

Name: Yash Jayeshkumar Pandya  
Email : yashp614929@gmail.com  
Project : Course Project: Application Deployment Using Docker

---

## **Overview**

The project `Application Deployment Using Docker` is used to deploy the project entitled 'Hotel Room Booking Application' which was implemented in a previous module/course to book rooms for a single hotel. In this project, I have created a microservices based backend application using spring boot framework. The deployment using docker mainly has the same project code structure and addition of deployment files. Ie. **Dockerfile** and **docker-compose** file.

The project is divided into three different microservices, which are as follows:

### Booking service

This service is exposed to the outer world and is responsible for collecting all information related to user booking.

### Payment service

This is a dummy payment service; this service is called by the booking service for initiating payment after confirming rooms.

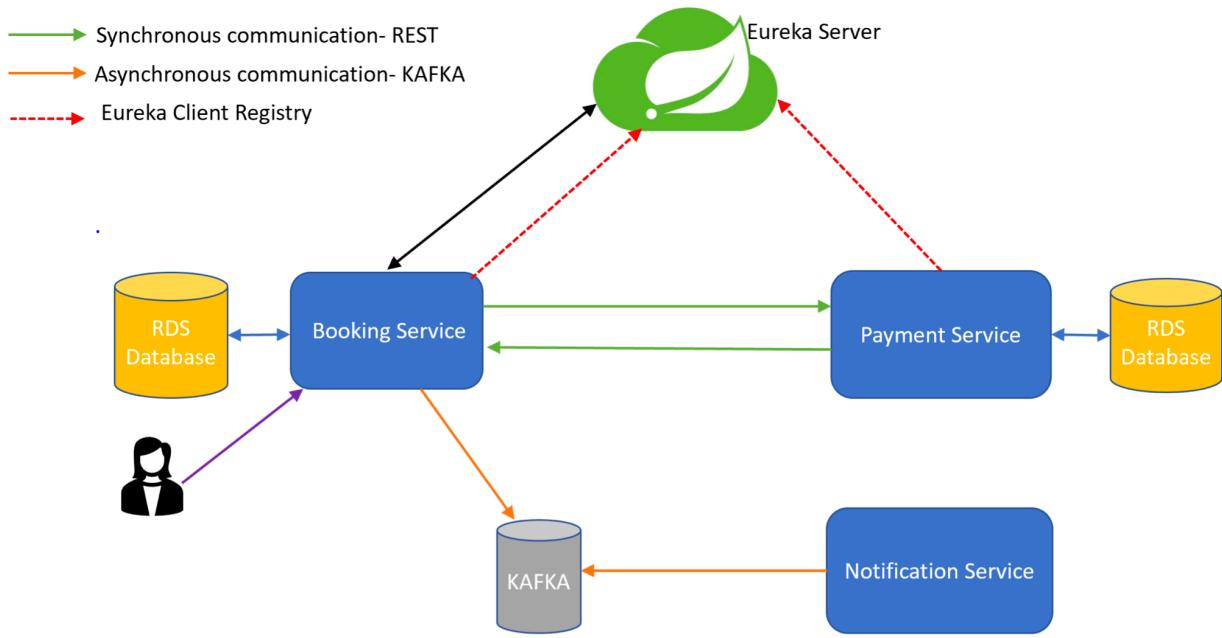
### Notification service

When payment is done and all data is saved properly, the booking service puts a message object on the Kafka topic, and then, the notification service reads that message from the Kafka topic and prints the message to the console.

## **Application Workflow**

Initially, the Booking service and Payment service register themselves on the Eureka server. Note that the Booking service interacts with the Notification service asynchronously and therefore, the Notification service need not register itself as a Eureka client.

The user initiates room booking using the 'Booking' service by providing information such as toDate, fromDate, aadharNumber and number of rooms required (number of Rooms).



The 'Booking' service returns back the list of room numbers and price associated and prompts the user to enter the payment details if they wish to continue ahead with the booking. It also stores the details provided by the user in its database.

If the user wants to go ahead with the booking, then they can simply provide the payment related details like bookingMode, upiId/ cardNumber to the 'Booking' service which will be further sent to the 'Payment' service. Based on this data, the payment service will perform a dummy transaction and return back the transaction Id associated with the transaction to the Booking service. All the information related to transactions is saved in the RDS database of the payment service.

With the help of Kafka and the Notification service, we send a notification to the user after the transaction is confirmed. (prints on the console of 'Notification' service, in this case)

## **Solution Architecture:**

All Services are combined into a single directory called 'Sweet-Home'. Booking service, Payment service and Eureka are a type of Spring boot application whereas Notification service is a simple java maven project.

Deployment cycle contains the following resources.

- AWS VPC
- AWS RDS
- AWS EC2
- AWS Elastic IPs
- AWS Security Groups
- Apache kafka
- MySQLWorkbench
- Postman

Solution approach is to **use a separate EC2 instance to install the apache kafka** and **do not containerize the kafka using the docker-compose**. Ie. **externalized kafka**. The connection to kafka is done using the public IP of the EC2 instance which is associated with the Elastic IP.



Firstly, a new VPC is created to launch resources that are being used while setting up the project.

The screenshot shows the AWS VPC Management Console interface. On the left, there's a sidebar with various navigation options under 'Your VPCs' and other sections like Security, Reachability, and DNS Firewall. The main area displays a table titled 'Your VPCs' with two entries:

Name	VPC ID	State	IPv4 CIDR	IPv6 CIDR (Network border group)	IPv6 pool
vpc-deeb52a3	vpc-deeb52a3	Available	172.31.0.0/16	-	-
docker-vpc	vpc-0d1bd31dce45ca543	Available	10.0.0.0/16	-	-

Two EC2 instances are created with the type t2.medium.

The screenshot shows the AWS EC2 Management Console interface. The left sidebar includes options for Instances, Images, Elastic Block Store, and Network & Security. The main content area displays a table titled 'Instances' with two entries:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...
kafka-ec2	i-0ff0ee27f22037dc4	Running	t2.medium	2/2 checks passed	No alarms	us-east-1e	ec2-54-159-178-122.co...	54.159.178.122
docker-ec2	i-022a1735cc53a5a42	Running	t2.medium	2/2 checks passed	No alarms	us-east-1e	ec2-52-1-204-116.com...	52.1.204.116

- **docker-ec2** instance is to deploy the application code using docker.
- **Kafka-ec2** instance is to install apache kafka for asynchronous communication for the booking and notification service.

New security group **docker-sg** is created and below inbound rules are added under this security group. This security group is associated with the **docker-ec2** where the application code is being deployed.

Note that all inbound rules are restricted to allow access only within developer/student's IP.

## Ports

- **22** is open to gain access to **docker-ec2** instance.
- **80** is open for all HTTP requests.
- **443** is open for all HTTPS requests.
- **8761** is open to access Eureka.
- **8080** is open to access booking service.
- **8083** is open to access payment service.

Name	Security group ID	Security group name	VPC ID	Description	Owner	Inbound rules count
sg-02a5c9c7fe15f9e4	docker-rds-sg	vpc-0d1bd31dce43ca343	Created by RDS manag...	S47022623928	3 Permission entries	
sg-067062d641cc5c63e	docker-sg	vpc-0d1bd31dce43ca343	docker-sg	S47022623928	6 Permission entries	
sg-0b25c3b5c1fc3e7c5	default	vpc-0d1bd31dce43ca343	default VPC security gr...	S47022623928	1 Permission entry	
sg-0f80caa4bb8b8ccdae7	kafka-sg	vpc-0d1bd31dce43ca343	kafka-sg	S47022623928	10 Permission entries	

Security group rule...	IP version	Type	Protocol	Port range	Source	Description
sgr-04994c215bdd1...	IPv4	Custom TCP	TCP	8761	173.177.134.208/32	Eureka
sgr-0c4586a673425d...	IPv4	SSH	TCP	22	173.177.134.208/32	-
sgr-0f12153fa40837ef17	IPv4	HTTP	TCP	80	173.177.134.208/32	-
sgr-0655d9a143916ecf	IPv4	Custom TCP	TCP	8080	173.177.134.208/32	booking-service
sgr-0842b99823d147...	IPv4	Custom TCP	TCP	8083	173.177.134.208/32	payment-service
sgr-0c835635d16156...	IPv4	HTTPS	TCP	443	173.177.134.208/32	-

Another security group **kafka-sg** is created and below inbound rules are added under this security group. This security group is associated with the **kafka-ec2** where the apache kafka is being deployed.

Note that all inbound rules are publicly opened.

## Ports

- **22** is open to gain access to **kafka-ec2** instance.
- **80** is open for all HTTP requests.
- **443** is open for all HTTPS requests.
- **2181** is open for apache kafka.
- **9092** is open for netcat.

The screenshot shows the AWS EC2 Management Console with the 'Security Groups' section selected. It displays a list of security groups, including 'docker-rds-sg', 'docker-sg', 'default', and 'kafka-sg'. The 'kafka-sg' row is highlighted with a blue selection bar. Below this, the 'Inbound rules (10)' section is shown, listing various ports and protocols that are publicly accessible. The rules include entries for port 443 (HTTPS), port 22 (SSH), port 80 (HTTP), port 2181 (Custom TCP), port 9092 (Custom TCP), and port 22 (SSH) again.

