Integration Knowledge Kit Developer Journal

RedHat JBoss EAP 7.1.0

A developer's journal of lessons learned and metrics to compare developer productivity and performance costs. The journal explores why platform choices count – covering the planning, expertise, and knowledge required to quickly and flexibly build and deploy highly scalable applications.

Index

Document Purpose and Brief Introduction to project scope

Introduction to JBoss

Difficulties faced in using JBoss

System and tools requirements

Required JBoss dependencies

Creation of Initial project

Build & Run the initial project

Creation of Controllers

Serving Static and Web pages

Supported Template Engines

Connection with Data resource

Application Level Security & authentication

Configuring Web Security

JBoss Rest Services (RestEasy JaxRS)

JBoss Messaging Services

Application start up configuration

Adding scheduler tasks

DevOps

Containers and Clouds

Adoption of IOT

Making Changes

Versioning

Conclusions

Document Purpose and Brief Introduction to project scope

This is a developer's journal where developers are going to write their experience installing and using JBoss t tools for developing the use case application with producer stack functionalities like Business Logic & Governance, Asynchronous processes and brokers, Big data distribution, Social Interaction, Front end computing, DevOps, Containers & Clouds and IOT.

For this purpose the following steps will be followed:

- Installation of Tools and Environment Setup
- Creation of basic JBoss project
- Goes on ……

Introduction to JBoss EAP

Red Hat JBoss Enterprise Application Platform 7 (JBoss EAP) is a middleware platform built on open standards and compliant with the Java Enterprise Edition 7 specification. It integrates WildFly Application Server 10 with high-availability clustering, messaging, distributed caching, and other technologies.

JBoss EAP includes a modular structure that allows service enabling only when required, improving startup speed.

The management console and management command-line interface (CLI) make editing XML configuration files unnecessary and add the ability to script and automate tasks. While you can still edit the XML files manually it's not suggested to do so.

JBoss EAP provides two operating modes for JBoss EAP instances: standalone server or managed domain. The standalone server operating mode represents running JBoss EAP as a single server instance. The managed domain operating mode allows for the management of multiple JBoss EAP instances from a single control point.

In addition, JBoss EAP includes APIs and development frameworks for quickly developing secure and scalable Java EE applications.

Some of the new Features of JBoss EAP 7 over 6 are

- Support for Java EE 7 and Java SE 8
- Optimized for container and cloud deployments
- Enhanced administration and management
- Undertow - improved scalability and performance
- Upgrade support featuring interoperability

Below are the list of products and tools offered by RedHat for the developers,

ACCELERATED DEVELOPMENT AND MANAGEMENT

Red Hat JBoss Data Grid
Red Hat JBoss Enterprise Application Platform
Red Hat JBoss Web Server

DEVELOPER TOOLS

Red Hat Application Migration Toolkit

Red Hat Development Suite
Red Hat JBoss Developer Studio

INTEGRATION AND AUTOMATION
Red Hat JBoss AMQ
Red Hat JBoss BRMS
Red Hat JBoss BPM Suite
Red Hat JBoss Data Virtualization
Red Hat JBoss Fuse

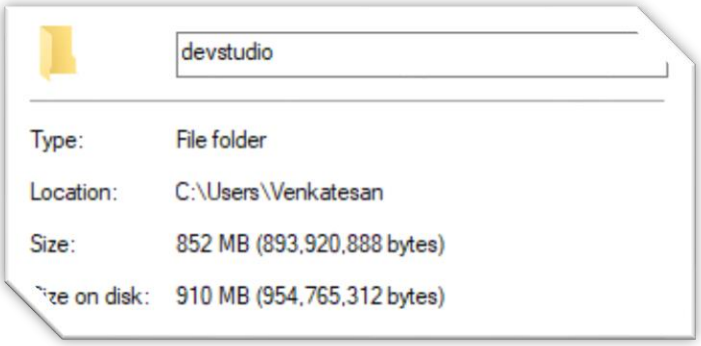
MOBILE
Red Hat Mobile Application Platform

CLOUD
Red Hat OpenShift Container Platform

LANGUAGES AND COMPILERS
OpenJDK

Difficulties faced in using JBoss

1. JBoss Developer Studio setup files are huge in size.



2. JBoss EAP and Developer Studio seems heavy weight considering the development of our usecase application and its features.
3. JBoss Admin console runs in 9990, So it's very common for the developers in windows to get the below error,
    - Address already in use: 9990, because nvidia Device drivers installed in windows are running in port 9990. So, We have only option to change the port number of JBoss Admin.
4. JBoss AS 7 is neat but the documentation is still quite lacking (and error messages not as useful as they could be)
5. Maven configuration and dependency errors
6. 403 forbidden error
    - Make sure, the Deployment Assembly is configured correctly
7. As MongoDB is an no-sql DB, there is no official JDBC driver. So, bean configuration and JBoss datasource properties like driverClassName, url cannot be configured. For that reason, We are forced to use Hibernate OGM or the native MongoDB libraries

8. java.util.zip.ZipException: invalid END header (bad central directory offset)
9. Not able to use “DatabaseServerLoginModule” for MongoDB
10. AsynchronousDispatcher error while using RestEasy
11.

System and tools requirements

- JDK 1.8 or later
- JBoss EAP 7.1.0
- Gradle 2.3+ or Maven 3.1.1+
- JBoss Studio

Required JBoss dependencies

```
<dependencyManagement>
	<dependencies>
		<dependency>
			<groupId>org.hibernate.ogm</groupId>
			<artifactId>hibernate-ogm-bom</artifactId>
			<type>pom</type>
			<version>${ogm.version}</version>
			<scope>import</scope>
		</dependency>
	</dependencies>
</dependencyManagement>

<dependencies>
	<dependency>
		<groupId>org.springframework</groupId>
		<artifactId>spring-core</artifactId>
		<version>${spring-framework.version}</version>
	</dependency>
	<dependency>
		<groupId>org.springframework</groupId>
		<artifactId>spring-webmvc</artifactId>
		<version>${spring-framework.version}</version>
	</dependency>
	<dependency>
		<groupId>org.springframework</groupId>
		<artifactId>spring-orm</artifactId>
		<version>${spring-framework.version}</version>
	</dependency>
	<dependency>
	  <groupId>org.springframework</groupId>
	  <artifactId>spring-jdbc</artifactId>
	  <version>${spring-framework.version}</version>
	</dependency>

	<!-- https://mvnrepository.com/artifact/org.springframework/spring-messaging -->
	<dependency>
	  <groupId>org.springframework</groupId>
	  <artifactId>spring-messaging</artifactId>
	  <version>${spring-framework.version}</version>
	</dependency>

	<!-- https://mvnrepository.com/artifact/org.springframework/spring-websocket -->
	<dependency>
	  <groupId>org.springframework</groupId>
```

```
                    <artifactId>spring-websocket</artifactId>
                    <version>${spring-framework.version}</version>
                </dependency>


                <!-- https://mvnrepository.com/artifact/org.hibernate.ogm/hibernate-ogm-mongodb -->
                <dependency>
                    <groupId>org.hibernate.ogm</groupId>
                    <artifactId>hibernate-ogm-mongodb</artifactId>
                </dependency>

                <!-- https://mvnrepository.com/artifact/org.mongodb/mongo-java-driver -->
                <dependency>
                    <groupId>org.mongodb</groupId>
                    <artifactId>mongo-java-driver</artifactId>
                    <version>3.6.2</version>
                </dependency>
                <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->
                <dependency>
                    <groupId>com.fasterxml.jackson.core</groupId>
                    <artifactId>jackson-databind</artifactId>
                    <version>2.9.4</version>
                </dependency>
                <!-- https://mvnrepository.com/artifact/org.json/json -->
                <dependency>
                    <groupId>org.json</groupId>
                    <artifactId>json</artifactId>
                    <version>20171018</version>
                </dependency>

                    <dependency>
            <groupId>org.jboss.resteasy</groupId>
            <artifactId>resteasy-servlet-initializer</artifactId>
            <version>3.0.19.Final</version>
            <scope>provided</scope>
            <exclusions>
              <exclusion>
                <artifactId>resteasy-jaxrs</artifactId>
                <groupId>org.jboss.resteasy</groupId>
              </exclusion>
            </exclusions>
        </dependency>
        </dependencies>
```

Initial Hours

10 hours to draft the Use case application, breakdown the definitions and to do the project planning.

