



Week 7 - Project Report

By : Kanchan Rai

Topic : Spring Security & Dockerizing Flight Booking Microservices

The following document contains the description and explanation of the project **Spring Security & Dockerizing Flight Booking Microservices**. This project has all the required requirements of the Week 6 project. As well as the spring security and dockerization of each microservice and running them as containers.

INDEX

1. Project Over View

- 1.1 System Architecture
- 1.2 Dockerization and Containerization

2. JCoco Coverage Report

3. SonarQube Report

4. JMeter Load Testing

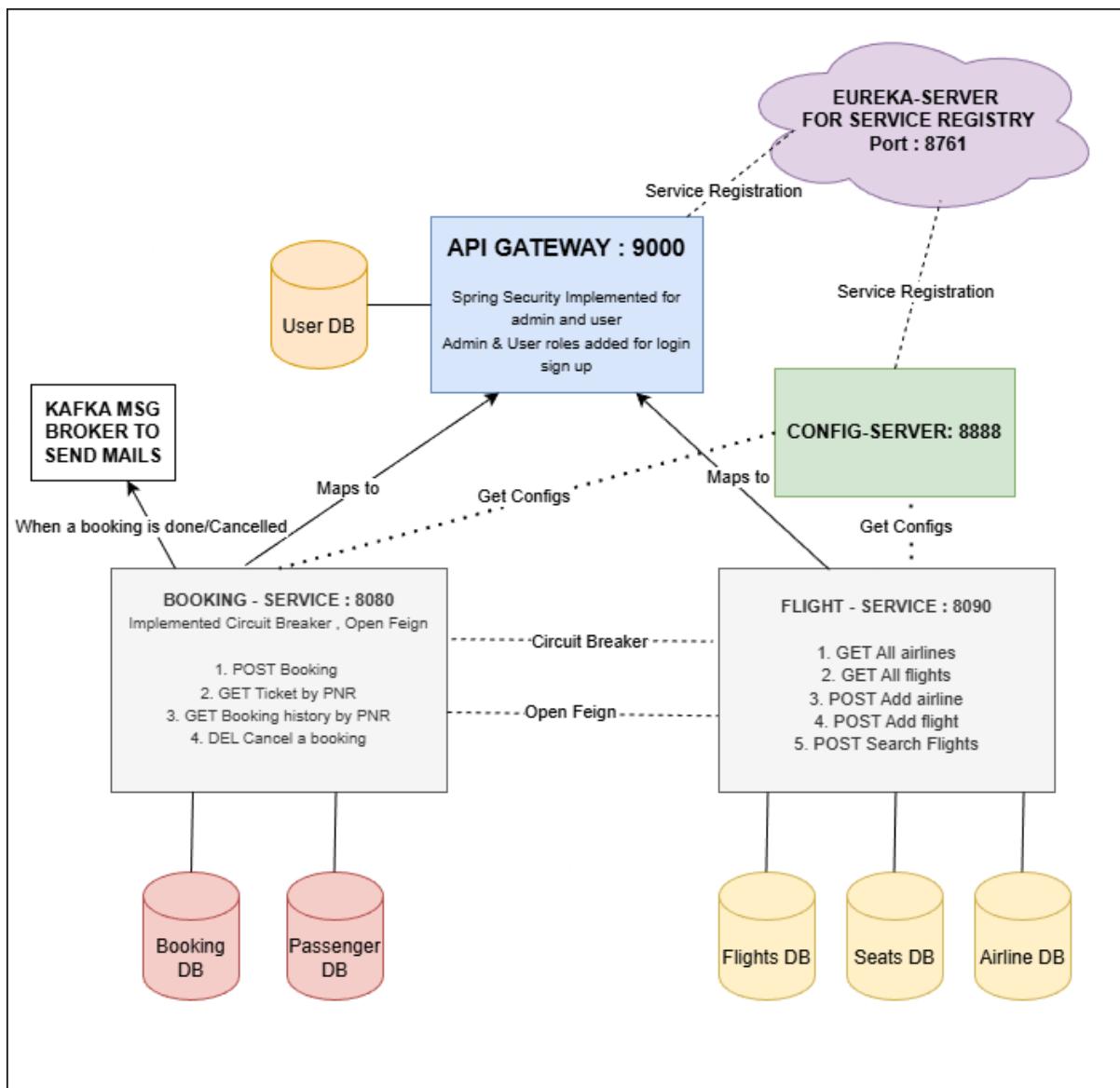
- 4.1 With 20 Threads
- 4.2 With 50 Threads
- 4.3 With 100 Threads

