БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет прикладной математики и информатики
Кафедра технологий программирования

Лабораторная работа №1

По дисциплине «Непрерывное интегрирование и сборка программного обеспечения»

Методические указания по выполнению лабораторной работы

Подготовила:
Давидовская М. И.,
Ст. преподаватель кафедры ТП

Минск, 2024 г.

Содержание

Цель работы

Изучение современных технологий контейнеризации.

Задачи работы

1. Установить средство контейнеризации docker.
2. Изучить применение и принципы docker.
3. Изучить создание многоконтейнерных инфраструктур.
4. Исследовать структуру файла Dockerfile и применить для конфигурации контейнеров.
5. Изучить утилиту docker-compose и структуру файла docker-compose.yml.
6. Развернуть проект из не менее 3х различных сервисов при помощи docker-compose.
7. Оформить отчёт в формате Markdown и создать Pull Request в git-репозитории.

Краткие теоретические сведения

Dockerfile и создание образов

Источник — https://practicum.yandex.ru/learn/yc-devops-container

Образ или **Image** — упакованный набор файлов приложения или сервиса. С помощью одного образа можно запустить множество контейнеров.

Образ включает код приложения, зависимости, конфигурацию, а также параметры окружения и команду для запуска основного процесса. Образы служат основой для создания контейнеров.

Контейнер или **Container** — запущенный и готовый к использованию экземпляр образа. Безопасность и стабильность процесса в контейнере обеспечивается изоляцией от остальной системы.

Разница между образом и контейнером в том, что образ предоставляет статический набор файлов, который может быть использован для создания множества контейнеров, тогда как контейнер — это динамическая сущность, которая представляет собой работающий экземпляр образа.

Чтобы собрать образ, нужен Dockerfile.

Dockerfile — конфигурационный файл с инструкциями, применяемыми при сборке Docker-образа и запуске контейнера.

Пример простого Dockerfile:

```
FROM ubuntu:22.04
RUN apt-get update && apt-get install -y nginx
COPY myapp /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Отразим на схеме слои. Размер слоёв тоже важно видеть — это поможет в дальнейшем сделать образ легче.

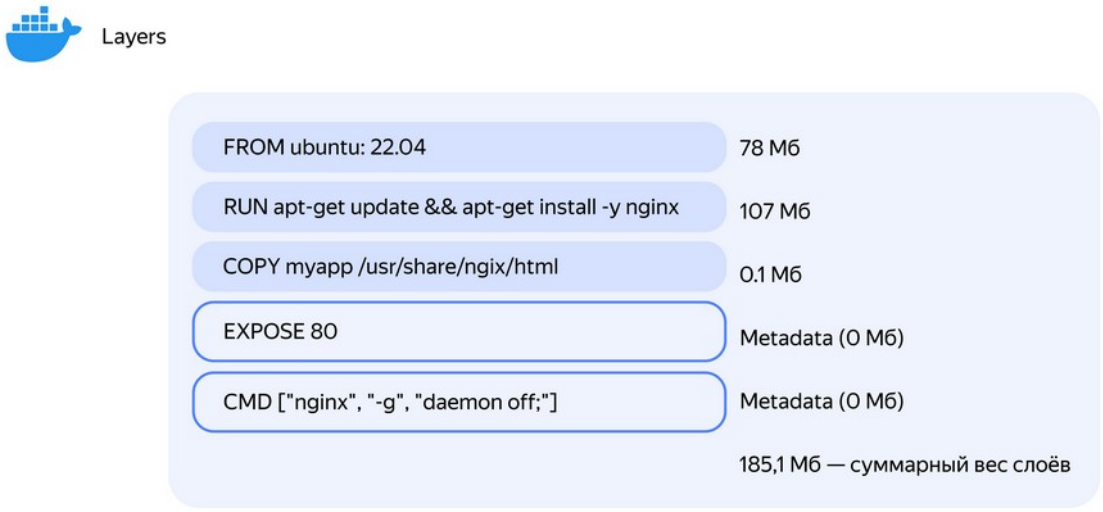


Figure 1: Слои контейнера Docker

Этот Dockerfile выполняет четыре действия:

1. Создаёт образ на основе Linux-дистрибутива Ubuntu 22.04.
2. Устанавливает веб-сервер Nginx.
3. Копирует содержимое каталога myapp в каталог `/usr/share/nginx/html`.
4. Сообщает что процесс в запущенном контейнере будет слушать порт `80` и запускать сам процесс `Nginx` соответствующей командой.

Инструкции `FROM`, `RUN`, а также `ADD` создают отдельные слои или layers итогового образа, который содержит изменения файлов относительно предыдущего слоя. Инструкции `EXPOSE` и `CMD` видны в истории изменений образа как слои. Их размер — `0`, потому что

изменений файлов в таких слоях не происходит, но добавляются метаданные к образу.

Команда `docker build -t mynginx:latest` создаст образ `mynginx:latest` согласно инструкциям в `Dockerfile`.

Вес итогового образа `185,1 Мб`. Он складывается из суммы весов всех слоёв. Для просмотра информации о слоях сборки используется команда `docker image history`.

Для просмотра подробной информации об образе можно воспользоваться командой `docker image inspect <имя:тег или идентификатор образа>`.

Например:

