02267 - Software Development for Web Services

Software Installation Guide

Kolkata

May 1, 2018

Danmarks Tekniske Universitet

Contents

Chapter 1

Running the Kolkata project

1.1 Public access to project

Repository contains of 4 different projects:

Kolkata (DTU Pay system)
KolkataTest
MobileSimulator
MerchantSimulator

KolkataTest, MobileSimulator and MerchantSimulator are standalone Java projects and Kolkata is the Java EE project which is deployed to server. Detailed building and deployment configurations for these project are mentioned in Chapter 3

1.2 Versions

Major dependency versions used in the Kolkata project

Maven	v.1.8	https://maven.apache.org/plugins/maven-compiler-pluginl
Java jdk	v.1.8.0	http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
cucumber	v.2.3.1	https://mvnrepository.com/artifact/io.cucumber/cucumber-jvm/2.3.1
Barcode4j	v.2.1	https://mvnrepository.com/artifact/net.sf.barcode4j/barcode4j/2.1
JUnit	v.4.12	https://mvnrepository.com/artifact/junit/junit/4.12
Javax.xml	v.1.1.1	https://mvnrepository.com/artifact/javax.xml.rpc/javax.xml.rpc-api/1.1.1
Javax.jms	v.2.0.1	https://mvnrepository.com/artifact/javax.jms/javax.jms-api/2.0.1
Javax.ejb	v.3.2	https://mvnrepository.com/artifact/javax.ejb/javax.ejb-api/3.2
Javax.ws.rs	v.2.0	https://mvnrepository.com/artifact/javax.ws.rs/javax.ws.rs-api/2.0

Chapter 2

Eclipse: Java and Java EE

2.1 Download link: Eclipse IDE

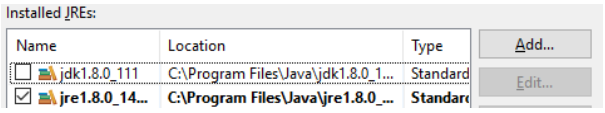
Eclipse was the IDE we used for our project. It provides a lot of features and auto-generate functionality which makes it easier for the developer.

Java EE is used when working with web applications/web services. You can add an extensions to get Java EE work in regular Eclipse or simply install Java EE as well. The link below gives access to downloads of both Eclipse for Java and Java EE. When this document was written Eclipse Oxygen is the newest version of Eclipse.

Download Eclipse IDE for Java og Java EE developers:

- https://www.eclipse.org/downloads/packages/release/Oxygen/2

Remember to have the latest version of Java, and to setup Eclipse to use `jdk` instead of `jre`. This is needed when you start to run your project with Maven, which will be explained later in this document. In your Eclipse IDE, do the following:
Choose `Window -> Preferences ->`
In the new Window, click on pane: `Java -> Installed JREs`
You should see something like this:

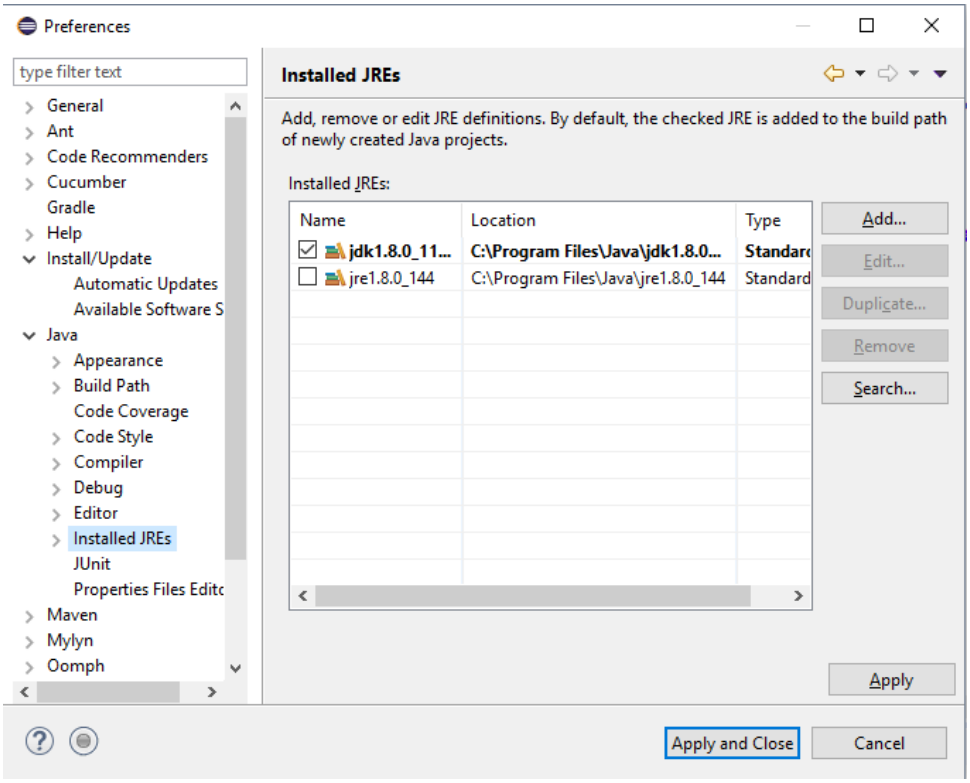


If this is not the case, click `Add`, choose `Standard VM` as JRE type and navigate to:
`Program Files -> Java ->` and click on the appearing `jdk` folder.

If you do not have any Java jdks installed, please download them from this website
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
and go through the above steps again.

When these steps are completed, you should have something like this (possibly with a newer version of

JRE/JDK):



Click `Apply and Close` and refresh your project.

2.2 Additional Eclipse packages

In Eclipse, we also installed several packages. Below is a list, this is not mandatory, but a good idea:

These can be added from `Help -> Marketplace` and search for the package.

Git - working with Git directly in Eclipse IDE, easy clone of repos and push/pull
m2e - maven to eclipse (for creating Maven projects)
Cucumber - for creating Cucumber tests

Chapter 3

Jenkins

Link to guide for installation and configuration of Jenkins on server through ssh, as provided during lectures is found here: https://goo.gl/zEiM5D. Modifications made by the group can be seen below. We recommend that you use the guide below, as some of the steps from the original source are misleading.

3.1 Introduction to Jenkins

Jenkins is an automation server to automate development processes with continuous integration. This allows you to run tests on a non-developer machine automatically every time someone pushes new code into the source repository.[1]

We installed Jenkins in the remote server `kolkata2.compute.dtu.dk`, the 2GB machine, that can be accessed at `http://139.59.136.18:8080`. We have created a user in our repository named "Jenkins" and in our project configurations, we added our repository from Jenkins interface. Builds are triggered every 5 minute and if there is a change in the source code, Jenkins pulls the latest commit and builds the project. Jenkins project configuration is detailed in Chapter 3.3.

Step 1-5 should only be created once either when configuring Jenkins or when configuring Wildfly. As Jenkins will be the first configuration task, it would be smart to do it here.

3.2 Initial setup

1. On the remote server (VM)
   Add a user to a group (to monitor the resources):
   `sudo groupadd <groupname>`
   `sudo useradd --create-home --shell /bin/bash --groups <groupname> user`.
2. If Java is not downloaded, get version 8 by:
   `sudo apt-get install oracle-java8-installer`

   Verify with the command:
   `java -version`
   `javac -version`

[1] https://www.quora.com/What-is-Jenkins-When-and-why-is-it-used

3. Configure the `$JAVA_HOME` variable
   Get the path of the installed java using:
   `sudo update-alternatives --config java`

   Get the path before
   `/jre/bin/java`

   use `sudo nano /etc/environment` and create a variable
   `$JAVA_HOME="YOUR_PATH"` (path found before)

4. Reload the file using: `source /etc/environment`
   Test it by using: `echo $JAVA_HOME`

5. Add the path variable in the user's `.bashrc` file
   Export `PATH$JAVA_HOME/jre/bin`: `$PATH`
   (the `/jre/bin` part depends on the result of `--config java` in point 3, but finally the path should point to java).
   Verify it by(so that .bashrc does not break the entire system): `source ~/.bashrc`

6. Downloading and installing Jenkins:
   Add the key to your system by:
   `wget -q -O - http://pkg.jenkins-ci.org/debian/jenkins-ci.org.key | sudo apt-key add`
   Add the URL of jenkins binary to the sources list by:

   ```
   sudo echo "deb http://pkg.jenkins-ci.org/debian binary/" >
   /etc/apt/sources.list.d/jenkins.list
   ```
   If the above does not work then try:

   ```
   sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian binary/ >
   /etc/apt/sources.list.d/jenkins.list'
   ```

