

# FLIGHT MANAGEMENT SYSTEM

(Implemented using microservices)

Debashrita Mandal

This microservices-based Flight Management System uses :

1. **Eureka Service Discovery**: For microservice registry
2. **OpenFeign and API Gateway**: For inter-service communication
3. **Resilience4j Circuit Breaker**: For fault tolerance
4. **Spring Cloud Config Server**: For centralized configuration
5. **RabbitMQ (using Docker)** : as an event-driven messaging layer (message broker)
6. **Notification Service (using Spring Mail)** for asynchronous email delivery.

The system persists data using **Spring Data JPA** with **MySQL**, supports validation using **Jakarta Validation**, and includes comprehensive **JUnit + Mockito** test coverage with **JaCoCo**.

## EUREKA SERVER

The screenshot shows the Eureka Server interface at [localhost:8761](http://localhost:8761). The main section is titled "System Status" and displays various system metrics:

Environment	test	Current time	2025-11-30T21:42:50 +0530
Data center	default	Uptime	00:01
		Lease expiration enabled	false
		Renews threshold	6
		Renews (last min)	1

Below this is the "DS Replicas" section, which lists instances currently registered with Eureka:

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - <a href="http://10.6.172.123:api-gateway:8765">10.6.172.123:api-gateway:8765</a>
BOOKING-MICROSERVICE	n/a (1)	(1)	UP (1) - <a href="http://10.6.172.123:booking-microservice:8082">10.6.172.123:booking-microservice:8082</a>
FLIGHT-MICROSERVICE	n/a (1)	(1)	UP (1) - <a href="http://10.6.172.123:flight-microservice:8081">10.6.172.123:flight-microservice:8081</a>

At the bottom is the "General Info" section.

## CIRCUIT BREAKER (Resilience4j)

The screenshot shows a POST request to <http://localhost:8082/api/flight/booking/79448598> using the Postman collection "FMS Microservice". The request body is:

```
1 {
2   "email": "debashritamandal852@gmail.com",
3   "name": "Debashrita Mandal",
4   "numberOfSeats": 2,
5   "passengers": [
6     { "name": "Debashrita Mandal", "gender": "F", "age": 21 },
7     { "name": "Aahana Banerjee", "gender": "F", "age": 22 }
]
```

The response status is 201 Created.

# RABBITMQ SERVER

The screenshot shows the Docker Desktop interface. On the left, a sidebar lists various sections: Ask Gordon (BETA), Containers (selected), Images, Volumes, Kubernetes, Builds, Models, MCP Toolkit (BETA), Docker Hub, Docker Scout, and Extensions. The main area is titled "Containers" with a "Give feedback" link. It displays resource usage: Container CPU usage at 0.48% / 800% (8 CPUs available) and Container memory usage at 133.3MB / 7.36GB. A "Show charts" button is present. Below this, a table lists containers, showing one named "rabbitmq" with a green dot indicating it is running. The table columns are Name, Container ID, Image, Port(s), CPU (%), and Actions. The "Actions" column for "rabbitmq" includes icons for Stop, Start, Open terminal, and Delete. A search bar and a "Only show running containers" toggle switch are also visible. At the bottom, a "Walkthroughs" section offers links to "Multi-container applications" (8 mins) and "Containerize your application" (\$ docker init 3 mins). A "View more in the Learning center" link is provided. The status bar at the bottom shows "Engine running", system resources (RAM 1.40 GB, CPU 0.13%, Disk 1.73 GB used / limit 1006.85 GB), and the version "v4.53.0".

Registered RabbitMQ on Docker Desktop.

