|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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 №4**

по дисциплине «Технологии виртуализации клиент-серверных приложений»

**Студент группы** ИКБО-20-19 Московка А.А.

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

(подпись студента)

**Руководитель практической работы** старший преподаватель Волков М.Ю.

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

(подпись руководителя)

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

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

Москва 2022

**Задание**

Вам необходимо создать Spring Boot сервис и произвести мониторинг его работы. В работе должны быть отражены все пункты создания docker-compose файла, проверена работоспособность сервиса и систем мониторинга. Произведена нагрузка на систему с целью проверки работоспособности. Все пункты должны быть отражены в отчете в формате снимков экрана.

Требования к системе:

1. Наличие сервера с CRUD набором для взаимодействия с базой данных и выгрузкой данных с GrayLog по эндпоинту
   1. CRUD набор должен содержать эндпоинты для добавления, обновления, чтения и удаления записей
   2. Количество используемых моделей данных должно быть не менее 3, а также содержать как минимум 1 связь 1-N, 1 связь N-N.
   3. При обращении к эндпоинту логов он должен выгружать данные в формате csv из GrayLog
2. PostgreSQL в качестве СУБД с которой будет производиться взаимодействие/сниматься метрики и логи
3. Zabbix для мониторинга базовой работоспособности сервера
4. Prometheus для сбора данных с PostgreSQL, Grafana для вывода этих данных
5. GrayLog для сбора данных с PostgreSQL
6. Adminer для управления базой данных PostgreSQL

**Ход работы**

Создадим Dockerfile CRUD приложения (Листинг 1).

Листинг 1 – Dockerfile

FROM openjdk:17.0.2

ADD app/target/app-0.0.1-SNAPSHOT.jar app.jar

ENTRYPOINT ["java", "-jar","app.jar"]

EXPOSE 8080

Также создадим docker-compose.yml файл (Листинг 2).

Листинг 2 - docker-compose.yml

version: '3.1'

services:

crud-app:

image: 'pr4app'

ports:

- 8080:8080

depends\_on:

postgres:

condition: service\_healthy

environment:

- SPRING\_DATASOURCE\_URL=jdbc:postgresql://postgres:5432/postgres

- SPRING\_DATASOURCE\_USERNAME=postgres

- SPRING\_DATASOURCE\_PASSWORD=password

- SPRING\_JPA\_HIBERNATE\_DDL\_AUTO=update

postgres:

image: postgres:14.5

ports:

- 5432:5432

environment:

- POSTGRES\_PASSWORD=password

- POSTGRES\_USER=postgres

- POSTGRES\_DB=postgres

healthcheck:

test: ["CMD-SHELL", "pg\_isready -U postgres"]

interval: 10s

timeout: 5s

retries: 5

adminer:

image: adminer

restart: always

ports:

- 8000:8080

depends\_on:

postgres:

condition: service\_healthy

grafana:

image: grafana/grafana

ports:

- 8002:3000

prometheus:

image: prom/prometheus

ports:

- 9090:9090

volumes:

- ./prometheus.yml:/etc/prometheus/prometheus.yml:ro

postgres-exporter:

image: prometheuscommunity/postgres-exporter

ports:

- 9187:9187

environment:

DATA\_SOURCE\_NAME: "postgresql://postgres:password@postgres:5432/postgres?sslmode=disable"

links:

- postgres

- prometheus

zabbix-agent:

image: zabbix/zabbix-agent

depends\_on:

- postgres

ports:

- 10050:10050

links:

- zabbix-server

zabbix-front:

image: zabbix/zabbix-web-nginx-pgsql

environment:

- ZBX\_SERVER\_HOST=zabbix-server

- DB\_SERVER\_HOST=postgres

- POSTGRES\_USER=postgres

- POSTGRES\_PASSWORD=password

- POSTGRES\_DB=postgres

depends\_on:

- postgres

ports:

- 8001:8080

zabbix-server:

image: zabbix/zabbix-server-pgsql

environment:

- DB\_SERVER\_HOST=postgres

- POSTGRES\_USER=postgres

- POSTGRES\_PASSWORD=password

- POSTGRES\_DB=postgres

depends\_on:

- postgres

ports:

- 10051:10051

Соберем образы и запустим контейнера (Рисунок 1).



