Understanding JBoss With Magnolia

- Document Overview
- Background Information on JBoss
- 1 - Determining Which JBoss Version To Use
- 2 - Determining Which Java Version To Use
- 3 - Verifying With Magnolia's Certified Stack
- 4 - Downloading JBoss
    - Versions Below JBoss EAP 7.0 / JBoss AS
    - Versions With JBoss EAP 7.0 and Above
- 5 - Adding Magnolia to JBoss
- 6 - Configuring Magnolia With JBoss
- 7 - Deploying Your Magnolia Instance
- 8 - Starting The Server
- 9 - Accessing The Page
- 10 - Stopping The Server
- Additional Information
    - More Information on the jboss-deployment-structure.xml
        - <jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
        - <exclude-subsystems>
        - <exclusions>
    - Databases
    - More Information on the modules.xml

Document Overview

I created this page to provide an easy way of understanding how JBoss works and how to get it working with Magnolia. This page is not a comprehensive analysis, but will continue to grow as I learn more about it. I will provide some helpful information on how JBoss is structured and what files and folders are relevant for our purposes.

If you would like to see these steps without the details, please see the Quickstart Guide here instead : Magnolia Quickstart With JBoss.

Since we mainly use Tomcat, I will try to compare some JBoss concepts and terms to some Tomcat or Magnolia terms so that you can have a reference as to how a certain file or function works.

Background Information on JBoss

It was very confusing for me to understand what JBoss, Redhat and all those names meant. So I'd like to start by defining these to clear up any confusion.

JBoss was the name of the company that created the product "**JBoss**". Later, **RedHat** bought the company but still had the **JBoss** product and maintained its name. With time, the **JBoss** product was renamed to **Wildfly** which is still the same product, but with a different name.

JBoss EAP (Enterprise Application Platform) is the enterprise version of **JBoss**. Saying **JBoss EAP** is like saying **Magnolia EE** or **EEPro**. Similarly, saying **J Boss AS** or **Wildfly** is like referring to the **Magnolia Community version**. The community counterpart for **JBoss EAP 7.0** and above is **Wildfly**. The community counterpart for versions of **JBoss EAP** before **7.0** is **JBoss AS**.

JBoss AS is also the development name of JBoss - it is the core, similar to saying "**magnolia-core**".

1 - Determining Which JBoss Version To Use

For testing purposes, we will be using Wildfly because EAP requires a subscription. In general, we will mostly be using Wildfly 11 and above. To see the correlation between EAP versions and Wildfly, see the chart here:

JBoss EAP Version	WildFly Version
JBoss EAP 6.0	JBoss AS 7.1
JBoss EAP 6.1	JBoss AS 7.2
JBoss EAP 6.2	JBoss AS 7.3
JBoss EAP 6.3	JBoss AS 7.4

JBoss EAP 6.4	JBoss AS 7.5
JBoss EAP 7.0	WildFly 10
JBoss EAP 7.1	WildFly 11
JBoss EAP 7.2	WildFly 14
JBoss EAP 7.3	WildFly 18

Here is an example:

If someone is using **JBoss EAP 7.0** to deploy **Magnolia**, you will need to use **Wildfly 10** as its counterpart.[(1)]

2 - Determining Which Java Version To Use

WildFly 10 and above requires **Java SE 8**. It will not run with **Java SE 6**, **Java SE 7** or **9**.[(2)(3)]

3 - Verifying With Magnolia's Certified Stack

You then need to verify with Magnolia's certified stack to ensure that the Java version is also compatible with the Magnolia version you wish to use: Magnolia Certified Stack.

Continuing the example:

I'd like to use **Magnolia 6.2** with **Wildfly 10**. I see that the certified stack does *not* include the usage of **Wildfly 10** with **Magnolia 6.2**. I then have to change my JBoss version to be using **Wildfly 11**.

4 - Downloading JBoss

Once you've determined what version you need, you then need to use one of the following links:

Versions Below JBoss EAP 7.0 / JBoss AS

If you are using a version below **JBoss EAP 7.0**, then **JBoss AS** is its community counterpart. You would then use this link: https://jbossas.jboss.org/downloads.

Download the ZIP file with the description saying "**Java EE Full & Web Distribution**".

Versions With JBoss EAP 7.0 and Above

If you are using **JBoss EAP 7.0** or above (**Wildfly 10** is its community counterpart)

Use this link: https://wildfly.org/downloads/.

