|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №8**

по дисциплине «Разработка клиент-серверных приложений»

**Студент группы** ИКБО-20-19 Анваржонов Ж. Т

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

**Руководитель практической работы** преподаватель Зарипов Е. В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Работа представлена «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Допущен к работе «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Москва 2022

**Выполнение работы**

Продемонстрируем реализацию сервера конфигурации с Git в качестве источника данных. Будем использовать Spring Cloud Config.

Добавим зависимости для подключения сервера конфигурации:

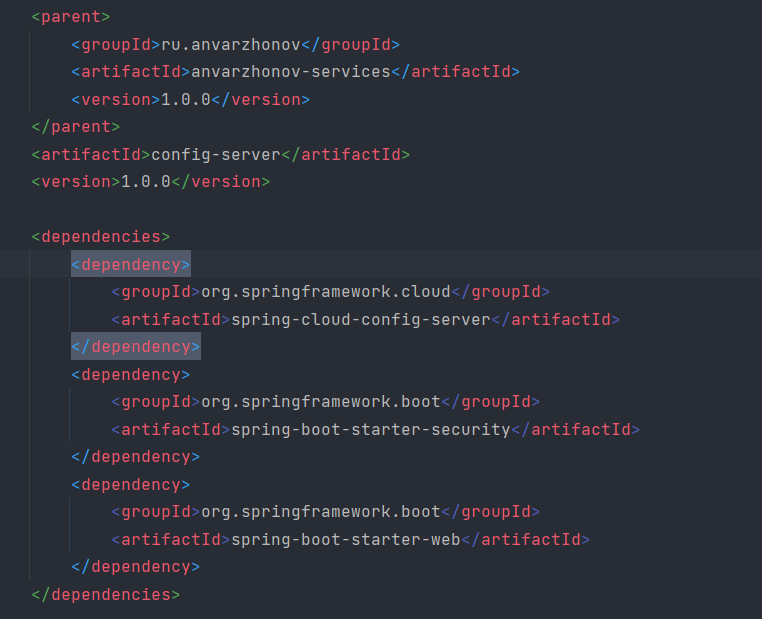


Рисунок 1 – Зависимости для сервера конфигурации

Все, что нужно сделать со стороны сервера конфигурации – включить аннотацию @EnableConfigServer в main классе:

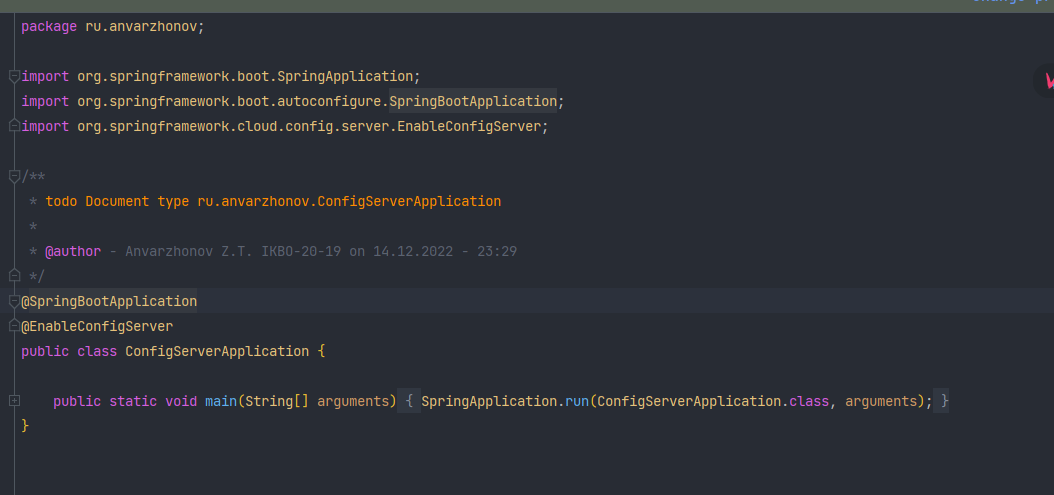


Рисунок 2 – Main класс с включенной аннотацией для включения сервера конфигурации

Добавим адрес гит url для подключения репозитория. Пока будем использовать локальный репозиторий:

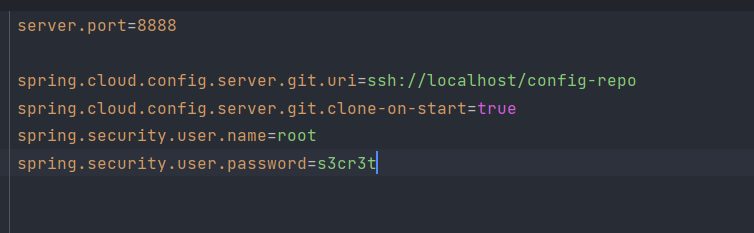


Рисунок 3 – Application.properties для config-server

Теперь чтобы получать конфигурацию из созданного сервера во всех сервисах в application.properties добавляем **spring.config.import**=<…> :

