Working with JBoss application Server 7.1

1. JBoss AS 7.x is an java based application server compliant to JavaEE 6 specification.
2. It supports all the technologies as mentioned in the JavaEE 6 specification document
3. Major Technologies being
   1. WebModule 3.0
      1. Servlets 3.0
      2. JSP 2.2
   2. JSF 2.1
   3. EJB 3.1
   4. JAX-WS 2.1
   5. RESTful WebServices (JAX-RS)
   6. CDI

**JBoss setup**

1. Download and install JDK
   1. To do details of platform specific JDK Configuration
2. Download and install JBoss AS 7
   1. To to details of JBoss download and install
3. Start and stop JBoss
4. Command line Administration interface
5. Start up JBoss in standalone mode
6. Start up JBoss in domain mode
7. Discuss JBoss AS 7 architecture
8. Discuss directory structure of JBoss install

**Start JBoss 7.1 in domain mode:**

1. Unzip JBoss 7 in two different locations
2. Name one of them as Master and other as slave
3. In my case I installed one in /home/training/jboss-as-7.1.1.Final and the other as /home/training/jboss-as-7.1.1.Final.slave
4. We call the first one as Master and the second as slave
5. The key point here is, when you start domain controller in Master configuration, you must have a management user in master domain who also will need to be mentioned in the remote host file with the encrypted password. Now as of Jboss7.1 the remote host name must match the management user name in master domain.

***Host name in slave config=management user in domain controller***

***If you are running the domain controller and slave in the same computer the do take care of the ports***

***Ports can be changed by modifying offset values in server configuration in your host.xml file***

<servers>

<server name="server-one" group="main-server-group">

<!-- Remote JPDA debugging for a specific server

<jvm name="default">

<jvm-options>

<option value="-Xrunjdwp:transport=dt\_socket,address=8787,server=y,suspend=n"/>

</jvm-options>

</jvm>

-->

<socket-bindings port-offset="550"/>

</server>

<server name="server-two" group="main-server-group" auto-start="true">

<!-- server-two avoids port conflicts by incrementing the ports in

the default socket-group declared in the server-group -->

<socket-bindings port-offset="350"/>

</server>

<server name="server-three" group="other-server-group" auto-start="false">

<!-- server-three avoids port conflicts by incrementing the ports in

the default socket-group declared in the server-group -->

<socket-bindings port-offset="450"/>

</server>

**Setup:**

on windows 7 you must start cmdline as Administrator

make sure you have proper network connection

**1. Machine1:**

1. Create a management user named “demo”

You can use the command <JBOSS\_HOME>/bin/add-user.sh/cmd

2. Encrypt the password and keep it safe at some location.

3. start JBoss domain controller as:

./domain.sh –b<your binding ip> -bmanagement=<your management ip>

**Machine2:**

1.modify the <JBOSS\_HOME>/configuration/host.xml .

As your domain is not running locally, you need to include the details of remote domain controller here:

1. Open host.xml and change the host name to management user name existing the domain controller:

<host name="demo" xmlns="urn:jboss:domain:1.2">

1. Insert the following code in the host.xml file after <security-realm.....> (just below security-realm element):

<server-identities>

<!-- Replace this with either a base64 password of your own, or use a vault with a vault expression -->

<secret value="d2VsY29tZTE="/> <!—this is the encrypted pwd value-->

</server-identities>

1. Comment under <domain-controller> element

<local/> and uncomment <remote> modify its value as:

*<remote host="your ip for domain controller" port="9999" security-realm="ManagementRealm" />*

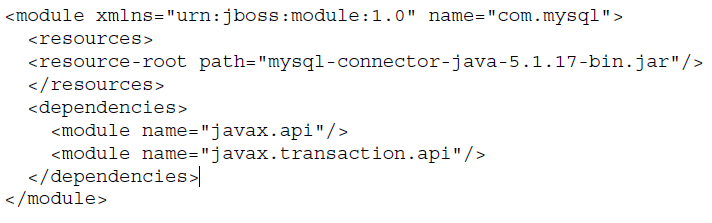
1. Change the port offset values in <server> section if required.
2. Start jboss as :

./domain.sh -b <binding IP for this machine> -Djboss.domain.master.address=<remote domain controller IP> -Djboss.domain.master.port=9999

