

Департамент образования города Москвы
Государственное автономное образовательное учреждение
высшего образования города Москвы
«Московский городской педагогический университет»

Институт цифрового образования
Департамент информатики, управления и технологий

ДИСЦИПЛИНА:

«Интеграция и развертывание программного обеспечения с помощью
контейнеров»

Лабораторная работа №3.1

Тема: «Docker Compose для мультиконтейнерных приложений»

Выполнила:

Студентка группы АДЭУ-211

Кравцова Алёна Евгеньевна

Руководитель:

Босенко Т.М

Москва

2024

Цель работы: освоить использование Docker Compose для управления многоконтейнерными приложениями.

Задачи:

1. Создать файл `docker-compose.yml` для указанного многоконтейнерного приложения.
2. Запустить приложение с помощью Docker Compose.
3. Проверить работоспособность приложения и взаимодействие между контейнерами.
4. Выполнить индивидуальное задание.

Индивидуальное задание:

st_93	Создать файл <code>docker-compose.yml</code> для системы учета сотрудников (Spring Boot + PostgreSQL).	Запустить приложение и проверить CRUD-операции.	Реализовать расчет средней зарплаты по отделам.
-------	--------------------------------------------------------------------------------------------------------	-------------------------------------------------	-------------------------------------------------

Шаг 1. Создание структуры проекта Spring Boot

Spring — это фреймворк для Java, на котором пишут веб-приложения и микросервисы. А Spring Boot — это расширение, которое упрощает и ускоряет работу со Spring. Оно представляет собой набор утилит, автоматизирующих настройки фреймворка.

Spring Boot разработан для ускорения создания веб-приложений. Он отличается от своего «родителя» тем, что не требует сложной настройки и имеет ряд встроенных инструментов, упрощающих написание кода.

Необходимо создать новый Spring Boot проект с зависимостями:

- Spring Web;
- Spring Data JPA;
- PostgreSQL Driver;

- Lombok;
- Micrometer Prometheus (для мониторинга).

Структура проекта:

employee-management/

- src/main/java/com/example/employees/

-- Employee.java

-- Department.java

-- EmployeeRepository.java

-- DepartmentRepository.java

-- EmployeeService.java

-- EmployeeController.java

-- DepartmentController.java

-- DataLoader.java

-- Application.java

- src/main/resources/application.yml

- pom.xml

- Dockerfile

- docker-compose.yml

- prometheus.yml

Структура представлена на рисунке 1.

