|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №3**

по дисциплине «Технологии виртуализации клиент-серверных приложений»

**Тема практической работы:**

**Студент группы** ИКБО-20-21 Мухаметшин А. Р.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

**Руководитель практической работы**  преподаватель Волков М.Ю.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Работа представлена «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Допущен к работе «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Москва 2024

**ОГЛАВЛЕНИЕ**

[Цель работы 3](#_Toc178103096)

[Ход работы 4](#_Toc178103097)

[Вывод 10](#_Toc178103098)

[Ответы на вопросы к практической работе 11](#_Toc178103099)

[Список использованной литературы 13](#_Toc178103100)

Цель работы

Целью данной работы является получение практических навыков по работе с программной платформой Docker, созданию образов и контейнеров.

Ход работы

Создадим веб-приложение, для которого позже будет создан Dockerfile. Это будет Python приложение, использующее фреймворки FastAPI и SQLAlchemy и пакетный менеджер Poetry. Это приложение реализует следующие ручки (Рисунок 1).

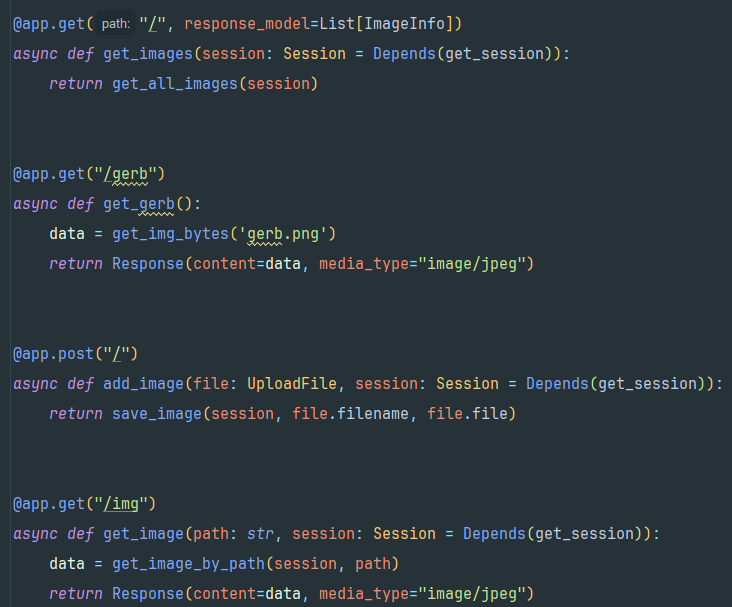


Рисунок 1 – Список реализованных ручек

Начнем писать Dockerfile. Начнем с базового образа, который потом позже будет использоваться при сборке. Он представлен на Рисунке 2.

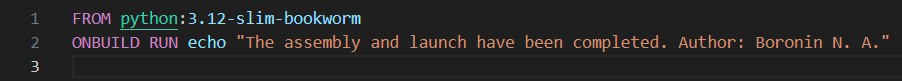


Рисунок 2 – Dockerfile базового образа

Этот образ будет использоваться на первом шаге многоэтапной сборки, как показано на Рисунке 3.