Рисунок 1 – Демонстрация сборки образа

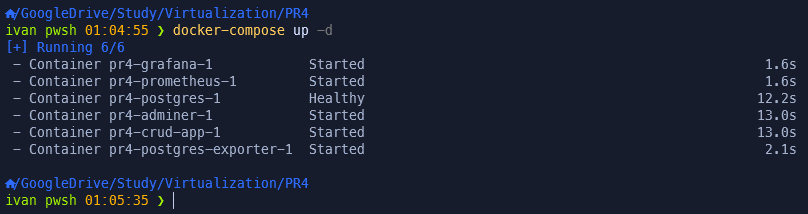


Рисунок 2 – Демонстрация запуска контейнеров

Проверим работу CRUD приложения, создадим нового пользователя (Рисунок 3).

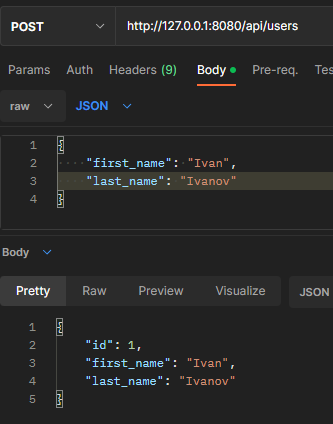


Рисунок 3 – Демонстрация работы CRUD приложения

Выведем список всех пользователей (Рисунок 4).

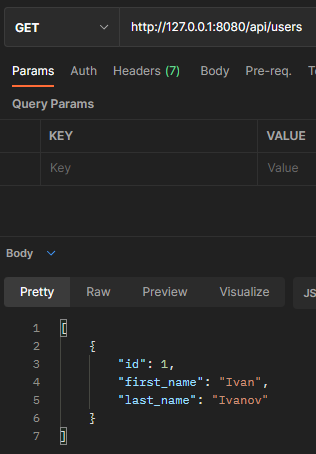


Рисунок 4 – Демонстрация работы CRUD приложения

Получим пользователя по его ID (Рисунок 5).

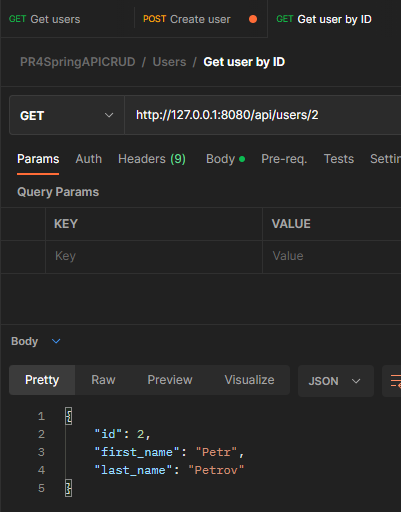


Рисунок 5 – Демонстрация работы CRUD приложения

Изменим информацию о пользователе по его ID (Рисунок 6, Рисунок 7)

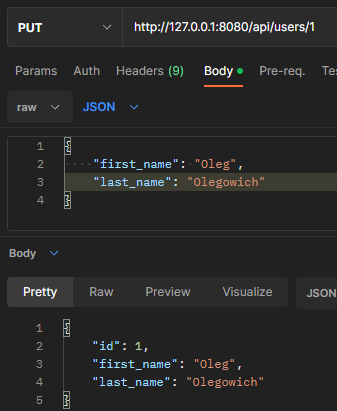


Рисунок 6 – Демонстрация работы CRUD приложения

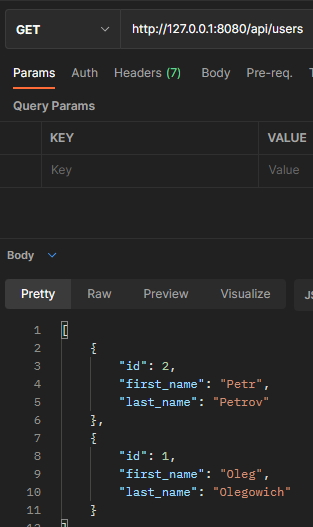


Рисунок 7 – Демонстрация работы CRUD приложения

Удалим пользователя по его ID (Рисунок 8, Рисунок 9).

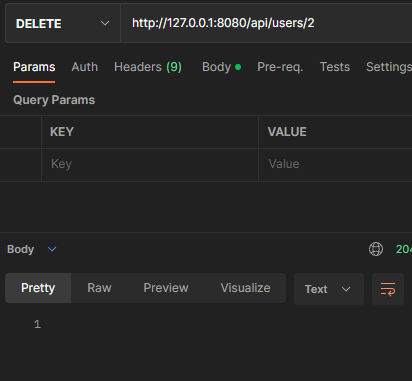


Рисунок 8 – Демонстрация работы CRUD приложения

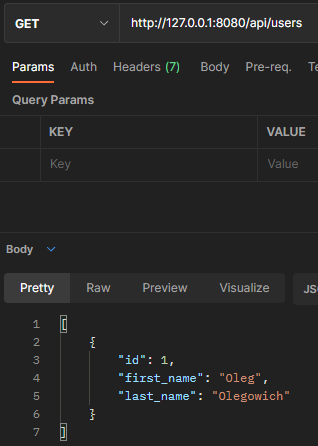


Рисунок 9 – Демонстрация работы CRUD приложения

Создадим новый пост (Рисунок 10).

