Maven Installation, Setup and Deployment

Overview

On a very high level all projects need to be built, tested, packaged, documented and deployed. Maven is essentially a project management and comprehension tool and as such provides a way to help with managing:

- Builds
- Documentation
- Reporting
- Dependencies
- SCMs
- Releases
- Distribution

Download and Installation

- Download Maven and Installation for different operational systems: http://maven.apache.org/download.cgi

Setup and Deployment

The following steps could be performed to setup Maven to use for Deployment:

1. Ensure Maven is installed on the machines from which you need to deploy (JRE 1.6 or later, Apache Maven 3.03 or later).
2. Get Read access to the Apigee Maven plugin repositories. This also means the machines you need to deploy from need to be able to contact these repositories to download library dependencies. Please contact your Apigee contact or support@apigee.com for more information. Apigee support will provide Customer with a settings.xml file that should be placed in ~/.m2/settings.xml.

   The maven settings.xml file should have the following entries:

Sample of Settings.xml file

```
<?xml version="1.0"encoding="UTF-8"?>
<settings
xmlns="http://maven.apache.org/SETTINGS/1.0.0"xmlns:xsi="http://www.w3.org/200
1/XMLSchema-instance"xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.
0
http://maven.apache.org/xsd/settings-1.0.0.xsd">
 <profiles>
  <profile>
   <repositories>
    <repository>
     <snapshots>
                  <enabled>false</enabled>
                 </snapshots>
                 <id>apigee-cs-libs-release</id>
     <name>libs-release</name>
     <url>http://repo.apigee.com:8081/artifactory/repo</url>
    </repository>
   </repositories>
   <pluginRepositories>
    <pluginRepository>
     <snapshots>
      <enabled>false</enabled>
     </snapshots>
     <id>apigee-cs-plugins-release</id>
     <name>plugins-release</name>
     <url>http://repo.apigee.com:8081/artifactory/repo</url>
    </pluginRepository>
   </pluginRepositories>
   <id>apigee-artifactory</id>
  </profile>
 </profiles>
 <servers>
  <server>
      <id>apigee-cs-libs-release</id>
         <username>client</username>
   <password>********</password>  <!-- obtain your access from apigee ops -->
     </server>
     <server>
   <id>apigee-cs-plugins-release</id>
         <username>client</username>
         <password>********</password> <!-- obtain your access from apigee ops
-->
  </server>
 </servers>
 <activeProfiles>
  <activeProfile>apigee-artifactory</activeProfile>
 </activeProfiles>
 <pluginGroups>
        <pluginGroup>com.apigee.cs</pluginGroup>
     <pluginGroup>com.apigee.build-tools.enterprise4g</pluginGroup>
 </pluginGroups>
</settings>
```

The passwords should be obtained as a result of submitting a request to support team.

3. Build a development workspace (base Apigee directory structure, API bundles' folder structure) by using Apigee export paradigm or Maven archetype.

   The following maven command with Maven archetype can be used to create Customer base file structure (parameters in the red color should be updated):

   mvn archetype:generate -DarchetypeGroupId=apigee -DarchetypeArtifactId=Customer-Root-archetype -DarchetypeVersion=3.0 -DarchetypeCatalog=local -DinteractiveMode=false -Dversion=1.0 -DgroupId=apigee -DartifactId=***name-of-root(typically company name)*** -DorgName=***Your_Customer_ORG_Name*** -DorgUserId=***your-user-id*** -DorgPassword=***changeit***

   Note that orgUserId and orgPassword, as their names suggest, are the credentials you use to log into the Apigee MS.

