Липецкий государственный технический университет

Кафедра прикладной математики

Отчет по лабораторной работе №6
«Контейнеризация.»

Студент _____ Стукановский А.О.
подпись, дата фамилия, инициалы

Группа ПМ-18

Руководитель
доц., к.п.н. кафедры АСУ _____ Кургасов В. В.
ученая степень, ученое звание подпись, дата фамилия, инициалы

Липецк 2020 г.

Содержание

Цель работы

Изучить современные методы разработки ПО в динамических и распределённых средах на примере контейнеров Docker.

Практическое задание

1) С помощью Docker Compose на своем компьютере поднять сборку nginx+php-fpm+postgres, продемонстрировать ее работоспособность, запустив внутри контейнера демо-проект на symfony. По умолчанию проект работает с sqlite-базой. Нужно заменить ее на postgres;

2) Создать новую БД в postgres.

3) Заменить DATABASE_URL в .env на строку подключения к postgres;

4) Создать схему БД и заполнить ее данными из фикстур, выполнив в консоли (
   php bin/console doctrine:schema:create,
   php bin/console doctrine:fixtures:load).

Проект должен открываться по адресу http://demo-symfony.local/ (Код проекта должен располагаться в папке на локальном хосте). Для компонентов nginx, fpm есть готовые docker-образы, их можно и нужно использовать. Нужно расшарить папки с локального хоста, настроить подключение к БД. В .env переменных для postgres нужно указать путь к папке, где будет лежать база, чтобы она не удалялась при остановке контейнера. На выходе должен получиться файл конфигурации docker-compose.yml и .env файл с настройками переменных окружения.

Дополнительные требования:

Postgres также должен работать внутри контейнера. В .env переменных нужно указать путь к папке на локальном хосте, где будут лежать файлы БД, чтобы она не удалялась при остановке контейнера.

Выполнение практического задания

Первоначально произведём клонирование текстового проекта с помощью команды «`get clone https://github.com/symfony/demo`», после чего совершаем переход в папку с проектом с помощью команды «`cd demo`».

Прежде чем запустить проект командой «`php bin/console server:start`», выполним установку php последней версии. После выполнения данного этапа в браузере стал доступен проект по адресу `http://localhost:8000`. Главное окно приложения имеет вид (рисунок 1).

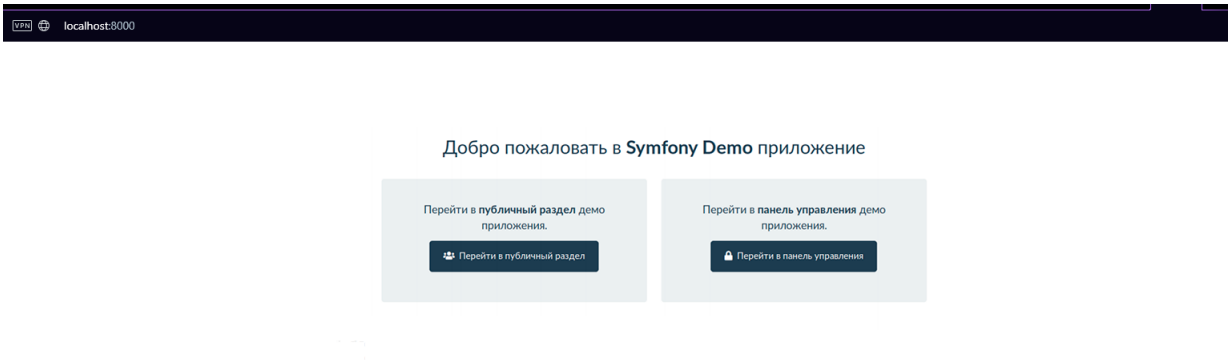

Рисунок 1.

