Exercises on Microservices with Spring Boot 3.0

1. **Build a User and Order Management System Problem:** Create two microservices:
   * **User Service** to manage users.
   * **Order Service** to manage orders placed by users.

Requirements:

* + Use **REST APIs**.
  + Communicate between services using **WebClient (Spring WebFlux)** or

OpenFeign.

* + Store data in **MySǪL** or **PostgreSǪL**.

Answer:

User Services:

@Entity

public class User {

@Id

@GeneratedValue

private Long id;

private String name;

private String email;

// getters and setters

}

public interface UserRepository extends JpaRepository<User, Long> {}

@RestController

@RequestMapping("/users")

public class UserController {

@Autowired private UserRepository repo;

@PostMapping

public User saveUser(@RequestBody User user) {

return repo.save(user);

}

@GetMapping("/{id}")

public User getUser(@PathVariable Long id) {

return repo.findById(id).orElseThrow();

}

}

spring.datasource.url=jdbc:mysql://localhost:3306/userdb

spring.datasource.username=root

spring.datasource.password=yourpassword

spring.jpa.hibernate.ddl-auto=update

**Order Services:**

@Entity

public class Order {

@Id

@GeneratedValue

private Long id;

private Long userId;

private String item;

private Double amount;

// getters and setters

}

public interface OrderRepository extends JpaRepository<Order, Long> {}

@FeignClient(name = "user-service", url = "http://localhost:8080")

public interface UserClient {

@GetMapping("/users/{id}")

User getUser(@PathVariable("id") Long id);

}

@RestController

@RequestMapping("/orders")

public class OrderController {

@Autowired private OrderRepository repo;

@Autowired private UserClient userClient;

@PostMapping

public Order placeOrder(@RequestBody Order order) {

userClient.getUser(order.getUserId()); // Validate user

return repo.save(order);

}

@GetMapping("/{id}")

public Order getOrder(@PathVariable Long id) {

return repo.findById(id).orElseThrow();

}

}

1. **Inventory Management System with Service Discovery Problem:** Create:
   * **Product Service**: Manage products and stock.
   * **Inventory Service**: Track stock levels for each product.

Requirements:

* + Use **Spring Cloud Netflix Eureka** for **service discovery**.
  + Implement **centralized configuration** using Spring Cloud Config Server.

**Answer:**

Setup Spring Cloud Config Server:

server:

port: 8888

spring:

cloud:

config:

server:

git:

uri: <https://github.com/your-org/config-repo>

Eureka Server:

server:

port: 8761

eureka:

client:

register-with-eureka: false

fetch-registry: false

@EnableEurekaServer

@SpringBootApplication

public class EurekaServerApp {}

Product Service:

@Entity

public class Product {

@Id

private Long id;

private String name;

private int stock;

}

spring:

application:

name: product-service

eureka:

client:

service-url:

defaultZone: <http://localhost:8761/eureka>

Inventory Service:

@FeignClient("product-service")

public interface ProductClient {

@GetMapping("/products/{id}")

Product getProduct(@PathVariable Long id);

}

@RestController

public class InventoryController {

@Autowired private ProductClient productClient;

@GetMapping("/stock/{id}")

public int getStock(@PathVariable Long id) {

Product product = productClient.getProduct(id);

return product.getStock();

}

}

1. **Implement an API Gateway**

**Problem:** Create an **API Gateway** to route requests to:

* + **Customer Service**
  + **Billing Service Requirements:**
  + Use **Spring Cloud Gateway**.
  + Implement **rate limiting**, **caching**, and **path rewriting**.

**Answer:**

spring:

cloud:

gateway:

routes:

- id: customer-service

uri: http://localhost:8081

predicates:

- Path=/customer/\*\*

filters:

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

- RequestRateLimiter=replenishRate=5,burstCapacity=10

- id: billing-service

uri: http://localhost:8082

predicates:

- Path=/billing/\*\*

filters:

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

- AddResponseHeader=X-Cache-Status, HIT

@Bean

public KeyResolver keyResolver() {

return exchange -> Mono.just(exchange.getRequest().getRemoteAddress().getAddress().getHostAddress());

}

1. **Resilient Microservices with Circuit Breaker Problem:** A **Payment Service** calls a slow third-party API. **Requirements:**
   * Implement **Circuit Breaker** and **fallback logic** using **Resilience4j**.
   * Log and monitor fallback events.

**Answer:**

**Payment Service:**

@Service

public class PaymentService {

@CircuitBreaker(name = "paymentCB", fallbackMethod = "fallbackPayment")

public String callPaymentAPI() {

// Simulate delay

Thread.sleep(5000);

return "Payment processed";

}

public String fallbackPayment(Throwable t) {

log.error("Fallback executed: " + t.getMessage());

return "Payment service is currently unavailable, please try later.";

}

}

@RestController

public class PaymentController {

@Autowired private PaymentService service;

@GetMapping("/pay")

public String makePayment() {

return service.callPaymentAPI();

}

}

resilience4j:

circuitbreaker:

instances:

paymentCB:

registerHealthIndicator: true

failureRateThreshold: 50

waitDurationInOpenState: 10s

permittedNumberOfCallsInHalfOpenState: 2