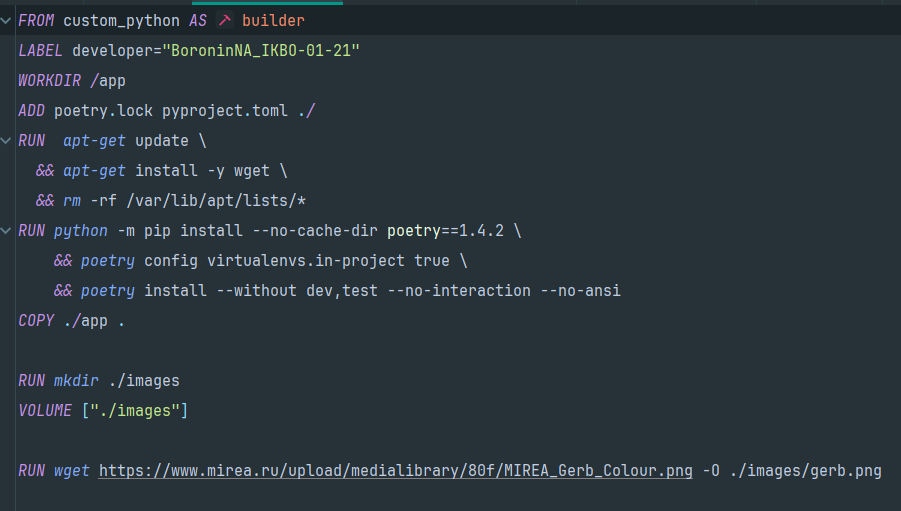


Рисунок 3 – Первый шаг сборки

Здесь устанавливается пакетный менеджер Poetry и с его помощью устанавливаются необходимые зависимости. Кроме того, скачивается утилита wget для получения файла MIREA\_Gerb\_Colour.png.

Далее перейдем ко второму шагу сборки, который будет использовать виртуальное окружение, созданное на первом шаге. Это показано на Рисунке 4.

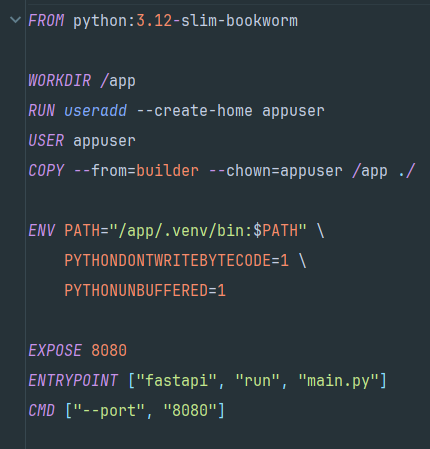


Рисунок 4 – Второй этап сборки

На данном шаге создается пользователь, от имени которого будут исполняться все операции в контейнере, копируется созданное окружение и пробрасываются необходимые переменные окружения. Основная точка входа – команда ENTRYPOINT, она представляет выражение, которое не должно изменяться. Далее следует команда CMD, предоставляющая аргументы по умолчанию. Аргументы, переданные при запуске контейнера, могут перезаписать CMD, но не влияют на ENTRYPOINT.

Также были необходимы переменные окружения для подключения к СУБД PostgreSQL. Они были описаны в .env файле, и использованы в docker-compose, как показано на Рисунке 5.

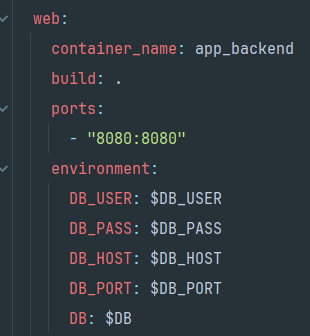


Рисунок 5 – Переменные окружения

Эти переменные используются в веб-приложении следующим образом (Рисунок 6).

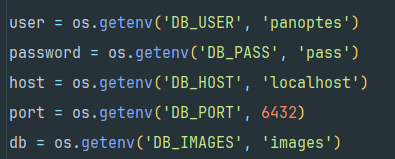


Рисунок 6 – Использование переменных окружения

Перейдем к тестированию приложения. На Рисунке 7 показан результат тестирования ручки получения герба РТУ МИРЭА.