5. All API Endpoints Testing & Results

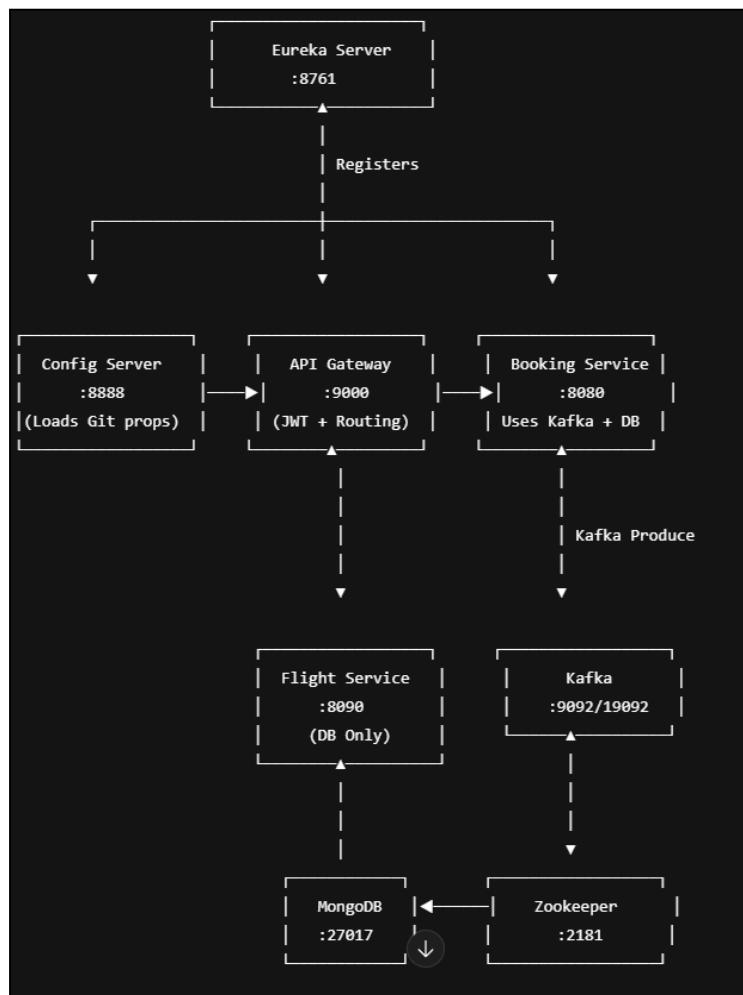
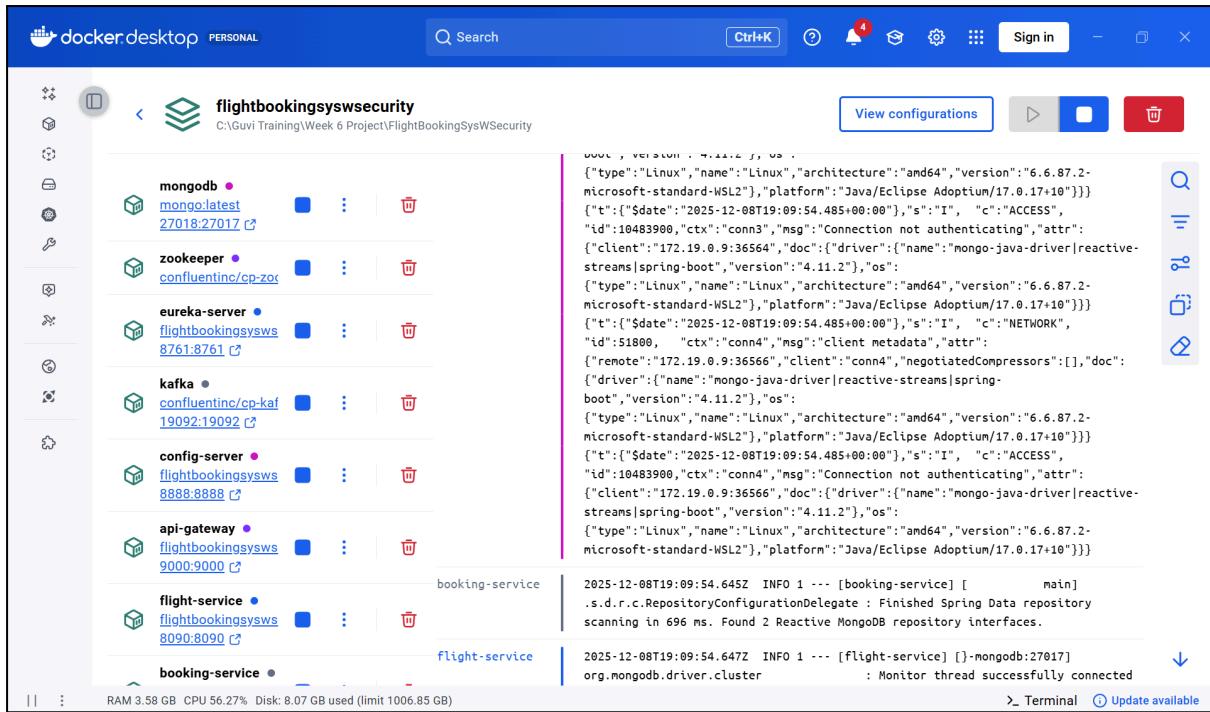
1. Project Overview

This project was a monolithic application with one single db called the “flightBookingSystem” in MongoDB, based on the reactive web flux in spring boot. The project has now been converted to a microservices architecture based project. There are two microservices namely “booking-service” and “flight-service”. These services are connected by a common api-gateway and a configuration server. In between these services proper load balancing has been applied along with the circuit breaker in between to ensure that all services stay up and running. These services have been registered on the netflix eureka server. Along with this apache-kafka message broker technique has also been applied to send notifications whenever a flight is booked or cancelled. Proper load testing using Apache JMeter and JUnit testing has been done.

