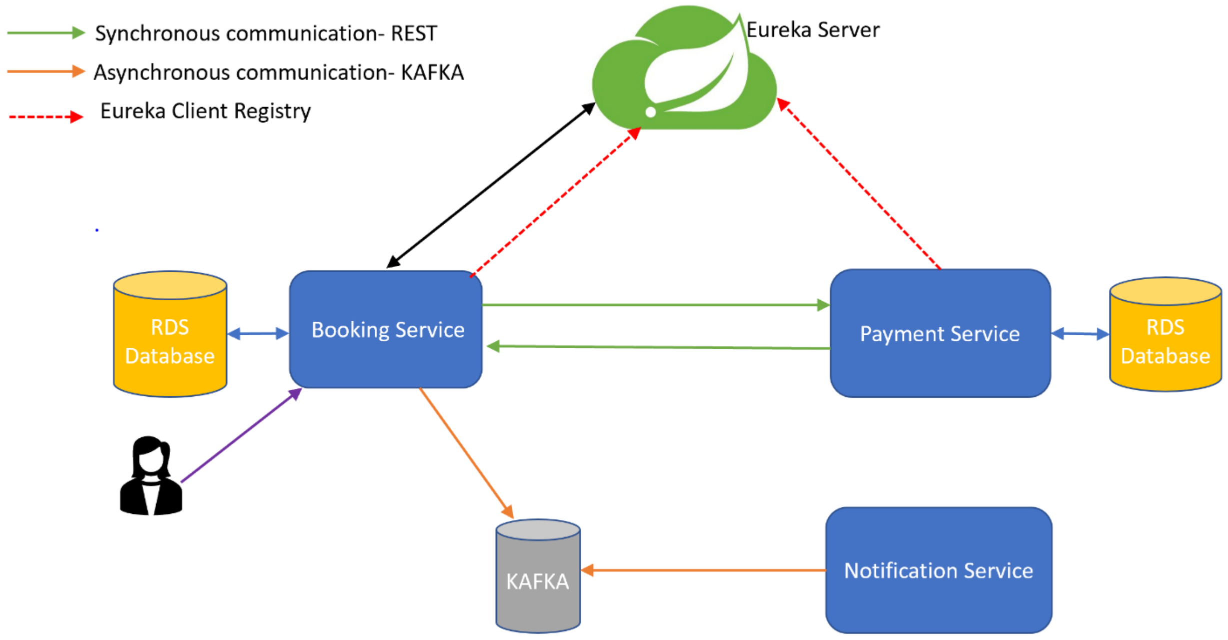
Coding Logic – Booking, Payment, Eureka & Notification Service Assignment

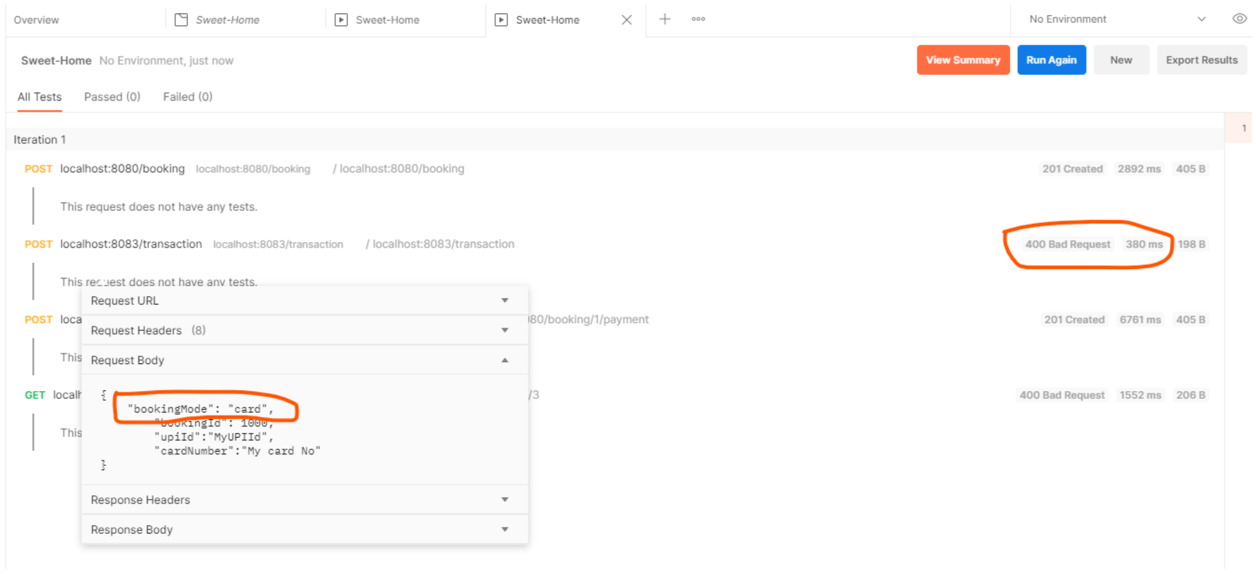
Table of Contents

Overall Application workflow:



Assumptions / Known Issues / Limitations wrt Assignment requirements:

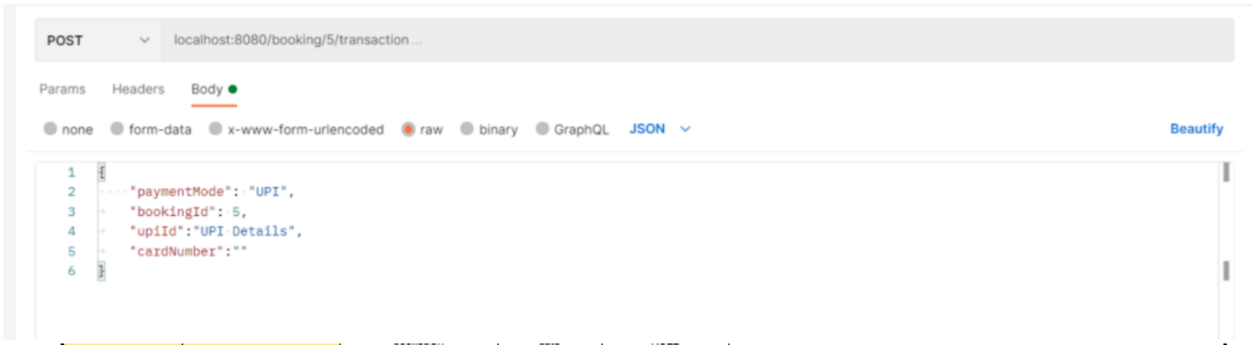
1. Given Postman API document, test for payment service endpoint 1 "*localhost:8083/booking*" is configured with wrong JSON request body. JSON key, "*paymentMode*" is wrongly mentioned as "*bookingMode*", this causing exception. **Handled it as "Invalid mode of payment" HTTP 400.**



```
Sweet-Home  No Environment, just now

Iteration 1
POST localhost:8080/booking  localhost:8080/booking  / localhost:8080/booking        201 Created  2892 ms  405 B
  This request does not have any tests.
POST localhost:8083/transaction  localhost:8083/transaction  / localhost:8083/transaction  400 Bad Request  380 ms  198 B
  This request does not have any tests.
POST loca  80/booking/1/payment                                                      201 Created  6761 ms  405 B
  This
GET localh  /3                                                                       400 Bad Request  1552 ms  206 B
  This
```

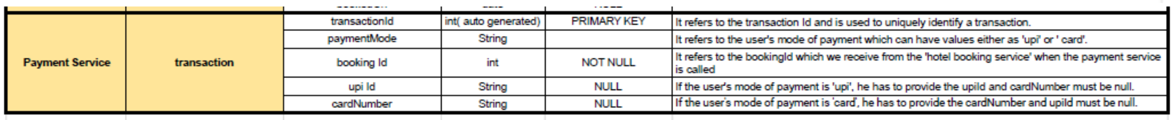
```
{
  "bookingMode": "card",
      "bookingId": 1000,
      "upiId":"MyUPIId",
      "cardNumber":"My card No"
}
```

2. paymentMode (upi / card) in Response Body is shown in ALL CAPS, whereas DB schema mentions card type in LOWERCASE – **Handled them assuming these as requirements.**



localhost:8080/booking/5/transaction

```
{
    "paymentMode": "UPI",
    "bookingId": 5,
    "upiId":"UPI Details",
    "cardNumber":""
}
```



Payment Service	transaction	transactionId	int(auto generated)	PRIMARY KEY	It refers to the transaction Id and is used to uniquely identify a transaction.
		paymentMode	String		It refers to the user's mode of payment which can have values either as 'upi' or ' card'.
		booking Id	int	NOT NULL	It refers to the bookingId which we receive from the 'hotel booking service' when the payment service is called
		upi Id	String	NULL	If the user's mode of payment is 'upi', he has to provide the upiId and cardNumber must be null.
		cardNumber	String	NULL	If the user's mode of payment is 'card', he has to provide the cardNumber and upiId must be null.

3. As it is assumed that Payment service endpoints are accessible only through Booking service, performing all validations in Booking Service alone isn't sufficient to handle the Postman tests, where tests hitting payment service endpoint directly where failing, as exceptions/validations aren't handled internal to Payment service: **Handled by implementing custom exception handling & validations in both Booking & payment services.**
4. If Card details are provided for UPI payment mode, along with cardNumber, then instead of throwing exception (that Invalid payment mode, 400) – **Handling it as explained below:**
   By nullifying the alternate payment details, if present, and storing only the data corresponding to paymentMode to the paymentService & transaction table of payments DB.
   (i.e) For below req body, upiId in table will be stored as null; only cardNumber will be saved.
   ```
   {
     "bookingId": 1,
     "paymentMode": "CARD",
     “upiId": "MyUPIId",
     "cardNumber": "My card No"
   }
   ```
5. Payment Service allows multiple transactions for same booking ID, i.e., Payments DB transaction table will/shall contain multiple unique transaction ID rows for same booking ID. **This validation is not mentioned in requirement hence not handled, considering assignment requirements.**

Service Implementations:

Following services are implemented are part of this assignment.

