**Exercises on Microservices with Spring Boot 3.0**

1. **Build a User and Order Management System**

**pom.xml dependencies:**

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-openfeign</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

</dependencies>

**application.yml**

server:

port: 8081

spring:

datasource:

url: jdbc:mysql://localhost:3306/userdb

username: root

password: yourpassword

jpa:

hibernate:

ddl-auto: update

show-sql: true

User.java

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

}

UserService.java

@Service

@RequiredArgsConstructor

public class UserService {

private final UserRepository userRepository;

public User saveUser(User user) {

return userRepository.save(user);

}

public Optional<User> getUser(Long id) {

return userRepository.findById(id);

}

}

UserController.java

@RestController

@RequestMapping("/users")

@RequiredArgsConstructor

public class UserController {

private final UserService userService;

@PostMapping

public ResponseEntity<User> create(@RequestBody User user) {

return ResponseEntity.ok(userService.saveUser(user));

}

@GetMapping("/{id}")

public ResponseEntity<User> getUser(@PathVariable Long id) {

return userService.getUser(id)

.map(ResponseEntity::ok)

.orElse(ResponseEntity.notFound().build());

}

}

**pom.xml**

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-config-server</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

</dependencies>

**ConfigServerApplication.java**

@SpringBootApplication

@EnableConfigServer

public class ConfigServerApplication {

public static void main(String[] args) {

SpringApplication.run(ConfigServerApplication.class, args);

}

}

**application.yml**

server:

port: 8888

spring:

cloud:

config:

server:

git:

uri: https://github.com/yourusername/config-repo

**2. EUREKA SERVER**

**pom.xml**

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

</dependencies>

**EurekaServerApplication.java**

@SpringBootApplication

@EnableEurekaServer

public class EurekaServerApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaServerApplication.class, args);

}

}

**application.yml**

server:

port: 8761

eureka:

client:

register-with-eureka: false

fetch-registry: false

**3. PRODUCT SERVICE**

**pom.xml**

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-config</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

</dependency>

</dependencies>

**bootstrap.yml**

spring:

application:

name: product-service

cloud:

config:

uri: http://localhost:8888

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka

**Product.java**

@Entity

@Data

public class Product {

@Id

@GeneratedValue

private Long id;

private String name;

private String description;

}

**ProductRepository.java**

public interface ProductRepository extends JpaRepository<Product, Long> {

}

**ProductController.java**

@RestController

@RequestMapping("/products")

@RequiredArgsConstructor

public class ProductController {

private final ProductRepository repository;

@PostMapping

public Product save(@RequestBody Product product) {

return repository.save(product);

}

@GetMapping

public List<Product> list() {

return repository.findAll();

}

}

**4. INVENTORY SERVICE**

**bootstrap.yml**

spring:

application:

name: inventory-service

cloud:

config:

uri: http://localhost:8888

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka

**Stock.java**

@Entity

@Data

public class Stock {

@Id

@GeneratedValue

private Long id;

private Long productId;

private int quantity;

}

**StockRepository.java**

public interface StockRepository extends JpaRepository<Stock, Long> {

List<Stock> findByProductId(Long productId);

}

**ProductClient.java (Feign)**

@FeignClient(name = "product-service")

public interface ProductClient {

@GetMapping("/products")

List<Product> getAllProducts();

}

**InventoryController.java**

@RestController

@RequestMapping("/stocks")

@RequiredArgsConstructor

public class InventoryController {

private final StockRepository stockRepo;

private final ProductClient productClient;

@GetMapping

public List<Stock> listStock() {

return stockRepo.findAll();

}

@PostMapping

public Stock save(@RequestBody Stock stock) {

return stockRepo.save(stock);

}

@GetMapping("/products")

public List<Product> getProductsFromProductService() {

return productClient.getAllProducts();

}

}

**1. API GATEWAY**

**pom.xml**

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-gateway</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>io.github.resilience4j</groupId>

<artifactId>resilience4j-spring-boot2</artifactId>

</dependency>

</dependencies>

**application.yml**

server:

port: 8080

spring:

application:

name: api-gateway

cloud:

gateway:

routes:

- id: customer-service

uri: lb://CUSTOMER-SERVICE

predicates:

- Path=/customer/\*\*

filters:

- RewritePath=/customer/(?<segment>.\*), /$\{segment}

- name: RequestRateLimiter

args:

redis-rate-limiter.replenishRate: 5

redis-rate-limiter.burstCapacity: 10

- AddResponseHeader=X-Cache, Cached

- id: billing-service

uri: lb://BILLING-SERVICE

predicates:

- Path=/billing/\*\*

filters:

- RewritePath=/billing/(?<segment>.\*), /$\{segment}

- name: RequestRateLimiter

args:

redis-rate-limiter.replenishRate: 3

redis-rate-limiter.burstCapacity: 5

- AddResponseHeader=X-Cache, Cached

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka

**Enable Redis Rate Limiting**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-redis-reactive</artifactId>

</dependency>

**ApiGatewayApplication.java**

@SpringBootApplication

@EnableDiscoveryClient

public class ApiGatewayApplication {

public static void main(String[] args) {

SpringApplication.run(ApiGatewayApplication.class, args);

}

}

**2. CUSTOMER-SERVICE**

**application.yml**

server:

port: 8081

spring:

application:

name: customer-service

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka

**CustomerController.java**

@RestController

@RequestMapping("/api")

public class CustomerController {

@GetMapping("/greet")

public String greet() {

return "Hello from Customer Service";

