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.

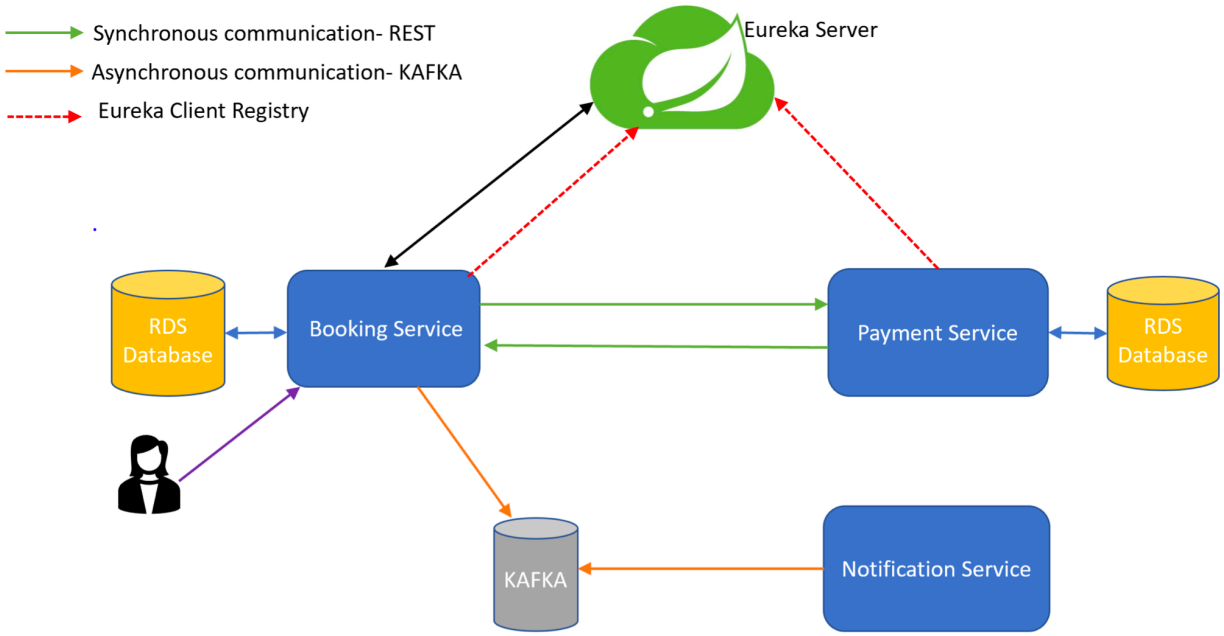
Notification service

> 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)

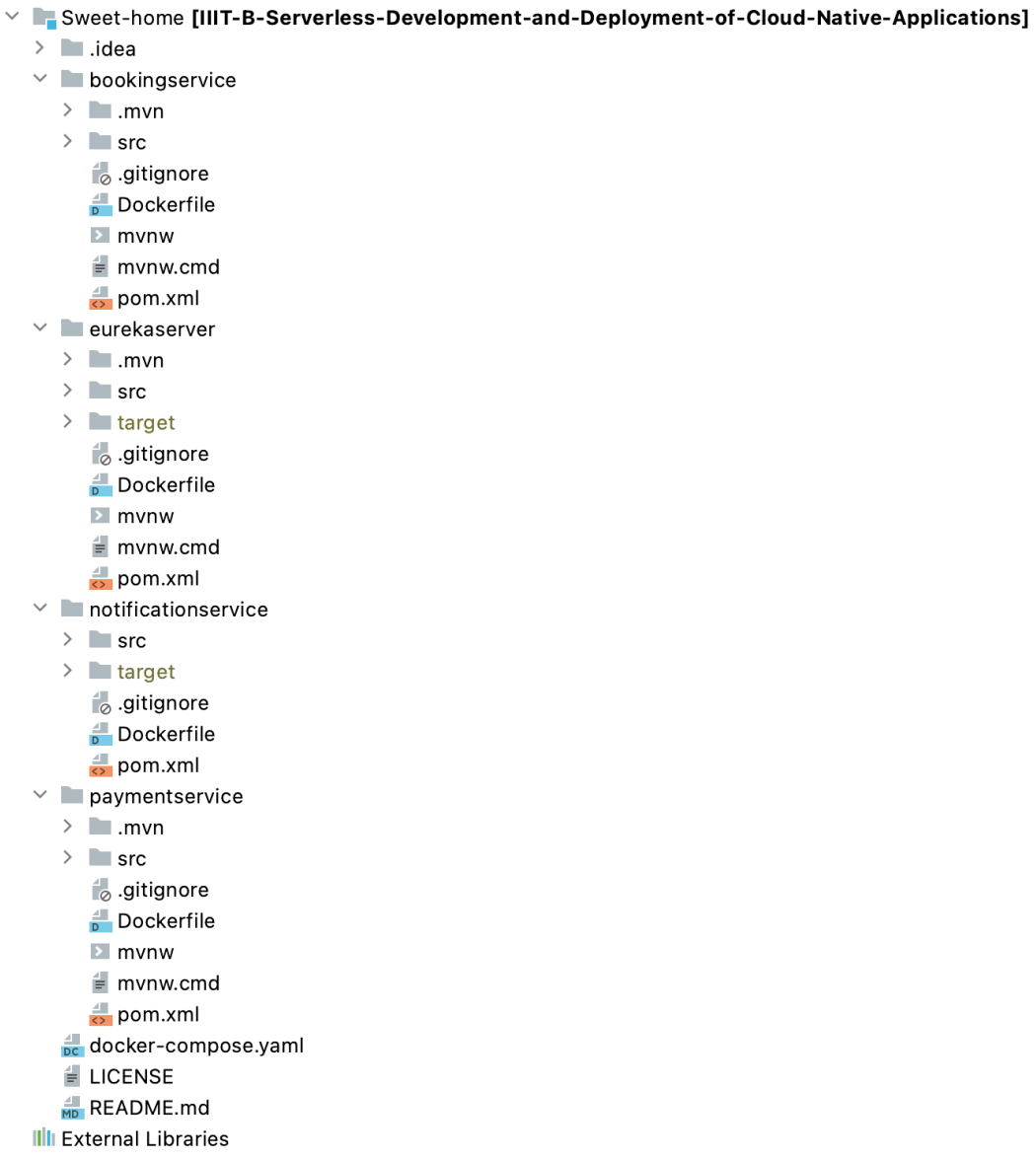
Solution Architecture:

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.

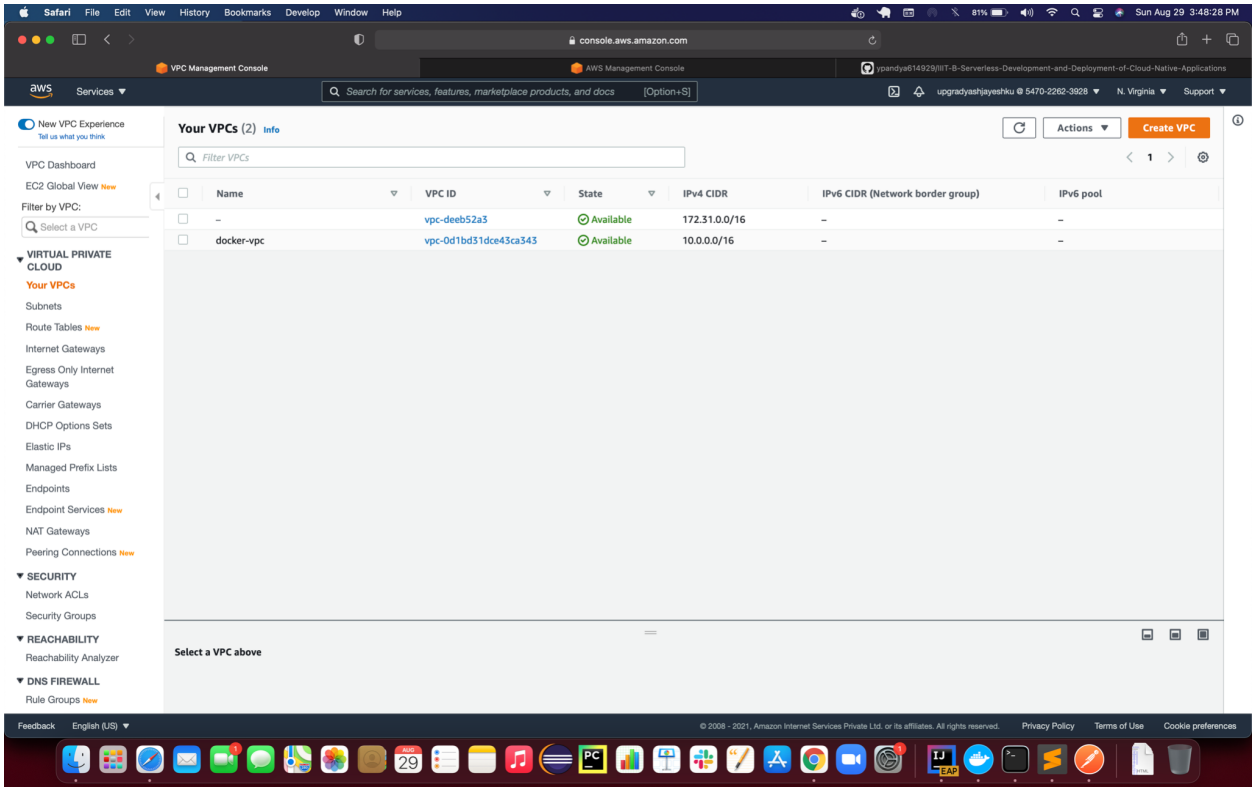
Deployment cycle contains the following resources.

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

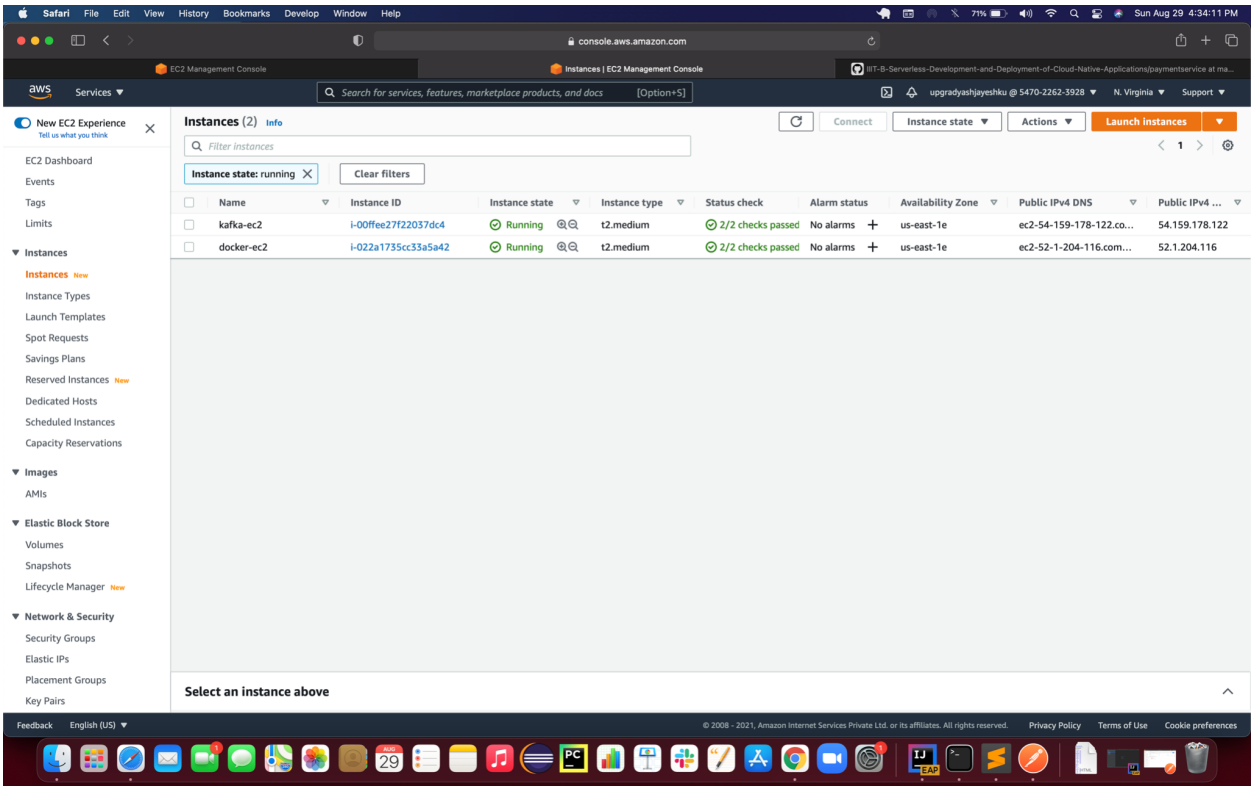
Solution approach is to **use a separate EC2 instance to install the apache kafka** and **do not containerize the kafka using the docker-compose**. Ie. **externalized kafka**. The connection to kafka is done using the public IP of the EC2 instance which is associated with the Elastic IP.