Name	Security group ID	Security group name	VPC ID	Description	Owner	Inbound rules count
-	sg-02a5c9c7ffe15f9e4	docker-rds-sg	vpc-0d1bd31dce43ca343	Created by RDS manag...	547022623928	3 Permission entries
-	sg-067d62d641c5cc63e	docker-sg	vpc-0d1bd31dce43ca343	docker-sg	547022623928	6 Permission entries
-	sg-0b25c3b5c1fc3e7c5	default	vpc-0d1bd31dce43ca343	default VPC security gr...	547022623928	1 Permission entry
<b>✓</b>	<b>sg-0f80ca4b8b8ccdae7</b>	<b>kafka-sg</b>	<b>vpc-0d1bd31dce43ca343</b>	<b>kafka-sg</b>	<b>547022623928</b>	<b>10 Permission entries</b>

Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Des
-	sgr-0f5af4d63104f0ee0	IPv4	HTTPS	TCP	443	0.0.0.0/0	-
-	sgr-0517ac104f35a2791	IPv6	SSH	TCP	22	::/0	-
-	sgr-038b4810654a8a...	IPv6	HTTP	TCP	80	::/0	-
-	sgr-0ae67ec72608f159d	IPv4	Custom TCP	TCP	2181	0.0.0.0/0	-
-	sgr-0cae67e76224a851	IPv6	HTTPS	TCP	443	::/0	-
-	sgr-03b6324fb6dbbf30f	IPv6	Custom TCP	TCP	2181	::/0	-
-	sgr-0de2892af9c25aec1	IPv4	Custom TCP	TCP	9092	0.0.0.0/0	-
-	sgr-06ea352357d9dc5...	IPv4	HTTP	TCP	80	0.0.0.0/0	-
-	sgr-0a53d1369fe879c74	IPv6	Custom TCP	TCP	9092	::/0	-
-	sgr-0a116969cf75bb856	IPv4	SSH	TCP	22	0.0.0.0/0	-

## EC2 instance connection

The single pem file is used to connect with both the ec2 instances. docker-ec2 and kafka-ec2.

```
ssh -i "docker-dep.pem" ubuntu@54.159.178.122
ssh -i "docker-dep.pem" ubuntu@52.1.204.116
```

The public IPs are used to connect with EC2 instances using the ssh command. However, the IP gets changed when the EC2 instance is stopped. In-order to make a common IP until the project deployment is completed, Elastic IPs are used. Each EC2 instance is associated with the elastic IP.

The screenshot shows the AWS EC2 Management Console interface. On the left, there's a sidebar with navigation links for EC2 Dashboard, Events, Tags, Limits, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Scheduled Instances, Capacity Reservations, Images, AMIs, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Network & Security, Security Groups, Elastic IPs (selected), Placement Groups, and Key Pairs. The main content area is titled "Elastic IP addresses (2)". It contains a table with two rows:

Name	Allocated IPv4 address	Type	Allocation ID	Associated instance ID	Private IP address	Associate
docker-ec2-ip	52.1.204.116	Public IP	eipalloc-0b6d594b6c6b42a15	i-022a1735cc33a5a42	10.0.0.90	eipass
kafka-ec2-ip	54.159.178.122	Public IP	eipalloc-04bcdadb0763caa12	i-0fffee27f22037dc4	10.0.0.179	eipass

## Docker and Docker-Compose Installation

docker is installed on the **docker-ec2** instance using the docker installation process and verifies the installation using the below command.

```
which docker  
docker -v
```

docker-compose is installed on the **docker-ec2** instance using the docker-compose installation process and verifies the installation using the below command.

```
docker-compose --version
```

A screenshot of a macOS desktop environment. At the top, a terminal window titled 'console.aws.amazon.com' is open, showing the command-line output:

```
ubuntu@ip-10-0-0-90:~$ which docker
/usr/bin/docker
ubuntu@ip-10-0-0-90:~$ docker --version
Docker Version 20.10.8, build 3967b7d
ubuntu@ip-10-0-0-90:~$
```

The main screen shows a browser window for 'console.aws.amazon.com'. The URL bar indicates the user is on the 'Downloads' page. The page content includes a sidebar with 'Security Groups', 'Elastic IPs', 'Placement Groups', and 'Key Pairs'. The main area displays the message 'Select an instance above'. Below the browser is a dock containing various application icons.

A screenshot of a macOS desktop environment, similar to the one above. A terminal window titled 'console.aws.amazon.com' is open, showing the command-line output:

```
ubuntu@ip-10-0-0-90:~/IIIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications$ ssh -i docker-dep.pem ubuntu@ec2-52-1-204-116.compute-1.amazonaws.com
Downloads — ubuntu@ip-10-0-0-90:~/IIIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications — ssh -i docker-dep.pem ubuntu@ec2-52-1-204-116.compute-1.amazonaws.com +
```

```
ubuntu@ip-10-0-0-90:~/IIIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications$ docker-compose --version
docker-compose version 1.17.1, build unknown
ubuntu@ip-10-0-0-90:~/IIIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications$
```

The main screen shows a browser window for 'console.aws.amazon.com'. The URL bar indicates the user is on the 'Downloads' page. The page content includes a sidebar with 'Security Groups', 'Elastic IPs', 'Placement Groups', and 'Key Pairs'. The main area displays the message 'Select an instance above'. Below the browser is a dock containing various application icons.

## RDS

The RDS database instance named **docker-app-rds** is created using the MySQL Engine. The RDS is created within the **docker-vpc**.

The screenshot shows the AWS RDS Management Console interface. On the left, there's a sidebar with various navigation options like Dashboard, Databases, Query Editor, etc. The main area is titled 'docker-app-rds' under 'Databases'. It has a 'Summary' section with details such as DB identifier (docker-app-rds), CPU usage (3.46%), Status (Available), Class (db.t2.micro), Role (Instance), Current activity (2 Connections), Engine (MySQL Community), and Region & AZ (us-east-1e). Below the summary is a tabbed section for 'Connectivity & security' which is currently selected. It displays endpoint information (Endpoint: docker-app-rds.cj1tpsfjqp8i.us-east-1.rds.amazonaws.com, Port: 3306), networking details (Availability zone: us-east-1e, VPC: docker-vpc (vpc-0d1bd31dce43ca343)), and security group associations (Subnet group: default-vpc-0d1bd31dce43ca343, Subnets: subnet-073b89fa998f1dc9a, subnet-0e7d6b4bf30ef39e1). A 'Security group rules (4)' section is also present. At the bottom, there's a standard Mac OS X dock with various application icons.

The security group **docker-rds-sg** is created with below inbound rules and associated with **docker-app-rds** RDS instance.

## Ports

- **3306** is open to connect RDS from developer/student's IP.
- **3306** is open to connect from the docker-sg security group.

The two databases are created using the MySQLWorkbench in the RDS database instance as shown below.

Booking service uses the below database.

- SweetHomeBooking

Payment service uses the below database.

- SweetHomePayment

**Security Groups (1/5) Info**

Name	Security group ID	Security group name	VPC ID	Description	Owner	Inbound rules count
sg-02a5c9c7ffe15f9e4	docker-rds-sg	Created by RDS manag...	vpc-0d1bd31dce43ca343	Created by RDS manag...	S47022623928	3 Permission entries
sg-067d62d641cc5c63e	docker-sg	Created by RDS manag...	vpc-0d1bd31dce43ca343	Created by RDS manag...	S47022623928	6 Permission entries
sg-0b25c3b5c1fc3e7c5	default	default VPC security gr...	vpc-0d1bd31dce43ca343	default VPC security gr...	S47022623928	1 Permission entry
sg-0f80ca4b8b8ccdae7	kafka-sg	Created by RDS manag...	vpc-0d1bd31dce43ca343	Created by RDS manag...	S47022623928	10 Permission entries

**sg-02a5c9c7ffe15f9e4 - docker-rds-sg**

**Inbound rules (3)**

Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description
sgr-0535cbfb16999f5	IPv4	MySQL/Aurora	TCP	3306	173.177.134.208/32	-	-
sgr-0f3a31392f66d387a	-	MySQL/Aurora	TCP	3306	sg-067d62d641cc5c63...	-	-
sgr-0282efcedbb541558	-	MySQL/Aurora	TCP	3306	sg-0f80ca4b8b8ccdae...	-	-

