**Setup up Deployment of Spring Boot "Microservice" Example Project (using Docker , Jenkins , CFM , ECR , ECS )**

Sample Java / Maven / Spring Boot (version 1.5.6) application code is used in this project

GitHub : <https://github.com/k-Richie/SpringCI/tree/main/app>

## **How to Run**

This application is packaged as a war which has Tomcat 8 embedded. No Tomcat or JBoss installation is necessary. You run it using the java -jar command.

* Clone this repository
* Make sure you are using JDK 1.8 and Maven 3.x

Java 1.8 : sudo apt-get install openjdk-8-jdk (ubuntu)

Maven 3.x : sudo apt install maven (ubuntu)

* You can build the project and run the tests by running mvn clean package
* Once successfully built, you can run the service by one of these two methods:

java -jar -Dspring.profiles.active=profile\_name target/spring-boot-rest-example-0.5.0.war

or

mvn spring-boot:run -Drun.arguments="spring.profiles.active=profile\_name"

Default profilename is “test” for in memory database support

**Points to remember**

1. Check the stdout or boot\_example.log file to make sure no exceptions are thrown

2. Once the application runs you should see something like this

Started Application in 22.285 seconds (JVM running for 23.032)

## 

## **About the Service**

The service is just a simple hotel review REST service. It uses an in-memory database (H2) to store the data. You can also do with a relational database like MySQL or PostgreSQL.

By default this service is made to use an external database MySQL

# **HOW we are Running the project with MySQL (external database)**

Here is what i did to back the services with MySQL, for example:

### **In pom.xml add:**

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

When you include this dependency in your Maven project's pom.xml file, Maven will automatically download the MySQL Connector/J library from the Maven Central Repository and make it available for your project to use.

### **Append this to the end of application.yml:(**src/main/resources/application.yml**)**

---

spring:

profiles: mysql

datasource:

driverClassName: com.mysql.jdbc.Driver

url: jdbc:mysql://<your\_mysql\_host\_or\_ip>/bootexample

username: <your\_mysql\_username>

password: <your\_mysql\_password>

jpa:

hibernate:

dialect: org.hibernate.dialect.MySQLInnoDBDialect

ddl-auto: update # todo: in non-dev environments, comment this out:

Hotel.service:

name: 'test profile:'

### **Then run is using the 'mysql' profile:**

java -jar -Dspring.profiles.active=mysql target/spring-boot-rest-example-0.5.0.war

or

mvn spring-boot:run -Drun.jvmArguments="-Dspring.profiles.active=mysql"

Modify the variables in the above code to make it work with your external MySQL database service.

**After your application setup seems to working fine and connectivity with your database looks good, we can move to dockerize the application.**

To dockerize the application, first we must ensure that all the values we are passing to the application are parameterized so that we can pass different values to run the application container according to our need.

**To do that modify the application.yml file**

---

spring:

profiles: mysql

datasource:

driverClassName: com.mysql.jdbc.Driver

**url: jdbc:mysql://${DB\_ENDPOINT}:${DB\_PORT}/${DB\_NAME}?useSSL=false**

**username: ${DB\_USERNAME}**

**password: ${DB\_PASSWORD}**

jpa:

hibernate:

dialect: org.hibernate.dialect.MySQLInnoDBDialect

ddl-auto: update # todo: in non-dev environments, comment this out:

hotel.service:

name: 'test profile:'

**Variables definitions:**

**DB\_ENDPOINT :** Endpoint of the MySQL database engine

**DB\_PORT :** Port at which database allow traffic

**DB\_NAME :** Name of the database

**DB\_USERNAME:** Username for the database source

**DB\_PASSWORD:** Password for the database source

After saving the file after modifications , you can just run:

**mvn clean install package**

This will create a war or jar file of the application along with target folder where it will be located.