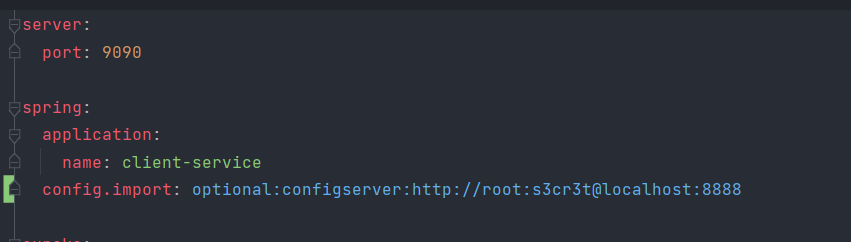


Рисунок 4 – Application.yml сервиса client-service

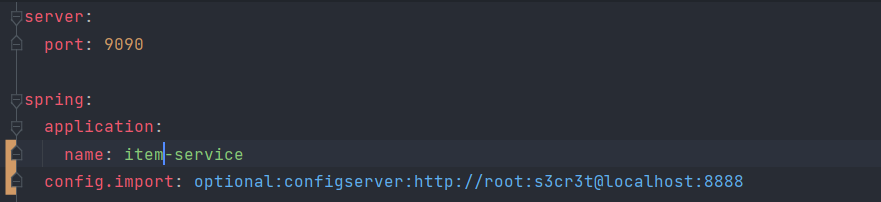


Рисунок 5 – Application.yml сервиса item-service

Продемонстрируем реализацию паттерна API Gateway c помощью Spring Cloud. Для этого добавим зависимость и укажем роуты к нашим сервисам:

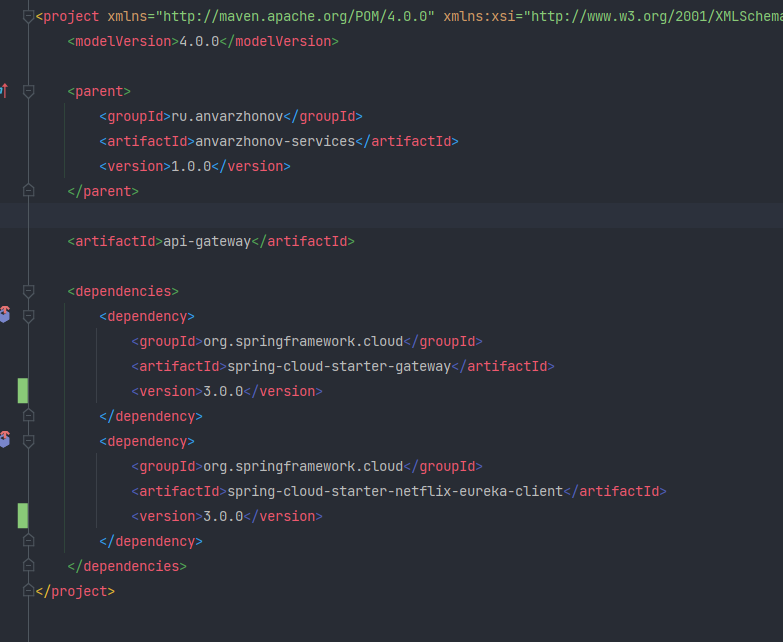


Рисунок 6 – Pom.xml api-gateway service

В следующем application.yml укажем роуты к нашим сервисам. В дальнейшем при настройки LoadBalancer’a поменяется адрес с localhost на lb, чтобы gateway сервис автоматически подбирал к какому экземпляру сервиса обращаться. Api gateway service будет работать на порту 8080 и будет единой точкой входа в нашу систему.

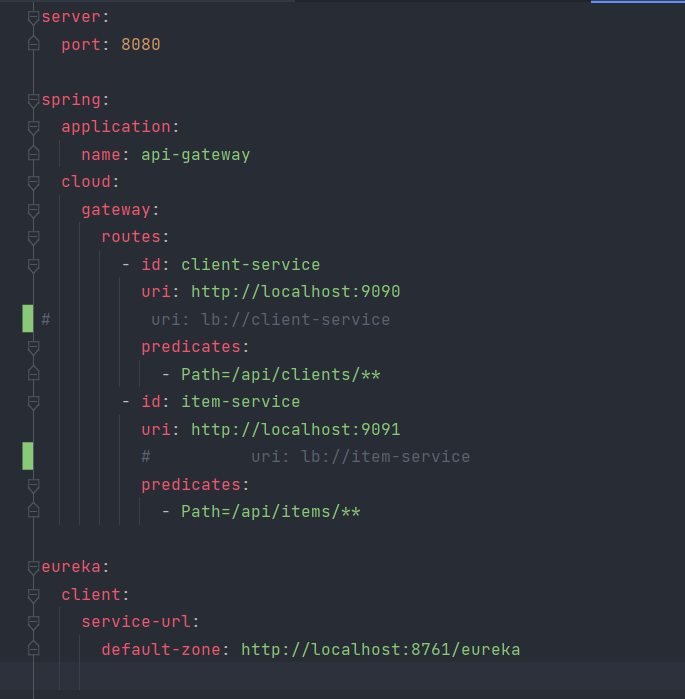


Рисунок 7 – Application.yml сервиса api-gateway

Продемонстрируем реализацию паттерна Service Discovery, который будет выявлять и регистрировать сервисы по заданному имени (spring.application.name). Реализацию будем делать с помощью Spring Cloud Eureka Server.

