5. СОЗДАНИЕ КОНТЕЙНЕРА DOCKER. РАБОТА С DOCKER COMPOSE

Изучая данный раздел программного обеспечения, студенты должны:

- изучить синтаксис *Dockerfile*;
- изучить синтаксис *Docker Compose*;
- составить файл типа *docker-compose*, содержащий два сервиса: один для клиент-серверного приложения, а второй – для базы данных;
- протестировать корректность работы приложения;
- настроить возможность подключения с личного компьютера к приложению, запущенному на виртуальной машине.

Исходные данные: виртуальный сервер, выбранный по указанию преподавателя.

1. Виртуализация и контейнеризация: Hyper-V, Docker и Kubernetes

Виртуализация — это технология создания изолированных виртуальных машин (ВМ), каждая из которых эмулирует полноценный компьютер с собственной операционной системой (ОС), приложениями и ресурсами.

Контейнеризация — это технология изоляции приложений на уровне операционной системы, где контейнеры разделяют ядро хостовой ОС, но имеют собственные пространства процессов, файловые системы и сетевые интерфейсы.

Ниже представлена сравнительная таблица некоторых характеристик виртуализации и контейнеризации:

Характеристика	Виртуализация	Контейнеризация
Уровень изоляции	Полная изоляция	Изоляция на уровне процессов
Ресурсоемкость	Высокая (гигабайты на ВМ)	Низкая (мегабайты на контейнер)
Время запуска	Длительное (минуты)	Быстрое (секунды)

Образ системы	Полная ОС со всеми компонентами	Только необходимые зависимости приложения
Управление	Отдельное управление каждой ВМ	Централизованное управление

1.1 Технологии виртуализации

Гипервизор (монитор виртуальных машин, VMM) — это специализированное программное обеспечение, которое позволяет запускать несколько операционных систем на одном физическом устройстве-сервере или компьютере. Физическое устройство, на котором установлен гипервизор, называется хостом или хост-машиной. Виртуальные машины, работающие на нём, — это гостевые системы.

Hyper-V — это технология виртуализации от Microsoft, гипервизор типа 1 (bare-metal), который работает непосредственно на аппаратном обеспечении сервера. Технология встроена в Windows Server и в клиентские версии Windows, начиная с Windows 8. При создании контейнеров *Docker* возможно создание с изоляцией *Hyper-V*.

1.2 Технологии контейнеризации

Наиболее популярными технологиями контейнеризации на данный момент являются *Docker* и *Kubernetes*.

Docker — это платформа, предназначенная для развертывания и управления контейнерными приложениями. Контейнеры позволяют упаковать приложение в единый образ, который можно запускать на любой системе, поддерживающей *Docker*. Это решает проблему несовместимости окружений, так как контейнеры гарантируют одинаковые условия работы программы независимо от используемой операционной системы или аппаратного обеспечения.

У *Docker* есть собственный встроенный оркестратор *Docker Swarm*, который позволяет управлять кластером как единым логическим узлом. Другим популярным инструментом оркестрации является *Docker Compose*. *Docker Compose* предназначен для управления контейнерами на одном хосте, а *Docker Swarm* – для управления контейнерами на нескольких хостах.

Kubernetes, в отличие от *Docker*, является не платформой, а наиболее популярным оркестратором, который автоматизирует задачи. Например, распределение нагрузки, восстановление, масштабирование и организация обновлений. Хотя для сложных сценариев оркестрации чаще выбирают *Kubernetes*, *Swarm* остается популярным решением благодаря своей простоте и тесной интеграции с другими сервисами *Docker*.

Docker и *Kubernetes* не являются взаимоисключающими технологиями, а часто использующимися вместе в едином рабочем пространстве (workflow). Для начала лучше всего освоить *Docker* и *Docker Compose* и разобраться в *Docker Swarm* для понимания базовых принципов оркестрации, а затем перейти к *Kubernetes* при необходимости развертываний с требованиями к масштабированию и отказоустойчивости.

В рамках данной лабораторной работы предлагается освоить *Docker* и *Docker Compose*, а также создать простое мультиконтейнерное приложение.

2. Установка программного обеспечения для работы с *Docker*

Запуском и установкой контейнеров можно управлять либо с помощью консоли, либо с помощью Docker Desktop, если работа ведется на Windows или macOS (Приложение 6). Если некоторые команды не работают, то перед `docker` добавьте `sudo` (например, `sudo docker info`).

Организационные команды:

- команда `docker` является основной командой *Docker*, выдающей информацию и справку по другим командам;

- команда `docker -v` осуществляет проверку версии / установлен ли *Docker* на компьютере;
- команда `docker info` выводит информацию о системе Docker, например, указывает на местоположение файловой структуры, в которой докер хранит все образы (поле *Docker Root Dir*).

Существует два основных подхода по работе с контейнерами:

1. Создание и запуск одиночного контейнера, описанного в *Dockerfile*. Для этого нужно использовать команды *Docker*.
2. Построение мультиконтейнерных приложений (например, клиент-серверное приложение и база данных). Для этого, в простом случае, нужно использовать команды *Docker Compose*.

Основные команды Docker для управления одиночным контейнером:

- команда `docker ps` показывает список запущенных контейнеров. Для просмотра полного списка в конце строки добавляется команда `-a`;
- команда `docker build .` собирает образ. Точка означает, что *Dockerfile* находится в той же директории, из которой запущена команда;
- команда `docker start [контейнер1] [контейнер2] [контейнер3]` запускает ранее созданные остановленные контейнеры. Используется имя контейнера или его ID;
- команда `docker stop [контейнер1] [контейнер2] [контейнер3]` останавливает ранее запущенные контейнеры. При этом данные контейнеров и сами контейнеры не удаляются;
- команда `docker stop $(docker ps -q)` останавливает все запущенные контейнеры;
- команда `docker container prune` удаляет все остановленные контейнеры.

Некоторые команды *Docker Compose* для управления мультиконтейнерными приложениями:

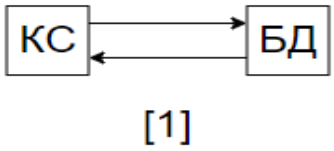
- команда `docker compose ps` отображает текущее состояние контейнеров;
- команда `docker compose up` собирает и запускает приложение со всеми контейнерами и образами с помощью файла *docker-compose.yml*, в котором контейнеры определены и объединены в одно приложение. Если файл не указан, то по умолчанию используется файл в текущем каталоге;
- в команде `docker compose up --build` флаг *--build* указывает *Docker Compose* на то, что сначала нужно пересобрать образы контейнеров перед их запуском. Это необходимо, если внесены изменения в *Dockerfile* или в файлы контекста сборки и нужно убедиться, что образы будут обновлены до запуска контейнеров. Без этого флага *Docker Compose* будет использовать существующие образы, если они уже были собраны;
- команда `docker compose start` запускает остановленные контейнеры;
- команда `docker compose stop` останавливает работу запущенных контейнеров без их удаления;
- команда `docker compose down` останавливает работу контейнеров и удаляет контейнеры;
- в команде `docker compose down -v` флаг *-v* указывает, что необходимо удалить *volumes* для данных контейнеров. Если не удалять *volumes*, то дальнейшее создание и пересборка контейнеров могут вызывать ошибку. Например, может не создаться новая база данных и не выполниться скрипт заполнения таблицы, так как система будет видеть по ней старые данные. При этом не будет доступа к данным, следовательно, нельзя будет выполнить запросы.

