Лабораторная работа №1

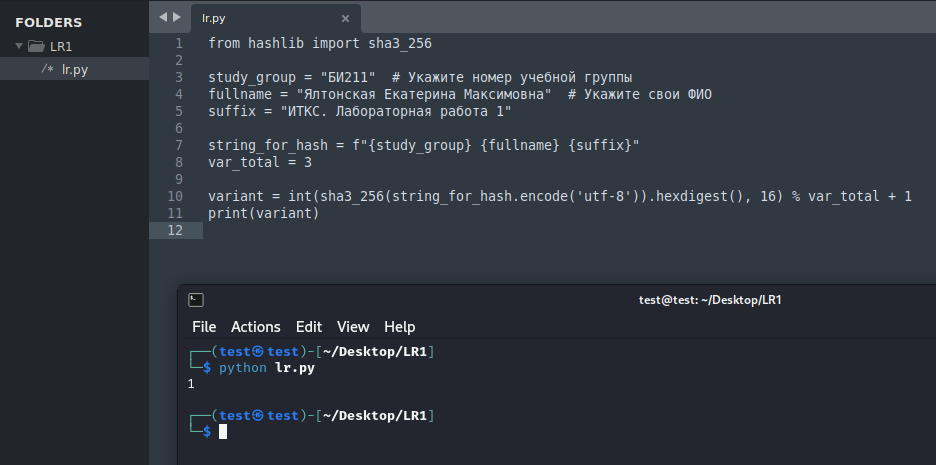
**Освоение Docker и Docker Compose**

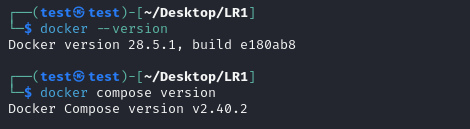
**Цель:** Научиться создавать Docker-образы, управлять