JBoss Java EE 6 Workshop

Dan Allen

JBoss Java EE 6 Workshop

Dan Allen

Table of Contents

This tutorial steps through creating, testing and deploying an application using the JBoss Java EE tools and runtimes.

Here’s the development stack we’ll be using:

- OpenShift
- git
- JBoss Forge
- JBoss Tools
- JBoss AS 7
- Arquillian

Hang on, cause it’s going to move *fast*.

Chapter 1. OpenShift Introduction

1.1. Why Java EE on OpenShift?

You’ve probably been hearing a lot about cloud (“to the cloud!”). But, if you’re like me, you just haven’t been that motivated to try it, or it just seemed like too much effort. After all, taking a Java EE application to the cloud typically meant:

- Breaking out a credit card (I hate being restricted from exploring)
- Implementing tons of hacks to get a Java EE application running on a cloud provider
- Not knowing how Java EE development is even compatible with cloud
- Lack of crystal clear documentation and examples

All that has changed. I assure you, you’re in for a pleasant surprise.

OpenShift [http://openshift.com] provides a free, cloud-based application platform for Java, Perl, PHP, Python, and Ruby applications using a shared-hosting model. Exploring the cloud has never been simpler. You can deploy a Java EE compliant application without requiring any hacks or workarounds. Best of all, take as much time as you like, because it’s all *free*. (Oh, and you can show off your applications to your friends, or even run applications for your own purposes).

It’s time to get going!

1.2. Creating an OpenShift account

Creating an OpenShift account is very straightforward.

1. Go to the OpenShift homepage [http://openshift.com]
2. Click on the "Signup and try it" button to open the registration form
3. Enter a valid email address, a password (twice) and pass the Captcha (I know, it’s tough)
4. Check your email and follow instructions in the welcome email

Welcome to OpenShift!

1.3. Installing Ruby-based OpenShift client tools

To interact with the OpenShift Platform-as-a-Service (PaaS), you’ll need one of the available OpenShift client tools installed. We’ll be using the Ruby-based commandline tool, rhc, in this tutorial.

To install the Ruby-based OpenShift client tools, you first need to install git and Ruby Gems.

On a Debian-based Linux distribution, it’s as easy as:

```
$> sudo apt-get install ruby rubygems
```

On a rpm-based Linux distribution, it’s as easy as:

```
$> sudo yum install ruby rubygems
```

Next, install the Red Hat Cloud (rhc) gem:

```
$> sudo gem install rhc
```

This will add a set of commands to your shell prefixed with rhc- (e.g., rhc-create-domain).

On some distributions the gem installation directory is not automatically added to the path. Find the location of your gems with gem environment and add it to you system’s PATH environment variable.

For more information see Installing OpenShift client tools on Linux, Windows and Mac OSX [https://redhat.com/openshift/community/kb/kb-e1000/installing-openshift-express-client-tools-on-non-rpm-based-systems].

1.4. Creating a domain for your applications

Before actually deploying an app to OpenShift, you first need to create a domain. A domain hosts one or more applications. The name of the domain will be used in the URL of your apps according to the following scheme:

http://[application name]-[domain name].rhcloud.com

Create a domain with the following command:

```
$> rhc-create-domain -n [domain name] -l [openshift email]
```

Use the same email address you used to register for your OpenShift account and enter your OpenShift password when prompted.

All OpenShift client tools prompt you for your OpenShift password (unless you supply it using the -p command flag)

The command will then create a pair of private and public keys as *libra_id_rsa* and *libra_id_rsa.pub* in your *$HOME/.ssh/* directory. It will ask you for a password to access the private key. **Don’t forget it!**

If you want to use an existing ssh key file, you can specify it explicitly in the *$HOME/.openshift/express.conf* file. You’ll also discover that your email is cached in this file, which means you don’t have to specify it in subsequent commands.

You can see a summary of your domain using the following command:

```
$> rhc-domain-info
```

You can also go to the dashboard [https://openshift.redhat.com/app/console/applications] to see your newly minted domain. You now have your own, free cloud. Woot!

1.5. Creating your first Java EE application on OpenShift

We now want to create a new application that uses the JBoss AS 7 cartridge. This allows us to deploy Java EE-compliant applications to OpenShift.

