properties have been made, go ahead and start apiman up again.

4.3.6. Configure one or more API to use MTLS

Now that the apiman MTLS feature has been configured, use the Manager UI to enable MTLS in one or more API. This can be done on the "Implementation" tab when you are configuring the details of your back-end endpoint (URL, type, and endpoint security).

4.4. How To: Use an External Keycloak Authentication Server

The apiman quickstart overlay ZIP comes with an embedded version of Keycloak that we use for authentication. You may already have a Keycloak instance that you use. This section explains how to modify apiman to use yours instead of ours.

4.4.1. High Level Overview

1. Create the apiman Realm in Keycloak
2. Configure the API Manager UI client in Keycloak
3. Point apiman at the remote Keycloak

4.4.2. Create the apiman Realm in Keycloak

You'll need to make sure you create the **apiman** realm in your Keycloak server. A quick way to do that is to use the Keycloak admin console to import the apiman realm file located here:

https://github.com/apiman/apiman/blob/master/distro/data/src/main/resources/data/apiman-realm.json

4.4.3. Configure the API Manager UI client in Keycloak

Once the apiman realm has been created or imported, make sure to configure the **Valid Redirect URIs** section of the **apimanui**, **apiman** and **apiman-gateway-api** clients. The default relative URL of each must be replaced by the full (absolute) public URL of your API Manager UI.

For example, the values may be something like:

apimanui
`https://apiman.myorg.com/apimanui/*`

apiman
`https://apiman.myorg.com/apiman/*`

apiman-gateway-api
`https://apiman.myorg.com/apiman-gateway-api/*`

Tip

Don't forget the "*" wildcard at the end of the URL.

4.4.4. Point apiman at the remote Keycloak

Finally, you must modify the **standalone-apiman.xml** configuration file to point apiman at the remote Keycloak server.

Make sure you know the full public URL of your Keycloak server and add it to the following section of **standalone-apiman.xml**:

```
<realm xmlns:kc="urn:jboss:domain:keycloak:1.0" name="apiman">
    <realm-public-key>MIIBIjAN<...>AQAB</realm-public-key>
    <auth-server-url>http://apiman.myorg.com:8080/auth</auth-server-url>
    <public-client>true</public-client>
    <ssl-required>NONE</ssl-required>
    <enable-cors>false</enable-cors>
    <principal-attribute>preferred_username</principal-attribute>
</realm>
```

Tip

If you manually created the apiman realm in Keycloak, you will also need to copy the realm's public key into <realm-public-key> above.

4.5. How To: Configure a Custom API Catalog

The API Manager has a feature where users can import APIs from a globally configured API Catalog. By default, apiman comes with a community catalog that contains a set of common public APIs such as Flickr and Netflix. When deploying apiman into an enterprise setting, it is often useful to replace the community API Catalog with something that lists out the internal APIs available within the enterprise.

4.5.1. High Level Overview

1. Describe your enterprise APIs as apiman API Catalog JSON
2. Make your enterprise API Catalog available in URL form
3. Point apiman at your enterprise API Catalog

4.5.1.1. Create a Custom Enterprise API Catalog JSON

The first thing you will need to do is express all of your enterprise APIs as a JSON file. The format of the JSON file is specific to apiman. You can find an example of the format here:

https://github.com/apiman/apiman-api-catalog/blob/master/catalog.json

4.5.1.2. Make Your Enterprise API Catalog Available

Now that you have a custom JSON based API Catalog, you need to make it available at a URL accessible to the API Manager. This can either be done by stashing it in some web server location so you have an http based URL, or you can store it locally on the API Manager server so as to have a valid file based URL.

4.5.1.3. Point apiman at Your Enterprise API Catalog

The last step is to make apiman aware of your custom API Catalog file. The catalog is configured in the **apiman.properties** file via these properties:

```
apiman-manager.api-
catalog.type=io.apiman.manager.api.core.catalog.JsonApiCatalog
apiman-manager.api-catalog.catalog-url=http://cdn.rawgit.com/apiman/apiman-
api-catalog/1.2.0.Final/catalog.json
```

Simply change the URL defined by the `apiman-manager.api-catalog.catalog-url` property and you're good to go!

Tip

For even more customization, you can actually implement your own API Catalog java class. This approach will allow you to find your APIs in whatever location they happen to be (e.g. a database, registry, etc). Please see the Developer Guide for more information on how to create a truly custom API Catalog.

4.6. How To: Use a Custom Plugin Registry

The API Manager uses a plugin registry to show admin users a list of available plugins that can be installed. Apiman comes with an official plugin registry that shows a list of the standard apiman plugins. If your enterprise implements a large number of custom policies, you may find it useful to replace the standard registry with one that includes your custom plugins in the list.

4.6.1. High Level Overview

1. Describe your enterprise plugins in a registry JSON file
2. Make your enterprise plugin registry available in URL form
3. Point apiman at your enterprise plugin registry

4.6.1.1. Create a Custom Enterprise Plugin Registry JSON

The first thing you will need to do is express all of your enterprise plugins as a JSON file. The format of the JSON file is specific to apiman. You can find an example of the format here:

https://github.com/apiman/apiman-plugin-registry/blob/master/registry.json

4.6.1.2. Make Your Enterprise Plugin Registry Available

Now that you have a custom JSON based plugin registry, you need to make it available at a URL accessible to the API Manager. This can either be done by stashing it in some web server location

so you have an http based URL, or you can store it locally on the API Manager server so as to have a valid file based URL.