1. Eureka Server
2. Booking Service, with backend RDS DB
3. Payment Service, with backend RDS DB
4. Notification Service, Listening to a Kafka messaging broker

Ensured Constructor Autowiring at all applicable instances, across the services.

1. Eureka Server – Implementation details

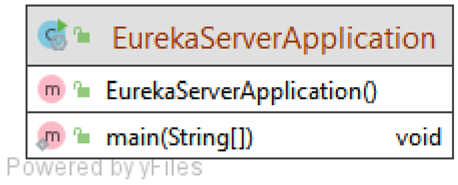
A standalone Spring Boot Application, configured as Eureka server, listening for clients in port 8761.

Dependencies added:

NetFlix Eureka Server dependencies

```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
```

Class Diagram:



Important Annotations:

@SpringBootApplication – Marks class as Spring Boot application, that shall contain Beans & triggers Spring Boot autoconfiguration & component scanning.

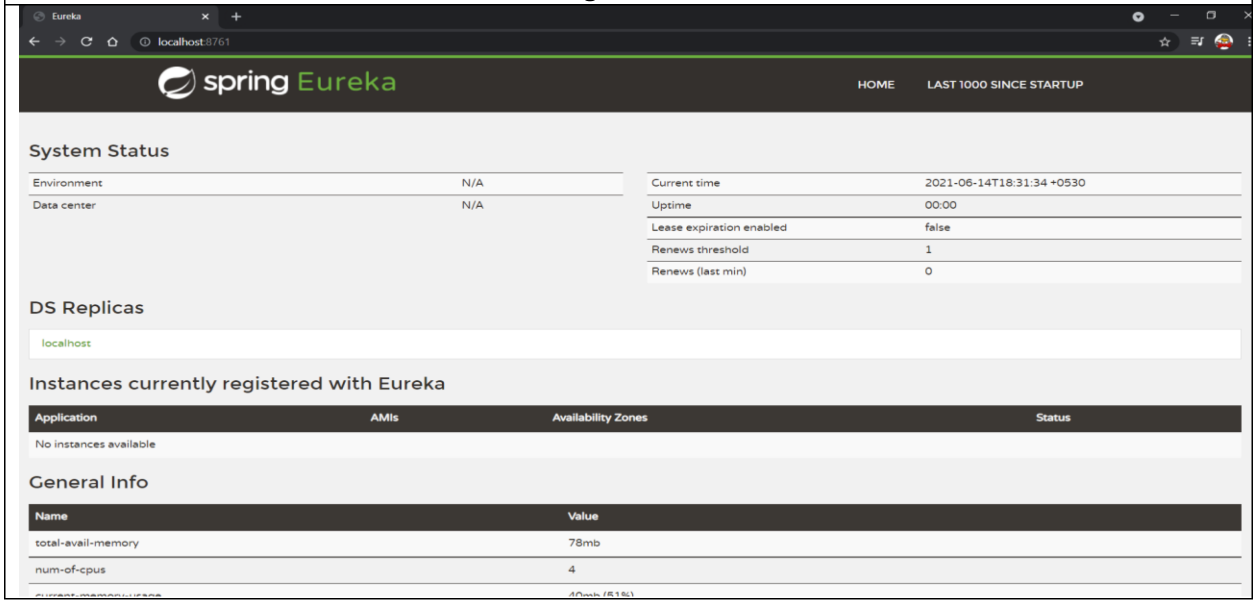
@EnableEurekaServer – Enables Eureka server in a SpringBootApplication.

Server Property configurations:

Eureka server configurations are added to application.yml. Client registration & discovery properties are set as false, to make current Spring Boot Application as a Eureka server, instead of Eureka Client. Server Port configurations are done here.

```
#Eureka server configurations
server:
  port: 8761

eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
```

Eureka Server screenshot – with no services registered.



System Status

Environment	N/A
Data center	N/A

Current time	2021-06-14T18:31:34 +0530
Uptime	00:00
Lease expiration enabled	false
Renews threshold	1
Renews (last min)	0

DS Replicas

localhost

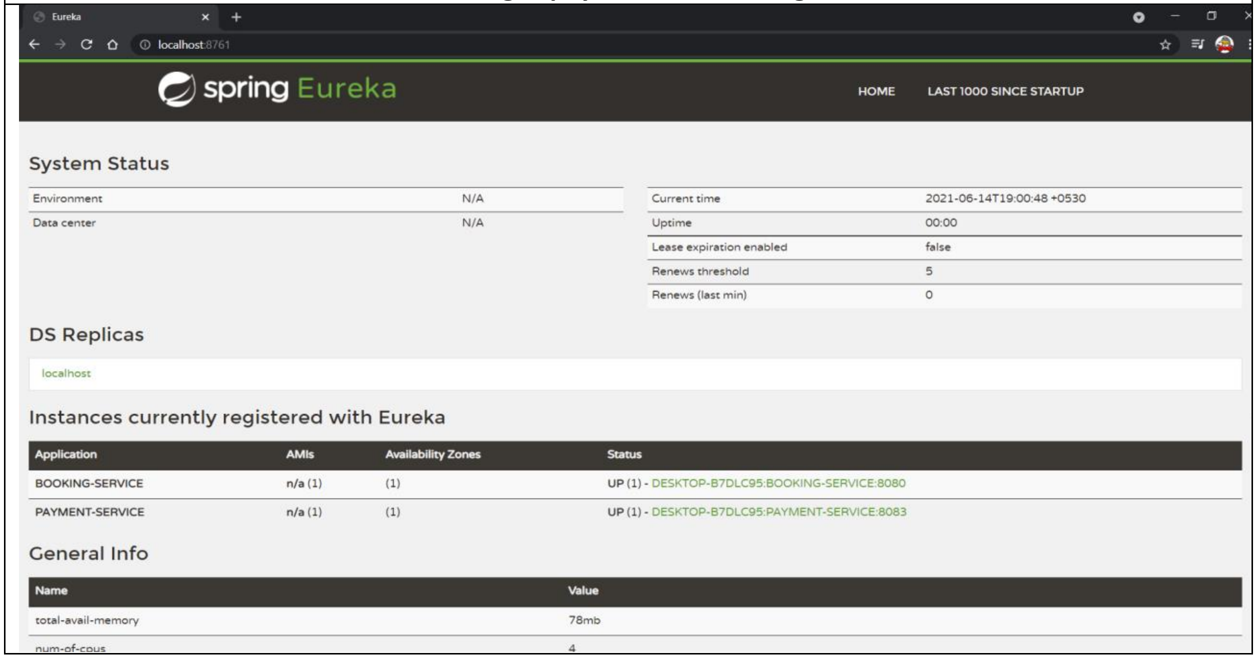
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

General Info

Name	Value
total-avail-memory	78mb
num-of-cpus	4
current-memory-usage	40mb (51%)

Eureka Server screenshot – with booking & payment services registered.



System Status

Environment	N/A
Data center	N/A

Current time	2021-06-14T19:00:48 +0530
Uptime	00:00
Lease expiration enabled	false
Renews threshold	5
Renews (last min)	0

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
BOOKING-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-B7DLC95:BOOKING-SERVICE:8080
PAYMENT-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-B7DLC95:PAYMENT-SERVICE:8083