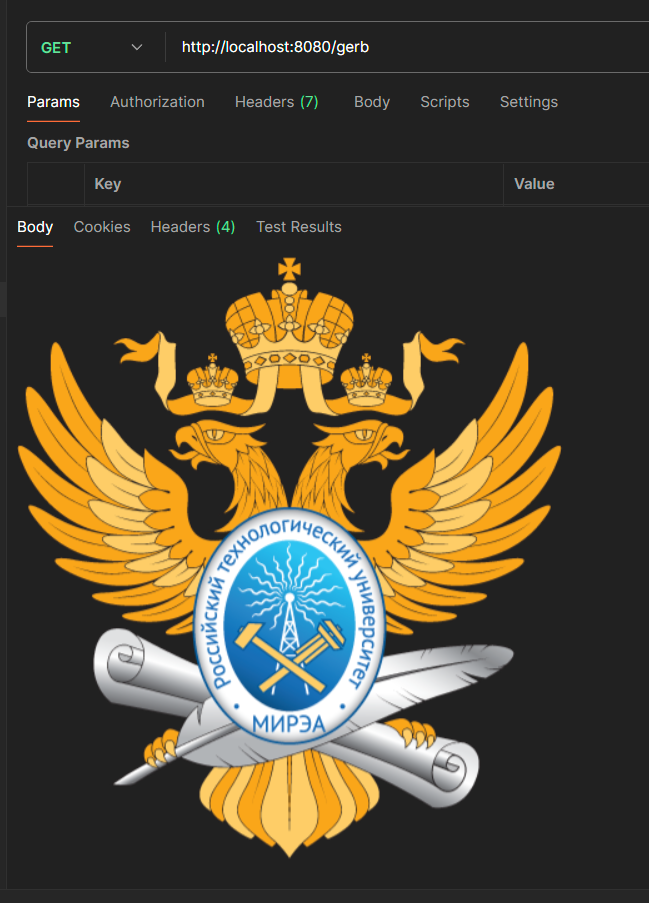


Рисунок 7 – Результат запроса на получения герба

На Рисунке 8 продемонстрирован результат тестирования добавления элемента.

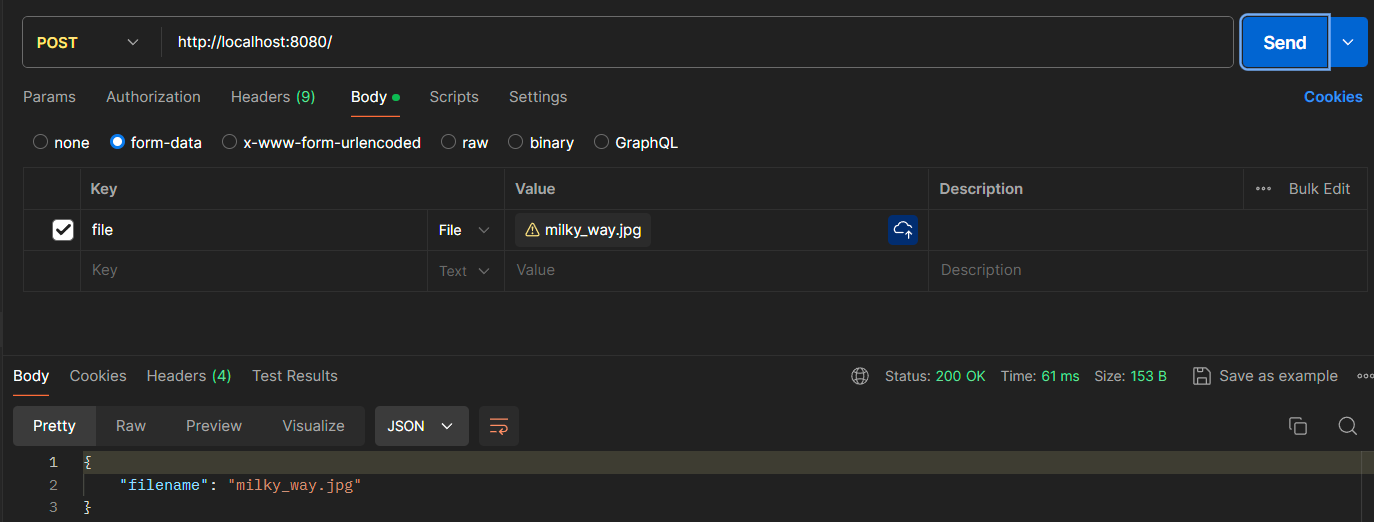


Рисунок 8 – Результат тестирования добавления элемента

На Рисунке 9 представлен результат тестирования вывода списка всех элементов.