Работа с *Docker Compose* используется чаще, чем просто с *Docker*, так как на практике редко имеется всего один контейнер для работы. Чаще всего есть несколько контейнеров, которые связаны и взаимодействуют между собой. В рамках данной лабораторной работы представлен вариант мультиконтейнерного приложения.

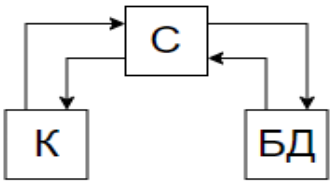
3. Создание клиент-серверного приложения и базы данных

Чтобы запустить контейнер, необходимо создать приложение, а затем прописать его сборку через файлы *Dockerfile* и *docker-compose.yml*. Можно объединять несколько контейнеров между собой с помощью портов и поля *depends_on*. При этом в каждом из контейнеров может быть своя среда, разные языки программирования и разные назначения.

Существуют два варианта работы с клиент-серверными приложениями:



[1]



[2]

[1] Простой вид, где один контейнер является приложением клиент-сервер, а второй – базой данных. Контейнер КС взаимодействует с БД, отправляя в нее запросы, получает их и выводит на экран.

[2] Вид, применяемый на практике. Обычно приложения клиента и сервера отличны друг от друга. К принимает запрос пользователя и отправляет его на С, где С взаимодействует с БД, а затем высылает ответ К. Данный вид более верный и безопасный, так как пользователь не взаимодействует напрямую с БД.

Для лабораторной работы будет достаточно вида №1.

В лабораторной работе используется связка Node.js и PostgreSQL. Структура папки общего приложения может выглядеть следующим образом:

- папка *app*, содержащая клиент-серверное приложение на Node.js;
    - файл *Dockerfile* в папке *app* для основного приложения;
- папка *db*, содержащая скрипт для создания таблиц и их заполнения;
- файл *.env*, содержащий переменные окружения для создания образа базы данных;
- основной файл *docker-compose.yml* для сборки *Docker*-приложения.

3.1 Создание базы данных

Для работы с PostgreSQL необходимо создать образ базы данных, а затем заполнить ее с помощью скрипта .sql. Образ будет создан в пункте 3. Скрипт должен содержать создание таблиц и их первоначальное заполнение.

Для данной лабораторной работы необходимо создание базы данных PostgreSQL. Ниже приведен пример скрипта для создания таблицы и ее заполнения:

```
CREATE TABLE IF NOT EXISTS users
(
     idu SERIAL PRIMARY KEY,
     login VARCHAR(30),
     pass VARCHAR(30),
     lastname VARCHAR(30),
     nameu VARCHAR(30),
     patronymics VARCHAR(30),
     dateofregistration TIMESTAMP,
     datelastconnect TIMESTAMP,
     statusu VARCHAR(30)
);
INSERT INTO users (login, pass, lastname, nameu, patronymics,
dateofregistration, datelastconnect, statusu)
VALUES
     ('kutuzova.ia', 'admin', 'Кутузова', 'Ирина', 'Александровна',
     '2021-04-06 19:00:14', '2021-04-06 19:00:14', 'admin'),
     ('mr. Goose', 'NotPavlin', 'Некий', 'Гусь', '-', '2021-04-06
     19:06:53', '2021-04-06 19:06:53', 'user');
```

В практическом применении *Docker* возможно использование образа любой базы данных (MySQL, MongoDB) и работа с ним.

3.2 Создание клиент-серверного веб-приложения

Для создания веб-приложения скачайте необходимые библиотеки, перейдя в папку *app*. Для каждого языка или надстройки все проверяется индивидуально. Возможно скачивание пакетов с помощью графического интерфейса Visual Studio или работа с Visual Studio Code. Перед переносом

приложения на виртуальный сервер, можно создать и проверить его на рабочем компьютере.

Для работы в Visual Studio Code на Node.js:

1) Проверьте наличие версий Node.js в системе командой `nvm list`. Если версии не найдены или команда не работает, то установить можно двумя способами: скачать с официального сайта nodejs.org или проверить наличие версий в папке C:\Windows\system32 командой `nvm list available`, а затем установить любую версию командой `nvm install номер_версии`.

2) Установите новые пакеты с помощью установщика пакетов `npm` (node package manager), перейдя в папку *app* и введя команду `npm install список_пакетов`, например `npm install pg express body-parser`.

3) Библиотеки автоматически добавятся в папку *app/node_modules*, а также обновятся два файла: *package.json* и *package-lock.json*.

Для работы с каждой базой данных используется свой пакет: например, для PostgreSQL - пакет *pg*, для работы с MySQL – пакет *mysql2*.

3.3 Соединение с базой данных

Создайте объект подключения, а затем укажите данные соединения:

```
// Пакет для работы с PostgreSQL
const {Pool} = require(«pg»);
// Пул соединения с базой данных
const pool = new Pool({
     // Название хоста базы данных
     host: process.env.DB_HOST,
     // Название базы данных
     database: process.env.DB_NAME,
     // Имя пользователя PostgreSQL
     user: process.env.DB_USER,
     // Пароль для пользователя
     password: process.env.DB_PASSWORD,
     // Порт 5432 – стандартный порт PostgreSQL, 3308 – для MySQL
     port: 5432
});
```

DB_HOST, DB_USER и другие – переменные окружения, которые прописываются в файле окружения *.env*:

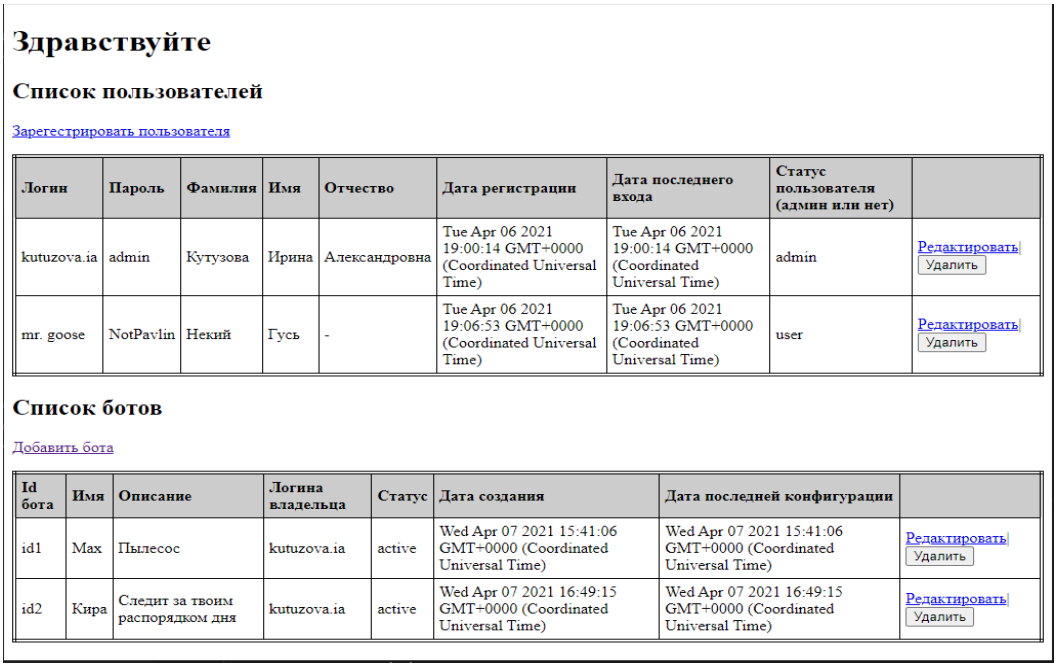
```
DB_NAME = useradmin
DB_USER = admin
DB_PASSWORD = admin
DB_HOST = db
```

Необходимо указать название базы данных, имя пользователя, его пароль, а также наименование хоста, которое позже будет использовано в *docker-compose.yml*. Созданный образ базы данных будет содержать то, что указано в файле переменных окружения. Файл переменных окружения обычно находится в той же папке, где и файл *docker-compose.yml*.

