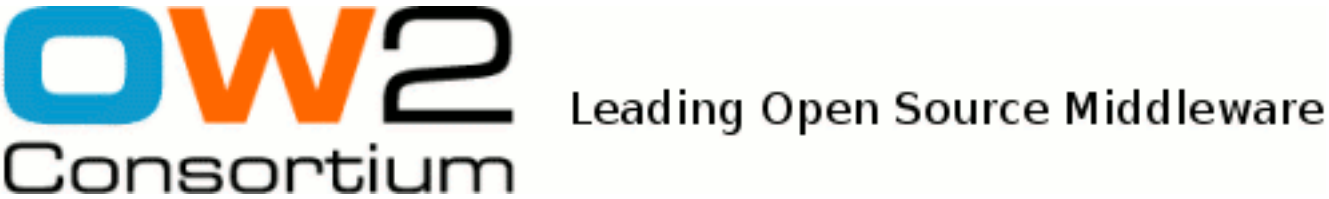

JOnAS Team (Mickaël Leduque, François Fornaciari)

- October 2008 -

Table of Contents

1. Deployment plans - Introduction 1
2. Repositories 2
    2.1. Creating the repository list 2
    2.2. Using the repository lists 2
    2.3. Accessing repositories through a proxy 3
3. Deployment plans 4
    3.1. Deployment type 4
    3.2. Information that can be provided as element or attribute 4
    3.3. Deployment common parts 5
    3.4. Deployment type specific documentation 5
        3.4.1. URL deployments 5
        3.4.2. Maven 2 deployments 6
        3.4.3. OBR deployments 6
    3.5. Mixing deployments types 7
4. Deployment plans in JOnAS 5 8
    4.1. Initial repositories 8
    4.2. The resourcemonitor service 8

List of Examples

2.1. Example of a repository file with three repositories. 2
3.1. example: namespace declaration 4
3.2. example: Maven2 deployment type 4
3.3. id as attribute 4
3.4. id as element 4
3.5. Deployment plan with URL deployments 6
3.6. Deployment plan with Maven2 deployments 6
3.7. Deployment plan with OBR deployments 7
3.8. Deployment plan with different deployment types 7
4.1. jonas.service with resourcemonitor service active 8

Chapter 1. Deployment plans - Introduction

A deployment plan is an XML file that describes a succession of resources to be deployed in the given order. The deployment plan only contains references to the deployable resources like for example, Java EE archives or an OSGi bundles. In order to deploy the resources on a server, the deployment plan has to be deployed on that server. The server will retrieve each resource from a repository before deploying it. The resources in a deployment plan may be stored in one or more repositories.

To deploy a deployment plan, one has to set up the repositories on the target server. The deployment plans will then be able to refer to these repositories.

The deployment plan functionnality supports currently three kinds of resources (both local or remote):

1. URL resources which point to valid URL locations.
2. Maven 2 resources which enable to deploy versioned resources.
3. OBR (OSGi Bundles Repository) resources which allow to benefit of the OBR resolver to deploy a requested OSGi bundle and its dependencies.

Note

The OBR deployment feature is currently partially supported as it only deploys specified OSGi bundle resources without their dependencies. The complete feature implementation is on-going.

Chapter 2. Repositories

The server maintains a list of repositories that can store the resources to be deployed using deployment plans.

2.1. Creating the repository list

The repositories list is an XML document with syntax described in the XML Schema Definition [http://jonas.ow2.org/ns/deployment-plan/repositories-1.0.xsd].

An example of a repository list file can be seen here.

The root node is `repositories` and must specify the following namespace: *http://jonas.ow2.org/ns/deployment-plan/repositories/1.0*.

Each child of the `repositories` node is a `repository` node and describes one repository. The `repository` nodes have a mandatory `id` attribute that identifies globally (i.e. at the server level) the repository.

The `repository` node then has two children nodes :

- `type` that specifies how the server can communicate with the repository (for example maven2, obr, url);
- `url` that gives the repository location.