We’ll assume you’ll be developing the application in the following folder: *~/demos/apps/sellmore*

Next, use the following command to create a slot in your OpenShift domain in which to run the application and the location where the local project should be created:

```
$> rhc-create-app -a sellmore -t jbossas-7.0 -r ~/demos/apps/sellmore
```

You’ll be prompted for your ssh key password that you created in the previous step.

Behind the scenes, OpenShift has created a git repository for you and cloned it locally. That’s how you’re going to "push" your application to the cloud. The cloned repository contains a Maven-based project structure (which you don’t have to use):

Project layout.

```
sellmore
|- .git/
|- .openshift/
|- deployments/
|- src/
|- .gitignore
|- pom.xml
`- README
```

The *README* describes all the special directories that pertain to OpenShift.

The OpenShift setup leaves behind a sample application which is going to get in our way later on. So first, let’s clear the path:

```
$> cd sellmore
$> git rm -r pom.xml src
$> git commit -m 'clear a path'
$> cd ..
```

If you’re working with another origin git repository (such as on github), we recommend renaming the OpenShift repository from origin to openshift:

```
$> cd sellmore
$> git remote rename origin openshift
$> cd ..
```

That separates the concern of managing your source code repository from deploying files to OpenShift.

You can see a summary of your application configuration using the following command:

```
$> rhc-domain-info
```

You can also go to the dashboard [https://openshift.redhat.com/app/console/applications] to see your application slot. If you click on the URL, you’ll see that a sample application is already running in the cloud. We’ll be replacing that soon enough.

If, for whatever reason, you need to delete your application, use this command:

```
$> rhc-ctl-app -a sellmore -c destroy
```

You’ll also want to delete your local .git repository (unless you mean to save it).

But now’s not the time to delete, it’s time to create!

Chapter 2. JBoss Forge Introduction

2.1. Why JBoss Forge?

Because starting a project is hard. It doesn't just take time, it takes mental energy. We want to save that energy for creating useful things. Trust me, even if copying and pasting 20 lines of build XML seems easy, somewhere along the line your going to find yourself roasting your brain. Let's toss the complexity over the wall and let a tool like Forge deal with it.

Forge is your monkey, or 10,000 of them.

2.2. Setting up Forge

To create our application, we're going to use JBoss Forge [http://jboss.org/forge]. Forge is a plugin-based framework for rapid development of standards-based Java applications.

Begin by grabbing Forge from the download area [https://repository.jboss.org/nexus/index.html#nexus-search;gav~org.jboss.forge~forge-distribution~~~~kw,versionexpand]. Then, unpack the distribution:

```
$> unzip forge-distribution-1.0.0.Beta5.zip
```

Move the extracted folder to the location of your choice and change into that directory in your console:

```
$> cd ~/opt/forge
```

Finally, run Forge:

```
$> ./bin/forge
```

To be sure everything is working okay, run the about command in the Forge shell:

```
$forge> about

  _____
 |  ___|__  _ __ __ _  ___
 | |_ / _ \| `__/ _` |/ _ \  \\
 |  _| (_) | | | (_| |  __/  //
 |_|  \___/|_|  \__, |\___|
                |___/

JBoss Forge, version [ 1.0.0.Beta5 ] - JBoss, by Red Hat, Inc. [ http://jboss.org/forge
```

Any command in this document prefixed with `$forge>` is intended to be run in the Forge shell.

Things look good. We're ready to create an application.

2.3. Generating an application with Forge

Forge allows you to create a Java EE application from scratch. We’re going to generate a point of sale application step-by-step in the Forge shell using the commands below (make sure Forge is running):

