**ЗНАКОМСТВО С ИНСТРУМЕНТАМИ DOCKER COMPOSE:**

**СОЗДАНИЕ ПРОСТОГО КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ**

**Можаев И.С.**

**Введение.**

В данной лабораторной работе будут изучены и продемонстрированы базовые инструменты для работы с Docker Compose, освоены методы связывания контейнеров, а также написано простое клиент-серверное приложение на Python с использованием Flask.

**Основная часть.**

**Теоретическая база**

Docker как программное обеспечение для контейнеризации позволяет пользователю запускать различные процессы и приложения изолированно от хостовой операционной системы, расходуя при этом небольшое количество ресурсов и времени. С помощью Docker можно создавать пространство для тестирования приложений, запуска различных служб, скриптов, настройки их соединений и т.п. Однако при развёртывании более крупных проектов, для реализации которых необходимо создание нескольких сервисов в отдельных контейнерах, становится очевидным, что собирать, запускать и отлаживать по очереди каждый контейнер, а также настраивать их сетевое взаимодействие громоздко и неудобно.

Именно для этих целей был создан Docker Compose – инструмент-менеджер для управления многоконтейнерными проектами. Docker Compose применяется при разработке систем, использующих различные сервисы, взаимосвязанные друг с другом, настройке внутренних зависимостей, а также для совместного использования таких приложений. Простейшим примером применения Docker Compose может служить сервер, подключённый к базе данных, или сервер, предоставляющий некоторые данные сервису-клиенту.

При разработке клиент-серверной архитектуры был использован язык программирования Python и библиотека Flask, предоставляющие простой и удобный синтаксис для создания веб-серверов.

**Цели и задачи**

- Научиться связывать контейнеры Docker

- Создать веб-сервер и клиент к нему

- Освоить инструменты работы с контейнерами и сетями в Docker Compose

- Настроить три контейнера (сервер, промежуточный сервер и клиент), два из которых изолированы друг от друга

**Технические требования**

- Операционная система семейства Linux (желательно Ubuntu не ниже 22.04) или виртуальная машина с такой системой

- Установленные Docker и Docker Compose

- Установленный Python и утилита pip, а также утилита ping

**Выполнение лабораторной работы**

**1. Создание двух контейнеров: сервер и клиент**

Создайте в рабочей директории нового проекта два каталога: server/ и client/. В каждом из них создайте Dockerfile и python-скрипт, например server.py и client.py.

Добавьте в каталог server/ файл index.html с некоторым текстом, который ваш сервер должен выводить. В файл server.py запишите следующий код, предоставляющий базовый веб-сервер на порту 1200 контейнера (по умолчанию использует файл index.html):

*#!/usr/bin/env python3*

*import http.server*

*with http.server.HTTPServer(("", 1200), http.server.SimpleHTTPRequestHandler) as*

*serv:*

*serv.serve\_forever()*

В файле Dockerfile сервера должен быть следующий текст:

*# сборка из последней версии python-образа*

*FROM python:latest*

*# копирование необходимых файлов в домашнюю директорию контейнера*

*COPY server.py .*

*COPY index.html .*

*# запуск скрипта сервера*

*CMD python server.py*

В файле client.py выполните запрос и вывод данных от сервера с помощью библиотеки requests:

*#!/usr/bin/env python3*

*import requests*

*response = requests.get("http://server:1200/")*

*print(response.text)*

В Dockerfile для клиента запишите следующие команды:

*# сборка из последней версии python-образа*

*FROM python:latest*

*# установка библиотеки requests, не входящей в стандартный пакет python*

*RUN pip install requests*

*# копирование исполняемого файла*

*COPY client.py .*

*# запуск скрипта*

*CMD python client.py*

**2. Связывание контейнеров**

Соберите образы из созданных Dockerfile с удобными названиями, например:

*docker build -t flask-server server/*

*docker build -t flask-client client/*

Чтобы соединить контейнеры между собой, можно создать виртуальную сеть, например my\_network:

*docker network create my\_network*

Запустите контейнер-сервер в фоновом режиме, подключив его к созданной сети и выполнив проброс портов хоста:

*docker run -d -p 1200:1200 --network=my\_network --name server flask-server*

Теперь запустите контейнер-клиент, также подключив его к сети my\_network и перенаправив потоки ввода-вывода в терминал:

*docker run -it --network=my\_network --name client flask-client*

В терминал должен вывестись текст, который вы поместили в файл index.html. Чтобы проверить работу веб-сервера, вы можете перейти по адресу http://localhost:1200/ в браузере и увидеть тот же самый текст.