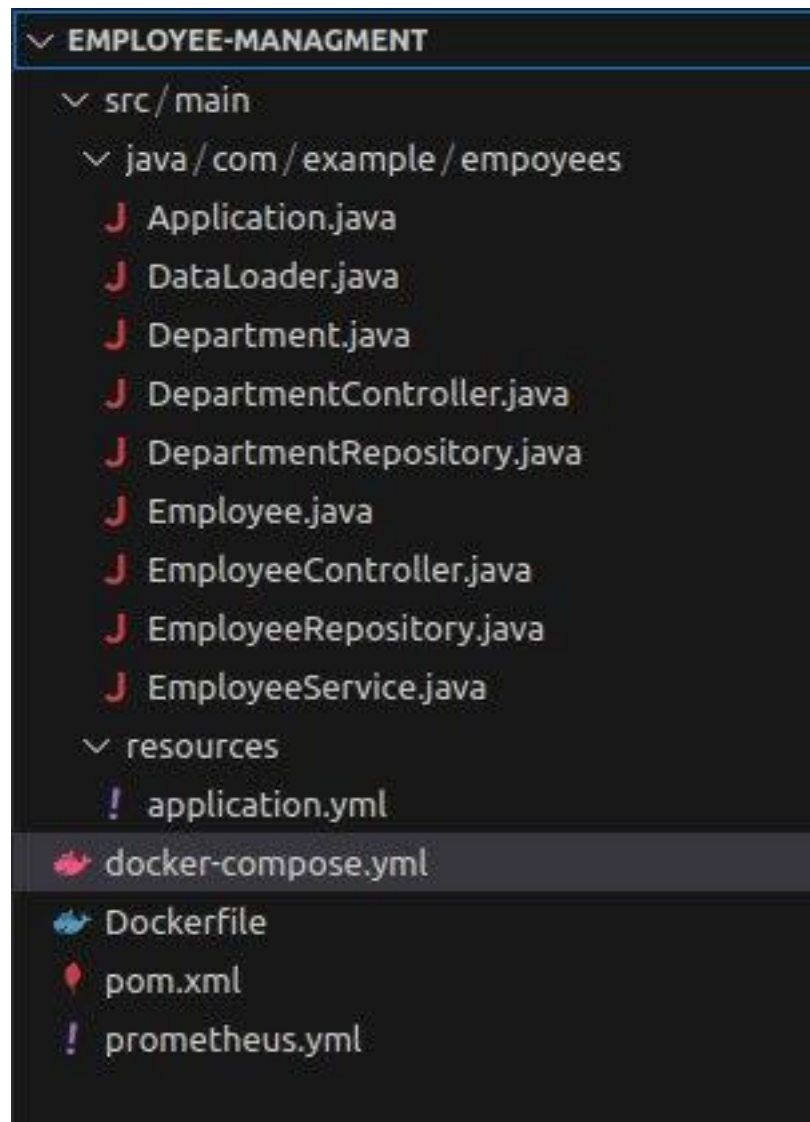


Рис. 1 – Структура проекта

Шаг 2. Создание приложения

Теперь подробнее рассмотрим каждый структурный компонент проекта.

1) Employee.java

Employee.java – модель данных сотрудника (Рис. 2).

```

package com.example.employees;

import jakarta.persistence.Entity;
import jakarta.persistence.Table;
import jakarta.persistence.Id;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.JoinColumn;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;

@Entity
@Table(name = "employees")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private Double salary;

    @ManyToOne
    @JoinColumn(name = "department_id")
    private Department department;
}

```

Рис. 2 – Employee.java

2) Department.java

Department.java – Модель данных отдела сотрудника (Рис. 3).

```

src > main > java > com > example > employees > Department.java
1  package com.example.employees;
2
3  import jakarta.persistence.Entity;
4  import jakarta.persistence.Table;
5  import jakarta.persistence.Id;
6  import jakarta.persistence.GeneratedValue;
7  import jakarta.persistence.GenerationType;
8  import lombok.Data;
9  import lombok.NoArgsConstructor;
10 import lombok.AllArgsConstructor;
11
12
13 @Entity
14 @Table(name = "departments")
15 @Data
16 @NoArgsConstructor
17 @AllArgsConstructor
18 public class Department {
19     @Id
20     @GeneratedValue(strategy = GenerationType.IDENTITY)
21     private Long id;
22     private String name;
23 }

```

Рис. 3 – Department.java

3) EmployeeRepository.java

Подготовим хранилище данных о сотрудниках (Рис. 4).

```
> main > java > com > example > employees > J EmployeeService.java
1 package com.example.employees;
2
3 import org.springframework.stereotype.Service;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import java.util.List;
6
7
8 @Service
9 public class EmployeeService {
10     @Autowired
11     private EmployeeRepository employeeRepository;
12
13     public List<Employee> getAllEmployees() {
14         return employeeRepository.findAll();
15     }
16
17     public List<Object[]> getAverageSalaryByDepartment() {
18         return employeeRepository.findAverageSalaryByDepartment();
19     }
20 }
```

Рис. 4 – EmployeeRepository.java

4) DepartmentRepository.java

Подготовим хранилище данных о сотрудниках (Рис. 5).

```
src > main > java > com > example > employees > J DepartmentRepository.java
1 package com.example.employees;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5
6 @Repository
7 public interface DepartmentRepository extends JpaRepository<Department, Long> {}
```