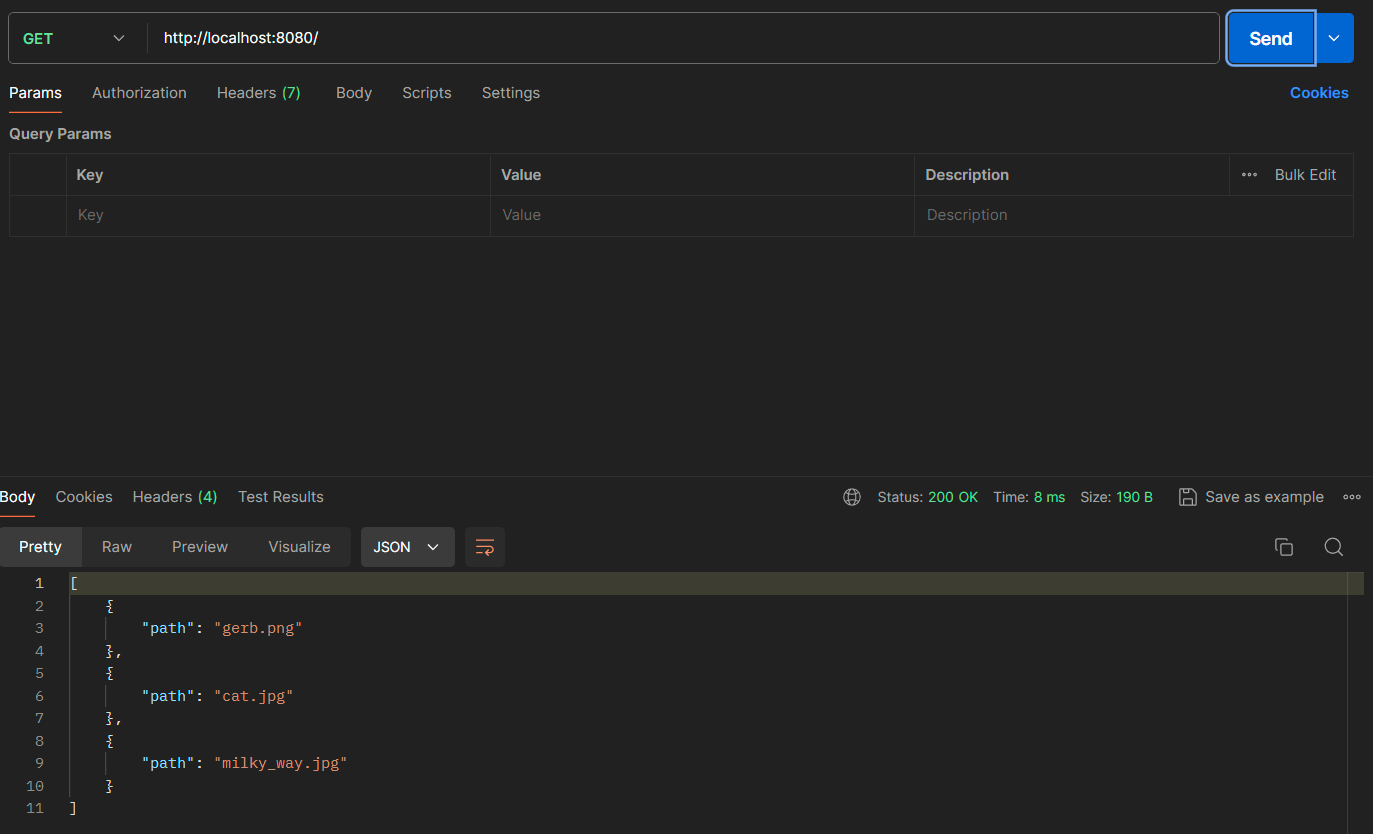


Рисунок 9 – Результат запроса на получение всех элементов

На Рисунке 10 показан результат запроса конкретного элемента.

