Getting Started

Very quick start

Codehaus Cargo can be directly run on any existing Maven Java EE project (WAR, EAR or other) by running:

```
mvn clean verify org.codehaus.cargo:cargo-maven2-plugin:run
```

This will create a default Jetty 7.x installed local container and start it using the Cargo Maven 2 / Maven 3 plugin with your Maven 2 / Maven 3 project's deployable (a WAR, for example) deployed to it; so you can run manual tests (as a first introduction).

What is magic is that if you now want to run the same tests with Tomcat 7.x you simply need to run (in one line):

```
mvn clean verify org.codehaus.cargo:cargo-maven2-plugin:run
    -Dcargo.maven.containerId=tomcat7x
    -Dcargo.maven.containerUrl=http://repo1.maven.org/maven2/org/apache/tomcat/tomcat/7.0.68/tomcat-7.0.68.zip
```

That command will automatically download Tomcat 7.0.68 from the specified URL (taking into account any proxy server setting you would have in Maven 2 / Maven 3), instantiate the container, create a local configuration with your application and run it. It will also save the downloaded container in the default directory (see the Maven 2 / Maven 3 Plugin Reference Guide for details), so it won't get downloaded when you run the same command twice.

Now, if you want to run this time on Glassfish 3.x with with the HTTP port set to `9000`, run:

```
mvn clean verify org.codehaus.cargo:cargo-maven2-plugin:run
    -Dcargo.maven.containerId=glassfish3x
    -Dcargo.maven.containerUrl=http://download.java.net/glassfish/3.1.1/release/glassfish-3.1.1.zip
    -Dcargo.servlet.port=9000
```

Codehaus Cargo's main advantage is that the commands and configuration remains the same for any version of any supported container - be it Tomcat, Jetty, JBoss, JOnAS, GlassFish, WebLogic, etc.

Like it? Well, keep on reading, then!

More examples

As usual the best way to learn to use a tool is through examples.

We have several Maven 2 / Maven 3 Archetypes that contain sample Maven 2 / Maven 3 projects with different use cases for the CARGO plugin, we would really recommend that you check them out. For more details, read here: Maven2 Archetypes.

In addition here are the typical uses cases covered by the plugin:

- Deploying to a running container
- Generating a container configuration deployment structure
- Merging WAR files
- Starting and stopping a container

The Cargo Maven plugin in detail

Here are the different goals available to call on this plugin:

Goals	Description
	Start a container. That goal will: • If the plugin configuration requires so, installs the container. • If the plugin configuration defines a container with a standalone local configuration, it will create the configuration. • If the plugin configuration contains one or more deployables, it will deploy these to the container automatically.

- Tomcat 7.x
- Tomcat 8.x
- Tomcat 9.x

- TomEE
  - TomEE 1.x
  - TomEE 7.x

- WebLogic
  - WebLogic 8.x
  - WebLogic 9.x
  - WebLogic 10.x
  - WebLogic 10.3.x
  - WebLogic 12.x
  - WebLogic 12.1.x
  - WebLogic 12.2.x

- WebSphere
  - WebSphere 8.5.x
  - WebSphere Liberty

- WildFly
  - WildFly 8.x
  - WildFly 9.x
  - WildFly 10.x

Extensions

- Ant
- Cargo Daemon
- Maven 2 / Maven 3

Independent extensions:

- Gradle
- Mule container

Project Info

- Roadmap
- News
- Issues
- Change log
- Source code
- Mailing lists
- License
- Credits

Development

- Project structure
- Importing sources
- Building
- Code style
- Contributing
- Release procedure
- Continous integration

build passed

Feeds

Site

News

	Maven instance quits (i.e., you see a `BUILD SUCCESSFUL` or `BUILD FAILED` message). If you want to start a container and perform manual testing, see our next goal `cargo:run`.
`cargo:run`	Start a container and wait for the user to press `CTRL + C` to stop. That goal will: • If the plugin configuration requires so, installs the container. • If the plugin configuration defines a container with a standalone local configuration, it will create the configuration. • If the plugin configuration contains one or more deployables, it will deploy these to the container automatically. • If the plugin configuration contains no deployables but the project's packaging is Java EE (WAR, EAR, etc.), it will deploy the project's deployable to to the container automatically. • And, of course, start the container and wait for the user to press `CTRL + C` to stop.
`cargo:stop`	Stop a container.
`cargo:restart`	Stop and start again a container. If the container was not running before calling `cargo:restart`, it will simply be started.
`cargo:configure`	Create the configuration for a local container, without starting it. Note that the `cargo:start` and `cargo:run` goals will also install the container automatically (but will not call `cargo:install`).
`cargo:package`	Package the local container.
`cargo:daemon-start`	Start a container via the daemon. Read more on: Cargo Daemon **Note**: The `daemon:start` goal is actually equivalent to a **restart** in CARGO's terms; in the case a container with the same `cargo.daemon.handleid` already exists then it will be stopped first before your container is started. This also implies that in the case the new container fails to start, the old one will not be restarted.
`cargo:daemon-stop`	Stop a container via the daemon. Read more on: Cargo Daemon
`cargo:deployer-deploy` (aliased to **`cargo:deploy`**)	Deploy a deployable to a running container. **Note**: The `cargo:start` and `cargo:run` do already deploy the deployables specified in the configuration to the container; as a result calling `cargo:deploy` for a container which has been started by CARGO in the same Maven project will most likely cause a second deployment of the same deployables (and might even fail).
`cargo:deployer-undeploy` (aliased to **`cargo:undeploy`**)	Undeploy a deployable from a running container.
`cargo:deployer-start`	Start a deployable already installed in a running container.
`cargo:deployer-stop`	Stop a deployed deployable without undeploying it.
`cargo:deployer-redeploy` (aliased to **`cargo:redeploy`**)	Undeploy and deploy again a deployable. If the deployable was not deployed before calling `cargo:deployer-redeploy` (or its alias `cargo:redeploy`) it will simply be deployed.
`cargo:uberwar`	Merge several WAR files into one.
`cargo:install`	Installs a container distribution on the file system. Note that the `cargo:start` goal will also install the container automatically (but will not call `cargo:install`).
`cargo:help`	Get help (list of available goals, available options, etc.).

The configuration elements are described in the Reference Guide section.