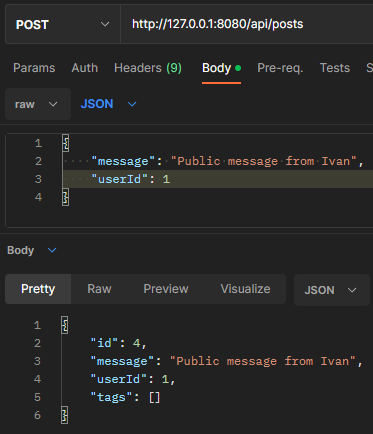


Рисунок 10 – Демонстрация работы CRUD приложения

Выведем список всех постов (Рисунок 11).

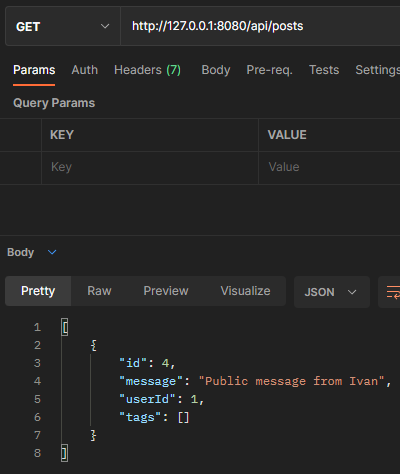


Рисунок 11 – Демонстрация работы CRUD приложения

Получим пост по его ID (Рисунок 12).

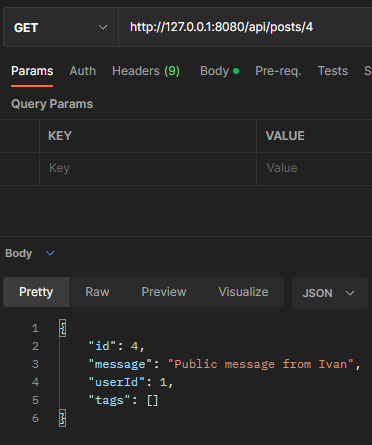


Рисунок 12 – Демонстрация работы CRUD приложения

Изменим пост по его ID (Рисунок 13, Рисунок 14).

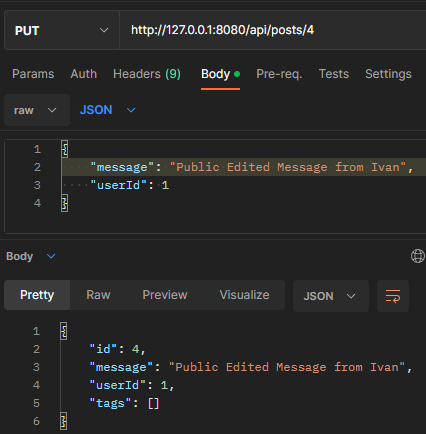


Рисунок 13 – Демонстрация работы CRUD приложения

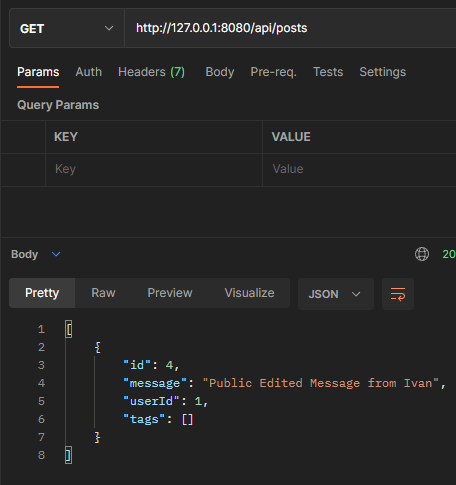


Рисунок 14 – Демонстрация работы CRUD приложения

Удалим пост по его ID (Рисунок 15, Рисунок 16).