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



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



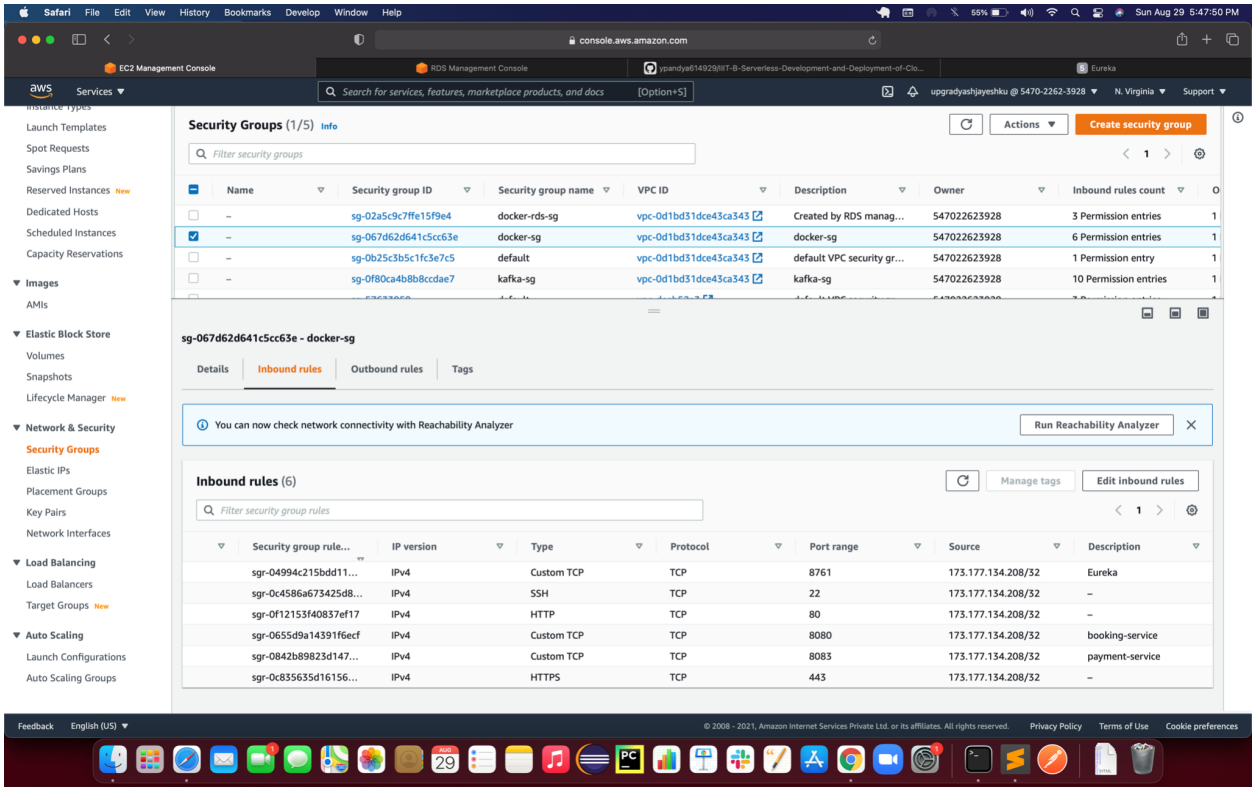
- **docker-ec2** instance is to deploy the application code using docker.
- **Kafka-ec2** instance is to install apache kafka for asynchronous communication for the booking and notification service.

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.

<u>Ports</u>

- **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.

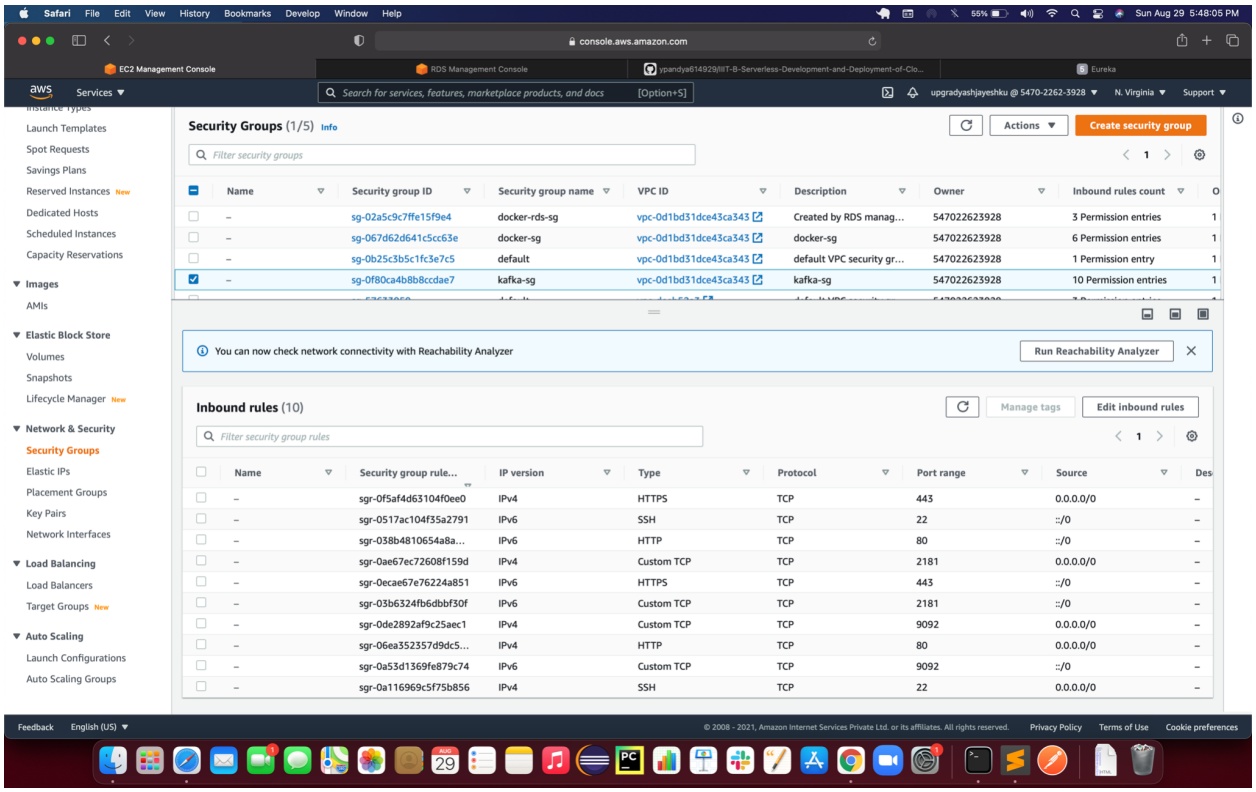


Another security group **kafka-sg** is created and below inbound rules are added under this security group. This security group is associated with the **kafka-ec2** where the apache kafka is being deployed.

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.

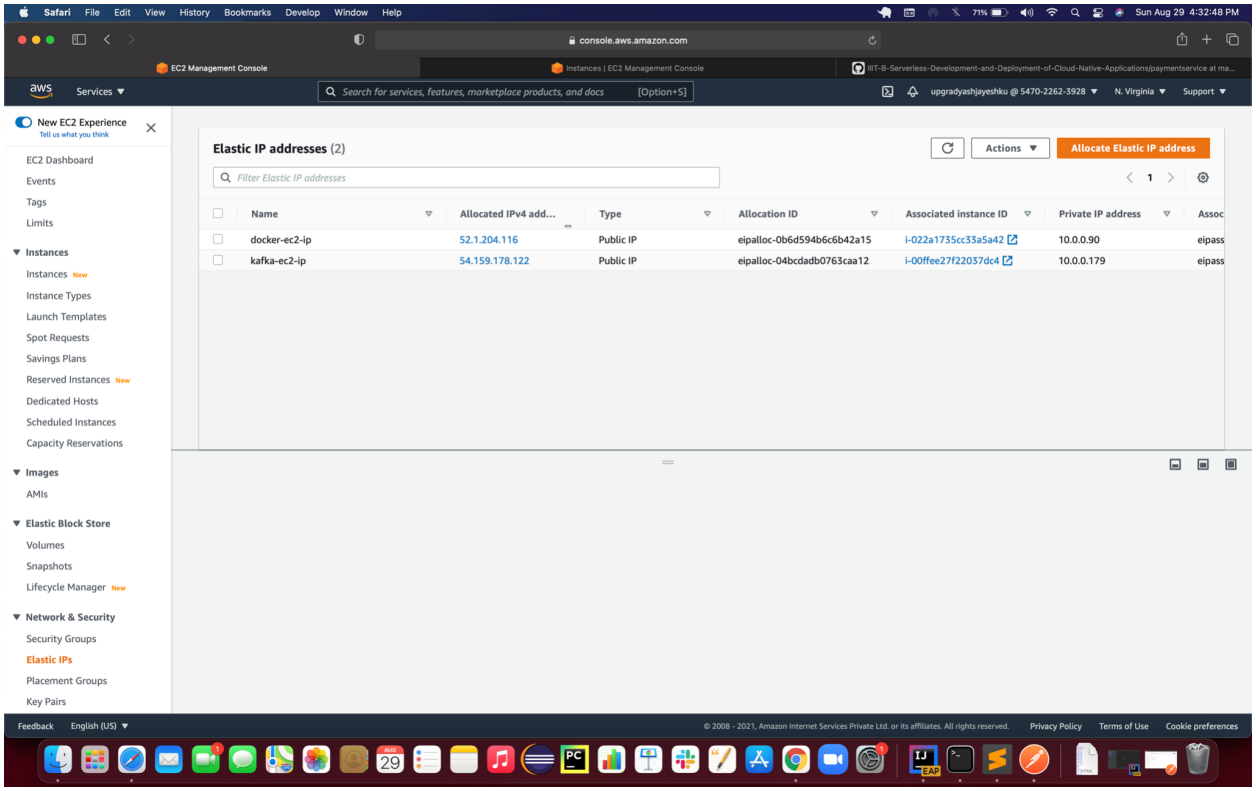


EC2 instance connection

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.