1.1. System Architecture



1.2 Dockerization Architecture Diagram



1.3 Eureka Registering

The screenshot shows the Spring Eureka dashboard with the following sections:

- System Status:** Displays environment (test), data center (default), current time (2025-12-01T22:49:44 +0530), uptime (00:05), lease expiration enabled (false), renew threshold (8), and renew (last min) (8).
- DS Replicas:** Shows instances registered with Eureka under the host [localhost](#).

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - 192.168.1.6:api-gateway:9000
BOOKING-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.6:booking-service:8080
CONFIG-SERVER	n/a (1)	(1)	UP (1) - 192.168.1.6:config-server:8888
FLIGHT-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.6:flight-service:8090
- General Info:** Displays system properties.

Name	Value
total-avail-memory	90mb
num-of-cpus	12

This is the Eureka Server where all four services are successfully registered, namely flight-service, booking-service, config-server, and api-gateway.

This project also implements Message Broker using Apache Kafka which is used whenever a flight is booked or cancelled in the booking service and a mail is sent to the passenger.

2. JCoco Coverage Report -

Coverage Report for FlightService - 96%

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.flightservice.service		96%		83%	10	46	2	100	1	19	0	3
com.flightservice.controller		70%		n/a	2	7	2	7	2	7	0	1
com.flightservice		37%		n/a	1	2	2	3	1	2	0	1
com.flightservice.request		100%		n/a	0	39	0	57	0	39	0	3
com.flightservice.exceptions		100%		90%	1	14	0	30	0	9	0	3
com.flightservice.model		100%		n/a	0	2	0	18	0	2	0	2
Total	32 of 867	96%	10 of 64	84%	14	110	6	215	4	78	0	13

Coverage Report for BookingService - 92%

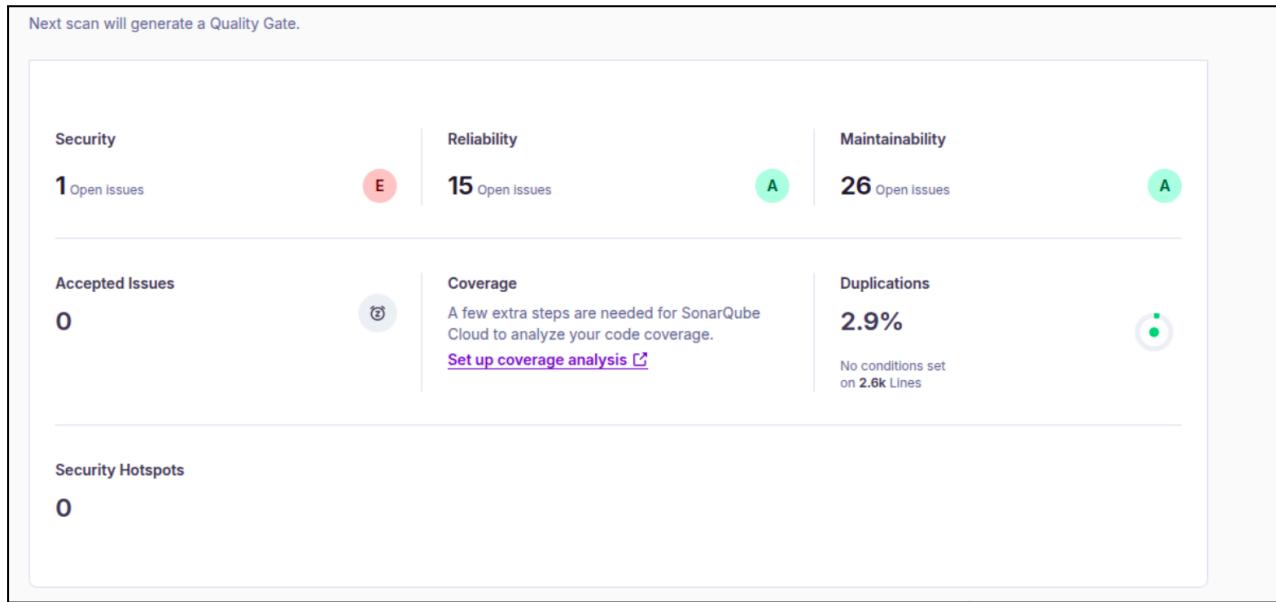
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.bookingservice.service		91%		82%	16	67	10	184	6	38	0	4
com.bookingservice.model		86%		n/a	10	30	10	53	10	30	0	6
com.bookingservice.requests		84%		n/a	4	24	5	35	4	24	0	2
com.bookingservice		37%		n/a	1	2	2	3	1	2	0	1
com.bookingservice.exceptions		100%		92%	1	16	0	42	0	9	0	3
com.bookingservice.client		100%		n/a	0	15	0	42	0	15	0	3
com.bookingservice.controller		100%		n/a	0	5	0	5	0	5	0	1
Total	110 of 1,384	92%	11 of 72	84%	32	159	27	364	21	123	0	20

