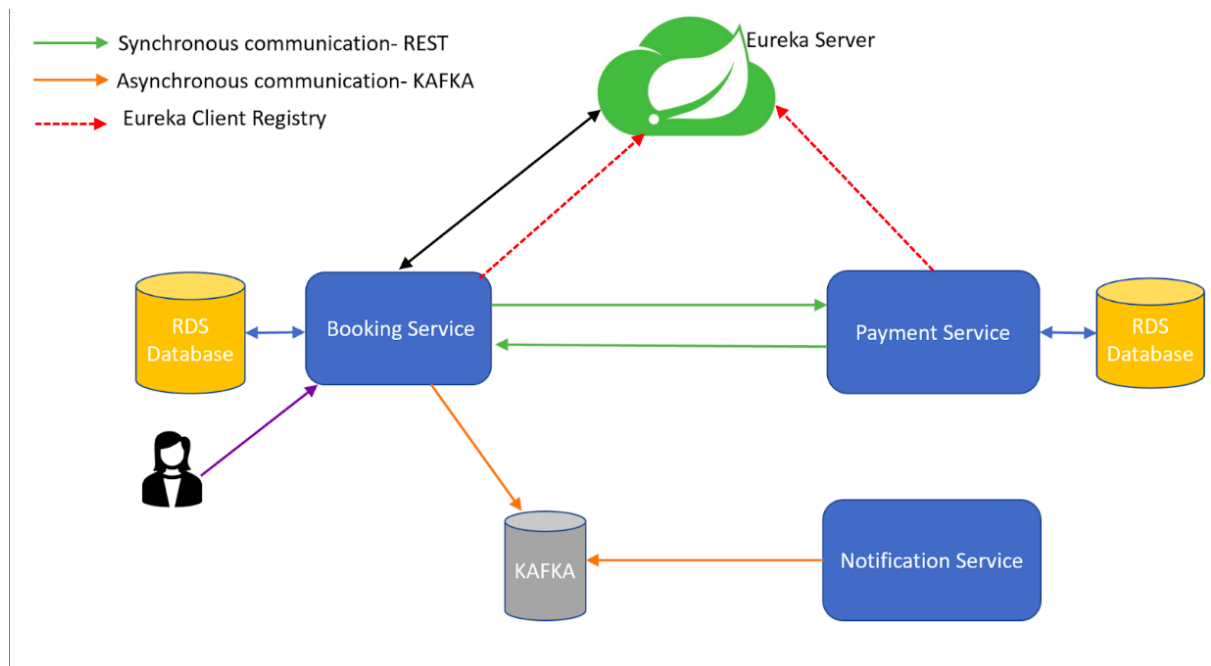


CodingLogic

using external KAFKA details explained below

LOGIC



- using external RDS database
- using external Kafka service
- passing details of kafa and database as environment in all the services so they can be picked up directly inside the pod
- docker files are placed inside folder of each service
- docker compose placed at parent of all the services folder
- path to build image from which docker file specified in docker compose file
- docker compose will take care of building images and running them by injecting environment variables as mentioned in the docker compose file

1. Installation and setup

→ **Docker setup**

- created VPC with name `vpc-69ce7f14`

The screenshot shows the AWS Management Console interface for a VPC named `vpc-69ce7f14`. The breadcrumb navigation at the top indicates the path: `VPC > Your VPCs > vpc-69ce7f14`. The VPC ID `vpc-69ce7f14` is prominently displayed at the top left, with an `Actions` dropdown menu to its right. Below this, there are two tabs: `Details` (selected) and `Info`. The `Details` tab contains a table of VPC configuration details:

Details			
VPC ID <code>vpc-69ce7f14</code>	State Available	DNS hostnames Enabled	DNS resolution Enabled
Tenancy Default	DHCP option set <code>dopt-cd3260b7</code>	Main route table <code>rtb-bf5a66c1</code>	Main network ACL <code>acl-b57169c8</code>
Default VPC Yes	IPv4 CIDR <code>172.31.0.0/16</code>	IPv6 pool -	IPv6 CIDR (Network border group) -
Network Address Usage metrics Disabled	Route 53 Resolver DNS Firewall rule groups -	Owner ID <code>063745262440</code>	

Below the details table, there are four tabs: `Resource map` (selected), `CIDRs`, `Flow logs`, and `Tags`. The `Resource map` tab shows a visual representation of the VPC resources. It includes three main sections:

- VPC**: Shows the VPC `vpc-69ce7f14` with a `Show details` link.
- Subnets (6)**: Shows subnets within the VPC, including `us-east-1a` and `subnet-e6523780`.
- Route tables (1)**: Shows the route table `rtb-bf5a66c1` which routes network traffic to resources.

