

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

Лабораторная работа 4

Выполнили:

Никитин Павел

Группа

K33402

Проверил:

Добряков Д. И.

Санкт-Петербург

2024 г.

## Задача

Необходимо упаковать ваше приложение в docker-контейнеры и обеспечить сетевое взаимодействие между различными частями вашего приложения, а также настроить общение микросервисов между собой посредством RabbitMQ. Делать это можно как с помощью docker-compose так и с помощью docker swarm. При разумном использовании swirl вы получите дополнительные баллы.

## Ход работы

Был использован docker-compose с образами postgres, prisma studio, rabbitmq а также образами полученными из микросервисов

```
1 version: '3.8'
2 services:
3   cart-service:
4     build:
5       context: .
6       dockerfile: apps/cart-service/Dockerfile
7     ports:
8       - '3001:3000'
9     networks:
10      - postgres
11
12   main-app:
13     build:
14       context: .
15       dockerfile: apps/main-app/Dockerfile
16     ports:
17       - '127.0.0.1:8000:8000'
18     networks:
19       - postgres
20     command: sh -c "yarn prisma migrate deploy && yarn nx run main-app:serve"
21
22   database:
23     image: postgres:16-alpine
24     environment:
25       - POSTGRES_USER=postgres
26       - POSTGRES_PASSWORD=123456789
27     ports:
28       - '9000:5432'
29     volumes:
30       - pgdata:/var/lib/postgresql/data
31     networks:
32       - postgres
33
34   prisma-studio:
35     container_name: prisma-studio
36     image: timothyjmilller/prisma-studio:latest
37     restart: unless-stopped
38     env_file:
39       - .env
40     ports:
41       - 5555:5555
42     networks:
43       - postgres
44
45   rabbitmq:
46     image: rabbitmq:3-management
47     ports:
48       - '5672:5672'
49       - '15672:15672'
50     networks:
51       - postgres
52
53 volumes:
54   pgdata:
55
56 networks:
57   postgres:
58     driver: bridge
59
```

Рисунок 1 - docker compose

Мы объединили все контейнеры в 1 сеть чтобы им было проще общаться  
Подключились через pgAdmin к базе данных запущенной в контейнере

