

# Spring Boot Microservices with Spring Cloud, k8s & Docker

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](mailto:embarkxofficial@gmail.com)

[www.embarkx.com](http://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*

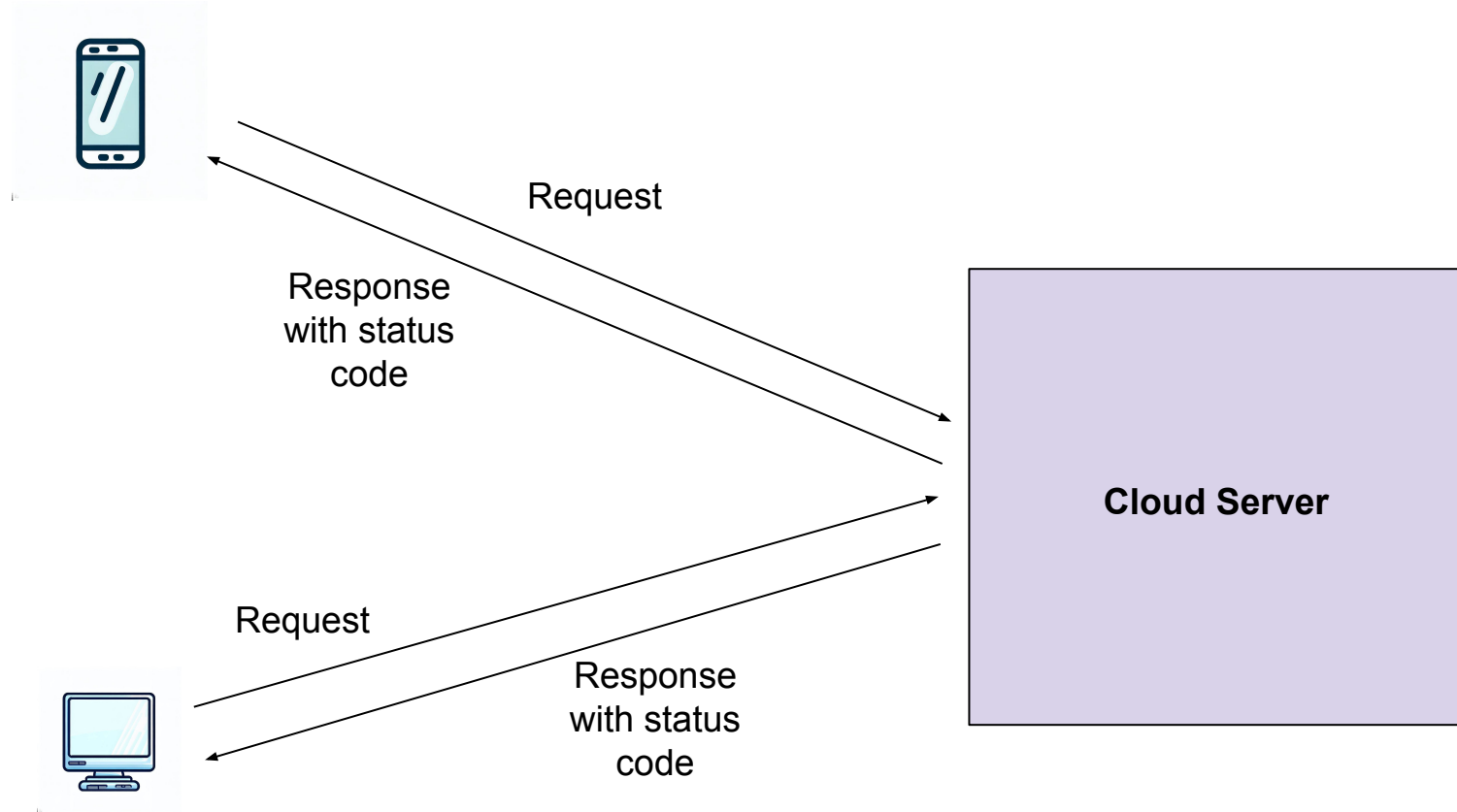
→ *External API's*

→ *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*

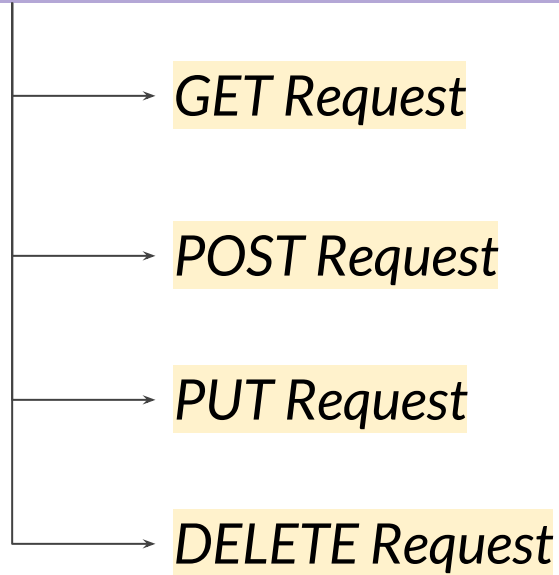
- *401 Unauthorized*
- *403 Forbidden*
- *404 Not Found*
- *500 Internal Server Error*

# 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...**

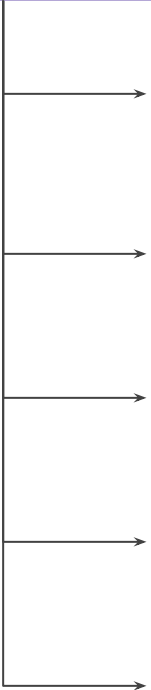
- *You could have prefabricated components?*
- *Could you assemble faster?*
- *Could you reduce errors?*
- *Would that make you fast?*

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 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)'.

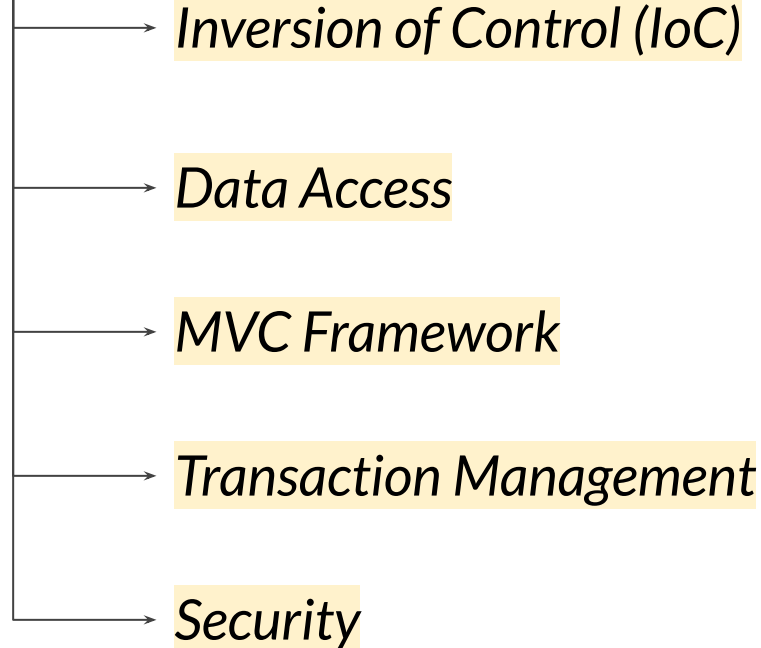
```
public interface MessageService {  
    String getMessage();  
}
```

```
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

- 
- Inversion of Control (IoC)
  - Data Access
  - MVC Framework
  - Transaction Management
  - Security

# *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*

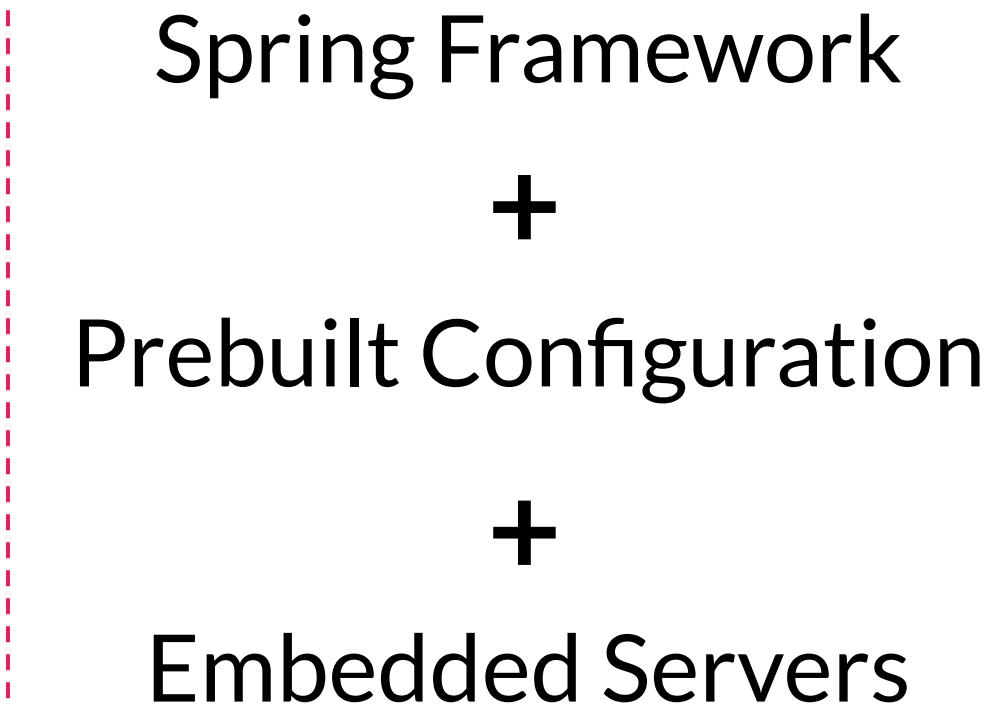
## Spring

## 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

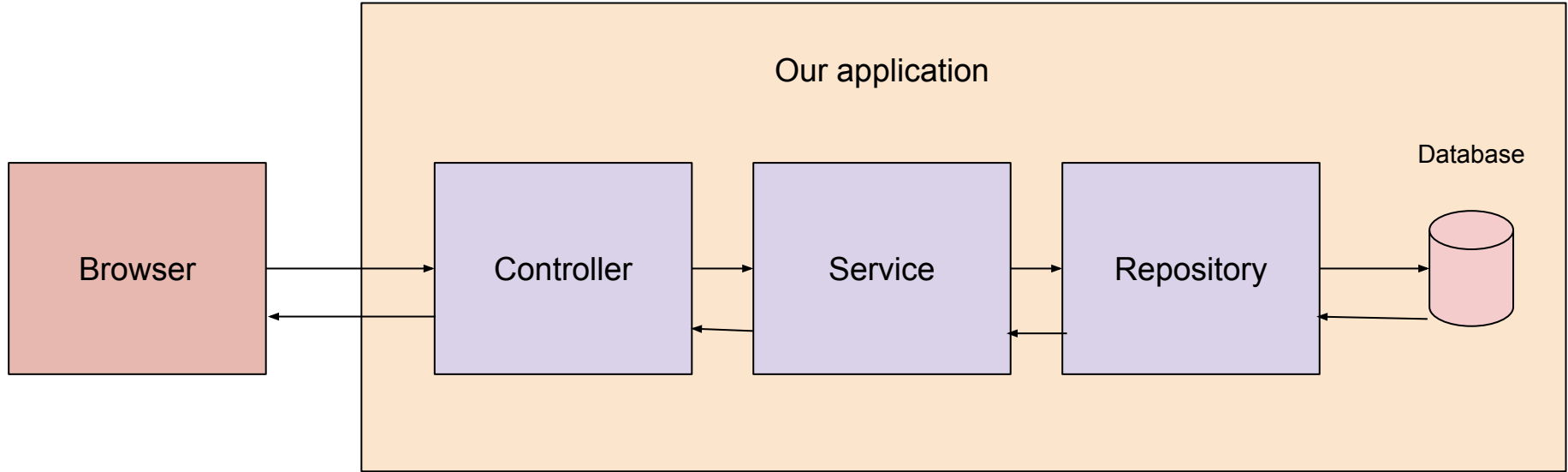
*Presentation layer presents the data and the application features to the user. This is the layer where in all the controller classes exist.*

## 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.*

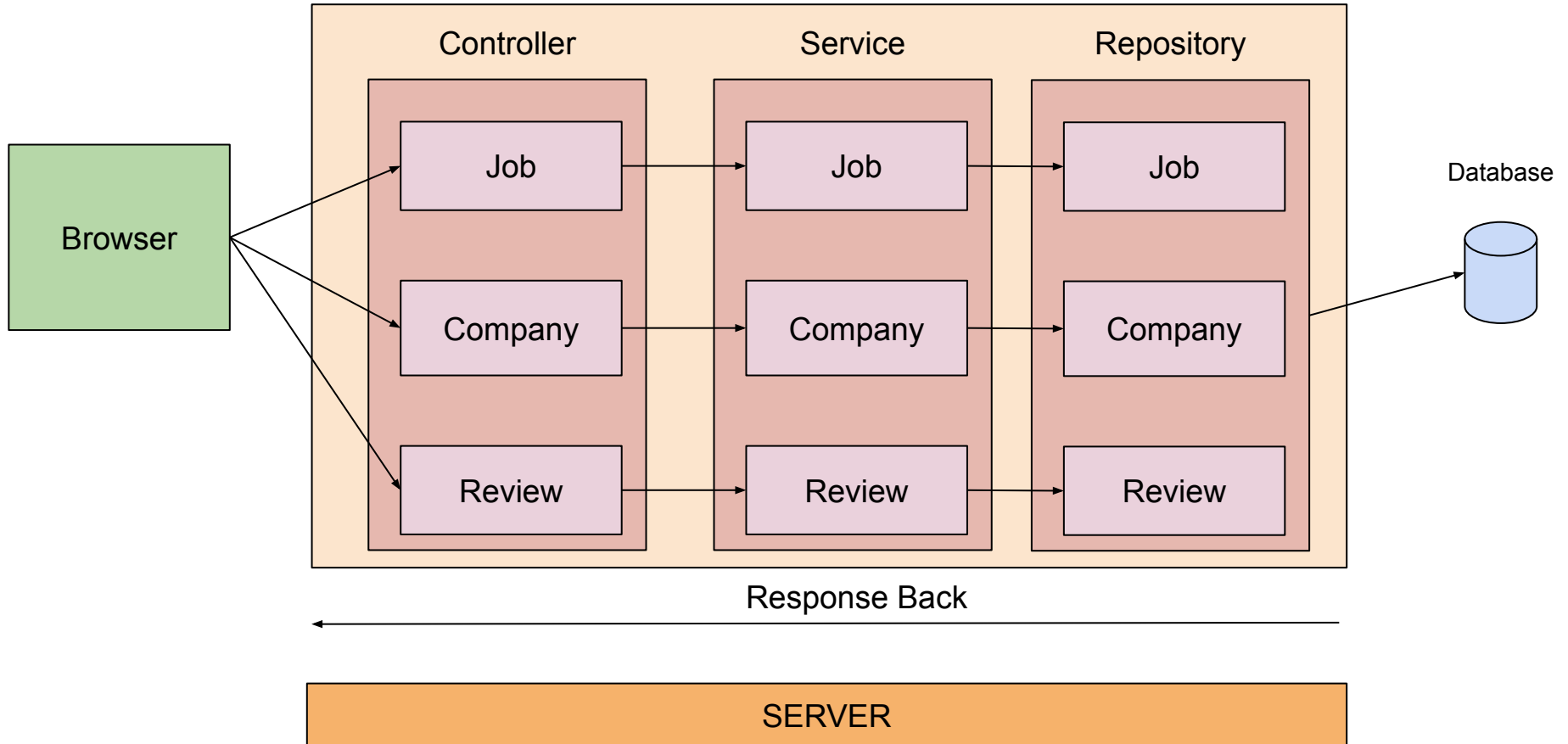


# Project Overview

Faisal Memon (EmbarkX)



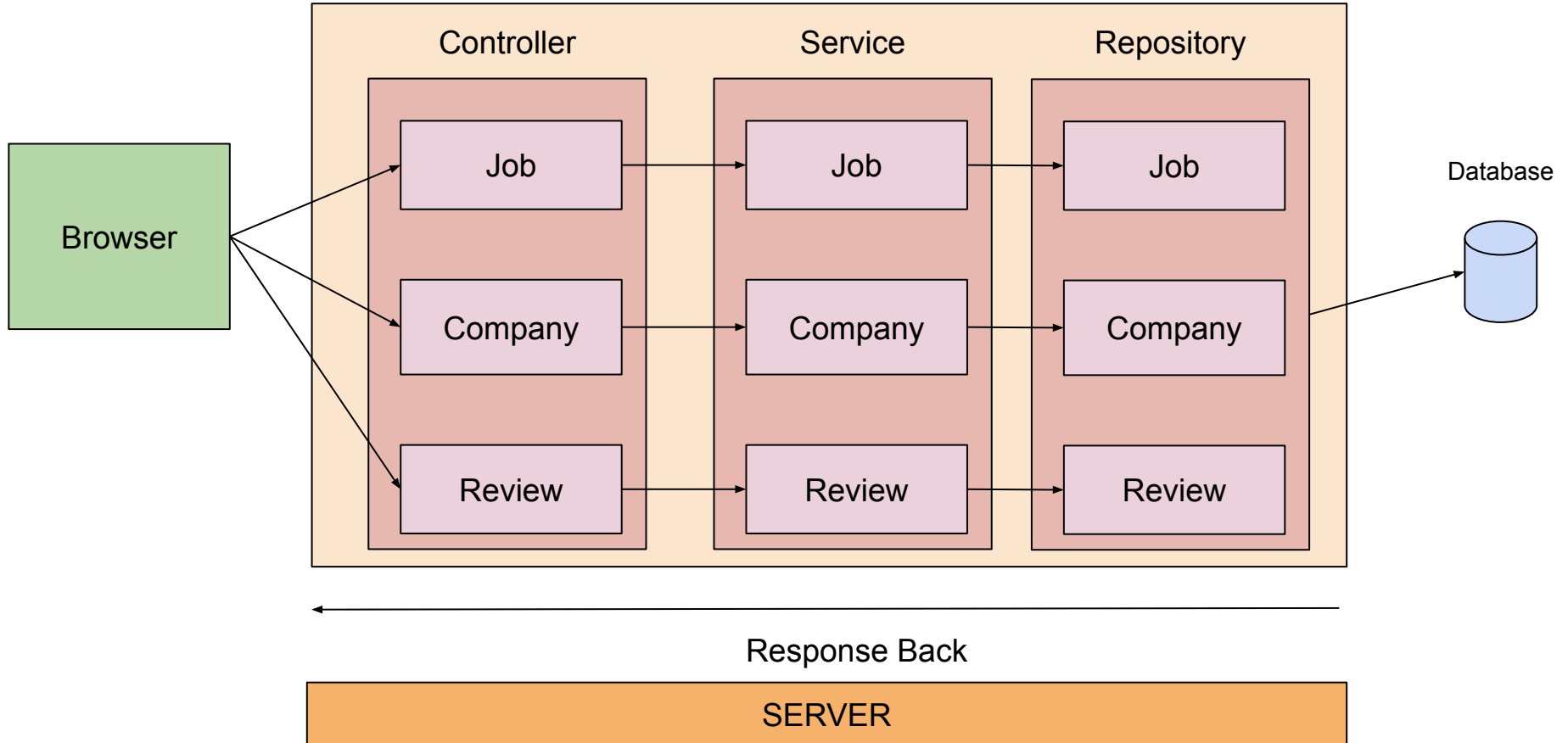
## OUR APPLICATION

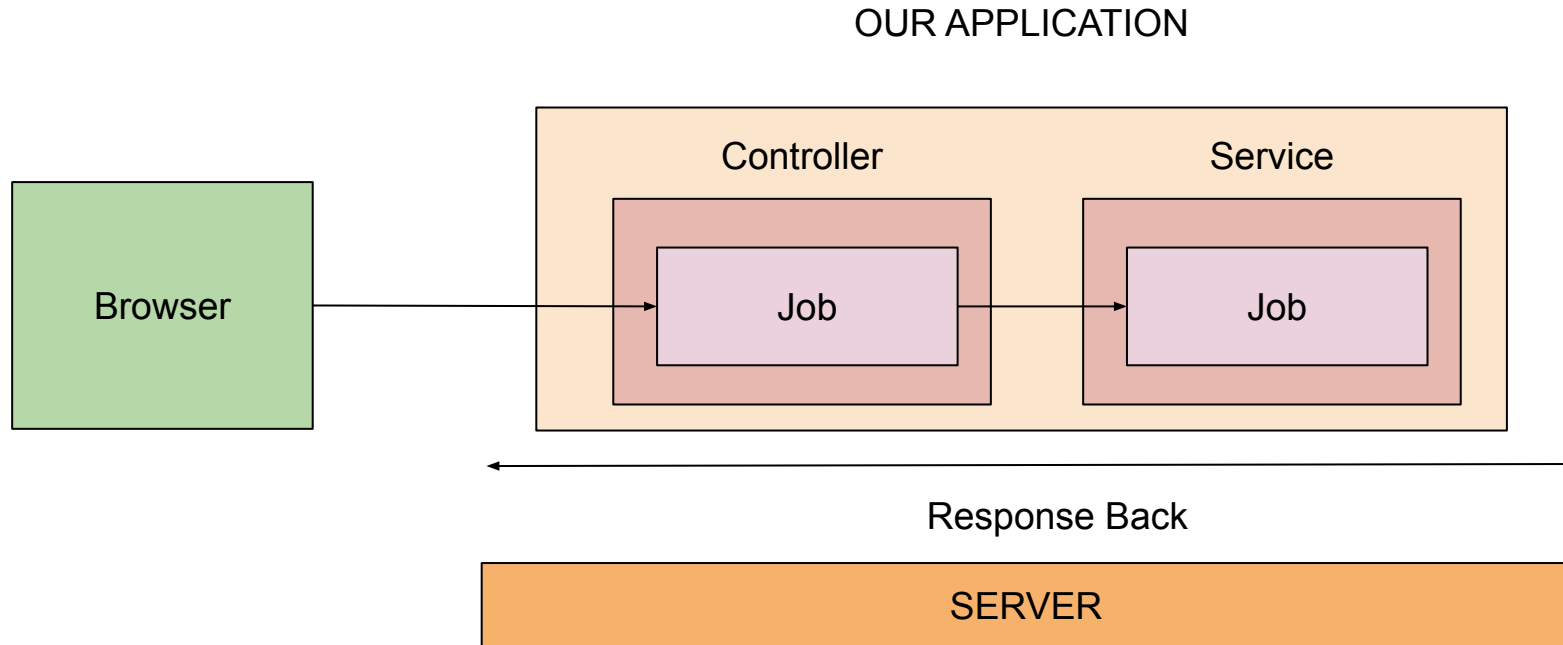


# Structuring Thoughts

Faisal Memon (EmbarkX)

## OUR APPLICATION





# ResponseEntity Class

Faisal Memon (EmbarkX)

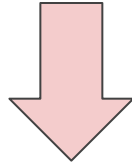
# **Need and Problem it solves**

- *Customization of HTTP Response*
- *Flexible Response Handling*
- *Consistent API Design*

# What is JPA?

Faisal Memon (EmbarkX)