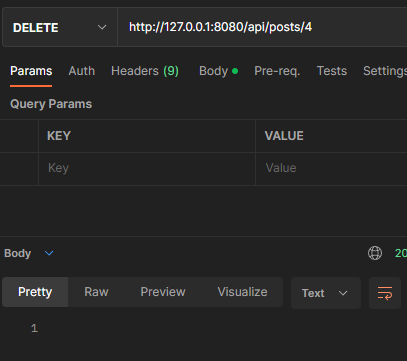


Рисунок 15 – Демонстрация работы CRUD приложения

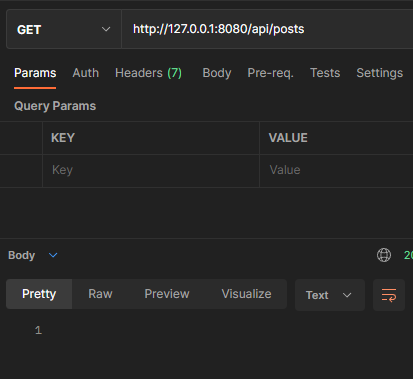


Рисунок 16 – Демонстрация работы CRUD приложения

Создадим новые теги и выведем их(Рисунок 17, Рисунок 18, Рисунок 19).

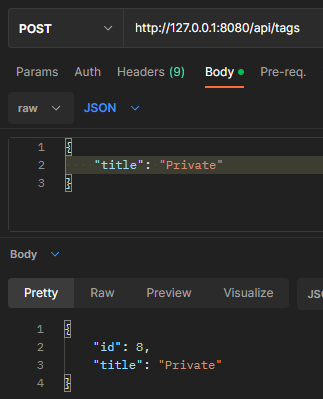


Рисунок 17 – Демонстрация работы CRUD приложения

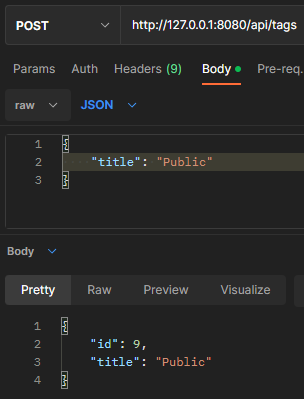


Рисунок 18 – Демонстрация работы CRUD приложения

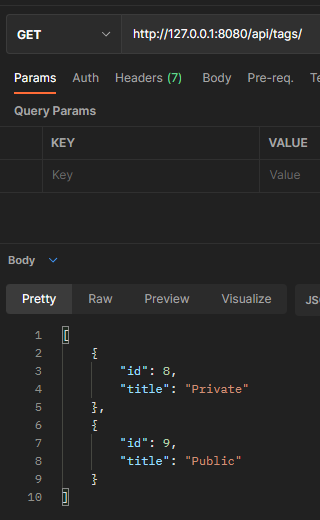


Рисунок 19 – Демонстрация работы CRUD приложения

Выведем тег по его ID (Рисунок 20).

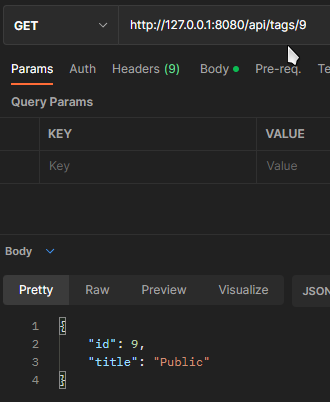


Рисунок 20 – Демонстрация работы CRUD приложения

Изменим тег по его ID (Рисунок 21, Рисунок 22).

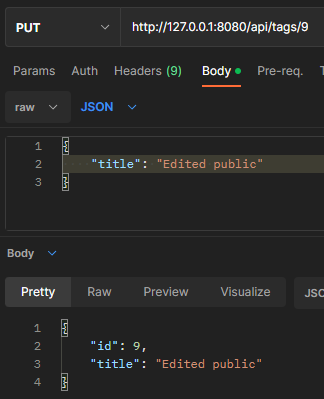


Рисунок 21 – Демонстрация работы CRUD приложения

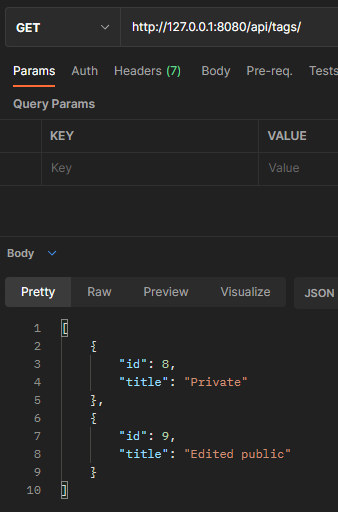


Рисунок 22 – Демонстрация работы CRUD приложения

Удалим тег по его ID (Рисунок 23, Рисунок 24).