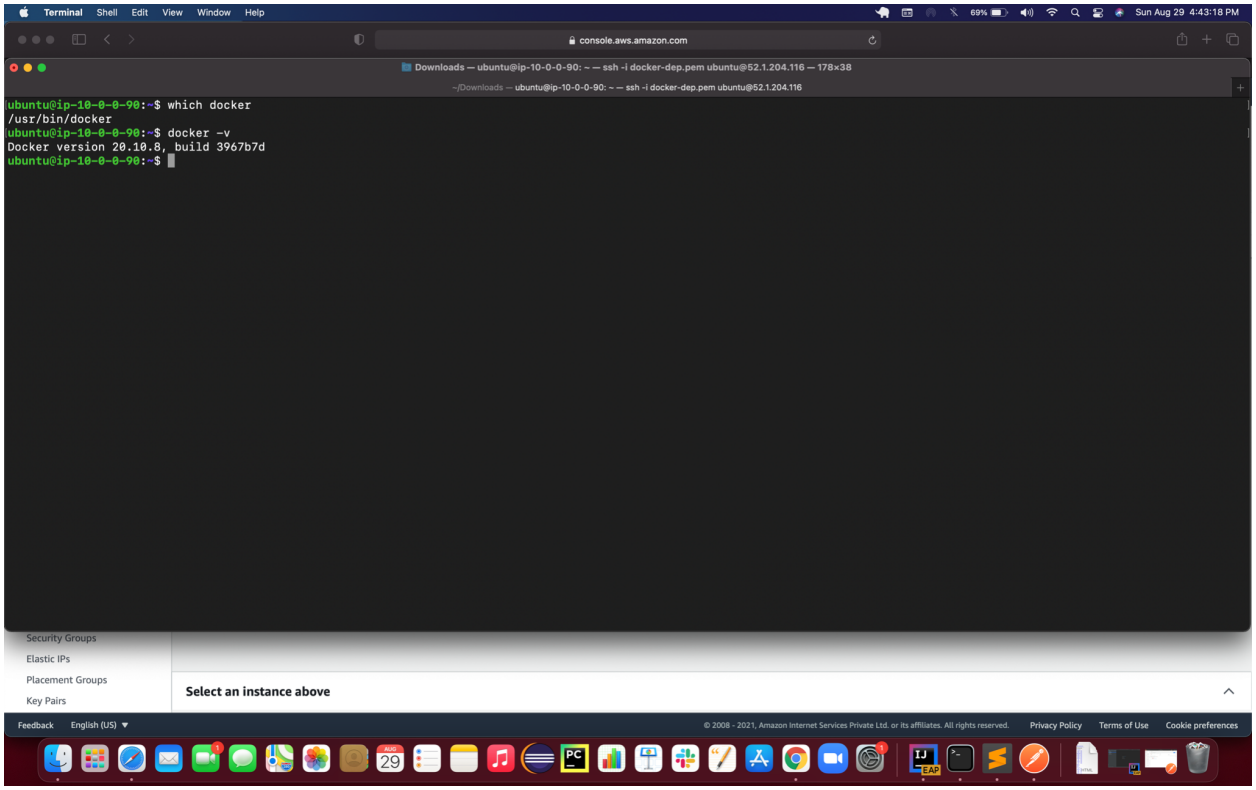
Docker and Docker-Compose Installation

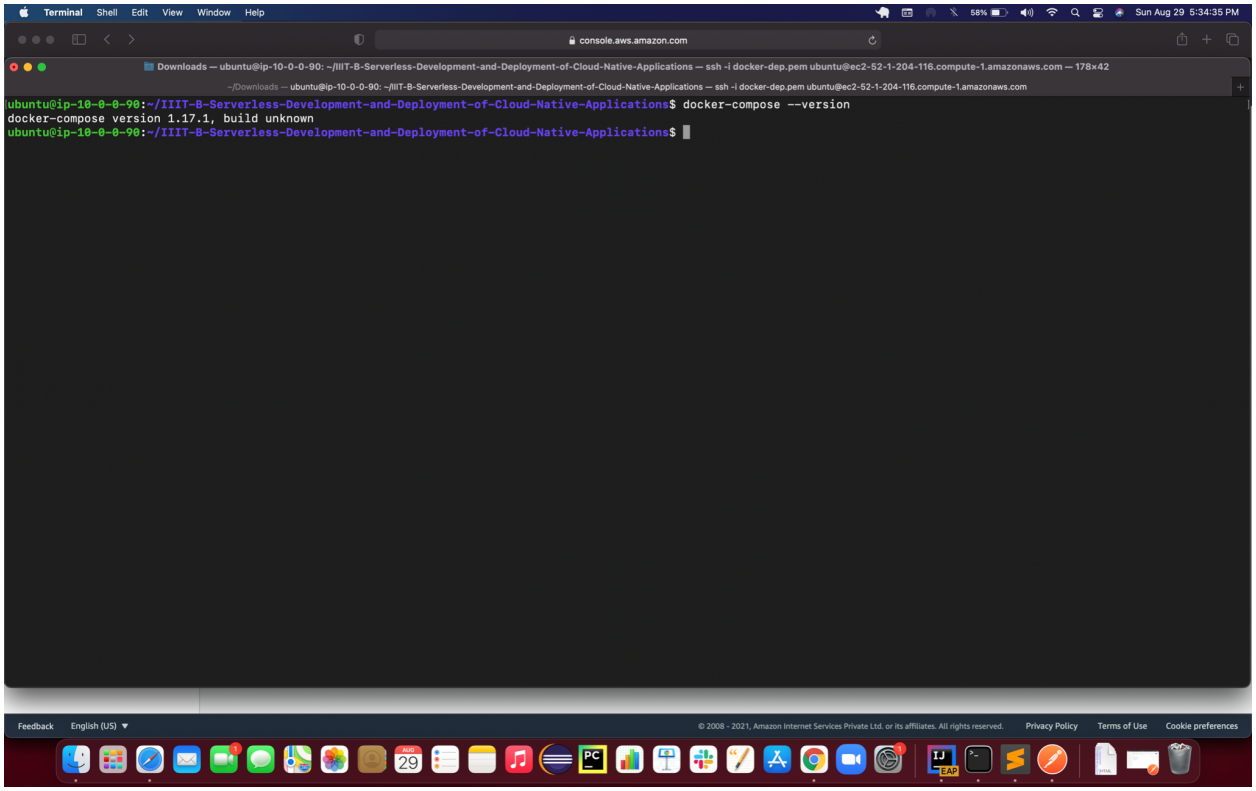
docker is installed on the **docker-ec2** instance using the docker installation process and verifies the installation using the below command.

```
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.

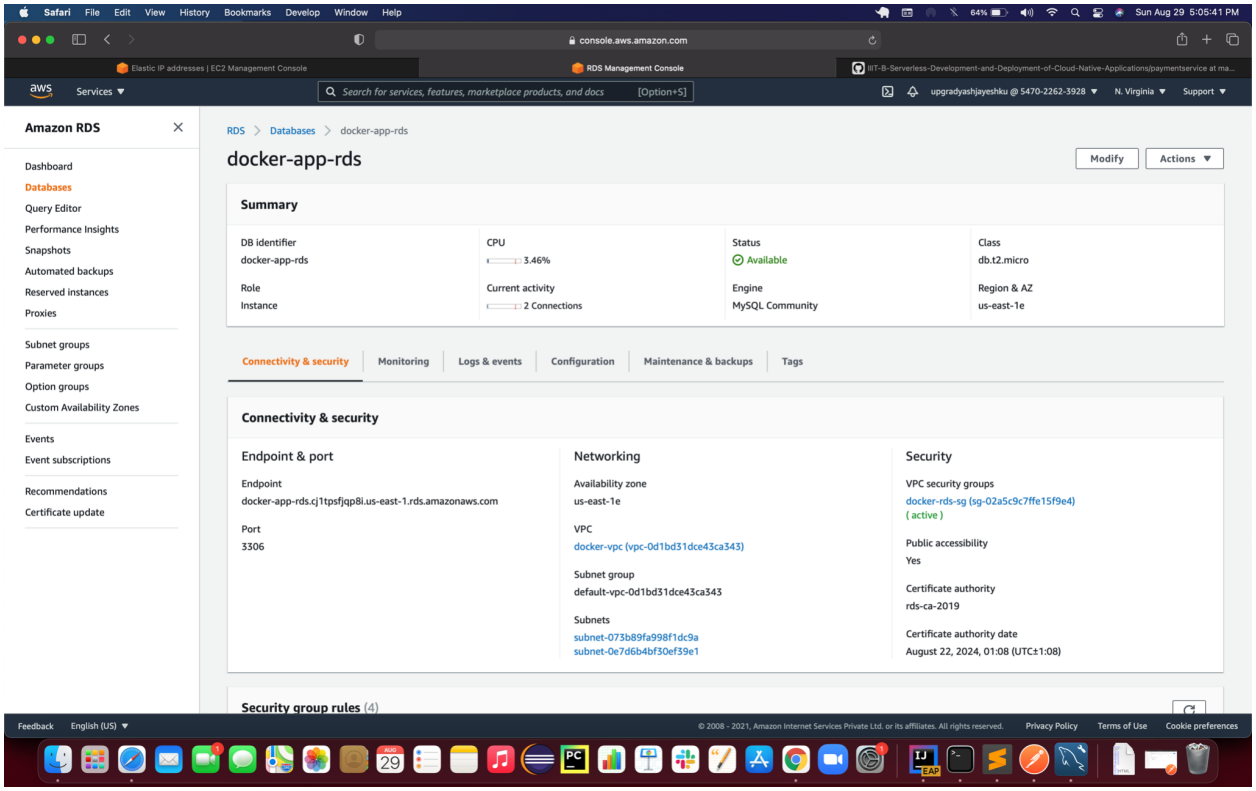
```
docker-compose --version
```


Terminal Shell Edit View Window Help
Sun Aug 29 4:43:18 PM
console.aws.amazon.com
Downloads — ubuntu@ip-10-0-0-90: ~ — ssh -i docker-dep.pem ubuntu@52.1.204.116 — 178×38
~/Downloads — ubuntu@ip-10-0-0-90: ~ — ssh -i docker-dep.pem ubuntu@52.1.204.116
ubuntu@ip-10-0-0-90:~$ which docker
/usr/bin/docker
ubuntu@ip-10-0-0-90:~$ docker -v
Docker version 20.10.8, build 3967b7d
ubuntu@ip-10-0-0-90:~$
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Select an instance above
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
Sun Aug 29 5:34:35 PM
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 — 178×42
~/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$ docker-compose --version
docker-compose version 1.17.1, build unknown
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

RDS

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



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.

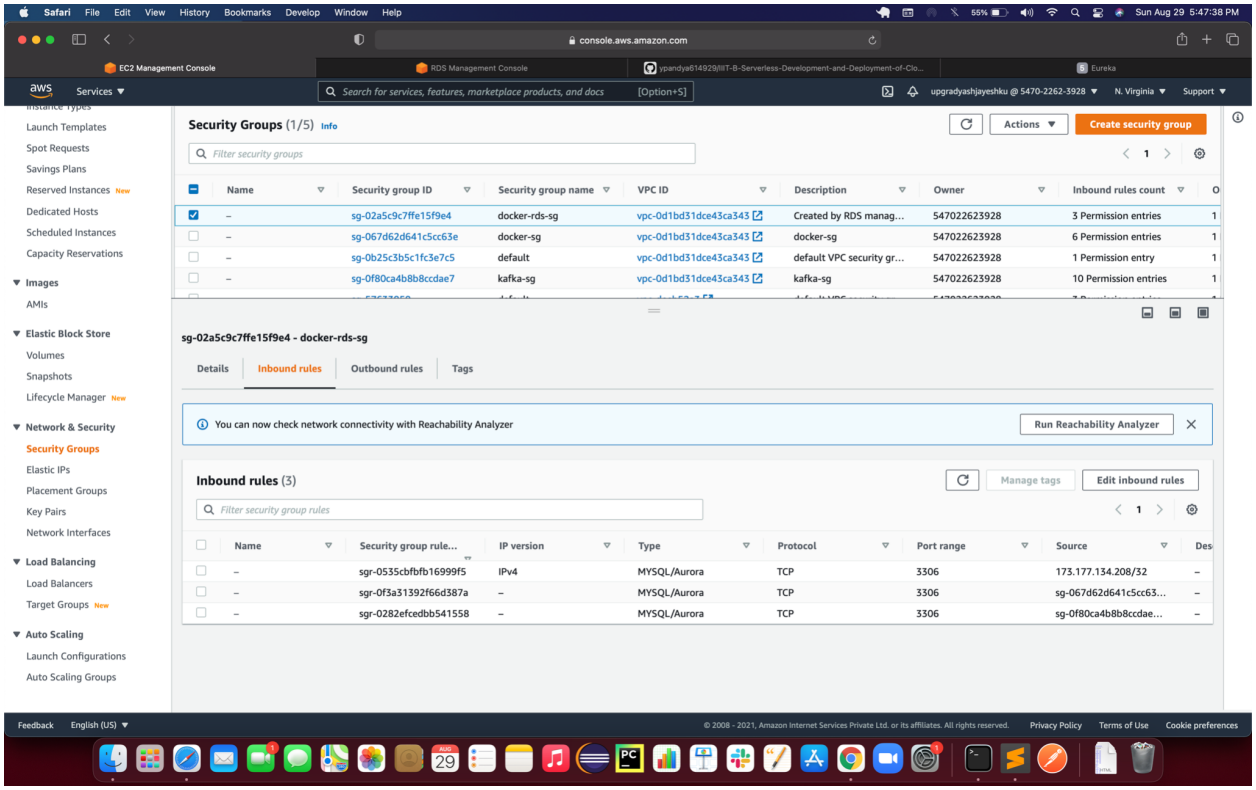
The two databases are created using the MySQLWorkbench in the RDS database instance as shown below.

Booking service uses the below database.