7. Update the system and install Jenkins:
   ```
   sudo apt-get update
   sudo apt-get install jenkins
   ```

8. Usually jenkins runs automatically else try:
   `sudo /etc/init.d/jenkins start`

   To stop it:
   `sudo /etc/init.d/jenkins stop`

9. Jenkins open's up at port `8080`, so find the IP address of the machine to connect to the port:
   Get the ip address by running: `/sbin/ifconfig | grep -A 2 "eth0"` (or the ip address for `eth0` (by just running `/sbin/ifconfig`))
   In the local browser try: `ip_address:8080` (`http://host_ip:8080/`)(Remember to put the found IP address)

10. **Follow the instructions and at the end the screen freezes, this can be overcome by modifying the contents of** `/var/lib/jenkins/secrets/<initialAdminPassword>` **before first contact with the browser**.

11. In Eclipse remember to get the **m2e** extension if you have not already.

12. Remember that compilers are available in jenkins/global for configuration and they have an option of either being installed or getting the system path variable to know where it is installed. We can also specify the path.

3.3 Project configuration

Adding our 4 projects to Jenkins and configuration of the project includes following steps. Step 1 and 2 will be same for all projects. Step 3 will be only different for **KolkataTest** and Step 4 will be different for all projects.

1. Create a `New Item` in your Jenkins interface and select `Freestyle Project`.

2. In the `Source Code Management` section, select `Git` and enter the Git repository link with credentials. (In our repository *username: Jenkins, password: Jenkins*) Select master branch to build.

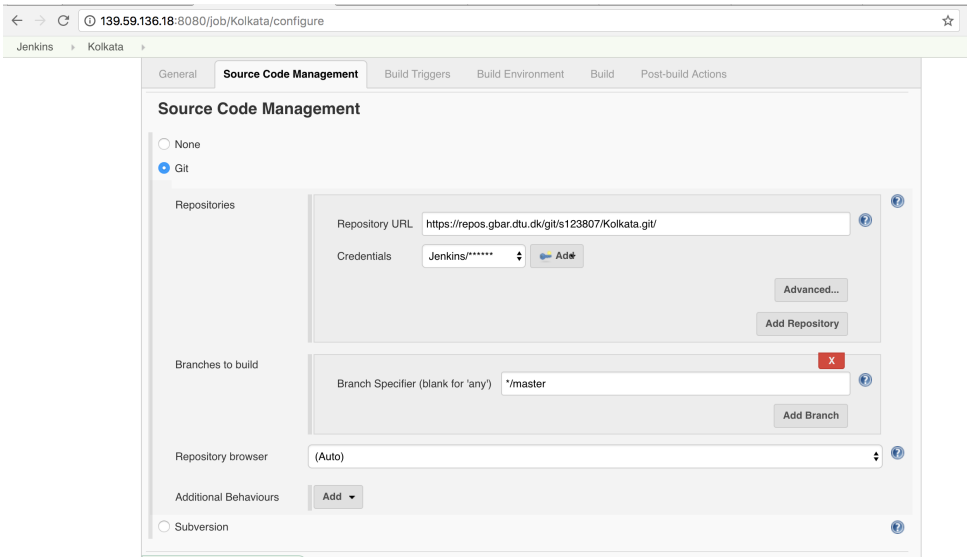


Figure 3.1: Jenkins Source Code Management configuration

3. Build Triggers

   (a) **Kolkata, Mobile Simulator, Merchant Simulator:**
   In the `Build Triggers` section, select `Poll SCM` and enter following poll script: `H/5 * * * *`. This script allows builds to be triggered in every 5 minutes and if there is a change in the source code, Jenkins pulls the latest commit and builds the project.

   (b) **KolkataTest:**
   In the `Build Triggers` section, select `Build after other projects are built` and enter 3 projects: Kolkata, Mobile Simulator, Merchant Simulator. This allows builds to be triggered after 3 projects are built.

4. Build

(a) **Kolkata:**
In the `Build` section, add a build step with following build script: "',

```
cd Kolkata
mvn clean package
"scp -i ~/kolkata target/Kolkata.war ...
root@kolkata4.compute.dtu.dk:~/wild*/standalone/deployments"
```

This script will create the .war file. The artifact is copied into the 4GB virtual machine to be deployed.

(b) **MobileSimulator:**
In the `Build` section, add a build step with following build script: "',

```
cd MobileSimulator
mvn clean package
```

This script will create the .jar file so that we can see if there is a problem with our Mobile Simulator.

(c) **MerchantSimulator:**
In the `Build` section, add a build step with following build script: "',

```
cd MerchantSimulator
mvn clean package
```

This script will create the .jar file so that we can see if there is a problem with our Merchant Simulator.