Coverage Report for Api Gateway -

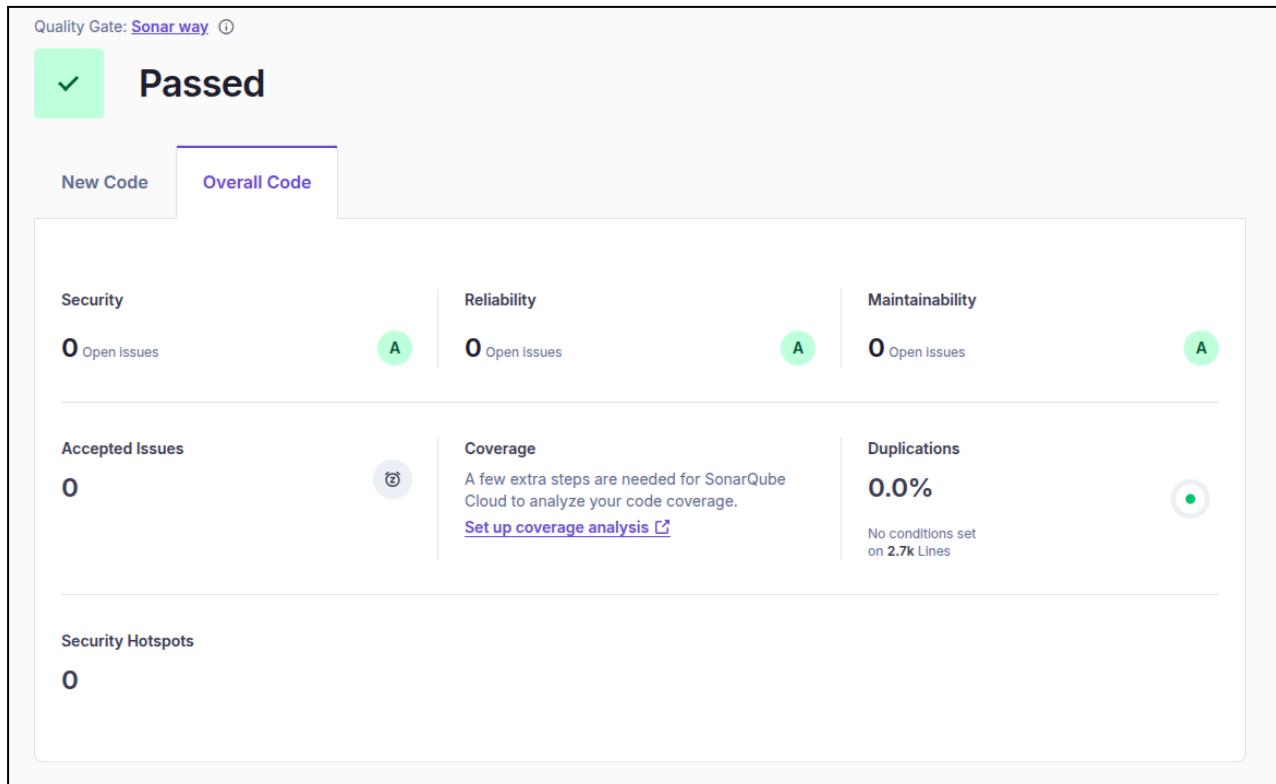
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.apigateway.controller		12%		0%	4	5	9	11	3	4	0	1
com.apigateway.model		25%		n/a	13	14	19	22	13	14	1	2
com.apigateway.security		93%		61%	10	25	6	72	0	12	0	3
com.apigateway		37%		n/a	1	2	2	3	1	2	0	1
Total	127 of 451	71%	12 of 28	57%	28	46	36	108	17	32	1	7

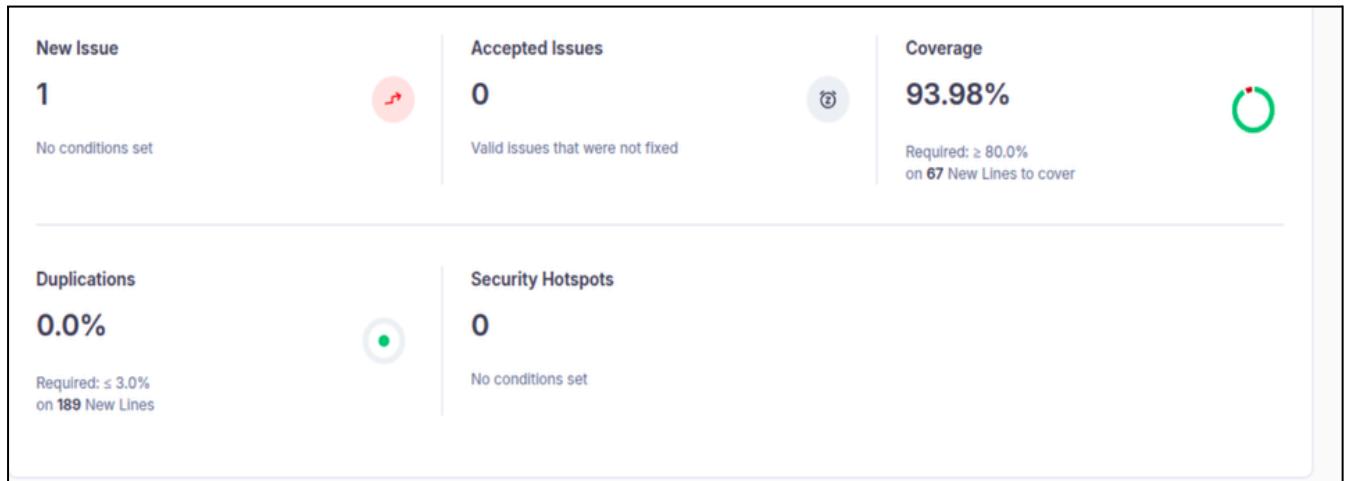
3. SonarQube Report

1. Before Fixing



Before fixing the project had 1 security issue, 15 reliability, 26 open issues and 2.9% code duplication. After fixing everything was reduced to 0.