- Created `ubuntu` EC2 instance with `instance type medium` with above created VPC as shown in below image with name `socket-large-instance`

EC2 > Instances > i-0876cff90fca0a49a

Instance summary for i-0876cff90fca0a49a (docker-large-instance) [Info](#)

Updated less than a minute ago

Refresh
Connect
Instance state ▼
Actions ▼

Instance ID i-0876cff90fca0a49a (docker-large-instance)	Public IPv4 address 44.204.67.178 open address	Private IPv4 addresses 172.31.81.152
IPv6 address -	Instance state Running	Public IPv4 DNS ec2-44-204-67-178.compute-1.amazonaws.com open address
Hostname type IP name: ip-172-31-81-152.ec2.internal	Private IP DNS name (IPv4 only) ip-172-31-81-152.ec2.internal	
Answer private resource DNS name IPv4 (A)	Instance type t2.medium	Elastic IP addresses -
Auto-assigned IP address 44.204.67.178 [Public IP]	VPC ID vpc-69ce7f14	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations. Learn more
IAM Role -	Subnet ID subnet-842241a5	Auto Scaling Group name -
IMDSv2 Optional		

[Details](#)
[Security](#)
[Networking](#)
[Storage](#)
[Status checks](#)
[Monitoring](#)
[Tags](#)

- updated security group inbound rules with required ports for connection from external as shown below

	Info	Info					
sgr-01214738b79269e9e	HTTP ▼	TCP	80	Custom ▼	<input type="text" value="0.0.0.0"/>	<input type="text" value="0.0.0.0"/>	Delete
sgr-0f15870b19ea4f15a	SSH ▼	TCP	22	Custom ▼	<input type="text" value="0.0.0.0"/>	<input type="text" value="0.0.0.0"/>	Delete
sgr-0201825e14d6c9163	Custom TCP ▼	TCP	9092	Custom ▼	<input type="text" value="0.0.0.0"/>	<input type="text" value="0.0.0.0"/>	Delete
sgr-084bc047ba3cfa23b	Custom TCP ▼	TCP	2181	Custom ▼	<input type="text" value="0.0.0.0"/>	<input type="text" value="0.0.0.0"/>	Delete
sgr-07b34c2abf60873d3	HTTPS ▼	TCP	443	Custom ▼	<input type="text" value="0.0.0.0"/>	<input type="text" value="0.0.0.0"/>	Delete
Add rule							

- Established `ssh connection to the instance` from public ip of Ec2 instance and public key

```

➤ AWS ssh -i ./ec2_key_pair_1.pem ubuntu@ec2-44-204-67-178.compute-1.amazonaws.com
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.19.0-1028-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Wed Jul 5 16:05:49 UTC 2023

System load:          0.13232421875
Usage of /:            29.8% of 23.0GB
Memory usage:         50%
Swap usage:           0%
Processes:            148
Users logged in:      1
IPv4 address for br-670ad7f45470: 172.24.0.1
IPv4 address for docker0: 172.17.0.1
IPv4 address for eth0: 172.31.81.152

 * Ubuntu Pro delivers the most comprehensive open source security and
   compliance features.

https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

24 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Last login: Wed Jul 5 16:05:50 2023 from 171.76.87.84
ubuntu@ip-172-31-81-152:~$

```

- Installed docker

```

● ubuntu@ip-172-31-81-152:~/LAB/FINAL_ASSIGNMENT/Sweet-home$ docker -v
Docker version 24.0.2, build cb74dfc
○ ubuntu@ip-172-31-81-152:~/LAB/FINAL_ASSIGNMENT/Sweet-home$

```

- transfer project files form local machine to Ec2

```

drwxr-xr-x 2 ubuntu ubuntu 4096 Jul 5 09:47 debugging/
drwxr-xr-x 8 ubuntu ubuntu 4096 Jul 5 09:01 upgrad-sweet-home-docker-deployment/
-rw-r--r-- 1 ubuntu ubuntu 175822 Jul 4 05:26 upgrad-sweet-home-docker-deployment.tar.gz
ubuntu@ip-172-31-81-152:~/LAB$

```

→ Kafka Setup

- using same VPC as created in docker setup `vpc-69ce7f14` and created a Ec2 instance with `t2.medium` named `large-kafka-host` for installing kafka

EC2 > Instances > i-08f80da2cd402702d

Instance summary for i-08f80da2cd402702d (large-kafka-host) [Info](#)

Updated less than a minute ago

[Refresh](#) [Connect](#) [Instance state](#) [Actions](#)

<p>Instance ID</p> <p>i-08f80da2cd402702d (large-kafka-host)</p> <p>IPv6 address</p> <p>–</p> <p>Hostname type</p> <p>IP name: ip-172-31-92-178.ec2.internal</p> <p>Answer private resource DNS name</p> <p>IPv4 (A)</p> <p>Auto-assigned IP address</p> <p>107.20.58.139 [Public IP]</p> <p>IAM Role</p> <p>–</p> <p>IMDSv2</p> <p>Optional</p>	<p>Public IPv4 address</p> <p>107.20.58.139 open address</p> <p>Instance state</p> <p>Running</p> <p>Private IP DNS name (IPv4 only)</p> <p>ip-172-31-92-178.ec2.internal</p> <p>Instance type</p> <p>t2.medium</p> <p>VPC ID</p> <p>vpc-69ce7f14</p> <p>Subnet ID</p> <p>subnet-842241a5</p>	<p>Private IPv4 addresses</p> <p>172.31.92.178</p> <p>Public IPv4 DNS</p> <p>ec2-107-20-58-139.compute-1.amazonaws.com open address</p> <p>Elastic IP addresses</p> <p>–</p> <p>AWS Compute Optimizer finding</p> <p>Opt-in to AWS Compute Optimizer for recommendations.</p> <p>Learn more</p> <p>Auto Scaling Group name</p> <p>–</p>
--	--	---