```
public class Job {  
    private Long id;  
    private String title;  
    private String description;  
    private String minSalary;  
    private String maxSalary;  
    private String location;  
}
```



id	title	description	minSalary	maxSalary	location
1	Senior Software Engineer	Senior Software Engineer	30000	40000	40000



# *Advantages of using JPA*

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

# Let's Understand Data Layer

**Faisal Memon (EmbarkX)**

## 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();  
getJob(int id);  
updateJob(Job employee);  
deleteJob(Job employee);
```

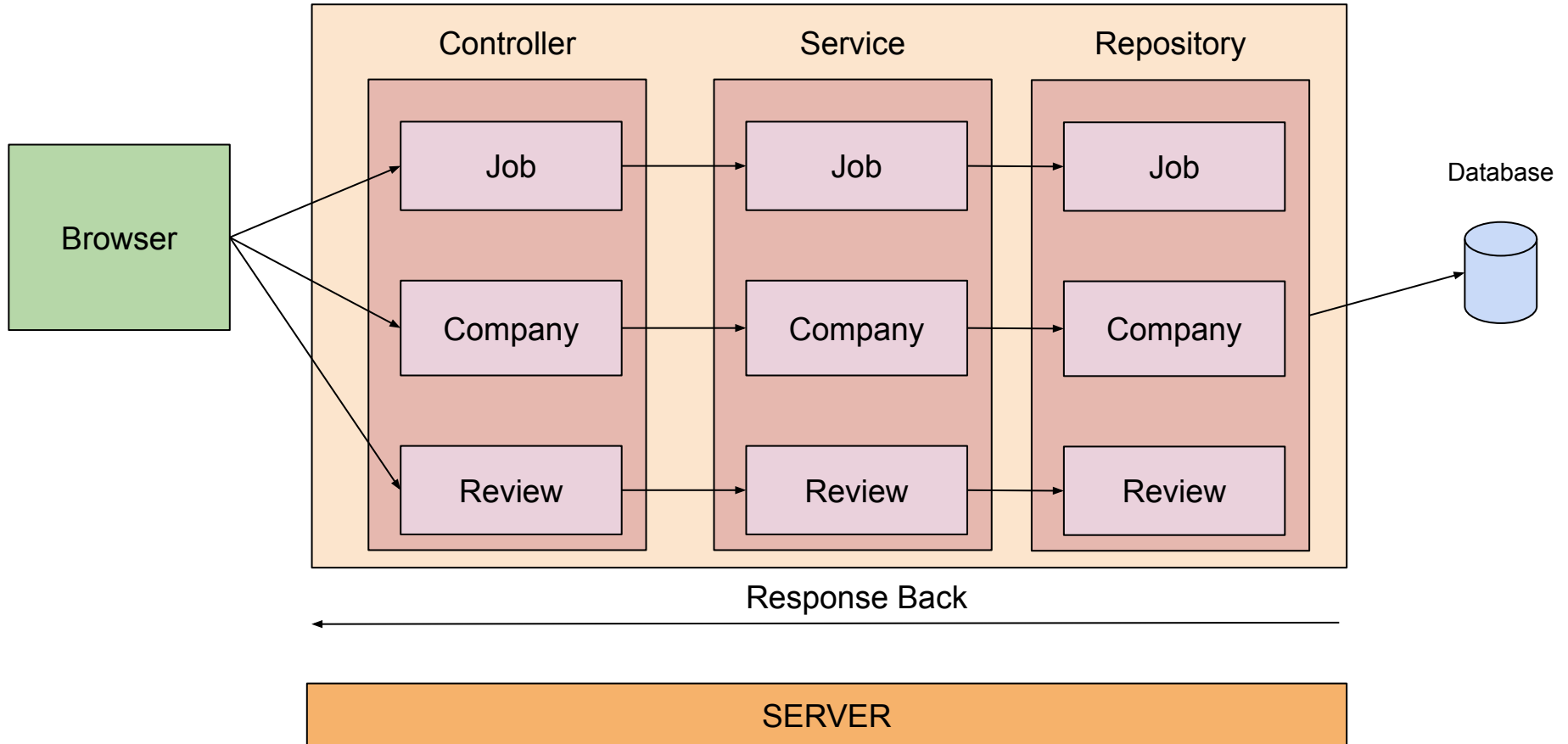
## Syntax

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

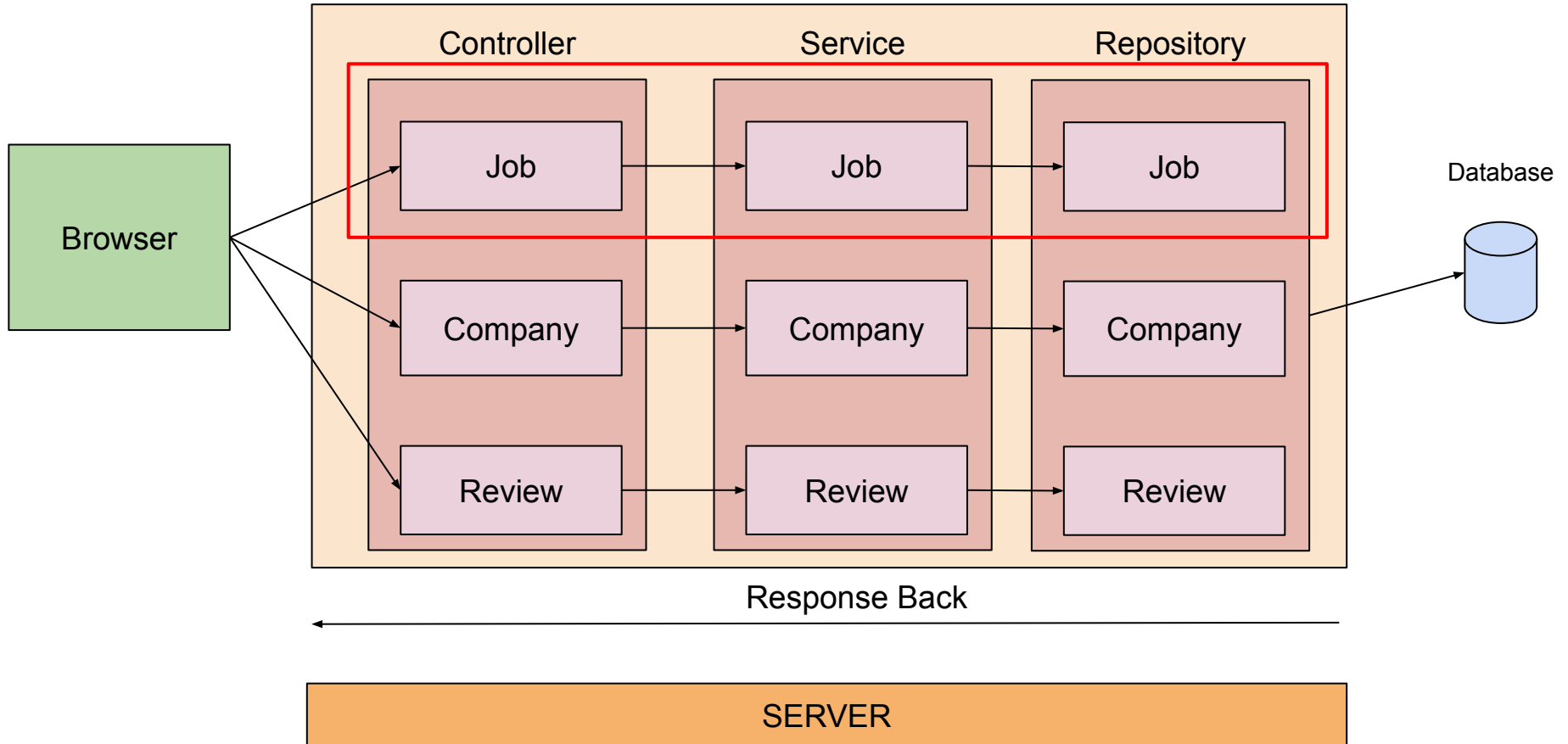
# What's Next?

Faisal Memon (EmbarkX)

## OUR APPLICATION



## OUR APPLICATION



# *Companies*

GET /companies

PUT /companies/{id}

POST /companies

DELETE /companies/{id}

GET /companies/{id}



# Reviews

GET /companies/{companyId}/reviews

POST /companies/{companyId}/reviews

GET /companies/{companyId}/reviews/{reviewId}

PUT /companies/{companyId}/reviews/{reviewId}

DELETE /companies/{companyId}/reviews/{reviewId}

# Thinking about defining Reviews API

Faisal Memon (EmbarkX)

# *Reviews*

GET /companies/{companyId}/reviews

POST /companies/{companyId}/reviews

GET /companies/{companyId}/reviews/{reviewId}

PUT /companies/{companyId}/reviews/{reviewId}

DELETE /companies/{companyId}/reviews/{reviewId}

# 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

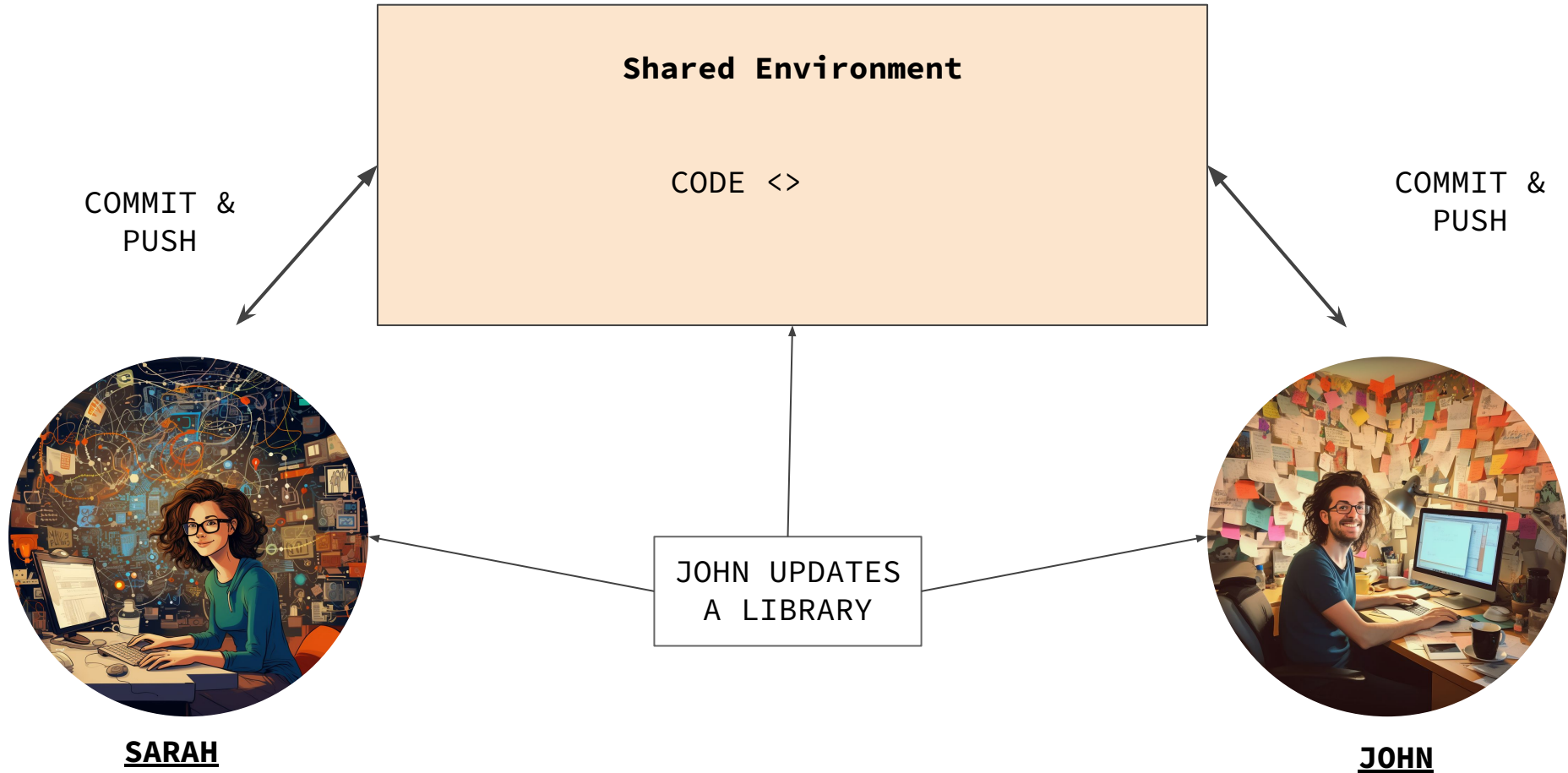
Faisal Memon (EmbarkX)



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

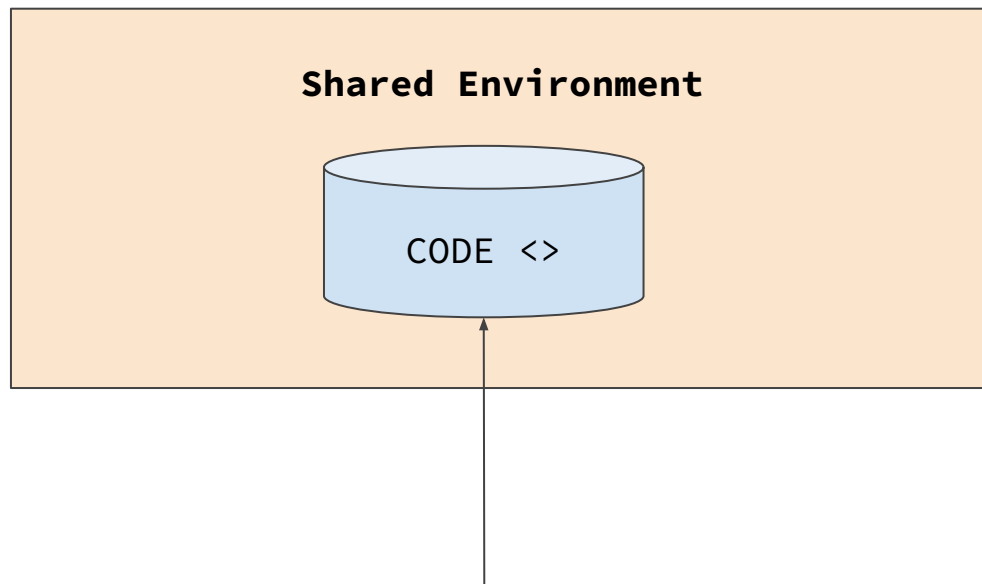
Faisal Memon (EmbarkX)



