

# Vert.x Maven Plugin

Kamesh Sampath, Roland Huß, Clement Escoffier

Version 1.0.7, 2017-04-21

# Vert.x Maven Plugin

1. Introduction .....	2
1.1. Using the plugin .....	2
1.2. Packaging .....	3
1.3. Running .....	4
1.3.1. In foreground using <b>vertx:run</b> .....	4
1.3.2. In background using <b>vertx:start</b> .....	4
2. Common Configurations .....	5
3. Common Run Configurations .....	6
4. Maven Goals .....	7
4.1. <b>vertx:setup</b> .....	7
4.1.1. Example .....	7
4.2. <b>vertx:initialize</b> .....	8
4.2.1. Configuration .....	8
4.2.2. Redeployment .....	9
4.3. <b>vertx:package</b> .....	9
4.3.1. Configuration .....	9
4.4. <b>vertx:run</b> .....	9
4.5. <b>vertx:debug</b> .....	11
4.6. <b>vertx:start</b> .....	11
4.7. <b>vertx:stop</b> .....	12
4.8. <b>vertx:setup</b> .....	12
4.8.1. Example .....	12
5. Example Configurations .....	14
5.1. vert.x:package Examples .....	14
5.1.1. Webjars and javascript dependencies .....	14
5.1.2. Using full names for webjars and javascript dependencies .....	15
5.2. vert.x:start Examples .....	16
5.2.1. start goal with custom java options .....	17
5.2.2. stopping one or more applications .....	17
5.2.3. Adding some extra options to run command .....	18
6. How does the redeploy work .....	19
6.1. References .....	19



# Chapter 1. Introduction

[Eclipse Vert.x](#) is a toolkit to build reactive and distributed systems on the top of the JVM. Vert.x applications can be developed in multiple languages such as Java, JavaScript, Groovy, Scala, Kotlin, Ceylon, Ruby... As a toolkit, Vert.x is un-opinionated.

The `maven-vertx-plugin` is a plugin for Apache Maven for packaging and running Vert.x applications. The plugin tries provide an opinionated way to build Vert.x application:

- applications are packaged as a *fat* jar (containing your code and all the dependencies including Vert.x)
- the application is configured with an `application.json` file or an `application.yml` file (translated to json).

The last version of the Vert.x Maven Plugin is **1.0.7**.

## 1.1. Using the plugin

The plugin provides a set of goals such as:

- `setup` - add the Vert.x Maven Plugin to your `pom.xml`
- `initialize` - manage *js* dependencies and *webjars*, also initiates the redeployment
- `package` - package your Vert.x application as a *fat* jar
- `run` - run your application
- `start / stop` - start your application in background / stop it

Generally, you will use the following configuration:

```

<project>
  ...
  <build>
    <plugins>
      ...
      <plugin>
        <groupId>io.fabric8</groupId>
        <artifactId>vertx-maven-plugin</artifactId>
        <version>${version}</version>
        <executions>
          <execution>
            <id>vmp</id>
            <goals>
              <goal>initialize</goal>
              <goal>package</goal>
            </goals>
          </execution>
        </executions>
        <configuration>
          <redeploy>true</redeploy>
        </configuration>
      </plugin>
      ...
    </plugins>
  </build>
  ...
</project>

```

## 1.2. Packaging

This plugin packaged the Vert.x application as a fat or über jar. These jars are executable, and so can be launched using: `java -jar <my-app.jar>`. The packaging process adds certain **MANIFEST.MF** entries that control how the application is launched. The plugin takes care of adding the required entries to the **MANIFEST.MF** with values configured using the [Common Configuration](#)

The following are the Manifest entries that will be added based on the configuration elements:

Table 1. Package configuration

Property	Manifest Attribute	Remarks
vertx.verticle	Main-Verticle	The main verticle, i.e. the entry point of your application. Used when the <code>Main-Class</code> is <code>io.vertx.core.Launcher</code> .
vertx.launcher	Main-Class	The main class used to start the application, defaults to <code>io.vertx.core.Launcher</code>

The package is generated using: `mvn clean package`

For more information on packaging refer to [vertx:package](#)

## 1.3. Running

The plugin allows running vert.x applications in following ways:

### 1.3.1. In foreground using [vertx:run](#)

The Vert.x application could also be run without building the fat jars, typically during the development mode. The [vertx:run](#) goal helps in running the application. The application will be run in as forked process with additional configurations based on the goals.