1. **Create a DataSource**
   1. We will configure mysql datasource with JBoss 7.1
   2. We will add the driver as a module in JBoss Server
   3. Modules are added to jboss as <JBOSS\_HOME>/modules/YourModule/main
   4. Create a folder structure as <JBOSS\_HOME>/modules/com/mysql/main



* 1. Our module will be “com.mysql”
  2. Copy the driver file for mysql to <JBOSS\_HOME>/modules/com/mysql/main
  3. Create an xml file named “module.xml” in the same folder
  4. Add the following entries in the module.xml file



1. Now your module is ready to be used.

Modify your server’s config file “standalone.xml” or “domain.xml”

Modify the section <subsystem xmlns="urn:jboss:domain:datasources:1.0">

Insert the following xml after <datasources> element

<datasource jta="false" jndi-name="java:/MySqlDS" pool-name="mySqlDS" enabled="true" use-ccm="false">

<connection-url>jdbc:mysql://localhost:3306/demo</connection-url>

<driver-class>com.mysql.jdbc.Driver</driver-class>

<driver>mysql5</driver>

<security>

<user-name>root</user-name>

<password>root</password>

</security>

<validation>

<validate-on-match>false</validate-on-match>

<background-validation>false</background-validation>

</validation>

<statement>

<share-prepared-statements>false</share-prepared-statements>

</statement>

</datasource>

Add the following xml fragment within the <drivers></drivers>

<driver name="mysql5" module="com.mysql">

<driver-class>com.mysql.jdbc.Driver</driver-class>

</driver>

Recall that our module is being referenced here as com.mysql and we are providing a name as mysql5 to our driver in turn this name is being used by <datasource> element

Restart your Server.

**Creating a cluster in Standalone Mode**

1. **Cluster in the same Box**
   1. **Create 2 copies of standalone folder as**
      1. **Standalone-node1**
      2. **Standalone-node2**
2. **Start Jboss node1 as**

./standalone.sh -c standalone-ha.xml -b 10.10.10.10 -bmanagement 10.10.10.10 -u 230.0.0.4 -Djboss.server.base.dir=../standalone-node1 -Djboss.node.name=node1 -Djboss.socket.binding.port-offset=100

1. Start JBoss node2 as :
2. ./standalone.sh -c standalone-ha.xml -b 10.10.10.10 -u 230.0.0.4 -Djboss.server.base.dir=../standalone-node2 -Djboss.node.name=node2 -Djboss.socket.binding.port-offset=200
3. As JBoss7 cluster is on-demand, you will not see any indication that a cluster is formed. You will be able to the cluster when you deploy a clustered application.

**Configure mod\_cluster with the above cluster: (Mod cluster in JBoss.7.1.1 contains a bug** bug JBPAPP-7516 **and automatic advertise mode will not work and you need to provide proxy-list to Jboss node)**

1. **Setup at httpd side:**
   1. Download, install and configure your httpd server according to your platform
   2. Get the shared object library from mod cluster download site <http://www.jboss.org/mod_cluster>
   3. We are using mod cluster 1.2.0.Final version
   4. Copy the required .so files (as per the docs) to httpd/modules folder.
   5. Copy the following text at the bottom of the conf/httpd.conf file
   6. Our apache server is listening on 8888 port, please update IP address as per system IP.

############### mod\_cluster Setting - STARTED ###############

LoadModule slotmem\_module modules/mod\_slotmem.so

LoadModule manager\_module modules/mod\_manager.so

LoadModule proxy\_cluster\_module modules/mod\_proxy\_cluster.so

LoadModule advertise\_module modules/mod\_advertise.so

#Listen 1.1.1.1:80

<VirtualHost \*:8888>

<Directory />

Order deny,allow

Allow from all

</Directory>

<Location /mod\_cluster-manager>

SetHandler mod\_cluster-manager

Order deny,allow

Allow from all

</Location>

EnableMCPMReceive

KeepAliveTimeout 60

ManagerBalancerName mycluster

#AdvertiseGroup 224.0.1.105:23364

AdvertiseFrequency 2

ServerAdvertise On

</VirtualHost>

############### mod\_cluster Setting - ENDED ###############

* 1. Start apache server
  2. You are done!

