Задание на выполнение практической работы №7
«Создание контейнера с веб сервером Nginx в Docker»
по дисциплине «Настройка и администрирование сервисного программного обеспечения»

Оформление:

1. Титульный лист должен включать:
   - название института,
   - название кафедры;
   - название дисциплины;
   - название практической работы;
   - вариант;
   - ФИО студента;
   - группу;
   - ФИО преподавателя.
2. Цель работы.
3. Теоретическое введение (не более страницы).
4. Выполнение работы.
   - задание;
   - решение;
   - результат (**команды писать текстом**, не нужно приводить скриншоты для кода. Заранее сохраняйте логи действий по работе с приложениями, фиксируйте все необходимые команды, которые имеет смысл поместить в отчет по работе **Скриншоты применять только там, где очевидно, что это картинка**.).
5. Выводы.
6. Библиографический список по ГОСТ (Рекомендуется пользоваться электронными библиотечными системами, например, e.lanbook.com для получения библиографических записей на необходимые книги по теме работы). Электронные источники информации оформлять в соответствии с ГОСТ http://inion.ru/site/assets/files/1085/gost_r_7_0-2021.pdf

Цель работы: получить навыки сборки Docker-контейнера с использованием Docker-compose.

Общие сведения

Docker применяется для управления отдельными контейнерами (сервисами), из которых состоит приложение.

Применяя Docker на пользовательской системе появляется возможность отделить компоненты разработки от системы и друг от друга. Таким образом, можно не устанавливать веб-сервер в операционную систему, а сделать несколько серверов в отдельных каталогах и потом в эти каталоги складывать проекты которые будут работать в готовом окружении, которое можно запускать когда понадобиться.

Docker Compose используется для одновременного управления несколькими контейнерами, входящими в состав приложения. Этот инструмент предлагает те же возможности, что и Docker, но позволяет работать с более сложными приложениями.

Пример. Требуется перевести проект на другую машину.

Без применения Docker Compose придётся переносить и перенастраивать сервисы по одному.

Если применяется Docker Compose, то перенос проекта на новый сервер происходит достаточно быстро. Для того чтобы завершить перенос проекта на новое место, нужно выполнить настройки и загрузить на новый сервер резервную копию базы данных.

Docker-compose позволяет запускать несколько контейнеров, связывать их и определять различные свойства контейнера в одном файле.

Этот файл называется docker-compose.yml.

Docker Compose управляет контейнерами, запускает их вместе, в нужной последовательности.

Docker-compose написан в формате YAML который похож на JSON или XML. В формате YAML имеют значения пробелы и табуляции, именно пробелами отделяются названия параметров от их значений.

Рассмотрим пример, в котором требуется создать клиент-серверное

приложение. Пример состоит из 11 этапов.

При создании конфигурации для Docker-compose необходимо определиться со структурой root директории.

Stage 1. Создание структуры root директории.

Root директория должна содержать следующие файлы и папки:

- папка client. Здесь будут находиться файлы клиентского приложения.
- папка server. Она будет содержать файлы, необходимые для обеспечения работы сервера.
- файл docker-compose.yml. Это файл Docker Compose, который будет содержать инструкции, необходимые для запуска и настройки сервисов.

В результате получится следующая структура корневой директории:

```
.
├── client/
├── server/
└── docker-compose.yml
2 directories, 1 file
```

Stage 2. Создание файлов в директории server/

Создадим в директории server/ следующие файлы:

- файл server.py. В нём будет находиться код сервера.
- файл index.html. В этом файле будет находиться фрагмент текста, который должно вывести клиентское приложение.
- файл Dockerfile. Это — файл Docker, который будет содержать инструкции, необходимые для создания окружения сервера.

В результате получится следующая структура директории server/:

```
.
├── server.py
├── index.html
└── Dockerfile
0 directories, 3 files
```

Stage 3. Редактирование файла server.py в директории server/

Файл server.py будет содержать следующий код:

```
#!/usr/bin/env python3

# Импорт системных библиотек python.
# Эти библиотеки будут использоваться для создания веб-сервера.
# Эти библиотеки устанавливаются вместе с Python.
import http.server
import socketserver

# Эта переменная нужна для обработки запросов клиента к серверу.
handler = http.server.SimpleHTTPRequestHandler

# Сервер будет работать на порте 1234.
# Номер порта пригодится в дальнейшем при работе с docker-compose.
with socketserver.TCPServer(("", 1234), handler) as httpd:

    # Эта команда заставляет сервер работать постоянно, ожидая запросов от клиента.
    httpd.serve_forever()
```

Этот код позволяет создать простой веб-сервер. Он будет отдавать клиентам файл index.html, содержимое которого позже будет выводиться на веб-

странице.