[Details](#) [Security](#) [Networking](#) [Storage](#) [Status checks](#) [Monitoring](#) [Tags](#)

- updated security group inbound rules for external access of kafka ports **2181** and **9092**

[Info](#) [Info](#)

sgr-01214738b79269e9e	HTTP	TCP	80	Custom	Q	0.0.0.0/0 X	Delete
sgr-0f15870b19ea4f15a	SSH	TCP	22	Custom	Q	0.0.0.0/0 X	Delete
sgr-0201825e14d6c9163	Custom TCP	TCP	9092	Custom	Q	0.0.0.0/0 X	Delete
sgr-084bc047ba3cfa23b	Custom TCP	TCP	2181	Custom	Q	0.0.0.0/0 X	Delete
sgr-07b34c2abf60873d3	HTTPS	TCP	443	Custom	Q	0.0.0.0/0 X	Delete

[Add rule](#)

- installed **kafka** by referring to the document provided
- started **zookeeper** and **kafka**

```
[ec2-user@ip-172-31-92-178 kafka_2.13-3.5.0]$ nc -vz localhost 2181
Ncat: Version 7.91 ( https://nmap.org/ncat )
Ncat: Connected to ::1:2181.
Ncat: 0 bytes sent, 0 bytes received in 0.04 seconds.
```

```
[ec2-user@ip-172-31-92-178 kafka_2.13-3.5.0]$ nc -vz localhost 9092
Ncat: Version 7.91 ( https://nmap.org/ncat )
Ncat: Connected to ::1:9092.
Ncat: 0 bytes sent, 0 bytes received in 0.04 seconds.
```

→ RDS setups

- created **RDS** with Database **MySQL** names **sweethome**

RDS > Databases > sweethome

sweethome

Modify Actions ▼

Summary

DB identifier sweethome	CPU 3.61%	Status Available	Class db.t2.micro
Role Instance	Current activity 20 Connections	Engine MySQL Community	Region & AZ us-east-1e

[Connectivity & security](#)
[Monitoring](#)
[Logs & events](#)
[Configuration](#)
[Zero-ETL integrations](#)
[Maintenance & backups](#)
[Tags](#)

Connectivity & security

<h5>Endpoint & port</h5> <p>Endpoint sweethome.cb9zqnw5buxr.us-east-1.rds.amazonaws.com</p> <p>Port 3306</p>	<h5>Networking</h5> <p>Availability Zone us-east-1e</p> <p>VPC vpc-69ce7f14</p> <p>Subnet group default-vpc-69ce7f14</p>	<h5>Security</h5> <p>VPC security groups sweetHomeVPC (sg-0089c3977e5751d44) Active</p> <p>Publicly accessible Yes</p>
--	--	--

- open port to access database from anywhere

EC2 > Security Groups > sg-0089c3977e5751d44 - sweetHomeVPC > Edit inbound rules

Edit inbound rules

Inbound rules control the incoming traffic that's allowed to reach the instance.

Inbound rules

Security group rule ID	Type	Protocol	Port range	Source	Description - optional	
sg-0024f7249adc6fd5e	MySQL/Aurora	TCP	3306	Custom 0.0.0.0/0		Delete

Add rule

Cancel Preview changes Save rules

- connect to mysql host from local host and create databases

```
➔ ASSIGNMENT mysql --host=sweethome.cb9zqnw5buxr.us-east-1.rds.amazonaws.com --user=admin --password=upgrad123
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 491
Server version: 5.7.42-log Source distribution

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| SweetHomeBooking |
| SweetHomePayment |
| innodb |
| mysql |
| performance_schema |
| sys |
+-----+
7 rows in set (0.34 sec)
```

2. Docker File creation

payment service

→ updated mysql connection made them to connect from environment variables passed

```
spring.datasource.url = jdbc:mysql://${MYSQL_HOST:localhost}:${MYSQL_HOST_PORT:3306}/SweetHomePayment
spring.datasource.username = ${MYSQL_USER:admin}
spring.datasource.password = ${MYSQL_PASSWORD:upgrad123}
```