```
docker inspect mynginx:latest
```

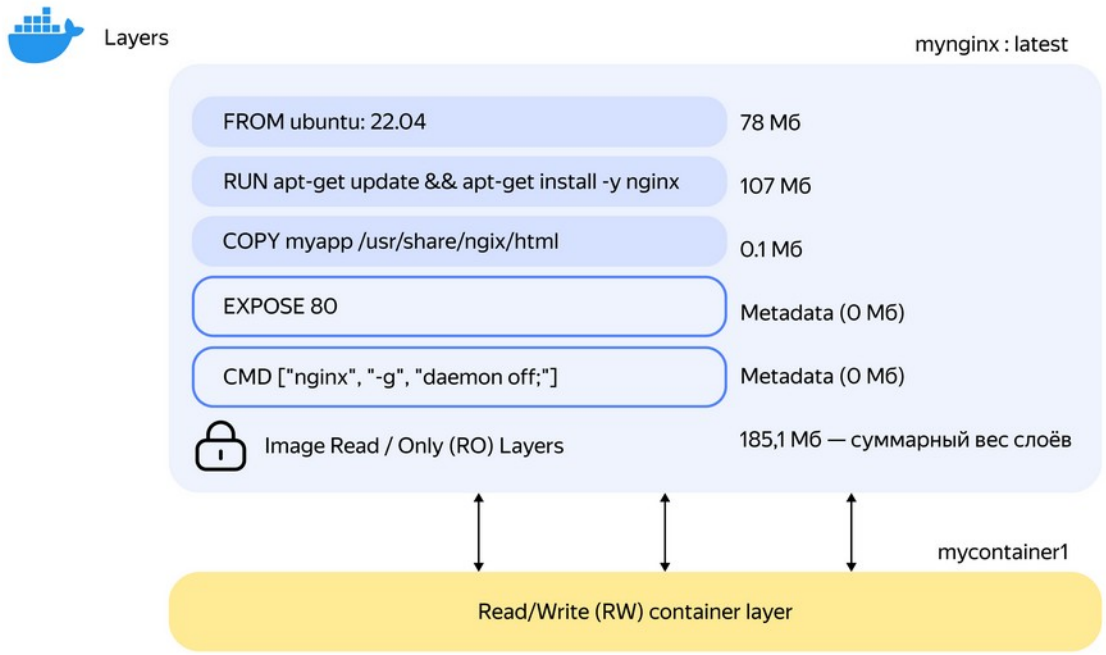


Figure 2: Слой чтения/записи в запущенном контейнере

Далее запускается контейнер.

Контейнер с именем `nginx-container` на базе образа `mynginx:latest` запускается командой

```
docker run --name nginx-container -p 80:80 mynginx:latest
```

Он будет слушать порт `80` на хостовой системе. При запуске контейнера создаётся ещё один перезаписываемый слой, который будет содержать файлы, изменённые в контейнере относительно образа, на базе которого он запущен.

Слои голубого цвета — образ контейнера до его запуска. После запуска появляется новый Read/Write слой — на схеме он жёлтого цвета.

Перезаписываемый слой может содержать временные файлы, создаваемые приложением во время работы в контейнере, например СУБД MysSQL может создавать временные таблицы на диске в случае, если во время обработки SQL-запроса они не помещаются в выделенной ему оперативной памяти.

Если на базе одного и того же образа будут запущены несколько контейнеров, для каждого будет создан свой отдельный перезаписываемый слой. Базовый образ при этом остаётся в единственном экземпляре.

Сборка образов выполняется на основе инструкций в `Dockerfile`. При создания образа в `Dockerfile` добавляются файлы, например, код проекта. Для запуска сборки используется команда `docker image build`. Этой команде обязательно передаётся как минимум один аргумент.

Аргумент или **контекст** — это каталог с набором файлов, которые используются при сборке образа контейнера.

Команда `docker image build` имеет следующие псевдонимы (aliases):

```
docker build
docker buildx build
docker builder build
```

Как работает docker build

Рассмотрим синтаксис:

```
docker image build [OPTIONS] PATH | URL | -
```

В этом примере:

- `OPTIONS` — необязательные опции запуска команды;
- `PATH` | `URL` | `-` — контексты сборки образа.

В большинстве случаев используется локальный путь `PATH`. При этом вместо него, в качестве контекста, можно указывать внешний Git-репозиторий через `URL` или `stdin` через `-`.

Если явно не указывать расположение `Dockerfile`, то команда `docker build` будет пытаться найти файл с названием `Dockerfile` в

каталоге контекста. Чтобы указать другое расположение `Dockerfile`, можно использовать опцию `-f <путь к Dockerfile>`.

Рассмотрим на примере. Имеется следующая структура каталогов:

```
# tree .
├── app1
│   ├── Dockerfile
│   └── run.py
├── app2
│   ├── Dockerfile
│   └── run.py
└── Dockerfile.custom
```

Запустим команду `docker build` с указанием контекста в папке `app1`:

```
docker build app1/
```

Для сборки будет использоваться файл `app1/Dockerfile`.

Теперь зададим в явном виде расположение `Dockerfile`:

```
docker build -f Dockerfile.custom ./
```

При сборке `Dockerfile.custom` будут доступны файлы и `app1/run.py`, и `app2/run.py`.

Инструкции Dockerfile

Напомним, что `Dockerfile` — это конфигурационный файл с инструкциями, применяемыми при сборке Docker-образа и запуске контейнера. Вот наиболее используемые из них:

Figure 3: Инструкции Dockerfile

Инструкции FROM, RUN, COPY и ADD участвуют в создании слоёв итогового образа. Другие инструкции отвечают за настройку, описание метаданных и указание действий, которые должны выполняться во время работы контейнера. Полный список инструкций описан в документации Docker.

Инструкция FROM в Dockerfile определяет базовый образ, который будет использоваться для создания нового образа. Базовый, или родительский, или parent образ может быть любым публичным или приватным образом, доступным в реестре образов по умолчанию — Docker Hub, или любом другом реестре образов.

Например, чтобы создать образ на основе официального образа Nginx, вы можете использовать следующую инструкцию:

```
FROM nginx:1.25
```

Это означает, что ваш новый образ будет основан на официальном образе Nginx, и все последующие инструкции будут выполняться в контексте этого образа.

Если вы хотите создать свой собственный базовый образ, не основываясь ни на каком другом, то в качестве базового образа следуют указывать образ scratch:

```
FROM scratch
```

Инструкция RUN в Dockerfile используется для выполнения команд внутри контейнера во время сборки образа. Например, инструкция устанавливает веб-сервер Nginx в контейнер на основе Debian.