контейнерами и развертывать мультисервисные приложения.

**Технологии:** Docker, Docker Compose, Python, PostgreSQL.

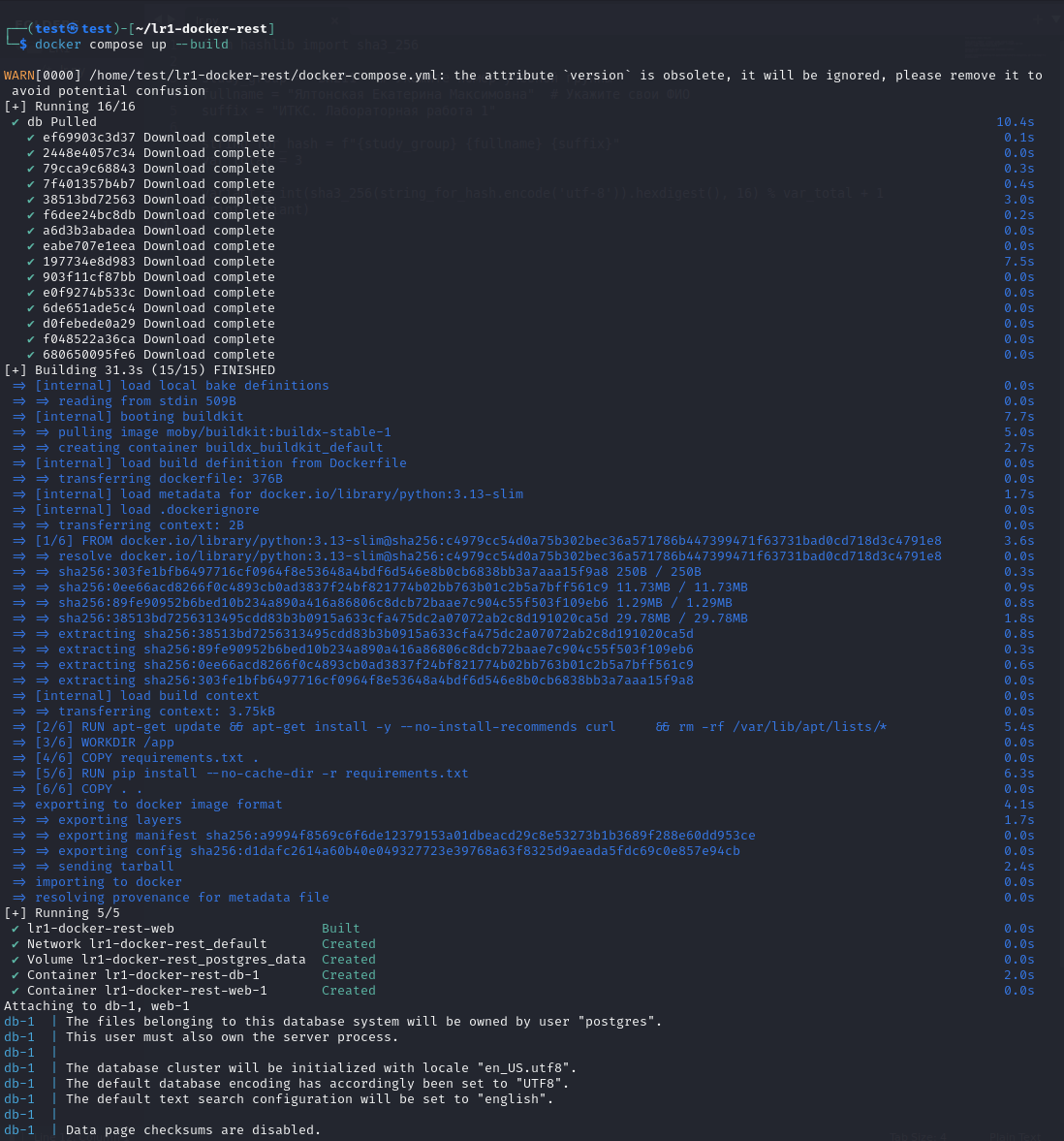
# Отчёт по Лабораторной работе №1.