The screenshot shows the RabbitMQ Management Interface (localhost:15672/#). The top navigation bar includes links to Authn | edX, Lecture Videos | Intr..., Analytics Academy, Machine Learning | ..., Build Your First Web..., Google Cybersecuri..., AI Ethics | IBM, and All Bookmarks. The title bar shows "RabbitMQ TM" and "RabbitMQ 3.11.28 Erlang 25.3.2.9". The top right shows "Refreshed 2025-12-01 11:58:42", "Virtual host All", "Cluster rabbit@5acd0416240f", and "User fms\_user Log out". The main menu has tabs for Overview (selected), Connections, Channels, Exchanges, Queues, and Admin. The "Overview" section has a "Totals" dropdown. It displays metrics: Queued messages (last minute), Currently idle, Message rates (last minute), and Global counts. Below are buttons for Connections: 0, Channels: 0, Exchanges: 7, Queues: 0, Consumers: 0. The "Nodes" section shows a table with one node: rabbit@5acd0416240f. The table columns are Name, File descriptors, Socket descriptors, Erlang processes, Memory, Disk space, Uptime, Info, and Reset stats. The "File descriptors" row shows 38 total and 1048576 available. The "Socket descriptors" row shows 0 total and 943629 available. The "Erlang processes" row shows 404 total and 1048576 available. The "Memory" row shows 145 MiB total and 3.0 GiB high watermark. The "Disk space" row shows 954 GiB total and 48 MiB low watermark. The "Uptime" row shows 3m 58s. The "Info" row shows basic (2), disc (2), rss (2). Buttons "This node" and "All nodes" are at the bottom. Below the nodes table are sections for Churn statistics, Ports and contexts, and Export definitions.

Logged into RabbitMQ.

**RabbitMQ**™ RabbitMQ 3.11.28 Erlang 25.3.2.9

Refreshed 2025-12-02 00:00:14 Refresh every 5 seconds

Virtual host All Cluster rabbit@5acd0416240f User fms\_user Log out

## Queues

All queues (1)

Pagination

Page 1 of 1 - Filter:   Regex ?

Displaying 1 item , page size up to: 100

Overview				Messages			Message rates			+/-
Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
/fms	booking.email.queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s

▶ Add a new queue

HTTP API Documentation Tutorials New releases Commercial edition Commercial support Discussions Discord Slack Plugins GitHub

Created a queue for Booking-Microservice.

**Gmail** Search mail

Compose

Inbox 6,904

Starred Snoozed Sent Drafts Purchases More

Labels +

Booking Confirmation - 4EA5863B Inbox x

debashritamandal852@gmail.com to me ▾

2:13 PM (0 minutes ago) star smiley

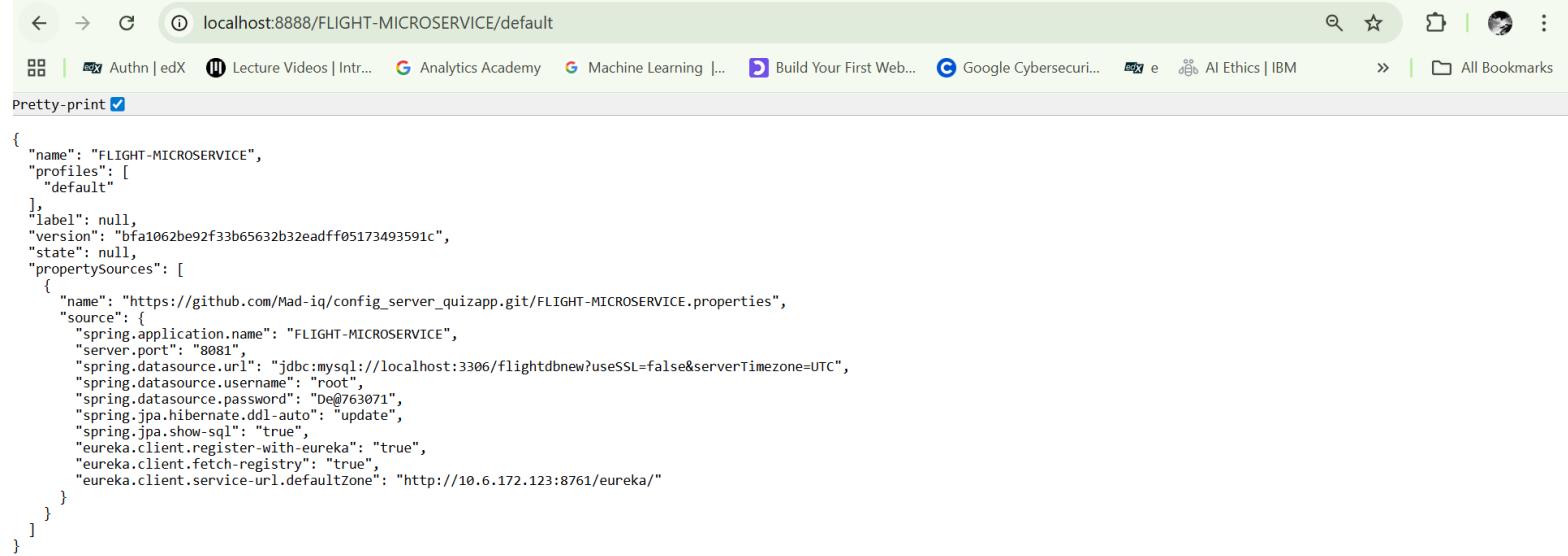
Hello Debashrita Mandal,  
Your booking is confirmed.  
PNR: 4EA5863B  
Flight ID: 79448598

