

Spring Boot Microservices Professional eCommerce Masterclass

Faisal Memon (EmbarkX)

Instructor: Faisal Memon

Company: EmbarkX.com

1. Personal Use Only

The materials provided in this course, including but not limited to PDF presentations, are intended for your personal use only. They are to be used solely for the purpose of learning and completing this course.

2. No Unauthorized Sharing or Distribution

You are not permitted to share, distribute, or publicly post any course materials on any websites, social media platforms, or other public forums without prior written consent from the instructor.

3. Intellectual Property

All course materials are protected by copyright laws and are the intellectual property of Faisal Memon and EmbarkX. Unauthorized use, reproduction, or distribution of these materials is strictly prohibited.

4. Reporting Violations

If you become aware of any unauthorized sharing or distribution of course materials, please report it immediately to [\[embarkxofficial@gmail.com\]](mailto:embarkxofficial@gmail.com).

5. Legal Action

We reserve the right to take legal action against individuals or entities found to be violating this usage policy.

Thank you for respecting these guidelines and helping us maintain the integrity of our course materials.

Contact Information

embarkxofficial@gmail.com

www.embarkx.com

Basics of API

Faisal Memon (EmbarkX)

API is Application Programming Interface

API is Application
Programming Interface

What does it mean?

Set of rules and protocols that allow one software application to interact and communicate with another

Restaurant Menu?

Why are API's needed?

- *Share data*
- *Speed up development*
- *Extend the reach and functionality of software*

Popular API's

- *Google Maps API*
- *Twitter API*
- *Facebook's Graph API*
- *Amazon S3 API*

Types of API

→ *Internal API's*

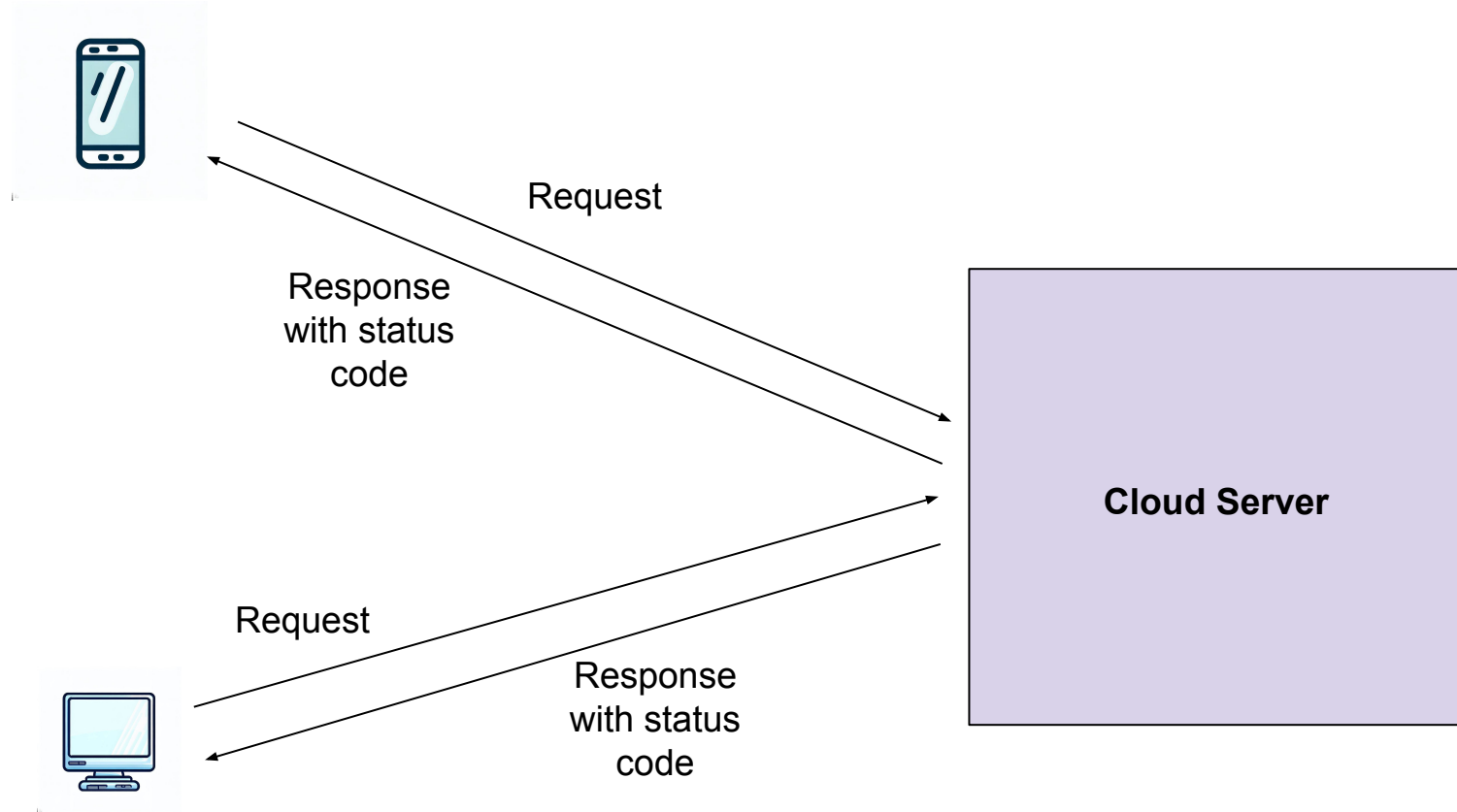
→ *External API's*

→ *Partner API's*

Status Codes in API

Faisal Memon (EmbarkX)

Need for Status Codes



Classification of Status Codes

→ *1xx (Informational)*

→ *2xx (Successful)*

→ *3xx (Redirection)*

→ *4xx (Client Error)*

→ *5xx (Server Error)*

Commonly used Status Codes

→ 200 OK

→ 201 Created

→ 204 No Content

→ 301 Moved Permanently

→ 400 Bad Request

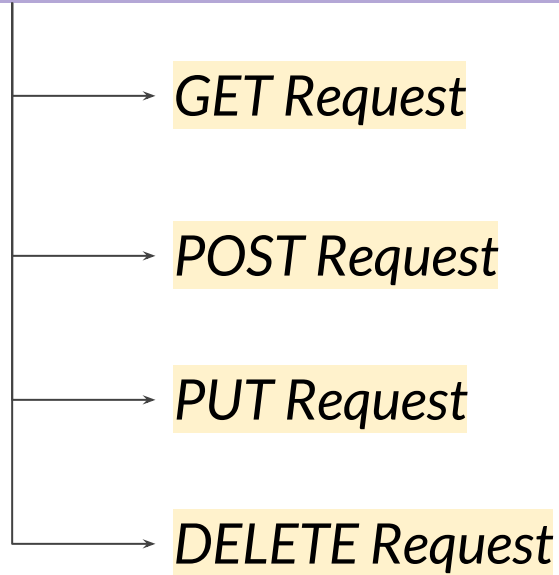
Commonly used Status Codes

- *401 Unauthorized*
- *403 Forbidden*
- *404 Not Found*
- *500 Internal Server Error*

Types of API Requests

Faisal Memon (EmbarkX)

Types of API requests



GET Request

- *Retrieve or GET resources from server*
- *Used only to read data*

POST Request

→ *Create resources from server*

PUT Request

→ *Update existing resources on Server*

DELETE Request

→ *Used to DELETE resources from Server*

What is a Web Framework?

Faisal Memon (EmbarkX)

Why do you need Web Framework?

- *Websites have a lot in common*
- *Security, Databases, URLs, Authentication....more*
- *Should you do this everytime from scratch?*

Think of building a House

- *You would need Blueprint and Tools*
- *That's how web development works*
- *Developers had to build from scratch*

What if...

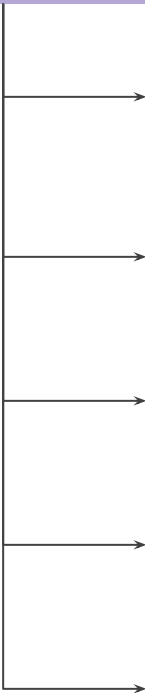
- *You could have prefabricated components?*
- *Could you assemble faster?*
- *Could you reduce errors?*
- *Would that make you fast?*

This is what a **Web
Framework** does!

What is Web Framework

Web Framework is nothing but collection of tools and modules that is needed to do standard tasks across every web application.

Popular Web Frameworks



Spring Boot (Java)

Django (Python)

Flask (Python)

Express (JavaScript)

Ruby on Rails (Ruby)

Introduction to Spring Framework

Faisal Memon (EmbarkX)

Spring takes away the
hassle

Features of Spring Framework



Inversion of Control (IoC)

A diagram consisting of a vertical line extending downwards from the bottom center of the title box, followed by a horizontal line extending to the right, ending in an arrowhead pointing towards the text 'Inversion of Control (IoC)'.

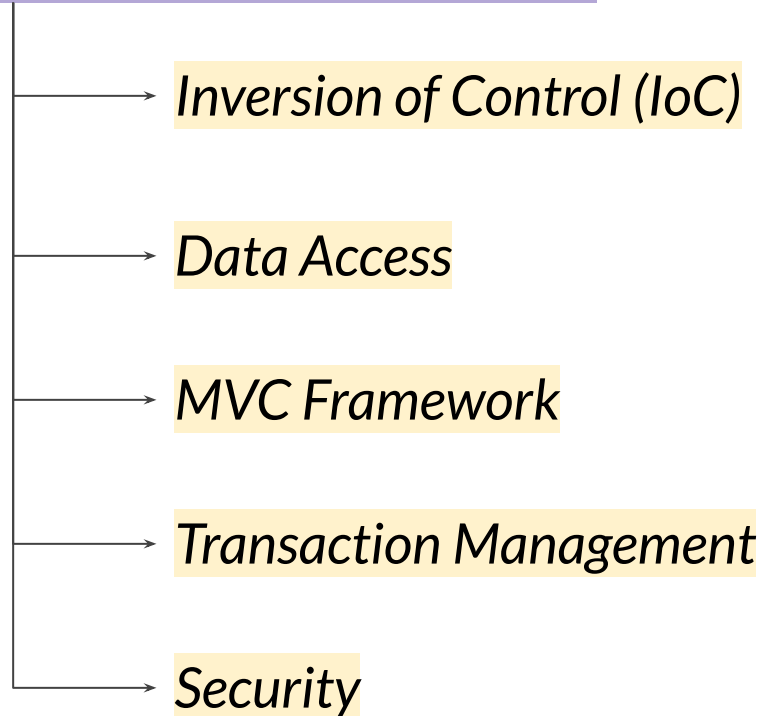
```
public interface MessageService {  
    String getMessage();  
}
```

```
public class EmailService implements MessageService  
{  
    public String getMessage() {  
        return "Email message";  
    }  
}
```



```
public class SMSClient {  
    private MessageService messageService;  
  
    public SMSClient() {  
        this.messageService = new EmailService(); //  
Dependency created within SMSClient  
    }  
  
    public void sendMessage() {  
        String message = messageService.getMessage();  
        // Logic to send SMS using the message  
    }  
}
```

Features of Spring Framework



Features of Spring Framework



```
graph LR; A[Features of Spring Framework] --> B[Testing Support]; A --> C[Internationalization (i18n) and Localization (l10n)];
```

Testing Support

Internationalization (i18n) and Localization (l10n)

History

- *Initially developed by Rod Johnson in 2002*
- *First version released in March 2004*
- *Since then, major developments and versions released*

What is Spring Boot?

Faisal Memon (EmbarkX)

What is Spring Boot?

Open-source, Java-based framework used to create stand-alone, production-grade Spring-based Applications

Spring

VS

Spring Boot

Lots of steps involved in setting up, configuration, writing boilerplate code, deployment of the app

Offers a set of pre-configured components or defaults, and eliminating the need for a lot of boilerplate code that was involved in setting up a Spring application

Spring boot = **Spring Framework**
+
Prebuilt Configuration
+
Embedded Servers

Components of Spring Boot

- *Spring Boot Starters*
- *Auto Configuration*
- *Spring Boot Actuator*
- *Embedded Server*
- *Spring Boot DevTools*

Advantages of Spring Boot

- *Stand alone and Quick Start*
- *Starter code*
- *Less configuration*
- *Reduced cost and application development time*

Why do developers love Spring Boot?

- *Java based*
- *Fast, easy*
- *Comes with embedded server*
- *Various plugins*
- *Avoids boilerplate code and configurations*

Spring Boot Architecture

Faisal Memon (EmbarkX)

Following are the different tiers

→ *Presentation layer*

→ *Service layer*

→ *Data access layer*

Presentation Layer

Presentation layer presents the data and the application features to the user. This is the layer where in all the controller classes exist.

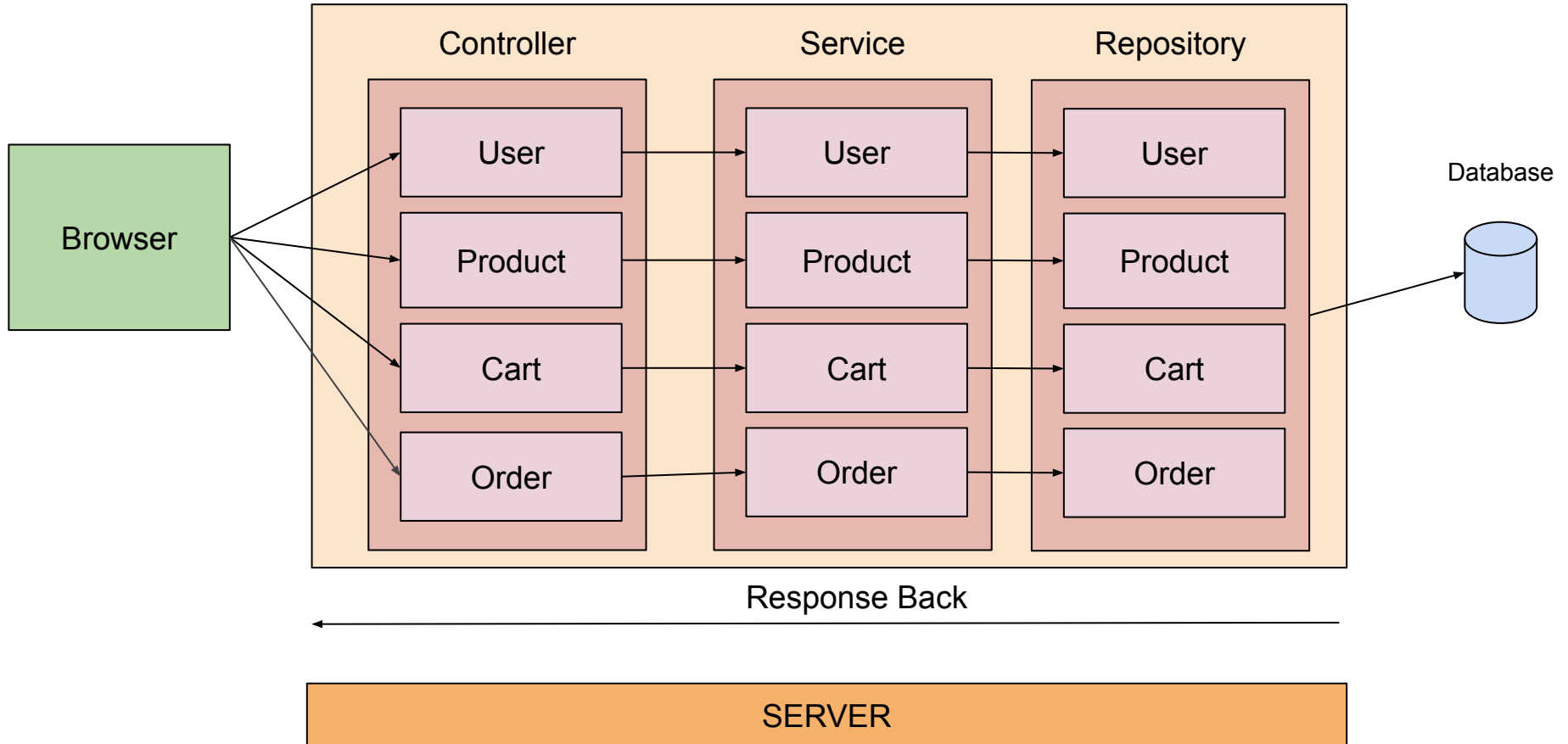
Service Layer

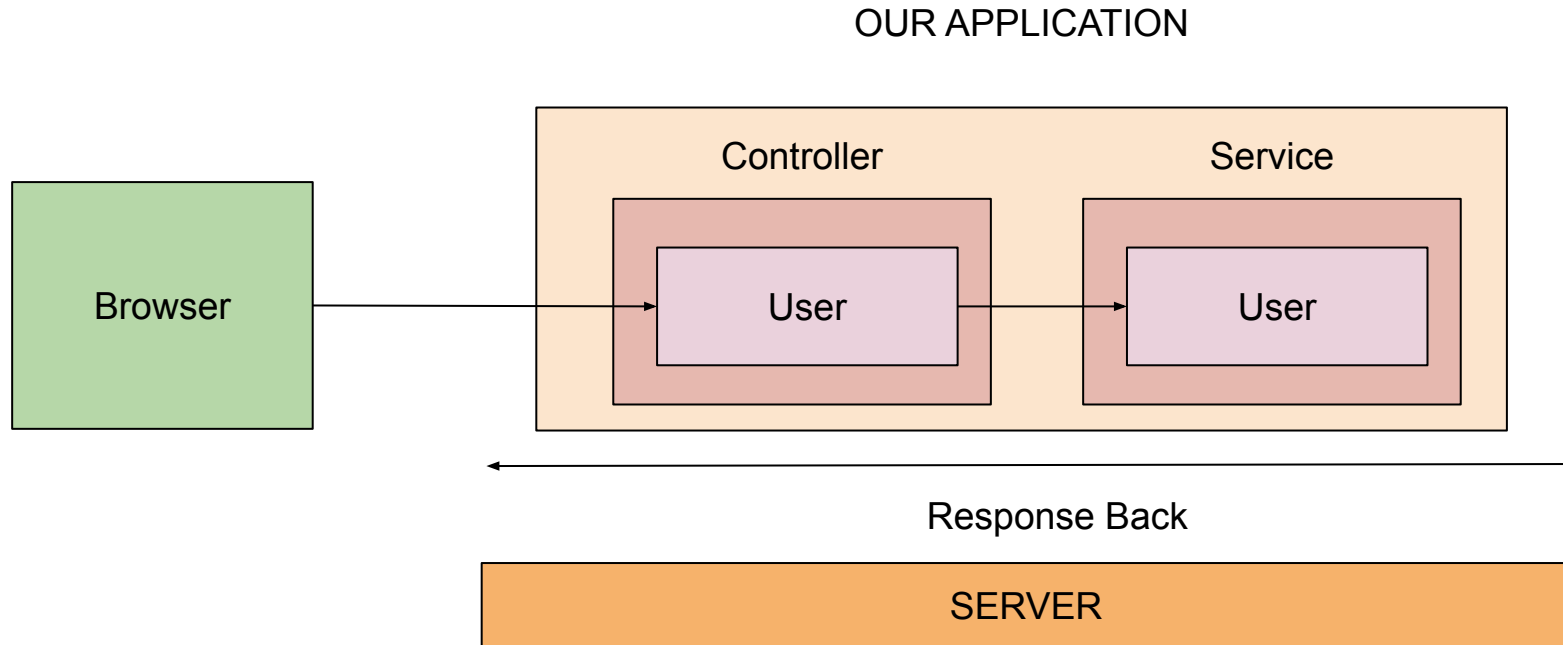
Service layer is where business logic resides in the application. Tasks such as evaluations, decision making, processing of data is done at this layer.

Data Access Layer

Data access layer is the layer where all the repository classes reside.

OUR APPLICATION

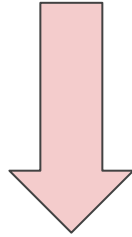




What is JPA?

Faisal Memon (EmbarkX)

```
public class User {  
    private Long id;  
    private String firstName;  
    private String lastName;  
}
```



id	firstName	lastName
1	Michael	Cole

Advantages of using JPA

- *Easy and Simple*
- *Makes querying easier*
- *Allows to save and update objects*
- *Easy integration with Spring Boot*

Easy and Simple

Without JPA (Manual SQL)

```
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/ecom", "user", "password");
PreparedStatement stmt = conn.prepareStatement("SELECT * FROM users WHERE id = ?");
stmt.setLong(1, 1);
ResultSet rs = stmt.executeQuery();

if (rs.next()) {
    User user = new User();
    user.setId(rs.getLong("id"));
    user.setName(rs.getString("name"));
    System.out.println(user);
}
```

With JPA (Much Simpler!)

```
User user = userRepository.findById(1L).orElse(null);
System.out.println(user);
```

Presentation Layer

Presentation layer presents the data and the application features to the user. This is the layer where in all the controller classes exist.

Service Layer

Service layer is where business logic resides in the application. Tasks such as evaluations, decision making, processing of data is done at this layer.

Data Access Layer

Data access layer is the layer where all the repository classes reside.

```
findAll();  
getUser(int id);  
updateUser(User user);  
deleteUser(User user);
```

Syntax

JpaRepository<entity-name>, <primary-key-type>

Introduction to Spring Boot Actuator

Faisal Memon (EmbarkX)

What is it?

Provides built-in production-ready features to monitor and manage your application

Why is it important?

Gives you the ability to monitor and manage your applications

Features

- *Built in endpoints*
- *Ability to view real time metrics*
- *Customizable*

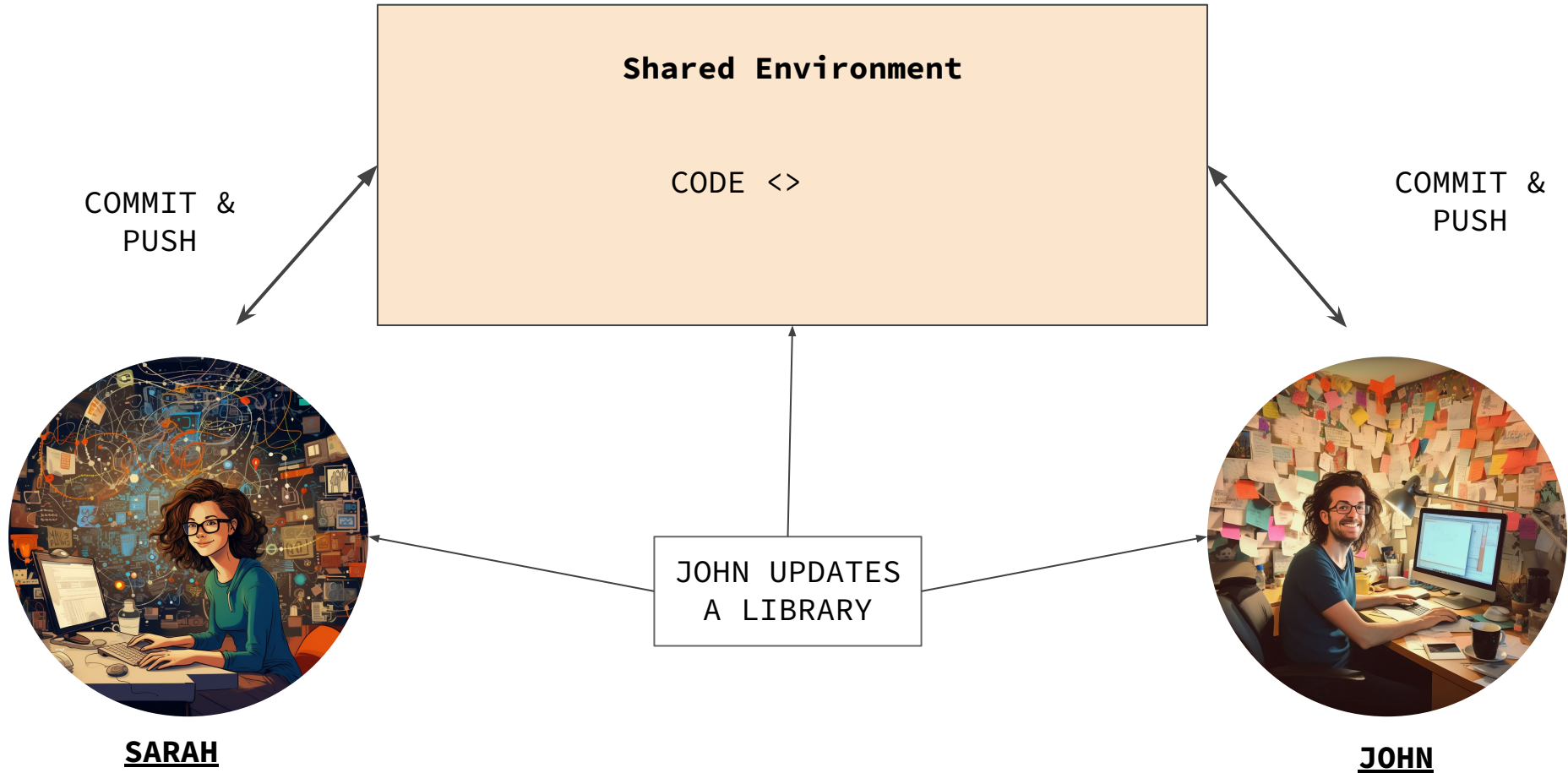
Understanding Actuator Endpoints

Faisal Memon (EmbarkX)

Endpoint	Purpose
/health	Shows application health information, useful for checking the status of the application, such as database connectivity, disk space, and custom health checks.
/info	Displays arbitrary application information, commonly used to display application version, git commit information, etc.
/metrics	Shows 'metrics' information that allows you to understand the performance and behavior of your running application.
/loggers	Allows you to query and modify the logging level of your application's loggers.
/beans	Provides a complete list of all the Spring beans in your application
/shutdown	Allows your application to be gracefully shut down

Introduction to Docker

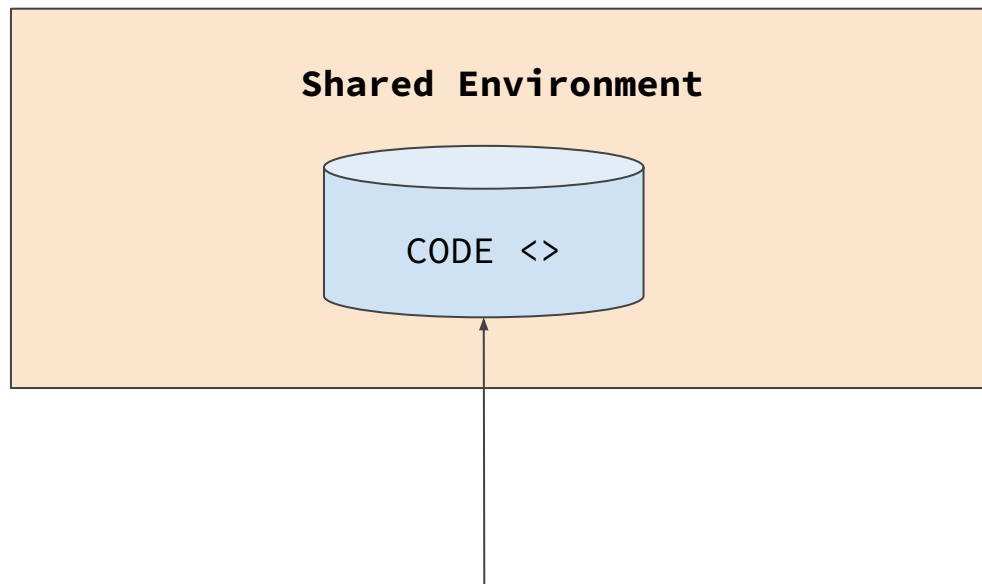
Faisal Memon (EmbarkX)



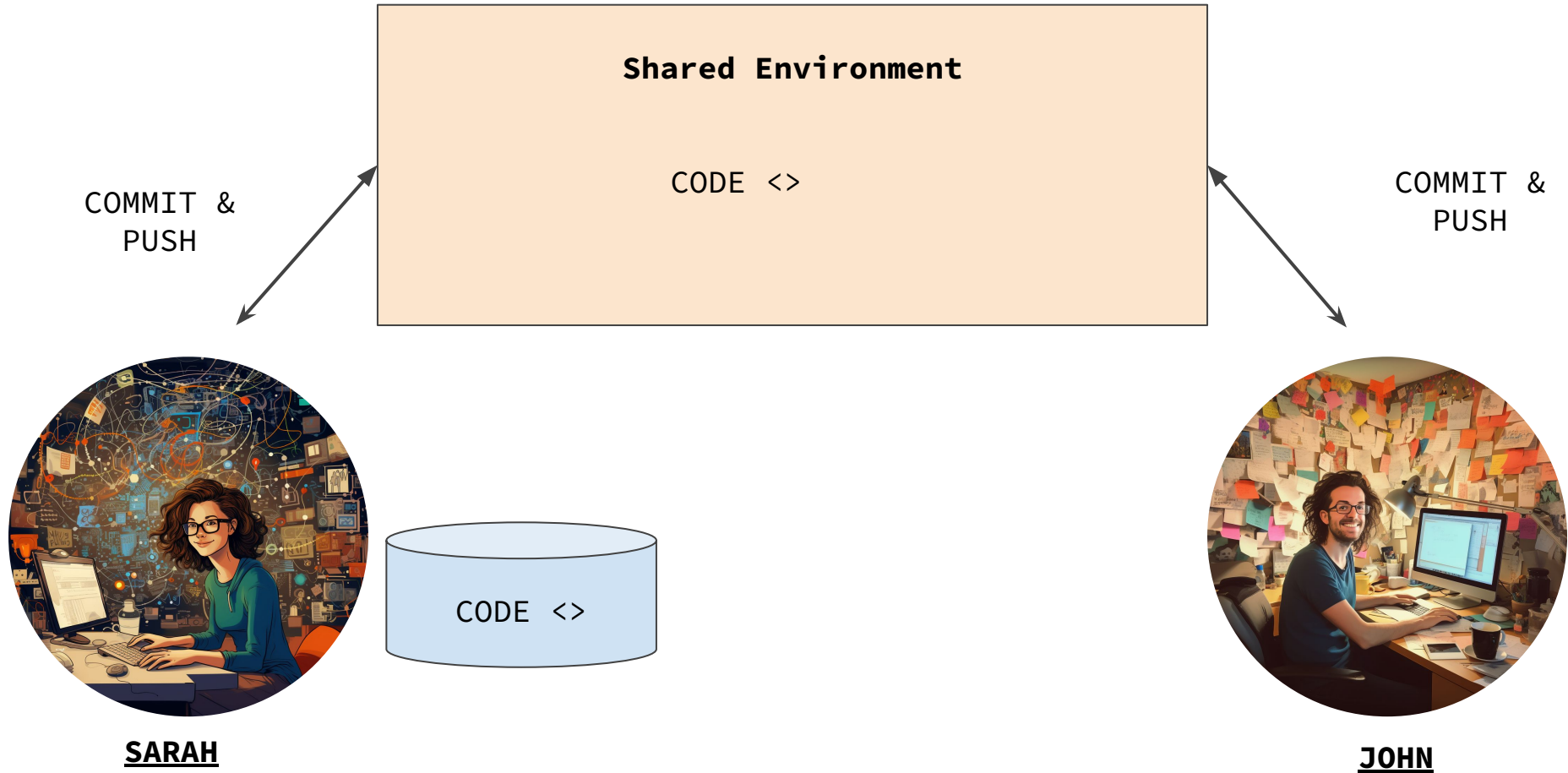
Welcome Docker

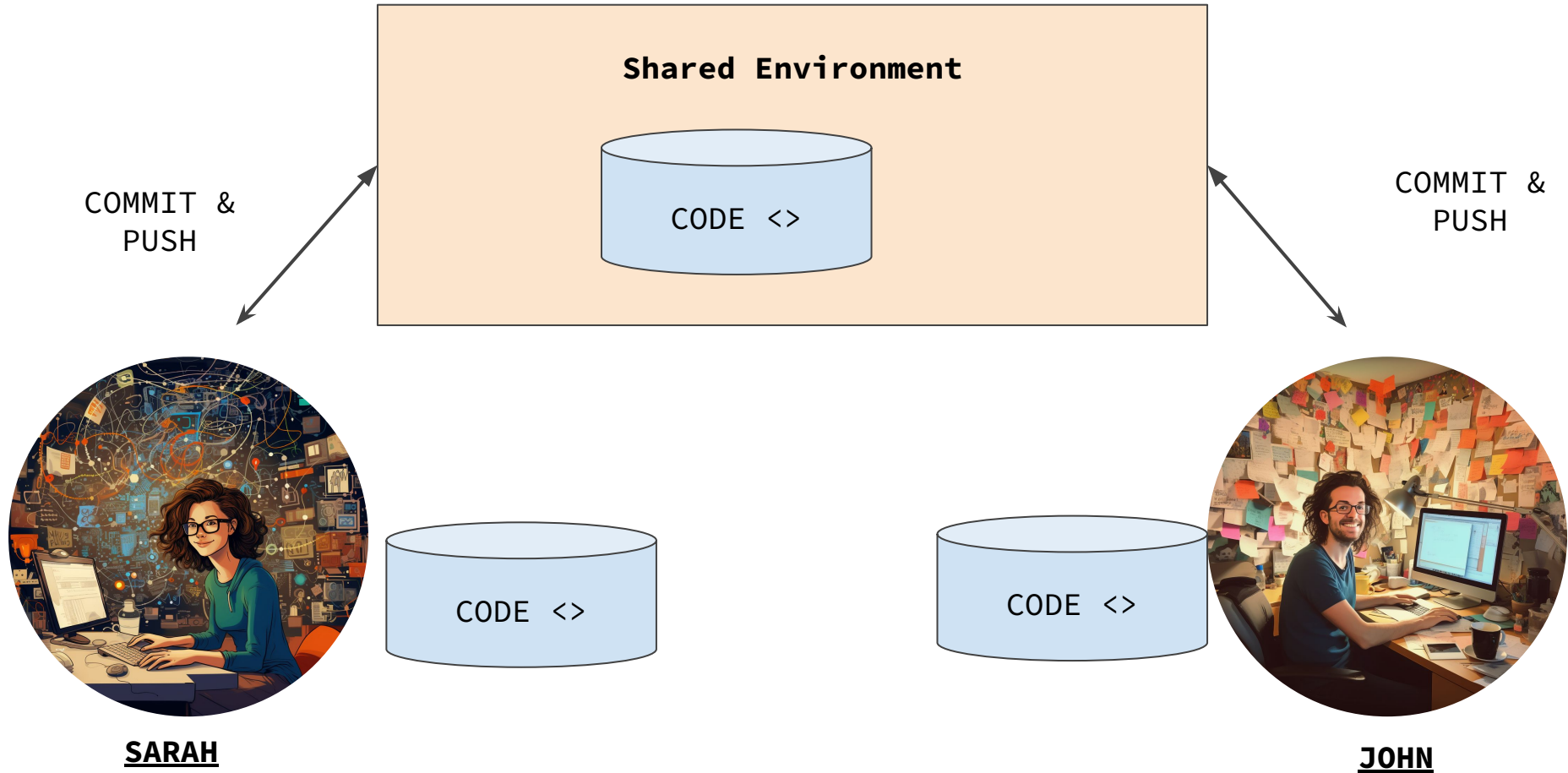
Shared Environment

CODE <>



Includes the **application code**, its **dependencies**, and the required **environment configuration**