16 hours to create the User Interface required for this use case application

5 hours to architect the service level calls and messaging services

2 hours for normalizing the Database according to the use case application.

4 hours to setup the dev environments and setup required software's

Creation of Initial project

Initial setup and server configuration details are followed as in below links,

https://developers.redhat.com/quickstarts/eap/helloworld/

https://developers.redhat.com/products/eap/hello-world/

After finishing the setup and configuring the maven repositories,

From Redhat central page, Select "helloworld-rs/"

Click Finish. This downloads the helloworld application and its Maven dependencies.

Once all the dependencies are downloaded, a dialog appears to tell you that your project is ready. Click Finish.

After clicking on finish, It took long time to get the dialog appear. Meanwhile, there are no notifications for the developers to understand the background tasks

JBoss Admin console runs in 9990, So it's very common for the developers in windows to get the below error,
- Address already in use: 9990, because nvidia Device drivers installed in windows are running in port 9990. So, We have only option to change the port number of JBoss Admin.

Edit the below configuration file
`JBOSS_HOME/standalone/configuration/standalone.xml`

or if you are using dev studio/eclipse, follow the below steps
- Click on the 'Server' View
- Expand the instance of JBoss you wish to run on (e.g. JBoss AS 7.1)
- Expand XML Configuration
- Expand Ports
- Right click on JBoss Web
- Select 'Change Value', and change the port number (e.g. 8082)

Let's create a basic Maven project and add features on top of it,

In JBoss Developer Studio,

choose File –> New -> Maven Project. If not found Maven Project, you select line Others. In the window that appears, you find maven project in search box, then click Next button.

Check on Use default Workspace location and click Next button, filter and select Artifact is "maven-archetype-webapp", and Next.

Details of the project:

Group Id: redhat-jboss -> Group ID will be used for the packaging of the future
Artifact Id: redhat-jboss -> name of project will show in eclipse
Version: 0.1-SNAPSHOT
Package: com.pushtotest.jboss

Click Finish button.

error : could not resolve archetype org.apache.maven.archetypes …

fixed by the below steps:

Open Window > Preferences
Select Maven > Archetypes
Click “Add Remote Catalog” and add the following:
Catalog File: http://repo1.maven.org/maven2/archetype-catalog.xml
Description: maven catalog

MVC framework is used to separate the data access layer, business logic code and the graphical user interface that has to be defined and designed to let the user interact with the application

We have only 2 options here, Either to use Spring MVC or JBoss Seam Framework,

Below are some of the reasons for selecting Spring MVC to develop the usecase application,

- Spring is light weight and It minimally invasive development with POJO.
- Spring achieves the loose coupling through dependency injection and interface based programming.
- Declarative programming through aspects and common conventions.
- Boilerplate reduction through aspects and templates.

Let's modify the project properties,

Select Project->Properties
Select Project facets, only check Dynamic Web Module, Java, Javascript.
Select Runtime tab in right and check jboss 7.1 Runtime
Select Java Build Path, in Source tab -> uncheck Allow output folders for source folders.

Edit pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>com.pushtotest.jboss</groupId>
	<artifactId>redhat-jboss</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>redhat-jboss</name>
	<url>http://maven.apache.org</url>

	<properties>
		<spring-framework.version>5.0.3.RELEASE</spring-framework.version>
		<ogm.version>5.0.0.CR1</ogm.version>
	</properties>

	<dependencies>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-core</artifactId>
			<version>${spring-framework.version}</version>
		</dependency>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-webmvc</artifactId>
			<version>${spring-framework.version}</version>
		</dependency>
	</dependencies>
	<build>
		<finalName>redhat-jboss</finalName>
		<resources>
```

```
                <resource>
                <directory>src/main/webapp</directory>
                </resource>
        </resources>
                <plugins>
                        <plugin>
                                <groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-compiler-plugin</artifactId>
                                <version>2.3.2</version>
                                <configuration>
                                        <source>1.8</source>
                                        <target>1.8</target>
                                        <encoding>UTF-8</encoding>
                                        <failOnMissingWebXml>false</failOnMissingWebXml>
                                        <warName>redhat-jboss</warName>
                                </configuration>
                        </plugin>
                </plugins>
        </build>
</project>
```

Create web.xml in WEB-INF folder

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
 <display-name>hello_world</display-name>

 <servlet>
    <servlet-name>redhat-jboss</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
       <param-name>contextConfigLocation</param-name>
       <param-value>/WEB-INF/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <context-param>
                <param-name>contextConfigLocation</param-name>
                <param-value></param-value>
  </context-param>

  <servlet-mapping>
    <servlet-name>redhat-jboss</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>

  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
</web-app>
```

Create servlet-context.xml in WEB-INF folder

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
```

```
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
context.xsd
    http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd">

   <context:component-scan base-package="com.pushtotest.jboss" />
   <!-- <context:annotation-config />  -->
   <!-- Enables the Spring MVC @Controller programming model -->
   <mvc:annotation-driven/>

   <mvc:resources mapping="/static/**" location="/WEB-INF/static/" />
   <bean
                        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
                        <property name="prefix">
                                <value>/WEB-INF/templates/</value>
                        </property>
                        <property name="suffix">
                                <value>.jsp</value>
                        </property>
            </bean>

            <!-- Activates various annotations to be detected in bean classes e.g: @Autowired -->
            <context:annotation-config />
</beans>
```

create HelloWorldController.java in com.pushtotest.jboss.controllers

```
package com.pushtotest.jboss;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class HelloWorldController {

            @RequestMapping(value="/hello",method = RequestMethod.GET)
            public String HelloWorld(Model model){
                        model.addAttribute("message", "Welcome to Spring MVC");
                        return "hello";
            }
}
```

create hello.jsp file in WEB-INF/templates directory

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Hello World</title>
</head>
<body>
            <h1>${message}</h1>
</body>
</html>
```

Start the JBoss EAP server and deploy the project as mentioned above.

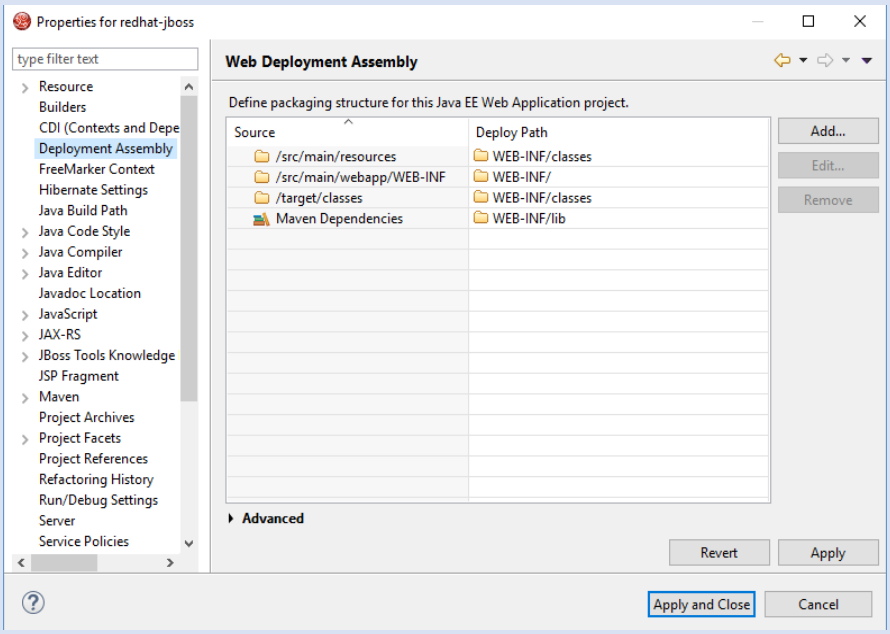
Error: Got Maven Configuration error
Solved: Deleted dependencies in lib folder & update Maven project

Error: WFLYSRV0161: Failed to get manifest for deployment
Solved: Deleted dependencies in lib folder & update Maven project

Error: Cannot change version of project facet Dynamic Web Module to 2.5.
Solved: Editing pom.xml

```
<configuration>
    <!-- Java EE 7 doesn't require web.xml, Maven needs to catch up! -->
    <failOnMissingWebXml>false</failOnMissingWebXml>
    <source>1.8</source> <!-- yours Java version -->
    <target>1.8</target>
 </configuration>
```

Error: 403 forbidden error
Solved: This error comes because the files are not placed correctly inside the war.
Make sure, the Deployment Assembly is configured correctly as below,



Build & Run the initial project

When you do project clean, War file and its classes are built automatically.

