**Summary Statement**

With the Apigee Enterprise, you can find building simple API’s quick and easy using our Cloud based User Interface to easily modify and build application bundles to facilitate most of your API management needs. This document will explore what happens when you need more control over the configuration and code you are pushing into your Apigee enterprise environment. Some of the key concepts included in this document will be how to manage configuration using file system based storage, how to effectively build configuration bundles, and how to deploy these bundles to the enterprise UI.

[Getting Started](#h.tiv6megv7rlt)

[Creating a development workspace using the Export Paradigm](#h.fy081a2qjy19)

[Creating a base directory on your computer](#h.etaoebwf2cjp)

[Create a new API](#h.rr49e3bvide5)

[Export your API](#h.ndknujg1zkzq)

[Decompress API Bundle](#h.yqcgaf5yz7ig)

[Creating a development workspace with a Maven ArcheType](#h.ly9nfwtja544)

[Building your base Apigee file structure with maven archetype](#h.3letye2t9ww5)

[Building API project folders with maven archetype](#h.d6ml6imn13j2)

[Building API bundles](#h.bi2bnlfweji9)

[What is an Apigee bundle?](#h.6o3c4onrcq31)

[Common API building strategies](#h.ru1jd9bktr49)

[Building Strategy - Zipping the file-system](#h.vvkk5ub7y4ma)

[Building Strategy - Maven](#h.c9yx3hedeg5x)

[POM.xml](#h.4bvqxogxkm73)

[Creating a maven compatible file structure](#h.aajq404sg2bn)

[parent-pom/pom.xml example](#h.35vwaddev1v9)

[pom.xml Sample](#h.xi7bhi39z2qu)

[Config.json](#h.w4cgdd6xtdk6)

[Config.json Data Structure](#h.otx7f72jd8qh)

# Getting Started

Often the most difficult and confusing thing in application development is figuring out how to build a common framework for creating new applications. Over time development teams have started using tools like Maven, Ant and Ivy to automate some of these functions. At Apigee we have use two primary methods for creating our development workspaces. The first and most widely used is the export paradigm, the second is using a maven archetype.

## Creating a development workspace using the Export Paradigm

Using this method is very simple and can be outlined by these key steps:

1. Create your base directory structure on your computer
2. Create a new API at enterprise.apigee.com or on your management server (On-Premise Customers Only)
3. Export your API
4. De-Compress your API Bundle

### Creating a base directory on your computer

The primary reason to use the export paradigm over the Maven archetype is to gain the flexibility to create your own file structure to contain your app-bundles. Because of the flexibility we will recommend placeholders for certain items that correspond to items that you will want to have on your file-system for effective management.

|  |  |
| --- | --- |
| Application bundles | This area will correspond to application bundles that you create to manage API’s. Or in simpler terms “Code” |
| Environments Resources | This placeholder can be used to store xml or json used to create resources on an environment. i.e. Cache, VirtualHosts, KeyValueMaps etc. |
| Documentation | We always recommend some general API documentation to live with the api configurations. |

### Create a new API

Instructions for creating a new API can be found at this link

<http://apigee.com/docs/api-platform/content/add-and-configure-your-first-api>

### Export your API

Instructions for exporting your API via an API can be found at this link

<http://apigee.com/docs/api/api_methods/105-export-an-api>

Instructions for exporting you API using the UI can be found at this link

<http://apigee.com/docs/api-platform/content/edit-api-revisions-offline#-a-class-jumplink-name-export-a-downloading-an-api-revision>

### Decompress API Bundle

The API bundle will come zipped, to achieve access to the contents use the un archiving utility of choice and unzip the file.

Once unzipped you will have a folder named apiproxy, this folder contains all of the configuration for your proxy and the folder structure needed for re-importing of the configuration.

The composition of the folder can be described as below.

{ApiName}.xml - A file that contains descriptors for the content

policies - A folder that contains xml policies

proxies - A folder that contains information about your proxy configurations (inbound)

targets - A folder that contains information about target configurations (outbound)

resources - A folder that contains any scripts (Java, js, and python)

Note: when creating scripts place your scripts in folders with the name of the script ext i.e. java, jsc, py

## 

# Building API bundles

### What is an Apigee bundle?

Apigee bundles can be described as a zipped file system composed of configuration, scripts and code. The file system when extracted is composed of the following structure.

|-apiproxy

|---proxies

|---resources

|---stepdefinitions

|---targets

### Common API building strategies

There are three common building strategies commonly used when doing file-system based development. All of these processes use the same underlying method of building the bundles. The underlying method comes in the form of using a zip utility to archive the intended file-system, this would also be the first strategy.