General Info

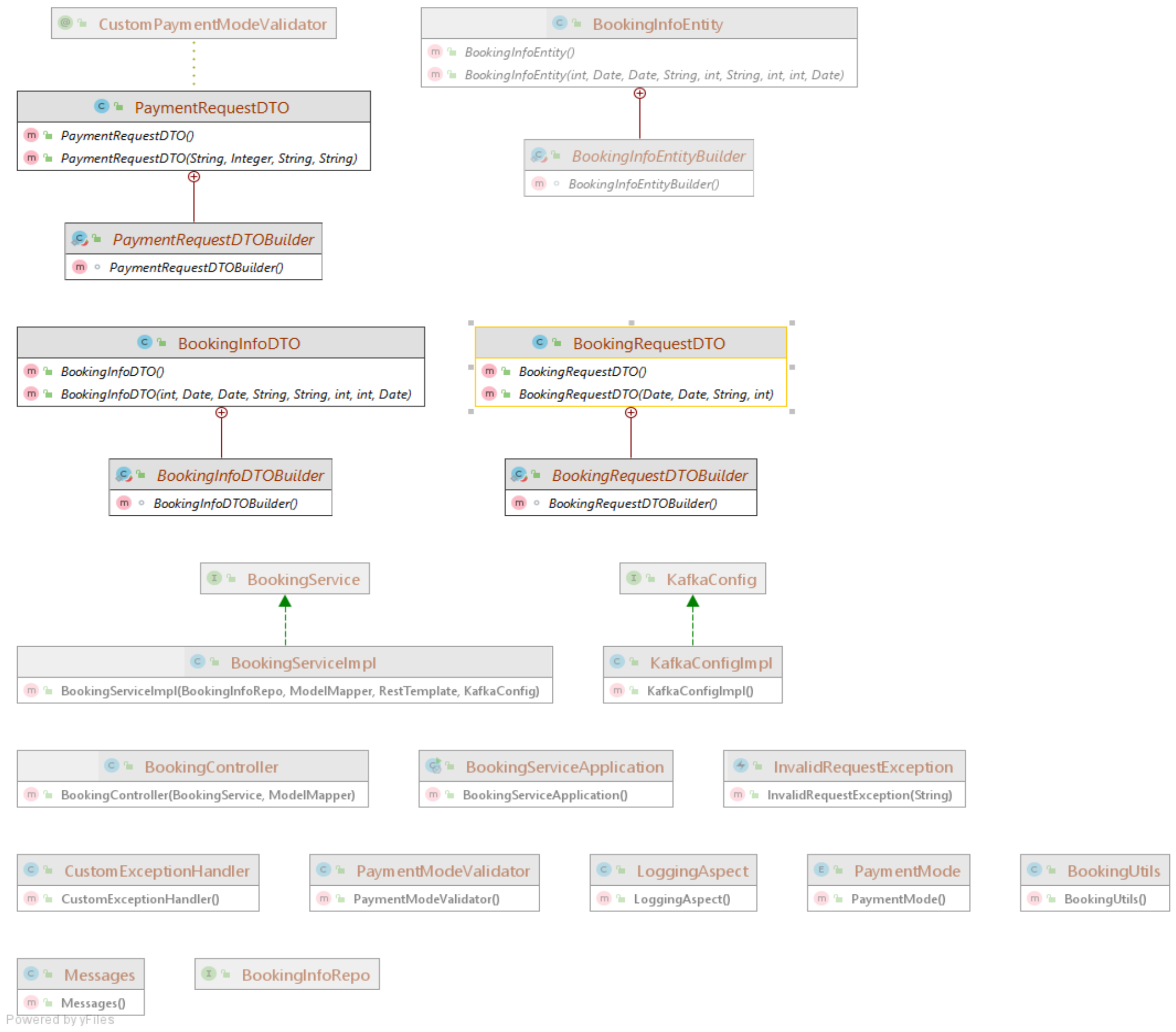
Name	Value
total-avail-memory	78mb
num-of-cpus	4

2. Booking Service (with backend RDS) – Implementation Details

Maven Spring Boot project configured with following package dependencies.

1. Spring Web – Core package with all necessary servers/services packed & configured to enable quicker & easier microservice development with less boilerplate code & infrastructure configs.
2. Eureka Client – To register with Eureka server to discover other services, here payment service
3. Spring JPA – Spring JPA for better integration with Java Persistent API, for CRUD operations, along with results paging & sorting support.
4. MySQL Driver – To enable application to communicate with MySQL server
5. Kafka Client – To register to Kafka EC2 server to send success message.
6. Lombok – To reduce boiler plate code by generating constructors, getters/setters, toString methods, as needed for the DTO & Entity pojo models.
7. ModelMapper – For convenient & easy DTO to Entity pojo mappings.
8. AOP – for custom Exception handling & logging (logging advice disabled/commented).

Class Diagram:



Class & Package Description:

Class Name	**Description, with major implementation considerations**
`BookingServiceApplication`	Annotated as @RestController & @SpringBootApplication Used **@LoadBalancer** for the **RestTemplate**, to enable discovering payment-service from the Eureka server running.
`BookingController`	Annotated as @RestController layer handles the following endpoint URIs URI 1: /booking Consumes `BookingRequestDTO` Produces `BookingInfoDTO` URI 2: /booking/{bookingId}/transaction Consumes `PaymentRequestDTO` Produces `BookingInfoDTO`
`BookingRequestDTO`	DTO containing user requested params checking for room booking availability. No Validations.
`BookingInfoDTO`	DTO containing Booking availability information sent to user based on the details requested by user through `BookingRequestDTO.` No Validations **Note**: `bookingId` - Field named as `id`, to match assignment naming convention requirement for the response

`PaymentRequestDTO`
`CustomPaymentModeValidator`
`PaymentModeValidator`
	Custom Class Constraint Validator
	`CustomPaymentModeValidator` is used with a custom Validator implementation in `PaymentModeValidator`
	This validates the payment request params as sent by user to Endpoint URI 2.
	`Validates following -`
	`1. Whether PaymentMode is any of payment modes, as supported in {@class PaymentMode}. (UPI or CARD)`
	`2. Checks if paymentMode is UPI then upiId is not null/Empty. Similarly, CardNumber should not be null/empty for CARD mode.`
	`Returns true, only if all above conditions satisfy; else false.`

`BookingInfoEntity`	Entity class that stores/tracks booking availability info as requested by user. The {@code transactionId} will be set to 0, by default. transactionId gets updated to valid transaction ID, once user initiates payment transaction (through endpoint "booking/{bookingId}/transaction") & makes a successful payment. `id` is (column name: `bookingId`)set as primary key, with AUTO generation strategy `roomPrice` is set nullable as false `transactionId` is set default to 0 **Note**: *It's been confirmed offline with Instructor that Schema referring to aadharNumber field to be unique is an error & hence haven't set unique constraint to true for this field.*
`BookingService`	Service Layer Implementation

`BookingServiceImpl`	Method `createBooking`: provides service implementation for URI 1. Uses Util Class implementations to generate random rom numbers within range of 0 to 100 & to calculate room price. Persists data to through Repository layer interface `BookingInfoRepo` Returns the `BookingInfoDTO` with booking availability details. Method `processBooking` provides service implementation for URI 2. Following checks/actions are performed here – 1. If No booking request ID is present in DB, then throw InvalidRequestException 2. Additionally, ensures if bookingId present in pathvariable matches with the bookingId present in RequestBody passed in. 3. Configure payment Request DTO to ensure that card is null for UPI mode & vice versa 4. Using Integer to avoid NPE while unboxing int warning, upon **RestTemplate.postForObject**() invocation. 5. Using RestTemplate for synchronous communication with Paymentservice, discovered through the Eureka Service Discovery 6. Publish/Produce Kafka Message, that will be brokered by the Kafka server running in EC2 instance and delivered to the NotificationService explained below.
`BookingInfoRepo`	Repository class. Using JpaRepository, to take advantage of both CRUD & results paging features inherited by JpaRepository, from CrudRepository & PagingAndSortingRepository

`CustomExceptionHandler`
`InvalidRequestException`
	Custom exception handler to handle error states & notify error in user readable form. * eg.,
	{
	"message": "Invalid mode of payment",
	"statusCode": 400
	}
	Not all the error messages are retrieved as it is expected from assignment requirement that the only one error message is displayed, and not as list of all error messages, like above.
	
	In addition, Exceptions triggered from custom Class Constraint Validator `CustomPaymentModeValidator` check failures are not retrievable through getFieldErrors(), while all the Global & Field errors i.e. Field errors, Class level validation errors are retrievable through getAllErrors().

`Messages`	Stores String Constants relevant to Error messages
`PaymentMode`	Enum class with Supported Payment modes (UPI / CARD)
`BookingUtils`	Utils class, to generate random room numbers, calculate no. of booking days & room price accordingly.

Properties configured:

Service running in port **8080** & named accordingly for Eureka discovery & AWS RDS DB connection

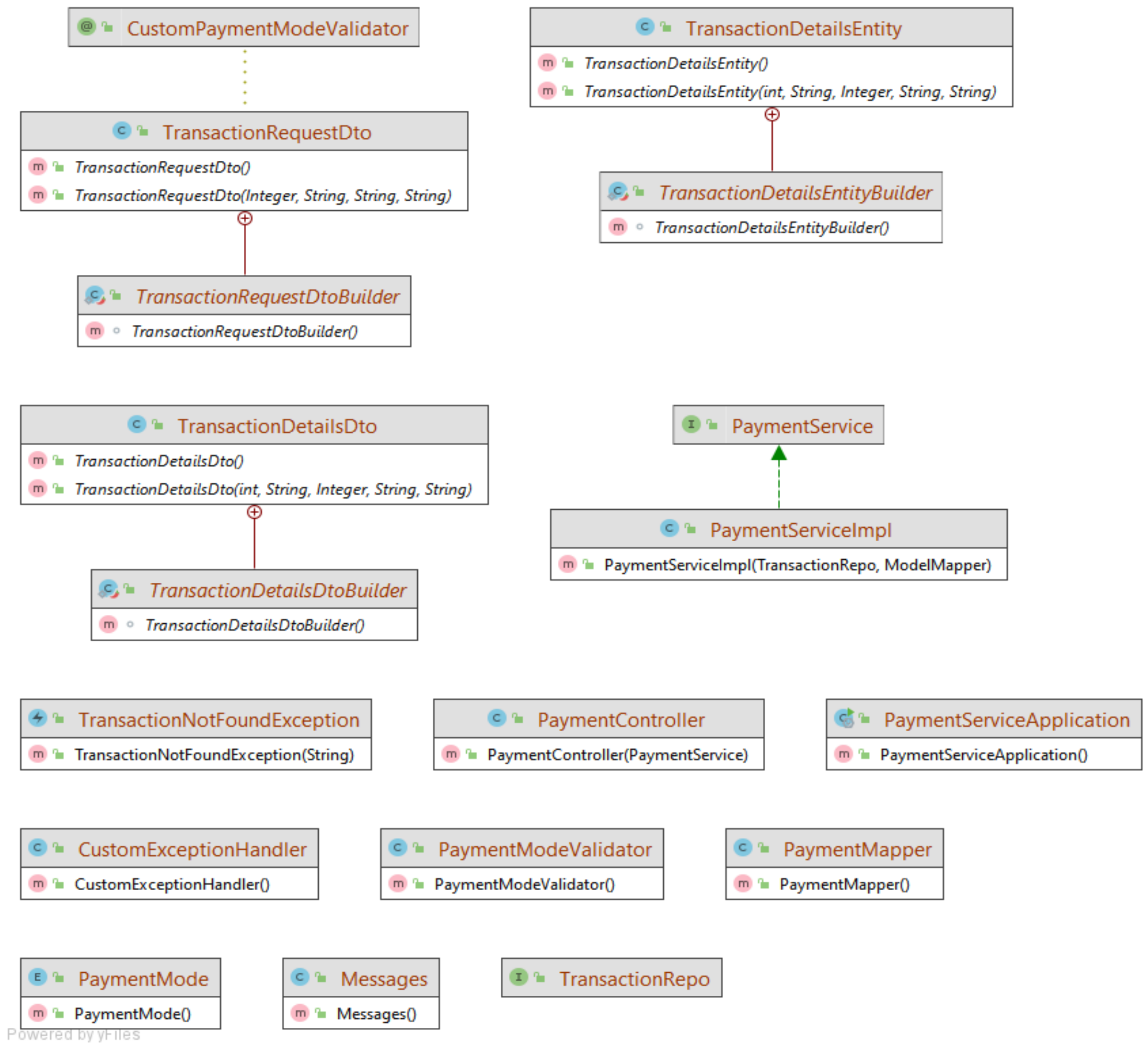
```
# Application Name & server port (also used by Eureka)
spring.application.name=BOOKING-SERVICE
server.port=8080
#
# RDS DB connection properties
spring.datasource.url=jdbc:mysql://bookings-db.cxjptry9wnmg.us-east-
1.rds.amazonaws.com/bookingsDB
spring.datasource.username=admin
spring.datasource.password=upgrad1234
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=create
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
#
##### URL Configuration #####
#Payments Service
paymentTransaction.url=http://PAYMENT-SERVICE/transaction
transactionDetails.url=http://PAYMENT-SERVICE/transaction/{transactionId}
#paymentTransaction.url=http://localhost:8083/transaction
#transactionDetails.url=http://localhost:8083/transaction/{transactionId}
```