1. **JBoss setup:** 
   1. Open <JBOSS\_HOME>/standalone-node1/standalone-ha.xml
   2. Update the <server> element with node name as standalone-node1

<server name="standalone-node1" xmlns="urn:jboss:domain:1.2">

1. Set proxy-list=”your apahce IP:apacheport” in modcluster subsystem as

<subsystem xmlns="urn:jboss:domain:modcluster:1.0">

<mod-cluster-config advertise-socket="modcluster" proxy-list="192.168.135.134:8888">

<dynamic-load-provider>

<load-metric type="busyness"/>

</dynamic-load-provider>

</mod-cluster-config>

</subsystem>

1. Update web subsystem with instance id as

<subsystem xmlns="urn:jboss:domain:web:1.1" default-virtual-server="default-host" instance-id="${jboss.node.name}" native="false">

Repeat the steps a-d for node2 also.

Restart Jboss nodes. Enjoy!

**JMS Configuration:**

1. By default Security is enabled in HornetQ and it raises an exception as Failed to create session : HornetQException[errorCode=105 message=Unable to validate user: **null**] if any JMS resource is accessed without proper security setup.
2. To correct this problem you need to add the following

<subsystem xmlns="urn:jboss:domain:messaging:1.1">

<hornetq-server>

<persistence-enabled>true</persistence-enabled>

<journal-file-size>102400</journal-file-size>

<journal-min-files>2</journal-min-files>

***<security-enabled>false</security-enabled>***

You may create a JMS Destination with an independent file as follows:

Just add this file to deployment folder

File name: test-jms.xml

<?xml version="1.0" encoding="UTF-8"?>

<messaging-deployment xmlns="urn:jboss:messaging-deployment:1.0">

<hornetq-server>

<jms-destinations>

<jms-queue name="QueueOne">

<entry name="java:jboss/exported/TestQueueOne"/>

<durable>true</durable>

</jms-queue>

<jms-topic name="TopicOne">

<entry name="java:jboss/exported/TestTopicOne"/>

</jms-topic>

</jms-destinations>

</hornetq-server>

</messaging-deployment>

**Configure SSL on Jboss 7:**

1. Open <JBOSS\_HOME><your config><config-file>
2. Navigate to the web subsystem configuration
3. In the connector section add

<connector name="https" protocol="HTTP/1.1" socket-binding="https" scheme="https" secure="true">

<ssl name="https" key-alias ="mykey" password="welcome1" certificate-key-file="C:\jboss-as-7.1.1.Final\ssl\mykeys.jks"/>

</connector>

Follow the xsd for the configuration for further information.

**Security Configuration**

Every security domain configuration is available in JBoss Server with the following JNDI name pattern:

**java:/jaas/<name of the domain in config file>**

e.g.

java:/jaas/mydomain

UserRoles Login Module (please configure properties files required)

Add the following entry in security-domains subsystem:

<security-domain name="mydomain" cache-type="default">

<authentication>

<login-module code="UsersRoles" flag="required">

<module-option name="usersProperties" value="${jboss.server.config.dir}/myusers.properties"/>

<module-option name="rolesProperties" value="${jboss.server.config.dir}/myroles.properties"/>

</login-module>

</authentication>

</security-domain>

Database Server Login Module:

<subsystem xmlns="urn:jboss:domain:security:1.0">

<security-domains>

   <security-domain name="turismo">

    <authentication>

          <login-module code="Database" flag="required">

               <module-option name="dsJndiName" value="java:turismo"/>

               <module-option name="principalsQuery" value="select password from Usuario where email=?"/>

               <module-option name="rolesQuery">

                 select ug.grupos\_groupName, 'Roles' from Usuario\_Grupo ug, Usuario u where u.id=ug.usuarios\_id and u.email=?

