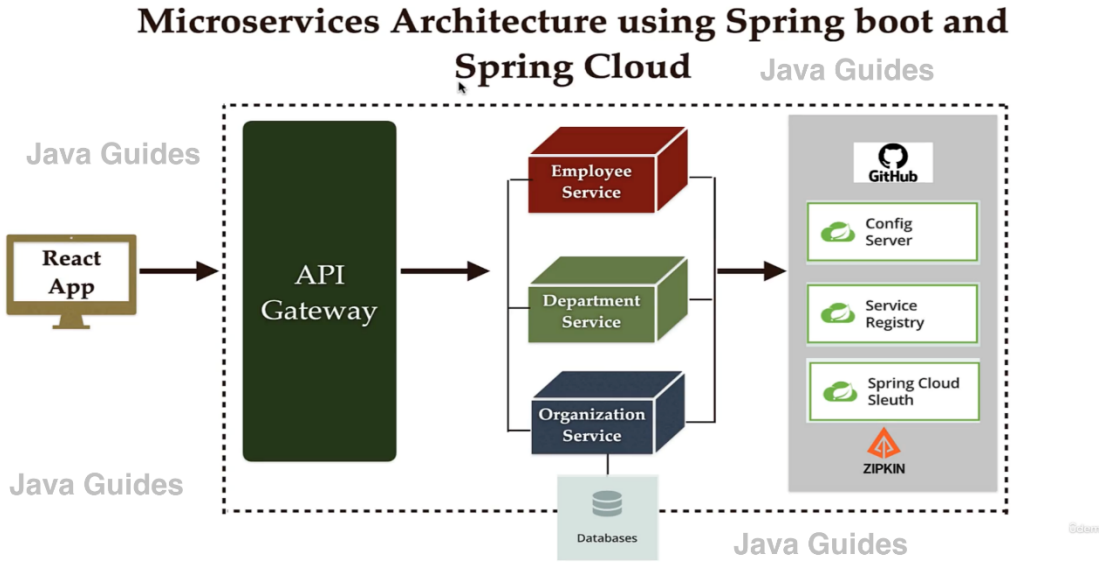
Spring Boot Microservices Architecture

Microservices Architecture using Spring Boot and Spring Cloud



Well, you can take any example of any project. For example, you can take an e-commerce application or you can take any healthcare domain-related application. So you can take any project as an example, but follow the same steps to create a microservice architecture using Spring Boot and Spring Cloud.

Core Microservices

Consider we have developed three core backend Spring boot microservices such as **employee service**, **department service,** and **organization service,** and all these three microservices have their own databases. You can use a relational database or NoSQL database as a database for these microservices. So whenever you create a microservice in your project, make sure that each microservice should have its own database. All right.

Microservices Communication

Once we build these 3 microservices. Next, we'll see how these microservices communicate with each other. Well, there are different ways to make a REST API call

from one microservice to another Microservice. For example, we can use a RestTemplate or WebClient or Spring cloud-provided open feign library. All right.

Well, there are two types of communication styles. One is synchronous and another is asynchronous.

In the case of synchronous, we can use the HTTP protocol to make an HTTP request from one microservice to the microservice.

And in the case of asynchronous communication, we have to use a message broker for asynchronous communication between multiple microservices. For example, we can use RabbitMQ or Apache Kafka as a message broker in order to make an asynchronous communication between multiple microservices and each microservice in a microservices project can expose REST APIs.

Registry and Discovery Pattern

Well, once we know how microservices communicate with each other, next you need to know how to implement a service Registry and discovery pattern in our microservices project.

Well, Spring Cloud provides a **Spring Cloud Netflix Eureka Based Service Registry** module that we can use to implement service registry and discovery patterns in our microservices project. Well, service Registry and discovery is a really essential pattern that we can use to avoid hard coding hostnames and ports.

Config-Server to Externalize the Configurations

Next, we will implement a config server to externalize the configurations of all these three microservices into a central place which is the git repository.

Well, Spring cloud provides a Spring cloud config module that we can use to implement a config server to externalize the configuration files of all these three microservices into a central place. We are going to use the git repository as storage for the config server.

API Gateway Pattern

Well, once we know how to use the config server to externalize the configuration files. Next, we have to implement an API gateway.

Well, API Gateway plays a very important role in our microservices architecture. So whenever a client wants to make a call to different microservices, the client has to remember the host names and ports of all these microservices. So there should be a solution where a client can send a request to the central component so that is where the API gateway comes into the picture. So whenever a client sends a request to the backend microservices, then the client has to send a request to the API gateway first, and then the API gateway based on the routing rules will route that request to the appropriate microservice. So this is how the API gateway plays an important role in a microservices architecture.

Well, Spring Cloud provides **Spring Cloud Gateway module** to implement API gateway patterns in a microservices architecture.

Distributed Tracing

Next, once you know how to implement an API gateway in a microservices project, next you can implement distributed tracing in a microservices architecture. Well, Spring Cloud provides a Spring Cloud sleuth module, which we can use to implement distributed tracing in our microservices project.

Well, along with Spring Cloud Sleuth, we'll also use Zipkin to visualize the tracing log information in a user interface. Well, Zipkin provides a user interface to track and trace information through web applications.

React Front Service

Next, you can use React/Angular to create a client-side service that will make a call to backend microservices.

Circuit Breaker Pattern

Next, you can implement a circuit breaker pattern in an employee service because the employee service is internally calling department service, and let's say due to some reason, department service is down then employee service won't get a response from the department server, isn't it? And then again, employee service will send an internal

server error to the API gateway and then API Gateway will send that response back to the client. All right. So in order to avoid this kind of issue, we can use a circuit breaker pattern.

So this circuit breaker pattern helps the employee service to avoid continuous calls to the department service Whenever department service is done and this circuit breaker pattern will help employee service to return some default response back to the API Gateway and the API Gateway will send that default response to the client.

All right. So this is a simple microservices architecture using Spring Boot and Spring Cloud.