3. Payment Service (with backend RDS DB) – Implementation Details

Maven Spring Boot project configured with following package dependencies.

1. Spring Web – Core package with all necessary servers/services packed & configured to enable quicker & easier microservice development with less boilerplate code & infrastructure configs.
2. Eureka Client – To register with Eureka server to discover other services, here payment service
3. Spring JPA – Spring JPA for better integration with Java Persistent API, for CRUD operations, along with results paging & sorting support.
4. MySQL Driver – To enable application to communicate with MySQL server
5. Lombok – To reduce boiler plate code by generating constructors, getters/setters, toString methods, as needed for the DTO & Entity pojo models.
6. ModelMapper – For convenient & easy DTO to Entity pojo mappings.
7. AOP – for custom Exception handling & logging (logging advice disabled/commented).

Class Diagram:


CustomPaymentModeValidator
TransactionDetailsEntity
TransactionDetailsEntity()
TransactionDetailsEntity(int, String, Integer, String, String)
TransactionRequestDto
TransactionRequestDto()
TransactionRequestDto(Integer, String, String, String)
TransactionDetailsEntityBuilder
TransactionDetailsEntityBuilder()
TransactionRequestDtoBuilder
TransactionRequestDtoBuilder()
TransactionDetailsDto
TransactionDetailsDto()
TransactionDetailsDto(int, String, Integer, String, String)
PaymentService
PaymentServiceImpl
PaymentServiceImpl(TransactionRepo, ModelMapper)
TransactionDetailsDtoBuilder
TransactionDetailsDtoBuilder()
TransactionNotFoundException
TransactionNotFoundException(String)
PaymentController
PaymentController(PaymentService)
PaymentServiceApplication
PaymentServiceApplication()
CustomExceptionHandler
CustomExceptionHandler()
PaymentModeValidator
PaymentModeValidator()
PaymentMapper
PaymentMapper()
PaymentMode
PaymentMode()
Messages
Messages()
TransactionRepo
Powered by yFiles

Class & Package Description:

Class Name	Description, with major implementation considerations
`PaymentServiceApplication`	Annotated as @RestController & @SpringBootApplication
`PaymentController`	Annotated as @RestController layer handles the following endpoint URIs URI 1: /transaction Consumes `TransactionRequestDto` Produces `Integer (transactionId)` URI 2: /transaction/{transactionId} Consumes `int transactionID` Produces `TransactionDetailsDto`

`TransactionRequestDto`	Custom Class Constraint Validator
	`CustomPaymentModeValidator` is used with a custom Validator implementation in `PaymentModeValidator`
	This validates the payment request params as sent by user to Endpoint URI 2.
	`Validates following -`
	` 1. Whether PaymentMode is any of payment modes, as supported in {@class PaymentMode}. (UPI or CARD)`
	` 2. Checks if PaymentMode is UPI then UpiID is not Empty. Similarly, CardNumber should not be empty for CARD mode.`
	Returns true, only if all above conditions satisfy; else false.
	
	Note: *Ensures that card details are stored repo in lower case as per assignment expectations in schema pdf*

`TransactionDetailsDto`	DTO containing transaction details to be sent as response for URI 2 **Note**: *Ensure that response object matches the expected upper case payment mode usage as per assignment expectations.*

`TransactionDetailsEntity`	Entity class that stores/tracks booking availability info as requested by user.
	`transactionId is Primary key with AUTO generation strategy`
	`bookingId is nullable set false`
	
	Note: *Ensures that card details are stored repo in lower case as per assignment expectations in schema pdf.*

`PaymentService` `PaymentServiceImpl`	Service Layer Implementation Method `makePayment` provides service implementation for URI 1. Handles make payment request from booking service and returns transaction ID. ensure that card is null for UPI mode, similarly upiId to null for CARD mode. Persists data to through Repository layer interface `TransactionRepo`

	Returns the `transactionID`. Method `fetchTransaction` provides service implementation for URI 2. Handles request to fetch transaction details of a particular transaction .
`TransactionRepo`	Repository class. Using JpaRepository, to take advantage of both CRUD & results paging features inherited by JpaRepository, from CrudRepository & PagingAndSortingRepository

`CustomExceptionHandler`
`InvalidRequestException`
	Custom exception handler to handle error states & notify error in user readable form. * eg.,
	{
	"message": "Invalid mode of payment",
	"statusCode": 400
	}
	Not all the error messages are retrieved as it is expected from assignment requirement that the only one error message is displayed, and not as list of all error messages, like above.
	
	In addition, Exceptions triggered from custom Class Constraint Validator CustomPaymentModeValidator check failures are not retrievable through getFieldErrors(), while all the Global & Field errors i.e. Field errors, Class level validation errors are retrievable through getAllErrors().

`Messages`	Stores String Constants relevant to Error messages
`PaymentMode`	Enum class with Supported Payment modes (UPI / CARD)
`PaymentMapper`	Custom mapper util to map to convert `TransactionRequestDto` to `TransactionDetailsEntity`

Properties configured:

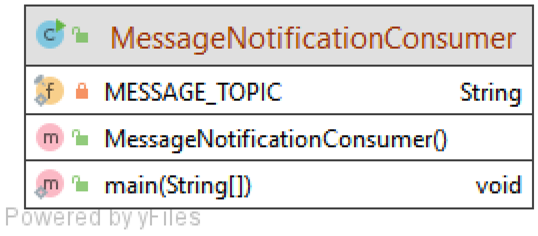
Service running in port **8083** & named accordingly for Eureka discovery & AWS RDS DB connection

```
# Application Name & server port (also used by Eureka)
spring.application.name=PAYMENT-SERVICE
server.port=8083
#
# RDS DB connection properties
spring.datasource.url=jdbc:mysql://payments-db.cxjptry9wnmg.us-east-
1.rds.amazonaws.com/paymentsDB
spring.datasource.username=admin
spring.datasource.password=upgrad1234
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=create
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

4. Notification Service (with Kafka broker configured in EC2) – Implementation Details

Standalone Javen Maven application configured with *kafka-clients* package dependency, that listens to kafka broker running in EC2 instace configured with a Elastic IP.

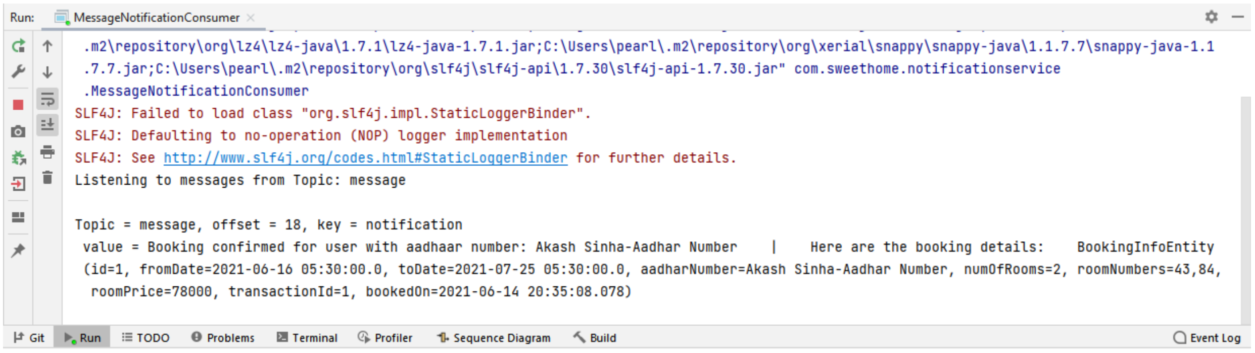
Class Diagram:



Class & Package Description:

`MessageNotificationConsumer:main()`	Configures kafka properties, bootstrap.servers, group.id, enable.auto.commit, commit intervals & key/value deserializers. Fetches the list of topics, here, topic name is “message” & keeps consumer polling for messages & closes consumer only upon program termination.

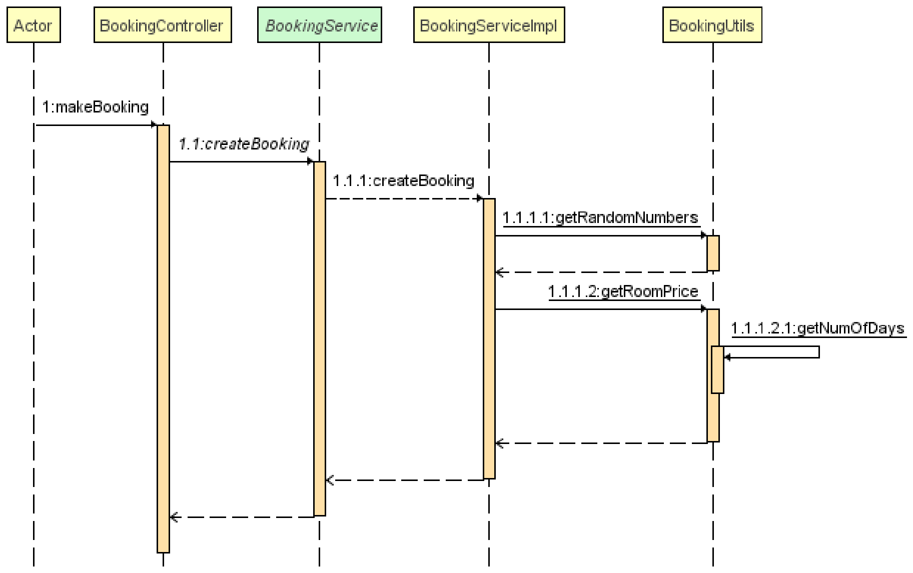
NotificationService Consumer screenshot (for given PostMan API documentation test):



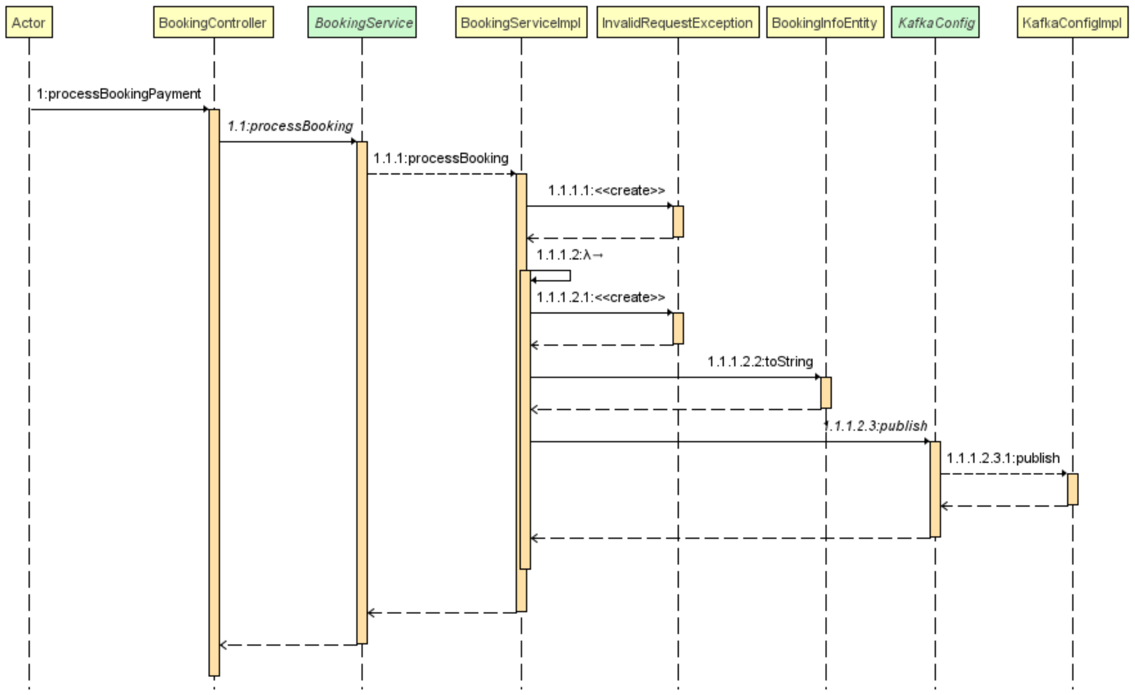
```
 .m2\repository\org\lz4\lz4-java\1.7.1\lz4-java-1.7.1.jar;C:\Users\pearl\.m2\repository\org\xerial\snappy\snappy-java\1.1.7.7\snappy-java-1.1
 .7.7.jar;C:\Users\pearl\.m2\repository\org\slf4j\slf4j-api\1.7.30\slf4j-api-1.7.30.jar" com.sweethome.notificationservice
 .MessageNotificationConsumer
SLF4J: 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.
Listening to messages from Topic: message