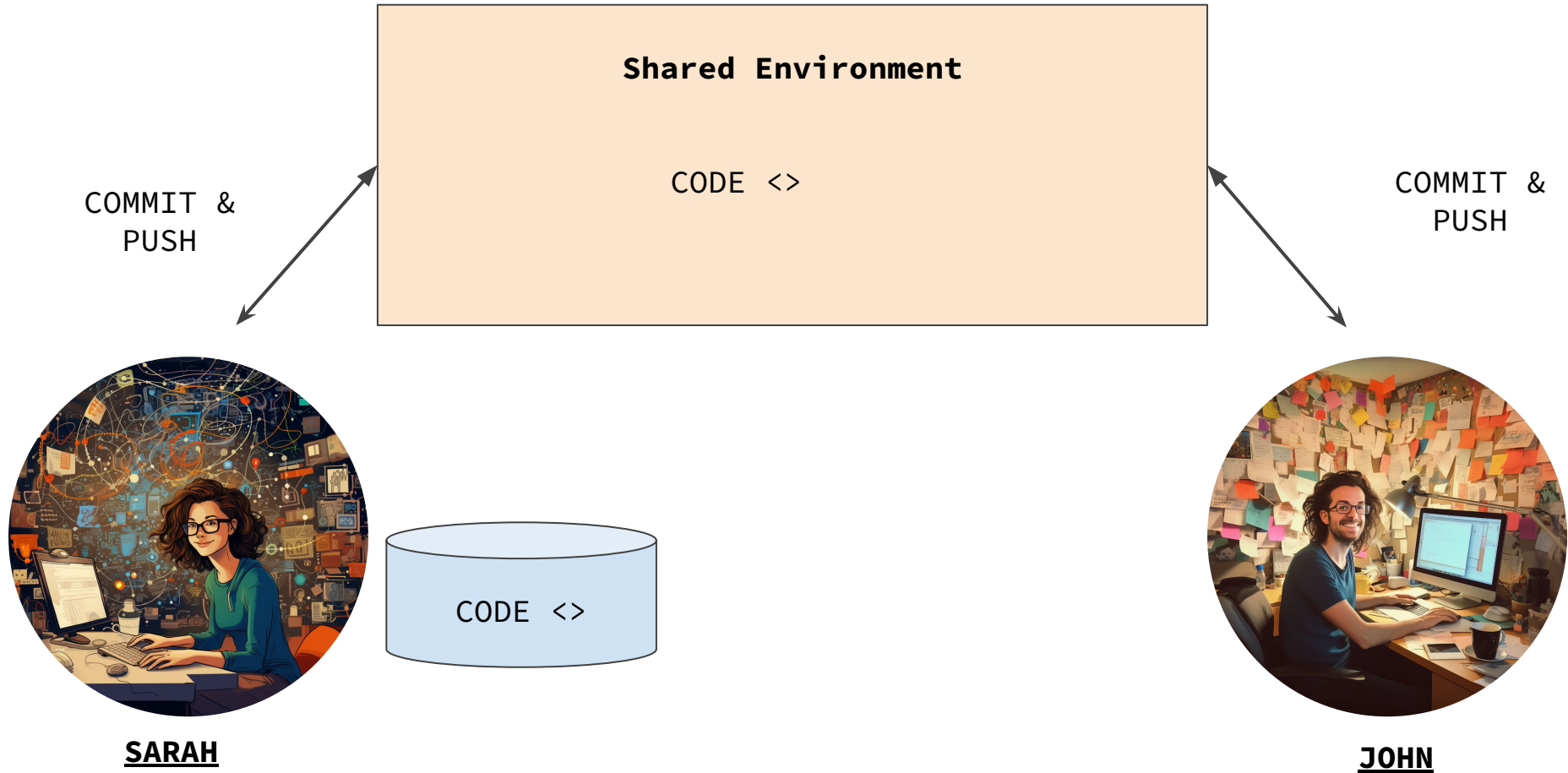
# Welcome Docker

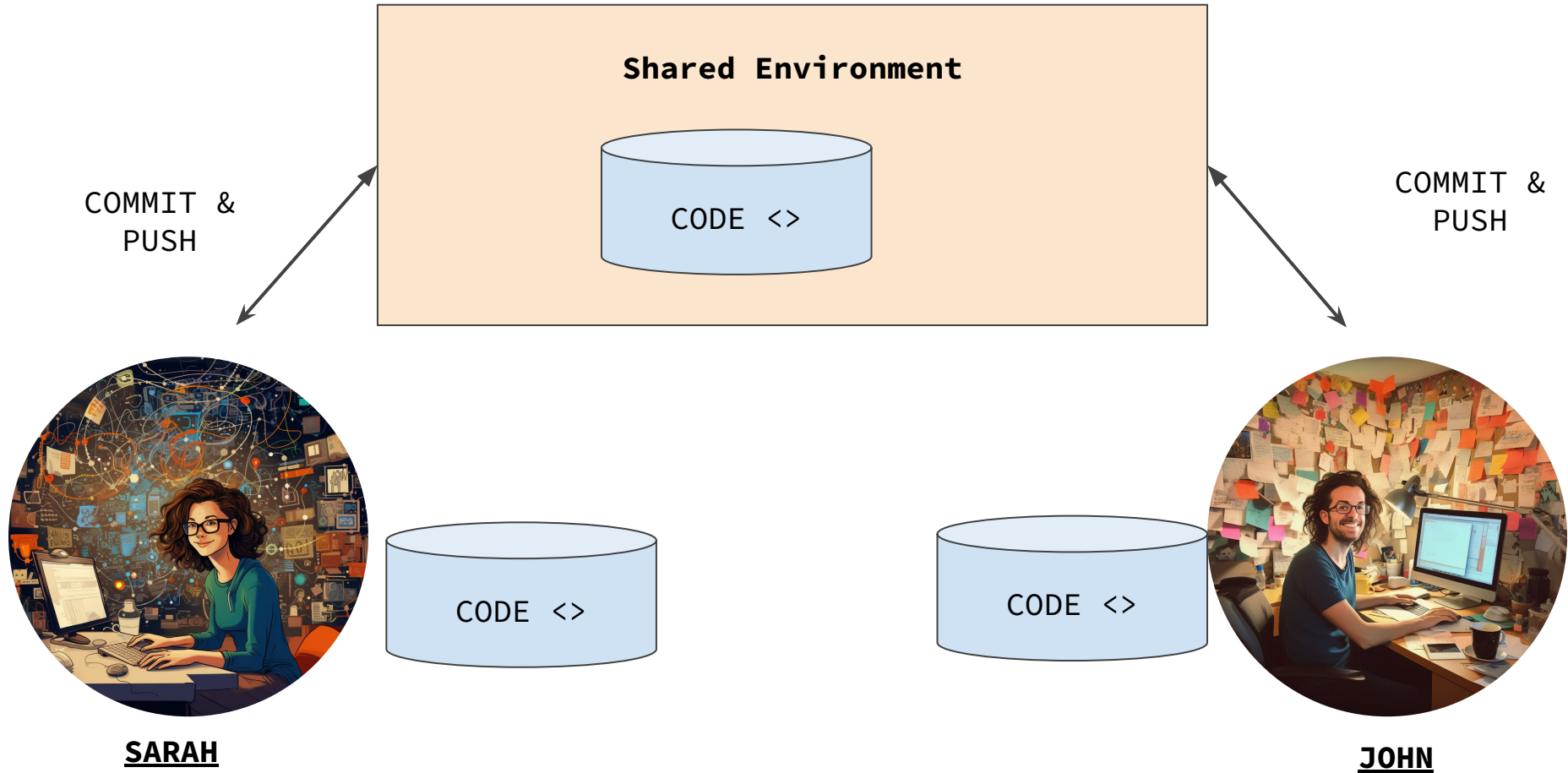
## Shared Environment

CODE <>



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



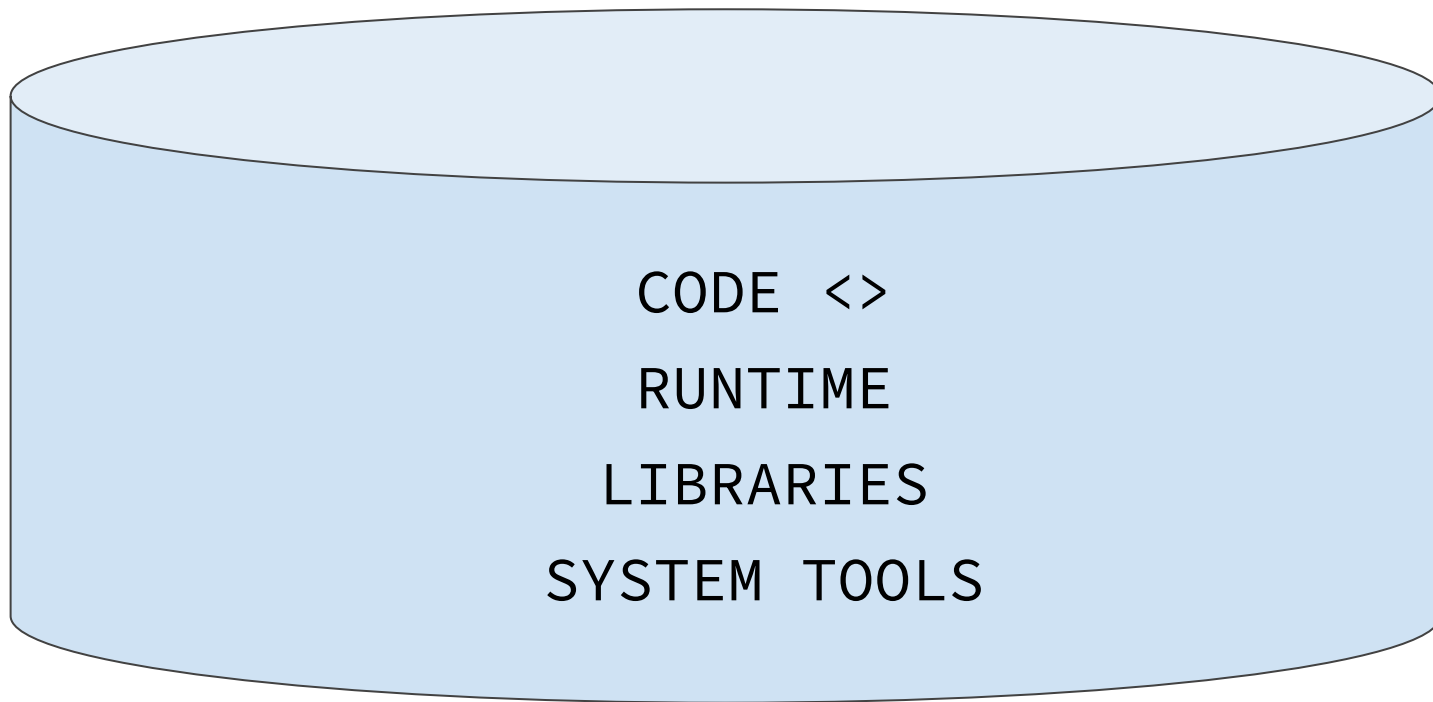




## **Let's Define Docker**

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

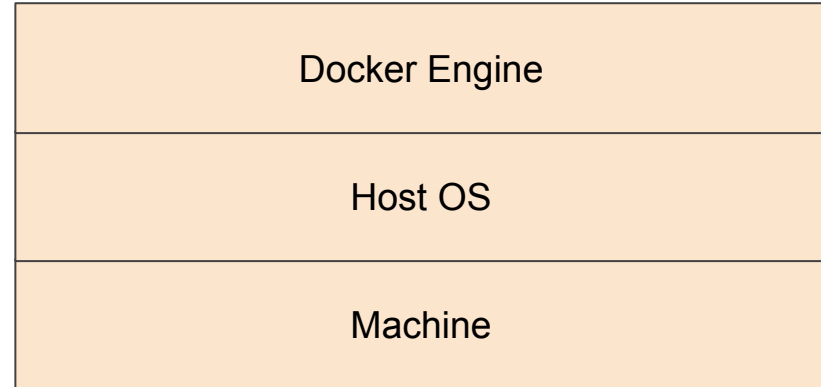
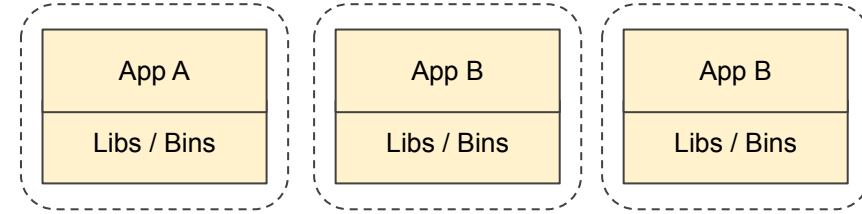
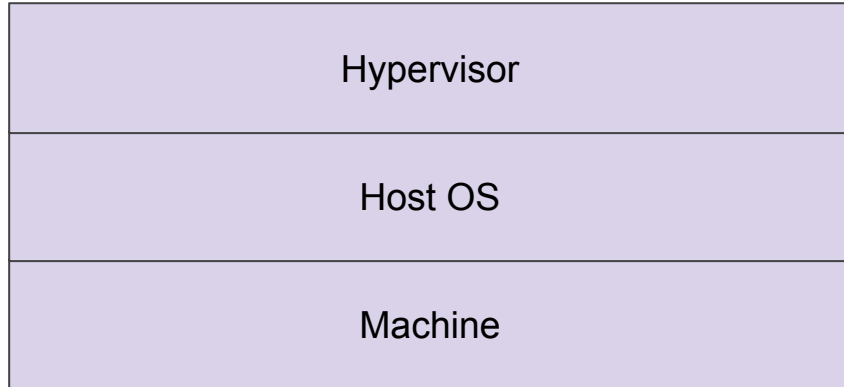
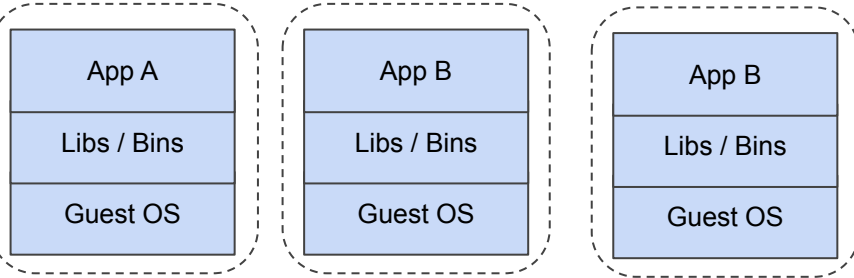
# **Docker Container**



# 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

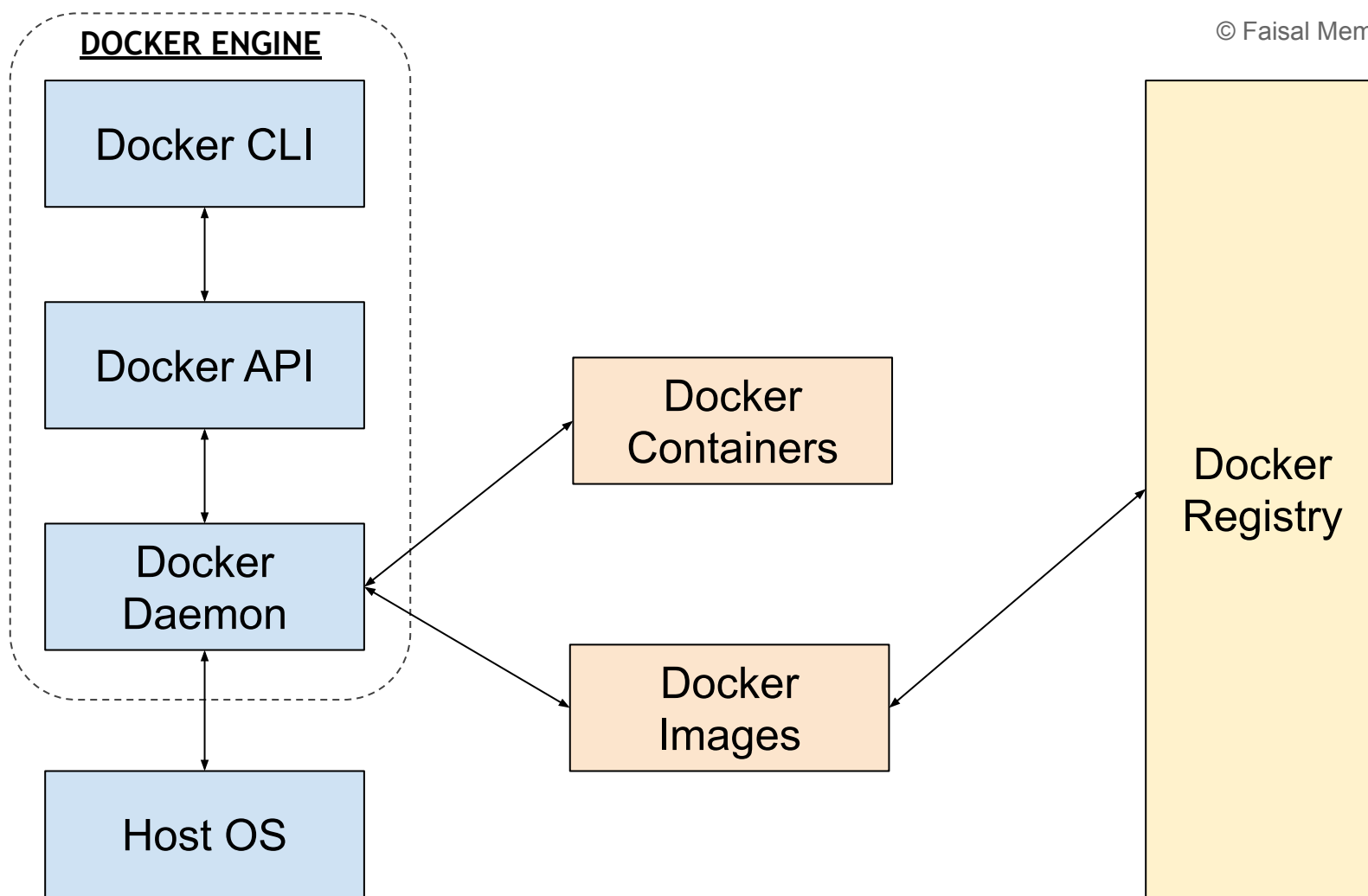


Parameters	Virtual Machines (VMs)	Docker Containers
<i>Size</i>	Relatively large and resource-intensive	Lightweight and resource-efficient
<i>Startup Time</i>	Longer boot time as full OS needs to start	Almost instant startup as no OS boot required
<i>Resource Utilization</i>	Utilizes more system resources (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