Forge commands to create a Java EE web project.

```
new-project; ❶

scaffold setup --scaffoldType faces; ❷
persistence setup --provider HIBERNATE --container JBOSS_AS7; ❸
validation setup --provider HIBERNATE_VALIDATOR; ❹

entity --named Customer --package ~.domain; ❺
field string --named firstName;
field string --named lastName;
field temporal --type DATE --named birthDate;
entity --named Item;
field string --named name;
field number --named price --type java.lang.Double;
field int --named stock;
cd ..;

entity --named ProductOrder; ❻
field manyToOne --named customer --fieldType ~.domain.Customer.java --inverseFieldName o
cd ../Customer.java;
entity --named Profile;
field string --named bio;
field string --named preferredName;
field string --named notes;
entity --named Address;
field string --named street;
field string --named city;
entity --named ZipCode;
field int --named code;
cd ../Address.java;

field manyToOne --named zip --fieldType ~.domain.ZipCode.java; ❼
cd ..;
cd Customer.java;
field manyToMany --named addresses --fieldType ~.domain.Address.java;
cd ..;
cd Address.java;
cd ../Customer.java;
field oneToOne --named profile --fieldType ~.domain.Profile.java;
cd ..;
cd ProductOrder.java;
field manyToMany --named items --fieldType ~.domain.Item.java;
cd ..;
cd ProductOrder.java;
field manyToOne --named shippingAddress --fieldType ~.domain.Address.java;
cd ..;

scaffold from-entity ~.domain.* --scaffoldType faces --overwrite; ❽
cd ~~;

rest setup; ❾
rest endpoint-from-entity ~.domain.*; ❿

build; 11

cd ~~; 12
echo "Project Info:"; project;
```

❶ Create a new project in the current directory
❷ Turn our Java project into a Web project with JSF, CDI, EJB
❸ Setup JPA
❹ Setup Bean Validation
❺ Create some JPA entities on which to base our application
❻ Create more entities, also add a relationship between Customer and their Orders
❼ Add more relationships between our entities
❽ Generate the UI for all of our entities at once
❾ Setup JAX-RS
❿ Generate CRUD endpoints
11 Build the project
12 Return to the project root directory and leave it in your hands

You’ve got a complete application, ready to deploy!

But wait! That sure seemed like a lot of typing. What’s really great about Forge is that it’s fine-grained enough to perform simple operations, but it can also compose those operations inside plugins or scripts!

You can take all of those commands and put them into a file named *generate.fsh*. You may also want to wrap the following two lines around the contents so that the commands run without pausing:

Use as first line.

```
set ACCEPT_DEFAULTS true; ❶
```

❶ Disables interactive commands

Use as last line.

```
set ACCEPT_DEFAULTS false; ❶
```

❶ Reenables interactive commands

Now you can build the application in a single command:

```
$forge> run generate.fsh
```

That’s more like it! Now, let’s get the application running!

Chapter 3. Application Deployment

3.1. Deploying your first Java EE application to OpenShift

There are two ways to deploy an application to OpenShift:

1. **Deploy the source**

   You can commit your source files and push them to the remote server using git, at which point the application will be built and deployed on the remote host. Alternatively, you can use a Jenkins slave to handle the build and deploy steps on the server. More on that later.

2. **Deploy a package**

   You can copy a pre-built war into *deployments/* (with the corresponding .dodeploy file for an exploded war) and use git to commit and push the file(s) to the remote server for deployment

In the first scenarios, you edit the files locally and let OpenShift build the app using Maven and deploy it to JBoss AS 7 once you push the changes using git. In the second scenario, you build the application locally and just push the final package to OpenShift, which it will deploy to JBoss AS 7.

We’re going to take the source route.

First, add the following profile to the end of the pom.xml file (inside the root element):

pom.xml (fragment).

```
<profiles>
  <profile>
   <!-- When built in OpenShift the 'openshift' profile will be used when invoking mvn.
   <!-- Use this profile for any OpenShift specific customization your app will need. --
   <!-- By default that is to put the resulting archive into the 'deployments' folder. -
   <!-- http://maven.apache.org/guides/mini/guide-building-for-different-environments.ht
   <id>openshift</id>
   <build>
      <finalName>sellmore</finalName>
      <plugins>
        <plugin>
          <artifactId>maven-war-plugin</artifactId>
          <version>2.1.1</version>
          <configuration>
            <outputDirectory>deployments</outputDirectory>
            <warName>ROOT</warName>
          </configuration>
        </plugin>
      </plugins>
    </build>
  </profile>
</profiles>
```

If you forget this profile, then the application will build on the OpenShift PaaS, but it will not be deployed to JBoss AS 7.

You may want to add the Eclipse project files to *.gitignore* so that they aren’t committed.

Next, we’ll add all the new files to git, commit them and push them to the server. You can perform these operations directly inside the Forge shell:

```
$forge> git add pom.xml src
$forge> git commit -a -m 'new project'
$forge> git push openshift master
```

You’ll see the OpenShift begin the build lifecycle on the server, which includes executing Maven and downloading the (nearby) internet. The console output you’re seeing is from the remote server being echoed into your local console.