Stage 4. Редактирование HTML-файла index.html в директории server/

Файл будет содержать следующий текст:

=DOCKER-COMPOSE READY=

Этот текст будет передаваться клиенту. Можно ввести любой другой текст.

Stage 5. Редактирование файла Dockerfile в директории server/

Dockerfile будет отвечать за организацию среды выполнения для Python-сервера. В качестве основы создаваемого образа воспользуемся официальным образом, предназначенным для выполнения программ, написанных на Python.

Содержимое Dockerfile будет таким:

```
# Dockerfile всегда должен начинаться с импорта базового образа.
# Для этого используется ключевое слово 'FROM'.
# Импортируем образ python (с DockerHub).
# В результате в качестве имени образа указываем 'python', а в качестве версии - 'latest'.

FROM python:latest

# Для того чтобы запустить в контейнере код, написанный на Python, нужно импортировать файлы 'server.py' и 'index.html'.
# Для того, чтобы это сделать используется ключевое слово 'ADD'.
# Первый параметр, 'server.py', представляет собой имя файла, хранящегося на компьютере.
# Второй параметр, '/server/', это путь, по которому нужно разместить указанный файл в образе.
```

```
# Помещаем файл в папку образа '/server/'.

ADD server.py /server/
ADD index.html /server/

# Воспользуемся командой 'WORKDIR'.
# Она позволяет изменить рабочую директорию образа.
# В качестве такой директории, в которой будут выполняться все команды, мы устанавливаем '/server/'.

WORKDIR /server/
```

Stage 6. Создание файлов в директории client/

Создадим в директории client/ следующие файлы:

- файл client.py. В нём будет находиться код клиента.
- файл Dockerfile. Это — файл Docker, который будет содержать инструкцию, описывающую создание среды для выполнения клиентского кода.

В результате получится следующая структура директории client/:

```
.
├── client.py
└── Dockerfile
0 directories, 2 files
```

Stage 7. Редактирование Python-файла client.py в директории client/

Файл client.py будет содержать следующий код:

```
#!/usr/bin/env python3

# Импортируем системную библиотеку Python.
```

```
# Она используется для загрузки файла 'index.html' с сервера.
# Эта библиотека устанавливается вместе с Python.
import urllib.request

# Эта переменная содержит запрос к 'http://localhost:1234/'.
# localhost указывает на то, что программа работает с локальным сервером.
# 1234 - это номер порта
fp = urllib.request.urlopen("http://localhost:1234/")

# 'encodedContent' соответствует закодированному ответу сервера ('index.html').
# 'decodedContent' соответствует раскодированному ответу сервера (то, что выведется на экран).
encodedContent = fp.read()
decodedContent = encodedContent.decode("utf8")

# Выводим содержимое файла, полученного с сервера ('index.html').
print(decodedContent)

# Закрываем соединение с сервером.
fp.close()
```

Благодаря этому коду клиентское приложение может загрузить данные с сервера и вывести их на экран.

Stage 8. Редактирование файла Dockerfile в директории client/

Dockerfile будет отвечать за организацию среды выполнения для Python-клиента.

Содержимое Dockerfile будет таким:

```
# То же самое, что и в серверном Dockerfile.
FROM python:latest

# Импортируем 'client.py' в папку '/client/'.
ADD client.py /client/

# Устанавливаем в качестве рабочей директории '/client/'.
WORKDIR /client/
```