```
RUN apt-get update && apt-get install -y nginx
```

Инструкция COPY используется для копирования файлов и папок из файловой системы хоста в файловую систему контейнера. Пример использования:

```
COPY ./src /app/src
```

В этом примере все файлы и папки из папки `./src` будут скопированы в каталог `/app/src` внутри контейнера.

Инструкция ADD схожа по синтаксису и назначению с инструкцией COPY. При этом помимо копирования файлов, ADD поддерживает возможность скачивания удалённых файлов, а также распаковки архивов:

Копирование локальных файлов:

```
ADD ./file.txt /app/file.txt
```

Копирование файлов с URL:

```
ADD https://example.com/file.txt /app/file.txt
```

Копирование содержимого локального архива:

```
ADD ./archive.tar.gz /app/
```

Инструкция ENTRYPOINT используется для указания команды, которая будет выполнена при запуске контейнера. Эта команда может быть любой командой, которую вы хотите запустить внутри контейнера, например, запуск веб-сервера или базы данных.

По умолчанию, если ENTRYPOINT не задан, то его значение — `/bin/sh -c`.

Инструкция CMD используется в качестве аргументов для ENTRYPOINT. Итоговая команда запуска контейнера складывается из объединения инструкций ENTRYPOINT + CMD.

В следующем примере при запуске контейнера выполнится команда `/usr/bin/python run.py`:

```
ENTRYPOINT ["/usr/bin/python"]
CMD ["run.py"]
```

Если ENTRYPOINT не задан, то команда запуска в следующем примере будет `/bin/sh/ -c python run.py`

```
CMD ["python", "run.py"]
```

В Dockerfile должна быть указана хотя бы одна из команд CMD или ENTRYPOINT.

Инструкция WORKDIR устанавливает рабочий каталог для последующего выполнения инструкций RUN, CMD, ENTRYPOINT, COPY и ADD. Она создаёт новый каталог, если он ещё не существует, и сохраняет его в контексте сборки.

Например, следующая команда скопирует файл run.py в каталог /app/run:

```
WORKDIR /app/run
COPY ./run.py ./run.py
ENTRYPOINT ["/usr/bin/python"]
CMD ["run.py"]
```

Задания

Методические указания

Все результаты лабораторной работы должны быть опубликованы в git-репозитории, ссылка на который доступна в курсе «Непрерывное интегрирование и сборка программного обеспечения»

Критерии оценивания

Для групп 11-13 выполнить все 5 заданий, для группы 14 — задания 1-4.

Содержание отчета

1. Цель работы.
2. Вариант задания.
3. Код приложений, конфигурационных файлов.
4. Ответы на контрольные вопросы.

Отчет должен быть опубликован в git-репозитории на github. Все результаты лабораторной работы должны быть опубликованы в git-репозитории, ссылка на который доступна в курсе «Непрерывное интегрирование и сборка программного обеспечения».

В файле Readme проекта на github должна быть ссылка на отчёт. Отчет опубликовать во внешнем хранилище или в репозитории в каталоге /docs. Если в лабораторной работе необходимо написать программу/ы, то отчёт должен результаты тестов по каждой программе и ответы на контрольные вопросы.

Пример оформления файла Readme может быть таким:

```
# Overview

Report on LabRabota1.

# Usage

// Заменить <<link>> и <<folder>> на соответствующие ссылки/названия
To check, please, preview report by <<link>> and source files
in <<folder>>.

# Author

Your name and group number.

# Additional Notes

// СКОПИРОВАТЬ И ВСТАВИТЬ ССЫЛКУ НА СВОЙ РЕПОЗИТОРИЙ,НАПРИМЕР
https://github.com/maryiad/lab3-task1-gr16-david
```

Каждая лабораторная работа содержит тексты задач и контрольные вопросы, ответы на которые проверяются преподавателем при приёме работы у студента.

Выполнение студентом лабораторной работы и сдача её результатов преподавателю происходит следующим образом:

1. Студент выполняет разработку программ.

2. В ходе разработки студент обязан следовать указаниям к данной задаче (в случае их наличия). Исходные тексты программ следует разрабатывать в соответствии с требованиями к оформлению, приведёнными в приложении.

3. Студент выполняет самостоятельную проверку исходного текста каждой разработанной программы и правильности её работы, а также свои знания по теме лабораторной работы.

Задание 1. Основы управления контейнерами и образами контейнеров Docker

В качестве операционной системы рекомендуется использовать ОС Ubuntu, установленную в виртуальной машине или же как вторая система.

1. Изучить руководства по установке и настройке Docker Engine:
   - Install Docker Engine
   - How To Install and Use Docker on Ubuntu 20.04
   - How to Install Docker on Ubuntu 24.04
2. Установить Docker Engine. При установке Docker Engine использовать руководство с сайта Docker.com для соответствующей версии ОС. Для выполнения первой лабораторной работы могут быть использованы ОС Windows и macOS. В окружениях Windows или macOS рекомендуется использовать Docker Desktop. Для ОС Linux — только консольный вариант Docker Engine.
   - Ознакомиться с командами `ps`, `run`, `container`, `image`, `stop`, `rm` и продемонстрировать вывод списка активных контейнеров, всех контейнеров, вывода списка образов и операции управления контейнерами.
   - На примере контейнера веб-сервера `nginx` продемонстрировать запуск контейнера с пробросом портов на основную (хостовую) систему и подключение к контейнеру из хостовой системы с помощью утилиты `curl` и в браузере:
   - `nginx:latest` с именем контейнера `mynginxlast`
   - `nginx:alpine` с именем контейнера `mynginxalpine`

- `nginx:1.26` с именем контейнера `mynginx1-26`

3. Продемонстрировать
   - запуск контейнера mynginxlast в неинтерактивном режиме и подключение к контейнеру с помощью манды **`exec`** и и опций **`-it`** (`-i — --interactive; -t — --tty`) и выход из него с помощью команды `exit`;
   - запуск контейнера `mynginxlast` в интерактивном режиме и выход из него с помощью команды `exit`.
   - В каталоге репозитория лабораторной работы создать каталог, например `task1`. В данном каталоге создать простой `html`-документ (веб-страницу) содержащий теги `html`, `head`, `title`, `body`, `h1`, `p`, в качестве содержимого которого указать название текущего задания, ФИО и номер группы студента. Продемонстрировать монтирование каталога `task1` как тома контейнера, например **`mynginxalpine`**, который является корневым каталогом для виртуального хоста (сайта) `nginx`.

Задание 2. Создание и публикация образа контейнера в репозиторий (реестр) контейнеров

1. Изучить методические рекомендации «Как работает docker build» и документацию по команде.
2. Собрать собственный образ на основе пункта 3 из задания 1 и опубликовать в репозиторий Docker Hub.

Задание 3. Настройка конфигурации многоконтейнерных систем в консоли и с помощью Dockerfile

1. Изучить методические рекомендации «Dockerfile и создание образов».
2. Создать приложением с хранением данных в базе данных, работающей в контейнере Docker.
3. Создать образ контейнера с приложением на основе примера ниже.

Пример 1 для задания 3

Создайте в каталоге репозитория для лабораторной работы папку `task3`.

Рассмотрим создание приложения «API на Python», которое будет использовать в некоторых запросах Postgres.

1️⃣ Запустите Postgres из уже готового образа:

```
docker run --name postgres-db \
-e POSTGRES_PASSWORD=apipass \
-e POSTGRES_DB=api \
-e POSTGRES_USER=apiuser \
-p 5432:5432 \
-d postgres:16.2-alpine
```

2️⃣ Создайте директорию `simple_python_app` в каталоге `task3` и перейдите в него:

```
mkdir ~/simple_python_app
cd ~/simple_python_app
```

3️⃣ Создайте файл `~/simple_python_app/app.py` со следующим содержимым:

```
from fastapi import FastAPI
import psycopg2
import os

app = FastAPI()

DATABASE_HOST = os.getenv("API_DB_HOST", "localhost")
DATABASE_PORT = os.getenv("API_DB_PORT", "5432")
DATABASE_NAME = os.getenv("API_DB_NAME", "api")
DATABASE_USER = os.getenv("API_DB_USER", "apiuser")
DATABASE_PASS = os.getenv("API_DB_PASS", "apipass")

DATABASE_URL = f"postgresql://{DATABASE_USER}:
{DATABASE_PASS}@{DATABASE_HOST}:
{DATABASE_PORT}/{DATABASE_NAME}"

conn = psycopg2.connect(DATABASE_URL)
cursor = conn.cursor()
```

```
@app.get("/")
async def root():
    cursor.execute(
        "SELECT version();"
    )
    item = cursor.fetchone()
    return {"message": "Hello World",
            "postgres_version": item[0]}

@app.get("/hello/{name}")
async def say_hello(name: str):
    return {"message": f"Hello {name}"}
```

4️⃣ Создайте файл `~/simple_python_app/requirements.dev.txt` со следующим содержимым:

```
fastapi==0.110.2
uvicorn[standard]==0.29.0
psycopg2-binary==2.9.9
```

5️⃣ Создайте файл `~/simple_python_app/requirements.txt` со следующим содержимым:

```
fastapi==0.110.2
uvicorn[standard]==0.29.0
psycopg2==2.9.9
```

6️⃣ Для локальной разработки Python разработчики используют virtual env для установки зависимостей в выделенном окружении. Для примера создайте такое окружение и установите зависимости `requirements.dev.txt`:

Для этого выполните следующие команды в каталоге `~/simple_python_app/`:

```
python3 -m venv venv                          Создание виртуального окружения
source ./venv/bin/activate                    Активация виртуального окружения
pip install -r requirements.dev.txt           Установка зависимостей
```

7️⃣ Перед тем, как собрать образ приложения на Python запустите его локально:

```
uvicorn app:app --host 0.0.0.0
```

8️⃣ Перейдите по ссылке `http://localhost:8000`.

9️⃣ В результате увидите ответ API:

```
{
message: "Hello World",
postgres_version: "PostgreSQL 16.2 on x86_64-pc-linux-musl, compiled by gcc (Alpine 13.2.1_git20231014) 13.2.1 20231014, 64-bit"
}
```

🔟 Остановите приложение, нажав `Ctrl+C`.

Для того, чтобы создать Docker-образ с приложением, вам необходим Dockerfile.

1️⃣1️⃣ Создайте файл `~/simple_python_app/Dockerfile` со следующим содержимым:

```
FROM python

# Копирование кода приложения
COPY . /app/

# Установка системных зависимостей для сборки зависимостей python
RUN apt-get update \
    && apt-get install -y gcc

# Установка зависмостей python
RUN pip install -r /app/requirements.txt

# Установка рабочей директории
WORKDIR /app

CMD ["uvicorn", "app:app", "--host", "0.0.0.0"]
```

1️⃣2️⃣ Запустите сборку приложения из папки `~/simple_python_app/`:

```
docker build -t simple_python_app .
```

1️⃣3️⃣ После успешной сборки запустите контейнер с приложением:

```
docker run -e API_DB_HOST=<внутрениий IP localhost> -p 8001:8000 simple_python_app
```

1️⃣4️⃣

Перейдите по ссылке `http://localhost:8001`. Получим тот же результат, что и при запуске приложения вне контейнера.

