Palash Bajpai

**Spring Boot**

Table of Contents

[**1.** **What is Spring framework?** 2](#_Toc189042511)

[1.1 Inversion of Control(IoC) 3](#_Toc189042512)

[1.2 Dependency Injection 4](#_Toc189042513)

[1.3 Types of Dependency Injection 7](#_Toc189042514)

[1.4 Types of Annotations 11](#_Toc189042515)

[**2.** **What is Spring Boot?** 13](#_Toc189042516)

[2.1. Key Features of Spring boot 13](#_Toc189042517)

[How is Spring Boot Different from Spring? 14](#_Toc189042518)

[2.2. Configuration Properties: 16](#_Toc189042519)

[2.2. Embedded Servers 17](#_Toc189042520)

[2.3 Actuators 18](#_Toc189042521)

[**3. Web Service** 19](#_Toc189042522)

[3.1 Types of web service 20](#_Toc189042523)

[**4. REST API with Spring Boot** 23](#_Toc189042524)

[4.1. Security 23](#_Toc189042525)

[**5. Intro to Microservice** 24](#_Toc189042526)

[5.1. Monolithic Architecture 25](#_Toc189042527)

[5.2. How do Microservices communicate? 26](#_Toc189042528)

[**6. Interview Questions** 29](#_Toc189042529)

[Why Use Spring Boot? 29](#_Toc189042530)

[Difference Between @Controller and @RestController 29](#_Toc189042531)

[Examples 30](#_Toc189042532)

# **1. Intro to Microservice**

Microservices, also known as the Microservices architecture, is a software development approach where a large application is divided into smaller, loosely coupled, and independently deployable services. Each service performs a specific function and communicates with other services using lightweight protocols.

**Core Concepts:**

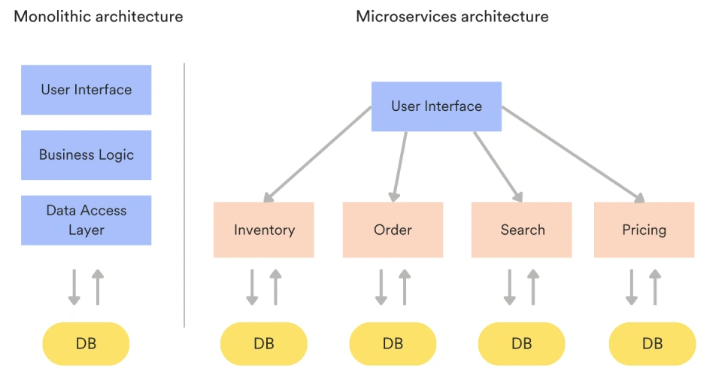
* **Decentralization:** Microservices decentralize development, allowing teams to work independently on different services.
* **Service Ownership:** Each microservice is owned by a specific team, ensuring accountability and focus.
* **Technology Agnostic:** Each service can use different technologies, programming languages, or databases.
* **Single Responsibility:** Each service is responsible for a specific task or business domain.

**Key Features:** Independent deployment and testing, Scalability (horizontal scaling), Modularity, Loose Coupling, Fault Isolation, and Technology Diversity and easier integration with modern technologies (DevOps, cloud-native).

**Disadvantages:** Increased complexity and delay in inter-communication, Dependency on robust monitoring and logging tools.

**Real world example:**

E-commerce platform: Amazon uses microservices to manage its vast ecosystem of functionalities like product search, recommendations, user authentication, and payments.

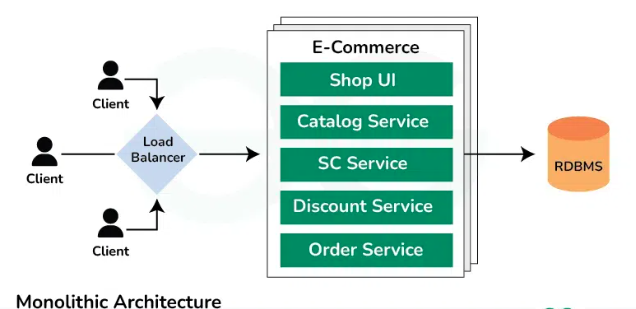


## 5.1. Monolithic Architecture

Monolithic architecture is a traditional software development approach where all components of an application—such as the user interface, business logic, and data layer—are tightly integrated into a single, unified codebase.

