Microservices Architecture with gRPC and REST APIs

Sandra Kumi

REVIEWER Mr. Thomas Darko

1. Introduction

Microservices architecture is an approach where a large application is broken down into smaller, independent services that communicate over the network. Each service focuses on a specific business capability and can be developed, deployed, and scaled independently. This document details the architecture of a microservices-based system with an API Gateway.

---

2. Architecture Overview

In this architecture, an API Gateway is used to route requests to different microservices. Each microservice runs in a separate Docker container, and the services communicate over the Docker network.

Key Components:

- API Gateway: Handles incoming requests and routes them to the appropriate microservice.
- Microservices: Independently running services responsible for specific functionalities (e.g., Product Service, Order Service).
- Service Discovery: Automatically finds and routes requests to running services.

2.1 Microservices Architecture Diagram

An overview of the microservices architecture is shown below. Services communicate through REST APIs, and the API Gateway forwards client requests to the appropriate service.

- API Gateway: Manages routing, load balancing, and security.
- Microservices: Product Service, Order Service, etc., each running on separate Docker containers.

---

3. API Gateway Configuration

3.1 API Gateway Properties

The API Gateway is configured using Spring Cloud Gateway to route requests to backend services.

```
spring.application.name=Api-gateway

server.port=8080
```

```
# Product Service Route

spring.cloud.gateway.routes[0].id=product-service

spring.cloud.gateway.routes[0].uri=http://product-service:8081

spring.cloud.gateway.routes[0].predicates=Path=/api/products/**


#Order Service Route

spring.cloud.gateway.routes[1].id=order-service
spring.cloud.gateway.routes[1].uri=http://order-service:8082
spring.cloud.gateway.routes[1].predicates=Path=/api/orders/**


# Service Discovery

spring.cloud.gateway.discovery.locator.enabled=true

spring.cloud.gateway.discovery.locator.lower-case-service-id=true
```

In the above configuration:

- Requests with the path "/api/products/**" are routed to the Product Service running on http://product-service:8081 and the path "/api/orders/**" are routed to the Order Service running on http://order-service:8082 .
- Service Discovery is enabled to dynamically discover services registered in the Docker network.

4. Services and Their Responsibilities

- **Product Catalog Service**
  - Manages product-related operations such as listing products, fetching details, and interacting with the database.
  - Uses RESTful APIs for external communication.
- **Order Service**
  - Manages order-related operations.
  - Uses gRPC for communication with other microservices.
- **API Gateway**

- Acts as an entry point for routing client requests to appropriate services (e.g., Product Catalog).
- Manages traffic and security for external APIs.

5. gRPC Setup for Order Service

5.1 gRPC Product Service Proto Definition

```
“syntax="proto3";

option java_multiple_files = true;
option java_package = "com.order.service.grpc";
option java_outer_classname = "ProductProto";

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

message ProductRequest {
  int64 order_id = 1;
}

message ProductResponse {
  int64 id = 1;
  string name = 2;
  string description = 3;
  double price = 4;
  int32 quantity = 5;
}”
```

5.2 ProductService Implementation

```
“@GrpcService
public class ProductgRPCService extends ProductServiceGrpc.ProductServiceImplBase {
    private final ProductRepo productRepo;

    public ProductgRPCService(ProductRepo productRepo) {
        this.productRepo = productRepo;
```

```
   }

   public void getProduct(ProductRequest request, StreamObserver<ProductResponse>
responseObserver) {
       Product product = productRepo.findById(request.getOrderId()).orElseThrow(() -> new
EntityNotFoundException("Product not found"));

       ProductResponse productResponse = ProductResponse.newBuilder()
               .setId(product.getProductId())
               .setName(product.getName())
               .setDescription(product.getDescription())
               .setPrice(product.getPrice())
               .setQuantity(product.getQuantity())
               .build();
       responseObserver.onNext(productResponse);
       responseObserver.onCompleted();
   }
}"
```

5.3 gRPC Server Configuration

```
@Configuration
public class GrpcConfig {


   @Bean
   public ManagedChannel managedChannel() {
       return ManagedChannelBuilder.forAddress("127.0.0.1", 9090)
               .usePlaintext()
               .build();
   }

   @Bean
   public ProductServiceGrpc.ProductServiceBlockingStub
productServiceBlockingStub(ManagedChannel managedChannel) {
       return ProductServiceGrpc.newBlockingStub(managedChannel());
   }
}
```

6. Product Catalog Service API Definition

6.1 Product Model

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String description;
    private Double price;
    private int quantity;
    // Getters and Setters
}
```

6.2 Product Repository

```
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {
}
```

6.3 Product Controller

```
@RestController
@RequestMapping("/api/products")
public class ProductController {

    @Autowired
    private ProductService productService;
    @GetMapping("/{id}")
    public ResponseEntity<Product> getProductById(@PathVariable Long id) {
        Product product = productService.getProductById(id);
        return ResponseEntity.ok(product);
    }

    @GetMapping
    public ResponseEntity<List<Product>> getAllProducts() {
        List<Product> products = productService.getAllProducts();
        return ResponseEntity.ok(products);
    }
}
```

7. Order Controller for gRPC Communication

```
@RestController
@RequestMapping("/api/orders")
public class OrderController {

    @GrpcClient("orderService")
    private OrderServiceGrpc.OrderServiceBlockingStub orderServiceStub;

    @GetMapping("/placeOrder")
    public ResponseEntity<String> placeOrder(@RequestParam("productId") int productId,
@RequestParam("quantity") int quantity) {
        OrderRequest request = OrderRequest.newBuilder()
                .setProductId(productId)
                .setQuantity(quantity)
```

```
            .build();

        OrderResponse response = orderServiceStub.createOrder(request);
        return ResponseEntity.ok(response.getMessage());
    }
}
```

8. Database Configuration (H2)

```
# H2 Database Configuration
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.h2.console.enabled=true
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update
```