# Release management

## Overview

The build & release management framework consist in:

- A git-based SCM (Gitlab)

- Nexus as a binary repository for the applications, shared between all the environments.

- Jenkins as the automation engine

To commit code into Gitlab triggers the build of the application. The build succeeds if the code compiles and successfully passes the unit tests. If the build succeeds the resulting binary artifact is deployed to Nexus and a fabric8 profile pointing to that artifact is generated into Fuse. System, integration and performance tests are successively triggered by the automation server. A manual step is required to promote the artifact to a “production” release version.

## Version management

**Working with Snaphots**

The maven release plugin is used to turn snaphot versions into release versions. Binaries resulting from a snapshot build are deployed in Nexus in a separated repository which is not known by the Fuse environment except for the Dev one.

**Fabric8 profile version**

When the application is a release, its version (2 first digits) is contained in the fabric8 profile name. The version must match the 2 first digits of the version of the binary in Nexus. In addition, the version of the binary must match the tag name in the SCM.

**HTTP GW and URI pattern**

The version of the application (first digit only) must be part of the URI pattern of it endpoint in order to be able to run multiple version at the same time.

* 1. Nexus deploy

Fuse is configured (see chapter8) to find the applications (hosted repository) and its dependencies (online repositories) from Nexus. So, the application has to first be deployed into Nexus prior to being usable in Fuse.

There is an “bug” with the maven “deploy” and “fabric8:deploy” commands when using snapshots versions that leads to a mismatch of versions between the 2 plugins. Therefore the deployment to Fuse must be done in 2 steps:

- mvn deploy

- mvn fabric8:deploy

Once the profile is deployed to Fabric, there is an extra step which consists in assigning the profile to a container.

Deployed artifacts are locally cached by each container. A change in a maven artifact will not be seen by Fuse unless the maven artifact version is incremented. However, snapshot applications being timestamped, a change in a snapshot version will automatically be detected and the profile will automatically be refreshed by the container upon deployment.

Deploying to nexus requires a <distributionManagement> section defined in the pom.xml that points to the location of a repository locally hosted by Nexus (this could also be set on the command line).

<distributionManagement>

<snapshotRepository>

<uniqueVersion>false</uniqueVersion>

<id>nexus-snapshots</id>

<url>http://10.40.11.24:8081/repository/maven-snapshots/</url>

<layout>default</layout>

</snapshotRepository>

<repository>

<uniqueVersion>true</uniqueVersio

<id>nexus-releases</id>

<url>http://10.40.11.24:8081/repository/maven-releases/</url>

<layout>default</layout>

</repository>

</distributionManagement>

To read artifacts is always allowed, even anonymously. However, a username and a password are needed to write (push) artifacts into maven repositories. Those credentials are defined in the maven settings.xml under a <server> section whose ID must match those defined in the <distributionManagement> section of the pom.xml.

<server>

<id>nexus-snapshots</id>

<username>admin</username>

<password>admin123</password>

</server>

<server>

<id>nexus-releases</id>

<username>admin</username>

<password>admin123</password>

</server>

To act as an internet proxy, Nexus must be configured as a global mirror in the maven settings.xml. The mirror will point point to a Nexus group that hides a list of remote repositories.

<mirrors>

<mirror>

<id>nexus</id>

<mirrorOf>\*</mirrorOf>

<url>http://10.40.11.24:8081/repository/public</url>

</mirror>

</mirrors>

* 1. Fabric deploy

The method we chose to deploy to Fabric is the fabric8 maven plugin.

The Fabric8 maven plugin is available in the FuseSource repository.

It must be defined and configured in the pom.xml

<plugin>

<groupId>io.fabric8</groupId>

<artifactId>fabric8-maven-plugin</artifactId>

<version>1.1.0.CR5</version>

</plugin>

The plugin can be customized in several ways:

* using plugin configuration parameters defined in the <configuration> section of the plugin's
* using Camel properties named as “fabric8:xxx”
* using java -Dyyy parameters on the maven command line

The following properties have been defined:

* The Fabric profile name is defined as a global Camel property so that it's visible at the top of the pom:

<fabric8.profile>phcc-pgw</fabric8.profile> // an hyphen is used in the name for grouping purpose

* The target deployment URL is defined as a command line parameter so that nothing has to be changed in the pom if the application has to be deployed somewhere else.

Maven fabric8:deploy -Dfabric8.jolokiaUrl=<http://10.40.11.24:8181/jolokia>

For production, use.25 or .26 or .27, it is the same

The parameter <upload>false</upload> is defined as a plugin property. This parameter tells not to deploy the binary artifact to the Fuse internal Maven repository. At runtime, Fuse will retrieve the artifact by looping through all the maven repositories configured in the Fabric agent, and should finally find it in Nexus.

To be able to deploy directly into the internal Fuse repository, an appropriate <server> section also must be defined in the maven settings.xml:

<server>

<id>fabric8.upload.repo</id>

<username>admin</username>

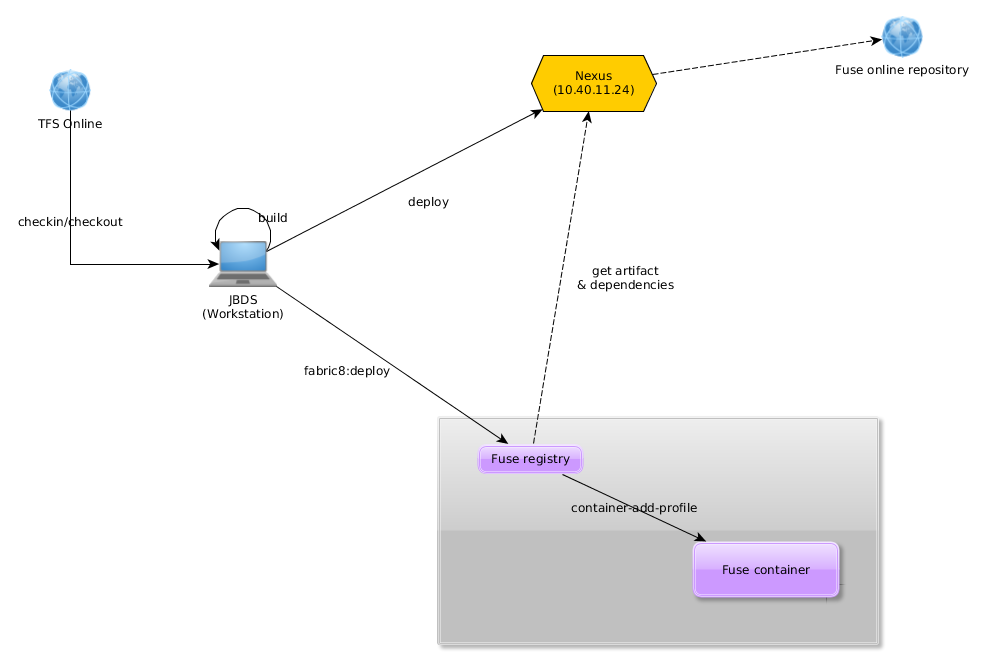
<password>admin</password>

</server>

* 1. Bundle development

Another important maven plugin is the maven-bundle-plugin from the Apache Felix project. This plugin allows the maven <packaging> to be changed to <bundle>. This packaging type allows maven to generate OSGI-compatible artifacts.

* 1. Deployment Diagram



## Release management diagram