To run the application vert.x application in foreground, execute the following maven command

```
mvn clean compile vertx:run
```

You can enable the *edeploy* mode rebuilding and restarting the application upon file changes.

For more information on configuration options refer to [vertx:run](#)

### 1.3.2. In background using [vertx:start](#)

This allows the application be run using the uber or fat jar as background process. This option also attaches a configurable id or an autogenerated id which could be used to stop the process using [vertx:stop](#)

To start the application vert.x application in background, execute the following maven command

```
mvn clean install vertx:start
```

For more information on configuration options refer to [vertx:start](#)

To stop the application vert.x application running in background, execute the following maven command

```
mvn vertx:stop
```

For more information on configuration options refer to [vertx:stop](#)

## Chapter 2. Common Configurations

All goals share the following configuration:

*Table 2. Package configuration*

Element	Description	Default	Property
verticle	Main verticle to start up		vertx.verticle
launcher	Main class to use	io.vertx.core.Launcher	vertx.launcher
skip	Skip the processing executed by this plugin	false	vertx.skip

# Chapter 3. Common Run Configurations

These are the common configuration shared by the run based goals such as **run**, **start** and **stop**.

Table 3. Run configuration

Element	Description	Property	Default
config	the application configuration file path that will be passed to the vertx launcher as <b>-conf</b> . If a yaml file is configured then it will be converted to json by the plugin. The converted file will be saved in <code>\${project.outputDir}/conf</code> directory	vertx.config	<code>\${basedir}/src/main/\${project.artifactId}.json</code> or <code>\${basedir}/src/main/\${project.artifactId}.yaml</code> or <code>\${basedir}/src/main/\${project.artifactId}.yml</code>
redeploy	controls whether vertx redeploy is enabled		false
workDirectory	The working directory of the running process of the application	vertx.directory	<code>\${project.basedir}</code>



Right now the plugin supports only file based Vert.x configuration



# Chapter 4. Maven Goals

This plugin supports the following goals which are explained in detail in the next sections.

Table 4. Plugin Goals

Goal	Description
<code>vertx:setup</code>	Add the vertx-maven-plugin to your <code>pom.xml</code> file
<code>vertx:initialize</code>	Copy <i>js</i> dependencies to <i>webroot</i> , unpack webjars to <i>webroot</i> and initialize the redeployment mode
<code>vertx:package</code>	Package Vert.x applications
<code>vertx:run</code>	Run a Vert.x application in foreground
<code>vertx:debug</code>	Debug a Vert.x application in foreground
<code>vertx:start</code>	Run a Vert.x application in daemon mode with specific id
<code>vertx:setup</code>	Sets up vert.x maven plugin and basic vert.x dependencies on an existing maven project
<code>vertx:stop</code>	Stops the vert.x application running in daemon mode

## 4.1. vertx:setup

This goal adds the Vert.x Maven Plugin to your `pom.xml` file. The plugin is configured with a default configuration.