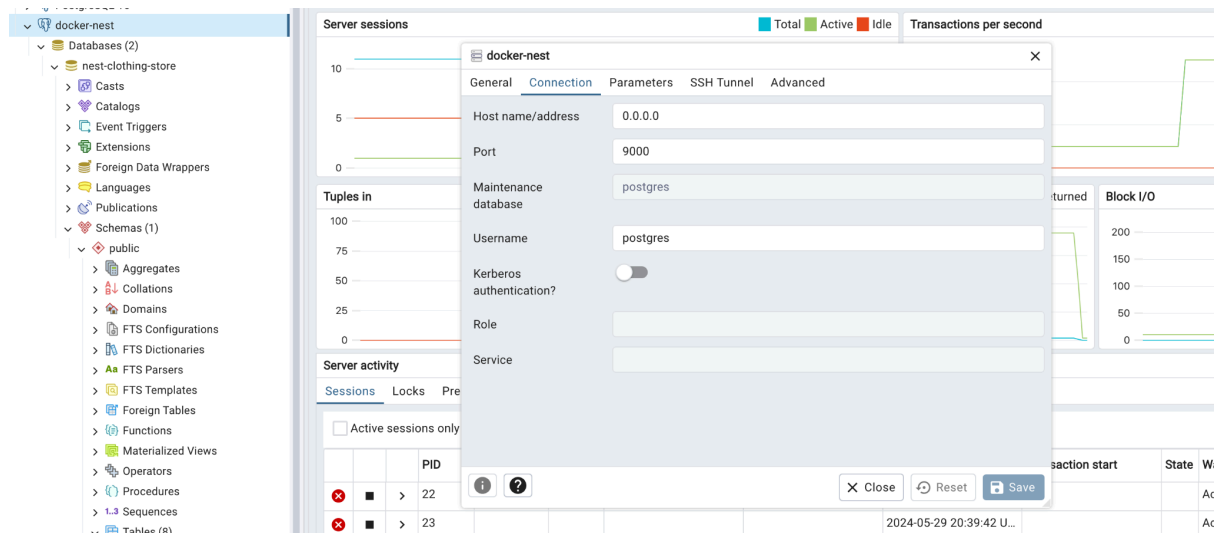


Рисунок 2 - PG admin

Также мы написали Dockerfile для каждого из сервисов

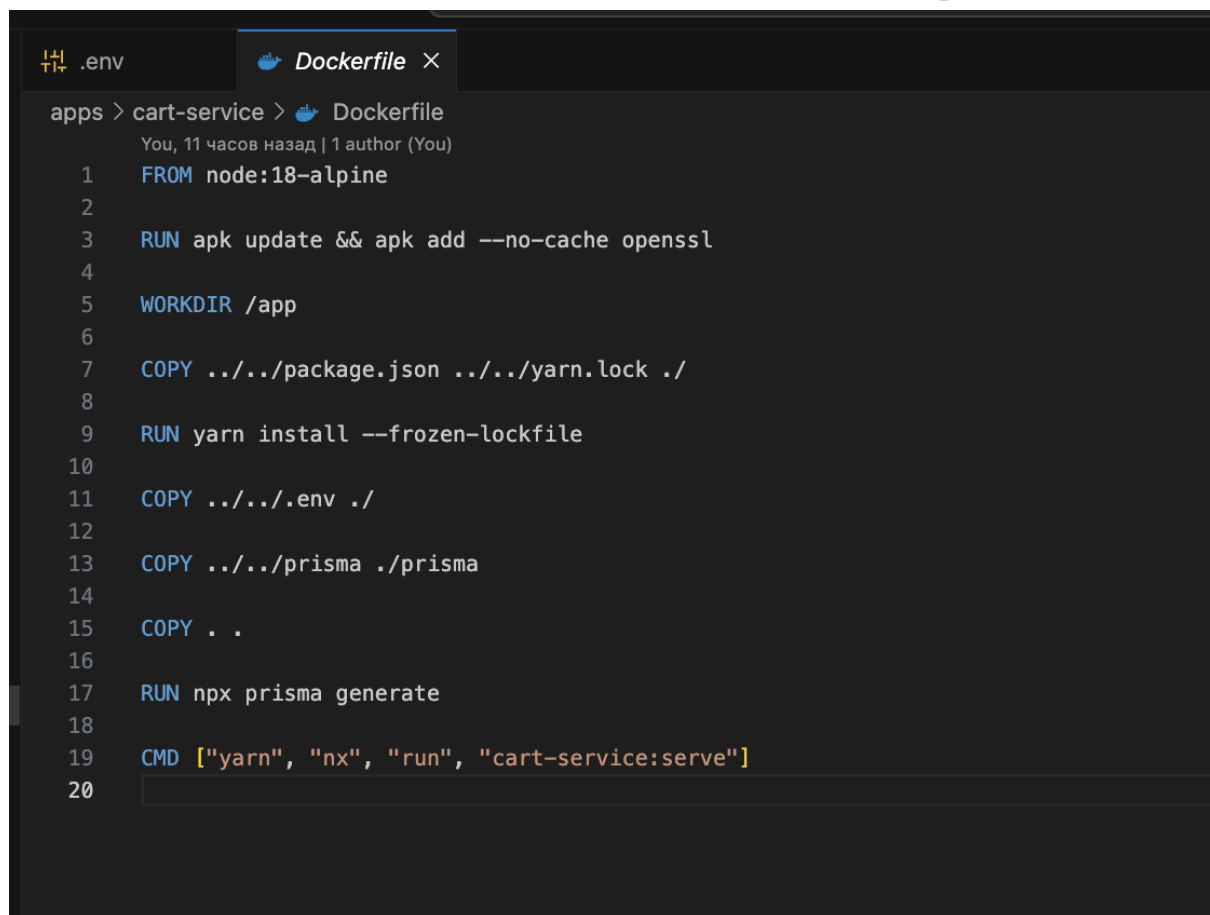


Рисунок 3 - Dockerfile для микросервисов

Добавили брокер сообщений в качестве транспорта между микросервисами:

```
async function bootstrap() {
  const app = await NestFactory.createMicroservice<MicroserviceOptions>(AppModule, {
    transport: Transport.RMQ,
    options: {
      urls: ['amqp://rabbitmq:5672'],
      queue: 'cart_service_queue',
      queueOptions: {
        durable: false,
      },
    },
  });

  await app.listen();
}
bootstrap();
```

Рисунок 4 - подключились к Rabbitmq

```
async function bootstrap() {
  const app = await NestFactory.create<NestExpressApplication>(AppModule);
  const config = app.get(ConfigService);
  const globalPrefix = 'api';

  app.useGlobalPipes(new ValidationPipe());
  app.setGlobalPrefix(globalPrefix);

  const swaggerConfig = new DocumentBuilder()
    .setTitle('nestjs-prisma-boilerplate')
    .setDescription('Store application')
    .setVersion('1.0')
    .addTag('auth')
    .addTag('user')
    .addBearerAuth()
    .build();
  const document = SwaggerModule.createDocument(app, swaggerConfig, {
    deepScanRoutes: true,
  });
  SwaggerModule.setup('api', app, document);

  app.connectMicroservice<MicroserviceOptions>({
    transport: Transport.RMQ,
    options: {
      urls: ['amqp://rabbitmq:5672'],
      queue: 'main_app_queue',
      queueOptions: {
        durable: false,
      },
    },
  });

  await app.listen(config.PORT);
  Logger.log(`🚀 Application is running on: http://localhost:${config.PORT}/${globalPrefix}`);
}

bootstrap();
```

