Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

**Отчет по лабораторным работам №9**

Дисциплина: «Разработка профессиональных приложений»

Docker

Выполнил:

студент группы ИВТАПбд-22

Архипова А.С.

Проверил:

преподаватель кафедры

«Вычислительная техника»

Исхаков И.И.

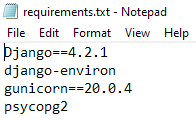
Ульяновск, [2023](https://github.com/ghub-ayrtom/DPA-lab5)

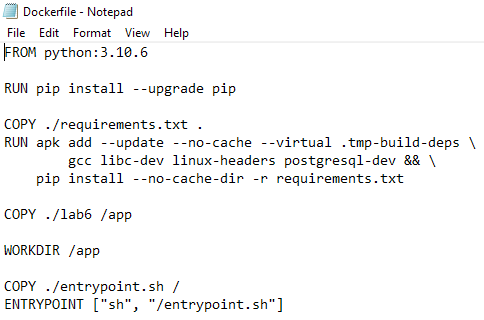
[**Постановка задачи**](https://github.com/ghub-ayrtom/DPA-lab5)

Для лабораторной работы №9 необходимо подготовить Docker Compose файл, позволяющий запустить 2 контейнера: 1. с автоматическим развертыванием всех необходимых библиотек и запуска gunicorn сервера; 2. для СУБД postgresql.

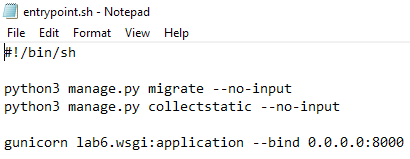
**Описание реализации**

Выполнение данной лабораторной работы я начала с создания в папке с python проектом текстового файла requirements.txt, в котором прописала все необходимые библиотеки, которые понадобятся для развёртывания Docker-контейнеров:



После этого, в той же папке с python проектом был создан файл Dockerfile. Это файл для предварительной работы, набор инструкций, который нужен для записи образа. В нем описывается, что должно находиться в образе, какие команды, зависимости и процессы он будет содержать. При запуске команды docker run программа сначала проверяет, есть ли нужный образ в локальном хранилище:

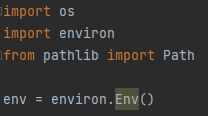
В его конце можно увидеть запуск скрипта entrypoint.sh, который выглядит следующим образом:

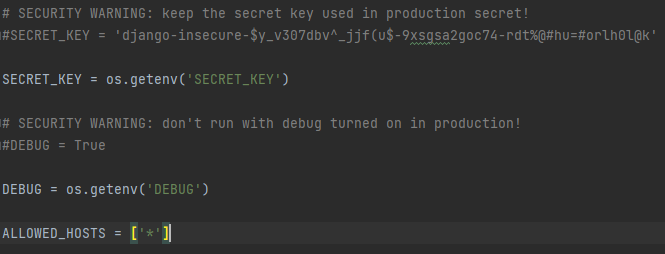


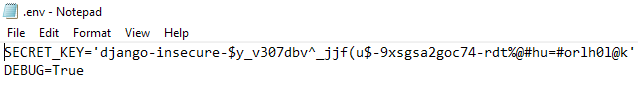
После этого был создан самый основной файл при работе с технологией Docker – это docker-compose.yml — конфигурационный файл в YAML-формате, описывающий логику запуска и взаимодействия контейнеров между собой и внешним миром. В сущности, инструкции заложенные в docker-compose.yml по логике работы идентичны ключам команды docker run:

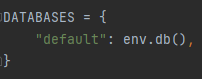


Для работы с Docker Compose также следующим образом был изменён файл settings.py нашего Django-приложения:





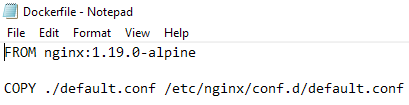






Дополнительно был создан 3 контейнер для развёртывания и запуска nginx сервера, но, как оказалось после, можно было обойтись и без него.

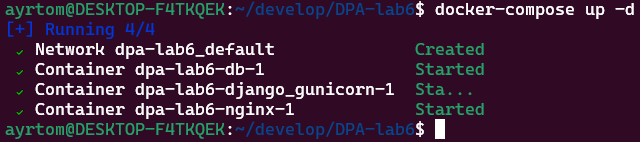
Для начала, в папке с python проектом DPA-lab6 был создан подкаталог nginx и в нём 2 файла: default.conf и ещё один Dockerfile, который имеет следующую структуру:



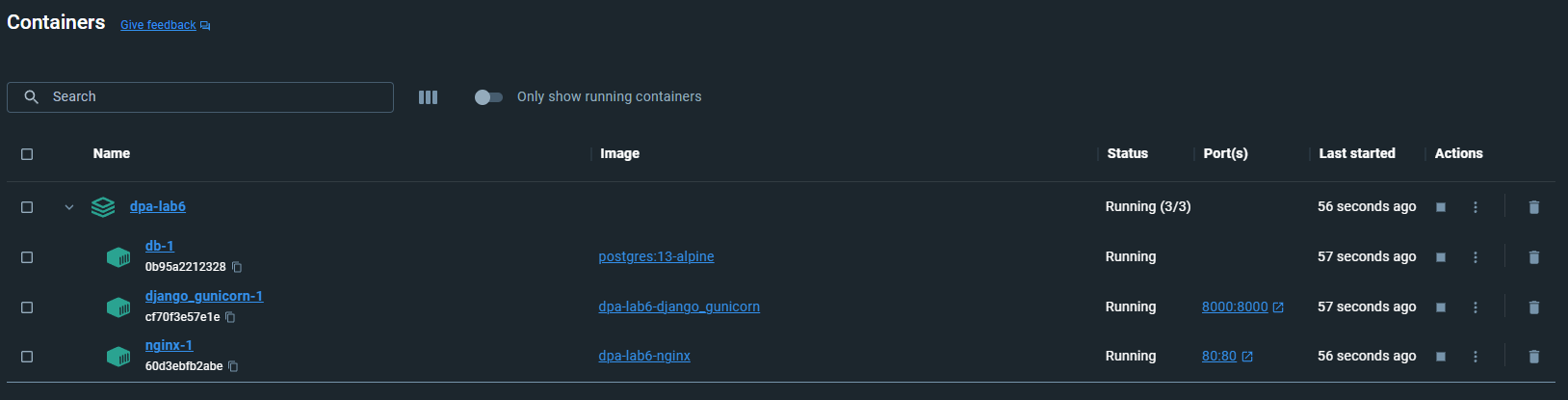
В нём мы командой FROM импортируем nginx версии 1.19.0. и заменяем конфиг nginx по умолчанию тем конфигом, который мы создали сами и который выглядит следующим образом:



Как видно, веб-сервер nginx занимает 80 порт, в свою очередь WSGI-сервер gunicorn занимает стандартный порт 8000, он расчитан на более серьёзную нагрузку, нежели локальный django-сервер, а также является более безопасным.

После того как все шаги выше были успешно выполнены, можно попробовать запустить docker-compose файл следующей командой в терминале WSL:

Развернулось 3 Docker-контейнера, которые выглядят следующим образом:



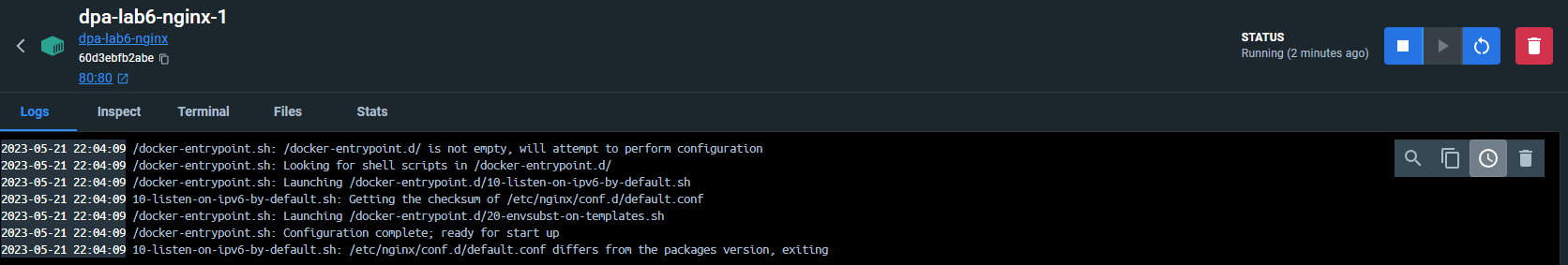
Контейнер dpa-lab6-db-1 отвечает за развёртывание СУБД postgresql:



Контейнер dpa-lab6-django-gunicorn-1 отвечает за автоматическое развёртывание всех необходимых библиотек и запуска gunicorn сервера:



Контейнер dpa-lab6-nginx-1 отвечает за запуск nginx-сервера:



Теперь можно попробовать зайти на сам локальный сервер gunicorn (порт: 8000) и посмотреть, всё ли работает.

**Описание возникших затруднений**

Основные затруднения при выполнении данной лабораторной работой были связаны с первичным знакомством с технологией контейнеризации Docker. Было достаточно непросто разобраться как во внутренней структуре самих файлов, так и в том, в каком порядке они должны создаваться и подключаться друг к другу.

Также, немало проблем возникло с подключением и установкой необходимых библиотек и их совместимостью.