Topic = message, offset = 18, key = notification
 value = Booking confirmed for user with aadhaar number: Akash Sinha-Aadhar Number    |    Here are the booking details:    BookingInfoEntity
 (id=1, fromDate=2021-06-16 05:30:00.0, toDate=2021-07-25 05:30:00.0, aadharNumber=Akash Sinha-Aadhar Number, numOfRooms=2, roomNumbers=43,84,
  roomPrice=78000, transactionId=1, bookedOn=2021-06-14 20:35:08.078)
```

Sequence Diagrams:

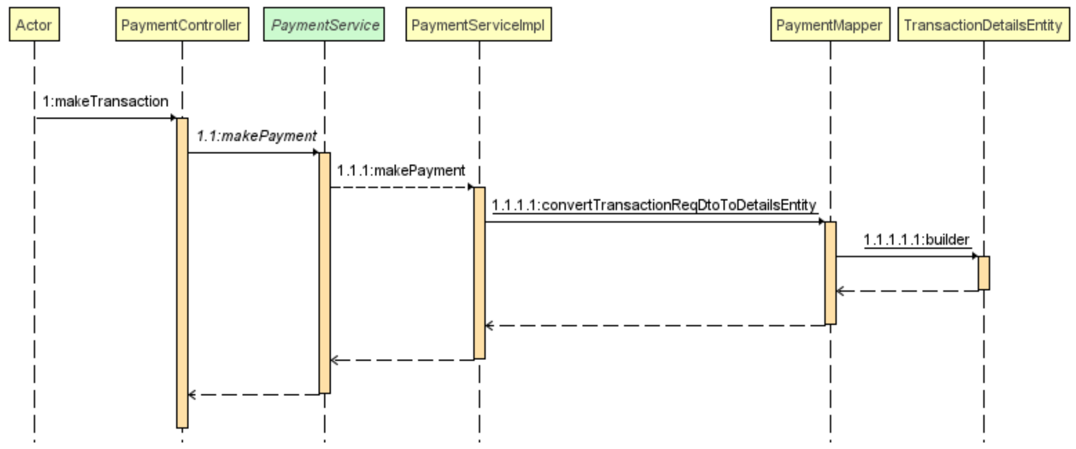
1. Booking Service: Endpoint 1: URI: *"/booking"*



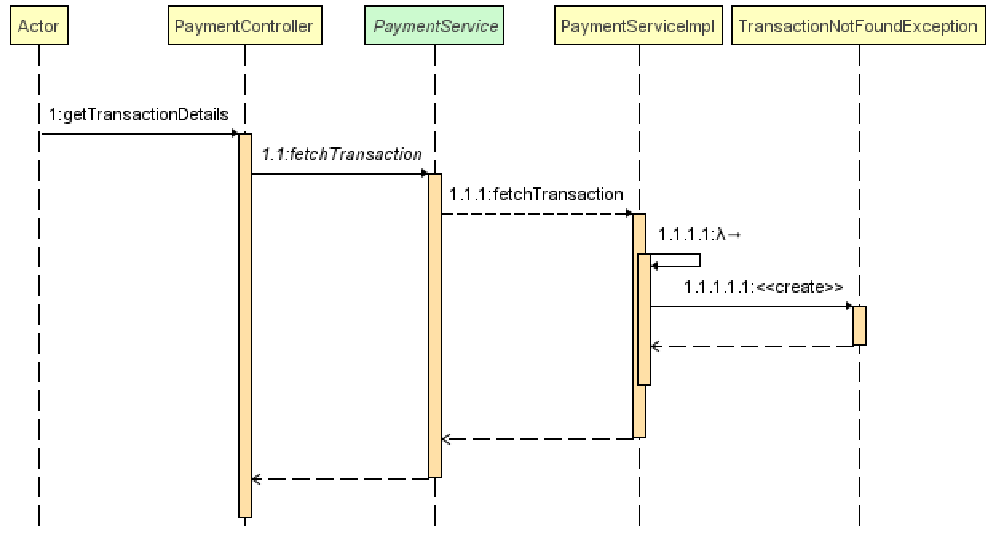
2. Booking Service: Endpoint 2: URI: *"booking/{bookingId}/transaction"*



3. Payment Service: Endpoint 1: URI: “*/transaction*”

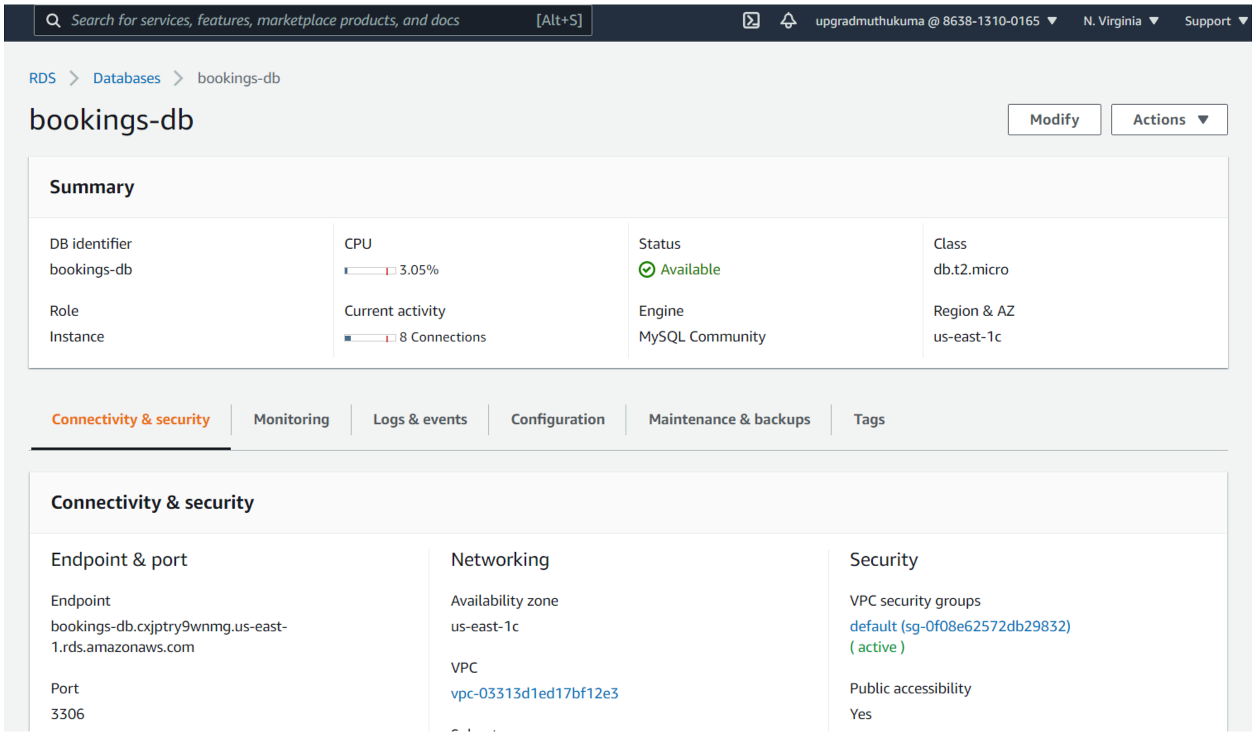


4. Payment Service: Endpoint 1: URI: “*/transaction/{transactionId}*”



ScreenShot References:

Booking Service - AWS RDS DB with security group config screenshot



Search for services, features, marketplace products, and docs [Alt+S] upgradmuthukuma @ 8638-1310-0165 N. Virginia Support

RDS > Databases > bookings-db

bookings-db

Modify Actions

Summary

DB identifier	CPU	Status	Class
bookings-db	3.05%	Available	db.t2.micro
Role	Current activity	Engine	Region & AZ
Instance	8 Connections	MySQL Community	us-east-1c

Connectivity & security | Monitoring | Logs & events | Configuration | Maintenance & backups | Tags

Connectivity & security

Endpoint & port

Endpoint
bookings-db.cxjptry9wnmg.us-east-1.rds.amazonaws.com

Port
3306

Networking

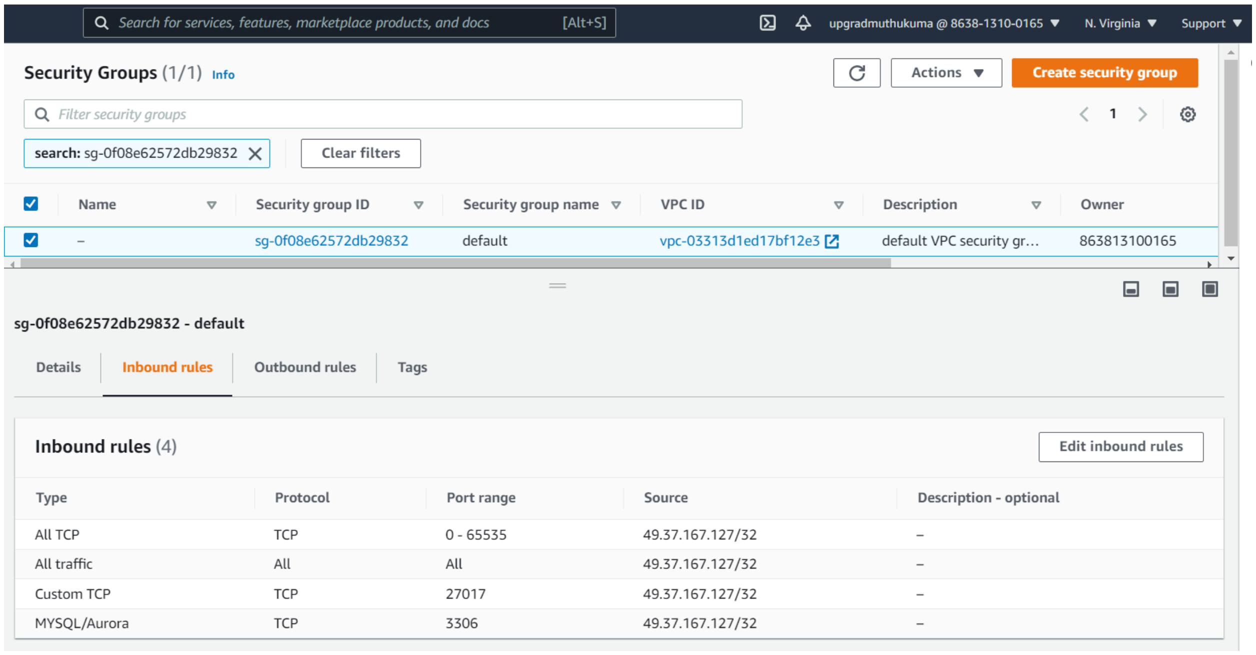
Availability zone
us-east-1c

VPC
vpc-03313d1ed17bf12e3

Security

VPC security groups
default (sg-0f08e62572db29832)
(active)

Public accessibility
Yes



Search for services, features, marketplace products, and docs [Alt+S] upgradmuthukuma @ 8638-1310-0165 N. Virginia Support

Security Groups (1/1) Info

Actions Create security group

Filter security groups

search: sg-0f08e62572db29832 Clear filters

Name	Security group ID	Security group name	VPC ID	Description	Owner
–	sg-0f08e62572db29832	default	vpc-03313d1ed17bf12e3	default VPC security gr...	863813100165

sg-0f08e62572db29832 - default

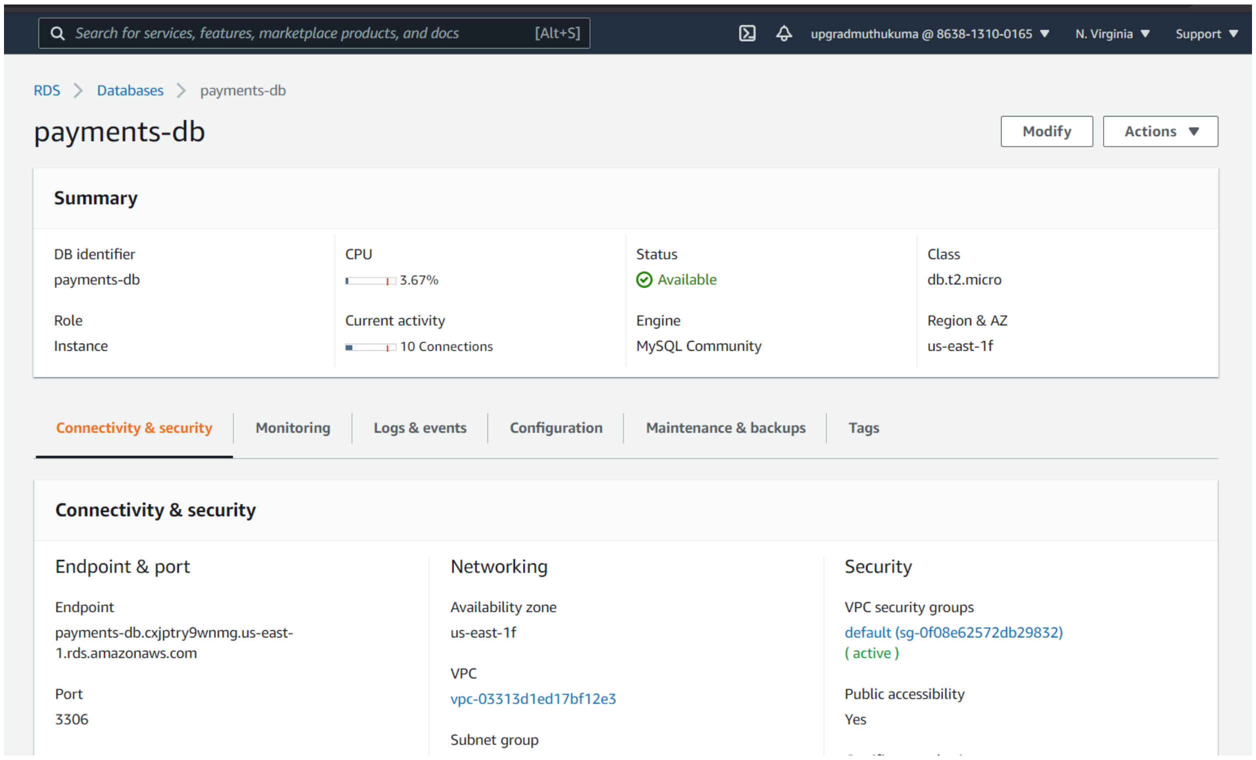
Details | Inbound rules | Outbound rules | Tags

Inbound rules (4)

Edit inbound rules

Type	Protocol	Port range	Source	Description - optional
All TCP	TCP	0 - 65535	49.37.167.127/32	–
All traffic	All	All	49.37.167.127/32	–
Custom TCP	TCP	27017	49.37.167.127/32	–
MYSQL/Aurora	TCP	3306	49.37.167.127/32	–

Payments Service - AWS RDS DB with security group config screenshot



Search for services, features, marketplace products, and docs [Alt+S]

upgradmuthukuma @ 8638-1310-0165 ▼ N. Virginia ▼ Support ▼

RDS > Databases > payments-db

payments-db

Modify | Actions ▼

Summary

DB identifier	CPU	Status	Class
payments-db	3.67%	Available	db.t2.micro
Role	Current activity	Engine	Region & AZ
Instance	10 Connections	MySQL Community	us-east-1f

Connectivity & security | Monitoring | Logs & events | Configuration | Maintenance & backups | Tags

Connectivity & security

Endpoint & port

Endpoint
payments-db.cxjptry9wnmg.us-east-1.rds.amazonaws.com

Port
3306

Networking

Availability zone
us-east-1f

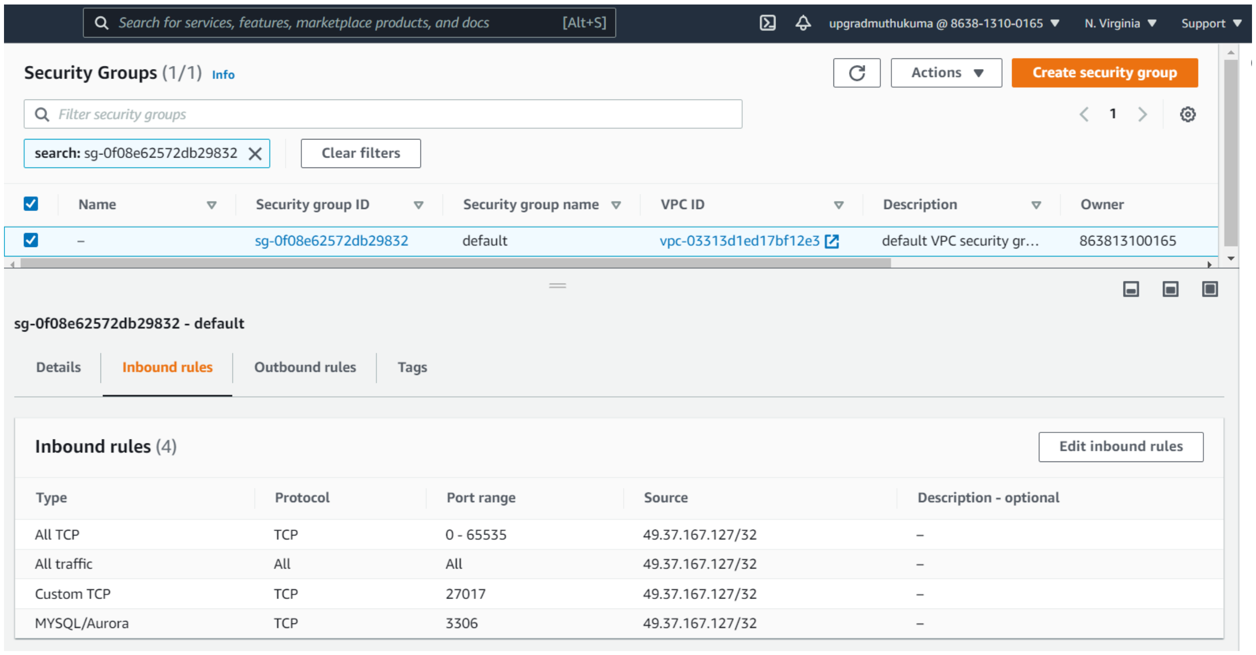
VPC
vpc-03313d1ed17bf12e3

Subnet group