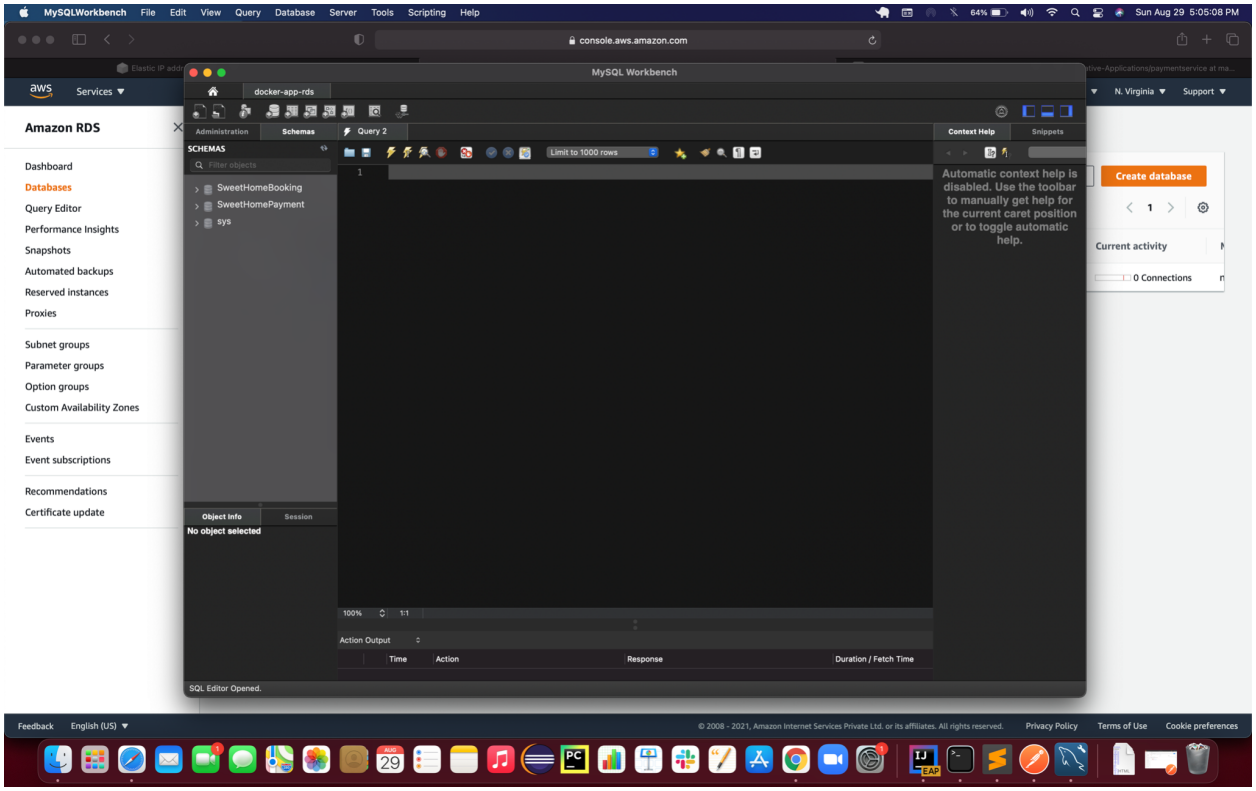
- SweetHomeBooking

Payment service uses the below database.

- SweetHomePayment



RDS is created with username as admin and password as upgrad123

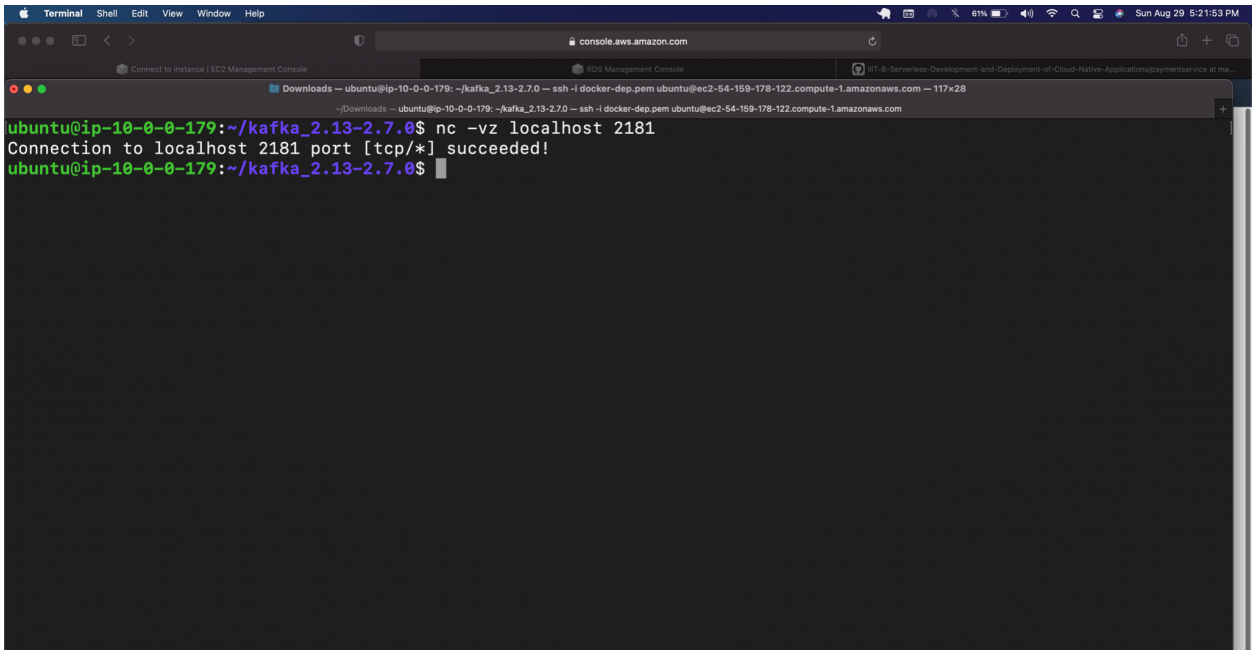


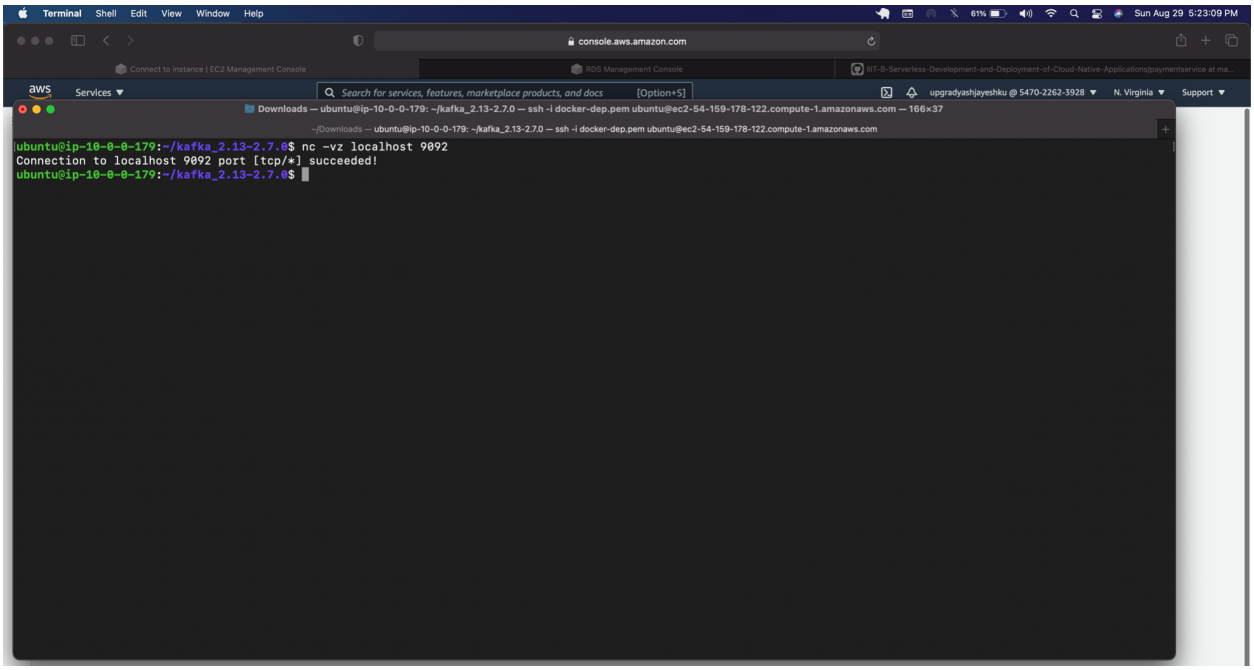
Kafka Installation

`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.



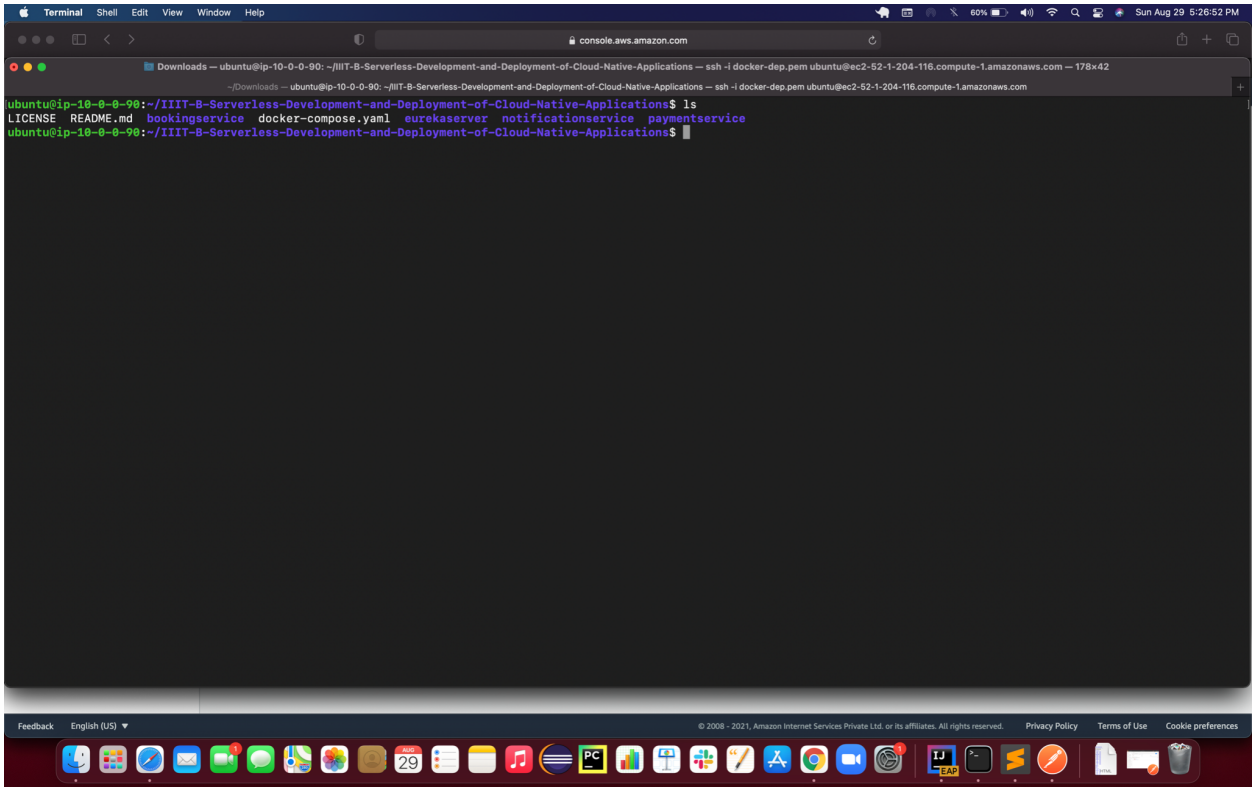


Application Deployment and Run instructions

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/ypandya614929/IIIT-B-Serverless-Development-and-Deployment-of-Cloud-Native-Applications.git`

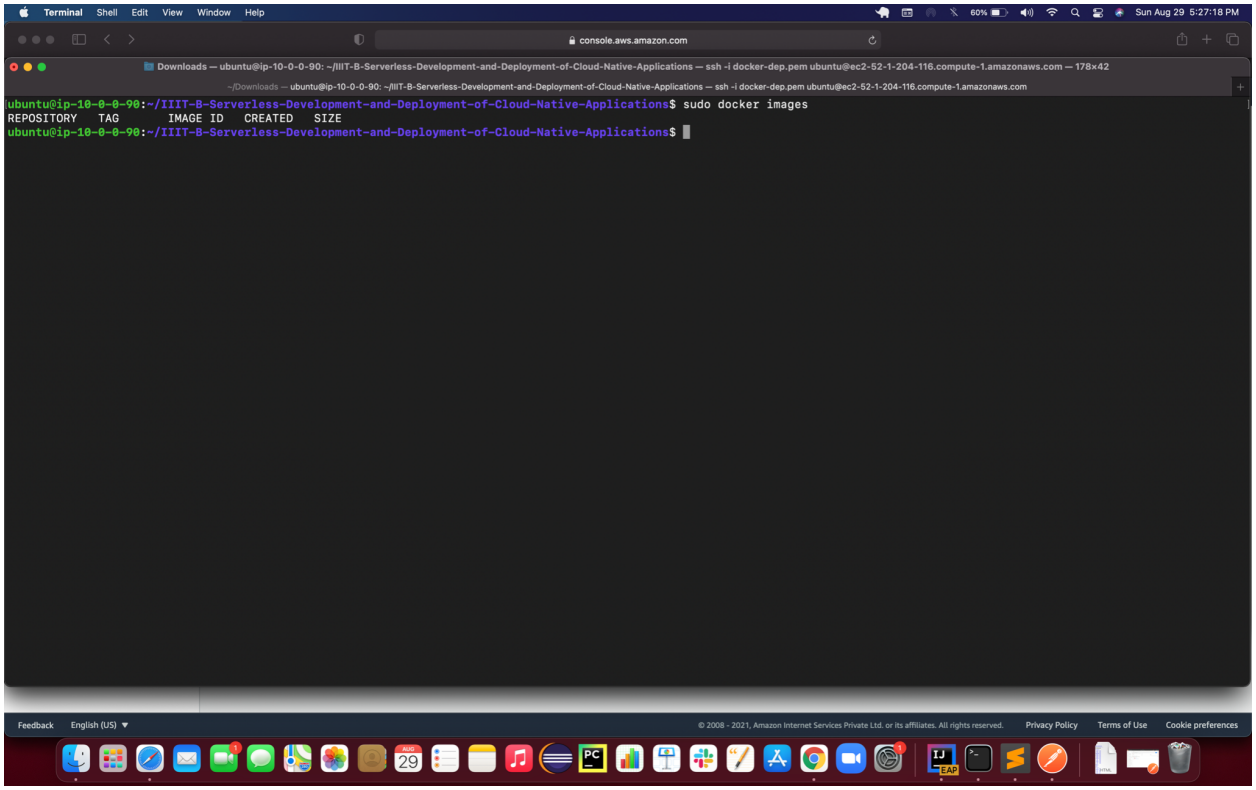
Please note that this is the private repository created under the developer/student's github account in order to simplify the process.

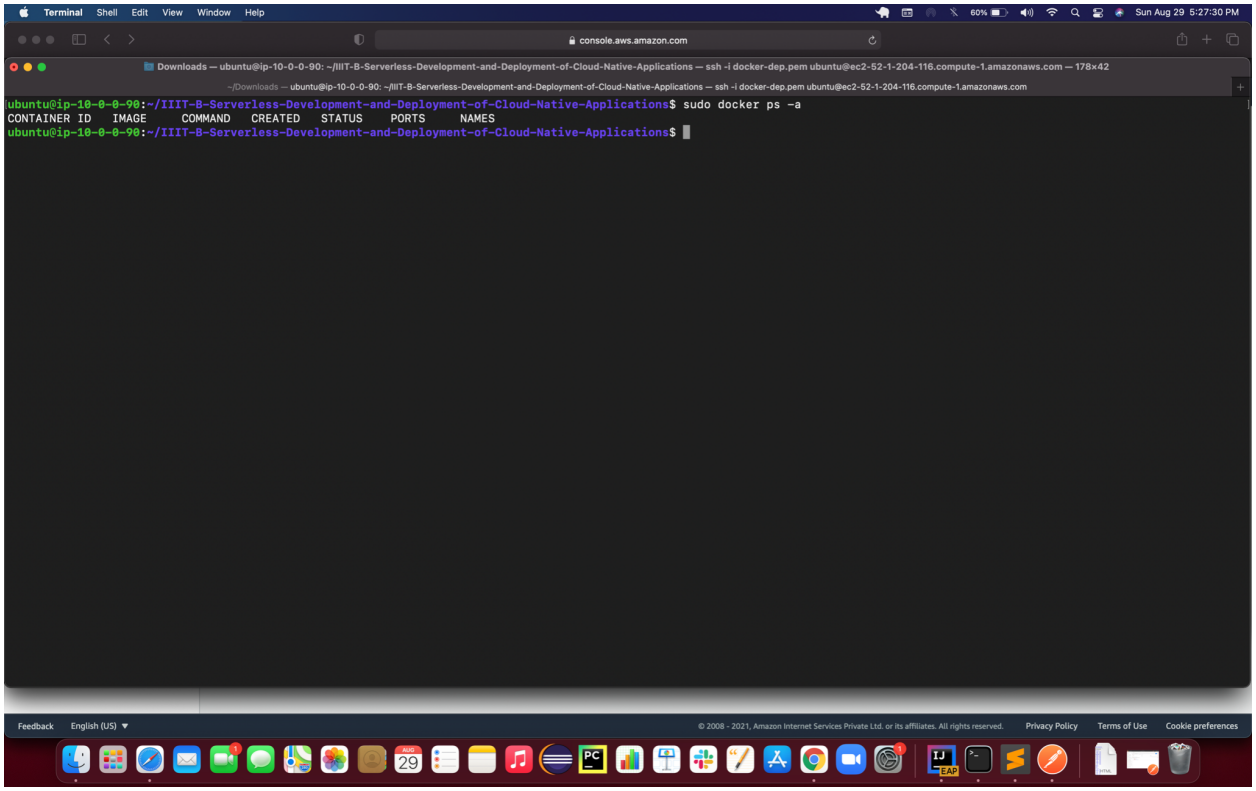


There are no docker images present initially and there are no containers running on the docker-ec2 instance. Both checks are verified using below commands.

`sudo docker images`

`sudo docker ps -a`


Terminal Shell Edit View Window Help
Sun Aug 29 5:27:18 PM
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 — 178×42
~/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
Sun Aug 29 5:27:30 PM
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 — 178×42
~/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

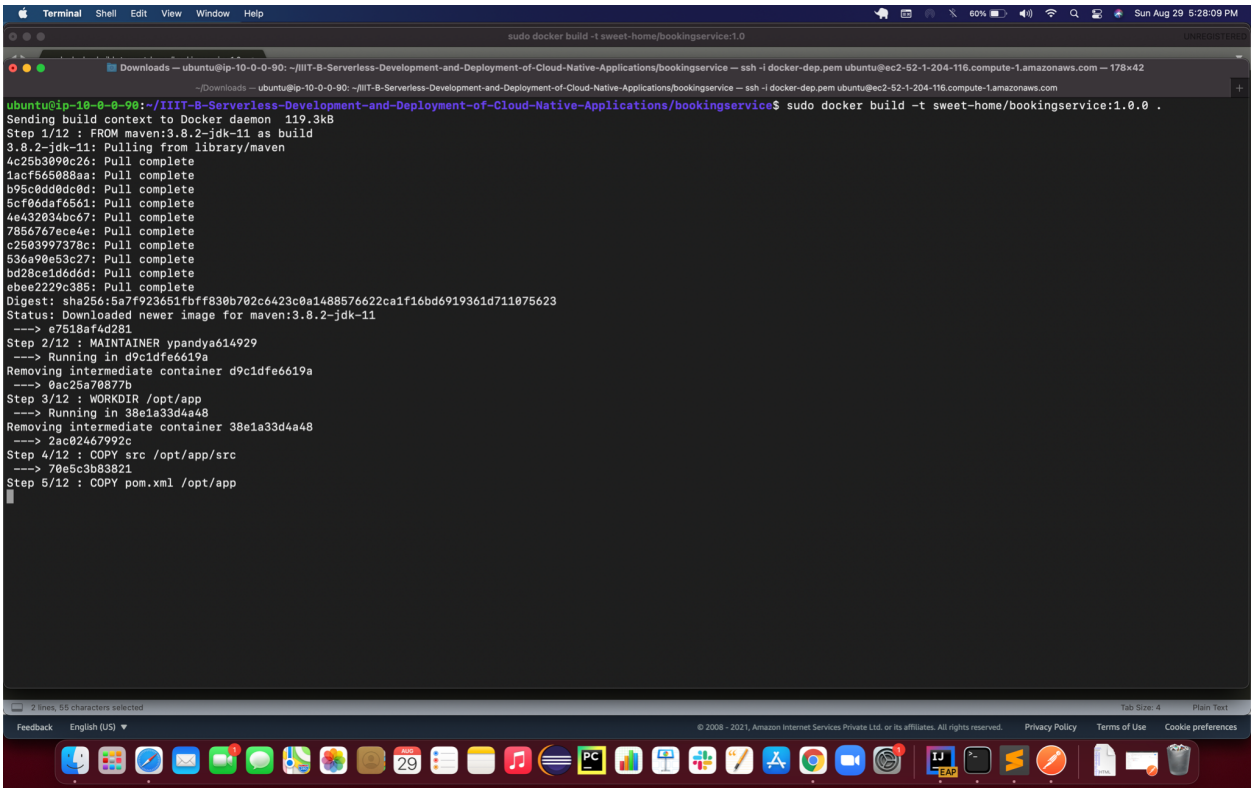
BookingService Image

The below commands are present in the Booking service Dockerfile.