Example 2.1. Example of a repository file with three repositories.

```
<?xml version="1.0" encoding="UTF-8"?>
<repositories
  xmlns="http://jonas.ow2.org/ns/deployment-plan/repositories/1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="repositories-1.0.xsd">

   <repository id="maven-repository">
     <type>maven2</type>
     <url>http://repo1.maven.org/maven2/</url>
   </repository>

   <repository id="http-repo">
     <type>url</type>
     <url>http://localhost/</url>
   </repository>

   <repository id="obr-repo">
     <type>obr</type>
     <url>file:///C:/obr/</url>
   </repository>

</repositories>
```

2.2. Using the repository lists

The repository lists can be deployed on a server in the same way as usual deployable resources (EAR, EJB-JAR, WAR...). This has to be done before deploying the deployment plan.

Note

In Easybeans, repositories lists can be deployed by copying the file into the easybeans-deploy directory like any other resource.

In JOnAS, there are two ways to manage the repository lists.

- You can deploy a repository list like any other resource, either by copying it into the deploy directory (if the development mode is active), or using Admin tools (JonasAdmin web interface or **`jonas admin -a filename`** command).
- You can use it as a configuration file, named `conf/initial-repositories.xml`.

2.3. Accessing repositories through a proxy

Proxy can be specified by setting the Java properties http.proxyHost and http.proxyPort. These properties are described in the Java SE documentation [http://java.sun.com/javase/6/docs/technotes/guides/net/properties.html].

Note

These properties have to be set even if only some Maven repositories are used: the global Maven configuration is ignored.

Chapter 3. Deployment plans

A deployment plan is a XML file where the children of the root node are *deployment* elements representing resources that will be deployed in the order in which they appear.

The exact syntax of the `deployment` node in the deployment plan depends on the *type* of the resource (maven2, obr, url...), but there are some parts that are common to all the deployment elements.

3.1. Deployment type

`deployment` is an abstract XML type, and concrete type must be declared. Concrete types are for example URL, Maven2 or OBR. Each deployment type is defined in its own XML Schema Description file (XSD file). In order to use one type, it is necessary to declare a namespace for it. Here is an example of namespace declaration :

Example 3.1. example: namespace declaration

```
<deployment-plan  xmlns="http://jonas.ow2.org/ns/deployment-plan/1.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:m2="http://jonas.ow2.org/ns/deployment-plan/maven2/1.0">
```

This a deployment plan contains a namespace for the Maven2 deployment type using the `m2:` prefix. We can then add a Maven2 deployment :

Example 3.2. example: Maven2 deployment type

```
<deployment-plan  xmlns="http://jonas.ow2.org/ns/deployment-plan/1.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:m2="http://jonas.ow2.org/ns/deployment-plan/maven2/1.0">
  <deployment xsi:type="m2:maven2-deploymentType ...
```

The information provided in each deployment element is divided in two parts:

- data that is specific to a deployment type (URL, Maven2, OBR)
- data that can be used in any deployment.

See the deployment type specific pages for the currently supported types : URL, Maven2 and OBR

3.2. Information that can be provided as element or attribute

Some information can be specified either as XML element or XML attribute. For example:

Example 3.3. id as attribute

```
<deployment id="deployment-1">
   ...
</deployment>
```

is the same as

Example 3.4. id as element

```
<deployment>
  <id>deployment-1</id>
  ...
</deployment>
```

It is the case for :

- `id`
- `repository-ref`
- `reloadable`
- `start`
- `reference`
- `startlevel`
- `starttransient`

3.3. Deployment common parts

The common part of the deployment syntax is described in the XML Schema Definition [http://jonas.ow2.org/ns/deployment-plan/deployment-plan-1.0.xsd].