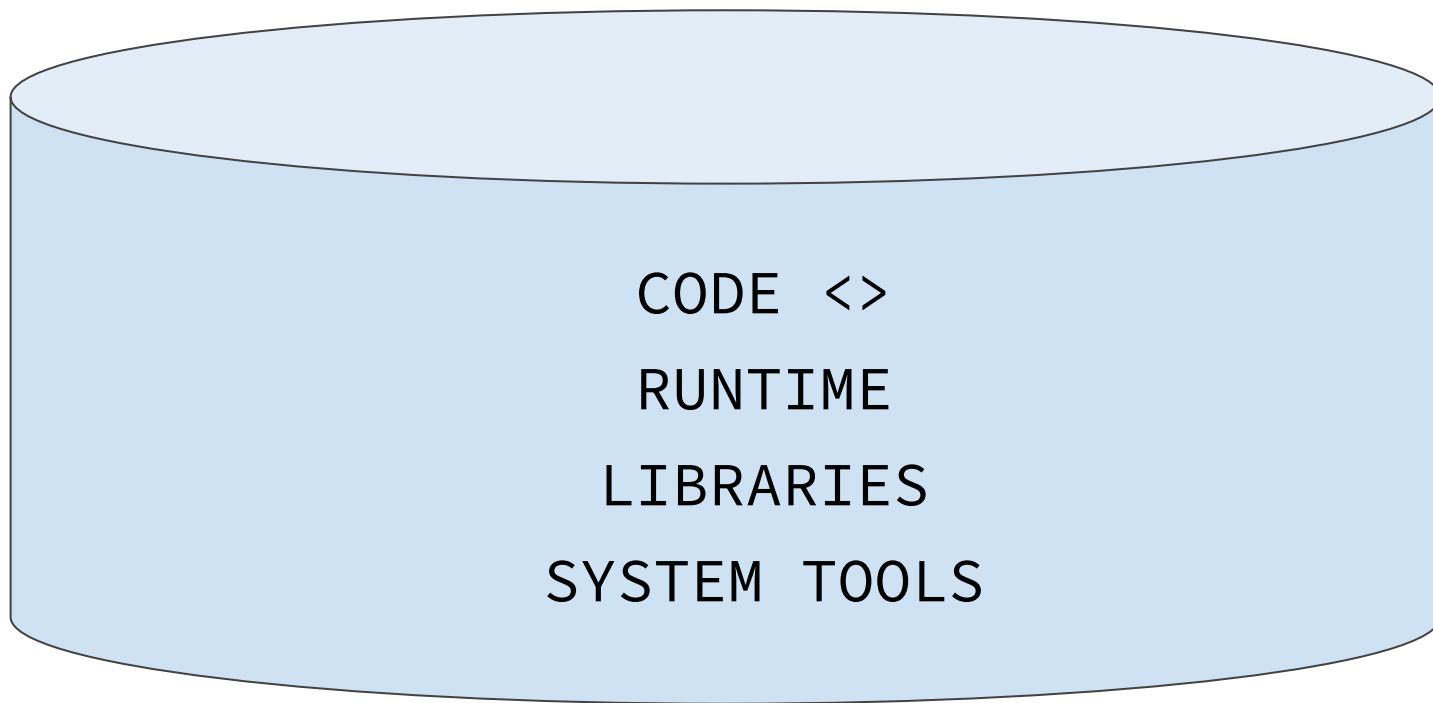




Let's Define Docker

Docker is an open-source platform that allows you to automate the deployment, scaling, and management of applications using containerization

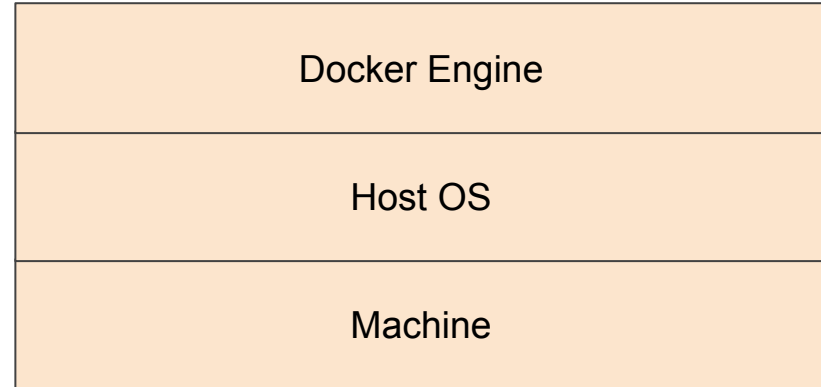
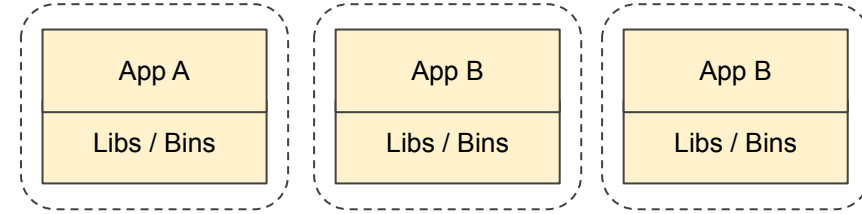
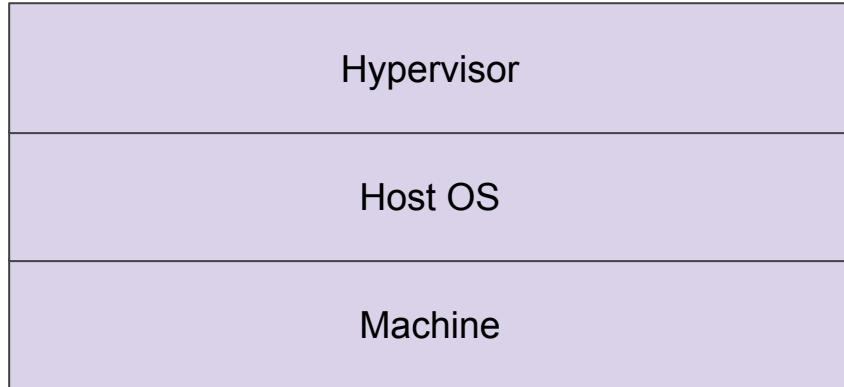
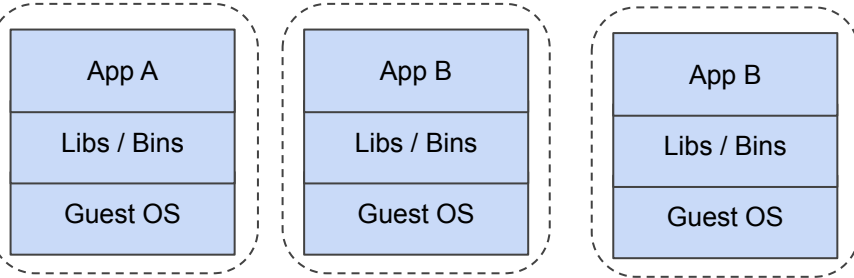
Docker Container



Virtual Machines

- *VMs act like separate computers inside your computer*
- *Each virtual machine behaves like a separate computer*
- *Virtual machines are created and managed by virtualization software*
- *They provide a flexible and scalable way to utilize hardware resources*

Docker over VM's

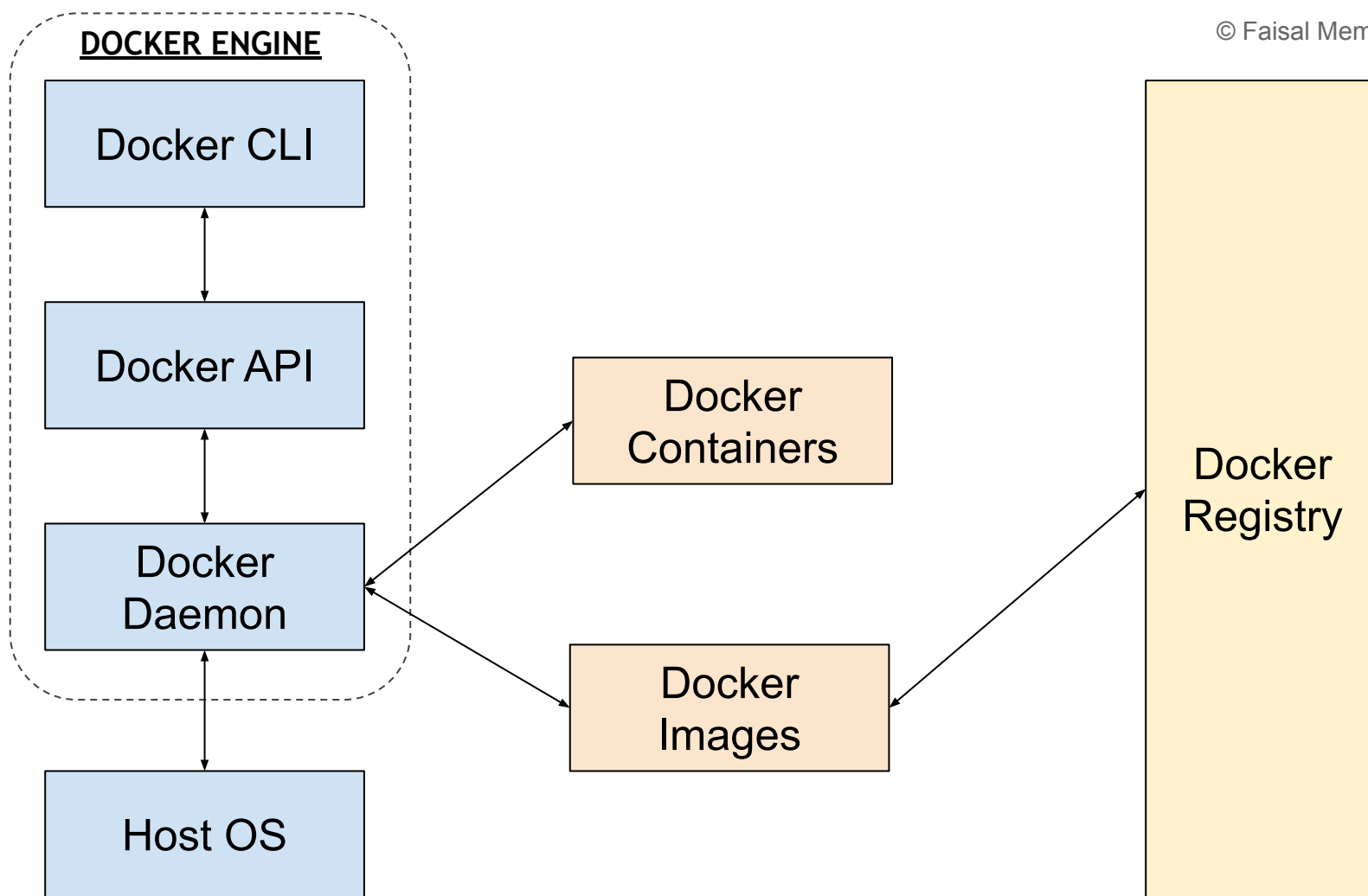


Parameters	Virtual Machines (VMs)	Docker Containers
<i>Size</i>	Relatively large and resource-intensive	Lightweight and resource-efficient
<i>Startup Time</i>	Longer boot time as full OS needs to start	Almost instant startup as no OS boot required
<i>Resource Utilization</i>	Utilizes more system resources (CPU, memory)	Utilizes fewer system resources
<i>Isolation</i>	Strong isolation between VMs	Isolated, but shares host OS kernel
<i>Portability</i>	Portable, but requires OS compatibility	Highly portable, independent of host OS

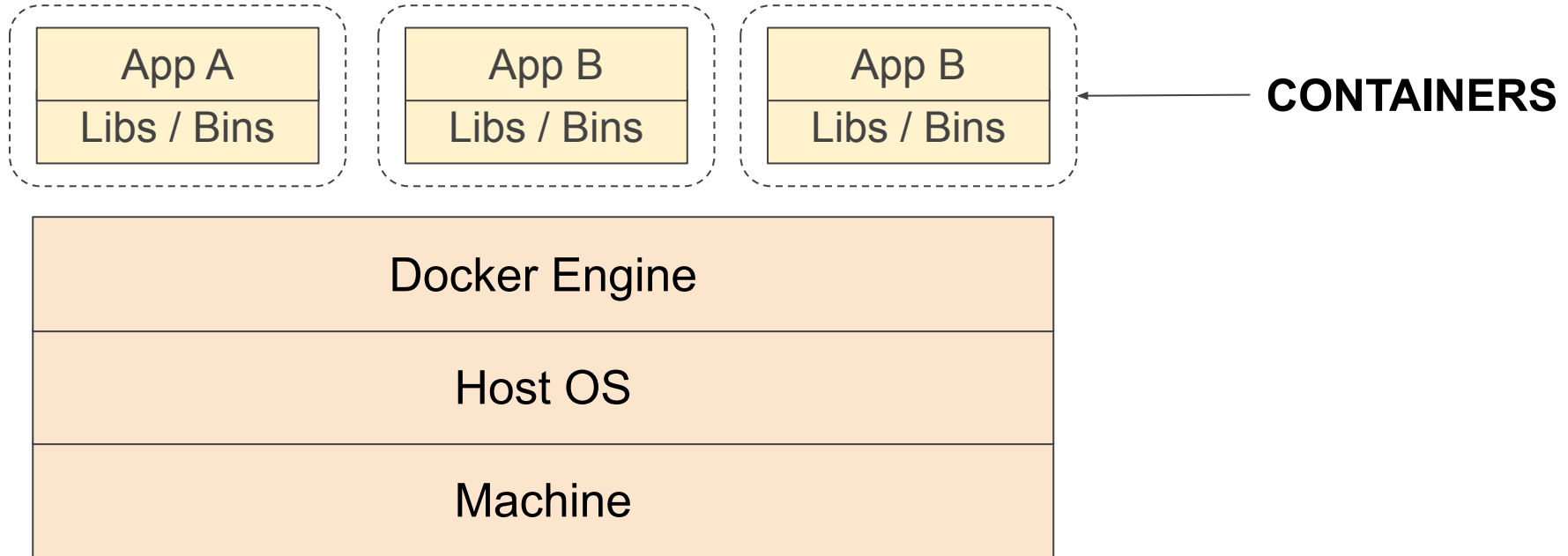
Parameters	Virtual Machines (VMs)	Docker Containers
<i>Scalability</i>	Scaling requires provisioning of new VMs	Easy to scale by creating more containers
<i>Ecosystem</i>	VM-specific tools and management frameworks	Docker ecosystem with extensive tooling
<i>Development Workflow</i>	Slower setup and provisioning process	Faster setup and dependency management
<i>Deployment Efficiency</i>	More overhead due to larger VM size	Efficient deployment with smaller container

Docker Architecture

Faisal Memon (EmbarkX)



Docker



Concepts in Docker

Faisal Memon (EmbarkX)

- **Images**: Docker images are templates that define the container and its dependencies.
- **Containers**: Containers are runtime environments created from Docker images.
- **Docker Engine**: The Docker Engine is the runtime that runs and manages containers

→ **Dockerfile**: A Dockerfile is a file that contains instructions to build a Docker image.

→ **Docker Hub**: Docker Hub is a cloud-based registry that hosts a vast collection of Docker images

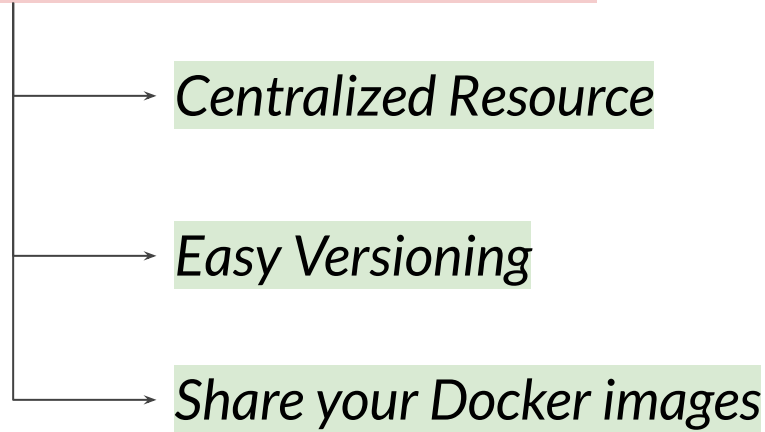
Docker Registry

Faisal Memon (EmbarkX)

What is it?

Docker Registry is a storage and distribution system for named Docker images

Importance of Docker Registry



Containerizing Our Spring Boot Application

Faisal Memon (EmbarkX)

Dockerfile

```
FROM openjdk:11
VOLUME /tmp
ADD target/my-app.jar my-app.jar
EXPOSE 8080
ENTRYPOINT [ "java", "-jar", "/my-app.jar" ]
```

Dockerfile

```
FROM openjdk:11
```

```
VOLUME /tmp
```

```
ADD target/my-app.jar my-app.jar
```

```
EXPOSE 8080
```

```
ENTRYPOINT [ "java", "-jar", "/my-app.jar" ]
```

Dockerfile

```
FROM openjdk:11
```

```
VOLUME /tmp
```

```
ADD target/my-app.jar my-app.jar
```

```
EXPOSE 8080
```

```
ENTRYPOINT [ "java", "-jar", "/my-app.jar" ]
```

Dockerfile

FROM openjdk:11

VOLUME /tmp

ADD target/my-app.jar my-app.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "/my-app.jar"]

Dockerfile

FROM openjdk:11

VOLUME /tmp

ADD target/my-app.jar my-app.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "/my-app.jar"]

Dockerfile

FROM openjdk:11

VOLUME /tmp

ADD target/my-app.jar my-app.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "/my-app.jar"]

How does the process work?

- *Cloud Native Buildpacks*
- *Spring Boot Maven Plugin*
- *Layering*
- *Paketo Buildpacks*
- *Result*

Advantages

- *No Dockerfile Needed*
- *Sensible Defaults*
- *Consistent Environment*
- *Security*
- *Layering & Efficiency*

Advantages

→ *Ease of use*

Docker Commands

Faisal Memon (EmbarkX)

Docker Commands

→ `docker pull <image>`

→ `docker push <username/image>`

→ `docker run -it -d -p <host-port>:<container-port>
--name <name> <image>`

→ `docker stop <container-id/container-name>`

→ `docker start <container-id/container-name>`

Docker Commands

→ `docker rm <container-id/container-name>`

→ `docker rmi <image-id/image-name>`

→ `docker ps`

→ `docker ps -a`

→ `docker images`

Docker Commands

→ `docker exec -it <container-name/container-id> bash`

→ `docker build -t <username/image> .`

→ `docker logs <container-name/container-id>`

→ `docker inspect <container-name/container-id>`

What Is PostgreSQL and Why Use It?

Faisal Memon (EmbarkX)

What is it?

*PostgreSQL is an object-relational database management system
(ORDBMS)*

Why is PostgreSQL popular?

- *SQL Compliance*
- *Extensibility*
- *Performance*
- *Strong Community Support*
- *Data Integrity*

Why PostgreSQL over H2?

- *Scalability*
- *Feature Set*
- *Ecosystem and Tools*
- *Durability*

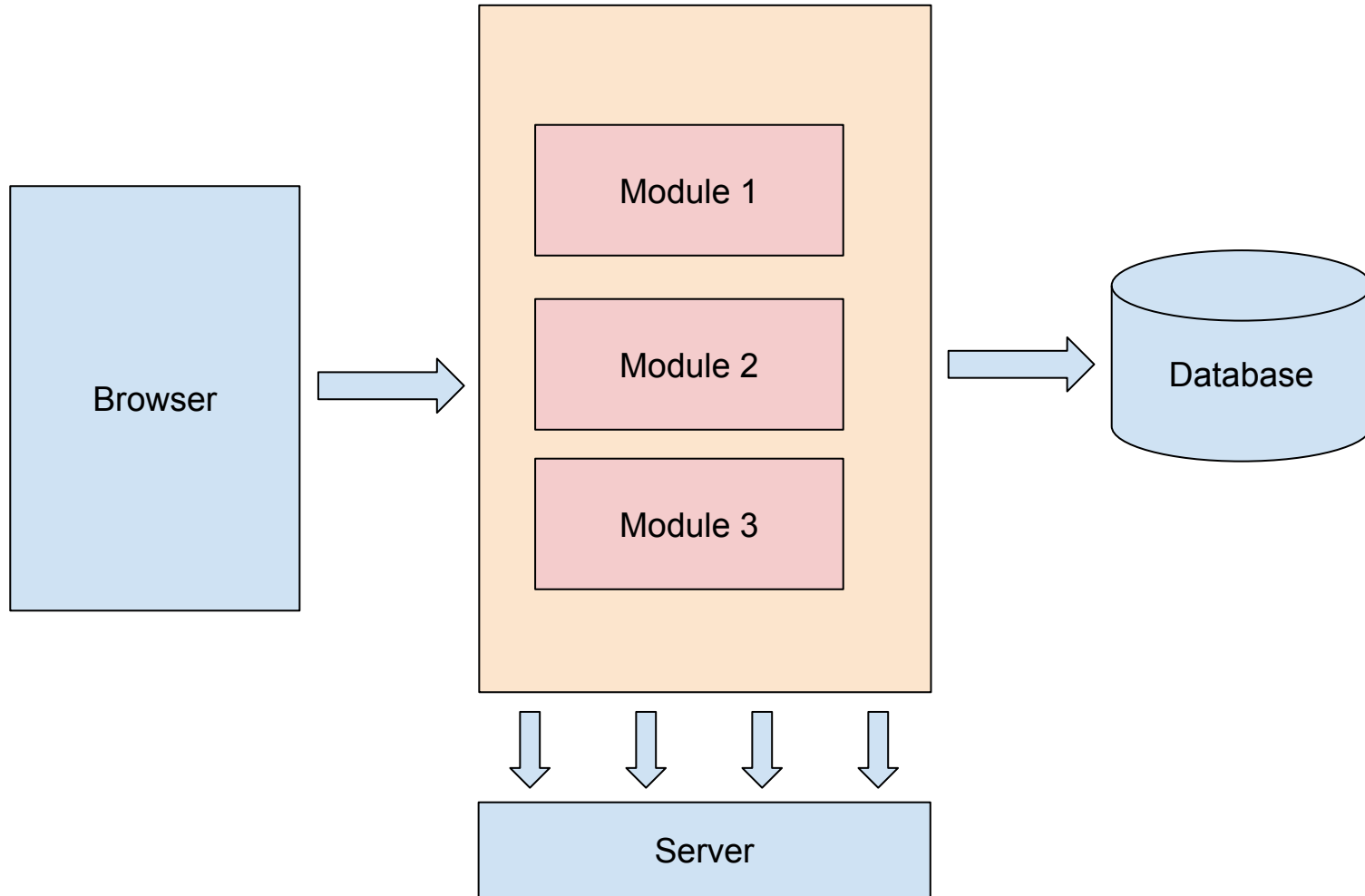
Before Microservices

Faisal Memon (EmbarkX)

World Before **Microservices**

Monolithic has everything
unified

Monolithic architecture is a design where all the components of an application are **interconnected** and **interdependent**



Problems with Monolithic Architecture

Faisal Memon (EmbarkX)

Problems

- *Difficult to Implement Changes*
- *Lack of Scalability*
- *Long-term Commitment to a Single Technology Stack*
- *Application Complexity and Its Effect on Development and Deployment*
- *Slowing Down of IDEs*

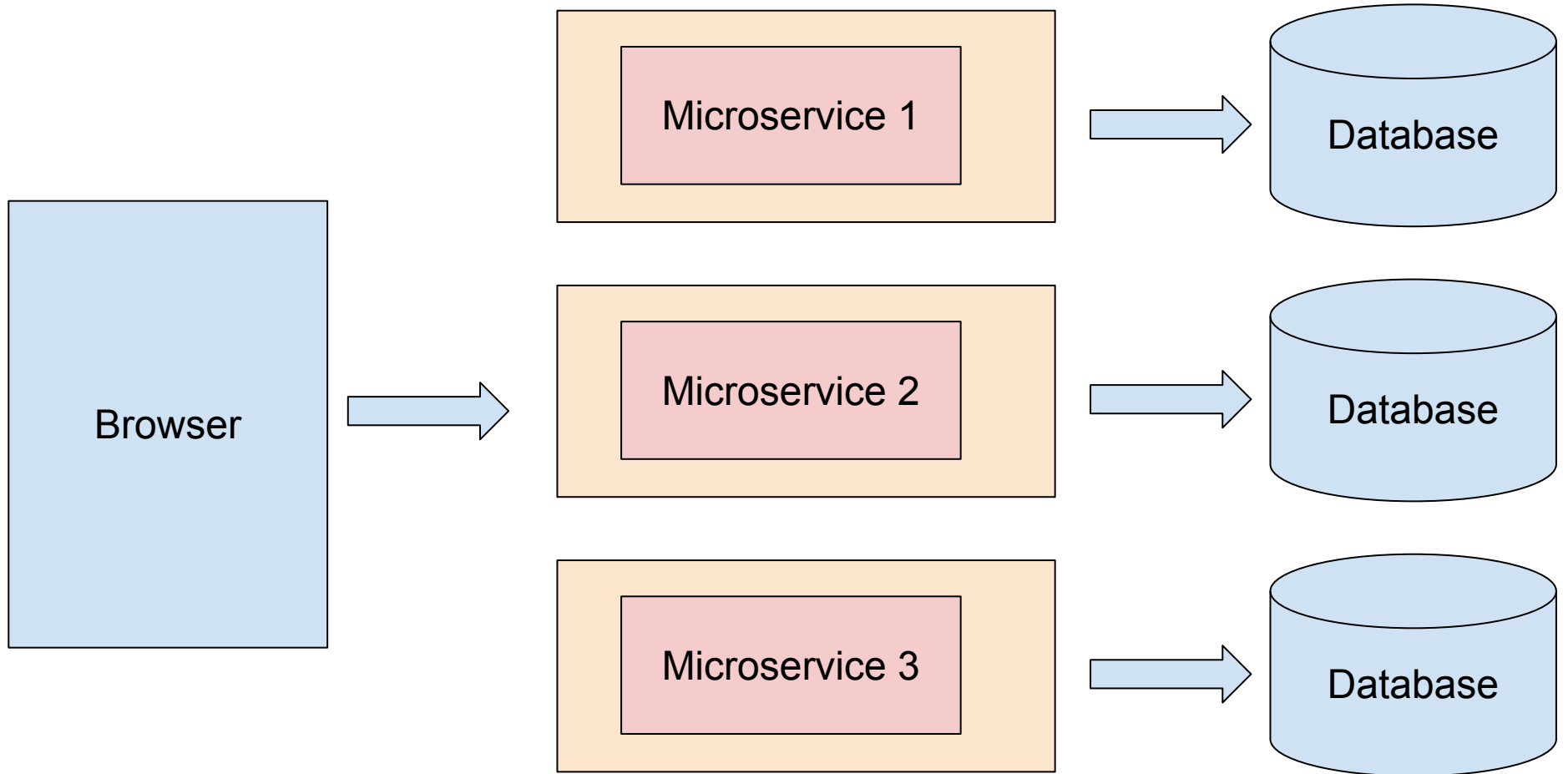
Problems

- *Increased Application Start Time*
- *Large Project Size*
- *Deploying for Small Changes*
- *Team Collaboration and Autonomy*

What are Microservices and Why do we need them?

Faisal Memon (EmbarkX)

Microservices structures an application as a collection of small autonomous services



Principles of Microservices

- *Single Responsibility*
- *Independence*
- *Decentralization*
- *Failure Isolation*
- *Continuous Delivery/Deployment*

Overcoming Monolithic Architecture Challenges with Microservices

Faisal Memon (EmbarkX)

How Microservices Address the Problems of Monolithic Architecture

→ *Scalability*

→ *Flexibility*

→ *Simplicity*

Case Study: Netflix

Principles of Microservices Architecture

Faisal Memon (EmbarkX)

Principles

→ *Single Responsibility*

→ *Bounded Context*

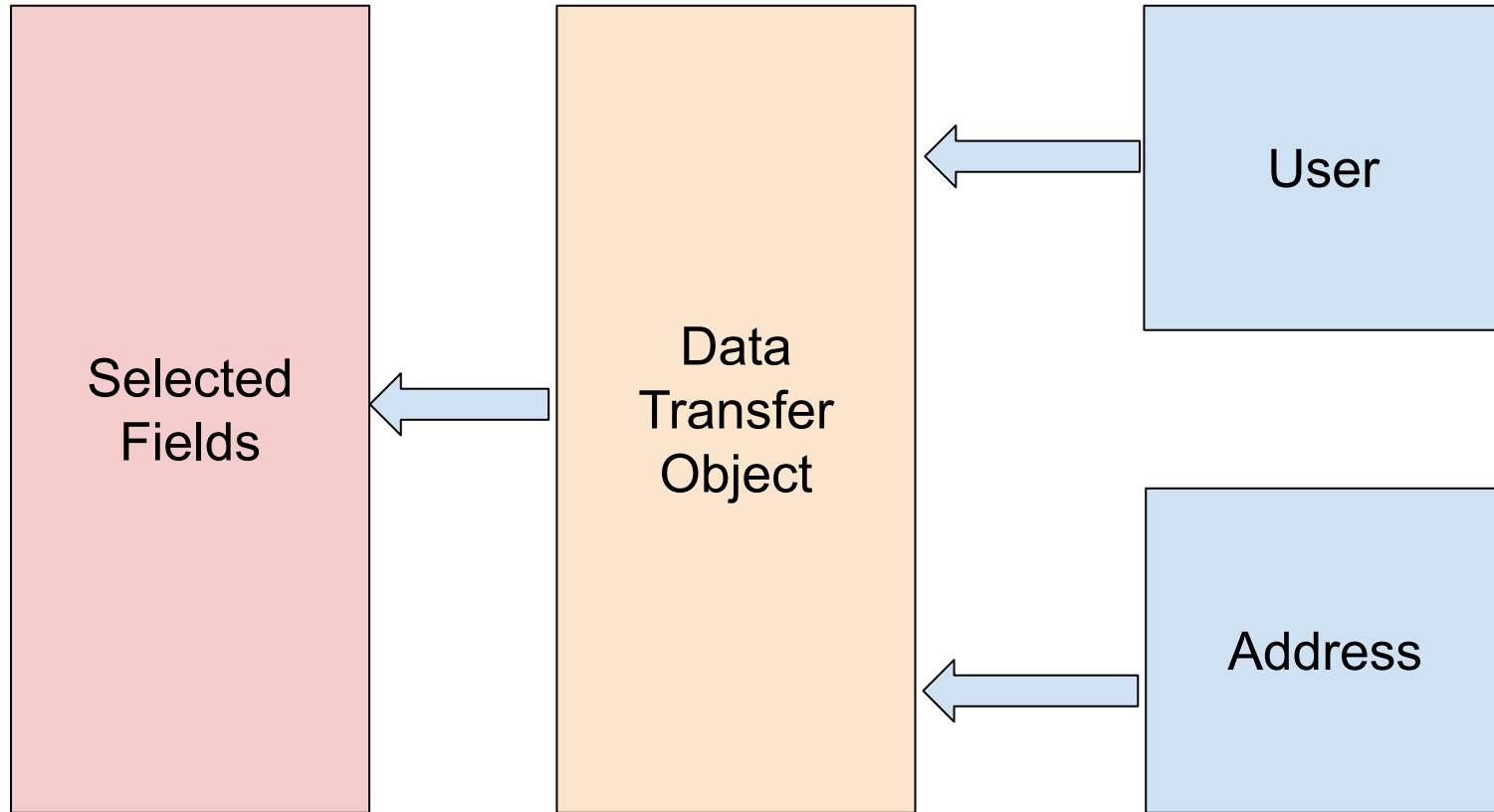
→ *Decentralized Data Management*

DTO Pattern

Faisal Memon (EmbarkX)

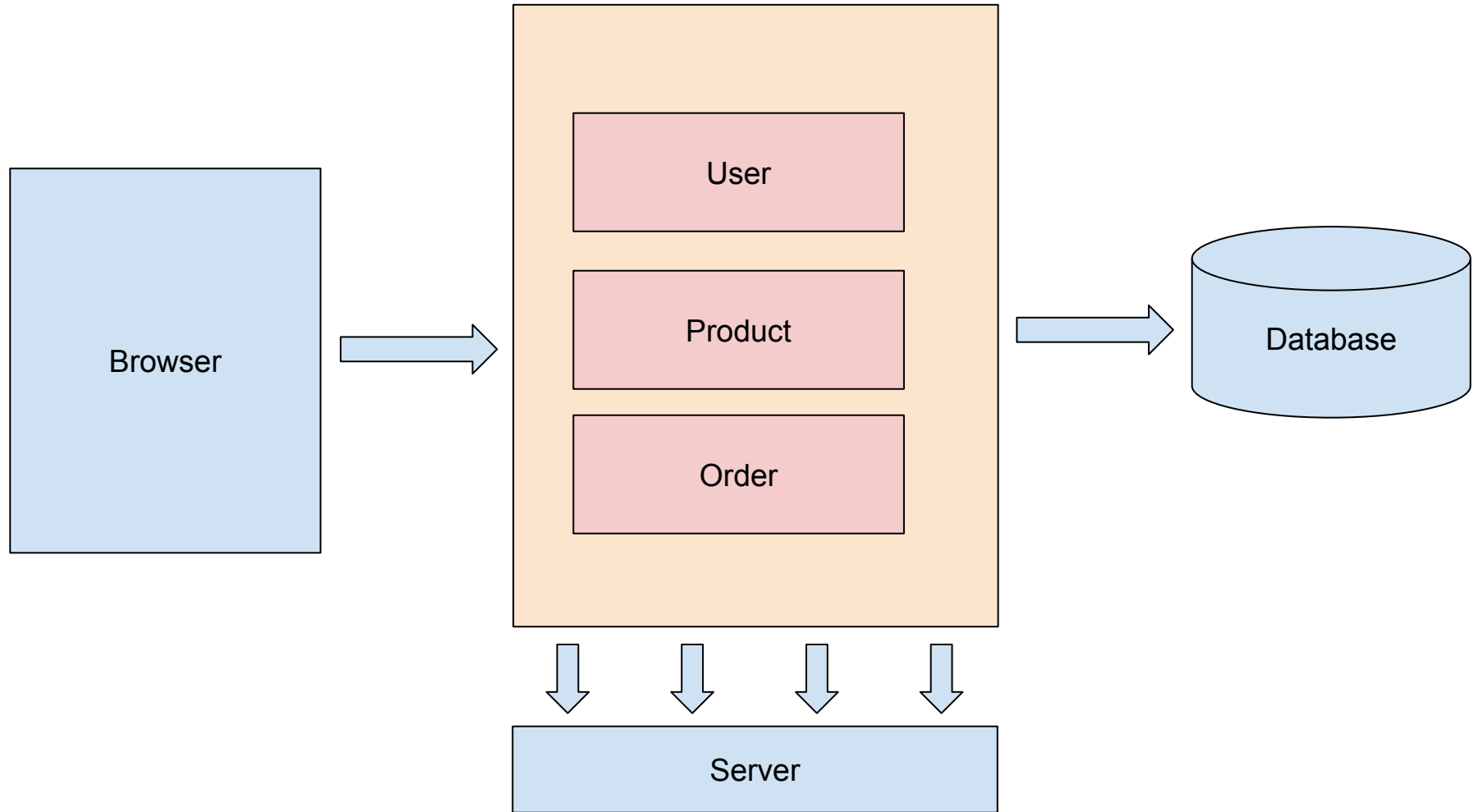
What is it?