Рис. 5 – DepartmentRepository.java

5) EmployeeService.java

Сервис для получения всех сотрудников, а также расчета средней зарплаты по отделу (Рис. 6).

```

package com.example.employees;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class EmployeeService {
    @Autowired
    private EmployeeRepository employeeRepository;

    // Получение всех сотрудников
    public List<Employee> getAllEmployees() {
        return employeeRepository.findAll();
    }

    // Получение сотрудника по ID
    public Optional<Employee> getEmployeeById(Long id) {
        return employeeRepository.findById(id);
    }

    // Создание нового сотрудника
    public Employee createEmployee(Employee employee) {
        return employeeRepository.save(employee);
    }

    // Обновление существующего сотрудника
    public Employee updateEmployee(Long id, Employee employeeDetails) {
        Employee employee = employeeRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Employee not found for this id :: " + id));

        employee.setName(employeeDetails.getName());
        employee.setSalary(employeeDetails.getSalary());
        employee.setDepartment(employeeDetails.getDepartment());
        return employeeRepository.save(employee);
    }

    // Удаление сотрудника
    public void deleteEmployee(Long id) {
        Employee employee = employeeRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Employee not found for this id :: " + id));
        employeeRepository.delete(employee);
    }

    // Получение средней зарплаты по отделам
    public List<Object[]> getAverageSalaryByDepartment() {
        return employeeRepository.findAverageSalaryByDepartment();
    }
}

```

Рис. 6 – EmployeeService.java

6) EmployeeController.java

Создание API для приложения с целью получения данных о сотрудниках (Рис. 7).


```

package com.example.employees;

import org.springframework.web.bind.annotation.*;
import org.springframework.beans.factory.annotation.Autowired;
import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/employees")
public class EmployeeController {
    @Autowired
    private EmployeeService employeeService;

    // Получение всех сотрудников
    @GetMapping
    public List<Employee> getAllEmployees() {
        return employeeService.getAllEmployees();
    }

    // Получение сотрудника по ID
    @GetMapping("/{id}")
    public Optional<Employee> getEmployeeById(@PathVariable Long id) {
        return employeeService.getEmployeeById(id);
    }

    // Создание нового сотрудника
    @PostMapping
    public Employee createEmployee(@RequestBody Employee employee) {
        return employeeService.createEmployee(employee);
    }

    // Обновление существующего сотрудника
    @PutMapping("/{id}")
    public Employee updateEmployee(@PathVariable Long id, @RequestBody Employee employeeDetails) {
        return employeeService.updateEmployee(id, employeeDetails);
    }

    // Удаление сотрудника
    @DeleteMapping("/{id}")
    public void deleteEmployee(@PathVariable Long id) {
        employeeService.deleteEmployee(id);
    }

    // Получение средней зарплаты по отделам
    @GetMapping("/average-salary")
    public List<Object[]> getAverageSalaryByDepartment() {
        return employeeService.getAverageSalaryByDepartment();
    }
}

```

Рис. 7 – EmployeeController.java

7) DepartmentController.java

Создание API для приложения с целью получения данных об отделах (Рис. 8).


```

src > main > java > com > example > employees > J DepartmentController.java
1  package com.example.employees;
2
3  import org.springframework.web.bind.annotation.RestController;
4  import org.springframework.web.bind.annotation.RequestMapping;
5  import org.springframework.web.bind.annotation.GetMapping;
6  import org.springframework.beans.factory.annotation.Autowired;
7  import java.util.List;
8
9
10 @RestController
11 @RequestMapping("/departments")
12 public class DepartmentController {
13     @Autowired
14     private DepartmentRepository departmentRepository;
15
16     @GetMapping
17     public List<Department> getAllDepartments() {
18         return departmentRepository.findAll();
19     }
20 }