```
    {
    message: "Hello World",
    postgres_version: "PostgreSQL 16.2 on x86_64-pc-linux-musl, compiled by gcc (Alpine 13.2.1_git20231014) 13.2.1 20231014, 64-bit"
    }
```

1️⃣5️⃣ Остановите запущенный контейнер приложения, нажав `Ctrl+C`

1️⃣6️⃣ Остановите и удалите контейнер с Postgres:

```
docker stop postgres-db
docker rm postgres-db
```

Пример 2 для задания 3

Примените разные техники оптимизации образа контейнера Docker и после каждого этапа **проверяйте размеры получаемых образов командой**

```
docker image ls simple_python_app
```

Перечень этапов для примеры 2 задания 3:

1. Соберите с помощью команды

```
docker build -t simple_python_app:v1 .
```

сборку приложения на основе Dockerfile из примера 1.

2. Оптимизируйте сборку, выполнив сортировку слоев в Dockerfile (см. ниже). На основе примера Dockerfile соберите simple_python_app:v2 командой:

```
docker build -t simple_python_app:v2 .
```

Пример Dockerfile с сортировкой слоев

```
FROM python
# -- Редко изменяемые операции --
# Установка системных зависимостей (в т.ч. для зависимостей python)
RUN apt-get update \
    && apt-get install -y gcc

# Копирование зависимостей python
COPY requirements.txt /app/
```

```
# Установка зависимостей python
RUN pip install -r /app/requirements.txt

# -- Часто изменяемые операции--
# Копирование кода приложения
COPY . /app/

# Установка рабочей директории
WORKDIR /app

CMD ["uvicorn", "app:app", "--host", "0.0.0.0"]
```

3. Включить только необходимые файлы в образ контейнера, например вместо копирования содержимого всего каталога скопировать только файл app.py и объединить команды копирования и установки зависимостей в одну команду. Вид Dockerfile будет как в примере ниже. Соберите версию образа 3, например командой:

```
docker build -t simple_python_app:v3 .
```

Пример Dockerfile с изменением количества файлов в образе контейнера приложения:

```
FROM python
# -- Медленные операции --
#   Установка   системных   зависимостей   (в   т.ч.   для
зависимостей python)
RUN apt-get update \
    && apt-get install -y gcc

# Установка зависмостей python
RUN                                                    --
mount=type=bind,source=requirements.txt,target=/app/requi
rements.txt \
  pip install -r /app/requirements.txt

# -- Быстрые операции --
# Копирование кода приложения
COPY app.py /app/
```

```
# Установка рабочей директории
WORKDIR /app

CMD ["uvicorn", "app:app", "--host", "0.0.0.0"]
```

4. Оптимизируйте Dockerfile (см. ниже), добавив удаление кеша приложений и менеджеров пакетов apt-get и pip-install. Соберите 4-ую версию образа:

```
docker build -t simple_python_app:v4 .
```

Пример Dockerfile с удалением кеша:

```
FROM python
# -- Медленные операции --
#   Установка   системных   зависимостей   (в   т.ч.   для
зависимостей python)
RUN apt-get update \
    && apt-get install -y --no-install-recommends gcc \
    && rm -rf /var/lib/apt/lists/*

# Установка зависмостей python
RUN                                                    --
mount=type=bind,source=requirements.txt,target=/app/requi
rements.txt \
       pip    install    --no-cache-dir    --no-deps    -r
/app/requirements.txt

# -- Быстрые операции --
# Копирование кода приложения
COPY app.py /app/

# Установка рабочей директории
WORKDIR /app

CMD ["uvicorn", "app:app", "--host", "0.0.0.0"]
```

5. Формирование облегченных образов Slim и Alpine.

5.1. Использовать новую версию образа `python:3.12.1-slim` и скопировать файлы из предыдущего образа. Пример Dockerfile ниже. Соберите новую версию образа командой:

```
docker build -t simple_python_app:v5 .
```

Пример Dockefile на основе slim образа

```
FROM python:3.12.1 AS builder
# -- Медленные операции --
#   Установка   системных   зависимостей   (в   т.ч.   для
зависимостей python)
RUN apt-get update \
    && apt-get install -y --no-install-recommends gcc \
    && rm -rf /var/lib/apt/lists/*

# Установка зависмостей python в диррекиторию /app/wheels
RUN                                                       --
mount=type=bind,source=requirements.txt,target=/app/requi
rements.txt \
        pip    wheel    --no-cache-dir    --no-deps    -r
/app/requirements.txt --wheel-dir /app/wheels

# ---------

FROM python:3.12.1-slim

# Копируем собранные файлы python из образа builder
COPY --from=builder /app/wheels /wheels

#  Установка  зависимостей,  которые  нужны  для  работы
приложения
RUN apt-get update \
     && apt-get install -y --no-install-recommends libpq-
dev \
    && rm -rf /var/lib/apt/lists/*

# Устанавливаем зависимости python не пересобирая их
RUN pip install --no-cache --no-cache-dir /wheels/*

# Копирование кода приложения
COPY app.py /app/
```

```
# Установка рабочей директории
WORKDIR /app

CMD ["uvicorn", "app:app", "--host", "0.0.0.0"]
```

5.2. Соберите приложение simple_python_app на основе Alpine. Для этого создайте в каталоге приложения отдельный файл Dockerfile.alpine со следующим содержимым (см. ниже). Соберите образ командой:

```
docker build -t simple_python_app:v5-alpine -f Dockerfile.alpine .
```

Файл Dockerfile.alpine:

```
FROM python:3.12.1-alpine AS builder
# -- Медленные операции --
#  Установка системных зависимостей (в т.ч. для зависимостей python)
RUN apk update \
    && apk add --no-cache gcc musl-dev postgresql-dev

# Установка зависмостей python в диррекиторию /app/wheels
RUN --mount=type=bind,source=requirements.txt,target=/app/requirements.txt \
        pip wheel --no-cache-dir --no-deps -r /app/requirements.txt --wheel-dir /app/wheels

# ---------

FROM python:3.12.1-alpine

# Копирование собранных файлов python из образа builder
COPY --from=builder /app/wheels /wheels

# Установка зависимостей, которые нужны для работы приложения
RUN apk add --no-cache libpq
# Устанавливаем зависимости python не пересобирая их
RUN pip install --no-cache --no-cache-dir /wheels/*
```

```
# Копирование кода приложения
COPY app.py /app/

# Установка рабочей директории
WORKDIR /app

CMD ["uvicorn", "app:app", "--host", "0.0.0.0"]
```

Размер `simple_python_app:v5-alpine` стал примерно в 2 раза меньше, чем сборка на основе Slim. Однако сборка приложения значительно отличается. Для дальнейшей работы рекомендуется выбрать образ на основе Slim.

6. Добавление пользователя, владельца приложения `simple_python_app`, отличного от `root`. Соберите образ с добавлением пользователя командой:

```
docker build -t simple_python_app:v6 .
```

Отобразите данные текущего пользователя в контейнере версии 6:

```
docker run --rm simple_python_app:v6 id
```

Отобразите данные текущего пользователя в контейнере версии 5:

```
docker run --rm simple_python_app:v5 id
```

Пример Dockerfile с добавлением пользователя:

```
FROM python:3.12.1 as builder
# -- Медленные операции --
#   Установка   системных   зависимостей   (в   т.ч.   для
зависимостей python)
RUN apt-get update \
    && apt-get install -y --no-install-recommends gcc \
    && rm -rf /var/lib/apt/lists/*

# Установка зависмостей python в диррекиторию /app/wheels
RUN                                                    --
mount=type=bind,source=requirements.txt,target=/app/requi
rements.txt \
        pip    wheel    --no-cache-dir    --no-deps    -r
/app/requirements.txt --wheel-dir /app/wheels
```

```
# ---------

FROM python:3.12.1-slim

# Копирование собранных файлов python из образа builder
COPY --from=builder /app/wheels /wheels

# Установка зависимостей, которые нужны для работы приложения
RUN apt-get update \
    && apt-get install -y --no-install-recommends libpq-dev \
    && rm -rf /var/lib/apt/lists/*

# Установка зависимости python, не пересобирая их
RUN pip install --no-cache --no-cache-dir /wheels/*

# Копирование кода приложения
COPY app.py /app/

# Добавление пользователя и назначение его на каталог приложения
RUN addgroup --system app \
  && adduser --system --group app \
  && chown -R app:app /app/
USER app

# Установка рабочей директории
WORKDIR /app

CMD ["uvicorn", "app:app", "--host", "0.0.0.0"]
```

Задание 4. Применение Docker Compose

1. Изучить руководства:
   - Руководство по Docker Compose для начинающих
   - Docker Compose в Docker Handbook
   - Docker Compose overview

2. Реализовать пример к заданию 4

Пример к заданию 4

В качестве примера используйте приложение `simple_python_app` из задания 3 и связанные с ним сервисы. Его компоненты:

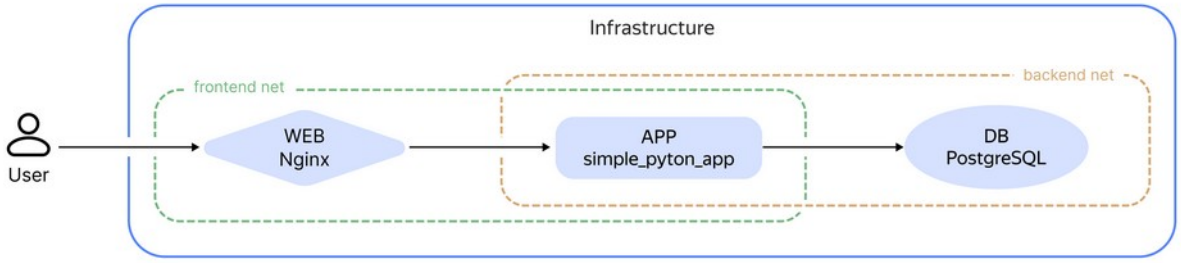


Figure 4: Ифраструктура приложения

Приложение содержит:

- **Веб-сервер** Nginx.
- **Бэкенд-сервис** Приложение `simple_python_app`.
- **Базу данных** PostgreSQL.

Настроим его оркестрацию с помощью Docker Compose. Для этого опишите инфраструктуру в манифесте `compose.yaml`, выполнив следующие действия:

1️⃣ Создайте каталог `task4/compose`. В нем создайте манифест `compose.yaml`, а также все необходимые файлы согласно инструкциям далее.

2️⃣ В каталоге `task4/compose` создайте директорию `simple_python_app` для исходного кода приложения.

3️⃣ В каталоге `task4/compose/simple_python_app` создайте файл с кодом приложения `app.py` со следующим содержимым:

```
from fastapi import FastAPI
import psycopg2
import os

app = FastAPI()

DATABASE_HOST = os.getenv("API_DB_HOST", "localhost")
DATABASE_PORT = os.getenv("API_DB_PORT", "5432")
```

```
    DATABASE_NAME = os.getenv("API_DB_NAME", "api")
    DATABASE_USER = os.getenv("API_DB_USER", "apiuser")
    DATABASE_PASS = os.getenv("API_DB_PASS", "apipass")

    DATABASE_URL     =     f"postgresql://{DATABASE_USER}:
{DATABASE_PASS}@{DATABASE_HOST}:
{DATABASE_PORT}/{DATABASE_NAME}"

    conn = psycopg2.connect(DATABASE_URL)
    cursor = conn.cursor()

    @app.get("/")
    async def root():
        cursor.execute(
            "SELECT version();"
        )
        item = cursor.fetchone()
        return {"message": "Hello World",
                "postgres_version": item[0]}

    @app.get("/hello/{name}")
    async def say_hello(name: str):
        return {"message": f"Hello {name}"}
```

4️⃣ Далее в каталоге task4/compose/simple_python_app создайте файл с зависимостями приложения requirements.txt:

```
    fastapi==0.110.2
    uvicorn[standard]==0.29.0
    psycopg2==2.9.9
```

5️⃣ В каталоге ~/compose/simple_python_app создайте Dockerfile:

```
FROM python:3.12.1 AS builder
# -- Редко изменяемые операции --
#  Установка системных зависимостей (в т. ч. для
зависимостей python)
RUN apt-get update \
    && apt-get install -y --no-install-recommends gcc \
    && rm -rf /var/lib/apt/lists/*
```