(d) **KolkataTest:**
In the `Build` section, add a build step with following build script: "',

```
cd MerchantSimulator
mvn clean install
cd ../MobileSimulator
mvn clean install
cd ../KolkataTest
mvn clean test
```

This script will create the 2 .jar files which are dependecies for KolkataTest. Then it will test KolkataTest project which is an outside project that connects to our remote server and invokes two simulators.

After configuration of the project you can see the project status in Jenkins interface as in Figure 3.2.

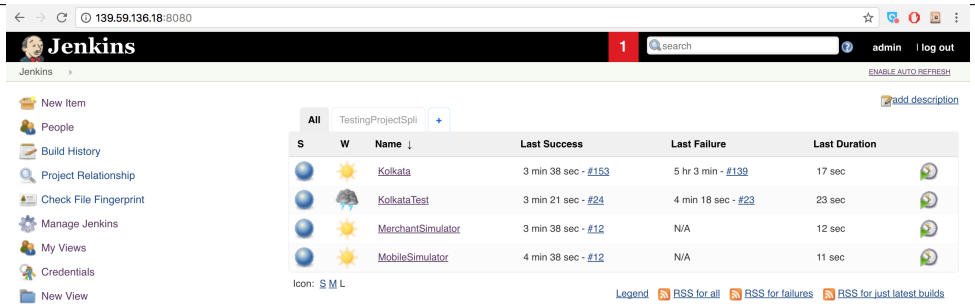


Figure 3.2: Jenkins home page

3.4 Andon plugin

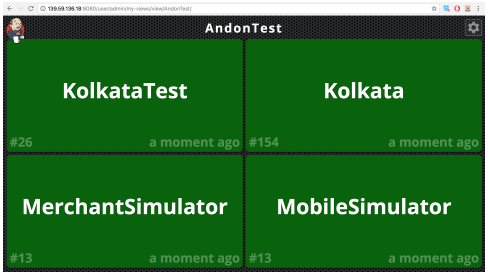


Figure 3.3: A screen shot of the chosen andon provided by the Build Monitor View plugin

In order to show our project status with an andon, we installed the `Build Monitor View`[2] you can get to the plugins page by navigating from `Manage Jenkins -> Manage Plugins` and creating it as a new view by clicking `New View` in the left-hand pane. This will open the create view page where you can add a name to your new view and then choose `Build Monitor View` as your view type in order to create the view with the new plugin. The plugin is a browser view showing the project status and the comment associated with the latest commit – there are a lot of different configurations, check them out.

[2] https://www.sharelatex.com/7316753753zfgsvykcsyws

Chapter 4

Wildfly

Link to guide for installation and configuration of Wildfly as provided during lectures is found here: https://goo.gl/FiW76X.
Modifications made by the group can be seen below, in the Section 4.2.

We installed Wildfly on the remote server `kolkata4.compute.dtu.dk`, the 4GB machine, that can be accessed at `http://159.89.31.4:8080`.

This installation guide is based upon commands in Unix bash. If you are working on a Windows machine, either run Putty or connect via a virtual machine.

4.1 Introduction to Wildfly

Wildfly is an application server developed by Red Hat. It is an extremely lightweight and powerful implementation of the Java Enterprise Edition.[1] In this project Wildfly running on a virtual machine will serve as our server.

Step 1-5 should only be created once either when configuring Jenkins or when configuring Wildfly. As Jenkins will be the first configuration task, it should have already been taken care of.

4.2 Configuration of Wildfly on remote server

1. On the remote server (VM)
   Add a user to a group (to monitor the resources):
   `sudo groupadd <groupname>`
   `sudo useradd --create-home --shell /bin/bash --groups <groupname> user`.
2. If Java is not downloaded, get version 8 by:
   `sudo apt-get install oracle-java8-installer`

   verify with the command:
   `java -version`
   `javac -version`

[1] https://docs.jboss.org/author/display/WFLY10/Documentation

3. Configure the `$JAVA_HOME` variable
   Get the path of the installed java using:
   `sudo update-alternatives --config java`

   get the path before
   `/jre/bin/java`

   use `sudo nano /etc/environment` and create a variable
   `$JAVA_HOME="YOUR_PATH"` (path found before)

4. Reload the file using: `source /etc/environment`
   test it by using: `echo $JAVA_HOME`

5. Add the path variable in the user's `.bashrc` file
   export `PATH$JAVA_HOME/jre/bin`: `$PATH`
   (the `/jre/bin` part depends on the result of `--config java` in point 3, but finally the path should point to java).