```

Рис. 8 – DepartmentController.java

8) DataLoader.java

DataLoader.java – файл, отвечающий за генерацию данных о сотрудниках (Рис. 9).

```

> main > java > com > example > employees > J DataLoader.java
1  package com.example.employees;
2
3  import org.springframework.boot.CommandLineRunner;
4  import org.springframework.stereotype.Component;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import java.util.Random;
7
8
9  @Component
10 public class DataLoader implements CommandLineRunner {
11     @Autowired
12     private EmployeeRepository employeeRepository;
13     @Autowired
14     private DepartmentRepository departmentRepository;
15
16     @Override
17     public void run(String... args) {
18         Department it = departmentRepository.save(new Department(null, "IT"));
19         Department hr = departmentRepository.save(new Department(null, "HR"));
20         Department finance = departmentRepository.save(new Department(null, "Finance"));
21
22         String[] names = {"Alice", "Bob", "Charlie", "David", "Eve", "Frank", "Grace", "Hank", "Ivy", "Jack",
23             "Kate", "Leo", "Mona", "Nina", "Oscar", "Paul", "Quinn", "Rachel", "Steve", "Tom",
24             "Uma", "Victor", "Wendy", "Xander", "Yvonne", "Zack", "Aaron", "Bella", "Chris", "Diana",
25             "Ethan", "Fiona", "George", "Hannah", "Isaac", "Julia", "Kevin", "Liam", "Megan", "Nathan",
26             "Olivia", "Peter", "Quincy", "Rose", "Sam", "Tina", "Ursula", "Vince", "Walter", "Xenia"};
27
28         Random random = new Random();
29         for (String name : names) {
30             Department department = random.nextBoolean() ? it : (random.nextBoolean() ? hr : finance);
31             double salary = 40000 + (random.nextDouble() * 60000); // Генерация зарплаты от 40к до 100к
32             employeeRepository.save(new Employee(null, name, salary, department));
33         }
34     }
35 }

```

Рис. 9 – DataLoader.java

9) Application.java

Application.java – файл, запускающий приложение (Рис. 10).

```

1  package com.example.employees;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6  @SpringBootApplication
7  public class Application {
8      public static void main(String[] args) {
9          SpringApplication.run(Application.class, args);
10     }
11 }
12

```

Рис. 10 – Application.java

10) application.yml

application.yml – файл, отвечающий за настройку приложения, а именно связывание его с БД и метриками Prometheus (Рис. 11).

```
1  spring:
2    datasource:
3      url: jdbc:postgresql://db:5432/employees_db
4      username: admin
5      password: admin
6    jpa:
7      hibernate:
8        ddl-auto: update
9      show-sql: true
10  management:
11    endpoints:
12      web:
13        exposure:
14          include: ["prometheus", "health", "metrics"]
15
16  metrics:
17    export:
18      prometheus:
19        enabled: true
20        distribution:
21          percentiles-histogram:
22            '[http.server.requests]': true
```

Рис. 11 – application.yml

11) pom.xml

pom.xml – файл зависимости приложения (Рис. 12).

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5    <modelVersion>4.0.0</modelVersion>
6
7    <parent>
8      <groupId>org.springframework.boot</groupId>
9      <artifactId>spring-boot-starter-parent</artifactId>
10     <version>3.2.2</version>
11     <relativePath />
12   </parent>
13
14   <groupId>com.example</groupId>
15   <artifactId>spring-boot-app</artifactId>
16   <version>1.0.0</version>
17   <packaging>jar</packaging>
18
19   <properties>
20     <java.version>17</java.version>
21     <lombok.version>1.18.30</lombok.version>
22     <postgresql.version>42.6.0</postgresql.version>
23   </properties>
24
25   <dependencies>
26     <!-- Spring Boot Starters -->
27     <dependency>
28       <groupId>org.springframework.boot</groupId>
29       <artifactId>spring-boot-starter-web</artifactId>
30     </dependency>
31
32     <dependency>

```

Рис. 12 – pom.xml

12) Dockerfile

Dockerfile – файл для конфигурирования приложения при запуске в Docker (Рис. 13).

```

Dockerfile
FROM maven:3.8.4-openjdk-17-slim AS build
WORKDIR /app
COPY . .
RUN mvn clean package -DskipTests

FROM openjdk:17-jdk-slim
WORKDIR /app
COPY --from=build /app/target/*.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]

```

Рис. 13 - Dockerfile

13) docker-compose.yml

Основной файл, который описывает 4-е контейнера и связь с БД, приложением, Prometheus и Grafana (Рис. 14).

