|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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: Docker**

**Студент группы** ИКБО-24-20 Рослов Павел

Дмитриевич

Постников Иван Романович

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

(подпись студента)

**Руководитель практической работы** Мельников Денис Александрович

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

(подпись руководителя)

Работа представлена «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Допущен к работе «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Москва 2023

**Задание 1.** Создать один или несколько Dockerfile, в которых каждая из нижеприведенных команд будет использована хотя бы 1 раз: FROM; RUN; LABEL; CMD; EXPOSE; ENV; ADD; COPY; ENTRYPOINT; VOLUME; USER; WORKDIR; ONBUILD.

Листинг 1 – файл Dockerfile с указанными командами

*# Используем официальный образ Node.js как базовый образ*

FROM node:14

*# Устанавливаем метаданные для образа*

LABEL maintainer="your-email@example.com"

LABEL version="1.0"

LABEL description="Dockerfile с использованием различных команд"

*# Задаем переменную окружения*

ENV NODE\_ENV=production

*# Создаем рабочую директорию*

WORKDIR /app

*# Копируем зависимости и файлы приложения в рабочую директорию*

COPY package\*.json ./

RUN npm install

COPY . .

*# Открываем порт, на котором будет работать приложение*

EXPOSE 3000

*# Задаем команду, которая будет выполняться при запуске контейнера*

CMD ["node", "app.js"]

*# Указываем точку входа (entrypoint) для контейнера*

ENTRYPOINT ["node"]

*# Создаем точку монтирования для хранения данных*

VOLUME /data

*# Задаем пользователя и группу, от имени которых будет выполняться приложение*

USER node

*# Действие, которое будет выполняться при сборке этого образа в качестве базового образа для другого Dockerfile*

ONBUILD ADD . /app/onbuild

ONBUILD RUN /usr/local/bin/python-build --dir /app/onbuild

**Задание 2.**

**Вывод**

В ходе данной практической работы были разобраны команды, прописанные в Dockerfile. Также было написано веб-приложение, собранное при помощи Docker и загруженное в DockerHub.

**КОНТРОЛЬНЫЕ ВОПРОСЫ:**

1. Процесс запуска приложения внутри контейнера Linux с использованием Docker:

* Создание Docker-образа: Сначала нужно создать Docker-образ, который содержит необходимое приложение и его зависимости. Для этого используется Dockerfile (см. вопрос 4).
* Сборка Docker-образа: Запускается команда docker build, которая выполняет инструкции из Dockerfile для создания образа.
* Запуск контейнера: После создания Docker-образа можно запустить контейнер с помощью команды docker run. Это создаст экземпляр контейнера из образа.
* Настройка сети и портов: Если приложение внутри контейнера использует сетевые ресурсы, то необходимо настроить сеть и маппинг портов для доступа к контейнеру с хоста.

1. Docker-образ - это легковесная, автономная и исполняемая среда, которая содержит все необходимое для запуска приложения, включая код, зависимости, среду выполнения и настройки. Он представляет собой статичный снимок файловой системы с добавленным приложением. Docker-образы служат для упрощения развертывания и масштабирования приложений в контейнерах Docker.
2. Файлы Dockerfile и Docker-Compose используются в Docker для описания и настройки контейнеров, но они выполняют разные функции:

* Dockerfile: Этот файл используется для создания Docker-образа. Он содержит инструкции по тому, как собрать образ, включая базовый образ, установку зависимостей, копирование файлов и другие настройки.
* Docker-Compose: Docker-Compose используется для определения и настройки множества контейнеров и их взаимосвязей в многоконтейнерных приложениях. В Docker-Compose файле определяются службы, их зависимости, параметры сети и другие параметры, которые необходимы для развертывания и управления множеством контейнеров.

1. Dockerfile - это текстовый файл, который содержит инструкции для автоматической сборки Docker-образа. В Dockerfile описываются шаги по установке зависимостей, копированию файлов, настройке среды выполнения и другие действия, необходимые для создания полностью настроенного образа, который можно использовать для запуска контейнера.
2. Политики перезапуска контейнера определяют, как Docker будет обрабатывать ситуации, когда контейнер завершает свою работу или неожиданно падает. Наиболее распространенными политиками перезапуска являются:

* "no" (по умолчанию): Контейнер не будет перезапускаться автоматически.
* "always": Контейнер всегда будет перезапускаться при завершении, даже если он завершился успешно.
* "unless-stopped": Контейнер будет перезапускаться только в том случае, если его остановили явно или если Docker перезапускается.
* Числовые значения: Можно указать конкретное количество попыток перезапуска контейнера перед окончательным прекращением.

1. В Docker существует несколько состояний контейнеров:

* Created: Контейнер создан, но еще не запущен.
* Running: Контейнер запущен и выполняет свои задачи.
* Paused: Выполнение контейнера приостановлено. Контейнер остается в паузе до явного возобновления.
* Stopped: Контейнер остановлен и не выполняет задачи. Может быть перезапущен.
* Exited: Контейнер завершил свое выполнение. Это конечное состояние для контейнера, если он не настроен на автоматический перезапуск.
* Dead: Контейнер больше не существует или не может быть запущен, например, из-за конфликтующих портов или недоступных ресурсов.