**HAND ON EXCERCISE-WEEK 5**

**1.** How do you expose a product catalog via REST in a microservice, and register it with a service registry?

xml

<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-web</artifactId>

</dependency>

**application.yml:**

spring:

application:

name: product-service

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka/

**ProductController.java**

@RestController

@RequestMapping("/products")

public class ProductController {

@GetMapping

public List<String> getAllProducts() {

return List.of("Laptop", "Mobile", "Tablet");

}

}

**2.**How do you communicate between microservices?

**pom.xml:**

<dependency>

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

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

</dependency>

**application.yml:**

spring:

application:

name: order-service

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka/

**ProductClient.java**

@FeignClient(name = "product-service")

public interface ProductClient {

@GetMapping("/products")

List<String> getProducts();

}

**OrderController.java**

@RestController

@RequestMapping("/orders")

public class OrderController {

@Autowired

private ProductClient productClient;

@GetMapping("/available-products")

public List<String> fetchAvailableProducts() {

return productClient.getProducts();

}

}

**3.**What happens if a microservice fails? How can you protect the system?

**pom.xml:**

<dependency>

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

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

</dependency>

**OrderController.java**

@CircuitBreaker(name = "productServiceCB", fallbackMethod = "fallbackProductList")

@GetMapping("/safe-products")

public List<String> safeFetchProducts() {

return productClient.getProducts();

}

public List<String> fallbackProductList(Throwable t) {

return List.of("Default Product");

}

**4.** How do you centralize routing for multiple microservices?

**application.yml:**

spring:

cloud:

gateway:

routes:

- id: product-service

uri: lb://product-service

predicates:

- Path=/products/\*\*

- id: order-service

uri: lb://order-service

predicates:

- Path=/orders/\*\*

**pom.xml:**

<dependency>

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

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

</dependency>

**5.** How do you manage configuration across multiple microservices?

**pom.xml:**

<dependency>

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

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

</dependency>

**ConfigServerApp.java**

@EnableConfigServer

@SpringBootApplication

public class ConfigServerApp {

public static void main(String[] args) {

SpringApplication.run(ConfigServerApp.class, args);

}

}

**application.yml**

server:

port: 8888

spring:

cloud:

config:

server:

git:

uri: https://github.com/your-org/microservice-configs

spring:

application:

name: product-service

cloud:

config:

uri: http://localhost:8888

**6.** How can you secure microservices using JWT tokens?

**JwtUtil.java**

public class JwtUtil {

private final String SECRET = "secret-key";

public String generateToken(String username) {

return Jwts.builder()

.setSubject(username)

.setExpiration(new Date(System.currentTimeMillis() + 900\_000))

.signWith(SignatureAlgorithm.HS256, SECRET)

.compact();

}

public String validateToken(String token) {

return Jwts.parser().setSigningKey(SECRET).parseClaimsJws(token).getBody().getSubject();

}

}

**TokenController.java**

@PostMapping("/token")

public String getToken(@RequestBody AuthRequest authRequest) {

return jwtUtil.generateToken(authRequest.getUsername());

}

Each protected microservice adds a OncePerRequestFilter to validate token in headers.

**7.** How do you trace requests across services?

**pom.xml:**

<dependency>

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

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

</dependency>

spring:

zipkin:

base-url: http://localhost:9411

sleuth:

sampler:

probability: 1.0