Security

VPC security groups
default (sg-0f08e62572db29832)
(active)

Public accessibility
Yes



Search for services, features, marketplace products, and docs [Alt+S]

upgradmuthukuma @ 8638-1310-0165 ▼ N. Virginia ▼ Support ▼

Security Groups (1/1) Info

Actions ▼ | Create security group

Filter security groups

search: sg-0f08e62572db29832 ✕ | Clear filters

✓	Name	Security group ID	Security group name	VPC ID	Description	Owner
✓	–	sg-0f08e62572db29832	default	vpc-03313d1ed17bf12e3	default VPC security gr...	863813100165

sg-0f08e62572db29832 - default

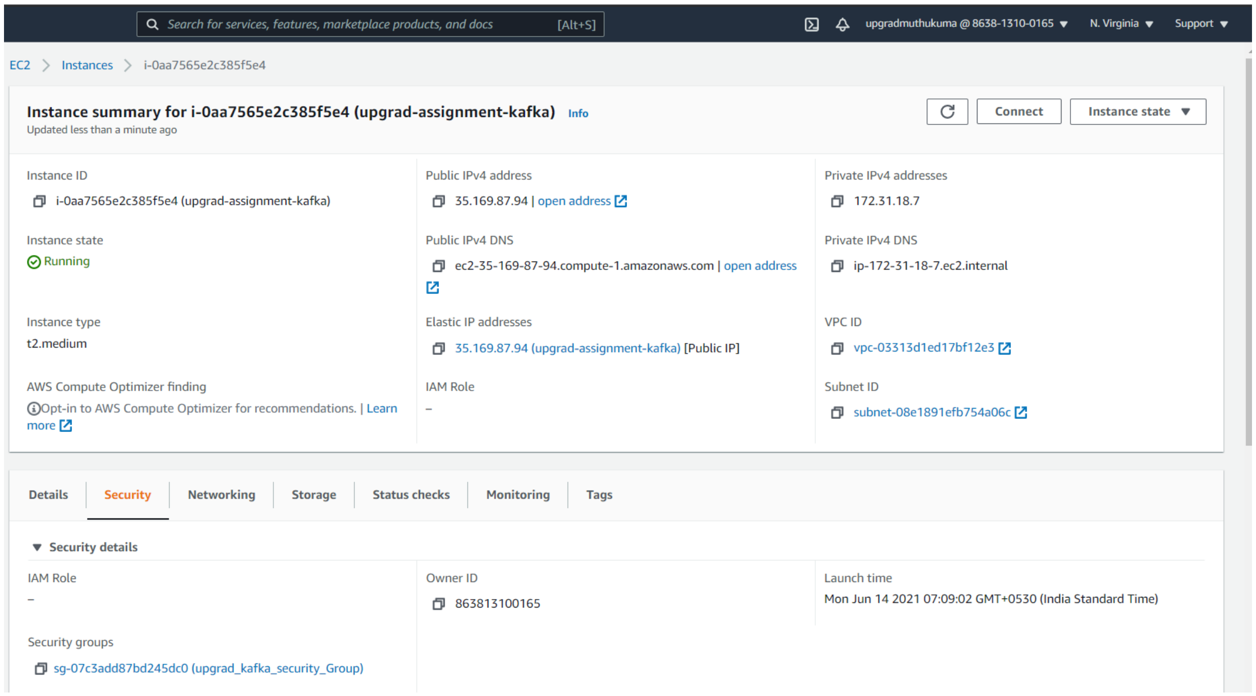
Details | Inbound rules | Outbound rules | Tags

Inbound rules (4)

Edit inbound rules

Type	Protocol	Port range	Source	Description - optional
All TCP	TCP	0 - 65535	49.37.167.127/32	–
All traffic	All	All	49.37.167.127/32	–
Custom TCP	TCP	27017	49.37.167.127/32	–
MYSQL/Aurora	TCP	3306	49.37.167.127/32	–

Kafka AWS configured on EC2, with Elastic IP: Security Group & console screenshots:



Search for services, features, marketplace products, and docs [Alt+S] upgradmuthukuma @ 8638-1310-0165 ▼ N. Virginia ▼ Support ▼

EC2 > Instances > i-0aa7565e2c385f5e4

Instance summary for i-0aa7565e2c385f5e4 (upgrad-assignment-kafka) Info
Updated less than a minute ago

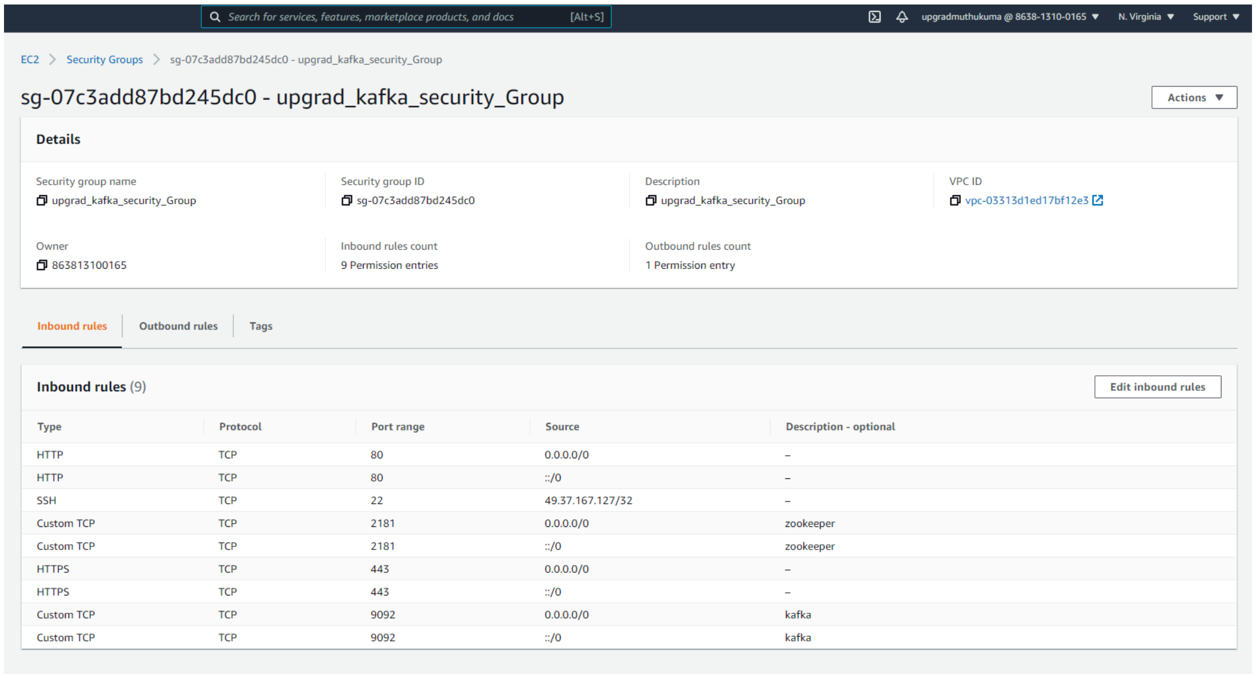
Connect | Instance state ▼

Instance ID: i-0aa7565e2c385f5e4 (upgrad-assignment-kafka)	Public IPv4 address: 35.169.87.94 | open address	Private IPv4 addresses: 172.31.18.7
Instance state: Running	Public IPv4 DNS: ec2-35-169-87-94.compute-1.amazonaws.com | open address	Private IPv4 DNS: ip-172-31-18-7.ec2.internal
Instance type: t2.medium	Elastic IP addresses: 35.169.87.94 (upgrad-assignment-kafka) [Public IP]	VPC ID: vpc-03313d1ed17bf12e3
AWS Compute Optimizer finding: Opt-in to AWS Compute Optimizer for recommendations. | Learn more	IAM Role: –	Subnet ID: subnet-08e1891efb754a06c

Details | Security | Networking | Storage | Status checks | Monitoring | Tags

▼ Security details

IAM Role: –	Owner ID: 863813100165	Launch time: Mon Jun 14 2021 07:09:02 GMT+0530 (India Standard Time)
Security groups: sg-07c3add87bd245dc0 (upgrad_kafka_security_Group)		



Search for services, features, marketplace products, and docs [Alt+S] upgradmuthukuma @ 8638-1310-0165 ▼ N. Virginia ▼ Support ▼

EC2 > Security Groups > sg-07c3add87bd245dc0 - upgrad_kafka_security_Group

sg-07c3add87bd245dc0 - upgrad_kafka_security_Group

Actions ▼

Details

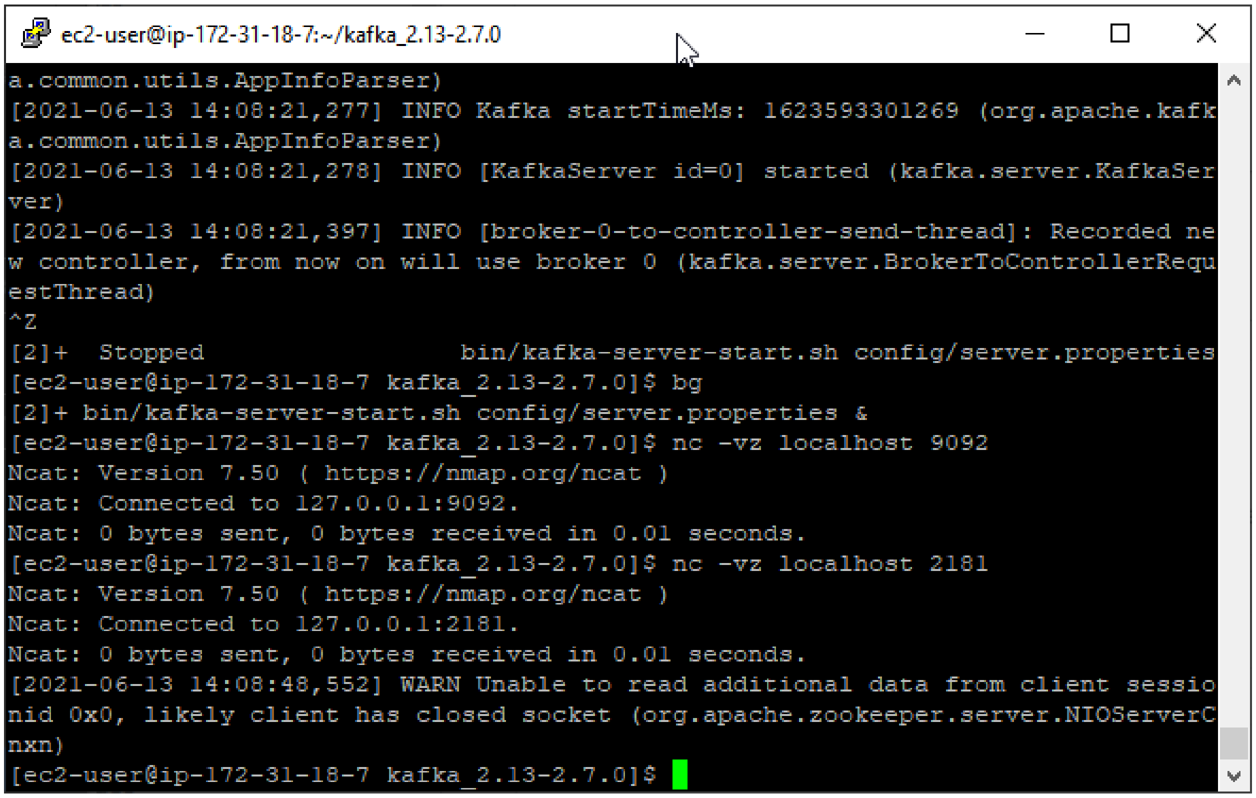
Security group name: upgrad_kafka_security_Group	Security group ID: sg-07c3add87bd245dc0	Description: upgrad_kafka_security_Group	VPC ID: vpc-03313d1ed17bf12e3
Owner: 863813100165	Inbound rules count: 9 Permission entries	Outbound rules count: 1 Permission entry	

Inbound rules | Outbound rules | Tags

Inbound rules (9) Edit inbound rules

Type	Protocol	Port range	Source	Description - optional
HTTP	TCP	80	0.0.0.0/0	–
HTTP	TCP	80	::/0	–
SSH	TCP	22	49.37.167.127/32	–
Custom TCP	TCP	2181	0.0.0.0/0	zookeeper
Custom TCP	TCP	2181	::/0	zookeeper
HTTPS	TCP	443	0.0.0.0/0	–
HTTPS	TCP	443	::/0	–
Custom TCP	TCP	9092	0.0.0.0/0	kafka
Custom TCP	TCP	9092	::/0	kafka

Zookeeper & Kafka running as background services.



ec2-user@ip-172-31-18-7:~/kafka_2.13-2.7.0

