МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

СЕРГИЕНКО ЛЕВ ЭДУАРДОВИЧ

Отчет по ЛАБОРАТОРНАЯ РАБОТА 1 ПО ДИСЦИПЛИНЕ «Непрерывное интегрирование и сборка программного обеспечения»

Методические указания по выполнению лабораторной работы

Преподаватель

Давидовская М.И.
Филиппов М.А.

2025

Цель работы

Изучение современных технологий контейнеризации.

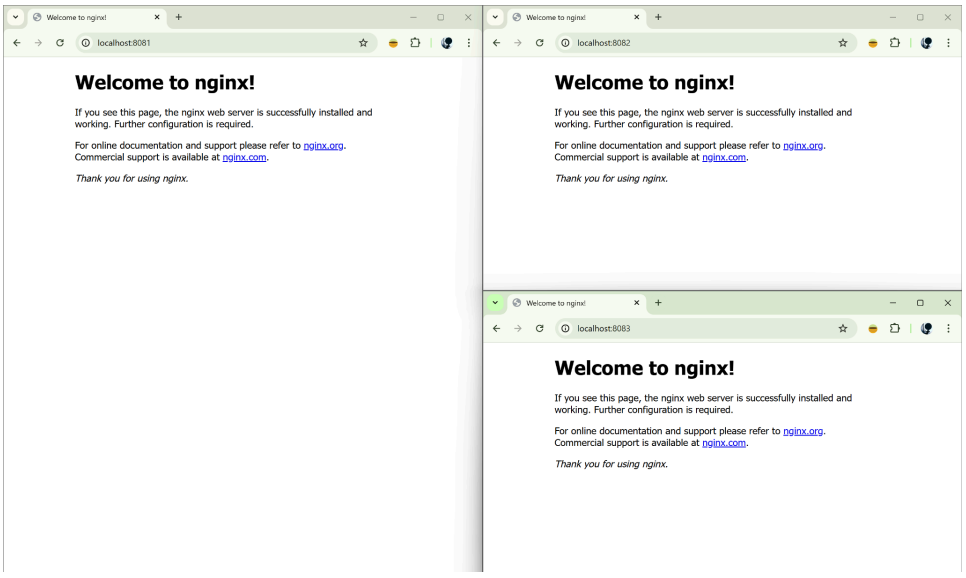
Код приложений, конфигурационных файлов.

Задание 1. Основы управления контейнерами и образами контейнеров Docker

2. 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

```
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP> docker run -d --name mynginxlast -p 8081:80 nginx:latest
 >> docker run -d --name mynginxalpine -p 8082:80 nginx:alpine
 >> docker run -d --name mynginx1-26 -p 8083:80 nginx:1.26
8cc7bbbcf496f3026fd2e53bb157fec3ffbb7fae5e68998badb5b0054fc242bd
07d2dd4c6b6737abdd88c76d91226cd5e4b134474b2240c2a163636a20945427
09221356af806bac03a02dc5e8f0f516d7a98ffcace97501e98eb32e4a6cd89d
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                    NAMES
09221356af80   nginx:1.26     "/docker-entrypoint.…"   3 seconds ago   Up 1 second    0.0.0.0:8083->80/tcp     mynginx1-26
07d2dd4c6b67   nginx:alpine   "/docker-entrypoint.…"   3 seconds ago   Up 2 seconds   0.0.0.0:8082->80/tcp     mynginxalpine
8cc7bbbcf496   nginx:latest   "/docker-entrypoint.…"   3 seconds ago   Up 2 seconds   0.0.0.0:8081->80/tcp     mynginxlast
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP>
```



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;

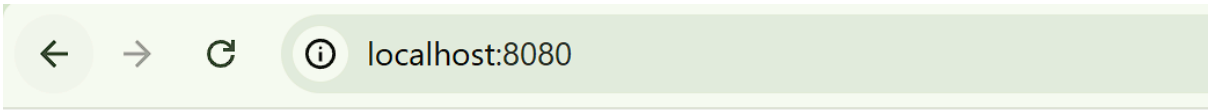
```
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP> docker run -d --name mynginxlast nginx:latest
e5864c6509e7ce287d930dc9ca3699d423478faec036aaa2c50d5625ff998561
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP> docker exec -it mynginxlast /bin/bash
root@e5864c6509e7:/# exit
exit
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP>
```

