Microservices

Product and Order Services with gRPC and API Gateway

Project Overview

In this project, I built a microservices architecture focused on a **Product Service** and an **Order Service**. My primary goals included creating microservices using RESTful APIs, setting up an API Gateway for efficient routing, and implementing **gRPC** for communication between microservices.

I developed the **Product Service** with REST endpoints, acting as a product catalog service, and then introduced **gRPC** where the **Product Service** was set up as the gRPC server. The **Order Service** acted as a gRPC client to request product details from the **Product Service**.

duce gRPC for efficient inter-service communication.

1. Create a Basic Microservice

I developed the **Product Catalog** microservice with RESTful endpoints that provided basic operations for managing product data like `productId`, `name`, `description`, `quantity`, and `price`.

2. Implement API Gateway

I set up an API Gateway to manage API requests and handle traffic routing. It efficiently directed incoming requests to the correct microservices, enabling centralized request handling for the architecture.

3. Integrate Microservices

I integrated the **Product Catalog** microservice with the API Gateway to ensure seamless routing and traffic management for client requests.

4. Explore gRPC

I introduced **gRPC** by setting up a new microservice, the **Order Service**, as the client and configured it to communicate with the **Product Service** (acting as the gRPC server). This communication occurred whenever the **Order Service** needed product information.

gRPC Setup

Proto Definition

I defined the **gRPC service** in a `.proto` file using the `proto3` syntax. This file contains the structure for the **ProductService** that retrieves product information.

```
syntax = "proto3";

package com.microservices.ecommerce.grpc;

option java_outer_classname = "ProductProto";

message ProductRequest {
  int64 productId = 1;
}

message ProductResponse {
  int64 productId = 1;
  string name = 2;
  string description = 3;
  double quantity = 4;
  double price = 5;
}

service ProductService {
  rpc GetProduct (ProductRequest) returns (ProductResponse);
}
```

Product Service Implementation

I implemented the **ProductService** as a gRPC server. The following code demonstrates how the service responds to requests and retrieves product information from a database:

```
package com.microservice.ecommerce.product;

import com.microservice.ecommerce.exception.ProductNotFoundException;
import com.microservices.ecommerce.grpc.ProductProto;
import com.microservices.ecommerce.grpc.ProductServiceGrpc;
import io.grpc.stub.StreamObserver;
import net.devh.boot.grpc.server.service.GrpcService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@GrpcService
public class ProductGRPTService extends
ProductServiceGrpc.ProductServiceImplBase {
    private final ProductRepository productRepository;
    private final Logger logger =
LoggerFactory.getLogger(ProductGRPTService.class);

    public ProductGRPTService(ProductRepository productRepository) {
        this.productRepository = productRepository;
    }

    @Override
    public void getProduct(ProductProto.ProductRequest request,
StreamObserver<ProductProto.ProductResponse> responseObserver) {
        Product product = productRepository.findById(request.getProductId())
                            .orElseThrow(() -> new
ProductNotFoundException(request.getProductId()));

        ProductProto.ProductResponse productResponse =
ProductProto.ProductResponse.newBuilder()
                .setProductId(product.getProductId())
                .setName(product.getName())
                .setDescription(product.getDescription())
                .setQuantity(product.getAvailableQuantity())
                .setPrice(product.getPrice())
                .build();

        responseObserver.onNext(productResponse);
        logger.info("Order service just requested a product with id {}",
product.getProductId());
        responseObserver.onCompleted();
    }
}
```

Conclusion

This lab was a great opportunity to explore microservice development and inter-service communication using gRPC. I successfully implemented a **Product Service** and **Order Service**, configured gRPC for efficient communication, and set up an API Gateway for centralized request routing. The API Gateway and gRPC combination enhanced the scalability and performance of the microservices architecture.