3.4 Обработка запросов к базе данных

В приложении необходимо реализовать 2-3 веб-страницы, а также добавить через них возможность добавления, изменения, удаления данных из базы данных пользователем. Например, это может быть веб-приложение администратора умного дома, в котором можно зарегистрироваться, авторизоваться, посмотреть список пользователей и их ботов, а также добавить, удалить или редактировать пользователя или бота. *Тематику веб-приложения необходимо придумать самостоятельно, обсудив её с преподавателем.* Возможный пример тем: сайт читателя библиотеки, сайт склада некоторых товаров, сайт умного дома, сайт зоопарка, сайт школы и списка учеников и другие. Для разработки веб-приложения воспользуйтесь списком литературы [4-8].

Пример страниц сайта администратора умного дома:



Здравствуйте

Список пользователей

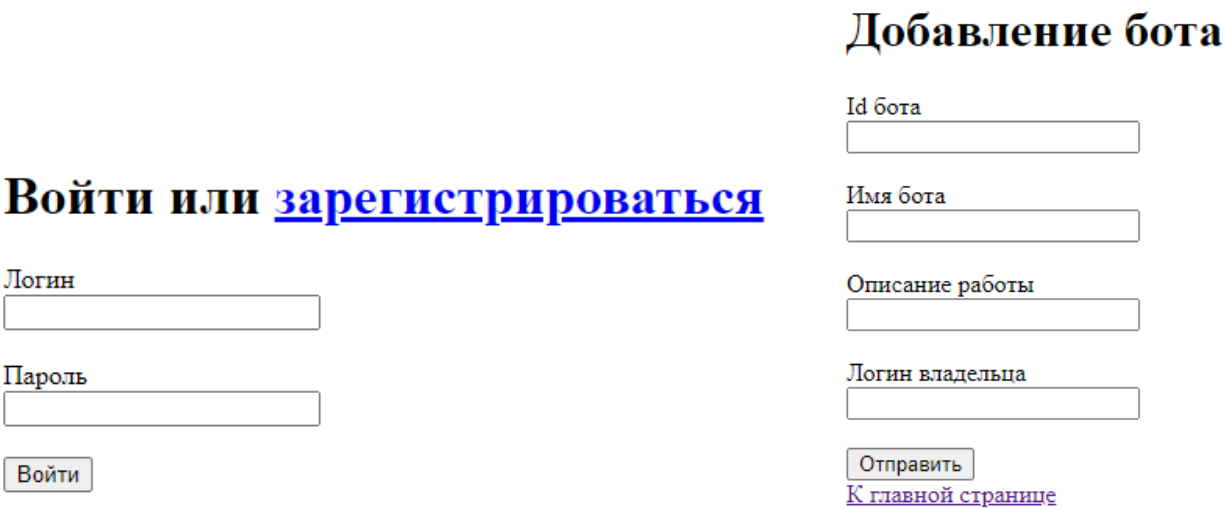
Зарегестрировать пользователя

Логин	Пароль	Фамилия	Имя	Отчество	Дата регистрации	Дата последнего входа	Статус пользователя (админ или нет)	
kutuzova.ia	admin	Кутузова	Ирина	Александровна	Tue Apr 06 2021 19:00:14 GMT+0000 (Coordinated Universal Time)	Tue Apr 06 2021 19:00:14 GMT+0000 (Coordinated Universal Time)	admin	Редактировать Удалить
mr. goose	NotPavlin	Некий	Гусь	-	Tue Apr 06 2021 19:06:53 GMT+0000 (Coordinated Universal Time)	Tue Apr 06 2021 19:06:53 GMT+0000 (Coordinated Universal Time)	user	Редактировать Удалить

Список ботов

Добавить бота

Id бота	Имя	Описание	Логина владельца	Статус	Дата создания	Дата последней конфигурации	
id1	Max	Пылесос	kutuzova.ia	active	Wed Apr 07 2021 15:41:06 GMT+0000 (Coordinated Universal Time)	Wed Apr 07 2021 15:41:06 GMT+0000 (Coordinated Universal Time)	Редактировать Удалить
id2	Кира	Следит за твоим распорядком дня	kutuzova.ia	active	Wed Apr 07 2021 16:49:15 GMT+0000 (Coordinated Universal Time)	Wed Apr 07 2021 16:49:15 GMT+0000 (Coordinated Universal Time)	Редактировать Удалить



Войти или зарегистрироваться

Логин

Пароль

Войти

Добавление бота

Id бота

Имя бота

Описание работы

Логин владельца

Отправить

К главной странице

При отправке запросов к базе данных различные пакеты и языки программирования используют свой интерфейс. Например, реализуют *get/post/update/delete* запросы.

Пример реализации регистрации пользователя:

```
app.post("/registration", urlencodedParser, function (req, res) {
    if (!req.body) return res.sendStatus(400);
    const login = req.body.login;
    const pass = req.body.pass;
    const nameu = req.body.nameu;
    const lastname = req.body.lastname;
    const patronymics = req.body.patronymics;
```

```
    let dateofregistration = new Date().toISOString().slice(0,
    19).replace('T', ' ');
    let datelastconnect = new Date().toISOString().slice(0,
    19).replace('T', ' ');
    const statusu = "user";
    pool.query("INSERT INTO users (login, pass, lastname, nameu,
    patronymics, dateofregistration, datelastconnect, statusu)
    VALUES ($1, $2, $3, $4, $5, $6, $7, $8)", [login, pass, nameu,
    lastname, patronymics, dateofregistration, datelastconnect,
    statusu], function (err, data) {
        if (err) return console.log(err);
        res.redirect("/signin");
    });
});
```

Пример реализации запроса на удаление из базы данных бота с некоторым id:

```
app.post(“/deletebot/:idb”, function (req, res) {
    const idb = req.params.idb;
    pool.query(“DELETE FROM bots WHERE idb=$1”, [idb], function
    (err, data) {
        if (err) return console.log(err);
        res.redirect(«/index»);
    });
});
```

Данные для запросов пользователь может вводить на экране или приложение само может определить id бота, данные по которому нужно получить (например, при нажатии кнопки «удалить» рядом с ботом, система видит id, вписанный в код создания элемента на html).