   When completed the archetype creates a structure that is composed of the following directories.

   |-*name-of-root (typically company name)*

   |---archive

   |-----docs

   |-----src

   |---binaries

   |---build-artifacts

   |---docs

   |-----customer

   |-----reference

   |-----solution

   |---src

   |-----analytics

   |-----gateway

   |-------parent-pom

   |-------test-app

   |---------apiproxy

   |----------proxies

   |----------resources

   |------------py

   |----------policies

   |----------targets

   |-----java

   |-----portal

   |---test

   If the Customer base folder structure were created by using either the Maven archetype or manually, Customer can now use Apigee second Maven archetype to build an API bundle's folder structure and maven project files called POM files. This should be run under <name-of-root>/src/gateway.

   The following command will generate a new API workspace:

   mvn archetype:generate -DarchetypeGroupId=apigee -DarchetypeArtifactId=proxy-archetype -DarchetypeVersion=2.0 -DarchetypeCatalog=local -DinteractiveMode=false -Dversion=1.0 -DgroupId=apigee -DartifactId=***NameOfAPI***

   When completed the archetype creates a structure that is composed of the following directories.

   |-apiproxy

   |---proxies

   |---resources

|---stepdefinitions

|---targets

4. Parent POM.xml file.

The contents of the parent pom folder will contain a single pom.xml file that will be automatically generated if the workspace were built by using the maven archetype. This file typically contains most of the configuration of maven and the plugin, it also contains credentials for the Apigee platform.

Sample of pom.xml for parent-pom directory.

```
<?xml version="1.0"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>apigee</groupId>
 <artifactId>parent-pom</artifactId>
 <packaging>pom</packaging>
 <version>1.0</version>
 <pluginRepositories>
  <pluginRepository>
   <id>central</id>
   <name>Maven Plugin Repository</name>
   <url>http://repo1.maven.org/maven2</url>
   <layout>default</layout>
   <snapshots>
    <enabled>false</enabled>
   </snapshots>
   <releases>
    <updatePolicy>never</updatePolicy>
   </releases>
  </pluginRepository>
  <pluginRepository>
   <id>apigee-cs-repo</id>
   <url>http://repo.apigee.com:8081/artifactory/repo</url>
   <snapshots>
    <enabled>true</enabled>
   </snapshots>
   <releases>
    <updatePolicy>never</updatePolicy>
   </releases>
  </pluginRepository>
 </pluginRepositories>
 <modules>
  <module>../sample-app</module>
 </modules>
 <build>
  <pluginManagement>
   <plugins>
    <plugin>
     <groupId>com.apigee.build-tools.enterprise4g</groupId>
     <artifactId>4G-gateway-maven-build-pack</artifactId>
     <version>0.0.15</version>
    </plugin>
   </plugins>
  </pluginManagement>
  <plugins>
   <plugin>
```

```
    <artifactId>maven-resources-plugin</artifactId>
    <version>2.3</version>
    <executions>
     <execution>
      <id>copy-resources-step</id>
      <phase>package</phase>
      <goals>
       <goal>copy-resources</goal>
      </goals>
      <configuration>
       <!-- this is important -->
       <overwrite>true</overwrite>
       <!-- target -->
       <outputDirectory>${basedir}/target/apiproxy</outputDirectory>
       <resources>
        <resource>
         <!-- source -->
         <directory>apiproxy</directory>
        </resource>
       </resources>
      </configuration>
     </execution>
    </executions>
   </plugin>
   <plugin>
    <groupId>com.apigee.build-tools.enterprise4g</groupId>
    <artifactId>4G-gateway-maven-build-pack</artifactId>
    <configuration>
     <skip>true</skip>
    </configuration>
   </plugin>
   <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>2.3.2</version>
    <configuration>
     <source>1.6</source>
     <target>1.6</target>
    </configuration>
   </plugin>
   <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.9</version>
    <configuration>
     <testFailureIgnore>false</testFailureIgnore>
    </configuration>
   </plugin>
  </plugins>
 </build>
 <!-- This is where you add the environment specific properties under various
profile names -->
 <profiles>
  <profile>
   <id>test</id>          <!-- This is used in the maven command -->
   <properties>
    <apigee.profile>test</apigee.profile> <!-- This is a profile name corresponds to
the config.json-->
    <apigee.env>test</apigee.env>   <!-- This is the same as environment name in the
```

```
4G org -->
    <apigee.hosturl>https://api.enterprise.apigee.com</apigee.hosturl>
    <apigee.apiversion>v1</apigee.apiversion>
    <!-- value of version in https://api.enterprise.apigee.com/v2 -->
    <apigee.org>yourOrg</apigee.org>
    <apigee.username>${username}</apigee.username>
    <apigee.password>${password}</apigee.password>
    <!-- Below options to support Apigee zero-downtime deployment -->
    <apigee.options>override</apigee.options>
    <apigee.override.delay>10</apigee.override.delay>
   </properties>
  </profile>
  <profile>
   <id>prod</id>
   <properties>
    <apigee.profile>prod</apigee.profile>
    <apigee.env>prod</apigee.env>
    <apigee.hosturl>https://api.enterprise.apigee.com</apigee.hosturl>
    <apigee.apiversion>v1</apigee.apiversion>
    <!-- value of version in https://api.enterprise.apigee.com/v2 -->
    <apigee.org>yourOrg</apigee.org>
    <apigee.username>${username}</apigee.username>
    <apigee.password>${password}</apigee.password>
    <!-- Below options to support Apigee zero-downtime deployment -->
    <apigee.options>override</apigee.options>
    <apigee.override.delay>10</apigee.override.delay>
   </properties>
  </profile>
 </profiles>
 <dependencies>
  <dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.8.2</version>
   <scope>test</scope>
  </dependency>
 </dependencies>
 <repositories>
  <repository>
   <id>apigee-cs-repo</id>
   <url>http://repo.apigee.com:8081/artifactory/repo</url>
```

```
  </repository>
 </repositories>
</project>
```