In-case if you are using Maven to build the project, run the below command

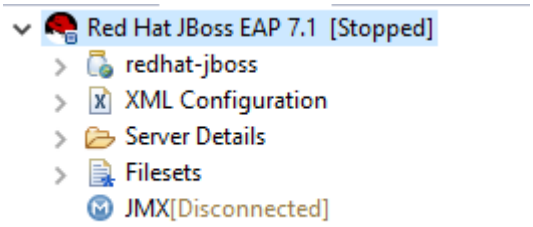
mvn clean deploy

To run the project from Developer Studio, follow the below steps,

Right click on Jboss EAP 7.1 and select “Add and Remove”

Select redhat-jboss in Available tab and Add to Configured tab, click finish button

In Server tab, click Publish to the server (Ctrl + Alt + P) to up your project to Server



Once Started, Application can be accessed through

http://localhost:8080/redhat-jboss/hello

5 hours to learn basics of jboss and create the initial project.

Creation of Controllers

In Spring's approach to building web sites, HTTP requests are handled by a controller. You can easily identify these requests by the @Controller annotation. In the following example, the Hello Controller handles GET requests for /hello

src/main/java/hello/HelloController.java

```
package hello;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class GreetingController {
   @RequestMapping("/hello")
   public String hello() {
      return "hello";
   }
}
```

There was lot of dependency errors,

It took 4 hours to configure proper debugging and to pass all the initial dependency errors

Serving static and web pages

Static pages

Create a new folder "static" inside "src/main/webapp/WEB-INF":

Edit web.xml:

```
<security-constraint>
   <web-resource-collection>
      <web-resource-name>StaticPages</web-resource-name>
      <url-pattern>/static/*</url-pattern>
```

```
  </web-resource-collection>
  <!-- OMIT auth-constraint -->
</security-constraint>
```

Edit servlet-context.xml:

```
<mvc:resources mapping="/static/**" location="/WEB-INF/static/" />
```

Web pages

In order to serve dynamic content you can choose between jsp or the template Engines. JBoss allows you to use both without any configuration files!

Access to all the static & dynamic page urls should be controlled more often, in-case if you are using Custom URL filters for security.

There are two options for template engines in JBoss,

Template Engine

- FreeMarker
- Apache Velocity

7 hours to accomplish the below tasks

- To do research on creating dynamic web pages
- Validating them by creating a standalone POC's
- Learning about FreeMarker & Velocity to understand its usage and its feasibility for the usecase application

Connection with Data resource

There are 3 ways you can connect to MongoDB in JBoss EAP

- Using Teiid Data virtualization
- Using MongoDB Java driver
- Using Hibernate OGM to map MongoDB

Let's use Hibernate OGM,

Hibernate Object/Grid Mapper (OGM) is a framework which provides Java Persistence (JPA) support for NoSQL solutions. It reuses Hibernate ORM's engine but persists entities into a NoSQL datastore instead of a relational database. This means you will be writing pure JPA code, which will be handled behind the scenes, by the OGM Engine.

ORM Maven dependencies:

```
        <dependencyManagement>
                <dependencies>
                        <dependency>
                                <groupId>org.hibernate.ogm</groupId>
```

```
                                        <artifactId>hibernate-ogm-bom</artifactId>
                                        <type>pom</type>
                                        <version>${ogm.version}</version>
                                        <scope>import</scope>
                                </dependency>
                        </dependencies>
                </dependencyManagement>

                <!-- https://mvnrepository.com/artifact/org.hibernate.ogm/hibernate-ogm-mongodb -->
                <dependency>
                        <groupId>org.hibernate.ogm</groupId>
                        <artifactId>hibernate-ogm-mongodb</artifactId>
                </dependency>
```

OGM Model Class:

```
package com.pushtotest.jboss.model;

import java.util.Random;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;


@Entity
@Table(name = "Login")
public class Login {
        @Id
        @GeneratedValue(strategy=GenerationType.IDENTITY)
        private String id;

        private String userId;
        private String password;
        private String firstName;
        private String lastName;
        private int balance;

        public int getBalance() {
                return balance;
        }
        public void setBalance(int balance) {
                this.balance = balance;
        }
        public String getId() {
                return id;
        }
        public void setId(String id) {
                this.id = id;
        }
        public String getUserId() {
                return userId;
        }
        public void setUserId(String userId) {
                this.userId = userId;
        }
```

```
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}
	public String getFirstName() {
		return firstName;
	}
	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}
	public String getLastName() {
		return lastName;
	}
	public void setLastName(String lastName) {
		this.lastName = lastName;
	}
}
```

OGM Dao Class:

```
package com.pushtotest.jboss.model;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;

import org.springframework.stereotype.Component;

import com.pushtotest.jboss.model.Login;

@Component
public class LoginDao {
	static EntityManagerFactory entityManagerFactory;

	public static EntityManagerFactory setUpEntityManagerFactory() {
		return entityManagerFactory = Persistence
				.createEntityManagerFactory("MongoJpa");
	}

	public void persist(Login user) {
		EntityManager em = setUpEntityManagerFactory()
				.createEntityManager();
		try {
			if(user != null) {
				em.getTransaction().begin();
				em.persist(user);
				em.getTransaction().commit();
			}
		} catch(Exception e) {
			e.printStackTrace();
			em.getTransaction().rollback();
		} finally {
			em.close();
		}
	}
```

```
	public Login findByUserId(String userId) {
		EntityManager em = setUpEntityManagerFactory()
					.createEntityManager();
		Login user = null;

		try {
			Query query = em.createQuery("SELECT * FROM Login where userId='" + userId + "'");
			user = (Login) query.getSingleResult();
		} catch(Exception e) {
			e.printStackTrace();
			em.getTransaction().rollback();
		} finally {
			em.close();
		}
		return user;
	}

	public Login findByUserNameAndPassword(String userId, String password) {
		EntityManager em = setUpEntityManagerFactory()
					.createEntityManager();
		Login user = null;

		try {
			Query query = em.createQuery("SELECT * FROM Login where userId='" + userId + "' and
password='" + password + "'");
			user = (Login) query.getSingleResult();
		} catch(Exception e) {
			e.printStackTrace();
			em.getTransaction().rollback();
		} finally {
			em.close();
		}
		return user;
	}

	public void updateBalance(String userId, int balance) {
		EntityManager em = setUpEntityManagerFactory()
					.createEntityManager();

		try {
			Login user = findByUserId(userId);
			if(user != null) {
			  em.getTransaction().begin();
			  user.setBalance(balance);
			  em.persist(user);
			  em.getTransaction().commit();
			}

		} catch (Exception e) {
			e.printStackTrace();
			em.getTransaction().rollback();
		} finally {
			em.close();
		}
	}
}
```

@Component is Spring annotation that tell the Spring container that we can use this class through Spring IoC (Dependency Injection).

We use JPA @PersistenceContext annotation that indicate dependency injection to an EntityManager. Spring injects a proper instance of EntityManager according to the servlet-context.xml configuration.

OGM Service Class:

```
package com.pushtotest.jboss.model;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import com.pushtotest.jboss.dao.LoginDao;
import com.pushtotest.jboss.interfaces.UserService;
import com.pushtotest.jboss.model.Login;

@Component
public class UserServiceImpl implements UserService {

	@Autowired
	private LoginDao loginDao;

	@Override
	public void save(Login user) {
		loginDao.persist(user);
	}

	@Override
	public Login findByUserId(String username) {
		return loginDao.findByUserId(username);
	}

	@Override
	public void updateBalance(String userId, int balance) {
		loginDao.updateBalance(userId, balance);
	}

	@Override
	public Login findByUserNameAndPassword(String userId, String password) {
		return loginDao.findByUserNameAndPassword(userId, password);
	}

}
```

We use Spring @Autowired annotation to inject LoginDao in our service class.