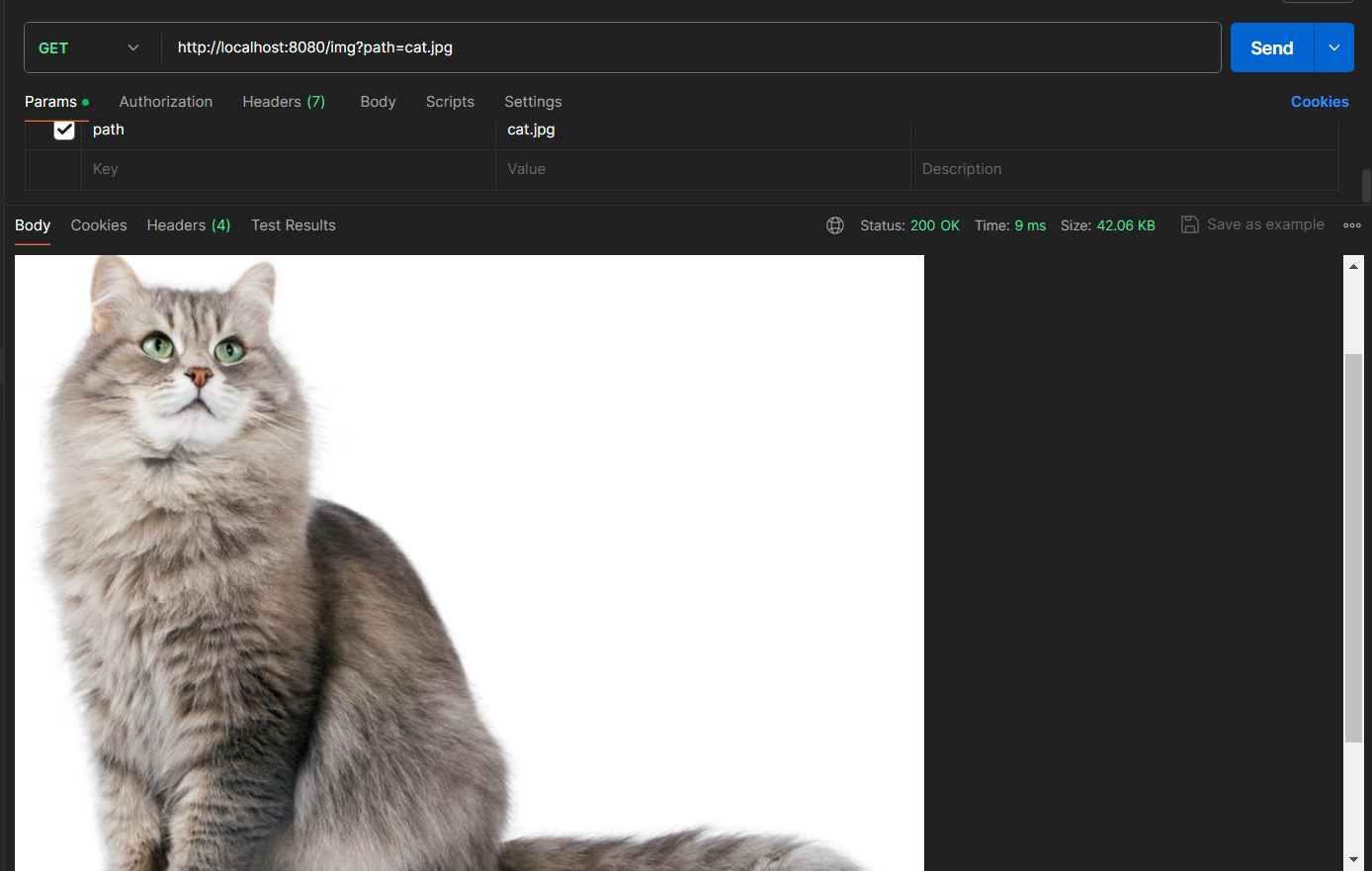


Рисунок 10 – Тестирование запроса на получение конкретного элемента

Продемонстрируем возможность запуска контейнеров (Рисунок 11) и логи сервера внутри контейнера (Рисунок 12).

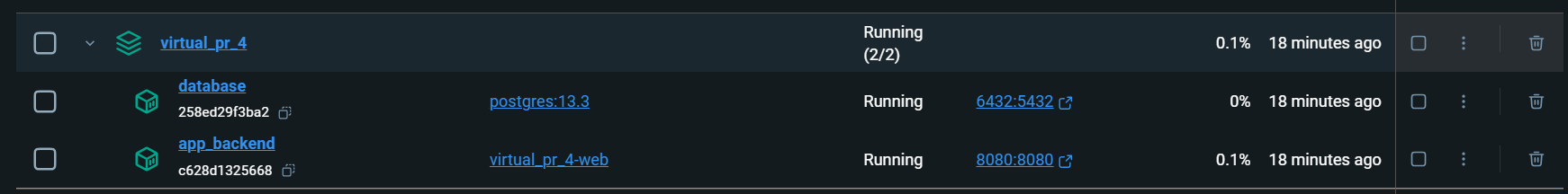


Рисунок 12 – Запущенные контейнеры

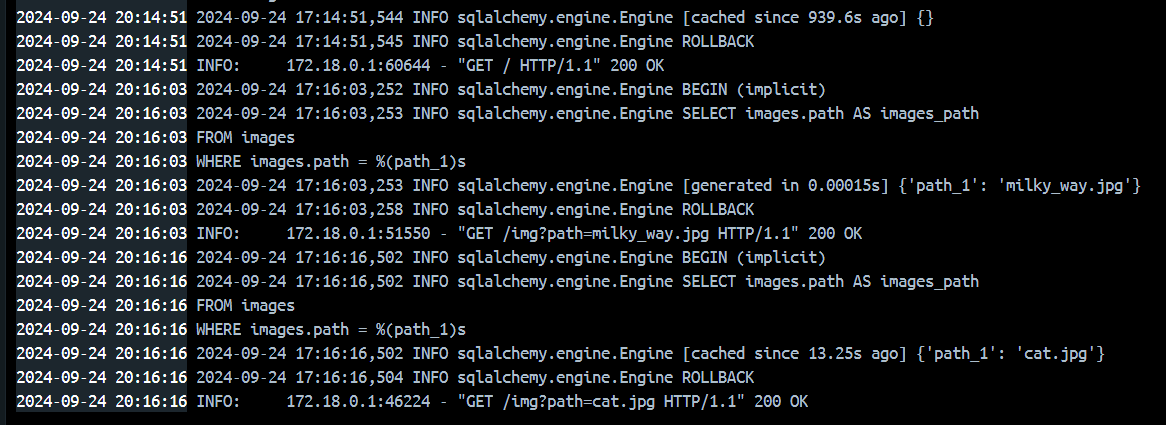


Рисунок 13 – Логи контейнера с приложением

Таким образом разработанное веб-приложение было собрано в контейнер, запущено при помощи Docker и протестировано на работоспособность.

Вывод

В ходе выполнения данной практической работы были получены практические навыки по работе с программной платформой Docker, созданию образов и контейнеров.