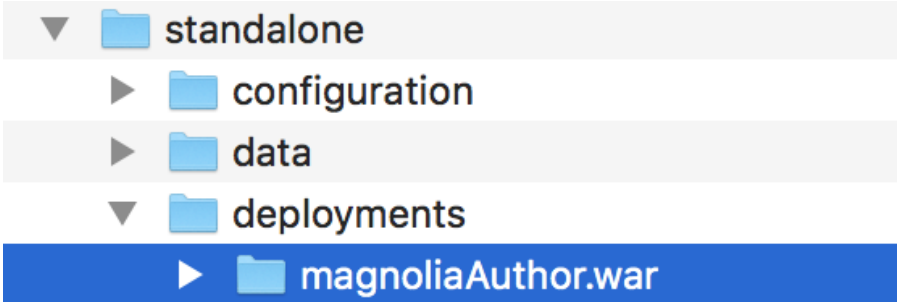
You want to download the ZIP file with the description saying "**Java EE Full & Web Distribution**"

Note
If you search for JBoss on a browser, you may bump into this link : https://developers.redhat.com/products/eap/download. However this is for the enterprise version, which we will not be using.

5 - Adding Magnolia to JBoss

You have two ways to deploy Jboss. One is with a domain and the other is with standalone configuration. I will only be covering how to use the standalone configuration.

- Unzip the file you downloaded and create a new folder within the "**standalone/deployments**" folder. Name this folder "**<name_of_Magnolia_instance>.war**". This is a regular folder and it will not be a WAR file, we will just be having ".war" as part of the name.



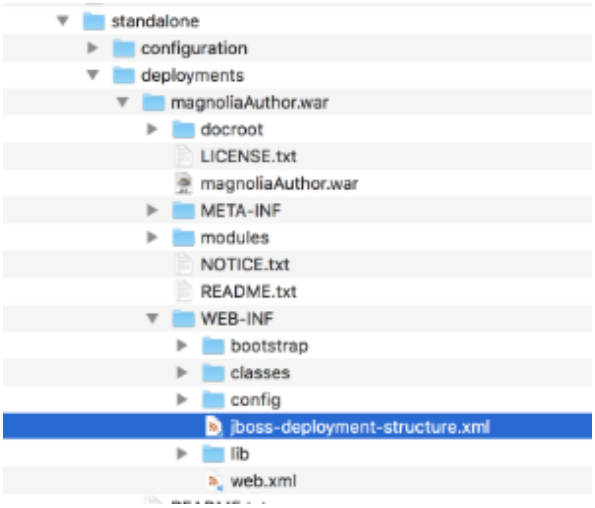
- Drop the Magnolia WAR file you have into this newly created folder.
- Explode the added WAR file : How to Explode a WAR

6 - Configuring Magnolia With JBoss

JBoss comes with many Java libraries for you to use that can be found within the "**modules**" folder of your downloaded JBoss. Magnolia also comes with many Java libraries, and these can be found within the "**WEB-INF/lib**" folder. As a result, some Java libraries declared in Magnolia are being repeated by the included libraries that already come with JBoss. Other deployments in JBoss can interfere with Magnolia. We must exclude certain repeated libraries to prevent library conflicts from occurring. We do this by using the

jboss-deployment-structure.xml file.

- Within **standalone/deployments/<name_of_Magnolia_instance>.war/WEB-INF** folder, create a new file and name it "**jboss-deployment-structure.xml**". This is the file we will be using to tell JBoss which dependencies to exclude from their built in list of Java libraries.



- Paste the following code into this newly created file:

Jboss-deployment-structure.xml

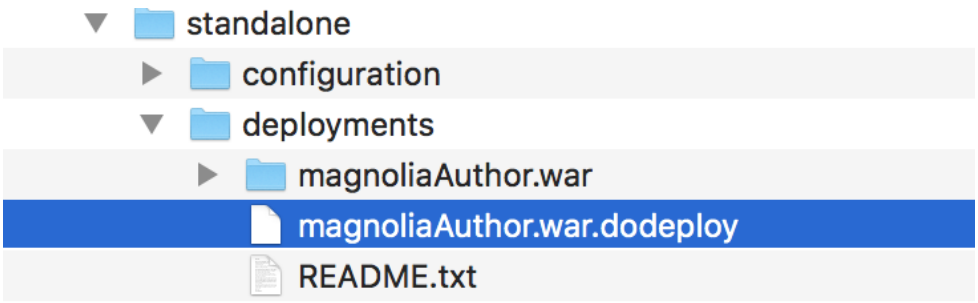
```
<?xml version="1.0"?>
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
    <deployment>
        <exclude-subsystems>
            <subsystem name="weld" />
            <subsystem name="jaxrs" />
            <subsystem name="webservices" />
        </exclude-subsystems>
        <exclusions>
            <module name="org.apache.log4j" />
            <module name="org.slf4j" />
            <module name="org.bouncycastle" />
        </exclusions>
    </deployment>
</jboss-deployment-structure>
```

- Open your **magnolia.properties** file and add the following line : **`magnolia.definitions.classpath=info.magnolia`**

7 - Deploying Your Magnolia Instance

Now that we have all of the configurations set up, all that is left to do is to deploy Magnolia. Normally, JBoss will check to see if any WAR file is present within the **standalone/deployments** folder. Since we've exploded the war and placed it in a folder, it does not detect that there is a WAR to deploy. We now have to tell JBoss to deploy the contents of an exploded folder.

