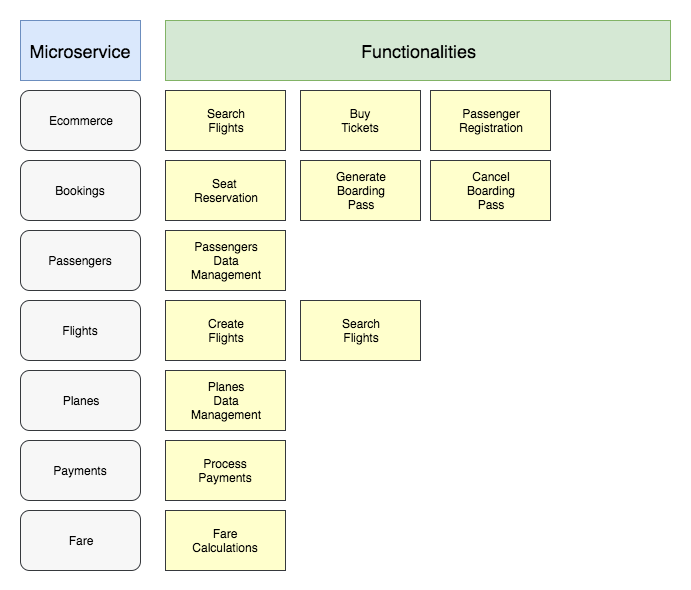
 FLIGHT MANAGEMENT SYSTEM

**PREPARED FOR**

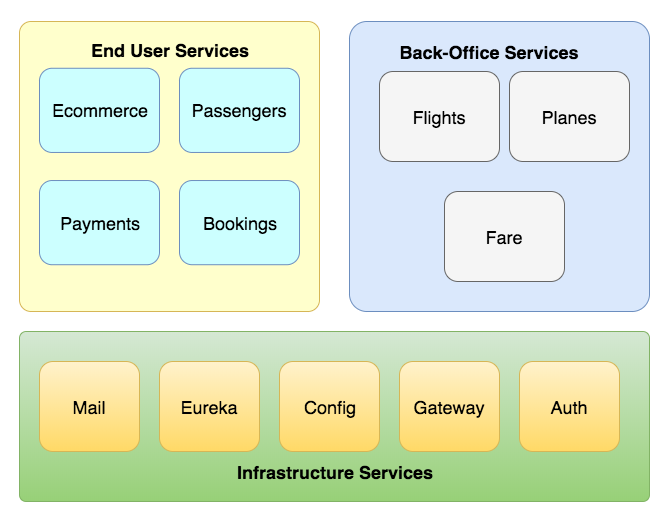
JEE Cloud Training Program Capgemini Technology Services

**Abstract:**

The Flight Management System is a Java-based booking solution for flight tickets. It consolidates data provided by different airline carriers and hence provides the user details and rates in real-time. Travelers may want to make changes in their bookings. The application allows them to book, cancel, view and update their bookings with ease. Other than this, it eases the management of bookings too. All the bookings, flights, schedules and routes can be viewed, added and modified on a single application by the administrator.



### **Solution diagram -** The following diagram illustrates the whole solution, which we will implement in the following chapters:



As we can see, there are different kinds of components. Some components will be exposed through the Gateway for end users, in our case, our customers. There is a category which the company users will use to register flights, for instance, where these microservices will be exposed on Gateway as well.

The infrastructure category will not be exposed over the internet, except the Gateway service. These services help the solution infrastructure and should be not exposed because there is sensitive data in there.

# The airline Bookings microservice

The airline Bookings microservice is a standard Spring Boot Application. There are some interactions with other services, such as the flights microservice.

These interactions were created using Hystrix to bring some desired behaviors, such as fault-tolerance and resilience, to the airline Bookings microservice.

# The airline Payments microservice

The Airline Payments is a microservice that gives payments confirmation for our Airline Ticket System.

# Turbine server

There are some integrations in our microservices group; the **Bookings microservice** calls the **Fares microservice** and the **Passengers microservice**, these integrations are done using Hystrix to make it more resilient and fault tolerant.

However, in the microservices world, there are several instances of service. This will require us to aggregate the Hystrix command metrics by instance. Managing the instances panel by panel is not a good idea. The Turbine server helps developers in this context.

By default, Turbine pulls metrics from servers run by Hystrix, but it is not recommended for cloud environments because it can consume high values of network bandwidth and it will increase the traffic costs. We will use Spring Cloud Stream RabbitMQ to push metrics to Turbine via the Advanced Message Queuing Protocol (AMQP).

# Mail microservice

The name is self-explanatory, this component will be responsible for sending emails. We will not configure an SMTP (Simple Mail Transfer Protocol) server, we will use SendGrid.

SendGrid is an SaaS (Software as a Service) service for emails, we will use this service to send emails to our Airline Ticket System. There are some triggers to send email, for example, when the user creates a booking and when the payment is accepted.

Our Mail microservice will listen to a queue. Then the integration will be done using the message broker. We choose this strategy because we do not need the feature that enables us to answer synchronously. Another essential characteristic is the retry policy when the communication is broken. This behavior can be done easily using the message strategy.

We are using RabbitMQ as a message broker. For this project, we will use RabbitMQ Reactor, which is a reactive implementation of RabbitMQ Java client.

# Authentication microservice

Security is essential for microservices applications, especially because of the distributed characteristics.

On the microservices architectural style, usually, there is a service that will act as an authentication service. It means this service will authenticate the requests in our microservices group.

Spring Cloud Security provides a declarative model to help developers enable security on applications. There is support for commons patterns such as OAuth 2.0. Also, Spring Boot Security enables Single Sign-On (SSO).

Spring Boot Security also supports relay SSO tokens integrating with Zuul proxy. It means the tokens will be passed to downstream microservices.

For our architecture, we will use the OAuth 2.0 and JWT patterns, both integrate with Zuul proxy.