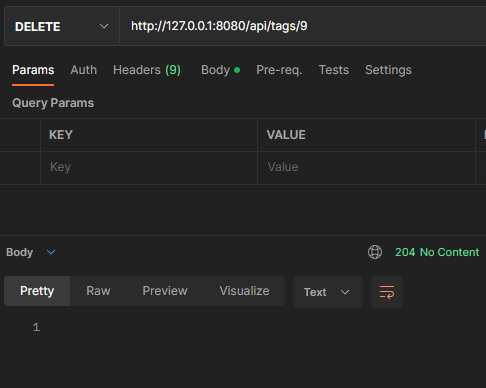


Рисунок 23 – Демонстрация работы CRUD приложения

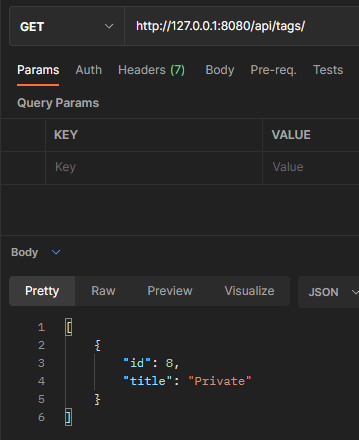


Рисунок 24 – Демонстрация работы CRUD приложения

Добавим тег к посту (Рисунок 25, Рисунок 26).

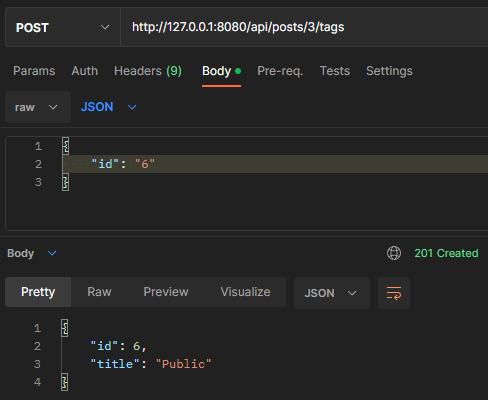


Рисунок 25 – Демонстрация работы CRUD приложения

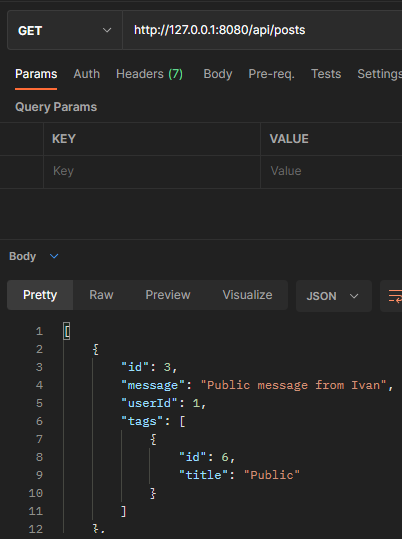


Рисунок 26 – Демонстрация работы CRUD приложения

Удалим тег с поста (Рисунок 27, Рисунок 28).

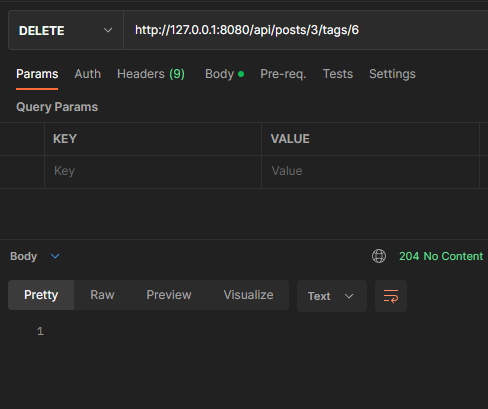


Рисунок 27 – Демонстрация работы CRUD приложения

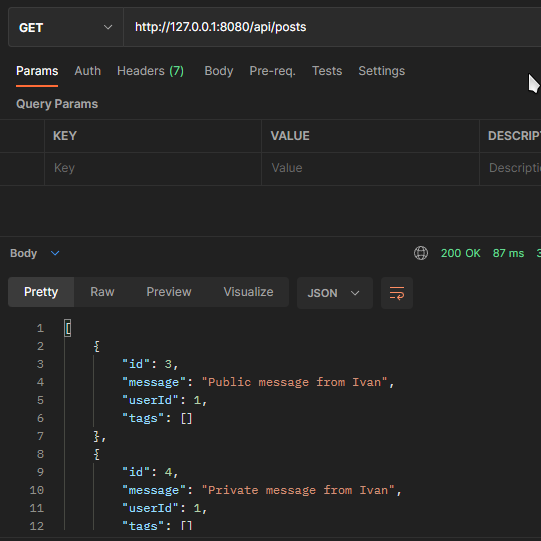


Рисунок 28 – Демонстрация работы CRUD приложения

Откроем панель Adminer и проверим его работу (Рисунок 29, Рисунок 30).