RDS is created with username as admin and password as upgrad123

**Amazon RDS**

**Databases**

**Query Editor**

**Performance Insights**

**Snapshots**

**Automated backups**

**Reserved instances**

**Proxies**

**Events**

**Event subscriptions**

**Recommendations**

**Certificate update**

**MySQL Workbench**

**Schemas**

No object selected

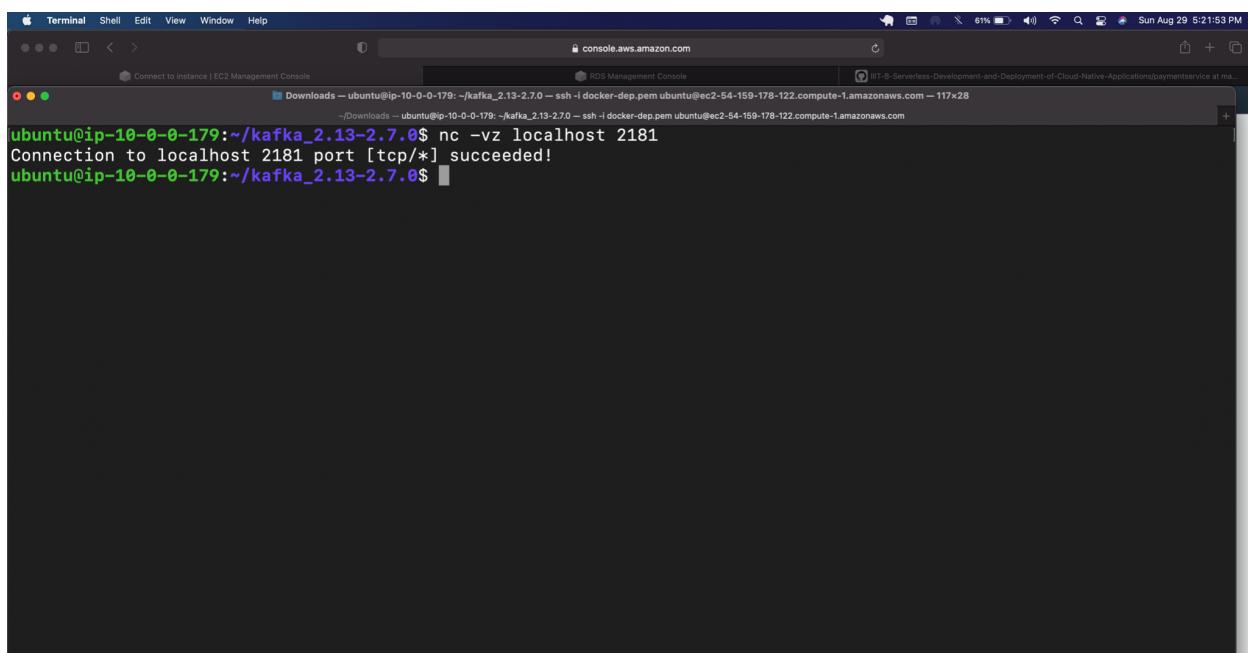
SQL Editor Opened.

## Kafka Installation

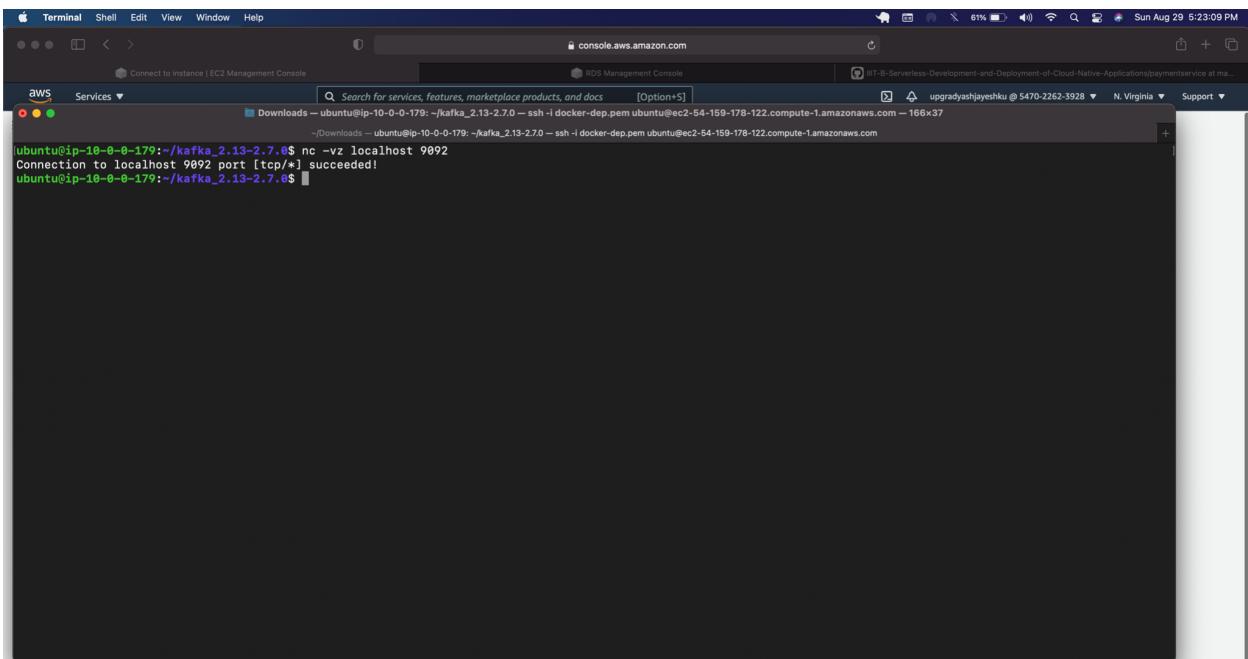
```
bin/zookeeper-server-start.sh config/zookeeper.properties
```

As described above **kafka-ec2** instance is used to install the apache kafka and run the kafka asynchronous communication for the notification service

Kafka and Netcat are installed using the installation process. The properties file is configured with the elastic IP of **kafka-ec2** instance. Both kafka and zookeeper are running and verified using netcat as shown below.



```
ubuntu@ip-10-0-0-179:~/kafka_2.13-2.7.0$ nc -vz localhost 2181
Connection to localhost 2181 port [tcp/*] succeeded!
```



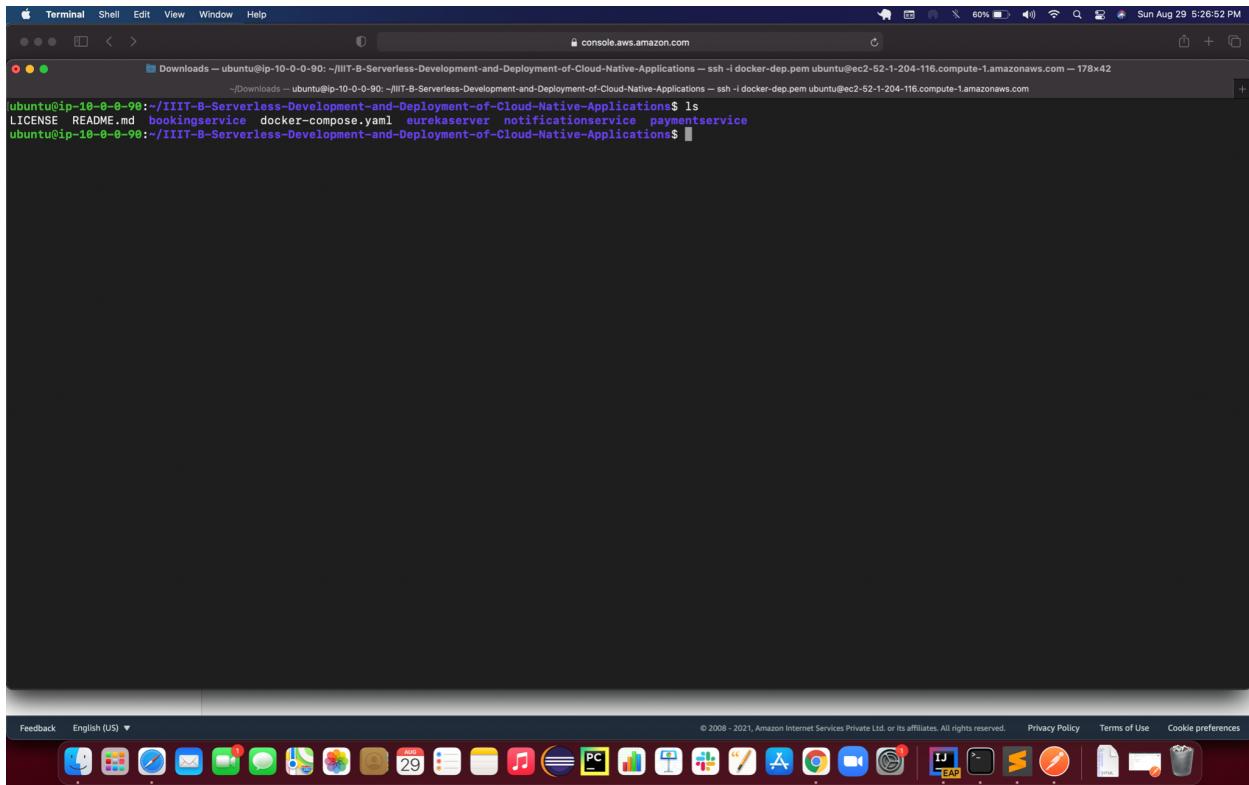
```
ubuntu@ip-10-0-0-179:~/kafka_2.13-2.7.0$ nc -vz localhost 9092
Connection to localhost 9092 port [tcp/*] succeeded!
```

## Application Deployment and Run instructions

The application code including the Dockerfiles and docker-compose file are grouped together and pushed to the github private repository. The codebase is pulled on to the docker-ec2 instance using the below command.

```
git clone  
https://github.com/ypandyaa614929/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications.git
```

Please note that this is the private repository created under the developer/student's github account in order to simplify the process.