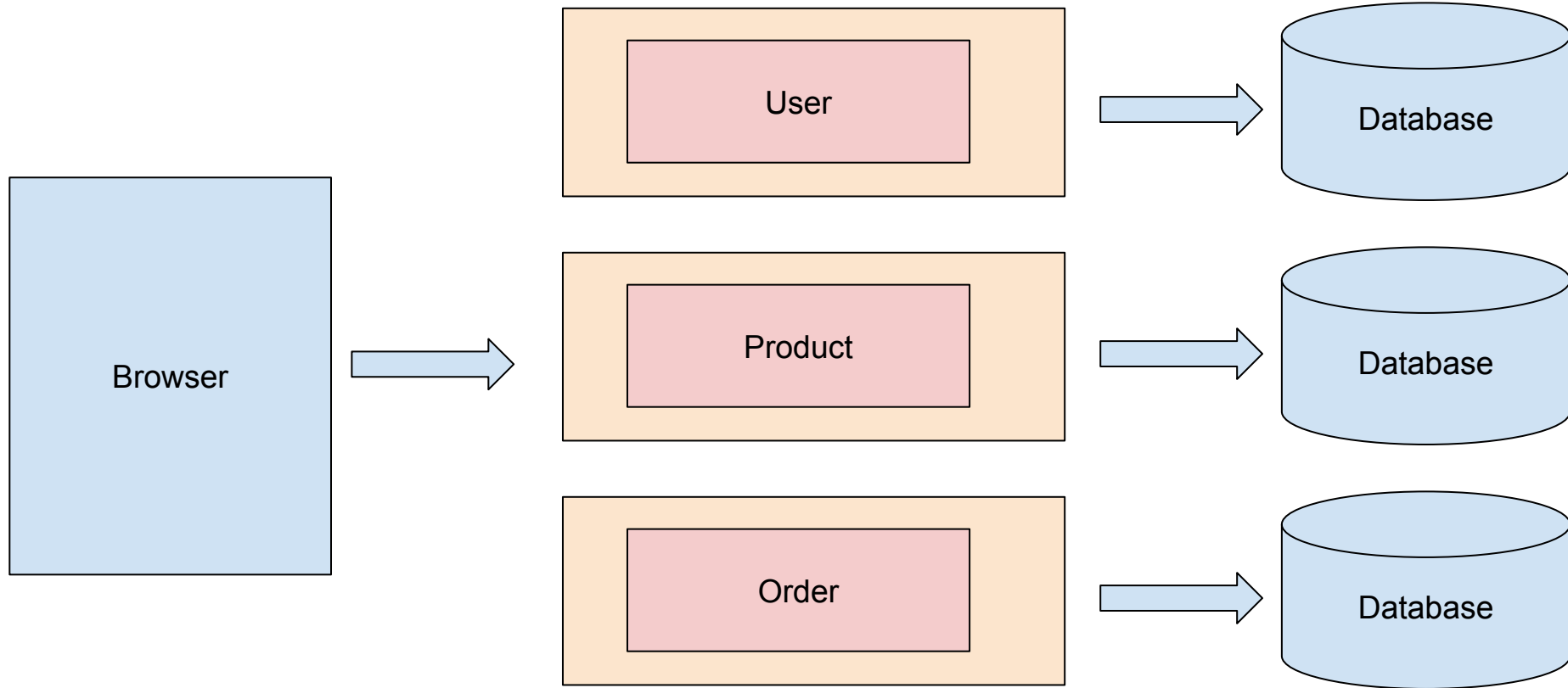
Design pattern used to transfer data between software application subsystems



Planning our changes

Faisal Memon (EmbarkX)

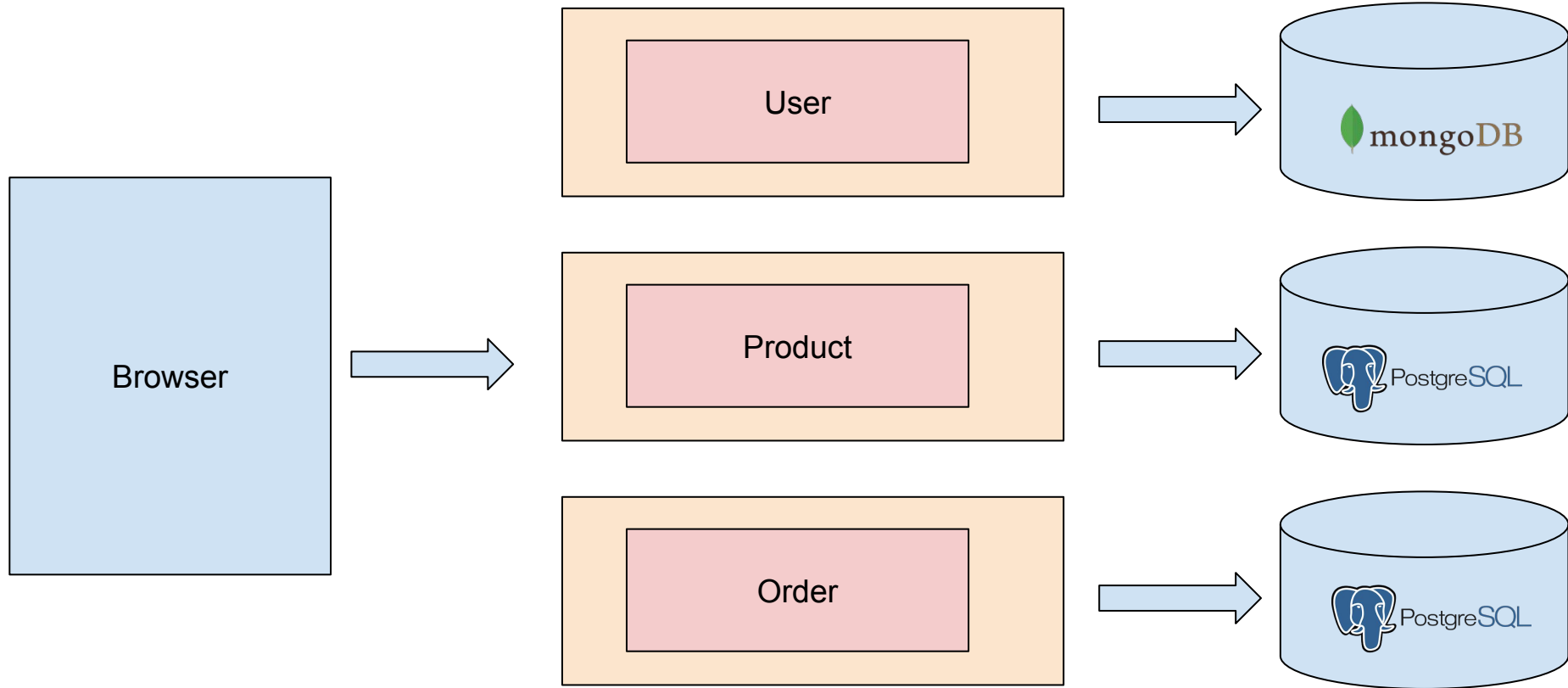




How are we going to structure our Microservices

Faisal Memon (EmbarkX)

Service	Port
Product	8081
User	8082
Order	8083



Configuration Management

Faisal Memon (EmbarkX)

Configuration Management

→ *Simply means managing and controlling the configurations of each microservice in the system*

→ *Configuration may include details such as database connections, external service URLs, caching settings, and more*

Database Configuration

```
spring.datasource.url=jdbc:mysql://localhost:3306/mydatabase
```

```
spring.datasource.username=mydbuser
```

```
spring.datasource.password=mydbpassword
```

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

Hibernate/JPA Settings

`spring.jpa.hibernate.ddl-auto=update`

`spring.jpa.show-sql=true`

Server Configuration

```
server.port=8080
```

```
server.servlet.context-path=/myapp
```

Email Configuration

```
spring.mail.host=smtp.gmail.com
```

```
spring.mail.port=587
```

```
spring.mail.username=myemail@gmail.com
```

```
spring.mail.password=myemailpassword
```

```
spring.mail.properties.mail.smtp.auth=true
```

```
spring.mail.properties.mail.smtp.starttls.enable=true
```

Logging Configuration

```
logging.level.root=INFO
```

```
logging.level.com.example.myapp=DEBUG
```

Application-Specific Properties

```
myapp.api.key=your_api_key
```

```
myapp.file.upload.path=/uploads
```

How does spring boot loads configuration

→ *Parsing and Environment*

→ *Relaxed Binding*

Challenges

- *Environment Management*
- *Security and Sensitive Data*
- *Consistency and Centralization*
- *Dynamic Updates and High Availability*
- *Monitoring and Versioning*

Spring Boot Externalized Configuration

Faisal Memon (EmbarkX)

Scenario

```
@Service
public class ProductService {
    // Hard-coded configuration - Bad Practice ✗
    private static final int MAX_PRODUCTS = 100;
    private static final String DB_URL = "jdbc:mysql://localhost:3306/ecomdb";
}
```

- *Need to recompile for changes*
- *Different values needed for different environments*
- *Security risks with sensitive data*
- *Hard to maintain across multiple services*

Externalized Configuration allows you to keep the configuration settings of your application separate from the codebase.

Ways of Externalizing Configurations

- Using `application.properties` or `application.yml`
- *Environment Variables*
- *Command-Line Arguments*
- *Configuration Files in External Locations*
- *Cloud Configuration (e.g., Spring Cloud Config Server)*

Configuration Formats

Faisal Memon (EmbarkX)

Configuration Formats

// 1. Using application.properties

```
server.port=8081  
spring.application.name=product-service  
spring.datasource.url=jdbc:mysql://localhost:3306/ecom-ms-product
```

// 2. Converting to application.yml

```
server:  
  port: 8081  
spring:  
  application:  
    name: product-service  
  datasource:  
    url: jdbc:mysql://localhost:3306/ecom-ms-product
```

application.properties

```
# Set the server port  
server.port=8081
```

```
# Set the Spring application name  
spring.application.name=product-service
```

```
# Set the datasource URL for MySQL  
spring.datasource.url=jdbc:mysql://localhost:3306/ecom-ms-product
```

→ *Simple key-value pairs*

→ *No hierarchy*

→ *Less readable for complex configurations*

application.yml

```
# Set the server port
```

```
server:
```

```
  port: 8081
```

```
# Set the Spring application name
```

```
spring:
```

```
  application:
```

```
    name: product-service
```

```
# Set the datasource URL for MySQL
```

```
datasource:
```

```
  url: jdbc:mysql://localhost:3306/ecom-ms-product
```

→ *Hierarchical structure*

→ *More readable*

→ *Requires consistent indentation*

YAML vs PROPERTIES

Parameter/Criteria	YAML (application.yml)	Properties (application.properties)
Format Type	Structured, uses indentation	Flat key-value pairs
Readability	More readable, especially for nested or hierarchical data	Simpler, but can become difficult to read with complex configurations
Hierarchy Representation	Supports nesting with indentation, easy to represent nested objects	No natural support for hierarchy, but can simulate it using dot notation
Complexity Handling	Well-suited for complex, nested configurations (e.g., lists, maps)	Best for simple configurations, less ideal for deeply nested data
File Size	Slightly larger due to indentation and structure	Generally more compact
Ease of Writing	Requires careful indentation and whitespace	Simpler to write but can become difficult to manage for complex configs
Configuration Duplication	Easier to avoid duplication due to structured format	Duplication can occur when keys are repeated across different levels

YAML vs PROPERTIES

Parameter/Criteria	YAML (application.yml)	Properties (application.properties)
Common Use Case	Large projects with multiple services or hierarchical data	Simple configurations or smaller projects
Error-Prone Nature	Can be error-prone if indentation is incorrect (e.g., extra spaces or missing indents)	Less error-prone for simple key-value pairs but becomes cluttered with complex structures
Flexibility in Structure	Flexible, can represent different types of data structures	More rigid structure due to flat key-value format
Spring Boot Support	Fully supported in Spring Boot	Fully supported in Spring Boot
File Extension	<code>.yaml</code> or <code>.yml</code>	<code>.properties</code>

Spring Boot Profiles

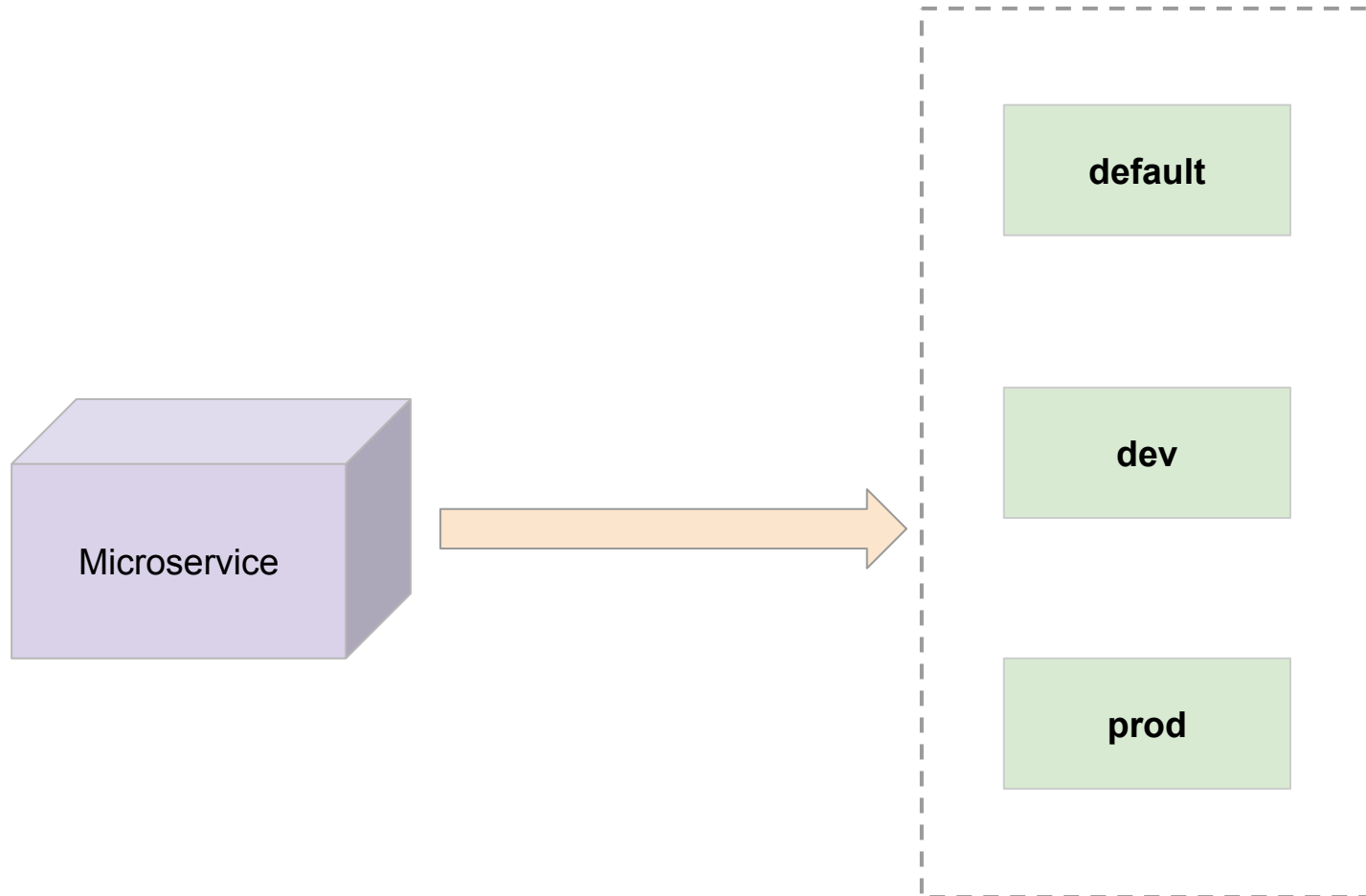
Faisal Memon (EmbarkX)

Introduction to Spring Boot Profiles

- Spring Boot profiles allow you to define multiple configurations for different environments (e.g., development, production, testing)
- These profiles are particularly useful in handling environment-specific settings such as database configurations, logging levels, and other application-specific properties
- Profiles enable Spring Boot applications to be more flexible and adaptable depending on whether they are running in development, testing, or production environments
- This means you can maintain separate configurations for each environment without manually changing configuration files when switching environments.

Why Use Profiles

- *Environment-Specific Configurations*
- *Ease of Switching*
- *Better Code Organization*



Benefits

- *Database Configuration*
- *Security*
- *Performance and Scaling*
- *External API Configurations*
- *Feature Toggles and Configuration Flags*
- *Easy Switching Between Environments*

How Spring Boot Resolves Configuration

Faisal Memon (EmbarkX)

Precedence Order

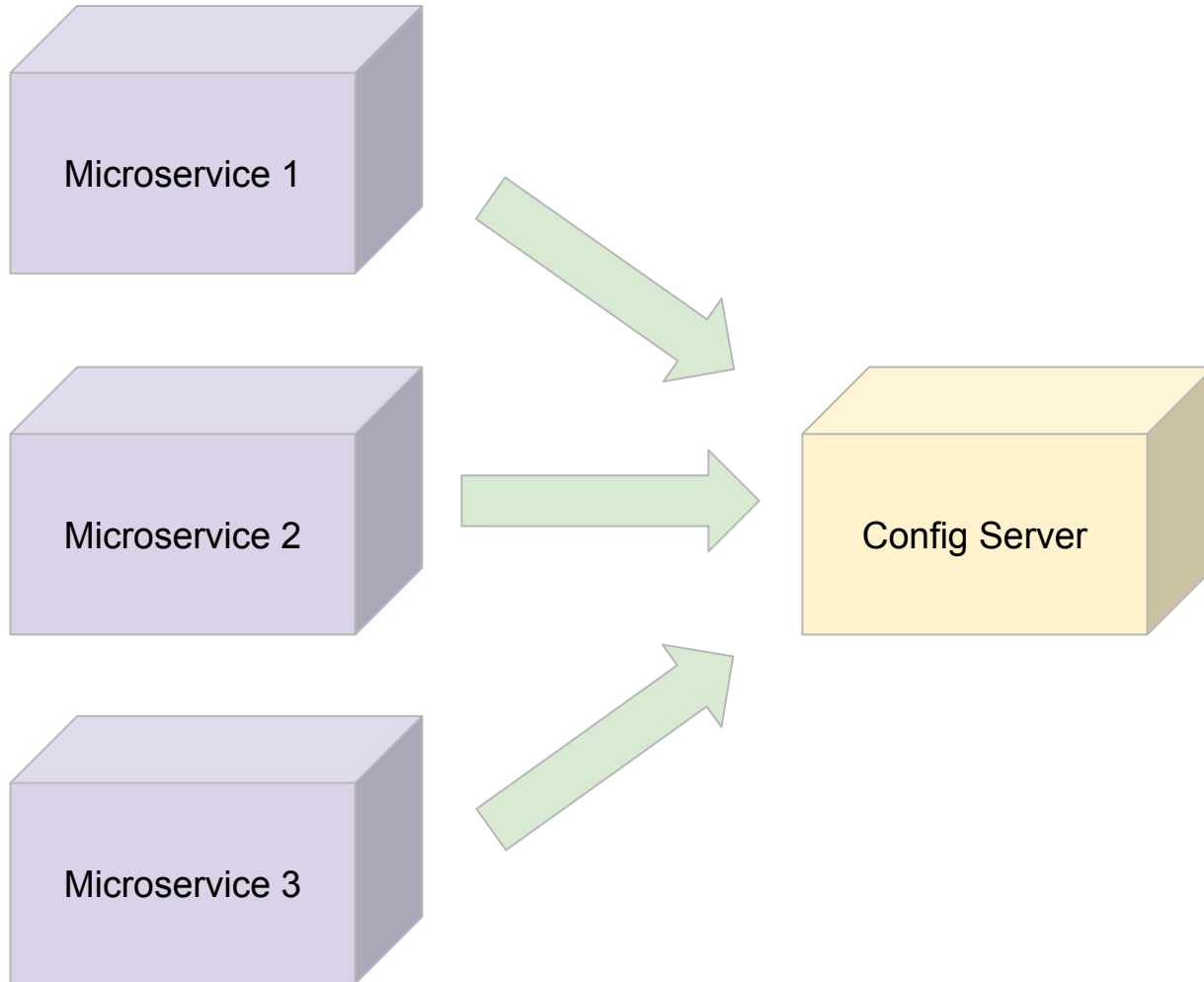
- Command-line arguments (*--build.id=12345*)
- Java system properties (*-Dbuild.id=12345*)
- OS environment variables (*export BUILD_ID=12345*)
- *application.properties* or *application.yml*
- Spring Cloud Config Server (if used)
- Default values inside the application code

Configuration Management with Spring Cloud Config Server

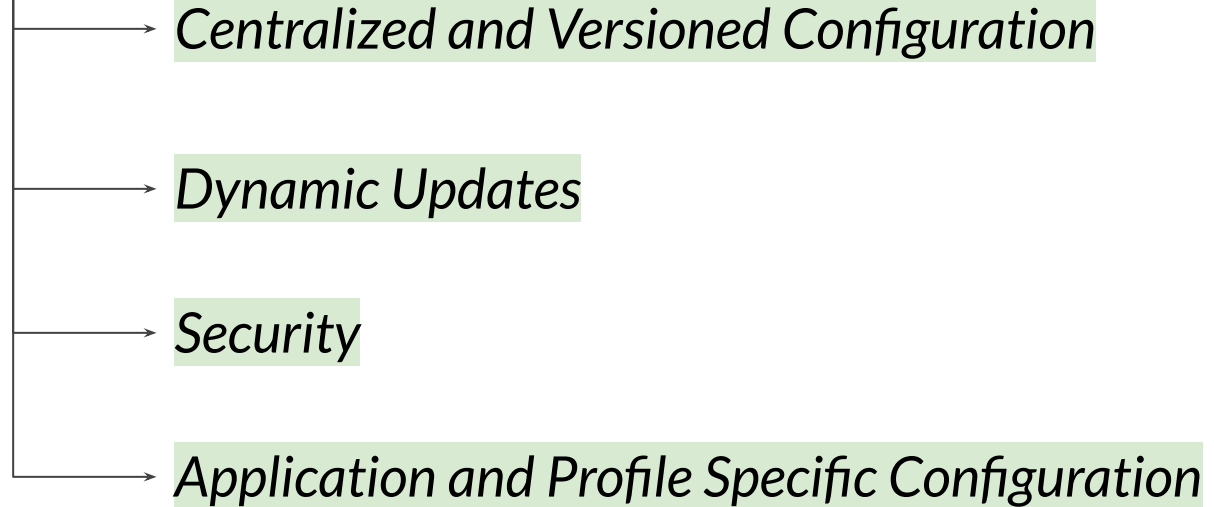
Faisal Memon (EmbarkX)

Configuration Management

- Simply means managing and controlling the configurations of each microservice in the system
- Configuration may include details such as database connections, external service URLs, caching settings, and more
- **Challenge:** As the number of Microservices increases in your architecture, managing the individual configurations can become a complex task.
- A centralized Config Server provides a central place for managing configurations across all microservices
- It simplifies configuration management and increases operational efficiency.



Features of a Config Server

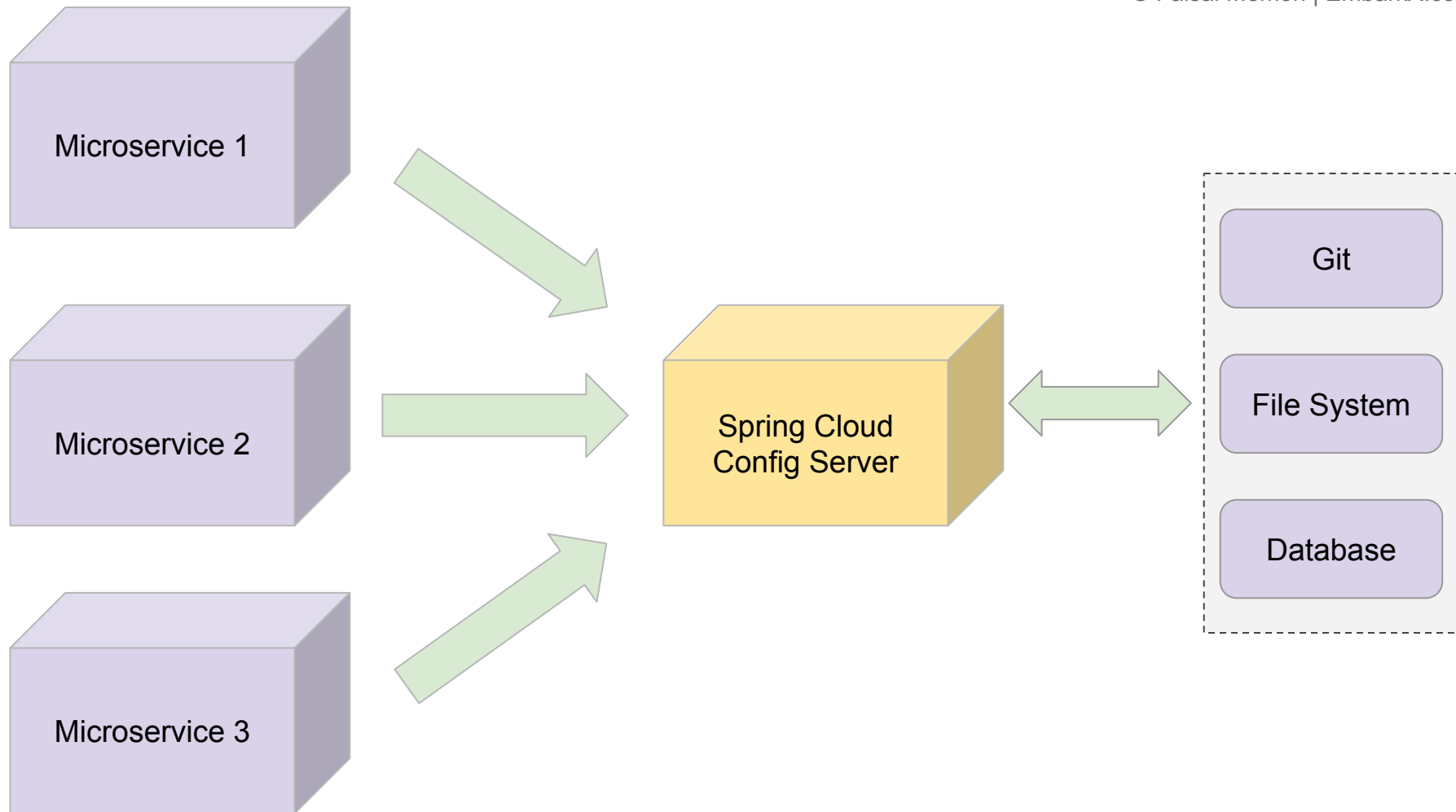
- 
- ```
graph LR; A[Features of a Config Server] --> B[Centralized and Versioned Configuration]; A --> C[Dynamic Updates]; A --> D[Security]; A --> E[Application and Profile Specific Configuration];
```
- *Centralized and Versioned Configuration*
  - *Dynamic Updates*
  - *Security*
  - *Application and Profile Specific Configuration*

# Benefits of a Config Server

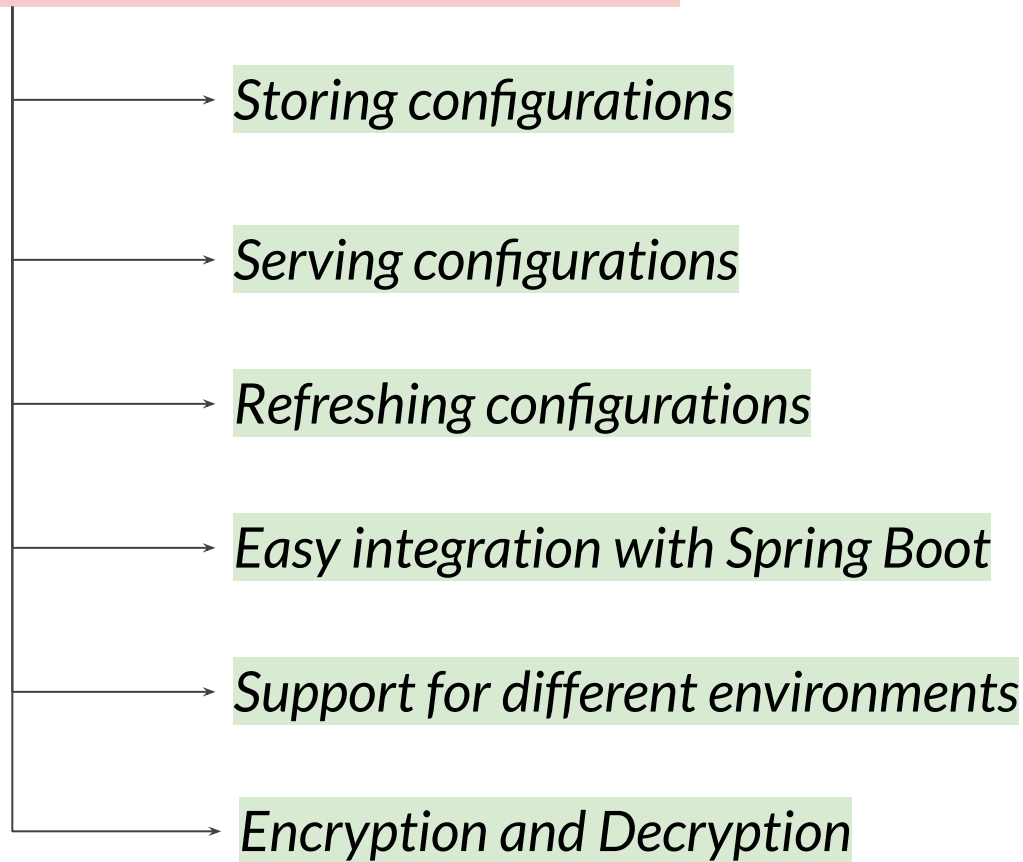
- *Single source of truth*
- *Easier to manage and update configurations*
- *Enhances security and control*
- *Easy to deploy and scale microservices*

**Spring Cloud Config Server** is part of the Spring Cloud project, a suite of tools specifically designed for building and managing cloud-native applications.





# Spring Cloud Config Server



# AES Encryption

Faisal Memon (EmbarkX)

**AES** is a secure way to  
lock and unlock data  
using a secret key.

# How AES Works

→ You Have a Message → "MySecretPassword"

→ You Use an AES Key (Example: "0E329232EA6D0D73")

→ AES Encrypts the Message → Converts "MySecretPassword" into "AQAB12345XYZ==" (random-looking text)

→ To Read the Message Again → Use the same AES key to decrypt it back to "MySecretPassword".

# **Key Features of AES**

- *Fast & Secure*
- *Uses a Fixed-Length Key*
- *Same Key for Encryption & Decryption*

**AES** is commonly used in Spring Cloud Config Server to protect sensitive configuration values like passwords and API keys!

# Using Keystore & RSA Key Pair

Faisal Memon (EmbarkX)



## **Keystore & RSA Key Pair**

- *Instead of using a single, plain-text key (like an AES key) directly in your configuration file, this method uses a keystore—a secure file that holds encryption keys*
- *With the keytool command, you generate an RSA key pair (one public and one private key) and store it in the keystore*
- *The key pair is then used by Spring Cloud Config Server to encrypt (lock) and decrypt (unlock) sensitive data like database passwords*

## How it Works

**Encryption:** When you want to secure a property, the Config Server uses the public part of the key pair (managed by the keystore) to encrypt the value.

**Decryption:** When a client requests the configuration, the server uses the private key to decrypt the property before sending it.

## **How Is It Different from AES?**

*AES is like having **one master key** for a safe, while RSA is like having a **lock and key pair** where one key is public (to lock) and another is private (to unlock)*

# **What Is Keytool and Why Use It?**

→ *Keytool is a command-line utility provided with the Java Development Kit (JDK)*

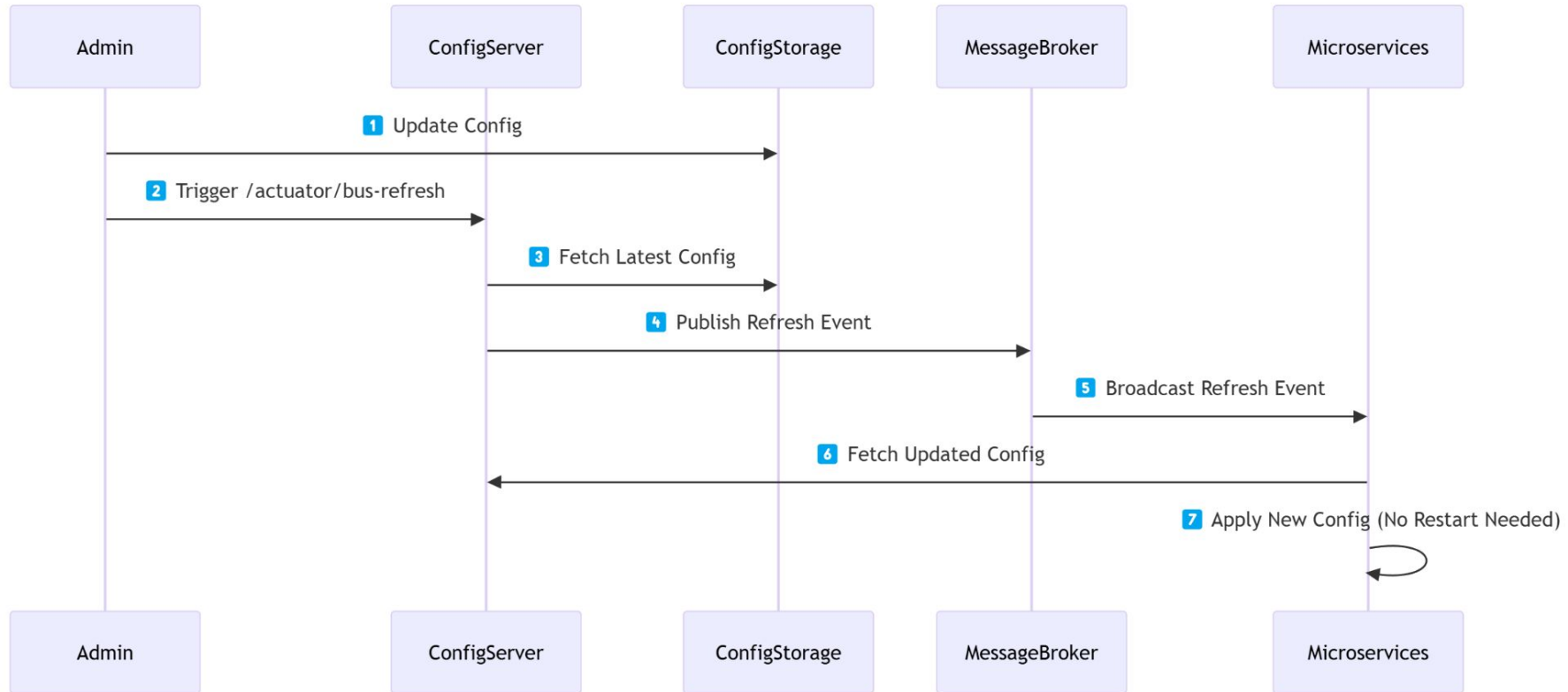
→ *It helps you create and manage keystores, generate key pairs (RSA keys, for example), and handle digital certificates.*

**AES** is a secure way to  
lock and unlock data  
using a secret key.

**AES** is commonly used in Spring Cloud Config Server to protect sensitive configuration values like passwords and API keys!