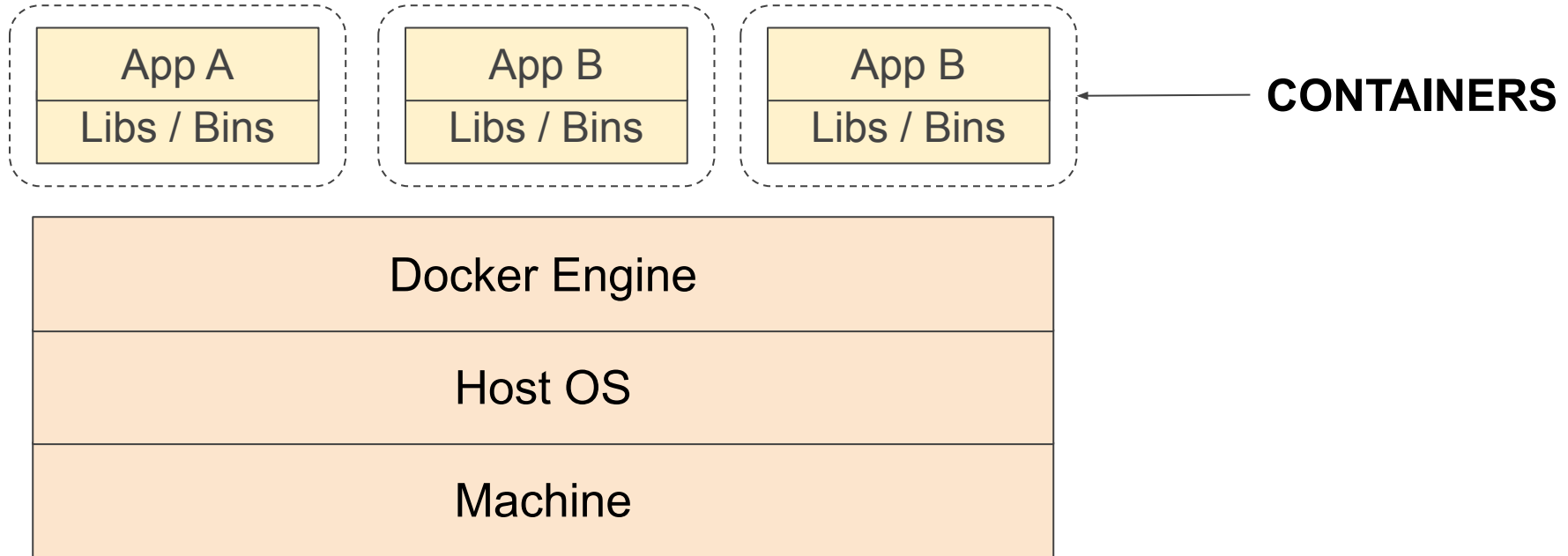
# Docker Architecture

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

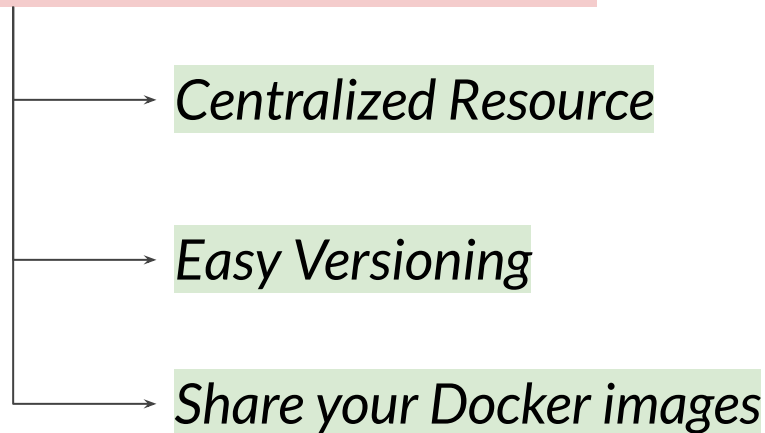
# Docker Registry

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*

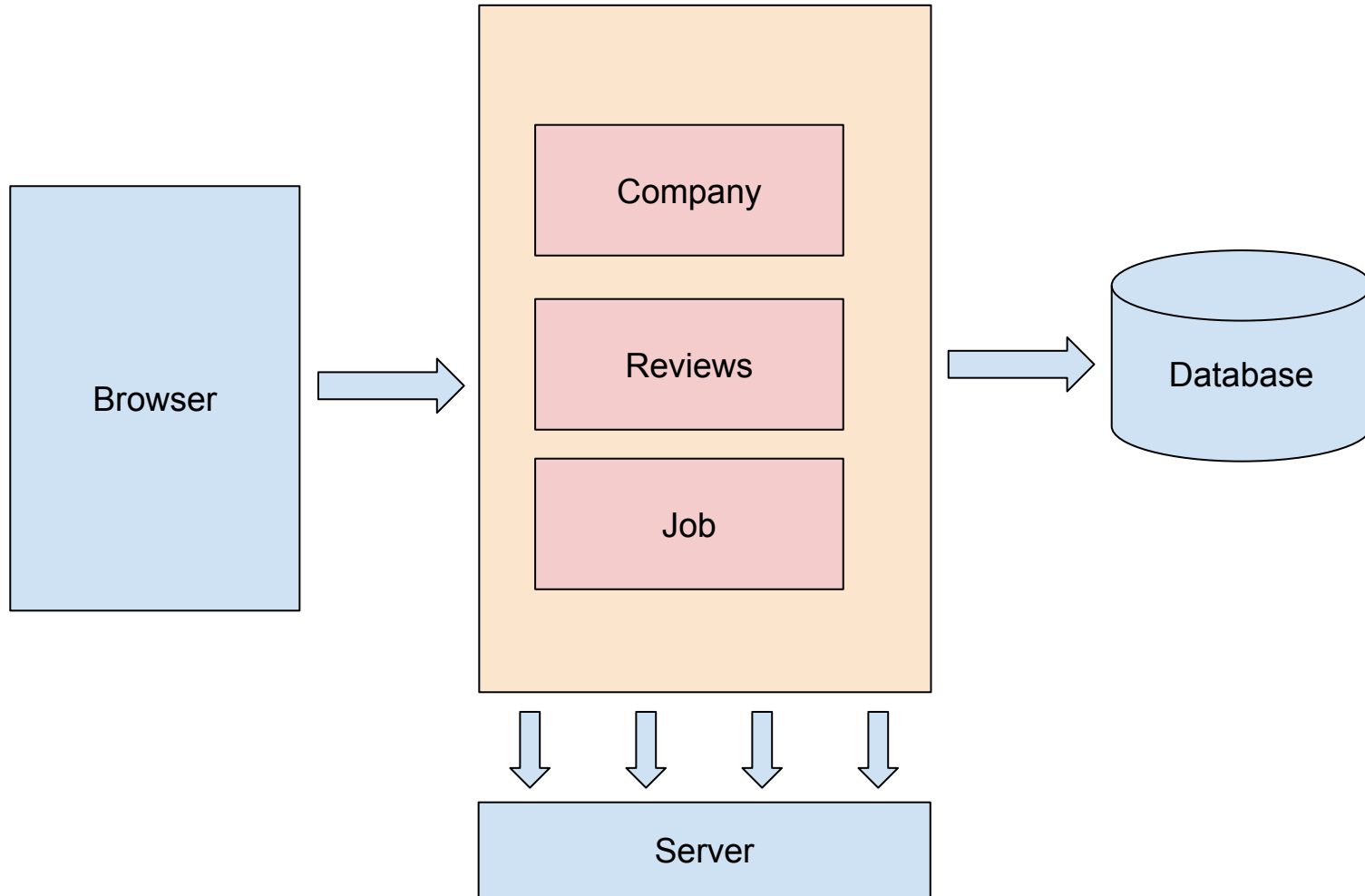
# Before Microservices

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*



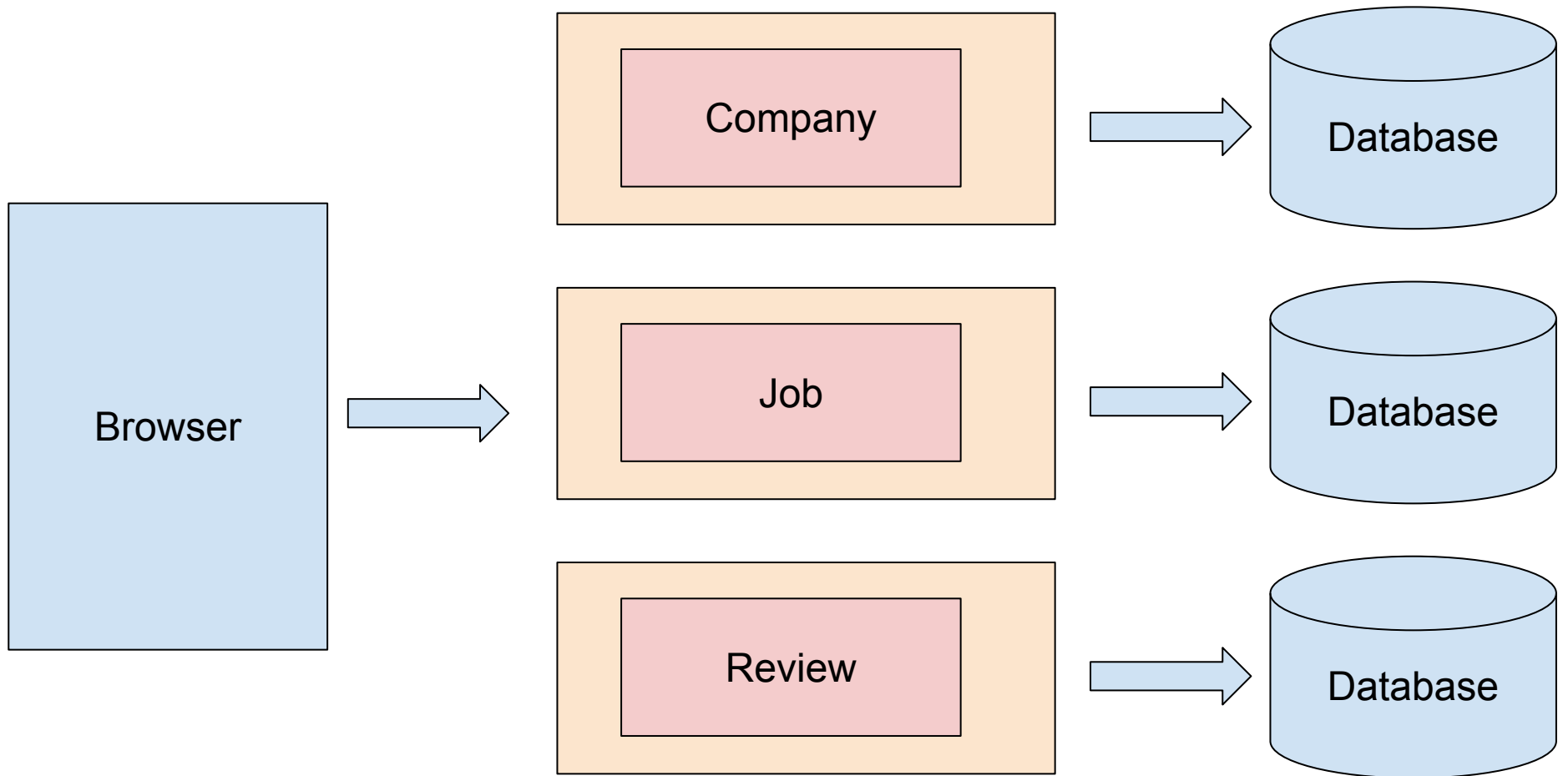
# Problems

- *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**

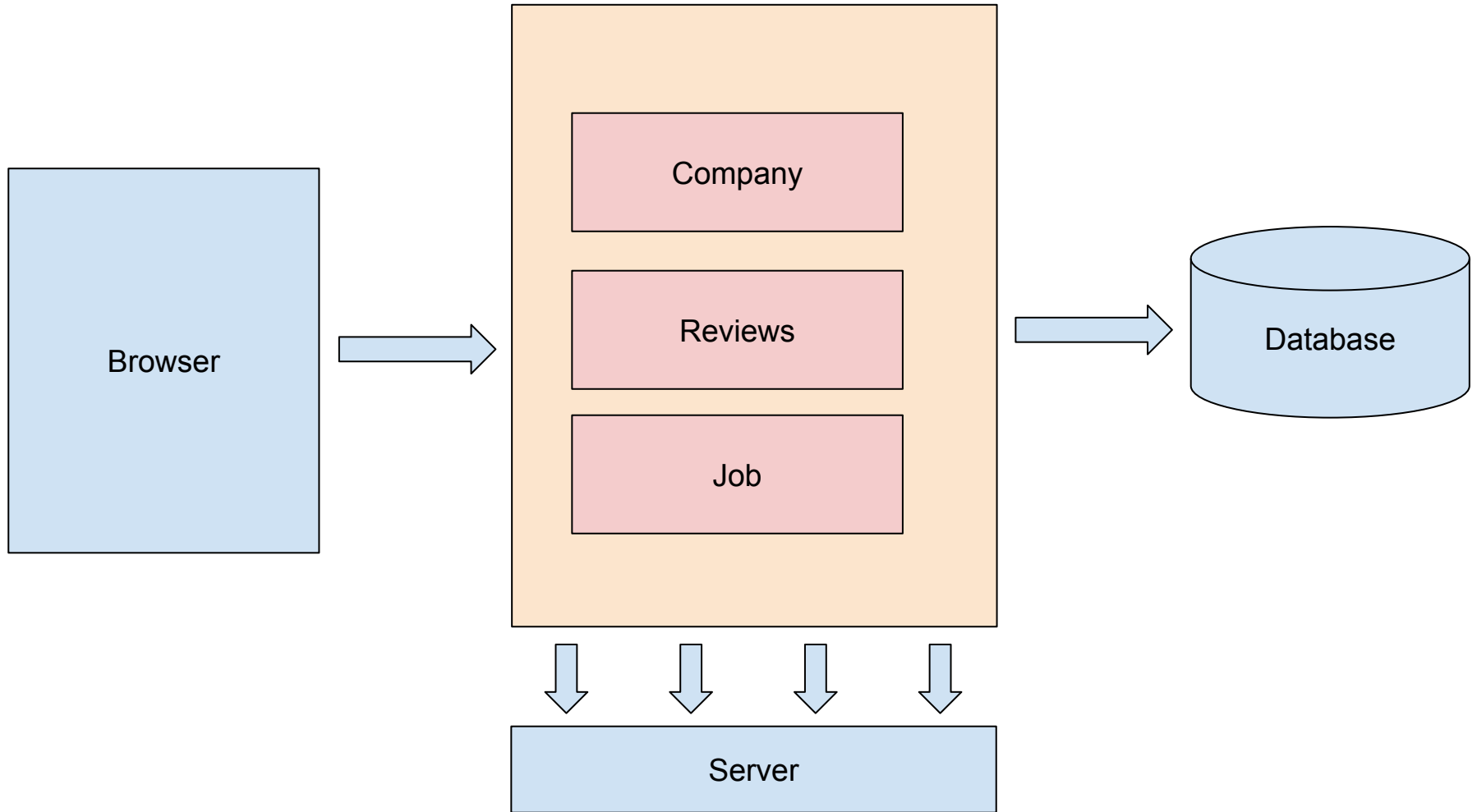
→ *Single Responsibility*

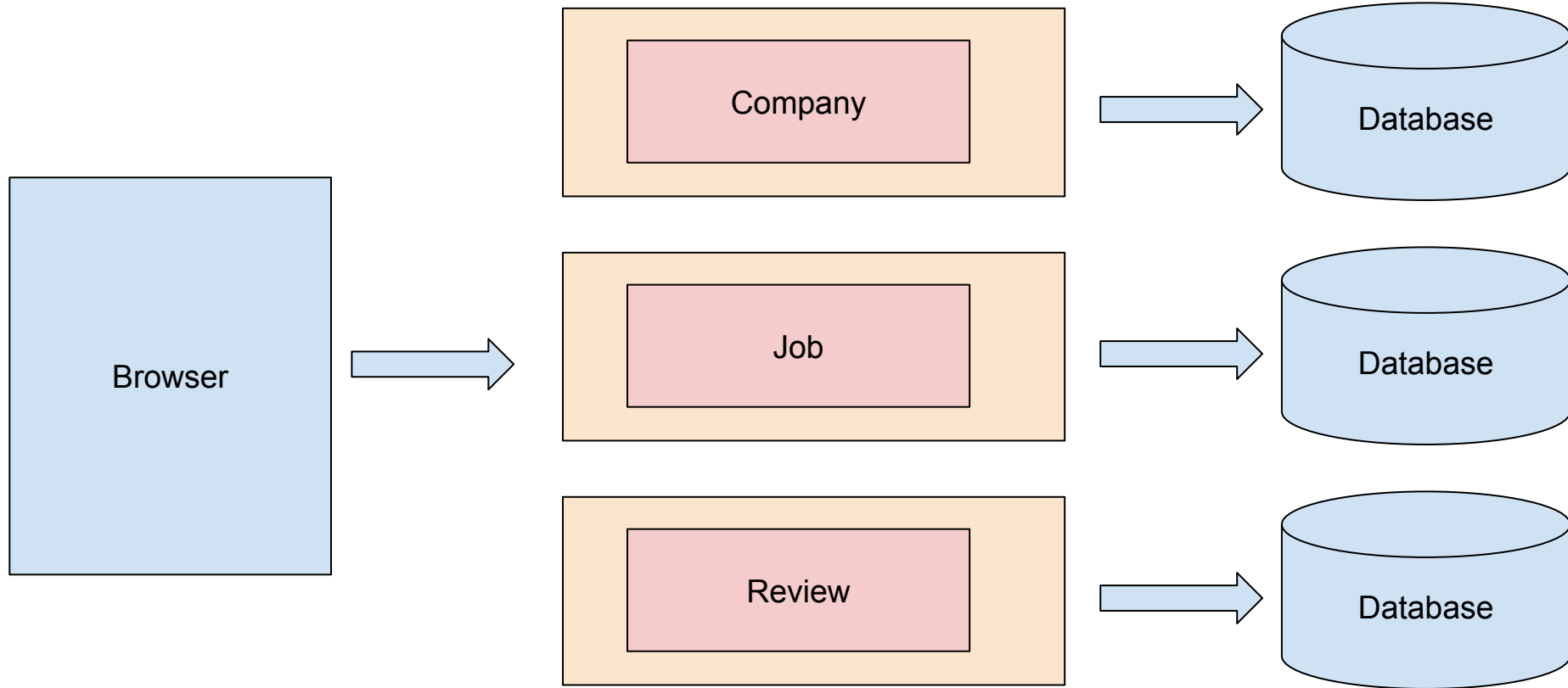
→ *Bounded Context*

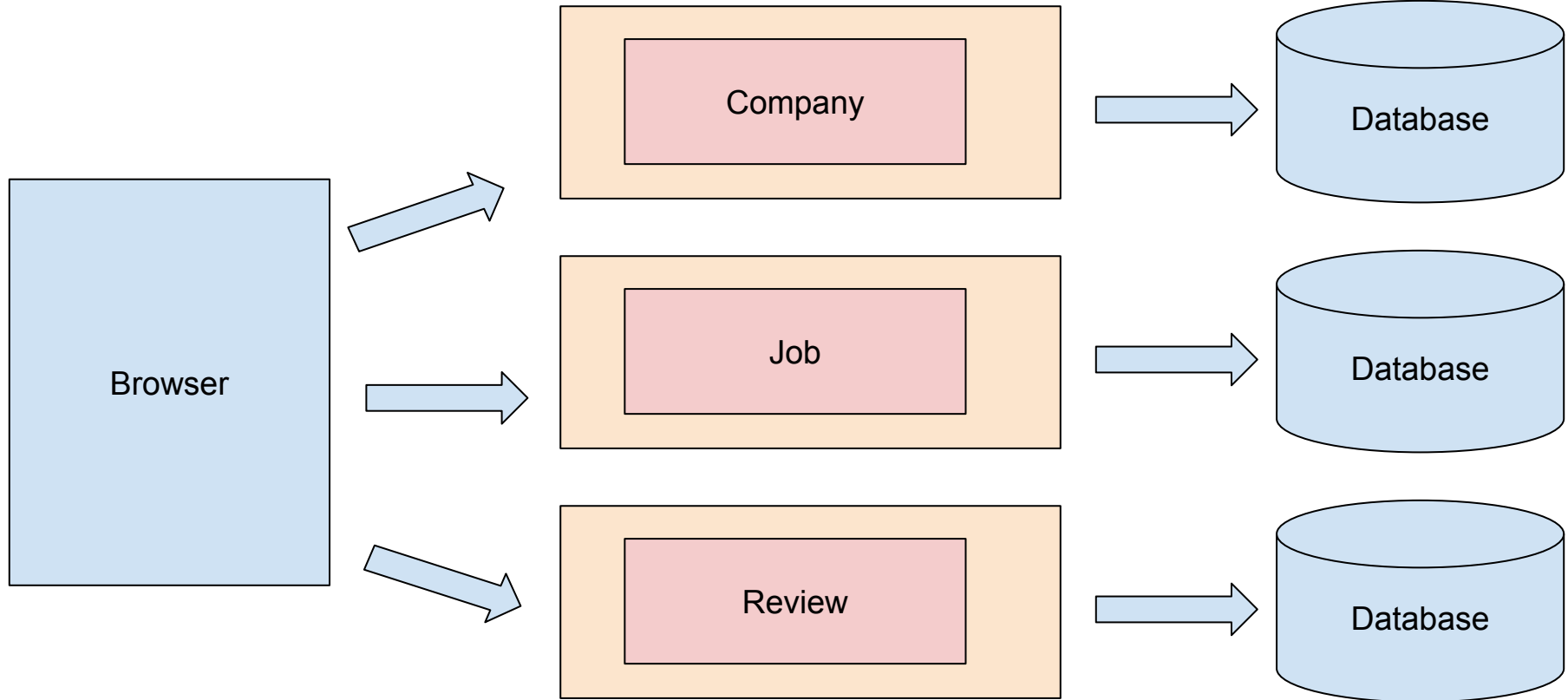
→ *Decentralized Data Management*

# Planning our changes

Faisal Memon (EmbarkX)







# How are we going to structure our Microservices

Faisal Memon (EmbarkX)



Service	Port
Company	8081
Jobs	8082
Reviews	8083

# Refactoring Review Service

Faisal Memon (EmbarkX)

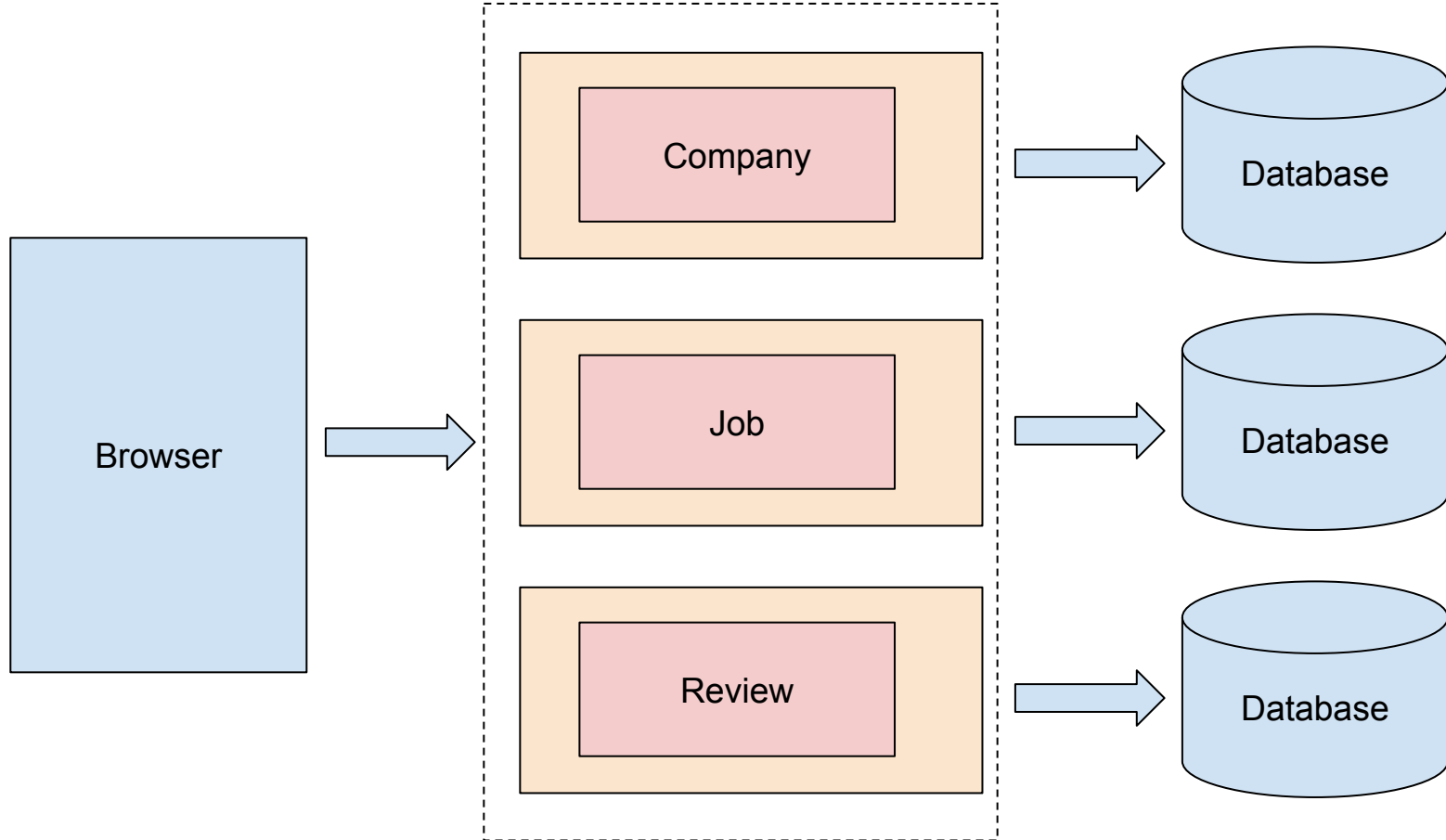
# Reviews

```
GET /companies/{companyId}/reviews
POST /companies/{companyId}/reviews
GET /companies/{companyId}/reviews/{reviewId}
PUT /companies/{companyId}/reviews/{reviewId}
DELETE /companies/{companyId}/reviews/{reviewId}
```

```
GET /reviews?companyId={companyId}
POST /reviews?companyId={companyId}
GET /reviews/{reviewId}
PUT /reviews/{reviewId}
DELETE /reviews/{reviewId}
```