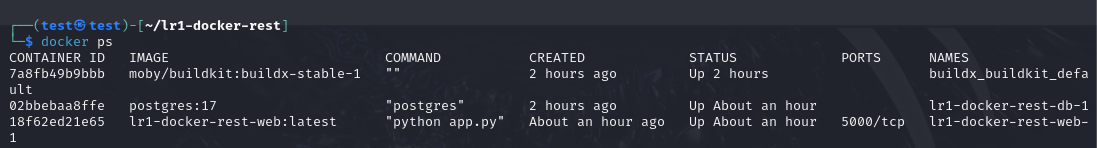
## Выполнение задания. Вариант 1: REST API для управления задачами (CRUD)

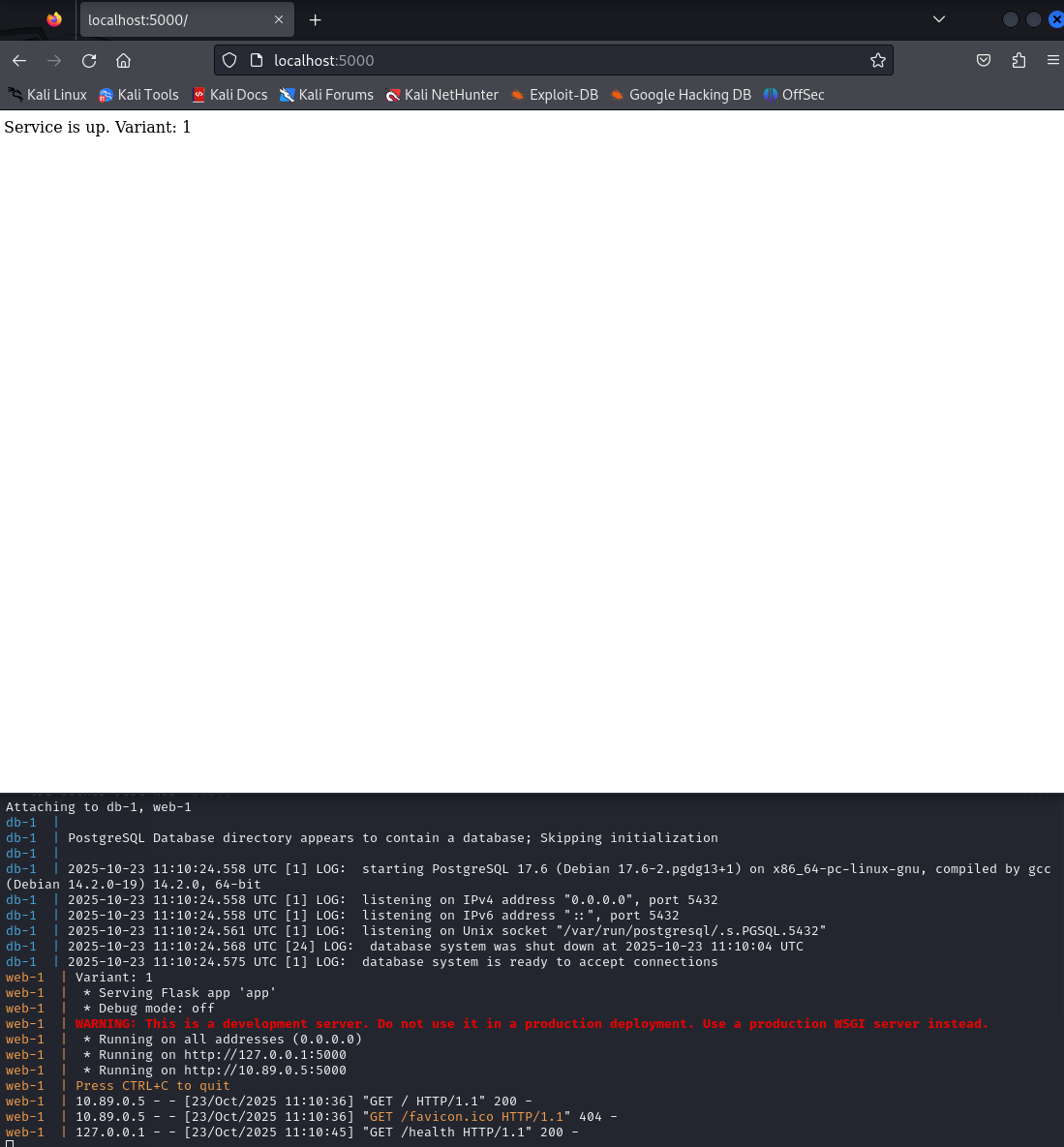
1. Определяю свой вариант. У меня 1 вариант.   
   
2. Устанавливаю Docker и Docker Compose.

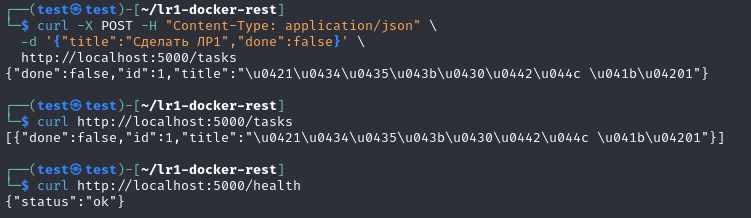


1. Создаю проект.  
   
2. Собираю и запускаю.





1. Открываю <http://localhost:5000/> в браузере.  
   
2. Проверяю API.