# Spring Cloud Bus Refresh

Faisal Memon (EmbarkX)





# Best Practices for Production

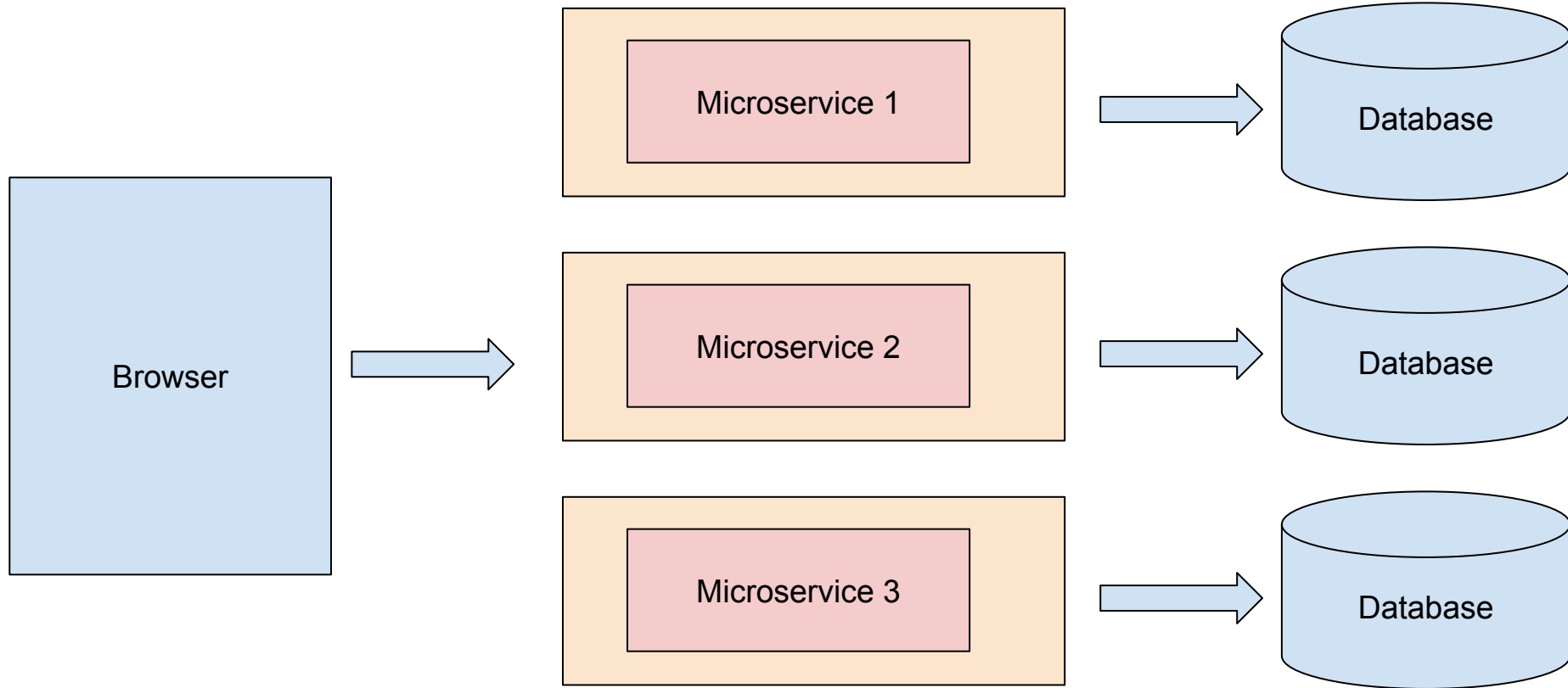
Faisal Memon (EmbarkX)

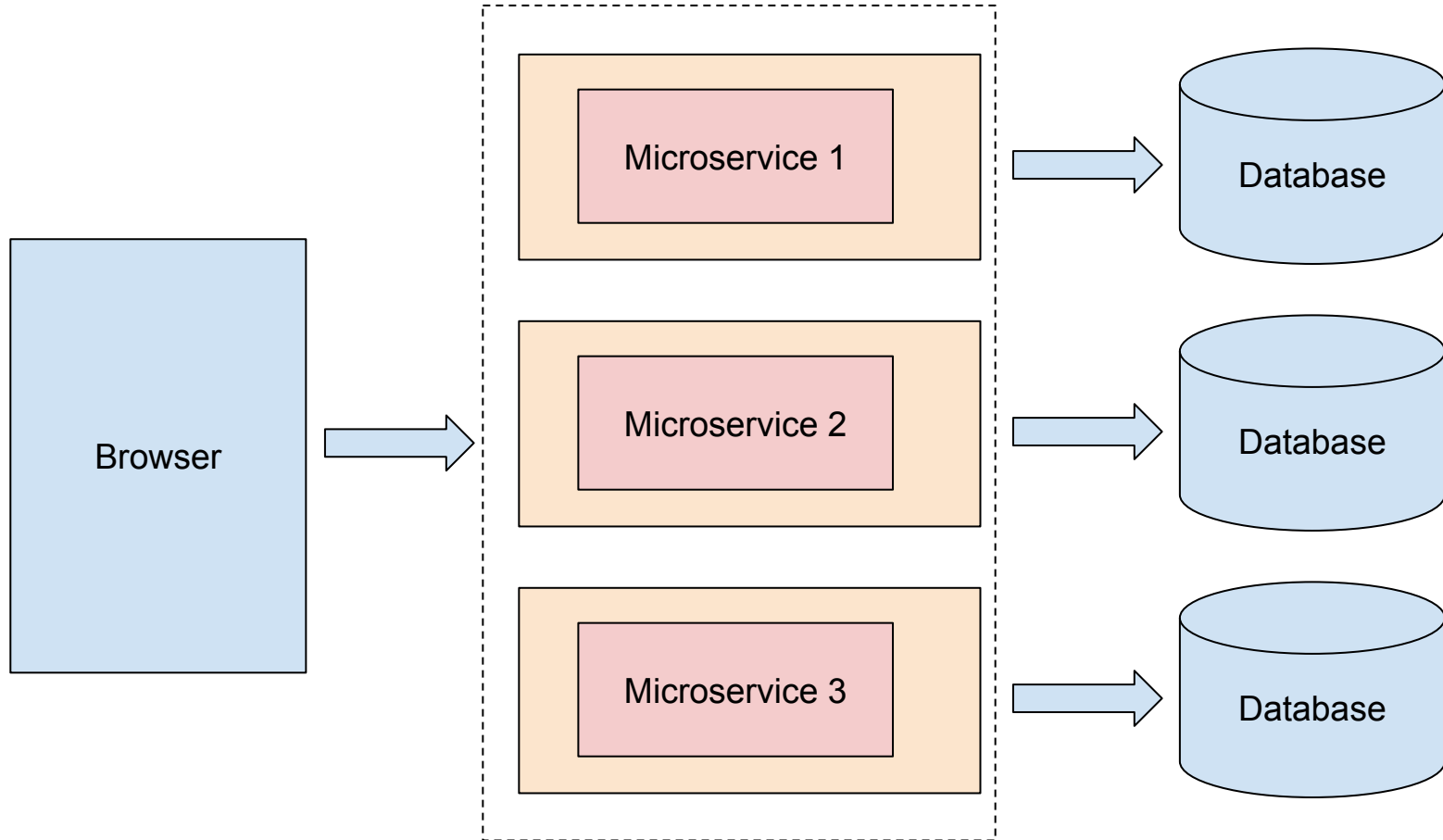
# **Best Practices for Production**

- *Externalize Configurations*
- *Use Profiles*
- *Encrypt Sensitive Data*
- *Use Secure Connections*
- *Access Control*

# Introduction to InterService Communication

Faisal Memon (EmbarkX)





# **Why is Inter-Service Communication so important?**

## ***Ways to implement***



```
graph LR; A[Ways to implement] --> B[Synchronous Communication]; A --> C[Asynchronous Communication]
```

*Synchronous Communication*

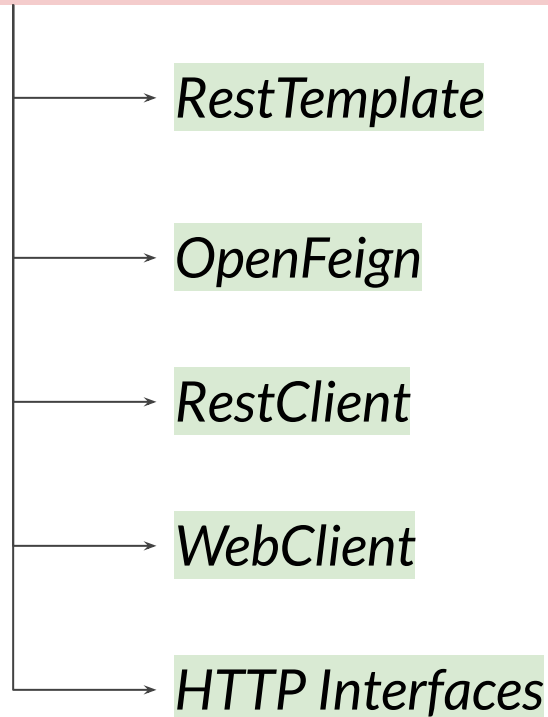
*Asynchronous Communication*

# Different Ways of Doing Synchronous Communication

Faisal Memon (EmbarkX)



# Synchronous Communication



# **Evolution of Synchronous communication**

- *The Early Days: Manual HTTP Calls and Low-Level Clients*
- *Introduction of Client-Side Load Balancing and Service Discovery*
- *Declarative HTTP Clients: OpenFeign and Beyond*
- *Other HTTP Clients*

# The Early Days: Manual HTTP Calls and Low-Level Clients

## **Direct HTTP Client Usage**

- In the initial stages, services communicated using basic HTTP clients (like Apache HttpClient) or even raw socket programming
- This required developers to manually handle request creation, response parsing, error handling, and retries.

## **Spring's RestTemplate**

- As microservices became more popular, Spring introduced RestTemplate
- Simplified making HTTP calls by providing higher-level methods
- Still required explicit configuration for aspects like timeouts, error handling, and manual integration with client-side load balancing

## Spring's RestTemplate

```
RestTemplate restTemplate = new RestTemplate();
String response =
restTemplate.getForObject("http://example.com/api/resource",
String.class);
```

- *Introduced in Spring for simplifying REST calls.*
- *Uses methods like getForObject() or postForEntity().*
- *Ideal for straightforward, blocking HTTP interactions.*

# Introduction of Client-Side Load Balancing & Service Discovery

## **Eureka**

→ To tackle issues like service instance management and fault tolerance, things like Eureka (for service discovery) were integrated with RestTemplate

→ This made interservice communication more resilient and dynamic, allowing calls to automatically distribute among available service instances.

## Declarative HTTP Clients: OpenFeign and Beyond

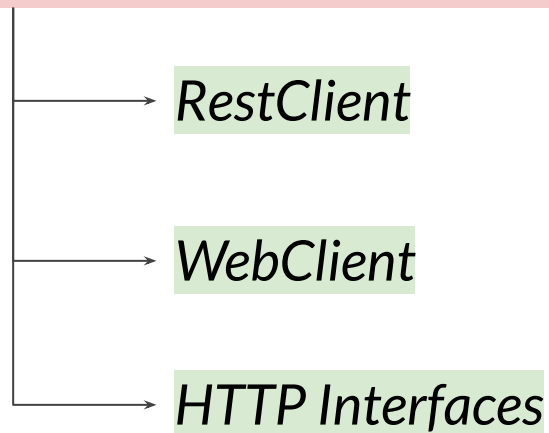
### OpenFeign

→ Recognizing the repetitive nature of REST call boilerplate, declarative clients such as **OpenFeign** emerged

→ With OpenFeign, developers can define a Java interface annotated with mapping annotations

```
@FeignClient(name = "example-service", url = "http://example.com")
public interface ExampleFeignClient {
 @GetMapping("/api/resource")
 String getResource();
}
```

# *New Rest Clients in Spring*



## Modern (New) REST Clients

### **WebClient**

- *Introduced in Spring 5 WebFlux for asynchronous operations.*
- *Supports reactive programming and integrates well with Mono and Flux*
- *Ideal for high-throughput applications.*

```
WebClient webClient = WebClient.create("http://example.com");
Mono<String> response = webClient.get()
 .uri("/api/resource")
 .retrieve()
 .bodyToMono(String.class);
```



## Modern (New) REST Clients

### **RestClient (New in Spring 6)**

→ *Modern alternative to RestTemplate, introduced in Spring 6.*

→ *Provides a fluent API similar to WebClient but remains synchronous.*

```
RestClient restClient = RestClient.create("http://example.com");
String response = restClient.get()
 .uri("/api/resource")
 .retrieve()
 .body(String.class);
```

## Modern (New) REST Clients

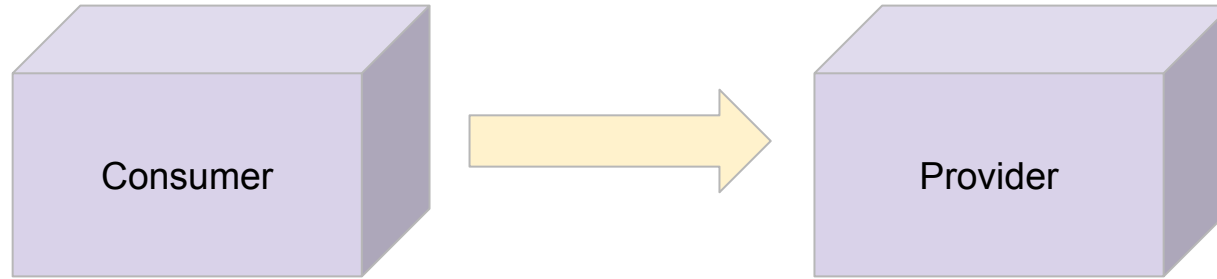
### **HTTP Interface (Spring 6)**

- Annotated interface with a generated, dynamic proxy implementation
- Similar to OpenFeign but built into Spring 6+ without requiring Spring Cloud.
- Uses `@HttpExchange` to declare REST clients in a declarative way.

```
@HttpExchange(url = "/api/resource", method = HttpMethod.GET)
public interface MyApiClient {
 String getResource();
}
```

# Inter Service Communication Scenarios in Our Project

Faisal Memon (EmbarkX)



# Which REST Client To Use?

Faisal Memon (EmbarkX)

# Choosing REST Client

## **Legacy vs. New Projects**

→ *For new projects, it's often better to use more modern alternatives.*

## **Synchronous vs. Asynchronous Needs**

→ *For blocking calls (where each HTTP call waits for a response), RestClient + HttpInterface is straightforward and easy to use.*

→ *If you need non-blocking, reactive behavior, WebClient is the modern choice, especially when handling high concurrency or I/O-bound operations.*

# Choosing REST Client

## **Programming Style and Developer Experience**

→ HTTP Interfaces let you define clients with simple interfaces and annotations, reducing boilerplate and making your code cleaner.

There's no  
**one-size-fits-all**  
answer



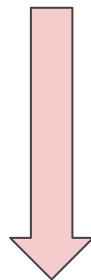
# Service Registry

Faisal Memon

## **What is it?**

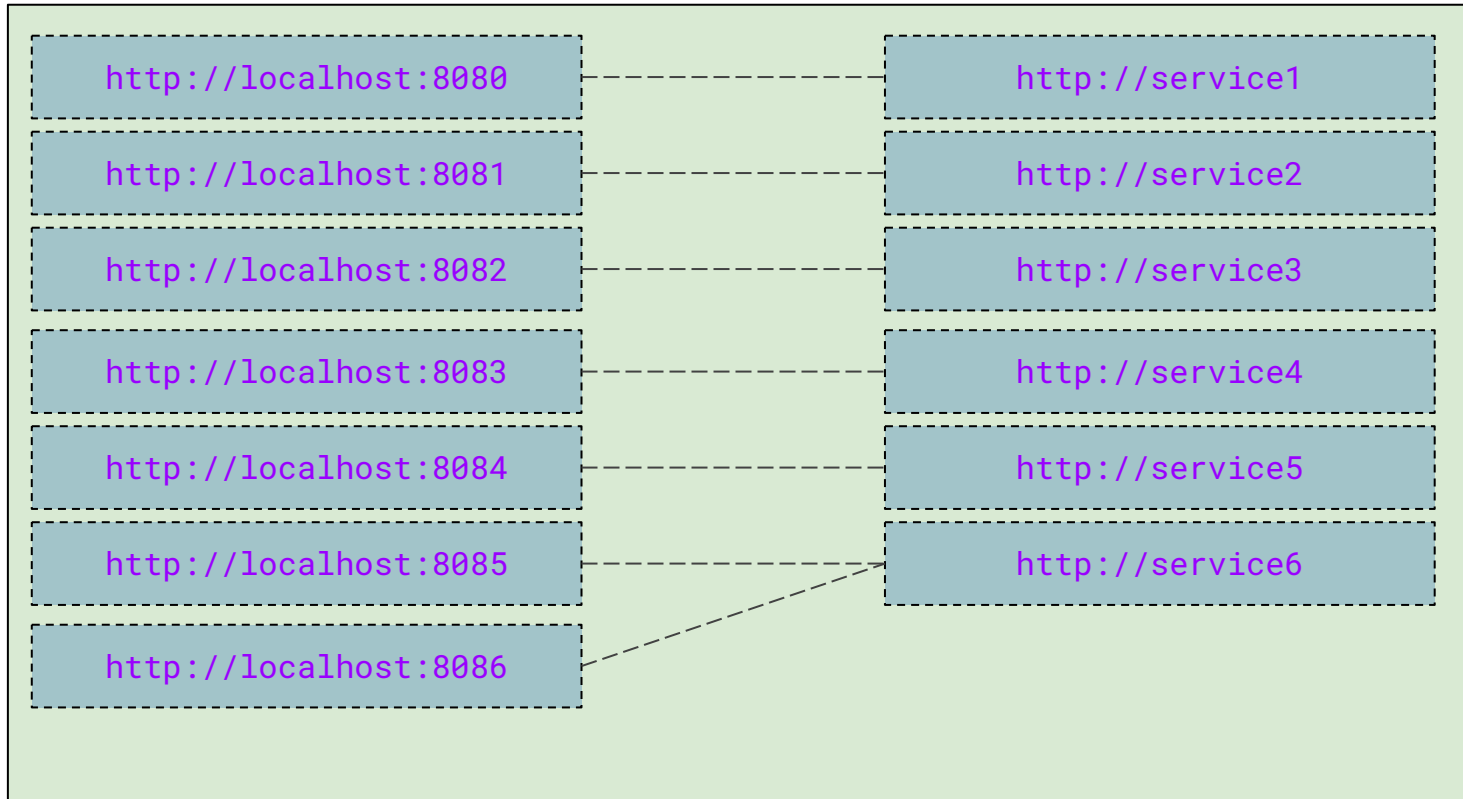
*A service registry is used in microservices architectures to enable dynamic service discovery*

`http://localhost:8081`



`http://provider`

# Service Registry



# Why do we need it?

@Bean

```
public RestClient restClient() {
 return RestClient.builder()
 .baseUrl("http://localhost:8081")
 .build();
}
```

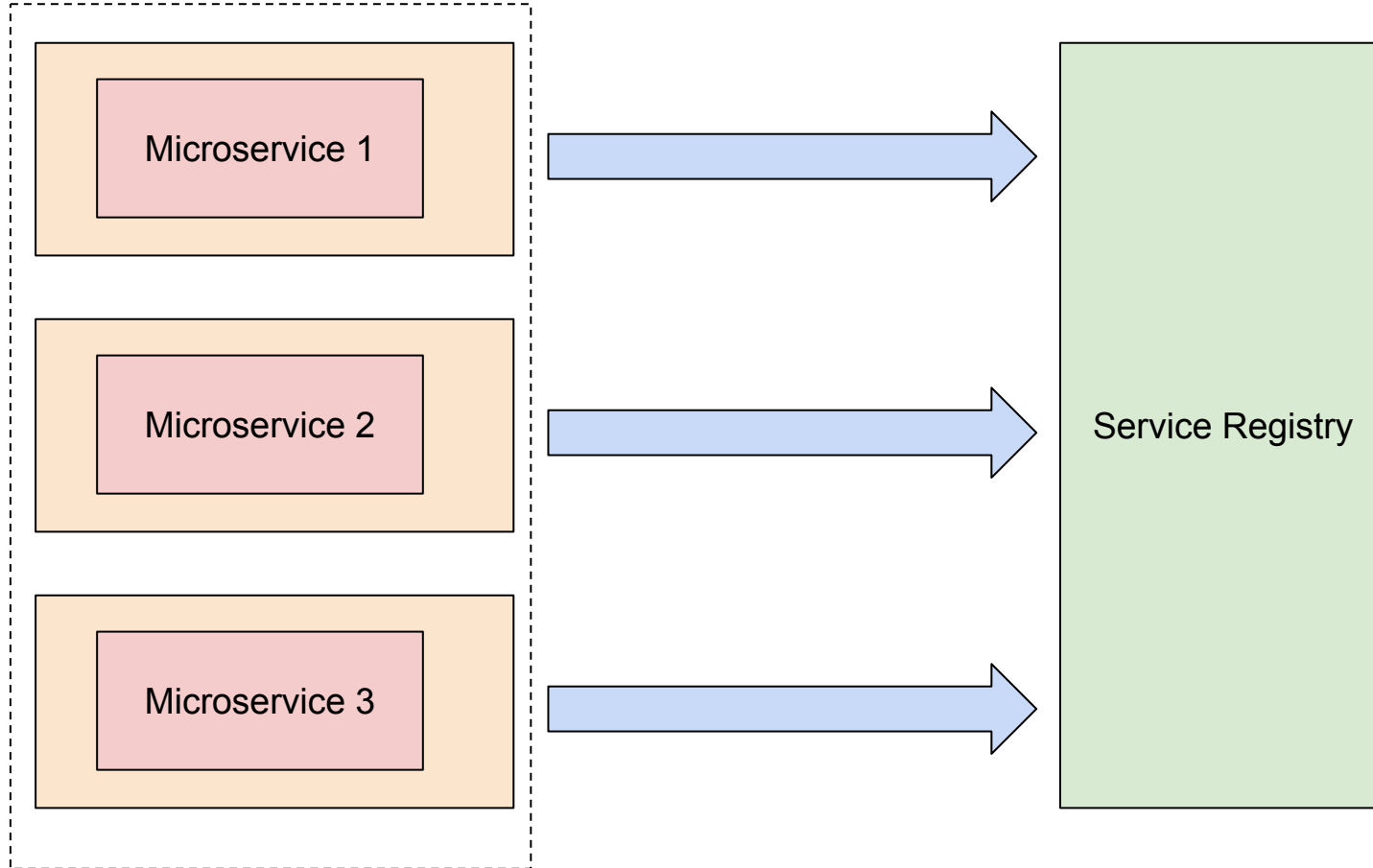
# ***Why do we need it?***

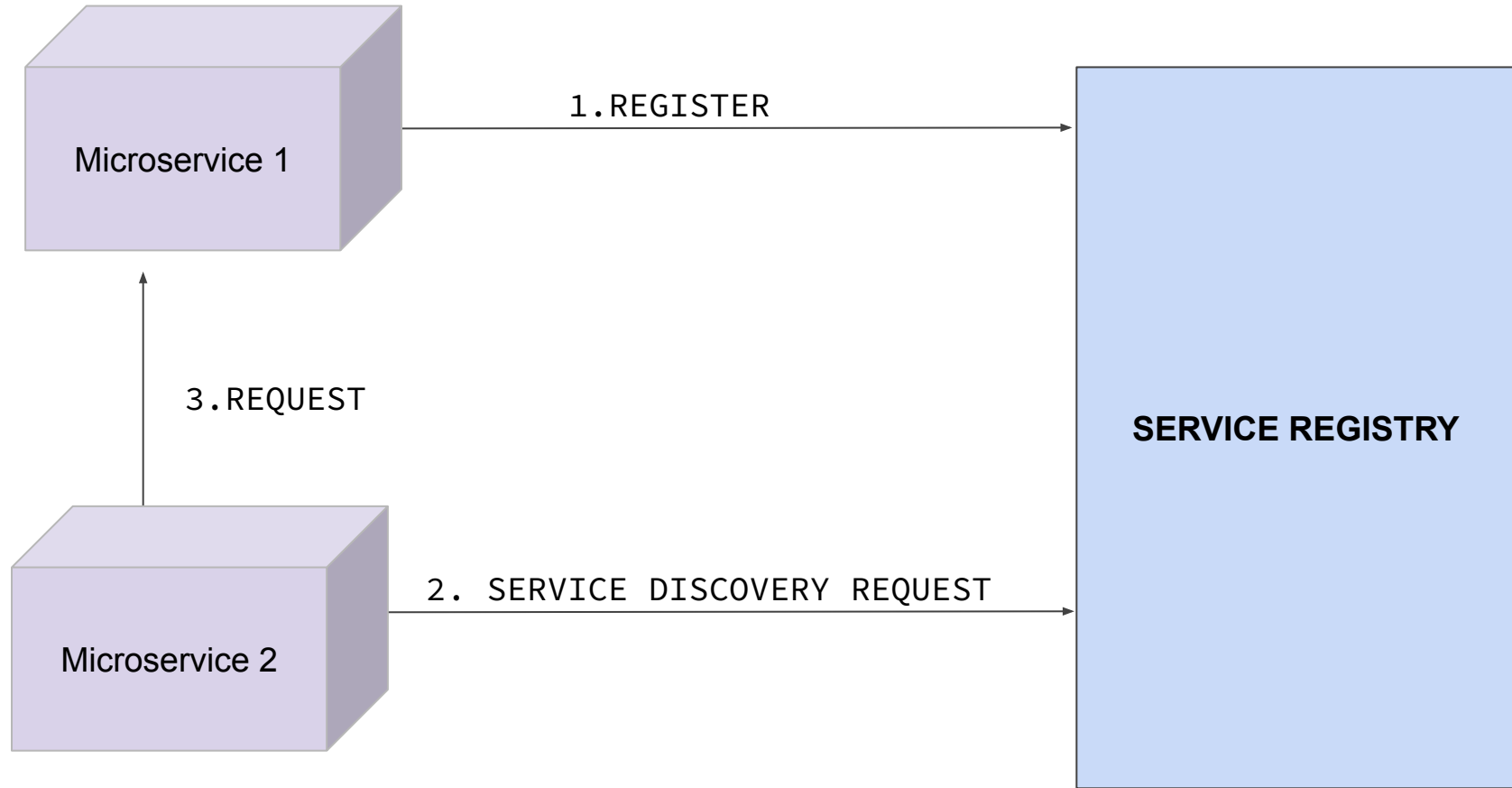
## ***Services can move***

*Services don't always stay at the same address*

## ***Dynamic environments***

*Services may start, stop, or scale dynamically*







Client Side Load  
Balancing into  
play

1. REGISTER

**SERVICE REGISTRY**

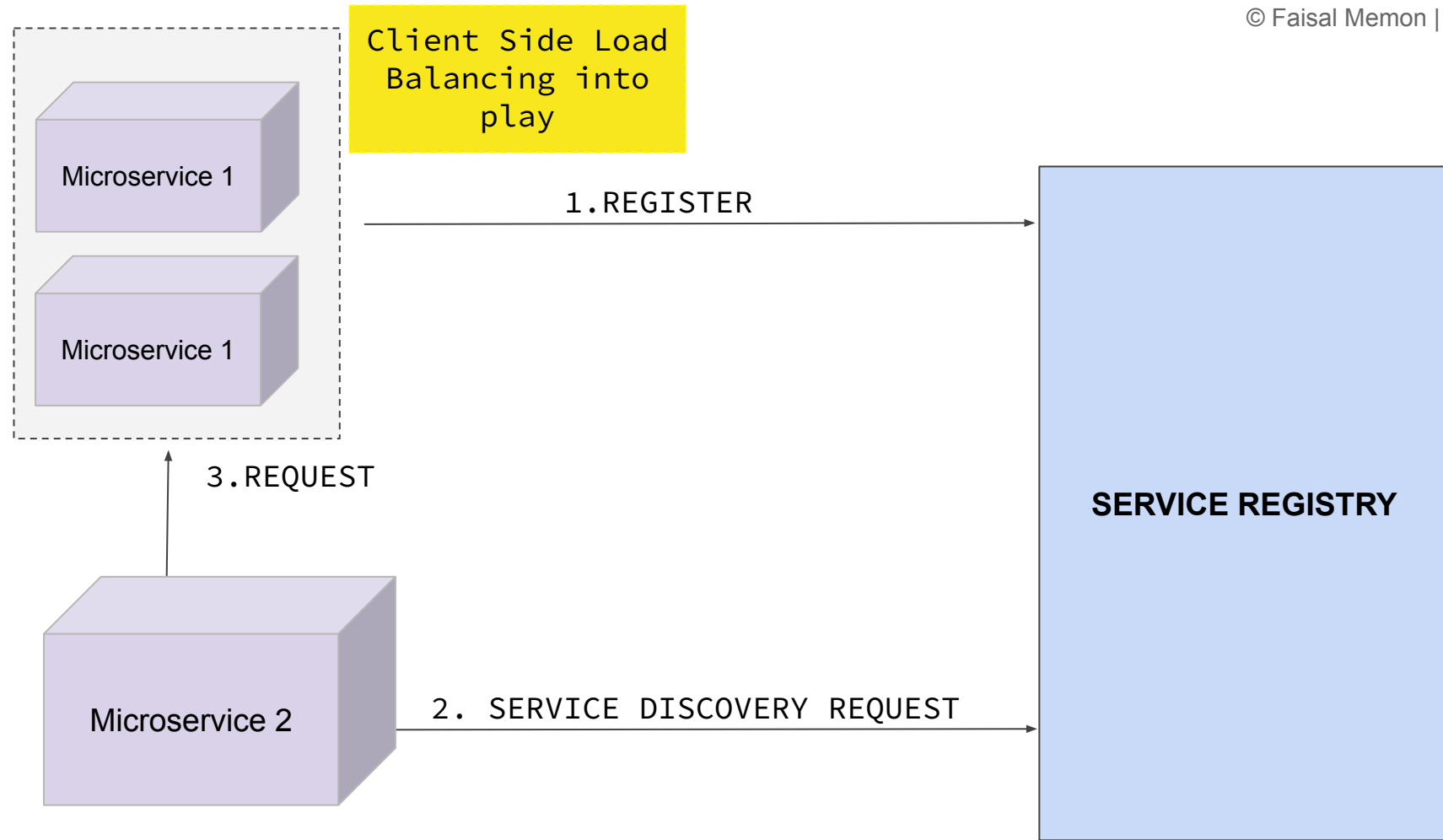
3. REQUEST

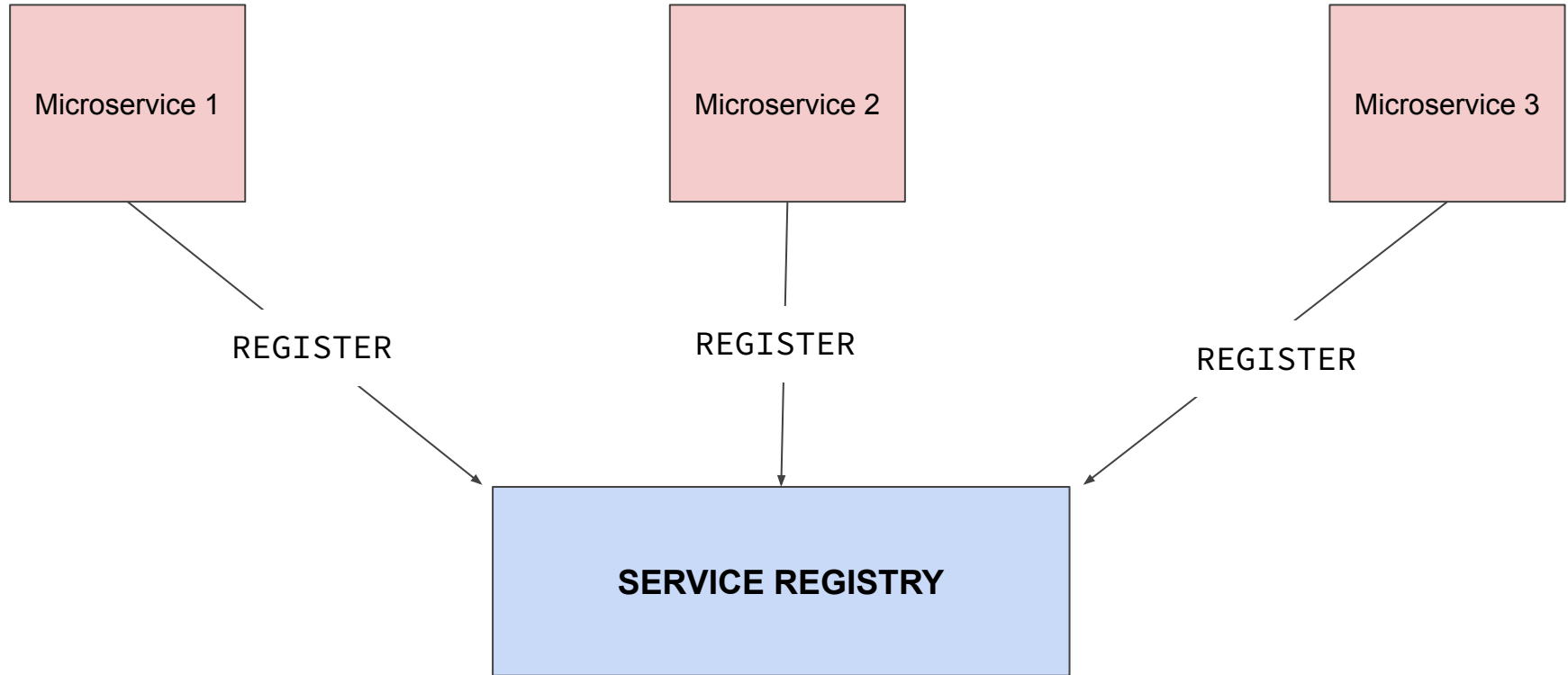
2. SERVICE DISCOVERY REQUEST

Microservice 1

Microservice 1

Microservice 2





# ***Why using a Service Registry is Beneficial?***

- 
- Dynamic Service Discovery*
  - Load Balancing*
  - Fault Tolerance and Resilience*
  - Scalability and Elasticity*
  - Service Monitoring and Health Checks*

# Spring Cloud Eureka

# How Things Worked Before?

Faisal Memon

## Older Approach

- 
- ```
graph LR; A[Older Approach] --> B[Hardcoded Service Endpoints]; A --> C[DNS-Based Service Discovery]; A --> D[Load Balancers (HAProxy, F5, Nginx)]; A --> E[Configuration Servers];
```
- *Hardcoded Service Endpoints*
 - *DNS-Based Service Discovery*
 - *Load Balancers (HAProxy, F5, Nginx)*
 - *Configuration Servers*

Hardcoded Service Endpoints

@Bean

```
public RestClient restClient() {  
    return RestClient.builder()  
        .baseUrl("http://localhost:8081")  
        .build();  
}
```

Hardcoded Service Endpoints

- *Manual Updates*
- *Scalability Issues*
- *No Load Balancing*

DNS-Based Service Discovery

```
String url = "http://product-service.example.com/api/products/123";  
ProductResponse response = restTemplate.getForObject(url,  
ProductResponse.class);
```

DNS-Based Service Discovery

- *DNS Caching*
- *Lack of Health Checks*
- *Slow Scaling*

Load Balancers (HAProxy, F5, Nginx)

```
String url = "http://load-balancer.example.com/api/products/123";  
ProductResponse response = restTemplate.getForObject(url,  
ProductResponse.class);
```

Load Balancers (HAProxy, F5, Nginx)

→ *Limited Auto-Discovery*

→ *Increased Latency*

Configuration Servers (Spring Cloud Config, Zookeeper)

```
@Value("${product.service.url}")
```

```
private String productServiceUrl;
```

```
ProductResponse response = restTemplate.getForObject(productServiceUrl  
+ "/api/products/123", ProductResponse.class);
```

Configuration Servers (Spring Cloud Config, Zookeeper)

→ *Not Dynamic*

→ *No Self-Healing*

→ *Still Needed Load Balancers*

Approach	Pros	Cons
Hardcoded IPs	Simple setup	No scalability, manual updates needed
DNS-based	Some level of abstraction	Slow updates, caching issues
Load Balancer	Distributes traffic	Extra latency
Config Server	Centralized settings	Not dynamic, restart needed
Service Registry	Auto-discovery, dynamic updates, built-in health checks	Adds dependency, requires additional service

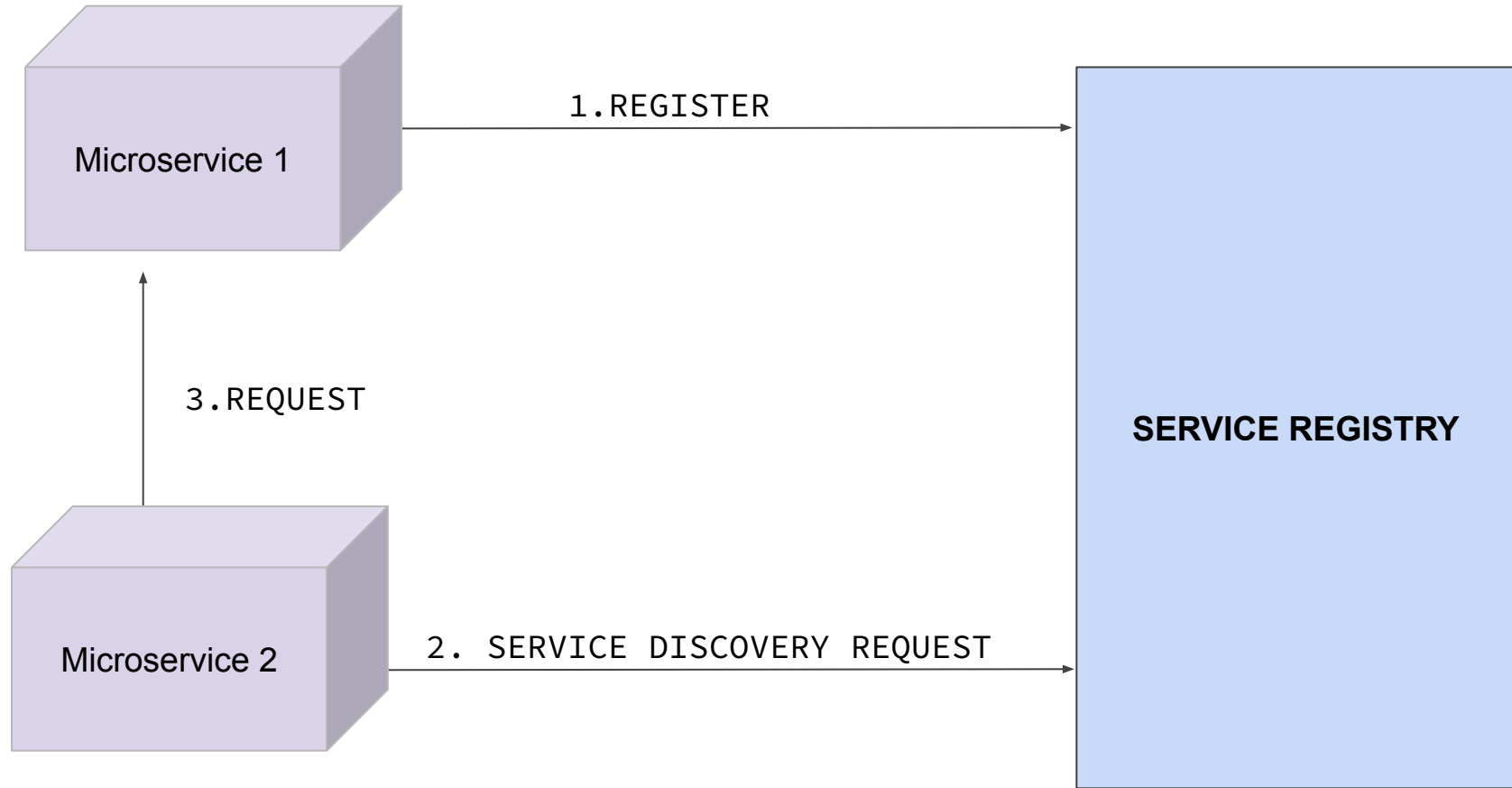
Shutting Down Registered Services Gracefully