   Verify it by(so that .bashrc does not break the entire system): `source ~/.bashrc`

6. Download wildfly from the latest repository using wget http://....(site of the tarball (`.gzip`-file)).
   Unpack the tarball with `tar -xvfz <filename>`.
   `x` - extract
   `v` - verbose
   `f` - file
   `z` - gzip
   If this is not working, try removing the `z` from `-xvfz`

   run the `standalone.sh` in the `wildfly*/bin` directory to make sure there are no errors.

7. Wildfly has a security mechanism that localizes it to its local host. Disable it by going to:
   `~/wildfly*/standalone/configuration/standalone.xml`

   replace the entire line of the `inet-address` with
   `<any-address/>`

   Note! This has to be done in several files!
   Find all files with a `grep` bash command localizing all files where the home IP address is set (`127.0.0.1`).

   Protip, use something like this
   `grep -rnw '127.0.0.1'`
   `-r` recursive
   `-n` line number
   `-w` match whole word

   Stear clear of the `<wsdl-*>` instances containing the IP address, these should stay unchanged.

8. Run Wildfly in the browser with command
   `https://ipaddress:port` (remember to input IP address of host)
   If Wildfly is set up correctly, on port `8080` you should see something like the following:

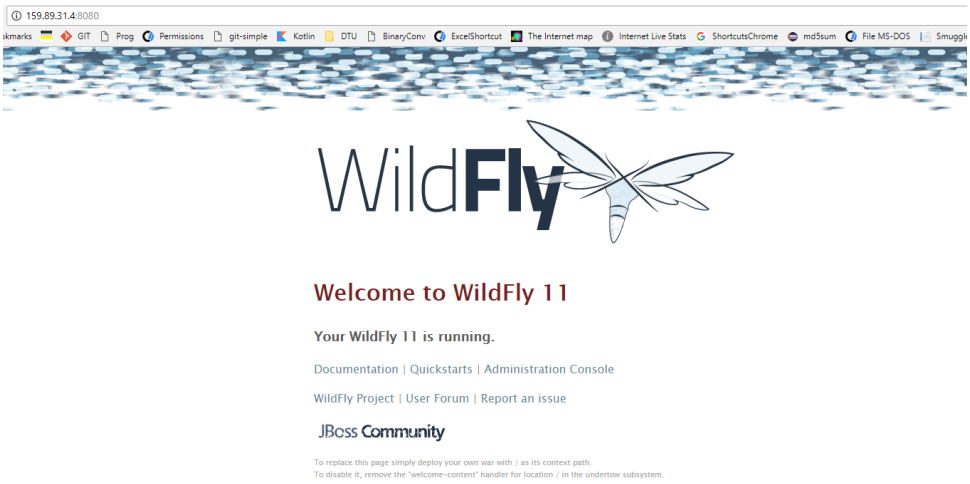


Figure 4.1: Response in browser when Wildfly is running successfully.

4.2.1 Setting up connection factories

Wildfly has a Management realm listening on port `9990`. From here you can create connection factories which are needed when added Web Applications and Message Driven Beans to your project. Follow this step-by-step screen shot guide.
Navigate to the Configuration page:

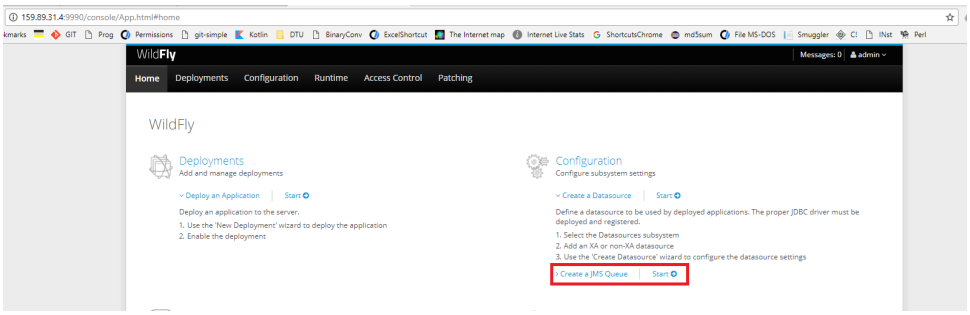


Figure 4.2: Navigation to the configuration page

Choose the correct filters:
`Subsystems -> Messaging Active JQ -> Messaging Provider -> Connections`, as indicated below