### Building Strategy - Zipping the file-system

This strategy as noted above is the basis for subsequent strategies. A common way to zip a the apigee files system is to use a file-system console like cmd (Windows) or Terminal (Mac OSX) then traverse your files system to the folder that contains the root folder that holds your configuration e.g. /yourApp/apiproxy/.. . Note that your containing folder should be named apiproxy.

Once you’re in the folder you can run this command *zip -r nameOfApi.zip \* apiproxy*

This method creates a zip that you can then import into your Enterprise Apigee account via one of the acceptable deploy models (See …).

### 

### Building Strategy - Maven

The maven build strategy is a good strategy if your current enterprise build and deploy strategies already use Maven or Maven compatible tools. Using the Apigee Maven plugin for build and deployment also gives you an extra set of configuration that you can use to change configuration elements based on where you want to deploy your Apigee bundle.

An Example Scenario for this use would be the maven configuration file to replace the number of requests it takes to trip a spike arrest policy. In your non-production environment you may want this policy to take effect when 10 requests a minute is surpassed, in production you may want this policy to trip when 10000 requests a minute is surpassed.

Using the maven configuration file you can build environment profiles which will allow you to do xpath replacements on certain policies.

When using maven there are some files that are involved that include configuration that are used by maven and the Apigee maven plugin.

|-apiproxy

|---proxies

|---resources

|---stepdefinitions

|---targets

|-pom.xml

|-config.json

### POM.xml

The pom.xml file is automatically generated if you built your workspace using the maven archetype. Despite this, you will want to inspect this file to ensure completeness and accuracy (for instance, the environments are generically entered as “test’ and “prod”). And if you did not use that method fret not, you can still use maven, you just need to create a couple of pom.xml files from scratch using examples in this document.

#### Creating a maven compatible file structure

First thing that is needed to make an existing workspace Maven-compatible is to add a directory called parent-pom in the directory that contains your application folders. For instance you can see the parent-pom folder in this structure is highlighted in blue.

|-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

|-----------stepdefinitions

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

|-----java

|-----portal

|---test

The contents of the parent pom folder will contain a single pom.xml file. This file typically contains most of the configuration of maven and the plugin, it also contains credentials for the Apigee platform.

#### parent-pom/pom.xml example

|  |
| --- |
| <?xml version="1.0"?>  <!--  Copyright (C) 2014 Apigee Corporation  Licensed under the Apache License, Version 2.0 (the "License");  you may not use this file except in compliance with the License.  You may obtain a copy of the License at  http://www.apache.org/licenses/LICENSE-2.0  Unless required by applicable law or agreed to in writing, software  distributed under the License is distributed on an "AS IS" BASIS,  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  See the License for the specific language governing permissions and  limitations under the License.  -->  <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>  </pluginRepositories>  <modules>  <module>../forcastweatherapi</module>  </modules>  <build>    <pluginManagement>  <plugins>  <plugin>  <groupId>io.apigee.build-tools.enterprise4g</groupId>  <artifactId>apigee-edge-maven-plugin</artifactId>  <version>1.0.0</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>${basedir}/target/apiproxy</outputDirectory>  <resources>  <resource>  <!-- source -->  <directory>apiproxy</directory>  </resource>  </resources>  </configuration>  </execution>  </executions>  </plugin>  <plugin>  <groupId>io.apigee.build-tools.enterprise4g</groupId>  <artifactId>apigee-edge-maven-plugin</artifactId>  <configuration>  <skip>true</skip>  </configuration>  </plugin>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>2.3.2</version>  <configuration>  <source>1.6</source>  <target>1.6</target>  </configuration>  </plugin>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-surefire-plugin</artifactId>  <version>2.9</version>  <configuration>  <testFailureIgnore>false</testFailureIgnore>  </configuration>  </plugin>      </plugins>  </build>      <!-- This is where you add the environment specific properties under various profile names -->  <profiles>  <profile>  <id>test</id>  <properties>  <apigee.profile>test</apigee.profile>  <apigee.env>test</apigee.env>  <apigee.hosturl>https://api.enterprise.apigee.com</apigee.hosturl>  <apigee.apiversion>v1</apigee.apiversion> <!-- value of version in https://api.enterprise.apigee.com/v2 -->  <apigee.org>demo30</apigee.org>  <apigee.username>${username}</apigee.username>  <apigee.password>${password}</apigee.password>  <apigee.options>clean,validate</apigee.options>  </properties>  </profile>  <profile>  <id>prod</id>  <properties>  <apigee.profile>prod</apigee.profile>  <apigee.env>prod</apigee.env>  <apigee.hosturl>https://api.enterprise.apigee.com</apigee.hosturl>  <apigee.apiversion>v1</apigee.apiversion> <!-- value of version in https://api.enterprise.apigee.com/v2 -->  <apigee.org>apigee-cs</apigee.org>  <apigee.username>${username}</apigee.username>  <apigee.password>${password}</apigee.password>  <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>  </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. The maven group id is malleable and is also marked in red for both pom examples, the only thing you should note when changing this is that they need to be consistent between applications.