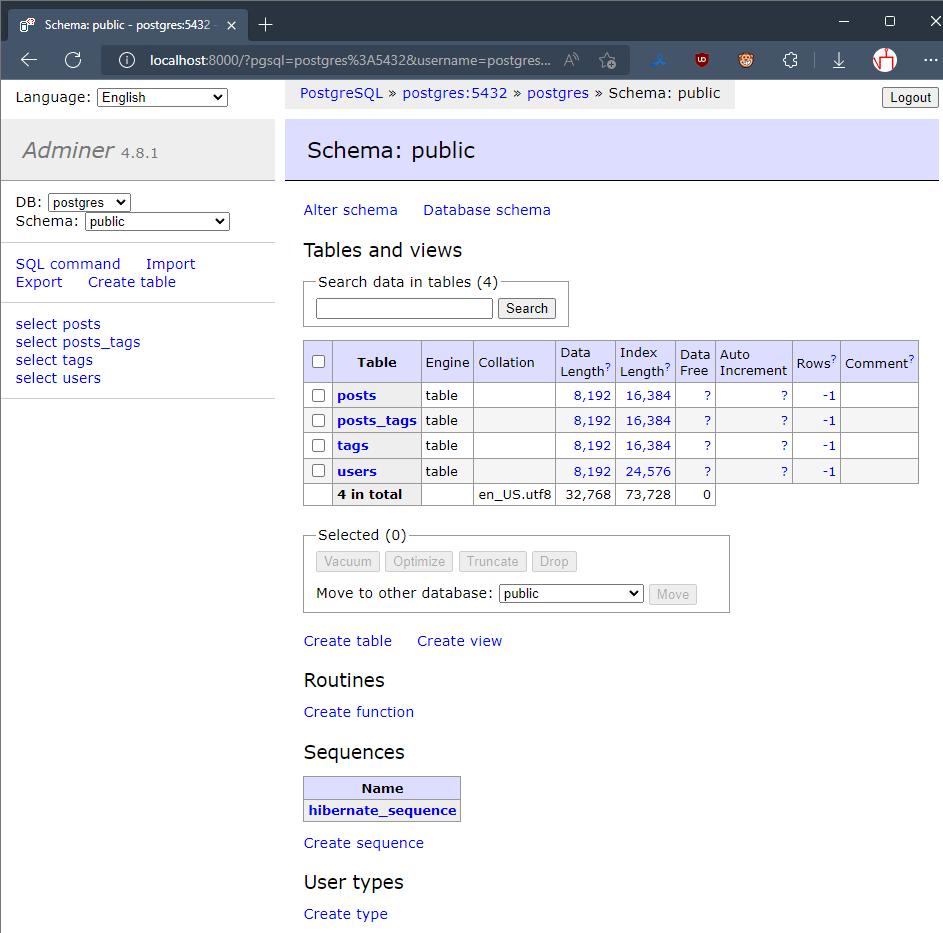


Рисунок 29 – Демонстрация работы панели Adminer

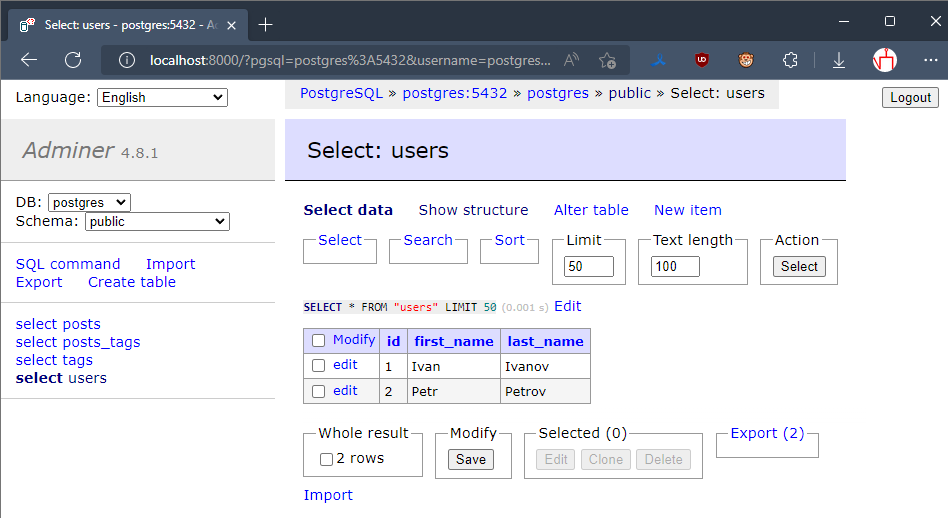


Рисунок 30 – Демонстрация работы панели Adminer

Проверим работу Prometheus и Grafana (Рисунок 31).

