1. What is Spring Cloud and why do we use it in microservices?

Answer:
Spring Cloud is a set of tools that help developers build and manage microservices by handling common challenges like service discovery, configuration management, fault tolerance, etc.

Real-time example:
Imagine you're building an **online food delivery system** with services like RestaurantService, OrderService, and DeliveryService. Spring Cloud helps:

- Register services with **Eureka Server** so they can discover each other.
- Manage configurations using **Spring Cloud Config Server**.
- Route requests via **API Gateway**.
- Implement circuit breakers using **Resilience4j** to handle failures.

Without Spring Cloud, you'd have to build all these manually.

2. How does Eureka Server help in a microservices architecture?

Answer:
Eureka is a **service registry**. Each microservice registers itself with Eureka so other services can discover and call it without hardcoding IPs or ports.

Real-time example:
Let's say the OrderService wants to call RestaurantService. Instead of calling http://localhost:8081, it just uses http://RESTAURANT-SERVICE. Eureka maps this to the actual running instance—even if it restarts or changes port.

3. What is an API Gateway and how is it used in Spring Cloud?

Answer:
An API Gateway is a single entry point for all client requests. It routes, filters, and handles cross-cutting concerns like security, logging, and rate-limiting.

Real-time example:
In your food app, instead of exposing all microservices to the frontend (OrderService, RestaurantService, DeliveryService), expose only **Spring Cloud Gateway**. It routes requests like:

- /orders/** → OrderService
- /restaurants/** → RestaurantService

You can also configure login checks here before routing.

4. How can you manage configuration across microservices in Spring Cloud?

Answer:
Using **Spring Cloud Config Server**, you centralize configuration in one place (like Git). Each service fetches its config during startup.

Real-time example:
You store all config files in a Git repo:

- order-service.properties
- restaurant-service.properties

When the OrderService starts, it fetches values like DB URLs, ports, etc., from the config server, not from local files. This ensures all configs are maintained centrally and version-controlled.

5. What is Resilience4j and how do you use circuit breakers in Spring Cloud?

Answer:
Resilience4j provides fault tolerance by using **circuit breakers**, retries, and rate-limiters. A **circuit breaker** stops calling a failed service temporarily and uses a fallback.

Real-time example:
If DeliveryService is down, OrderService uses a fallback method to return a message: *"Delivery info is currently unavailable. Please check back later."*

This avoids crashing the whole system due to one failed service.

6. What is Feign Client and how does it simplify inter-service communication?

Answer:
Feign is a declarative HTTP client. You just define an interface, and Spring generates the REST call implementation.

Real-time example:
In OrderService, to call RestaurantService:

java

CopyEdit

@FeignClient(name = "restaurant-service")

public interface RestaurantClient {

```
    @GetMapping("/restaurants/{id}")
    Restaurant getRestaurant(@PathVariable Long id);
}
```

No need to write RestTemplate boilerplate—this interface does it all.

7. What is the use of Spring Cloud Config Client and how does it work?

Answer:
The **Config Client** pulls configurations from the **Config Server** at startup or via refresh.

Real-time example:
Imagine you need to change the database password for PaymentService. Instead of opening the app and modifying application.properties, you update it in Git. On the next startup (or by calling /actuator/refresh), the Config Client fetches the updated password automatically.

8. What is Load Balancing in Spring Cloud and how is it done?

Answer:
Spring Cloud uses **Ribbon** or **Spring Cloud LoadBalancer** to distribute requests across multiple instances of a service.

Real-time example:
You have three instances of InventoryService. When OrderService makes requests, they're distributed across all three for performance and fault tolerance.

9. How can you refresh configuration without restarting the microservice?

Answer:
Use **Spring Actuator** with the /actuator/refresh endpoint, along with @RefreshScope.

Real-time example:
Your support team changes the max order limit in the config repo. Instead of restarting OrderService, you just call the /actuator/refresh endpoint and the change is reflected live.

10. What is Zipkin and Sleuth in Spring Cloud?

Answer:

- **Sleuth** adds trace and span IDs to logs.

- **Zipkin** visualizes the path of a request through microservices.

Real-time example:
A user complains that placing an order is slow. You check **Zipkin UI**, trace the flow (Gateway → OrderService → PaymentService) and find delay in PaymentService. Sleuth logs help debug which part is slow.

11. What are common patterns supported by Spring Cloud?

Answer:

- **Service Discovery**
- **Config Management**
- **API Gateway**
- **Circuit Breaker**
- **Distributed Tracing**
- **Load Balancing**

Real-time example:
In a movie booking platform, Spring Cloud allows you to build a resilient architecture where services like MovieService, UserService, and BookingService work seamlessly across cloud environments.

12. What happens if a microservice is down and another service calls it?

Answer:
Without protection, it causes failures or hangs. With **Resilience4j Circuit Breaker**, it switches to fallback logic.

Real-time example:
If MovieDetailsService is down, the BookingService can return: *"Movie details currently not available. Please try later."*—instead of crashing.