Выполним установку бесплатной версии Docker на Ubuntu 19, который в официальном репозитории поставщика называется docker-ce. Для успешной установки выполним следующие команды:

1) `sudo apt install apt-transport-https ca-certificates curl software-properties-common`
2) `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -`
3) `sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu disco test"`
4) `sudo apt update`
5) `sudo apt install docker-ce`

Скачиваем файл и сохраняем его под другим именем `/usr/local/bin/docker-compose` (рисунок 2).

```
artem@ubuntuserver:~$ sodu curl -L https://github.com/docker/compose/releases/download/1.25.0-rc4/do
cker-compose-'uname -s'-'uname -m' -o /usr/local/bin/docker-compose
sodu: command not found
artem@ubuntuserver:~$ sudo curl -L https://github.com/docker/compose/releases/download/1.25.0-rc4/do
cker-compose-'uname -s'-'uname -m' -o /usr/local/bin/docker-compose
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1438    0  1438    0     0   7083      0 --:--:-- --:--:-- --:--:--  7049
artem@ubuntuserver:~$ _
```

Рисунок 2.

Изменим права доступа к данному файлу, позволив всем пользователям выполнять его (рисунок 3).

```
artem@ubuntuserver:~$ sudo chmod +x /usr/bin/docker-compose
chmod: cannot access '/usr/bin/docker-compose': No such file or directory
artem@ubuntuserver:~$ sudo chmod +x /usrlocal/bin/docker-compose
chmod: cannot access '/usrlocal/bin/docker-compose': No such file or directory
artem@ubuntuserver:~$ sudo chmod +x /usr/local/bin/docker-compose
artem@ubuntuserver:~$ _
```

Рисунок 3.

Создадим жёсткую символическую ссылку `/usr/bin/docker-compose` (рисунок 4).

```
artem@ubuntuserver:~$ sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
artem@ubuntuserver:~$
```

Рисунок 4.

Выполним установку «`sudo apt install composer`». Composer – менеджер зависимостей PHP. Он отслеживает необходимые проекту библиотеки, извлекает и устанавливает их.

Следующим шагом установим postgresql, зайдём в консоль управления и создадим базу данных «`demo`» и пользователя «demodb_user». Проверим наличие созданной базы данных и пользователя (рисунок 5).

```
                                  List of databases
   Name    |  Owner   | Encoding | Collate |  Ctype  |    Access privileges
-----------+----------+----------+---------+---------+-------------------------
 demo      | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =Tc/postgres           +
           |          |          |         |         | postgres=CTc/postgres  +
           |          |          |         |         | demodb_user=CTc/postgres
 postgres  | postgres | UTF8     | C.UTF-8 | C.UTF-8 |
 template0 | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres            +
           |          |          |         |         | postgres=CTc/postgres
 template1 | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres            +
           |          |          |         |         | postgres=CTc/postgres
(4 rows)

postgres=# \du
                                   List of roles
  Role name  |                         Attributes                         | Member of
-------------+------------------------------------------------------------+-----------
 demo_user   | Superuser, Create role, Create DB                          | {}
 demodb_user |                                                            | {}
 postgres    | Superuser, Create role, Create DB, Replication, Bypass RLS | {}

postgres=#
```

Рисунок 5.

Отредактируем настройки окружения в `.env` (рисунок 6).