3.2. OpenShift Build Lifecycle

The OpenShift build lifecycle comprises four steps:

1. **Pre-Receive**

   Occurs when you run a git push command, but before the push is fully committed.

2. **Build**

   Builds your application, downloads required dependencies, executes the *.openshift/action_hooks/build* script and prepares everything for deployment.

3. **Deploy**

   Performs any required tasks necessary to prepare the application for starting, including running the *.openshift/action_hooks/deploy* script. This step occurs immediately before the application is issued a start command.

4. **Post-Deploy**

   Allows for interaction with the running application, including running the *.openshift/action_hooks/post_deploy* script. This step occurs immediately after the application is started.

When the build is done, you’ll notice that the application is deployed to JBoss AS 7. You can now visit the application URL again to see the application running.

http://sellmore-[domain name].rhcloud.com

You should see the Forge welcome page and a list of items in the sidebar you can create, read, update and delete (CRUD).

If you want to push out a new change, simply update a file, then use git to commit and push again:

```
$forge> git commit -a -m 'first change'
$forge> git push openshift master
```

The OpenShift build lifecycle will kick off again. Shortly after it completes, the change will be visible in the application.

Chapter 4. Application Management

4.1. Managing OpenShift applications from a shell environment

OpenShift isn't just a black box (black cloud?), it's Linux and it's open! That means you can shell into your cloud just as you would any (decent) hosting environment.

So what's the login? It's embedded there in the git repository URL. Let's find it.

```
$> git remote show -n openshift
```

You can also get the same information using:

```
$> rhc-domain-info -a
```

You are looking for the ssh username and host in the form `username@hostname`. Once you've got that, just pass it to ssh and the authentication will be handled by the ssh key you setup earlier. Here's the syntax:

```
$> ssh [UUID]@[application name]-[domain name].rhcloud.com
```

There's a lot of power in that shell environment. You can type help to get a list of speciality commands (such as starting, stopping or restarting your app), or use just about any Linux shell command you know. Be sure to pay attention to what you're typing, though rest assured that the box is running on RHEL secured with SELinux.

4.2. Viewing the log files

There are two ways to view (tail) the log files of your application. You can use the client tool:

```
$> rhc-tail-files -a sellmore
```

Or you can shell into the server and use the built-in tail command:

```
$> tail_all
```

You can also use the regular tail command in the remote shell environment.

4.3. Restarting or stopping your application

You can control your application directly without pushing files through git. One way is to use the client tool from your location machine:

```
$> rhc-ctl-app -c restart
```

You can also shell into your domain and execute a command using one of the special commands provided:

```
$> ctl_app restart
```

In addition to restart, you can use the commands start, stop, etc.

Chapter 5. Persistence Storage

5.1. Preserving the database between restarts

You may have noticed that each time we restart the application, the data gets lost. There are two ways to resolve this:

1. Update tables rather that dropping and recreating them on deployment

2. Save the data to a safe location on disk

The first setting is a feature of Hibernate (or alternate JPA provider) and is changed using the following property in *src/main/resources/META-INF/persistence.xml*:

src/main/resources/META-INF/persistence.xml (fragment).

```
<property name="hibernate.hbm2ddl.auto" value="update"/>
```

The second feature depends on the database you are using. If you are using the provided H2 database, you’ll likely want to change the configuration in *.openshift/config/standalone.xml* to use the OpenShift data directory:

.openshift/config/standalone.xml (fragment).

```
<connection-url>jdbc:h2:${OPENSHIFT_DATA_DIR}/test;DB_CLOSE_DELAY=-1</connection-url>
```

The other approach is just to use a regular client-server database (e.g., MySQL or PostgreSQL), which we’ll do later.

5.2. Persisting data to a MySQL database

OpenShift provides us with several add-on services (cartridges) we can use. To see the list of available cartridges, issue the following command:

```
$> rhc-ctl-app -a sellmore -L

List of supported embedded cartridges:

  postgresql-8.4, metrics-0.1, mysql-5.1, jenkins-client-1.4,
  10gen-mms-agent-0.1, phpmyadmin-3.4, rockmongo-1.1, mongodb-2.0
```

Oh goody! Lots of options :)

Let’s install mysql-5.1 cartridge:

```
$> rhc-ctl-app -a sellmore -e add-mysql-5.1

Mysql 5.1 database added. Please make note of these credentials:

   Root User: admin
   Root Password: xxxxx
   Database Name: sellmore

Connection URL: mysql://127.1.47.1:3306/

You can manage your new Mysql database by also embedding phpmyadmin-3.4.
```

The name of the database is the same as the name of the application.