There are no docker images present initially and there are no containers running on the docker-ec2 instance. Both checks are verified using below commands.

```
sudo docker images
```

```
sudo docker ps -a
```

```
Terminal Shell Edit View Window Help
console.aws.amazon.com
Downloads — ubuntu@ip-10-0-0-90: ~/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications — ssh -i docker-dep.pem ubuntu@ec2-52-1-204-116.compute-1.amazonaws.com — 178x42
~/Downloads — ubuntu@ip-10-0-0-90: ~/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications — ssh -i docker-dep.pem ubuntu@ec2-52-1-204-116.compute-1.amazonaws.com
ubuntu@ip-10-0-0-90:~/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications$ sudo docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
ubuntu@ip-10-0-0-90:~/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications$
```

Feedback English (US) ▾

© 2008 – 2021, Amazon Internet Services Private Ltd. or its affiliates. All rights reserved. Privacy Policy Terms of Use Cookie preferences

```
Terminal Shell Edit View Window Help
console.aws.amazon.com
Downloads — ubuntu@ip-10-0-0-90: ~/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications — ssh -i docker-dep.pem ubuntu@ec2-52-1-204-116.compute-1.amazonaws.com — 178x42
~/Downloads — ubuntu@ip-10-0-0-90: ~/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications — ssh -i docker-dep.pem ubuntu@ec2-52-1-204-116.compute-1.amazonaws.com
ubuntu@ip-10-0-0-90:~/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications$ sudo docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
ubuntu@ip-10-0-0-90:~/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications$
```

Feedback English (US) ▾

© 2008 – 2021, Amazon Internet Services Private Ltd. or its affiliates. All rights reserved. Privacy Policy Terms of Use Cookie preferences

## BookingService Image

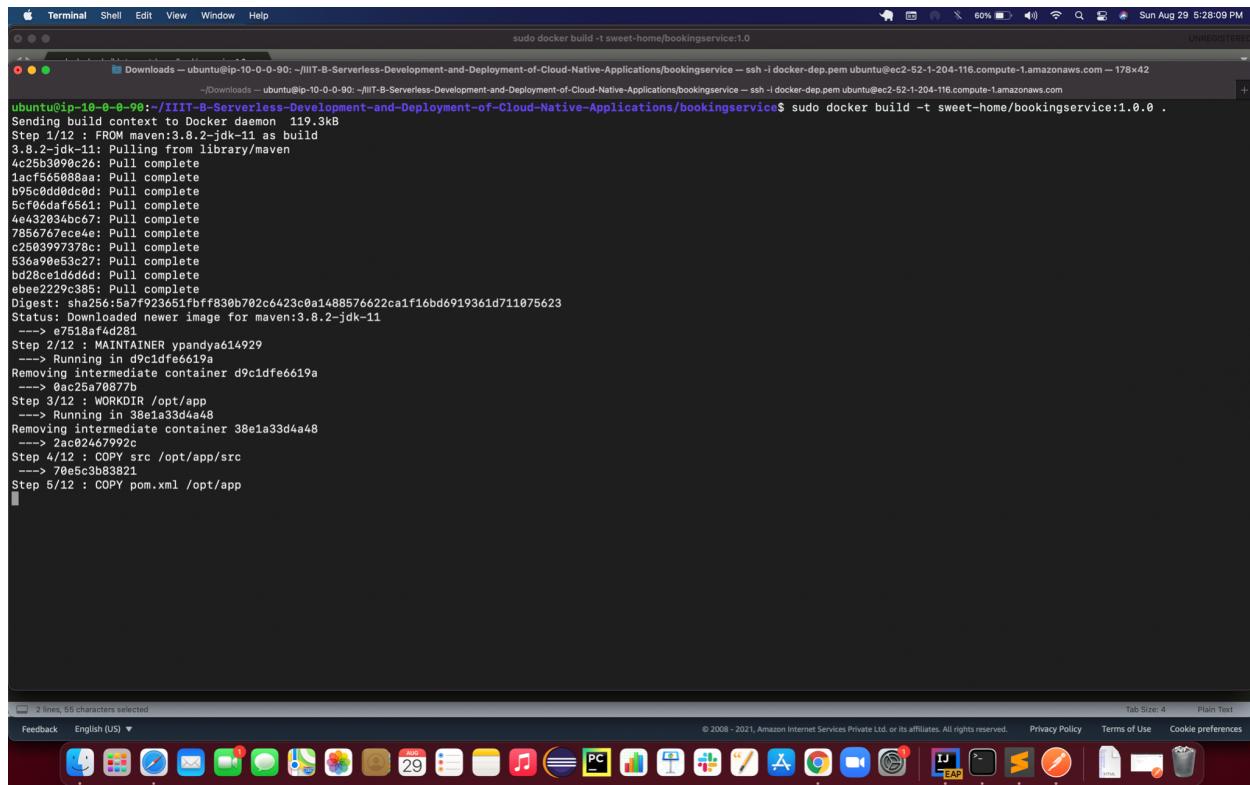
The below commands are present in the Booking service Dockerfile.

```
# 1
FROM maven:3.8.2-jdk-11 as build
MAINTAINER ypandyaa614929
WORKDIR /opt/app
COPY src /opt/app/src
COPY pom.xml /opt/app
RUN mvn -f /opt/app/pom.xml clean package

# 2
FROM openjdk:14-jdk-alpine
MAINTAINER ypandyaa614929
WORKDIR /opt/app
COPY --from=build /opt/app/target/bookingService.jar
/opt/app/bookingService.jar
EXPOSE 8080
ENTRYPOINT [ "java", "-jar", "/opt/app/bookingService.jar"]
```

The docker image of the booking service is built using the following command.

```
sudo docker build -t sweet-home/bookingService:1.0.0 .
```



The screenshot shows a macOS terminal window with the title bar "Terminal". The window contains the command "sudo docker build -t sweet-home/bookingService:1.0 .". The terminal output shows the build process for the BookingService image:

```
sudo docker build -t sweet-home/bookingService:1.0
[sudo] password for ubuntu: 
Sending build context to Docker daemon 119.3kB
Step 1/12 : FROM maven:3.8.2-jdk-11 as build
3.8.2-jdk-11: Pulling from library/maven
4c25b3099c26: Pull complete
1acf565088aa: Pull complete
b95c0d9d0dc9d: Pull complete
5cF04d4af6561: Pull complete
4e432934bc67: Pull complete
7856767ece4e: Pull complete
c250397378c: Pull complete
536a90e53c27: Pull complete
bd28ce1d6d6d: Pull complete
ebee229c385: Pull complete
Digest: sha256:b57f923651fbff838b782c6423c0a1488576622ca1f16bd6919361d711075623
Status: Downloaded newer image for maven:3.8.2-jdk-11
--> e7518af4d281
Step 2/12 : MAINTAINER ypandyaa614929
--> Running in d9c1dfe6619a
Removing intermediate container d9c1dfe6619a
--> 9ac25a70877b
Step 3/12 : WORKDIR /opt/app
--> Running in 38e1a33d4a48
Removing intermediate container 38e1a33d4a48
--> 2ac024467992c
Step 4/12 : COPY src /opt/app/src
--> 7085c3b83821
Step 5/12 : COPY pom.xml /opt/app
```

## PaymentService Image

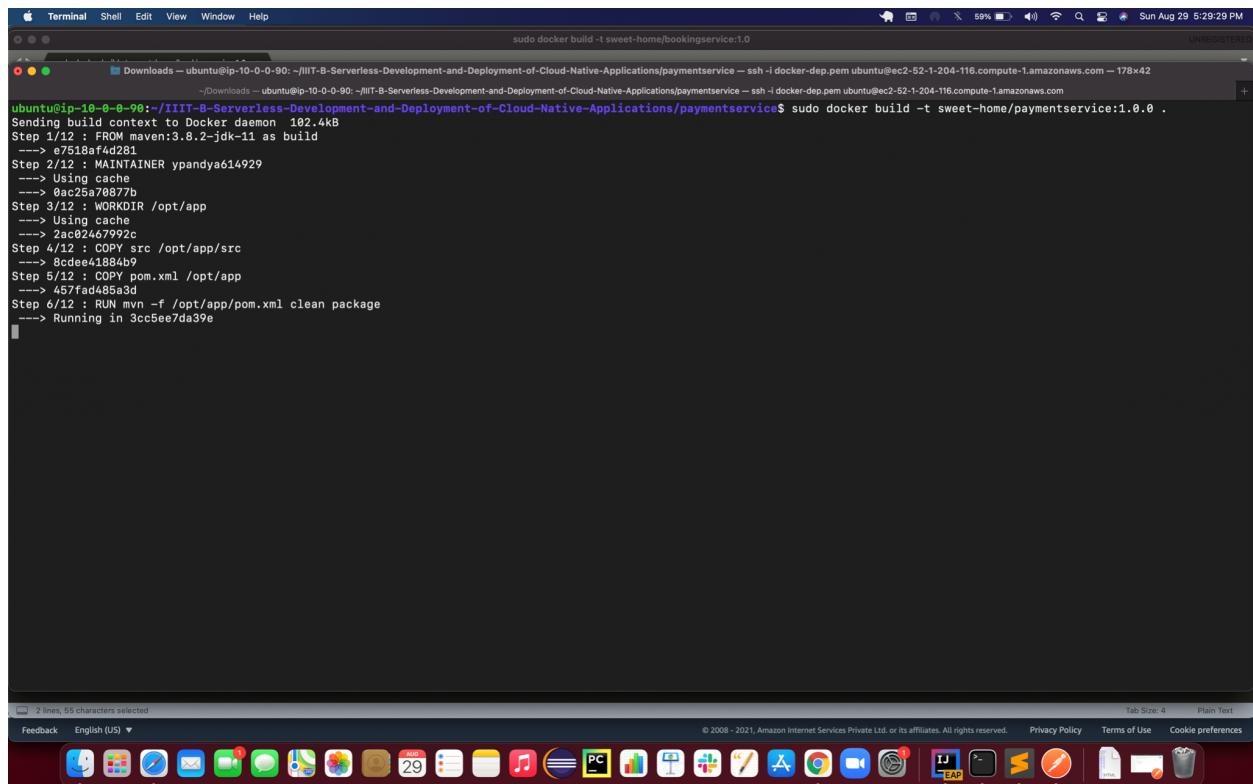
The below commands are present in the Payment service Dockerfile.

```
# 1
FROM maven:3.8.2-jdk-11 as build
MAINTAINER ypandyaa614929
WORKDIR /opt/app
COPY src /opt/app/src
COPY pom.xml /opt/app
RUN mvn -f /opt/app/pom.xml clean package

# 2
FROM openjdk:14-jdk-alpine
MAINTAINER ypandyaa614929
WORKDIR /opt/app
COPY --from=build /opt/app/target/paymentService.jar
/opt/app/paymentService.jar
EXPOSE 8083
ENTRYPOINT [ "java", "-jar", "/opt/app/paymentService.jar"]
```