# 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*

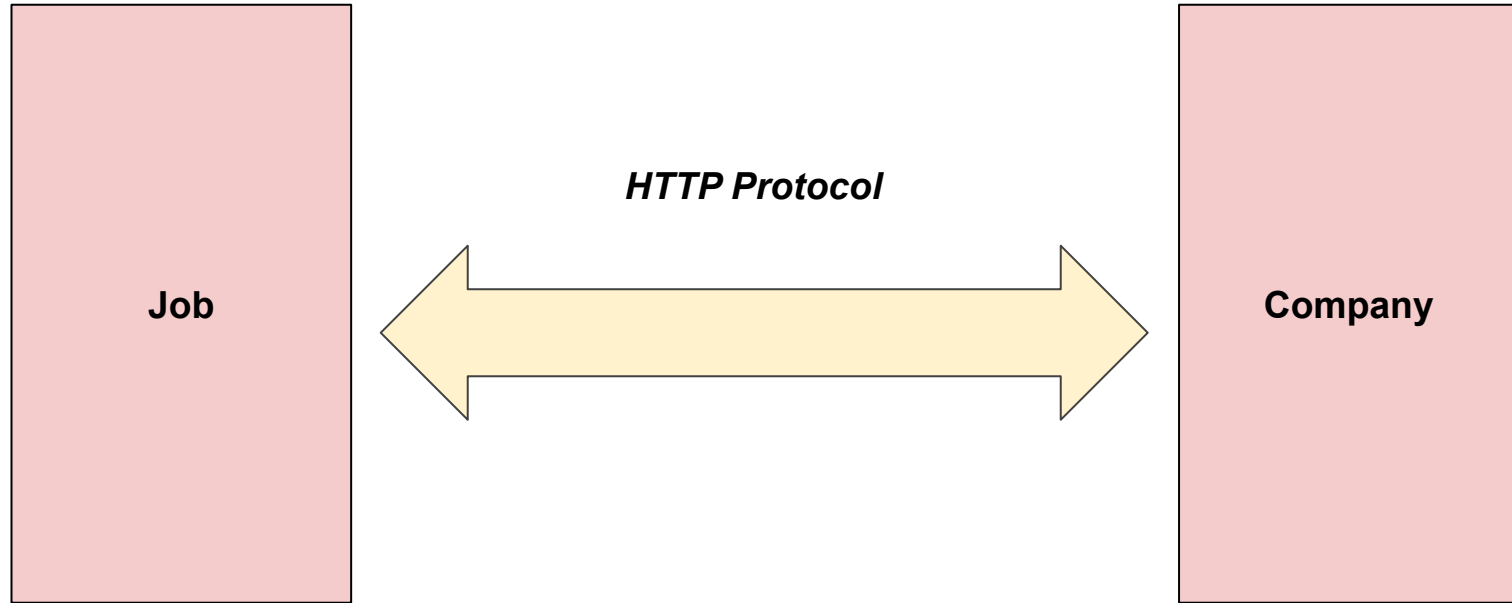
*Asynchronous Communication*

# What Is REST Template and Why Do You Need It?

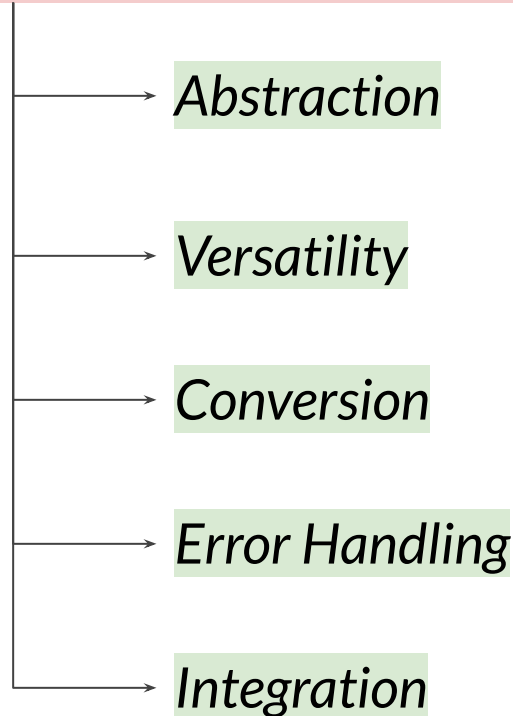
Faisal Memon



# RestTemplate



# *Features and Advantages of RestTemplate*

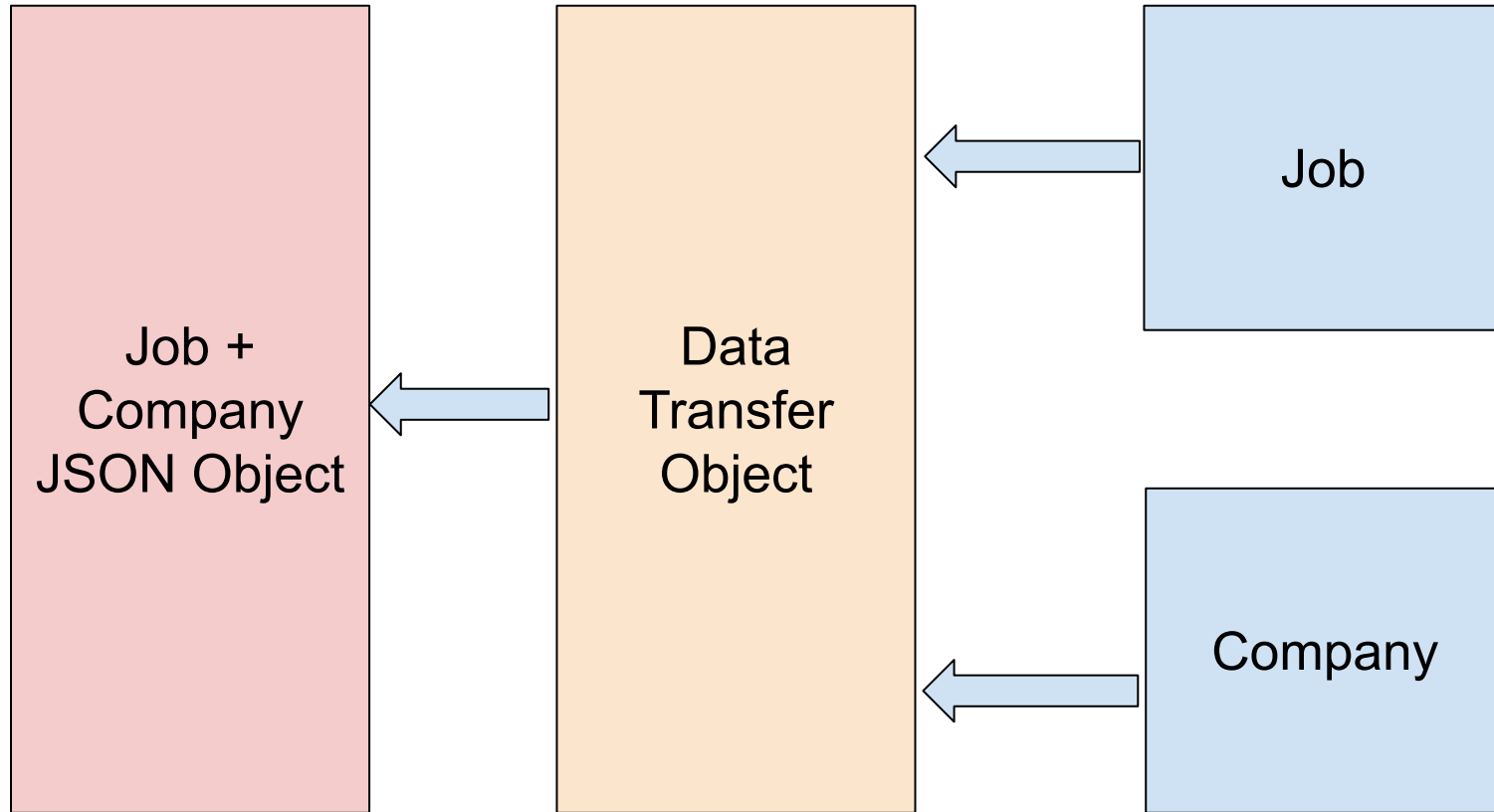


# DTO Pattern

Faisal Memon (EmbarkX)

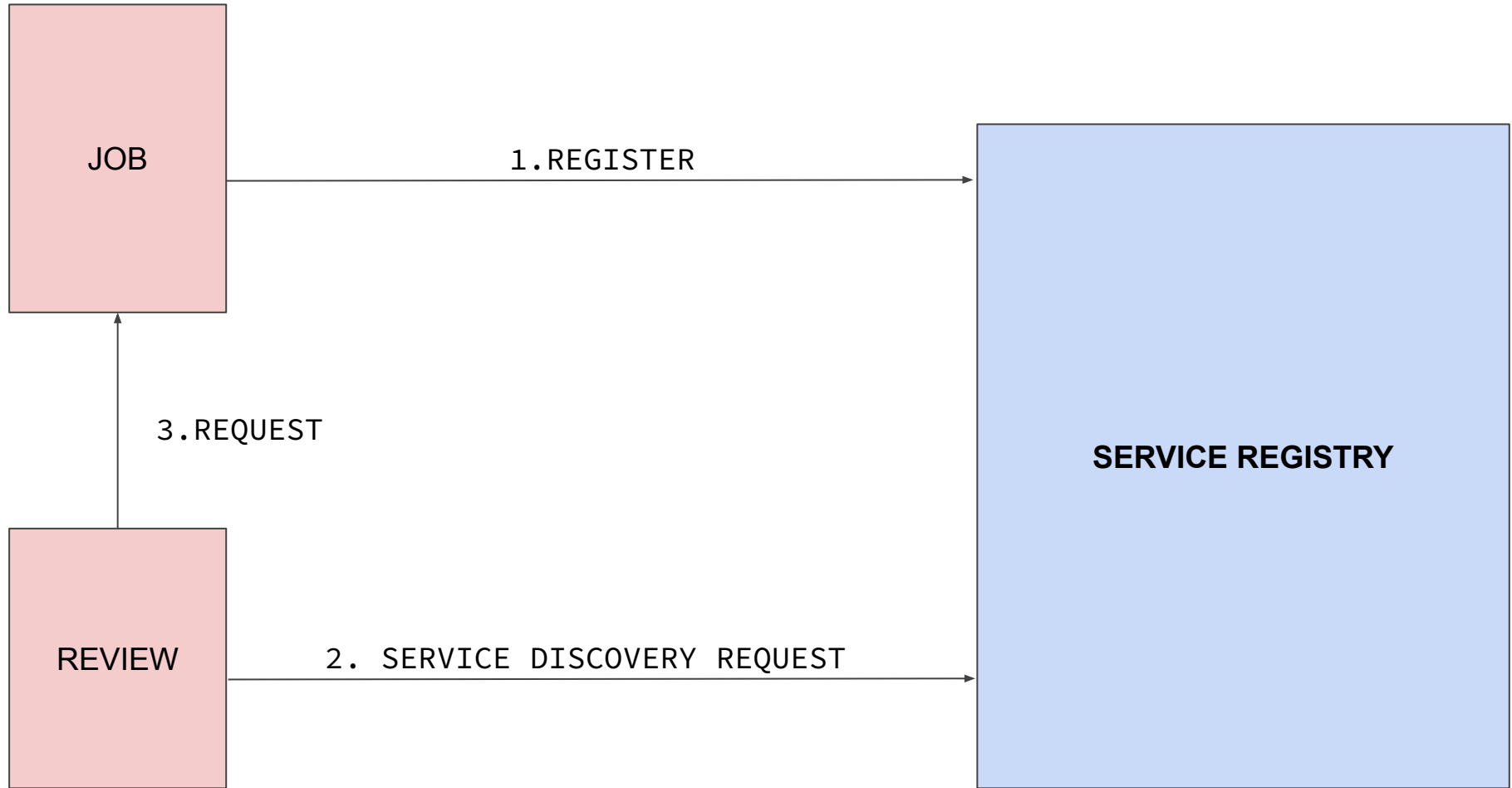
## **What is it?**

*Design pattern used to transfer data between software application subsystems*

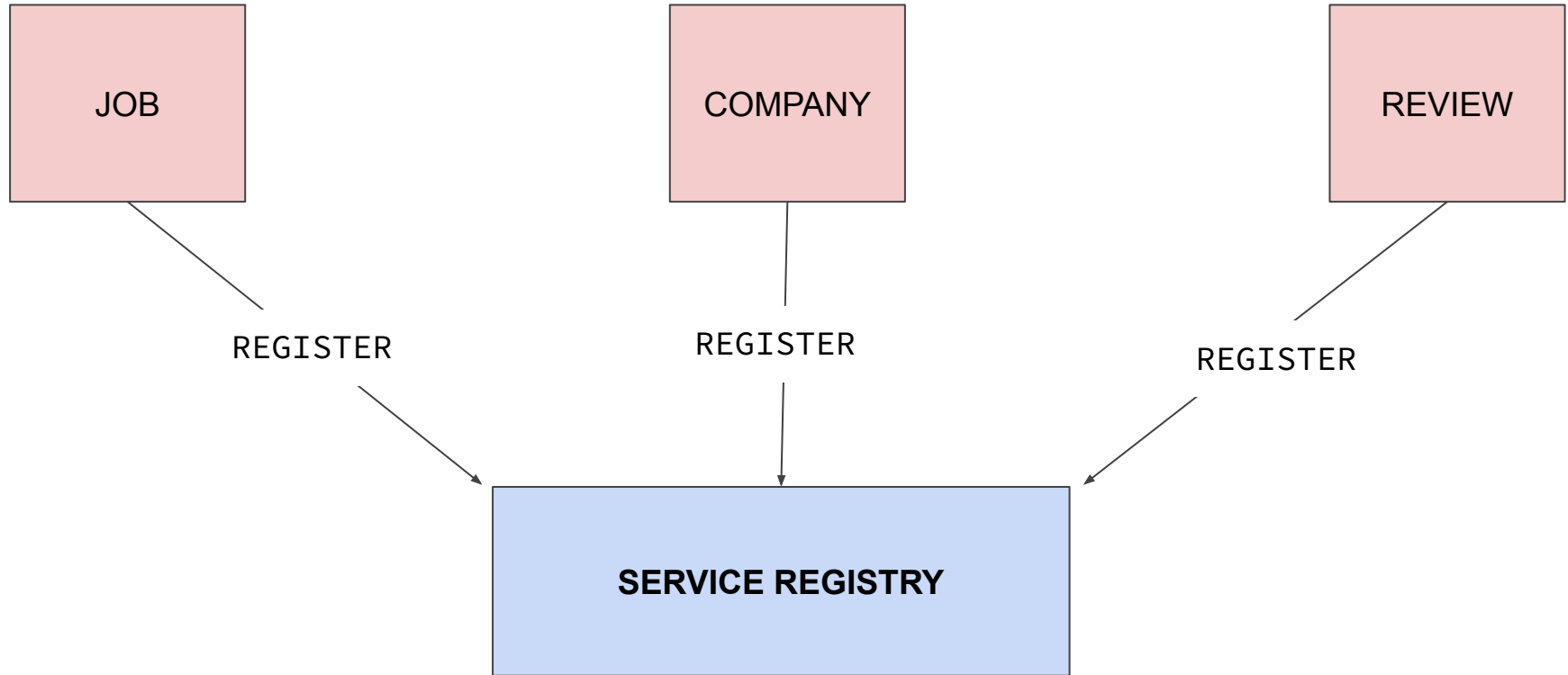


# Behind the Scenes of Eureka Server

Faisal Memon (EmbarkX)







# **Behind the scenes**

→ *Heartbeat Signal*

→ *Heartbeat Monitoring*

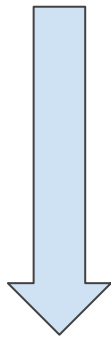
# Open Feign: An Introduction

Faisal Memon

## **What is it?**

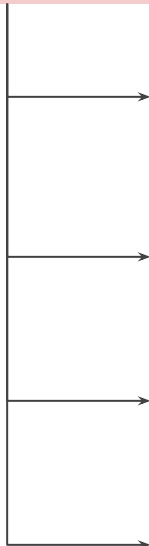
*Feign is a declarative web service client designed to make writing HTTP clients easier*

```
Company company =  
restTemplate.getForObject("http://COMPANY-SERVICE/companies/" +  
job.getCompanyId(), Company.class);
```



```
Company company = companyClient.getCompany(job.getCompanyId());
```

# Why use OpenFeign



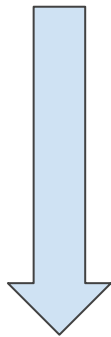
*Ease of use*

*Integration with Eureka*

*Built-in Load Balancing with Ribbon*

*Support for FallBacks and Circuit Breakers*

```
Company company =  
restTemplate.getForObject("http://COMPANY-SERVICE/companies/" +  
job.getCompanyId(), Company.class);
```



```
Company company = companyClient.getCompany(job.getCompanyId());
```

# Introduction to Distributed Tracing


Faisal Memon (EmbarkX)



*How would you track a request from start to end?*

***Distributed Tracing*** enables you to trace your request from  
start to end

# ***Problems that Distributed Tracing Solves***

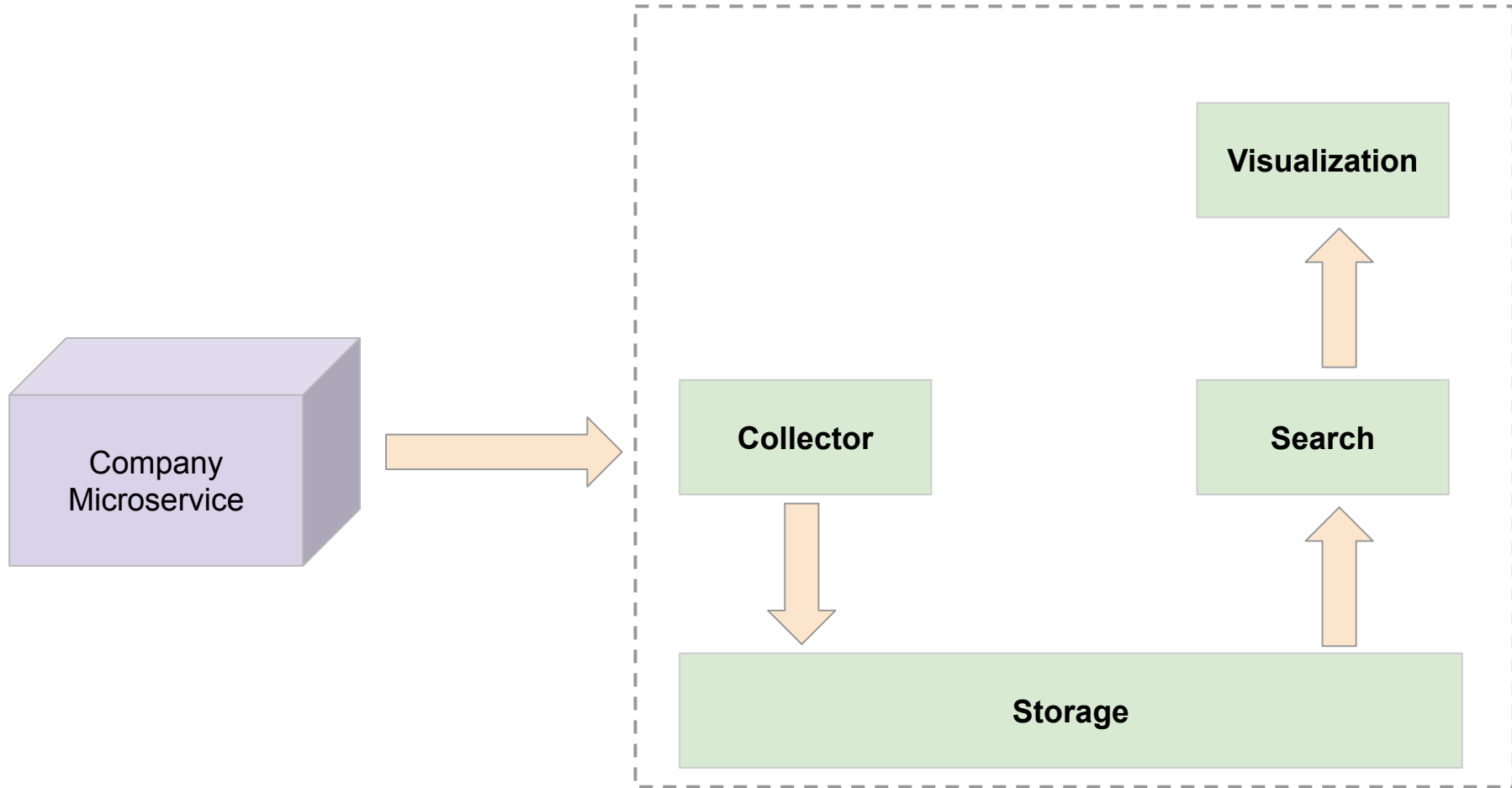
- 
- *Request Visualization*
  - *Identify Performance Bottleneck*
  - *Error Analysis and Debugging*
  - *Tracking Dependency*
  - *Performance Optimization*

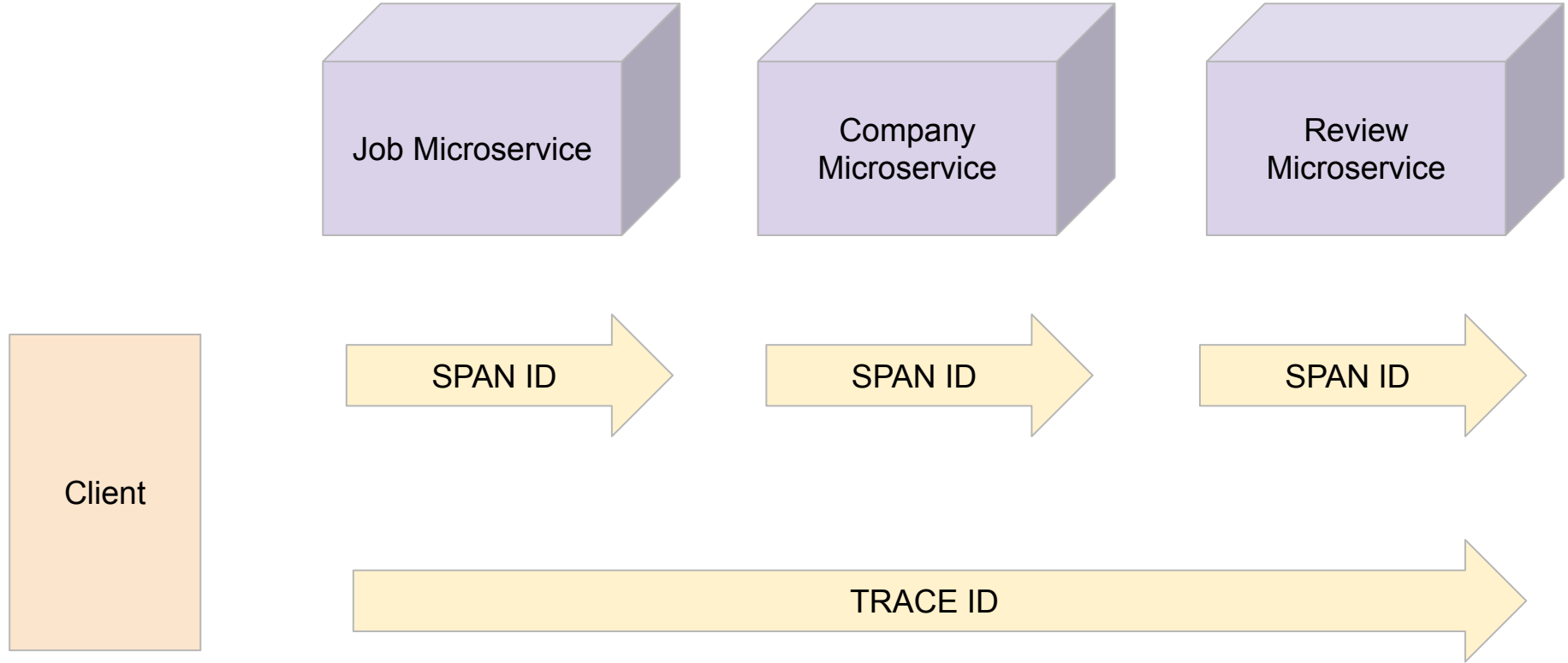
# Introduction to Zipkin

Faisal Memon (EmbarkX)

# **What is Zipkin?**

*Zipkin is an open-source distributed tracing system*





# Introduction to Micrometer

Faisal Memon (EmbarkX)



## **What is Micrometer?**

*Micrometer provides insights that help you keep tabs on your application's performance*

# **How Micrometer Helps**

- *Helps you collect metrics from your application*
- *Acts as a middleman or a bridge between your application and the metrics collection systems*
- *It offers a vendor-neutral interface*
- *You can abstract away the complexities of interacting with different metrics collection systems*
- *Micrometer simplifies the process of collecting metrics from your application*