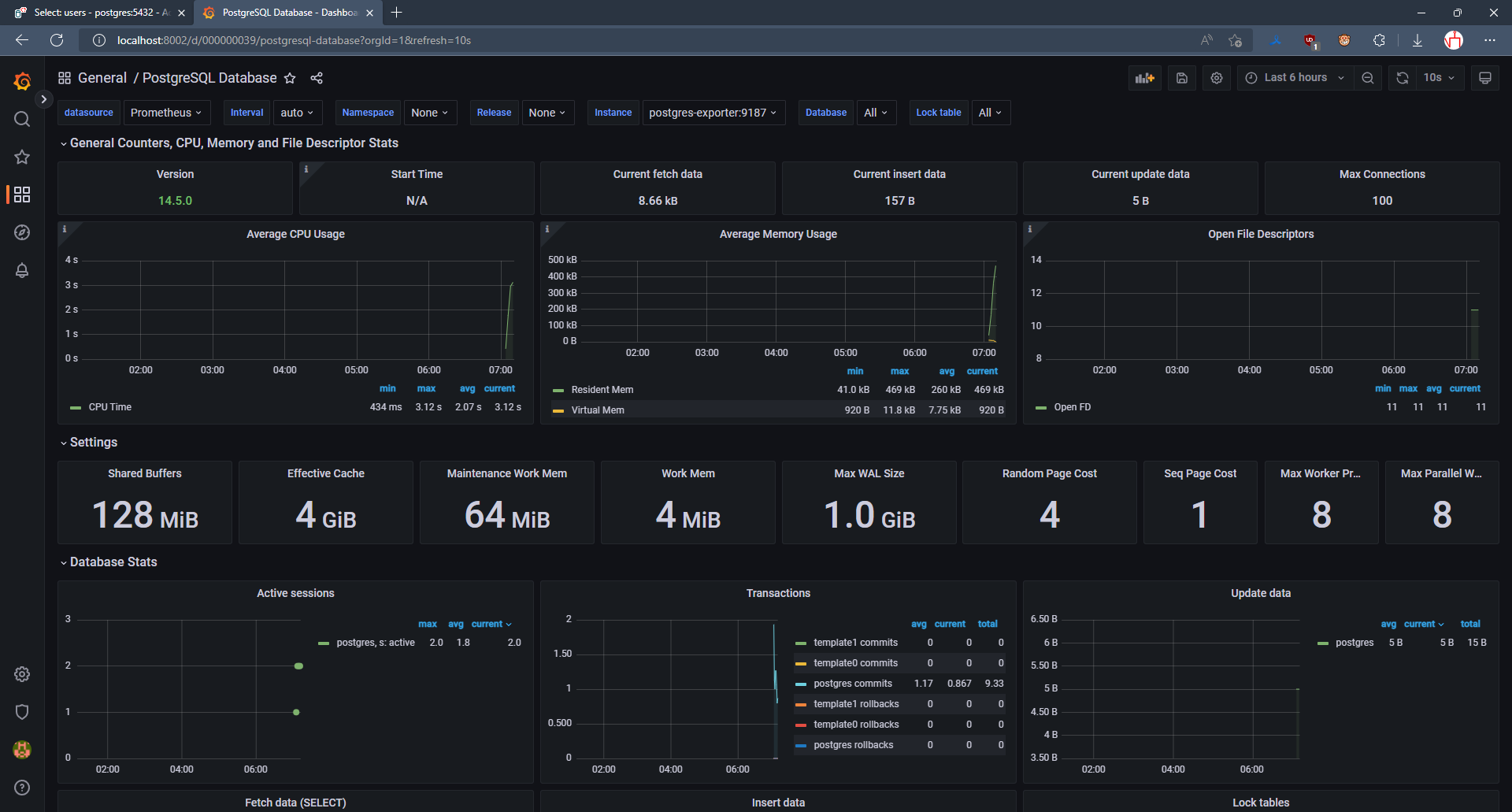


Рисунок 31 – Демонстрация работы Prometheus и Grafana

**Вывод**

В результате выполнения данной практической работы были получены навыки работы с docker-compose, Adminer, GrayLog, Prometheus, Grafana, Zabbix, PostgreSQL, Spring Boot.