To do this, you create a new file within **standalone/deployments** and name it "**<name_of_folder_with_exploded_WAR>.dodeploy**". As an example, if you named the folder **magnoliaAuthor.war**, the name of this new file will be "**magnoliaAuthor.war.dodeploy**". No other changes will be made to this file, it is left blank.



Every deployment you will notice that this file will change to have the suffix **.deployed** of **.failed**. To redeploy, you would just rename any of these back to .**dodeploy**. These are considered "**marker**" files and are used by JBoss as instructions for deployments or to provide statuses. If interested, you can click on the **README.txt** found in the **deployments** folder for more information on what those suffixes mean.

Note

In the case of having an actual WAR, file instead of an exploded one, JBoss will detect that there is something to deploy. If that is the case, then the .**dodeploy** file will *not* be necessary.

8 - Starting The Server

Open a console window and change directory to "**<JBoss_folder>/bin**". And execute the following command: **./standalone.sh**

9 - Accessing The Page

The page name is the name of the folder you created. Similar to Tomcat, in the above example, the location will be: **localhost:8080/magnoliaAuthor**

10 - Stopping The Server

You can shut off the server by doing **ctrl+c** on the console that you started the server with.

Additional Information

More Information on the jboss-deployment-structure.xml

Jboss-deployment-structure.xml

```
<?xml version="1.0"?>
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
    <deployment>
        <exclude-subsystems>
            <subsystem name="weld" />
            <subsystem name="jaxrs" />
            <subsystem name="webservices" />
        </exclude-subsystems>
        <exclusions>
            <module name="org.apache.log4j" />
            <module name="org.slf4j" />
            <module name="org.bouncycastle" />
        </exclusions>
    </deployment>
</jboss-deployment-structure>
```

This is the same file used earlier.

<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">	This element is telling the xml file what version of the element structure to use. For our purposes, you can see this line as the standard when creating **jboss-deployment-structure.xml** files. **"urn:jboss:deployment-structure:1.2"** This is referring to a version that is found here: https://github.com/jboss-modules/jboss-modules/tree/1.x/src/main/resources/schema This element is saying that you will be using the structure found within the file definition declared in the link provided. No additional action is required here
<exclude-subsystems>	Subsystem are like groups of modules. They split the application into smaller independent functionalities, which is why you can exclude them. These live within the path: **system/layers/base/org/jboss/as**
<exclusions>	Within this element segment, you will list the modules you wish to exclude from the given JBoss Java libraries. We are doing this because they are already found within Magnolia. This is the format you will use: **<module name = "name_of_module">** With the above code you will see that the exclusions refer to the following folder paths: **modules/system/layers/base/org/apache/log4j** **modules/system/layers/base/org/slf4j** **modules/system/layers/base/org/bouncycastle**

Databases

Magnolia takes care of getting your databases set up. You would do this via the various jackrabbit-bundle files located within the **WEB-INF/config/repo-conf** folder.

More Information on the modules.xml

We do not need to edit this file, but understanding what it does can help further understand what we are doing. We can compare this to a Magnolia module's POM file. Let's open one to see how it looks like. Try going to this file path so that the below explanation can make more sense. In this example I will use the file located within **modules/system/layers/base/org/apache/lucene/main**.

modules/system/layers/base/org/apache/lucene/main/modules.xml

```
<module xmlns="urn:jboss:module:1.3" name="org.apache.lucene">
   <properties>
       <property name="jboss.api" value="private"/>
   </properties>

   <resources>
       <resource-root path="lucene-analyzers-common-5.3.1.jar"/>
       <resource-root path="lucene-core-5.3.1.jar"/>
       <resource-root path="lucene-facet-5.3.1.jar"/>
       <resource-root path="lucene-queries-5.3.1.jar"/>
       <resource-root path="lucene-queryparser-5.3.1.jar"/>
   </resources>

   <dependencies>
       <module name="javax.api"/>
       <module name="org.apache.lucene.internal" optional="true" services="import"/>
   </dependencies>
</module>
```

<resources>	The elements here are referring to individual JAR files located within the **modules/system/layers/base/org/apache/lucene/main** folder. JBoss will read this file to understand that these are the JARs which the **modules/system/layers/base/org/apache/lucene/main** module is using.
<dependencies>	references to modules that have paths to locations within **modules/system/layers/base.**