OpenShift is telling us that the phpmyadmin cartridge is also available, so we’ll add it as well.

```
$> rhc-ctl-app -a sellmore -e add-phpmyadmin-3.4

phpMyAdmin 3.4 added. Please make note of these credentials:

   Root User: admin
   Root Password: xxxxx

URL: https://sellmore-[domain name].rhcloud.com/phpmyadmin/
```

Open a browser and go to the URL shown, then login as admin with the password reported by the previous command.

It’s a good idea to create another user with limited privileges (select, insert, update, delete, create, index and drop) on the same database.

You can also shell into the domain and control MySQL using the MySQL client. You’ll need to connect using the hostname provided when you added the cartridge since it’s running on a different interface (not through a local socket).

```
$> mysql -u $OPENSHIFT_DB_USERNAME -p$OPENSHIFT_DB_PASSWORD -h $OPENSHIFT_DB_HOST
```

Now we’ll configure our application to use OpenShift’s MySQL database when running in the cloud.

5.3. Switching the application datastore to MySQL

The JBoss AS 7 cartridge comes configured out of the box with datasources for H2 (embedded), MySQL and PostgreSQL. The datasources for MySQL and PostgreSQL are enabled automatically when the respective cartridges are added. You can find this configuration in *.openshift/config/standalone.xml*.

Here’s the datasource name that cooresponds to the MySQL connection pool:

java:jboss/datasources/MysqlDS

The connection URL uses values that are automatically populated via environment variables maintained by OpenShift.

jdbc:mysql://${OPENSHIFT_DB_HOST}:${OPENSHIFT_DB_PORT}/${OPENSHIFT_APP_NAME}

All you need to do is open up the *src/main/resources/META-INF/persistence.xml* and set the JTA datasource:

src/main/resources/META-INF/persistence.xml (fragment).

```
<jta-data-source>java:jboss/datasources/MysqlDS</jta-data-source>
```

If you want to use PostgreSQL, follow the steps above for setting up MySQL, but replace it with the PostgreSQL cartridge (postgresql-8.4). Then, you’ll use this datasource in your persistence.xml:

src/main/resources/META-INF/persistence.xml (fragment).

```
<jta-data-source>java:jboss/datasources/PostgreSQLDS</jta-data-source>
```

You can connect to the PostgreSQL prompt on the domain using this command:

```
$> psql -h $OPENSHIFT_DB_HOST -U $OPENSHIFT_DB_USERNAME -d $OPENSHIFT_APP_NAME
```

Chapter 6. Advanced Deployment

6.1. Building with Jenkins

Jenkins is a continous integration (CI) server. When installed in an OpenShift environment, Jenkins takes over as the build manager for your application. You now have two options for how to build and deploy on OpenShift:

Building without Jenkins
: Uses your application space as part of the build and test process. Because of this, the application is stopped to free memory while the build is running.

Building with Jenkins
: Uses dedicated application space that can be larger then the application runtime space. Because the build happens in its own dedicated jail, the running application is not shutdown or changed in any way until after the build is a success.

Here are the benefits to using Jenkins:

- Archived build information
- No application downtime during the build process
- Failed builds do not get deployed (leaving the previous working version in place).
- Jenkins builders have additional resources like memory and storage
- A large community of Jenkins plugins (300+)

To enable Jenkins to use with an existing application, you first create a dedicated jenkins application:

```
$> rhc-create-app -a builds -t jenkins-1.4
```

Then you add the Jenkins client to your own application:

```
$> rhc-ctl-app -a sellmore -e add-jenkins-client-1.4
```

Make a note of the admin account password for Jenkins and point your browser at the following URL:

http://builds-[domain name].rhcloud.com

Once you are there, you can click "log in" in the top right of the Jenkins window to sign in and start tweaking the Jenkins configuration.

Now you simply have to do a git push to remote branch and Jenkins will take over building and deploying your application.

The pre-Jenkins way of doing this was to fire off a command line build and dump the output to the screen. You’ll notice that this output is replaced with a URL where you can view the output and status of the build.