4.6.1.3. Point apiman at Your Enterprise Plugin Registry

The last step is to make apiman aware of your custom plugin registry file. The registry is configured in the **apiman.properties** file via the following property:

```
apiman-manager.plugins.registries=http://cdn.rawgit.com/apiman/apiman-
plugin-registry/1.2.0.Final/registry.json
```

The value of this property is a comma-separated list of URLs. Each URL in the list should point to a valid plugin registry JSON file. So to include your enterprise plugins in the list, simply add the URL to your plugin registry to the end of the existing list.

4.7. How To: Use Property Replacement in Policy Config

It is often useful to externalize certain information that varies from one deployment environment to another. For example, you may have an LDAP server for authentication, but you have one in the Test deployment environment and a different one in Production. Rather than configure your apiman policies differently in each environment (to match the actual LDAP connection info) you can externalize those settings into system properties or environment variables. Once that is done, you can refer to those properties/variables in your apiman policy configuration.

4.7.1. High Level Overview

1. Externalize values into system properties or environment variables
2. Reference a system property or environment variable in a policy

4.7.1.1. Externalize Values

Depending on your deployment strategy, how you do this may vary. If you are using WildFly, for example, you can set system properties in the standalone.xml file or by passing them in via -D parameters on startup (not recommended). For more information, see:

https://docs.jboss.org/author/display/WFLY8/General+configuration+concepts

Describing all approaches to setting system properties and environment variables is out of scope for this document.

4.7.1.2. Reference a System Property or Environment Variable

Once you have some values externalized into system properties or environment variables, you can reference them easily in your apiman policies. All you need to do is use the Ant style syntax to refer to your externalized values, like this:

```
${MY_ENVIRONMENT_VARIABLE}
```

A variable of this style can be used in any apiman policy configuration field. The variables are resolved when the policy configuration is first loaded, and then cached. To change a value, you must restart your server.

Tip

When resolving variables, if there is an environment variable with the same name as a system property, the value of the **system property** will be used.

4.8. How To: Use a SQL Database to Store and Retrieve Metrics (Replacing Elasticsearch)

Sometimes you just don't want to use Elasticsearch to store your metrics, even though it's really good at it. We typically recommend that you stick with ES if possible, because it gives you the option of installing Kibana, opening up a whole new world of advanced analytics over the metric data. However, if all you want is the basic graphs/charts shown in the apiman UI, you can easily switch to storing metric data in a database.

4.8.1. High Level Overview

1. Deploy an appropriate JDBC datasource
2. Enable the API Gateway JDBC initializer
3. Configure the API Gateway metrics provider to be JDBC
4. Configure the API Manager metrics accessor to be JDBC

4.8.1.1. Deploy an appropriate JDBC datasource

The first thing to do is deploy a datasource into WildFly that will be used by the JDBC metrics implementations. There are a number of ways to do this, and documenting how to create a data-source in WildFly is outside the scope of this guide. However, an example -ds.xml descriptor file, specifically for this purpose, can be found here:

https://github.com/apiman/apiman/blob/master/distro/data/src/main/resources/sample-configs/apiman_gw-ds_mysql.xml

Tip

Make sure you take note of the datasource jndi-name - you will need it when configuring the metrics implementations in apiman.properties

Tip

Don't forget to also deploy the appropriate JDBC driver JAR for your database.

4.8.1.2. Enable the API Gateway JDBC initializer

Of course, there is a database schema that must be installed in your database, which creates the tables and indexes needed. This schema can be automatically created by apiman the first time it is used. To enable this feature, you must configure the API Gateway JDBC initializer in the **apiman.properties** file, like so:

```
# Initializer for the Gateway JDBC
# ################################
apiman.jdbc.datasource.jndi-location=java:jboss/datasources/apiman-gateway
apiman-gateway.initializers=jdbc
apiman-
gateway.initializers.jdbc=io.apiman.gateway.engine.jdbc.JdbcInitializer
apiman-gateway.initializers.jdbc.datasource.jndi-location=
${apiman.jdbc.datasource.jndi-location}
apiman-gateway.initializers.jdbc.datasource.type=mysql5
```

This initializer will run whenever apiman starts up, and it will install the API Gateway schema/DDL into the configured database so that the metrics JDBC implementations can function properly.

Note that you will need to set the correct value of **apiman-gateway.initializers.jdbc.datasource.type** based on the specific database you will be using. Possible values for this property include:

- h2
- mysql5
- postgresql9
- mssql11
- oracle12

4.8.1.3. Configure the API Gateway metrics provider to be JDBC

Next, you just need to tell apiman that you want to use JDBC instead of Elasticsearch for the storage of metrics data. This can be done again by setting a specific property in **apiman.properties**:

```
apiman-gateway.metrics=io.apiman.gateway.engine.jdbc.JdbcMetrics
apiman-gateway.metrics.datasource.jndi-location=
${apiman.jdbc.datasource.jndi-location}
```

The metrics will be stored in JDBC (asynchronously) instead of Elasticsearch!

4.8.1.4. Configure the API Manager metrics accessor to be JDBC

It doesn't help us to store the metrics data in SQL unless we also query the data when showing information in the UI. For this reason you will need to configure the API Manager to use JDBC to **retrieve** metric information instead of using Elasticsearch. Again, this is done by editing the **apiman.properties** file:

```
# API Manager metrics settings
# ############################
apiman-manager.metrics.type=io.apiman.manager.api.jdbc.JdbcMetricsAccessor
apiman-manager.metrics.datasource.jndi-location=
${apiman.jdbc.datasource.jndi-location}
```

Tip

If you are deploying the API Manager and API Gateway separately (on different nodes), make sure you edit the correct apiman.properties files on the correct nodes. Feel free to make all of the above changes to all your nodes, since any properties in apiman.properties will be ignored on nodes where certain components are excluded.