**Core Concepts:**

* **Unified Codebase:** All modules (e.g., authentication, payment, reporting) are part of one application.
* **Single Deployment Unit:** The entire application is deployed as a single unit on servers.
* **Centralized Data Management**: Typically uses a single database for all features and services.
* **Tightly Coupled Components:** Changes in one module often affect others, requiring comprehensive testing.



**Advantages: -** Faster(inter module communication not required) , Easier Deployment lifecycle), ideal for smaller project or project which don’t need high scalability.

**Disadvantages: -** Scalability challenges, harder to debug, difficulty in maintenance and updates, issue in one place bring down entire system, lack of flexibility for modern needs.

To overcome all this issues, we use microservice architecture.

## 5.2. How do Microservices communicate?

**Endpoint:**

* An endpoint is a URL or address where a microservice exposes its functionality.
* Example: https://api.example.com/orders could be an endpoint for retrieving order details.

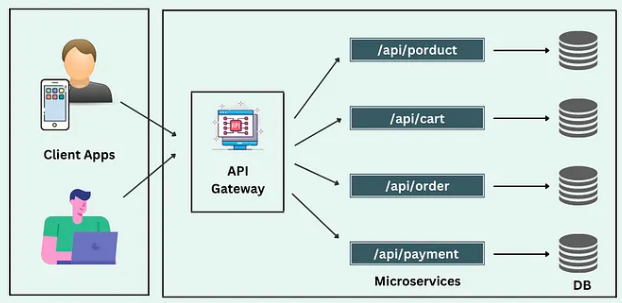
**API Gateways:**

An API gateway is an API management tool that sits between a client and a collection of backend services. An API gateway acts as a reverse proxy to accept all application programming interface (API) calls, aggregate the various services required to fulfill them, and return the appropriate result.

* An API Gateway acts as a single entry point for client requests to multiple services.
* Functions of an API Gateway:

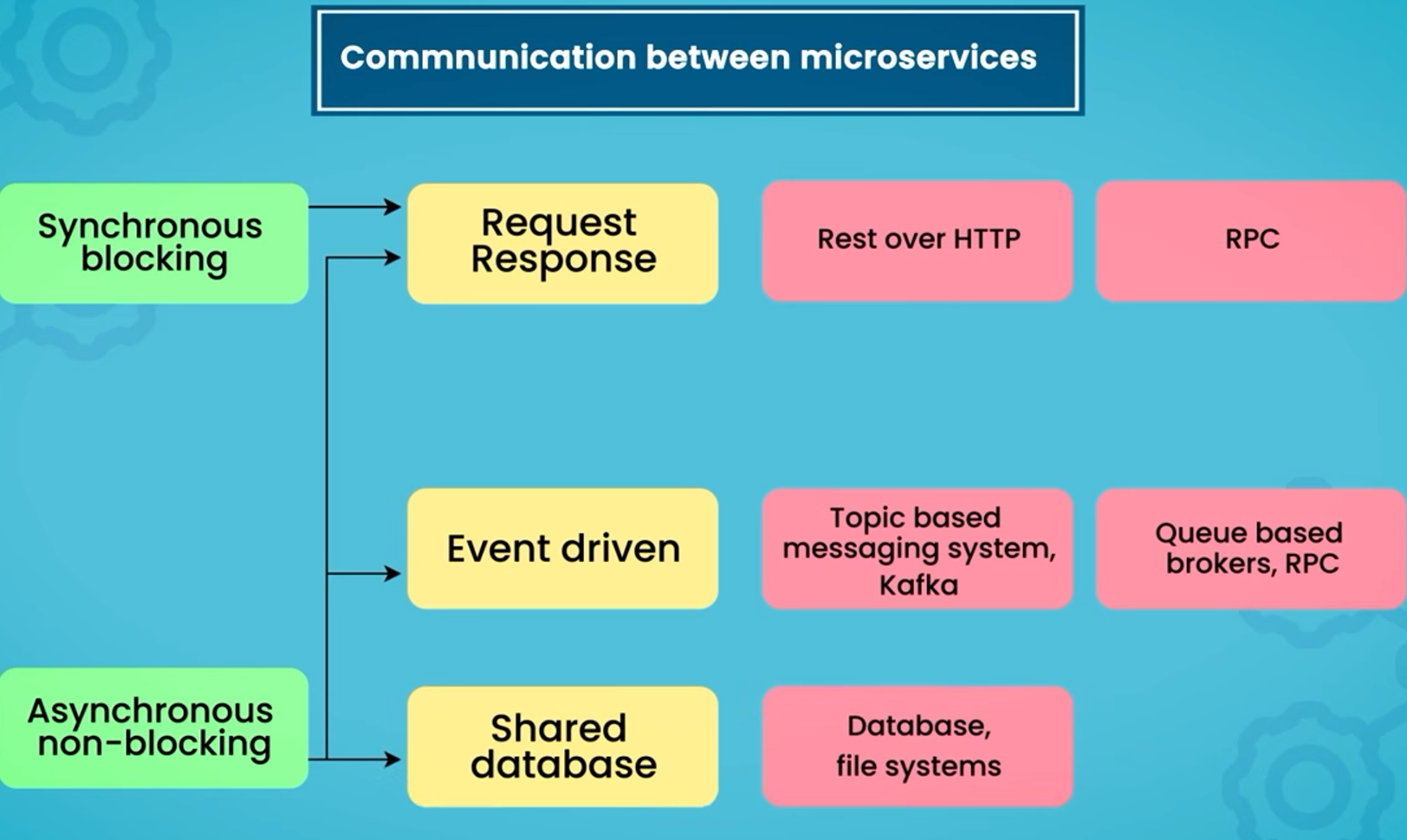
1. **Routing Requests:** Directs requests to the appropriate service.
2. **Authentication:** Ensures only authorized users can access the services.
3. **Monitoring and Logging:** Tracks request data, performance, and errors.

* Example: A user logs into a mobile app, and the API Gateway routes the request to the authentication service**.**



Communication between Microservices

Microservices need to communicate with each other to work as a cohesive system. This communication can be synchronous or asynchronous.