4. Создание контейнера *Docker*

4.1 *Dockerfile* для создания образа основного приложения (*app*)

В первую очередь создайте *Dockerfile* с необходимыми сведениями. Этот *Dockerfile* должен находиться в той же папке, что и основное приложение.

```
# Базовый образ
# latest – автоматическая установка последней версии node
FROM node:latest
# Рабочая директория
WORKDIR /app
# Копирование файлов в образ
```

```
COPY package*.json ./
# Установка пакетов
RUN npm install
# Открытие порта
EXPOSE host_app_port
# Команды при запуске контейнера
CMD [ «npm», «start» ]
```

Для данного кода:

- команде FROM укажите базовый образ, на основе которого произойдет сборка пользовательского образа;
- команде WORKDIR укажите рабочую директорию, содержимое которой положится в образ;
- команда COPY скопирует файлы в создающийся образ. В случае примера это перечень пакетов, необходимых для работы приложения клиента-сервера, который копируется в корневую папку образа;
- после команды RUN укажите команды, которые должны выполниться при сборке образа (например, установка необходимых пакетов);
- команде EXPOSE укажите на порты в контейнере *Docker*, которые должны быть доступны другим контейнерам или хост-системе;
- команда CMD автоматически выполняется при старте контейнера, например, это может быть запуск приложения или какого-либо скрипта.

4.2 Создание файла *docker-compose.yml*

Файл *docker-compose.yml* является основным файлом для построения контейнера и объединения в нем образов приложения и базы данных. Общий вид файла выглядит следующим образом:

```
# Части приложения – образы/контейнеры, взаимодействующие между собой
services:
     # Образ базы данных
     db:
          // Подробнее в пункте 3.2.1 //
     # Образ приложения клиент-сервер
```

```
    app:
        // Подробнее в пункте 3.2.2 //
volumes:
    db:
```

4.2.1 Services: db

Создание образа базы данных:

```
db:
    # Образ PostrgreSQL
    image: postgres:latest
    # Местоположение файла переменных окружения
    env_file: ./.env
    # Переменные окружения
    environment:
        POSTGRES_USER: $DB_USER
        POSTGRES_PASSWORD: $DB_PASSWORD
        POSTGRES_DB: $DB_NAME
    # Хранилище данных для контейнера
    volumes:
        - db:/var/lib/postgres
        - ./db:/docker-entrypoint-initdb.d/
    # Флаг перезапуска контейнера
    restart: always
```

Данный код создает готовый образ базы данных версии *postgres:latest*, используя переменные окружения для создания базы данных: *POSTGRES_USER*, *POSTGRES_PASSWORD*, *POSTGRES_DB*, взятые из файла *.env*. *Volumes* – необходимые данные для работы базы данных, а также хранения новых записей.

4.2.2 Services: app

Создание образа клиент-серверного приложения:

```
app:
    # Название образа
    image: myapp
    # По какой папке строится образ
    build: ./app
    # Зависимость от образа базы данных
    depends_on:
    - db
```

```
# Местоположение файла переменных окружения
env_file: ./.env
# Переменные окружения
environment:
     DB_HOST: db
     POSTGRES_USER: $DB_USER
     POSTGRES_PASSWORD: $DB_PASSWORD
     POSTGRES_DB: $DB_NAME
# Порты
ports:
     - host_app_port:docker_app_port
# Хранилище данных для контейнера
volumes:
     - ./app:/app
     - /app/node_modules
```

Данный код создает образ клиент-серверного приложения, построенного по папке *app*.

Depends_on определяет порядок запуска и остановки сервисов, то есть, в коде приложение зависит от *bd* и должно создаться после создания образа базы данных. При удалении сначала удаляется *app*, потом *bd*. Если имеется три контейнера (клиент, сервер, база данных), то необходимо указать, что сначала создается база данных, потом сервер, потом клиент. При этом клиент зависит от сервера, а сервер – от базы данных.

host_app_port – это порт, прописанный для связи с приложением в *Dockerfile* (поле EXPOSE). Двоеточие означает переадресацию. Например, *ports '3000:3001'* означает, что пользователь работает с портом *3000*, подключаясь к нему через *localhost* (*localhost:3000/main.html*). Программа перенаправляет запрос на порт докера *3001*, который прописан в файлах сервера.

Пример кода в файле сервера для *docker_app_port = 3001*:

```
app.listen(3001, function () {
     console.log(«Сервер ожидает подключения…»);
});
```

Порты выбираются произвольно. Главное проверить, чтобы они не были заняты другими программами или процессами. Проверить можно с помощью команды `ss -ltnp | grep -w ':port'`. Например, для занятого порта *3000*:

```
[root@localhost ~]# ss -ltnp | grep -w ':3000'
LISTEN 0      4096               [::]:3000          [::]:*    users:(("docker-proxy"
,pid=4190,fd=8))
```

Если порт в данный момент не занят, то в ответ на команду ничего не выведется.

5. Установка *Docker* на виртуальный сервер

Используйте сервер из ранних лабораторных работ или создайте новый. Если при установке пакетов сервер зависает или выдаёт «Killed» - для установки не хватает либо vCPU, либо ОЗУ.

Можно удостовериться, что в данный момент на виртуальном сервере не установлен *Docker*:

```
[root@localhost ~]# docker
-bash: docker: command not found
```

Для установки необходимо выполнить следующие команды. Если при установке не хватает прав, добавьте в начало каждой команды `sudo` перед `dnf`.

```
dnf -y install dnf-plugins-core
```

Установка репозитория:

```
dnf config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo
```

Проверка установки репозитория:

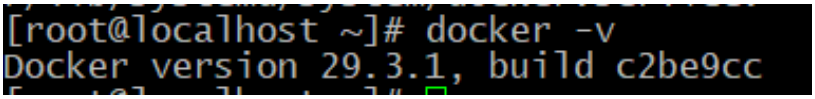
```
dnf list docker-ce
```

```
[root@localhost ~]# dnf list docker-ce
Docker CE Stable - x86_64                  106 kB/s |  72 kB     00:00
Available Packages
docker-ce.x86_64              3:29.3.1-1.el9                   docker-ce-stable
```

```
dnf install docker-ce docker-ce-cli containerd.io docker-buildx-
plugin docker-compose-plugin
usermod -aG docker $(whoami)
newgrp docker
systemctl enable --now docker
```

