

Deployit Cookbook - Using tags to control deployments

Version 3.8.0

Table of Contents

Table of Contents	2
Cookbook - Using tags to control deployments	3
Setup	3
Deploying without tags	3
Using tags	4
Changing the Manifest and Maven POM	5

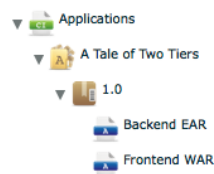
Cookbook - Using tags to control deployments

Deployit's AutoMap function is a powerful way to do one-click deployment mappings. But sometimes, for example during an automated Maven deployment, you need more fine-grained control over which deployable goes into what container. You can use *tags* to indicate what should go where.

This Cookbook describes how to map the components of a DAR file to different servers. We will have a frontend War that is run on Tomcat and a backend Ear that needs a JEE server. Using tags, this can be done in an easy and intuitive way.

Setup

In our example, we will use the fictional application "A Tale of Two Tiers". It contains two artifacts: 'Backend EAR' and 'Frontend WAR':

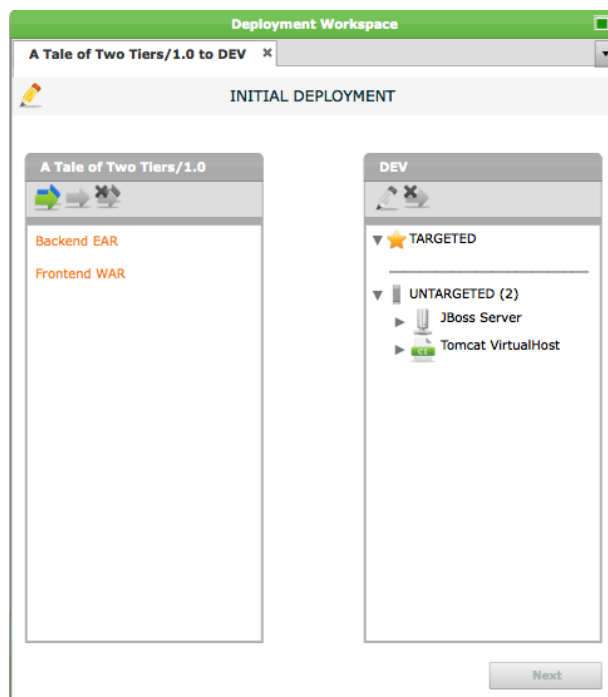


The WAR is a frontend application that will need to be deployed to Tomcat. The backend EAR is a full-fledged JEE application that will need JBoss to run.

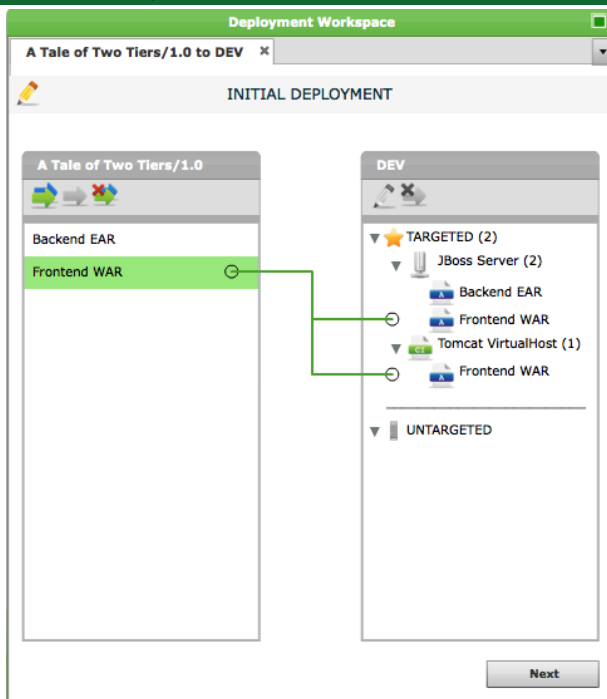
We have an example environment called DEV, that contains **JBoss Server** and **Tomcat VirtualHost**

Deploying without tags

We start a deployment in the UI by dragging the 1.0 version of "A Tale of Two Tiers" to the Package space and the DEV environment to the Environment space.



We press the "auto map" button (green and blue arrow) to create a deployment mapping. But it doesn't do what we want: the Frontend EAR is mapped to both Tomcat *and* JBoss, since JBoss is perfectly capable of running WAR files as well.

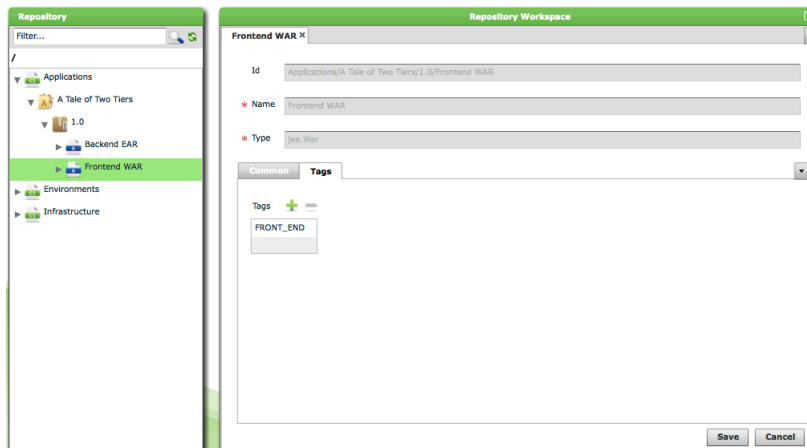


The link from Frontend WAR to JBoss Server has to be removed manually. You can do this by clicking on "Frontend WAR" under "JBoss server" and clicking the green arrow with the red cross. But this is inconvenient. If there are more servers, the amount of manual tweaking will get out of hand pretty quickly. Moreover, automated deployments from Maven will not be able to do this and will fail.