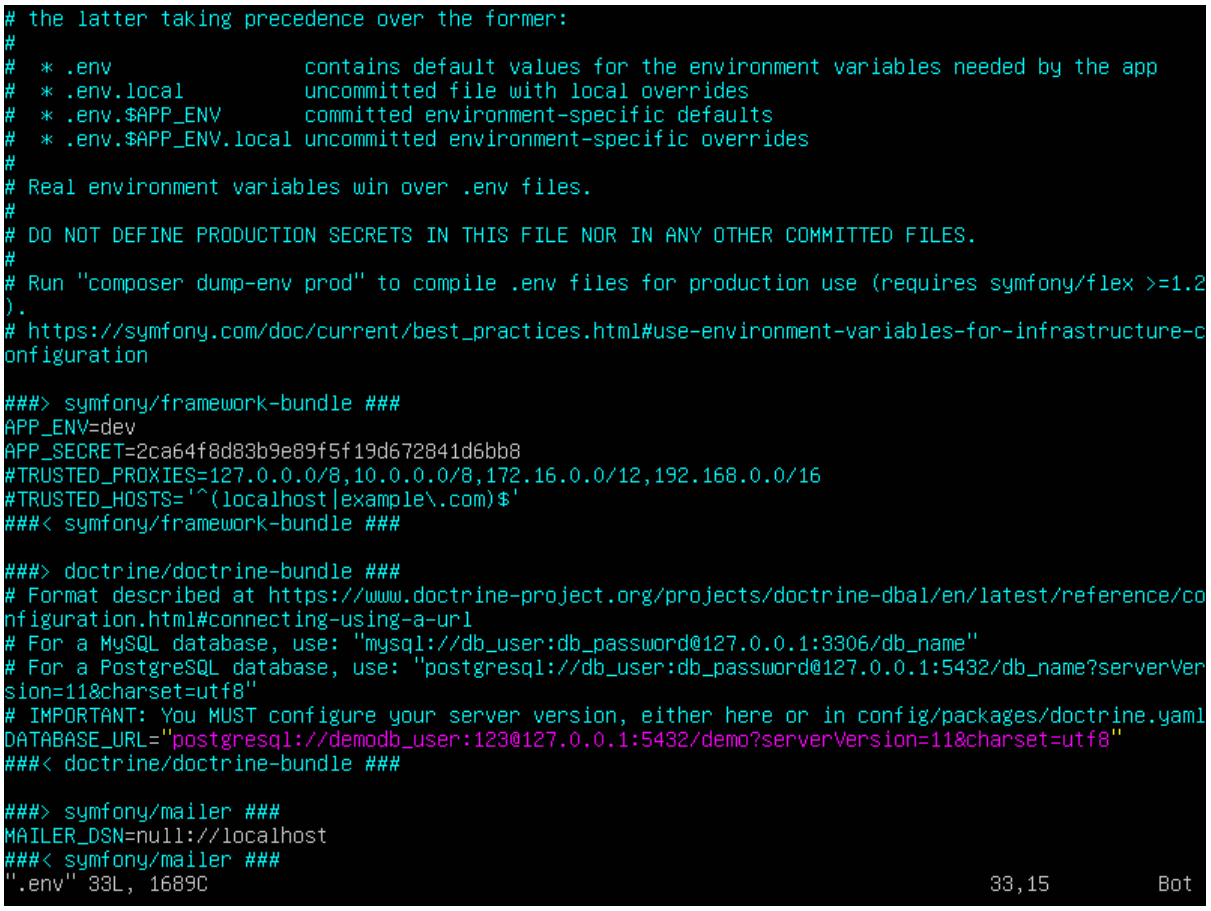

Рисунок 6.

Далее загрузим схему БД командой «`php bin/console doctrine:schema:create`» (рисунок 7) и сами данные командой «`php bin/console doctrine:fixtures:load`» (рисунок 8).

Рисунок 7.

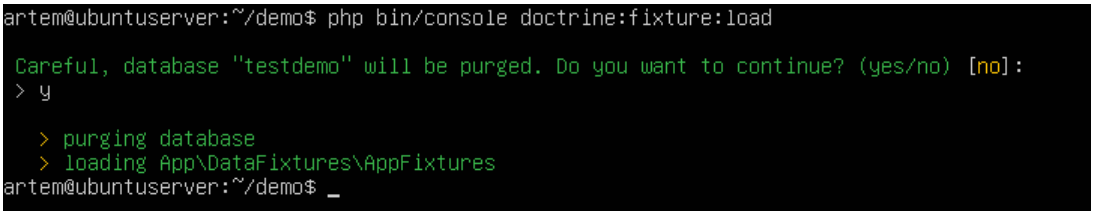

Рисунок 8.

В папке с проектом создадим файл `Dockerfile` и заполним его следующим содержимым (рисунок 9):

```
FROM richarvey/nginx-php-fpm
WORKDIR /var>>/www/html/demo
COPY composer.json ./
RUN COMPOSER_MEMORY_LIMIT=-1 composer install
COPY . .
EXPOSE 8000
CMD ["php", "-S", "0.0.0.0:8000", "-t", "public/"]
```

Рисунок 9.