### 4.1.1. Example

```
mvn io.fabric8:vertx-maven-plugin:1.0.7:setup
```

The setup goal by default uses the plugin property `vertx-core-version` from the plugin properties file `vertx-maven-plugin.properties` as the vert.x version of the project, if you wish to override the vert.x version, then you can run the same command as above with `-DvertxVersion=<your-vertx-version>`

e.g.

```
mvn io.fabric8:vertx-maven-plugin:{version}:setup -DvertxVersion=3.4.0
```

This will configure the vert.x and its dependencies to `3.4.0` i.e. Maven project property `vertx.version` set to `3.4.0`

You can also generate a project if you don't have a `pom.xml` file:

```
mvn io.fabric8:vertx-maven-plugin:1.0.7:setup \
  -DprojectId=org.acme \
  -DprojectArtifactId=acme-project \
  -DprojectVersion=1.0-SNAPSHOT \ # default to 1.0-SNAPSHOT
  -Dverticle=org.acme.Foo \
  -Ddependencies=web,jmx,mongo
```

The **verticle** parameter creates a new verticle class file. The **dependencies** parameters specifies the Vert.x dependencies you need.

Dependencies are selected from a list of know dependencies matching the one listed on <http://vertx.io/docs>. In addition, you can configure dependencies using the following syntax: **groupId:artifactId:version:classifier** such as in:

```
io.vertx:vertx-codetrans => Inherit version from BOM
commons-io:commons-io:2.5 => Regular dependency
io.vertx:vertx-template-engines:3.4.1:shaded => Dependency with classifier
```

## 4.2. vertx:initialize

This goal has several aims:

- copy *js* dependencies to the *webroot* directory
- unpack *webjars* dependencies to the *webroot* directory
- initialize the *recording* of the build used for the redeployment

### 4.2.1. Configuration

The initialize goal has the following parameters apart from the ones mentioned in [Common Configuration](#)

Table 5. Package Configuration

Element	Description	Property	Default
webRoot	The location where <i>js</i> dependencies and <i>webjars</i> are copied.	&nbsp;	\${project.baseDir}/target/classes/webroot
stripWebJarVersion	Whether or not the version is stripped when unpacking webjars	&nbsp;	true
stripJavaScriptDependencyVersion	Whether or not the version is stripped when copying the JavaScript file	&nbsp;	true

The `webroot` directory is generally used by the `StaticHandler` from Vert.x Web.

### 4.2.2. Redeployment

The `initialize` goal is used to configure the redeployment used in `vertx:run`. It starts observing the executed plugins in your build to *replay* them when a file changes.

## 4.3. vertx:package

This goal packages a Vert.x application as fat or über jar with its dependencies bundled as part of the jar.

### 4.3.1. Configuration

The package goal has the following parameters apart from the ones mentioned in [Common Configuration](#)

Table 6. Package Configuration

Element	Description	Property	Default
service Provider Combination	Whether or not SPI files ( <code>META-INF/services</code> ) need to be combined. Accepted values as <code>combine</code> and <code>none</code> .	<code>&lt;property&gt;</code>	<code>combine</code>
classifier	The classifier to add to the artifact generated. If given, the artifact will be attached with that classifier and the main artifact will be deployed as the main artifact. If this is not given (default), it will replace the main artifact and only the Vert.x (fat) jar artifact will be deployed (in the Maven sense). Attaching the artifact allows to deploy it alongside to the original one. Attachment is controlled using the <code>attach</code> parameter.	<code>&lt;property&gt;</code>	<code>&lt;property&gt;</code>
attach	Whether or not the created archive needs to be attached to the project. If attached, the fat jar is installed and deployed alongside the main artifact (plain jar). Notice that you can't disable attachment if the classifier is not set (it would mean detaching the main artifact).	<code>&lt;property&gt;</code>	<code>true</code>

## 4.4. vertx:run

This goal allows to run the Vert.x application as part of the maven build. The application is always run as a forked process.