                </module-option>

               <module-option name="hashAlgorithm">SHA-256</module-option>

               <module-option name="hashEncoding">base64</module-option>

          </login-module>

     </authentication>

**</security-domain>**  
 **</security-domains>**  
**</subsystem>**

**Alternative of Database login Module:**

<security-domain name="mysqldomain" cache-type="default">

<authentication>

<login-module code="Database" flag="required">

<module-option name="dsJndiName" value="java:/datasources/MySqlDS"/>

<module-option name="principalsQuery" value="select password from USERS where username=?"/>

<module-option name="rolesQuery" value="select role, 'Roles' from ROLES where username=?"/>

</login-module>

</authentication>

</security-domain>

In order to get working with this configuration, you have first have to create the required tables and insert some sample data in it:

create table users(username varchar(32) primary key,password varchar(32));

create table roles(username varchar(32),role varchar(20));

insert into users(username,password) values('scott','scott123');

insert into users(username,password) values('pavan','pavan123');

insert into users(username,password) values('shantanu','shan123');

insert into roles(username,role) values('scott','manager');

insert into roles(username,role) values('pavan','user');

insert into roles(username,role) values('shantanu','manager');

LDAP Login Module:

<security-domain name="ldapdomain" cache-type="default">

<authentication>

<login-module code="LdapExtended" flag="required">

<module-option name="java.naming.factory.initial" value="com.sun. jndi.ldap.LdapCtxFactory"/>

<module-option name="java.naming.provider.url" value="ldap:// localhost:10389"/>

<module-option name="java.naming.security.authentication" value="simple"/>

<module-option name="bindDN" value="uid=admin,ou=system"/>

<module-option name="bindCredential" value="secret"/>

<module-option name="baseCtxDN" value="ou=People,dc=example,dc=com"/>

<module-option name="baseFilter" value="(uid={0})"/>

<module-option name="rolesCtxDN" value="ou=Roles,dc=example,dc=com"/>

<module-option name="roleFilter" value="(member={1})"/>

<module-option name="roleAttributeID" value="cn"/>

<module-option name="searchScope" value="ONELEVEL\_SCOPE"/>

<module-option name="allowEmptyPasswords" value="true"/>

</login-module>

</authentication>

</security-domain>

**JBoss-cli:**

start a server:

[domain@192.168.135.130:9999 /] /host=master/server-config=server-three:start

{

"outcome" => "success",

"result" => "STARTED"

}

stop a server:

[domain@192.168.135.130:9999 /] /host=master/server-config=server-three:stop

{

"outcome" => "success",

"result" => "STOPPING"

}

CLI Commands:

1. **Read System Properties**

[standalone@localhost:9999 /] /core-service=platform-mbean/type=runtime/:read-attribute(name=system-properties)

[standalone@IP\_ADDRESS:9999 /] /system-property=foo:add(value=bar)

[standalone@IP\_ADDRESS:9999 /] /system-property=foo:read-resource

{

    "outcome" => "success",

    "result" => {"value" => "bar"}

}

[standalone@IP\_ADDRESS:9999 /] /system-property=foo:remove

{"outcome" => "success"}

|  |
| --- |
| [domain@IP\_ADDRESS:9999 /] /system-property=foo:add(value=bar)  [domain@IP\_ADDRESS:9999 /] /system-property=foo:read-resource  [domain@IP\_ADDRESS:9999 /] /system-property=foo:remove |

**Host and its server instances**

|  |
| --- |
| [domain@IP\_ADDRESS:9999 /] /host=master/system-property=foo:add(value=bar)  [domain@IP\_ADDRESS:9999 /] /host=master/system-property=foo:read-resource  [domain@IP\_ADDRESS:9999 /] /host=master/system-property=foo:remove |

**Just one server instance**

|  |
| --- |
| [domain@IP\_ADDRESS:9999 /] /host=master/server-config=server-one/system-property=foo:add(value=bar)  [domain@IP\_ADDRESS:9999 /] /host=master/server-config=server-one/system-property=foo:read-resource  [domain@IP\_ADDRESS:9999 /] /host=master/server-config=server-one/system-property=foo:remove |

### Overview of all system properties

Overview of all system properties in JBoss AS7+ including OS system properties and properties specified on command line using -D, -P or --properties arguments.