Создадим файл `docker-compose.yml` и заполним его следующим содержимым (рисунок 10):

```
version: "3"
services:
        app:
                container_name: docker-node-mongo
                restart: always
                build: .
                ports:
                        - "80"
                links:
                        - postgres
        postgres:
                container_name: postgres
                image: postgres
                ports:
                        - "5432"
                environment:
                        POSTGRES_USER: demodb_user
                        POSTGRES_PASSWORD: 123
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"docker-compose.yml" 19L, 530C                                  8,29          All
```

Рисунок 10.

Выполним команду «`docker-compose up -d`», которая выполнит скачивание образов и установку зависимостей composer. Видим, что контейнер работает (рисунок 11).

```
docker-node-mongo is up-to-date
Attaching to postgres, docker-node-mongo
docker-node-mongo | [Sun Dec 27 17:43:59 2020] PHP 7.4.10 Development Server (http://0.0.0.0:8000) s
tarted
docker-node-mongo | [Fri Jan  8 10:29:59 2021] PHP 7.4.10 Development Server (http://0.0.0.0:8000) s
tarted
docker-node-mongo | [Fri Jan  8 10:38:19 2021] PHP 7.4.10 Development Server (http://0.0.0.0:8000) s
tarted
docker-node-mongo | [Fri Jan  8 12:20:17 2021] PHP 7.4.10 Development Server (http://0.0.0.0:8000) s
tarted
docker-node-mongo | [Sat Jan  9 06:00:15 2021] PHP 7.4.10 Development Server (http://0.0.0.0:8000) s
tarted
docker-node-mongo | [Mon Jan 11 12:57:15 2021] PHP 7.4.10 Development Server (http://0.0.0.0:8000) s
tarted
docker-node-mongo | [Fri Jan 29 13:18:06 2021] PHP 7.4.10 Development Server (http://0.0.0.0:8000) s
tarted
docker-node-mongo | [Tue Feb  2 14:37:56 2021] PHP 7.4.10 Development Server (http://0.0.0.0:8000) s
tarted
postgres          |
postgres          | PostgreSQL Database directory appears to contain a database; Skipping initialization
postgres          |
postgres          | 2021-02-02 15:11:52.767 UTC [1] LOG:  starting PostgreSQL 13.1 (Debian 13.1-1.pgdg100+
1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 8.3.0-6) 8.3.0, 64-bit
postgres          | 2021-02-02 15:11:52.801 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
postgres          | 2021-02-02 15:11:52.802 UTC [1] LOG:  listening on IPv6 address "::", port 5432
postgres          | 2021-02-02 15:11:52.805 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s
.PGSQL.5432"
postgres          | 2021-02-02 15:11:52.815 UTC [26] LOG:  database system was interrupted; last known up
at 2020-12-27 17:43:49 UTC
postgres          | 2021-02-02 15:11:53.008 UTC [26] LOG:  database system was not properly shut down; aut
omatic recovery in progress
postgres          | 2021-02-02 15:11:53.012 UTC [26] LOG:  redo starts at 0/15CD5A0
postgres          | 2021-02-02 15:11:53.012 UTC [26] LOG:  invalid record length at 0/15CD5D8: wanted 24,
got 0
postgres          | 2021-02-02 15:11:53.012 UTC [26] LOG:  redo done at 0/15CD5A0
postgres          | 2021-02-02 15:11:53.099 UTC [1] LOG:  database system is ready to accept connections
```

Рисунок 11.