**Moving to Dockerization**

**Make sure you have docker installed on your system**

[**https://docs.docker.com/engine/install/ubuntu/**](https://docs.docker.com/engine/install/ubuntu/)

1. Create a Dockerfile: Create a file named "Dockerfile" in the root directory of your Spring Boot project. The Dockerfile contains instructions for building a Docker image for your application.
2. Specify a base image: In the Dockerfile, start by specifying a base image that includes the necessary runtime environment for your application, such as OpenJDK or AdoptOpenJDK. For example:

**FROM openjdk:8**

3. Copy the application files: Use the COPY instruction in the Dockerfile to copy your Spring

Boot application's JAR file into the Docker image. For example:

**COPY target/spring-boot-rest-example-0.5.0.war spring-boot-rest-example-0.5.0.war**

4. Expose ports (if necessary): If your Spring Boot application listens on a specific port, you , can use the EXPOSE instruction to expose that port in the Docker image. For example:

**EXPOSE 8090 8091 3306**

5. Define the command to run the application: Use the CMD or Entrypoint instruction to define the command that will be executed when a container is created from the Docker image. Typically, you'll specify a command to run the Java application using the java -jar command. For example:

**ENTRYPOINT ["java", "-jar", "/spring-boot-rest-example-0.5.0.war"]**

6. Build the Docker image: Open a terminal or command prompt, navigate to the directory containing the Dockerfile, and run the following command to build the Docker image:

**docker build -t my-application-image .**

7. Run the Docker container: Once the Docker image is built, you can run a container from the

image using the following command:

**docker run -p 8090:8090 my-application-image**

*IF EVERYTHING SEEMS TO WORKING FINE WE CAN MOVE TO NEXT STEPS*

**Pushing Docker Image to ECR**

* Install and configure the AWS CLI: Ensure that you have the AWS CLI installed and configured with your AWS credentials. You can install the AWS CLI by following the instructions in the AWS Command Line Interface User Guide.

**aws ecr create-repository -- my-repo** <ecrRepoName>**--region** <Region>

* Grant proper permissions: Ensure that your AWS credentials have the necessary permissions to perform actions on the ECR repository.

ecr:CreateRepository: Allows the user to create an ECR repository.

ecr:PutImage: Allows the user to push Docker images to the ECR repository.

ecr:DescribeRepositories: Allows the user to list and describe ECR repositories.

* Tag the Docker image: After building your Docker image, tag it using the ECR repository URI. Run the following command to tag your image:

**docker tag my-application-image:latest <account-id>.dkr.ecr.<region>.amazonaws.com/my-repo:latest**

* Authenticate Docker with ECR: Run the following AWS CLI command to authenticate Docker with ECR:

**aws ecr get-login-password --region <region> | docker login --username AWS --password-stdin <account-id>.dkr.ecr.<region>.amazonaws.com**

* Push the Docker image to ECR: Run the following command to push the Docker image to the ECR repository:

**docker push <account-id>.dkr.ecr.<region>.amazonaws.com/my-repo:latest**

**Note: Replace <region> with the AWS region where your ECR repository is located and <account-id> with your AWS account ID. Replace my-repo with the name of your ECR repository.**

**Provisioning Infrastructure to run the application on ECS**

GitHub : [https://github.com/k-Richie/SpringCI/tree/main](https://github.com/k-Richie/SpringCI/tree/main/app)

How to get started:

Prerequisite:-Install and configure the AWS CLI: Ensure that you have the AWS CLI installed and configured with your AWS credentials. You can install the AWS CLI by following the instructions in the AWS Command Line Interface User Guide.

https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

STEP1 (**VPC)**: Create a cloudformation template and Parameter file of VPC which consists of Vpc,2 public subnets (subnet-1 and subnet-2) and 2 private subnets (subnet-3 and subnet-4). The internet-facing ALB is launched in subnet-1 and subnet-2. The public subnet-1 hosts a bastion host that provides secure access to instances in private subnets. The private subnets, subnet-3 and subnet-4, host the backend application servers and RDS instances.