```
# 1
FROM maven:3.8.2-jdk-11 as build
MAINTAINER ypandya614929
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 ypandya614929
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 .



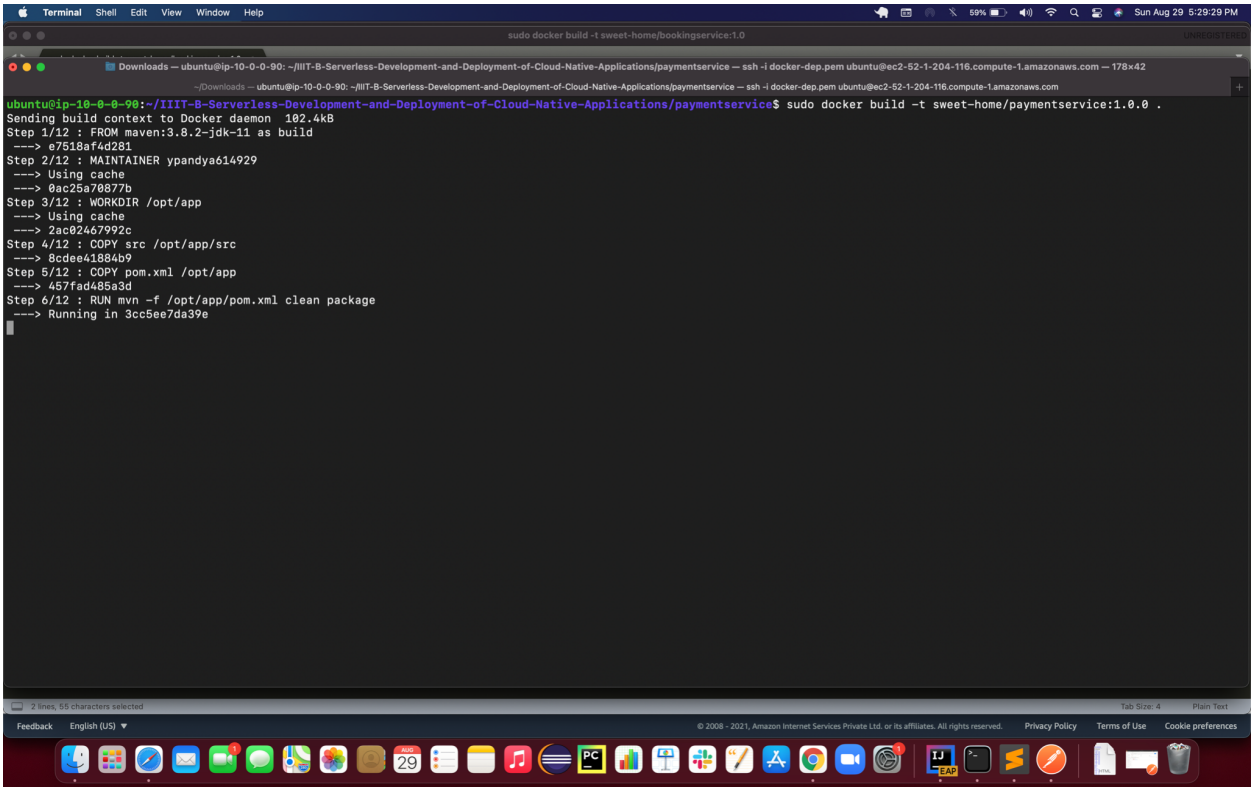
PaymentService Image

The below commands are present in the Payment service Dockerfile.

```
# 1
FROM maven:3.8.2-jdk-11 as build
MAINTAINER ypandya614929
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 ypandya614929
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 .



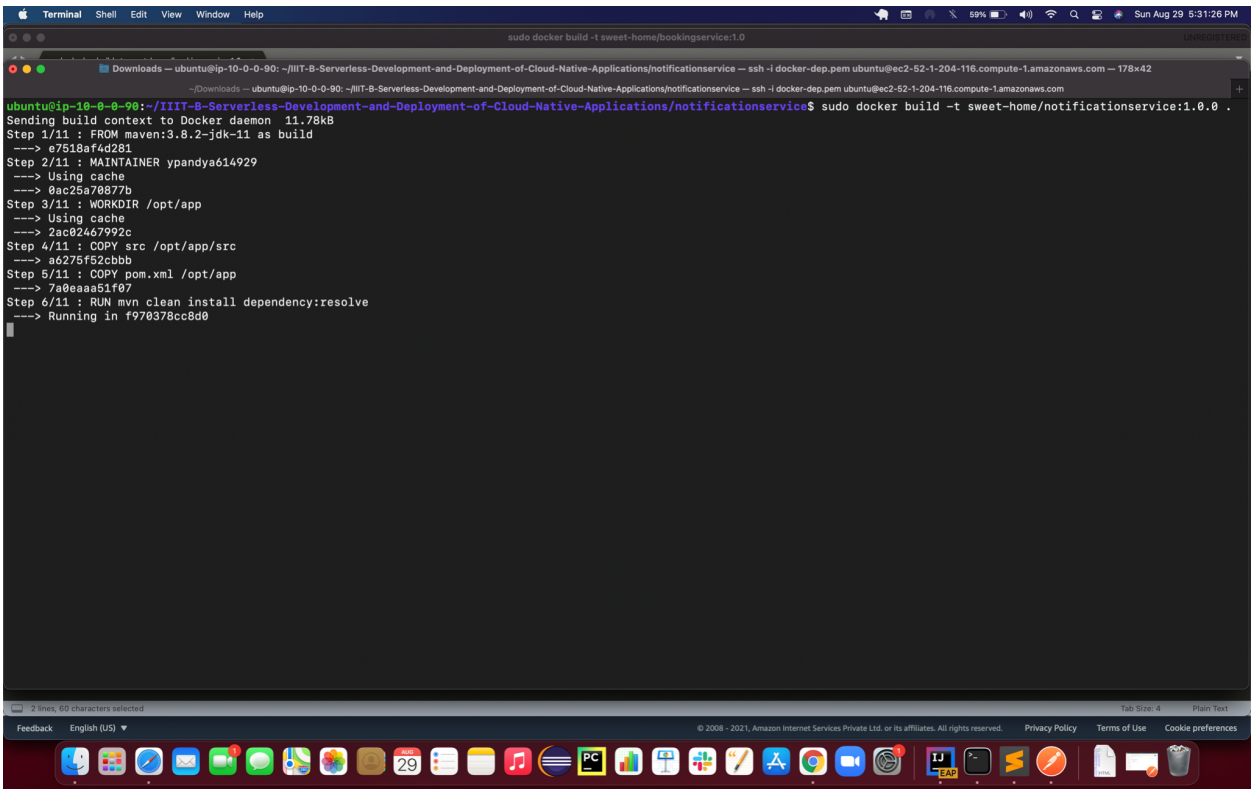
NotificationService Image

The below commands are present in the Notification service Dockerfile.

```
# 1
FROM maven:3.8.2-jdk-11 as build
MAINTAINER ypandya614929
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 ypandya614929
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 .