Those are the information pieces that are common to all deployment types:

- `id`: this information is mandatory and gives a identifier to the deployment plan.
- `repository-ref`: this information is optional. The value is the id of a repository known by the server. When given, the server will only search the resource on this repository.
- `reloadable`: this information is a boolean (true or false), is optional and its default value is true. When set to true, after the resource is correctly deployed, the server will check the resource on the repository periodically. If the resources changes, the server will redeploy it.
- `start`: this information is a boolean (true or false), is optional and its default value is true. When set to true, if the resource to deploy is an OSGi bundle, the server will install and start the bundle on the framework. When set to false, the server will only install the bundle on the framework.
- `reference`: this information is a boolean (true or false), is optional and its default value is false. When set to true, if the resource to deploy is an OSGi bundle, the server will install the OSGi bundle as a referenced JAR file, i.e. the JAR of the bundle (and its embedded JARs) will not be copied in the cache of the OSGi framework. When set to false, the server will install the bundle on the framework in the standard way.
- `startlevel`: this information is a number greater than or equal to 1, is optional and its default value is 1. When set to true, if the resource to deploy is an OSGi bundle, the server will install the OSGi bundle with the given start level. When the OSGi framework is launched, the framework will enter start level one and all bundles which are assigned to start level one are started. The framework will continue to increase the start level, starting bundles at each start level, until the framework has reached a beginning start level.
- `starttransient`: this information is a boolean (true or false), is optional and its default value is true. When set to true, if the resource to deploy is an OSGi bundle, the bundle start operation is transient, which means that the persistent autostart setting of the bundle is not modified.

3.4. Deployment type specific documentation

3.4.1. URL deployments

3.4.1.1. Specific data

The only URL deployment specific data is the `resource` that can only be specified as a *XML element*.

- *url:resource* (required) : the file name of the resource.

3.4.1.2. Full URL deployment plan

There are few possible variations in the URL deployment. Here is a complete deployment plan with a URL deployment.

Example 3.5. Deployment plan with URL deployments

```
<?xml version="1.0" encoding="UTF-8"?>
  <deployment-plan xmlns="http://jonas.ow2.org/ns/deployment-plan/1.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:url="http://jonas.ow2.org/ns/deployment-plan/url/1.0">

   <deployment xsi:type="url:url-deploymentType"  id="dep1" reloadable="true">
    <url:resource>easybeans-example-statelessbean-1.0.1-SNAPSHOT.jar</url:resource>
   </deployment>

  </deployment-plan>
```

3.4.2. Maven 2 deployments

3.4.2.1. Maven2 specific data

The Maven 2 deployments specific data can only be specified as *XML elements*. Here are those elements :

- *m2:groupId* (required) : the maven group id of the artifact.
- *m2:artifactId* (required) : the maven artifact id of the artifact.
- *m2:version* (required) : the version of the artifact.
- *m2:type* (optional with default value: jar) : the maven type of the artifact (jar, war, ear, ...).
- *m2:classifier* (optional with empty default value) : the maven classifier.

3.4.2.2. Full Maven2 deployment plan example

The following deployment plan contains two Maven2 deployments. The first one doesn't specify the resource type, so it is assumed to be `jar`. The second one has a `ear` type.

Example 3.6. Deployment plan with Maven2 deployments

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment-plan xmlns="http://jonas.ow2.org/ns/deployment-plan/1.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:m2="http://jonas.ow2.org/ns/deployment-plan/maven2/1.0">

  <deployment xsi:type="m2:maven2-deploymentType" id="dep2">
    <m2:groupId>org.ow2.easybeans</m2:groupId>
    <m2:artifactId>easybeans-example-statefulbean</m2:artifactId>
    <m2:version>1.1.0-SNAPSHOT</m2:version>
  </deployment>

  <deployment xsi:type="m2:maven2-deploymentType" id="dep3">
    <m2:groupId>org.ow2.easybeans</m2:groupId>
    <m2:artifactId>example-server</m2:artifactId>
    <m2:version>1.1.0-SNAPSHOT</m2:version>
    <m2:type>ear</m2:type>
  </deployment>