# Common problems

Faisal Memon (EmbarkX)

**"Missing Trace" problem**

# **Best Practices**

→ *Use consistent naming conventions*

→ *Secure your Zipkin server*

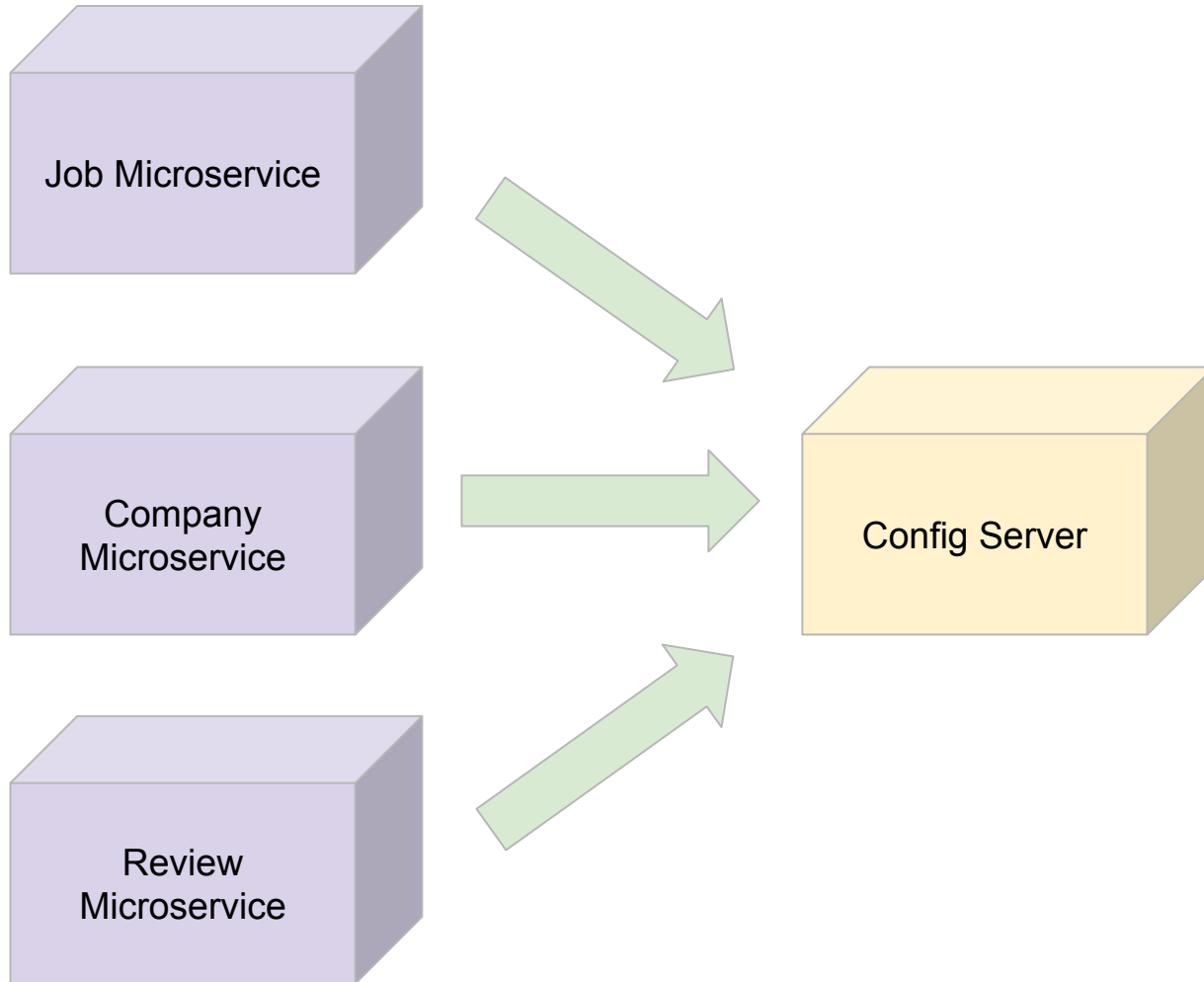
→ *Know that - Distributed tracing does come with a performance impact*

# Introduction to Configuration Management

**Faisal Memon (EmbarkX)**

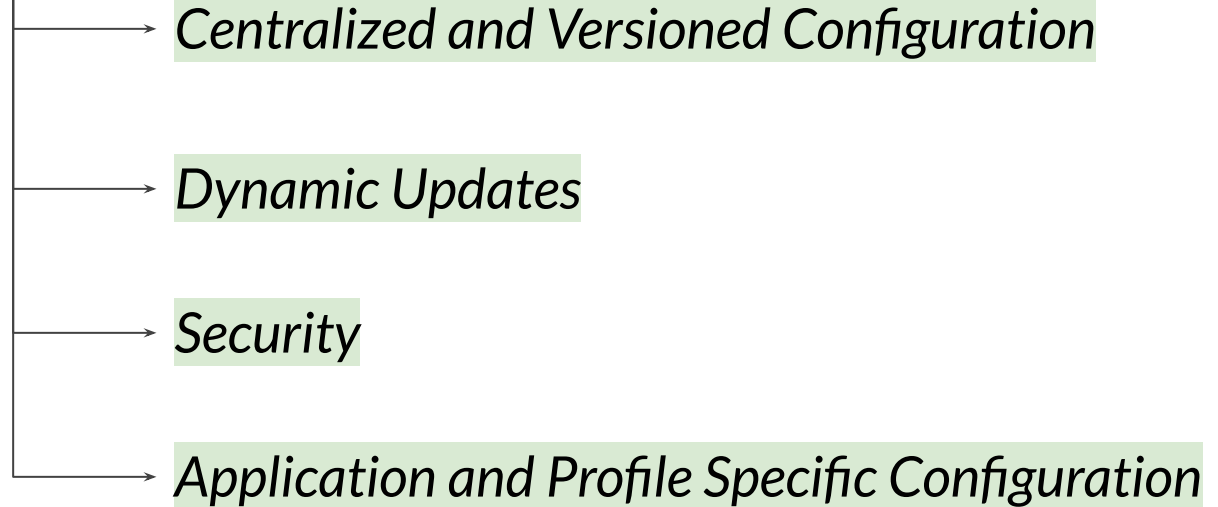
# Configuration Management

- 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
- **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*

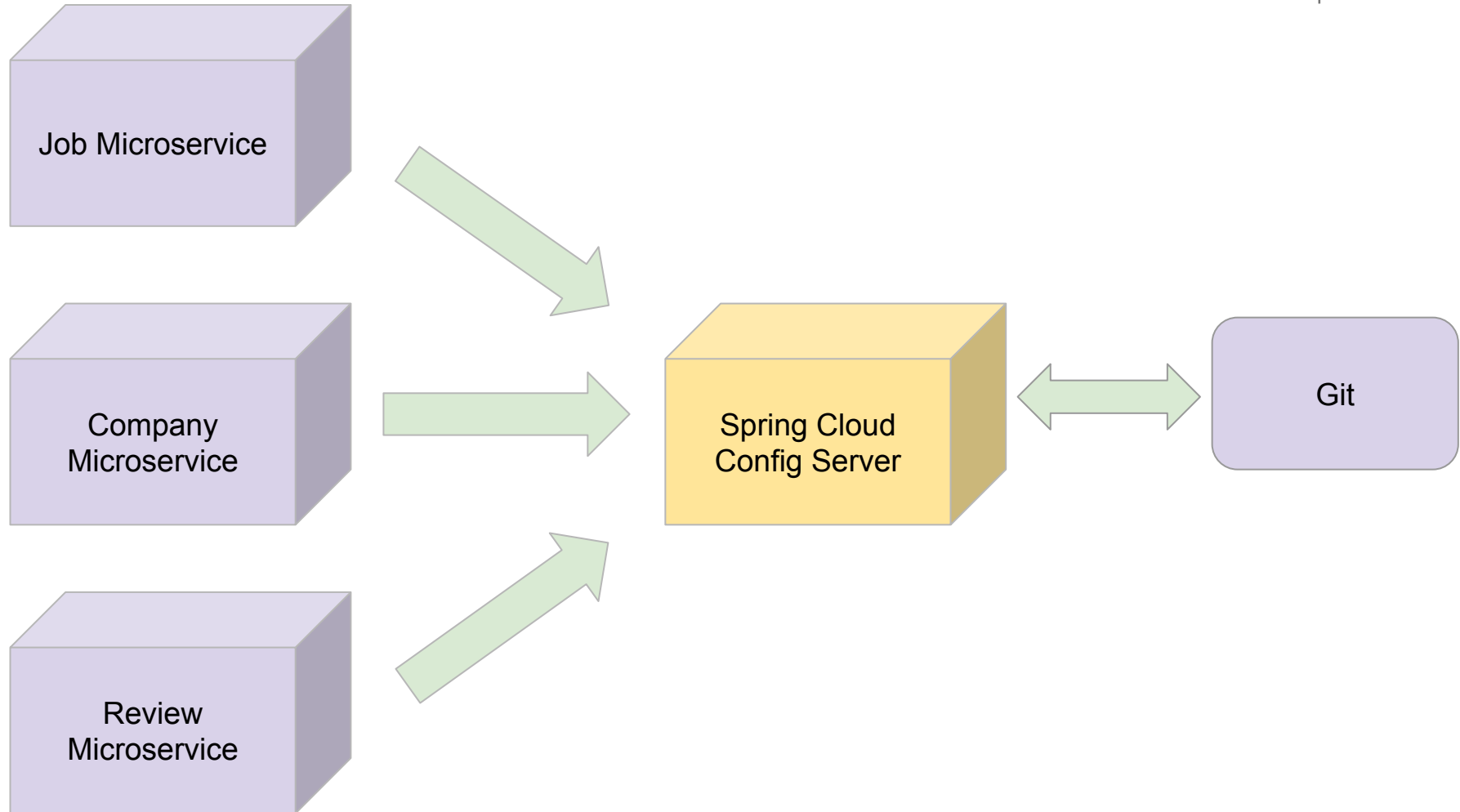
# Benefits of a Config Server

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

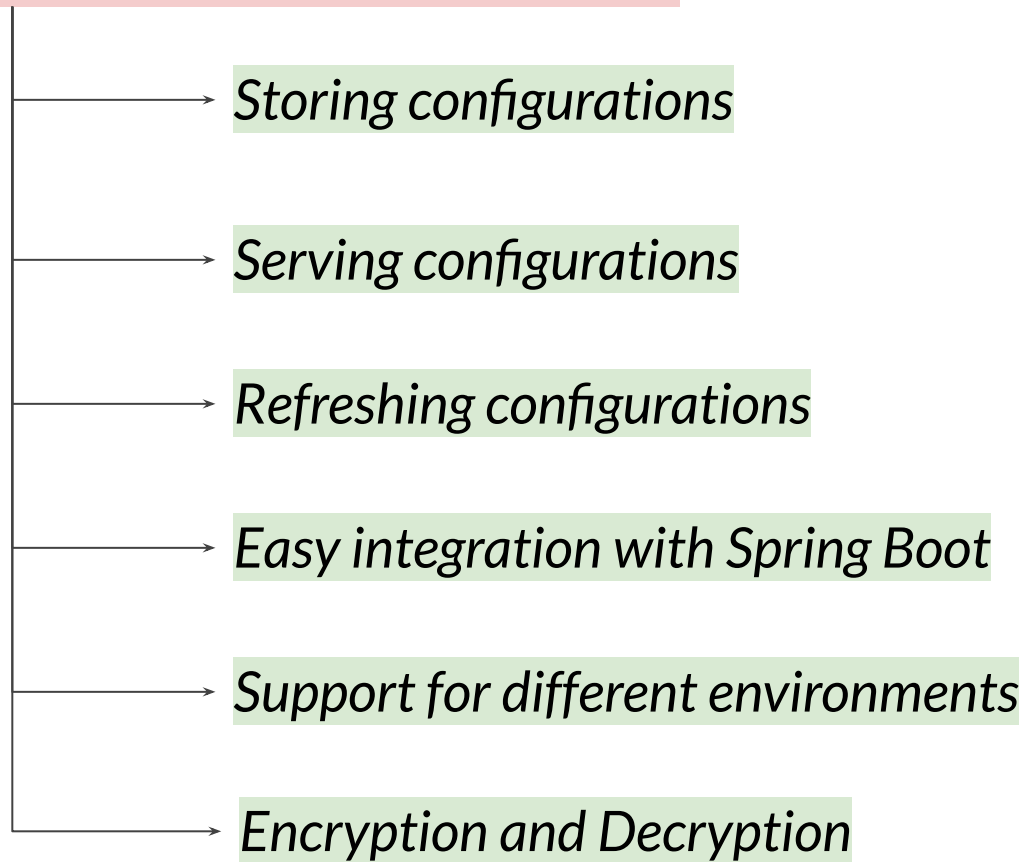
# Spring Cloud Config Server

Faisal Memon (EmbarkX)

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

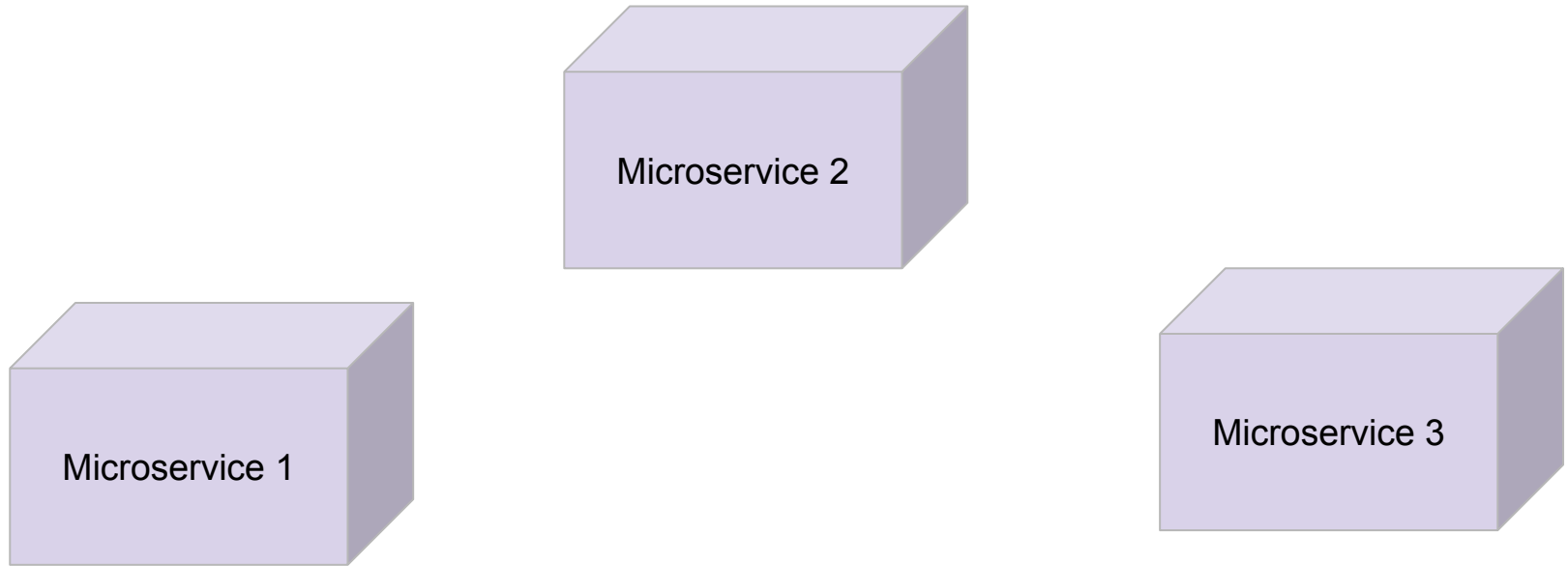


# Spring Cloud Config Server



# Introduction to API Gateways

Faisal Memon (EmbarkX)

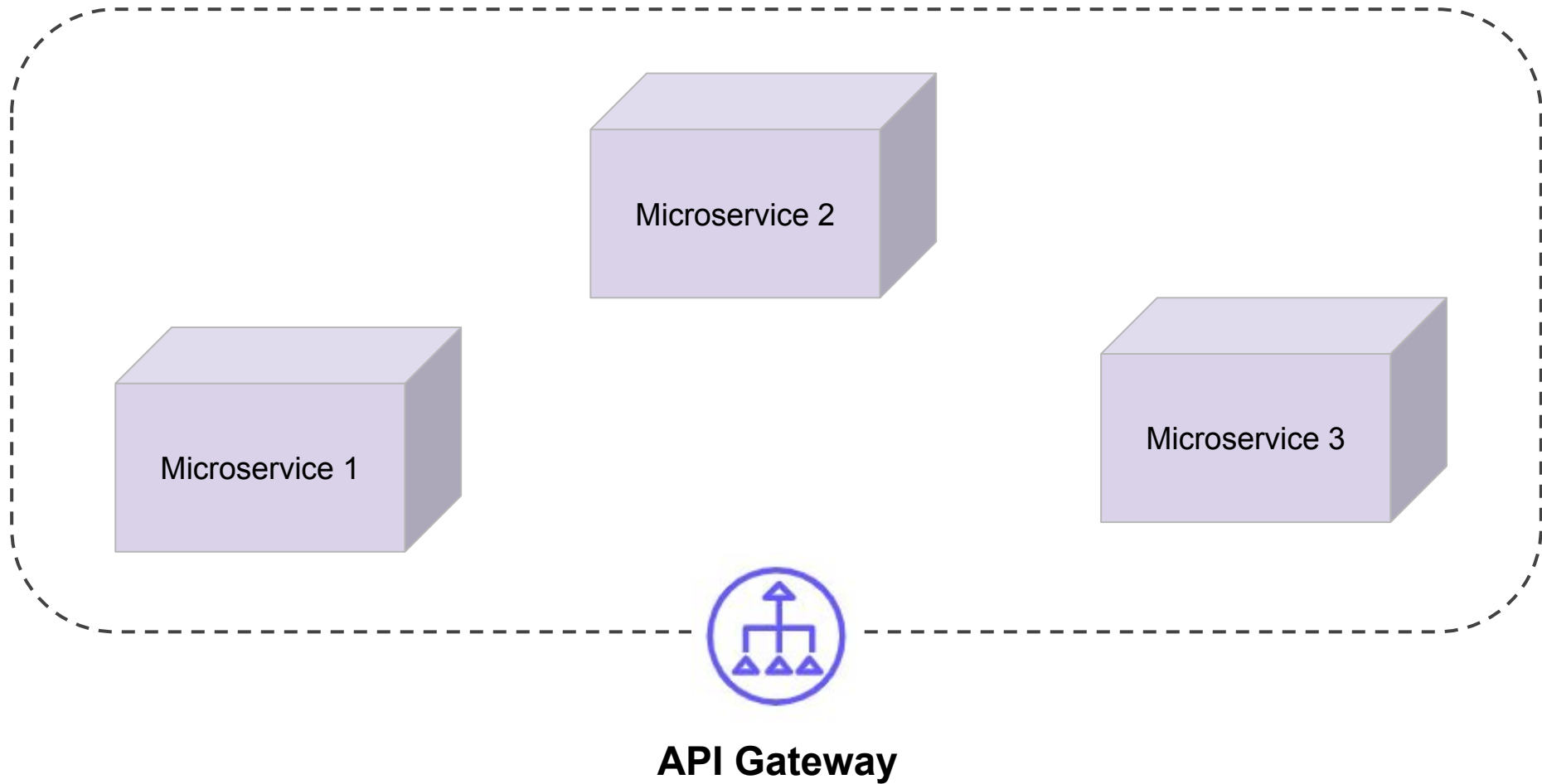


**Microservices  
Architecture**

=

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





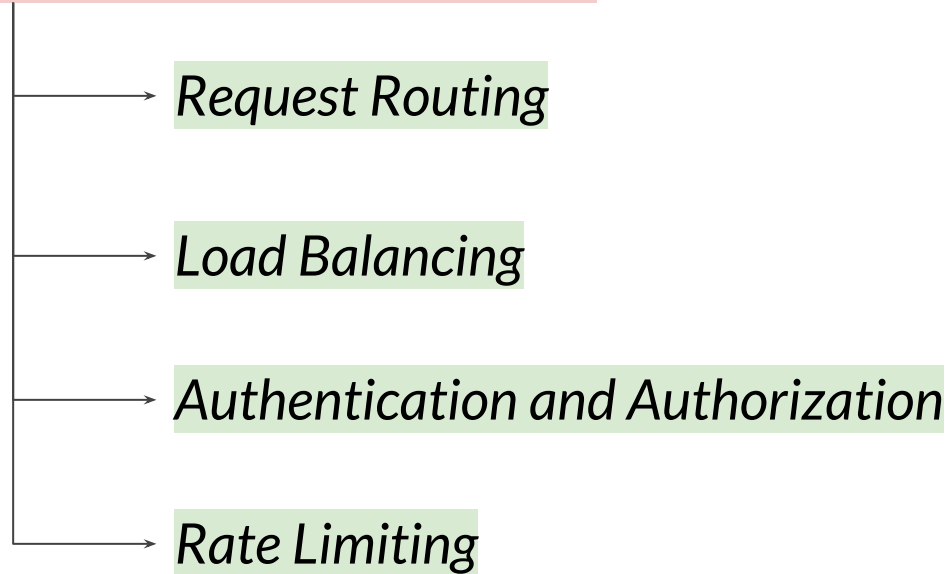
# Advantages

- *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*

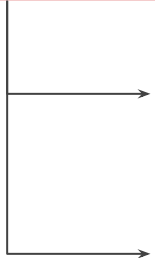
# API Gateway Functions

Faisal Memon (EmbarkX)

