Spring Cloud Microservices

1. Overview of microservices
2. Microservices application example
3. Implementing circuit breaker behaviour

1. Overview of Microservices

- The challenges
- What are microservices?
- Characteristics of a microservice architecture
- Benefits of the microservices approach
- Microservices and the cloud
- Microservices and Spring

The Challenges

- Globalization and interconnectivity place new demands on organizations and IT departments…
  - Applications need to communicate with many external service providers over the Internet - the age of silo applications is over
  - Customers expect incremental product updates and feature upgrades, rather than complete product releases once a year
  - Architectures must be flexible enough to scale out across multiple servers quickly when volume spikes
  - Availability and resilience in the worldwide market are essential

What are Microservices?

- According to Wiki:

Microservices is a specialisation of an implementation approach for service-oriented architectures (SOA) used to build flexible, independently deployable software systems.

Services in a **microservice architecture (MSA)** are processes that communicate with each other over a network in order to fulfil a goal. These services use technology-agnostic protocols.

The microservices approach is a first realisation of SOA that followed the introduction of DevOps and is becoming more popular for building continuously deployed systems.

Characteristics of a Microservice Architecture

- Microservices are a move away from monolithic architectures
  - Functionality is delivered as fine-grained distributed components

- Each microservice is **highly cohesive**
  - Has responsibility for a very specific piece of domain logic
  - Has well-defined boundaries
  - The implementation technology of a microservice is irrelevant

- Microservices are **loosely coupled**
  - Each microservice is deployed independently of other ones
  - Communicate via technology-neutral protocols, e.g. HTTP, JSON

Benefits of the Microservices Approach

- Scalability
  - Microservices can be distributed across multiple servers
  - Easier to scale-out specific services as needed
- Flexibility
  - Microservices offer a finer level of granularity than traditional apps
  - Easier to compose and rearrange to deliver new functionality
- Resilience
  - Microservices are decoupled, so they degrade/fail in isolation
  - Failures can be contained locally, without crashing the whole app

Microservices and the Cloud

- Microservices are ideally suited for deployment on the cloud
  - Easy to deploy individually
  - Typically small in size, so it's OK to start up a large number of the same microservice if demand spikes
  - Increases scalability and resilience

Microservices and Spring

- Spring Boot and Spring Cloud are well suited to microservices

- Spring Boot
    - Focuses on common core development features for creating and packaging REST-oriented microservices

- Spring Cloud
    - Makes it simple to deploy and operate microservices in the cloud (public or private)

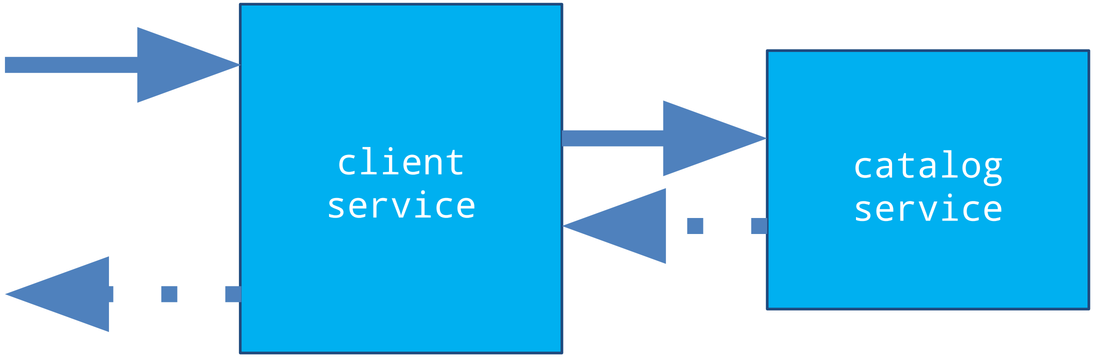
2. Microservices Application Example

- Overview
- Implementing the catalog service
- Implementing the client service

Overview

- In this section we'll show a complete (simple) example of how to create a Spring Cloud microservice application

- There are two Spring Boot applications in the demo:
  - `demo-16-clientservice`
  - `demo-16-catalogservice`



Implementing the Catalog Service (1 of 2)

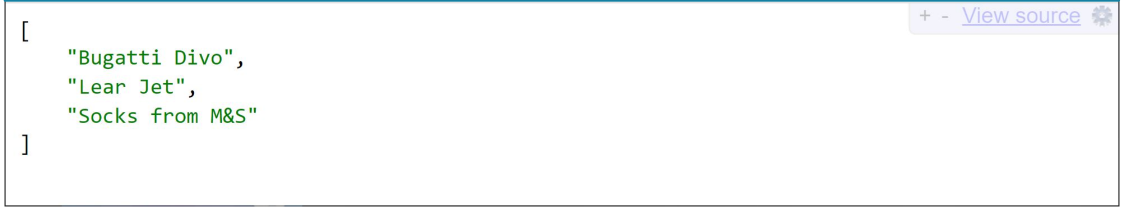
- The "catalog" service is a Spring Boot application with a REST service that returns catalog info
  - See `demo-16-catalogservice`
  - The `server.port` property is `8081`