OGM Bean Configuration:

Create persistence.xml under src/main/resources/META-INF folder

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
	xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
```

```
        <persistence-unit name="MongoJpa" transaction-type="RESOURCE_LOCAL">
                <provider>org.hibernate.ogm.jpa.HibernateOgmPersistence</provider>
                <class>com.pushtotest.jboss.model.Login</class>
                <class>com.pushtotest.jboss.model.BetGames</class>
                <class>com.pushtotest.jboss.model.BetOdds</class>
                <class>com.pushtotest.jboss.model.Notifications</class>
                <properties>
                        <property name="hibernate.ogm.datastore.provider" value="mongodb" />
                        <property name="hibernate.ogm.datastore.database" value="test" />
                        <property name="hibernate.ogm.datastore.create_database"
                                value="true" />
                </properties>
        </persistence-unit>
</persistence>
```

As MongoDB is an no-sql DB, there is no official JDBC driver. So, bean configuration properties like driverClassName, url cannot be configured. For that reason, It is not possible to use bean configuration for MongoDB datasource

Solution: Use OGM provider configuration using persistence.xml as above.

Issues:

Is it possible to define datasource configuration to a MongoDB?

If not, Will it be supported in the future?

How to connect to mongoDB with JBoss EAP 7.x?

Resolution

MongoDB is not supported and there are no plans as of now to get it enlisted in our supported/certified configurations as listed below: JBoss Enterprise Application Platform (EAP) 6 Supported Configurations and JBoss Enterprise Application Platform (EAP) 7 Supported Configurations

It doesn't look like there is a compliant JDBC driver for MongoDB. MongoDB will not provide a JDBC driver due to the reasons discussed in https://jira.mongodb.org/browse/JAVA-539. There is a third party JDBC driver http://sourceforge.net/projects/mongojdbcdriver/?source=directory but MongoDB may not map too well to java.SQL.ResultSets.

However, if what is wanted is the ability to configure connection pooling and similar, rather than a SQL interface, one can use a JCA RAR. Something like this https://github.com/ebirn/mongodb-connector

Most commonly, below error occurs

java.util.zip.ZipException: invalid END header (bad central directory offset)

Solution: dependency jar files are getting corrupted some times, Delete the dependencies in lib and update the maven project

java.lang.NoSuchMethodError:
org.hibernate.boot.model.source.spi.AttributePath.isPartOfCollectionElement

Solution: Spring OGM dependency problems, Use dependency hierarchy to identify the compatible dependency jars

Got few dependency errors, took some time to know JBoss has less support on MongoDB

7 hours overall to implement the JBoss data source for MongoDB.

Application Level Security & authentication

We have to use Java EE Standard authentication module

A part of the Java EE specification is security for web and EE applications, which makes it possible both to specify declarative constraints in your web.xml (such as "role X is required to access resources at URLs "/protected/*") and to control it programatically, i.e. verifying that the user has a particular role (see HttpServletRequest.isUserInRole).

It works as follows:

Edit web.xml:

- Login configuration – primarily whether to use browser prompt (basic) or a custom login form and a name for the login realm
  - The custom form uses "magic" values for the post action and the fields, starting with j_, which are intercepted and processed by the server
- The roles used in your application
- What roles are required for accessing particular URL patterns (default: none)
- Whether HTTPS is required for some parts of the application

Tell your application server how to authenticate users for that login realm, usually by associating its name with one of the available login modules in the configuration (the modules ranging from simple file-based user list to LDAP and Kerberos support).

Edit JBoss EAP 7.1.0/ standalone/configuration/standalone.xml

```
<security-domain name="form-auth" cache-type="default">
  <authentication>
    <login-module code="com.pushtotest.jboss.security.UserSecurityService" flag="required"/>
  </authentication>
</security-domain>
```

The code attribute should contain the fully qualified name of your login module class and the security-domain's name must match the declaration in jboss-web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web>
 <security-domain>form-auth</security-domain>
 <disable-audit>true</disable-audit>
</jboss-web>
```

We need to override the UsernamePasswordLoginModule as below,

```
package com.pushtotest.jboss.security;
```

```
import java.security.acl.Group;
import java.util.Map;

import javax.security.auth.Subject;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.login.LoginException;
import javax.servlet.http.HttpServletRequest;

import org.bson.types.ObjectId;
import org.jboss.security.SimpleGroup;
import org.jboss.security.SimplePrincipal;
import org.jboss.security.auth.spi.UsernamePasswordLoginModule;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.MongoClient;
import com.pushtotest.jboss.model.Login;

/**
 * The simples username and password based login module possible,
 * extending JBoss' {@link UsernamePasswordLoginModule}.
 */
@Component
public class UserSecurityService extends UsernamePasswordLoginModule {

        @SuppressWarnings("rawtypes")
   public void initialize(Subject subject, CallbackHandler callbackHandler,
         Map sharedState,
         Map options) {
      // We could read options passed via <module-option> in standalone.xml if there were any here
      // For an example see http://docs.redhat.com/docs/en-
US/JBoss_Enterprise_Application_Platform/5/html/Security_Guide/sect-Custom_LoginModule_Example.html

      // We could also f.ex. lookup a data source in JNDI
      // For an example see
http://www.docjar.com/html/api/org/jboss/security/auth/spi/DatabaseServerLoginModule.java.html
      super.initialize(subject, callbackHandler, sharedState, options);
   }

   /**
    * (required) The UsernamePasswordLoginModule modules compares the result of this
    * method with the actual password.
    */
   @Override
   protected String getUsersPassword() throws LoginException {
         String[] userCredentials = getUsernameAndPassword();
         String userName = userCredentials[0];
         //String ePassword = userCredentials[1];
         System.out.format("Security Service: authenticating user '%s'\n", userName);

         Login user = null;

         try {
```

```
                        //user = userService.findByUserId(userName);
                        user = passwordFromDB(userName);

                        if(user != null) {
                                String password = user.getPassword();
                                System.out.format("Security Service: UserName:" + userName + " and password in DB:
'%s'\n", password);
                                HttpServletRequest request = (HttpServletRequest)
javax.security.jacc.PolicyContext.getContext("javax.servlet.http.HttpServletRequest");
                                request.getSession().setAttribute("user", user);
                                return password;
                        }

                }catch(Exception e) {
                        e.printStackTrace();
                }
        return "notValid";
    }

    private Login passwordFromDB(String userName) {
                MongoClient mongoConnection = new MongoClient("localhost");
                        DB database = mongoConnection.getDB("test");
                        DBCollection collection = database.getCollection("Login");
                        BasicDBObject query = new BasicDBObject();
                        query.put("userId", userName);

                        DBCursor cursor = collection.find(query);
                        DBObject dbObj = null;
                        Login user = null;

                        try {
                                while (cursor.hasNext()) {
                                        dbObj = cursor.next();
                                        user = new Login();
                                        ObjectId objId = (ObjectId) dbObj.get("_id");
                                        user.setId(objId.toString());
                                        user.setUserId((String) dbObj.get("userId"));
                                        user.setPassword((String) dbObj.get("password"));
                                        user.setFirstName((String) dbObj.get("firstName"));
                                        user.setLastName((String) dbObj.get("lastName"));
                                        user.setBalance((int) dbObj.get("balance"));
                                }
                        } finally {
                                mongoConnection.close();
                                cursor.close();
                        }
                        return user;
    }


    /**
     * (required) The groups of the user, there must be at least one group called
     * "Roles" (though it likely can be empty) containing the roles the user has.
     */
    @Override
    protected Group[] getRoleSets() throws LoginException {
        SimpleGroup group = new SimpleGroup("Roles");
        try {
```

```
            group.addMember(new SimplePrincipal("USER"));
        } catch (Exception e) {
            throw new LoginException("Failed to create group member for " + group);
        }
        return new Group[] { group };
    }

}
```

And Web.xml should contain the Login configuration.

```
<!-- Define the Login Configuration for this Application -->
 <login-config>
   <auth-method>FORM</auth-method>
   <realm-name>BettingApplication</realm-name>
   <form-login-config>
     <form-login-page>/login</form-login-page>
     <form-error-page>/error</form-error-page>
   </form-login-config>
 </login-config>
```

Once the Java EE authentication is added, Jboss server filters the requests of url's starting with j_. Login page should be configured as below

```
   <form method="post" action="j_security_check"
   class="col s10 m8 l6 xl6 offset-s1 offset-m2 offset-l3 center-align signin-form">
   <div class="titleText">Log in to your account</div>
   <div class="row">
      <div class="input-field col s12">
         <input type="email" name="j_username" id="userId" class="validate" />
         <label for="signin-userId" data-error="Invalid email">Email</label>
      </div>
   </div>
   <div class="row">
      <div class="input-field col s12">
         <input type="password" name="j_password" id="password"
            class="validate" /> <label for="signin-password">Password</label>
      </div>
   </div>
   <div id="signin-error" class="row"
      th:classappend="${showError} ? show : hide">
      <div class="error-field col s12">
         <label for="signin-error" ${errorMessage}"></label>
      </div>
   </div>
   <button type="submit" class="waves-effect waves-light btn"
      name="action">Sign In</button>
   <div id="signupBtn" class="signupBtn">
      <a href="registration">Sign up for an account</a>
   </div>
</form>
```

Not able to use "DatabaseServerLoginModule",

As MongoDB is an no-sql DB, there is no official JDBC driver. So, JNDI datasource properties like driverClassName, url cannot be configured. For that reason, We are forced to use either UsernamePasswordLogin Module or the Custom Login Module

5 Hours to do the above implementations

Configuring Web Security

Security is a fundamental part of any enterprise application. You need to be able to restrict who is allowed to access your applications and control what operations application users may perform.