- запуск контейнера mynginxlast в интерактивном режиме и выход из него с помощью команды exit.

```
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP> docker run -it --name mynginxlast nginx:latest /bin/bash
root@cc3091f49450:/# exit
exit
```

- В каталоге репозитория лабораторной работы создать каталог, например task1. В данном каталоге создать простой html документ (веб-страницу) содержащий теги. Продемонстрировать монтирование каталога task1 как тома контейнера, например mynginxalpine, который является корневым каталогом для виртуального хоста (сайта) nginx.

```
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP> docker run -d --name mynginxalpine -v .\task1:/usr/share/nginx/html -p 8080:80 nginx:alpine
a0569cc5780842235f48e2d3c96cf47044a68b8de438f36e67f17ee889b79ded
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP>
```



Лабораторная работа 1

ФИО: Sergienko Lev E

Группа: 12 MSS

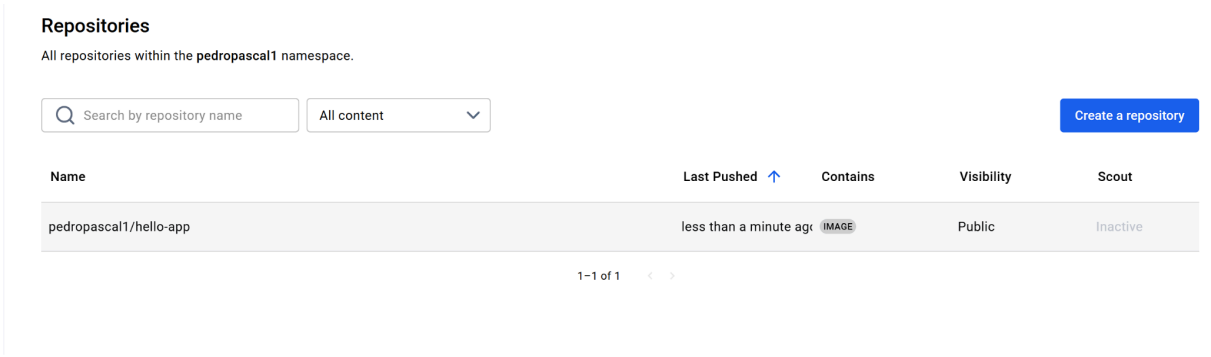
Задание 2. Создание и публикация образа контейнера в репозиторий (реестр) контейнеров

2. Собрать собственный образ на основе пункта 3 из задания 1 и опубликовать в репозиторий Docker Hub.

```
build:
    docker build -t pedropascal1/hello-app:1.0 .

start:
    docker run -d -p 8081:80 --name mypage pedropascal1/hello-app:1.0

push:
    docker push pedropascal1/hello-app:1.0
```



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

1. Создать приложением с хранением данных в базе данных, работающей в контейнере Docker. Создать образ контейнера с приложением на основе примера.

```
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP\task3\simple_python_app> 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
Unable to find image 'postgres:16.2-alpine' locally
16.2-alpine: Pulling from library/postgres
4abcf2066143: Download complete
904e95badb7d: Download complete
8f6103a2e811: Download complete
128d1b74d24d: Download complete
696a345da38f: Download complete
0a392327555d: Download complete
0e14a31643e8: Download complete
02b106837f9f: Download complete
d34b010d3edc: Download complete
Digest: sha256:951bfda460300925caa3949eaa092ba022e9aec191bbea9056a39e2382260b27
Status: Downloaded newer image for postgres:16.2-alpine
c2406ccc45ef32004f8e06e6b52809966209bee88088486be0652dc8cbeeb593
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP\task3\simple_python_app> docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED          STATUS          PORTS                    NAMES
c2406ccc45ef   postgres:16.2-alpine   "docker-entrypoint.s…"   10 seconds ago   Up 8 seconds   0.0.0.0:5432->5432/tcp   postgres-db
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP\task3\simple_python_app>
```