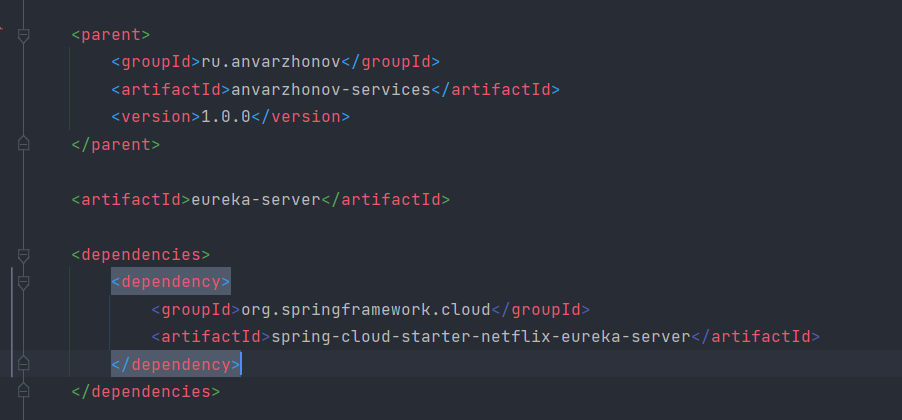


Рисунок 8 – Добавление зависимости для подключения eureka-server

Чтобы включить eureka сервер необходимо только включить аннотацию **EnableEurekaServer** со стороны сервера и **EnableEurekaClient** со стороны всех сервисов, которых хотим зарегистрировать

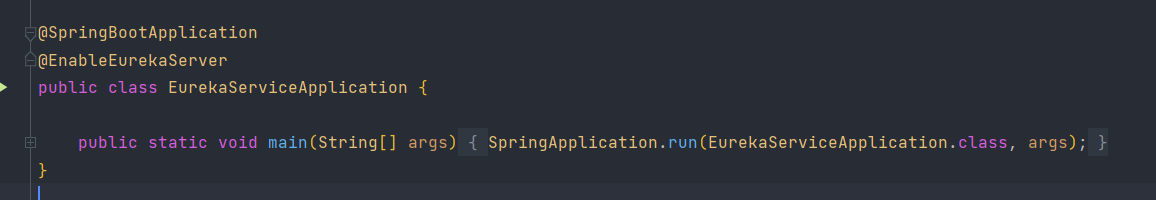


Рисунок 9 – Main класс с аннотацией enableEurekaServer

Запустим и сервер и покажем главную страницу, на которой пока нет зарегистрированных сервисов:

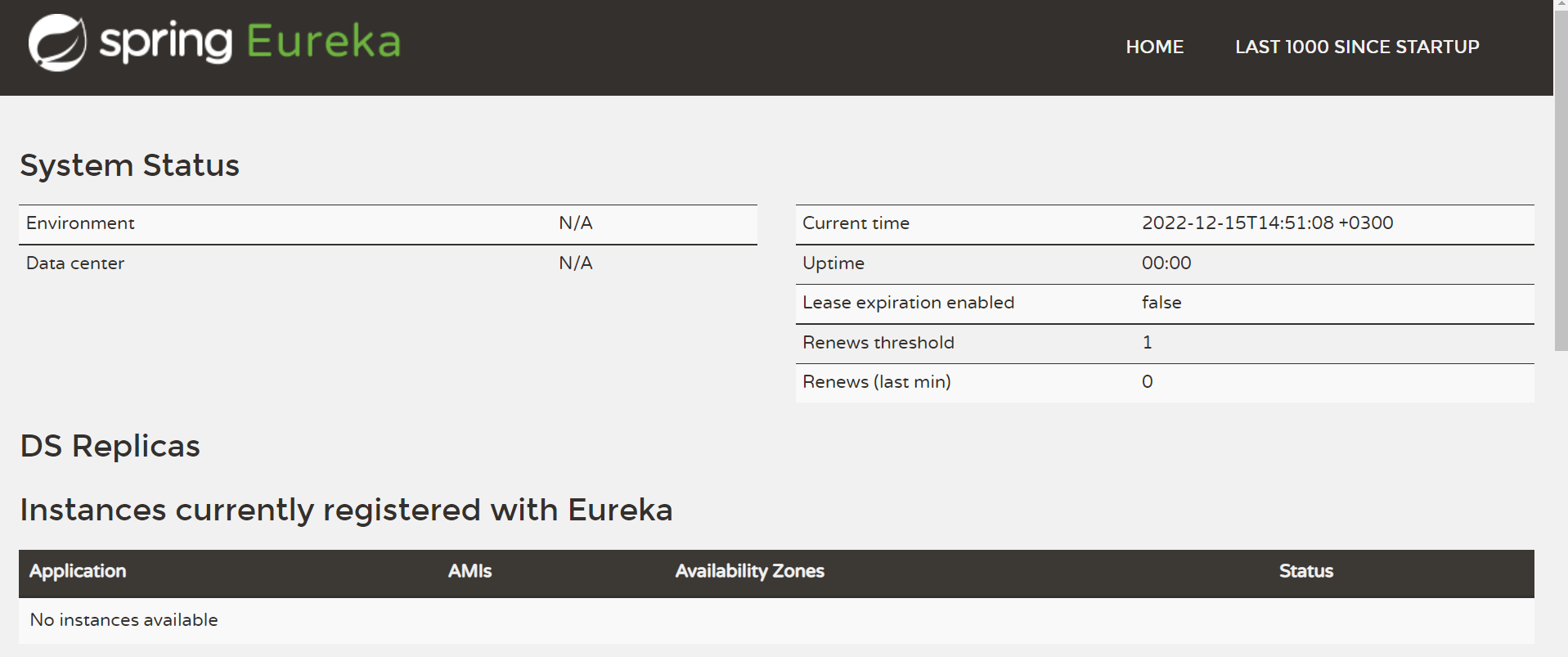


Рисунок 10 – Сервер spring eureka

Запустим наши микросервисы и продемонстрируем регистрации их в eureka server.