→ wrote `Dockerfile` to build images

```
FROM maven:3.8.2-jdk-11 AS builder
WORKDIR /usr/src/app
COPY src ./src
COPY pom.xml .
RUN mvn clean install -DskipTests

FROM openjdk:11.0.11-jdk-slim
WORKDIR /usr/app
COPY --from=builder /usr/src/app/target/paymentService.jar /usr/app/paymentService.jar
ENV PATH="${PATH}:${JAVA_HOME}/bin"
EXPOSE 8083
ENTRYPOINT ["java", "-jar", "/usr/app/paymentService.jar"]
```

booking service

→ updated mysql connection made them to connect from environment variables passed

```
spring.datasource.url = jdbc:mysql://${MYSQL_HOST:localhost}:${MYSQL_HOST_PORT:3306}/SweetHomeBooking
spring.datasource.username = ${MYSQL_USER:admin}
spring.datasource.password = ${MYSQL_PASSWORD:upgrad123}
```

→ updated `kafka` connection details to be acceptable from environment variables

```
spring.datasource.url = jdbc:mysql://${MYSQL_HOST:localhost}:${MYSQL_HOST_PORT:3306}/SweetHomeBooking
spring.datasource.username = ${MYSQL_USER:admin}
spring.datasource.password = ${MYSQL_PASSWORD:upgrad123}
```

→ wrote `Dockerfile` to build images

```
FROM maven:3.8.2-jdk-11 AS builder
WORKDIR /usr/src/app
COPY src ./src
COPY pom.xml .
RUN mvn clean install -DskipTests

FROM openjdk:11.0.11-jdk-slim
WORKDIR /usr/app
COPY --from=builder /usr/src/app/target/bookingService.jar /usr/app/bookingService.jar
ENV PATH="${PATH}:${JAVA_HOME}/bin"
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "/usr/app/bookingService.jar"]
```

notification service

→ updated `kafka` connection details to be acceptable from environment variables

```
String kafkaConnectionString = System.getenv("KAFKA_HOST")+" ":""+System.getenv("KAFKA_HOST_PORT");
properties.put("bootstrap.servers", kafkaConnectionString);
```

→ wrote docker file to build image

```
FROM maven:3.8.2-jdk-11 AS builder
WORKDIR /usr/src/app
COPY src ./src
COPY pom.xml .
RUN mvn clean compile assembly:single

FROM openjdk:11.0.11-jdk-slim
WORKDIR /usr/app
COPY --from=builder /usr/src/app/target/notificationService-jar-with-dependencies.jar /usr/app/notificationService.jar
ENV PATH="${PATH}:${JAVA_HOME}/bin"
#EXPOSE 8761
ENTRYPOINT ["java", "-jar", "/usr/app/notificationService.jar"]
```

eureka service

→ wrote docker file for eureka

```
FROM maven:3.8.2-jdk-11 AS builder
WORKDIR /usr/src/app
COPY src ./src
COPY pom.xml .
RUN mvn clean install -DskipTests

FROM openjdk:11.0.11-jdk-slim
WORKDIR /usr/app
COPY --from=builder /usr/src/app/target/eurekaServer.jar /usr/app/eurekaServer.jar
ENV PATH="${PATH}:${JAVA_HOME}/bin"
EXPOSE 8761
ENTRYPOINT ["java", "-jar", "/usr/app/eurekaServer.jar"]
```

3. Docker-compose file

- docker compose file will `build images` and `run them in specified network`
- docker compose file contains environment variables which are given at time of run
- these environment variables are the `ip for external kafka` and `IP of sql database` and respective `ports`
- while running these values to be filled and just fire `sudo docker compose up`

```
version: '3.3'

services:

  eureka-service:
    build: eureka-server
    container_name: eureka-server
    image: sweet-home/eureka-server:1.0.1
    ports:
      - "8761:8761"
    networks:
      - "microservicesnet"
    hostname: eureka-service

  notification-service:
    build: notification-service
    container_name: notification-service
    image: sweet-home/notification-service:1.0.1
    environment:
      KAFKA_HOST: "107.20.58.139"
      KAFKA_HOST_PORT: "9092"
    networks:
      - "microservicesnet"
    hostname: notification-service

  booking-service:
    build: booking-service
    container_name: booking-service
    image: sweet-home/booking-service:1.0.1
    ports:
```

```

- "8080:8080"
networks:
  - "microservicesnet"
environment:
  MYSQL_HOST: "sweethome.cb9zqnw5buxr.us-east-1.rds.amazonaws.com"
  MYSQL_HOST_PORT: 3306
  MYSQL_USER: admin
  MYSQL_PASSWORD: upgrad123
  KAFKA_HOST: "107.20.58.139"
  KAFKA_HOST_PORT: "9092"
depends_on:
  - eurekaservice
hostname: booking-service

paymentservice:
  build: paymentservice
  container_name: payment-service
  image: sweet-home/paymentservice:1.0.1
  ports:
    - "8083:8083"
  networks:
    - "microservicesnet"
  environment:
    MYSQL_HOST: "sweethome.cb9zqnw5buxr.us-east-1.rds.amazonaws.com"
    MYSQL_HOST_PORT: 3306
    MYSQL_USER: admin
    MYSQL_PASSWORD: upgrad123
    KAFKA_HOST: "107.20.58.139"
    KAFKA_HOST_PORT: "9092"
  depends_on:
    - eurekaservice
  hostname: payment-service

networks:
  microservicesnet:
    driver: bridge

```

4. Service deployment

Instruction to run

Pre requisites

- docker installed with compose plugin
- RDS is up and running with the databases `SweetHomeBooking` and `SweetHomePayment`
- Kafka is up and running as a external kafka
- zip folder of the project

STEPS

1. unzip the project folder `Sweet-home.zip` then you will end up with `Sweet-home` folder
2. `cd Sweet-home`
3. open `docker-compose.yml` with any editor example: `vi docker-compose.yml`
4. update `KAFKA_HOST`, `KAFKA_HOST_PORT`, `MYSQL_HOST`, `MYSQL_HOST_PORT`, `MYSQL_USER`, `MYSQL_PASSWORD` with your proper working values.