запуск базы

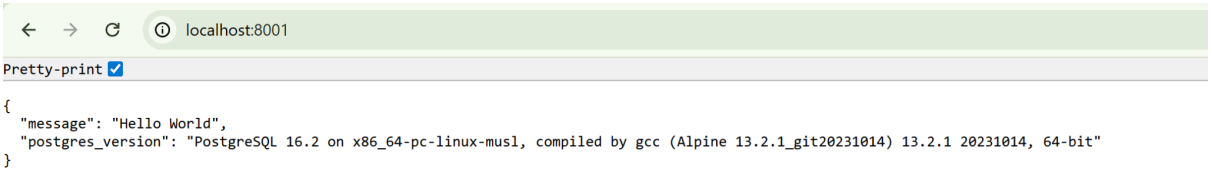
```
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP\task3\simple_python_app> docker build -t my_app .
[+] Building 2.7s (11/11) FINISHED                                                                                                  docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                                              0.0s
 => => transferring dockerfile: 230B                                                                                                              0.0s
 => [internal] load metadata for docker.io/library/python:3.11                                                                                    1.7s
 => [auth] library/python:pull token for registry-1.docker.io                                                                                     0.0s
 => [internal] load .dockerignore                                                                                                                 0.0s
 => => transferring context: 2B                                                                                                                   0.0s
 => [internal] load build context                                                                                                                 0.0s
 => => transferring context: 419B                                                                                                                 0.0s
 => [1/5] FROM docker.io/library/python:3.11@sha256:3afa472a60099bc640f313807190b16e2cecfadfc8302d9c3fd67e7706db54e5                              0.0s
 => => resolve docker.io/library/python:3.11@sha256:3afa472a60099bc640f313807190b16e2cecfadfc8302d9c3fd67e7706db54e5                              0.0s
 => CACHED [2/5] COPY . /app/                                                                                                                     0.0s
 => CACHED [3/5] RUN apt-get update && apt-get install -y gcc                                                                                     0.0s
 => CACHED [4/5] RUN pip install -r /app/requirements.txt                                                                                         0.0s
 => CACHED [5/5] WORKDIR /app                                                                                                                     0.0s
 => exporting to image                                                                                                                            1.0s
 => => exporting layers                                                                                                                           0.0s
 => => exporting manifest sha256:88aca6e419b95044c2615bd46443f4933de6a3442d956caa804168d237fa75d5                                                 0.0s
 => => exporting config sha256:e42fb816d209a316e9ddad529d5d985dfb08562db2871a5fc248a61f07b23612                                                   0.0s
 => => exporting attestation manifest sha256:a9355a24c3cde4ac46f7ff86fe1709ff696648dfe790fb8e6678051673fac169                                     0.0s
 => => exporting manifest list sha256:403efb861864ef18216018325d0e6508dbff9f71d0b487c5d86a3ad49e4568be                                           0.0s
 => => naming to docker.io/library/my_app:latest                                                                                                  0.0s
 => => unpacking to docker.io/library/my_app:latest                                                                                               0.9s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/py1vzbvp7tnzm0520zd608pvg
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP\task3\simple_python_app>
```

Собираем образ

```
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP\task3\simple_python_app> docker run -e API_DB_HOST=host.docker.internal -p 8001:8000 my_app
INFO:     Started server process [1]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

запуск приложения




```
{
  "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"
}
```


Результат

2. Примените разные техники оптимизации образа контейнера Docker

```
docker build -t simple_python_app:v1 -f ./simple_python_app/Dockerfile ./simple_python_app
docker build -t simple_python_app:v2 -f ./simple_python_app/Dockerfile.v2 ./simple_python_app
docker build -t simple_python_app:v3 -f ./simple_python_app/Dockerfile.v3 ./simple_python_app
docker build -t simple_python_app:v4 -f ./simple_python_app/Dockerfile.v4 ./simple_python_app
docker build -t simple_python_app:v5 -f ./simple_python_app/Dockerfile.v5 ./simple_python_app
docker build -t simple_python_app:v5.alpine -f ./simple_python_app/Dockerfile.v5.alpine ./simple_python_app
docker build -t simple_python_app:v6 -f ./simple_python_app/Dockerfile.v6 ./simple_python_app
docker image ls simple_python_app
```

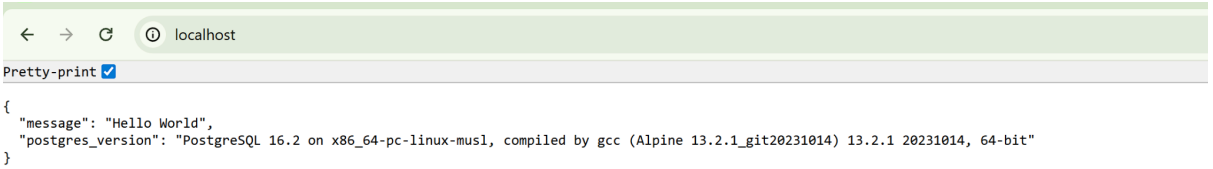
```
REPOSITORY          TAG          IMAGE ID       CREATED                  SIZE
simple_python_app   v6           ef69fd71c80e   Less than a second ago   252MB
simple_python_app   v5.alpine    e175e9a88454   4 seconds ago            134MB
simple_python_app   v5           fc8150aeb1a2   43 seconds ago           252MB
simple_python_app   v4           9c9b7b2247da   About a minute ago       1.62GB
simple_python_app   v2           1eb6e854c413   About a minute ago       1.74GB
simple_python_app   v1           cda3cec84fdb   About a minute ago       1.74GB
simple_python_app   v3           f32c3f5bd1cf   3 minutes ago            1.74GB
```

Задание 4. Применение Docker Compose

1. Реализовать пример к заданию 4