Stage 9. Редактирование файла docker-compose.yml в root директории

Содержимое файла будет таким:

```
# Файл docker-compose должен начинаться с тега версии.
# Используем "3.8" так как это - самая свежая версия на момент написания этого кода.
# Подробнее о различиях в версиях конфигурации docker-compose можно посмотреть на официальном ресурсе https://docs.docker.com/compose/compose-file/compose-versioning/#compatibility-matrix
version: "3.8"

# Следует учитывать, что docker-compose работает с сервисами.
# 1 сервис = 1 контейнер.
# Сервисом может быть клиент, сервер, сервер баз данных...
# Раздел, в котором будут описаны сервисы, начинается с 'services'.

services:

  # В примере создаётся клиентское и серверное приложения.
  # Это означает, что нужно два сервиса.
  # Первый сервис (контейнер): сервер.
```

```
  # Назвать его можно так, как нужно разработчику.
  # Понятное название сервиса помогает определить его роль.
  # Поэтому, для именования соответствующего сервиса, используется ключевое слово 'server'.

  server:

    # Ключевое слово "build" позволяет задать
    # путь к файлу Dockerfile, который нужно использовать для создания образа,
    # который позволит запустить сервис.
    # Здесь 'server/' соответствует пути к папке сервера,
    # которая содержит соответствующий Dockerfile.

    build: server/

    # Команда, которую нужно запустить после создания образа.
    # Следующая команда означает запуск "python ./server.py".

    command: python ./server.py

    # Не забываем, что в качестве порта в 'server/server.py' указан порт 1234.
    # Если мы хотим обратиться к серверу с нашего компьютера (находясь за пределами контейнера),
    # мы должны организовать перенаправление этого порта на порт компьютера.
    # Сделать это можно используя ключевое слово 'ports'.
    # При его использовании применяется следующая конструкция: [порт компьютера]:[порт контейнера]
    # В нашем случае нужно использовать порт компьютера 1234 и организовать его связь с портом
```

```
    # 1234 контейнера (так как именно на этот порт сервер
    # ожидает поступления запросов).

    ports:
      - 1234:1234

  # Второй сервис (контейнер): клиент.
  # Этот сервис назван 'client'.

  client:
    # Здесь 'client/ соответствует пути к папке, которая содержит
    # файл Dockerfile для клиентской части системы.

    build: client/

    # Команда, которую нужно запустить после создания образа.
    # Следующая команда означает запуск "python ./client.py".

    command: python ./client.py

    # Ключевое слово 'network_mode' используется для описания типа сети.
    # Тут мы указываем то, что контейнер может обращаться к 'localhost' компьютера.

    network_mode: host

    # Ключевое слово 'depends_on' позволяет указывать, должен ли сервис,
    # прежде чем запуститься, ждать, когда будут готовы к работе другие сервисы.
    # Нам нужно, чтобы сервис 'client' дождался бы готовности к работе сервиса
```

'server'.

```
depends_on:
  - server
```

Stage 10. Сборка проекта приложением docker-compose

В docker-compose.yml внесены все необходимые инструкции, теперь проект нужно собрать.

```
$ docker-compose build
```

Stage 11. Запуск проекта приложением docker-compose

Для запуска проекта необходимо выполнить команду:

```
$ docker-compose up
```

Аналогом команды при работе с отдельными контейнерами, выполняется команда docker run.

После выполнения этой команды в терминале должен появиться текст, загруженный клиентом с сервера: =DOCKER-COMPOSE READY=.

Сервер использует порт основной операционной системы 1234 для обслуживания запросов клиента. Поэтому, если перейти в браузере по адресу http://localhost:1234/, в нём будет отображена страница с текстом =DOCKER-COMPOSE READY=

Краткая справка по командам docker-compose

Чтобы просмотреть справку по приложению:

```
$ docker-compose –help
```

В docker-compose есть две пары команд для запуска и остановки сервисов. up и down управляют сервисом целиком, в то время как start и stop запускают отдельные компонент-контейнеры.

Команда запускает сервисы с нуля: создаёт сети, если нужно, собирает Dockerfile и запускает контейнеры. Если добавить -d параметр (как и в docker run -d), то сервис запустится в фоновом процессе.

$ docker-compose up -d

Дополнительно, compose добавляет к именам сетей и контейнеров префикс (такой же, как имя текущей папки), но внутри созданной им сети можно использовать как оригинальное имя (web, db) так и измененное (dockercompose_db_1).

Команда останавливает и затем удаляет контейнеры и другие ресурсы (в том числе сети):

$ docker-compose down

Команды запуска и остановки сервис-контейнеров

Иногда в уже запущенном сервисе нужно остановить какой-нибудь компонент-контейнер не удаляя его. Для этого нужно использовать команды start и stop.