The docker image of the payment service is built using the following command.

```
sudo docker build -t sweet-home/paymentservice:1.0.0 .
```



The screenshot shows a macOS terminal window with the title bar "Terminal". The window contains the command "sudo docker build -t sweet-home/paymentservice:1.0 ." followed by the Docker build output. The output shows the steps of the build process, including pulling the Maven base image, copying the application code, and running Maven to build the project. The terminal window is part of the Dock, and the status bar at the bottom indicates the date and time as "Sun Aug 29 5:29:29 PM".

```
sudo docker build -t sweet-home/paymentservice:1.0
ubuntu@ip-10-0-0-90:/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications/paymentservice$ sudo docker build -t sweet-home/paymentservice:1.0.0 .
Sending build context to Docker daemon 102.4kB
Step 1/12 : FROM maven:3.8.2-jdk-11 as build
--> e7518af4d281
Step 2/12 : MAINTAINER ypandyaa614929
--> 8ac25a70877b
Step 3/12 : WORKDIR /opt/app
--> Using cache
--> 2ac02467992c
Step 4/12 : COPY src /opt/app/src
--> 8cdee41884b9
Step 5/12 : COPY pom.xml /opt/app
--> 457fad485a3d
Step 6/12 : RUN mvn -f /opt/app/pom.xml clean package
--> Running in 3cc5ee7da39e
```

## NotificationService Image

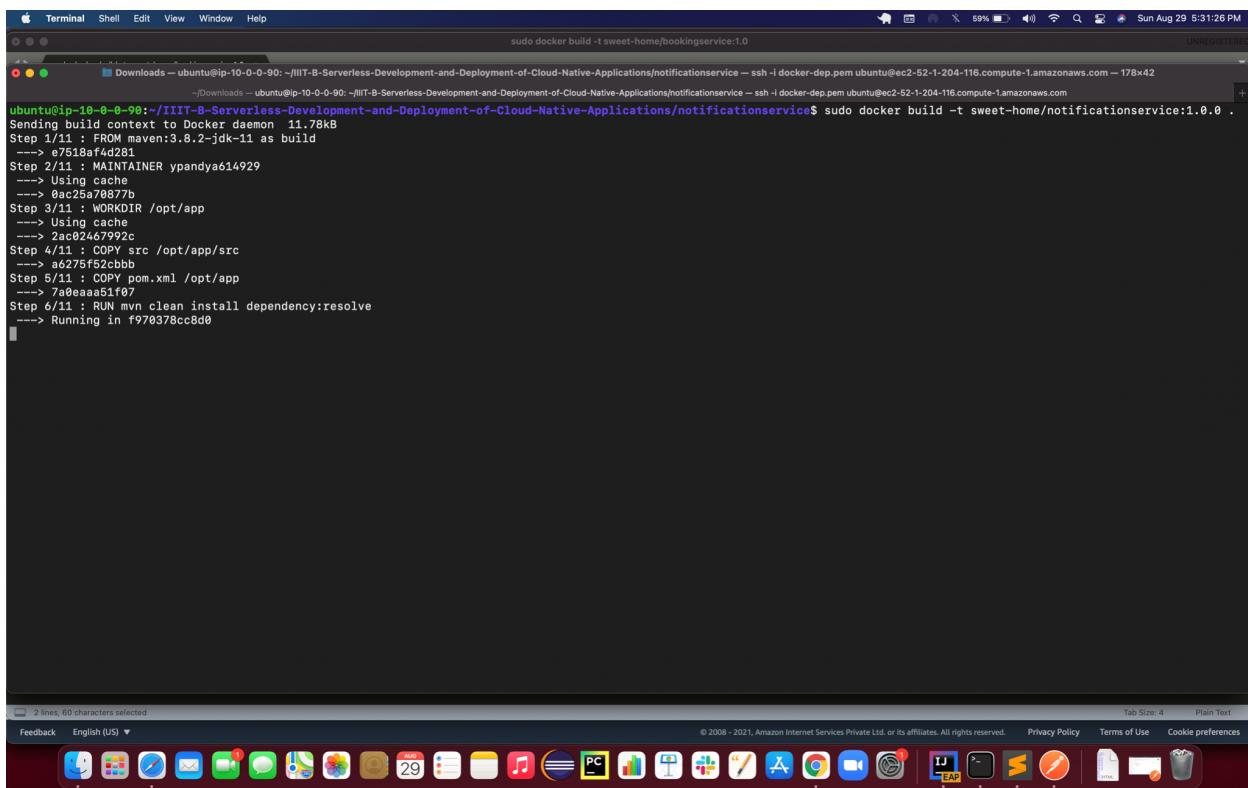
The below commands are present in the Notification service Dockerfile.

```
# 1
FROM maven:3.8.2-jdk-11 as build
MAINTAINER ypandyaa614929
WORKDIR /opt/app
COPY src /opt/app/src
COPY pom.xml /opt/app
RUN mvn clean install dependency:resolve

# 2
FROM openjdk:14-jdk-alpine
MAINTAINER ypandyaa614929
WORKDIR /opt/app
COPY --from=build
/opt/app/target/notificationService-jar-with-dependencies.jar
/opt/app/notificationService-jar-with-dependencies.jar
ENTRYPOINT [ "java", "-jar",
"/opt/app/notificationService-jar-with-dependencies.jar"]
```

The docker image of the notification service is built using the following command.

```
sudo docker build -t sweet-home/notificationservice:1.0.0 .
```



```
sudo docker build -t sweet-home/notificationservice:1.0.0 .
ubuntu@ip-10-0-0-90:~/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications/notificationservice$ sudo docker build -t sweet-home/notificationservice:1.0.0 .
Sending build context to Docker daemon 11.78kB
Step 1/11 : FROM maven:3.8.2-jdk-11 as build
--> e7518af4d281
Step 2/11 : MAINTAINER ypandyaa614929
--> Using cache
--> 9ac25a70877b
Step 3/11 : WORKDIR /opt/app
--> Using cache
--> 2ac02467992c
Step 4/11 : COPY src /opt/app/src
--> a6275ff52cbbb
Step 5/11 : COPY pom.xml /opt/app
--> 7a0eaaa51f07
Step 6/11 : RUN mvn clean install dependency:resolve
--> Running in f970378cc8d0
```

## EurekaServer Image

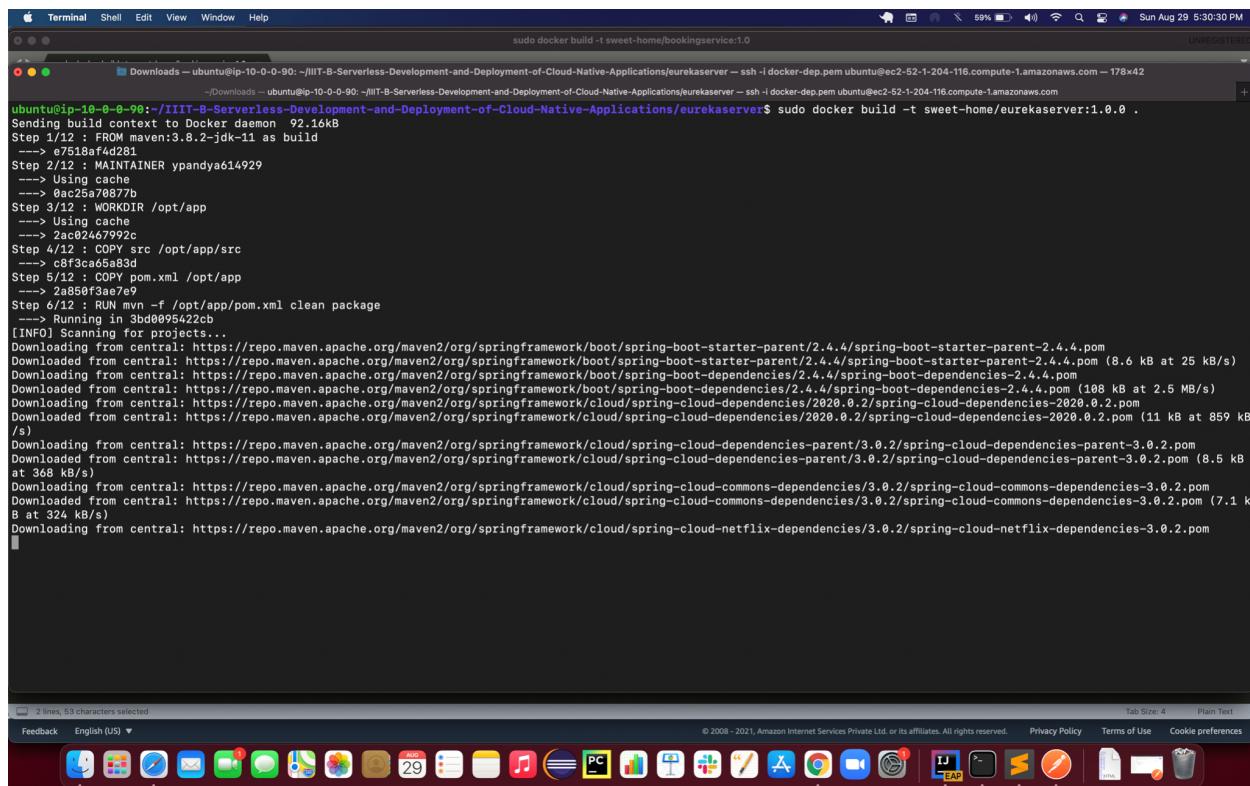
The below commands are present in the Eureka server Dockerfile.

```
# 1
FROM maven:3.8.2-jdk-11 as build
MAINTAINER ypandyaa614929
WORKDIR /opt/app
COPY src /opt/app/src
COPY pom.xml /opt/app
RUN mvn -f /opt/app/pom.xml clean package

# 2
FROM openjdk:14-jdk-alpine
MAINTAINER ypandyaa614929
WORKDIR /opt/app
COPY --from=build /opt/app/target/eurekaServer.jar
/opt/app/eurekaServer.jar
EXPOSE 8761
ENTRYPOINT [ "java", "-jar", "/opt/app/eurekaServer.jar"]
```

