NAGELLA PRAJITHRAJ

Day 7: Spring Boot and Microservices - Scalable Traffic Monitoring

Task 1: Migrate to Spring Boot for a Streamlined Setup of Microservices for Different City Zones

Set Up Spring Boot Project:

Use Spring Initializr to create a new Spring Boot project. Add dependencies for Spring Web and other necessary components.

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-config</artifactId>
    </dependency>
</dependencies>
```

1.

Create Microservices for Different City Zones:

Develop separate microservices for different city zones, e.g., ZoneA, ZoneB, etc. Each service will have its own controller and service layer.

ZoneA Service:

java

```
@SpringBootApplication
public class ZoneAApplication {
    public static void main(String[] args) {
```

```
        SpringApplication.run(ZoneAApplication.class, args);
    }
}

@RestController
@RequestMapping("/zoneA")
public class ZoneAController {

    @Autowired
    private ZoneAService zoneAService;

    @GetMapping("/traffic")
    public TrafficData getTrafficData() {
        return zoneAService.getTrafficData();
    }
}

@Service
public class ZoneAService {

    public TrafficData getTrafficData() {
        // Logic to fetch and return traffic data for ZoneA
        return new TrafficData();
    }
}

public class TrafficData {
    // Traffic data fields and methods
}
```

2.

Set Up Application Properties:
Configure application properties for each microservice.
`application.properties`
properties
Copy code

```
spring.application.name=zone-a-service
eureka.client.service-url.defaultZone=http://localhost:8761/eureka/
server.port=8081
```

3.

Task 2: Implement Eureka for Service Discovery Among Traffic Monitoring Microservices

Set Up Eureka Server:
Create a new Spring Boot application for the Eureka server.
EurekaServerApplication.java:
java
Copy code

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(EurekaServerApplication.class, args);
    }
}
```

application.properties:
properties
Copy code

```
spring.application.name=eureka-server
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

1.

Configure Eureka Client in Microservices:
Ensure each microservice registers with the Eureka server.
`application.properties` for microservices:
properties
Copy code

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

2.

Task 3: Configure Spring Cloud Config for Managing Microservice Settings During Peak and Off-Peak Hours

Set Up Spring Cloud Config Server:
Create a Spring Boot application for the Config server.
ConfigServerApplication.java:
java
Copy code

```
@SpringBootApplication
@EnableConfigServer
```

```
public class ConfigServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(ConfigServerApplication.class, args);
    }
}
```

application.properties:

properties

Copy code

```
spring.application.name=config-server
server.port=8888
spring.cloud.config.server.git.uri=https://github.com/your-config-repo
```

1.

Create Configuration Files in Git Repository:

Store configurations for different environments (e.g., peak and off-peak) in a Git repository.

zone-a-service.yml:

yaml

Copy code

```
peak:
  traffic:
    updateInterval: 5
off-peak:
  traffic:
    updateInterval: 30
```

2.

Configure Microservices to Use Spring Cloud Config:

Update the `application.properties` of each microservice to use the Config server.

`application.properties` for microservices:

properties

Copy code

```
spring.application.name=zone-a-service
eureka.client.service-url.defaultZone=http://localhost:8761/eureka/
server.port=8081
spring.cloud.config.uri=http://localhost:8888
spring.profiles.active=peak
```

3.

Use Configuration Properties in Services:
Inject configuration properties into your services.
ZoneAService.java:
java
Copy code

```
@Service
@ConfigurationProperties(prefix = "traffic")
public class ZoneAService {

    private int updateInterval;

    public TrafficData getTrafficData() {
        // Use updateInterval for fetching traffic data
        return new TrafficData();
    }

    public void setUpdateInterval(int updateInterval) {
        this.updateInterval = updateInterval;
    }
}
```

4.

By migrating to Spring Boot and setting up microservices for different city zones, you can achieve a streamlined and scalable architecture for traffic monitoring. Implementing Eureka for service discovery and Spring Cloud Config for dynamic configuration management during peak and off-peak hours ensures a robust and flexible system.

4o