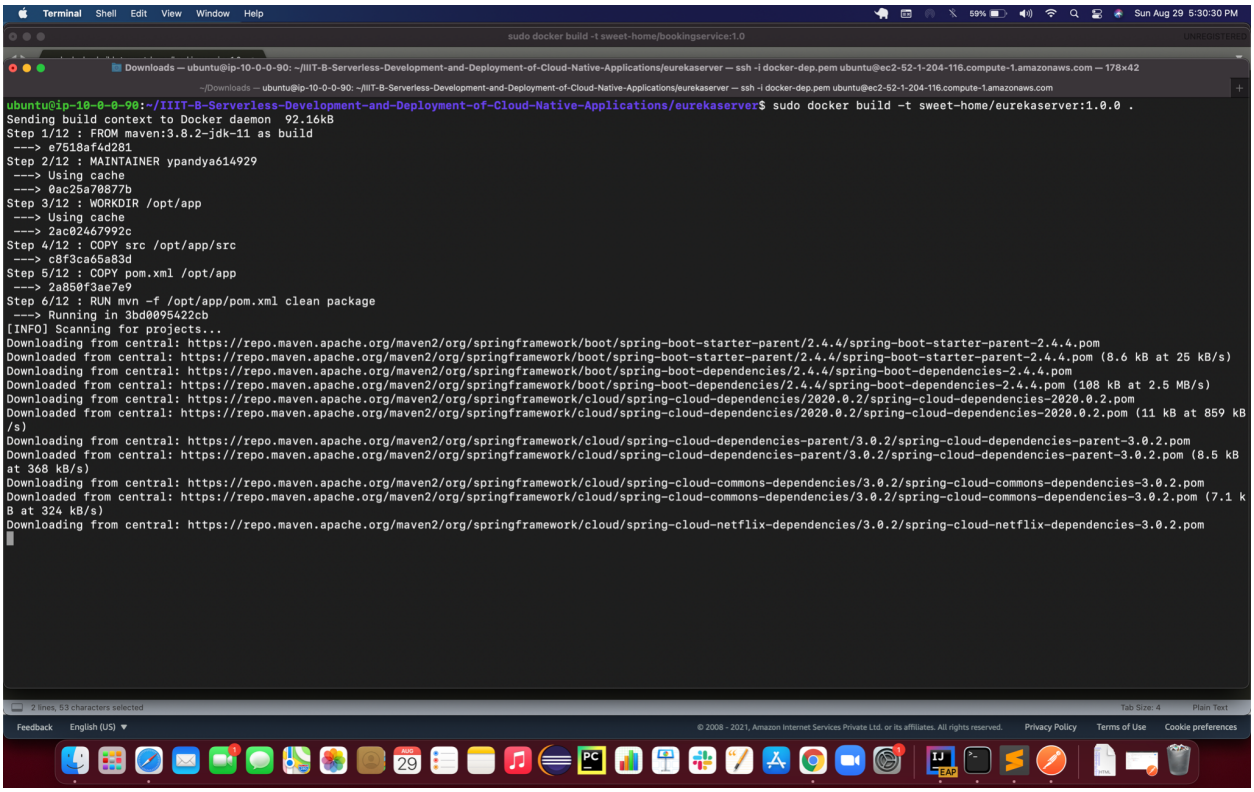
EurekaServer Image

The below commands are present in the Eureka server Dockerfile.

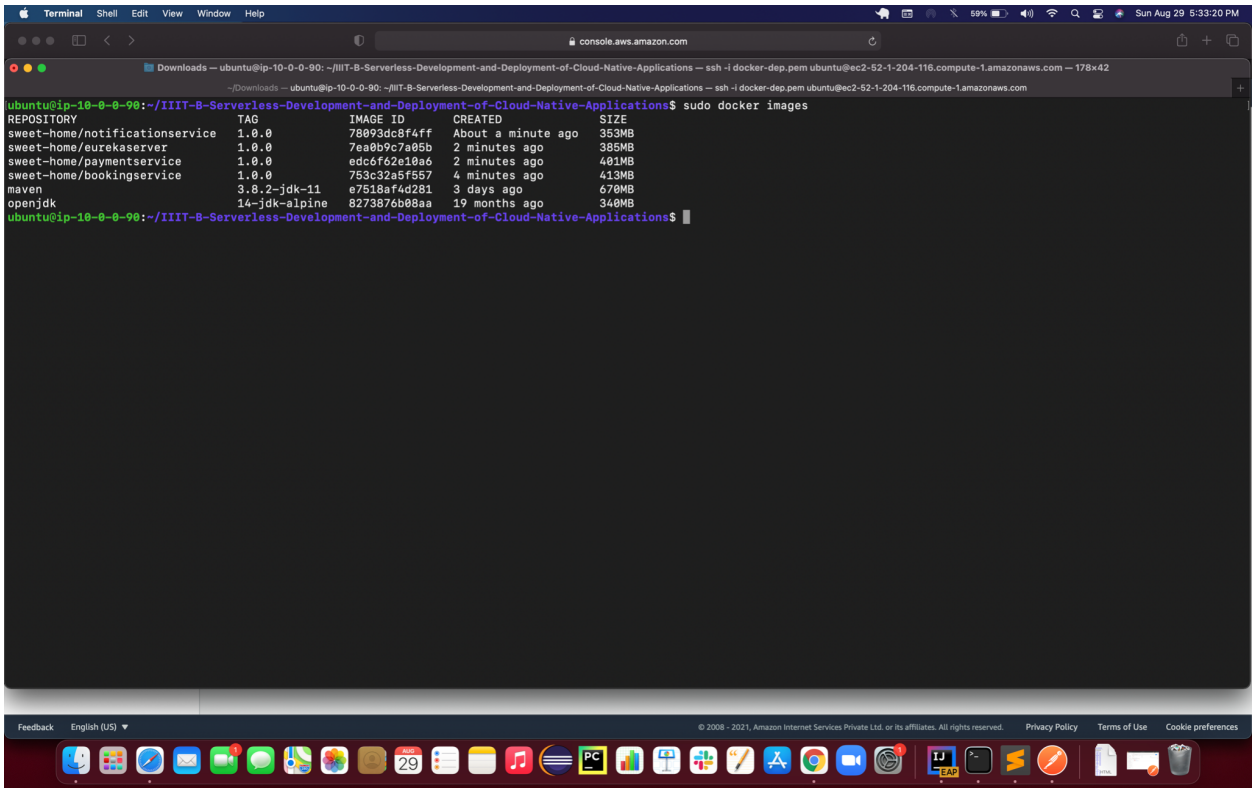
```
# 1
FROM maven:3.8.2-jdk-11 as build
MAINTAINER ypandya614929
WORKDIR /opt/app
COPY src /opt/app/src
COPY pom.xml /opt/app
RUN mvn -f /opt/app/pom.xml clean package

# 2
FROM openjdk:14-jdk-alpine
MAINTAINER ypandya614929
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 .



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



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.

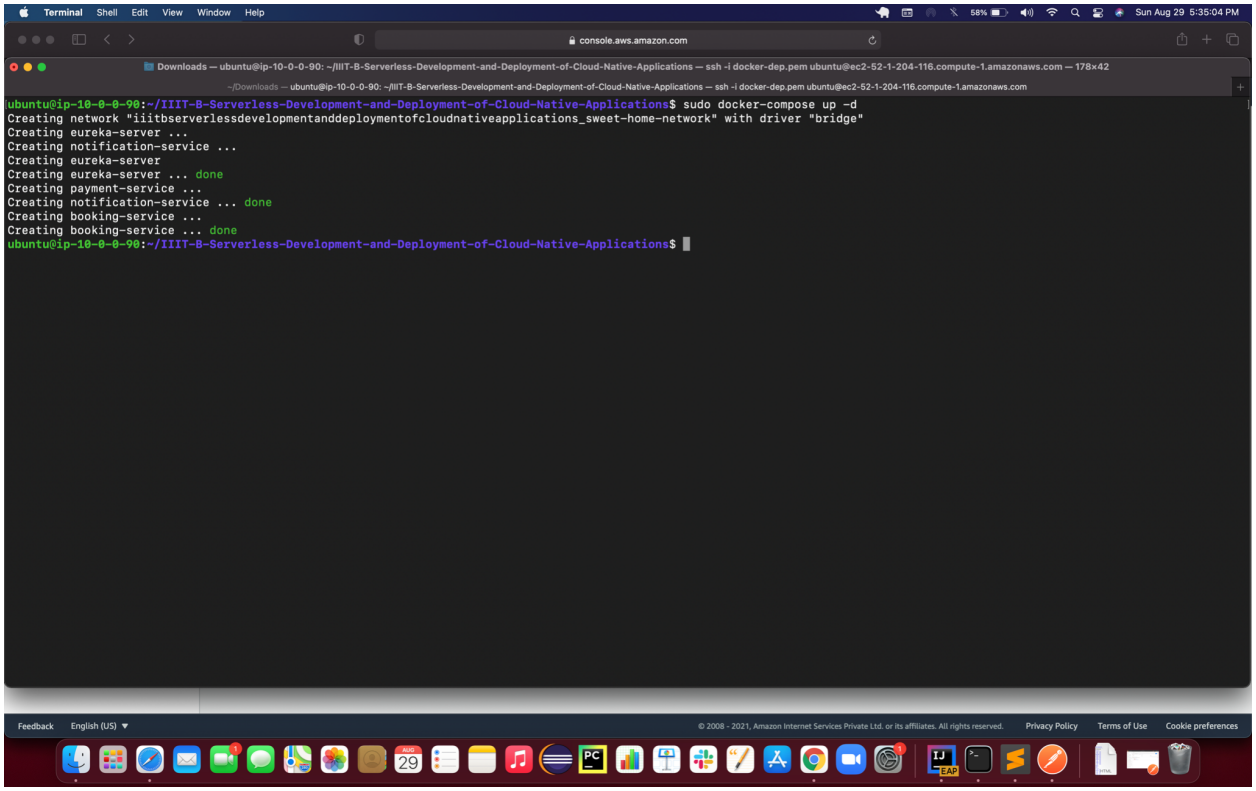
```
version: '3'
```

```yaml
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.