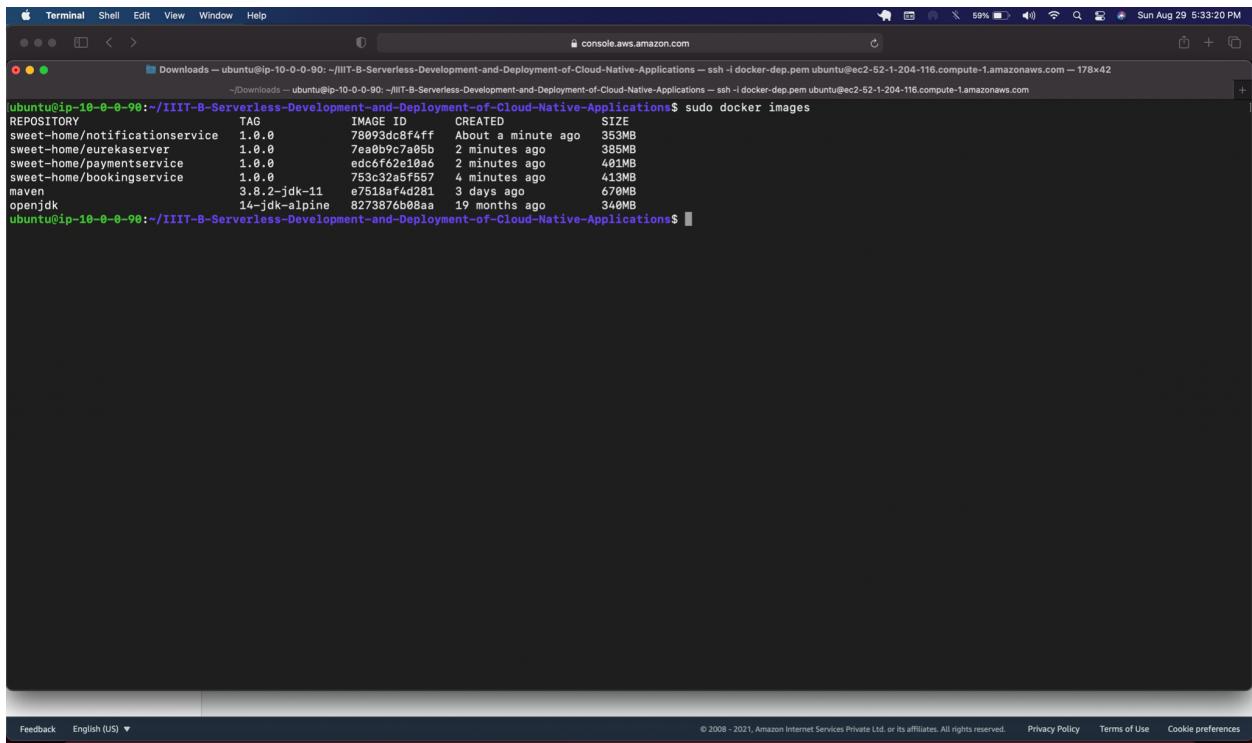
The docker image of the eureka server is built using the following command.

```
sudo docker build -t sweet-home/eurekaserver:1.0.0 .
```



```
sudo docker build -t sweet-home/bookingService:1.0.0 .
[sudo] password for ubuntu: 
Sending build context to Docker daemon 92.16kB
Step 1/12 : FROM maven:3.8.2-jdk-11 as build
--> e7518af4d281
Step 2/12 : MAINTAINER ypandyaa614929
--> Using cache
--> 9ac25a70877b
Step 3/12 : WORKDIR /opt/app
--> Using cache
--> 2ac02467992c
Step 4/12 : COPY src /opt/app/src
--> c8f3ca65a83d
Step 5/12 : COPY pom.xml /opt/app
--> 2a850f3ae7e9
Step 6/12 : RUN mvn -f /opt/app/pom.xml clean package
--> Running in 3bd0095422cb
[INFO] Scanning for projects...
Downloading from central: https://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-starter-parent/2.4.4/spring-boot-starter-parent-2.4.4.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-starter-parent/2.4.4/spring-boot-starter-parent-2.4.4.pom (8.6 kB at 25 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-dependencies/2.4.4/spring-boot-dependencies-2.4.4.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-dependencies/2.4.4/spring-boot-dependencies-2.4.4.pom (108 kB at 2.5 MB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/springframework/cloud/spring-cloud-dependencies/2020.0.2/spring-cloud-dependencies-2020.0.2.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/springframework/cloud/spring-cloud-dependencies/2020.0.2/spring-cloud-dependencies-2020.0.2.pom (11 kB at 859 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/springframework/cloud/spring-cloud-dependencies-parent/3.0.2/spring-cloud-dependencies-parent-3.0.2.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/springframework/cloud/spring-cloud-dependencies-parent/3.0.2/spring-cloud-dependencies-parent-3.0.2.pom (8.5 kB at 368 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/springframework/cloud/spring-cloud-commons-dependencies/3.0.2/spring-cloud-commons-dependencies-3.0.2.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/springframework/cloud/spring-cloud-commons-dependencies/3.0.2/spring-cloud-commons-dependencies-3.0.2.pom (7.1 kB at 324 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/springframework/cloud/spring-cloud-netflix-dependencies/3.0.2/spring-cloud-netflix-dependencies-3.0.2.pom
```

All the necessary images are built and can be verified as below.



```
ubuntu@ip-10-0-0-90:~/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications$ sudo docker images
REPOSITORY          TAG      IMAGE ID      CREATED             SIZE
sweet-home/notificationservice  1.0.0    78093dc8f4ff  About a minute ago  353MB
sweet-home/eurekaserver        1.0.0    7ea0b9c7a05b  2 minutes ago       385MB
sweet-home/paymentservice      1.0.0    edc6f62e10a6  2 minutes ago       491MB
sweet-home/bookingservice      1.0.0    753c32a5f557  4 minutes ago       413MB
maven                  3.8.2-jdk-11  e7518af4d281  3 days ago        670MB
openjdk                14-jdk-alpine  8273876b08aa  19 months ago      340MB
ubuntu@ip-10-0-0-90:~/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications$
```

Further, after building all the images of the micro services, the deployment is carried out by running the docker-compose file.

docker-compose file contains all the specifications related to containers and which image need to use to start/run the container. All networks and volumes can also be added into the file in-order to create based on the necessity.

The sequence of container creation can be specified via `depend_on`. Environment classified as a collection of variables that can be used while starting the application from the container. Port mapping is necessary in which one has to specify the port of the container and the mapping of that port with the host machine.

In our docker-compose file we have added `eurekaserver`, `paymentservice`, `bookingservice` and `notificationservice`.