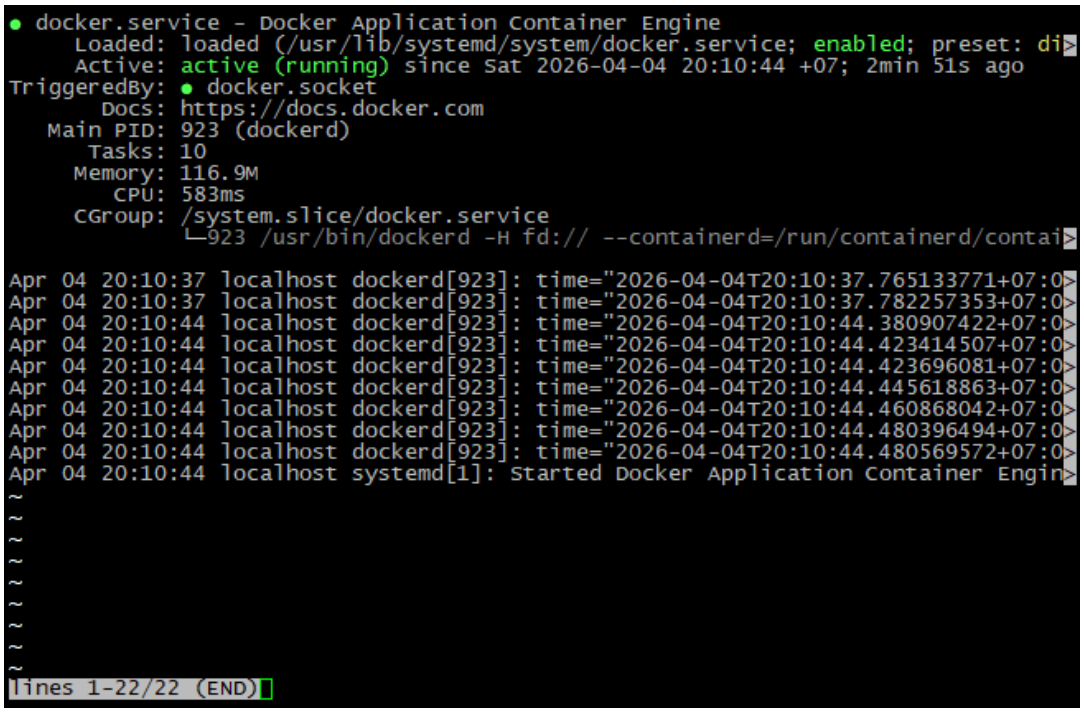
Проверка установки *Docker*:

```
docker -v
```



Проверить корректность запуска службы можно через вывод её текущего состояния:

```
systemctl status docker
```



После установки *Docker* необходимо дополнительно установить *Docker Compose*. Для скачивания актуальной версии *Docker Compose* в каталог */usr/local/bin/* проверьте ее последнюю версию по ссылке: *https://github.com/docker/compose/releases*. В приведенном ниже коде необходимо заменить *version* на нужную версию *Docker Compose*, например *v5.1.1*.

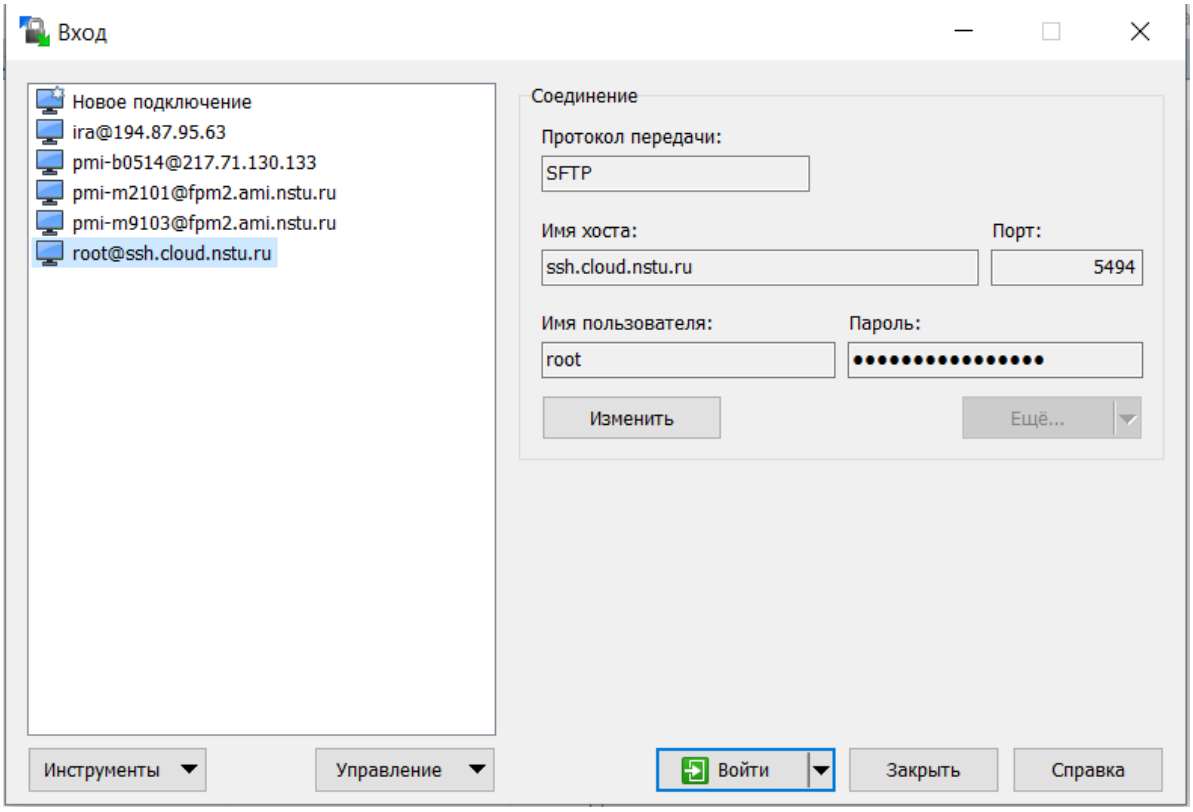
```
curl -L
https://github.com/docker/compose/releases/download/version/docker-
compose-linux-x86_64 -o /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
```

Проверка установки *Docker Compose*:

```
docker compose version
```

```
[root@localhost ~]# docker compose version
Docker Compose version v5.1.1
```

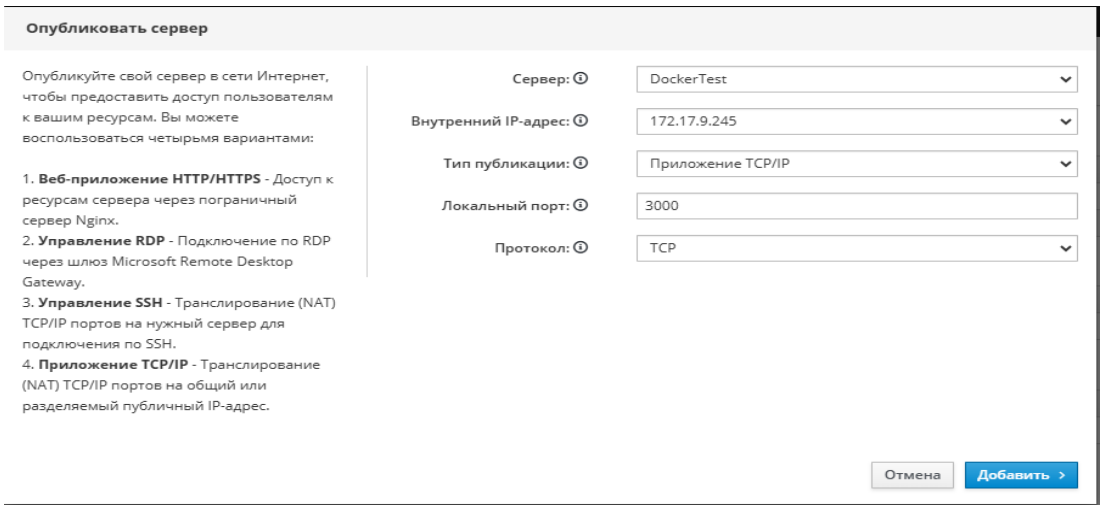
Для переноса данных может пригодиться WinSCP – аналог Putty с графическим интерфейсом. Для подключения нужно выбрать кнопку «Новое соединение». На новой вкладке выберите протокол SFTP и введите остальные данные, которые использовались для подключения в Putty. Соединение можно сохранить.