The J2EE specifications define a simple role-based security model for EJBs and web components. The JBoss component framework that handles security is the JBossSX extension framework. The JBossSX security extension provides support for both the role-based declarative J2EE security model and integration of custom security via a security proxy layer.

The default implementation of the declarative security model is based on Java Authentication and Authorization Service (JAAS) login modules and subjects. The security proxy layer allows custom security that cannot be described using the declarative model to be added to an EJB in a way that is independent of the EJB business object.

It has to be configured from the Application level. Considering the security levels of the usecase application, entries in web.xml should look like,

```
<!-- Define a Security Constraint on this Application -->
<security-constraint>
 <web-resource-collection>
   <web-resource-name>error</web-resource-name>
   <url-pattern>/error</url-pattern>
 </web-resource-collection>
 <!-- OMIT auth-constraint -->
          </security-constraint>
          <security-constraint>
 <web-resource-collection>
   <web-resource-name>registration</web-resource-name>
   <url-pattern>/registration</url-pattern>
 </web-resource-collection>
 <!-- OMIT auth-constraint -->
          </security-constraint>
           <security-constraint>
 <web-resource-collection>
   <web-resource-name>StaticPages</web-resource-name>
   <url-pattern>/static/*</url-pattern>
 </web-resource-collection>
 <!-- OMIT auth-constraint -->
          </security-constraint>
<security-constraint>
 <web-resource-collection>
  <web-resource-name>BettingApplication</web-resource-name>
  <url-pattern>/*</url-pattern>
 </web-resource-collection>
 <auth-constraint>
   <role-name>USER</role-name>
 </auth-constraint>
</security-constraint>
```

and the security roles should be configured as,

```
 <!-- Security roles referenced by this web application -->
<security-role>
 <description>
  The role a user needs to be allowed to log in to the application
 </description>
 <role-name>USER</role-name>
</security-role>
```

1 hour to apply the web security

JBoss Rest Services (RestEasy jax-RS)

We are having more options to implement Rest services in JBoss environment, Some of the ways are to use

- Jersey jax-RS
- Spring RestController
- RestEasy jax-RS
  and more.

In Spring's approach to building RESTful web services, HTTP requests are handled by a controller.

These components are easily identified by the @RestController annotation, and the RestController classes handles GET requests for various rest calls

Let's consider using RestEasy, as it is already built in with JBoss EAP

Edit pom.xml

```
        <dependency>
            <groupId>org.jboss.resteasy</groupId>
            <artifactId>resteasy-servlet-initializer</artifactId>
            <version>3.0.19.Final</version>
            <scope>provided</scope>
            <exclusions>
              <exclusion>
                <artifactId>resteasy-jaxrs</artifactId>
                <groupId>org.jboss.resteasy</groupId>
              </exclusion>
            </exclusions>
        </dependency>
```

Configure listener and servlet to support RESTEasy

```
<!-- Auto scan REST service -->
  <context-param>
                <param-name>resteasy.scan</param-name>
                <param-value>true</param-value>
        </context-param>
```

```
        <!-- this need same with resteasy servlet url-pattern -->
        <context-param>
                <param-name>resteasy.servlet.mapping.prefix</param-name>
                <param-value>/rest</param-value>
        </context-param>
        <listener>
                <listener-class>
                        org.jboss.resteasy.plugins.server.servlet.ResteasyBootstrap
                </listener-class>
        </listener>

        <servlet>
                <servlet-name>resteasy-servlet</servlet-name>
                <servlet-class>
                        org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher
                </servlet-class>
        </servlet>

        <servlet-mapping>
                <servlet-name>resteasy-servlet</servlet-name>
                <url-pattern>/rest/*</url-pattern>
        </servlet-mapping>

  <servlet-mapping>
    <servlet-name>redhat-jboss</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
```

RestService class:

```
package com.pushtotest.jboss.rest;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Iterator;
import java.util.List;
import java.util.Random;
import java.util.TimeZone;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.QueryParam;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import com.pushtotest.jboss.interfaces.BettingGamesService;
import com.pushtotest.jboss.interfaces.BettingOddsService;
import com.pushtotest.jboss.interfaces.NotificationService;
import com.pushtotest.jboss.model.BettingGames;
import com.pushtotest.jboss.model.BettingOdds;
import com.pushtotest.jboss.model.Notifications;
import com.pushtotest.jboss.model.UserIdentification;

@Path("/api")
public class RestService {

	@Autowired
  private BettingGamesService gamesRepository;

	@Autowired
  private BettingOddsService oddsRepository;

	@Autowired
  private NotificationService nRepository;

  @GET
	@Path("/gamesType")
  public String gamesType() {
	//List<BettingGames> bettingGamesList = gamesRepository.findAllDistinctByType();
	List<BettingGames> bettingGamesList = gamesRepository.findAll();
	Iterator<BettingGames> itr = bettingGamesList.iterator();

	JSONArray jAr = new JSONArray();
	while(itr.hasNext()){
		/*JSONObject jObj = new JSONObject();
		jObj.put("game", arg1)*/
		BettingGames bg = itr.next();
		jAr.put(bg.getType());
	}
	return jAr.toString();
  }

  @GET
	@Path("/gamesList")
  public String gamesList(@QueryParam("game") String gameType) throws JSONException {
	List<BettingGames> bettingGamesList = gamesRepository.findByType(gameType);
	Iterator<BettingGames> itr = bettingGamesList.iterator();

	JSONArray jAr = new JSONArray();
	while(itr.hasNext()){
		JSONObject jObj = new JSONObject();
		BettingGames bg = itr.next();
		jObj.put("game_id", bg.getGameId());
```

```
                jObj.put("team1", bg.getTeam1());
                jObj.put("team2", bg.getTeam2());

                SimpleDateFormat parser=new SimpleDateFormat("EEE, dd MMM yyyy HH:mm:ss
z");
                parser.setTimeZone(TimeZone.getTimeZone("GMT"));
                jObj.put("date", parser.format(new Date()));

                jObj.put("homebet", "-");
                jObj.put("drawbet", "-");
                jObj.put("awaybet", "-");
                jAr.put(jObj);
        }
        return jAr.toString();
  }

  @GET
        @Path("/createNotif")
  public String createNotif(@QueryParam("userId") String userId) throws JSONException {
        Notifications notif = new Notifications(userId, "UNREAD", "Sample Title", "Sample
Description");
        notif.setCreatedDate(new Date());
        nRepository.save(notif);
        return "done";
  }

  @GET
        @Path("/markAsRead")
  public String createNotif(@QueryParam("userId") String userId, @QueryParam("nId") String nId)
throws JSONException {
        Notifications notif = nRepository.findById(nId);
        notif.setStatus("READ");
        nRepository.save(notif);
        return "done";
  }

  @GET
        @Path("/getReadNotifications")
  public String greeting(@QueryParam("userId") String userId) throws Exception {
    List<Notifications> nList = nRepository.findByUserIdAndStatus(userId, "READ");
    Iterator<Notifications> itr = nList.iterator();

    JSONArray jAr = new JSONArray();
    while(itr.hasNext()){
        Notifications notif = itr.next();
        JSONObject jObj = new JSONObject();
        jObj.put("nId", notif.getId());
        Date cDate = notif.getCreatedDate();
```

```
        SimpleDateFormat sdf = new SimpleDateFormat("MM-DD-YYYY");
        jObj.put("date", sdf.format(cDate));
        jObj.put("title", notif.getTitle());
        jObj.put("description", notif.getDescription());
        jObj.put("status", notif.getStatus());

        jAr.put(jObj);
    }
    return jAr.toString();
  }

  @GET
        @Path("/lockAnOdd")
  public String lockAnOdd(@QueryParam("userId") String userId, @QueryParam("gameId") String
gameId, @QueryParam("oddType") String oddType) throws JSONException {
        List<BettingOdds> bettingOddsList = oddsRepository.findByGameIdAndStatus(gameId,
"active");
        Iterator<BettingOdds> itr = bettingOddsList.iterator();
        boolean result = true;
        while(itr.hasNext()){
                BettingOdds bo = itr.next();
                String lockStr = (bo.getLock() != null) ? bo.getLock() : "[]";
                if(lockStr.contains(userId)) result = false;
                else{
                        JSONArray jAr = new JSONArray(lockStr);
                        jAr.put(userId);
                        bo.setLock(jAr.toString());
                        if(oddType.equals("home")){
                                JSONArray homeList = new JSONArray((bo.getHome() != null) ?
bo.getHome() : "[]");
                                homeList.put(userId);
                        bo.setHome(homeList.toString());
                        } else if(oddType.equals("draw")){
                                JSONArray drawList = new JSONArray((bo.getDraw() != null) ?
bo.getDraw() : "[]");
                                drawList.put(userId);
                        bo.setHome(drawList.toString());
                        } else if(oddType.equals("away")){
                                JSONArray awayList = new JSONArray((bo.getAway() != null) ?
bo.getAway() : "[]");
                                awayList.put(userId);
                        bo.setHome(awayList.toString());
                        }
                        oddsRepository.save(bo);
                }
        }
        return ""+result;
  }
```

```
}
```