}

}

**3. BILLING-SERVICE**

**application.yml**

server:

port: 8082

spring:

application:

name: billing-service

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka

**BillingController.java**

@RestController

@RequestMapping("/api")

public class BillingController {

@GetMapping("/invoice")

public String invoice() {

return "Invoice generated by Billing Service";

}

}

**pom.xml**

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>io.github.resilience4j</groupId>

<artifactId>resilience4j-spring-boot3</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-aop</artifactId>

</dependency>

</dependencies>

**application.yml**

server:

port: 8083

resilience4j:

circuitbreaker:

instances:

paymentService:

registerHealthIndicator: true

slidingWindowSize: 5

minimumNumberOfCalls: 5

failureRateThreshold: 50

waitDurationInOpenState: 10s

permittedNumberOfCallsInHalfOpenState: 2

automaticTransitionFromOpenToHalfOpenEnabled: true

management:

endpoints:

web:

exposure:

include: "\*"

**ThirdPartyClient.java**

@Service

public class ThirdPartyClient {

public String callSlowApi() {

try {

Thread.sleep(3000); // simulate 3s delay

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

if (new Random().nextBoolean()) {

throw new RuntimeException("Third-party API timeout");

}

return "Success from third-party API";

}

}

**PaymentService.java**

@Service

@RequiredArgsConstructor

public class PaymentService {

private final ThirdPartyClient thirdPartyClient;

private static final Logger logger = LoggerFactory.getLogger(PaymentService.class);

@CircuitBreaker(name = "paymentService", fallbackMethod = "fallbackResponse")

public String makePayment() {

return thirdPartyClient.callSlowApi();

}

public String fallbackResponse(Throwable t) {

logger.warn("Fallback triggered due to: {}", t.toString());

return "Payment failed temporarily. Please try again later.";

}

}

**PaymentController.java**

@RestController

@RequiredArgsConstructor

public class PaymentController {

private final PaymentService paymentService;

@GetMapping("/pay")

public String pay() {

return paymentService.makePayment();

}

}

**Microservices with API gateway**

**AccountController.java**

package com.cognizant.account.controller;

import org.springframework.web.bind.annotation.\*;

import java.util.Map;import java.util.HashMap;

@RestController@RequestMapping("/accounts")public class AccountController {

@GetMapping("/{number}")

public Map<String, Object> getAccountDetails(@PathVariable String number) {

Map<String, Object> response = new HashMap<>();

response.put("number", number);

response.put("type", "savings");

response.put("balance", 234343);

return response;

}

}

**LoanController.java**

package com.cognizant.loan.controller;

import org.springframework.web.bind.annotation.\*;

import java.util.Map;import java.util.HashMap;

@RestController@RequestMapping("/loans")public class LoanController {

@GetMapping("/{number}")

public Map<String, Object> getLoanDetails(@PathVariable String number) {

Map<String, Object> response = new HashMap<>();

response.put("number", number);

response.put("type", "car");

response.put("loan", 400000);

response.put("emi", 3258);

response.put("tenure", 18);

return response;

}

}

### **pom.xml**

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>2023.0.0</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies></dependencyManagement>

### **application.properties**

server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

logging.level.com.netflix.eureka=OFF

logging.level.com.netflix.discovery=OFF

### **EurekaDiscoveryServerApplication.java**

@SpringBootApplication@EnableEurekaServerpublic class EurekaDiscoveryServerApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaDiscoveryServerApplication.class, args);

}

}

**application.properties**

server.port=8080

spring.application.name=account-service

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

### **AccountApplication.java**

@SpringBootApplication@EnableDiscoveryClientpublic class AccountApplication {

public static void main(String[] args) {

SpringApplication.run(AccountApplication.class, args);

}

}

**AccountController.java**

@RestController@RequestMapping("/accounts")public class AccountController {

@GetMapping("/{number}")

public Map<String, Object> getAccountDetails(@PathVariable String number) {

Map<String, Object> map = new HashMap<>();

map.put("number", number);

map.put("type", "savings");

map.put("balance", 234343);

return map;

}

}

## **Loan Microservice**

### **application.properties**

server.port=8082

spring.application.name=greet-service

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

### **GreetServiceApplication.java**

@SpringBootApplication@EnableDiscoveryClientpublic class GreetServiceApplication {

public static void main(String[] args) {

SpringApplication.run(GreetServiceApplication.class, args);

}

}

### **GreetController.java**

@RestController@RequestMapping("/greet")public class GreetController {

@GetMapping

public String greet() {

return "Hello World";

}

}

## **API Gateway**

### **application.properties**

server.port=9090

spring.application.name=api-gateway

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

spring.cloud.gateway.discovery.locator.enabled=true

spring.cloud.gateway.discovery.locator.lower-case-service-id=true

### **ApiGatewayApplication.java**

@SpringBootApplication@EnableDiscoveryClientpublic class ApiGatewayApplication {

public static void main(String[] args) {

SpringApplication.run(ApiGatewayApplication.class, args);

}

}

## **Global Logging Filter in API Gateway**

### **LogFilter.java**

@Componentpublic class LogFilter implements GlobalFilter, Ordered {

private static final Logger logger = LoggerFactory.getLogger(LogFilter.class);

@Override

public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {

logger.info("Incoming Request: " + exchange.getRequest().getURI());

return chain.filter(exchange);

}

@Override

public int getOrder() {

return -1;

}

}