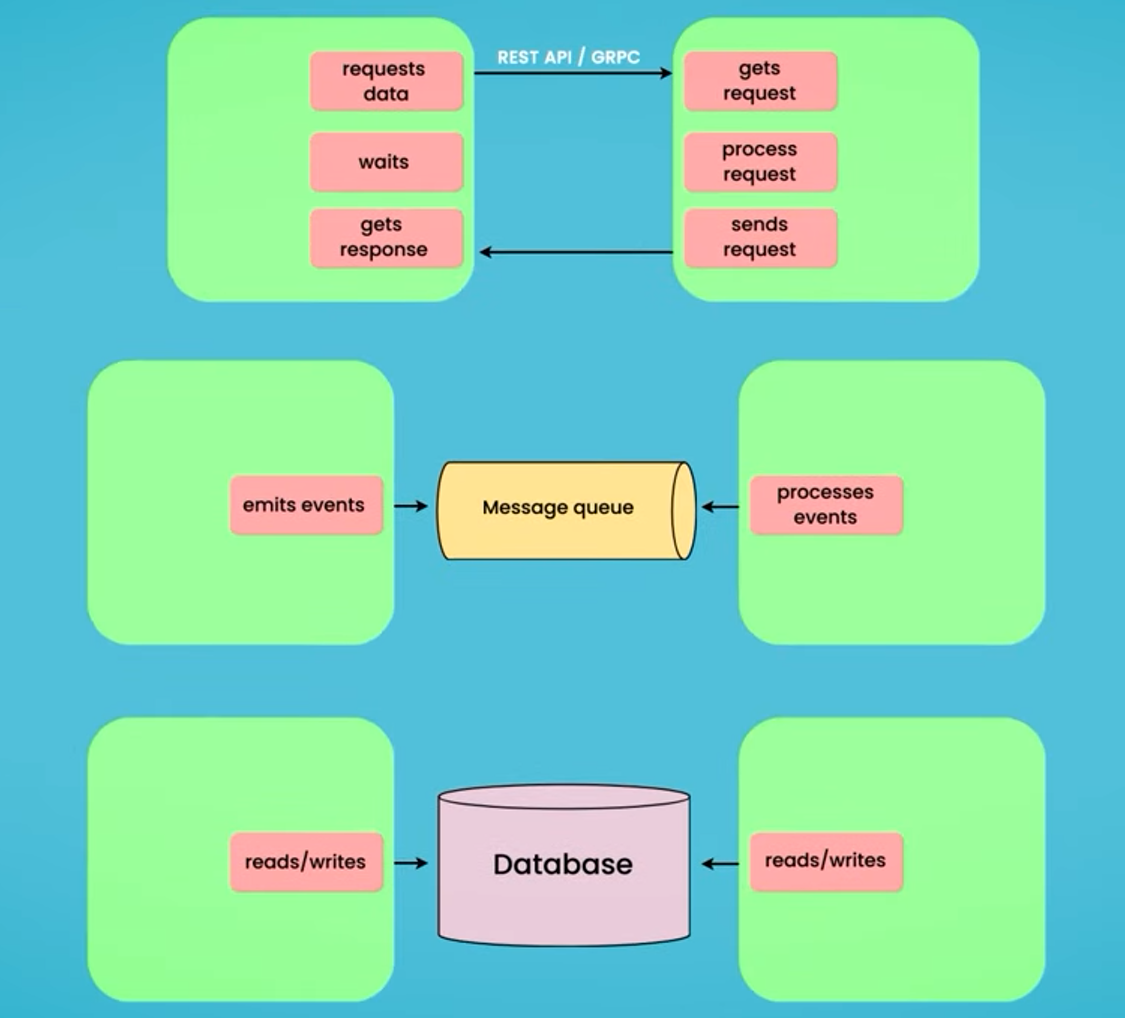
a) Synchronous Communication

**REST (Representational State Transfer):**

* A widely-used protocol that allows services to communicate over HTTP.
* Simple and stateless; uses methods like GET, POST, PUT, DELETE.
* Example: A payment service requests order details from an order service via a REST API.

**gRPC (Google Remote Procedure Call):**

* A high-performance framework that uses Protocol Buffers for communication.
* Faster and more efficient than REST, especially for internal service communication.
* Example: A billing service calls the inventory service using gRPC.



b) Asynchronous Communication

**Message Queues (e.g., RabbitMQ, ActiveMQ):**

* Messages are sent to a queue, and consumers process them at their own pace.
* Example: An order service places a message in the "shipping queue" for the shipping service to process later.

**Kafka (Apache Kafka):**

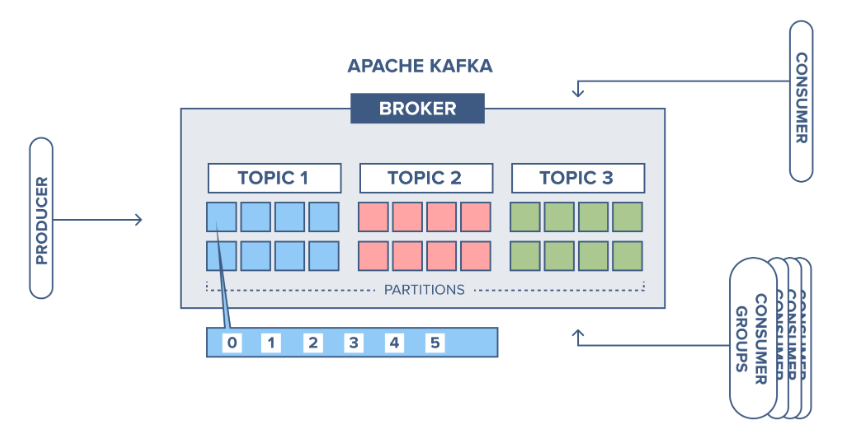
A distributed event-streaming platform designed for handling large volumes of data. Used for real-time event processing, logging, or communication between services.

