

FLIGHT MANAGEMENT SYSTEM

Debashrita Mandal

INTRODUCTION

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 dashboard. At the top, there's a header bar with a search icon, a star icon, a folder icon, and a refresh icon. Below the header, the URL is shown as localhost:8761. The main content area has a title "System Status" followed by a table with two rows. The first row shows "Environment" as "test" and "Current time" as "2025-11-30T21:42:50 +0530". The second row shows "Data center" as "default" and "Uptime" as "00:01". Below this is another table with four rows, showing "Lease expiration enabled" as "false", "Renews threshold" as "6", and "Renews (last min)" as "1". Under the "System Status" section, there's a heading "DS Replicas" and a table titled "Instances currently registered with Eureka" with three rows. The first row is for "API-GATEWAY" with status "UP (1) - 10.6.172.123:api-gateway:8765". The second row is for "BOOKING-MICROSERVICE" with status "UP (1) - 10.6.172.123:booking-microservice:8082". The third row is for "FLIGHT-MICROSERVICE" with status "UP (1) - 10.6.172.123:flight-microservice:8081". At the bottom of the dashboard, there's a "General Info" section.

CIRCUIT BREAKER (Resilience4j)

The screenshot shows the Postman application interface. On the left, there's a sidebar with a tree view of collections: "FMS", "FMS Microservice" (which is expanded to show "POST Add inventory", "POST search flights", and "POST Book Ticket"), "FMS_Reactive", "FMS_Reactive_Mongo", "New Collection", "Quiz_microservices", "Spring_Mongo_Reactive", "Springboot_Mongodb", and "WebfluxTutorial". The main area shows a "Book Ticket" collection. A "POST" request is selected with the URL "http://localhost:8082/api/flight/booking/79448598". The "Body" tab is active, showing a JSON payload:

```
1 {  
2   "email": "debashritamandal1852@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 }  
]
```

Below the body, the response is shown as a "201 Created" status with a timestamp of "3.10 s" and a size of "253 B". The response body is also JSON:

```
1 {  
2   "message": "Cannot book right now: Flight service is unavailable",  
3   "status": "FAILED"  
4 }
```

RABBITMQ SERVER

The screenshot shows the Docker Desktop interface. On the left sidebar, under the 'Containers' section, there is a single entry for a 'rabbitmq' container. The container details show it's using 0.48% of CPU and 133.3MB of memory. Below the sidebar, there are sections for 'Walkthroughs' featuring 'Multi-container applications' and 'Containerize your application'. At the bottom, system status information is displayed: 'Engine running', 'RAM 1.40 GB CPU 0.13%', 'Disk: 1.73 GB used (limit 1006.85 GB)', and a version indicator 'v4.53.0'.

Registered RabbitMQ on Docker Desktop.

The screenshot shows the RabbitMQ Management Interface at localhost:15672/#/. The top navigation bar includes links for Authn | edX, Lecture Videos | Intr..., Analytics Academy, Machine Learning |..., Build Your First Web..., Google Cybersecuri..., AI Ethics | IBM, and All Bookmarks. The main dashboard displays cluster statistics: RabbitMQ 3.11.28, Erlang 25.3.2.9, Refreshed 2025-12-01 11:58:42, Virtual host All, Cluster rabbit@5acd0416240f, User fms_user, and Log out. The 'Overview' tab is selected, showing 'Totals' data: Queued messages (last minute), Currently idle, Message rates (last minute), Currently idle, and Global counts. Below this, a table provides detailed resource usage for a single node: Name (rabbit@5acd0416240f), File descriptors (38), Socket descriptors (0), Erlang processes (404), Memory (145 MiB), Disk space (954 GiB), Uptime (3m 58s), Info (basic, disc, 2, rss), and Reset stats (+/-). Other tabs include Connections, Channels, Exchanges, Queues, and Admin.