Chapter 7. Integration Testing

7.1. Writing real cloud tests with Arquillian

Bring your tests to the runtime instead of managing the runtime from your test. Isn’t the cloud one of those runtimes? It sure is!

Let’s use Arquillian to write some tests that run on a local JBoss AS instance. Later, we’ll get them running on OpenShift.

Setting up Arquillian requires thought. Let’s put those 10,000 monkeys to work again. Open up Forge and see if it can find a plugin to help us get started with Arquillian.

```
$forge> forge find-plugin arquillian
```

Sure enough, there it is!

```
- arquillian (org.arquillian.forge:arquillian-plugin:::1.0.0-SNAPSHOT)
       Author: Paul Bakker <paul.bakker.nl@gmail.com>
       Website: http://www.jboss.org/arquillian
       Location: git://github.com/forge/plugin-arquillian.git
       Tags: arquillian, jboss, testing, junit, testng, integration testing, tests, CDI,
       Description: Integration Testing Framework
```

Let’s snag it.

```
$forge> forge install-plugin arquillian
```

That will clone the plugin source, build it and install it into the Forge shell. Once it’s finished, we can get straight to the Arquillian setup. We’ll first create a profile for a JBoss AS 7 instance running locally.

```
$forge> setup arquillian --container JBOSS_AS_REMOTE_7.X
```

At the time of writing, the plugin puts the Arquillian BOM dependency in the wrong section. Move it into the dependencyManagement section below the others:

pom.xml.

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.arquillian</groupId>
      <artifactId>arquillian-bom</artifactId>
      <version>1.0.0.CR7</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

You can also remove the version from the `arquillian-junit-container` dependency. Both of these problems will be fixed in the next release of the plugin.

We can also use the Forge Arquillian plugin to create tests for us. Let’s create an integration test for one of the services created earlier:

```
$forge> arquillian create-test --class com.acme.sellmore.rest.ItemEndpoint --enableJPA
```

This test is going to read and write to a database. You probably don't want to mix test data with application data, so first copy the JPA descriptor (persistence.xml) to the test classpath and prefix the file with test- so it doesn't get mixed up:

```
$forge> cd ~~
$forge> cp src/main/resources/META-INF/persistence.xml src/test/resources/test-persisten
```

Make sure the *test-persistence.xml* uses the ExampleDS datasource (or whatever you want to use for tests).

Next, open up the test in your editor so we can work it into a useful test. Begin by updating the ShrinkWrap archive builder to snag the JPA descriptor from the test classpath (instead of the production one):

src/test/java/com/acme/sellmore/rest/ItemEndpointTest.java (fragment).

```
.addAsManifestResource("test-persistence.xml", "persistence.xml")
```

Assign the `@Test` method a meaninful name and replace the contents with logic to validate that an item can be created in one transaction and retrieved in another:

src/test/java/com/acme/sellmore/rest/ItemEndpointTest.java (fragment).

```
@Test
public void should_insert_and_select_item() {
    Item item = new Item();
    item.setName("Widget");
    item.setPrice(5.0);
    item.setStock(100);
    item = itemendpoint.create(item);
    Long id = item.getId();
    Assert.assertNotNull(id);
    Assert.assertTrue(id > 0);
    Assert.assertEquals(item.getVersion(), 0);

    item = itemendpoint.findById(id);
    Assert.assertNotNull(item);
    Assert.assertEquals("Widget", item.getName());
}
```

The test is ready to run. First, start JBoss AS 7.

```
$> cd $JBOSS_HOME
$> ./bin/standalone.sh
```

Run the Arquillian test on this instance by activating the cooresponding profile when running the Maven test command:

```
$forge> test --profile JBOSS_AS_REMOTE_7.X
```

If things go we'll, the tests will pass and you'll see some Hibernate queries in the JBoss AS console. "Green bar!"

The previous test runs inside the container. Let's write another test that acts as a client to the REST endpoint. To keep effort to a minimum, we'll use the Apache HttpComponents HttpClient [http://

hc.apache.org/httpcomponents-client-ga] library to invoke the HTTP endpoints. We can get Forge to add it to our build:

```
$forge> project add-dependency org.apache.httpcomponents:httpclient:4.1.2:test
```

Let’s REST!