После подключения справа откроется файловая система сервера, слева – ваша файловая система.

6. Запуск и проверка контейнера *Docker*

Для проверки работы приложения необходимо подключиться к нему по внешнему адресу. Для этого дополнительно опубликуйте сервер в сети по протоколу TCP.

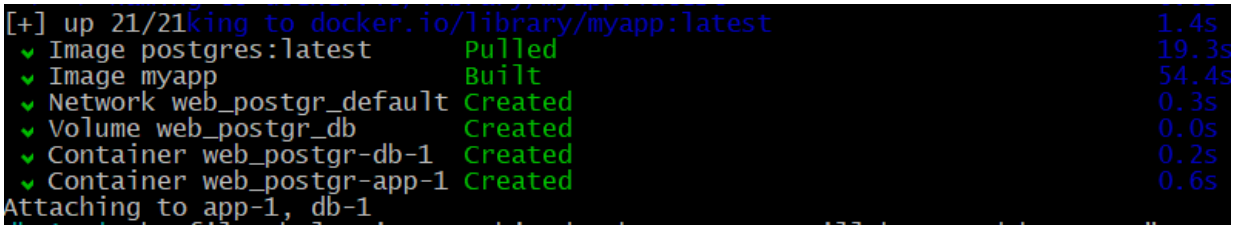


В общем случае публикация происходит на отдельный, выделенный IP-адрес 217.71.129.139 и TCP/IP порт. В качестве локального порта введите порт хоста (*host_app_port*).

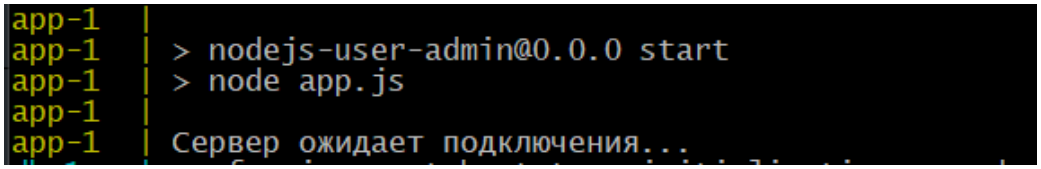
Приложения (TCP, UDP)

Виртуальный сервер	Протокол	Внутренний адрес	Внешний адрес
DockerTest	TCP	172.17.9.245:3000	217.71.129.139:4402

В консоли перейдите в каталог, в котором расположены папки с приложениями и файл *docker-compose.yml*. Сборка и запуск приложения осуществляется с помощью команды `docker compose up --build`. Все контейнеры должны быть созданы:



Проверьте готовность сервера принимать входящие запросы пользователя:



Проверьте создание базы данных, таблиц и вставку данных в таблицы:

```
db-1   | server started
db-1   | CREATE DATABASE
db-1   |
db-1   |
db-1   | /usr/local/bin/docker-entrypoint.sh: running /docker-entrypoint-initdb.
d/postgr.sql
db-1   | CREATE TABLE
db-1   | INSERT 0 2
db-1   | CREATE TABLE
db-1   | INSERT 0 2
```

Проверьте готовность базы данных к подключению:

```
db-1   | 2026-04-04 13:42:49.710 UTC [1] LOG:  database system is ready to accep
t connections
```

Останов работы контейнеров и образов возможен с помощью команды *CTRL+C* в командной строке. Удаление созданных контейнеров и образов осуществляется с помощью команды `docker compose down -v`:

```
[+] down 4/4
 ✔ Container web_postgr-app-1 Removed
 ✔ Container web_postgr-db-1  Removed
 ✔ Volume web_postgr_db       Removed
 ✔ Network web_postgr_default Removed
```

Для проверки работы запустите контейнер и проверьте работоспособность ссылки *http://внешний_адрес:port/page*, где *port* – порт, прописанный облачной системой, *page* – некоторая страница, которая ранее была прописана в файле клиент-серверного приложения. Для примера выше это адрес *http://217.71.129.139:4402/index*. Если все сделано верно, то страница загрузится, и данные отобразятся.

Далее необходимо проверить работоспособность других ссылок, таких как «авторизация», «регистрация» и других, а также возможность добавления/редактирования/удаления объектов в базе данных через веб-приложение. Полученные результаты отобразите в отчете.

Содержимое базы данных внутри контейнера в консоли можно посмотреть с помощью следующей последовательности команд:

1) Для просмотра базы данных нужно узнать ID образа базы данных с помощью команды `docker ps`. В примере ниже ID_контейнера = 1476b5e6156f:

```
[root@localhost web_postgr]# docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED         STATUS
       PORTS                                                 NAMES
af29f668bfcc   myapp             "docker-entrypoint.s…"   3 minutes ago   Up About a mi
nute   3000/tcp, 0.0.0.0:3000->3001/tcp, [::]:3000->3001/tcp   web_postgr-app-1
1476b5e6156f   postgres:latest   "docker-entrypoint.s…"   3 minutes ago   Up About a mi
nute   5432/tcp                                              web_postgr-db-1
```

2) С помощью команды `docker exec -it ID_контейнера bash` можно зайти внутрь контейнера. Если команда не сработала, то необходимо добавить `sudo` в начале строки:

```
[root@localhost web_postgr]# docker exec -it 1476b5e6156f bash
root@1476b5e6156f:/#
```

3) Команда `psql -U <Имя_пользователя> <Имя_БД>` позволяет зайти в базу данных под нужным пользователем (прописанным при создании контейнера базы данных):

```
root@1476b5e6156f:/# psql -U admin useradmin
psql (18.3 (Debian 18.3-1.pgdg13+1))
Type "help" for help.

useradmin=#
```

Далее выполните различные запросы к таблицам. Основной запрос `SELECT * FROM pg_catalog.pg_tables;` - просмотр существующих таблиц:

```
useradmin=# SELECT * FROM pg_catalog.pg_tables;
    schemaname     |        tablename        | tableowner | tablespace | hasindexes |
ity
-------------------+-------------------------+------------+------------+------------+
----
 public            | users                   | admin      |            | t          |
 public            | bots                    | admin      |            | t          |
 pg_catalog        | pg_statistic            | admin      |            | t          |
```

Также проверьте другие запросы, которые база данных должна выполнять в созданном приложении. Отразите в отчете работу с консолью. Выполните к базе данных 2-3 запроса и сравните результаты в консоли с результатами, выдаваемыми веб-приложением.

Контрольные вопросы:

1. Что такое виртуализация? В чем различие между виртуализацией и контейнеризацией.