Перейдём на страницу сайта и проверим всё ли работает корректно. Как видно, сайт работает(рисунок 12)

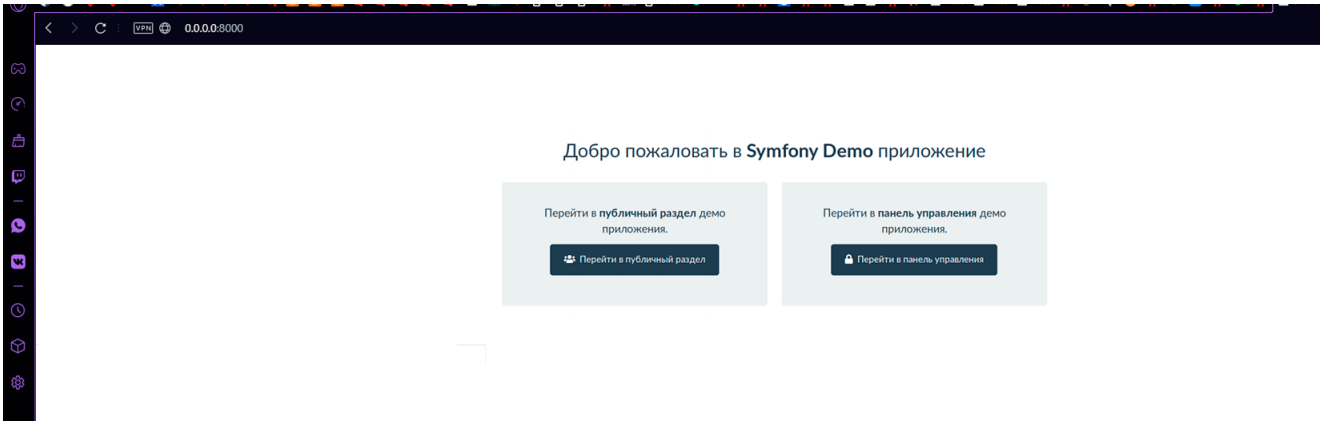

Рисунок 12.

Осуществим настройку контейнеров. Внутри папки тестрового проекта создадим папку docker «`mkdir docker`» и перейдём в неё «`cd docker`» .

Внутри папки docker создадим файл docker-compose.yaml с конфигурацией всех контейнеров (рисунок 13).

Рисунок 13.

Также в папке docker создадим папки nginx «`mkdir nginx`» и php-fpm «`mkdir php-fpm`». В папке nginx создадим файл Dockerfile с конфигурацией контейнера nginx (рисунок 14).

```
FROM nginx:latest
COPY default.conf /etc/nginx/conf.d/
```

Рисунок 14.

В папке php-fpm создадим файл Dockerfile с конфигурацией контейнера php-fpm (рисунок 15).

```
FROM php-fpm
RUN apt-get update
RUN apt-get install -y zliblg-dev libpq-dev git libicu-dev libxml2-dev \
&& docker-php-ext-configure int1 \
&& docker-php-ext-install int1 \
&& docker-php-ext-configure pgsql -with-pgsql=/usr/local/pgsql \
&& docker-php-ext-install pdo pdo_pgsql pgsql \
&& docker-php-ext-install zip xml
WORKDIR /var/www/symfony
```

Рисунок 15.

Перезапустим контейнеры командой «`docker-compose up -d`» и перейдём по адресу localhost.local, чтобы проверить работоспособность сайта (рисунок 16).

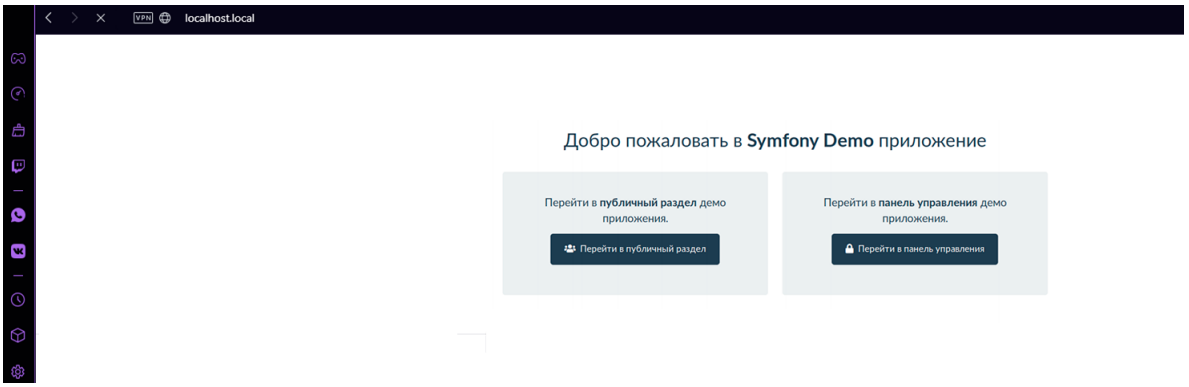
localhost.local
Добро пожаловать в Symfony Demo приложение
Перейти в публичный раздел демо приложения.
Перейти в публичный раздел
Перейти в панель управления демо приложения.
Перейти в панель управления