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 Smile Forward More

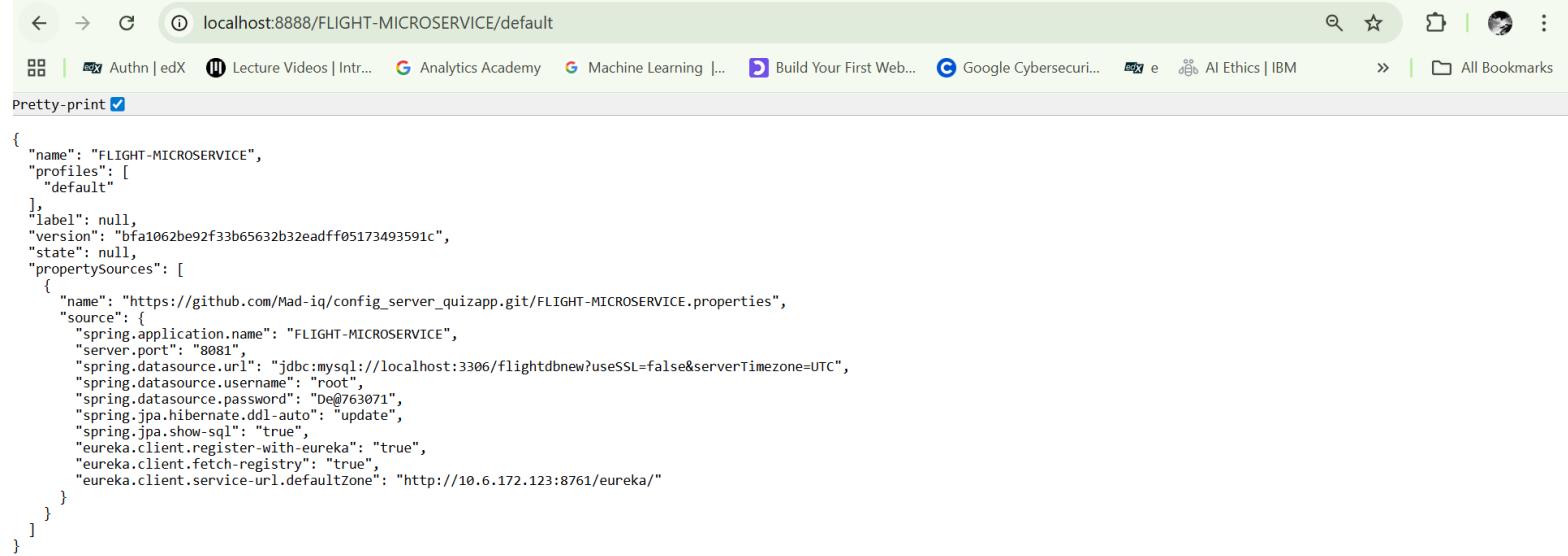
Hello Debashrita Mandal,
Your booking is confirmed.
PNR: 4EA5863B
Flight ID: 79448598