Rest services can be accessed through

http://localhost:8080/redhat-jboss/rest/api/gamesType

AsynchronousDispatcher error

Solution:

If deploying to JBoss 7.x you need to change the scope of your resteasy dependencies to provided. This is because those particular libraries are already included in JBoss as modules

5 hours to do frontend integrations, definitions and implementations for all the web services calls.

JBoss Messaging Services

Web sockets are a way of providing two way socket style interfaces between a web browser and a web server, with the server able to push information, rather than only responding to browser HTTP "pull" requests.

Plain sockets, however, are a good way of providing synchronous communications between applications. If the application receiving the messages can process them synchronously - as soon as they are sent - regular sockets may be a good solution.

Message queues are intended for asynchronous communications - cases where a message, after being sent, may need to be stored for a while before the recipient picks up the message and acts on it. Because of the need for storage, message queues require a separate server to store the message, or in some cases a database server if you already have one.

JMS is also an another mechanism for the Messaging services in the usecase app, Message Queue is the reference JMS implementation for Java EE. It is a piece of software that glues senders and receivers so that they can communicate without knowing much about each other (they both need to know about the queue, of course) and do not need to implement networking code, handling failure, routing one message to many receivers etc. The system works even if senders and receivers are never alive at the same time, as queues also serve as a temporary storage for undelivered messages. Aside from that, queues can provide additional services, like authorization, transactions etc.

Let’s consider using websocket,

Add the below dependency to the pom.xml

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-messaging -->
                        <dependency>
                           <groupId>org.springframework</groupId>
                           <artifactId>spring-messaging</artifactId>
                           <version>${spring-framework.version}</version>
                        </dependency>
```

The service will accept messages containing a id in a STOMP message whose body is a JSON object. If the id given is "523456789884545", then the message might look something like this:

```
{
   "id": "523456789884545"
}
```

To model the message carrying the id, you can create a plain old Java object with a id property and a corresponding getId() method:

src/java/main/com/pushtotest/jboss/model/UserIdentification.java

```
package com.pushtotest.jboss.model;

public class UserIdentification {

   private String id;

   public UserIdentification() {
   }

   public UserIdentification(String id) {
      this.id = id;
   }

   public String getId() {
      return id;
   }

   public void setId(String id) {
      this.id = id;
   }
}
```

Upon receiving the message and extracting the id, the service will process it by creating a JSON string response and publishing that JSON string on a separate queue that the client is subscribed to.

Spring will use the Jackson JSON library to automatically marshal instances of type Greeting into JSON.

Next, you'll create a controller to receive the hello message and send a greeting message.

Create a message-handling controller

In Spring's approach to working with STOMP messaging, STOMP messages can be routed to @Controller classes. For example the NotificationController is mapped to handle messages to destination "/notify".

src/java/main/com/pushtotest/jboss/controller/NotificationController.java

```
package com.pushtotest.jboss.controllers;
```

```
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Iterator;
import java.util.List;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.messaging.handler.annotation.DestinationVariable;
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.messaging.handler.annotation.SendTo;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

import com.pushtotest.jboss.interfaces.UserService;
import com.pushtotest.jboss.model.BettingGames;
import com.pushtotest.jboss.model.BettingOdds;
import com.pushtotest.jboss.model.Notifications;
import com.pushtotest.jboss.model.UserIdentification;
import com.pushtotest.jboss.repository.interfaces.BettingGamesRepository;
import com.pushtotest.jboss.repository.interfaces.BettingOddsRepository;
import com.pushtotest.jboss.repository.interfaces.NotificationRepository;

@Controller
public class NotificationController {

        @Autowired
  private UserService userService;

        @Autowired
  private BettingGamesRepository gamesRepository;

        @Autowired
  private BettingOddsRepository oddsRepository;

        @Autowired
  private NotificationRepository nRepository;

  @MessageMapping("/notify")
  @SendTo("/topic/notifications")
  public String greeting(UserIdentification uId) throws Exception {
    Thread.sleep(1000); // simulated delay
    //userService.updateBalance(userId, balance);
    List<Notifications> nList = nRepository.findByUserIdAndStatus(uId.getId(), "UNREAD");
    Iterator<Notifications> itr = nList.iterator();

    JSONArray jAr = new JSONArray();
    while(itr.hasNext()){
        Notifications notif = itr.next();
        JSONObject jObj = new JSONObject();
        jObj.put("nId", notif.getId());
        Date cDate = notif.getCreatedDate();
        SimpleDateFormat sdf = new SimpleDateFormat("MM-DD-YYYY");
        jObj.put("date", sdf.format(cDate));
        jObj.put("title", notif.getTitle());
        jObj.put("description", notif.getDescription());
```

```
            jObj.put("status", notif.getStatus());

            jAr.put(jObj);
        }
        return jAr.toString();
    }

    @MessageMapping("/getOdds")
    @SendTo("/topic/bettingodds")
    public String getOdds(UserIdentification uId) throws JSONException {
            //List<BettingGames> bettingGamesList = gamesRepository.findAllDistinctByType();
            List<BettingGames> bettingGamesList = gamesRepository.findAll();
            Iterator<BettingGames> itr = bettingGamesList.iterator();

            JSONArray jAr = new JSONArray();

            while(itr.hasNext()){
                    BettingGames bg = itr.next();
                    List<BettingGames> gamesList = gamesRepository.findByType(bg.getType());
                    Iterator<BettingGames> itr1 = gamesList.iterator();

                    while(itr.hasNext()){
                            BettingGames bg1 = itr1.next();
                            List<BettingOdds> bettingOddsList =
oddsRepository.findByGameIdAndStatus(bg1.getGameId(), "active");
                            Iterator<BettingOdds> itr2 = bettingOddsList.iterator();
                            while(itr2.hasNext()){
                                    JSONObject jObj = new JSONObject();
                                    BettingOdds bo = itr2.next();
                                    jObj.put("game_id", bo.getGameId());
                                    JSONArray totalList = new JSONArray((bo.getLock() != null) ? bo.getLock() : "[]");
                                    JSONArray homeList = new JSONArray((bo.getHome() != null) ? bo.getHome() : "[]");
                                    JSONArray drawList = new JSONArray((bo.getDraw() != null) ? bo.getDraw() : "[]");
                                    JSONArray awayList = new JSONArray((bo.getAway() != null) ? bo.getAway() : "[]");