**Basics of Kafka and Message Brokers:**

* A **message broker** like Kafka handles the transfer of messages between services.
* Kafka allows services to publish events to "topics," which other services can subscribe to and consume.
* Example: When a user places an order, the order service publishes an "Order Created" event to Kafka. The inventory and shipping services subscribe to this event to update stock and schedule shipping.

**Event Publishing and Subscription Patterns:**

* **Publishing:** A service creates and sends an event (e.g., "User Registered").
* **Subscription:** Other services listen for and react to those events (e.g., the notification service sends a welcome email).



# **2. Start microservices**

## 5.2. Cloud Native and 12-Factor app

**Endpoint:**

* An endpoint is a URL or address where a microservice exposes its functionality.
* Example: https://api.example.com/orders could be an endpoint for retrieving order details.

## 2.2. Quiz app

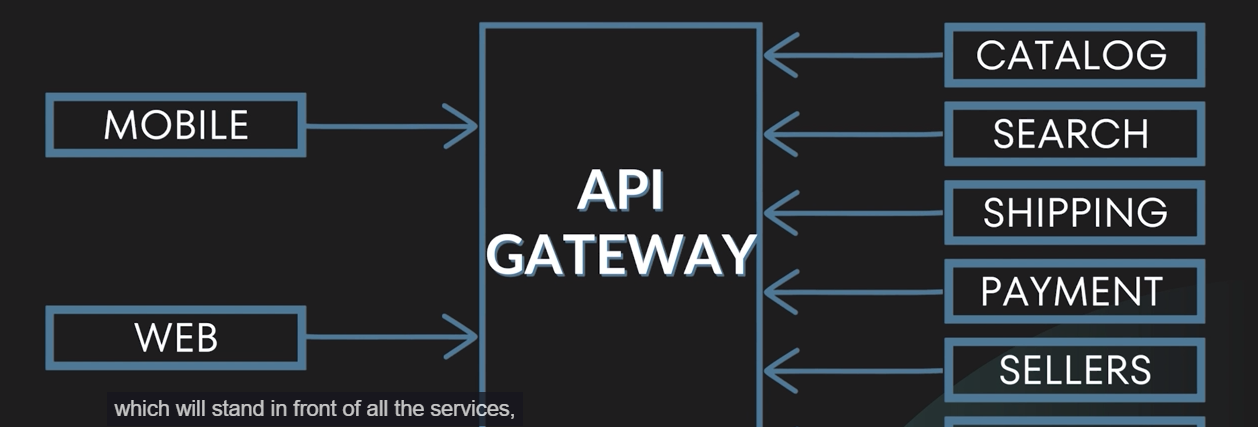
2.2. Service Registry

Using Rest template and Feign client

2.2. Load Balancing

Env variable se print port number, run in 2 different ports to verify1

2.2. API Gateway



2.2. Circuit Breaker

# **6. Interview Questions**

**3. How Web Services Solve the Issue of Sharing Code?**

Problems with Sharing Code Directly:

* **Compatibility Issues:** Sharing a JAR file restricts usage to specific programming languages and platforms.
* **Distribution Challenges:** Every time the code changes, the updated JAR must be redistributed.
* **Versioning Problems:** Managing multiple versions of the same library is complex.
* **Limited Accessibility:** Sharing a JAR file does not support real-time or remote access.

Solution via Web Services:

* **Real-Time Access:** The client consumes the service in real-time, ensuring the latest version of the code is used.
* **Language Agnosticism:** Web services provide data in standardized formats like XML or JSON, enabling usage across languages.
* **Centralized Updates:** Updates to the service are made on the server, automatically reflecting for all clients.
* **Ease of Integration:** With standard protocols like HTTP and REST, integration with web services is straightforward.

POJO vs Bean