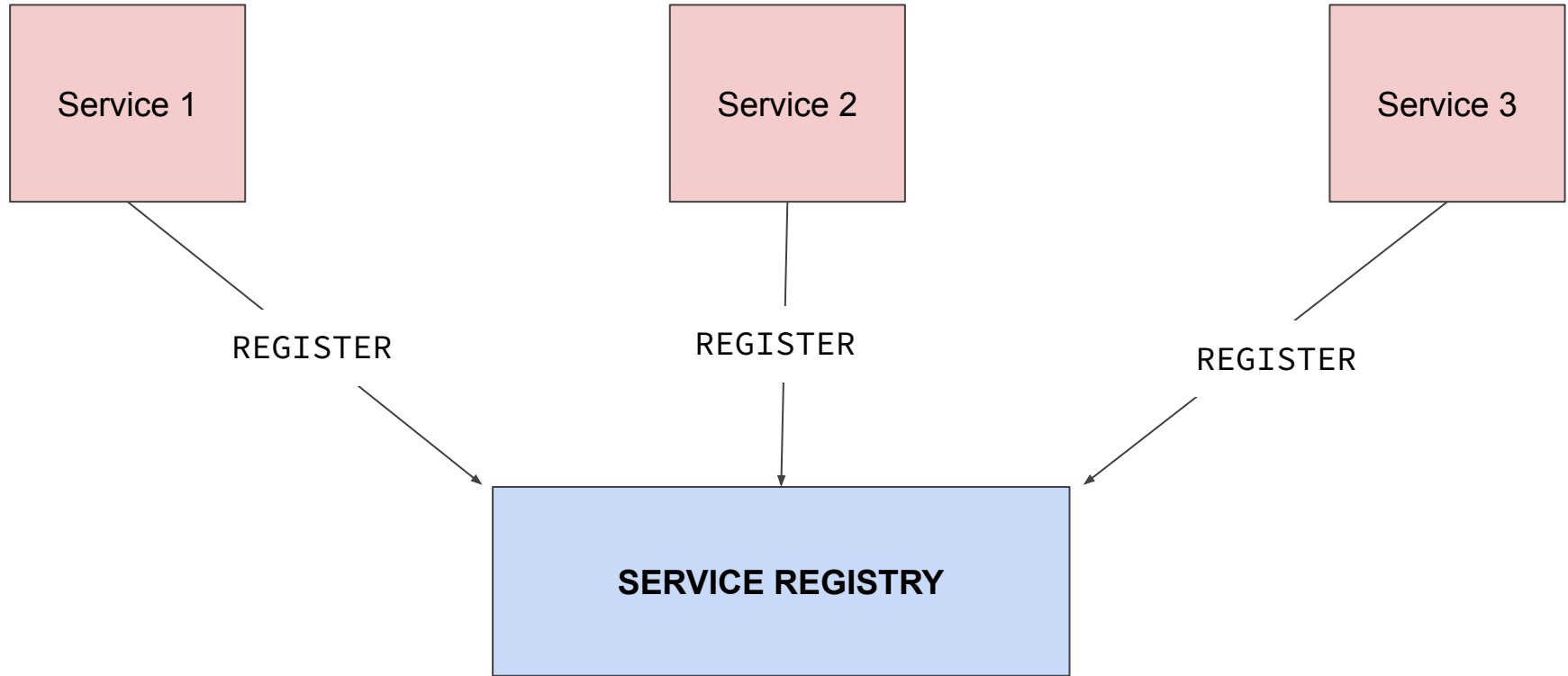
Faisal Memon (EmbarkX)

✗ Not Good Approach	✓ Best Approach
● Pressing Red Button in IntelliJ	↺ Use Actuator Shutdown (<code>/actuator/shutdown</code>)
✗ Kills process immediately	✓ Ensures proper cleanup
✗ Eureka might still consider the instance alive	✓ Eureka removes the instance properly
✗ Might leave open DB/connections	✓ Handles all resource cleanup

Behind the Scenes of Eureka Server

Faisal Memon (EmbarkX)





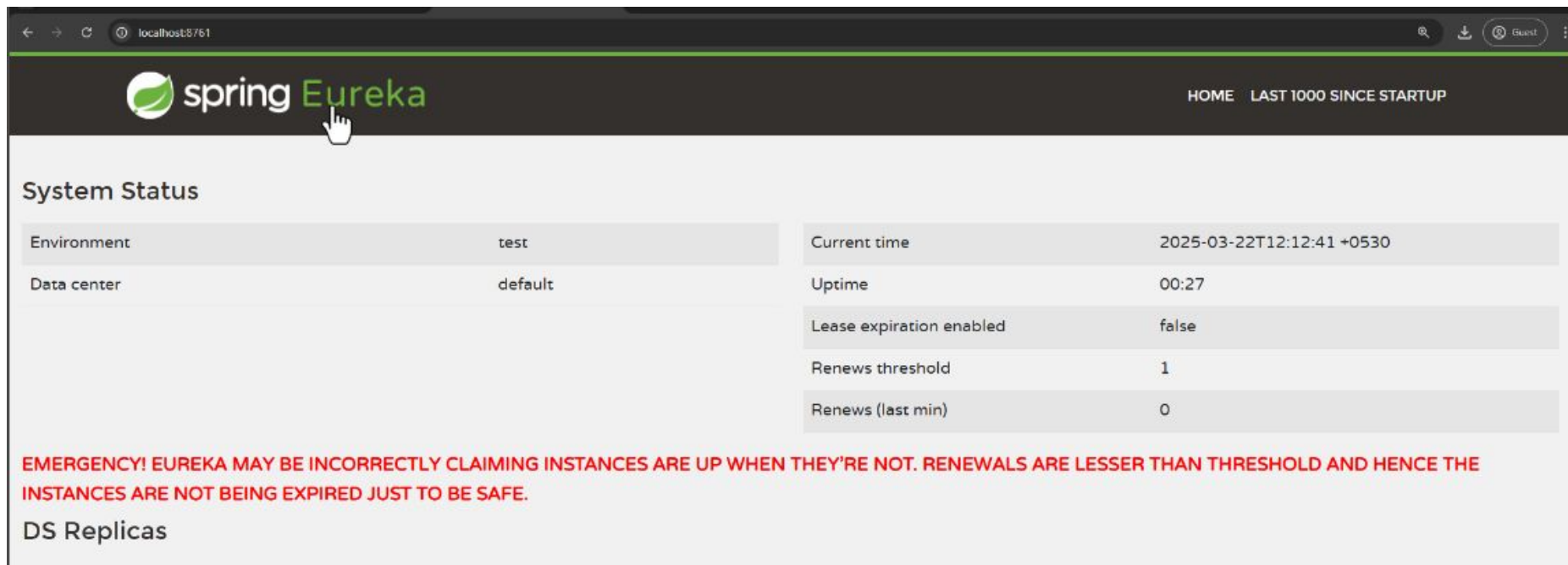
Behind the scenes

→ *Heartbeat Signal*

→ *Heartbeat Monitoring*

Self Preservation Mode Eureka

Faisal Memon (EmbarkX)



The screenshot shows the Spring Eureka web interface in a browser window. The address bar shows 'localhost:8761'. The header has the 'spring Eureka' logo and navigation links for 'HOME' and 'LAST 1000 SINCE STARTUP'. The main content area is titled 'System Status' and contains two tables. The left table shows 'Environment: test' and 'Data center: default'. The right table shows system metrics: 'Current time: 2025-03-22T12:12:41 +0530', 'Uptime: 00:27', 'Lease expiration enabled: false', 'Renews threshold: 1', and 'Renews (last min): 0'. Below these tables is a red emergency warning message. At the bottom, there is a section for 'DS Replicas'.

localhost:8761

spring Eureka

HOME LAST 1000 SINCE STARTUP

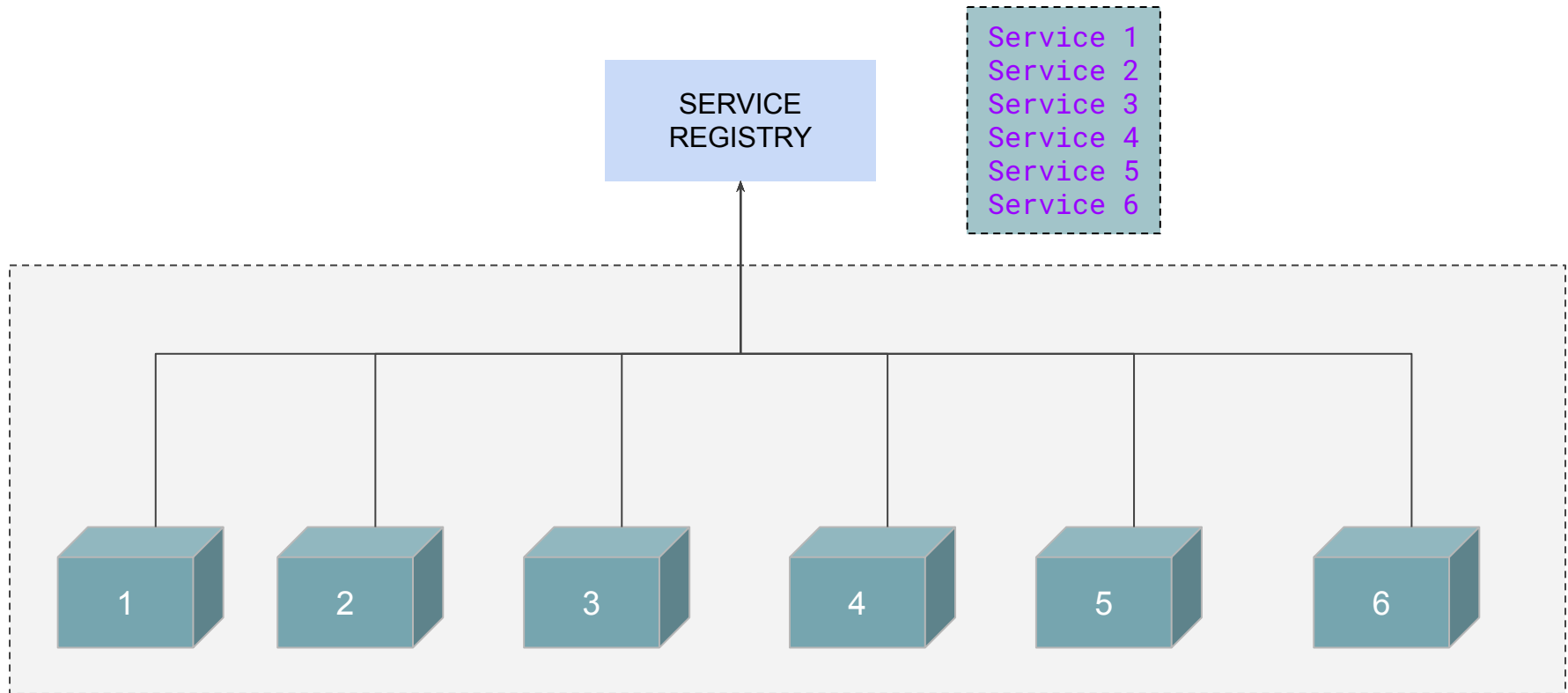
System Status

Environment	test
Data center	default

Current time	2025-03-22T12:12:41 +0530
Uptime	00:27
Lease expiration enabled	false
Renews threshold	1
Renews (last min)	0

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

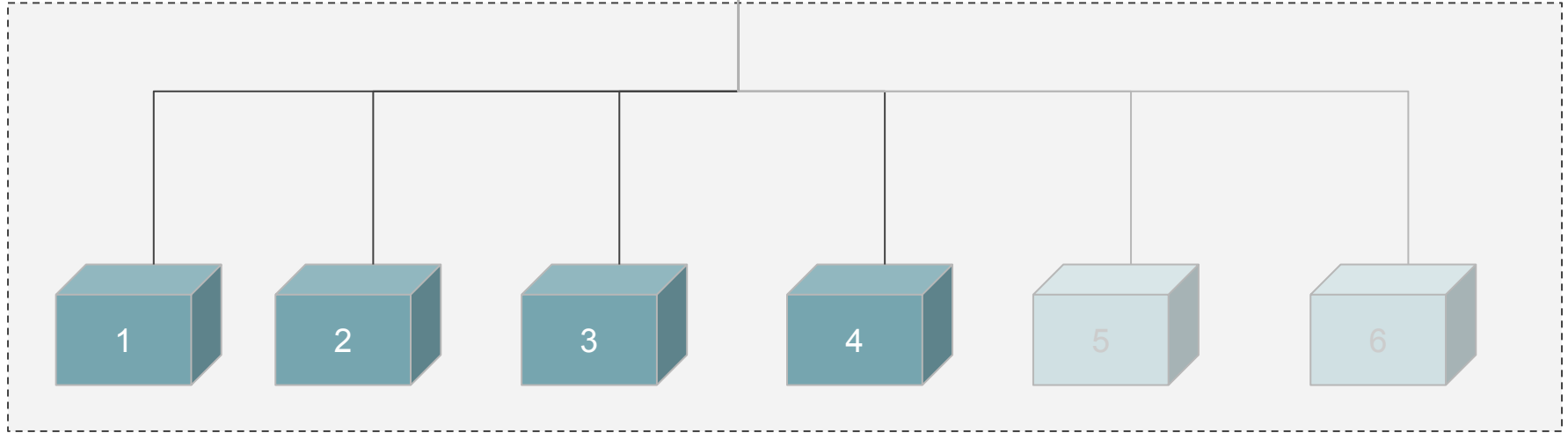
DS Replicas



Self
Preservation
Mode

SERVICE
REGISTRY

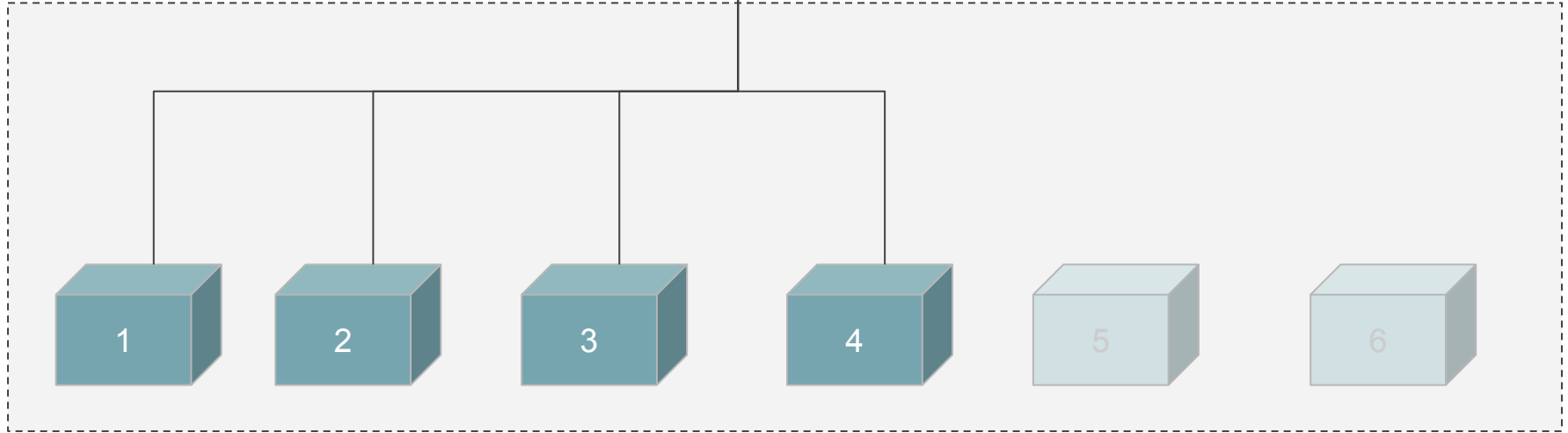
Service 1
Service 2
Service 3
Service 4
Service 5
Service 6



Self
Preservation
Mode

SERVICE
REGISTRY

Service 1
Service 2
Service 3
Service 4



Configurations for Self-Preservation

`eureka.server.enableSelfPreservation=true`

If disabled, Eureka will evict instances immediately when heartbeats drop, which can be risky in unstable networks

`eureka.server.self-preservation-percentage: 0.85`

This threshold is used to calculate the expected number of heartbeats per minute.

`eureka.instance.lease-renewal-interval-in-seconds=60`

This sets the interval at which a service instance sends a heartbeat (renewal) to the Eureka server.

Configurations for Self-Preservation

`eureka.server.eviction-interval-timer-in-ms=60*1000`

A scheduler runs at this interval (every 60 seconds) to check for and evict expired instances based on the lease-expiration-duration setting.

`eureka.server.renewal-threshold-update-interval-ms=15601000`

Every 15 minutes, Eureka recalculates the expected heartbeats per minute based on the current registry size.

Important Eureka Server Configuration Settings

Faisal Memon (EmbarkX)

Eureka Client Settings (Microservice Side)

Property	Default	Description
<code>eureka.client.register-with-eureka</code>	<code>true</code>	Whether the service should register itself with Eureka.
<code>eureka.client.fetch-registry</code>	<code>true</code>	Whether to fetch registry info from Eureka.
<code>eureka.instance.lease-renewal-interval-in-seconds</code>	<code>30</code>	How often (in seconds) the service sends heartbeats to Eureka.
<code>eureka.instance.lease-expiration-duration-in-seconds</code>	<code>90</code>	Time after which the service is removed if no heartbeats are received.
<code>eureka.instance.hostname</code>	Auto-detected	Used for service instance identification.

Eureka Server Settings

Property	Default	Description
<code>eureka.server.enable-self-preservation</code>	<code>true</code>	Prevents automatic removal of instances if many services go down at once.
<code>eureka.server.eviction-interval-timer-in-ms</code>	<code>60000</code> (60s)	How often Eureka cleans up stale instances.
<code>eureka.server.expected-client-renewal-interval-seconds</code>	<code>30</code>	Expected heartbeat interval from services.
<code>eureka.instance.prefer-ip-address</code>	<code>false</code>	If <code>true</code> , registers instances using their IP address instead of hostname.
<code>eureka.instance.instance-id</code>	Auto-generated	Custom identifier for the service instance.

Optimized Configuration for Faster Deregistration

```
eureka.instance.lease-renewal-interval-in-seconds=5  
# Send heartbeat every 5s
```

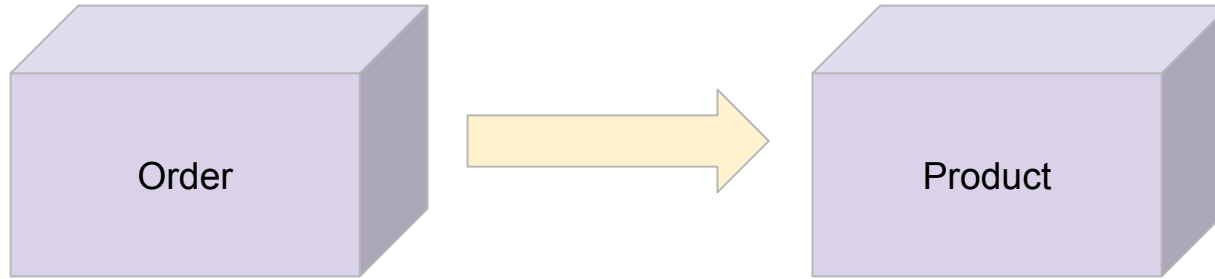
```
eureka.instance.lease-expiration-duration-in-seconds=10  
# Remove instance after 10s of no heartbeat
```

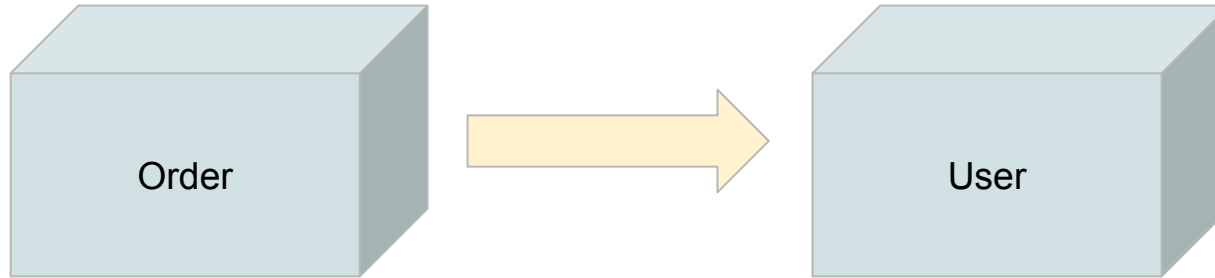

Remember

- **Shutting down is NOT enough** because Eureka removes instances only after a timeout (default: 90s)
- Explicitly unregistering helps **immediately remove** services and prevent requests from hitting a dead instance.
- Optimizing configuration can **speed up failover and service removal**.

Inter Service Communication Scenarios in Our Project

Faisal Memon (EmbarkX)





Observability

Faisal Memon (EmbarkX)



Observability is the ability to understand the internal state of a system by analyzing the data it produces - like logs, metrics, and traces.

Monitoring vs. Observability

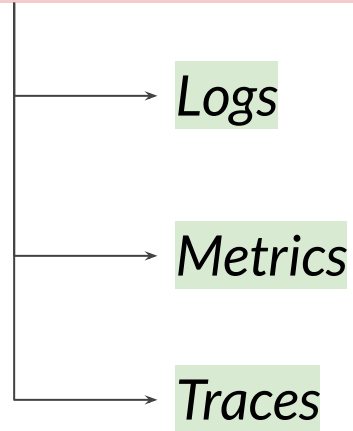
Feature	Monitoring	Observability
Definition	Watching system health and performance	Understanding why things happen
Approach	Collect predefined data	Explores unknown issues
Example	Checking if a server is running	Debugging why a request failed

Monitoring vs. Observability

*Monitoring is like a smoke
detector*

*Observability is like an
investigation*

Three Pillars of Observability



Logs - What Happened

→ *Logs are text-based records of events happening in a system.*

→ *Tools for logs - ELK Stack (Elasticsearch, Logstash, Kibana), Splunk*

→ *In a food delivery app, a log might look like this:*

`[2025-01-29 12:00:15] INFO OrderService: Order ID 1234 placed successfully.`

`[2025-01-29 12:00:20] ERROR PaymentService: Payment failed for Order ID 1234.`

Metrics - How is the System Performing

- Metrics are numerical values that help track system performance over time.
- Tools for Metrics - Prometheus + Grafana, Datadog, New Relic
- In a video streaming app, some important metrics could be:
 - CPU usage:** 80% (Is the system overloaded?)
 - Response time:** 500ms (Are users experiencing delays?)
 - Error rate:** 5% (Are too many requests failing?)

Traces - How Requests Flow Through the System

- *Traces track how a request moves across multiple microservices.*
- *Tools for Traces - Zipkin, Jaeger, OpenTelemetry*
- *In a **ride-sharing app** (like Uber), when a user books a ride, the request passes through multiple services:*
 - User Service** (Verify user)*
 - Ride Service** (Find available drivers)*
 - Payment Service** (Process payment)*
 - Notification Service** (Send confirmation)*

Why Observability is Critical in Microservices?

→ *Finding Problems Faster*

→ *Improving Performance*

→ *Better Debugging*

→ *Scaling Efficiently*

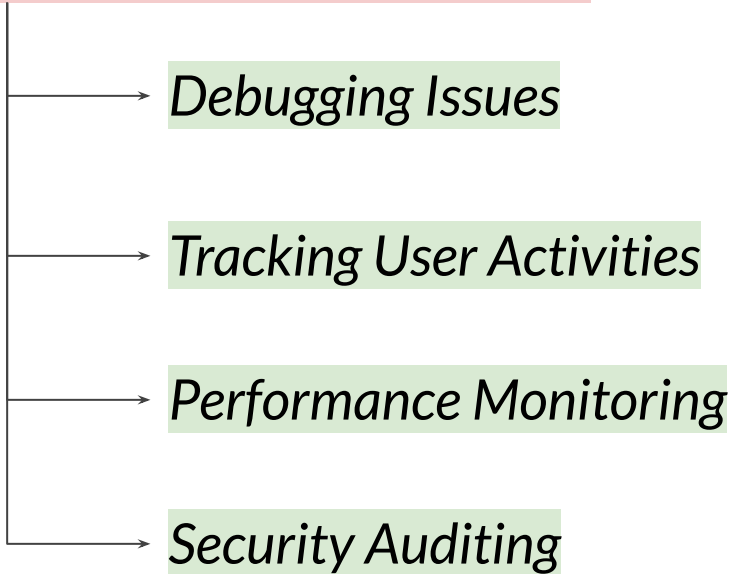
Observability ensures that
a microservices system
stays **healthy, fast, and
reliable.**

Logs

Faisal Memon (EmbarkX)

Logs are messages written
by applications

Why Are Logs Important?



```
graph LR; A[Why Are Logs Important?] --> B[Debugging Issues]; A --> C[Tracking User Activities]; A --> D[Performance Monitoring]; A --> E[Security Auditing];
```

Debugging Issues

Tracking User Activities

Performance Monitoring

Security Auditing

```
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController
```

```
public class OrderController {
```

```
    private static final Logger logger =  
    LoggerFactory.getLogger(OrderController.class);
```

```
    @GetMapping("/order")
```

```
    public String placeOrder(@RequestParam String item) {
```

```
        logger.info("Order received for item: {}", item); // Log message
```

```
        return "Order placed for " + item;
```

```
    }
```

```
}
```

```
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController
```

```
public class OrderController {
```






```
    private static final Logger logger =  
LoggerFactory.getLogger(OrderController.class);
```

```
    @GetMapping("/order")  
    public String placeOrder(@RequestParam String item) {  
        logger.info("Order received for item: {}", item); // Log message  
        return "Order placed for " + item;  
    }
```

```
}
```

```
2025-02-07 12:00:15 INFO OrderController - Order received for item: Laptop
```

Types of Logs (Logging Levels)

Log Level	Purpose	Example Scenario
TRACE 	Very detailed, used for debugging	Step-by-step execution of a function
DEBUG 	Useful for developers, but not in production	Checking if data is correct
INFO 	General system events	Order placed, user logged in
WARN 	Something is wrong, but not critical	Payment took too long
ERROR 	Something failed	Payment failed, database error

Normal Logging

Each microservice writes logs in separate files, but searching for an issue requires checking logs in multiple places.

Centralized Logging

*Instead of checking multiple log files, we collect logs from all services into one central system like **Elasticsearch + Kibana** (ELK Stack) or **Loki + Grafana**.*

Logging Best Practices

Always Use Structured Logging

→ *Structured logs are easier to search in centralized logging systems.*

```
logger.info("Order placed for item: {}", item);
```

Use the Right Log Level

Use **INFO** for general events (User login, Order placed)

Use **DEBUG** for development testing

Use **ERROR** only for real failures

```
logger.info("Payment failed for order 123"); // This should be ERROR  
logger.error("Payment failed for order: {}", orderId);
```

Filter Sensitive Data in Logs

→ *Never expose sensitive data in logs*

```
logger.info("User logged in with password: {}", password); // Not Good  
logger.info("User logged in: {}", userId); // Good
```

Best Practices

- *Always use structured logging*
- *Use the right logging levels (INFO, DEBUG, WARN, ERROR)*
- *Filter sensitive data in logs*

Understanding Logging in Spring Boot

Faisal Memon (EmbarkX)

Spring Boot **does not**
log messages directly

Your Code

```
logger.info("Message")
```



SLF4J (Logging Facade)

Acts as a middleman



Logging Framework

*Logback, Log4j2, JUL
(Default: Logback)*



Log Output

Console, File, etc.

Component	Role
SLF4J (Simple Logging Facade for Java)	Interface (middleman) for logging. It forwards logs to the actual logging framework.
Logback (Default in Spring Boot)	Processes logs , applies formatting, filtering, and decides where to send logs.
Log Output	The final destination for logs (Console, File, Database, etc.).

Metrics and Monitoring in Microservices With Grafana

Faisal Memon (EmbarkX)

Metrics - How is the System Performing

- Metrics are quantitative measurements used to track the performance, health, and behavior of a system
- Metrics help you measure things like how fast your services are responding, how much memory they are using, how many requests they are handling, and whether there are any errors or failures.

Why Are Metrics Important?



Microservices

Your Applications



Prometheus

Metrics Collection



Grafana

Visualizing Metrics



Alerts

Optional

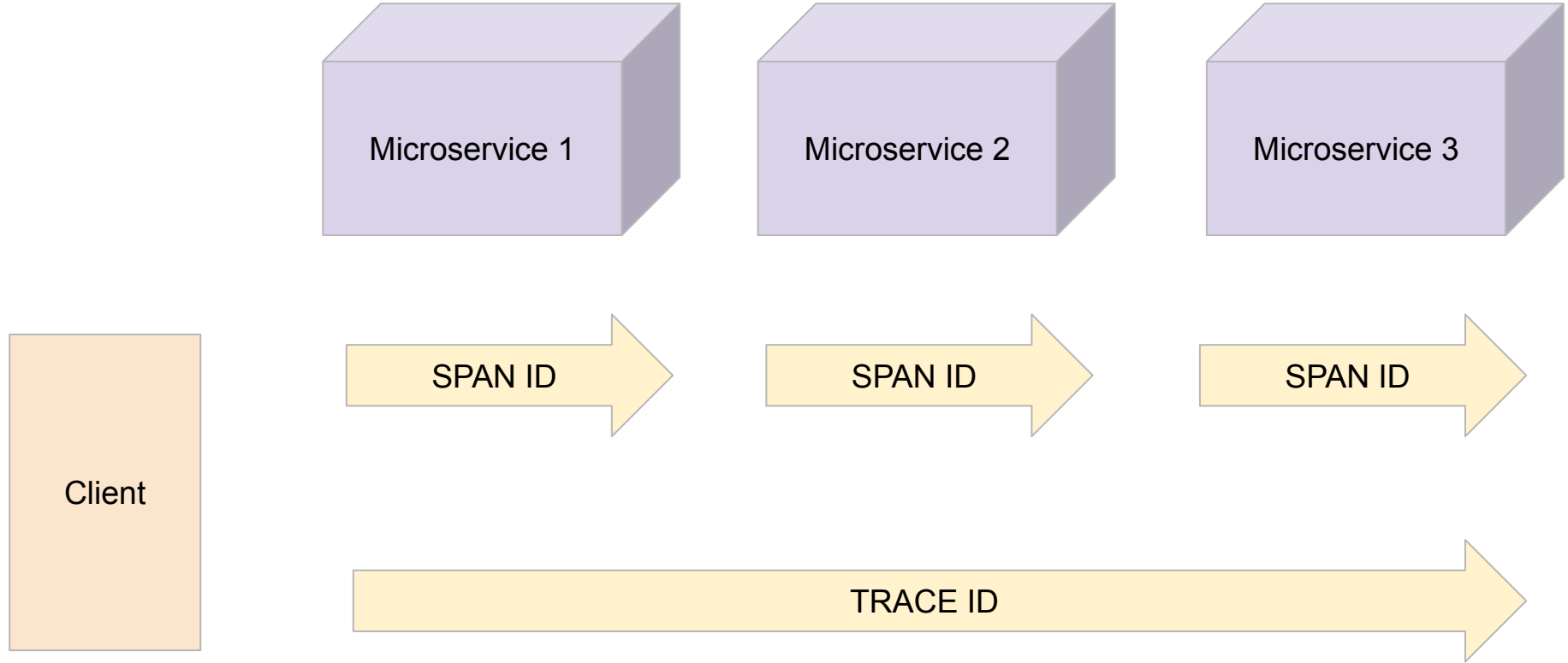
Distributed Tracing

Faisal Memon (EmbarkX)

A way to track requests as they
move through **different services** in
a microservices architecture

Imagine you're using a food delivery app...

1. You place an order → The request goes to the Order Service.
2. The Order Service talks to the Payment Service for payment.
3. The Payment Service communicates with the Inventory Service to check stock.
4. The Inventory Service updates the Database.



Order Placed (Trace ID: 12345)

└─ Order Service (Span: 1)

| └─ Payment Service (Span: 2)

| └─ Inventory Service (Span: 3)

| └─ Notification Service (Span: 4)

└─ Database Update (Span: 5)

Benefits

```
graph LR; A[Benefits] --> B[Find Performance Issues]; A --> C[Debugging Failures]; A --> D[Improve Reliability]; A --> E[Optimize Microservices];
```

Find Performance Issues

Debugging Failures

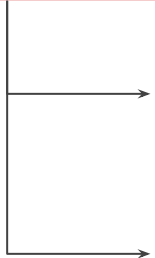
Improve Reliability

Optimize Microservices

The Challenge

Faisal Memon (EmbarkX)

Challenges Right Now



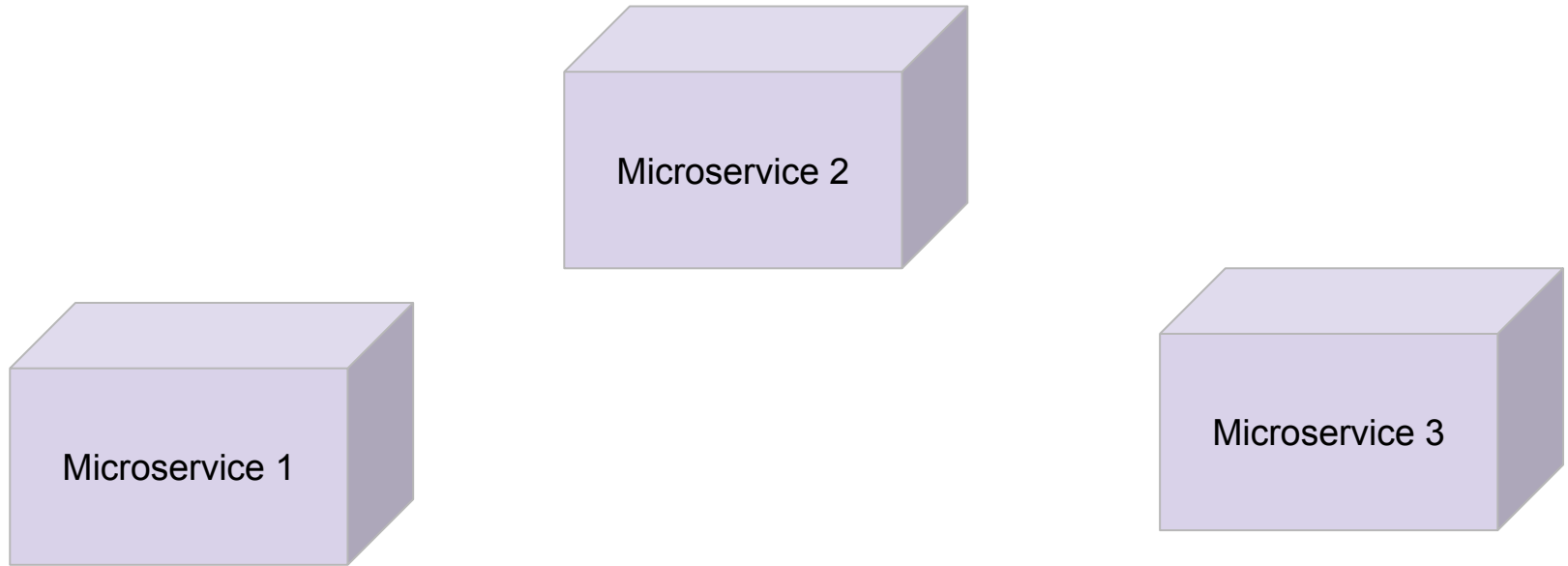
```
graph LR; A[Challenges Right Now] --> B[Communication Issue [Remember every service URLs]]; A --> C[Security + Authentication]
```

Communication Issue [Remember every service URLs]

Security + Authentication

Introduction to API Gateways

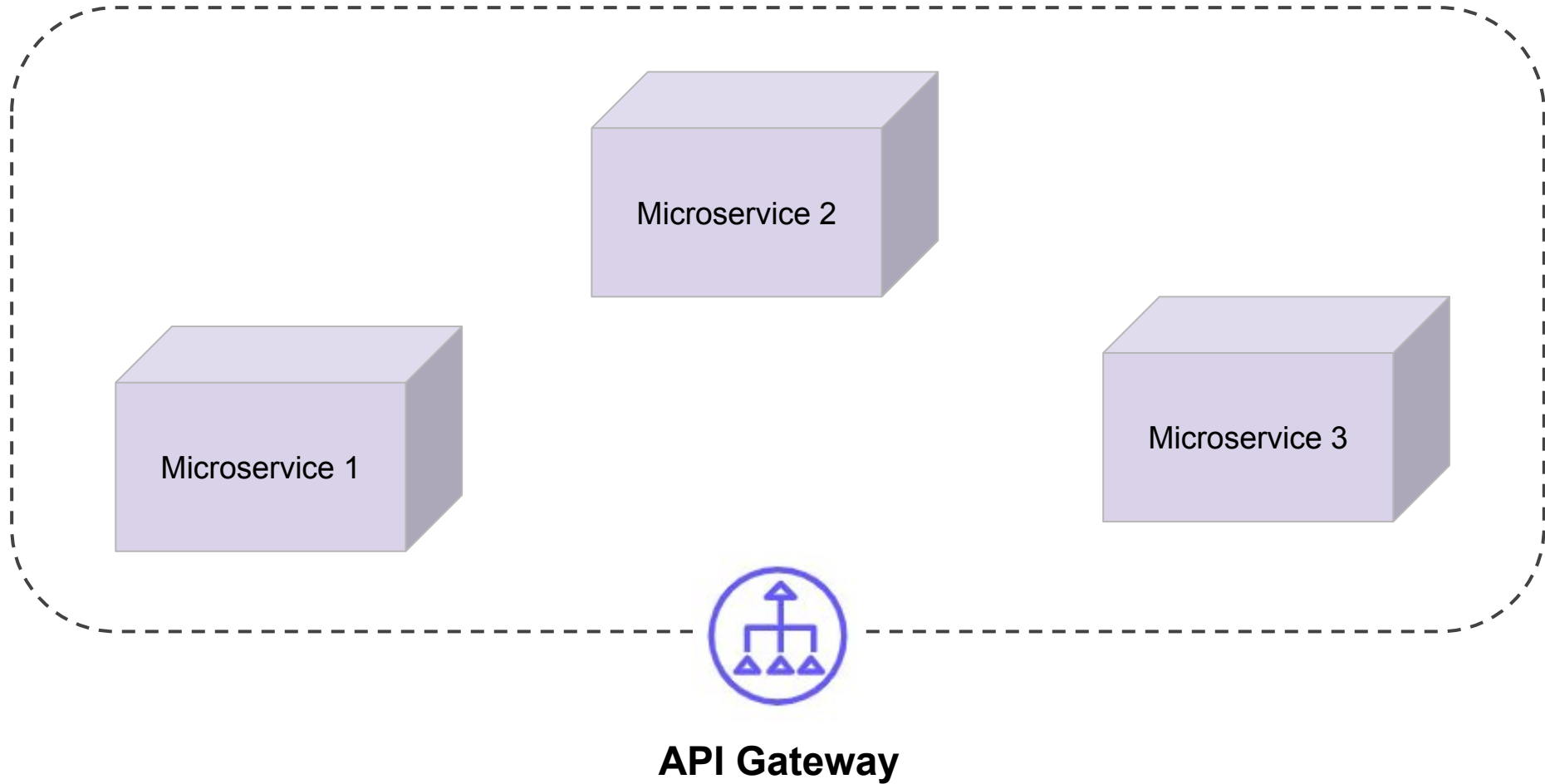
Faisal Memon (EmbarkX)