$ docker-compose stop web

#Stopping dockercompose_web_1 ... done

#...do stuff

$ docker-compose start web

#Starting web ... done

Команда выводит журналы сервисов:

$ docker-compose logs -f [service name]

Команда вывода списка контейнеров:

$ docker-compose ps

На экране появится следующий вывод процессов docker:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e90e12f70418	nginx	"nginx -g 'daemon off"	6 minutes ago	Up 5 minutes	80/tcp, 443/tcp	nginx_nginx_1

Команда exec

Запускает командную строку указанного контейнера

Чтобы внести изменения в файловую систему контейнера, используйте ID – контейнера (в данном случае это e90e12f70418) и команду docker exec. Команда запустит оболочку в контейнере.

$ docker exec -it e90e12f70418 /bin/bash

Опция –t открывает терминал, а опция –i запускает интерактивный режим. Опция /bin/bash открывает оболочку bash для запущенного контейнера (команда, выполняемая в контейнере).

Так будет выглядеть системная подсказка контейнера с ID e90e12f70418, к которому обратились:

root@e90e12f70418:/#

Теперь можно взаимодействовать с контейнером из командной строки.

Кроме того, нужно помнить, что большинство контейнеров создаются как минимальные установки Linux, поэтому некоторые команды и утилиты в них недоступны.

Имейте ввиду: все изменения будут утеряны, как только контейнер перезапустится, если каталог не входит в том данных.

Другой пример exec:

$ docker-compose exec web bash

где web – контейнер

bash – выполняемая команда в контейнере

Общий вид команды exec:

$ docker-compose exec [service name] [command]

Например, команда может выглядеть так:

$ docker-compose exec server ls

Команда вывода списка образов:

$ docker-compose images

Команда запуска сервисов, описанных в docker-compose.yml

$ docker-compose up -d

Ключ -d указывает docker compose запускать контейнеры в фоновом режиме.

При необходимости с помощью команды docker-compose up можно запустить конкретный сервис:

$ docker-compose up my_container1

Сервис должен быть описан в docker-compose.yml

Задание на практическую работу

Используя инструкцию, представленную в общих сведениях Docker-compose создать следующие сервисы. Работу каждого сервиса продемонстрировать преподавателю.

1. Повторить пример, который был рассмотрен в разделе «Общие сведения». Проект назвать docker_py. Обязательно продемонстрировать сервис при обсуждении с преподавателем отчёта практической работы.
2. Создать аналогичный сервис как в примере, рассмотренном в разделе «Общие сведения», но вместо языка Python использовать язык JavaScript. Проект назвать docker_js. Обязательно продемонстрировать сервис при обсуждении с преподавателем отчёта практической работы.
3. Создать сервис как в примере, рассмотренном в разделе «Общие сведения», но вместо сервера на Python использовать веб сервер Apache. Проект назвать docker_apache. Обязательно продемонстрировать сервис при обсуждении с преподавателем отчёта практической работы.

4. Создать сервис как в примере, рассмотренном в разделе «Общие сведения», но вместо сервера на Python использовать веб сервер Nginx. Проект назвать docker_nginx. Обязательно продемонстрировать сервис при обсуждении с преподавателем отчёта практической работы.
5. Создать сервис в состав которого входят следующие компоненты apache, mysql, phpMyAdmin. Проект назвать docker_phpmyadmin. Обязательно продемонстрировать сервис при обсуждении с преподавателем отчёта практической работы.

Рекомендуемые материалы для ознакомления

DOCKER + DOCKER COMPOSE – ЗАПУСК КОНТЕЙНЕРОВ ДЛЯ ВЕБРАЗРАБОТКИ. Раздел: Docker - конфигурирование отдельного проекта – URL: https://atlogex.com/docker-start/ (дата обращения: 25.03.2021). Рекомендуется читать с указанного раздела.

Официальная документация Dockerfile reference – URL: https://docs.docker.com/engine/reference/builder/ (дата обращения: 25.03.2021)

Сейерс, Э. Х. Docker на практике / Э. Х. Сейерс, А. Милл ; перевод с английского Д. А. Беликов. — Москва : ДМК Пресс, 2020. — 516 с. — ISBN 978-5-97060-772-5. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/131719 (дата обращения: 23.03.2021). — Режим доступа: для авториз. пользователей.

Книга доступна также по адресу: https://vk.com/wall-51126445_66963 (дата обращения: 25.03.2021)