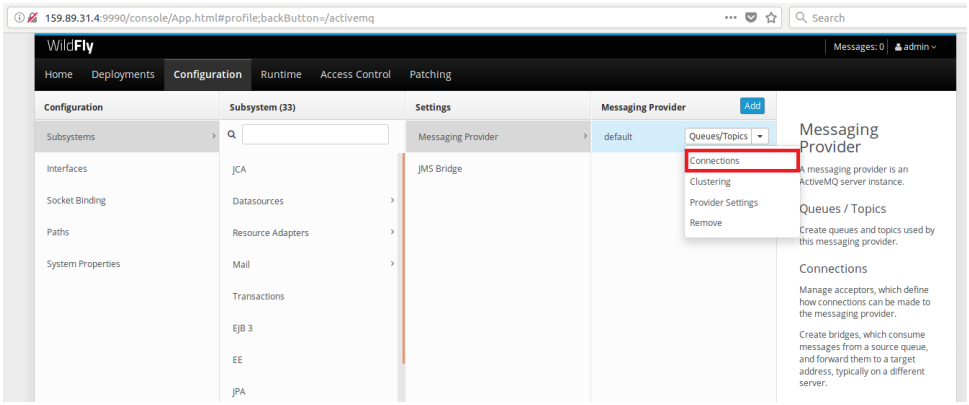


Figure 4.3: Navigating to the connection factory settings

After completing the above steps and choosing `Connections` from the left-side pane, you should see the following:

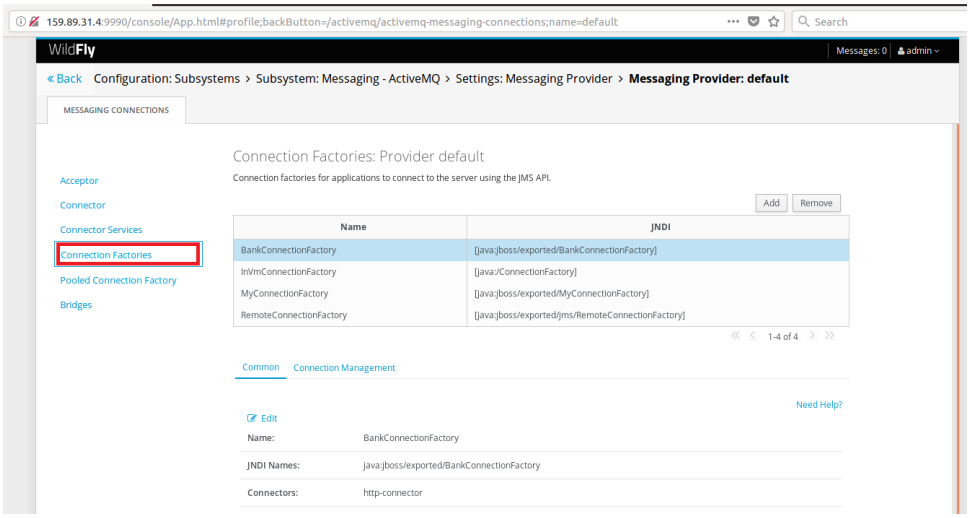


Figure 4.4: Setting up a connection factory

By choosing `Add` you will be presented with the view for adding a new Connection Factory.
Our connection factory is as follows:
Name: `MyConnectionFactory`
JNDI Name: `java:jboss/exported/MyConnectionFactory`
Connector: `http-connector`

4.2.2 Setting up queues

Wildfly has a Management realm listening on port `9990`. From here you can create queues which are needed when adding Web Applications and Message Driven Beans to your project. Follow this step-by-step screen shot guide.