Reply Forward

① Upgrade →

Received an email after successful booking on the provided booking email id.

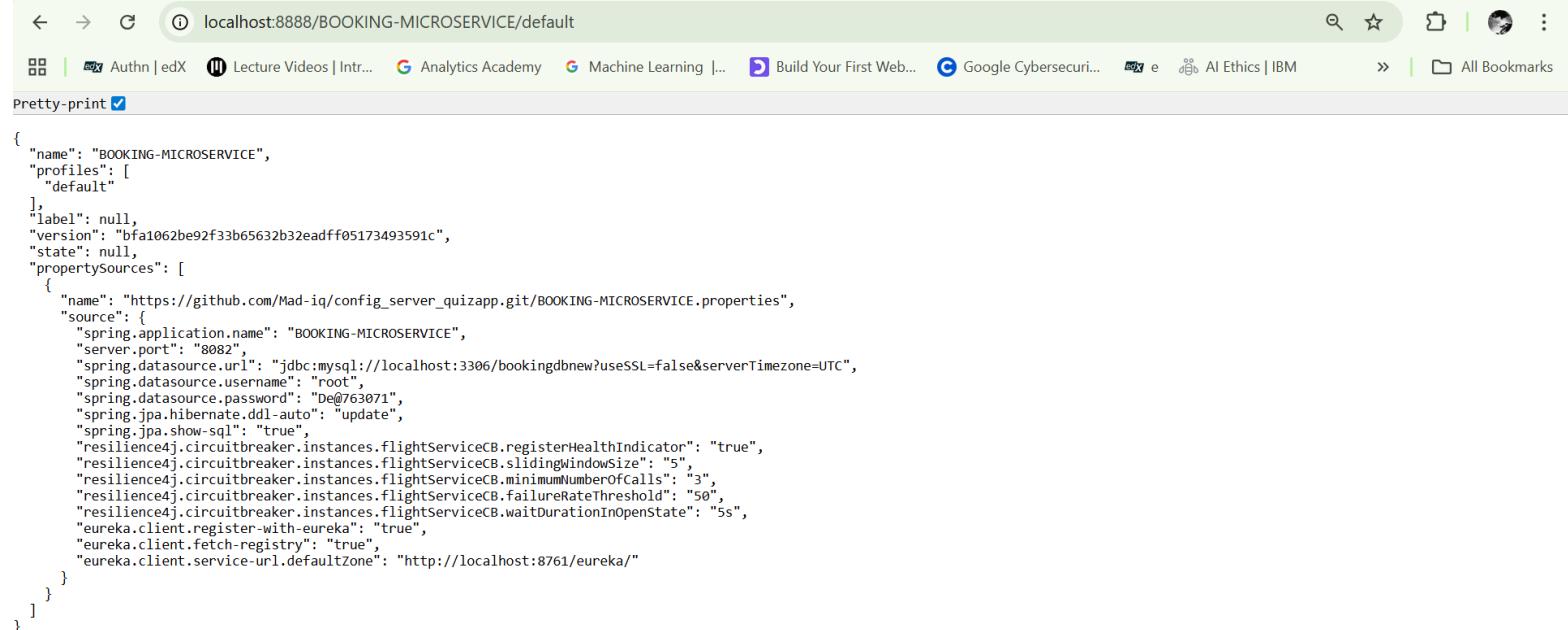
## CONFIG SERVER



A screenshot of a web browser window titled "localhost:8888/FLIGHT-MICROSERVICE/default". The page displays a JSON object representing the configuration for the FLIGHT-MICROSERVICE. The JSON structure includes fields for name, profiles, label, version, state, and propertySources, which contain detailed Spring Boot configuration properties like database URL, port, and Eureka service discovery.

```
{
  "name": "FLIGHT-MICROSERVICE",
  "profiles": [
    "default"
  ],
  "label": null,
  "version": "bfa1062be92f33b65632b32eadff05173493591c",
  "state": null,
  "propertySources": [
    {
      "name": "https://github.com/Mad-iq/config_server_quizapp.git/FLIGHT-MICROSERVICE.properties",
      "source": {
        "spring.application.name": "FLIGHT-MICROSERVICE",
        "server.port": "8081",
        "spring.datasource.url": "jdbc:mysql://localhost:3306/flightdbnew?useSSL=false&serverTimezone=UTC",
        "spring.datasource.username": "root",
        "spring.datasource.password": "De@763071",
        "spring.jpa.hibernate.ddl-auto": "update",
        "spring.jpa.show-sql": "true",
        "eureka.client.register-with-eureka": "true",
        "eureka.client.fetch-registry": "true",
        "eureka.client.service-url.defaultZone": "http://10.6.172.123:8761/eureka/"
      }
    }
  ]
}
```

Uploaded the application.properties for Flight-microservice into the config server.



A screenshot of a web browser window titled "localhost:8888/BOOKING-MICROSERVICE/default". The page displays a JSON object representing the configuration for the BOOKING-MICROSERVICE. Similar to the FLIGHT-MICROSERVICE, it includes fields for name, profiles, label, version, state, and propertySources, with detailed configuration properties for the booking database and Eureka integration.

```
{
  "name": "BOOKING-MICROSERVICE",
  "profiles": [
    "default"
  ],
  "label": null,
  "version": "bfa1062be92f33b65632b32eadff05173493591c",
  "state": null,
  "propertySources": [
    {
      "name": "https://github.com/Mad-iq/config_server_quizapp.git/BOOKING-MICROSERVICE.properties",