Note1. In case of manual creating a Maven compatible file structure, should be created the directory "parent-pom" in the directory that contains Customer application folders (see the folder in red color in above file structure).
Note2. The following entries in some XML file elements could be changed to match Customer environment: "groupId", "id" (for each profile sections), "apigee.profile", "apigee.env", "apigee.hosturl", "apigee.org". The contents of "apigee.profile", "apigee.env", and "id" elements should match the profile the Customer wants to use and is matched with environment name. The value of the "apigee.hosturl" element should match the value in the example if Customer is an enterprise cloud user. If Customer is an on-premise user this url would be the location of Customer management servers host and port. Port is by default 8080. The value of the "apigee.org" element should match the organization provided when Customer environment was initially setup, in most cases this includes the name of the company. For on premise installations, the org is setup when you run installation scripts.

Note3. The "apigee.override.delay", "apigee.delay,apigee.options" are optional elements. The "apigee.override.delay" could be specified (in milliseconds) . This will ensure to add a delay between the operations like delete, import, activate, deactivate etc .

Note4. The "apigee.options" element can have the following values: **clean** (this option will delete the last deployed revision in an environment), **validate** (this option will validate a bundle before importing . Thus if you want strict validation then its required), **in active** (this option will import the bundle without activating the bundle), **override** (this option is used for seamless deployment and should be supplied with apigee.override.delay parameter. The apigee.override.delay expects delay to be given in seconds), **update** (this option will update the deployed revision . This is similar to import with validation but no new revision is created. If there any errors in the bundle, error is thrown and the existing bundle is left intact. In case the revision they are trying to update is deployed, it will internally trigger undeployment and deployment. It is completely in the background and not visible in the response. ***It is advised not to update the deployed revision***. (UI could show a warning or something in this case).

Note5. The "apigee.options" combination could be given with comma-separated values. The precedence order of options are -> override , update , (clean, inactive, validate, force).