Reply Forward

① Upgrade →

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

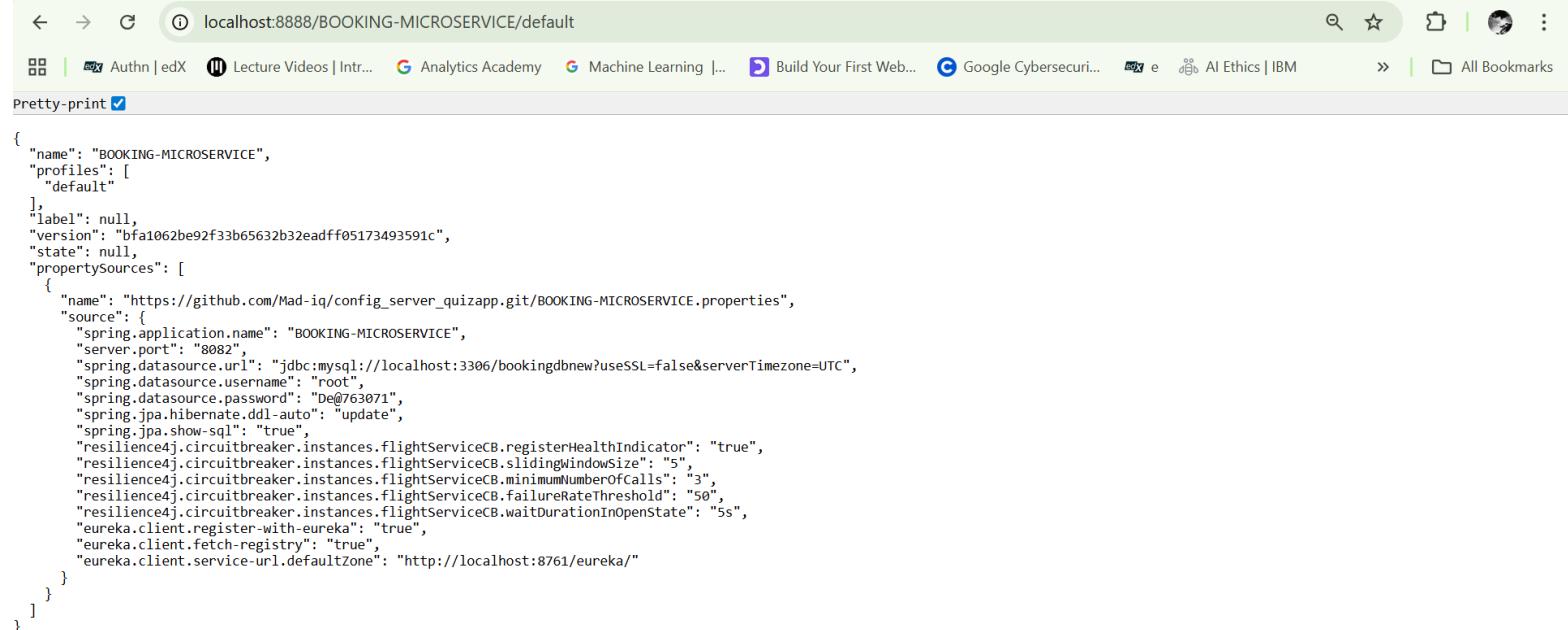
CONFIG SERVER



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 client settings.

```
{
  "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 application interface. On the left, the sidebar lists collections: FMS, FMS Microservice, FMS_Reactive, FMS_Reactive_Mongo, New Collection, Quiz_microservices, Spring_Mongo_Reactive, Springboot_Mongodb, and WebfluxTutorial. The FMS Microservice collection is expanded, showing methods: POST Add inventory, POST search flights, POST Book Ticket, GET Search by pnr, and GET Search by mail. The FMS_Reactive collection is also expanded. The main workspace shows a POST request to 'Add inventory' at 'http://localhost:8765/FLIGHT-MICROSERVICE/api/flight'. The request body is raw JSON:

```
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 the Postman application interface. The sidebar lists collections: Quiz_I, FMS, FMS Microservice, FMS_Reactive, FMS_Reactive_Mongo, New Collection, Quiz_microservices, Spring_Mongo_Reactive, Springboot_Mongodb, and WebfluxTutorial. The FMS Microservice collection is expanded, showing methods: POST Add inventory, POST search flights, POST Book Ticket, GET Search by pnr, and GET Search by mail. The FMS_Reactive collection is also expanded. The main workspace shows a POST request to 'Add inventory' at 'http://localhost:8765/FLIGHT-MICROSERVICE/api/flight'. The request body is raw JSON:

```
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 }
```

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"
```

View Find and replace Console Postbot Runner Start Proxy Cookies Vault Trash

3. Book Tickets

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, POST Book Ticket (which is selected), GET Search by pnr, and GET Search by mail. 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
}
```

View Find and replace Console Postbot Runner Start Proxy Cookies Vault Trash

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 interface with a test plan containing three thread groups:

- Thread Group (20 users):** Contains one 'HTTP Request' configuration. The 'Basic' tab shows the protocol as 'http', server name as 'localhost', and port number as '8082'. The 'Advanced' tab shows a 'Path' of '/api/flight/ticket/D4F30643'. Parameters are listed below.
- Thread Group (50 Users):** Contains one 'HTTP Request' configuration. The 'Basic' tab shows the protocol as 'http', server name as 'localhost', and port number as '8082'. The 'Advanced' tab shows a 'Path' of '/api/flight/ticket/D4F30643'. Parameters are listed below.
- Thread Group (100 Users):** Contains one 'HTTP Request' configuration. The 'Basic' tab shows the protocol as 'http', server name as 'localhost', and port number as '8082'. The 'Advanced' tab shows a 'Path' of '/api/flight/ticket/D4F30643'. Parameters are listed below.