      "source": {
        "spring.application.name": "BOOKING-MICROSERVICE",
        "server.port": "8082",
        "spring.datasource.url": "jdbc:mysql://localhost:3306/bookingdbnew?useSSL=false&serverTimezone=UTC",
        "spring.datasource.username": "root",
        "spring.datasource.password": "De@763071",
        "spring.jpa.hibernate.ddl-auto": "update",
        "spring.jpa.show-sql": "true",
        "resilience4j.circuitbreaker.instances.flightserviceCB.registerHealthIndicator": "true",
        "resilience4j.circuitbreaker.instances.flightserviceCB.slidingWindowSize": "5",
        "resilience4j.circuitbreaker.instances.flightserviceCB.minimumNumberOfCalls": "3",
        "resilience4j.circuitbreaker.instances.flightserviceCB.failureRateThreshold": "50",
        "resilience4j.circuitbreaker.instances.flightserviceCB.waitDurationInOpenState": "5s",
        "eureka.client.register-with-eureka": "true",
        "eureka.client.fetch-registry": "true",
        "eureka.client.service-url.defaultZone": "http://localhost:8761/eureka/"
      }
    }
  ]
}
```

Uploaded the application.properties for Booking-microservice into the config server.

# API-GATEWAY

The screenshot shows the API-Gateway interface. On the left, there's a sidebar with collections like FMS, FMS Microservice, and Quiz\_microservices. The main area shows a POST request to 'Add inventory' with the URL `http://localhost:8765/FLIGHT-MICROSERVICE/api/flight`. The request body is a JSON object:

```
1 {
2   "airlineName": "IndiGo",
3   "source": "DELHI",
4   "destination": "MUMBAI",
5   "startDate": "2025-12-10T09:30:00",
6   "endDate": "2025-12-10T11:45:00",
7   "availableSeats": 10,
8   "ticketPrice": 4500,
9   "mealStatus": true
10 }
```

The response status is 201 Created, with a time of 259 ms and a size of 182 B. The response body is:

```
1 {
2   "message": "Flight added successfully",
3   "flightId": "AFA9CCF7"
4 }
```

## POSTMAN TESTING

### 1. Add Inventory

The screenshot shows Postman testing the 'Add inventory' endpoint. The request URL is `http://localhost:8765/FLIGHT-MICROSERVICE/api/flight`. The request body is the same JSON object as in the API-Gateway screenshot.

