Spring Cloud Interview Questions

In this post we will look at Spring Cloud questions. Examples are provided with explanation.

Q: What is Spring Cloud?
A: Spring Cloud Stream App Starters are Spring Boot based Spring Integration applications that provide integration with external systems. Spring Cloud Task. A short-lived microservices framework to quickly build applications that perform finite amounts of data processing.
Spring Cloud (/spring/springcloud)

Q: What are the advantages of using Spring Cloud?
A: When developing distributed microservices with Spring Boot we face the following issues-

- **Complexity associated with distributed systems-**
  This overhead includes network issues, Latency overhead, Bandwidth issues, security issues.
- **Service Discovery-**
  Service discovery tools manage how processes and services in a cluster can find and talk to one another. It involves a directory of services, registering services in that directory, and then being able to lookup and connect to services in that directory.
- **Redundancy-**
  Redundancy issues in distributed systems.
- **Loadbalancing-**
  Load balancing improves the distribution of workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units, or disk drives.
- **Performance issues-**
  Performance issues due to various operational overheads.
- **Deployment complexities-**
  Requirement of Devops skills.

Q: What does one mean by Service Registration and Discovery? How is it implemented in Spring Cloud?
A: When we start a project, we usally have all the configurations in the properties file. As more and more services are developed and deployed, adding and modifying these properties become more complex. Some services might go down, while some the location might change. This manual changing of properties may create issues.

Eureka Service Registration and Discovery helps in such scenarios. As all services are registered to the Eureka server and lookup done by calling the Eureka Server, any change in service locations need not be handled and is taken care of
Microservice Registration and Discovery with Spring cloud using Netflix Eureka.
(/spring/spring_eurekaregister)

Q: What does one mean by Load Balancing ? How is it implemented in Spring Cloud?
A: In computing, load balancing improves the distribution of workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units, or disk drives. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource. Using multiple components with load balancing instead of a single component may increase reliability and availability through redundancy. Load balancing usually involves dedicated software or hardware, such as a multilayer switch or a Domain Name System server process.
In SpringCloud this can be implemented using Netflix Ribbon.
Spring Cloud- Netflix Eureka + Ribbon Simple Example (/spring/spring_ribbon)
Q: What is Hystrix? How does it implement Fault Tolerance?
A: In computing, load balancing improves the distribution of workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units, or disk drives. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource. Using multiple components with load balancing instead of a single component may increase reliability and availability through redundancy. Load balancing usually involves dedicated software or hardware, such as a multilayer switch or a Domain Name System server process.
In SpringCloud this can be implemented using Netflix Ribbon.
Spring Cloud- Netflix Eureka + Ribbon Simple Example (/spring/spring_ribbon)

Q: What is Hystrix? How does it implement Fault Tolerance?
A: Hystrix is a latency and fault tolerance library designed to isolate points of access to remote systems, services and 3rd party libraries, stop cascading failure and enable resilience in complex distributed systems where failure is inevitable.
Usually for systems developed using Microservices architecture, there are many microservices involved. These microservices collaborate with each other.
Consider the following microservices-

Suppose if the microservice 9 in the above diagram failed, then using the traditional approach we will propagate an exception. But this will still cause the whole system to crash anyways.
This problem gets more complex as the number of microservices increase. The number of microservices can be as high as 1000. This is where hystrix comes into picture
We will be the Fallback method feature of Hystrix for this scenario. We have two services employee-consumer consuming the service exposed by the employee-producer.
The simplified diagram is as below-

Now suppose due to some reason the employee-producer exposed service throws an exception. In this case using Hystrix we define a fallback method. This fallback method should have the same return type as the exposed service. In case of exception in the exposed service the fallback method will return some value. Spring Cloud- Netflix Hystrix Fallback method Simple Example (/spring/spring_hystrix)

Q: What is Hystrix Circuit Breaker? Need for it?

A: Due to some reason the employee-producer exposed service throws an exception. In this case using Hystrix we defined a fallback method. In case of exception in the exposed service the fallback method returned some default value.

If the exceptions keep on occuring in the firstPage method() then the Hystrix circuit will break and the employee consumer will skip the firtsPage method all together and directly call the fallback method.
The purpose of circuit breaker is to give time to the first page method or other methods that the firstpage method might be calling and is causing the exception to recover. It might happen that on less load the issue causing the exceptions have better chance of recovering

Spring Cloud- Circuit Breaker using Netflix Hystrix Simple Example (/spring/spring_hystrix_circuitbreaker)

Q: What is Netflix Feign? What are its advantages?
A: Feign is a java to http client binder inspired by Retrofit, JAXRS-2.0, and WebSocket. Feign's first goal was reducing the complexity of binding Denominator uniformly to http apis regardless of restfulness. Previous examples in the employee-consumer we consumed the REST services exposed by the employee-producer using **REST Template**

But we had to write a lot of code to perform following-

- For Load balancing using Ribbon.
- Getting the Service instance and then the Base URL.
- Make use of the REST Template for consuming service.

The previous code was as below

```
 @Controller
public class ConsumerControllerClient {

        @Autowired
        private LoadBalancerClient loadBalancer;

        public void getEmployee() throws RestClientException, IOException {

                ServiceInstance serviceInstance=loadBalancer.choose("employee-p

                System.out.println(serviceInstance.getUri());

                String baseUrl=serviceInstance.getUri().toString();

                baseUrl=baseUrl+"/employee";

                RestTemplate restTemplate = new RestTemplate();
                ResponseEntity<String> response=null;
                try{
                response=restTemplate.exchange(baseUrl,
                                HttpMethod.GET, getHeaders(),String.class);
                }catch (Exception ex)
                {
                        System.out.println(ex);
                }
                System.out.println(response.getBody());
        }
```

The previous code, there are chances of exceptions like NullPointer and is not optimal. We will see how the call is made much easier and cleaner using Netflix Feign. If the Netflix Ribbon dependency is also in the classpath, then Feign also takes care of load balancing by default.
Spring Cloud- Netflix Feign Simple Example (/spring/spring-cloud-netflix-feign-tutorial)

Q: What is Spring Cloud Bus? Need for it?
A: Consider the scenario that we have multiple applications reading the properties using the Spring Cloud Config and the Spring Cloud Config in turn reads these properties from GIT.
Consider the below example where multiple employee producer modules are getting the property for Eureka Registration from Employee Config Module.

What will happen if suppose the eureka registration property in GIT changes to point to another Eureka server. In such a scenario we will have to restart the services to get the updated properties. There is another way of using Actuator Endpoint **/refresh**. But we will have to individually call this url for each of the modules. For example if Employee Producer1 is deployed on port 8080 then call

http://localhost:8080/refresh. Similarly for Employee Producer2 **http://localhost:8081/refresh** and so on. This is again cumbersome. This is where Spring Cloud Bus comes into picture.

The Spring Cloud Bus provides feature to refresh configurations across multiple instances. So in above example if we refresh for Employee Producer1, then it will automatically refresh for all other required modules. This is particularly useful if we are having multiple microservice up and running. This is achieved by connecting all microservices to a single message broker. Whenever an instance is refreshed, this event is subscribed to all the microservices listening to this broker and they also get refreshed. The refresh to any single instance can be made by using the endpoint **/bus/refresh**

Spring Cloud Tutorial - Publish Events Using Spring Cloud Bus (/spring/cloud-stream-bus)