Code repository :

* Template repo: <https://github.com/k-Richie/SpringCI/blob/main/template/vpc.yaml>
* Parameter repo:<https://github.com/k-Richie/SpringCI/blob/main/parameters/vpcPara.json>

**Command to Provision it:-**aws cloudformation create-stack --stack-name <your stack name> --template-body file://<your Vpc template file name> --parameters file://parameters/<your Vpc parameter file name> --region <your region>

STEP2 (**RDS)**:Create RDS template and Parameter file for connecting the application with the database.(For this application I have used Mysql).

Code repository :

* Template repo: https://github.com/k-Richie/SpringCI/blob/main/template/rds.yaml
* Parameter repo:<https://github.com/k-Richie/SpringCI/blob/main/parameters/rdsPara.json>

**Command to Provision it:-**aws cloudformation create-stack --stack-name <your stack name> --template-body file://<your RDS template file name> --parameters file://parameters/<your RDS parameter file name> --region <your region>

STEP3 (**CLUSTER)**:Create ECS Cluster where services will be run.(Note:- Choose EC2 Launchtype which manages EC2 instances to host the container).

Code repository :

* Template repo:https://github.com/k-Richie/SpringCI/blob/main/template/ecs-cluster.yaml
* Parameter repo:<https://github.com/k-Richie/SpringCI/blob/main/parameters/clusterPara.json>

**Command to Provision it:-**aws cloudformation create-stack --stack-name <your stack name> --template-body file://<your Cluster template file name> --parameters file://parameters/<your Cluster parameter file name> --region <your region>

STEP4 (**HOST & SERVICE):**Create ECS host and services.(Note:-Choose ECS optimized ami-id while launching the EC2 instance).

Code repository :

* Host Parameter repo:https://github.com/k-Richie/SpringCI/blob/main/parameters/ecs-asgPara.json
* Host Template repo:<https://github.com/k-Richie/SpringCI/blob/main/template/ecs-asg.yaml>
* Service Template repo:https://github.com/k-Richie/SpringCI/blob/main/template/ecs-services.yaml
* Service Parameter repo:https://github.com/k-Richie/SpringCI/blob/main/parameters/servicePara.json

Points to keep in mind:-

* Choose ECS optimized ami-id.
* Attach IAM role ([ecsTaskExecutionRole](https://us-east-1.console.aws.amazon.com/iamv2/home?region=us-east-1#/roles/details/ecsTaskExecutionRole)).
* For further information refer to the following documentation to launch an Amazon ECS Linux container instance**:-**

https://docs.aws.amazon.com/AmazonECS/latest/developerguide/launch\_container\_instance.html

**Command to Provision host:-**aws cloudformation create-stack --stack-name <your stack name> --template-body file://<your Host template file name> --parameters file://parameters/<your Host parameter file name> --region <your region>

**Command to Provision service:-**aws cloudformation create-stack --stack-name <your stack name> --template-body file://<your Service template file name> --parameters file://parameters/<your Service parameter file name> --region <your region>

**Now your service will be up and running.**

You can access the UI page of your application through the DNS of Load Balancer your provisioned. **http://ELB\_DNS:8090/swagger-ui.html**

**Using the Service**

### **Create a hotel resource**

POST /example/v1/hotels

Accept: application/json

Content-Type: application/json

{

"name" : "C1class",

"description" : "Very basic, small rooms but clean",

"city" : "Delhi",

"rating" : 4

}

RESPONSE: HTTP 201 (Created)

Location header: [http://DNS:8090/example/v1/hotels/1](http://dns:8090/example/v1/hotels/1)

### **Update a hotel resource**

PUT /example/v1/hotels/1

Accept: application/json

Content-Type: application/json

{

"name" : "C1class",

"description" : "Very basic, big rooms but clean",

"city" : "Santa Ana",

"rating" : 3

}

RESPONSE: HTTP 204 (No Content)

### **Retrieve a paginated list of hotels**

http://lDNS:8090/example/v1/hotels?page=0&size=10

Response: HTTP 200

Content: paginated list