**Microservices
Architecture**

=

A monolithic application is broken down into
smaller, loosely coupled services



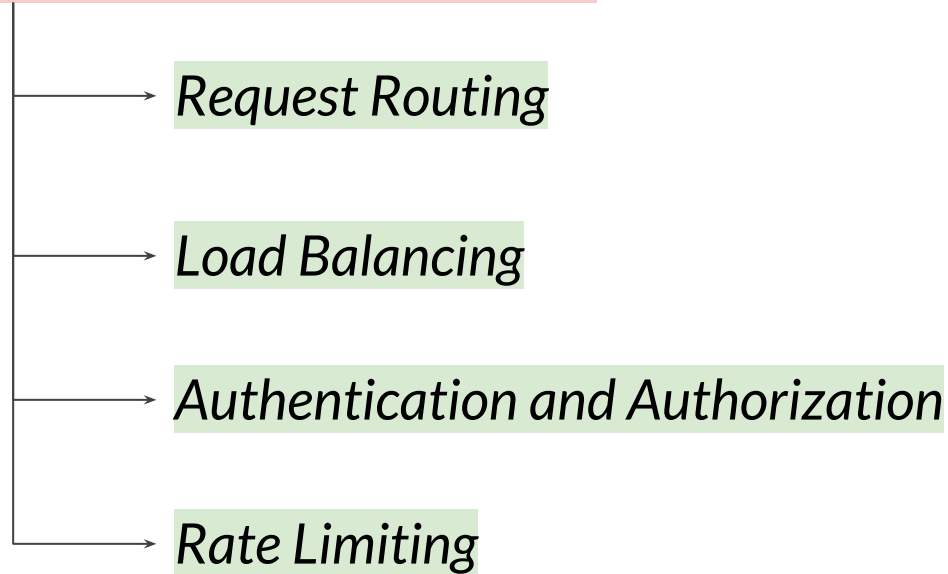
Advantages

- *It encapsulates the internal system architecture*
- *Handle cross-cutting concerns like security, load balancing, rate limiting, and analytics*
- *Can authenticate incoming requests and pass only valid requests to the services*
- *Can aggregate responses from different microservices*
- *Plays a crucial role in simplifying client interactions and managing cross-cutting concerns*

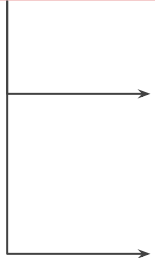
API Gateway Functions

Faisal Memon (EmbarkX)

Capabilities of API Gateway



Capabilities of API Gateway



Request and Response Transformation

Aggregation of Data from Multiple Services

Routes in yml/properties vs Java Code

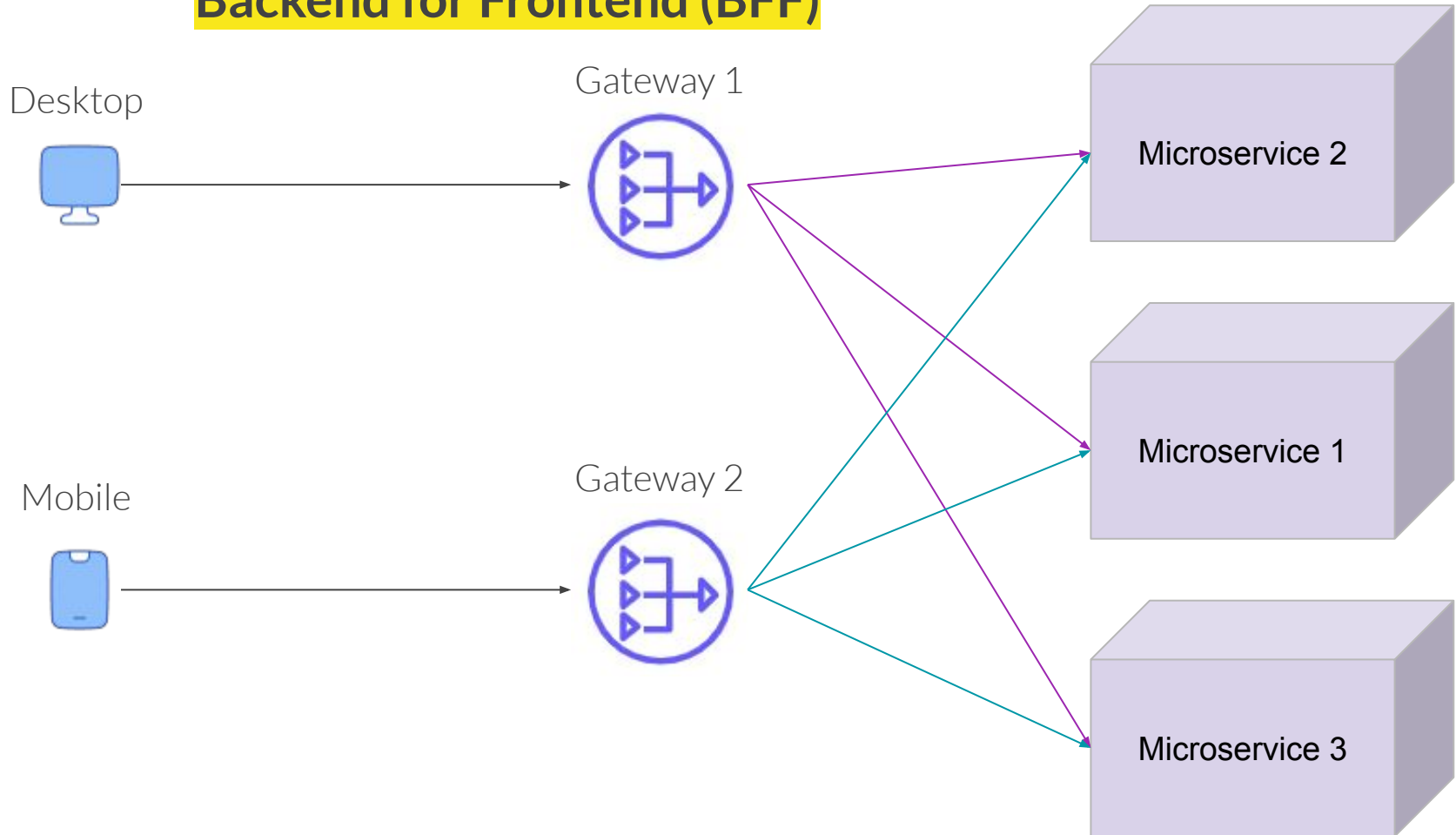
Faisal Memon (EmbarkX)

Feature	YAML Configuration	Java Configuration
Complex Filters	✗ Not possible	✓ Fully customizable
Readability	✓ Easier for simple routes	✗ More code required
Custom Logic (e.g., JWT Auth, Rate Limiting)	✗ Limited	✓ Flexible

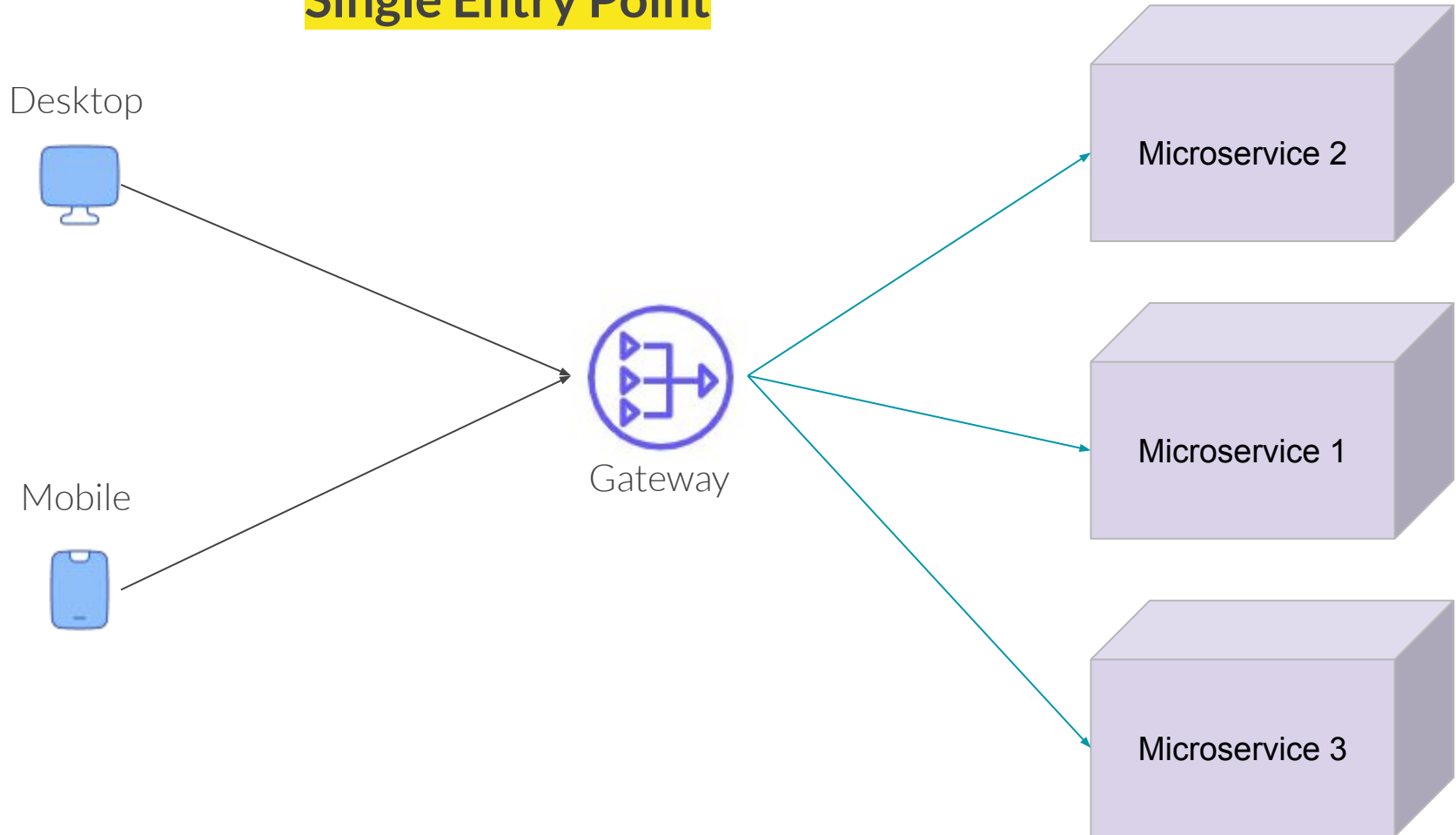
API Gateway Patterns & Best Practices

Faisal Memon (EmbarkX)

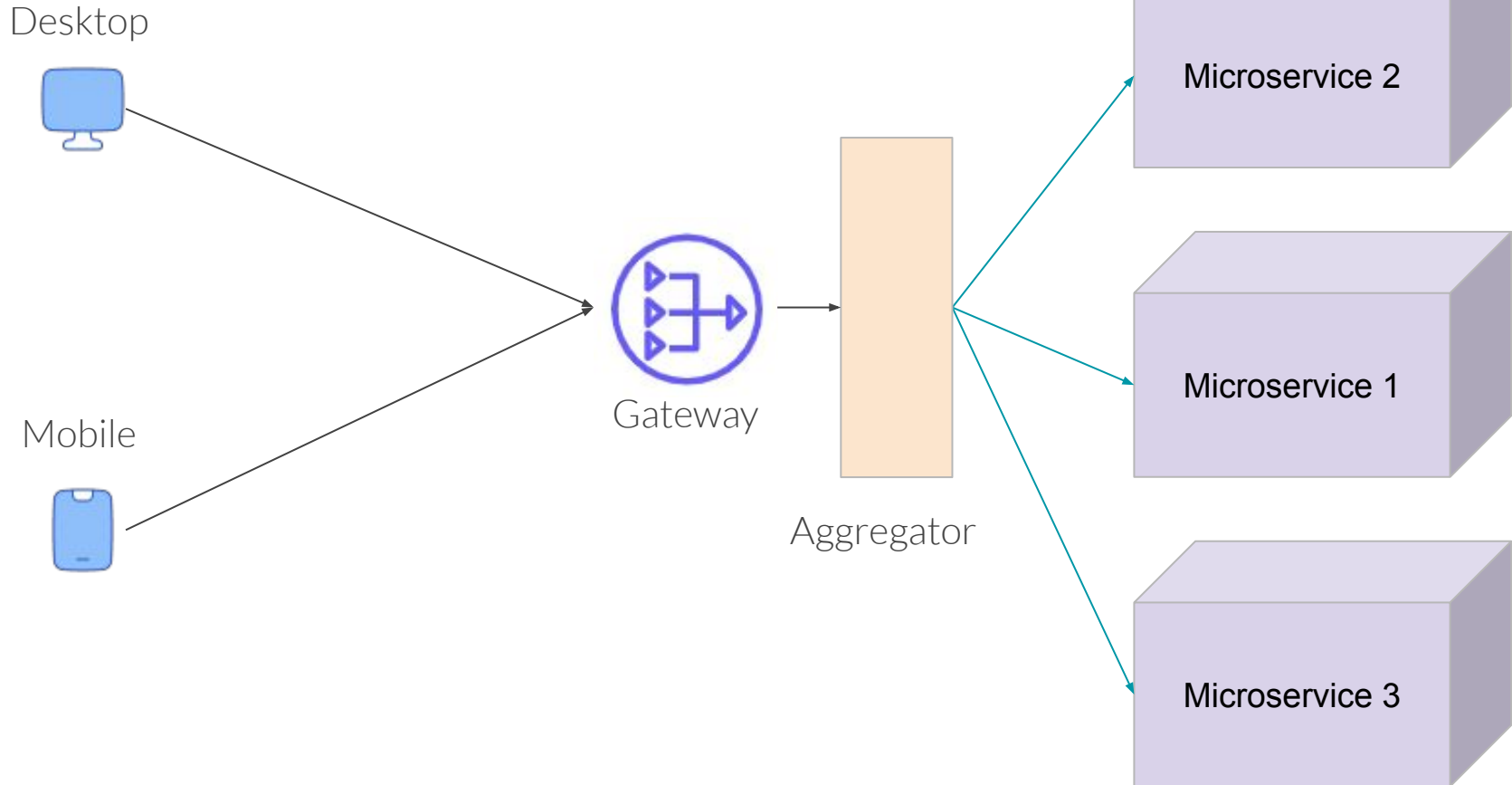
Backend for Frontend (BFF)



Single Entry Point



Aggregation Gateway



Best Practices for API Gateways



Use Load Balancing

Implement Authentication & Security

Handle Errors Gracefully

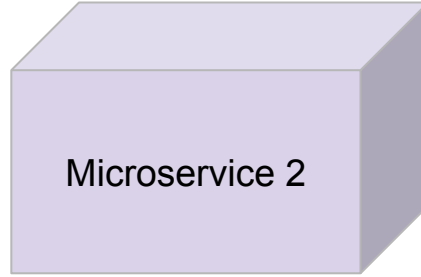
Introduction to Fault Tolerance

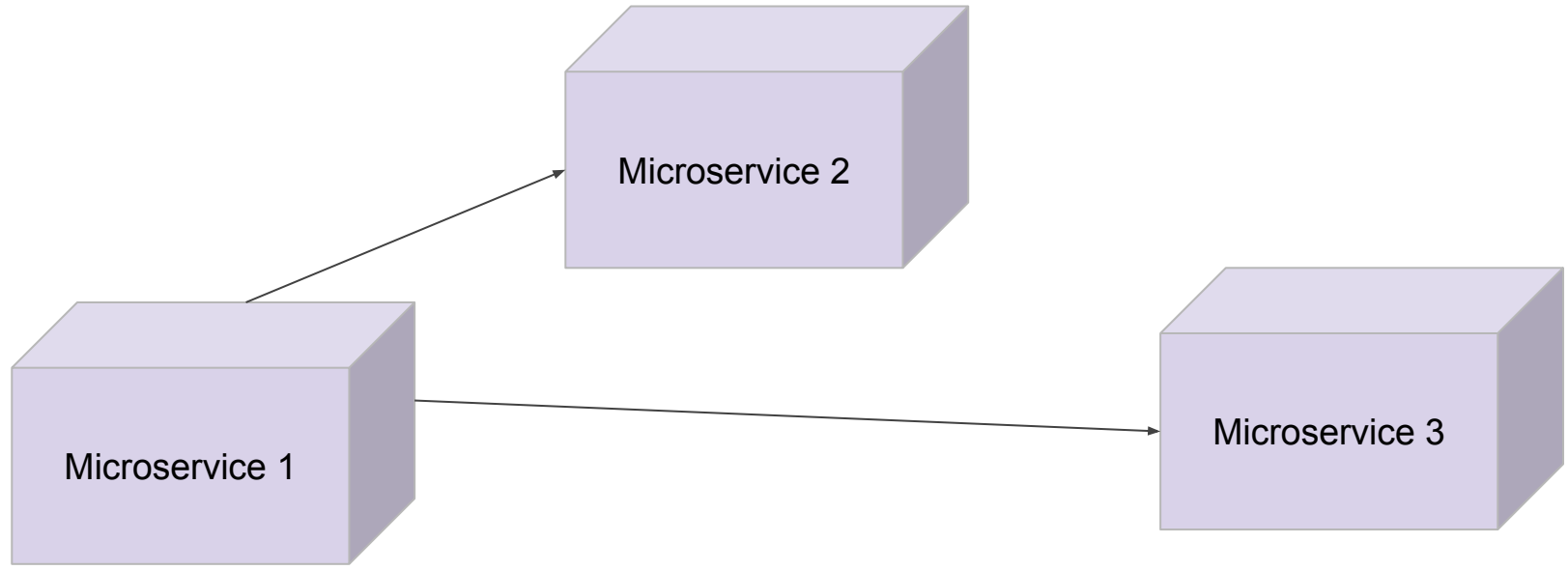
Faisal Memon (EmbarkX)

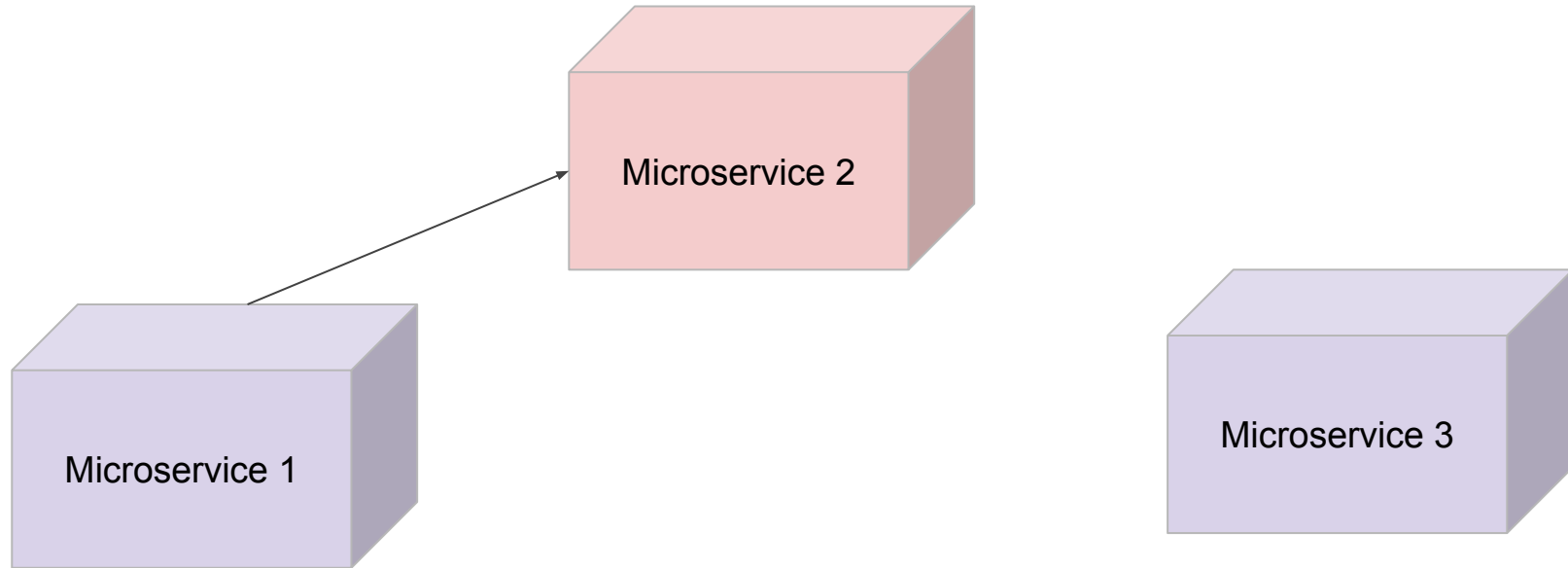
Fault Tolerance

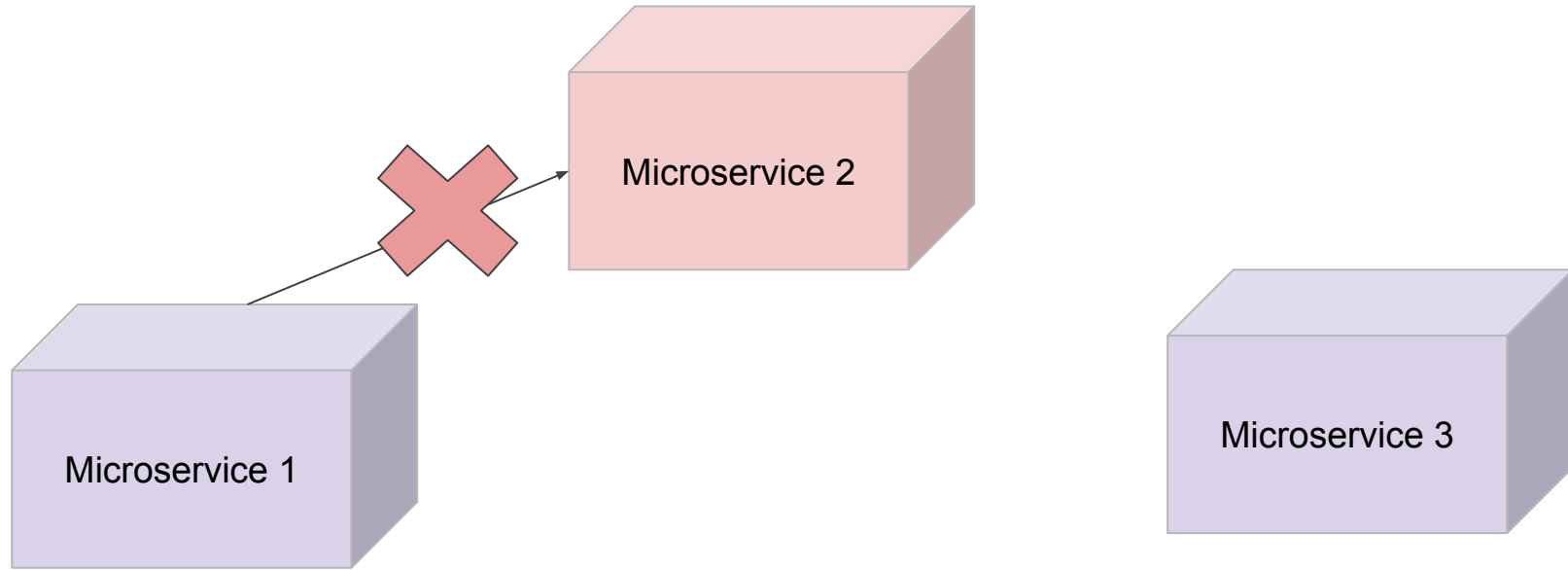
Ability to continue operating without interruption











Fault Tolerance

- *Fault tolerance prevents small failures from crashing the entire system.*
- *It helps microservices work smoothly even if one part fails.*

Understanding Failures and Strategies for Fault Tolerance

Faisal Memon

Failures in Microservices



```
graph LR; A[Failures in Microservices] --> B[Network Failures]; A --> C[Service Unavailability]; A --> D[High Latency Issues (Slow Response)];
```

Network Failures

Service Unavailability

High Latency Issues (Slow Response)

Network Failures

What and why?

Networks can fail because of bad connections, traffic overload, hardware issues

How to handle it?

- *Retry Mechanism – Try again a few times before giving up. If the Payment Service doesn't respond, wait 2 seconds and try again.*
- *Timeouts – Don't wait forever. Stop waiting after 3-5 seconds and show a message.*
- *Fallback Response – Show a friendly message like: "Payment service is currently unavailable. Please try again later."*

Service Unavailability

What and why?

A microservice stops working completely because crashes, maintenance, server failures.

How to handle it?

- *Circuit Breaker Pattern – If a service keeps failing, stop calling it for some time and show a message.*
- *Load Balancing – Have multiple copies of a service so if one crashes, another can handle the request.*
- *Service Discovery – Use tools like Eureka to find healthy services and avoid calling the broken ones.*

High Latency Issues

What and why?

Some services take too long to respond because of too many requests, slow database queries, external API delays

How to handle it?

- *Caching – Store frequently used data (like product details) in memory (Redis, Hazelcast) so it loads instantly.*
- *Asynchronous Processing – Instead of making the user wait, send the request to a queue (Kafka, RabbitMQ) and process it in the background.*
- *Timeouts & Fallbacks – If the service is slow, stop waiting after 3-5 seconds and show a default list of products.*

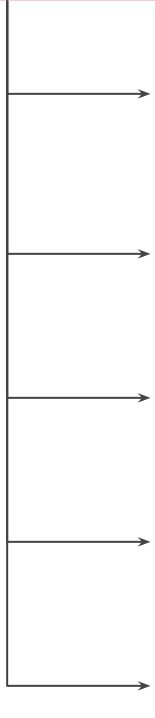
Introduction to Resilience4J

Faisal Memon

Resilience

Ability or capacity to recover quickly from difficulties

Techniques



Retries

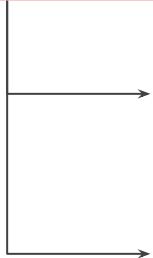
Rate Limiting

Bulkheads

Circuit Breakers

Fallbacks

Techniques



```
graph TD; Techniques[Techniques] --> Timeouts[Timeouts]; Techniques --> GracefulDegradation[Graceful Degradation];
```

Timeouts

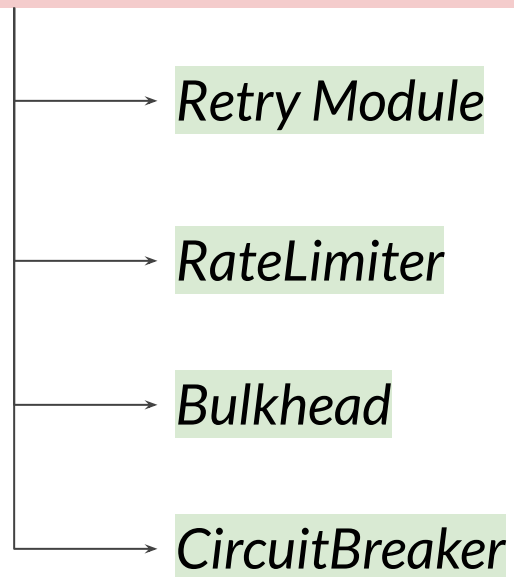
Graceful Degradation

Resilience4J is a lightweight, easy-to-use fault
tolerance library

Why it's a good choice?

- *Easy integration with Spring Boot*
- *Built for functional programming paradigms*

Resilience4J Modules



Retry Module

- *It's not uncommon for a network call or a method invocation to fail temporarily*
- *We might want to retry the operation a few times before giving up*
- *Retry module enables to easily implement retry logic in our applications*

RateLimiter

- *We might have a service which can handle only a certain number of requests in given time*
- *RateLimiter module allows us to enforce restrictions and protect our services from too many requests*

Bulkhead

- *Isolates failures and prevents them from cascading through the system*
- *Limit the amount of parallel executions or concurrent calls to prevent system resources from being exhausted*

CircuitBreaker

- *Used to prevent a network or service failure from cascading to other services*
- *Circuit breaker 'trips' or opens and prevents further calls to the service*

Retry Pattern

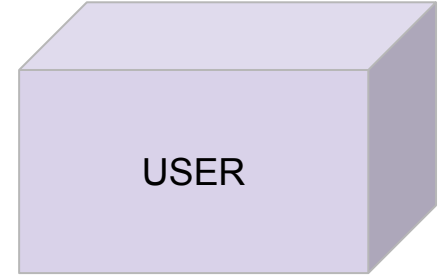
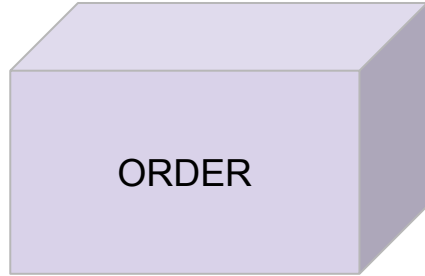
Faisal Memon

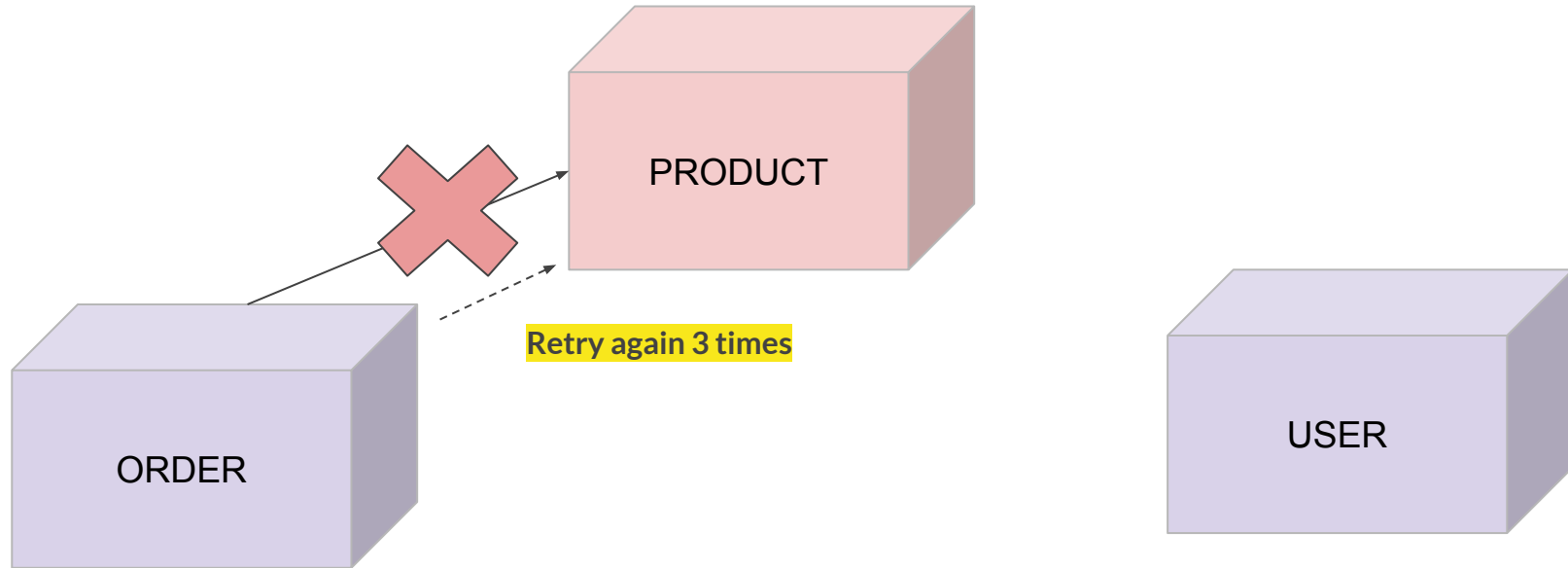
Retry Mechanism

- *A retry mechanism means trying again if something fails the first time*
- *Handle temporary issues and prevents unnecessary failures*
- *Imagine retrying to send failed message on -
WhatsApp*

When to use and when to avoid?

- *Retries are helpful when failures are temporary like bad network, service overload*
- *Retries are not always helpful when there is permanent failure*





```
resilience4j:
  retry:
    instances:
      exampleRetry:
        maxAttempts: 3
        waitDuration: 500ms
        intervalFunction:
          type: exponential
          initialInterval: 500ms
          multiplier: 1.5
          maxInterval: 5s
        retryExceptions:
          - java.io.IOException
          - java.sql.SQLException
        ignoreExceptions:
          - com.example.exception.NonRetriableException
        failAfterMaxAttempts: false
```

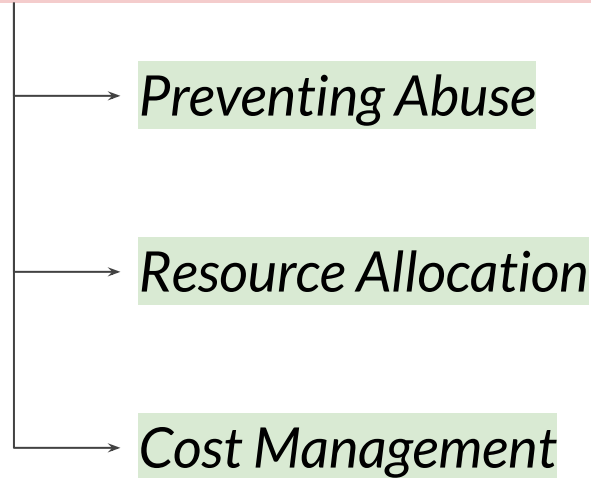

What Is Rate Limiting and Why Is It Needed?

Faisal Memon (EmbarkX)

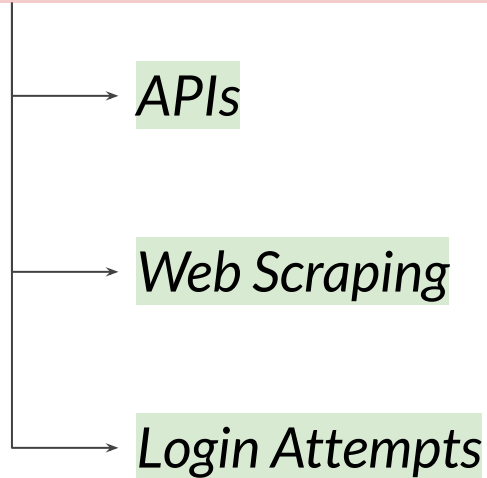
Rate Limiting

Rate limiting is a technique for limiting network traffic.