The goal has the following exclusive configuration apart from the applicable [Common Run Configuration](#)

Element	Description	Property	Default
runArgs	The list of arguments that can be passed as part of <code>run</code> command. You can also pass the required run arguments using the system property. e.g. <code>-Dvertx.runArgs="--worker --instances=2"</code> . The system property takes precedence over the plugin configurations	vertx.runArgs	&nbsp;
config	The configuration file to use to configure the application. This property is passed as the <code>-config</code> option to vertx run.	&nbsp;	src/main/config/application.json or src/main/config/application.yml
jvmArgs	The Java Options that will be used when starting the application, these are the values that are typically passed to vert.x applications using <code>--java-opts</code> .	vertx.jvmArguments	<i>no arguments</i>
redeploy	Whether or not the redeployment is enabled	&nbsp;	false
redeployScanPeriod	The file system check period (in milliseconds)	vertx.redeploy.scan.period	1000
redeployGracePeriod	The amount of time (in milliseconds) to wait between 2 re-deployments	vertx.redeploy.grace.period	1000
redeployTerminationPeriod	the amount of time (in milliseconds) to wait after having stopped the application (before launching user command). This is useful on Windows, where the process is not killed immediately.	vertx.redeploy.termination.period	1000 (1 second)
worker	If specified the verticle will be a worker verticle	&nbsp;	false

When the redeployment is enabled, it replays the plugin configured between the *generate-source* and *process-classes* phases.

So to start a Vert.x application just launch:

```
mvn vertx:run
```

If the sources are not compiled, the plugin executes `mvn compile` for you.

**IMPORTANT:** `vertx:run` starts the application in a forked JVM.

## 4.5. vertx:debug

This goal allows to debug the Vert.x application. The application is started in a JVM with the debugging enabled. In your IDE, you need to create a **remote** debug configuration to plug the debugger.

The goal does not have any exclusive configuration, [Common Run Configuration](#) defines all the applicable configurations for the goal.

Element	Description	Property	Default
config	The configuration file to use to configure the application. This property is passed as the <b>-config</b> option to vertx run.	&nbsp;	src/main/config/application.json or src/main/config/application.yml
jvmArgs	The Java Options that will be used when starting the application, these are the values that are typically passed to vert.x applications using --java-opts.	vertx.jvmArguments	<i>no arguments</i>
debugPort	The debugger port	debug.port	5005
debugSuspend	Whether or not the application must wait until a debugger is attached to start	debug.suspend	false

**IMPORTANT:** the redeploy mode and debug cannot be used together. When **vertx:debug** is launched, the redeployment mode is disabled.

## 4.6. vertx:start

This goal allows to start the Vert.x application as a background process from maven build. This goal triggers the vert.x **start** command, passing the configuration values as mentioned below.

Table 7. Run configuration

Element	Description	Property	Default
startMode	The property to decide how the vert.x application will be started in background. The application can be started in <b>jar</b> mode in which the application will be packaged as fat jar and started, or can be run in <b>exploded</b> mode where the application will be launched with exploded <i>classesDirectory</i> and maven dependencies to the classpath	vertx.start.mode	jar

Element	Description	Property	Default
appId	The application id that will added as <b>-id</b> option to the vert.x start command	vertx.app.id	If this is not passed a default uuid will be generated and set as appId
jvmArgs	The Java Options that will be used when starting the application, these are the values that are typically passed to vert.x applications using <code>--java-opts</code>	vertx.jvmArguments	

Apart from the above list of exclusive start configuration, the goal shares the common **Common Run Configuration** with the following configuration ignored by the goal,

- redeploy

## 4.7. vertx:stop

This goal allows to stop the vert.x application running as background process from maven build. This goal triggers the vert.x **stop** command, passing the configuration values as mentioned below.

Table 8. Run configuration

Element	Description	Property	Default
appIds	The application id's that will stopped using the vert.x stop command		If this is not passed, the vertx-start-proc.id file present workingDirectory will be read for the application id

Apart from the above list of exclusive start configuration, the goal shares the common **Common Run Configuration** with the following configuration ignored by the goal,

- redeploy

## 4.8. vertx:setup

This goal adds the Vert.x Maven Plugin to your `pom.xml` file. The plugin is configured with a default configuration.