- Take a look at the endpoints in `CatalogController`:
  - `/catalog`
  - `/catalog/{index}`

Implementing the Catalog Service (2 of 2)

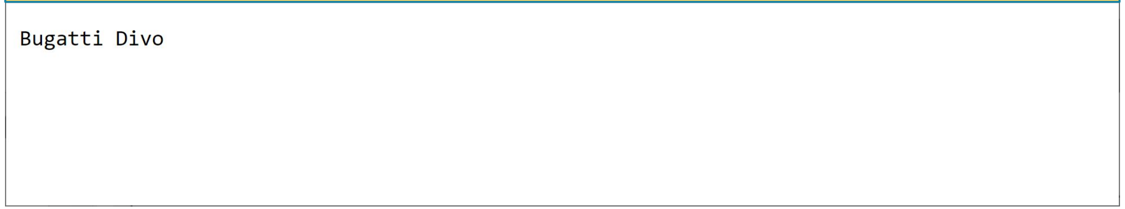
- Run the catalog app and ping the following URLs…

```
http://localhost:8081/catalog
```



```
[
    "Bugatti Divo",
    "Lear Jet",
    "Socks from M&S"
]
```

```
http://localhost:8081/catalog/0
```



```
Bugatti Divo
```

Implementing the Client Service (1 of 3)

- The "client" service is another Spring Boot application with a REST service
    - See `demo-16-clientservice`
    - The `server.port` property is `8080`

- Take a look at the endpoint in `ClientController`:
    - `/client/{index}`

Implementing the Client Service (2 of 3)

- The "client" service invokes the "catalog" service
  - Using a Spring `RestTemplate`

HTTP request

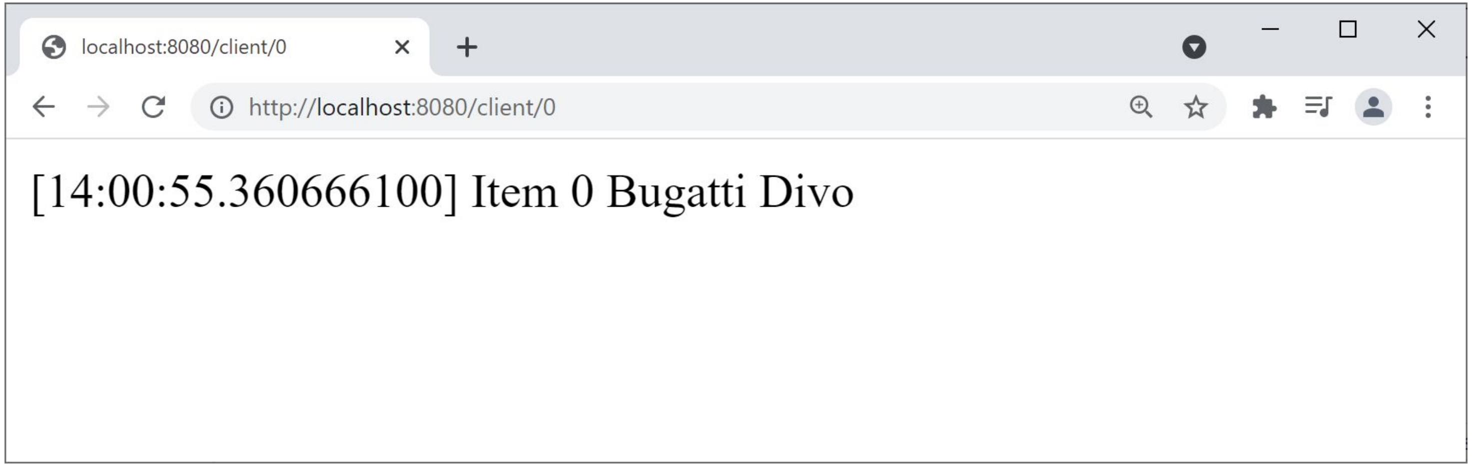
Catalog service

```
@RestController
public class ClientController {

   @GetMapping("/client/{index}")
   public String getItem(@PathVariable int index){

      URI catalogUrl = URI.create("http://localhost:8081/catalog/" + index);
      RestTemplate restTemplate = new RestTemplate();

      String result = restTemplate.getForObject(catalogUrl, String.class);
      return String.format("[%s] Item %d %s", LocalTime.now(), index, result);
   }
}
```

