

Deploying to a running container

Cargo supports [deploying](#) to an already running container. This feature is called [Hot Deployment]. You call it by using the (`cargo:deploy`) goal (e.g. `mvn cargo:deploy`).

Note that you can also do [static deployment](#) by simply defining the deployables to deploy in the `<configuration>` element as shown in the [Starting and stopping a container](#) tutorial. In that case the deployables will be deployed before the container starts.

i Not all containers have a Deployer implemented

We haven't finished implementing Deployers for all containers yet. Please [check](#) if your favorite container has it implemented. If not you'll need to deploy your deployables by defining them in a standalone local configuration as shown in the [Starting and stopping a container](#) use case.

i About WAR contexts

Many containers have their specific files for redefining context roots (Tomcat has `context.xml`, JBoss has `jboss-web.xml`, etc.). If your WAR has such a file, the server will most probably use the context root defined in that file instead of the one you specify using the CARGO deployer.

Using a local deployer

A local [deployer](#) is a deployer that deploys deployables on a [Local Container](#) (i.e. a container installed on the same machine where the deployer is executing). Thus you'll need to use a local container id in `<containerId>`. You can check that by reviewing the supported [container list](#) and selecting the container you wish to use.

Example of doing a local deploy to an existing configuration:

```
[...]
<configuration>

  <!-- Container configuration -->
  <container>
    <containerId>resin3x</containerId>
    <home>c:/apps/resin-3.0.9</home> or
    <zipUrlInstaller>

    <url>http://www.caucho.com/download/resin-3.
    0.9.zip</url>
```

```
<downloadDir>${project.build.directory}/down  
loads</downloadDir>
```

```
<extractDir>${project.build.directory}/extra  
cts</extractDir>  
  </zipUrlInstaller>  
</container>
```

```
<!-- Configuration to use with the  
container -->
```

```
<configuration>  
  <type>existing</type>  
  <properties>  
    [...]   
  </properties>  
</configuration>
```

```
<!-- Deployer configuration -->
```

```
<deployer>  
  <type>local</type>  
</deployer>
```

```
<!-- Deployables configuration -->
```

```
<deployables>  
  <deployable>  
    <groupId>war group id</groupId>  
    <artifactId>war artifact  
id</artifactId>  
    <type>war</type>  
    <properties>  
      <context>optional root
```

```
context</context>
  </properties>
  <pingURL>optional url to ping to know
if deployable is done or not</pingURL>
  <pingTimeout>optional timeout to ping
(default 20000 milliseconds)</pingTimeout>
</deployable>
<deployable>
  <groupId>ear group id</groupId>
  <artifactId>ear artifact
id</artifactId>
  <type>ear</type>
  <pingURL>optional url to ping to know
if deployable is done or not</pingURL>
  <pingTimeout>optional timeout to ping
(default 20000 milliseconds)</pingTimeout>
</deployable>
[...]
```

```
    </deployables>
</configuration>
[...]
```

As for the `cargo:start` goal if your project is a J2EE project then the generated artifact will be automatically added to the list of deployables to deploy.

Using a remote deployer

A remote [deployer](#) is a deployer that deploys deployables on a [Remote Container](#) (i.e. a container that is running and that has been started externally from Cargo). Thus you'll need to use an id for a remote container in `<containerId>` and a [Runtime Configuration](#).

Example of doing a remote deploy using a runtime configuration:

```
[...]
<configuration>

    <!-- Container configuration -->
    <container>
        <containerId>tomcat6x</containerId>
        <type>remote</type>
    </container>

    <!-- Configuration to use with the
container -->
    <configuration>
        <type>runtime</type>
        <properties>

<cargo.remote.username>username</cargo.remote
e.username>

<cargo.remote.password>password</cargo.remote
```

```
e.password>
  </properties>
</configuration>

<!-- Deployer configuration -->
<deployer>
  <type>remote</type>
</deployer>

<!-- Deployables configuration -->
<deployables>
  <deployable>
    <groupId>war group id</groupId>
    <artifactId>war artifact
id</artifactId>
    <type>war</type>
    <properties>
      <context>optional root
context</context>
    </properties>
    <pingURL>optional url to ping to know
if deployable is done or not</pingURL>
    <pingTimeout>optional timeout to ping
(default 20000 milliseconds)</pingTimeout>
  </deployable>
  [...]
</deployables>
```

```
</configuration>
[...]
```

As you can see the information to connect and do the deployment to the remote container is specified in the runtime configuration (`cargo.remote.username` and `cargo.remote.password`). The properties to define are deployer-dependent.

Here's an another example, this time deploying the current project's artifact to a running JBoss 4.2.x container using a default 8080 port and default authorizations:

```
[...]
<configuration>
  <container>
    <containerId>jboss42x</containerId>
    <type>remote</type>
  </container>
</configuration>
[...]
```

Just type `mvn cargo:deploy`. Notice that we haven't specified a `<deployer>` element nor a `<configuration>` one: this is because the plugin is smart enough to create default instances for you.

Handling of deployment or undeployment failures

If the deployment, undeployment or redeployment is done without any watchdog, CARGO will rely on the error codes return by the container's deployer to detect failures.

This is sometimes not desired: for example, if one calls `cargo:undeploy` but the target deployable (be it a web application, EAR, etc.) is actually not deployed, then the call to `cargo:undeploy` will return a failure indicating that the deployable to undeploy is not deployed.

To work around the issue, make sure you define your deployable together with a watchdog. For example:

Deployable with a ping URL

```
<deployables>
  <deployable>

  <location>${project.build.directory}/${project.build.finalName}.${project.packaging}</location>

  <pingURL>http://localhost:${servlet.port}/mycontext/index.html</pingURL>
  </deployable>
</deployables>
```

In that case:

- When you will call `cargo:deploy`:
 - If the deployer returns a failure, CARGO will log a message The deployment has failed, with the error returned by the deployer and a severity `INFO`.
 - It will then start the watchdog, to check if the deployable is deployed.
 - If the watchdog detects that the deployable is indeed deployed, the overall result is a `BUILD SUCCESSFUL`
 - If the watchdog detects that the deployable is not deployed, you then get a `BUILD FAILURE`
- When you will call `cargo:undeploy`:
 - If the deployer returns a failure, CARGO will log a message The undeployment has failed, with the error returned by the deployer and a severity `INFO`.
 - It will then start the watchdog, to check if the deployable is not deployed.
 - If the watchdog detects that the deployable is indeed not deployed, the overall result is a `BUILD SUCCESSFUL`
 - If the watchdog detects that the deployable is still deployed, you then get a `BUILD FAILURE`

This functionality is available in the Java API, ANT tasks and Maven2/Maven3 plugin.