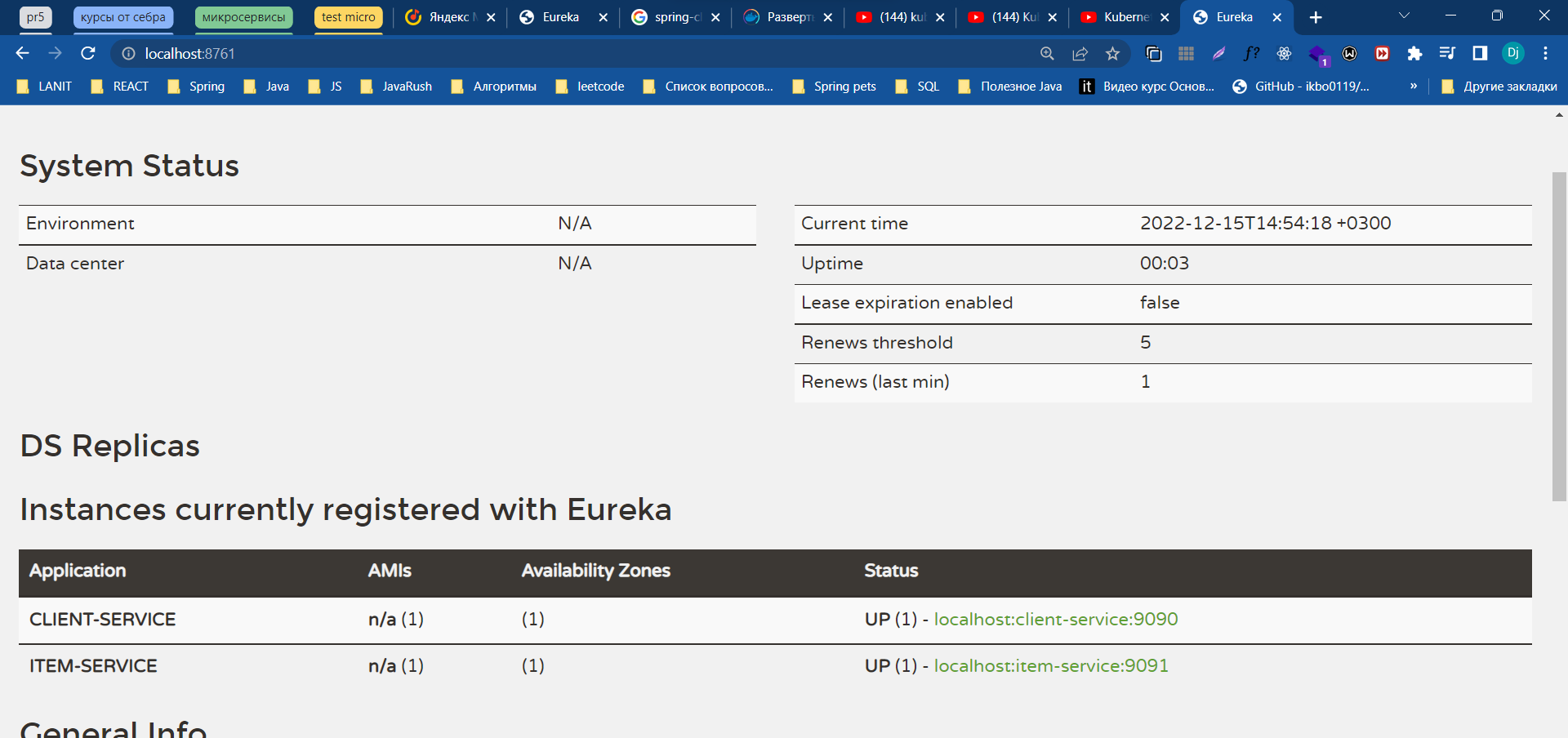


Рисунок 11 – Демонстрация зарегистрированных микросервисов в eureka server

Как видно после старта двух микросервисов в качестве инстансов появились экземпляры с названиями сервисом.

Запустим также api gateway:

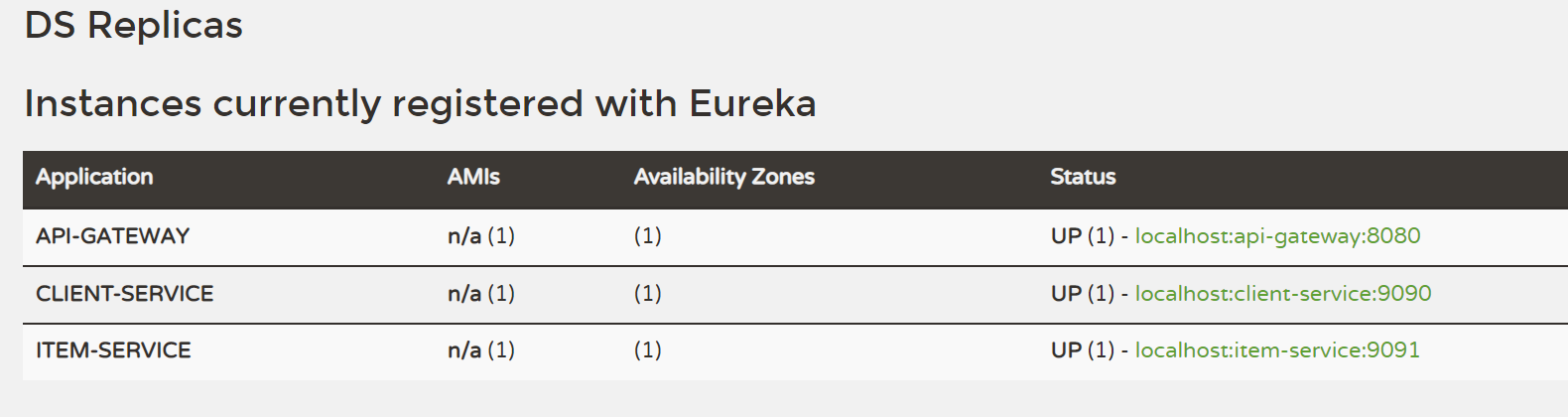


Рисунок 12 – Регистриция api gateway сервиса

Продемонстрируем реализацию балансировщика нагрузки. Балансировщик нагрузки будет настраивать на стороне сервиса api-gateway, который и будет выбирать к какому экземпляру сервиса направлять запрос