Pearson

Implementing the Client Service (3 of 3)

- Run the client app and ping the following URL…
  - `http://localhost:8080/client/0`



```
[14:00:55.360666100] Item 0 Bugatti Divo
```

3. Implementing Circuit Breaker Behaviour

- Overview
- Circuit breakers in Spring Cloud
- Spring Cloud circuit breaker dependency
- Spring Cloud circuit breaker example
- Seeing a circuit breaker in action

Overview

- In a microservice application, services call other services
    - E.g. ServiceA calls ServiceB, ServiceB calls ServiceC, etc.

- If any service is down, you get a ripple effect of failures
    - E.g. if ServiceC is down…
    - Then ServiceB will fail (because it depends on ServiceC)
    - Then ServiceA will fail (because it depends on ServiceB), etc.

- To avoid the ripple effect of failures, use a **circuit breaker**
    - Specify a fallback method that can be called, if a service fails

Circuit Breakers in Spring Cloud

- Spring Cloud provides a circuit breaker API
  - Via the `CircuitBreakerFactory` class

- `CircuitBreakerFactory` is an abstraction over various circuit breaker implementations, including:
  - Resilience4J (we'll use this)
  - Netfix Hystrix
  - Sentinel
  - Spring Retry

Spring Cloud Circuit Breaker Dependency

- To use the Resilience4J circuit breaker implementation, add the following dependency to the pom file in your (client) project :

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>
    <version>2.0.2</version>
</dependency>                                                   pom.xml (client project)
```

- Once you've added this dependency, Spring Boot autoconfig will automatically create a Resilience4J bean
  - This bean is exposed via `CircuitBreakerFactory`
  - See next slide for an example of how to use a circuit breaker…

Spring Cloud Circuit Breaker Example

HTTP request

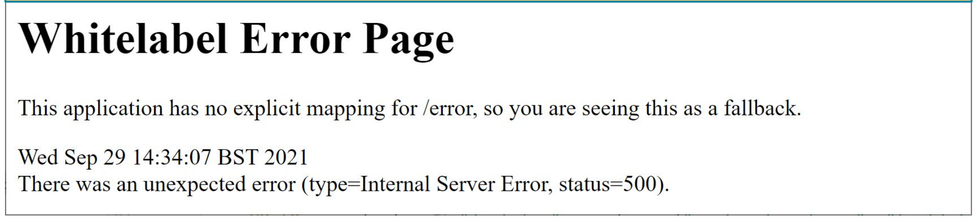
```
@RestController
public class ClientWithFallbackController {

   @Autowired
   private CircuitBreakerFactory factory;

   @GetMapping("/clientWithFallback/{index}")
   public String getItem(@PathVariable int index){

      URI catalogUrl = URI.create("http://localhost:8081/catalog/" + index);
      RestTemplate restTemplate = new RestTemplate();

      CircuitBreaker circuitBreaker = factory.create("circuitbreaker");
      String result = circuitBreaker.run(
               ()  -> restTemplate.getForObject(catalogUrl, String.class),
               err -> getFallback(index));
      return String.format("[%s] Item %d %s", LocalTime.now(), index, result);
   }

   public String getFallback(int i) { return "FALLBACK-ITEM-" + i;}
}
```

Catalog service

Seeing a Circuit Breaker in Action

- To see the effect of the circuit breaker, follow these steps:
  - Stop the catalog service
  - Then ping the following client endpoints…

```
http://localhost:8080/client/0
```

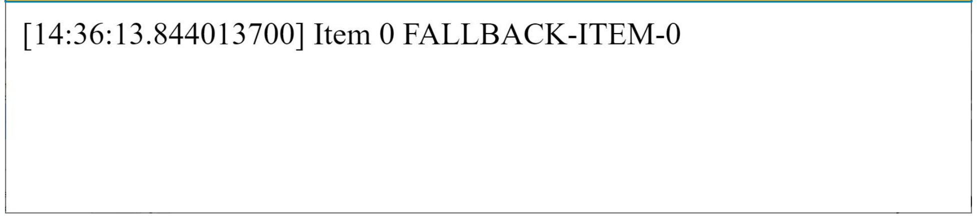


Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Wed Sep 29 14:34:07 BST 2021
There was an unexpected error (type=Internal Server Error, status=500).

```
http://localhost:8080/clientWithFallback/0
```



```
[14:36:13.844013700] Item 0 FALLBACK-ITEM-0
```

Summary

- Overview of microservices
- Microservices application example
- Implementing circuit breaker behaviour