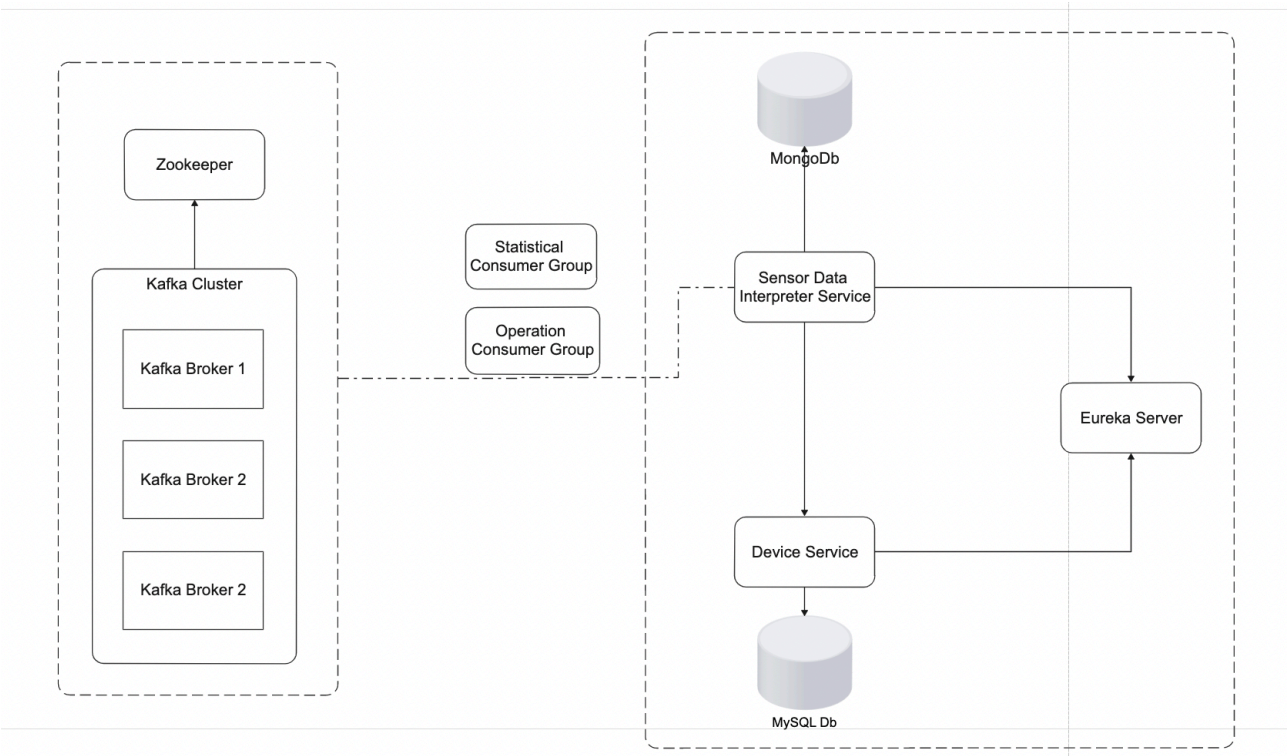
Sensor Data Interpreter Uygulaması

Proje Github Url

Ön koşullar: Docker

Genel Mimari



Kullanılan Teknolojiler

1. **Sensor Data Interpreter Service:**

- Spring Boot 3.0 (Aynı topic üzerinde birden fazla dinleyici için, non-blocking retry mekanizmasında sağlanan özelliklerden dolayı bu versiyon tercih edilmiştir. Detay)
- Java 17 (Spring 3.0 ile uyumlu minimum versiyon)
- Kafka, Zookeeper
- MongoDb

2. **Device Service:** Spring Boot 2 , Java 11, MySQL
3. **Eureka Service:** Spring Boot 2 , Java 11

Docker-compose kullanılarak bu servisler ayağa kaldırılmaktadır. Ek olarak MongoDb yi görüntüleyebilmek için MongoExpress(container: mongo-ui) , Kafka Cluster (Broker ve Queue) detaylarını görüntüleyebilmek için de Kafdrop(container name: kafka-ui) konfigürasyonları yapılmıştır.