The response status is 201 Created, with a time of 259 ms and a size of 182 B. The response body is:

```
1 {
2   "message": "Flight added successfully",
3   "flightId": "AFA9CCF7"
4 }
```

## 2. Search Flights

The screenshot shows the Postman application interface. On the left, the sidebar lists collections: FMS, FMS Microservice, FMS\_Reactive, and others. The 'FMS Microservice' collection is expanded, showing methods: POST Add inventory, POST search flights (which is selected), POST Book Ticket, GET Search by pnr, and GET Search by mail. The main workspace shows a POST request to 'http://localhost:8765/FLIGHT-MICROSERVICE/api/flight/search'. The request body is empty. The response status is '200 OK' with a response time of 564 ms, size of 307 B, and a green 'Save Response' button. The response body is a JSON object:

```
availableSeats: 8,
airline: "IndiGo",
flightId: "79448598",
availableSeatNumbers: [
  "2B",
  "2C",
  "1C",
  "2D",
  "1D",
  "1E",
  "1F",
  "2A"
],
price: 4500.0,
dateTime: "2025-12-10T09:30"
```

## 3. Book Tickets

The screenshot shows the Postman application interface. The sidebar lists collections: FMS, FMS Microservice, FMS\_Reactive, and others. The 'FMS Microservice' collection is expanded, showing methods: POST Add inventory, POST search flights, and POST Book Ticket (which is selected). The main workspace shows a POST request to 'http://localhost:8765/BOOKING-MICROSERVICE/api/flight/booking/79448598'. The request body is a JSON object:

```
passengers: [
  { "name": "Debashrita Mandal", "gender": "F", "age": 21 },
  { "name": "Aahana Banerjee", "gender": "F", "age": 22 }
],
mealPreference: "VEG",
seatNumbers: ["2A", "2B"]
```

The response status is '201 Created' with a response time of 421 ms, size of 190 B, and a green 'Save Response' button. The response body is a JSON object:

```
{
  "pnr": "D13A32F4",
  "message": "Booking successful",
  "totalPrice": 9000.0
}
```

## 4. Search by PNR

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:8765/BOOKING-MICROSERVICE/api/flight/ticket/D4F30643`. The response body is a JSON object:

```
1 {  
2   "pnr": "D4F30643",  
3   "flightId": "79448598",  
4   "status": "CONFIRMED",  
5   "seatNumbers": [  
6     "1A",  
7     "1B"  
8   ],  
9   "email": "debashritamandal852@gmail.com",  
10  "passengers": [  
11    {  
12      "id": 1,  
13      "name": "Debashrita Mandal",  
14      "gender": "F",  
15      "..."  
16    }  
17  ]  
18}
```

## 5. Search by email

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:8765/BOOKING-MICROSERVICE/api/flight/booking/history/debashritamandal852@gmail.com`. The response body is a JSON object:

```
1 {  
2   "email": "debashritamandal852@gmail.com",  
3   "history": [  
4     {  
5       "status": "CONFIRMED",  
6       "flightId": "79448598",  
7       "pnr": "D4F30643",  
8       "date": "2025-12-10"  
9     },  
10    {  
11      "status": "CONFIRMED",  
12      "flightId": "79448598",  
13      "pnr": "D13A32F4",  
14    }  
15  ]  
16}
```

## 6. Delete ticket

The screenshot shows the Postman interface with the following details:

- Left Sidebar (kspase):**
  - FMS
  - FMS Microservice
    - POST Add inventory
    - POST search flights
    - POST Book Ticket
    - GET Search by pn
    - GET Search by mail
    - GET Delete flights
  - FMS\_Reactive
  - FMS\_Reactive\_Mongo
  - New Collection
  - Quiz\_microservices
    - POST add questions
    - GET get questions by category
    - GET generate random id
    - GET get all questions
    - POST get questions by id
    - POST score
    - POST create quiz
- Header Bar:** POST Book Ticke • POST search flight • GET Search by pn • FMS Microserv • DEL Delete flight: • + • No environment
- Request URL:** FMS Microservice / Delete flights
- Method:** DELETE
- URL:** http://localhost:8082/api/flight/booking/cancel/D4F30643
- Params:** Key Value Description
- Body:** JSON
 

```

1 {
2   "pnr": "D4F30643",
3   "message": "Ticket cancelled successfully"
4 }
```
- Response:** 200 OK (832 ms, 224 B) | Save Response
- Bottom Tools:** Postbot, Runner, Start Proxy, Cookies, Vault, Trash