**Standalone**

|  |
| --- |
| [standalone@IP\_ADDRESS:9999 /] /core-service=platform-mbean/type=runtime:read-attribute(name=system-properties) |

**Domain**

|  |
| --- |
| [domain@IP\_ADDRESS:9999 /] /host=master/core-service=platform-mbean/type=runtime:read-attribute(name=system-properties)  [domain@IP\_ADDRESS:9999 /] /host=master/server=server-one/core-service=platform-mbean/type=runtime:read-attribute(name=system-properties) |

### List Subsystems

### To list server extensions, profiles, subsystems, network interfaces, or socket-binding-groups: jboss-cli.sh -c "ls subsystem" jboss-cli.sh -c "ls extension" jboss-cli.sh -c "ls profile" jboss-cli.sh -c "ls interface" jboss-cli.sh -c "ls socket-binding-group"

### OR

[standalone@localhost:9999 /] /:read-children-names(child-type=subsystem)

### List description of available attributes and childs

### View configuration as XML for domain model or host model

Assume you have a host that is called master

|  |
| --- |
| [domain@localhost:9999 /] /host=master:read-config-as-xml |

Just for the domain or standalone

|  |
| --- |
| [domain@localhost:9999 /] :read-config-as-xml |

### How to get interface address

[standalone@localhost:9999 /] cd interface=public

[standalone@localhost:9999 interface=public] :read-resource(include-runtime=true)

[standalone@localhost:9999 interface=public] :read-attribute(name=resolved-address)

It's similar for domain, just specify path to server instance:

|  |
| --- |
| [domain@localhost:9999 /] /host=master/server=server-one/interface=public:read-attribute(name=resolved-address) |

**Deployment**

1. **:read-children-names(child-type=deployment) --🡪 lists all deployments**

## Standalone mode

First, we need to make sure there is a standalone AS7 instance is running and the CLI is connected to its controller, e.g.

Connected to standalone controller at localhost:9999

[standalone@localhost:9999 /]

Now to deploy an application, all that's necessary is to type in deploy and the path to the package (the tab-completion will help to navigate the filesystem), e.g.

[standalone@localhost:9999 /] deploy ../../../../testsuite/smoke/target/deployments/test-deployment.sar

'test-deployment.sar' deployed successfully.

You can make sure that the application is deployed by checking deployments in the standalone.xml

<deployments>

    <deployment name="test-deployment.sar" runtime-name="test-deployment.sar" sha1="af4edddaa426ccc367fed33cb67553ba21e0bc3d"/>

</deployments>

To undeploy this application its name has to be passed in as the argument to undeploy (if you type in undeploy, the tab-completion will list all the deployed applications and help complete the deployment name), e.g.

[standalone@localhost:9999 /] undeploy test-deployment.sar

Successfully undeployed test-deployment.sar.

Now if you check the standalone.xml you'll see that the deployment is gone from the list.

Based on this simple example you can see that the deploy command actually automatically performs two steps: uploads the content and enables it. And the undeploy command, by default, disables the application and removes its content from the repository.

If necessary though, you can just upload the content to the repository w/o enabling it. The deploy command has --disabled switch for that, e.g.

[standalone@localhost:9999 /] deploy ../../../../testsuite/smoke/target/deployments/test-deployment.sar --disabled

'test-deployment.sar' deployed successfully.

Which will result in the following in the standalone.xml

<deployments>

    <deployment name="test-deployment.sar" runtime-name="test-deployment.sar" sha1="18a29bb79220c29f0c1c4c6cf65e466c7f347713" enabled="false"/>

</deployments>

Then to enable the deployment you'll need to specify just its name as the argument (if you press the tab key after '--name=' it'll help you autocomplete the value)

[standalone@localhost:9999 /] deploy --name=test-deployment.sar

'test-deployment.sar' deployed successfully.

<deployments>

    <deployment name="test-deployment.sar" runtime-name="test-deployment.sar" sha1="18a29bb79220c29f0c1c4c6cf65e466c7f347713"/>

</deployments>

Now, when undeploying, If you don't want to remove the content but just disable (stop) the running application then you can use --keep-content switch, e.g.

[standalone@localhost:9999 /] undeploy test-deployment.sar --keep-content

Successfully undeployed test-deployment.sar.

If you press the tab key after the deployment name, it'll autocomplete the --keep-content argument. And if you now look into the standalone.xml, you'll see

<deployments>

   <deployment name="test-deployment.sar" runtime-name="test-deployment.sar" sha1="af4edddaa426ccc367fed33cb67553ba21e0bc3d" enabled="false"/>

</deployments>

If you want to re-deploy an already deployed application (whether enabled or disabled) and replace its content with the new version, the deploy command will fail unless you specify --force argument, e.g.