5. after filling values by referring to samples given fire following command to run the application
6. `sudo docker compose up`

after successful execution of above command images will be built and services will be starting

running docker images

```

ubuntu@ip-172-31-81-152:~/LAB/FINAL_ASSIGNMENT/Sweet-home$ sudo docker compose ps
NAME                IMAGE                                COMMAND                                SERVICE    CREATED   STATUS    PORTS
booking-service     sweet-home/booking-service:1.0.1    "java -jar /usr/app/..."          bookingservice  6 minutes ago    Up 6 minutes    0.0.0.0:80
eureka-server       sweet-home/eureka-server:1.0.1      "java -jar /usr/app/..."          eureka-service  6 minutes ago    Up 6 minutes    0.0.0.0:87
notification-service sweet-home/notification-service:1.0.1 "java -jar /usr/app/..."          notification-service  6 minutes ago    Up 6 minutes    0.0.0.0:80
payment-service     sweet-home/payment-service:1.0.1     "java -jar /usr/app/..."          payment-service  6 minutes ago    Up 6 minutes    0.0.0.0:80

```

docker images created after docker compose up

```

ubuntu@ip-172-31-81-152:~/LAB/FINAL_ASSIGNMENT/Sweet-home$ sudo docker images
REPOSITORY              TAG          IMAGE ID       CREATED        SIZE
sweet-home/notification-service  1.0.1       6fe850ba7f67  13 hours ago  435MB
sweet-home/booking-service      1.0.1       fddc074d9332  15 hours ago  494MB
sweet-home/payment-service       1.0.1       eef13b3e43da  23 hours ago  482MB
sweet-home/eureka-server        1.0.1       66019c1c89fc  23 hours ago  466MB

```

eureka server

after deploying the services they will be appearing in eureka

The screenshot shows the Spring Eureka web interface. The top navigation bar includes 'HOME' and 'LAST 1000 SINCE STARTUP'. The main content area is divided into several sections:

- System Status:** A table showing environment and data center details.

Environment	N/A	Current time	2023-07-05T18:14:49 +0000
Data center	N/A	Uptime	00:44
		Lease expiration enabled	false
		Renews threshold	5
		Renews (last min)	4
- EMERGENCY!** A red warning message: "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:** A section for distributed system replicas.
- Instances currently registered with Eureka:** A table listing registered services.

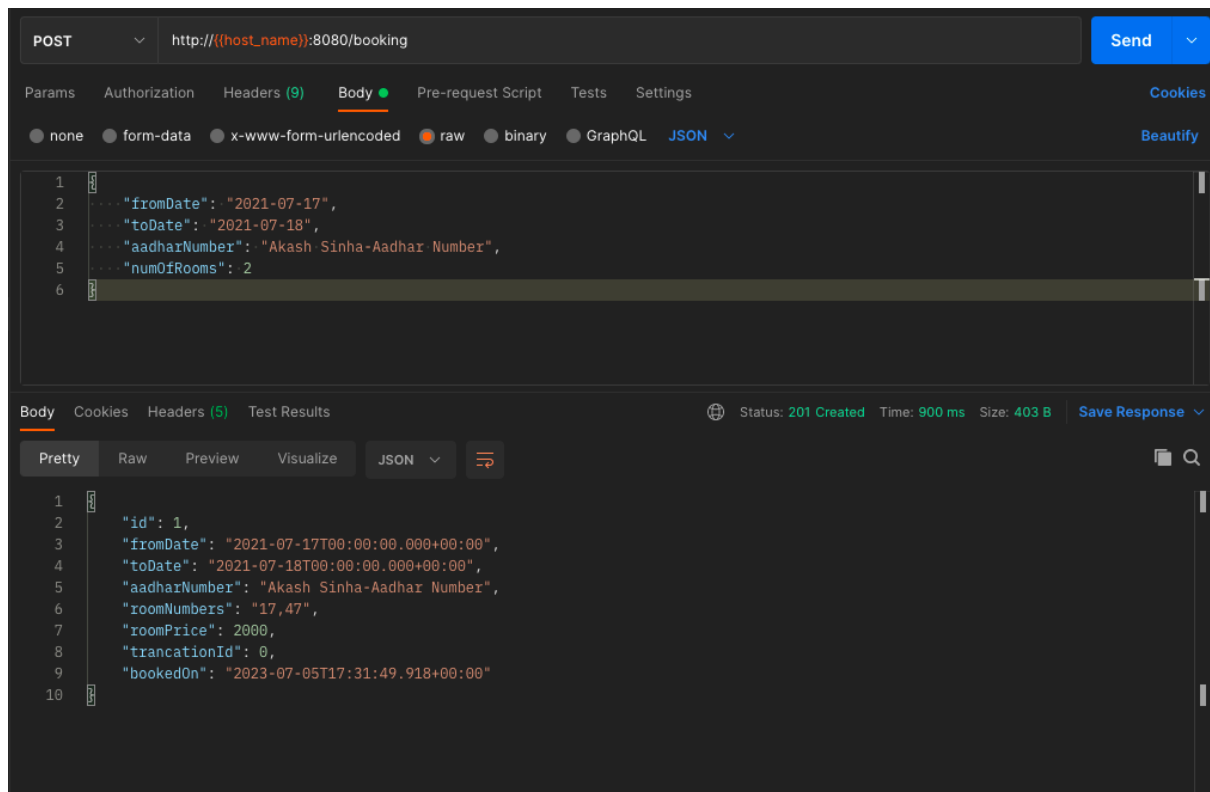
Application	AMIs	Availability Zones	Status
BOOKING-SERVICE	n/a (1)	(1)	UP (1) - booking-service:booking-service
PAYMENT-SERVICE	n/a (1)	(1)	UP (1) - payment-service:payment-service:8083
- General Info:** A section for general information.