</deployment-plan>
```

3.4.3. OBR deployments

3.4.3.1. OBR specific data

Many notions used here are defined or described in the OSGi RFC 112 Bundle Repository [http://www.osgi.org/download/rfc-0112_BundleRepository.pdf]. The OBR deployment gives many ways

to define a resource. You can use as many descriptions as you want, only one search will be issued, built as the conjunction of these descriptions. The result will be considered valid only if there is *one and only one* matching resource. The OBR specific data can only be specified as *XML elements*.

Here are the possible descriptions:

- *obr:bundle-symbolic-name* (optional) : the OBR symbolic name of the resource.
- *obr:bundle-version* (optional) : the OBR version of the resource.
- *obr:filter* (optional) : a OSGi/LDAP filter (see the RFC) that describes the resource.
- *obr:require-service* (optional) : the resource must provide the required service.

3.4.3.2. Full OBR deployment example

The following deployment plan contains two OBR deployments. Both require a resource by `symbolic-name` + `version`, but the first uses the dedicated XML elements while the second uses a *filter*.

Example 3.7. Deployment plan with OBR deployments

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment-plan xmlns="http://jonas.ow2.org/ns/deployment-plan/1.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:obr="http://jonas.ow2.org/ns/deployment-plan/obr/1.0">
  <deployment xsi:type="obr:obr-deploymentType"  id="dep-obr-1">
     <obr:bundle-symbolic-name>org.ow2.easybeans.examples.entitybean</obr:bundle-symbolic-name>
     <obr:bundle-version>=1.1.0-SNAPSHOT<obr:bundle-version>
  </deployment>

  <deployment xsi:type="obr:obr-deploymentType"  id="dep-obr-2">
     <obr:filter>(&(symbolicname=org.ow2.easybeans.examples.statefullbean)(version=1.1.0-SNAPSHOT))</obr:filter>
  </deployment>
</deployment-plan>
```

3.5. Mixing deployments types

It is possible to use deployments of different types in the same deployment plan. To achieve that, one just has to declare all the required namespaces.

The following example mixes URL and Maven2 deployments in the same deployment plan

Example 3.8. Deployment plan with different deployment types

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment-plan xmlns="http://jonas.ow2.org/ns/deployment-plan/1.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:url="http://jonas.ow2.org/ns/deployment-plan/url/1.0"
        xmlns:m2="http://jonas.ow2.org/ns/deployment-plan/maven2/1.0">

  <deployment xsi:type="url:url-deploymentType"  id="dep1">
    <url:resource>easybeans-example-statefulbean-1.0.1-SNAPSHOT.jar</url:resource>
    <repository-ref>home-repository</repository-ref>
  </deployment>

  <deployment xsi:type="m2:maven2-deploymentType" id="dep3">
    <m2:groupId>org.ow2.easybeans</m2:groupId>
    <m2:artifactId>example-server</m2:artifactId>
    <m2:version>1.1.0-SNAPSHOT</m2:version>
    <m2:type>ear</m2:type>
    <reloadable>true</reloadable>
  </deployment>

</deployment-plan>
```

Chapter 4. Deployment plans in JOnAS 5

4.1. Initial repositories

The `conf/initial-repositories.xml` file allows to add repositories to the list of availaible repositories at JOnAS server startup.

This file follows the same syntax as the repository list XML files.

4.2. The resourcemonitor service

There is one service that is related to deployment plans in JOnAS 5.

- the resource monitoring service, named resourcemonitor in the `conf/jonas.properties` file.

You can activate this service by adding it to the jonas.services list in `conf/jonas.properties`. For example:

Example 4.1. jonas.service with resourcemonitor service active

```
jonas.services  registry,jmx,jtm,db,security,wm,wc,resource,ejb2,ejb3,ws,web,ear,depmonitor,resourcemonitor
```

will activate the resource monitoring service.

This service is only useful if deployment plans with `reloadable` deployments are deployed.

The `conf/jonas.properties` file permits to configure the following options:

- jonas.service.resourcemonitor.monitorInterval: the time in milliseconds between two resource checks. If the option is not provided, this has a default value of 30 000 (30 seconds)