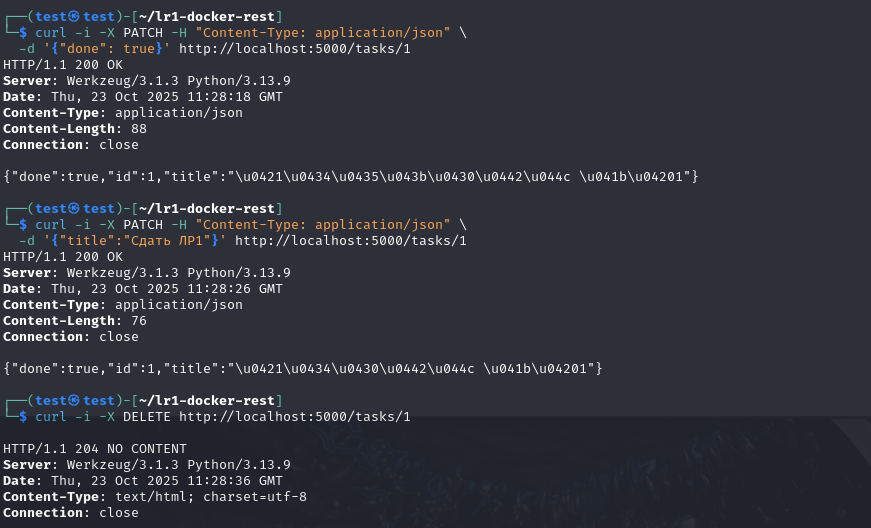


1. Добавляю CRUD. Редактирую app.py.

CRUD — это 4 базовые операции над ресурсами:

* **C (Create)** → POST /tasks — создать задачу. (уже есть)
* **R (Read)** → GET /tasks и (желательно) GET /tasks/<id> — прочитать список/одну запись. (есть)
* **U (Update)** → PATCH /tasks/<id> — частично обновить поля (title, done).
* **D (Delete)** → DELETE /tasks/<id> — удалить запись.

HTTP-коды: 200 (GET/ PATCH), 201 (POST), 204 (DELETE), 400 (невалидный ввод), 404 (нет такого id).

1. Проверяю остальное.  
   

По скринам видно, что:

* PATCH /tasks/1 даёт **200 OK** и возвращает обновлённую задачу — обновление работает.
* DELETE /tasks/1 даёт **204 No Content** — удаление корректно.

## Ответы на контрольные вопросы.

1. **Зачем нужны volumes для PostgreSQL в Docker Compose?**

Именованные тома обеспечивают постоянство данных вне жизненного цикла контейнера. При пересоздании контейнера данные, смонтированные в /var/lib/postgresql/data, сохраняются в volume (например, postgres\_data), что позволяет безопасно обновлять/перезапускать сервис без потери БД и упрощает резервное копирование.

1. **Как Docker обеспечивает изоляцию процессов?**

Изоляция достигается через Linux **namespaces** (PID, NET, MNT, UTS, IPC, USER), которые разделяют видимость процессов, сети и файловых систем, и **cgroups**, которые ограничивают ресурсы (CPU/память/IO). Дополнительно применяются урезанные capabilities и профили безопасности (seccomp/AppArmor/SELinux). Контейнеры разделяют ядро хоста и не являются ВМ.

1. **Почему в healthcheck используется curl, а не «проверка порта»?**

Открытый порт не гарантирует работоспособность приложения. curl -f проверяет уровень HTTP/приложения (ожидает 2xx/3xx, учитывает таймаут/код ответа), поэтому при ошибке маршрута /health (например, 500) контейнер корректно помечается как **unhealthy**, что надёжнее простой проверки сокета.

1. **Чем отличается CMD от ENTRYPOINT в Dockerfile?**

ENTRYPOINT задаёт неизменяемую исполняемую команду контейнера, а CMD — её аргументы по умолчанию. При совместном использовании итоговая команда — **ENTRYPOINT + CMD**; аргументы, переданные в docker run, заменяют CMD, но не ENTRYPOINT (напр., ENTRYPOINT ["python"], CMD ["app.py"] → python app.py; с аргументом tools.py будет python tools.py).

## Вывод.

В Docker Compose развернут воспроизводимый стенд: Flask-сервис с PostgreSQL, реализован полный REST-CRUD над задачами и маршрут /health. Сохранность данных обеспечена именованным томом; корректность работы подтверждена curl-тестами, HTTP-кодами, логами и healthchecks, учтена задержка готовности БД.

## Ссылка на репозиторий.