Sigh. There’s no better way to do this at the moment, so copy the previous test and rename it to `ItemEndpointClientTest` (rename both the file and the class name). Then, replace the class definition with the following source:

src/test/java/com/acme/sellmore/rest/ItemEndpointClientTest.java.

```
@RunWith(Arquillian.class)
public class ItemEndpointClientTest {
    @ArquillianResource
    private URL deploymentUrl;

    @Deployment(testable = false)
    public static WebArchive createDeployment() {
        return ShrinkWrap.create(WebArchive.class, "test.war")
                .addClasses(Item.class, ItemEndpoint.class)
                .addAsResource("META-INF/persistence.xml")
                .addAsWebInfResource(EmptyAsset.INSTANCE, "beans.xml")
                .setWebXML(new File("src/main/webapp/WEB-INF/web.xml"));
    }

    @Test
    public void should_post_update_and_get_item() {
        DefaultHttpClient client = new DefaultHttpClient();
        String itemResourceUrl = deploymentUrl + "rest/item";

        String ITEM_XML = "<item>%1$s<name>Widget</name><price>5.0</price><stock>%3$d</s

        // POST new item
        HttpPost post = new HttpPost(itemResourceUrl);
        post.setEntity(createXmlEntity(String.format(ITEM_XML, "", "", 99)));

        String result = execute(post, client);
        assertEquals(String.format(ITEM_XML, "<id>1</id>", "<version>0</version>", 99),

        // PUT update to item 1
        HttpPut put = new HttpPut(itemResourceUrl + "/1");
        put.setEntity(createXmlEntity(String.format(ITEM_XML, "", "", 98)));

        execute(put, client);

        // GET item 1
        HttpGet get = new HttpGet(itemResourceUrl + "/1");
        get.setHeader("Accepts", MediaType.APPLICATION_XML);

        result = execute(get, client);
        assertEquals(String.format(ITEM_XML, "<id>1</id>", "<version>1</version>", 98),

        client.getConnectionManager().shutdown();
    }
}
```

Also add these two private helper methods (to hide away some of the boilerplate code):

src/test/java/com/acme/sellmore/rest/ItemEndpointClientTest.java (fragment).

```
private HttpEntity createXmlEntity(final String xml) {
    ContentProducer cp = new ContentProducer() {
        public void writeTo(OutputStream outstream) throws IOException {
            Writer writer = new OutputStreamWriter(outstream, "UTF-8");
            writer.write(xml);
            writer.flush();
        }
    };

    AbstractHttpEntity entity = new EntityTemplate(cp);
    entity.setContentType(MediaType.APPLICATION_XML);
    return entity;
}

private String execute(final HttpUriRequest request, final HttpClient client) {
    try {
        System.out.println(request.getMethod() + " " + request.getURI());
        return client.execute(request, new BasicResponseHandler())
                .replaceFirst("<\\?xml.*\\?>", "").trim();
    }
    catch (Exception e) {
        e.printStackTrace();
        Assert.fail(e.getMessage());
        return null;
    }
    finally {
        request.abort();
    }
}
```

Let’s see if these endpoints do what they claim to do.

```
$forge> test --profile JBOSS_AS_REMOTE_7.X
```

If you get a test failure, it may be because the type the endpoints are configured to consume is incorrect. Open the `ItemEndpoint` class and replace all instances of `@Consumes` with:

src/main/java/com/acme/sellmore/rest/ItemEndpoint.java.

```
@Consumes(MediaType.APPLICATION_XML)
```

Run the tests again. With any luck, this time you’ll be chanting “Green bar!”

7.2. Running the Arquillian tests on OpenShift

Okay, now you can say it. "Let’s take it to the cloud!" If they work there, they’ll work anywhere :)

It’s up to you whether you want to run the tests on the same OpenShift application as the production application or whether you want to create a dedicated application. We’ll assume you’re going to create a dedicated application. Let’s call it ike.

```
$> rhc-create-app -t jbossas-7.0 -a ike
```

You’ll also need an Arquillian profile. The Forge plugin doesn’t honor the OpenShift adapter yet, so you’ll have splice this profile into the pom.xml by hand:

pom.xml (fragment).

```
<profile>
  <id>OPENSHIFT_1.X</id>
  <dependencies>
    <dependency>
      <groupId>org.jboss.arquillian.container</groupId>
      <artifactId>arquillian-openshift-express</artifactId>
      <version>1.0.0.Alpha1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</profile>
```

The Arquillian OpenShift adapter also uses git push to deploy the test archive. In order for that to work, it needs to know where it’s pushing. In other words, it needs a little configuration.