to see eureka service <http://<Ec2-instanceIP>:8761/> replace [Ec2-instanceIP](#) with ec2 public ip

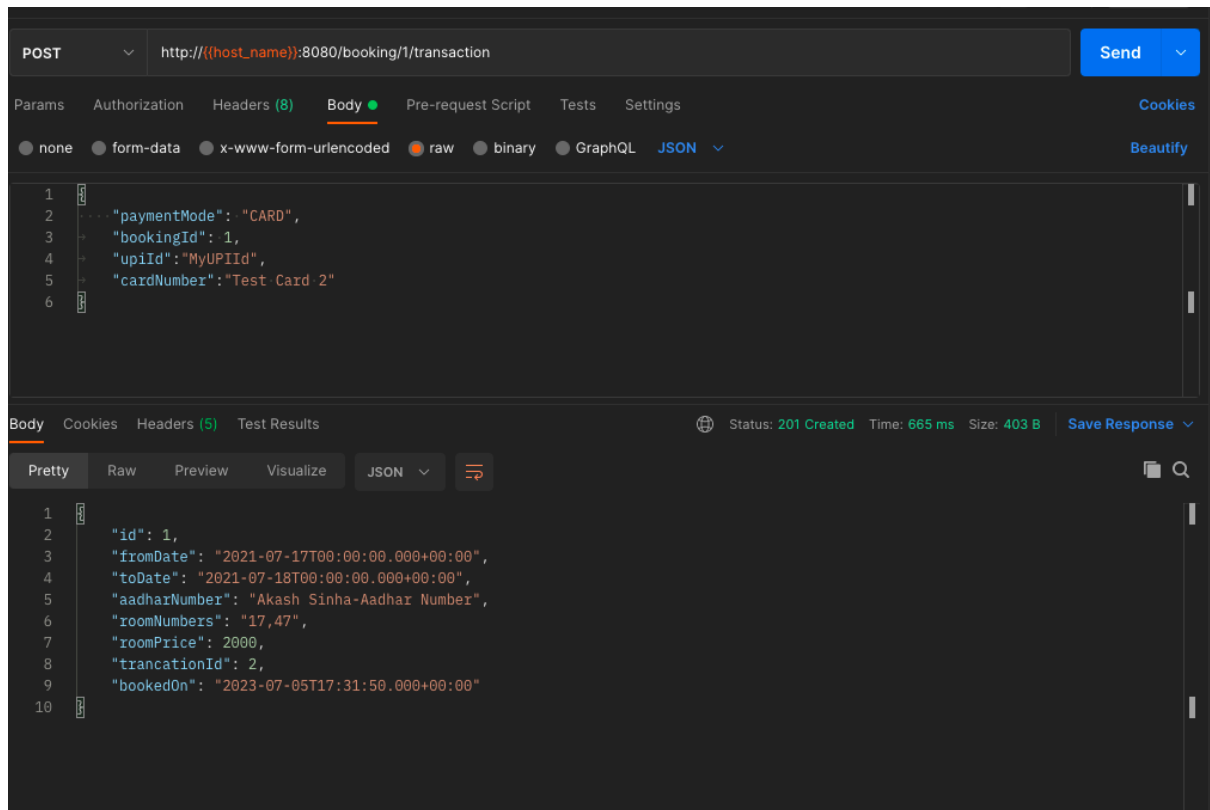
5. APIs testing

[booking service](#)

