Microservices Interview Questions and Answers

---

1. Microservices Design Patterns

- Service Registry & Discovery: Utilize Eureka for automatic service detection.
- API Gateway: Use Spring Cloud Gateway for routing.
- Circuit Breaker: Implement Resilience4j for fault tolerance.
- Event-Driven Architecture: Leverage Kafka or RabbitMQ for asynchronous communication.
- Database Per Microservice: Each microservice manages its own database.
- Sidecar Pattern: Attach additional functionalities (like logging and monitoring) to services.

---

2. Communication Between Microservices

a) Synchronous Communication

- REST APIs (HTTP) are used for direct calls.
- Example: Using RestTemplate to make synchronous calls.

java

Copy

```
@RestController
@RequestMapping("/orders")
public class OrderController {
  private final RestTemplate restTemplate;

  public OrderController(RestTemplate restTemplate) {
    this.restTemplate = restTemplate;
  }
```

```
    @GetMapping("/{id}")
    public Order getOrder(@PathVariable Long id) {
        Customer customer = restTemplate.getForObject("http://CUSTOMER-SERVICE/customers/" + id, Customer.class);
        return new Order(id, customer);
    }
}
```

b) Asynchronous Communication

- Messaging queues like Kafka or RabbitMQ are used for asynchronous communication.
- Example: Sending a message using RabbitMQ.

```
java
Copy
@Service
public class OrderService {
    @Autowired private RabbitTemplate rabbitTemplate;

    public void placeOrder(Order order) {
        rabbitTemplate.convertAndSend("orderQueue", order);
    }
}
```

---

3. Handling Failure in Microservices

- Circuit Breaker Pattern: Prevents system failure by blocking requests when a service is down.
- Retry Mechanism: Retries failed requests.
- Fallback Mechanism: Provides an alternative response when the service fails.

- Monitoring & Logging: Use ELK Stack or Prometheus & Grafana for monitoring and logging.

Circuit Breaker Example (Resilience4j)

java

Copy

```
@Retry(name = "orderService", fallbackMethod = "fallbackMethod")
public String callOrderService() {
   return restTemplate.getForObject("http://ORDER-SERVICE/orders", String.class);
}


public String fallbackMethod(Exception e) {
   return "Order service is currently unavailable. Please try later.";
}
```

---

4. Load Balancing in Microservices

Client-Side Load Balancing (Ribbon)

java

Copy

```
@LoadBalanced
@Bean
public RestTemplate restTemplate() {
   return new RestTemplate();
}
```

Server-Side Load Balancing (Spring Cloud Gateway)

yaml

Copy

```
spring:
  cloud:
    gateway:
      routes:
        - id: order-service
          uri: lb://ORDER-SERVICE
          predicates:
            - Path=/orders/**
```

---

5. Service Discovery Using Eureka

Eureka Server Configuration

java

Copy

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(EurekaServerApplication.class, args);
    }
}
```

Eureka Client Configuration

yaml

Copy

```
spring:
  application:
    name: order-service
```

```
eureka:
 client:
  service-url:
   defaultZone: http://localhost:8761/eureka/
```

---

6. Centralized Configuration Using Spring Cloud Config & Git

Config Server

java

Copy

```
@SpringBootApplication
@EnableConfigServer
public class ConfigServerApplication {
  public static void main(String[] args) {
    SpringApplication.run(ConfigServerApplication.class, args);
  }
}
```

Application Properties for Config Server

yaml

Copy

```
spring:
 cloud:
  config:
   server:
    git:
     uri: https://github.com/your-repo/config-repo
```

Client Application Configuration

yaml

Copy

```
spring:
 application:
  name: order-service
 config:
  import: optional:configserver:http://localhost:8888
```

Step-by-Step Guide to Building Microservices

Step 1: Create a Spring Boot Application

1. Create a Spring Boot project using Spring Initializr.
2. Add dependencies: Spring Web, Spring Boot Actuator, Spring Cloud Dependencies.

Step 2: Implement Microservices

1. Create independent services (Order, Payment, Customer).
2. Define REST endpoints for each service.

Step 3: Set Up Eureka for Service Discovery

1. Create Eureka Server.
2. Register microservices as Eureka Clients.

Step 4: Implement API Gateway

1. Use Spring Cloud Gateway for routing between services.

Step 5: Implement Circuit Breaker and Resilience4j

1. Configure Resilience4j for handling service failures and retries.

Step 6: Implement Centralized Configuration

1. Use Spring Cloud Config with a GitHub repository to centralize configurations.

Step 7: Implement Load Balancing

1. Use Ribbon for client-side load balancing or Spring Cloud Gateway for server-side load balancing.

Step 8: Implement Communication Mechanism

1. Use REST for synchronous communication between services.
2. Use Kafka or RabbitMQ for asynchronous messaging between services.

Step 9: Deploy Using Docker and Kubernetes

1. Create Docker images for each microservice.
2. Deploy the services in Kubernetes clusters for scalability and management.

Microservices Architecture Flow Diagram

lua

Copy