Ответы на вопросы к практической работе

**1. Опишите процесс запуска приложения внутри контейнера Linux, используя Docker.**

Важно отметить, что Docker использует технологии, такие как namespaces и cgroups в Linux, для изоляции контейнеров. Namespaces создает отдельные пространства имен для процессов, пользователей, сетевых интерфейсов и других ресурсов. Cgroups контролируют использование ресурсов (ЦП, память, дисковое пространство) контейнерами.

Сначала командой docker build создает новый образ из Dockerfile. Далее Docker Daemon получает этот запрос и проверяет, существует ли указанный образ на локальной машине (если нет, он загрузит его из Docker Hub или другого реестра).

Наконец Docker создает новый процесс для контейнера, используя команды, заданные в CMD или ENTRYPOINT в Dockerfile

**2. Что такое образ Docker и для чего он нужен?**

Образ Docker (Docker Image) — это статический файл, который содержит все необходимые компоненты для запуска приложения внутри контейнера. Образ включает в себя:

- базовую операционную систему (например, Ubuntu, Alpine и др.);

- все зависимости и библиотеки, которые требуются для работы приложения;

- само приложение или его код;

- конфигурационные файлы;

- команды для его настройки и запуска.

**3. Как соотносятся между собой файлы Dockerfile и Docker-Compose?**

Цель Dockerfile – описать как создать образ Docker. Задача docker-compose – управлять множеством конейтеров docer как одним приложением.

**4. Что такое Dockerfile?**

Dockerfile — это текстовый файл, который содержит все команды и инструкции, необходимые для автоматической сборки образа Docker. Он описывает процесс создания образа, включая исходный базовый образ, установки программного обеспечения, копирование файлов и выполнение конфигурационных шагов.

**5. Опишите политики перезапуска контейнера.**

Основные политики перезапуска:

- no: контейнер не будет автоматически перезапущен. Это поведение по умолчанию, если не указана другая политика.

- always: контейнер будет автоматически перезапущен всегда, если он завершил свою работу (независимо от причины: ошибка, остановка и т. д.).

- unless-stopped: подобно always, но без автоматического перезапуска, если контейнер был остановлен вручную.

- on-failure: контейнер будет автоматически перезапущен только в случае его завершения с ненулевым статусом (т.е., если произошла ошибка).

**6. Назовите все возможные состояния контейнеров.**

Основные состояния контейнеров:

- сreated: контейнер был создан, но еще не запущен. Это начальное состояние после выполнения команды docker create;

- running: контейнер в данный момент выполняется и активен. Он находится в этом состоянии после выполнения команды docker run или docker start;

- paused: контейнер был приостановлен, и его выполнение временно остановлено. Это состояние можно получить с помощью команды docker pause. Приостановленный контейнер можно снова запустить с помощью команды docker unpause;

- exited: контейнер завершил свою работу. Он может остановиться по разным причинам, например, завершение процесса, выполняемого в контейнере, либо ошибка во время выполнения;

- dead: это состояние возникает, если контейнер соответствует выполнению процесса, но Docker не может его правильно контролировать. Это может быть вызвано различными проблемами, такими как сбой в хост-системе или проблемы с сетевыми настройками. Подобное состояние может наблюдаться в случае, если дочерний процесс контейнера завершается некорректно.

Список использованной литературы

1. 50 вопросов по Docker, которые задают на собеседованиях, и ответы на них | Хабр. — Текст: электронный [сайт]. — URL: https://habr.com/ru/company/southbridge/blog/528206/;

2. Docker Documentation | Docker Documentation — Текст: электронный [сайт]. — URL: <https://docs.docker.com/>;

3 Что такое режим Docker Swarm и когда его использовать? — CloudSavvy ИТ | Cpab. — Текст: электронный [сайт]. — URL: https://cpab.ru/chtotakoe-rezhim-docker-swarm-i-kogda-ego-ispo4lzovat-cloudsavvy-it/;

4 Dockerfile reference | Docker Documentation — Текст: электронный [сайт]. — URL: <https://docs.docker.com/engine/reference/builder/>;