```

version: '3.8'
services:
  db:
    image: postgres:15
    container_name: postgres_db
    restart: always
    environment:
      POSTGRES_DB: employees_db
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: admin
    ports:
      - "5432:5432"
    volumes:
      - pgdata:/var/lib/postgresql/data

  app:
    build: .
    container_name: spring_app
    depends_on:
      - db
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://db:5432/employees_db
      SPRING_DATASOURCE_USERNAME: admin
      SPRING_DATASOURCE_PASSWORD: admin
    ports:
      - "8080:8080"
    restart: always

  prometheus:
    image: prom/prometheus
    container_name: prometheus
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    ports:
      - "9090:9090"
    depends_on:
      - app

  grafana:
    image: grafana/grafana
    container_name: grafana
    ports:
      - "3000:3000"
    depends_on:
      - prometheus
    volumes:
      - grafana_data:/var/lib/grafana

volumes:
  pgdata:
  grafana_data:

```

Рис. 14 – docker-compose.yml

14) prometheus.yml

Файл конфигурации Prometheus с приложением (Рис. 15).

```
prometheus.yml
global:
  scrape_interval: 15s
scrape_configs:
  - job_name: 'spring'
    metrics_path: '/actuator/prometheus'
    static_configs:
      - targets: ['app:8080']
```

Рис. 15 – prometheus.yml

Шаг 3. Запуск приложения

Сделаем *Docker compose build up -d --build* для сборки контейнеров и запуска приложения (Рис. 16).