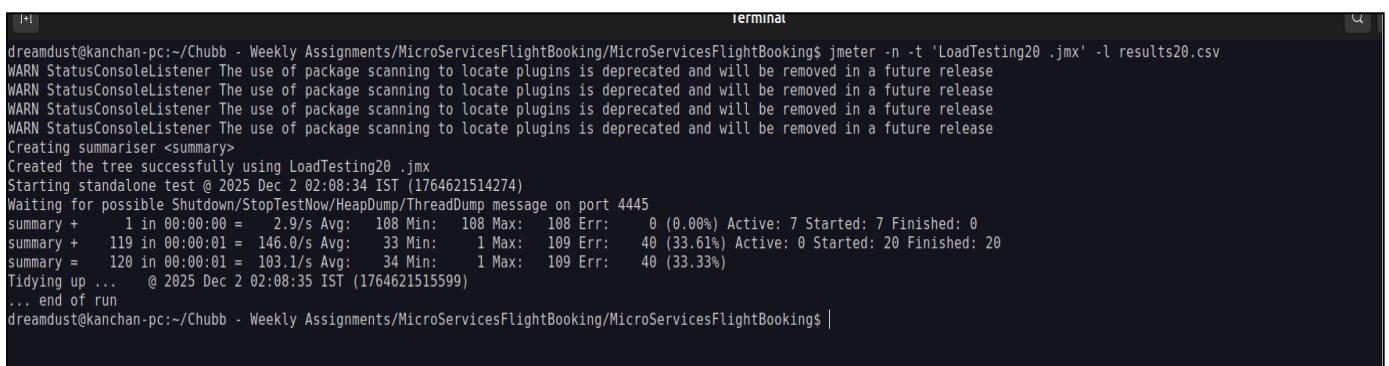
4. JMeter Load Testing

4.1 With 20 Threads

The JMeter testing for 20 threads across the get,post, and delete requests have been carried out successfully through CLI as well as GUI.

Command used and result stored in a csv file -

```
jmeter -n -t 'LoadTesting20.jmx' -l result20.csv
```



```
dreamdust@kanchan-pc:~/Chubb - Weekly Assignments/MicroServicesFlightBooking/MicroServicesFlightBooking$ jmeter -n -t 'LoadTesting20.jmx' -l results20.csv
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
Creating summariser <summary>
Created the tree successfully using LoadTesting20.jmx
Starting standalone test @ 2025 Dec 2 02:08:34 IST (1764621514274)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
summary + 1 in 00:00:00 = 2.9/s Avg: 108 Min: 108 Max: 108 Err: 0 (0.00%) Active: 7 Started: 7 Finished: 20
summary + 119 in 00:00:01 = 146.0/s Avg: 33 Min: 1 Max: 109 Err: 40 (33.61%) Active: 0 Started: 20 Finished: 20
summary = 120 in 00:00:01 = 103.1/s Avg: 34 Min: 1 Max: 109 Err: 40 (33.33%)
Tidying up ... @ 2025 Dec 2 02:08:35 IST (1764621515599)
... end of run
dreamdust@kanchan-pc:~/Chubb - Weekly Assignments/MicroServicesFlightBooking/MicroServicesFlightBooking$ |
```

Result CSV-

timeStamp	elapsed	label	responseCode	responseMessage
1764621514660	108	Get All Flights	200	OK
1764621514660	108	Get All Flights	200	OK
1764621514715	53	Get All Flights	200	OK
1764621514660	108	Get All Flights	200	OK
1764621514660	109	Get All Flights	200	OK
1764621514664	104	Get All Flights	200	OK
1764621514763	33	Get All Flights	200	OK
1764621514778	35	Get All Airlines	200	OK
1764621514778	36	Get All Airlines	200	OK
1764621514778	43	Get All Airlines	200	OK
1764621514778	44	Get All Airlines	200	OK
1764621514782	40	Get All Airlines	200	OK
1764621514778	43	Get All Airlines	200	OK

4.2 With 50 Threads

The JMeter testing for 50 threads across the get,post, and delete requests have been carried out successfully through CLI as well as GUI.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get All Flights	50	568	245	1134	195.83	0.00%	26.8/sec	69.49	5.45	2654.3
Get All Airlines	50	622	248	1397	246.03	0.00%	31.0/sec	80.24	6.29	2653.8
Post Add a flight	50	10	1	41	10.26	100.00%	39.4/sec	84.92	0.00	2208.7
Post Search a ...	50	11	1	44	12.41	100.00%	40.7/sec	87.45	0.00	2203.0
Post Book a fli...	50	295	56	702	204.29	0.00%	38.7/sec	8.47	17.50	224.0
Del Cancel a b...	50	231	46	650	149.71	0.00%	43.3/sec	4.22	9.00	100.0
TOTAL	300	290	1	1397	291.80	33.33%	124.1/sec	202.91	22.06	1674.0