Servislere Detaylı Bakış

Sensor Data Interpreter Servis :

- Uygulama docker-compose ile ayağa kalkarken script ile mongodbde bu servis için yeni bir kullanıcı oluşturulmaktadır. (mongo-init.js)
- Uygulama discovery servise bağlanmaktadır.
- Uygulama ayağa kalktıktan 10 saniye sonra çalışan bir job ile kuyruğa her iki mesaj grubu için de kayıtlar atmaktadır. Retry mekanizmasını görmek için bir tane hatalı kayıt atılmaktadır. MongoUI üzerinden kontrol edilebilir. (Normalde bu mimaride producer işini yapan başka bir microservis olmalı ama ana case e odaklanmak için produceri bu servis içerisinde ele aldım)
- Bu serviste Case Study dokumanında belirtilen ana kuyruğa gelen mesajları okumak için Kafka Consumer Group mantığı kullanılmıştır. 1 adet Zookeeper server, 3 Broker ve 2 consumer group oluşturulmuştur.
- Gelen mesajlar iki consumer group tarafından okunmaktadır. Groupların direk ilgili mesajları okumaları için Kafka konfigürasyonunda filtreleme kullanılmıştır.
- Her iki groupta mesaj okunurken hata alması durumunda **non-blocking retry** ve **dltHandler** özellikleri kullanılmıştır. Hata alan mesaj 2 kere **daha** okunmaya çalışılacak yine de hata alırsa **dltHandler** tarafından mesaj mongoDbye yazılacaktır. Mesajdaki validation hataları gibi bazı exceptionlar retry mekanizmasından özellikle hariç tutulmuştur.
- Operasyonel, istatistiksel ve başarısız olan mesajlar mongodb de ayrı collectionlara yazılmakta ve bu kayıtlar Mongo UI üzerinden ulaşılabilir.
- Operasyonel group consumer, gelen operasyonel mesajı okuyup device-service ten **FeignClient** ile cihaz tipine göre optimal sıcaklık bilgilerini çekmektedir. Mesajlardan yorumlanarak elde edilen bilgiler “computedSensorDataMessages" collectionına yazılr. FeignClienttan dönen bazı hataların retry mekanizmasından hariç tutulması için device servisten dönen exceptionlar ayrı bir configurasyonla customize edilmiştir.

Fakat bu kısmı, retry ve dlthandler mekanizmasının işlendiğini senaryoyu job ile tetikleyerek size gösterebilmek için yoruma aldım.

```
//exclude = {FeignApiClientException.class,
NotFoundException.class, ValidationException.class})

exclude = {NotFoundException.class,
ValidationException.class})
```

- Device ve verilen zaman aralığında device a ait lokasyon bilgilerini almak için açılan endpointin detaylarını ve örnek requesti swagger sayfasında bulabilirsiniz.
- Endpoint belirli bir formatta date ve time bilgisi almaktadır. Gerekli validasyonların yapılması için custom JsonDesirializier eklenmiştir.(CustomDateDeserializer)
- Endpointten hata alınması durumunda genel bir api error response oluşturulmuş ve custom exceptionlara ait hata mesajı dönülmüştür.

Device Service:

- Uygulama docker-compose ile ayağa kalkerken MySQL de yeni bir db ve kullanıcı oluşturulmakta, ayrıca docker init script ile device tipine göre optimal sıcaklık bilgileri db ye yazılmaktadır.
- Uygulama discovery servise bağlanmaktadır.
- Açılan endpoint ile servis MySQL veritabanına bağlanıp ilgili dataları dönmektedir.

```
http://localhost:8084/optimalTemperature/{deviceType}
```

Eureka Server:

- Mikroservislerimiz bu servise register oluyorlar ve birbirleri arasındaki iletişimi tek noktadan sağlamış oluyoruz.

Docker-Compose ile Servisleri Çalıştırma

1. Uygulamayı çalıştırmak için projenin olduğu klasöre gidin ve aşağıdaki komutu çalıştırın.

```
docker compose up -d
```

2. Tüm containerlar başarılı bir şekilde çalışmaya başladığında Sensor Data Interpreter Servisi ve diğer servislere aşağıdaki adreslerden ulaşabilirsiniz.

	Url	Ek Bilgi
Sensor Data Interpreter Service	http://localhost:8080/	
Sensor Data Interpreter Swagger UI	http://localhost:8080/swagger-ui/index.html	
Device Service	http://localhost:8084/	
Eureka Server	http://localhost:8761/	Discovery Service
Mongo Express	http://localhost:8081/	MongoDb UI
Kafdrop (Kafka UI)	http://localhost:9000/	Kafka Cluster ve Brokerlar için UI
Php My Admin	http://localhost:8082/	MySql diye bağlanmak ve detayları görüntülemek için kullanılmıştır. Kullanıcı adı ve şifre için application.yaml dosyasını kontrol ediniz.

3. Tüm servisleri durdurmak için aşağıdaki komutu kullanabilirsiniz.

```
docker compose down
```