```
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP\task4\compose> docker compose up -d
[+] Running 5/5
 ✓ Network simple-python-app_backend-net           Created                                                0.0s
 ✓ Network simple-python-app_frontend-net          Created                                                0.0s
 ✓ Container simple-python-app-db-1                Healthy                                                2.1s
 ✓ Container simple-python-app-simple_python_app-1 Started                                                2.4s
 ✓ Container simple-python-app-web-1               Started                                                2.7s
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP\task4\compose>
```

запуск приложения



тест

```
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP\task4\compose> docker compose down
 [+] Running 5/5
  ✓ Container simple-python-app-web-1               Removed                                               10.5s
  ✓ Container simple-python-app-simple_python_app-1 Removed                                                0.9s
  ✓ Container simple-python-app-db-1                Removed                                                0.5s
  ✓ Network simple-python-app_backend-net           Removed                                                0.3s
  ✓ Network simple-python-app_frontend-net          Removed                                                0.5s
PS C:\Users\lev\Desktop\devops\devops-gr12b-lab1-foreverNP\task4\compose>
```

down

Задание 5. Развернуть инфраструктуру проекта

В качестве 5 задания хочу защитить свою курсовую работу, в рамках которой был спроектирован и разработан полноценный сервис Mermate для генерации диаграмм по текстовому запросу пользователя с помощью ИИ.

В рамках проекта было разработано: 2 backend сервиса на python и golang, frontend на vue. Использованы 2 бд, redis и mongo, nginx как прокси, grafana, loki, alloy для мониторинга.

Все собиралось и запускалось с помощью докера.

Привожу docker compose:

```
services:
    nginx:
        image: nginx:1.28.0-alpine
        container_name: nginx
        ports:
            - "8080:80"
        volumes:
            -
./nginx/templates/default.conf.template:/etc/nginx/templates/default.conf.tem
plate
            - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
        environment:
            - APP_PORT=${APP_PORT}
        depends_on:
            app:
                condition: service_healthy
        networks:
            - app_network
        restart: unless-stopped

    ################################ The application services
################################
    app:
        build:
            context: ../.
            dockerfile: Dockerfile
        container_name: mermate
        depends_on:
            redis:
                condition: service_healthy
            mongo:
                condition: service_healthy
        healthcheck:
            test: ["CMD", "wget", "-qSO-",
"0.0.0.0:${APP_PORT}/healthz"]
            interval: 30s
            timeout: 10s
            retries: 3
            start_period: 5s
        environment:
            - APP_NAME=${APP_NAME}
            - APP_PORT=${APP_PORT}
            - MONGO_HOST=mongo
```

```
                - MONGO_PORT=27017
                - MONGO_USERNAME=${MONGO_USERNAME}
                - MONGO_PASSWORD=${MONGO_PASSWORD}
                - MONGO_DATABASE=${MONGO_DATABASE}
                - MONGO_CONNECT_TIMEOUT=${MONGO_CONNECT_TIMEOUT}
                - REDIS_ADDR=redis:6379
                - REDIS_PASSWORD=${REDIS_PASSWORD}
                - AUTH_SECRET=${AUTH_SECRET}
                - AUTH_ACCESS_TOKEN_TTL=${AUTH_ACCESS_TOKEN_TTL}
                - AUTH_REFRESH_TOKEN_TTL=${AUTH_REFRESH_TOKEN_TTL}
                - USER_BCRYPT_COST=${USER_BCRYPT_COST}
                - AI_SERVICE_ADDRESS=ai_service:50051
            volumes:
                - app_logs:/var/log/mermate
            networks:
                - app_network
            restart: unless-stopped

        ai_service:
            build:
                context: ../ai_service
                dockerfile: Dockerfile
            container_name: ai_service
            environment:
                - AI_SERVICE_GH_ENDPOINT=${AI_SERVICE_GH_ENDPOINT}
                - AI_SERVICE_GH_API_KEY=${AI_SERVICE_GH_API_KEY}
                - AI_SERVICE_GG_API_KEY=${AI_SERVICE_GG_API_KEY}
                - AI_SERVICE_SERVER_PORT=50051
            volumes:
                - ai_service_logs:/var/log/ai_service
            networks:
                - app_network
            restart: unless-stopped