# Capabilities of API Gateway

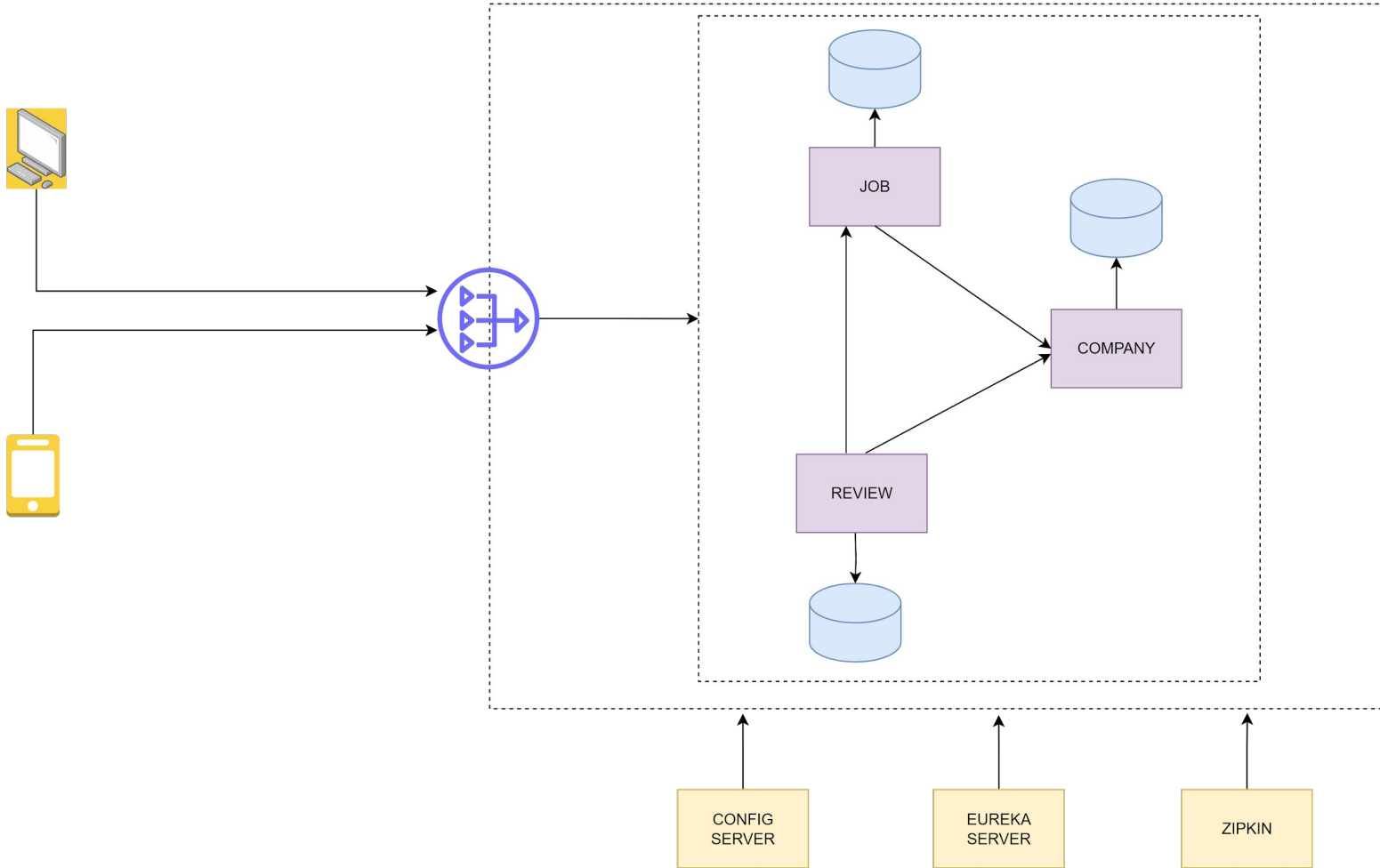


# Capabilities of API Gateway



*Request and Response Transformation*

*Aggregation of Data from Multiple Services*



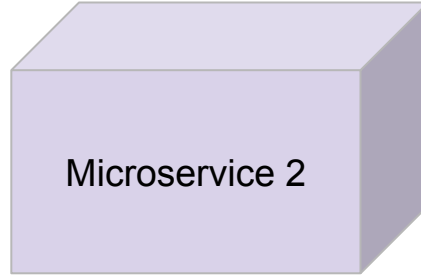
# Introduction to Fault Tolerance

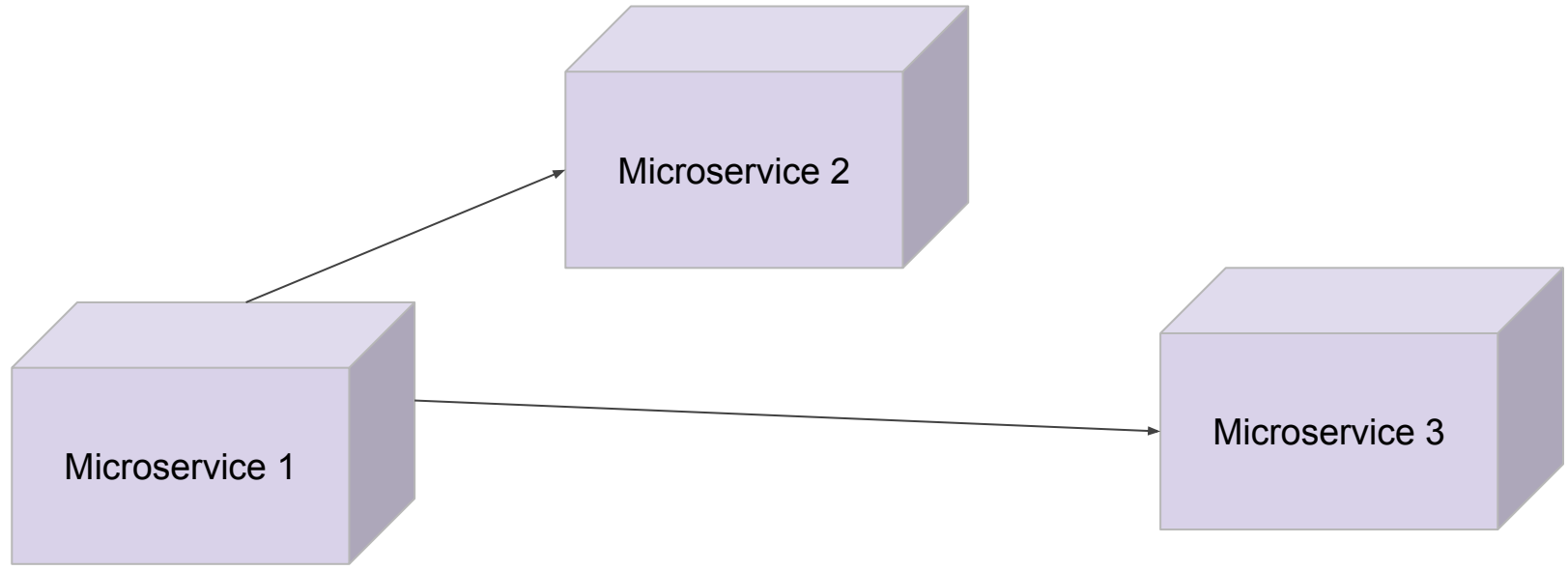
Faisal Memon (EmbarkX)

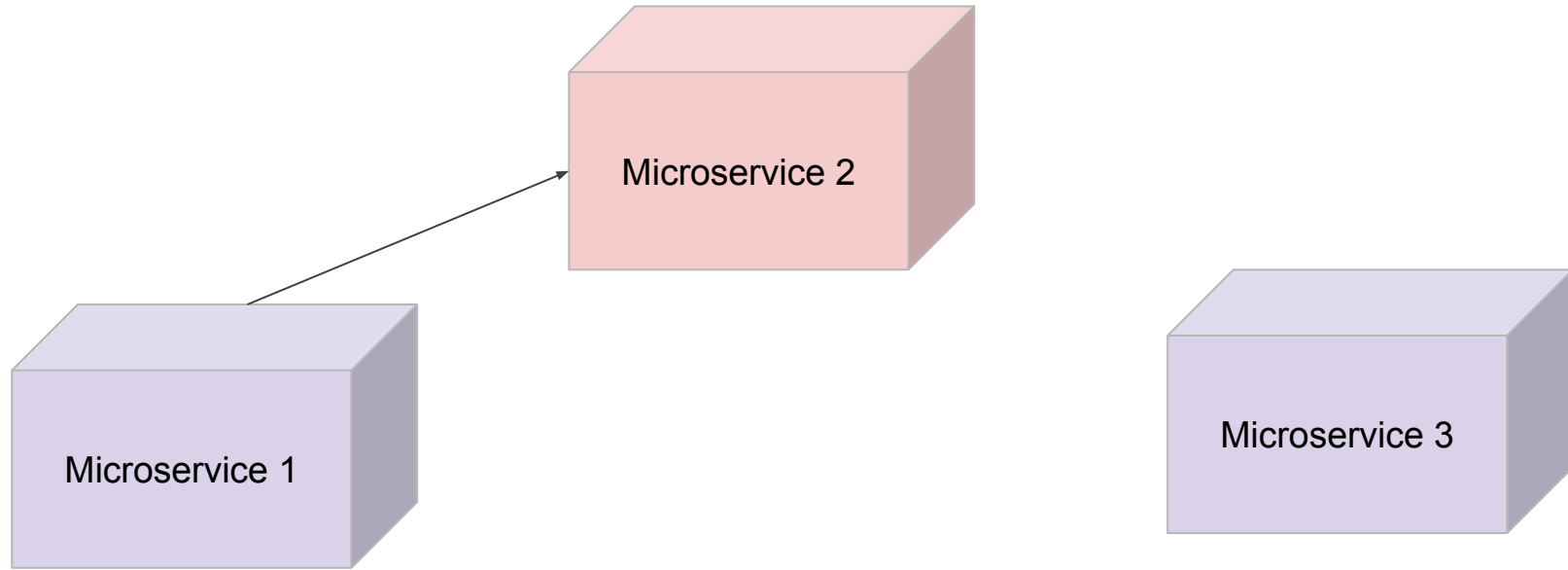
# **Fault Tolerance**

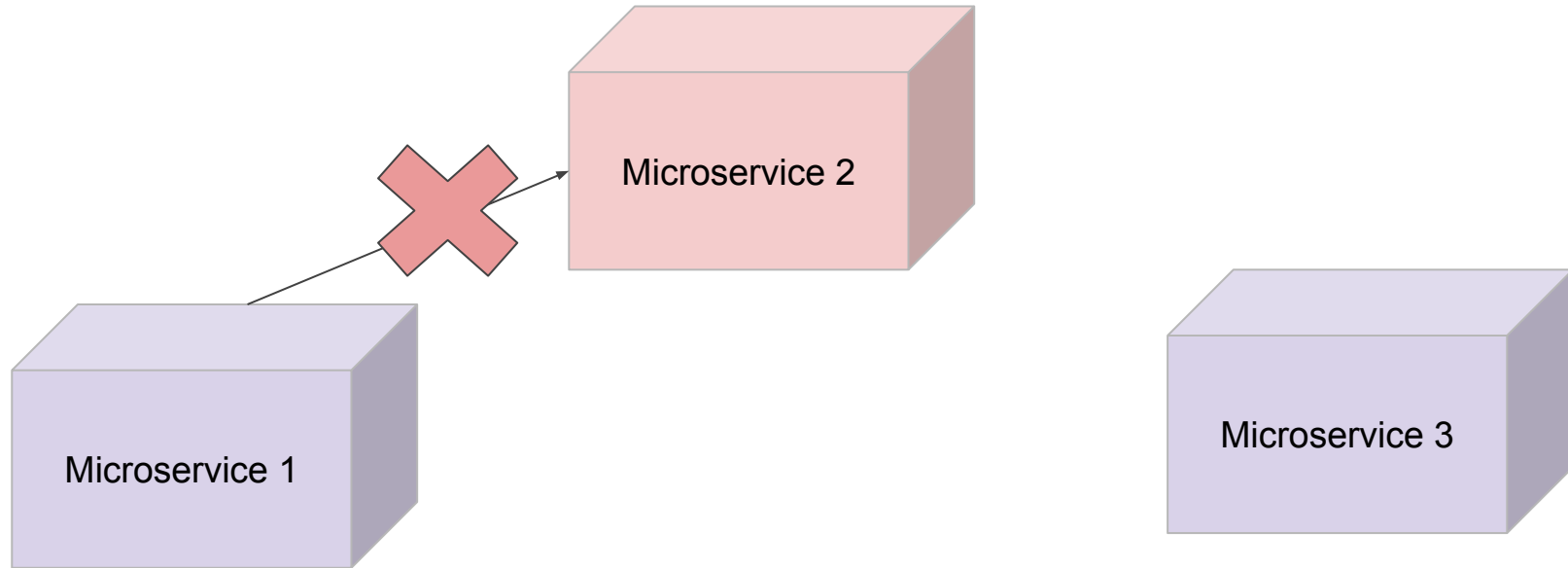
*Ability to continue operating without interruption*



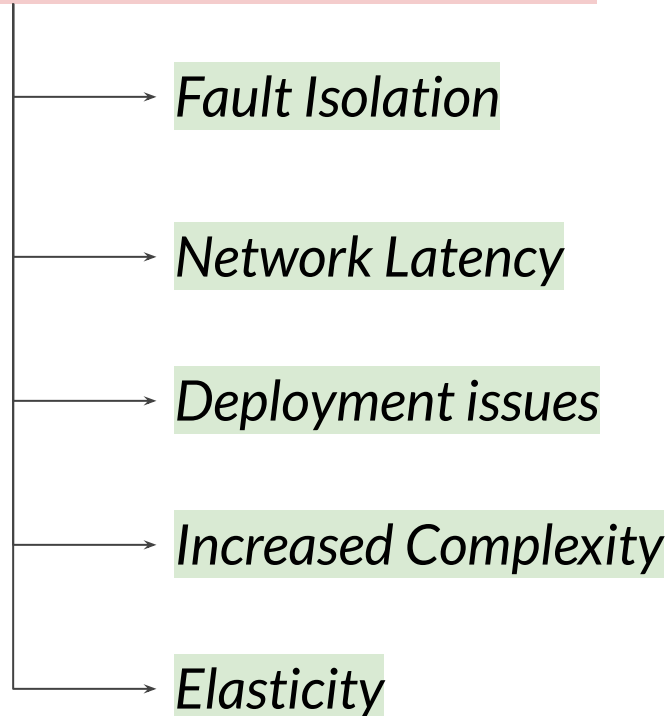








# *Need for Fault Tolerance*



# ***Need for Fault Tolerance***



*Tolerating External Failures*

# Introduction to Resilience4J

Faisal Memon

# **Resilience**

*Ability or capacity to recover quickly from difficulties*



# Techniques



Retries

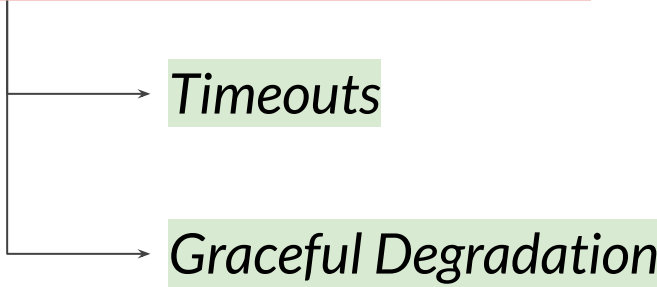
Rate Limiting

Bulkheads

Circuit Breakers

Fallbacks

# *Techniques*



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

*Timeouts*

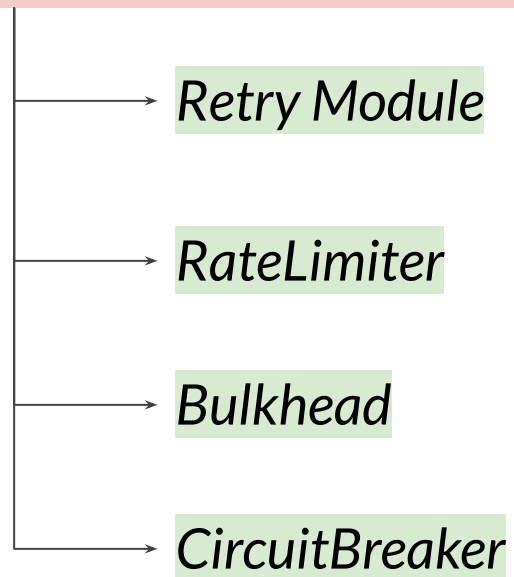
*Graceful Degradation*

***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*

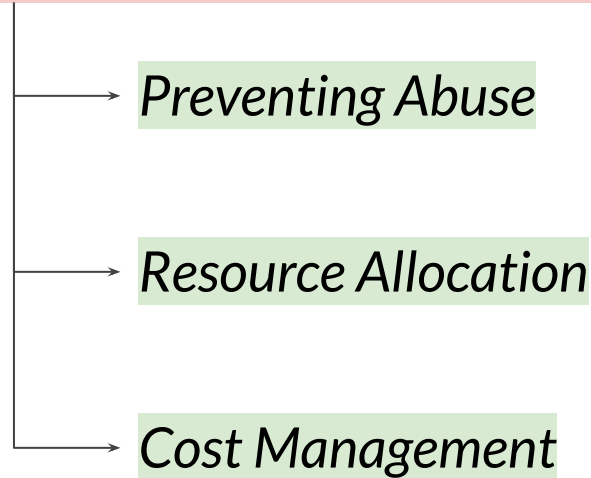
# What Is Rate Limiting and Why Is It Needed?

Faisal Memon (EmbarkX)

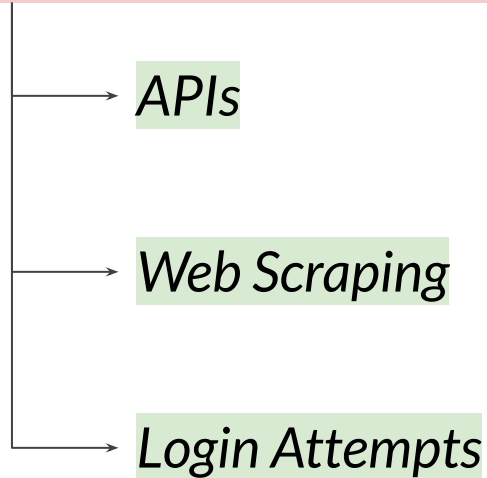
# **Rate Limiting**

*Rate limiting is a technique for limiting network traffic.*

# ***Importance of Rate Limiting***



# *Use Cases of Rate Limiting*



# Distributed Denial of Service (DDoS) attacks

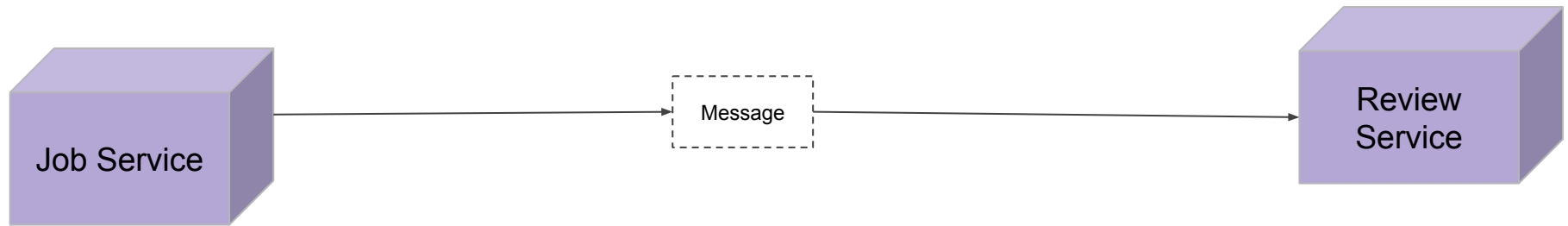
# Rate Limiting with Resilience4J in Spring Boot

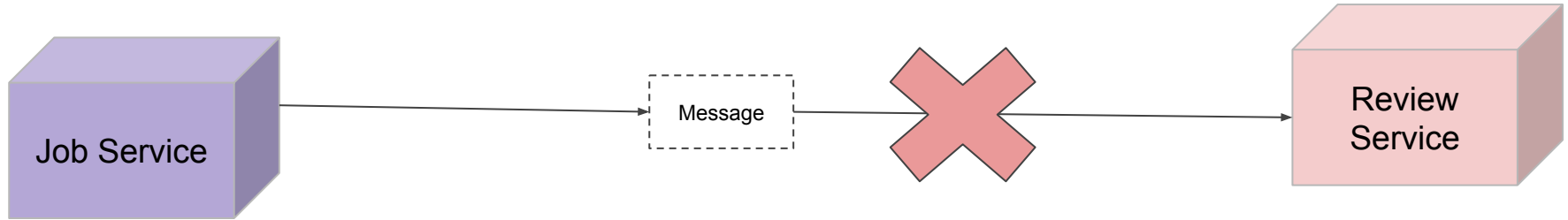
Thank you



# What are Message Queues?

Faisal Memon (EmbarkX)



