WildFly 8 Hands-on Lab

Arun Gupta

Revision 1.0
Jan 27, 2014

Table of Contents

List of Figures

Chapter 1. Introduction

WildFly 8 (nee JBoss Application Server 7) is a lightweight, flexible, Java EE 7 compliant, polyglot, and open source application server.

This self-paced hands-on lab introduces the basic administrative features of WildFly 8. A comprehensive lab that covers Java EE 7 development using WildFly is available at: https://github.com/javaee-samples/javaee7-hol.

This document is a work in progress. The latest version of this document is available at http://github.com/arun-gupta/wildfly-lab. Please send pull requests to contribute or file issues.

1.1. Getting Started

Download the WildFly 8 'Application Server Distribution' zip file from http://wildfly.org/downloads/ and unzip.

Lets get familiar with the directory structure.

Directory	Purpose
appclient	Configuration files, deployment content, log files, and writeable areas used by the application client from this installation
bin	Startup scripts, startup configuration files and other command line utilities
docs	XML Schema definition files and sample configurations
domain	Configuration files, deployment content, log files, and writeable areas used by the managed domain from this installation
domain/lib/ext	Directory for installed library jars referenced by applications using the Extension-List mechanism for this domain
domain/servers	Writable area for each application server instance that runs in this domain. Each server has its own directory., created when the server is started.
modules	Various modules used in the server
standalone	Configuration files, deployment content, log files, and writeable areas used by the single standalone server from this installation
standalone/deployments	Directory for filesystem-based automatic deployment

Directory	Purpose
standalone/lib/ext	Directory for installed library jars referenced by applications using the Extension-List mechanism for this instance
welcome-content	Default welcome page content

1.1.1. Standalone and Managed Domain

WildFly 8 can be started in two different modes:

1. A *standalone* server is an independent instance. It can be started using `bin/standalone.sh` launch script on Unix-variety of machines and `bin/standalone.bat` on Windows.
2. A *managed domain* that allows you to run and manage a multi-server topology and manage multiple WildFly instances from a single control point. It can be started using `bin/domain.sh` launch script on Unix-variety of machines and `bin/domain.bat` on Windows.
   A Domain Controller is the central management control point for the collection of servers in the 'domain'. More details about starting WildFly in managed domain mode, server topology and central management is explained in ???.

1.1.2. Web Profile and Full Platform

Java EE defines *profiles* that represent a configuration of the full platform suited to a particular class of applications. The Java EE Web Profile is defined as a separate specification in the platform, and is defined as a subset of technologies contained in the platform and targeted toward the developers of modern web applications.

WildFly supports both Web Profile and Full Platform by defininig 'configuration profiles' in different files. These profiles are defined in 'standalone/configuration/*.xml' files for standalone instance and 'domain/configuration/domain.xml' for managed domain.

1. Start a WildFly instance in web profile

   ```
   ./bin/standalone.sh
   ```

   This starts WildFly standalone instance server in web profile. By default it refers to `standalone\configuration\standalone.xml` configuration file which offers more capabilities than just web profile.

Access http://localhost:8080 and see the following page to make sure the server is running correctly.

Figure 1.1.

Use Ctrl+C to stop the running server.

2. Start in full platform by specifying the configuration as:

```
./bin/standalone.sh -c standalone-full.xml
```

The name of the exact configuration file from ‘standalone/configuration’ directory is specified as the parameter.

Use Ctrl+C to stop the running server.

1.1.3. Multiple instances on the same machine

Multiple instances of standalone server can be started on the same machine by specifying a port offset option to the `standalone.sh` or `standalone.bat` script.

1. Start another instance of standalone server at port 9080 using the following command:

```
./bin/standalone.sh -Djboss.socket.binding.port-offset=1000
```

   This will start another standalone instance server on port 9080 (=8080 + 1000) as the default application port and 10990 (=9990 + 1000) as the default management port. These port numbers would differ if non-default ports were used.

   After the server is stated the welcome page is now accessible at http://localhost:9080.

   Use Ctrl+C to stop the running server.

Chapter 2. Deploying Applications to WildFly

Purpose: This section explains different ways to deploy Java EE 7 applications to WildFly.

Make sure to start a WildFly standalone instance for each sub-section unless stated otherwise. A standalone instance can be started as:

```
standalone.sh
```

2.1. File system

Deploying archives such as war, ear, and rar files using file system is as simple as copying to `standalone/deployments` directory. The deployment can be configured for auto-deploy or manual mode.

Purpose: This section explains how to deploy a Java EE 7 application to WildFly using filesystem commands.

Any archive can be placed in `standalone/deployments` directory and is automatically deployed to the server. This is the default behavior.

This deployment mode is only intended to be used during development phase. Managed domain mode is recommended if running production systems.

1. Download a Java EE 7 sample application from https://github.com/arun-gupta/wildfly-lab/blob/master/samples/javaee7/javaee7-1.0-SNAPSHOT.war?raw=true.
2. Copy the war file to `standalone/deployments` directory using the following command:

```
cp target/javaee7-1.0-SNAPSHOT.war /Users/arungupta/workspaces/wildfly/
build/target/wildfly-8.0.0.Final-SNAPSHOT/standalone/deployments
```

3. Verify the deployed application by accessing the webpage http://localhost:8080/javaee7-1.0-SNAPSHOT/EmployeeList and see the output as:

Figure 2.1.

2.2. Administration Console

The Admin Console is a web-based administration console and available at http://localhost:9990/console.

Purpose: This section explains how to deploy a Java EE 7 application to WildFly using web-based Administration Console.

1. A new user in the management realm needs to be added before the admin console can be accessed.
   A new user can be added in the management realm using `add-user.sh` script. This can be done non-interactively by giving the following command:

   ```
   add-user.sh -u sheldon -p bazinga
   ```

   The output of the command looks like:

   ```
   Added user 'sheldon' to file '/Users/arungupta/workspaces/wildfly/
   build/target/wildfly-8.0.0.Final-SNAPSHOT/standalone/configuration/
   mgmt-users.properties'
   Added user 'sheldon' to file '/Users/arungupta/workspaces/wildfly/
   build/target/wildfly-8.0.0.Final-SNAPSHOT/domain/configuration/mgmt-
   users.properties'
   ```

   The directory names ‘standalone’ and ‘domain’ indicate that the user is added to both standalone instance and managed domain. The file name indicates that it is added to the management realm.

A user can be added to application realm by including `-a` switch.

Alternatively, the user can be added interactively by invoking the script and entering the values on the prompt as shown:

```
add-user.sh

What type of user do you wish to add?
 a) Management User (mgmt-users.properties)
 b) Application User (application-users.properties)
(a): <take default>

Enter the details of the new user to add.
Using realm 'ManagementRealm' as discovered from the existing property
 files.
Username : <sheldon>
Password requirements are listed below. To modify these restrictions
 edit the add-user.properties configuration file.
 - The password must not be one of the following restricted values
 {root, admin, administrator}
 - The password must contain at least 8 characters, 1 alphanumeric
 character(s), 1 digit(s), 1 non-alphanumeric symbol(s)
 - The password must be different from the username
Password : <bazinga>
JBAS015269: Password must have at least 8 characters!
Are you sure you want to use the password entered yes/no? <yes>
Re-enter Password : <bazinga>
What groups do you want this user to belong to? (Please enter a comma
 separated list, or leave blank for none)[  ]:
About to add user 'sheldon' for realm 'ManagementRealm'
Is this correct yes/no? <yes>
Added user 'sheldon' to file '/Users/arungupta/workspaces/wildfly/
build/target/wildfly-8.0.0.Final-SNAPSHOT/standalone/configuration/
mgmt-users.properties'
Added user 'sheldon' to file '/Users/arungupta/workspaces/wildfly/
build/target/wildfly-8.0.0.Final-SNAPSHOT/domain/configuration/mgmt-
users.properties'
Added user 'sheldon' with groups  to file '/Users/arungupta/workspaces/
wildfly/build/target/wildfly-8.0.0.Final-SNAPSHOT/standalone/
configuration/mgmt-groups.properties'
Added user 'sheldon' with groups  to file '/Users/arungupta/workspaces/
wildfly/build/target/wildfly-8.0.0.Final-SNAPSHOT/domain/configuration/
mgmt-groups.properties'
```

```
Is this new user going to be used for one AS process to connect to
 another AS process?
e.g. for a slave host controller connecting to the master or for a
 Remoting connection for server to server EJB calls.
yes/no? <no>
```

Now accessing the console at http://localhost:9990/console prompts for a username and password. Enter the username ‘sheldon’ and password ‘bazinga’ and then the console is shown as:

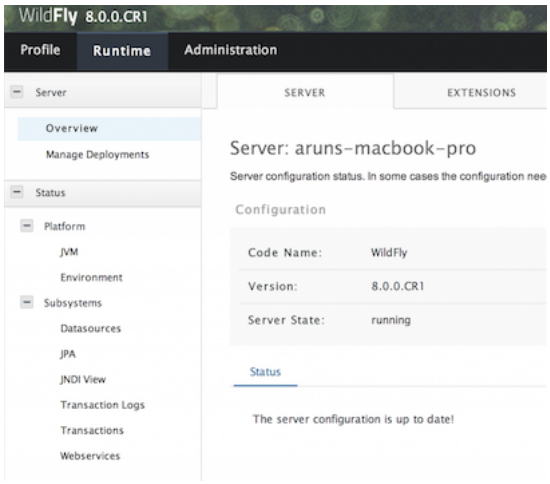

Figure 2.2.

The added user can be disabled by giving the following command:

```
add-user.sh -u sheldon -d
```

User can be deleted by deleting the corresponding line from `domain/configuration/mgmt-users.properties` and `standalone/configuration/mgmt-users.properties` files.