Note6. Flow without "apigee.options": import–>undeploy(last active)–>deploy (new revision)

5. API bundle's POM.xml file

The pom.xml file should be created under each API bundle.Once Customer have moved over to an application folder Customer need create a pom.xml file, this file will refer back to the parent pom file and because of that will need minimal changes from the configuration that is show in the example. The value of the "groupId" element should match the content of the same element in the parent pom.xml file. The value of the "artifactId" element should be unique, typically set to the folder name or the name of the API. The value of the "name" should match the artifact name.

Sample of Pom.xml for API bundle

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <parent>
  <artifactId>parent-pom</artifactId>
  <groupId>apigee</groupId>
  <version>1.0</version>
  <relativePath>../parent-pom/pom.xml</relativePath>
 </parent>
 <modelVersion>4.0.0</modelVersion>
 <groupId>apigee</groupId>
 <artifactId>proxyname</artifactId>
 <version>1.0</version>
 <name>proxyname</name>
 <packaging>pom</packaging>
 <build>
  <plugins>
   <plugin>
    <artifactId>maven-clean-plugin</artifactId>
    <version>2.5</version>
```

```
 <executions>
  <execution>
   <id>auto-clean</id>
   <phase>initialize</phase>
   <goals>
    <goal>clean</goal>
   </goals>
  </execution>
 </executions>
</plugin>
<plugin>
 <artifactId>maven-resources-plugin</artifactId>
 <version>2.6</version>
 <executions>
  <execution>
   <id>copy-resources-step</id>
   <phase>package</phase>
   <goals>
    <goal>copy-resources</goal>
   </goals>
   <configuration>
    <!-- this is important -->
    <overwrite>true</overwrite>
    <!-- target -->
    <outputDirectory>${basedir}/target/apiproxy</outputDirectory>
    <resources>
     <resource>
      <!-- source -->
      <directory>apiproxy</directory>
     </resource>
    </resources>
   </configuration>
  </execution>
 </executions>
</plugin>
<plugin>
 <groupId>com.apigee.build-tools.enterprise4g</groupId>
 <artifactId>4G-gateway-maven-build-pack</artifactId>
 <configuration>
  <skip>false</skip>
  <!-- Use this module level config to skip module build. Make it true -->
 </configuration>
 <executions>
  <execution>
   <id>configure-bundle-step</id>
   <phase>package</phase>
   <goals>
    <goal>configure</goal>
   </goals>
  </execution>
 </executions>
</plugin>
```

```
  </plugins>
 </build>
</project>
```

With the pom.xml files created Customer now only need to create a config.json file.

5. Config.json file.

   The config.json acts as Customer build time configuration modification descriptor. The file's root object is called configurations, configurations is an array of proxy configurations scoped to an environment. **Note**: it is important that the name of the configurations match the name of the profiles in the parent-pom. For instance, in the example below you have two configurations one for the test profile and one for the production profile. This example also shows how you can use xpath to replace environment specific settings.

Sample of config.json file.

```
{
    "configurations": [
        {
            "name": "prod",
            "policies": [
                {
                    "name": "ServiceCallout.xml",
                    "tokens": [
                        {
                            "xpath": "/ServiceCallout/HTTPTargetConnection/URL",
                            "value": "https://somepath"
                        }
                    ]
                },
                {
                    "name": "QuotaPolicy.xml",
                    "tokens": [{
                            "xpath": "/QuotaPolicy/@enabled",
                            "value": "true"
                        }
                    ]
                }
            ],
            "proxies": [
                {
                    "name": "default.xml",
                    "tokens": [
                        {
                            "xpath":
"/ProxyEndpoint/HTTPProxyConnection/VirtualHost",
                            "value": "some value"
                        }
                    ]
                }
            ],
            "targets": [
                {
                    "name": "default.xml",
                    "tokens": [
                        {
                            "xpath": "/TargetEndpoint/HTTPTargetConnection/URL",
```

