To write a Spring Boot application that prints "Hello, World!" to the console, you will need to complete the following steps:

1. Setting up a new Maven project in your development environment.

mvn archetype:generate -DgroupId=com.example -DartifactId=hello-world -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

2. Adding the Spring Boot starter dependencies to Maven pom.xml file:

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

3. Create a new Java class and annotate it with @SpringBootApplication to enable Spring Boot.

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class HelloWorldApplication {

public static void main(String[] args) {

SpringApplication.run(HelloWorldApplication.class, args);

}

}

4. Creating a new controller class that will handle HTTP requests and return a response. In this example, we will create a simple controller that returns "Hello, World!" when the /welcome URL is accessed.

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

@Controller

public class HelloWorldController {

@RequestMapping("/welcome.html")

@ResponseBody

public String sayHello() {

return "Hello, World!";

}

}

5. Run the main method in the HelloWorldApplication class to start the application. Then, open a web browser and navigate to [http://localhost:8080/](http://localhost:8080/hello)welcome.html

To connect a Spring Boot application to a database using Testcontainers

1. Adding the Testcontainers dependency in our project. In pom.xml file:

<dependency>

<groupId>org.testcontainers</groupId>

<artifactId>testcontainers</artifactId>

<version>1.14.3</version>

<scope>test</scope>

</dependency>

2. Defining the database container in our test code.

spring.datasource.url=jdbc:postgresql://${POSTGRES\_HOST:localhost}:${POSTGRES\_PORT:5432}/redcarpetup

spring.datasource.username=postgres

spring.datasource.password=1234

3. Starting the database container before running our tests

@BeforeClass

public static void startDb() {

postgres.start();

}

4. Stopping the database container after running our tests.

@AfterClass

public static void stopDb() {

postgres.stop();

}

Writing a terraform script to deploy the spring boot application to AWS Lambda

To deploy a Spring Boot application to AWS Lambda using Terraform, we will need to perform the following steps:

1. Create an AWS IAM role with permissions to access Lambda and other necessary AWS resources.
2. Build the Spring Boot application and package it as a ZIP file.
3. Create a Terraform configuration file that defines the Lambda function and the required AWS resources, such as the IAM role and an Amazon S3 bucket to store the application ZIP file.
4. Use the “terraform init” command to initialize the working directory which includes the Terraform configuration files and installs any required plugins.
5. Use the “terraform apply” command to create the resources and deploy the application to AWS Lambda.

I have attached the files.