Рисунок 13 – Правки в application.yml сервиса api-gateway

Продемонстрируем взаимодействие между микросервисами с использованием FeignClient. Для этого в первую очередь добавляем новую зависимость openfeign во все наши микросервисы, которые общаются между собой:

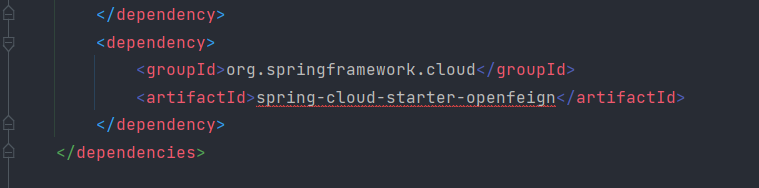


Рисунок 14 – Добавление зависимости openfeign

Создадим интерфейс ItemFeignClient в сервисе client-service, который будет обращаться по эндпоинту /api/items/findClientsItems чтобы получить список арендованных одним клиентом вещей.

Покажем обращение к feign клиенту в сервисе ClientService:

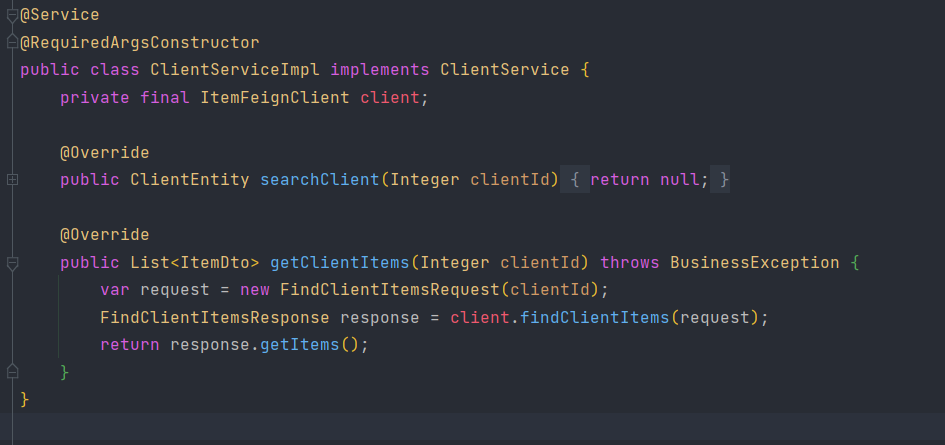


Рисунок 15 – Сервис получение списка арендованных вещей у клиента

Продемонстрируем взаимодействие каждого микросервиса с базой данных postgresql. Для этого в каждом микросервисе добавим настройку для подключения к БД:

В сервис client-service будем подключаться к БД client\_db, а в item-service к item\_db, таким образом каждый микросервис будет использовать отдельный экземпляр базы данных:

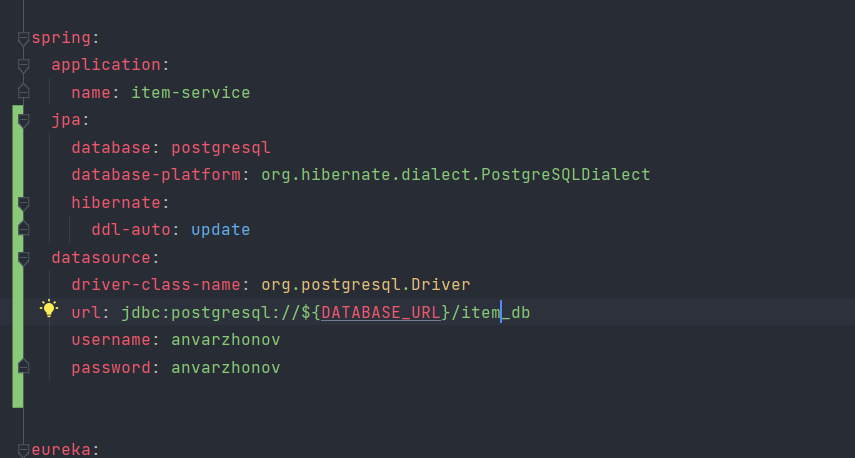


Рисунок 16 –Настройки подключения к бд

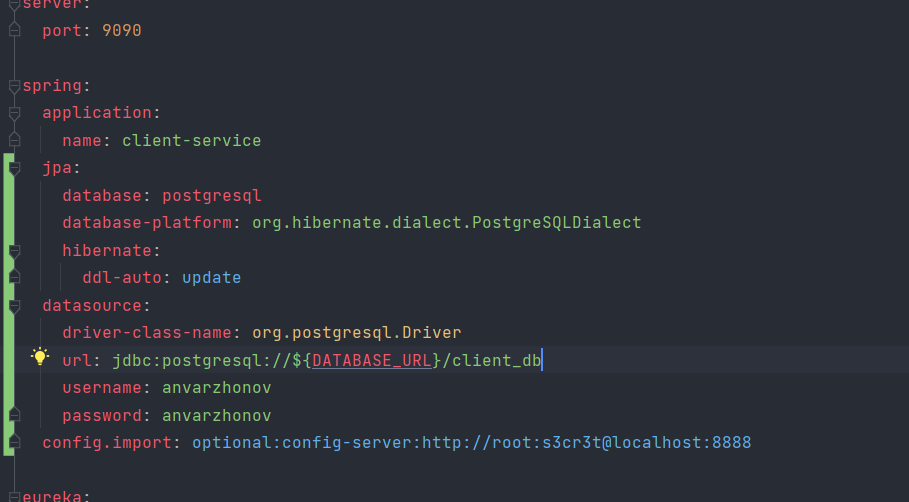


Рисунок 17 – Подключение к бд сервиса client\_db

Репозитории для работы с БД:

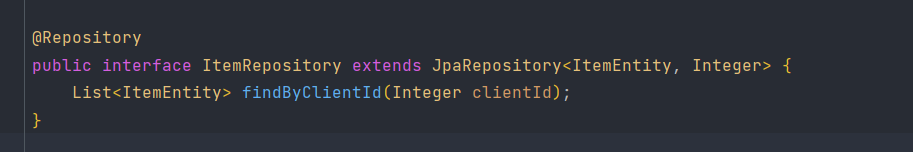


Рисунок 18 – Репозиторий для работы с БД item\_db

Запустим в docker-compose каждый экземпляр бд:

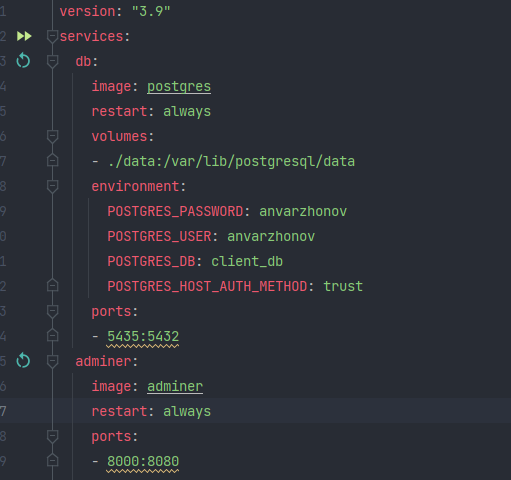


Рисунок 19 – Настройки docker-compose для БД

Демонстрация подключения к БД:

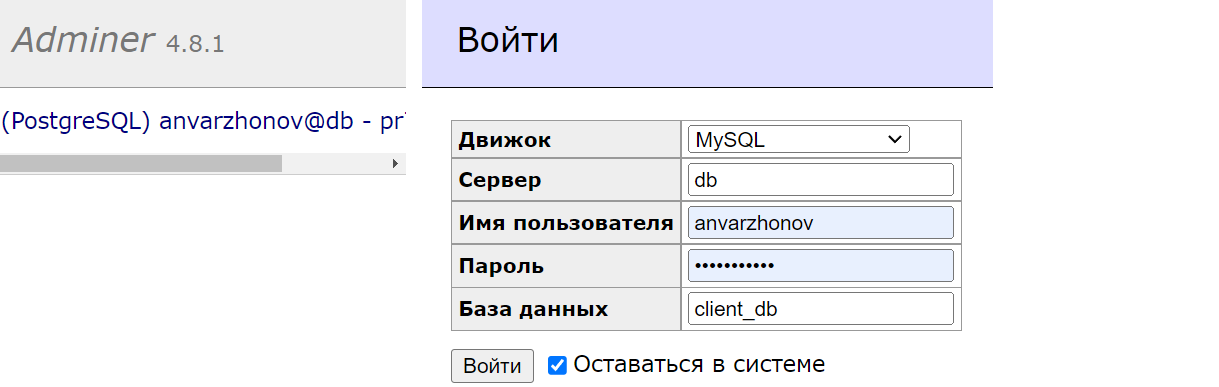


Рисунок 20 – Подключение к бд client\_db

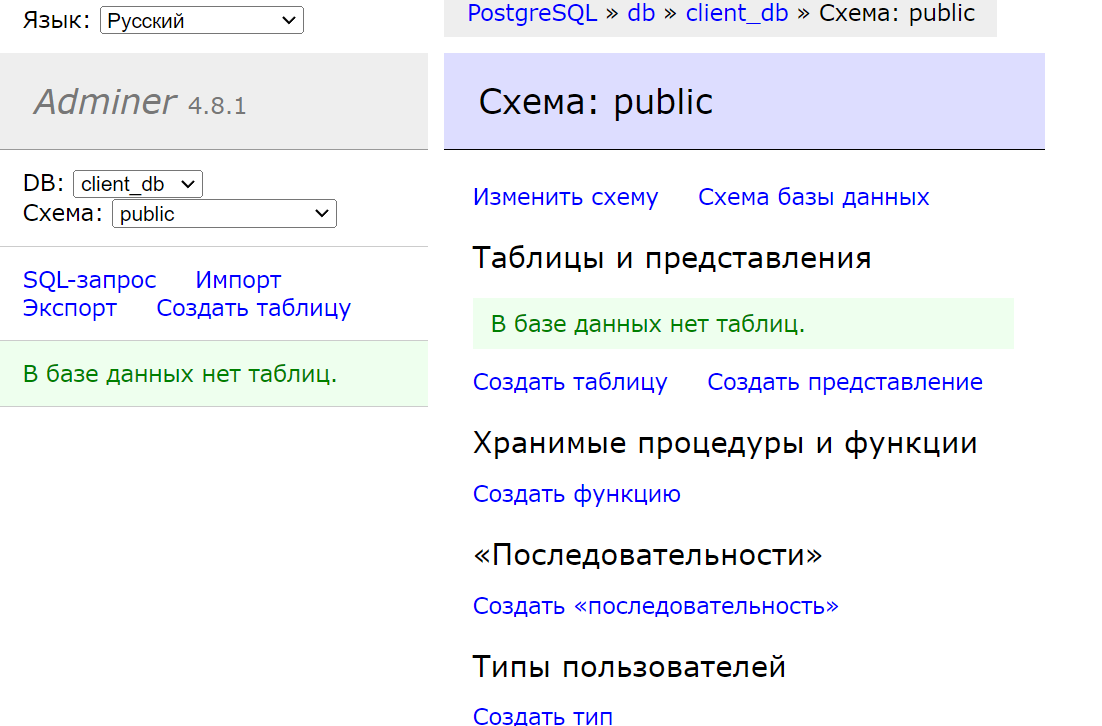


Рисунок 21 – База данных client\_db

Продемонстрируем файлы для развертывания сервисов в kubernetes. Для этого нам нужно для каждого сервиса создать deployment и service файлы.

1. Client-service

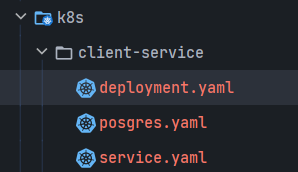


Рисунок 22 – Структура файлов

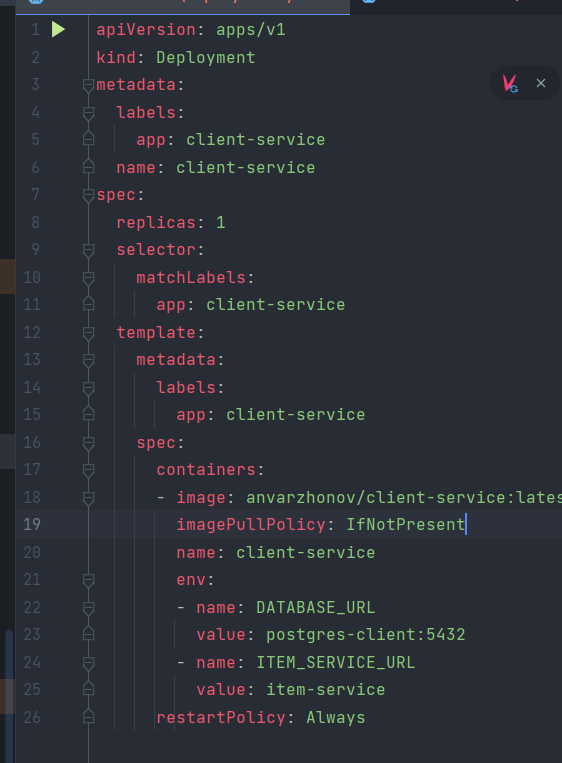


Рисунок 23 – Deployment.yml

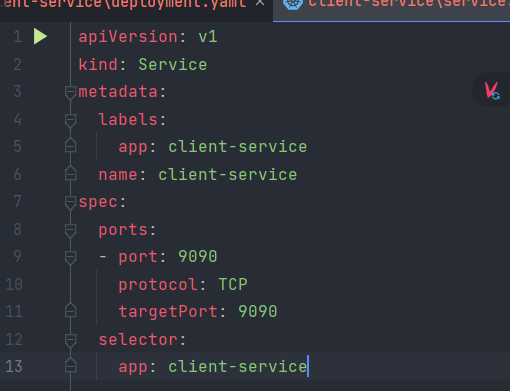


Рисунок 24 – Service.yaml

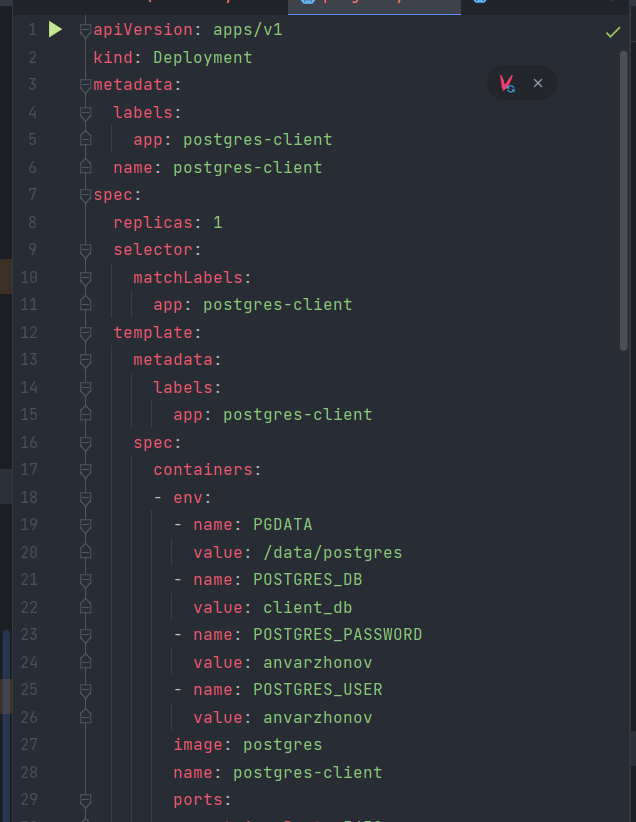


Рисунок 25 – Файл для развертывания Postgres

1. item-service

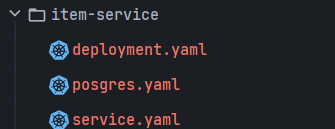


Рисунок 26 – Структура файлов



Рисунок 27 – Deployment.yml

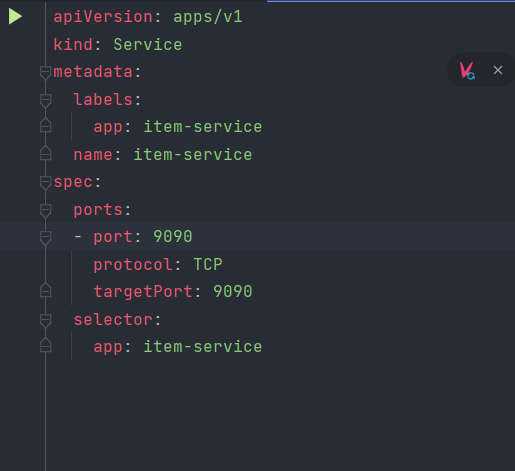


Рисунок 28 – Service.yaml

Соберем файлы с помощью kubectl, Рисунок 29.