Рисунок 16.

Вопросы для самопроверки

Назовите отличия использования контейнеров по сравнению с виртуализацией.

A. Меньшие накладные расходы на инфраструктуру

Назовите основные компоненты Docker.

B. Контейнеры

Какие технологии используются для работы с контейнерами?

C. Контрольные группы (cgroups)

Найдите соответствие между компонентом и его описание:

образы – доступные только для чтения шаблоны приложений;

контейнеры – изолированные при помощи технологий операционной системы пользовательские окружения, в которых выполняются приложения;

реестры (репозитории) – сетевые хранилища образов.

В чём отличие контейнеров от виртуализации?

Виртуализация обеспечивает одновременную работу нескольких операционных систем на одном компьютере.

Контейнеры используют одно и то же ядро операционной системы и изолируют процессы приложения от остальной системы.

Для одновременной работы нескольких ОС в одной программе, которая реализует виртуализацию, требуется больше системных ресурсов, чем для аналогичной конфигурации на основе контейнеров. Ресурсы, как правило, не безграничны, поэтому, чем меньше «весят» приложения, тем плотнее их можно разместить на серверах. Все Linux-контейнеры, работающие на компьютере, используют одну и ту же ОС, поэтому приложения и сервисы остаются легковесными и работают в параллельном режиме, не тормозя друг друга.

Перечислите основные команды утилиты Docker с их кратким описанием.

docker run – Создание нового контейнера образа и его выполнение.

docker start – Запуск существующего контейнера. Данная команда используется в случае необходимости повторного запуска давшего сбой или совершившего выход контейнера.

docker stop — Остановка существующего контейнера Docker.

docker build – Сборка образа Docker из файла докер (dockerfile) и контекста сборки. Контекст сборки — это набор файлов, расположенных по определенному пути.

docker ps – Вывод списка запущенных контейнеров. Можно передать несколько параметров: q — «тихий» режим, в котором команда выводит только id контейнеров; -a — показывает все контейнеры, а не только запущенные.

docker exec -ti – Выполнение команды оболочки внутри конкретного контейнера.

Каким образом осуществляется поиск образов контейнеров?

Изначально Docker проверяет локальный репозиторий на наличие нужного образа. Если образ не найден, Docker проверяет удаленный репозиторий Docker Hub.

Каким образом осуществляется запуск контейнера?

Docker выполняет инициализацию и запуск ранее созданного по образу контейнера по имени. Для запуска Docker необходимо иметь права root-пользователя или пользователя из группы Docker, создаваемой автоматически во время установки сервиса. Если попробовать запустить в терминале Docker без этих прав или, не являясь пользователем группы docker, появится ситуация, когда не запускается контейнер.

Что значит управлять состоянием контейнеров?

Контейнеры Docker могут находиться в нескольких состояниях:

Исполняется (running) – контейнер работает. В выводе docker ps можно увидеть статус «Up» и время, в течение которого он исполняется.

Создан (created) – контейнер создан, но в настоящий момент не выполняется. Такое состояние будет у контейнера после команды docker create.

Завершил исполнение (exited) – контейнер завершил исполнение.

Поставлен на паузу (paused) – процесс контейнера остановлен, но существует. Поставить контейнер на паузу можно при помощи команды docker pause.