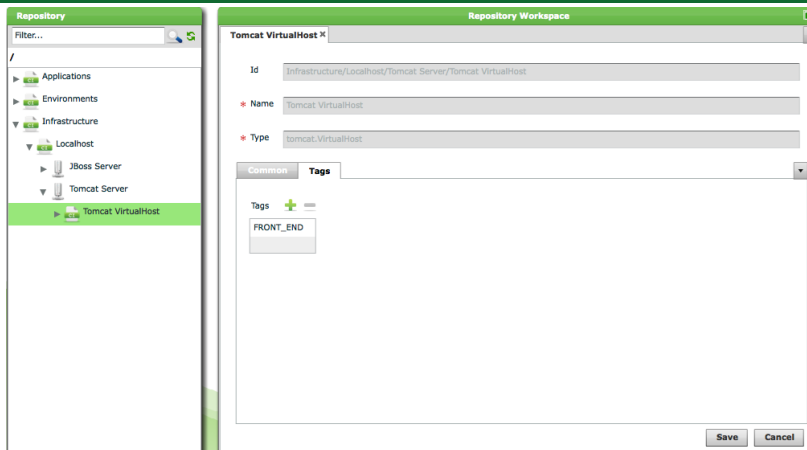
Using tags

The solution here is to use **tags**. Tags are simple labels that you can attach to any deployable (for example a WAR file) and to any container (say, the Tomcat server). If tags are present, the mapping algorithm will only create a deployment mapping if both deployable and container share a certain tags. Note that an item may contain multiple tags.

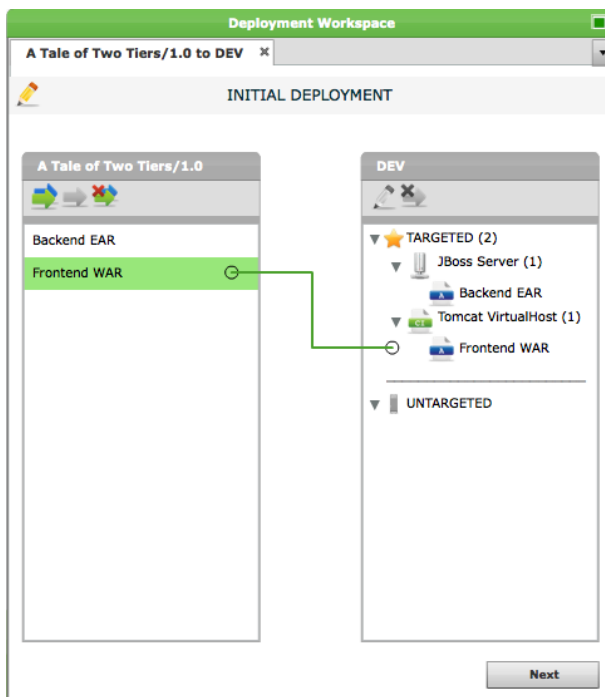
Let's see how this works out in practice. In the repository browser, we open the **Frontend WAR** and add a tag **FRONT_END** in the Tags tab.



Do the same for the **Tomcat VirtualHost**.



If we do a deployment mapping now, we will see that the Frontend WAR file will be no longer mapped to JBoss, because the JBoss server doesn't have the FRONT_END tag.



Changing the Manifest and Maven POM

This is all fine, but we're still left with a manual step: adding the tags in the repository browser. This is a great way to test if tags work as expected, but for real deployments you want a more structural solution. We need to make sure that the tags are already in the DAR file when we import the application version.

To do so, we have to add the tags to the `MANIFEST.MF` file. Luckily, that is quite easy. Tags are modeled as a property of type Set of Strings. So here's an example of the war entry in the manifest file, with tags added.

```
Name: Frontend-1.0.war
CI-Name: Frontend WAR
CI-Type: jee.War
CI-Tags-EntryValue-1: FRONT_END
CI-Tags-EntryValue-2: LONDON
```

Using the Deployit Maven plugin, the correct snippet would be:

```
...  
<deployable>  
  <name>Frontend-1.0.war</name>  
  <type>jee.War</type>  
  <groupId>com.xebialabs.deployit.cookbook</groupId>  
  <artifactId>Frontend</artifactId>  
  <tags>  
    <tag>FRONT_END</tag>  
    <tag>LONDON</tag>  
  </tags>  
</deployable>  
...
```

A full example can be found in the [Maven Plugin documentation](#).