Importance of Rate Limiting



Use Cases of Rate Limiting



Distributed Denial of Service (DDoS) attacks

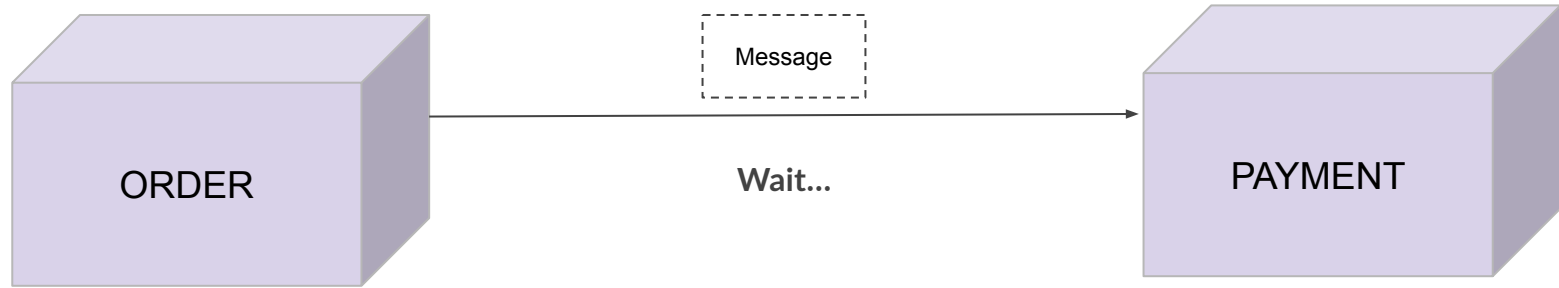
Rate Limiting with Resilience4J in Spring Boot

Thank you

Introduction to Asynchronous Communication

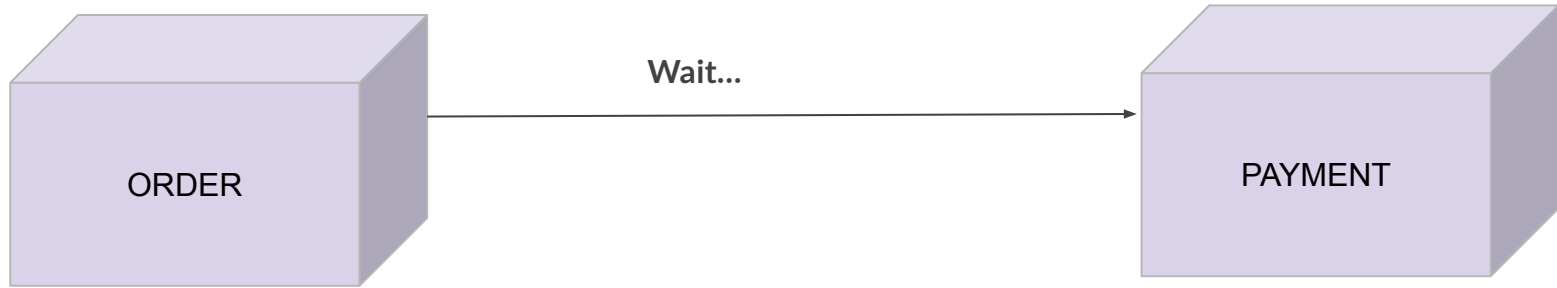
Faisal Memon (EmbarkX)











The Need



```
graph LR; A["The Need"] --> B["Better Performance"]; A --> C["Fault Tolerance"]; A --> D["Scalability"]; A --> E["Loose Coupling"];
```

Better Performance

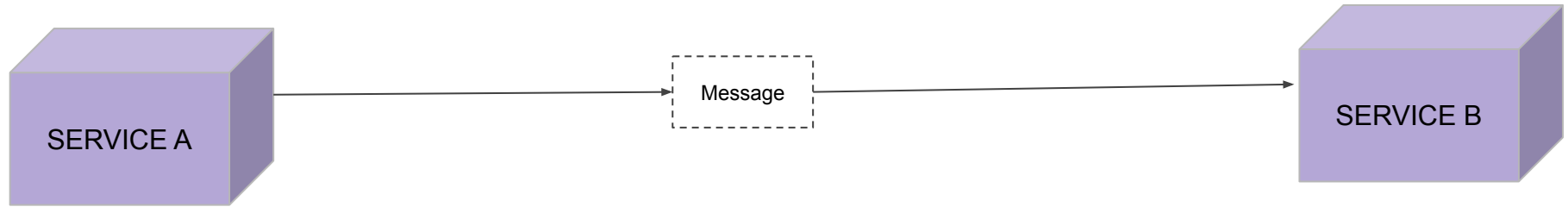
Fault Tolerance

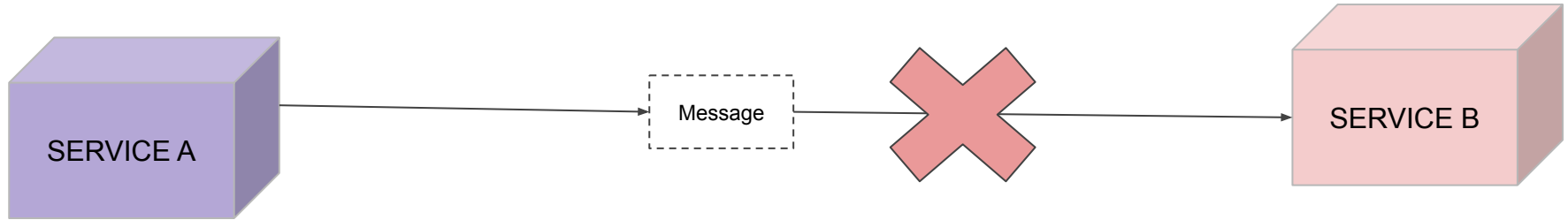
Scalability

Loose Coupling

What are Message Queues?

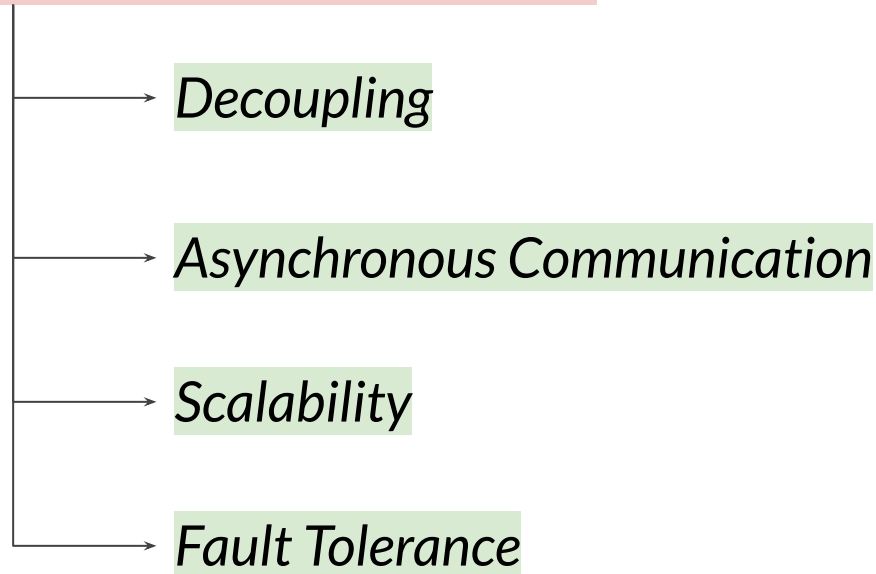
Faisal Memon (EmbarkX)



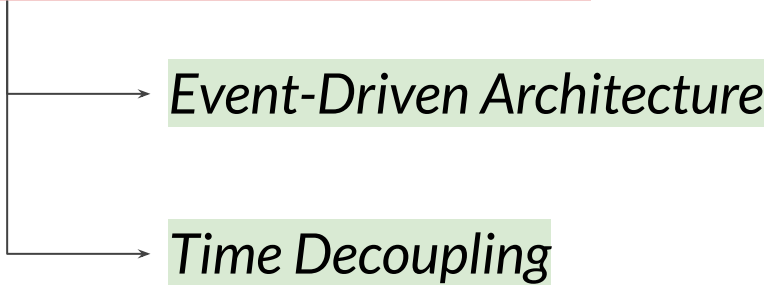




Need for Message Queues



Need for Message Queues



```
graph LR; A[Need for Message Queues] --> B[Event-Driven Architecture]; A --> C[Time Decoupling];
```

Event-Driven Architecture

Time Decoupling

Message Queues

A message queue is a form of asynchronous service-to-service communication used in serverless and microservices architectures



Demonstrating Importance of Message Queues

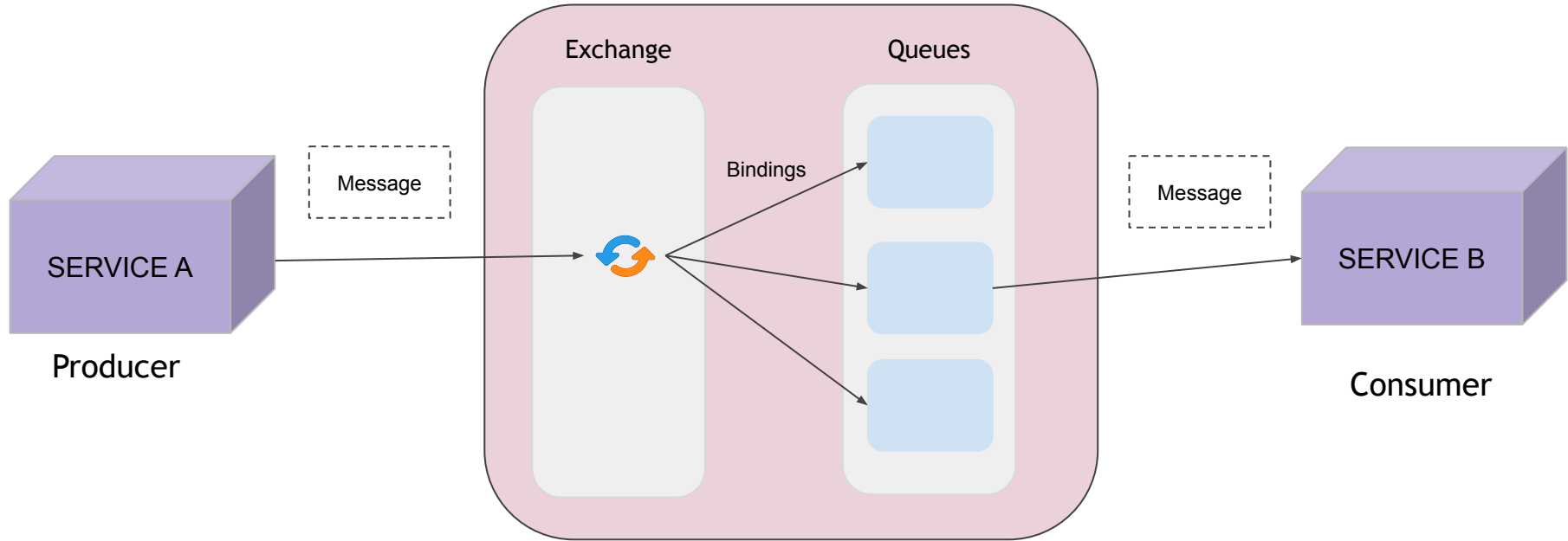
Faisal Memon (EmbarkX)



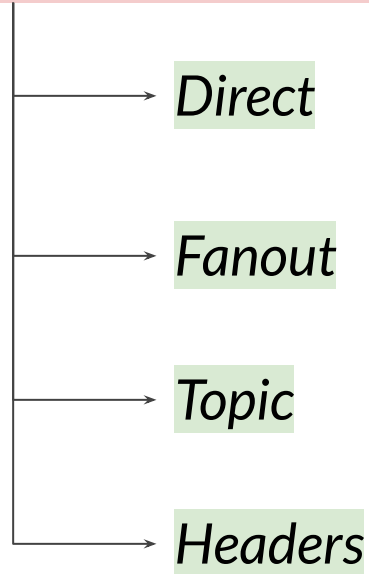
Messaging Exchanges and Its Types

Faisal Memon (EmbarkX)

What is an Exchange?

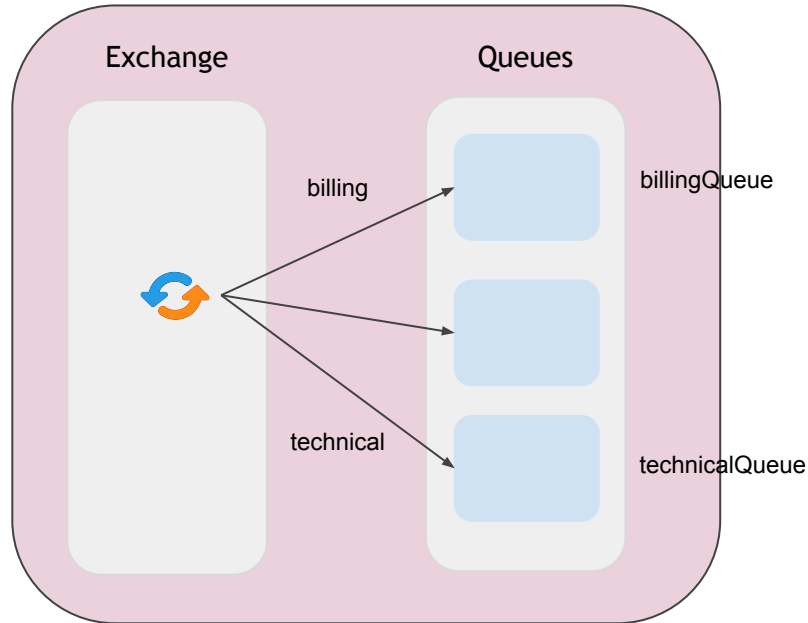


Message Exchange Types



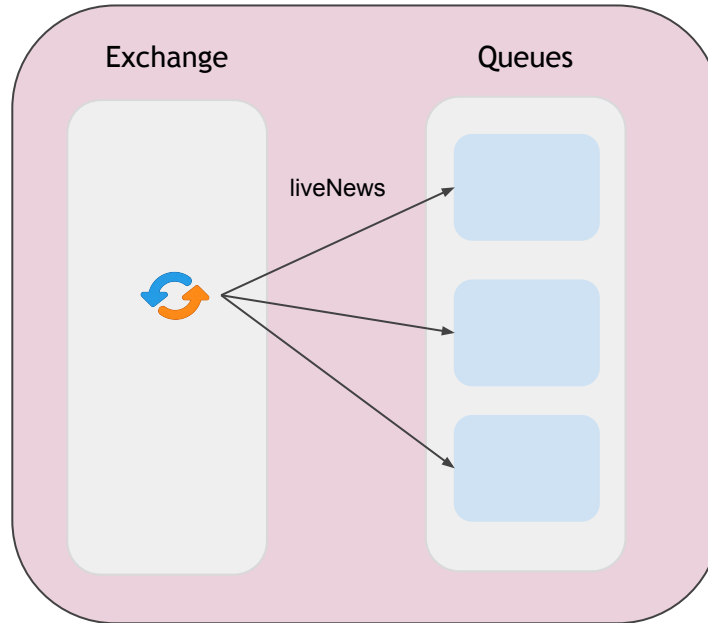
Direct Exchange

Routes messages to specific queue based on routing key



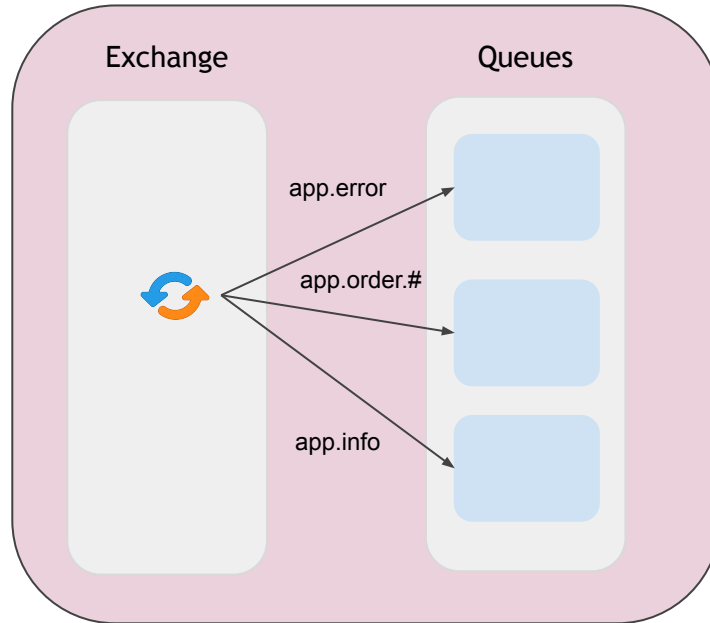
Fanout Exchange

A Fanout Exchange broadcasts every message to all bound queues.



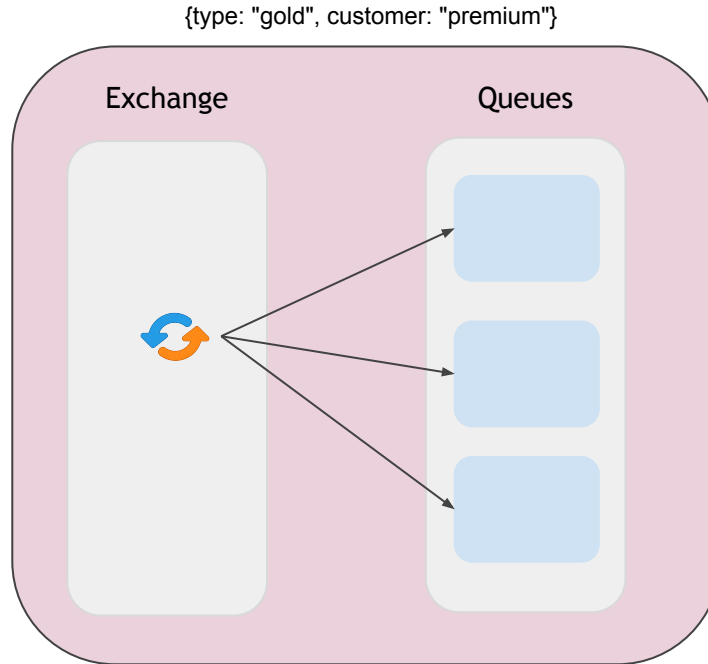
Topic Exchange

A Topic Exchange routes messages dynamically based on patterns in routing keys.



Headers Exchange

A Headers Exchange routes messages based on key-value pairs in message headers.



Summary of Exchange Types

Exchange Type	Routing Key?	Message Delivery	Use Case
Direct Exchange	✓ Yes	Sent to one matching queue	Order Processing, Support Tickets
Fanout Exchange	✗ No	Sent to all queues	Notifications, Broadcast Messages
Topic Exchange	✓ Yes (with patterns)	Sent to queues matching patterns	Logging, Event-Driven Systems
Headers Exchange	✗ No (Uses headers)	Sent to queues matching headers	Targeted Marketing, Metadata-Based Routing

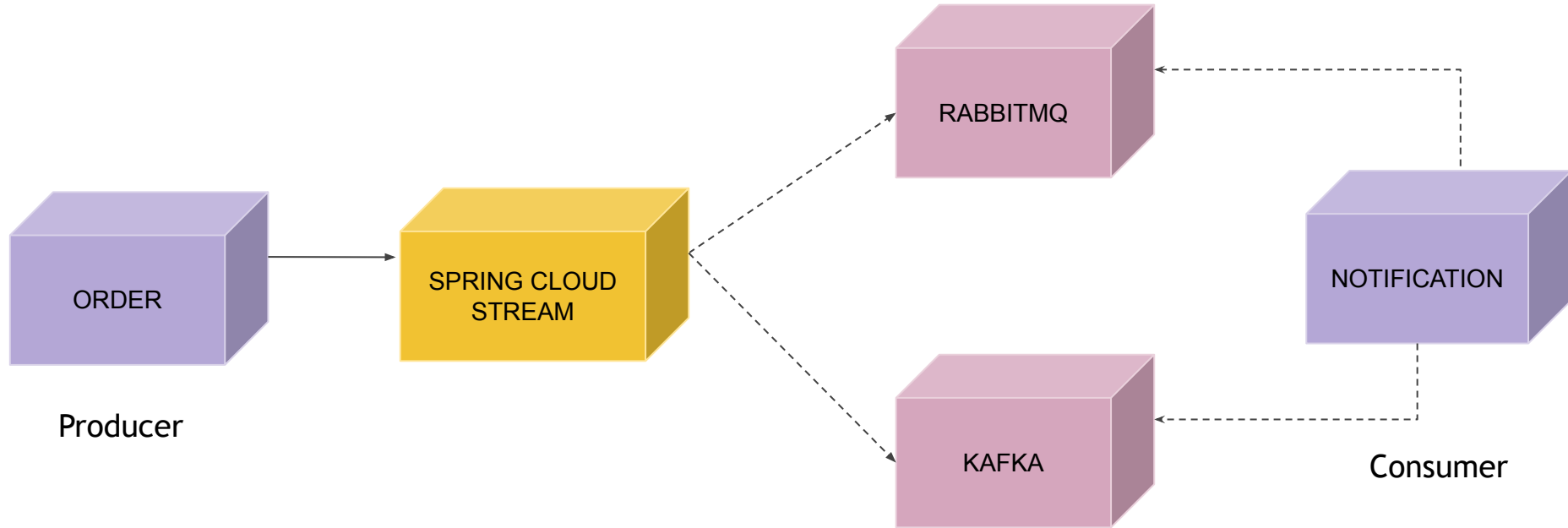
Section Overview

Faisal Memon (EmbarkX)



Using Spring Cloud Functions and Spring Cloud Streams

Faisal Memon (EmbarkX)

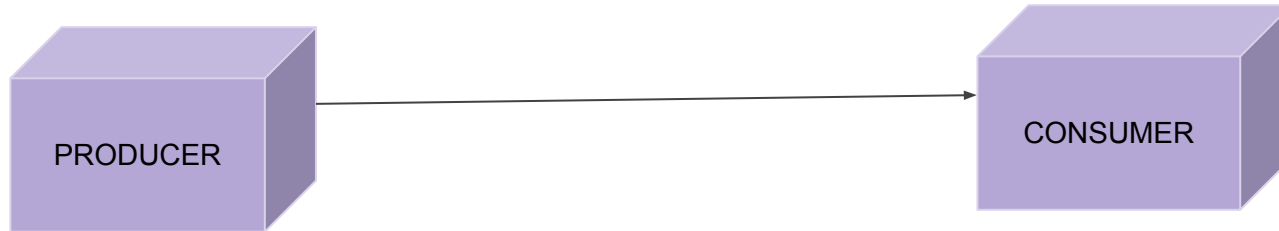


Different Messaging Models

Faisal Memon (EmbarkX)

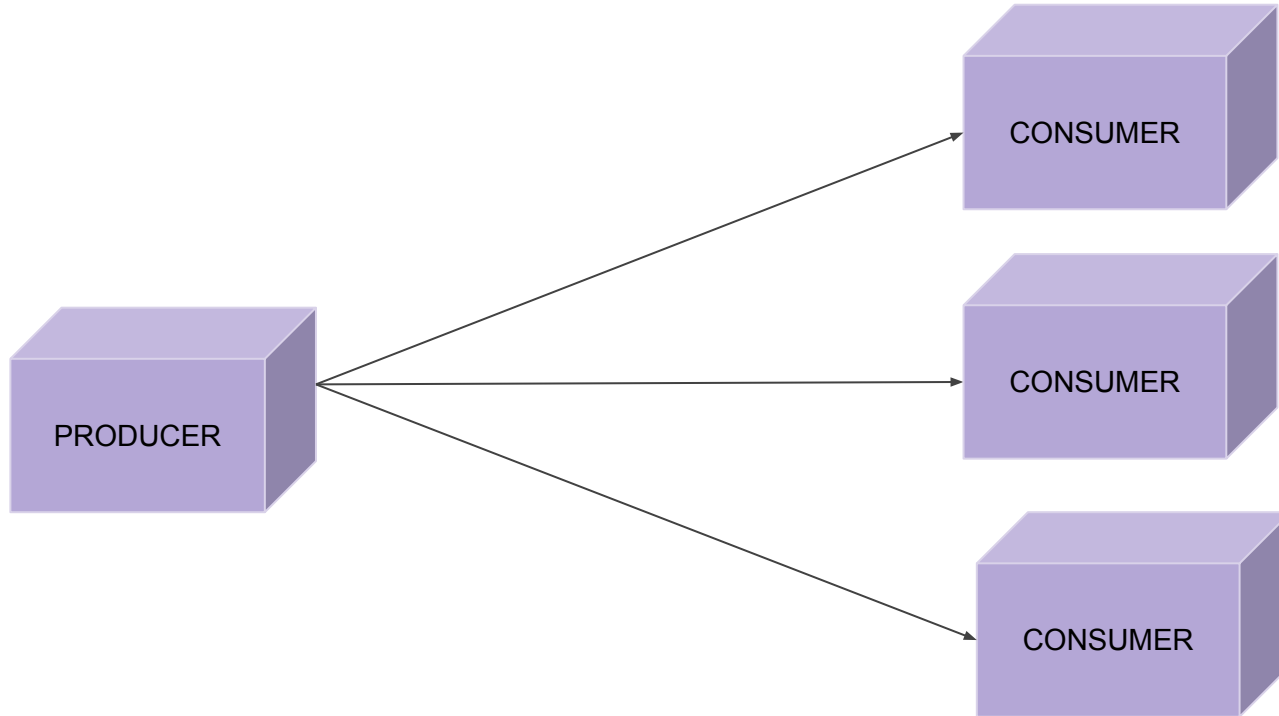
Queue (Point-to-Point Messaging Model)

One message is delivered to one consumer only



Publish-Subscribe (Pub-Sub Model)

Messages are broadcasted to multiple subscribers.



RabbitMQ and Kafka

→ *RabbitMQ is mainly built around Queue (Point-to-Point) but it can do Pub-Sub using fanout/topic exchanges.*

→ *Kafka is naturally built for Pub-Sub (events are broadcast to anyone interested), though it can also mimic queue-like behavior.*

Things to Remember About Kafka

Faisal Memon (EmbarkX)

Remember About Kafka

→ *Kafka is NOT a Database*

→ *You Can't Search in Kafka*

RabbitMQ vs Kafka

Faisal Memon (EmbarkX)

Feature	RabbitMQ	Kafka
Message Model	Traditional messaging (push-based).	Streaming platform (pull-based, event log).
Delivery Style	Messages are pushed to consumers.	Consumers pull messages when ready.
Persistence	Messages are persisted until delivered or expired.	Events are stored for a fixed time (like 7 days) even after consumption.
Ordering	Per queue — not global.	Per partition — very strict ordering inside partitions.
Scaling	Good for many queues and many consumers; not optimized for millions/sec throughput.	Designed for huge scale, high-throughput (millions of events/sec).
Use Case	Tasks, commands, small reliable messages, RPC-style comms.	Event sourcing, real-time analytics, data streaming pipelines.

Feature	RabbitMQ	Kafka
Acknowledgement	ACK/NACK per message.	Offset tracking — consumers remember their read position.
Latency	Very low (good for real-time communication).	Slightly higher, optimized for huge volume and replaying old messages.
Message Replay	Hard — once consumed and deleted, can't get it back.	Easy — consumer can re-read old events from Kafka.
Built-in Routing	Exchanges (direct, fanout, topic) give smart routing.	No built-in routing; you manage topics and partitions.
Streaming Support	Yes (newer versions) via RabbitMQ Streams plugin — but still maturing.	Built from day 1 for streaming.
Setup and Management	Simple to medium complexity.	Heavier setup (Zookeeper needed sometimes), tuning needed.

Choose RabbitMQ when

- You need fast, reliable delivery of small messages (e.g., order placed → send email).
- You want built-in routing logic (direct, topic, fanout exchanges).
- You want to acknowledge or retry individual messages easily.
- You are building task queues (e.g., send SMS, generate PDF).
- You don't need to store data long-term — just deliver and forget.

Choose Kafka when

- You need to store and stream huge volumes of data long-term (logs, events, metrics).
- You want to replay events (analytics, rebuilding state, event sourcing).
- You have many consumers reading the same data independently.
- You want to build real-time data pipelines (stream processing, machine learning ingestion).
- You care more about scalability and throughput than super-low latency.

Keycloak Masterclass

Faisal Memon (EmbarkX)

Security, Authentication and Authorization

Faisal Memon (EmbarkX)

Security in applications
means protecting your app,
users, and data from threats.

Authentication means
verifying users identity.

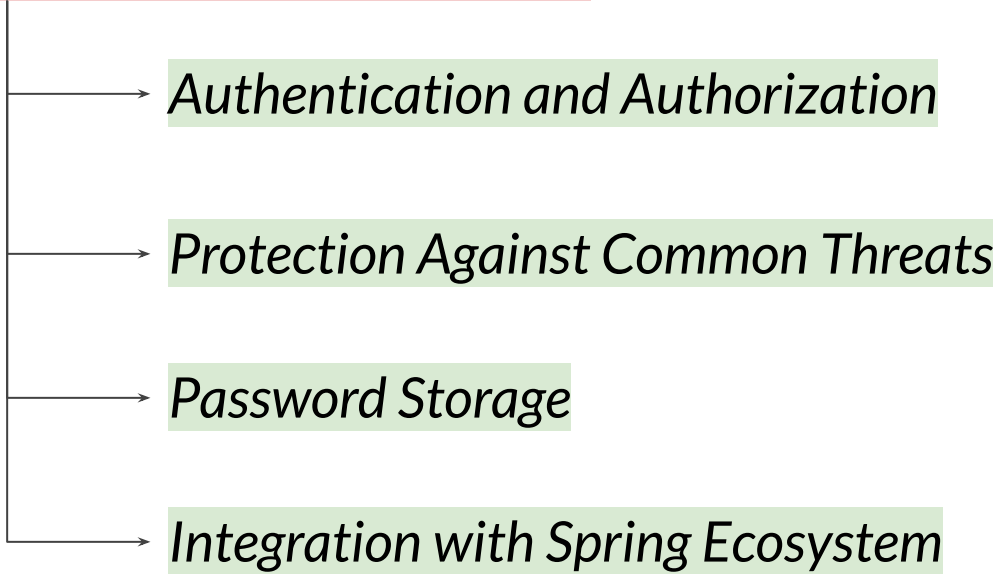
Authorization tells you
what you are allowed to do?

Why These Matter in Software?

Spring Security: Importance & Benefits

Faisal Memon (EmbarkX)

Spring Security



```
graph LR; A[Spring Security] --> B[Authentication and Authorization]; A --> C[Protection Against Common Threats]; A --> D[Password Storage]; A --> E[Integration with Spring Ecosystem];
```

Authentication and Authorization

Protection Against Common Threats

Password Storage

Integration with Spring Ecosystem

Imagine doing all by yourself

Why Use Spring Security

- *Comprehensive and Customizable*
- *Community and Support*
- *Declarative Security*
- *Integration Capabilities*

Why Use Spring Security

→ *Regular Updates*

→ *Ease of Use with Spring Boot*

Thank you

IAM: Introduction and Problem Statement

Faisal Memon (EmbarkX)

Why Do We Need Authentication and Authorization?

→ *Security*

→ *User Management*

→ *Compliance*

Imagine you are building 3
applications - website, admin
dashboard, mobile app

Problems

→ *Managing Logins for Every App*

→ *Security Risks*

→ *Integration Nightmares*

Instead of doing all this by
yourself, **use a tool that is
built for this.**

Identity and Access Management (IAM)

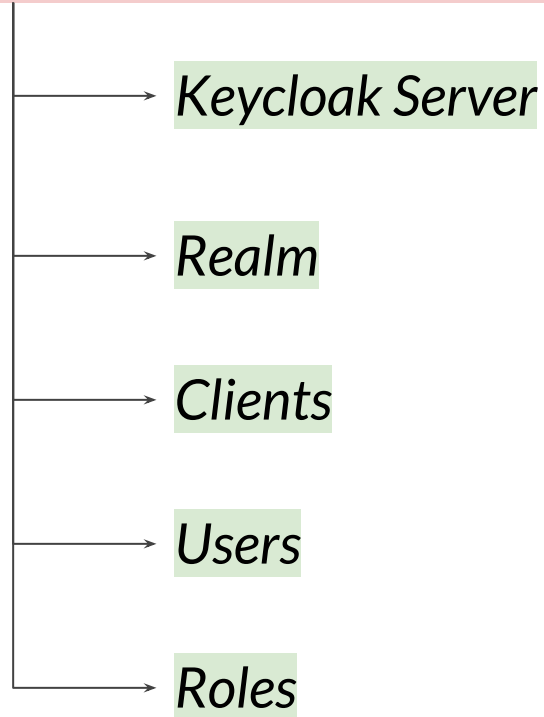
- *Handle login, logout, password reset.*
- *Manage user roles (who is admin, who is viewer).*
- *Support social logins like Google or Facebook.*
- *Give you control over what users can access.*

Keycloak = Centralized,
Production-Ready Identity &
Access Management

Keycloak Architecture and Core Concepts

Faisal Memon (EmbarkX)

Concepts



Concepts

```
graph TD; Concepts[Concepts] --> IP[Identity Providers]; Concepts --> P[Protocols: OIDC and SAML];
```

Identity Providers

Protocols: OIDC and SAML

Concepts

KeyCloak Server

This is the main system that runs and manages everything. It gives you:

- *A login screen*
- *A dashboard to manage users and apps*
- *APIs to connect your app with Keycloak*

Realm

A Realm is like a workspace or a group inside Keycloak.

Concepts

Clients

A **Client** is just an *application* (web, mobile, API) that connects to Keycloak for login.

Users

These are your *actual users* - Employees, Customers, Admins

Roles

Roles define *permissions*.

Concepts

Identity Providers

*Keycloak supports this using **Identity Providers**.*

Protocols

These are the languages Keycloak uses to talk to apps. These are OIDC (OpenID Connect) and SAML.

Summary

Term	Think of It As...	Example
Realm	Group of users + apps	HR realm, Sales realm
Client	An app that uses Keycloak login	React app, mobile app
User	A person who logs in	John, Admin Jane
Role	What a user can do	viewer, editor, admin
Identity Provider	External login option	Google, GitHub, Facebook
OIDC / SAML	Protocols for secure login communication	App ↔ Keycloak login handshake

KeyCloak Server

This is the main system that runs and manages everything.

Realm

Client 1

Different Protocols for Authentication with Keycloak

Faisal Memon (EmbarkX)

An authentication protocol is like a set of rules that helps two systems **talk securely** when logging in a user.

Authentication Protocols

OIDC (OpenID Connect)

Modern and used in most apps

- *Built on top of OAuth2*
- *Uses JWT tokens*
- *Works with modern apps: React, Angular, Spring Boot, Node.js, mobile apps*

SAML (Security Assertion Markup Language)



Used in older enterprise apps

- *Based on XML*
- *Often used in large organizations and legacy systems*
- *More common in Java EE, older ERP systems, or SaaS apps like Salesforce*

What to use?

- *If you're building modern web/mobile apps* → **OIDC**
- *If you're integrating with older enterprise apps* → **SAML**
- *Keycloak supports both, so you can use whichever your app requires*

Authentication Protocols

Feature	OIDC	SAML
Format	JSON (JWT tokens)	XML
Use Case	Modern apps, REST APIs	Legacy apps, enterprise systems
Token Type	Access/ID tokens	SAML Assertions
Easier to Integrate	 Yes	 Not as easy
Popular With	React, Angular, Spring Boot	SAP, Salesforce, Oracle apps

OAuth2

Faisal Memon (EmbarkX)

You had to **share** your
credentials all the time

Problems

→ *Security Risk*

→ *Limited Control*

→ *Inconvenience*

What is OAuth?

OAuth (Open Authorization) is a standard protocol that allows users to grant third-party applications access to their information without sharing their passwords.

Why is OAuth Needed?

OAuth is needed to enable secure and easy access to user information by third-party applications without compromising the user's credentials (like passwords).

OAuth solves the problem of sharing sensitive login credentials directly with third-party applications.

How OAuth 2.0 Works

- *User clicks "Login with Google"*
- *App sends you to Google to log in*
- *You give permission & Google says: "OK!"*
- *App receives a token from Google*
- *App uses that token to talk to Google (not your password)*

Summary

- **What:** OAuth lets apps access your information without needing your password.
- **Why:** It's safer because you don't have to share your password with other apps.
- **Problem Solved:** Before OAuth, apps needed your password to get your info, which was risky.
- **How It Worked Before:** You had to give your password to every app, which was unsafe and inconvenient.
- **How OAuth Works Now:** You log in through a trusted service (like Google), give permission, and the app gets a special token to access your info without needing your password.

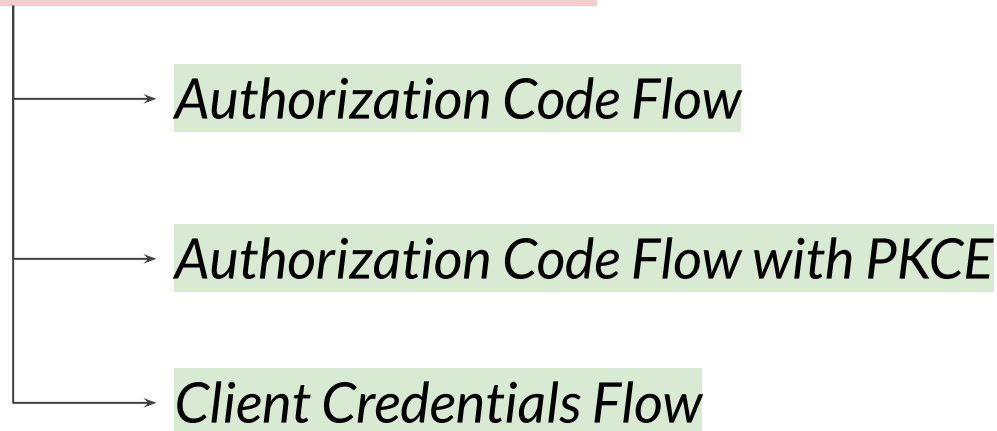
Key Terms

- **Resource Owner (User):** person who owns the account
- **Third-Party Application:** This is the application that wants to access to your account
- **Resource Server:** This is the server that holds data that application wants to access.
- **Authorization Server:** This server handles the authentication (logging in) and authorization (granting permissions)
- **Client:** This is the application that requests access to the resource server on behalf of the user.