Запущен заново (restarting) – контейнер рестартует. Остановленный контейнер можно рестартовать. Рестарт означает, что контейнер заново запустится с теми же идентификатором и настройками сети, что были до остановки. Однако это будет уже другой процесс с другим PID.

«Умер» (dead) – контейнер перестал функционировать вследствие сбоя.

Управление состоянием контейнеров - это контроль хода выполнения контейнера и перевод его в конкретный режим в любой момент времени.

Как изолировать контейнер?

Для изоляции контейнера достаточно правильно сконфигурировать файлы Dockerfile и docker-compose.yaml. По умолчанию контейнеры запускаются от прав root пользователя.

Опишите последовательность создания новых образов, назначение Dockerfile?

Dockerfile - это специальный скрипт, в котором содержатся команды и инструкции, которые будут в заданной последовательности выполнены для сборки нового образа Docker.

Для создания нового образа необходимо:

1) Указать базовый образ для создания нового. Эта команда должна быть первой в Dockerfile.

2) Добавить необходимые слои.

3) Выполнить нужные операции.

4) Развернуть рабочее окружение внутри контейнера с необходимыми зависимостями.

Движок Docker-а при запуске распарсит Dockerfile и создаст из него соответствующий образ (Image), который был описан.

Возможно ли работать с контейнерами Docker без одноимённого движка?

С контейнерами Docker можно работать без одноимённого движка в среде другой виртуализации Kubernetes.

Опишите назначение системы оркестрации контейнеров Kubernetes. Перечислите основные объекты Kubernetes.

Kubernetes — это система с открытым исходным кодом, предназначенная для оркестрации контейнеров. Она позволяет создавать, обновлять и масштабировать контейнеры, не беспокоясь о вынужденном простое. Использование Kubernetes позволяет организовать многократное использование контейнеров и переключение между ними.

Основные объекты:

1) Поды — неделимая единица в платформе Kubernetes. При создании развёртывания в Kubernetes, создаются поды с контейнерами внутри (в отличие от непосредственного создания контейнеров). Каждый Pod-объект связан с узлом, на котором он размещён, и остаётся там до окончания работы (согласно стратегии перезапуска) либо удаления. В случае неисправности узла такой же под будет распределён на другие доступные узлы в кластере.

2) Узел — это рабочая машина в Kubernetes, которая в зависимости от кластера может быть либо виртуальной, либо физической. Каждый узел управляется мастером (ведущим узлом). Узел может содержать несколько подов, которые мастер Kubernetes автоматически размещает на разные узлы кластера. Ведущий узел при автоматическом планировании (распределении подов по узлам) учитывает доступные ресурсы на каждом узле.

3) Том - общий ресурс хранения для совместного использования из контейнеров, развёрнутых в пределах одного пода.

4) Все объекты управления (узлы, поды, контейнеры) в Kubernetes помечаются метками, селекторы меток — это запросы, которые позволяют получить ссылку на объекты, соответствующие какой-то из меток; метки и селекторы — это главный механизм Kubernetes, который позволяет выбрать, какой из объектов следует использовать для запрашиваемой операции.

5) Сервисом в Kubernetes называют совокупность логически связанных наборов подов и политик доступа к ним.

6) Контроллер — это процесс, который управляет состоянием кластера, пытаясь привести его от фактического к желаемому; он делает это, оперируя набором подов, который определяется с помощью селекторов меток, являющихся частью определения контроллера.

7) Операторы — специализированный вид программного обеспечения Kubernetes, предназначенный для включения в кластер сервисов, сохраняющих своё состояние между выполнениями, таких как СУБД, системы мониторинга или кэширования. Назначение операторов — предоставить возможность управления stateful-приложениями в кластере Kubernetes прозрачным способом и скрыть подробности их настроек от основного процесса управления кластером Kubernetes.

Вывод

В ходе лабораторной работы были изучены современные методы разработки ПО в динамических и распределённых средах на примере контейнеров Docker.

Список литературы

[1] Львовский, С.М. Набор и верстка в системе LaTeX [Текст] / С.М. Львовский. М.: МЦНМО, 2006. — 448 с.