The docker-compose file is built using the below content.

```
version: '3'
```

```

services:
  eurekaserver:
    build: eurekaserver
    container_name: eureka-server
    image: sweet-home/eurekaserver:1.0.0
    ports:
      - "8761:8761"
    networks:
      - sweet-home-network
    environment:
      EUREKA_HOST_NAME: 52.1.204.116
      EUREKA_PORT: 8761

  paymentservice:
    build: paymentservice
    container_name: payment-service
    image: sweet-home/paymentservice:1.0.0
    ports:
      - "8083:8083"
    networks:
      - sweet-home-network
    environment:
      EUREKA_HOST_NAME: eurekaserver
      EUREKA_PORT: 8761
      PAYMENT_SVC_PORT: 8083
    depends_on:
      - eurekaserver

  bookingservice:
    build: bookingservice
    container_name: booking-service
    image: sweet-home/bookingservice:1.0.0
    ports:
      - "8080:8080"
    networks:
      - sweet-home-network
    environment:
      EUREKA_HOST_NAME: eurekaserver
      EUREKA_PORT: 8761
      BOOKING_SVC_PORT: 8080
    depends_on:
      - eurekaserver
      - notificationservice

  notificationservice:
    build: notificationservice
    container_name: notification-service
    image: sweet-home/notificationservice:1.0.0
    networks:

```

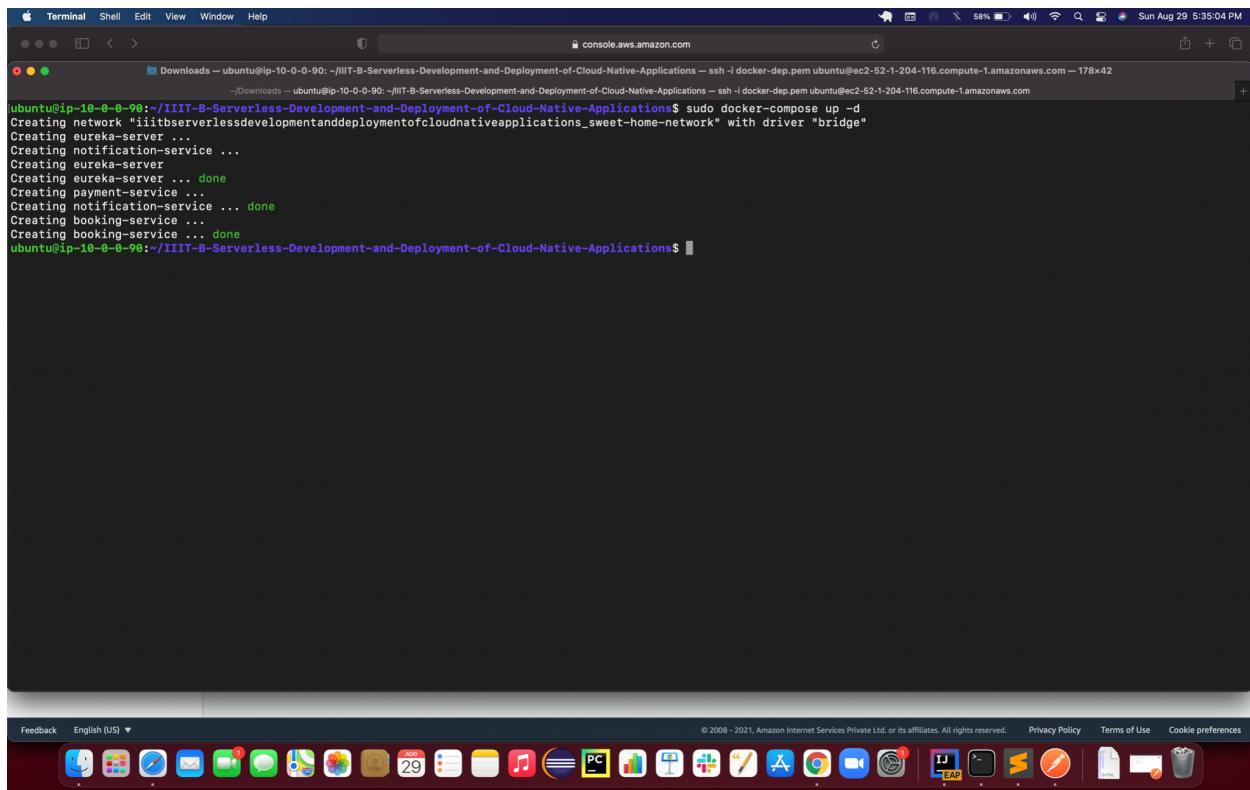
```
- sweet-home-network
```

```
networks:
```

```
sweet-home-network:  
  driver: bridge
```

To start the containers based on definition from the docker-compose can be done using the below command.

```
sudo docker-compose up -d
```



```
ubuntu@ip-10-0-0-90:~/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications$ sudo docker-compose up -d  
Creating network "iiitsserverlessdevelopmentanddeploymentofcloudnativeapplications_sweet-home-network" with driver "bridge"  
Creating nureka-server ...  
Creating notification-service ...  
Creating nureka-server  
Creating nureka-server ... done  
Creating payment-service ...  
Creating notification-service ... done  
Creating booking-service ...  
Creating booking-service ... done  
ubuntu@ip-10-0-0-90:~/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications$
```

Here, a bridge type of network named sweet-home-network is created as the docker-compose start building architecture.

All the services are up and running on specified ports and can be verified using below docker command.

```
sudo docker ps -a
```

```

ubuntu@ip-10-0-0-90:~/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications$ sudo docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
7f8162c952d4 sweet-home/bookingservice:1.0.0 "java -jar /opt/app_..." 3 minutes ago Up 3 minutes 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp booking-service
cebd279be033 sweet-home/paymentsservice:1.0.0 "java -jar /opt/app_..." 3 minutes ago Up 3 minutes 0.0.0.0:8083->8083/tcp, :::8083->8083/tcp payment-service
7bc45e3b7f15 sweet-home/eurekashervert:1.0.0 "java -jar /opt/app_..." 3 minutes ago Up 3 minutes 0.0.0.0:8761->8761/tcp, :::8761->8761/tcp eureka-server
a283ab56ce0 sweet-home/notificationsservice:1.0.0 "java -jar /opt/app_..." 3 minutes ago Up 3 minutes notification-service
ubuntu@ip-10-0-0-90:~/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications$ 

```

Eureka can be viewed from the browser by accessing with the **docker-ec2** public IP which is Elastic IP **docker-ec2-ip** and the eureka port **8761**

## System Status

Environment	N/A
Data center	N/A
Current time	2021-08-29T21:42:41 +0000
Uptime	00:07
Lease expiration enabled	false
Renews threshold	5
Renews (last min)	4

**EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.**

## DS Replicas

### Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
BOOKING-SERVICE	n/a (1)	(1)	UP (1) - 7f8162c952d4:booking-service:8080
PAYMENT-SERVICE	n/a (1)	(1)	UP (1) - cebd279be033:payment-service:8083

## General Info

Name	Value
total-avail-memory	95mb
num-of-cpus	2
current-memory-usage	39mb (41%)
server-upptime	00:07
registered-replicas	
unavailable-replicas	

## Verification

All the micro services are running on a specified port, all micro services are verified using the postman.

Booking service endpoints are defined as below and running on port 8080

POST /booking

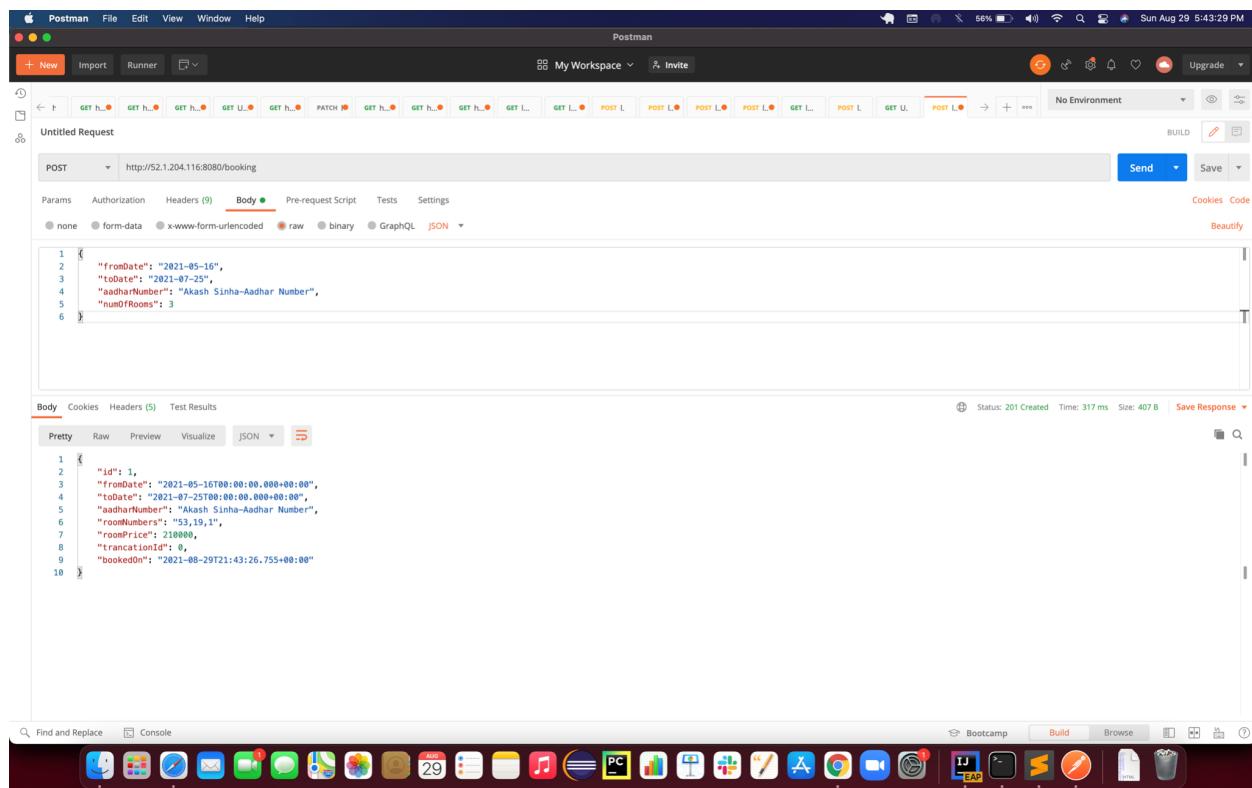
POST /booking/{id}/transaction

Payment service endpoints are defined as below and running on port 8083

POST /transaction

GET /transaction/{id}

All micro services test cases are attached as below and data validation is also performed and verified using the MySQLWorkbench in order to validate data consistency.



```
1 {
2   "id": 1,
3   "fromDate": "2021-05-16",
4   "toDate": "2021-07-25",
5   "aadharNumber": "Akash Sinha-Aadhar Number",
6   "num0fRooms": 3
7 }
```

