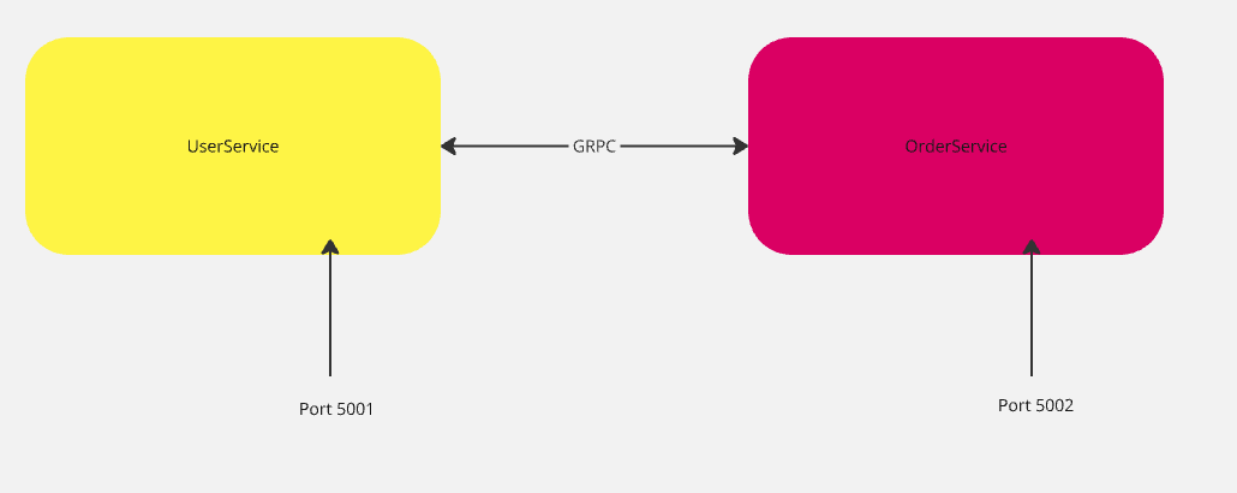
Разработка микросервисной архитектуры с использованием gRPC

Целью работы была разработка и развёртывание микросервисной архитектуры на основе Java, gRPC и Docker. Два микросервиса — UserService (управление пользователями) и OrderService (управление заказами) — были выделены в отдельные модули. Взаимодействие между ними происходит по gRPC-протоколу. Для удобного развертывания использовались Docker и Docker Compose.



- **UserService**: Управляет информацией о пользователях (CreateUser, GetUser).
- **OrderService**: Создаёт заказы и при необходимости может обращаться к UserService за информацией о пользователе.

Основные функции:

- UserService: CreateUser, GetUser.
- OrderService: CreateOrder, (опционально GetOrder).

В файлах .proto определены сервисы и сообщения:

UserService.proto:

```
syntax = "proto3";

package com.example.user;

option java_multiple_files = true;
option java_package = "com.example.userservice.proto";
option java_outer_classname = "UserServiceProto";

service UserService {
  rpc CreateUser(CreateUserRequest) returns (UserResponse);
  rpc GetUser(GetUserRequest) returns (UserResponse);
}

// Запрос на создание пользователя
message CreateUserRequest {
  string name = 1;
  string email = 2;
}

// Ответ с данными пользователя
message UserResponse {
  string user_id = 1;
  string name = 2;
  string email = 3;
}

// Запрос на получение данных пользователя
message GetUserRequest {
  string user_id = 1;
}
```

OrderService.proto:

```
syntax = "proto3";

package com.example.order;

option java_multiple_files = true;
option java_package = "com.example.orderservice.proto";
option java_outer_classname = "OrderServiceProto";

service OrderService {
  rpc CreateOrder(CreateOrderRequest) returns (OrderResponse);
  rpc GetOrder(GetOrderRequest) returns (OrderResponse);
}

// Запрос на создание пользователя
message CreateOrderRequest {
  string user_id = 1;
  double amount = 2;
}

// Ответ с данными пользователя
message OrderResponse {
  string order_id = 1;
  string user_id = 2;
  double amount = 3;
}

// Запрос на получение данных пользователя
message GetOrderRequest {
  string order_id = 1;
}
```

Для каждого сервиса написан Dockerfile

```dockerfile
# Используем Maven для сборки
FROM maven:3.9.4-eclipse-temurin-21 AS build
WORKDIR /app

# Копируем весь проект
COPY . .

# Переходим в папку order_service и собираем только этот модуль
WORKDIR /app/order_service
RUN mvn clean package -DskipTests

# Создаём лёгкий образ для запуска
FROM eclipse-temurin:21-jre
WORKDIR /app
COPY --from=build /app/order_service/target/order_service-1.0-SNAPSHOT.jar order_service.jar
EXPOSE 5002
ENTRYPOINT ["java", "-jar", "order_service.jar"]
```

```dockerfile
# Используем Maven для сборки
FROM maven:3.9.4-eclipse-temurin-21 AS build
WORKDIR /app
# Копируем весь проект
COPY . .
# Переходим в папку user_service и собираем только этот модуль
WORKDIR /app/user_service
RUN mvn clean package -DskipTests

# Создаём лёгкий образ для запуска
FROM eclipse-temurin:21-jre
WORKDIR /app
COPY --from=build /app/user_service/target/user_service-1.0-SNAPSHOT.jar user_service.jar
EXPOSE 5001
ENTRYPOINT ["java", "-jar", "user_service.jar"]
```

docker-compose.yml:

```yaml
version: '3.8'
services:
  user_service:
    build:
      context: .
      dockerfile: user_service/Dockerfile
    ports:
      - "5001:5001"

  order_service:
    build:
      context: .
      dockerfile: order_service/Dockerfile
    ports:
      - "5002:5002"
```

Тестирование через Postman (gRPC):