[standalone@localhost:9999 /] deploy ../../../../testsuite/smoke/target/deployments/test-deployment.sar

'test-deployment.sar' is already deployed (use --force to force re-deploy).

[standalone@localhost:9999 /] deploy ../../../../testsuite/smoke/target/deployments/test-deployment.sar --force'test-deployment.sar' re-deployed successfully.

Removal of a disabled deployment is not different from an enabled deployment, e.g.

[standalone@localhost:9999 /] undeploy test-deployment.sar

Successfully undeployed test-deployment.sar.

TODO: the command result messages could be more specific...

If you check the standalone.xml you'll see that the deployment was removed from the server's deployment repository.

## Domain mode

Make sure the domain is up and the CLI is connected to its controller, e.g.

Connected to domain controller at localhost:9999

[domain@localhost:9999 /]

The main difference in deploying and undeploying in the domain mode is that you have to specify the server groups to which these operations should apply. In case of deploy, you can choose between two arguments to specify the server groups:

--all-server-groups    - to apply to all the available server groups;

--server-groups    - to apply the operation to the spicific list of server groups separated by commas.

E.g.

[domain@localhost:9999 /] deploy ../../../../testsuite/smoke/target/deployments/test-deployment.sar --all-server-groups

'test-deployment.sar' deployed successfully.

If you now looking into the domain.xml, you'll see something like this

    <deployments>

        <deployment name="test-deployment.sar" runtime-name="test-deployment.sar" sha1="af4edddaa426ccc367fed33cb67553ba21e0bc3d"/>

    </deployments>

    <server-groups>

        <server-group name="main-server-group" profile="default">

            <jvm name="default">

                <heap size="64m" max-size="512m"/>

            </jvm>

            <socket-binding-group ref="standard-sockets"/>

            <deployments>

                <deployment name="test-deployment.sar" runtime-name="test-deployment.sar" sha1="af4edddaa426ccc367fed33cb67553ba21e0bc3d"/>

            </deployments>

        </server-group>

        <server-group name="other-server-group" profile="default">

            <jvm name="default">

                <heap size="64m" max-size="512m"/>

            </jvm>

            <socket-binding-group ref="standard-sockets"/>

            <deployments>

                <deployment name="test-deployment.sar" runtime-name="test-deployment.sar" sha1="af4edddaa426ccc367fed33cb67553ba21e0bc3d"/>

            </deployments>

        </server-group>

    </server-groups>

The corresponding undeploy command (i.e. the one that would undo the just performed deploy command) is

[domain@localhost:9999 /] undeploy test-deployment.sar --all-relevant-server-groups

Successfully undeployed test-deployment.sar.

Notice that undeploy uses --all-relevant-server-groups instead of --all-server-groups. --all-relevant-server-groups means to undeploy the application from all the server groups in which the deployment is enabled (which might be a subset of all the available server groups).

And if you check the domain.xml now, you'll see that the deployment was removed from the domain deployment repository.

The following command is an equivalent to the deploy command using --all-server-groups above (assuming main-server-group and other-server-group are all the available server groups)

[domain@localhost:9999 /] deploy ../../../../testsuite/smoke/target/deployments/test-deployment.sar --server-groups=main-server-group,other-server-group

'test-deployment.sar' deployed successfully.

BTW, the tab-completion will help completing the value for --server-groups. You can check and make sure that the domain.xml looks the same as above after the deploy.

Now, suppose, we want to undeploy the application from only one server group, e.g.

[domain@localhost:9999 /] undeploy test-deployment.sar --server-groups=other-server-group --keep-content

Successfully undeployed test-deployment.sar.

Note, since the application is still supposed to be enabled in the main-server-group --keep-content must be specified, otherwise the command will fail. You can now check the domain.xml.

    <deployments>

        <deployment name="test-deployment.sar" runtime-name="test-deployment.sar" sha1="af4edddaa426ccc367fed33cb67553ba21e0bc3d"/>

    </deployments>

    <server-groups>

        <server-group name="main-server-group" profile="default">

            <jvm name="default">

                <heap size="64m" max-size="512m"/>

            </jvm>

            <socket-binding-group ref="standard-sockets"/>

            <deployments>

                <deployment name="test-deployment.sar" runtime-name="test-deployment.sar" sha1="af4edddaa426ccc367fed33cb67553ba21e0bc3d"/>

            </deployments>

        </server-group>

        <server-group name="other-server-group" profile="default">

            <jvm name="default">

                <heap size="64m" max-size="512m"/>

            </jvm>

            <socket-binding-group ref="standard-sockets"/>

        </server-group>

    </server-groups>

As you can see, the deployment is not just disabled but completely removed from the other-server-group. At the moment it is handled this way to keep the implementation of deploy/undeploy simpler.