Рисунок 5 -Также подключили main-app

```
canceled
+ nest-clothing-store docker compose up --build
[+] Building 42.9s (18/24)
=> [main-app internal] load build definition from Dockerfile
=> => transferring dockerfile: 364B
=> [main-app internal] load metadata for docker.io/library/node:18-alpine
=> [cart-service internal] load build definition from Dockerfile
=> => transferring dockerfile: 368B
=> [main-app internal] load .dockerignore
=> => transferring context: 2B
=> [cart-service internal] load .dockerignore
=> => transferring context: 2B
=> [cart-service 1/9] FROM docker.io/library/node:18-alpine@sha256:5069da655539e2e986ce3fd1757f24a41b846958566c89ff4a48874434d73749
=> [main-app internal] load build context
=> => transferring context: 4.35MB
=> [cart-service internal] load build context
=> => transferring context: 4.35MB
=> CACHED [cart-service 2/9] RUN apk update && apk add --no-cache openssl
=> CACHED [cart-service 3/9] WORKDIR /app
=> CACHED [cart-service 4/9] COPY ../../package.json ../../yarn.lock ./
=> CACHED [cart-service 5/9] RUN yarn install --frozen-lockfile
=> CACHED [cart-service 6/9] COPY ../../.env ./
=> CACHED [cart-service 7/9] COPY ../../prisma ./prisma
=> [cart-service 8/9] COPY . .
=> [cart-service 9/9] RUN npx prisma generate
=> [main-app] exporting to image
=> => exporting layers
=> => writing image sha256:ec6013d5184f81497fa6a1d2923a70dd5047a992a4f94b07128de4b61c51557
=> => naming to docker.io/library/nest-clothing-store-main-app
=> [cart-service] exporting to image
=> => exporting layers
=> => writing image sha256:959d77ec1bb710f3fd855b7603eebe5d466ba892dec54691af394bd9a148e51f
=> => naming to docker.io/library/nest-clothing-store-cart-service
[+] Running 5/5
✔ Container nest-clothing-store-database-1 Created
✔ Container prisma-studio Created
✔ Container nest-clothing-store-main-app-1 Recreated
✔ Container nest-clothing-store-cart-service-1 Recreated
```

Рисунок 6 - запустили build

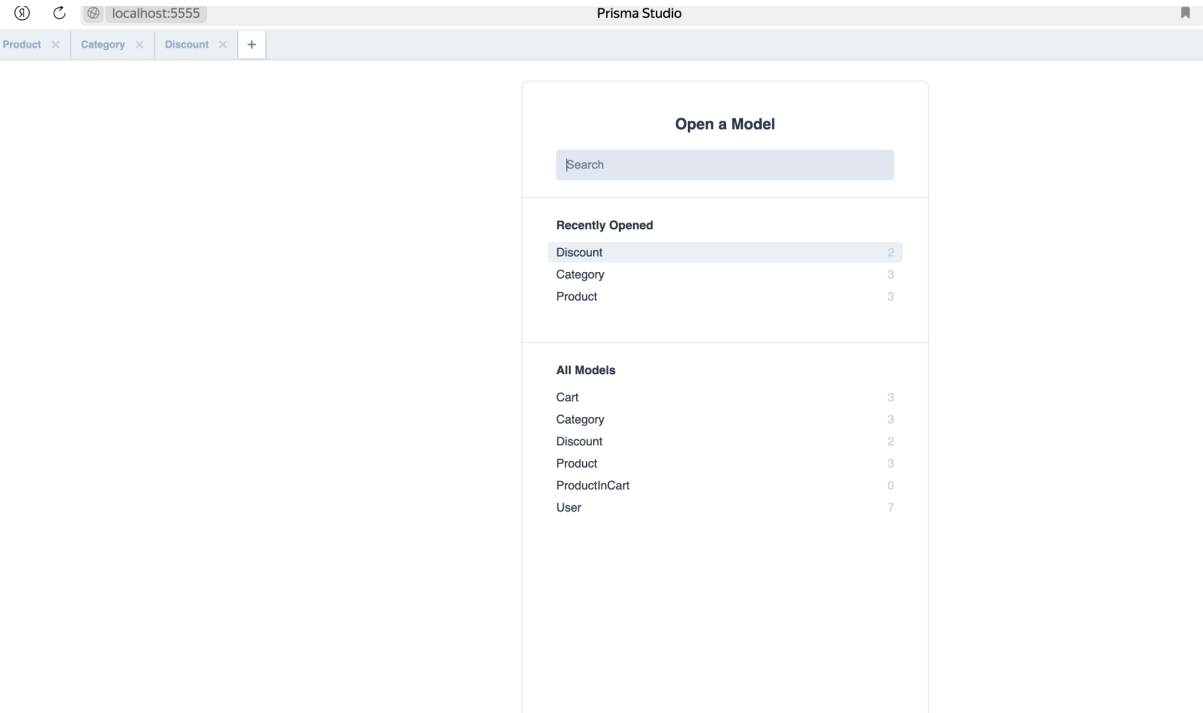


Рисунок 7 - подняли prisma

### Вывод

Научился работать с docker, docker compose узнал о брокерах сообщений, смог соединить это воедино, поднять и пробросить порты. Также научился пользоваться dockerhub и искать образы.