Example

- **Resource Owner (User):** You want to give PrintMyPhotos access to your photos without giving them your Google account password
- **Third-Party Application:** This is the application that wants to access your photos to print them
- **Resource Server:** This is the server that holds your photos and has the data that PrintMyPhotos wants to access.
- **Authorization Server:** This server handles the authentication (logging in) and authorization (granting permissions) for Google services.
- **Client:** This is the application that requests access to the resource server (Google Photos) on behalf of the user.

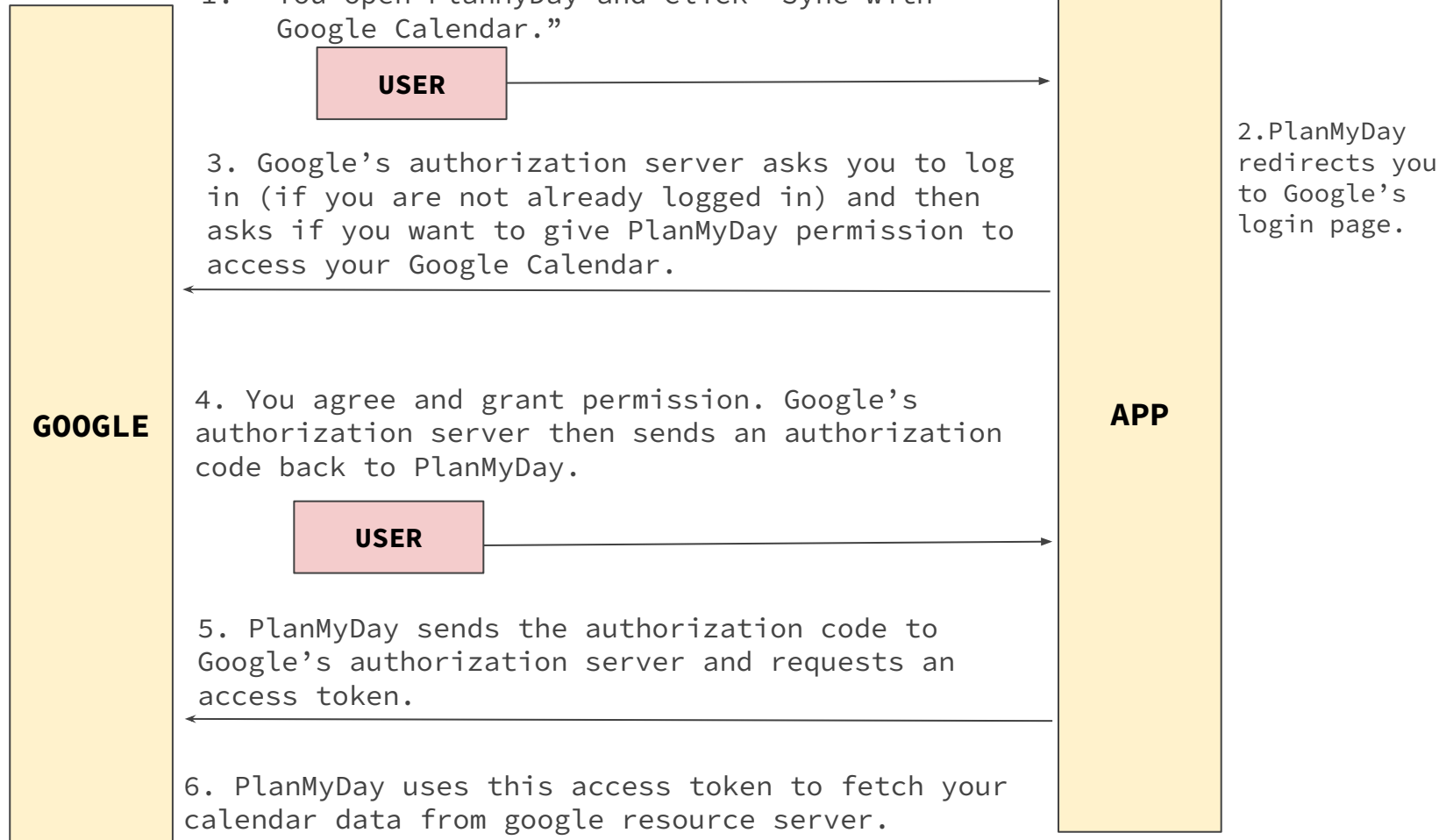
OAuth 2.0 Flows



What is Authorization Code Flow?

Faisal Memon (EmbarkX)

It's the most common and secure way for apps to let users log in using a third-party system (like Google, Facebook, or Keycloak) — without sharing their password with the app.



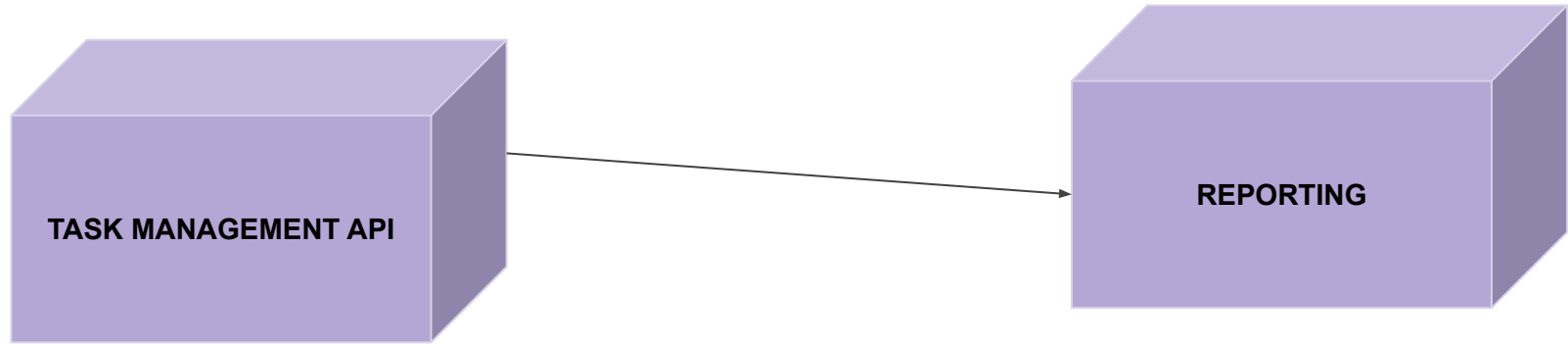
Why Is This Flow Safe?

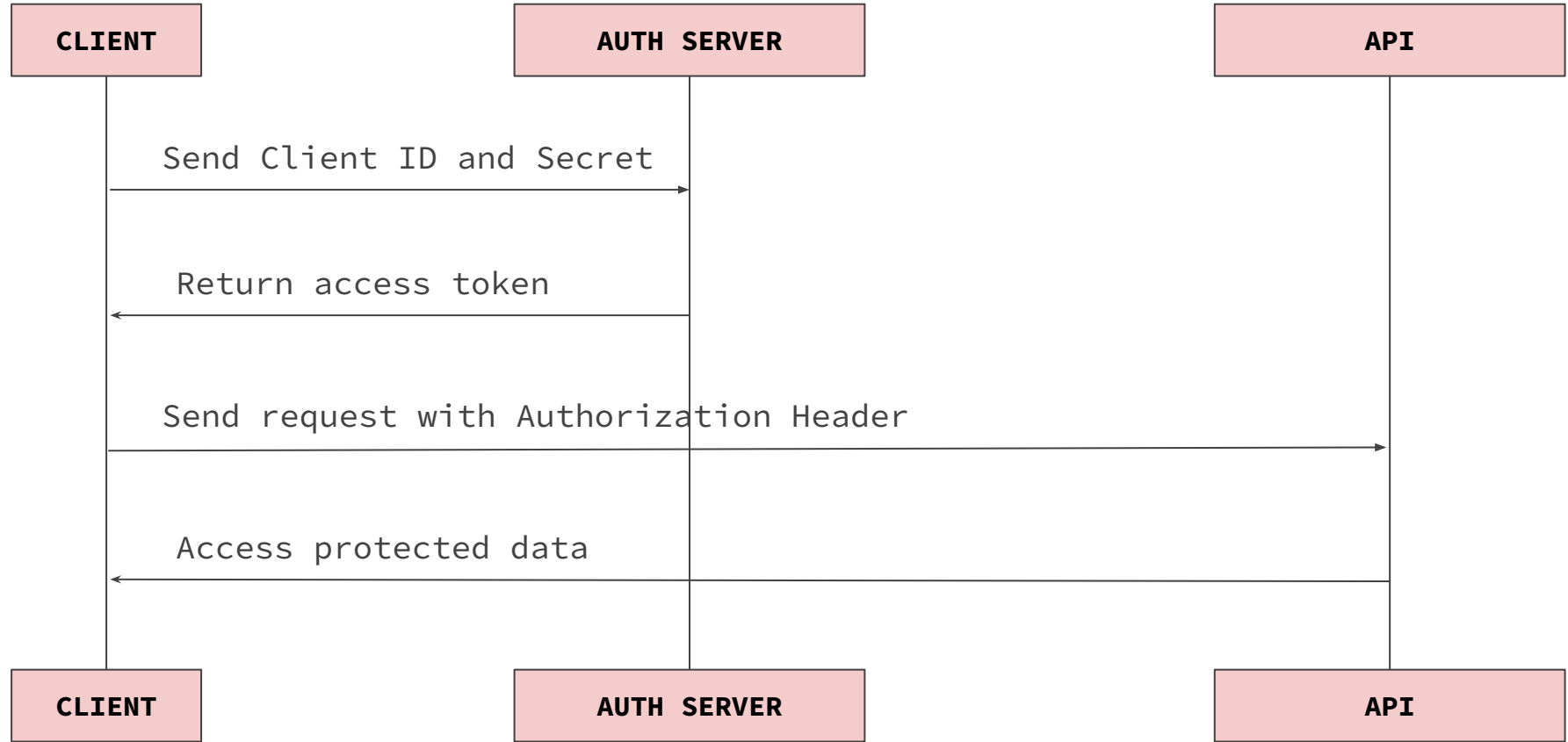
- *The authorization code is temporary and short-lived*
- *Password is entered on Google's secure login page, not on App*
- *The token is exchanged server-to-server, not visible in the browser*
- *It supports extra security layers like PKCE and HTTPS*

Client Credentials Flow

Faisal Memon (EmbarkX)

Client Credentials Flow is an OAuth 2.0 flow used when machines (not users) talk to each other.





When to use?

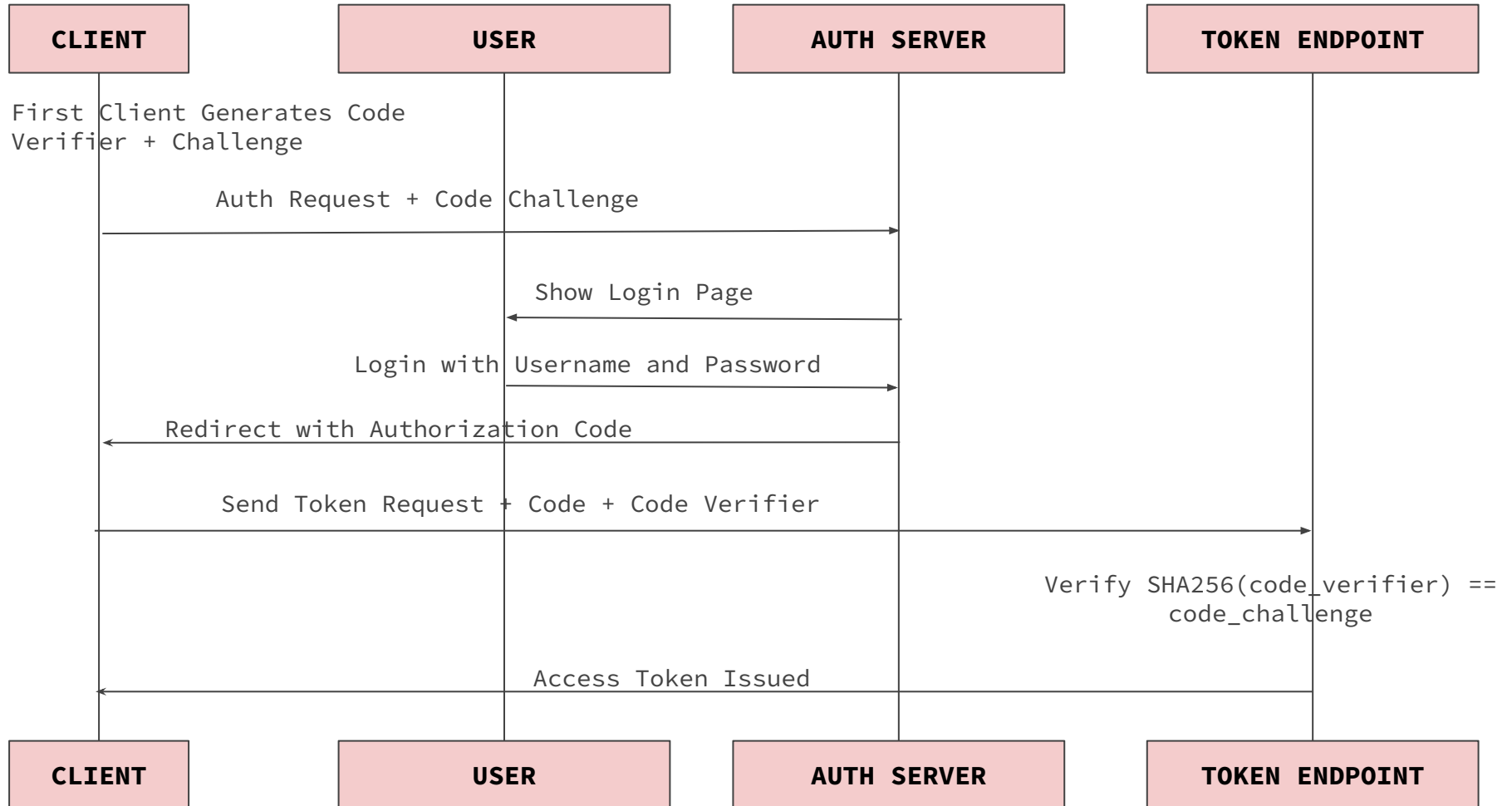
- *System-to-system (machine-to-machine) calls*
- *Backend jobs, cron tasks, microservices communication*
- *No end-user interaction required*

PKCE Flow

Faisal Memon (EmbarkX)

PKCE = Proof Key for Code Exchange

Extension to OAuth 2.0 used to make
authorization code flow more secure



Securing Microservices

Faisal Memon (EmbarkX)

Single Entry Point

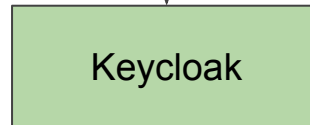
Desktop



Mobile



Gateway



Keycloak

Microservice 2

Microservice 1

Microservice 3

Behind internal Firewall / VPC

What to expect next?

Faisal Memon (EmbarkX)

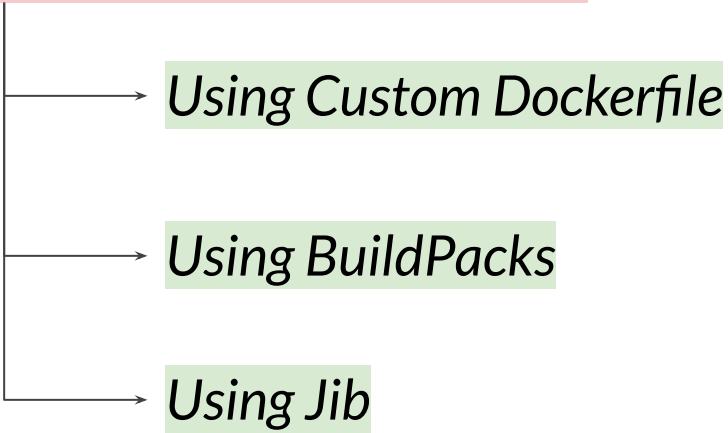
What to expect?

- *Docker, Containerization and different ways of containerizing your services*
- *Comprehensive and very practical*
- *Be production ready*

Different Ways to Containerize Spring Boot Applications

Faisal Memon (EmbarkX)

Different Ways



```
graph LR; A[Different Ways] --> B[Using Custom Dockerfile]; A --> C[Using BuildPacks]; A --> D[Using Jib];
```

Using Custom Dockerfile

Using BuildPacks

Using Jib

Using Dockerfile

```
FROM openjdk:11
VOLUME /tmp
ADD target/my-app.jar my-app.jar
EXPOSE 8080
ENTRYPOINT [ "java", "-jar", "/my-app.jar" ]
```

Pros

- *Full control over layers and image structure*
- *Works everywhere (Kubernetes, CI/CD)*

Cons

- *You manage everything (base image, JDK, layers, etc.)*

Using BuildPacks

- *Since Spring Boot 2.3+, you can use the Cloud Native Buildpacks*
- *Help you generate a container image without writing a Dockerfile*

Pros

- *No need to write a Dockerfile*
- *Automatically includes correct JDK version*
- *Follows best practices for Spring apps*
- *Supports layers and image caching*

Cons

- *Limited customization*
- *Slower initial builds (but faster rebuilds)*

Using Jib

- *Jib is a Maven/Gradle plugin that builds Docker images without Docker installed, directly from the build tool.*

Pros

- *No Dockerfile needed*
- *Builds are reproducible and fast*
- *Works well in CI/CD*

Cons

- *More Maven/Gradle setup*
- *Not ideal if you want custom Docker layers*

Which to use?

Approach	Control	Simplicity	Best for
Dockerfile	High	Medium	Advanced devs, Kubernetes users
Buildpacks	Medium	Easy	Most Spring Boot apps (from 2.3+)
Jib	Medium	Easy	CI/CD pipelines, cloud projects

Important Docker Commands

Faisal Memon (EmbarkX)

docker system prune -a --volumes --force

Explanation:

This command removes all stopped containers, all networks not used by at least one container, all dangling images, all build cache, and all volumes not used by at least one container, without confirmation.

- **docker system prune** – removes unused containers, networks, images (not in use), and cache
- **-a** – removes *all* images, not just dangling ones
- **--volumes** – includes named volumes (careful with persistent data)
- **--force** – skips confirmation prompts

docker network prune --force

Explanation:

Removes all Docker networks that are not currently used by any running or stopped containers, and it does so without prompting for confirmation. This helps free up disk space by cleaning up obsolete network configurations.


- `docker system prune` – removes unused networks
- `--force` – skips confirmation prompts

`docker rm -f $(docker ps -q)`

Explanation:

This command forcibly removes all currently running Docker containers.

- `docker ps -q`: Lists the IDs of all running Docker containers.
- `docker rm -f`: Forcibly removes the specified containers.
- `$(...)`: This is command substitution, meaning the output of `docker ps -q` (the IDs of running containers) is passed as arguments to the `docker rm -f` command.

 *This will immediately stop and remove all running containers, so use it carefully in dev or staging – not on prod machines unless you're absolutely sure.*

Using Dockerfile

Faisal Memon (EmbarkX)

Dockerfile

1 → Build Your Spring Boot App to get .jar file

```
./mvnw clean package -DskipTests
```

OR

```
mvnw clean package -DskipTests
```

2 → Create Dockerfile

```
FROM eclipse-temurin:23-jdk-ubi9-minimal
```

```
WORKDIR /app
```

```
COPY target/*.jar app.jar
```

```
EXPOSE 8082
```

```
ENTRYPOINT ["java", "-jar", "app.jar"]
```


Dockerfile

3 → Navigate to directory where your project resides via your terminal

```
cd <path>
```

4 → Build docker image

```
docker build -t config-server ./config-server
```

5 → Run docker image

```
docker run --env-file configserver/.env -d --name config-server  
-p 8888:8888 config-server
```

Each time you change code and rebuild .jar

- `./mvnw clean package`
- `docker build -t config-server ./config-server`
- `docker run --env-file configserver/.env -d --name config-server
-p 8888:8888 config-server`

Resolving Errors

Faisal Memon (EmbarkX)

ERROR 1

```
→ order git:(main) x ./mvnw clean package -DskipTests
zsh: permission denied: ./mvnw
→ order git:(main) x
```

SOLUTION

Run this once to give it execute permission:

```
chmod +x ./mvnw
```

Then run:

```
./mvnw clean package -DskipTests
```

ERROR 2

```
→ ecom-microservices git:(main) ✗ docker run -d --name eureka -p 8761:8761 eureka
docker: Error response from daemon: Conflict. The container name "/eureka" is already in use b
y container "dc11c219734545cca7fa163c17b4e72fbc5fbd34fda987c8b901226f15ba2f3d". You have to re
move (or rename) that container to be able to reuse that name.
See 'docker run --help'.

→ ecom-microservices git:(main) ✗ docker run -d --name eureka -p 8761:8761 eureka
docker: Error response from daemon: Conflict. The container name "/eureka" is already in use b
y container "dc11c219734545cca7fa163c17b4e72fbc5fbd34fda987c8b901226f15ba2f3d". You have to re
move (or rename) that container to be able to reuse that name.
See 'docker run --help'.

→ ecom-microservices git:(main) ✗ docker rm -f eureka

eureka

→ ecom-microservices git:(main) ✗ docker run -d --name eureka -p 8761:8761 eureka
e587c382612096ef47f0a2941bc613fa1d66bbc925f8c7a39fa9547098c350e1
```

docker rm -f eureka

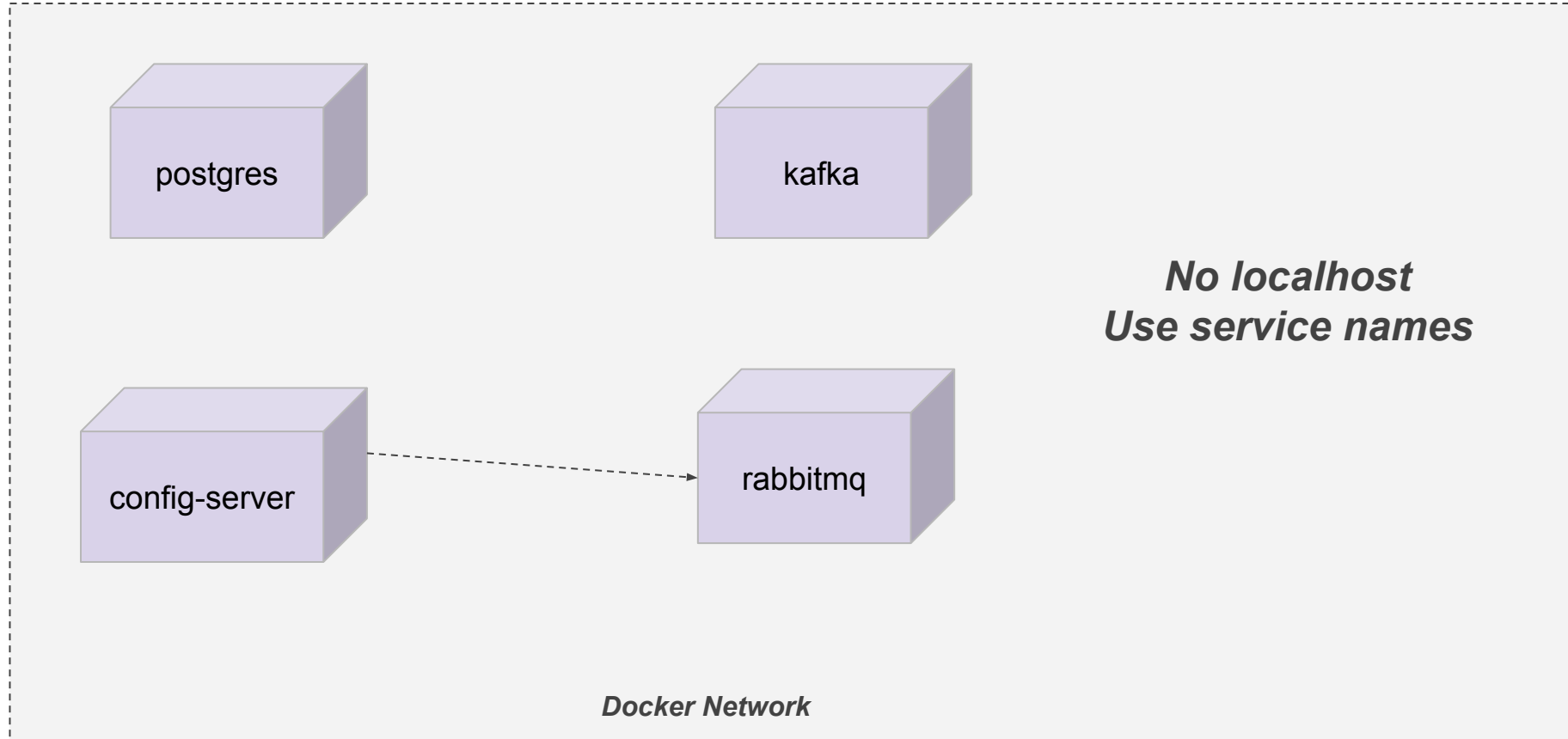
ERROR 3

```
[INFO] Compiling 1 source file with javac [debug parameters release 24] to target/classes
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
[INFO] Total time: 2.275 s
[INFO] Finished at: 2025-06-13T19:14:42+05:30
[INFO] -----
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.13.0:compile (
default-compile) on project eureka: Fatal error compiling: error: release version 24 not suppo
rted -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the followin
g articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoExecutionException
→ eureka git:(main) ×
```




Use right java version in pom.xml `<java.version>23</java.version>`

Configuration Management in Docker

Faisal Memon (EmbarkX)



Summary

Reason	Explanation
 Separate config	Because Docker environment is different from your local machine
 Use service names	Because containers need a way to identify each other on the Docker network
 Don't use <code>localhost</code>	It will only refer to itself (the current container), not others

Buildpacks

Faisal Memon (EmbarkX)

Buildpacks are tools
that **automate**
container image
creation.

You don't write instructions —
it **detects, builds,** and
packages your app
automatically.

```
mvn spring-boot:build-image
```

Buildpacks were originally
created by **Heroku**

Your Image








Start Command & Metadata

Your Spring Boot .jar

App Dependencies

Java Runtime

Why Use Buildpacks for Spring Boot?

Advantage	Explanation
 Zero Docker knowledge	No need to write <code>Dockerfile</code> .
 Faster rebuilds	Layer caching improves speed.
 Secure defaults	Uses best practices for image building.
 Smart detection	Automatically figures out what's needed.
 Cloud ready	Built for Kubernetes, Heroku, GCP, etc.

When Might You NOT Use Buildpacks?

- *When you need custom image behavior*
- *When your app has non-standard packaging.*
- *When you're already comfortable with Dockerfile-based builds.*

Build image with mention of image name

```
mvnw spring-boot:build-image  
-Dspring-boot.build-image.imageName=<image-name>
```

Build image without mention of image name

```
mvnw spring-boot:build-image -DskipTests
```

Run image

```
docker run -d -p 8080:8080 <image-name>
```

Additional Configurations

Faisal Memon (EmbarkX)

```

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <image>
          <!-- ✅ 1. Image name -->
          <name>demo/my-app</name>

          <!-- ✅ 2. Builder image (default is Paketo) -->
          <builder>paketobuildpacks/builder:base</builder>

          <!-- ✅ 3. Run image (used as base during runtime) -->
          <runImage>paketobuildpacks/run:base-cnb</runImage>

          <!-- ✅ 4. Environment variables passed to buildpack -->
          <env>
            <BP_JVM_VERSION>21</BP_JVM_VERSION>
            <BP_JAVA_OPTS>-Xmx512m</BP_JAVA_OPTS>
          </env>

          <!-- ✅ 5. Docker publish settings (if pushing to registry) -->
          <publish>true</publish> <!-- optional -->
          <docker>
            <username>your-docker-username</username>
            <password>your-docker-password</password>
          </docker>
        </image>
      </configuration>
    </plugin>
  </plugins>
</build>

```

Option	Description
<code><name></code>	The image name and optionally tag (e.g., <code>my-app:1.0</code>).
<code><builder></code>	The builder image used to build the app (default is Paketo's <code>base</code>).
<code><runImage></code>	Optional. Base image used during runtime.
<code><env></code>	Set environment variables used during build (e.g., JVM version, memory limits).
<code><publish></code>	If <code>true</code> , pushes the built image to a remote Docker registry.
<code><docker></code>	Credentials used if you're publishing the image to DockerHub or another registry.

Observability in Docker

Faisal Memon (EmbarkX)

Networks & Services

These services are connected to the main backend network used for application services and system components

backend

These services are strictly part of your observability/logging stack






loki




















Introduction to Jib

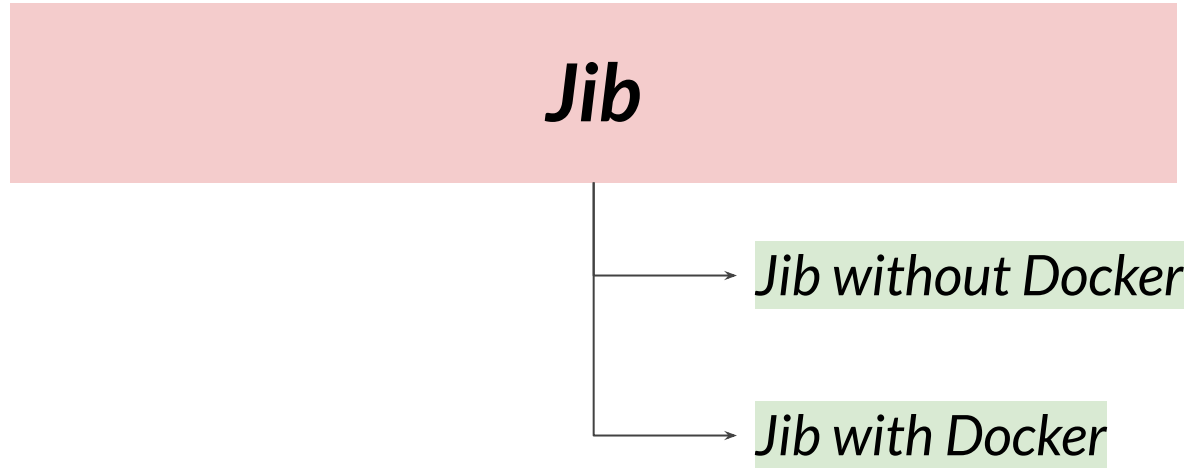
Faisal Memon (EmbarkX)

Jib is a powerful tool from Google that lets you containerize Java applications without writing a Dockerfile.

Why Jib?

-  *No Dockerfile needed*
-  *Fast image builds (layered caching)*
-  *Secure: No docker daemon required*
-  *Push directly to DockerHub, GCR, ECR, etc.*
-  *Works inside CI/CD systems like GitHub Actions, GitLab, Jenkins*

Feature	Jib	Buildpacks (Pack CLI / Spring Boot Buildpacks)
 Tool Type	Maven/Gradle plugin	CLI tool (pack) or integrated in Spring Boot 2.3+ (spring-boot:build-image)
 Docker Daemon Needed	 No (can build and push without Docker)	 Optional (uses Docker by default, but can use remote builders)
 Build Tool Integration	 Tightly integrated with Maven/Gradle	 Spring Boot integration or external CLI
 Customization	 Fine-grained (layers, JVM flags, ports, environment)	 Less control unless you customize the buildpack stack
 Speed	 Very fast , uses caching well	 Fast , but slower than Jib for pure Java builds
 Layers Optimized	 Yes , Java-specific layering	 Yes , generalized layering
 Language Support	 Only for JVM (Java, Kotlin)	 Polyglot (Java, Node.js, Go, Python, etc.)



Jib without Docker

mvn jib:build

- Builds the image using Java, not Docker
- Pushes the image directly to a remote container registry (like Docker Hub, GCR, ECR)
- Doesn't need Docker installed or running locally

When to use?

- On CI servers (like GitHub Actions, GitLab CI, etc.)
- You want faster, repeatable builds without Docker daemon
- You're pushing images directly to a registry (like Docker Hub)

Jib with Docker

mvn jib:dockerBuild

- Builds the image using Jib
- Loads it directly into your local Docker daemon
- Doesn't push to Docker Hub

When to use?

- You want to run the image locally for testing
- You plan to use docker-compose or docker run right away

```
<plugin>
  <groupId>com.google.cloud.tools</groupId>
  <artifactId>jib-maven-plugin</artifactId>
  <version>3.4.0</version>
  <configuration>
    <from>
      <image>eclipse-temurin:21-jdk</image>
    </from>
    <to>
      <image>yourdockerhubusername/yourimage</image>
      <auth>
        <username>${env.DOCKER_USERNAME}</username>
        <password>${env.DOCKER_PASSWORD}</password>
      </auth>
    </to>
    <container>
      <mainClass>your.main.Class</mainClass>
    </container>
  </configuration>
</plugin>
```

Feature	jib:build (No Docker)	jib:dockerBuild (With Docker)
Docker installed	✗ Not needed	✓ Required
Pushes to Docker Hub	✓ Yes	✗ No
Loads image into local Docker	✗ No	✓ Yes
CI/CD friendly	✓ Yes	✗ No (unless Docker is installed)
Use with docker-compose	✗ Pull from registry first	✓ Immediately usable

Common Jib Commands and Configurations to Know

Faisal Memon (EmbarkX)

`mvn jib:dockerBuild`

What it does:

- Builds your app and creates a Docker image locally (in your Docker daemon).
- It does not push the image anywhere.

When to use:

- You're testing locally.
- You want to run: *docker run your-image-name*

mvn compile jib:build

Builds your app and pushes the image directly to a remote registry (like Docker Hub or GCR). Doesn't require Docker installed

When to use:

- You're in CI/CD.
- You want to deploy to Kubernetes or cloud directly.
- You want to skip local Docker setup.

mvn compile jib:buildTar

What it does:

- Creates a .tar file of your image that you can load into Docker manually.

When to use:

- You want to transfer or store the image offline.
- You want to run: *docker load < image.tar*.








mvn clean compile jib:dockerBuild

What it does:

- Cleans the previous build output (target/ folder).
- Compiles your Java code from scratch.
- Builds the Docker image locally with Jib.

When to use:

- You want a fresh build (to avoid any leftover files).
- You've made structural changes (e.g., dependency changes, added classes).
- You're preparing for a clean deploy or test.

Command	What It Does	When to Use
<code>mvn compile jib:dockerBuild</code>	 Builds the Docker image locally (no push), without cleaning old files	Fast dev builds
<code>mvn clean compile jib:dockerBuild</code>	 Clean build + Docker image locally	Reliable fresh build
<code>mvn compile jib:build</code>	 Builds image and pushes to Docker Hub or any registry	When you want to push to remote
<code>mvn clean compile jib:build</code>	 Clean + build + push to Docker Hub	Clean push for CI/CD
<code>mvn jib:buildTar</code>	 Exports the image as a <code>.tar</code> file	Use with <code>docker load</code> , offline work
<code>mvn jib:dockerBuild -Dimage=your-image-name</code>	 Custom image name built locally	Local custom naming
<code>mvn jib:build -Dimage=registry-name/image:tag</code>	 Push image to a custom registry	Production/staging deployment

Important Configurations

Base Image (important!)

```
<from>  
  <image>gcr.io/distroless/java21</image>  
</from>
```

Target Image (where you push it)

```
<to>  
  <image>your-dockerhub-username/your-app-name</image>  
</to>
```

Add Ports

```
<container>  
  <ports>  
    <port>8080</port>  
  </ports>  
</container>
```

Important Configurations

Set Environment Variables

```
<container>  
  <environment>  
    <SPRING_PROFILES_ACTIVE>docker</SPRING_PROFILES_ACTIVE>  
  </environment>  
</container>
```

Use Local Docker Image as Base (optional)

```
<from>  
  <image>eclipse-temurin:21-jre</image>  
</from>
```


When to use which method of containerizing

Faisal Memon (EmbarkX)

Jib (No Docker Needed to Build)

Use when:

- You're new to Docker.
- You don't want to install Docker just to build images.
- You're using Maven or Gradle already.
- You want simple, fast container builds that push to Docker Hub or GCR.

Pros:

- No Docker needed to build.
- Fast incremental builds.
- Integrated into your Maven/Gradle workflow.

Cons:

- Limited control over the base image.
- Not great for multi-container setups (without Compose or k8s).

Dockerfile + Docker CLI

Use when:

- You need full control over how your image is built.
- You want to customize the container (e.g., install packages).
- You're comfortable writing a Dockerfile.

Pros:

- Full flexibility.
- Industry standard.
- Easy to share.

Cons:

- Requires Docker installed locally.
- Manual steps if not scripted.

Spring Boot's Built-in spring-boot-maven-plugin with build-image

Use when:

- You want a quick default image with zero setup.
- You're fine with default behavior (no Dockerfile).
- You're using Spring Boot 2.3+

Pros:

- Very easy to use.
- Uses Paketo buildpacks under the hood.
- Great for simple apps.

Cons:

- Less control.
- Slightly larger image size.

Goal or Context	Best Method
You're just starting out	Jib or Spring Boot plugin
You need custom layers, scripts	Dockerfile
No Docker on your machine	Jib (jib:build)
CI/CD pipeline needs optimized builds	Jib
You want full Docker control	Dockerfile