                                    jObj.put("homebet", homeList.length() + "/" + totalList.length());
                                    jObj.put("drawbet", drawList.length() + "/" + totalList.length());
                                    jObj.put("awaybet", awayList.length() + "/" + totalList.length());
                                    jAr.put(jObj);
                            }
                    }
            }
            //return jAr.toString();
            return uId.getId() + "=" + new Date().getTime();
    }

}
```

The @MessageMapping annotation ensures that if a message is sent to destination "/notify", then the notifyUser() method is called.

Internally, the implementation of the method simulates a processing delay by causing the thread to sleep for 1 second. This is to demonstrate that after the client sends a message, the server can take as long as it needs to process the message asynchronously. The client may continue with whatever work it needs to do without waiting on the response.

After the 1 second delay, the notifyUser() method creates a Greeting object and returns it. The return value is broadcast to all subscribers to "/topic/notify" as specified in the @SendTo annotation.

Configure Spring for STOMP messaging

Now that the essential components of the service are created, you can configure Spring to enable WebSocket and STOMP messaging.

Create a Java class named WebSocketConfig that looks like this:

src/main/java/com/pushtotest/jboss/config/WebSocketConfig.java

```
package com.pushtotest.jboss.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.messaging.simp.config.MessageBrokerRegistry;
import org.springframework.web.socket.config.annotation.AbstractWebSocketMessageBrokerConfigurer;
import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;
import org.springframework.web.socket.config.annotation.StompEndpointRegistry;

@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig extends AbstractWebSocketMessageBrokerConfigurer {

   @Override
   public void configureMessageBroker(MessageBrokerRegistry config) {
      config.enableSimpleBroker("/topic");
      config.setApplicationDestinationPrefixes("/app");
   }

   @Override
   public void registerStompEndpoints(StompEndpointRegistry registry) {
      registry.addEndpoint("/messaging-service").withSockJS();
   }

}
```

This HTML file imports the SockJS and STOMP javascript libraries that will be used to communicate with our server using STOMP over websocket. Let's create the script file:

src/main/webapp/WEB-INF/static/js/script.js

```
var stompClient = null;

function setConnected(connected) {
   if (connected) {} else {}
}

function connect() {
   var socket = new SockJS('/messaging-service');
   stompClient = Stomp.over(socket);
   stompClient.connect({}, function (frame) {
      setConnected(true);
      //console.log('Connected: ' + frame);
      stompClient.subscribe('/topic/notifications', function (response) {
                response_data = JSON.parse(response.body);
                if(response_data.length){
                        !messageIcon.classList.contains('unreadMsg-count') ? messageIcon.classList.add('unreadMsg-
count') : "";
                                messageIcon.setAttribute('data-unreadMsg-count',response_data.length);
```

```
                                        if(page == "message"){
                updateMessage(response_data);
            }
                }else{
                                messageIcon.classList.contains('unreadMsg-count') ? messageIcon.classList.remove('unreadMsg-
count') : "";
                                messageIcon.removeAttribute('data-unreadMsg-count');
                }
      });

      stompClient.subscribe('/topic/bettingodds', function (response) {
         console.log(response_data);
      });
   });
}

function disconnect() {
   if (stompClient !== null) {
      stompClient.disconnect();
   }
   setConnected(false);
   //console.log("Disconnected");
}

function askForNotification() {
   stompClient.send("/app/notify", {}, JSON.stringify({'id': loggedInUser}));
}

function askForBettingUpdates(){
                stompClient.send("/app/getOdds", {}, JSON.stringify({'id': loggedInUser}));
}

if(loggedInUser != null && loggedInUser !=""){
                connect();
                setInterval(function(){
                                askForNotification();
                }, 1000);
                var intervalTime = 10000 + (Math.floor(Math.random() * 10) + 1) * 1000;

                setInterval(function(){
                                askForBettingUpdates();
                }, intervalTime);

                //profile page menu
                document.getElementsByClassName('menuWrap')[0].style.display = "block";
}
```

The main piece of this JavaScript file to pay attention to is the connect() and askForNotifications() functions.

The connect() function uses SockJS and stomp.js to open a connection to "/messaging-service", which is where our SockJS server is waiting for connections. Upon a successful connection, the client subscribes to the "/topic/notifications" destination, where the server will publish greeting messages. When a notification is received on that destination, it will update in UI.

The askForNotifications() function retrieves the name entered by the user and uses the STOMP client to send it to the "/app/notify" destination

STOMP does not, however, deal in queues and topics—it uses a SEND semantic with a “destination” string. The broker must map onto something that it understands internally such as a topic, queue, or

exchange. Consumers then SUBSCRIBE to those destinations. Since those destinations are not mandated in the specification, different brokers may support different flavours of destination. So, it's not always straightforward to port code between brokers.

3 hours to do add additional code in Javascript, Configuration and messaging implementations

Application start up configuration

Implement Startup listener class with the ServletContextListener interface , which make your class able to receive notifications from JBoss Server when it starts and shut-downs .

For example:

```
public class ApplicationContextListener implements ServletContextListener {
   /**This method will run when the web application starts***/
   public void contextInitialized(ServletContextEvent sce) {
      //List<BettingGames> bettingGamesList = gamesRepository.findAllDistinctByType();
                        List<BettingGames> bettingGamesList = gamesRepository.findAll();
            Iterator<BettingGames> itr = bettingGamesList.iterator();

            while(itr.hasNext()){
                        BettingGames bg = itr.next();
                        List<BettingGames> gamesList = gamesRepository.findByType(bg.getType());
                        Iterator<BettingGames> itr1 = gamesList.iterator();

                        while(itr.hasNext()){
                                    BettingGames bg1 = itr1.next();
                                    List<BettingOdds> bettingOddsList =
oddsRepository.findByGameIdAndStatus(bg1.getGameId(), "active");
                                    Iterator<BettingOdds> itr2 = bettingOddsList.iterator();
                                    while(itr2.hasNext()){
                                                BettingOdds bo = itr2.next();
                                                bo.setStatus("inactive");
                                                oddsRepository.save(bo);
                                    }
                        }
            }

            /** Starts new Bets*/

                        while(itr.hasNext()){
                        BettingGames bg = itr.next();
                        List<BettingGames> gamesList = gamesRepository.findByType(bg.getType());
                        Iterator<BettingGames> itr1 = gamesList.iterator();

                        while(itr.hasNext()){
                                    BettingGames bg1 = itr1.next();
                                    BettingOdds bo = new BettingOdds();
                                    bo.setGameId(bg1.getGameId());
                                    bo.setHome("[]");
                                    bo.setDraw("[]");
                                    bo.setAway("[]");
                                    bo.setLock("[]");
                                    bo.setStartDate(new Date());
                                    bo.setStatus("active");
                        }
            }
            return;
```

```
  }
}
```

Register your ApplicationContextListener in the web.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <listener>
        <listener-class>com.pushtotest.jboss.config.ApplicationContextListener </listener-class>
  </listener>
</web-app>
```

When the JBoss starts , the contextInitialized() in the ApplicationContextListener will run too.

2 hours to align the application startup configurations according to the usecase app.