Кроме того, можно связать два контейнера без создания пользовательской виртуальной сети, дав ссылку контейнеру-клиенту на контейнер-сервер:

*docker run -d -p 1200:1200 --name server flask-server*

*docker run -it --link server:server --name client flask-client*

**3. Сборка проекта с помощью Docker Compose**

Очевидно, что постоянно использовать перегруженные разнообразными ключами команды сборки и запуска контейнеров по отдельности и настраивать их взаимодействие неудобно. Конечно, можно оформить отдельный bash-скрипт, последовательно выполняющий необходимые команды. Но в таком случае отладка и настройка контейнеров только усложнятся. Поэтому используем Docker Compose как удобный и понятный инструмент для работы с многоконтейнерной системой.

Создайте файл docker-compose.yml в основной директории вашего проекта. Внесите в файл следующие ключи и команды:

*# последняя версия Docker Compose*

*version: "3.8"*

*# список сервисов*

*services:*

*# название*

*server:*

*# сборка из указанной директории*

*build: server/*

*# проброс портов хоста*

*ports:*

*- 1200:1200*

*client:*

*build: client/*

*# запуск только после запуска сервера*

*depends\_on:*

*- server*

Теперь выполните сборку и запуск контейнеров двумя простыми командами:

*docker compose build*

*docker compose up*

Если всё выполнено верно, в терминале должен отобразиться всё тот же текст html-страницы, полученный клиентом от сервера.

**4. Задача лабораторной работы**

Выполните все указанные в предыдущих пунктах команды, продемонстрируйте их результаты с помощью скриншотов.

Используя изученные команды и прикреплённые ниже источники, реализуйте проект Docker Compose со следующей архитектурой:

- сервер, предоставляющий по указанному порту статичную html-страницу;

- промежуточный сервер, запрашивающий у первого сервера эти данные, добавляющий к ним собственный текст и предоставляющий по определённому порту полученную страницу;

- клиент, запрашивающий у промежуточного сервера вторую страницу и выводящий результат в консоль.

Для создания веб-серверов рекомендуется использовать библиотеку Flask.

Контейнеры клиента и первого сервера не должны иметь доступа друг к другу. Продемонстрируйте содержимое файлов, используемых в проекте, скриншоты из терминала и браузера при запросе по указанным портам. Протестируйте соединение между каждой парой контейнеров с помощью утилиты ping и прикрепите соответствующие скриншоты.

**Демонстрация результатов**

Результаты выполнения пунктов 1 и 2 можно найти в текущем репозитории, в каталоге version0/. Команды, используемые в консоли, содержатся в файле command.md.

Результаты выполнения пункта 3 настоящей лабораторной работы содержатся в каталоге version1/.

Пример выполнения задачи, поставленной в лабораторной работе, представлен в каталоге version2/.

**Выводы**

В лабораторной работе были показаны базовые инструменты для работы с менеджером многоконтейнерных проектов Docker Compose как наиболее удобного средства для разработки сложных сервисов. Изучены основные команды для составления docker-compose.yml файлов, механизмы использования виртуальных сетей в Docker Compose и портов хоста и контейнеров. Кроме того, были освоены методы работы с библиотекой создания веб-серверов Flask и библиотекой клиентских запросов Requests на языке Python.

**Список использованных источников**:

1. Хабр: Руководство по Docker Compose для начинающих – <https://habr.com/ru/companies/ruvds/articles/450312/>
2. Хабр: Docker Compose – <https://habr.com/ru/companies/otus/articles/731854/>
3. Docker Docs – <https://docs.docker.com/>
4. Docker Docs: Docker Compose overview – <https://docs.docker.com/compose/>
5. Python 3.12.1 documentation – <https://docs.python.org/3/>
6. Flask documentation – https://flask.palletsprojects.com/en/3.0.x/
7. Хабр: Flask для начинающих– <https://habr.com/ru/articles/783574/>
8. Requests 2.31.0 documentation – <https://requests.readthedocs.io/en/latest/>