### 4.8.1. Example

```
mvn io.fabric8:vertx-maven-plugin:1.0.7:setup
```

The setup goal by default uses the plugin property `vertx-core-version` from the plugin properties file `vertx-maven-plugin.properties` as the vert.x version of the project, if you wish to override the vertx

version, then you can run the same command as above with `-DvertxVersion=<your-vertx-version>`

e.g.

```
mvn io.fabric8:vertx-maven-plugin:{version}:setup -DvertxVersion=3.4.0
```

This will configure the vert.x and its dependencies to `3.4.0` i.e. Maven project property `vertx.version` set to `3.4.0`

You can also generate a project if you don't have a `pom.xml` file:

```
mvn io.fabric8:vertx-maven-plugin:1.0.7:setup \  
  -DprojectId=org.acme \  
  -DprojectArtifactId=acme-project \  
  -DprojectVersion=1.0-SNAPSHOT \ # default to 1.0-SNAPSHOT \  
  -Dverticle=org.acme.Foo \  
  -Ddependencies=web,jmx,mongo
```

The `verticle` parameter creates a new verticle class file. The `dependencies` parameters specifies the Vert.x dependencies you need.

Dependencies are selected from a list of know dependencies matching the one listed on <http://vertx.io/docs>. In addition, you can configure dependencies using the following syntax: `groupId:artifactId:version:classifier` such as in:

```
io.vertx:vertx-codetrans => Inherit version from BOM  
commons-io:commons-io:2.5 => Regular dependency  
io.vertx:vertx-template-engines:3.4.1:shaded => Dependency with classifier
```

# Chapter 5. Example Configurations

The following sections shows example plugin snippets for the goals provided by the plugin.



please update the plugin version as needed (latest is **1.0.7**).

## 5.1. vert.x:package Examples

This is the default configuration. It builds the fat jar in the *package* phase, and support *.js* and webjars dependencies. `mvn compile vertx:run` enables the redeployment.

```
<plugin>
  <groupId>io.fabric8</groupId>
  <artifactId>vertx-maven-plugin</artifactId>
  <version>${version}</version>
  <executions>
    <execution>
      <id>vmp</id>
      <goals>
        <goal>initialize</goal>
        <goal>package</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <redeploy>true</redeploy>
  </configuration>
</plugin>
```

### 5.1.1. Webjars and javascript dependencies

When the *js* dependencies and the webjars are handled, it stripped the version by default. Let's illustrate this with the following dependencies:

```
<dependency>
  <groupId>io.vertx</groupId>
  <artifactId>vertx-web</artifactId>
  <version>3.4.1</version>
  <classifier>client</classifier>
  <type>js</type>
</dependency>
<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>jquery</artifactId>
  <version>3.1.1</version>
</dependency>
```



The first dependency is a `js` dependency (`<type>js</type>`). The plugin resolves the dependency and copy the file to the `webroot` directory (the default is `${project.baseDir}/target/classes/webroot`). The output file name is: `vertx-web-client.js`.

The second dependency is a webjar. A `webjar` is a jar file containing web resources. The plugin extracts the web resource to the `webroot` directory. In this example, it creates: `${project.baseDir}/target/classes/webroot/jquery/jquery.js` (and other includes resources).

### 5.1.2. Using full names for webjars and javascript dependencies

By default the plugin stripped the version. However this behavior can be disabled using:

```
<plugin>
  <groupId>io.fabric8</groupId>
  <artifactId>vertx-maven-plugin</artifactId>
  <version>${version}</version>
  <executions>
    <execution>
      <id>vmp</id>
      <goals>
        <goal>initialize</goal>
        <goal>package</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <redeploy>true</redeploy>
    <stripWebJarVersion>>false</stripWebJarVersion>
    <stripJavaScriptDependencyVersion>>false</stripJavaScriptDependencyVersion>
  </configuration>
</plugin>
```

In this case, the `vertx-web-client.js` output file is: `vertx-web-3.4.1-client.js`, while for jquery, resources are unpacked in `${project.baseDir}/target/classes/webroot/jquery/3.1.1`. You can always look into the `target/classes/webroot` directory to check the output files.

