Step 2

BDD Testing and the SpockFramework

2.1 Setup Groovy and the SpockFramework

We are already following the RED-GREEN-REFACTOR pattern of Test Driven Development. We've already used Vert.x' JUnit-compatible library vertx-unit.

At Red Hat's Open Innovation Labs, we prefer BDD style tests. For most apps we like the Cucumber Framework. Asynchronous testing is different though, and Eclipse Vert.x is asynchronous. For this lab we wil use the SpockFramework which has excellent support for asynchronous testing.

The Spock dependencies have already been added to our POM but are listed below so that you can familiarize yourself with them.

```
<!-- Optional dependencies for using Spock -->
<dependency> <!-- use a specific Groovy version rather than the one specified by
spock-core -->
    <groupId>org.codehaus.groovy</groupId>
    <artifactId>groovy-all</artifactId>
    <version>2.4.13</version>
</dependency>
<dependency> <!-- enables mocking of classes (in addition to interfaces) -->
    <groupId>net.bytebuddy</groupId>
    <artifactId>byte-buddy</artifactId>
    <version>1.6.5</version>
    <scope>test</scope>
</dependency>
<dependency> <!-- enables mocking of classes without default constructor (together
with CGLIB) -->
    <groupId>org.objenesis</groupId>
    <artifactId>objenesis</artifactId>
    <version>2.5.1</version>
    <scope>test</scope>
</dependency>
```

Spock is written in Groovy, and we have also added the gmavenx-plugin and the Surefire and JaCoCo plugins for tests and code coverage:

```
<plugin>
    <!-- The gmavenplus plugin is used to compile Groovy code. To learn more about
this plugin,
    visit https://github.com/groovy/GMavenPlus/wiki -->
    <groupId>org.codehaus.gmavenplus</groupId>
    <artifactId>gmavenplus-plugin</artifactId>
    <version>1.6</version>
    <executions>
        <execution>
            <goals>
                <goal>compile</goal>
                <goal>compileTests</goal>
            </goals>
        </execution>
    </executions>
</plugin>
<plugin>    <!-- Configure the Maven SureFire plugin to use Groovy Spec files for test
-->
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.20.1</version>
    <configuration>
        <useFile>false</useFile>
        <includes>
            <include>**/*Spec.java</include>
        </includes>
    </configuration>
</plugin>
<plugin>    <!-- Configure JaCoCo to be able to extract code coverage information -->
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.7.6.201602180812</version>
    <executions>
        <execution>
            <id>jacoco-initialize</id>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
        <execution>
            <id>jacoco-site</id>
            <phase>test</phase>
            <goals>
                <goal>report</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

2.2 Our First Spock Test

Create a directory "src/test/groovy" and add the package, "com/redhat/qcon/insult.".

TIP	You may need to designate this folder as a source folder in your include

Create a new Groovy script, "MainVerticleSpec." The convention in BDD testing is to end your class names with, "Spec."

The behavior we are expecting is that our microservice insults us when we call "/api/insult/v1." Type or paste the following code in the Groovy script:

```
package com.redhat.qcon.insult

import io.vertx.core.Vertx
import io.vertx.core.http.HttpClientResponse
import spock.lang.Shared
import spock.lang.Specification
import spock.util.concurrent.AsyncConditions

class MainVerticleSpec extends Specification {

    @Shared
    Vertx vertx ①

    def setup() {
        vertx = Vertx.vertx() ②
        def async = new AsyncConditions(1)
        vertx.deployVerticle("com.redhat.qcon.insult.MainVerticle", { res ->
            async.evaluate {
                assert res.succeeded()
            }
        })
        async.await(10)③
    }

    def "Test Being Insulted"() {④
        given: "An instance of Spock's AsyncConditions"
            def async = new AsyncConditions(2)

        when: "We make an HTTP request to the server"
            vertx.createHttpClient().get(80, "localhost", "/api/v1/insult").handler({
response ->
                async.evaluate {
                    assert response.statusCode() == 200
                    assert response.getHeader("Content-Type") == "application/json"
                }
                response.bodyHandler({ buffer ->
                    async.evaluate {
                        assert buffer.toJsonObject().getString("insult") ==
"testinsult"
                    }
                })
            }).end()⑤

        then: "We expect to see an insult"
            async.await(10)
    }
}
```

① this is the local instance of Vert.x that will be fired up in the test

② we initialize our local Vert.x

③ Like most things in Eclipse Vert.x deploying the Verticle is asynchronous so we account for this by creating a Spock AsyncCondition and waiting for up to 10 seconds.

④ our test method

⑤ here we use Vert.x' built in WebClient in our test method. The WebClient is useful when calling external http endpoints both in our application and is equally useful in our test cases Run the test in your IDE or directly on the command line with Maven. Your test should fail because MainVerticle isn’t doing anything yet.include