API for booking the rooms



api for making transaction for booked rooms



corresponding pod logs for booking service on request

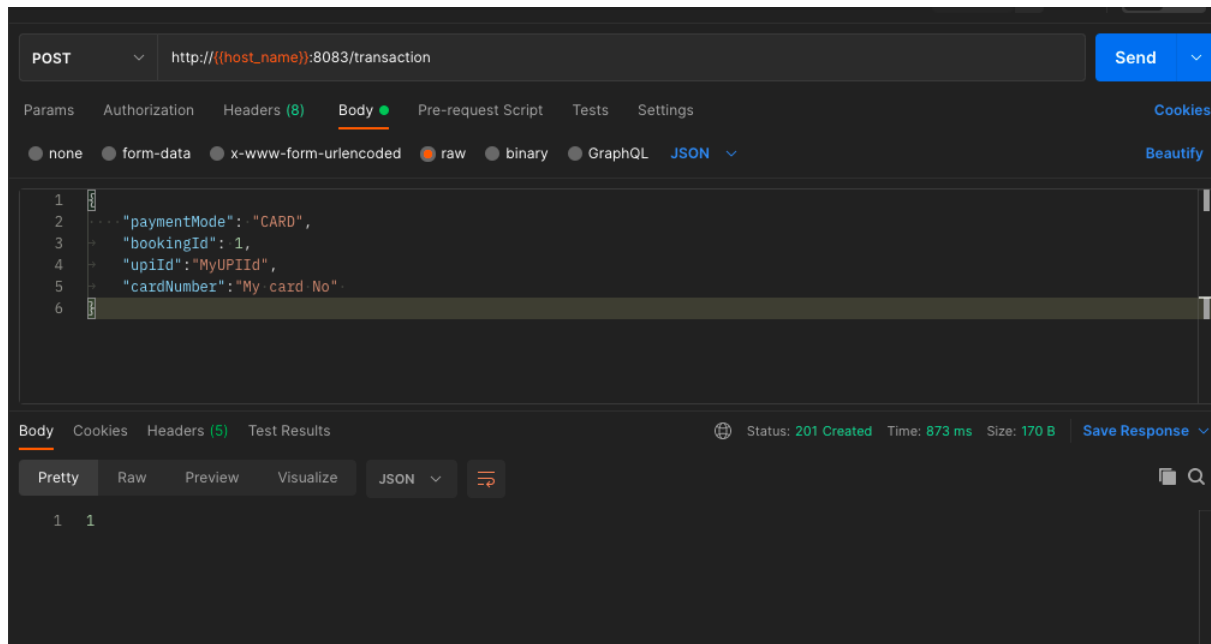
```

2023-07-05 17:31:42.308 INFO 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Setting all instance registry info from the Eureka server
2023-07-05 17:31:42.598 INFO 1 --- [nio-8080-exec-0] com.netflix.discovery.DiscoveryClient : The response status is 200
2023-07-05 17:31:49.863 INFO 1 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2023-07-05 17:31:49.863 INFO 1 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2023-07-05 17:31:49.865 INFO 1 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms
Number of Days: 1
BookingInfoEntity(fromDate=Sat Jul 17 00:00:00 UTC 2021, toDate=Sun Jul 18 00:00:00 UTC 2021, aadharNumber='Akash Sinha-Aadhar Number', roomNumbers='17,47', roomPrice=2000, transactionId='0', bookedOn=Wed Jul 05 17:31:49 UTC 2023)
Hibernate: select next_val as id_val from hibernate_sequence for update
Hibernate: update hibernate_sequence set next_val= ? where next_val=?
Hibernate: insert into hotel_booking (aadhar_number, booked_on, from_date, room_numbers, room_price, to_date, transaction_id, id) values (?, ?, ?, ?, ?, ?, ?, ?)
PaymentDto{bookingId=1, paymentMode='CARD', upiId='MyUPIId', cardNumber='Test Card 2'}
Hibernate: select bookinginfo_id as id1_0_0_, bookinginfo_aadhar_number as aadhar_n2_0_0_, bookinginfo_booked_on as booked_o3_0_0_, bookinginfo_from_date as from_date4_0_0_, bookinginfo_room_numbers as room_num5_0_0_, bookinginfo_room_price as room_pri6_0_0_, bookinginfo_to_date as to_date7_0_0_, bookinginfo_transaction_id as transacti8_0_0_ from hotel_booking bookinginfo where bookinginfo_id=?
Hibernate: update hotel_booking set aadhar_number=?, booked_on=?, from_date=?, room_numbers=?, room_price=?, to_date=?, transaction_id=? where id=?
2023-07-05 17:36:11.325 INFO 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via configuration
ubuntu@ip-172-31-81-152:~/LAB/FINAL_ASSIGNMENT/Sweet-home$

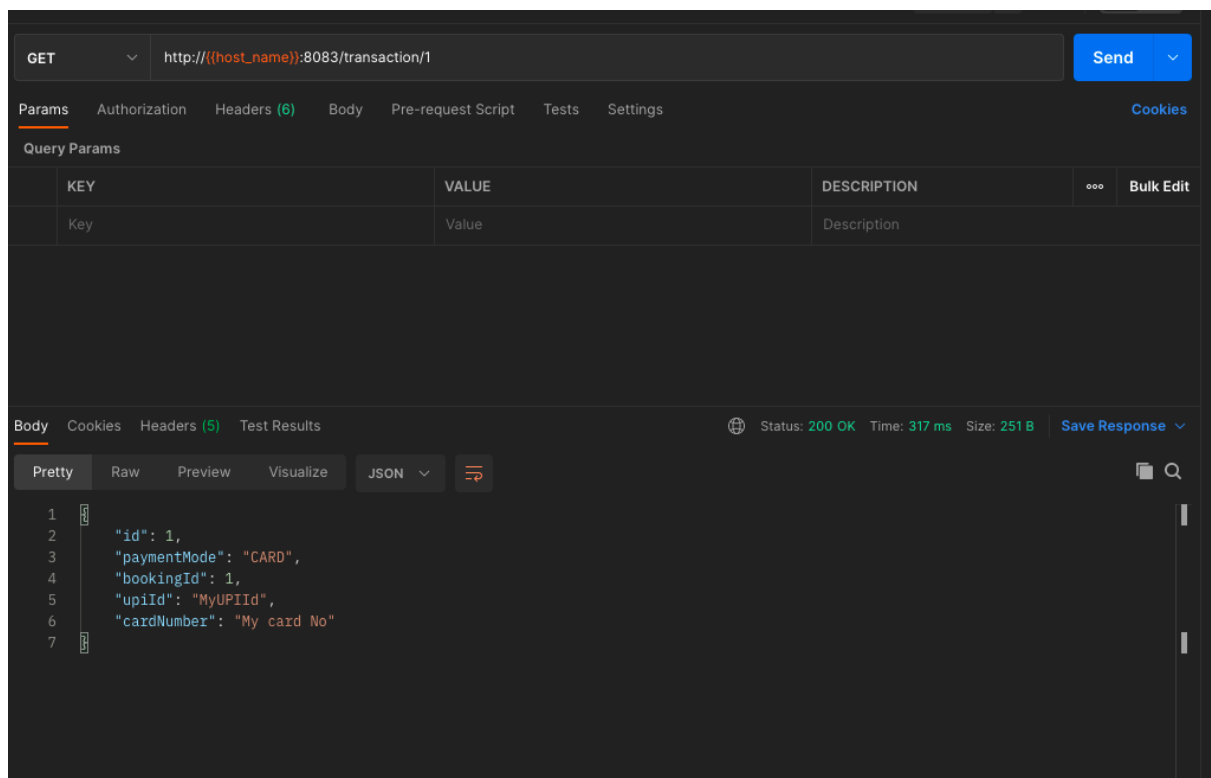
```