```
Compose now can delegate build to bake for better performances
Just set COMPOSE_BAKE=true
[*] building 119-de (14/14) FINISHED
-> [app internal] load build definition from Dockerfile
-> [app internal] load metadata for docker.io/library/openjdk:17-jdk-slim
-> [app internal] load metadata for docker.io/library/maven:3.8.4-openjdk-17-slim
-> [app internal] load .dockerignore
-> [app internal] transferring context: 28
-> [app build 1/4] FROM docker.io/library/openjdk:17-jdk-slim@sha256:150deb7b386bad085dcf6c781b9b9023a23096087b637c09a50c8019cab86f8
-> [app internal] load build context
-> [app internal] transferring context: 4.43kB
-> [app stage-1 1/3] FROM docker.io/library/openjdk:17-jdk-slim@sha256:150deb7b386bad085dcf6c781b9b9023a23096087b637c09a50c8019cab86f8
-> CACHED [app build 2/4] WORKDIR /app
-> [app build 3/4] COPY --from=build /app/target/*.jar app.jar
-> [app build 4/4] RUN mvn clean package -DskipTests
-> CACHED [app stage-1 2/3] WORKDIR /app
-> [app stage-1 3/3] COPY --from=build /app/target/*.jar app.jar
-> [app] exporting to image
-> [app] exporting layers
-> [app] writing image sha256:80ace0995839f270009fc1d70f51d750c5b094c4083e0e3e0c0ccaf99
-> [app] naming to docker.io/library/employee-management-app
-> [app] resolving provenance for metadata file
[*] Running 6/6
✔ app
✔ Network employee-managment_default
✔ Container employee-managment-prometheus-1
✔ Container postgres_db
✔ Container spring_app
✔ Container employee-managment-grafana-1
```

Рис. 16 – Сборка контейнера

Через *docker ps* посмотрим запущенные контейнеры (Рис. 17).