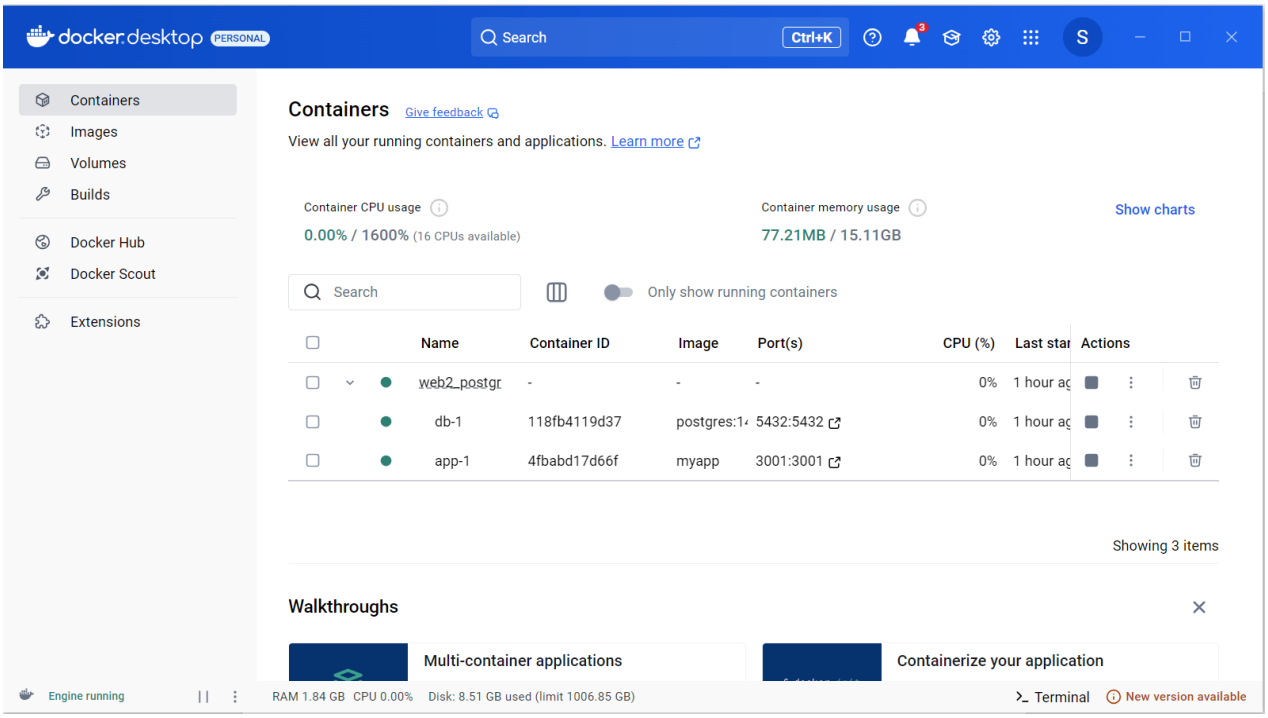
2. Что такое контейнер *Docker*? Расскажите о жизненном цикле контейнера *Docker*.
3. Команды *Docker*. Как определить состояние контейнера *Docker*?
4. Что такое образ *Docker*? Что такое файл *Dockerfile*?
5. Каким образом *Docker* сохраняет данные? Если вы остановите контейнер, потеряете ли данные?
6. Что такое *Docker Compose* и когда он применяется? В чем различие между *Docker* и *Docker Compose*, между *Docker Swarm* и *Docker Compose*?
7. Команды *Docker Compose*. Как определить состояние *Docker Compose*?
8. Что такое файл *docker-compose.yml*? Для чего он необходим? Что в нём находится?
9. Присоединение к базе данных из приложения. Запрос на подключение. Переменные окружения.

Приложение 6

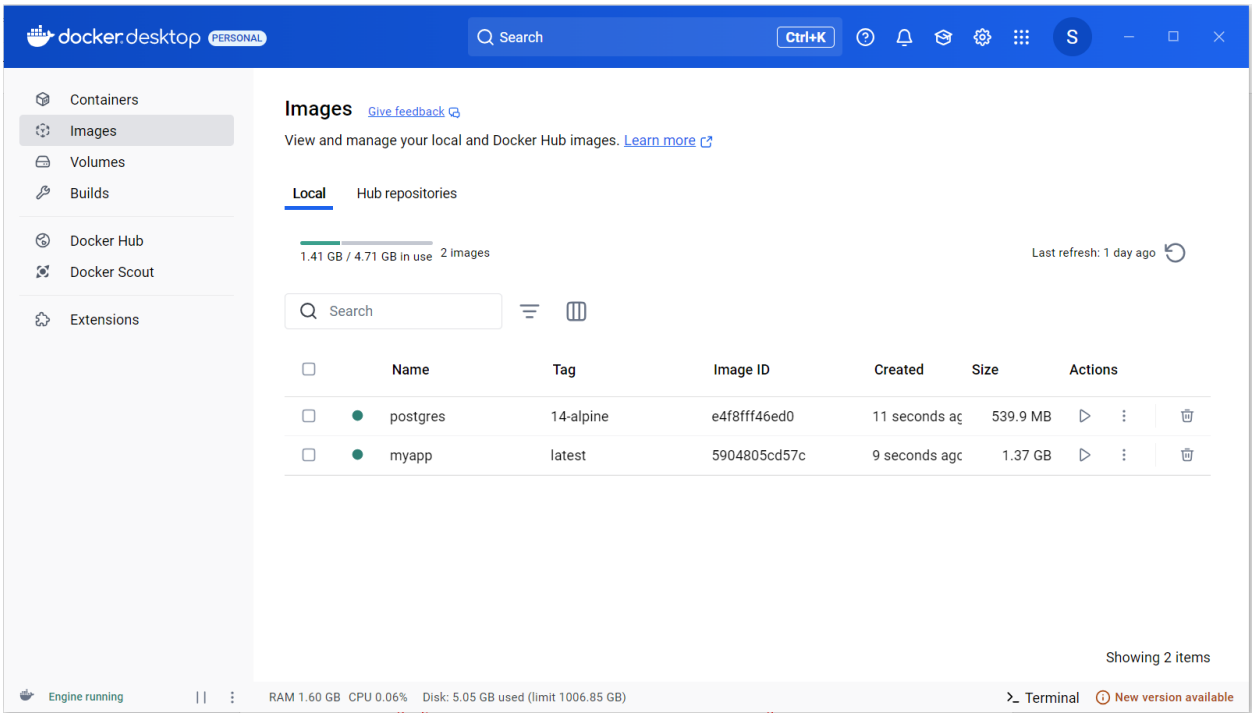
Docker Desktop для работы на Windows и MacOS

1. Перейти по ссылке *https://www.docker.com/products/docker-desktop/#/windows* и скачать *Docker Desktop* для выбранной ОС. *Docker* для Mac и *Docker* для Windows – это отдельные продукты, которые запускают виртуальную машину Linux внутри операционной системы хоста и создают контейнеры внутри этой машины.
2. Установить приложение *Docker Desktop*.