[paymentservice](#)

API for payment service payment details



API for payment service get payment details



corresponding pod logs on firing request to service

```

Hibernate: insert into payment (booking_id, card_number, payment_mode, upi_id, id) values (?, ?, ?, ?, ?)
Hibernate: select paymentdet0_id as id1_0_0_, paymentdet0_booking_id as booking_2_0_0_, paymentdet0_card_number as card_num3_0_0_, paymentdet0_payment_mode
as payment_4_0_0_, paymentdet0_upi_id as upi_id5_0_0_ from payment paymentdet0 where paymentdet0_id=?
2023-07-05 17:36:10.560 INFO 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via configuration

```

notification service

logs of notification on getting message on registered topic in kafka

```
ubuntu@ip-172-31-81-152:~/LAB/FINAL_ASSIGNMENT/Sweet-home$ sudo docker logs de0b7da7a43
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 aadhaar number: Akash Sinha-Aadhar Number | Here are the booking details: BookingInfoEntity{fromDate=2021-07-17 00:00:00.0, toDate=2021-07-18 00:00:00.0, aadharNumber='Akash Sinha-Aadhar Number', roomNumbers='17,47', roomPrice=2000, transactionId='2', bookedOn=2023-07-05 17:31:50.0}
ubuntu@ip-172-31-81-152:~/LAB/FINAL_ASSIGNMENT/Sweet-home$
```

Instruction to run

Pre requisites

- docker installed with compose plugin
 - RDS is up and running with the databases `SweetHomeBooking` and `SweetHomePayment`
- Kafka is up and running as a external kafka
- zip folder of the project

STEPS

1. unzip the project folder `Sweet-home.zip` then you will end up with `Sweet-home` folder
2. `cd Sweet-home`
3. open `docker-compose.yml` with any editor example: `vi docker-compose.yml`
4. update `KAFKA_HOST`, `KAFKA_HOST_PORT`, `MYSQL_HOST`, `MYSQL_HOST_PORT`, `MYSQL_USER`, `MYSQL_PASSWORD` with your proper working values.
5. after filling values by referring to samples, execute below command to run the application from sweet-home folder where `docker-compose.yml` is present
6. `sudo docker compose up`
7. application will be up after that

contributions

Vinod Patil:

- written docker-compose.yml
- written bookingservice docker file and changes in booking service

Rupa Cherlopalli:

- written paymentservice docker file and changes in paymentservice