The 'Parameters' section for the first thread group shows:

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

The screenshot shows the Apache JMeter interface with the title "Summary Report1.jmx". The left sidebar displays a tree view of the test plan, including "Test Plan", "Thread Group (20 users)" (selected), "Thread Group (50 Users)", and "Thread Group (100 Users)". Each group contains an "HTTP Request" and a "Summary Report" item. The main panel is titled "Summary Report" and contains fields for "Name" (Summary Report) and "Comments" (Write results to file / Read from file). A "Filename" input field is present with a "Browse..." button. Below these are buttons for "Log/Display Only" (Errors, Successes) and "Configure". A detailed table follows, showing performance metrics for each thread group. The table has columns: Label, # Samples, Average, Min, Max, Std. Dev., Error %, Throughput, Received KB/sec, Sent KB/sec, and Avg. Bytes. Data rows include "HTTP Request" and "TOTAL" for each group, with values matching the summary report.

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

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

- Title Bar:** Summary Report1.jmx (C:\Users\KJIT\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
- Icon Bar:** Includes icons for Test Plan, Thread Group, HTTP Request, Summary Report, View Results Tree, and others.
- Left Panel (Tree View):**
 - 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
- Center Panel (View Results Tree Configuration):**
 - Name: View Results Tree
 - Comments: Write results to file / Read from file
 - Filename:
 - Log/Display Only: Errors Successes
 - Search: Case sensitive Regular exp.
 - Text dropdown: Sampler result
 - Content pane: Shows 15 "HTTP Request" entries, each with a green checkmark icon.
- Bottom Panel:** Scroll automatically? Raw Parsed

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
 - 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:** A central dialog titled "View Results Tree" is open.
 - 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 with Case sensitive and Regular exp. checkboxes, and Search and Reset buttons.
 - Text Area: Shows a list of "Sampler result" entries, each preceded by a green checkmark and labeled "HTTP Request".
 - Bottom Buttons: Raw, Parsed, and Scroll automatically? checkboxes.
- Header:** 00:01:14 ⚠ 0 0/170

3. 100 USERS

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

- Thread Group (20 users): Contains an HTTP Request configuration.
- Thread Group (50 Users): Contains an HTTP Request configuration.
- Thread Group (100 Users): Contains an HTTP Request configuration, which is currently selected.

The selected HTTP Request configuration has the following settings:

- Basic** tab selected.
- Web Server** section: Protocol [http], Server Name or IP: localhost, Port Number: 8082.
- HTTP Request** section: Method: GET, Path: /api/flight/ticket/D4F30643. Checkboxes: Redirect Automatically (unchecked), Follow Redirects (checked), Use KeepAlive (checked), Use multipart/form-data (unchecked), Browser-compatible headers (unchecked).
- Parameters** tab selected. A table for sending 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 four thread groups: "Thread Group (20 users)", "Thread Group (50 Users)", "Thread Group (100 Users)", and a selected "Summary Report". The main panel is titled "Summary Report" and contains configuration fields: "Name: Summary Report", "Comments: Write results to file / Read from file", and a "Filename" input field with a "Browse..." button. Below these are two tables showing performance metrics. The first table has columns: Label, # Samples, Average, Min, Max, Std. Dev., Error %, Throughput, Received KB/sec, Sent KB/sec, and Avg. Bytes. It shows data for "HTTP Request" and "TOTAL". The second table is partially visible below it.

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

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

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

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 elements under "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 "View Results Tree" panel is open on the right, titled "View Results Tree". It has fields for "Name" (set to "View Results Tree") and "Comments". There are options to "Write results to file / Read from file" and a "Filename" input field with a "Browse..." button. Below these are "Search:" and "Text" dropdown fields, and "Case sensitive" and "Regular exp." checkboxes. There are also "Search" and "Reset" buttons. The main area shows a tree view with "Sampler result" expanded, displaying multiple "HTTP Request" entries, each with a green checkmark icon. At the bottom, there are "Raw" and "Parsed" tabs and a checkbox for "Scroll automatically?".