Body Cookies Headers (5) Test Results

```
1 {
2   "id": 1,
3   "fromDate": "2021-05-16T00:00:00.000+00:00",
4   "toDate": "2021-07-25T00:00:00.000+00:00",
5   "aadharNumber": "Akash Sinha-Aadhar Number",
6   "roomNumbers": "53,19,1",
7   "roomPrice": 210000,
8   "transcationId": 0,
9   "bookedOn": "2021-08-29T21:43:26.755+00:00"
10 }
```

Postman

Untitled Request

POST http://52.1.204.116:8083/transaction

Body (JSON)

```
1 {
2   "paymentMode": "CARD",
3   "bookingId": 1,
4   "upId": "MyUPIID",
5   "cardNumber": "My card No"
6 }
```

Status: 201 Created Time: 359 ms Size: 170 B Save Response

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

Find and Replace Console

Bootcamp Build Browse

Postman

Untitled Request

POST http://52.1.204.116:8080/booking/1/transaction

Body (JSON)

```
1 {
2   "paymentMode": "CARD",
3   "bookingId": 1,
4   "upId": "MyUPIID",
5   "cardNumber": "Test Card 2"
6 }
```

Status: 201 Created Time: 425 ms Size: 407 B Save Response

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "fromDate": "2021-05-16T00:00:00+00:00",
4   "toDate": "2021-07-25T00:00:00+00:00",
5   "aadharNumber": "Akash.Sinha-Aadhar Number",
6   "roomNumbers": "53,19,1",
7   "roomPrice": 210000,
8   "trancationId": 2,
9   "bookedOn": "2021-08-29T21:43:27.000+00:00"
10 }
```

Find and Replace Console

Bootcamp Build Browse

The screenshot shows the Postman application interface. At the top, the menu bar includes Apple, Postman, File, Edit, View, Window, Help, and various status icons. The main toolbar has buttons for + New, Import, Runner, and a search bar. The workspace title is "My Workspace". A banner at the top right says "Sun Aug 29 5:45:20 PM". Below the toolbar, there's a list of recent requests and a "No Environment" message. The current request is titled "Untitled Request" and is a GET request to <http://52.1.204.116:8083/transaction/2>. The "Params" tab is selected. The response body is displayed in JSON format:

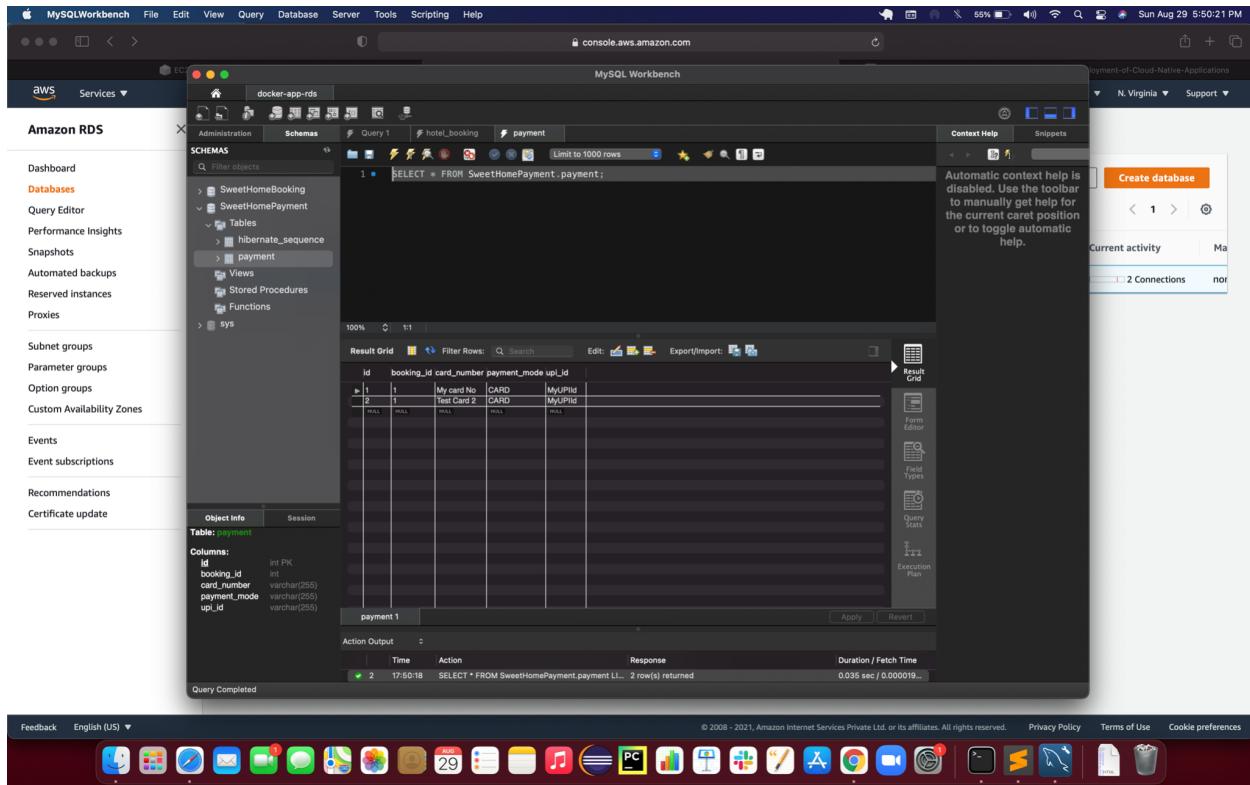
```
1 {
2     "id": 2,
3     "paymentMode": "CARD",
4     "bookingId": 1,
5     "upId": "MyUpId",
6     "cardNumber": "Test Card 2"
7 }
```

The status bar at the bottom indicates Status: 200 OK, Time: 77 ms, Size: 252 B, and Save Response.

The screenshot shows the MySQL Workbench application. The top navigation bar includes Apple, MySQLWorkbench, File, Edit, View, Query, Database, Server, Tools, Scripting, Help, and a connection to "console.aws.amazon.com". The left sidebar shows the AWS Services menu with "Amazon RDS" selected, displaying options like Dashboard, Databases, Query Editor, Performance Insights, Snapshots, Automated backups, Reserved instances, Proxies, Subnet groups, Parameter groups, Option groups, Custom Availability Zones, Events, Event subscriptions, and Recommendations. The main area is titled "MySQL Workbench" and shows a database connection to "docker-app-rds". It displays the "Schemas" tree with "SweetHomeBooking" schema selected, containing "Tables", "Views", "Stored Procedures", and "Functions". A query editor window is open with the SQL command: "SELECT \* FROM SweetHomeBooking.hotel\_booking;". The results grid shows one row of data:

id	aadhar_number	booked_on	from_date	room_numbers	room_price	to_date	tranction_id
1	Aakash Sinha-Aadhar Number	2021-08-29 21:43:27	2021-05-16 00:00:00	53,19,1	210000	2021-07-25 00:00:00	2

The status bar at the bottom indicates Feedback, English (US), © 2008 - 2021, Amazon Internet Services Private Ltd, or its affiliates. All rights reserved., Privacy Policy, Terms of Use, and Cookie preferences.



Notification service is connected with kafka and booking service, so for every successful booking service notification is being sent and printed on the console once the booking service writes the log into the kafka topic and the notification service asynchronously connects and reads the data from the kafka topic. This can only be done after a transaction is confirmed.

To preview notification service logs, one has to preview the logs of the notification service container.

Docker has a built-in command to analyse the container logs. The command to preview the logs from the notification container is as shown below.

```
sudo docker logs notification-service
```

```
Not Secure - 52.1.204.116
ubuntu@ip-10-0-0-90:~/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications$ sudo docker logs notification-service
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
message
Booking confirmed for user with aadhar number: Akash Sinha-Aadhar Number | Here are the booking details: BookingInfoEntity{fromDate=2021-05-16 00:00:00.0, toDate=2021-07-25 00:00:00.0, aadharNumber='Akash Sinha-Aadhar Number', roomNumbers='53,19,1', roomPrice=21000.0, transactionId='2', bookedOn=2021-08-29 21:43:27.0}
ubuntu@ip-10-0-0-90:~/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications$
```

## Instructions

- The RDS database URL needs to be mapped in the booking and payment service properties and username and password too (if changes).
- EC2 Elastic IP of **kafka-ec2** instance needs to be mapped in the notification service and booking service.
- Eureka ie. **docker-ec2** Elastic IP needs to be mapped in the docker-compose file as **EUREKA\_HOST\_NAME** within Environment section of **eurekaserver**

## Note

- Kafka is externalized using the two EC2 instances approach.