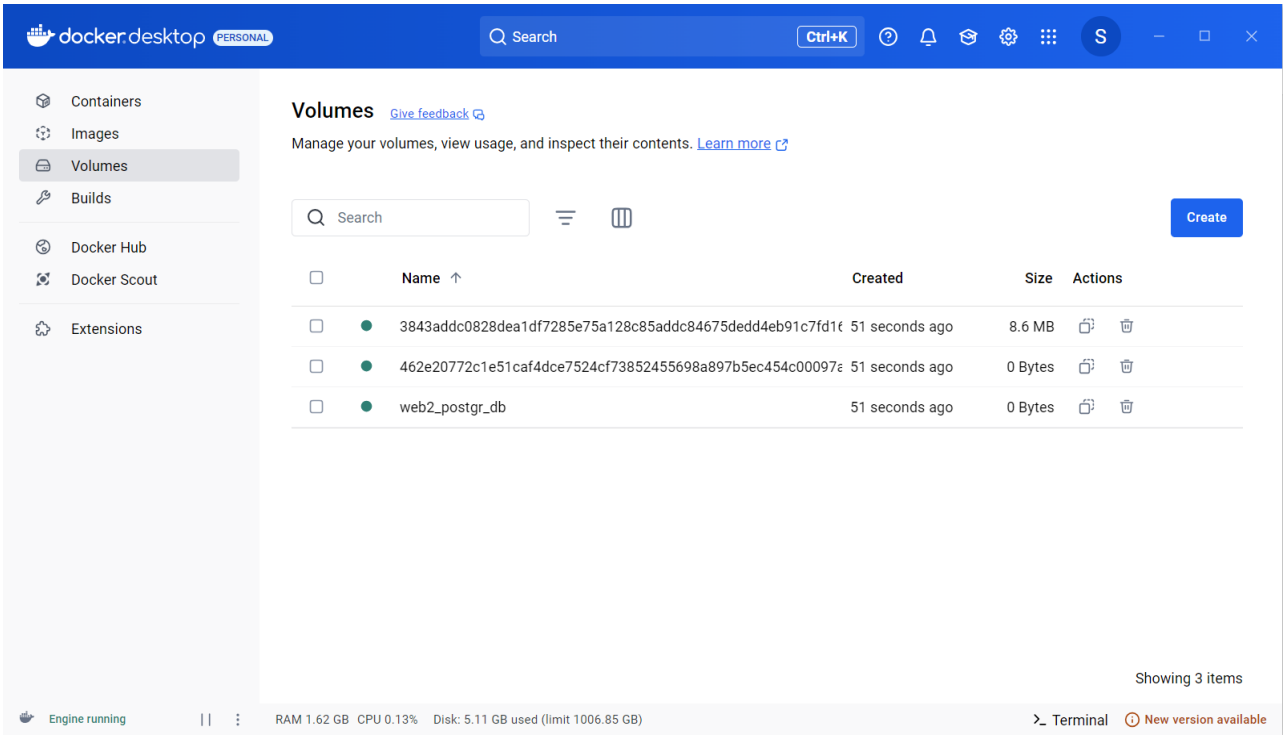
Containers - список существующих контейнеров. Обратите внимание, что на данном примере представлено два разных контейнера db-1 и app-1, объединенных в одно приложение web2-postrgr.



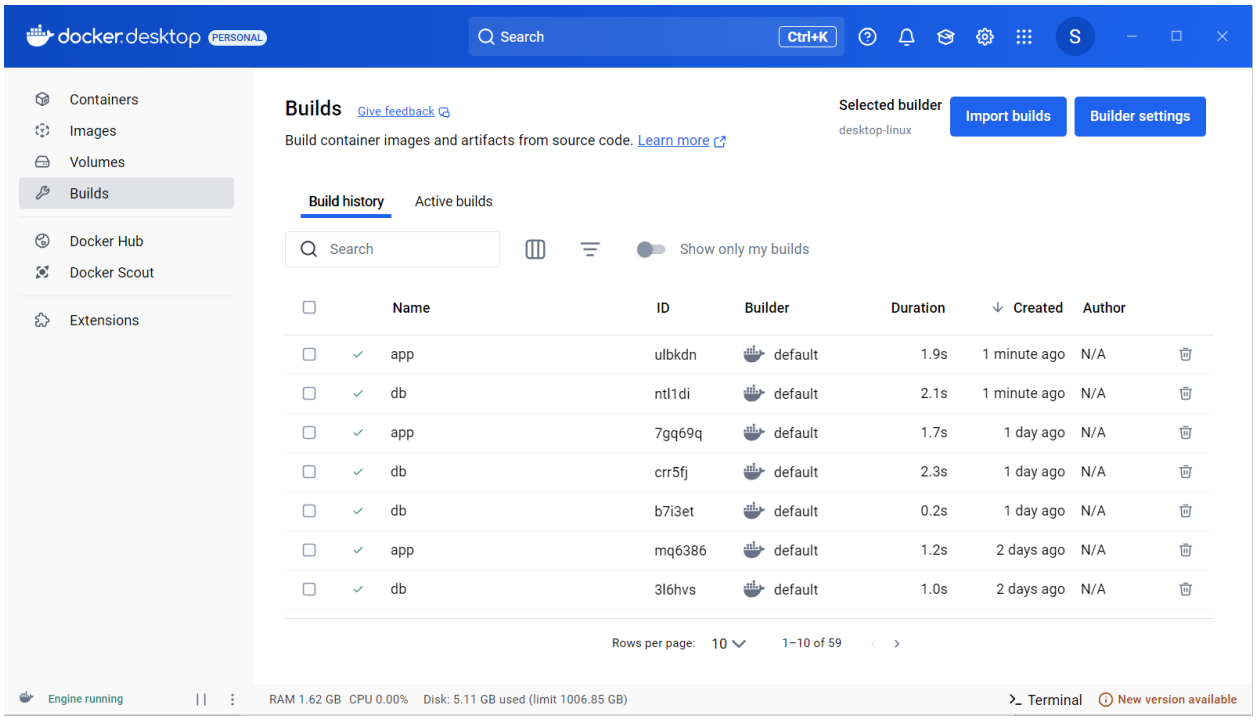
Images – список существующих образов. Во вкладке Hub repositories можно посмотреть образы из пользовательского репозитория образов.



Volumes – хранилище данных для контейнеров. Не имеет привязки к жизненному циклу контейнера. Необходимо для сохранения информации внутри контейнера после его перезапуска (например, добавленные после создания контейнера данные в базе данных).



Builds – история созданий контейнеров и образов.



БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Стасышин В.М., Стасышина Т.Л. Язык структурных запросов SQL. Начальное знакомство. Новосибирск: Изд-во НГТУ, 2023.
2. Стасышин В.М. Проектирование информационных систем и баз данных. Учебное пособие. Новосибирск: Изд-во НГТУ, 2012.
3. Документация к PostgreSQL 18 [сайт]. – 2025 – URL: https://postgrespro.ru/docs/postgresql/current (дата обращения: 04.04.2026).
4. Официальный образ Docker базы данных PostgreSQL [сайт]. – URL: https://hub.docker.com/_/postgres (дата обращения: 04.04.2026).
5. Репозиторий официальных образов Docker [сайт]. – URL: https://hub.docker.com/search?badges=official (дата обращения: 04.04.2026).
6. Документация пакета pg для работы с базой данных PostgreSQL на Node.js [сайт]. – URL: https://node-postgres.com/announcements (дата обращения: 04.04.2026).
7. Официальный сайт языка создания шаблонов Handlebars [сайт]. – URL: https://handlebarsjs.com/guide/ (дата обращения: 04.04.2026).
8. Практическое руководство по Node.js, создание web-приложения с помощью Handlebars, работа с базой данных [сайт]. – URL: https://metanit.com/web/nodejs/ (дата обращения: 04.04.2026).
9. Git: *https://git-scm.com/doc*
10. GitLab документация: *https://docs.gitlab.com*
11. Markdown: *https://www.markdownguide.org*
12. Mermaid диаграммы: *https://mermaid.js.org*