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
5e016d29e18	grafana/grafana	"/run.sh"	About a minute ago	Up About a minute	0.0.0.0:3000->3000/tcp, [::]:3000->3000/tcp	employee-managment-grafana-1
9171d51634e0	employee-managment-app	"java -jar app.jar"	About a minute ago	Up About a minute	0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp	spring_app
81bac74b40cf	postgres:15	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp	postgres_db
b36859edf2e8	prom/prometheus	"/bin/prometheus --c..."	About a minute ago	Up About a minute	0.0.0.0:9090->9090/tcp, [::]:9090->9090/tcp	employee-managment-prometheus-1

Рис. 17 – Просмотр всех запущенных контейнеров

Исходя из рисунка 17 видно, что все контейнеры успешно запущены и работают.

Шаг 4. Проверка CRUD операций

Read: выводим всех сотрудников (Рис. 19).

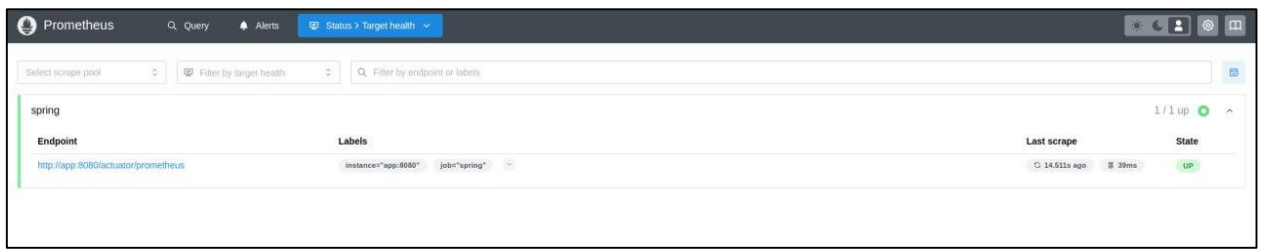


Рис. 22 – Проверка взаимодействия

Далее выведем количества HTTP-запросов, обработанных сервером, с момента его последнего запуска (Рис. 23).

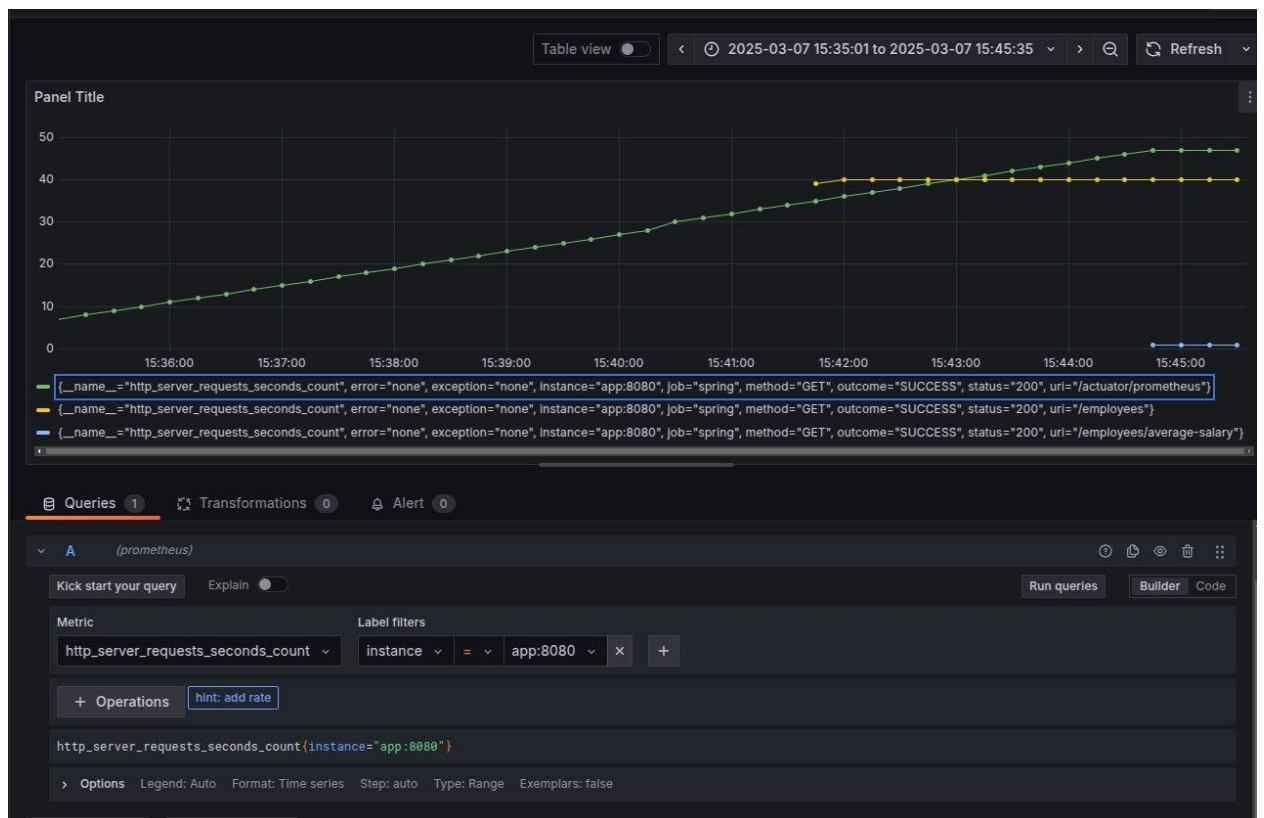


Рис. 23 – Количество HTTP-запросов

Далее с помощью Grafana выведем данные запроса `http://localhost:8080/employees/average-salary` по средней ЗП по отделу (Рис. 24).

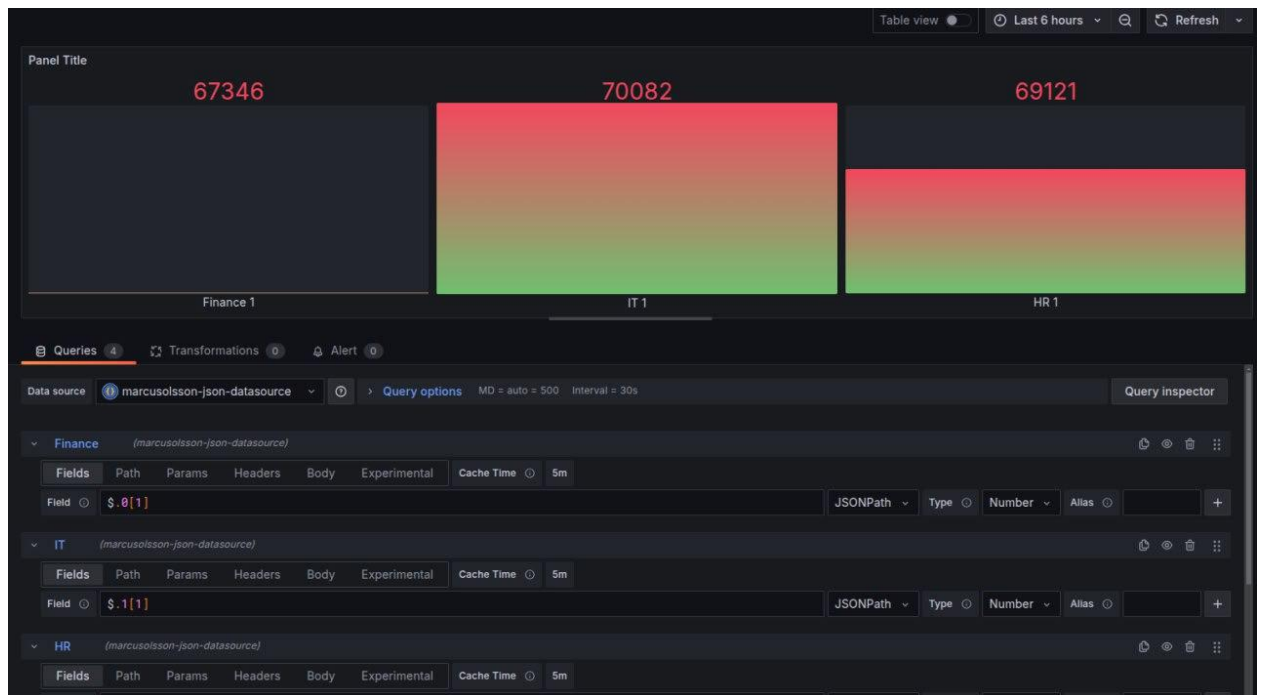


Рис. 24 – Дашборд о средней ЗП по отделу

Заключение: таким образом, в ходе выполнения работы было создано многоконтейнерное приложение на spring boot с 4-мя контейнерами. Были сгенерированы данные, успешно загружены в БД, а также подключен Prometheus и Grafana. Все задачи были выполнены.