To deploy the application to the other-server-group again you need to pass its name as the value of the --name argument to the deploy command, e.g.

[domain@localhost:9999 /] deploy --name=test-deployment.sar --server-groups=other-server-group

'test-deployment.sar' deployed successfully.

You can check the domain.xml to make sure that the deployment is back in the other-server-group.

Removing content from the domain deployment repository which is not referenced from any of the available server groups doesn't require specification of server groups. E.g.

[domain@localhost:9999 /] undeploy test-deployment.sar --all-relevant-server-groups --keep-content

Successfully undeployed test-deployment.sar.

    <deployments>

        <deployment name="test-deployment.sar" runtime-name="test-deployment.sar" sha1="af4edddaa426ccc367fed33cb67553ba21e0bc3d"/>

    </deployments>

    <server-groups>

        <server-group name="main-server-group" profile="default">

            <jvm name="default">

                <heap size="64m" max-size="512m"/>

            </jvm>

            <socket-binding-group ref="standard-sockets"/>

        </server-group>

        <server-group name="other-server-group" profile="default">

            <jvm name="default">

                <heap size="64m" max-size="512m"/>

            </jvm>

            <socket-binding-group ref="standard-sockets"/>

        </server-group>

    </server-groups>

[domain@localhost:9999 /] undeploy test-deployment.sar

Successfully undeployed test-deployment.sar.

Note, this is possible only the deployment isn't referenced from any of the available server groups, otherwise the command will fail. If you want to remove the deployment from the deployment repository in this case you'll have to specify either --all-relevant-server-groups or list them in --server-groups.

If you now check the domain.xml you'll see that the deployment has been removed from the domain deployment repository.

It is also possible to just upload the content to the domain's deployment repostiory w/o enabling it in any of the available server groups. E.g.

[domain@localhost:9999 /] deploy ../../../../testsuite/smoke/target/deployments/test-deployment.sar --disabled

'test-deployment.sar' deployed successfully.

After that, the domain.xml will look like this

<deployments>

        <deployment name="test-deployment.sar" runtime-name="test-deployment.sar" sha1="404ccff4db24ac37e603ec557f5d32f88087258a"/>

    </deployments>

    <server-groups>

        <server-group name="main-server-group" profile="default">

            <jvm name="default">

                <heap size="64m" max-size="512m"/>

            </jvm>

            <socket-binding-group ref="standard-sockets"/>

        </server-group>

        <server-group name="other-server-group" profile="default">

            <jvm name="default">

                <heap size="64m" max-size="512m"/>

            </jvm>

            <socket-binding-group ref="standard-sockets"/>

        </server-group>

    </server-groups>

**Encrypted Password for Data Source:**

In AS7 you can use the SecureIdentityLoginModule to add an encrypted password domain. For instance, you can define a security domain in standalone.xml or domain.xml:

<security-domain name="EncryptedPassword">

<authentication>

<login-module code="SecureIdentityLogin" flag="required">

<module-option name="username" value="test"/>

<module-option name="password" value="encrypted\_password"/>

</login-module>

</authentication>

</security-domain>

Then you can add this security domain in your particular data source that uses this userid/pwd combination in standalone.xml or domain.xml:

<datasource ... >

.....

<security>

<security-domain>EncryptedPassword</security-domain>

</security>

</datasource>

To encrypt the password itself, you can run this command (please verify the versions of picketbox jar and logging jar in your particular AS7 download to substitute accordingly):

java -cp $JBOSS\_HOME/modules/org/picketbox/main/picketbox-4.0.6.<beta|final>.jar:$JBOSS\_HOME/modules/org/jboss/logging/main/jboss-logging-3.1.0.<some\_version>.jar:$CLASSPATH org.picketbox.datasource.security.SecureIdentityLoginModule password