Finally, you can also configure the `webRoot` dir:

```

<plugin>
  <groupId>io.fabric8</groupId>
  <artifactId>vertx-maven-plugin</artifactId>
  <version>${version}</version>
  <executions>
    <execution>
      <id>vmp</id>
      <goals>
        <goal>initialize</goal>
        <goal>package</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <webRoot>target/classes/assets</webRoot>
    <redeploy>true</redeploy>
  </configuration>
</plugin>

```

Notice that the `webRoot` directory is relative to the project directory.

## 5.2. vert.x:start Examples

You can start your application in background using: `mvn clean package vertx:start`. The application is started in background. An id is automatically associated with the process. This id can be specified:

```
mvn clean package vertx:start -Dvertx.app.id=my-vertx-app
```

This id is useful to stop the application:

```
mvn vertx:stop -Dvertx.app.id=my-vertx-app
```

The application id can be set in the `pom.xml` file:

```

<plugin>
  <groupId>io.fabric8</groupId>
  <artifactId>vertx-maven-plugin</artifactId>
  <version>${version}</version>
  <executions>
    <execution>
      <phase>initialize</phase>
      <phase>package</phase>
    </execution>
  </executions>
  <configuration>
    <appId>my-app-id</appId>
  </configuration>
</plugin>

```

### 5.2.1. start goal with custom java options

Because it's a forked process, passing Java options needs to be done explicitly using a specific property:

```

<plugin>
  <groupId>io.fabric8</groupId>
  <artifactId>vertx-maven-plugin</artifactId>
  <version>${version}</version>
  <executions>
    <execution>
      <phase>initialize</phase>
      <phase>package</phase>
    </execution>
  </executions>
  <configuration>
    <jvmArgs> ①
      <jvmArg>-Xms512m</jvmArg>
      <jvmArg>-Xmx1024m</jvmArg>
    </jvmArgs>
  </configuration>
</plugin>

```

① The jvm arguments that gets passed as `--java-opts` to the vert.x application

You can also pass these parameters in the command line:

```
mvn clean package vertx:start -Dvertx.jvmArguments=-Xms512m -D-Dfoo=bar
```

### 5.2.2. stopping one or more applications

When you have configured to [\[start-with-app-id\]](#) or know the application ids, then you can add list

of application ids as shown below to trigger stop of the those applications

```
<plugin>
  <groupId>io.fabric8</groupId>
  <artifactId>vertx-maven-plugin</artifactId>
  <version>${version}</version>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>start</goal>
      </goals>
      <configuration>
        <appIds> ①
          <appId>my-app-id-1</appId>
          <appId>my-app-id-2</appId>
        </appIds>
      </configuration>
    </execution>
  </executions>
</plugin>
```

① List of custom unique application ids

### 5.2.3. Adding some extra options to run command

```
<plugin>
  <groupId>io.fabric8</groupId>
  <artifactId>vertx-maven-plugin</artifactId>
  <version>${version}</version>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>run</goal>
      </goals>
      <configuration>
        <runArgs>
          <runArg>--cluster</runArg>
        </runArgs>
      </configuration>
    </execution>
  </executions>
</plugin>
```

# Chapter 6. How does the redeploy work

During the *initialize* phase, the plugin start observing the mojos (Maven plugins) that are executed. When the Vert.x application is launched, it watches for changed in `src/main`. When a file is changed (created, updated or deleted), it replays the executed mojos. It executes all the mojos from the *generate-sources* to the *process-classes* phases, using the same configuration are the initial (observed) one.

When these mojos are executed, they may update files in `target/classes`. The Vert.x application has been launched to observe changed from this location and restart when change happens. The application is restarted completely, *i.e.* stopped and restarted.

Such mechanism let you use any Maven plugin (executed in the right set of phase). The plugin is re-executed and then the Vert.x application is restarted.

## 6.1. References

[Maven Properties](#)