```
# Установка зависимостей python в директорию /app/wheels
RUN --mount=type=bind,source=requirements.txt,target=/app/requirements.txt \
    pip wheel --no-cache-dir --no-deps -r /app/requirements.txt --wheel-dir /app/wheels

# ---------

FROM python:3.12.1-slim

# Копирование собранных файлов python из образа builder
COPY --from=builder /app/wheels /wheels

# Установка зависимостей, которые нужны для работы приложения
RUN apt-get update \
    && apt-get install -y --no-install-recommends libpq-dev \
    && rm -rf /var/lib/apt/lists/*

# Устанавливаем зависимости python не пересобирая их
RUN pip install --no-cache --no-cache-dir /wheels/*

# Копирование кода приложения
COPY app.py /app/

# Добавление пользователя и назначение его на каталог приложения
RUN addgroup --system app \
  && adduser --system --group app \
  && chown -R app:app /app/
USER app

# Установка рабочей директории
WORKDIR /app

ENTRYPOINT ["uvicorn"]
```

```
CMD ["app:app", "--host", "0.0.0.0"]
```

Файлы для сборки образа с приложением готовы.

6️⃣ В каталоге task4/compose создайте каталог nginx.

7️⃣ В каталоге и `~/compose/nginx` создайте файл `app.conf` с конфигурацией сайта:

```
server {
  listen       80;
  server_name  localhost;

  error_page   500 502 503 504  /50x.html;
  location = /50x.html {
    root   /usr/share/nginx/html;
  }

  location / {
    proxy_pass   http://simple_python_app:8000;
    proxy_set_header Host $host;
                     proxy_set_header     X-Forwarded-For
$proxy_add_x_forwarded_for;
    proxy_set_header X-Real-IP $remote_addr;
  }
}
```

Конфигурация nginx готова.

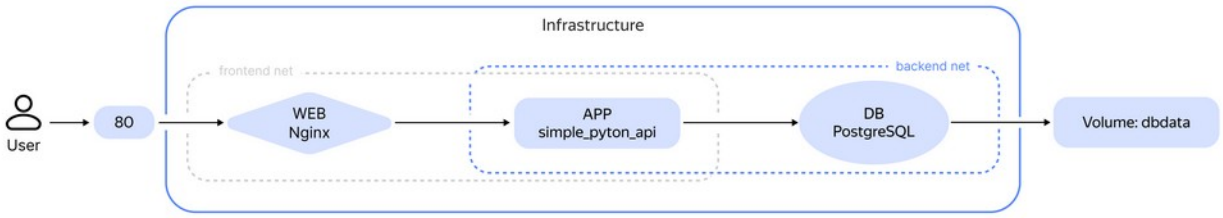
8️⃣Создайте манифест `compose.yaml` следующего вида:

```
name: simple-python-app
services:
  web:
    image: nginx:alpine
    ports:
      - "80:80"
    networks:
      - frontend-net
    volumes:
      - type: bind
        source: ./nginx/app.conf
        target: /etc/nginx/conf.d/default.conf
        read_only: true
    depends_on:
```

```
      - simple_python_app

  simple_python_app:
    image: simple_python_app:${APP_VERSION:-latest}
    build: ./simple_python_app
    environment:
      API_DB_HOST: db
      API_DB_PASS: apipass
      API_DB_NAME: api
      API_DB_USER: apiuser
    networks:
      - frontend-net
      - backend-net
    depends_on:
      db:
        condition: service_healthy
        restart: true

  db:
    image: postgres:16.2-alpine
    environment:
      POSTGRES_PASSWORD: apipass
      POSTGRES_DB: api
      POSTGRES_USER: apiuser
    volumes:
      - dbdata:/var/lib/postgresql/data
    networks:
      - backend-net
    healthcheck:
      test: ["CMD-SHELL", "pg_isready"]
      interval: 10s
      timeout: 5s
      retries: 10
      start_period: 30s
      start_interval: 1s

networks:
  frontend-net:
```

```
  backend-net:

volumes:
  dbdata:
```

Корневым параметром будет `name: simple-python-app`. Манифест `compose.yaml` реализует схему как на рисунке ниже.



9️⃣ Запустите проект с помощью команды
`docker compose up -d`
🔟 Откройте сайт в браузере.
1️⃣1️⃣ Остановите проект с помощью команды
`docker compose down -v`

Задание 5. Развернуть инфраструктуру проекта

Реализовать проект согласно варианту или проект по дисциплине ПЧМИ, если соответствует требованиям. Поднять БД в контейнере, подключить к ней ПО, создающее структуру БД посредствам системы миграции (liqubase), а также реализующее основные CRUD-операции в рамках предметной области. Для ПО сделать возможность локальной работы в базе, а также поднятие в докер-контейнере внутри одной докер сети с базой. Подключить swagger. Для каждого контейнера создать Dockerfile. Продемонстрировать управление контейнерами с помощью Docker Compose.

Продемонстрировать работу через него. У вас должна быть всего одна таблица в базе данных.

Варианты

Вар№	Задание
1.	Ведется учет читателей в библиотеке. Имеется возможность добавить читателя, изъять читательский билет, отредактировать и выдать книгу на

Вар№	Задание
	руки. Сформировать отчет о том, сколько всего читателей и сколько из них имеет книги на руках.
2.	Ведется учет студентов. Имеется возможность принять студента, отчислить, отредактировать и перевести на новый курс. Сформировать отчет о том, сколько студентов на каких курсах в данный момент обучаются.
3.	Ведется учет посетителей в прокате. Имеется возможность добавить посетителя, прекратить отношения с посетителем, отредактировать запись о посетителе и выдать диск на руки. Сформировать отчет о том, сколько посетителей имеют диски в данный момент на руках и сколько всего в зарегистрировано посетителей в прокате.
4.	Ведется учет сотрудников по организации. Имеется возможность принять человека
5.	Ведется учет животных в специальной гостинице. Имеется возможность принять животное на определенный срок; продлить срок пребывания; выписать домой. Сформировать отчет о том, сколько животных в данный момент в гостинице, а сколько выписаны домой.
6.	Ведется учет автомобилей в прокате. Имеется возможность принять новый автомобиль, выдать автомобиль на руки, списать автомобиль. Сформировать отчет о том, сколько автомобилей в данный момент находятся на руках, сколько списано, а сколько в салоне.
7.	Ведется учет поставщиков. Имеется возможность оформить контракт с новым поставщиком, закончить контракт с одним из имеющихся поставщиков, отредактировать информацию о поставщике. Сформировать отчет о количестве оформленных и прекращенных контрактов.
8.	Ведется учет компьютеров в магазине. Имеется возможность принять новый компьютер, оформить продажу имеющегося, отредактировать информацию. Сформировать отчет о количестве имеющихся и количестве проданных компьютеров.
9.	Ведется учет надзирателей. Имеется возможность принять нового надзирателя, уволить надзирателя, отредактировать информацию. Сформировать отчет о том, сколько надзирателей работает, а сколько уволено.
10.	Ведется учет посетителей салона красоты. Имеется возможность завести нового клиента, прекратить отношения с существующим, отредактировать запись. Сформировать отчет количестве клиентов и количестве бывших клиентов.

Вар№	Задание
11.	Ведется учет работников салона красоты. Имеется возможность принять работника, отредактировать данные, уволить. Сформировать отчет о том, сколько работников уволено, а сколько работает.
12.	Ведется учет школьников в кружке. Имеется возможность принять школьника в кружок, отчислить за неуспеваемость, отредактировать информацию. Сформировать отчет о том, сколько школьников посещают кружок, а сколько отчислено.
13.	Ведется учет клиентов турфирмы. Имеется возможность выписать тур клиенту, прекратить отношения с клиентом, отредактировать информацию. Сформировать отчет количестве выписанных туров и количестве ушедших клиентов.
14.	Ведется учет преподавателей ВУЗа. Имеется возможность принять человека на должность, уволить по определенной причине, назначить предмет для чтения. Сформировать отчет о том, сколько преподавателей работает, а сколько уволено.
15.	Ведется учет руководителей кружков. Имеется возможность принять нового человека, перевести из кружка в кружок, указать количество обучающихся в кружке. Сформировать отчет о том, сколько человек обучается у каждого из руководителей.
16.	Ведется учет работников ветклиники. Имеется возможность принять работника, внести данные о повышении квалификации, уволить. Сформировать отчет о том, сколько работников уволено, а сколько работает.
17.	Ведется учет автомобилей в автоцентре. Имеется возможность принять новый автомобиль, продать автомобиль, отредактировать информацию. Сформировать отчет о том, сколько автомобилей в салоне, а сколько продано.
18.	Ведется учет контрагентов. Имеется возможность завести нового контрагента и открыть с ним договор, закрыть существующий договор, назначить нового ответственного за договор менеджера. Сформировать отчет о том, сколько договоров действует, а сколько закрыты.
19.	Ведется учет книг и читателей в библиотеке. Имеется возможность выдать книгу на руки, выдать для чтения в читальном зале, получить возвращенную книгу. Сформировать отчет о том, сколько книг в данный момент находятся на руках, в библиотеке, в читальном зале.
20.	Ведется учет больных. Имеется возможность принять больного, назначить ему лечение, выписать домой. Сформировать отчет о том,

Вар№	Задание
	сколько больных в данный момент проходят лечение в больнице, а сколько выписано.
21.	Ведется учет студентов по специальностям. Имеется возможность принять студента на определенную специальность, отчислить за неуспеваемость, перевести со специальности на специальность. Сформировать отчет о том, сколько студентов и на каких специальностях в данный момент обучаются, а также сколько было отчислено.
22.	Ведется учет дисков в прокате. Имеется возможность выдать диск на руки, принять новый диск в прокат, получить диск обратно. Сформировать отчет о том, сколько дисков в данный момент на руках, а сколько в прокате
23.	Ведется учет сотрудников организации. Имеется возможность принять человека на определенную должность, уволить по определенной причине, повысить зарплату. Сформировать отчет о том, сколько сотрудников работают, а сколько уволено
24.	Ведется учет менеджеров. Имеется возможность принять менеджера, уволить, изменить количество заключенных им договоров. Сформировать отчет о том, сколько договоров, какой менеджер заключил.
25.	Ведется учет животных в специальной гостинице. Имеется возможность принять животное на определенный срок и прикрепить к нему работника; продлить срок пребывания; выписать домой. Сформировать отчет о том, сколько животных в гостинице, сколько отправлено домой.
26.	Ведется учет успеваемости учащихся техникума на потоке. Имеется возможность принять обучающегося, проставить среднюю оценку за предметы, отчислить. Сформировать отчет об успеваемости учащихся в порядке убывания.
27.	Ведется учет заключенных. Имеется возможность принять нового заключенного, отпустить заключенного домой, сменить надзирателя. Сформировать отчет о том, какие заключенные отбывают наказания и какие уже отбыли.

Контрольные вопросы

1. Что такое и зачем нужен Docker? Альтернативные системы?
2. Как получить Docker-образ, что это такое?
3. Как запустить контейнер? Как получить доступ к его портам?
4. Как просмотреть логи контейнера?

5. Как сохранить данные внутри контейнера между его перезапусками?

6. Как подключить контейнеры к одной сети? Какие есть альтернативные варианты?7. Почему контейнеры могут обращться между собой по имени (хэшу, если его нет)?

8. Что такое метки (docker tag)?

9. Как удалить ненужные образа и контейнеры?

10. Как запустить что-то внутри работающего контейнера?

11. Как узнать, какие файлы изменяет программа внутри контейнера?

12. Когда происходит завершение контейнера? Как сделать?

13. Перезапустите сборку собранного образа, оцените время пересборки, объясните

причины.

14. К какому число слоев стремиться в образе, правила оптимизации?

15. Опишите базовые команды Dockerfile, что они делают, где смотреть документацию?

16. Что такое контекст сборки, как его оптимизировать?

17. Основные возможности Docker Compose.