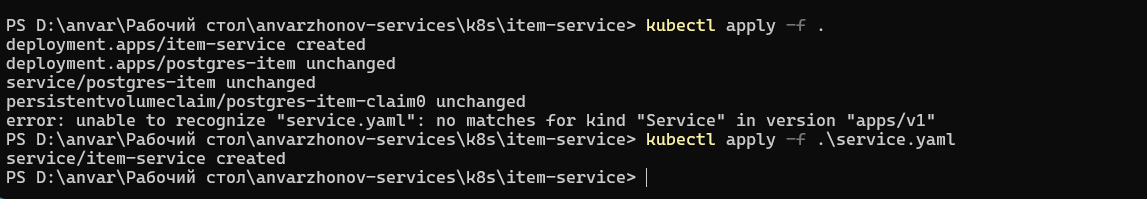


Рисунок 29 – Фрагмент kubectl

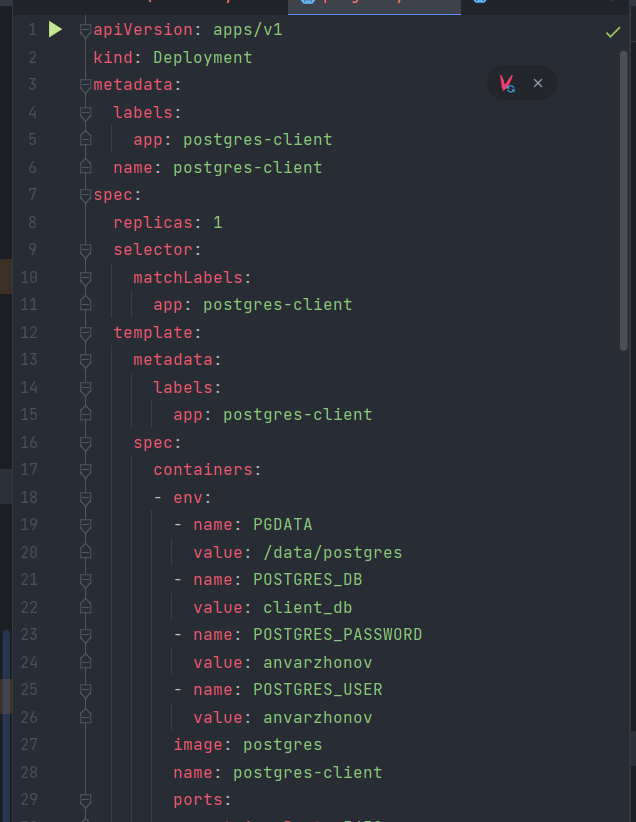


Рисунок 29 –Файл для развертывания Postgres

Продемонстрируем поднятые поды. На рисунке видно 2 микросервиса и 2 экземпляра для БД postgresql, Рисунок 30

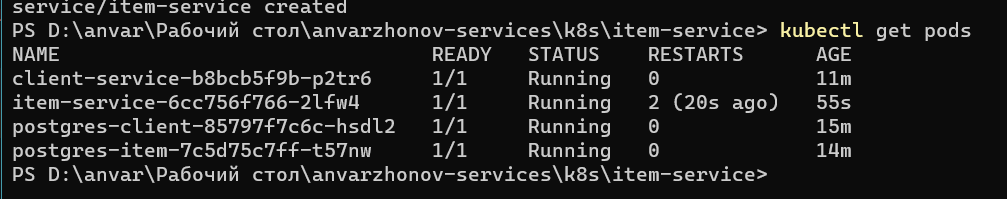


Рисунок 30 – Поднятые в kubernetes поды

Как видно все сервисы успешно запустились. Покажем еще объекты Service, Рисунок 31.

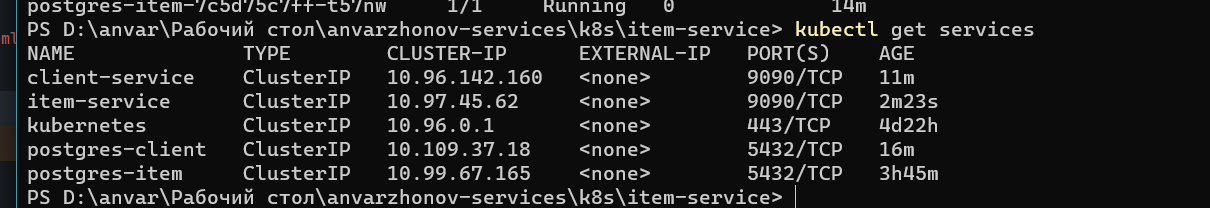


Рисунок 31 – services

**Вывод:**

В ходе практической работы была разработана и развернута в kubernetes микросервисная система предметной области аренда вещей с использованием инструментов Spring Cloud.