Adding scheduler tasks

We have an option to use Quartz scheduler. In the earlier versions, Quartz was bundled along with Application server, Now We have to setup manually

Edit pom.xml

```
<dependency>
            <groupId>org.quartz-scheduler</groupId>
            <artifactId>quartz</artifactId>
            <version>2.1.5</version>
</dependency>
```

Install Quartz as one of the modules in the Application Server. Edit module.xml

```
 <module xmlns="urn:jboss:module:1.1" name="org.quartz">
 <resources>
  <resource-root path="quartz-2.2.3.jar"/>
  <resource-root path="quartz-jobs-2.2.3.jar"/>
  <resource-root path="c3p0-0.9.1.1.jar"/>
 </resources>
 <dependencies>
      <module name="org.slf4j"/>
      <module name="org.apache.log4j"/>
      <module name="javax.api"/>
 </dependencies>
</module>
```

Include the deployment of Quartz in the deployment as below

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="org.quartz" />
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

Quartz Servlet Application,

```
package com.pushtotest.jboss.config.scheduler;

import java.io.IOException;
import java.util.Date;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.*;

import org.quartz.*;
import org.quartz.impl.*;
import static org.quartz.JobBuilder.*;
import static org.quartz.TriggerBuilder.*;
import static org.quartz.DateBuilder.*;

@WebServlet("/applicationScheduler")
public class ApplicationScheduler extends HttpServlet {

   private static final long serialVersionUID = 1L;

   protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      try {
      SchedulerFactory sf = new StdSchedulerFactory();
      Scheduler sched = sf.getScheduler();
      sched.start();


      Date runTime = evenMinuteDate(new Date());
      // Trigger the job to run on the next round minute
      Trigger trigger = newTrigger()
        .withIdentity("trigger1", "group1")
        .startAt(runTime)
        .build();
      // Define job instance
      JobDetail job1 = newJob(HelloJob.class)
        .withIdentity("job1", "group1")
        .build();

      // Schedule the job with the trigger
      sched.scheduleJob(job1, trigger);

     // Set response content type
      response.setContentType("text/html");

     // Actual logic goes here.
      PrintWriter out = response.getWriter();
      out.println("<h1>Quartz Job Scheduled in a minute</h1>");
     }

       catch ( Exception de) {
```

```
            throw new IOException(de.getMessage());
        }
    }

}
```

2 hours to implement the Quartz scheduler tasks.

DevOps

DevOps combines both the development and operation teams together to work across the entire application life cycle including the build, test, deploy, monitor, manage and plan release process.

Major components of DevOps are,

- Continuous Integration & Delivery
- Microservices
- Infrastructure as code

Continuous Integration & Delivery

We will do the continuous integration with JBoss and Jenkins

Once the Jenkins setup is done, let's create the bootstrap job:

- Go to "New item" >> "Pipeline" named "BettingApp"
- "Pipeline Definition" >> "Pipeline script from SCM" >> "Git" >> add the repository url
- "Additional Behaviours" >> "Clean before checkout"

Triggers and other settings can be fine-tuned later.

This job now checks out or clones the entered repository and looks by default for a file called "Jenkinsfile" that has to contain build pipeline dsl: https://jenkins.io/doc/pipeline/

```
import groovy.json.JsonSlurper;


node{
   stage 'Build, Test and Package'
   env.PATH = "${tool 'apache-maven-3.3.9'}/bin:${env.PATH}"
   git url: "https://github.com/.../betting-app.git"
   def commitid = sh(returnStdout: true, script: 'git rev-parse HEAD').trim()
   def workspacePath = pwd()
   sh "echo ${commitid} > ${workspacePath}/expectedCommitid.txt"
   sh "mvn clean build -Dcommitid=${commitid}"
}


node{
   stage 'Stop, Deploy and Start'
   // shutdown
```

```
    sh 'curl -X POST http://localhost:8080/shutdown || true'
    // start the application
    sh ‘mvn clean install run’
    // wait for application to respond
    sh 'while ! httping -qc1 http://localhost:8080  ; do sleep 1 ; done'
}

node{
    stage 'Smoketest'
    def workspacePath = pwd()
    sh "curl --retry-delay 10 --retry 5 http://localhost:8080/info -o ${workspacePath}/info.json"
    if (deploymentOk()){
        return 0
    } else {
        return 1
    }
}

def deploymentOk(){
    def workspacePath = pwd()
    expectedCommitid = new File("${workspacePath}/expectedCommitid.txt").text.trim()
    actualCommitid = readCommitidFromJson()
    println "expected commitid from txt: ${expectedCommitid}"
    println "actual commitid from json: ${actualCommitid}"
    return expectedCommitid == actualCommitid
}

def readCommitidFromJson() {
    def workspacePath = pwd()
    def slurper = new JsonSlurper()
    def json = slurper.parseText(new File("${workspacePath}/info.json").text)
    def commitid = json.app.commitid
    return commitid
}
```

It has three steps,

Build, Test and Package:

determination of the git commit id and running the maven build with -Dcommitid parameter (so the information is available at runtime in /info endpoint)

Stop, Deploy and Start:

using the shutdown hook to stop the app, copy the artifact and start it

Smoketest:

retrieve commitid from /info endpoint and compare it with the one that was used for building the artifact.

After committing this file to the repository and triggering the job, the job will pick up the file and run the pipeline. For security reasons all the groovy scripts are sandboxed and so the build will fail with the below error message

```
org.jenkinsci.plugins.scriptsecurity.sandbox.RejectedAccessException: Scripts not permitted to use staticMethod
org.codehaus.groovy.runtime.DefaultGroovyMethods getText java.io.File

  at org.jenkinsci.plugins.scriptsecurity.sandbox.whitelists.StaticWhitelist.rejectStaticMethod(StaticWhitelist.java:190)

  at org.jenkinsci.plugins.scriptsecurity.sandbox.groovy.SandboxInterceptor$8.reject(SandboxInterceptor.java:272)

  at org.jenkinsci.plugins.scriptsecurity.sandbox.groovy.SandboxInterceptor.onGetProperty(SandboxInterceptor.java:363)
```

This shall avoid damages on the Jenkins infrastructure by executing invasive operations through groovy code. Administrators can approve usage of specific method signatures under "Manage Jenkins" >> "In-process script-aproval" or in config xml "scriptApproval.xml"

So, with this pipeline you combine the sources of the application with the complete build process in a revision-safe system and devs can individually modify or extend their build-process without any changes on Jenkins itself.

Microservices

Let's use the aws-serverless-java-container library to run a RedHat JBoss application (built on Spring Framework) in AWS Lambda.

Here are the step by step functionalities,

- Import "aws-serverless-java-container-spring" dependency
- Create the Lambda Handler
- Package the Application
- Publish your Lambda Function

Infrastructure as code

Infrastructure as Code (IaC) is the process of managing and configuring servers and environments through human-readable definition files. When done properly, it allows infrastructure to be rebuilt consistently and to be automated - no more logging in as root on production servers.

This can save your team a lot of pain, especially when working across multiple environments.

There are many tools to accomplish this,

- AWS CloudFormation
- Chef
- Puppet
- Saltstack
- Docker
- Vagrant and many more

10 hours to explore the devops opportunities

Containers and Clouds

Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications, whether on laptops, data center VMs, or the cloud.

To deploy an application to a JBoss EAP instance running inside a container, you can do one of the followings:

- Deploy the application to a running container using the management console or management CLI.
- Use the COPY command in the Dockerfile to deploy the application with the deployment scanner when the container starts.

Let's look at how the second option works, To use the deployment scanner to deploy an application when the container starts, you can use the COPY command in the Dockerfile.

Example Dockerfile for Deploying an Application Using the Deployment Scanner

```
FROM registry.access.redhat.com/jboss-eap-7-tech-preview/eap70
COPY app.war $JBOSS_HOME/standalone/deployments/
USER root
RUN chown jboss:jboss $JBOSS_HOME/standalone/deployments/app.war
USER jboss
```

The above example copies the app.war file to a location which is scanned by the deployment scanner and updates the file to have the appropriate permissions. This will cause app.war to be deployed when the JBoss EAP instance is started as the container is started.

2 hours to setup docker instance and to deploy the EAP application

Making Changes

The management command-line interface (CLI) is a command-line administration tool for JBoss EAP.

Use the management CLI to start and stop servers, deploy and undeploy applications, configure system settings, and perform other administrative tasks. Operations can be performed in batch mode, allowing multiple tasks to be run as a group.

Here are steps to follow,

- Deploy the initial application inside a docker container using the above-mentioned steps.
- Make the changes
- Deploy/un-deploy/Re-deploy the application using the Management CLI

1 hour to auto configure the build while making changes

Versioning

Here's the implementation of URI path versioning inside Jax-RS @Path.

```
@Path("/api")
public class RestService {
   @Path({"/v1.0", "/v1.1"})
   public String createNotif(@QueryParam("userId") String userId) throws JSONException {
            Notifications notif = new Notifications(userId, "UNREAD", "Sample Title", "Sample Description");
            notif.setCreatedDate(new Date());
            nRepository.save(notif);
            return "done";
   }

   @Path("/v1.2")
   public String createNotif(@QueryParam("userId") String userId, @QueryParam("nId") String nId) throws JSONException {
            Notifications notif = nRepository.findById(nId);
            notif.setStatus("READ");
            nRepository.save(notif);
            return "done";
   }
}
```

30 mins to accomplish versioning

Conclusions