## JACOCO AND SONARQUBE REPORTS:

### FLIGHT-MICROSERVICE:

#### 1. JACOCO REPORT

Code Coverage: 93%

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.flight.exception	82%		100%	n/a	1	6	2	14	1	5	0	1
com.flight	0%		n/a		2	2	3	3	2	2	1	1
com.flight.service	97%		85%		3	16	0	76	0	6	0	1
com.flight.controller	91%		100%		1	7	1	11	1	5	0	1
com.flight.model	100%		n/a		0	2	0	10	0	2	0	2
Total	32 of 532	93%	3 of 26	88%	7	33	6	114	4	20	1	6

Created with JaCoCo 0.8.10 202304240956

## 2. SONARQUBE REPORT

Code Coverage: 93.6%

The screenshot shows the SonarQube interface for the project 'flight-microservice'. The main summary page displays various quality metrics. Key highlights include:

- Security:** 0 Open issues (Grade A)
- Reliability:** 0 Open issues (Grade A)
- Maintainability:** 35 Open issues (Grade A)
- Coverage:** 93.6% (No conditions set on 114 Lines to cover)
- Duplications:** 0.0% (No conditions set on 659 Lines)

The left sidebar shows navigation options like Overview, Main Branch (selected), Pull Requests, Branches, Information, and Administration.

## BOOKING-MICROSERVICE

### 1. JACOCO REPORT:

Code Coverage: 90%

The screenshot shows the Jacoco report for the 'booking-microservice'. The report provides detailed coverage analysis across various elements. Key findings include:

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.booking.service	93%	93%	72%	72%	10	30	9	117	0	10	0	1
com.booking.config	0%	0%	n/a	n/a	4	4	6	6	4	4	1	1
com.booking.publisher	0%	0%	n/a	n/a	1	1	3	3	1	1	1	1
com.booking	37%	37%	n/a	n/a	1	2	2	3	1	2	0	1
com.booking.model	100%	100%	n/a	n/a	0	3	0	10	0	3	0	3
com.booking.exception	100%	100%	100%	100%	0	7	0	16	0	6	0	1
com.booking.controller	100%	100%	n/a	n/a	0	4	0	5	0	4	0	1
Total	74 of 769	90%	11 of 42	73%	16	51	20	160	6	30	2	9

## 2. SONARQUBE REPORT

Code Coverage: 84.7%

The screenshot shows the SonarQube web interface for the project 'booking-microservice'. The main dashboard displays various metrics:

- Security:** 0 Open issues (Grade A)
- Reliability:** 0 Open issues (Grade A)
- Maintainability:** 38 Open issues (Grade A)
- Accepted Issues:** 0
- Coverage:** 84.7% (No conditions set on 160 Lines to cover)
- Duplications:** 0.0% (No conditions set on 806 Lines)

The left sidebar shows navigation options like Overview, Main Branch (selected), Pull Requests, and Branches.

## JMETER TESTING

### 1. 20 USERS

The screenshot shows the Apache JMeter 5.6.3 interface with a test plan containing three thread groups:

- Thread Group (20 users):** Contains an HTTP Request configuration with the following settings:
  - Name: HTTP Request
  - Comments: (empty)
  - Protocol [http]: http
  - Server Name or IP: localhost
  - Port Number: 8082
  - HTTP Request:
    - Method: GET
    - Path: /api/flight/ticket/D4F30643
    - Follow Redirects: checked
    - Use KeepAlive: checked
    - Content encoding: (empty)
    - Redirect Automatically: unchecked
    - Use multipart/form-data: unchecked
    - Browser-compatible headers: unchecked
  - Parameters tab (Send Parameters With the Request):