4.3 With 100 Threads

jmeter -n -t ‘LoadTesting100 .jmx’ -l result100.csv

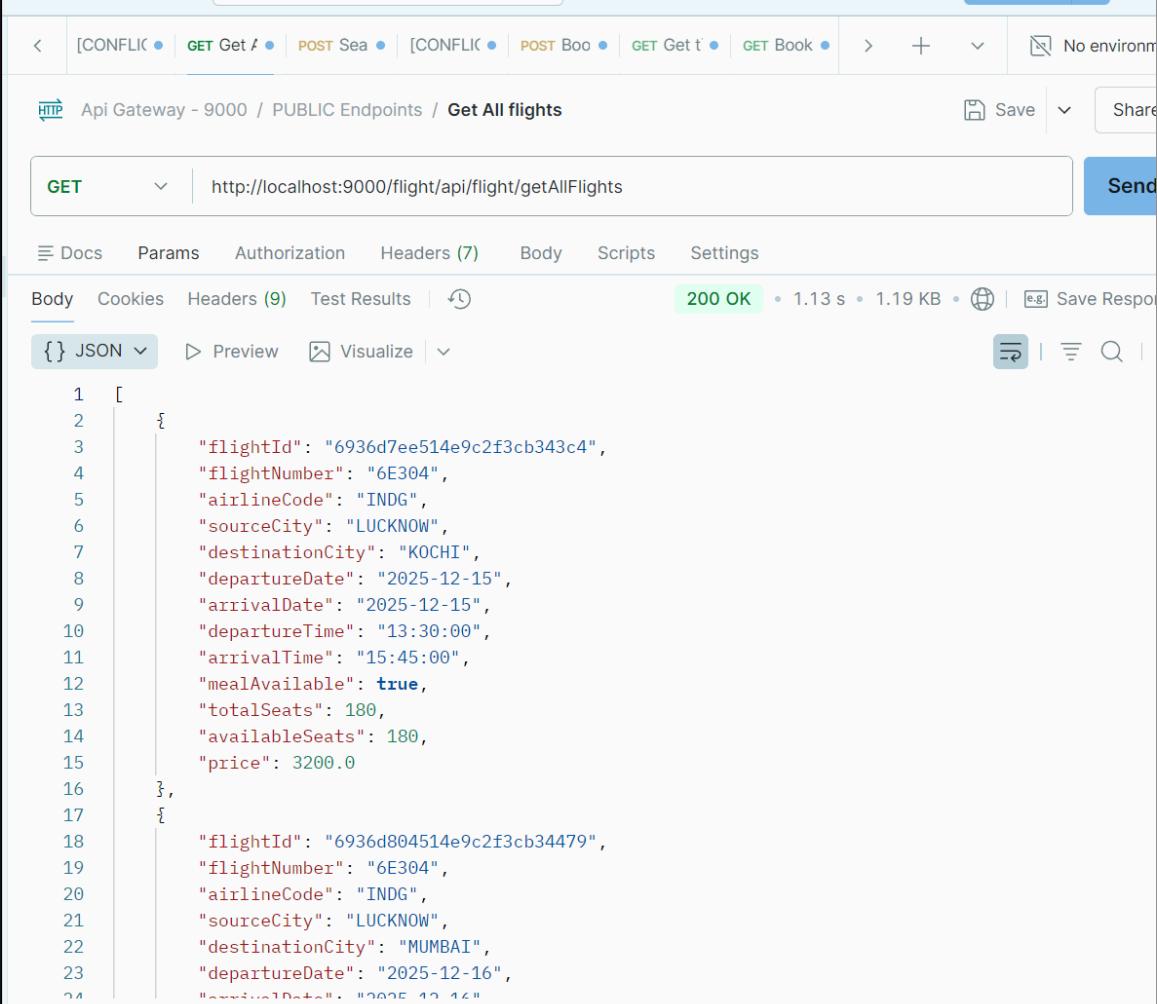
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get All Flights	200	553	203	806	139.40	0.00%	151.2/sec	391.82	30.71	2654.1
Get All Airlines	200	667	215	1081	213.94	0.00%	150.0/sec	389.13	30.48	2655.8
Post Book a fli...	200	283	20	738	190.77	0.00%	103.2/sec	22.57	46.66	224.0
Del Cancel a b...	200	157	47	668	127.68	0.00%	107.0/sec	10.45	22.26	100.0
TOTAL	800	415	20	1081	266.72	0.00%	249.4/sec	343.01	66.48	1408.5

5. Running the docker container

API Gateway on port 9000

5.1 Public Routes (No authorization required)

1. GET All flights



The screenshot shows the Postman interface with the following details:

- URL: `http://localhost:9000/flight/api/flight/getAllFlights`
- Method: `GET`
- Response Status: `200 OK`
- Response Time: `1.13 s`
- Response Size: `1.19 KB`
- Response Headers:
 - Content-Type: application/json
 - Content-Length: 113
 - Date: Mon, 11 Dec 2023 10:45:00 GMT
 - X-Powered-By: Express
- Response Body (JSON):

```
1  [
2   {
3     "flightId": "6936d7ee514e9c2f3cb343c4",
4     "flightNumber": "6E304",
5     "airlineCode": "INDG",
6     "sourceCity": "LUCKNOW",
7     "destinationCity": "KOCHI",
8     "departureDate": "2025-12-15",
9     "arrivalDate": "2025-12-15",
10    "departureTime": "13:30:00",
11    "arrivalTime": "15:45:00",
12    "mealAvailable": true,
13    "totalSeats": 180,
14    "availableSeats": 180,
15    "price": 3200.0
16  },
17  {
18    "flightId": "6936d804514e9c2f3cb34479",
19    "flightNumber": "6E304",
20    "airlineCode": "INDG",
21    "sourceCity": "LUCKNOW",
22    "destinationCity": "MUMBAI",
23    "departureDate": "2025-12-16",
24    "arrivalDate": "2025-12-16"
```