**Ответы на контрольные вопросы**

1. Назовите основные различия между Prometheus и Zabbix.

Zabbix и Prometheus могут использоваться в различных сценариях мониторинга, без какой-либо специализации в любом из них. Zabbix старше Prometheus и, вероятно, более стабилен, с более готовыми к использованию решениями.

Zabbix имеет ядро, написанное на C и webUI на основе PHP, также использует "агенты" (клиентские программы), написанные в с. Прометей написан на языке го. Zabbix хранит данные в СУБД (MySQL, PostgreSQL, Oracle, sqlite) пользователя выбор. Prometheus использует собственную базу данных, встроенную в бэкэнд-процесс (это нереляционная база данных, специально разработанная для хранения данных мониторинга таким же образом до OpenTSDBмодель данных).

Zabbix по умолчанию использует модель "pull", когда сервер подключается к агентам на каждой машине мониторинга, агенты периодически собирают информацию и отправляют ее на сервер. Альтернативой является режим" активных проверок", когда агенты устанавливают соединение с сервером и отправляют данные к нему, когда это нужно. Прометей предпочитает "вытягивать" модель, когда сервер собирает информацию с клиентских машин. Но!--13-->Prometheus Push Gateway может использоваться в случаях, когда необходима модель" push".

Prometheus требует, чтобы приложение было инструментировано с клиентской библиотекой Prometheus (доступной на разных языках программирования) для подготовки метрик. Но для мониторинга системы или программного обеспечения, которые не могут быть инструментированы, есть официальная "черный ящик экспортер" это позволяет зондировать конечные точки по целому ряду протоколов; кроме того, широкое распространение сторонние "экспортеры" и инструменты доступны, чтобы помочь выставить метрики для Prometheus (подобно "агенты" для Zabbix). Одним из таких инструментов является telegraf (https://github.com/influxdata/telegraf). Zabbix использует собственный tcp-протокол связи между агентами и сервером. Prometheus использует HTTP с буферами протокола (+текстовый формат для удобства использовать с завитком).

Zabbix предлагает собственный webUI для визуализации. Prometheus предлагает базовый инструмент для изучения собранных данных и визуализации их в простых графиках на своем собственном сервере, а также предлагает минимальный Dashboard builder PromDash. Но Prometheus разработан и поддерживается современными инструментами визуализации, такими как Grafana.

Zabbix поддерживает оповещение в своем ядре. Prometheus предлагает решение для оповещения, которое отделено от его ядро в Alertmanager приложение.

1. Как можно запустить две базы PostgreSQL в одном docker-compose файле чтобы они работали одновременно и таблицы внутри не пересекались?

Запустить два контейнера

1. Назовите виды мониторинга систем.

Nagios, Zabbix, Cacti, OpenNMS, Icinga

1. При помощи чего можно передавать конфигурационные переменные в контейнер?

Задав их в Dockerfile, docker-compose или в аргументах запуска образа.

1. Назовите основные различия Docker Swarm и Docker Compose.

Docker Compose используется для конфигурации нескольких контейнеров на одном и том же хосте, когда Docker Swarm инструмент оркестровки контейнеров.

1. Назовите пример задачи, которую невозможно выполнить, используя один лишь docker-compose без Dockerfile

Использование собственного образа

**Список использованных источников**

1. 50 вопросов по Docker, которые задают на собеседованиях, и ответы на них | Хабр. — Текст: электронный [сайт]. — URL: https://habr.com/ru/company/southbridge/blog/528206/
2. Docker Documentation | Docker Documentation — Текст: электронный [сайт]. — URL: — Текст: электронный [сайт]. — URL: https://docs.docker.com/
3. Zabbix Documentation — Текст: электронный [сайт]. — URL: https://www.zabbix.com/manuals
4. Prometheus Documentation — Текст: электронный [сайт]. — URL: https://prometheus.io/docs/introduction/overview/
5. Grafana Documentation — Текст: электронный [сайт]. — URL: https://grafana.com/docs/
6. GrayLog Documentation — Текст: электронный [сайт]. — URL: https://docs.graylog.org/