Name:	Value:	URL Encode?	Content-Type:	Include Equals?
(empty)	(empty)	(empty)	(empty)	(empty)
- Thread Group (50 Users):** Contains an HTTP Request configuration with the following settings:
  - Protocol [http]: http
  - Server Name or IP: localhost
  - Port Number: 8082
  - HTTP Request:
    - Method: GET
    - Path: /api/flight/ticket/D4F30643
    - Follow Redirects: checked
    - Use KeepAlive: checked
    - Content encoding: (empty)
    - Redirect Automatically: unchecked
    - Use multipart/form-data: unchecked
    - Browser-compatible headers: unchecked
- Thread Group (100 Users):** Contains an HTTP Request configuration with the following settings:
  - Protocol [http]: http
  - Server Name or IP: localhost
  - Port Number: 8082
  - HTTP Request:
    - Method: GET
    - Path: /api/flight/ticket/D4F30643
    - Follow Redirects: checked
    - Use KeepAlive: checked
    - Content encoding: (empty)
    - Redirect Automatically: unchecked
    - Use multipart/form-data: unchecked
    - Browser-compatible headers: unchecked

The screenshot shows the Apache JMeter interface with a dark theme. The left sidebar displays a tree view of the test plan, including 'Test Plan', 'Thread Group (20 users)', 'Thread Group (50 Users)', and 'Thread Group (100 Users)'. Each group contains an 'HTTP Request' and a 'Summary Report' element. The 'Summary Report' under the first thread group is currently selected, highlighted with a blue background.

The main panel is titled 'Summary Report' and contains the following fields:

- Name:
- Comments:
- Filename:
- Log/Display Only:  Errors  Successes

Below these fields is a table showing performance metrics:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	20	6	3	14	2.29	0.00%	4.2/sec	1.60	0.58	389.0
TOTAL	20	6	3	14	2.29	0.00%	4.2/sec	1.60	0.58	389.0
:::										

At the bottom of the table are two checkboxes:  Include group name in label? and  Save Table Header. There is also a  button.

## 2. 50 USERS

Summary Report1.jmx (C:\Users\KIT\Downloads\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\Summary Report1.jmx) - Apache JMeter (5.6.3)

File Edit Search Run Options Tools Help

0:00:14 0/0/170

Test Plan

- Thread Group (20 users)
  - HTTP Request
  - Summary Report
  - View Results Tree
- Thread Group (50 Users)
  - HTTP Request
  - Summary Report
  - View Results Tree
- Thread Group (100 Users)
  - HTTP Request
  - Summary Report
  - View Results Tree

HTTP Request

Name: HTTP Request

Comments:

Basic Advanced

Web Server

Protocol [http]: http Server Name or IP: localhost Port Number: 8082

HTTP Request

Method: GET Path: /api/flight/ticket/D4F30643 Content encoding:

Redirect Automatically Follow Redirects Use KeepAlive Use multipart/form-data Browser-compatible headers

Parameters Body Data Files Upload

Send Parameters With the Request:

Name:	Value	URL Encode?	Content-Type	Include Equals?
-------	-------	-------------	--------------	-----------------

Detail Add Add from Clipboard Delete Up Down

Summary Report1.jmx (C:\Users\KIT\Downloads\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\Summary Report1.jmx) - Apache JMeter (5.6.3)

File Edit Search Run Options Tools Help

0:00:14 0/0/170

Test Plan

- Thread Group (20 users)
  - HTTP Request
  - Summary Report
  - View Results Tree
- Thread Group (50 Users)
  - HTTP Request
  - Summary Report
  - View Results Tree
- Thread Group (100 Users)
  - HTTP Request
  - Summary Report
  - View Results Tree

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename: Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB...	Sent KB/sec	Avg. Bytes
HTTP Request	100	5	3	13	1.73	0.00%	4.6/min	0.03	0.01	389.0
TOTAL	100	5	3	13	1.73	0.00%	4.6/min	0.03	0.01	389.0

Include group name in label? Save Table Data Save Table Header

The screenshot shows the Apache JMeter interface with the following details:

- Title Bar:** Summary Report1.jmx (C:\Users\KIT\Downloads\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\Summary Report1.jmx) - Apache JMeter (5.6.3)
- Toolbar:** File, Edit, Search, Run, Options, Tools, Help
- Test Plan Tree:** The left sidebar displays the Test Plan structure:
  - Test Plan
  - Thread Group (20 users)
    - HTTP Request
    - Summary Report
    - View Results Tree (selected)
  - Thread Group (50 Users)
    - HTTP Request
    - Summary Report
    - View Results Tree
  - Thread Group (100 Users)
    - HTTP Request
    - Summary Report
    - View Results Tree
- View Results Tree Dialog:** The main panel shows the configuration for viewing results:
  - Name: View Results Tree
  - Comments: Write results to file / Read from file
  - Filename: [empty input field]
  - Buttons: Browse..., Log/Display Only: Errors, Successes, Configure
  - Search: Search bar, Case sensitive, Regular exp., Search, Reset buttons
  - Text Area: Sampler result (displaying 15 repeated "HTTP Request" entries)
  - Bottom Buttons: Raw, Parsed, Scroll automatically?
- Header:** 00:01:14 ⚠ 0 0/170

### **3. 100 USERS**

The screenshot shows the Apache JMeter 5.6.3 interface. The left sidebar displays a tree view of the test plan:

- Test Plan
- Thread Group (20 users)
  - HTTP Request
  - Summary Report
  - View Results Tree
- Thread Group (50 Users)
  - HTTP Request
  - Summary Report
  - View Results Tree
- Thread Group (100 Users)
  - HTTP Request
  - Summary Report
  - View Results Tree

The "HTTP Request" under the "Thread Group (100 Users)" is selected and highlighted with a blue background. The main panel on the right shows the configuration for this selected request:

**HTTP Request**

Name: HTTP Request  
Comments:

**Basic** **Advanced**

- Web Server  
Protocol [http]: http Server Name or IP: localhost Port Number: 8082

- HTTP Request  
Method: GET Path: /api/flight/ticket/D4F30643  
 Redirect Automatically  Follow Redirects  Use KeepAlive  Use multipart/form-data  Browser-compatible headers

**Parameters** **Body Data** **Files Upload**

Send Parameters With the Request:

Name:	Value	URL Encode?	Content-Type	Include Equals?
-------	-------	-------------	--------------	-----------------

Buttons at the bottom: Detail, Add, Add from Clipboard, Delete, Up, Down

The screenshot shows the Apache JMeter interface with a dark theme. The left sidebar displays a tree view of the test plan, including a 'Test Plan' node and three 'Thread Group' nodes under it. The first Thread Group has 20 users, the second has 50 users, and the third has 100 users. Each group contains an 'HTTP Request' sampler and a 'Summary Report' listener. The 'Summary Report' listener for the 100-user group is currently selected, highlighted with a blue background. The main panel on the right is titled 'Summary Report' and contains configuration fields for 'Name' (set to 'Summary Report'), 'Comments', and 'Filename'. Below these are buttons for 'Browse...', 'Log/Display Only', and 'Configure'. A large table provides detailed performance metrics for the 'HTTP Request' sampler across all three thread groups. The table includes columns for Label, # Samples, Average, Min, Max, Std. Dev., Error %, Throughput, Received KB/sec, Sent KB/sec, and Avg. Bytes. The data shows 300 samples for each group, with averages around 6 ms and errors near 0.00%. The total throughput is listed as 13.4/min.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	300	6	3	16	1.52	0.00%	13.4/min	0.08	0.03	389.0
TOTAL	300	6	3	16	1.52	0.00%	13.4/min	0.08	0.03	389.0

The screenshot shows the Apache JMeter interface with the title bar "Summary Report1.jmx (C:\Users\KIT\Downloads\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\Summary Report1.jmx) - Apache JMeter (5.6.3)". The menu bar includes File, Edit, Search, Run, Options, Tools, and Help. The toolbar contains various icons for file operations like Open, Save, Print, and Help. The left sidebar displays a hierarchical tree of test plans and thread groups, with "View Results Tree" selected under the third thread group. The main panel is titled "View Results Tree" and contains fields for "Name" (set to "View Results Tree"), "Comments", and options to "Write results to file / Read from file". It also includes a "Filename" field with a "Browse..." button, and checkboxes for "Log/Display Only", "Errors", "Successes", and "Configure". Below these are search and filter controls: "Search:" with a "Case sensitive" and "Regular exp." checkbox, and "Reset" and "Search" buttons. The main results area is titled "Sampler result" and lists 15 "HTTP Request" entries, each preceded by a green checkmark icon. At the bottom, there are buttons for "Raw" and "Parsed" data formats, and a checkbox for "Scroll automatically?".