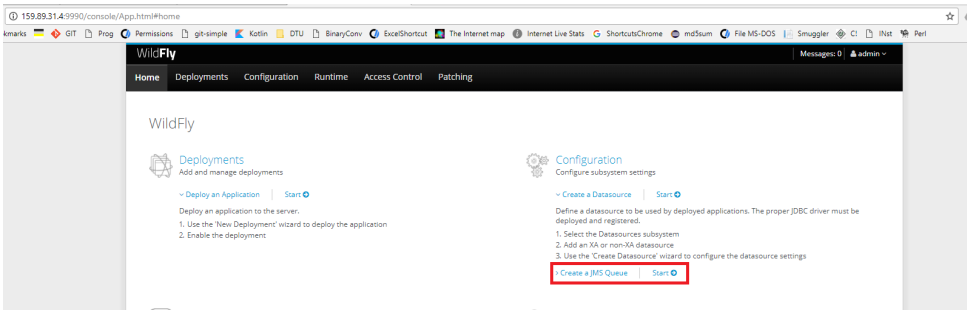


Figure 4.5: Navigation to the configuration page

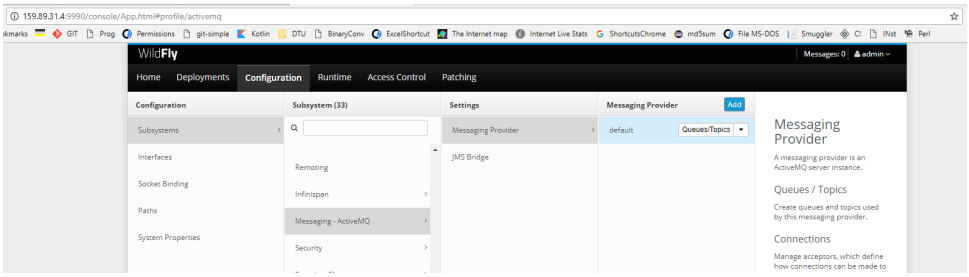


Figure 4.6: Creating a new queue: steps to get to the correct management page

The `Queue/Topics` choice is default, click it to activate it and move to the next page.

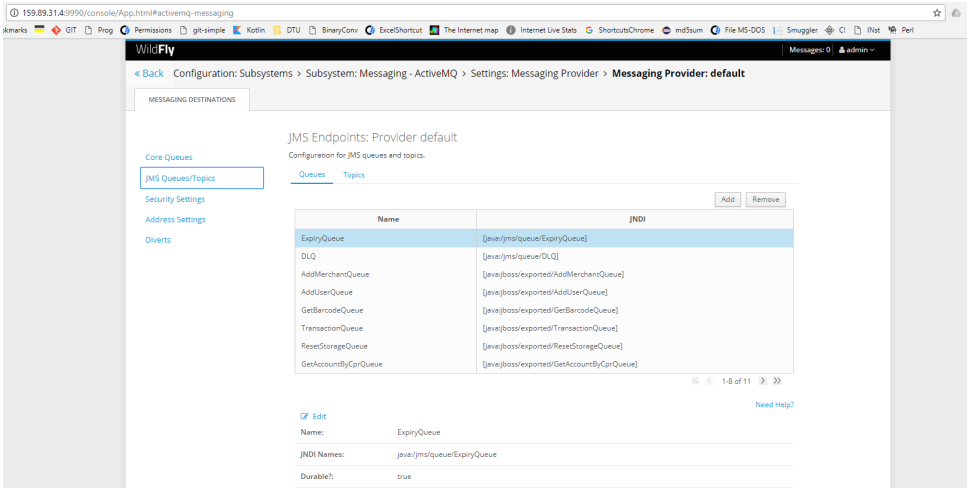


Figure 4.7: Adding the queue

To add the queue, click Add and fill out Name, JNDI and keep the Durable checkbox checked. The format of the queue to be added should look similar to those seen in the picture. Make sure you get the name and JNDI right. Our queues can be seen in Figure 4.8.

Name	JNDI
ExpiryQueue	[java:/jms/queue/ExpiryQueue]
DLQ	[java:/jms/queue/DLQ]
AddMerchantQueue	[java:jboss/exported/AddMerchantQueue]
AddUserQueue	[java:jboss/exported/AddUserQueue]
GetBarcodeQueue	[java:jboss/exported/GetBarcodeQueue]
TransactionQueue	[java:jboss/exported/TransactionQueue]
ResetStorageQueue	[java:jboss/exported/ResetStorageQueue]
GetAccountByCprQueue	[java:jboss/exported/GetAccountByCprQueue]
CheckMerchantQueue	[java:jboss/exported/CheckMerchantQueue]
CheckUserQueue	[java:jboss/exported/CheckUserQueue]
TransferBankQueue	[java:jboss/exported/TransferBankQueue]

Figure 4.8: Our queues

Add as many queues as you like, make sure to get the names correct. As they are added here they will be directly accessible in your project.

4.2.3 Disabling security

In order to use JMS on Wildfly, we needed to disable security. Follow this step-by-step screen shot guide from the management port `localhost:9990`.

