Maven2 Archetypes

Maven2 Archetypes

Maven2 has a powerful feature called Archetype.

As part of the Cargo project we have created some archetypes that you can use to get a working sample project that uses the Cargo Maven2 plugin.

Single Webapp Module Archetype

This is the most basic example, showing how to configure Cargo and Maven2 to run functional tests directly from a single webapp module. To run it, execute (one one line):

```
mvn archetype:generate
-DarchetypeGroupId=org.codehaus.cargo

-DarchetypeArtifactId=cargo-archetype-webapp
-single-module

-DarchetypeVersion=<version>
```

where **<version>** is the version of the Cargo Archetype, as a result the CARGO version to use.

The Maven archetype plugin will then ask you what groupId, artifactId, version, and package you would like to use for generation. Once your Maven project is generated, simply run:

```
mvn clean verify
```

That command will:

- Build the web application
- If needed, download Jetty
- Configure Jetty
- Start the Jetty container
- Deploy your web application to the Jetty container
- Run some tests on the deployed application
    - We've used classic JUnit tests, but you can use whatever you like, really
- Stop the Jetty container

That archetype also demonstrates another powerful feature of CARGO coupled with Maven. If you now run:

```
mvn clean verify -P tomcat6x
```

the same steps will be done with Tomcat instead of Jetty; thanks to Maven's profiles.

The archetype actually ships with Maven profiles for the **jetty7x** (default), **jetty6x**, **tomcat6x**, **jonas5x**, **jboss51x**, **jboss71x**, **glassfish3x** and **geronimo3x** containers. We tried to show with these archetypes as many examples as possible, so you will find that:

- Some of these profiles define a **standalone** container:
  - The profiles **jetty7x**, **jonas5x**, **jboss71x** and **geronimo3x** download the server using the ArtifactInstaller.
  - The profiles **jboss51x**, **glassfish3x** and **tomcat6x** download the server using the ZipUrlInstaller.
- The profile **jetty6x** define an **embedded** container.

You can of course add and use any other container from the Containers list.

Separate Functional Test Module Archetype

Shows how to configure Cargo and Maven2 to run functional tests by creating a functional tests module next to the webapp module. To run it, execute (one one line):

```
mvn archetype:generate
-DarchetypeGroupId=org.codehaus.cargo

-DarchetypeArtifactId=cargo-archetype-webapp
-functional-tests-module

-DarchetypeVersion=<version>
```

where **<version>** is the version of the Cargo Archetype, as a result the CARGO version to use.

The remaining instructions for this archetype remain the same as the *Single Webapp Module Archetype* archetype.

Datasource Definition Archetype

Shows how to configure Cargo and Maven2 to for a more complex setup, in particular with datasource definitions. This includes:

- Definition of an Apache Derby datasource in the container's configuration
- Adding in the Apache Derby driver to the container

Of course, the same kind of setup works for any other datasource provider and driver.

To run it, execute (one one line):

```
mvn archetype:generate
-DarchetypeGroupId=org.codehaus.cargo

-DarchetypeArtifactId=cargo-archetype-webapp
-with-datasource

-DarchetypeVersion=<version>
```

where **<version>** is the version of the Cargo Archetype, as a result the CARGO version to use.

The overall remaining instructions for this archetype remain the same as the *Single Webapp Module Archetype* arch etype, with the following differences:

- Not all containers support datasources. For details configuring datasource, please read: DataSource and Resource Support.
- The profiles that ship with this archetype are **jetty7x** (default), **tomcat6x**, **glassfish3x**, **jboss51x**, **jboss71x**, **jonas5x** and **geronimo3x**.

As in the other examples, what's interesting to see is that the datasource definition on CARGO remains the same even if you switch container; as CARGO handles all the container-specific datasource setup steps.

Webapp Creation and Remote Deployment Archetype

This example shows you how to use Cargo to deploy your webapp to a running container remotely and perform integration tests on it. That container can therefore be running on any machine. To run it, execute (one one line):

```
mvn archetype:generate
-DarchetypeGroupId=org.codehaus.cargo

-DarchetypeArtifactId=cargo-archetype-remote
-deployment

-DarchetypeVersion=<version>
```

where **<version>** is the version of the Cargo Archetype, as a result the CARGO version to use.

Once your Maven project is generated, simply run:

```
mvn clean verify
```

That command will:

- Build the web application
- Connect to the remote Tomcat manager
- Deploy your application
- Run integration tests to verify that the application is running file
    - We've used classic JUnit tests, but you can use whatever you like, really
- Undeploy your application

This archetype supports the following properties that can be changed by passing `-DpropertyName=propertyValue` arguments to the `mvn` command:

- **servlet.port** changes the target servlet port, the archetype sets it to `8080`
- **hostname** changes the target servlet's hostname, the archetype sets it to `localhost`
- **username** sets the username, the archetype sets it to `admin`
- **password** sets the password, the archetype sets it to an empty password

This archetype ships with Maven profiles for the following containers:

- **tomcat6x** (default), using the Tomcat manager
- **jetty6x** and **jetty7x**. Before using the Jetty remote deployer, please read: Jetty Remote Deployer.
- **jonas5x**
- **jboss51x** and **jboss7x**, including adding in the special dependencies for the JBoss remote deployer. For more details, please read: JBoss Remote Deployer.
- **glassfish3x**, including adding in the GlassFish-specific dependencies for JSR88. For more details, please read: JSR88.

Daemon Archetype

Shows how to configure Cargo and Maven2 to use the Cargo Daemon's features in order to remotely start the container, deploy your deployable on it, do some tests, and shut the server down remotely. To run it, execute (one one line):

```
mvn archetype:generate
-DarchetypeGroupId=org.codehaus.cargo

-DarchetypeArtifactId=cargo-archetype-daemon

-DarchetypeVersion=<version>
```

where **<version>** is the version of the Cargo Archetype, as a result the CARGO version to use.

The Maven archetype plugin will then ask you what groupId, artifactId, version, and package you would like to use for generation. Once your Maven project is generated, you need to first start the Cargo Daemon. Then, simply run:

```
mvn clean verify
-Dhostname=<daemon.hostname>
-Ddaemon.port=<daemon.port>
```

where **<daemon.hostname>** is the hostname / IP address of the machine running the Cargo Daemon and **<daemon.port>** the port for it.That command will:

- Build the web application
- If needed, download Jetty
- Send the downloaded Jetty package to the remote Cargo Daemon instance
- Configure Jetty on the remote machine
- Start the Jetty container on the remote machine
- Deploy your web application to the Jetty container on the remote machine
- Run some tests on the deployed application
    - We've used classic JUnit tests, but you can use whatever you like, really
- Stop the Jetty container on the remote machine

The archetype actually ships with Maven profiles for the **jetty7x** (default), **tomcat6x**, **jonas5x**, **jboss51x**, **jboss71x**, **glassfish3x** and **geronimo3x** containers.

You can of course add and use any other container from the Containers list.