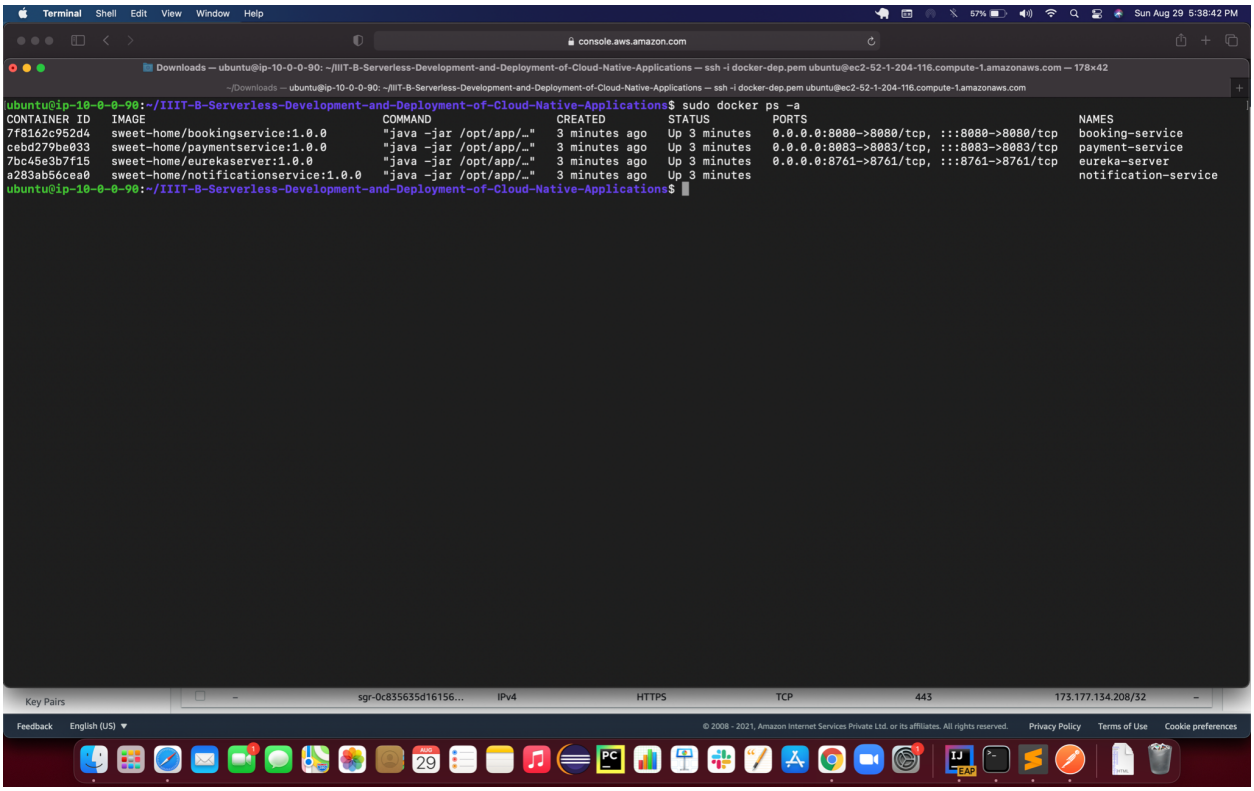
sudo docker-compose up -d



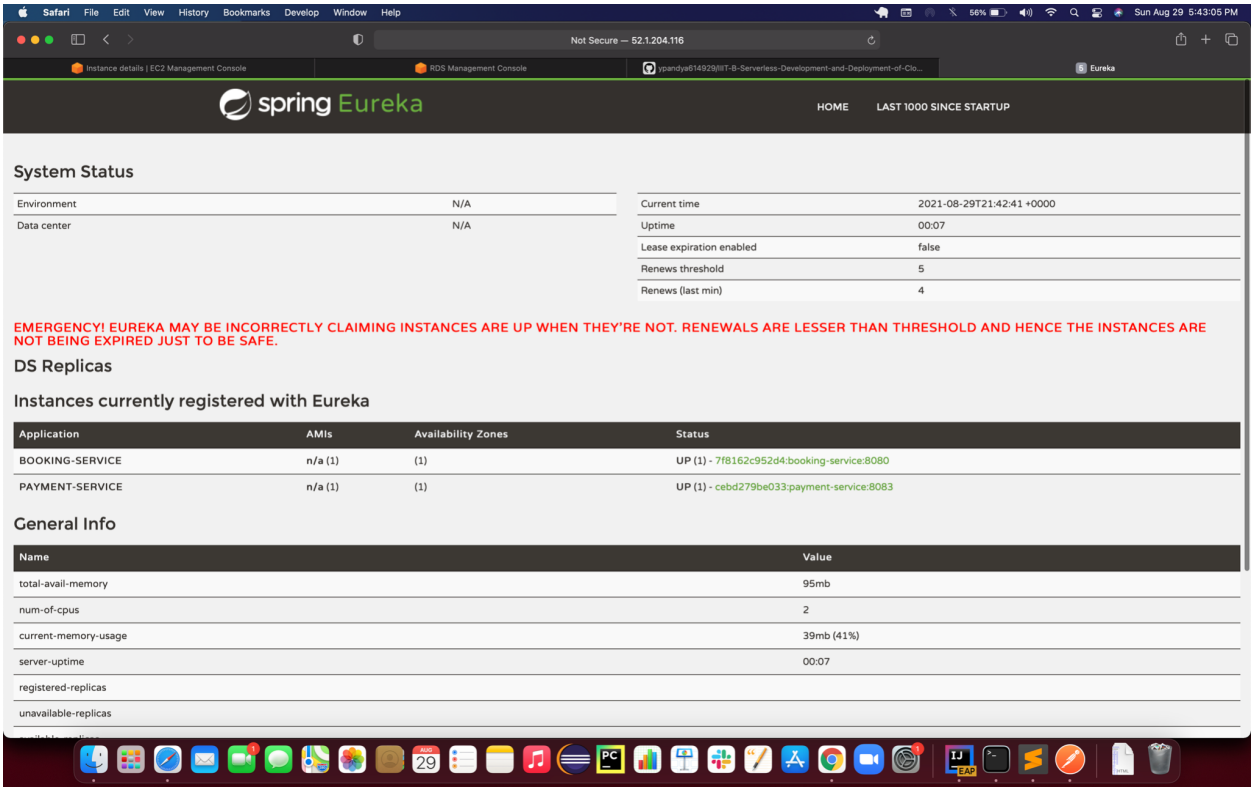
Here, a bridge type of network named sweet-home-network is created as the docker-compose start building architecture.

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

sudo docker ps -a



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**



Verification

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

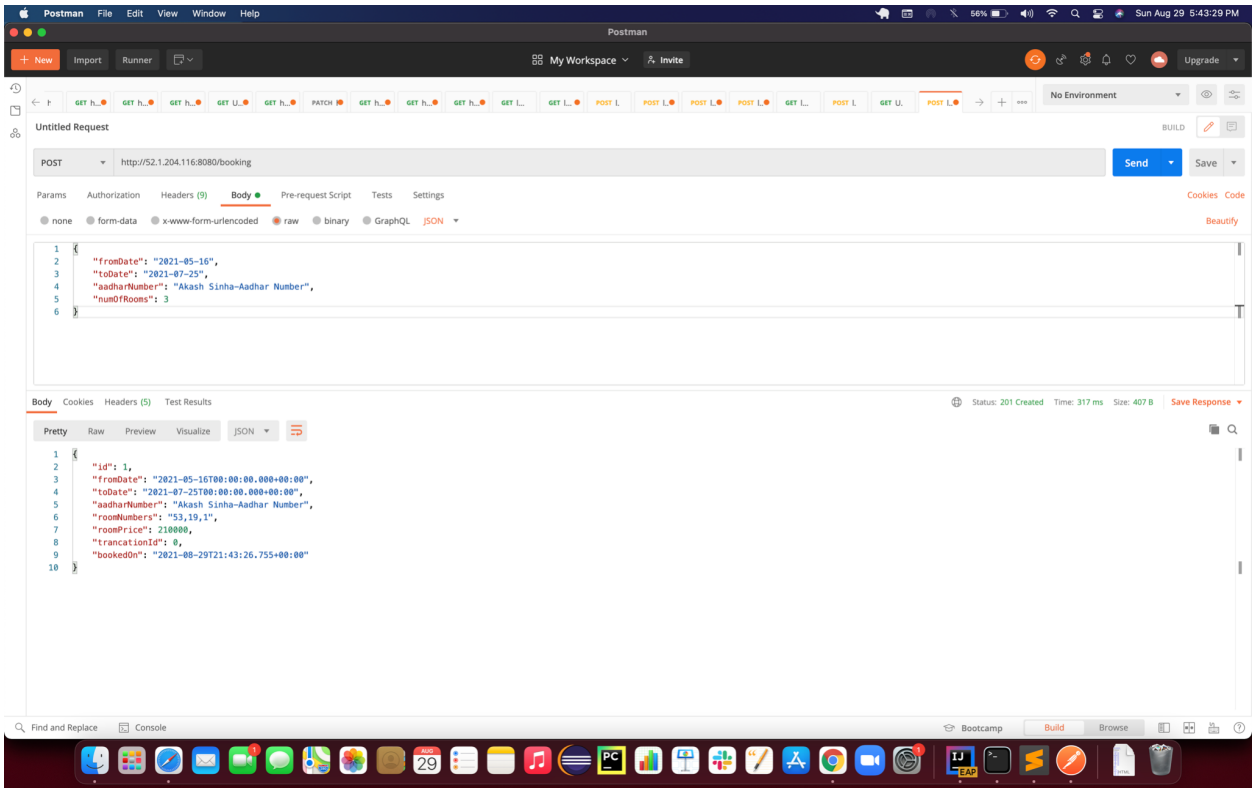
Booking service endpoints are defined as below and running on port 8080

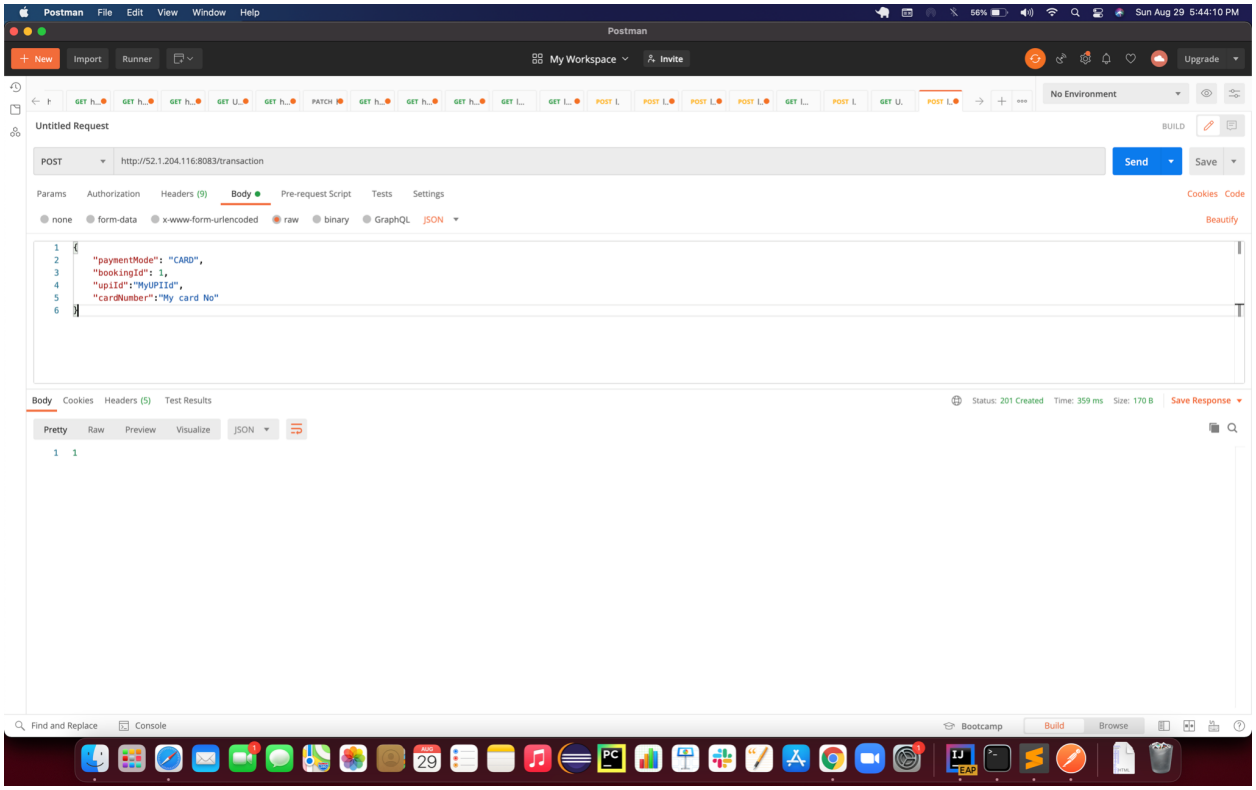
POST /booking
POST /booking/{id}/transaction

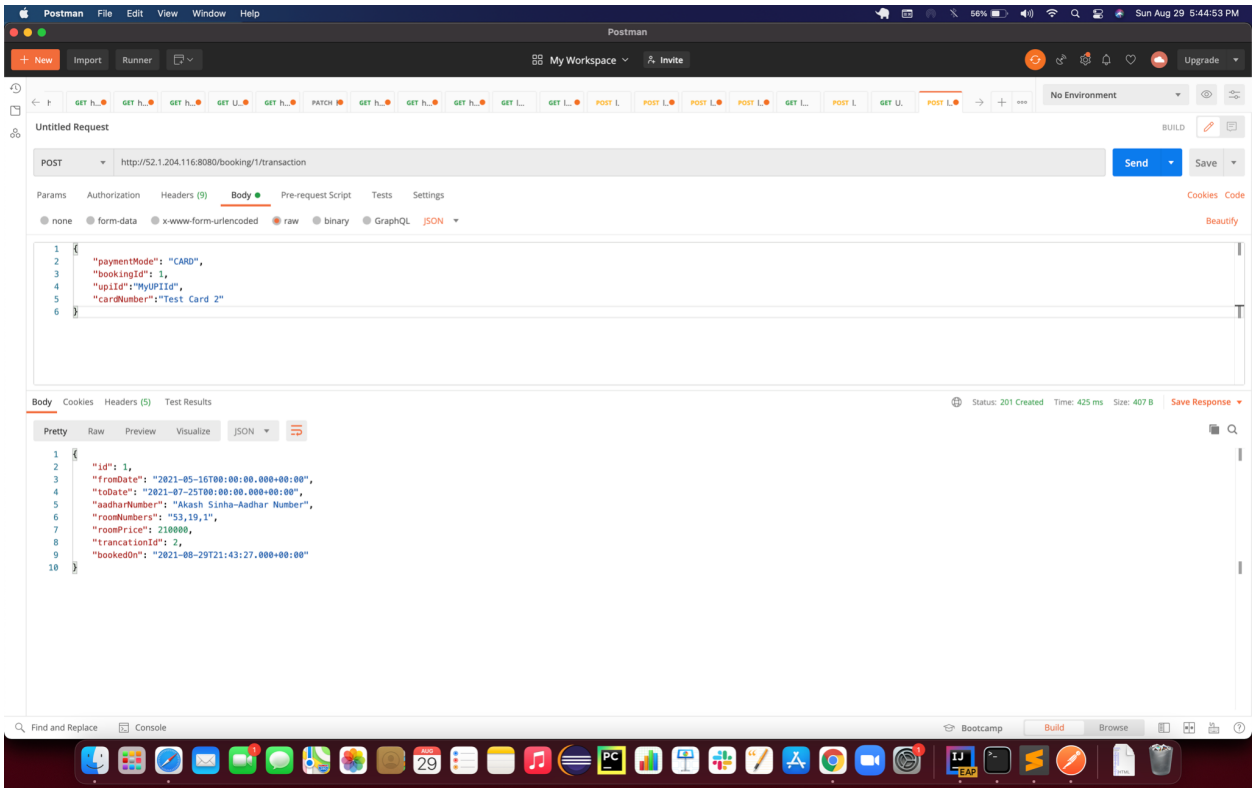
Payment service endpoints are defined as below and running on port 8083