2. POST Search Flights

The screenshot shows the API Gateway interface at <http://localhost:9000> under the PUBLIC Endpoints section. A POST request is being made to <http://localhost:9000/flight/api/flight/search>. The request body contains the following JSON payload:

```
1 {
2   "sourceCity": "LUCKNOW",
3   "destinationCity": "KOCHE",
4   "travelDate": "2025-12-15",
5   "tripType": "ONE WAY"
6 }
```

The response status is 200 OK, and the response body is a JSON array containing one flight record:

```
1 [
2   {
3     "flightId": "6936d7ee514e9c2f3cb343c4",
4     "flightNumber": "6E304",
5     "airlineCode": "INDG",
6     "sourceCity": "LUCKNOW",
7     "destinationCity": "KOCHE",
8     "departureDate": "2025-12-15",
9     "arrivalDate": "2025-12-15",
10    "departureTime": "13:30:00",
11    "arrivalTime": "15:45:00",
12    "mealAvailable": true,
13    "totalSeats": 180,
14    "availableSeats": 180,
15    "price": 3200.0
16  }
17 ]
```

3. GET All Airlines

The screenshot shows the API Gateway interface at <http://localhost:9000> under the PUBLIC Endpoints section. A GET request is being made to <http://localhost:9000/flight/api/flight/getAllAirlines>. The response status is 200 OK, and the response body is a JSON array containing four airline records:

```
1 [
2   {
3     "airlineCode": "VIS",
4     "airlineName": "Vistaara"
5   },
6   {
7     "airlineCode": "INDG",
8     "airlineName": "Indigo"
9   },
10  {
11    "airlineCode": "AI",
12    "airlineName": "Air India"
13  },
14  {
15    "airlineCode": "SJ",
16    "airlineName": "Spice Jet"
17  }
18 ]
```

5.2 Sign Up & Login

1. Admin Sign Up

The screenshot shows a Postman request to `http://localhost:9000/auth/register`. The request method is `POST`. The body is set to `raw JSON` and contains the following data:

```
1 {
2     "username": "admin",
3     "password": "pass123",
4     "fullName": "Admin User",
5     "email": "admin@example.com",
6     "role": "ROLE_ADMIN"
7 }
```

The response status is `201 Created` with the message `User registered successfully`.

2. Admin Login

The screenshot shows a Postman request to `http://localhost:9000/auth/login`. The request method is `POST`. The body is set to `raw JSON` and contains the following data:

```
1 {
2     "username": "admin",
3     "password": "pass123"
4 }
```

The response status is `200 OK` with a response time of `112 ms`. The response body is a JWT token:

```
1 eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbjIsInJvbGUiOiJST0xFXFETUl0IiwiaWF0IjoxNzY1MjAxODA4LCJ1eHAiOiE3NjUyMDU0MDh9.yVGIKFHarGtu3gAwytIF_JYb1GYzG7BLcv0e_8nkcA
```

3. User Signup

The screenshot shows a Postman request for 'User Signup' at `http://localhost:9000/auth/register`. The request method is `POST`. The body is set to `raw JSON` and contains the following data:

```
1 {
2   "username": "user2",
3   "password": "pass1",
4   "fullName": "User1",
5   "email": "admin@example.com",
6   "role": "ROLE_USER"
7 }
```

The response status is `201 Created`, and the response body is `User registered successfully`.

4. User Login

The screenshot shows a Postman request for 'User Login' at `http://localhost:9000/auth/login`. The request method is `POST`. The body is set to `raw JSON` and contains the following data:

```
1 {
2   "username": "user2",
3   "password": "pass1"
4 }
```

The response status is `200 OK`, and the response body is a long encoded string: `eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMiIsInJvbGUIOiJST0xFX1VTRVIiLCJpYXQiOjE3NjUyMDUzMzQsImV4cCI6MTc2NTIwODk3NH0.E91mPSQhhwixUqt9Q8RVe6i-9WPrsEmab1j00eFS6uYA`.

5.3 Admin Endpoints

1. POST Add an airline

The screenshot shows the Postman interface with the following details:

- HTTP:** Api Gateway - 9000 / ADMIN Endpoints / Add an airline
- Method:** POST
- URL:** http://localhost:9000/flight/api/flight/addAirline
- Headers:** (10)
- Body:** (raw) JSON
- Request Body:**

```
1 {
2   "airlineCode": "SJ",
3   "airlineName": "Spice Jet"
4 }
```
- Response Status:** 201 Created
- Response Body:**

```
1 {
2   "airlineCode": "SJ",
3   "airlineName": "Spice Jet"
4 }
```

2. POST Add a flight

The screenshot shows the Postman interface with the following details:

- HTTP:** Api Gateway - 9000 / ADMIN Endpoints / Add a flight
- Method:** POST
- URL:** http://localhost:9000/flight/api/flight/airline/inventory/add
- Headers:** (10)
- Body:** (raw) JSON
- Request Body:**

```
1 {
2   "airlineCode": "AI",
3   "flightNumber": "AI304",
4   "sourceCity": "KOCHI",
5   "destinationCity": "MUMBAI",
6   "departureDate": "2025-12-16",
7   "departureTime": "10:50",
8   "arrivalDate": "2025-12-16",
9   "arrivalTime": "12:55",
10  "totalSeats": 180,
11  "price": 3200,
12  "mealAvailable": true
13 }
```
- Response Status:** 201 Created
- Response Body:**

```
1 {
2   "flightId": "6936d81c514e9c2f3cb3452e"
3 }
```

5.4 User Endpoints

1. POST Book a flight

HTTP Api Gateway - 9000 / USER Endpoints / Book a flight - ONE WAY

POST <http://localhost:9000/booking/api/booking/6936d7ee514e9c2f3cb343c4>

Docs Params Authorization Headers (10) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {  
2   "tripType": "ONE_WAY",  
3   "contactName": "Kanchan",  
4   "contactEmail": "kanchanrai2307@gmail.com",  
5   "passengers": [  
6     {  
7       "name": "John",  
8       "age": 26,  
9       "gender": "MALE",  
10      "seatOutbound": "S1"  
11    }  
12  ]  
13 }
```

Body Cookies Headers (9) Test Results 201 Created 5.88ms

{ } JSON ▾ Preview Visualize ▾

```
1 {  
2   "bookingId": "6936f09f48beee49f4f6ab6a",  
3   "tripType": "ONE_WAY",  
4   "outboundFlightId": "6936d7ee514e9c2f3cb343c4",  
5   "returnFlight": null,  
6   "pnrOutbound": "4D6C79",  
7   "passengers": null,  
8   "warnings": [],  
9   "pnrReturn": null,  
10  "contactName": "Kanchan",  
11  "contactEmail": "kanchanrai2307@gmail.com",  
12  "totalPassenger": 1,
```

← ⌂ ⚠️ 🗑️ 📧 📂 ⏮

Ticket booked - PNR 4D6C79 Inbox x

Summarise this email

 **kanchanrai2307@gmail.com**
to me ▾

Your ticket is booked.
PNR: 4D6C79
Return PNR: N/A
Outbound flight: 6936d7ee514e9c2f3cb343c4
Return flight: N/A
Passengers: 1
Status: CONFIRMED

2. DEL Cancel a flight by pnr

HTTP Api Gateway - 9000 / USER Endpoints / Cancel a booking by pnr

DELETE http://localhost:9000/booking/api/booking/cancel/4D6C79

Docs Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body Cookies Headers (9) Test Results | 200 OK • 4.6

{ } JSON ▾ ▷ Preview ⚡ Visualize ▾

```
1 {  
2   "message": "Booking cancelled"  
3 }
```

Ticket cancelled - PNR 4D6C79 Inbox x

Summarise this email

 **kanchanrai2307@gmail.com**
to me ▾

Your ticket is cancelled.
PNR: 4D6C79
Return PNR: N/A
Outbound flight: 6936d7ee514e9c2f3cb343c4
Return flight: N/A
Passengers: 1
Status: CANCELLED

5.4 User + Admin points

1. GET Ticket by pnr with validation
2. GET Booking History by mail

HTTP Api Gateway - 9000 / USER + ADMIN Endpoints / Get ticket by pnr

GET http://localhost:9000/booking/api/booking/ticket/0409F2

Docs Params Authorization Headers (8) Body Scripts Settings

Body Cookies Headers (9) Test Results | 404

{ } JSON ▾ ▷ Preview ⚡ Debug with AI | ▾

```
1 {
2   "error": "PNR not found"
3 }
```

HTTP Api Gateway - 9000 / USER + ADMIN Endpoints / Booking history by email

GET http://localhost:9000/booking/api/booking/history/kanchanrai2307@gmail.com

Docs Params Authorization Headers (8) Body Scripts Settings

Body Cookies Headers (9) Test Results | 200 OK • 94 ms • 3.29 KB

{ } JSON ▾ ▷ Preview ⚡ Visualize | ▾

```
1 [
2   {
3     "bookingId": "6936d87cd2308978ca9f246f",
4     "tripType": "ONE WAY",
5     "outboundFlightId": "6936d7ee514e9c2f3cb343c4",
6     "returnFlight": null,
7     "pnrOutbound": "CD9B13",
8     "passengers": null,
9     "warnings": null,
10    "pnrReturn": null,
11    "contactName": "Kanchan",
12    "contactEmail": "kanchanrai2307@gmail.com",
13    "totalPassenger": 1,
14    "status": "CONFIRMED"
15  },
16  {
17    "bookingId": "6936df61c7e6700af5a8e199",
18    "tripType": "ONE WAY",
19    "outboundFlightId": "6936d7ee514e9c2f3cb343c4",
20    "returnFlight": null,
21    "pnrOutbound": "8EC7F6",
22    "passenger": null,
23    "warnings": null,
24    "pnrReturn": null,
25    "contactName": "Kanchan",
26    "contactEmail": "kanchanrai2307@gmail.com",
27    "totalPassenger": 1,
28    "status": "CONFIRMED"
```