2. Click on ‘Manage Deployments’ to see the output as:

Figure 2.3.

3. Click on ‘Add’ button to see:

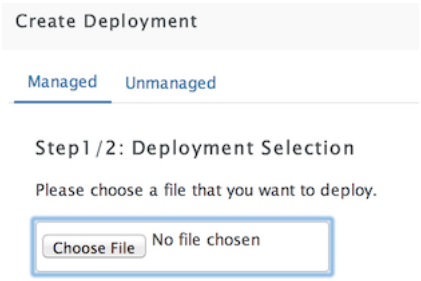

Figure 2.4.

4. Click on ‘Choose File’ button and select the war file.
   Click on ‘Next >>’ button. Take the defaults and click ‘Save’.
5. By default the application is not enabled and the console shows:

Figure 2.5.

Click on ‘En/Disable’ button, click on ‘Confirm’ and now the app is ready to be used. Access the application http://localhost:8080/javaee7-1.0-SNAPSHOT.war/EmployeeList.

2.3. Command Line Interface (jboss-cli)

Purpose: This section explains how to deploy a Java EE 7 application to WildFly using jboss-cli.

`jboss-cli.sh` or `jboss-cli.bat` is Command Line Interface management tool for a standalone server or a managed domain. It allows a user to connect to a standalone server or domain controller and execute management operations.

More details about using jboss-cli for managing the server are available in Chapter 3, *Management using Command Line Interface (jboss-cli)*.

Purpose: This section explains how to deploy a Java EE 7 application to WildFly using 'jboss-cli'.

'jboss-cli' can be used to deploy applications using the interactive console or in a non-interactive manner.

2.3.1. Interactive

1. Use jboss-cli to connect with the existing standalone instance by giving the following command:

```
jboss-cli.sh -c
```

   The `-c` switch connects using the default host ('localhost') and management port ('9990'). These values are specified in 'jboss-cli.xml' and can be updated.

   This opens up the 'jboss-cli' interactive console and shows the following prompt:

```
[standalone@localhost:9990 /]
```

   The prompt indicates that 'jboss-cli' is connected to a standalone instance's management port.

2. Deploy the application by giving the following command in console:

```
deploy target/javaee7-1.0-SNAPSHOT.war
```

   The directory name of the war file in the command may be different depending upon how 'jboss-cli' was invoked. Verify the server log to ensure that the application was redeployed. Look for specific timestamp in the log entries.

`--force` switch can be included in the command to replace the existing application.

3. Verify the deployment status by typing the following command `deployment-info` in the console:

```
deployment-info
```

and see the output as:

```
NAME                       RUNTIME-NAME               PERSISTENT ENABLED
 STATUS
javaee7-1.0-SNAPSHOT.war javaee7-1.0-SNAPSHOT.war true       true    OK
```

Verify the server log to ensure that the application was deployed. Look for specific timestamp in the log entries.

4. Undeploy the application by giving the following command:

```
undeploy javaee7-1.0-SNAPSHOT.war
```

5. Type `exit` 'quit' to exit 'jboss-cli' interactive console.

2.3.2. Non-interactive

Non-interactive mode allows to support scripts and other types of command line or batch processing.

1. Deploy the application using the following command:

```
jboss-cli.sh --connect --command="deploy target/javaee7-1.0-
SNAPSHOT.war --force"
```

The directory name of the war file in the command may be different depending upon how 'jboss-cli' was invoked. Verify the server log to ensure that the application was redeployed. Look for specific timestamp in the log entries.

Verify the deployed application at http://localhost:8080/javaee7/EmployeeList and look for a similar output.

2. Verify the deployment status by giving the following command:

```
jboss-cli.sh --connect --command=deployment-info
```

3. Undeploy the application by giving the following command:

```
jboss-cli.sh --connect --command="undeploy javaee7-1.0-SNAPSHOT.war"
```

2.4. Curl

curl is a free and popular command line tool for transfering data using URL syntax. If you don't have it installed on your machine then it can be downloaded from http://curl.haxx.se/download.html.

Purpose: This section explains how to deploy a Java EE 7 application to WildFly using curl.

Deploying applications using curl is a two-step process.

1. Upload your archive to WildFly using the following command

```
curl -F "file=@target/javaee7-1.0-SNAPSHOT.war" --digest http://
sheldon:bazinga@localhost:9990/management/add-content
```

   This command:

   a. Makes a POST request using form-encoded (`-F`) data with one field (`file`) defining the location of the WAR file

   b. `target/javaee7-1.0-SNAPSHOT.war` is the location of the WAR file

   c. `sheldon` is the administrative user with password `bazinga`

   d. `localhost:9090` is the default management host and port for WildFly instance

   e. WildFly management port uses digest authentication and that is defined using `-digest`

   f. Prints the output as something like:

```
{“outcome” : “success”, “result” : { “BYTES_VALUE” :
 “+Dg9u1ALXacrndNdLrT3DQSaqjw=” }}
```

2. Deploy the uploaded archive using the following command:

```
curl -S -H "Content-Type: application/json" -d '{"content":[{"hash":
 {"BYTES_VALUE" : "+Dg9u1ALXacrndNdLrT3DQSaqjw="}}], "address":
 [{"deployment":"javaee7-1.0-SNAPSHOT.war"}], "operation":"add",
 "enabled":"true"}' --digest http://sheldon:bazinga@localhost:9990/
management
```

   This command:

a. Sends a POST request (`-d`) with JSON payload
b. Value assigned to `result` name in the JSON response of previous command is assigned to `hash` name in this command
c. Content type of the payload is explicitly specified to be `application/json` using `-H`
d. `add` command triggers the deployment of the archive
e. Application archive is enabled as well, as opposed to not by default
f. As in previous command, `sheldon` is the administrative user with password `bazinga`
g. As in previous command, `localhost:9090` is the default management host and port for WildFly instance
h. As in previous command, WildFly management port uses digest authentication and that is defined using `-digest`

2.5. HTTP APIs (TBD)

http://localhost:9990/management?operation=deploy

Purpose: This section explains how to deploy a Java EE 7 application to WildFly using HTTP API.

2.6. Maven plugin

The wildfly-maven-plugin is used to deploy, redeploy, undeploy or run your application. You can also deploy or undeploy artifacts, such as JDBC drivers, and add or remove resources. There is also the ability to execute CLI commands.

Purpose: This section explains how to deploy Java EE 7 applications to WildFly use the maven plugin.

1. Add the following fragment to samples/javaee7/pom.xml:

```
<plugin>
    <groupId>org.wildfly.plugins</groupId>
    <artifactId>wildfly-maven-plugin</artifactId>
    <version>1.0.0.Beta1</version>
    <executions>
        <execution>
            <phase>install</phase>
```

```
            <goals>
                <goal>deploy</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

along with other <plugin> elements.

This adds the 'wildfly-maven-plugin' description to 'pom.xml'. It also invokes the plugin 'deploy' target during the standard maven 'install' phase.

2. Start a WildFly instance as:

```
standalone.sh
```

3. Deploy the application by giving the command:

```
mvn wildfly:deploy
```

or

```
mvn install
```

4. Access the web page at http://localhost:8080/javaee7/TestServlet and see the output as:

Figure 2.6.

2.7. Java

Purpose: This section explains how to deploy a Java EE 7 application to WildFly using Java Management APIs.

A standalone WildFly process or a managed domain can be configured to listen for remote management requests using its “native management interface”. This interface uses an open protocol based on the JBoss Remoting library. JBoss Remoting is used to establish a communication channel from the client to the process being managed. Once the communication channel is established the primary traffic over the channel is management requests initiated by the client and asynchronous responses from the target process.

Deployment using a custom Java class is a two-step process:

1. First step is to build a deployment plan that describe the steps of uploading the archive to the content repository and deploying it.
2. Second step involves executing the deployment plan on the application server instance.

Lets create an application that will allow us to deploy a Java EE 7 application using a Java-based class.

1. Create a maven project as:

```
mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes
 -DgroupId=org.wildfly.samples -DartifactId=deployment -
DinteractiveMode=false
```

2. Open the created Maven project in NetBeans, Eclipse, IntelliJ or any other editor of your choice. The custom client must have maven artifact ‘org.jboss.as:jboss-as-controller-client’ and its dependencies on the classpath. ‘org.jboss:jboss-dmr’ is maven artifact for detyped representation of the management model and must be added to the list of maven dependencies as well.
   This can be achieved by adding the following dependencies in `pom.xml`:

```
<dependency>
    <groupId>org.jboss.as</groupId>
    <artifactId>jboss-as-controller-client</artifactId>
    <version>7.2.0.Final</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.jboss</groupId>
    <artifactId>jboss-dmr</artifactId>
    <version>1.2.0.Final</version>
    <scope>test</scope>
```

```
</dependency>
```

Note that both the dependencies are added in ‘test’ scope.

3. Edit the `src/test/java/org/wildfly/samples/AppTest.java` file and add a method as shown:

```
ModelControllerClient createClient(final InetAddress host, final int
 port,
            final String username, final String password) {

   final CallbackHandler callbackHandler = new CallbackHandler() {

       @Override
       public void handle(Callback[] callbacks) throws IOException,
 UnsupportedCallbackException {
           for (Callback current : callbacks) {
               if (current instanceof NameCallback) {
                   NameCallback ncb = (NameCallback) current;
                   ncb.setName(username);
               } else if (current instanceof PasswordCallback) {
                   PasswordCallback pcb = (PasswordCallback) current;
                   pcb.setPassword(password.toCharArray());
               } else {
                   throw new UnsupportedCallbackException(current);
               }
           }
       }
   };

   return ModelControllerClient.Factory.create(host, port,
 callbackHandler);
}
```

In this code:

a. `ModelControllerClient` is used to manage a WildFly server instance or a Domain Controller or slave Host Controller.

b. A new instance is created by specifying host, port, and a Callback handler to obtain authentication information for the call. Typically these credentials are obtained from a dialog box, such as a GUI-base client. In our case the credentials will be passed when this method is invoked.

4. Update `testApp` method such that it looks like:

```
public void testApp() throws IOException, InterruptedException,
 ExecutionException {
    // create client
    ModelControllerClient client =
 createClient(InetAddress.getByName("localhost"),
        9999,
        "sheldon",
        "bazinga");

    // connect
    ServerDeploymentManager manager =
 ServerDeploymentManager.Factory.create(client);

    // build a deployment plan
    DeploymentPlan plan = manager
        .newDeploymentPlan()
        .add(new File("/Users/arungupta/workspaces/wildfly-lab/samples/
javaee7/target/javaee7-1.0-SNAPSHOT.war"))
        .andDeploy()
        .build();

    // run it
    ServerDeploymentPlanResult result = manager.execute(plan).get();
}
```

Make sure to resolve imports from `org.jboss.as.controller.client.helpers.standalone.*` package.

In this code:

a. A new instance of client for controlling application server management model is created by specifying the correct host, port, username, and password. Note that the username and password are the ones specified earlier.

b. `ServerDeploymentManager` is the primary deployment interface for a standalone instance. A new instance is created by using the previously created `ModelControllerClient`.

c. `newDeploymentPlan` initiates the creation of a new `DeploymentPlan`. Archive is added to the repository using `add` method and deployed using `andDeploy` method. Finally the deployment plan is built using `build` method. Note the location of archive may be different in your environment.

d. Finally the deployment plan is executed using `execute` method.

5. Perform the deployment by running the test as:

```
mvn test
```

6. Check the server log to ensure that the application is deployed. The following server log entries should be visible:

```
14:08:57,052 INFO  [org.wildfly.extension.undertow] (MSC service thread
 1-9) JBAS017534: Register web context: /javaee7-1.0-SNAPSHOT
14:08:57,063 INFO  [org.jboss.as.server] (management-handler-thread -
 17) JBAS018559: Deployed "javaee7-1.0-SNAPSHOT.war" (runtime-name :
 "javaee7-1.0-SNAPSHOT.war")
```

You may see the error message as:

```
JBAS014616: Operation ("add") failed - address: ([("deployment" =>
 "javaee7-1.0-SNAPSHOT.war")]) - failure description: "JBAS014803:
 Duplicate resource [(\"deployment\" => \"javaee7-1.0-SNAPSHOT.war\")]"
```

This occurs if the archive has been previously deployed.

`addDeploy` method may be replaced with `andReplace("javaee7-1.0-SNAPSHOT.war")` to replace a previously deployed archive.

2.8. Ruby (TBD)

Purpose: This section explains how to deploy a Java EE 7 application to WildFly using Ruby.

Send a pull request ?

2.9. Perl (TBD)

Purpose: This section explains how to deploy a Java EE 7 application to WildFly using Perl.

Send a pull request ?

Chapter 3. Management using Command Line Interface (jboss-cli)

Purpose: This section explains how to manage WildFly using jboss-cli.

WildFly comes with a Command Line Interface management tool for a standalone server or a managed domain. This script is in the `bin` directory of WildFly distrbution and can be invoked by calling `jboss-cli.sh` for Mac/Linux based systems or `jboss-cli.bat` for Windows. It allows a user to connect to a standalone server or domain controller and execute management operations available through the management model.

3.1. Connecting to a server

1. Start a WildFly standalone server, if not already running, using the following command:

```
standalone.sh
```

2. Use 'jboss-cli' to connect with this instance by giving the following command:

```
jboss-cli.sh -c
```

   The `-c` switch connects using the default host ('localhost') and management port ('9990'). These values are specified in 'bin/jboss-cli.xml' and can be updated.

   This opens up the 'jboss-cli' interactive console and shows the following prompt:

```
[standalone@localhost:9990 /]
```

   The prompt indicates that 'jboss-cli' is connected to a standalone instance's default management port.

3. If WildFly instance is running on a different host and/or port, then `--controller` switch can be used to specify that information.

   a. In another shell, start another WildFly instance on a different port using the following command:

```
./bin/standalone.sh -Djboss.socket.binding.port-offset=10000
```

This will start another WildFly standalone instance on application port 18080 and management port 19090.

b. Use 'jboss-cli' to connect with this instance by giving the following command:

```
jboss-cli.sh -c --controller=localhost:19990
```

This opens up the 'jboss-cli' interactive console and shows the following prompt:

```
[standalone@localhost:19990 /]
```

The prompt indicates that 'jboss-cli' is connected to a standalone instance on port '19990'.

c. Type 'exit' or 'quit' in the console to exit.

3.2. Resources, Operations, and Commands

WildFly internal management model consists of *management resources* that that are added, removed, or modified by using *operations* and *commands*. Operations are low level but a comprehensive way to manage the server. Commands are more user-friendly, although most of them still translate into operation requests and some of them even into a few composite operation requests.

All resources are organized in a tree. The path to the node in the tree for a particular resource is its *address* and is identified by `/`.

Each resource expose information about their state as *attributes*.

Each resource may support child resources.

All resources expose metadata that describes their attributes, operations, and child types. This metadata can be queried by invoking one or more of the *global operations* supported by the resource.

Command and operation request history is enabled by default. While in the command line session, you can use the arrow keys to go back and forth in the history of commands and operations.

3.2.1. Operations

1. The operations require one of the following prefixes:

a. `:` to execute against the current node

Type `:` at the prompt and press tab key to see a complete list of operations. This will show the following output:

```
[standalone@localhost:9990 /] :
add-namespace                read-operation-names         take-
snapshot
add-schema-location          read-resource                undefine-
attribute
delete-snapshot              read-resource-description    upload-
deployment-bytes
full-replace-deployment      reload                       upload-
deployment-stream
list-snapshots               remove-namespace             upload-
deployment-url
read-attribute               remove-schema-location       validate-
address
read-children-names          replace-deployment           validate-
operation
read-children-resources      resolve-expression           whoami
read-children-types          resolve-internet-address     write-
attribute
read-config-as-xml           server-set-restart-required
read-operation-description   shutdown
```

Operations can be auto completed by using the tab key. For example, type `:r` at the prompt and press tab key to see the list of operations beginning with that letter.

Read a simple attribute using `read-attribute` operation as shown:

```
[standalone@localhost:9990 /] :read-attribute(name=release-version)
```

This will show the output as:

```
{
    "outcome" => "success",
    "result" => "8.0.0.Final-SNAPSHOT"
}
```

b. `./` to execute against a child node of the current path

```
[standalone@localhost:9990 deployment] ./javaee7-1.0-
SNAPSHOT.war:read-resource
```

This command reads resource's attribute, the deployed war file in this case, values along with either basic (default) or complete information about any child resources. Detailed information about the resource can be obtained by adding `recursive=true` parameters as:

```
[standalone@localhost:9990 /deployment] ./javaee7-1.0-
SNAPSHOT.war:read-resource(recursive=true)
```

c. `/` to execute an operation against the root node

```
[standalone@localhost:9990 /] /socket-binding-group=standard-
sockets/socket-binding=http:read-attribute(name=bound-port)
```

This command traverses a path and reads an attribute on the leaf node. In this case it reads the port and displays the output as:

```
{
    "outcome" => "success",
    "result" => 8080
}
```

2. Read the complete list of operations supported by a resource by typing `read-operation-names` at the prompt and see the output as:

```
[standalone@localhost:9990 /] :read-operation-names
{
    "outcome" => "success",
    "result" => [
        "add-namespace",
        "add-schema-location",
        "delete-snapshot",
        "full-replace-deployment",
        "list-snapshots",
        "read-attribute",
        "read-children-names",
        "read-children-resources",
        "read-children-types",
        "read-config-as-xml",
        "read-operation-description",
```

```
            "read-operation-names",
            "read-resource",
            "read-resource-description",
            "reload",
            "remove-namespace",
            "remove-schema-location",
            "replace-deployment",
            "resolve-expression",
            "resolve-internet-address",
            "server-set-restart-required",
            "shutdown",
            "take-snapshot",
            "undefine-attribute",
            "upload-deployment-bytes",
            "upload-deployment-stream",
            "upload-deployment-url",
            "validate-address",
            "validate-operation",
            "whoami",
            "write-attribute"
        ]
}
```

3. A number of operations can be applied to every resource. Such operations are called *global operations*. `read-operation-names` is one such global operation. Another commonly used global operation is `read-resource` that reads resource's attribute values along with either basic or complete information about any child resources.
4. Read complete detail about `read-operation-description` operation by giving the command:

```
[standalone@localhost:9990 /] :read-operation-description(name=read-
operation-names)
```

and see the output as:

```
{
    "outcome" => "success",
    "result" => {
        "operation-name" => "read-operation-names",
        "description" => "Gets the names of all the operations for the
 given resource",
        "request-properties" => {"access-control" => {
            "type" => BOOLEAN,
```

```
            "description" => "If 'true' only operations the user is
 allowed to see are returned, and filtered operations are listed in the
 'access-control' response header.",
            "expressions-allowed" => false,
            "required" => false,
            "nillable" => true,
            "default" => false
        }},
        "reply-properties" => {
            "type" => LIST,
            "value-type" => STRING,
            "description" => "The operation names"
        },
        "read-only" => true
    }
}
```

Try this operation for some other operations and read the output.

3.2.2. Commands

1. Type `help --commands` at the jboss-cli prompt to see a complete list of commands available in current context. This will show the following output:

```
[standalone@localhost:9990 /] help --commands
alias               deploy              if                  read-
attribute       undeploy
batch               deployment-info     jdbc-driver-info    read-
operation       unset
cd                  deployment-overlay  ls                  reload
          version
clear               echo                module              run-batch
          xa-data-source
command             echo-dmr            patch               set
          :
connect             help                pwd                 shutdown
data-source         history             quit                try
```

This can also be achieved by pressing the tab key at the prompt. The list of commands depends upon the current context, i.e. it may change based upon the node address in the domain management model.

Commands can be auto completed by using the tab key. For example, type letter `d` at the prompt and press tab key to see the list of commands beginning with that letter. Enter space after choosing the command and press tab key again to see the list of arguments to the command.

2. Help for any command is available by typing the command name and using `--help` option. For example:

```
[standalone@localhost:9990 /] deploy --help
```

will show the following output:

```
SYNOPSIS

    deploy ((file_path | --url=deployment_url)
                [--script=script_name] [--name=deployment_name]
                [--runtime-name=deployment_runtime_name]
                [--force | --disabled] [--unmanaged])
           | --name=deployment_name
           [--server-groups=group_name (,group_name)* | --all-server-
groups]
           [--headers={operation_header (;operation_header)*}]

DESCRIPTION

  Deploys the application designated by the file_path or enables an
 already
  existing but disabled in the repository deployment designated by the
 name
  . . .
```

3. `ls` command list the contents of a node path including node types and attributes. Giving this command on the root node shows the following output:

```
[standalone@localhost:9990 /] ls
core-service                        management-minor-version=0
deployment                          name=aruns-macbook-pro
deployment-overlay                  namespaces=[]
extension                           process-type=Server
interface                           product-name=undefined
path                                product-version=undefined
socket-binding-group                profile-name=undefined
subsystem                           release-codename=WildFly
```

```
system-property                          release-version=8.0.0.Final-
SNAPSHOT
launch-type=STANDALONE                   running-mode=NORMAL
management-major-version=2               schema-locations=[]
management-micro-version=0               server-state=running
```

All entries with name/value pairs are attributes and every thing else is a node.

4. `cd` command changes the current node path to the specified argument. Change the path to 'management' node by typing the command:

```
[standalone@localhost:9990 /] cd core-service=management
[standalone@localhost:9990 core-service=management]
```

The command line prompt in the first line shows that the command was issued from the root node. The prompt in the second line shows the updated node name.

5. Deploy an application and check its status by typing the following commands:

```
[standalone@localhost:9990 /] deploy ~/workspaces/wildfly-lab/samples/
javaee7/target/javaee7-1.0-SNAPSHOT.war --force
[standalone@localhost:9990 /] deployment-info
NAME                      RUNTIME-NAME              PERSISTENT ENABLED
 STATUS
javaee7-1.0-SNAPSHOT.war javaee7-1.0-SNAPSHOT.war true       true    OK
```

6. Change the HTTP application port from a default value of 8080 to 8090 by giving the following command:

```
[standalone@localhost:9990 /] /socket-binding-group=standard-sockets/
socket-binding=http:write-attribute(name=port,value=8090)
```

and see the output as:

```
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
```

The command output indicates that the server should be reloaded. This can be achieved by typing `reload` command at the prompt. Now the application is

accessible at http://8090/javaee7-1.0-SNAPSHOT/EmployeeList instead of the port 8080.

Any change to the management model is persisted to the configuration file. Lets change the port back to 8080 by giving the following command:

```
[standalone@localhost:9990 /] /socket-binding-group=standard-sockets/
socket-binding=http:write-attribute(name=port,value=8080)
```

3.3. Batch

The batch mode allows one to group commands and operations and execute them together as an atomic unit, i.e., if at least one of the commands or operations fails, all the other successfully executed commands and operations in the batch are rolled back.

Only the commands that translate into operation requests are allowed in the batch. The batch, actually, translates into a *composite* operation request.

Batch mode can be composed interactively using jboss-cli prompt or non-interactively where the set of commands and operations are saved in a file and loaded at the prompt.

3.3.1. Interactive Batch

1. Start batch mode by typing the `batch` command:

```
[standalone@localhost:9990 /] batch
[standalone@localhost:9990 / #]
```

   The prompt changes to `#` indicating that the CLI is in batch mode.

2. Enter the operations and commands that need to be included in batch:

```
[standalone@localhost:9990 / #] data-source add --name=myDataSource
 --jndi-name=java:jboss/datasources/MyDataSource --user-name=sa --
password=sa --driver-name=h2 --connection-url=jdbc:h2:mem:myData
[standalone@localhost:9990 / #] deploy ~/workspaces/wildfly-lab/
samples/javaee7/javaee7-1.0-SNAPSHOT.war
```

   This command is creating a JDBC resource and deploys an application that uses it.

3. Finally run the commands entered in the batch by giving the following command:

```
[standalone@localhost:9990 / #] run-batch
```

If the command is executed successfully then it is discarded and the CLI leaves the batch mode. If any of the command or operation in the batch fails then the CLI gives an error and all steps executed so far are rolled back.

3.3.2. Non-interactive Batch

Non-interactive batch is useful for set of commands and operations that are executed frequently. Such commands and operations can be saved to a file and later used as argument to 'batch' command.

1. Save the following commands in a text file:

```
/subsystem=datasources/data-source="java:jboss/datasources/
MyDataSource":add(jndi-name="java:jboss/datasources/MyDataSource",
 driver-name="h2", connection-url="jdbc:h2:mem:myData", user-name="sa",
 password="sa")
deploy ~/workspaces/wildfly-lab/samples/javaee7/target/javaee7-1.0-
SNAPSHOT.war
```

   and save the file as 'myScript.txt'.

2. Run the interactive CLI as:

```
standalone.sh
```

3. Load the script file using `batch` command:

```
[standalone@localhost:9990 /] batch --file=myScript.txt
```

   + Run the batch using the `run-job` command.

   Alternatively, the file may be loaded and executed in one command:

```
[standalone@localhost:9990 /] run-batch --file=myScript.txt
```

3.4. Environment variables

CLI supports variables and are resolved during command line parsing phase. They are useful to store frequently used nodepaths, complex commands or operations, or any other text that needs a shorter and easy to use name.

Variables set during a CLI session are not persisted when the session is terminated. The variables may be stored in `.jbossclirc` in which case they are persisted across different sessions.

3.4.1. Non-persistent Variables

1. Set a new variable as:

```
[standalone@localhost:9990 /] set default_port=/socket-binding-
group=standard-sockets/socket-binding=http:read-attribute(name=bound-
port)
```

   This code defines a new variable `default_port` and sets its value to the defined operation. Variable names are expected to follow Java identifier format.

2. Use this variable in CLI to execute the command:

```
[standalone@localhost:9990 /] $default_port
```

   to see the output as:

```
{
    "outcome" => "success",
    "result" => 8090,
    "response-headers" => {"process-state" => "reload-required"}
}
```

3.4.2. Persistent Variables

Persistent variables are stored in `.jbossclirc` file. The location of this file is checked in the following order:

1. value of system property `jboss.cli.rc`
2. user's working directory (as defined by `user.dir` system property)
3. `bin` directory

A default `.jbossclirc` is already included in the `bin` directory and can be used as a template for user-specific environment setup.

The file contains 'set' commands to define the variables, such as:

```
set default_port=/socket-binding-group=standard-sockets/socket-
binding=http:read-attribute(name=bound-port)
```

3.5. GUI

CLI can be started with a GUI instead of a command line. It allows you to browse through different nodes and commands and operations supported on a node. Commands are automatically created and can be submitted to the server. Applications can be deployed and undeployed as well.

1. Type the following command to start CLI with GUI:

   ```
   standalone.sh --gui
   ```

   The complete domain model is shown in a separate window as:

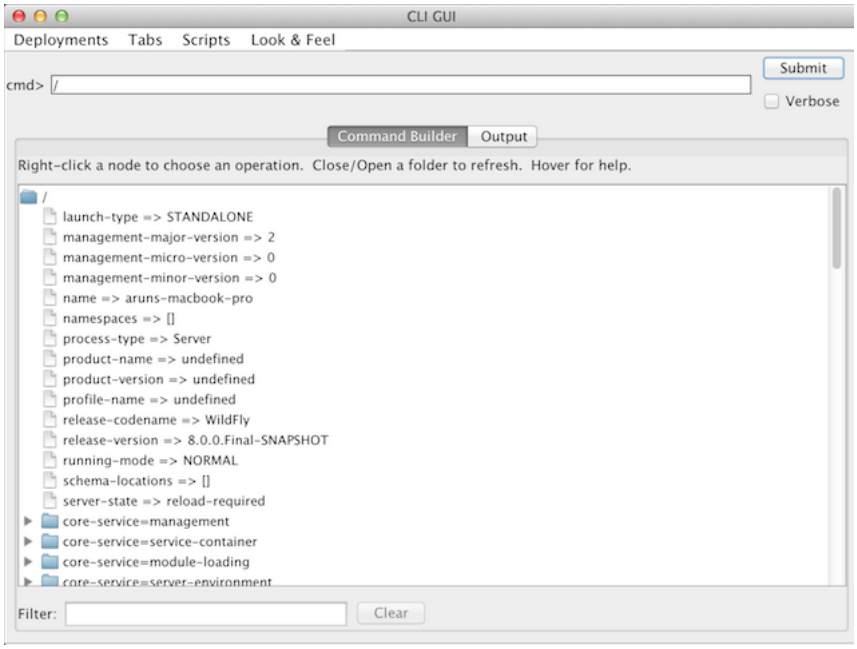

Figure 3.1.

2. Right-click on any node to see the list of supported operations as shown:

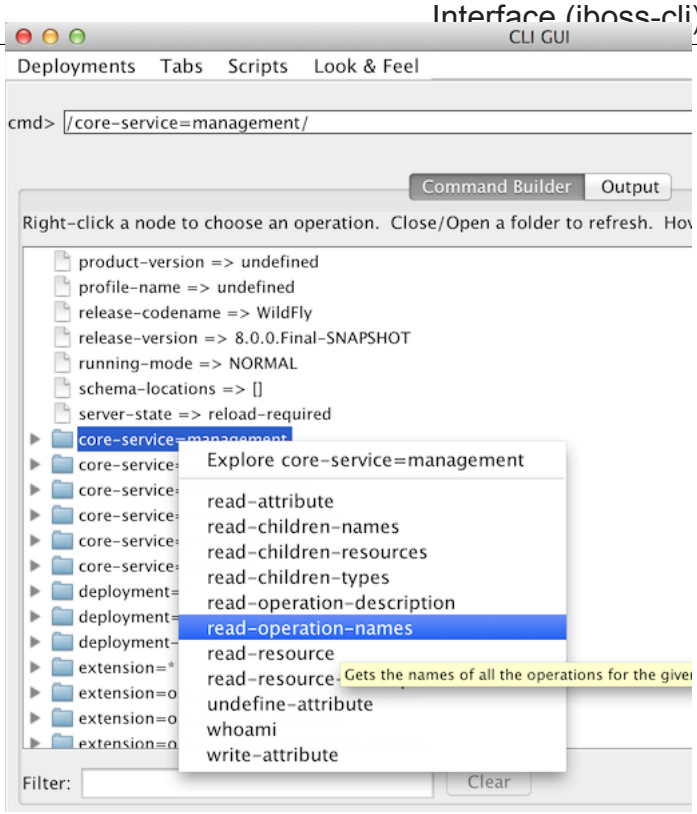

Figure 3.2.

The command is dynamically created and populated in the ‘cmd>’ text box.

3. This command can be submitted to the server by clicking on ‘Submit’ button. Command output is shown:

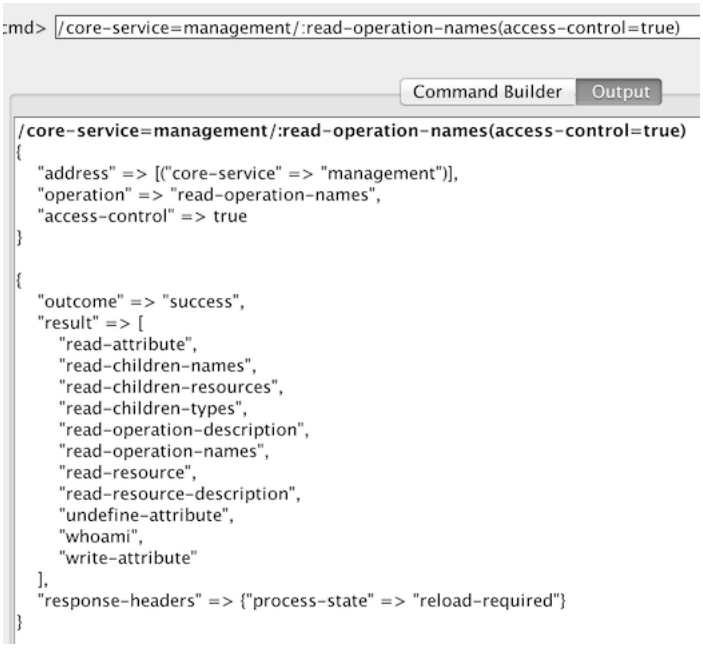

Figure 3.3.

4. Type ‘serv’ in ‘Filter’ box to search for any nodes and attributes that contains this phrase. The output is shown as:

Figure 3.4.

Chapter 4. Role Based Access Control

Role Based Access Control (RBAC) is the ability to restrict access to system or certain portions of it to authorized users. For JBoss AS 7.x, the web-based administrative console had an all-or-nothing approach. This means user authenticated with management security realm will have all the privileges. This may not be appropriate for mission-critical deployments and a finer-grained control may be required. WildFly 8 introduces RBAC using different roles.

Purpose: This section explains how to configure and use RBAC for WildFly.

There are seven pre-defined roles in two different categories. First four roles are where users are locked out of sensitive data and the next three level roles where users are allowed to deal with sensitive data.

The pre-defined roles are explained below:

Role	Permissions
Monitor	• Has the fewest permissions • Can only read configuration and current runtime state • No access to sensitive resources or data or audit logging resources
Operator	• All permissions of Monitor • Can modify the runtime state, e.g. reload or shutdown the server, pause/resume JMS destination, flush database connection pool. • Does not have permission to modify persistent state.
Maintainer	• All permissions of Operator • Can modify the persistent state, e.g. deploy an application, setting up new data sources, add a JMS destination
Deployer	• All permissions of Maintainer • Permission is restricted to applications only, cannot make changes to container configuration
Administrator	• All permissions of Maintainer • View and modify sensitive data such as access control system

Role	Permissions
	• No access to administrative audit logging system
Auditor	• All permissions of Monitor • View and modify resources to administrative audit logging system • Cannot modify sensitive resources or data outside auditing, can read any sensitive data
Super User	• Has all the permissions • Equivalent to administrator in previous versions

4.1. Default super user

By default, any user added to the management realm and not belonging to a group is in “Super User” role.

1. Start the server as standalone instance if not already running:

```
./bin/standalone.sh
```

2. Access Admin Console at http://localhost:9990/console. This prompts for authentication as shown:

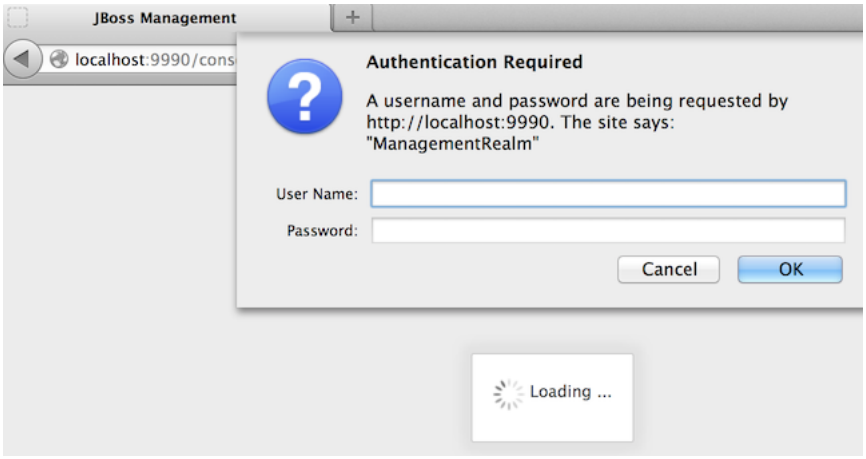

Figure 4.1.

Enter the user name ‘sheldon’ and password ‘bazinga’ (as created earlier). Admin console should look like:

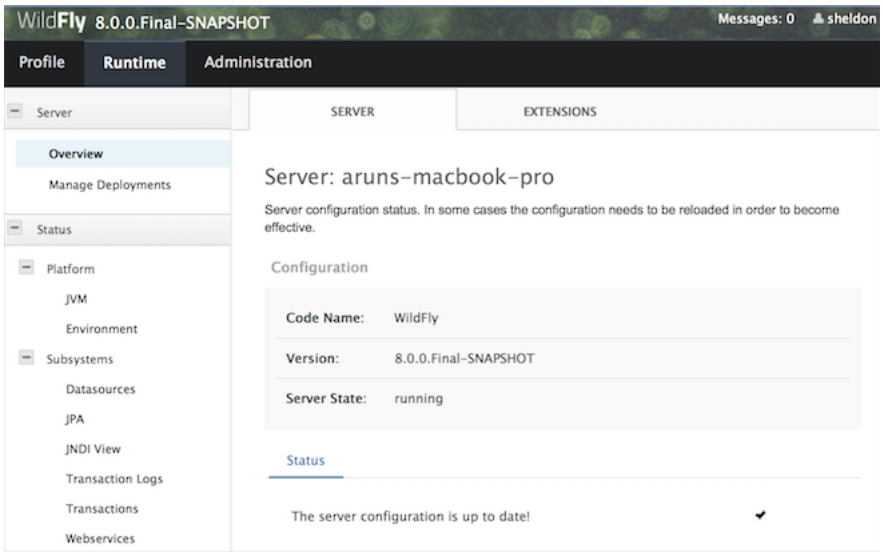

Figure 4.2.

This view shows:

a. A new ‘Administration’ tab that allows to map users to roles. This will be done in a later section.

b. Information about the logged in user is shown on top-right as:

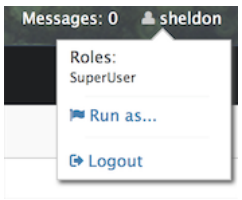

Figure 4.3.

Notice the logged in user name is shown.

c. A user in ‘Super User'' role can act to run in any role by clicking on ‘Run as’. Click on ‘Run as’ and select drop-down list box to see the list of available roles as:

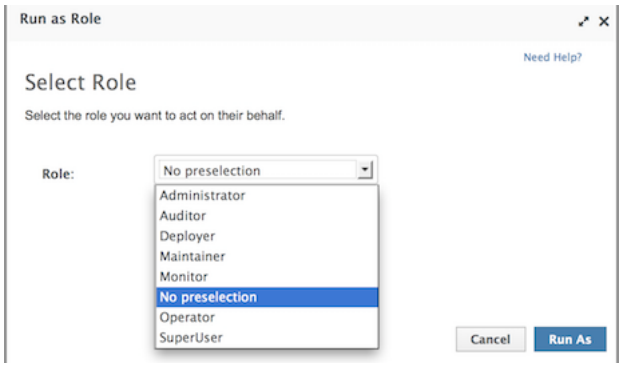

Figure 4.4.

d. Select the ‘Monitor’ role and click on ‘Run As’.
The application has to be reloaded for changes to take effect. Click on ‘Confirm’ to reload the application. After the reload, clicking on the user on top-right in admin console will display the selected role as ‘Run as Monitor’ as shown:

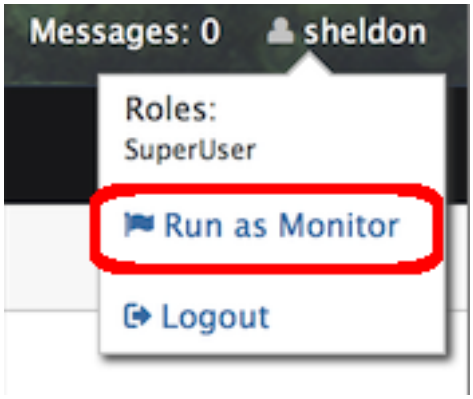

Figure 4.5.

i. Click on ‘Manage Deployments’ and check that ‘Add’, ‘Remove’, and similar buttons are not present as shown:

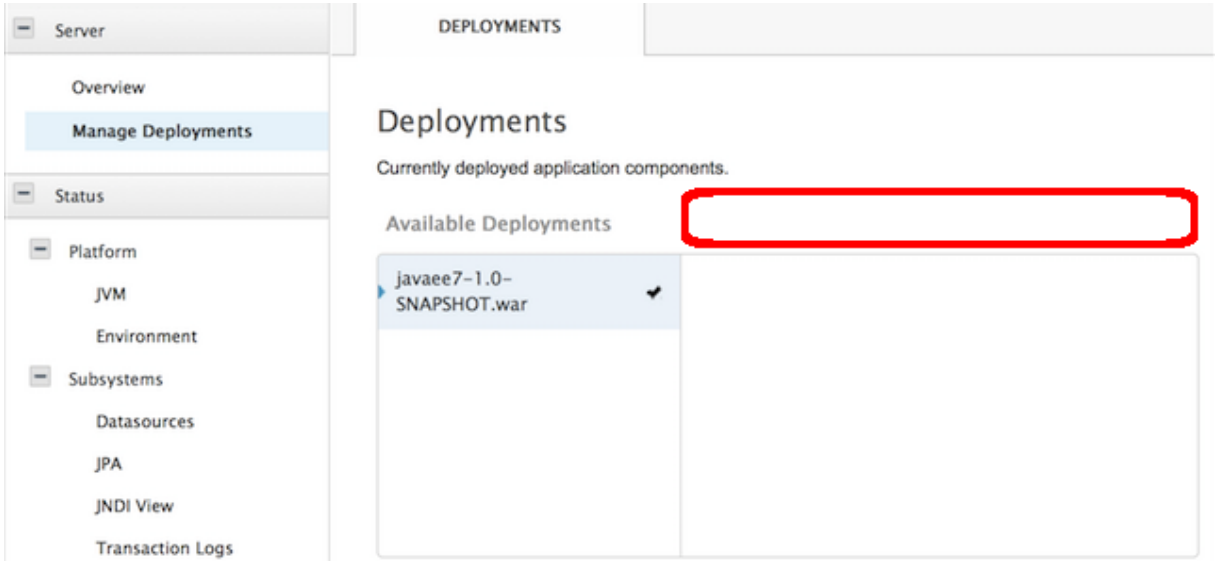

Figure 4.6.

ii. Click on ‘Profile’, ‘Data Sources’ and check that all data sources are visible but not editable. This is identified by the fact that ‘Add’, ‘Remove’, and ‘Disable’ buttons are not available as shown.

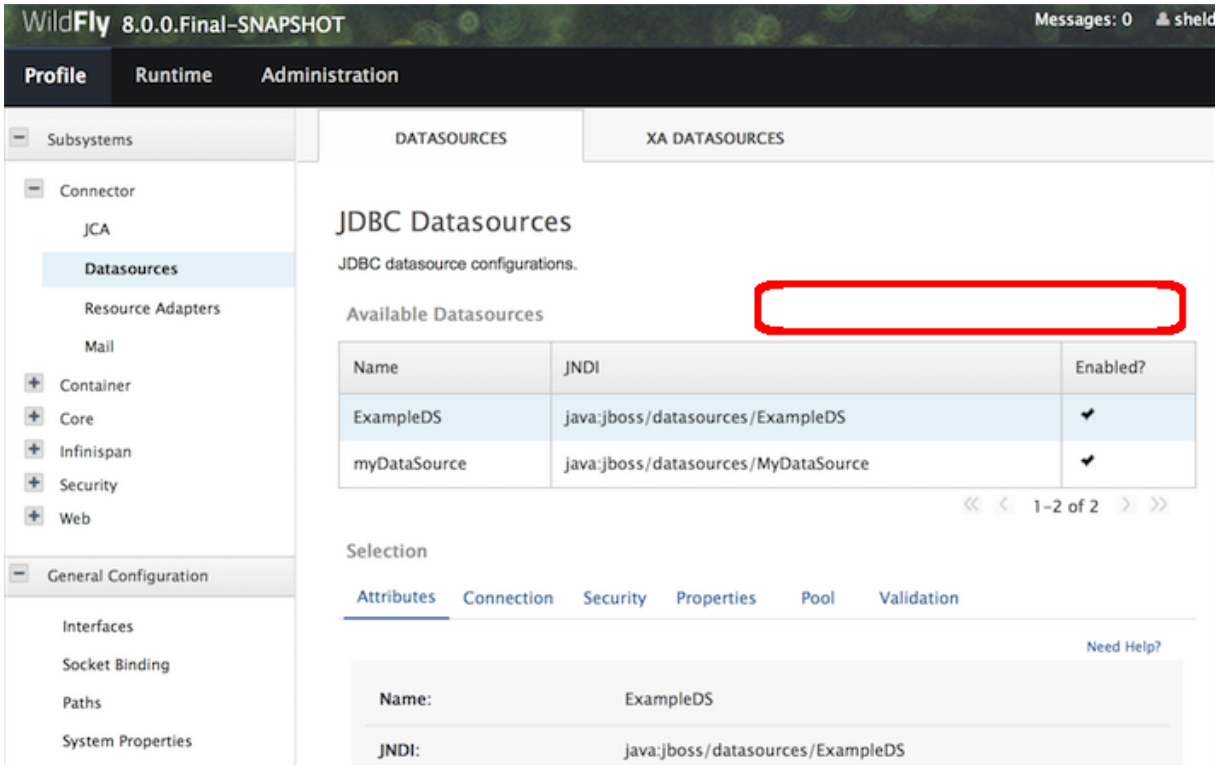

Figure 4.7.

iii. Click on ‘Administration’ tab and make sure the user does not have access to it as shown:

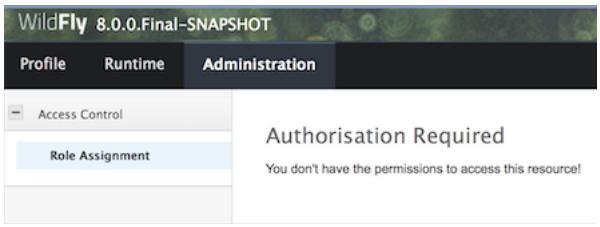

Figure 4.8.

e. Feel free to select other roles and observe how different options are enabled/disabled.

4.2. Configure ‘rbac’ access control provider

WildFly 8 comes with two access control providers:

- ‘simple’ provider, the default one, gives all privileges to any authenticated administrator. This provides compatibility with older releases.
- ‘rbac’ provider allows you to setup configuration that will map users to different roles.

1. Configure ‘rbac’ access control provider by giving the following command:

```
jboss-cli.sh -c --command="/core-service=management/
access=authorization:write-attribute(name=provider,value=rbac)"
```

This command will change authorization provider to “rbac” and will produce the output as:

```
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
```

2. The server needs to be restarted as the authorization provider is changed. Give the following command to restart the server:

```
./bin/jboss-cli.sh -c --command="reload"
```

If the server is running in managed domain then it can be restarted by additionally specifying `--host=master` in the command.

Check the server log to confirm server restarted, look for specific time stamps.

3. Any existing roles need to be explicitly mapped after the access control provider is changed. Map the user 'sheldon' to the role 'Super User' by giving the following command:

```
jboss-cli.sh -c --command="/core-service=management/
access=authorization/role-mapping=SuperUser/include=user-
sheldon:add(name=sheldon,type=USER)"
```

4. On top-right in admin console, click on the username and click on 'Logout' and then 'Confirm'.

5. Enter the login credentials again (username is 'sheldon' and password is 'bazinga') to login back into the admin console.

4.3. Map users, groups, and roles

WildFly introduces the concept of "groups" in security realms. Users can be directly associated with a role, or can belong to a group and then a group can be associated with a role.

1. Add two users in different groups using `bin/adduser.sh` script

   a. Add first user in a group by giving the following command:

```
add-user.sh -u penny -p penny1 -g just4fun
Added user 'penny' to file '/Users/arungupta/workspaces/wildfly/
build/target/wildfly-8.0.0.Final-SNAPSHOT/standalone/configuration/
mgmt-users.properties'
Added user 'penny' to file '/Users/arungupta/workspaces/wildfly/
build/target/wildfly-8.0.0.Final-SNAPSHOT/domain/configuration/mgmt-
users.properties'
Added user 'penny' with groups just4fun to file '/Users/arungupta/
workspaces/wildfly/build/target/wildfly-8.0.0.Final-SNAPSHOT/
standalone/configuration/mgmt-groups.properties'
Added user 'penny' with groups just4fun to file '/Users/arungupta/
workspaces/wildfly/build/target/wildfly-8.0.0.Final-SNAPSHOT/domain/
configuration/mgmt-groups.properties'
```

b. Add another user in a different group by giving the following command:

```
add-user.sh -u leonard -p leonard1 -g geek
Added user 'leonard' to file '/Users/arungupta/workspaces/wildfly/
build/target/wildfly-8.0.0.Final-SNAPSHOT/standalone/configuration/
mgmt-users.properties'
Added user 'leonard' to file '/Users/arungupta/workspaces/wildfly/
build/target/wildfly-8.0.0.Final-SNAPSHOT/domain/configuration/mgmt-
users.properties'
Added user 'leonard' with groups geek to file '/Users/arungupta/
workspaces/wildfly/build/target/wildfly-8.0.0.Final-SNAPSHOT/
standalone/configuration/mgmt-groups.properties'
Added user 'leonard' with groups geek to file '/Users/arungupta/
workspaces/wildfly/build/target/wildfly-8.0.0.Final-SNAPSHOT/domain/
configuration/mgmt-groups.properties'
```

These commands creates the following users:

User	Password	Group
penny	penny1	just4fun
leonard	leonard1	geek

Both users are added for standalone instance and managed domain.

2. Click on ‘Administration’ tab to see an output as:

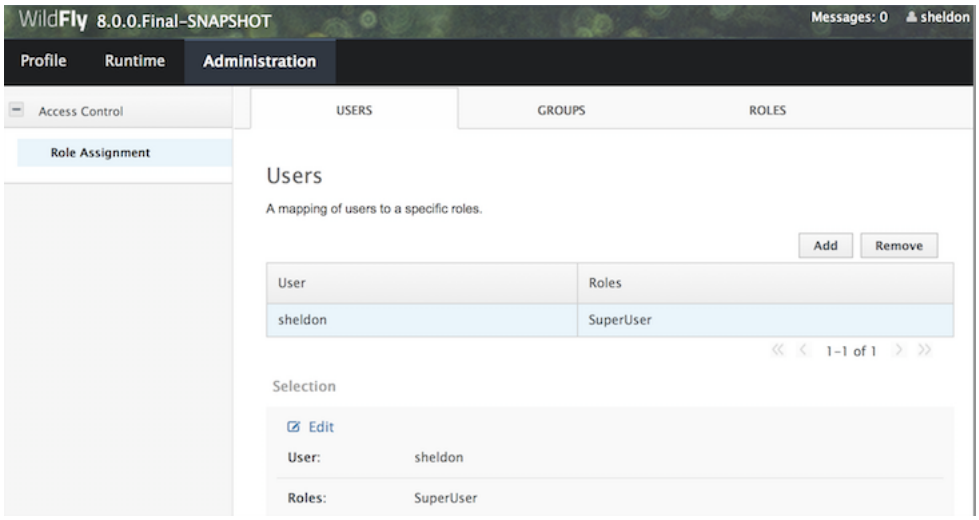

Figure 4.9.

Previously assigned user/role mapping is already shown here.

3. Click on ‘Add’ to assign a new role to user mapping. Type ‘penny’ in ‘User’ textbox and select ‘Monitor’ role as shown:

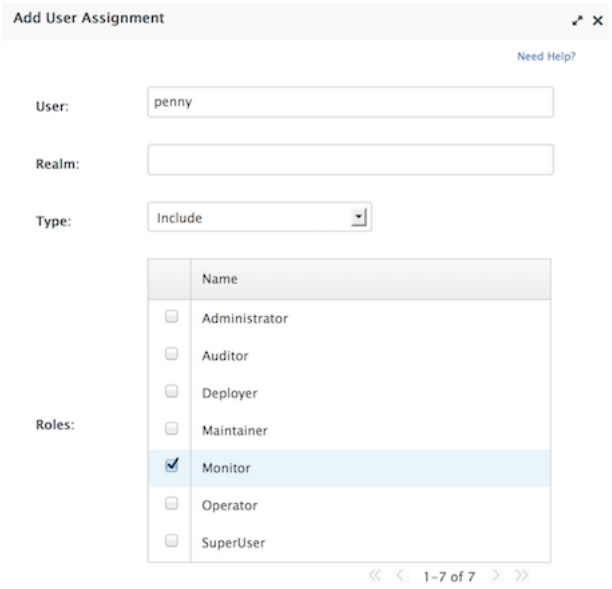

Figure 4.10.

Click on ‘Save’.

Multiple roles may be assigned to each user.

4. Assign ‘Administrator’ role to user ‘leonard’. The updated admin console looks like as shown:

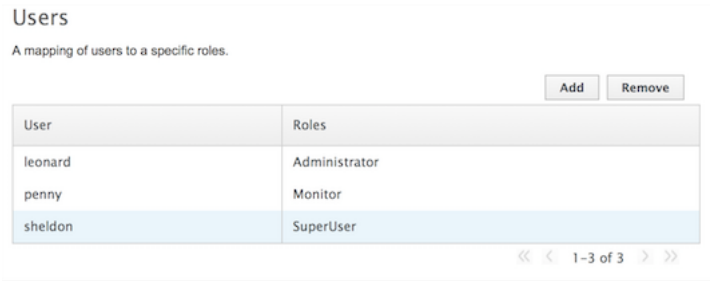

Figure 4.11.

Groups, and thus all users in that group, can be assigned one or more roles by clicking on ‘GROUPS’ tab.

4.4. Logging in as different users

1. Click on top-right and select ‘Logout’ to log out of admin console. Login again by using the username ‘penny’ and password ‘penny1’. Note that this user was assigned ‘Monitor’ role.
2. Top-right of admin console shows the logged in user name.

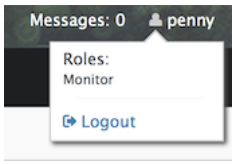

Figure 4.12.

Note that ‘Run as’ is not available any more.

3. Click on ‘Manage Deployments’ to see the output as shown:

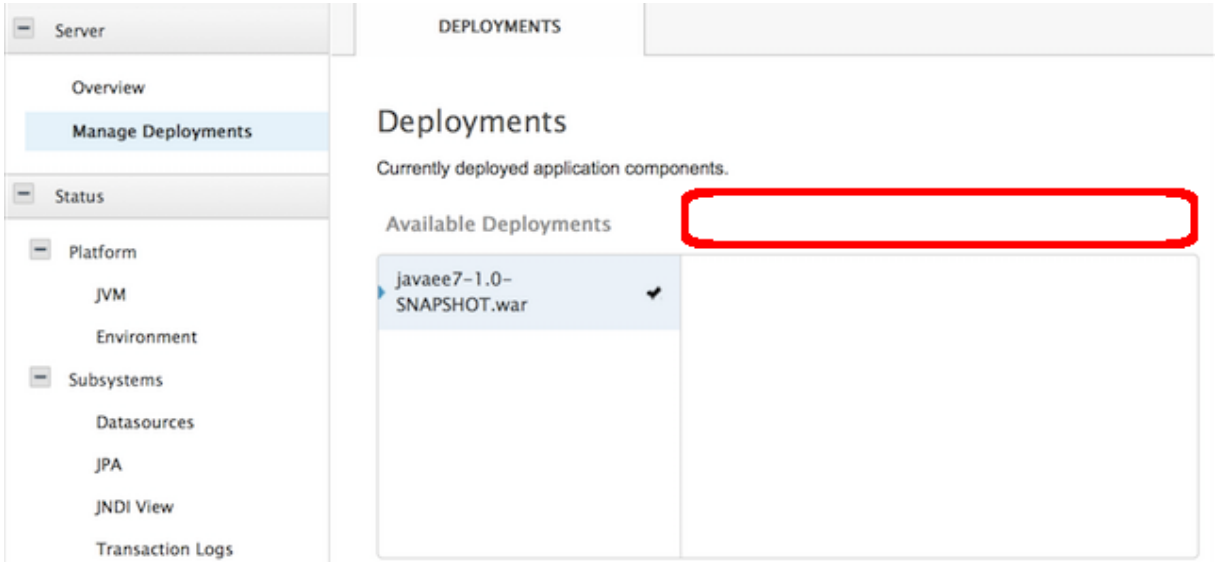

Figure 4.13.

This role permits only monitoring and ‘Add’, ‘Remove’, ‘En/Disable’, and ‘Replace’ buttons are not available.

4. Click on ‘Administration’ tab to see a permissiond denied output as:

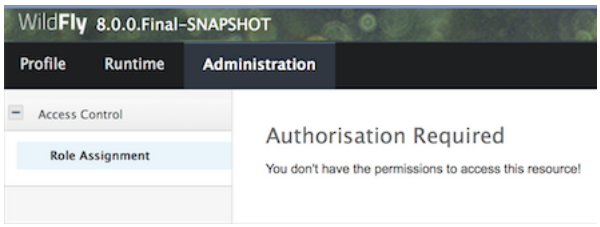

Figure 4.14.

5. Click on top-right and select ‘Logout’ to log out of admin console. Login again by using the username ‘leonard’ and password ‘leonard1’. Note that this user was assigned ‘Administrator’ role.
6. Top-right of admin console shows the logged in user name.

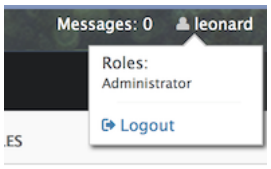

Figure 4.15.

Note that ‘Run as’ is not available any more.

7. Click on ‘Deployments’ and confirm that new deployments can be added or existing can be replace, removed, enabled or disabled by the presence of buttons.
8. Click on ‘Administration’ tab and confirm that all information is visible and editable.

4.5. Filtering out commands in ‘jboss-cli’

CLI or ‘jboss-cli’ can authenticate against local WildFly without prompting the user for a username and password. This mechanism only works if the user running the CLI has read access to the “standalone/tmp/auth” directory or “domain/tmp/auth” folder under the respective WildFly installation. If the local mechanism fails then the CLI will fallback to prompting for a username and password.

Alternatively authentication can be forced by explicitly specifying `user` and `password` options.

1. Connect using ‘jboss-cli’ using the following command:

```
jboss-cli.sh --user=penny --password=penny1 -c
```

Note that the user ‘penny’ is in Monitor role.

2. Type `data-source` on the CLI console and TAB to see the following output:

```
[standalone@localhost:9990 /] data-source
--help                                flush-gracefully-connection-in-
pool
--name=                               flush-idle-connection-in-pool
add                                   flush-invalid-connection-in-pool
disable                               read-resource
enable                                remove
flush-all-connection-in-pool          test-connection-in-pool
```

   Note that all commands and attributes, even those not permitted for Monitor role, are shown.

3. Try to add a new data source using the following command:

```
[standalone@localhost:9990 /] data-source add --name=testDataSource
```

   This command gives the following error:

```
JBAS013456: Unauthorized to execute operation 'add' for resource '[
    ("subsystem" => "datasources"),
    ("data-source" => "testDataSource")
]' -- "JBAS013475: Permission denied"
```

   This is because ‘Monitor’ role does not have permission to add data sources.

4. Type `exit` or `quit` to exit out of CLI console.
5. Edit `bin/jboss-cli.xml` and change the following element:

```
<access-control>false</access-control>
```

   to

```
<access-control>true</access-control>
```

   This element filter out the command and attribute suggestions displayed based on user’s permissions.

6. Connect using ‘jboss-cli’ using the following command:

```
./bin/jboss-cli.sh --user=penny --password=penny1 -c
```

7. Type `data-source` on the CLI console and TAB to see the following output:

```
[standalone@localhost:9990 /] data-source
--help          --name=         read-resource
```

Note that only permitted commands and attributes are shown.

Try some other commands and see which ones are accessible or not.

Chapter 5. Audit Logging

WildFly comes with audit logging built in for management operations affecting the management model. The audit log records can be logged to a file on the server or to syslog. Syslog is a better choice for audit logging since you can log to a remote syslog server, and secure the authentication to happen over TLS with client certificate authentication. This section will only cover default logging in file.

5.1. Turn on audit logging

By default audit logging is turned off and needs to be turned on.

1. Connect using 'jboss-cli' using the following command:

```
jboss-cli.sh --user=sheldon --password=bazinga -c
```

   Note you need to provide user and password credentials of a user in Auditor or Super User role.

2. Enable audit logging by giving the following command:

```
[standalone@localhost:9990 /] /core-service=management/access=audit/
logger=audit-log:write-attribute(name=enabled,value=true)
```

   This will display the output as:

```
{"outcome" => "success"}
```

   If the user is not in Auditor or Super User role then the following error message will be displayed:

```
{
    "outcome" => "failed",
    "failure-description" => "JBAS013456: Unauthorized to execute
 operation 'write-attribute' for resource '[
    (\"core-service\" => \"management\"),
    (\"access\" => \"audit\"),
    (\"logger\" => \"audit-log\")
]' -- \"JBAS013475: Permission denied\"",
    "rolled-back" => true
}
```

5.2. Logging JSON records to file

By default audit log records are formatted using JSON and are generated in `standalone/data/audit-log.log`. This file is generated after audit log is enabled and looks like:

```
{
    "type" : "core",
    "r/o" : false,
    "booting" : false,
    "version" : "8.0.0.Final-SNAPSHOT",
    "user" : "sheldon",
    "domainUUID" : null,
    "access" : "NATIVE",
    "remote-address" : "127.0.0.1/127.0.0.1",
    "success" : true,
    "ops" : [{
        "address" : [
            {
                "core-service" : "management"
            },
            {
                "access" : "audit"
            },
            {
                "logger" : "audit-log"
            }
        ],
        "operation" : "write-attribute",
        "name" : "enabled",
        "value" : true,
        "operation-headers" : {
            "caller-type" : "user",
            "access-mechanism" : "NATIVE"
        }
    }]
}
```

The main fields in this JSON record are explained below:

Field Name	Possible Values
`type`	• 'core' indicates it is a management operation • 'jms' indicates it is from the jms subsystem

Field Name	Possible Values
`r/o`	• 'true' if the operation does not change the management model • 'false' otherwise
`booting`	• 'true' is the operation was executed during the bootup process • 'false' otherwise
`user`	The username of the authenticated user. If the operation is logged via the CLI on the same machine as the running server, then the special $local user is used
`access`	• 'native' The operation came in through the native management interface, for example the CLI • 'http' The operation came in through the domain HTTP interface, for example the admin console • 'jmx' The operation came in through the JMX subsystem. See JMX for how to configure audit logging for JMX
`ops`	List of operations being executed serialized to JSON

Try a few other management operations and notice how `audit-log.log` is updated. For example, trying to add a new data source using the following command:

```
[standalone@localhost:9990 /] data-source add --name=myDataSource2 --jndi-
name=java:jboss/datasources/MyDataSource2 --user-name=sa --password=sa --
driver-name=h2 --connection-url=jdbc:h2:mem:myData
```

will generate the following audit log record:

```
{
    "type" : "core",
    "r/o" : false,
    "booting" : false,
    "version" : "8.0.0.Final-SNAPSHOT",
    "user" : "sheldon",
    "domainUUID" : null,
    "access" : "NATIVE",
    "remote-address" : "127.0.0.1/127.0.0.1",
    "success" : true,
    "ops" : [{
        "address" : [
            {
                "subsystem" : "datasources"
```

```
        },
        {
            "data-source" : "myDataSource2"
        }
    ],
    "operation" : "add",
    "user-name" : "sa",
    "password" : "sa",
    "jndi-name" : "java:jboss/datasources/MyDataSource2",
    "connection-url" : "jdbc:h2:mem:myData",
    "driver-name" : "h2",
    "operation-headers" : {
        "caller-type" : "user",
        "access-mechanism" : "NATIVE"
    },
    "driver-class" : null,
    "datasource-class" : null,
    "new-connection-sql" : null,
    "url-delimiter" : null,
    "url-selector-strategy-class-name" : null,
    "use-java-context" : null,
    "jta" : null,
    "max-pool-size" : null,
    "min-pool-size" : null,
    "initial-pool-size" : null,
    "pool-prefill" : null,
    "pool-use-strict-min" : null,
    "capacity-incrementer-class" : null,
    "capacity-decrementer-class" : null,
    "security-domain" : null,
    "reauth-plugin-class-name" : null,
    "flush-strategy" : null,
    "allow-multiple-users" : null,
    "connection-listener-class" : null,
    "connection-properties" : null,
    "prepared-statements-cache-size" : null,
    "share-prepared-statements" : null,
    "track-statements" : null,
    "allocation-retry" : null,
    "allocation-retry-wait-millis" : null,
    "blocking-timeout-wait-millis" : null,
    "idle-timeout-minutes" : null,
    "query-timeout" : null,
    "use-try-lock" : null,
    "set-tx-query-timeout" : null,
    "transaction-isolation" : null,
```

```
        "check-valid-connection-sql" : null,
        "exception-sorter-class-name" : null,
        "stale-connection-checker-class-name" : null,
        "valid-connection-checker-class-name" : null,
        "background-validation-millis" : null,
        "background-validation" : null,
        "use-fast-fail" : null,
        "validate-on-match" : null,
        "spy" : null,
        "use-ccm" : null,
        "enabled" : null,
        "reauth-plugin-properties" : null,
        "exception-sorter-properties" : null,
        "stale-connection-checker-properties" : null,
        "valid-connection-checker-properties" : null,
        "connection-listener-property" : null,
        "capacity-incrementer-properties" : null,
        "capacity-decrementer-properties" : null
    }]
}
```

Similarly deploying an application as:

```
[standalone@localhost:9990 /] deploy javaee7-1.0-SNAPSHOT.war --force
```

shows the following audit log record:

```
{
    "type" : "core",
    "r/o" : false,
    "booting" : false,
    "version" : "8.0.0.Final-SNAPSHOT",
    "user" : "sheldon",
    "domainUUID" : null,
    "access" : "NATIVE",
    "remote-address" : "127.0.0.1/127.0.0.1",
    "success" : true,
    "ops" : [{
        "operation" : "full-replace-deployment",
        "name" : "javaee7-1.0-SNAPSHOT.war",
        "content" : [{"input-stream-index" : 0}],
        "operation-headers" : {
            "caller-type" : "user",
            "access-mechanism" : "NATIVE"
```

```
        },
        "address" : null
    }]
}
```

Chapter 6. Acknowledgements

The following JBoss community members graciously reviewed and contributed to this hands-on lab:

- Jorge Solórzano (twitter handle ?)

Thank you very much for providing the valuable feedback!

Please send pull requests at http://github.com/arun-gupta/wildfly-lab to contribute or file issues.