```
a.common.utils.AppInfoParser)
[2021-06-13 14:08:21,277] INFO Kafka startTimeMs: 1623593301269 (org.apache.kafk
a.common.utils.AppInfoParser)
[2021-06-13 14:08:21,278] INFO [KafkaServer id=0] started (kafka.server.KafkaSer
ver)
[2021-06-13 14:08:21,397] INFO [broker-0-to-controller-send-thread]: Recorded ne
w controller, from now on will use broker 0 (kafka.server.BrokerToControllerRequ
estThread)
^Z
[2]+  Stopped                 bin/kafka-server-start.sh config/server.properties
[ec2-user@ip-172-31-18-7 kafka_2.13-2.7.0]$ bg
[2]+ bin/kafka-server-start.sh config/server.properties &
[ec2-user@ip-172-31-18-7 kafka_2.13-2.7.0]$ nc -vz localhost 9092
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Connected to 127.0.0.1:9092.
Ncat: 0 bytes sent, 0 bytes received in 0.01 seconds.
[ec2-user@ip-172-31-18-7 kafka_2.13-2.7.0]$ nc -vz localhost 2181
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Connected to 127.0.0.1:2181.
Ncat: 0 bytes sent, 0 bytes received in 0.01 seconds.
[2021-06-13 14:08:48,552] WARN Unable to read additional data from client sessio
nid 0x0, likely client has closed socket (org.apache.zookeeper.server.NIOServerC
nxn)
[ec2-user@ip-172-31-18-7 kafka_2.13-2.7.0]$
```

<<End of Document>>