**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), **inactive** (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)

You will note that there are some entries in the xml file that are colored in red text. These entries would be changed to match your environment these entries should be filled out to match your existing environment.

Once you have the parent pom files configured to your liking you will want to traverse to your application directory;

|-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

|-----------stepdefinitions

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

|-----java

|-----portal

|---test

Once you have traversed to the application folder you should 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.

#### pom.xml Sample

|  |
| --- |
| <!--  Copyright (C) 2014 Apigee Corporation  Licensed under the Apache License, Version 2.0 (the "License");  you may not use this file except in compliance with the License.  You may obtain a copy of the License at  http://www.apache.org/licenses/LICENSE-2.0  Unless required by applicable law or agreed to in writing, software  distributed under the License is distributed on an "AS IS" BASIS,  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  See the License for the specific language governing permissions and  limitations under the License.  -->  <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>forecastweatherapi</artifactId>  <version>1.0</version>  <name>forecastweatherapi</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>io.apigee.build-tools.enterprise4g</groupId>  <artifactId>apigee-edge-maven-plugin</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> |

**groupId** element’s content should match that of the same element in the parent pom.xml. You can leave this as apigee for default.

**artifactId** element’s content should be a unique name, typically set to the folder name or the name of the API.

**name** element’s content should match the artifact name.

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

### Config.json

The config.json acts as your 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.

#### Config.json Data Structure

**configuration** - an array of API definitions

**name** - Name of the maven profile

**proxies** - Array of proxy definitions, directly correlates to the proxies folder in your API Bundle

**name** - Name of file to configure

**tokens** - Array of Actions to Invoke on Elements

**xpath** - Path to element that your want to change the value of

**value** - The replacement value

**policies** - Array of proxy definitions, directly correlates to the policies or step definition folder in your API Bundle

**name** - Name of file to configure

**tokens** - Array of Actions to Invoke on Elements

**xpath** - Path to element that your want to change the value of

**value** - The replacement value

**targets** - Array of proxy definitions, directly correlates to the proxies folder in your API Bundle

**name** - Name of file to configure

**tokens** - Array of Actions to Invoke on Elements

**xpath** - Path to element that your want to change the value of

**value** - The replacement value

**config.json Example**

|  |
| --- |
| {  **"configurations"**:[  {  **"name"**:"test",  **"proxies"**:[  {  **"name"**:"default.xml",  **"tokens"**:[  {  **"xpath"**:"/ProxyEndpoint/HTTPProxyConnection/BasePath",  **"value"**:"/somepath/"  }  ]  }  ],  **"policies"**:[  ],  **"targets"**:[  {  **"name"**:"default.xml",  **"tokens"**:[  {  **"xpath"**:"/TargetEndpoint/HTTPTargetConnection/URL",  **"value"**:"http://test.api.testapi.com/proxy/test/testapp.svc"  }  ]  }  ]  }, {  **"name"**:"production",  **"proxies"**:[  {  **"name"**:"default.xml",  **"tokens"**:[  {  **"xpath"**:"/ProxyEndpoint/HTTPProxyConnection/BasePath",  **"value"**:"/someProductioPpath/"  }  ]  }  ],  **"policies"**:[  ],  **"targets"**:[  {  **"name"**:"default.xml",  **"tokens"**:[  {  **"xpath"**:"/TargetEndpoint/HTTPTargetConnection/URL",  **"value"**:"http://api.testapi.com/proxy/test/testapp.svc"  }  ]  }  ]  }  ]  } |

Deploying

To deploy, run the following command line in the subdirectory of your API proxy name:

<name of root>/src/gateway/<API proxy name>

**mvn apigee-enterprise:deploy -P<env name> -Dusername=<MS admin email> -Dpassword=<MS admin pw>**

For example:

mvn apigee-enterprise:deploy -Pprod -Dusername=admin@toopowerful.com -Dpassword=too\_powerful\_password