Seed an arquillian.xml descriptor using a known container (in this case, JBoss AS 7 remote):

```
$forge> arquillian configure-container --profile JBOSS_AS_REMOTE_7.X
```

Next, replace the container element with the following XML:

src/test/resources/arquillian.xml (fragment).

```
<container qualifier="OPENSHIFT_1.X">
  <configuration>
    <property name="namespace">mojavelinux</property>
    <property name="application">ike</property>
    <property name="sshUserName">02b0951a5ed54c98b54c41a7f2efbda8</property>
    <!-- Passphrase can be specified using the environment variable SSH_PASSPHRASE -->
    <!-- <property name="passphrase"></property> -->
    <property name="login">dan.j.allen@gmail.com</property>
  </configuration>
</container>
```

You can either put the passphrase for your SSH key in the descriptor or you can export the `SSH_PASSPHRASE` environment variable:

```
$> export SSH_PASSPHRASE=[libra_id_rsa passphrase]
```

To activate this container configuration, write the name of the qualifier to the *arquillian.launch* file (alternatively, you can select the configuration using the `-Darquillian.launch` flag when you run Maven):

```
$> echo "OPENSHIFT_1.X" > src/test/resources/arquillian.launch
```

Are you ready to see some tests run in the cloud?

```
$forge> test --profile OPENSHIFT_1.X
```

You may want to tail the log files in another terminal to moniter the progress:

```
$> rhc-tail-files -a ike
```

If you can’t see the green bar, look above you :)

Chapter 8. Hosting Configuration

8.1. Making your application a top-level domain

Do we expect that you’ll use *.rhcloud.com for all of your public websites? Of course not! That’s where the alias feature comes in.

You can create a domain alias for any OpenShift application using this command:

```
$> rhc-ctl-app -a sellmore -c add-alias --alias sellmore.com
```

Next, you point the DNS for your domain name to the IP address of your OpenShift server (or you can cheat by putting it in */etc/hosts*).

Now you can access the application from the following URL:

http://sellmore.com

Congratulations! You’re OpenShift-hosted.

Chapter 9. Summary

In this tutorial, we learned how to:

- Register an account at OpenShift
- Install the Ruby-based OpenShift client tools
- Create our own OpenShift domain
- Create an OpenShift application using the JBoss AS 7 cartridge on that domain
- Add a remote OpenShift git repo to our own repo to deploy an existing app
- Deploy a Java EE application to OpenShift
- Work with the in-memory database
- Configure H2 to persist the database file to the application’s data directory
- Configure MySQL and phpmyadmin cartridges in OpenShift
- Configure our Java EE application to use the MySQL database running on the OpenShift domain

Chapter 10. Resources

- Git repository for this tutorial
  http://tinyurl.com/dcjbug-jboss-workshop
- OpenShift homepage
  http://openshift.com
- OpenShift dashboard
  https://openshift.redhat.com/app/console/applications
- OpenShift Documentation
  http://docs.redhat.com/docs/en-US/OpenShift/2.0/html/User_Guide/index.html
- OpenShift Knowledge Base
  https://redhat.com/openshift/community/kb
- Installing OpenShift client tools on Linux, Windows and Mac OSX
  https://redhat.com/openshift/community/kb/kb-e1000/installing-openshift-express-client-tools-on-non-rpm-based-systems
- Apps prepared for rapid deployment to OpenShift
  https://www.redhat.com/openshift/community/kb/kb-e1021-rundown-on-the-github-hosted-git-repositories-available-for-rapid-deployment
- OpenShift resources for JBoss AS
  https://www.redhat.com/openshift/community/page/jboss-resources
- JBoss Forge download [https://repository.jboss.org/nexus/index.html#nexus-search;gav~org.jboss.forge~forge-distribution~~~~kw,versionexpand]
- JBoss Java EE quickstarts repository
  https://github.com/jbossas/quickstart
- Deploy a Play! application on OpenShift (provided a lot of details for this workshop)
  https://github.com/opensas/play-demo/wiki/Step-12.5---deploy-to-openshift
- How JBoss AS 7 was configured for OpenShift
  https://community.jboss.org/blogs/scott.stark/2011/08/10/jbossas7-configuration-in-openshift-express