        ############################### Database services
###############################
        redis:
            image: redis:7.4.2
            container_name: redis
            volumes:
                - redis_data:/data
            command: ["redis-server", "--requirepass",
"${REDIS_PASSWORD}"]
            networks:
                - app_network
```

```
            healthcheck:
                test: ["CMD", "redis-cli", "-a", "${REDIS_PASSWORD}",
"ping"]
                interval: 30s
                timeout: 10s
                retries: 3
                start_period: 5s
            restart: unless-stopped

        mongo:
            image: mongo:8.0.6
            container_name: mongo
            volumes:
                - mongo_data:/data/db
            environment:
                MONGO_INITDB_ROOT_USERNAME: ${MONGO_USERNAME}
                MONGO_INITDB_ROOT_PASSWORD: ${MONGO_PASSWORD}
            networks:
                - app_network
            healthcheck:
                test:
                    [
                        "CMD",
                        "mongosh",
                        "--username",
                        "${MONGO_USERNAME}",
                        "--password",
                        "${MONGO_PASSWORD}",
                        "--authenticationDatabase",
                        "admin",
                        "--quiet",
                        "--eval",
                        "db.adminCommand('ping')",
                    ]
                interval: 30s
                timeout: 10s
                retries: 3
                start_period: 5s
            restart: unless-stopped

        ################################ Observability services
################################
        grafana:
            image: grafana/grafana:11.6.0
            container_name: grafana
```

```
            ports:
                - "3000:3000"
            environment:
                - GF_SECURITY_ADMIN_USER=${GRAFANA_ADMIN_USER}
                - GF_SECURITY_ADMIN_PASSWORD=${GRAFANA_ADMIN_PASSWORD}
                - GF_USERS_ALLOW_SIGN_UP=false
            volumes:
                - ./grafana/provisioning:/etc/grafana/provisioning
            depends_on:
                - loki
            networks:
                - app_network
            restart: unless-stopped

        alloy:
            image: grafana/alloy:v1.8.1
            container_name: alloy
            volumes:
                - ./alloy/config.alloy:/etc/alloy/config.alloy
                - app_logs:/var/log/mermate:ro
                - ai_service_logs:/var/log/ai_service:ro
            command: run --server.http.listen-addr=0.0.0.0:12345
--storage.path=/var/lib/alloy/data /etc/alloy/config.alloy
            networks:
                - app_network
            depends_on:
                app:
                    condition: service_healthy
            restart: unless-stopped

        loki:
            image: grafana/loki:2.9.14
            container_name: loki
            volumes:
                - loki_data:/loki
                - ./loki/loki-config.yaml:/etc/loki/local-config.yaml
            command: -config.file=/etc/loki/local-config.yaml
            networks:
                - app_network
            restart: unless-stopped

    volumes:
        redis_data:
        mongo_data:
        loki_data:
```

```
        app_logs:
        ai_service_logs:

networks:
    app_network:
        driver: bridge
```

Ответы на контрольные вопросы.

1. Что такое и зачем нужен Docker? Альтернативные системы?

Docker - движок контейнеризации: упаковывает приложение + зависимости в лёгкие изолированные контейнеры для воспроизводимости, портируемости и быстрой доставки. Даёт изоляцию процессов, сеть, файловую систему и образную модель.

Альтернативы: Podman (daemonless, совместим с Docker CLI), containerd/CRI-O (рантаймы), LXC/LXD (полноценные контейнеры/с реальной init), systemd-nspawn.

2. Как получить Docker-образ, что это такое?

Образ - слоистый read-only шаблон файловой системы + метаданные (команда запуска, переменные и т.д.). Получить: docker pull repo/image:tag (скачать из registry) или docker build -t myimage:tag . (собрать из Dockerfile).

3. Как запустить контейнер? Как получить доступ к его портам?

Запуск: docker run [опции] image - примеры: -d, --name, -e, -v. Порты: проброс с хоста -p <hostPort>:<containerPort>; при создании пользовательской сети контейнеры общаются по внутреннему IP/имени; --network host даёт доступ к портам хоста.

4. Как просмотреть логи контейнера?

docker logs <container>; -f - follow, --tail N - последние N строк.

5. Как сохранить данные внутри контейнера между его перезапусками?

Использовать Volume или bind-mount: -v /host/path:/container/path (bind) или -v myvolume:/container/path (named volume). Volumes - предпочтительны для персистентных данных.

6. Как подключить контейнеры к одной сети? Какие есть альтернативные варианты?

Создать сеть: docker network create mynet и запускать с --network mynet или docker network connect. По умолчанию есть bridge; альтернативы: host, overlay (для swarm/кластера), macvlan. В Compose сети задаются автоматически и удобно управляются.

7. Почему контейнеры могут общаться между собой по имени (хэшу, если его нет)?

Потому что Docker встроил DNS-резолвер для пользовательских сетей: имя контейнера/сервиса становится DNS-записью, Docker internal DNS возвращает IP. Если имени нет, можно использовать короткий ID (хэш) - Docker разрешает имена и частичные ID в CLI.

8. Что такое метки (docker tag)?

tag - метка изображения в формате repository:tag (напр. nginx:1.25). docker tag связывает локальный image ID с читаемым именем/тегом. Теги помогают версионировать/идентифицировать образы.

9. Как удалить ненужные образа и контейнеры?

Остановить/удалить контейнер: docker stop <c> -> docker rm <c>. Удалить образ: docker rmi <image>.

10. Как запустить что-то внутри работающего контейнера?

docker exec -it <container> /bin/sh или /bin/bash - интерактивная сессия. Одноразовая команда: docker exec <container> mycommand.

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

docker diff <container> покажет изменённые / добавленные / удалённые файлы относительно образа.

12. Когда происходит завершение контейнера? Как сделать?

Контейнер завершится, когда завершится его главный процесс (PID 1). Остановить принудительно: docker stop <c>, docker kill <c> (SIGKILL).

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

Как перезапустить: docker build . (или docker build --no-cache . для полной пересборки). Время пересборки зависит от: хит-кэша слоёв (если слой не изменился, Docker использует кэш - быстро), тяжести шагов (установка пакетов, скачивание ресурсов), размера контекста и IO. Полная пересборка (--no-cache) = суммарное время всех шагов; при удачном кэше пересборка может выполняться за секунды/минуты, иначе - десятки минут.

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

Нет жёсткого числа; цель - разумное количество и минимальный размер. Правила: объединять команды RUN там, где это логично (сохранение кэша и уменьшение финального числа слоёв), использовать многоступенчатую сборку (multi-stage) для уменьшения

финального артефакта, очищать кеши в том же RUN (например apt-get clean && rm -rf /var/lib/apt/lists/*), минимизировать контекст и ненужные файлы, использовать официальные slim-базы.

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

FROM - базовый образ;
RUN - выполняет команды при сборке;
CMD - команда по умолчанию при запуске контейнера (можно переопределить);
ENTRYPOINT - фиксированная точка входа;
COPY / ADD - копируют файлы (ADD умеет распаковывать URL/архивы);
EXPOSE - метаданные про порты;
ENV - переменные окружения;
WORKDIR - рабочая директория;
USER - переключить пользователя;
VOLUME - объявить точку для монтирования;
ARG - аргументы сборки;
LABEL - метаданные;
HEALTHCHECK - проверка здоровья.

Документация: официальная страница Dockerfile reference в документации Docker.

16. Что такое контекст сборки, как его оптимизировать?

Контекст сборки - это набор файлов/папок (обычно текущая директория), которые отправляются демон-серверу Docker при docker build. Всё, что не в контексте, недоступно для COPY/ADD. Оптимизация: минимизировать размер контекста, исключая лишние файлы через .dockerignore (node_modules, .git, большие артефакты), располагать Dockerfile в компактном каталоге, копировать только нужные файлы в образ (COPY с конкретными путями).

17. Основные возможности Docker Compose.

Compose - инструмент для определения и запуска многоконтейнерных приложений через docker-compose.yml. Возможности: декларативное описание сервисов, сетей и томов; build или image; ports, volumes, environment, depends_on; удобные команды up, down, logs, exec; профиль/override файлов и интеграция с переменными окружения. Отлично для локальной разработки и простого оркестрования.