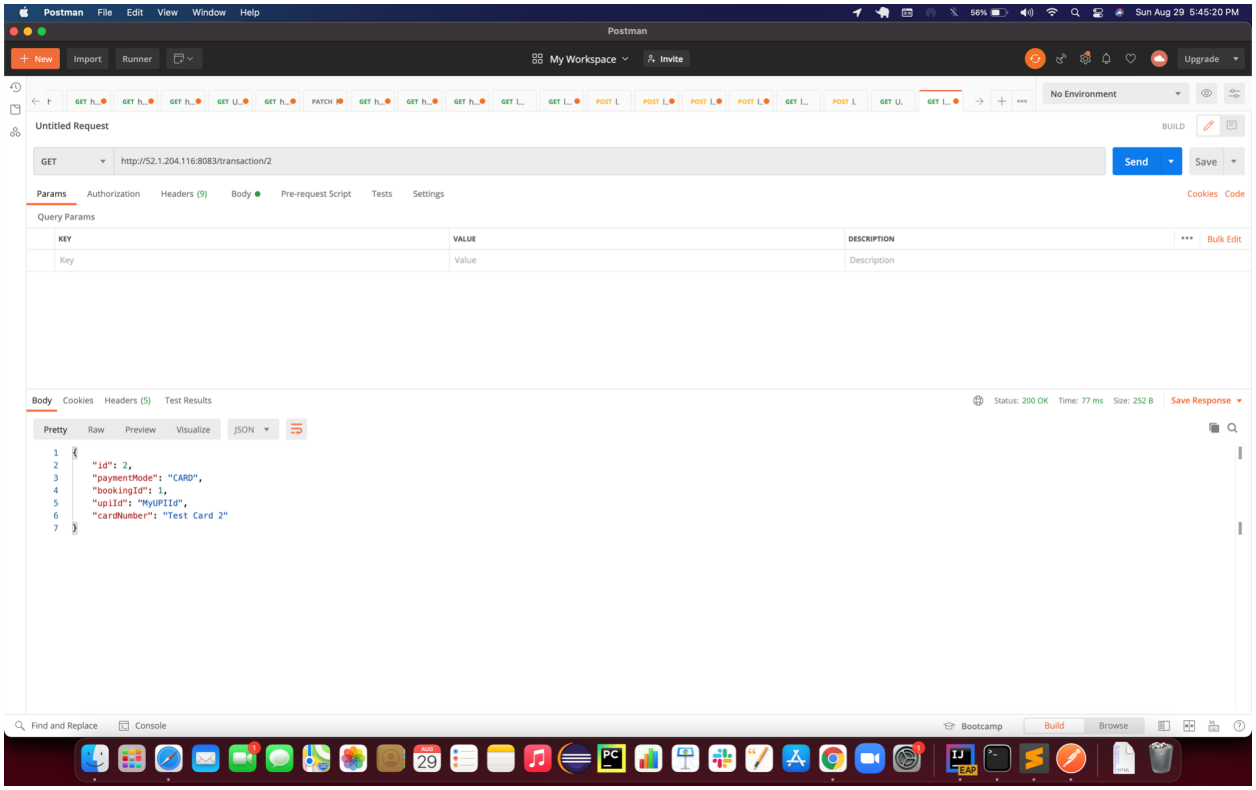
POST /transaction
GET /transaction/{id}

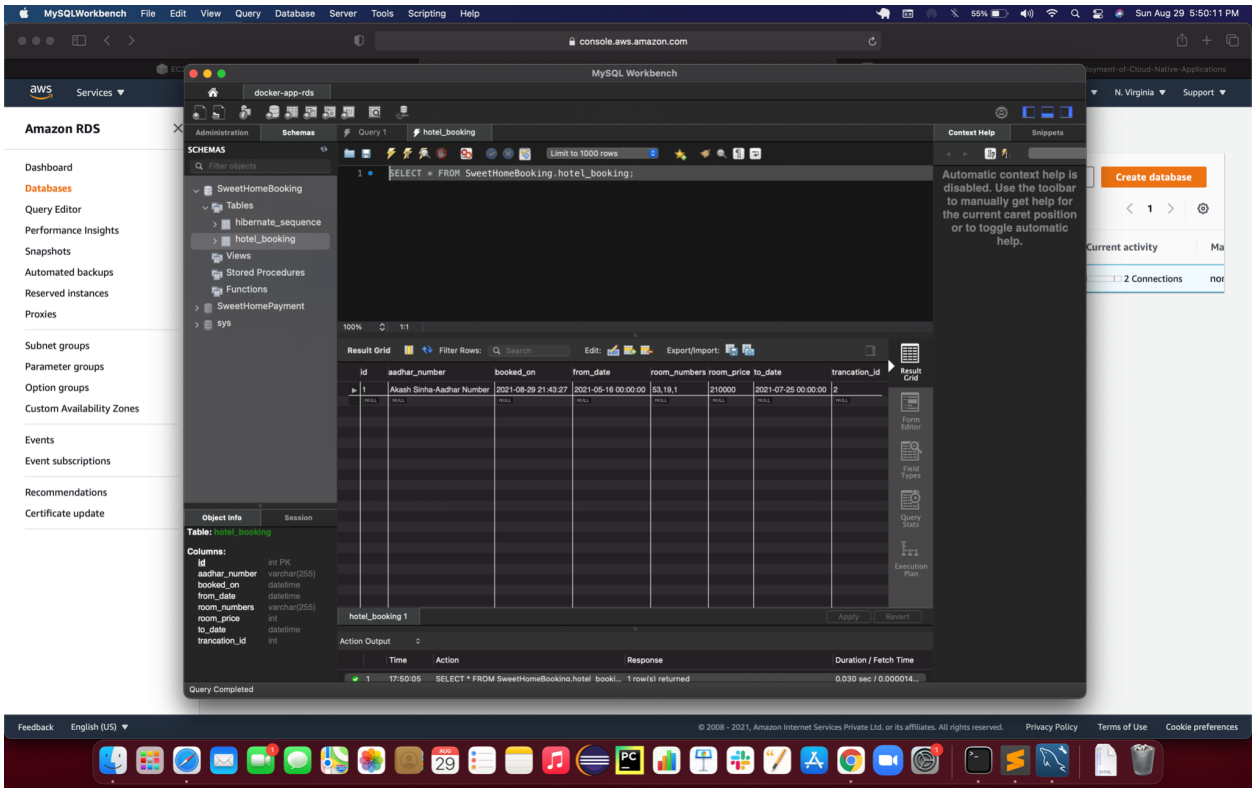
All micro services test cases are attached as below and data validation is also performed and verified using the MySQLWorkbench in order to validate data consistency.

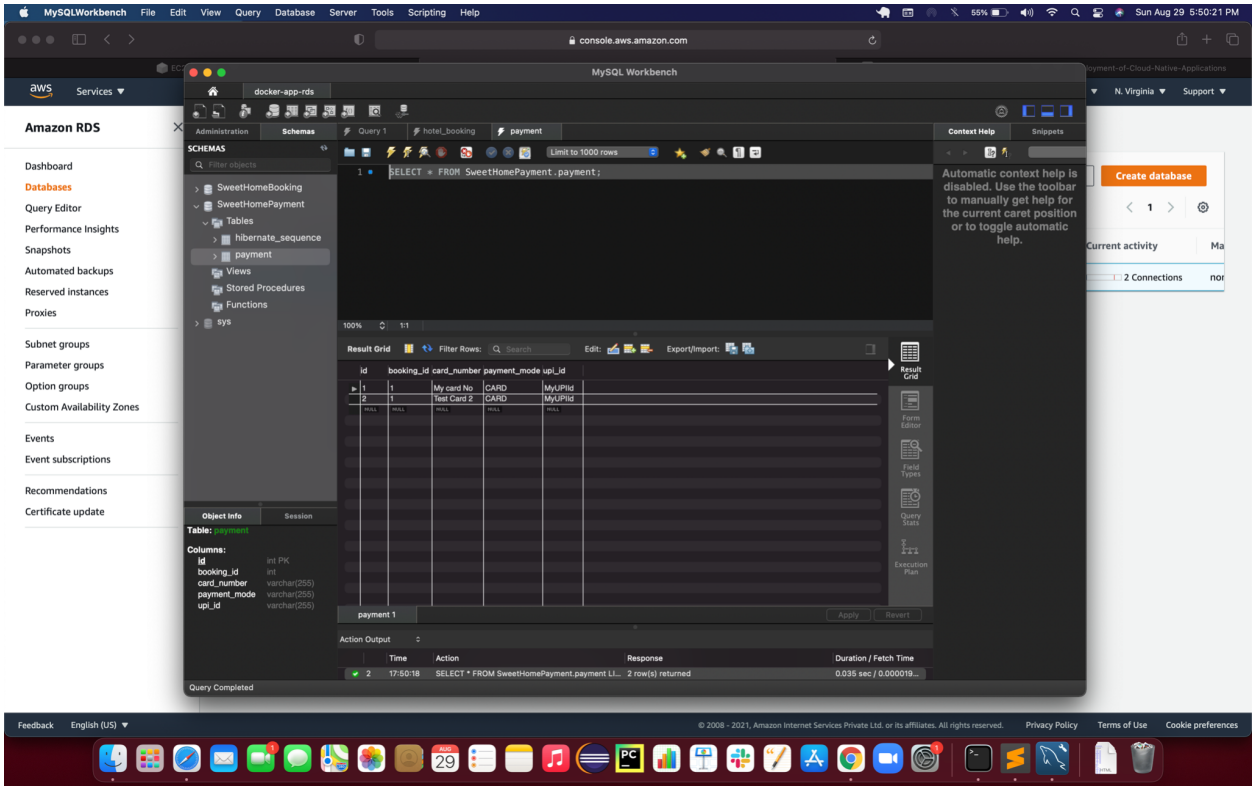



Postman
Untitled Request
POST
http://52.1.204.116:8083/transaction
Send
Save
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"paymentMode": "CARD",
"bookingId": 1,
"upiId":"MyUPIId",
"cardNumber":"My card No"
}
Body
Cookies
Headers (5)
Test Results
Status: 201 Created
Time: 359 ms
Size: 170 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
1


Postman
Untitled Request
POST
http://52.1.204.116:8080/booking/1/transaction
Send
Save
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"paymentMode": "CARD",
"bookingId": 1,
"upiId":"MyUPIId",
"cardNumber":"Test Card 2"
}
Body
Cookies
Headers (5)
Test Results
Status: 201 Created
Time: 425 ms
Size: 407 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"fromDate": "2021-05-16T00:00:00.000+00:00",
"toDate": "2021-07-25T00:00:00.000+00:00",
"aadharNumber": "Akash Sinha-Aadhar Number",
"roomNumbers": "53,19,1",
"roomPrice": 210000,
"trancationId": 2,
"bookedOn": "2021-08-29T21:43:27.000+00:00"
}


Postman
Untitled Request
GET
http://52.1.204.116:8083/transaction/2
Send
Save
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Query Params
KEY
VALUE
DESCRIPTION
Body
Cookies
Headers (5)
Test Results
Status: 200 OK
Time: 77 ms
Size: 252 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 2,
"paymentMode": "CARD",
"bookingId": 1,
"upiId": "MyUPIId",
"cardNumber": "Test Card 2"
}


MySQL Workbench
Amazon RDS
Dashboard
Databases
Query Editor
Performance Insights
Snapshots
Automated backups
Reserved instances
Proxies
Subnet groups
Parameter groups
Option groups
Custom Availability Zones
Events
Event subscriptions
Recommendations
Certificate update
SCHEMAS
SweetHomeBooking
Tables
hibernate_sequence
hotel_booking
Views
Stored Procedures
Functions
SweetHomePayment
sys
SELECT * FROM SweetHomeBooking.hotel_booking;
Result Grid
id
aadhar_number
booked_on
from_date
room_numbers
room_price
to_date
trancation_id
Akash 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
Query Completed
Create database
2 Connections

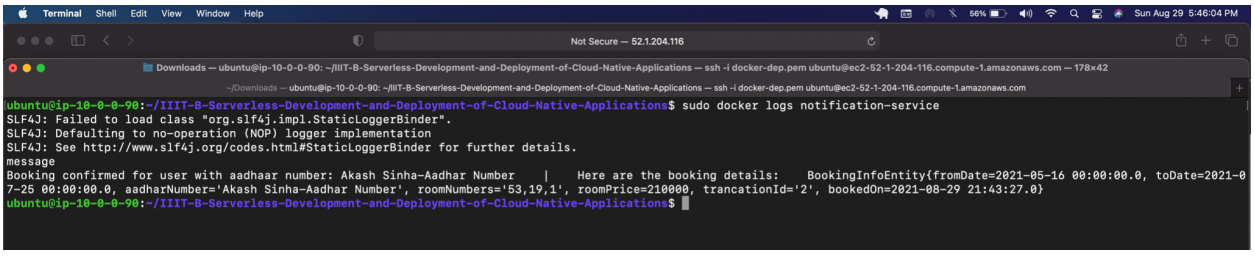


Notification service is connected with kafka and booking service, so for every successful booking service notification is being sent and printed on the console once the booking service writes the log into the kafka topic and the notification service asynchronously connects and reads the data from the 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
```



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.