Navigate to the Configuration page:

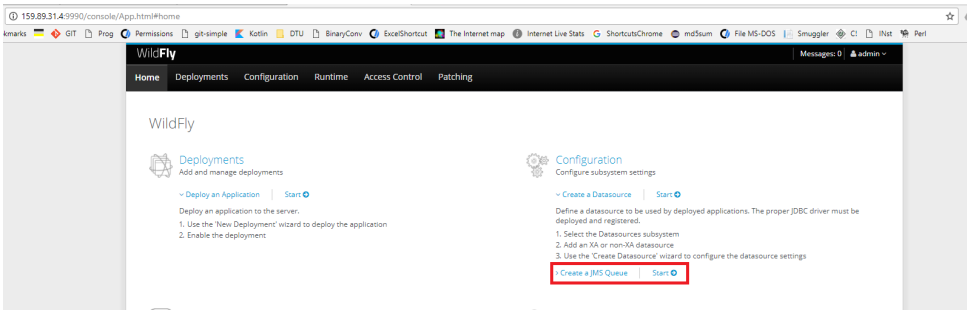


Figure 4.9: Navigation to the configuration page

Choose the correct filters:
`Subsystems -> Messaging Active JQ -> Messaging Provider -> Provider Settings`, as indicated below

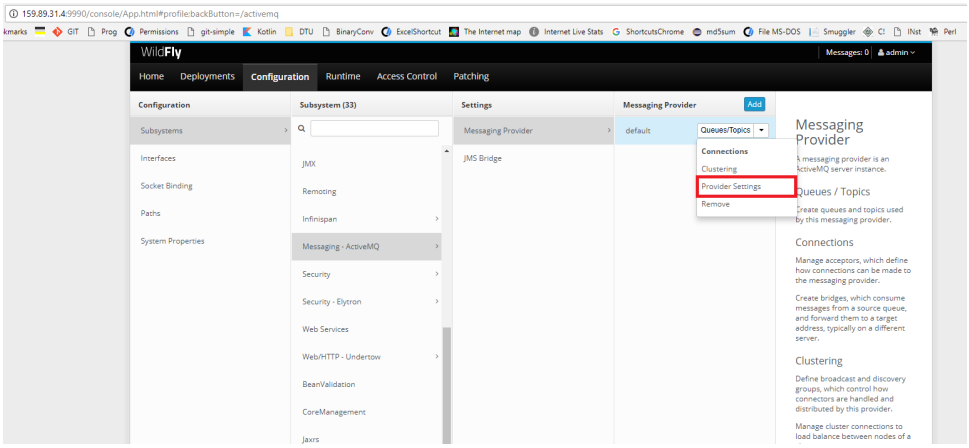


Figure 4.10: Navigating to the connection factory settings

A pop-up within the browser window will open. Choose the `Security` tab and edit the `Security enabled` field such that it looks like the screen shot below:

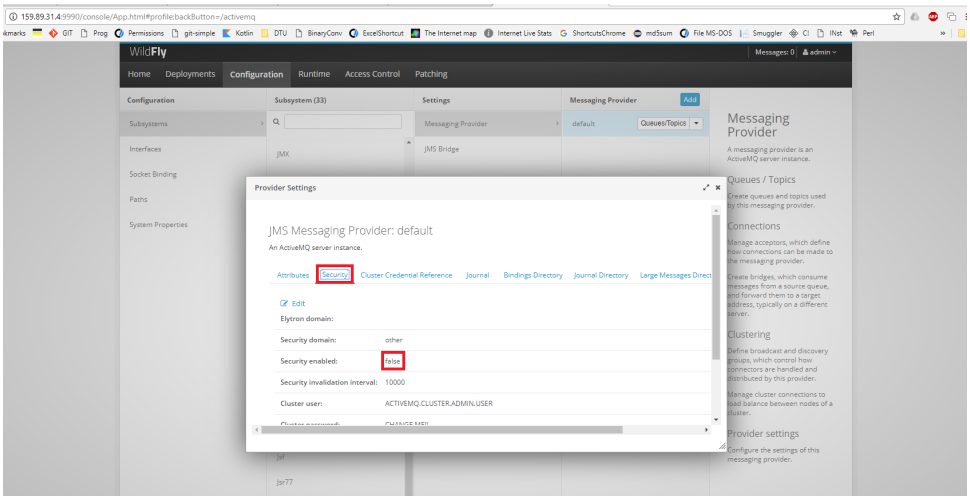


Figure 4.11: Navigating to the connection factory settings

Refresh the page to check that the new provider settings has been saved to the system configurations.