```
                            "value": "https://somepath value"
                        }
                    ]
                }
            ]
        },
        {
            "name": "test",
            "policies": [
                {
                    "name": "ServiceCallout.xml",
                    "tokens": [
                        {
                            "xpath": "/ServiceCallout/HTTPTargetConnection/URL",
                            "value": "https://somepath value"
                        },
         {
                 "xpath": "/ServiceCallout/HTTPTargetConnection/SSLInfo/KeyStore",
                 "value": "some value"
                      },
                      {
                 "xpath": "/ServiceCallout/HTTPTargetConnection/SSLInfo/KeyAlias",
                 "value": "some value"
                  }
                    ]
                }
            ],
            "proxies": [
                {
                    "name": "default.xml",
                    "tokens": [
                        {
                            "xpath":
"/ProxyEndpoint/HTTPProxyConnection/VirtualHost",
                            "value": "some value"
                        }
                    ]
                }
            ],
            "targets": [
                {
                    "name": "default.xml",
                    "tokens": [
                        {
                            "xpath": "/TargetEndpoint/HTTPTargetConnection/URL",
                            "value": "https://some value"
                        }
                    ]
                }
```

```
            ]
        }
}
```

Note1. The "**configuration**" section contains an array of API definitions; the "name" is a name of the Maven profile.

Note2. The "**proxies**" section contains an array of proxy definitions, directly correlates to the proxies folder in Customer API Bundle; the "**name**" is a Name of file to configure; the "**tokens**" block contains an array of Actions to Invoke on Elements; the "**x path**" element contains a Path to element that your want to change the value of; the "**value**" element contains the replacement value.

Note3. The "**policies**" section contains an array of policies definitions, directly correlates to the policies or step definition folder in Customer API Bundle; the "**name**" is a Name of file to configure; the "**tokens**" block contains an array of Actions to Invoke on Elements; the "**xpath**" element contains a Path to element that your want to change the value of; the "**value**" element contains the replacement value.

Note4. The "**targets**" section contains an array of target proxy definitions, directly correlates to the proxies folder in your API Bundle; the "**name**" is a Name of file to configure; the "**tokens**" block contains an array of Actions to Invoke on Elements; the "**x path**" element contains a Path to element that your want to change the value of; the "**value**" element contains the replacement value.

Samples of most used maven commands see below.

Import and Deployment Bundle

To import and deploy use:

```
mvn apigee-enterprise:install -P{profile-id value as in parent pom}
-Dusername=test@user.com -Dpassword=P@55w05d (where {} are place holders for values
based on the environment)
```

Import Bundle

To Import only:

```
mvn apigee-enterprise:install -P{profile-id value as in parent pom}
-Dbuild.option=deploy-inactive -Dusername={user apigee edge login} -Dpassword={user
apigee edge login}
```

Undeploy Bundle

To undeploy:

```
mvn apigee-enterprise:install -P{profile-id value as in parent pom} -D
build.option=undeploy -Dusername={user apigee edge login} -Dpassword={user apigee
edge login} (recommended as this works as saving from the UI, which does import and
deploy)
```

Generate Proxy Bundle

To generate proxy zip file with apiproxy folder and artifacts use:

```
mvn apigee-enterprise:package -P{profile-id value as in parent pom}
-Dbuild.option=deploy-inactive -Dusername={user apigee edge login} -Dpassword={user
apigee edge login}.
```

Related Artifacts

- Maven Tutorial - http://docs.codehaus.org/display/MAVENUSER/The+Maven+2+tutorial
- Maven build/deployment/import Scripts samples: https://github.com/apigeecs/devops-tools/tree/master/4G-gateway-maven-build-pack/samples
- Other tools to run deployments available from GitHub https://github.com/apigee/api-platform-samples/tree/master/tools
- Maven Getting Started Guide: http://maven.apache.org/guides/getting-started/
- http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html
- Tutorial how to set up Maven, Git, and Jenkins to work together: http://obscuredclarity.blogspot.com/2012/04/continuous-integration-using-jenkins.html

Sample Configuration Files

- maven-config-files.zip
- sample-config.json