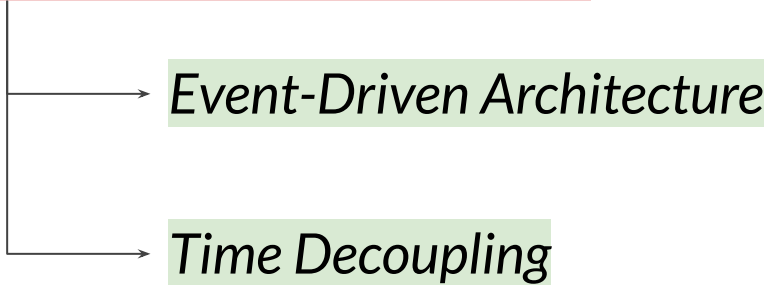




# *Need for Message Queues*



# *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

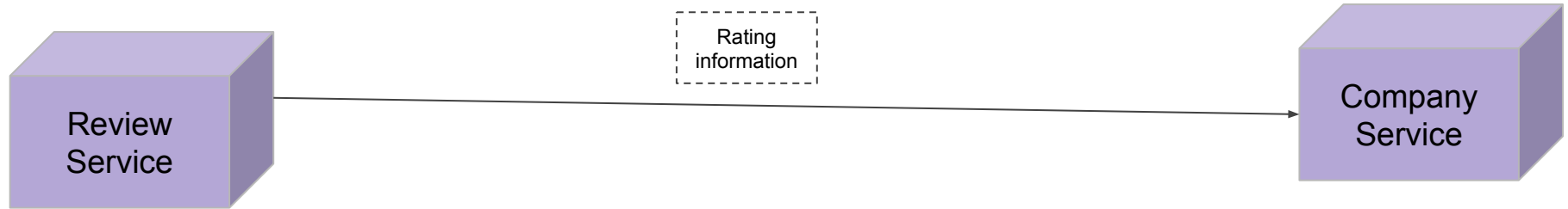
Faisal Memon (EmbarkX)



Thank you

# What are we going to build?

Faisal Memon (EmbarkX)





# Publishing Messages to RabbitMQ

Faisal Memon (EmbarkX)

# **Publishing Messages to RabbitMQ**

- *Define RabbitMQConfiguration Class*
- *Create a DTO*
- *Create a Producer to Send Messages (In Review)*
- *Send message once a Review is created*



# Consuming Messages from RabbitMQ

Faisal Memon (EmbarkX)

# Consuming Messages from RabbitMQ

- *Define RabbitMQConfiguration Class*
- *Create a DTO*
- *Create a Consumer to Receive Messages*
- *Update the ratings against a Company*

# Steps to Package Microservices into JARs

Faisal Memon (EmbarkX)

## Commands

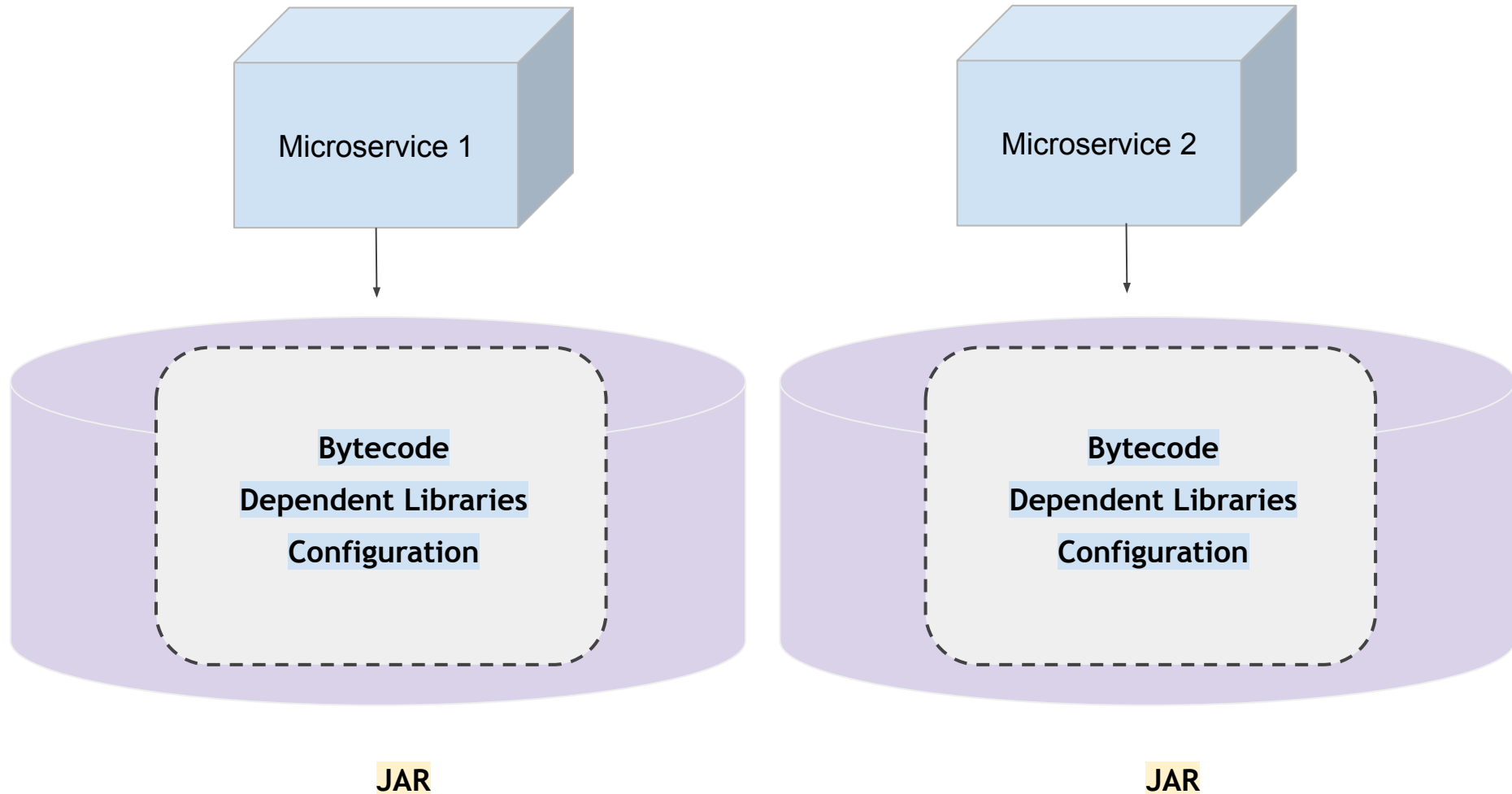
→ *mvn clean*

→ *mvn package*

→ *mvn clean package*

# Advanced Topics in Packaging

Faisal Memon (EmbarkX)



## ***Fat Jars***

*A 'fat' JAR is a JAR that contains not only the compiled bytecode of our application, but also all of its dependencies*

## ***Skinny Jars***

*A 'skinny' JAR is a JAR that contains only the compiled bytecode of our application*

# **Addressing Common Issues**

- *Conflicting versions of the same library*
- *Failure of the application to find its main class*



# Best Practices for Packaging Microservices

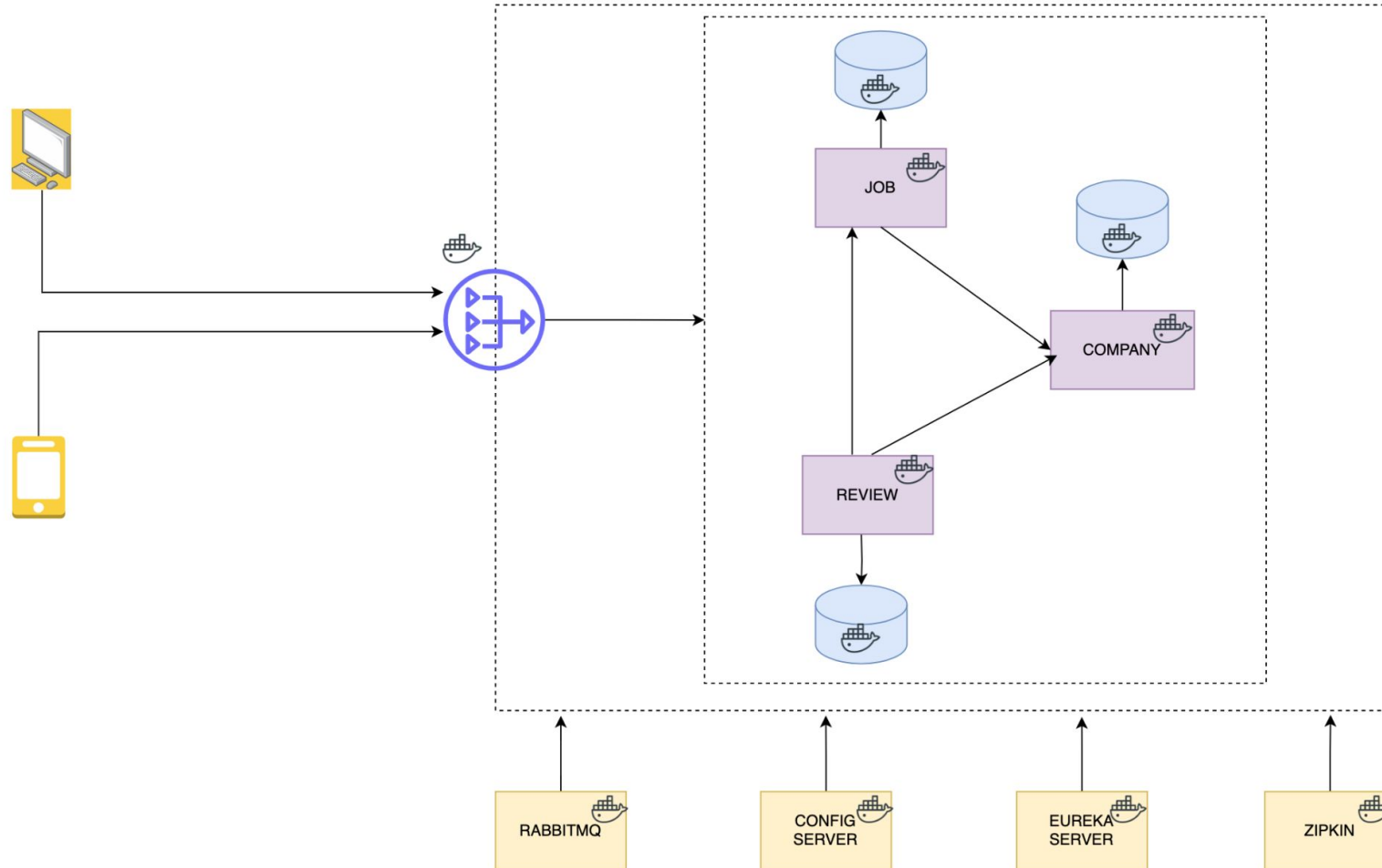
Faisal Memon (EmbarkX)

# **Best Practices**

- *Importance of Keeping Services Independent*
- *Keeping Track of Versions*
- *Continuous Integration/Continuous Delivery (CI/CD)*

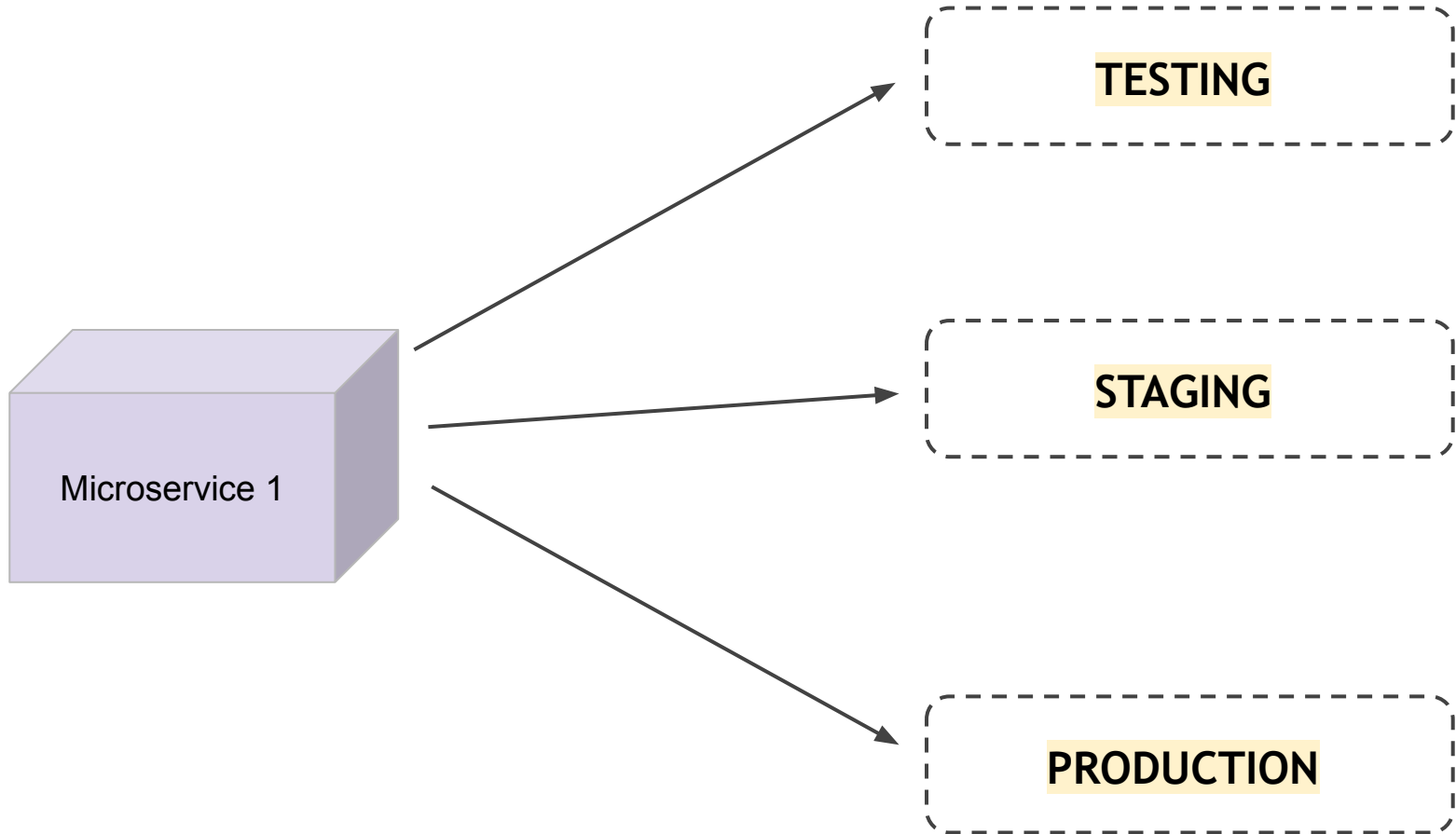
# Containerizing Spring Boot Microservices

Faisal Memon (EmbarkX)



# Introduction to Spring Boot Profiles

Faisal Memon (EmbarkX)



# What is Kubernetes?

Faisal Memon (EmbarkX)

## **Raise your hand!**

If you've ever been scared by  
the number of containers you  
had to manage in Docker



The story of Kubernetes begins  
with **Google**

Google built a system called  
**'Borg'**

‘Borg’ was open sourced and  
called **Kubernetes** or **k8s**

# Kubernetes

→ *Kubernetes is essentially a platform designed to completely manage the life cycle of containerized applications using methods that provide predictability, scalability, and high availability*

→ *You can define how your applications should run and the ways they should be able to interact with other applications or the outside world*

# Benefits of Kubernetes

Faisal Memon (EmbarkX)

# **Benefits**

- *Service Discovery and Load Balancing*
- *Automated Rollouts and Rollbacks*
- *Horizontal Scaling*

# **Benefits**

→ *Self-Healing*

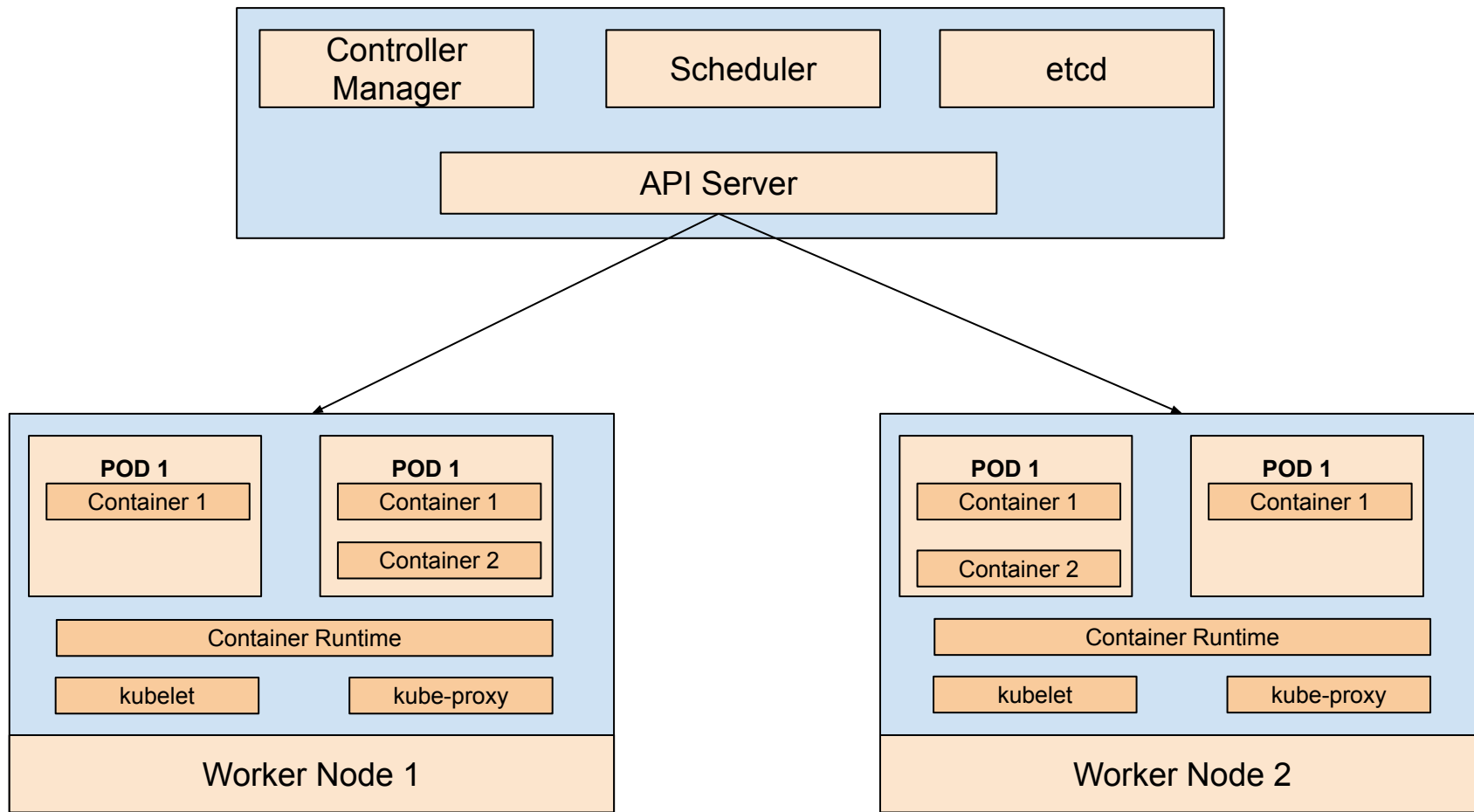
→ *Secret and Configuration Management*

# Kubernetes Architecture

Faisal Memon (EmbarkX)



## CONTROL PANE



# Pods

Faisal Memon (EmbarkX)

# What is a Pod

- A pod groups one or more containers and their shared resources, such as volumes (storage), IP address, and network ports.
- Containers within a pod run on the same worker node and share the same lifecycle.
- Pods are ephemeral and can be created, scheduled, and destroyed dynamically

## **Pod is not...**

→ *A pod is not a durable entity*

→ *Pods are not designed for horizontal scaling on their own*

# Key considerations

- *Designed to be stateless*
- *Communicate with each other within the same cluster using localhost*
- *Pods are assigned a unique IP address within the cluster*

## **Key considerations**

- *Lifecycle and availability of pods are managed by Kubernetes*
- *Pods can have associated labels and annotations*

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
  - name: redis
    image: redis:6.2.5
```



# Service

Faisal Memon (EmbarkX)

A Service in Kubernetes is an abstraction that defines a logical set of Pods and a policy by which to access them, sometimes called a **micro-service**

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

*Pods in Kubernetes are ephemeral, they can be created and destroyed*

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

# Exposing your application

Faisal Memon (EmbarkX)

# *Types of Service*

→ *ClusterIP*

→ *NodePort*

→ *LoadBalancer*

# ReplicaSets

Faisal Memon (EmbarkX)

A ReplicaSet is a Kubernetes object used for managing and scaling a set of identical pod replicas.



# **Why do you need identical Pods?**

- *High Availability*
- *Load Balancing*
- *Scaling*
- *Rolling Updates*
- *Service Discovery and Load Balancing*

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

*Pods in Kubernetes are ephemeral, they can be created and destroyed*

# **What a ReplicaSet is not**

→ *A ReplicaSet is not designed to handle rolling updates or deployments*

→ *It does not provide declarative updates to the pods it manages*

## *What to keep in mind*

- *ReplicaSets use a selector to identify the pods it manages*
- *You specify the desired number of replicas*
- *If a pod managed by a ReplicaSet fails or gets deleted, the ReplicaSet replaces it automatically to maintain the desired replica count*

## *What to keep in mind*

→ *ReplicaSets are often used together with Deployments*

→ *When updating the configuration of the pods, such as image versions or resource requirements, it's recommended to use Deployments*

# We don't need API Gateway / Eureka Server anymore?

Faisal Memon (EmbarkX)

Do we need API Gateway and  
Eureka Server?

# Eureka (Service Discovery)



Thank you