

**Липецкий государственный технический университет**

**Факультет автоматизации и информатики**

**Кафедра автоматизированных систем управления**

**ЛАБОРАТОРНАЯ РАБОТА №6**

**по дисциплине «Операционная система Linux»**

**Контейнеризация**

Студент

Осипов А.А.

Группа АИ-18

Руководитель

Кургасов В.В.

Липецк 2020 г.

## Оглавление

Цель работы .....	3
Задание кафедры.....	4
Ход работы .....	5
Вывод .....	13
Контрольные вопросы.....	14

## Цель работы

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

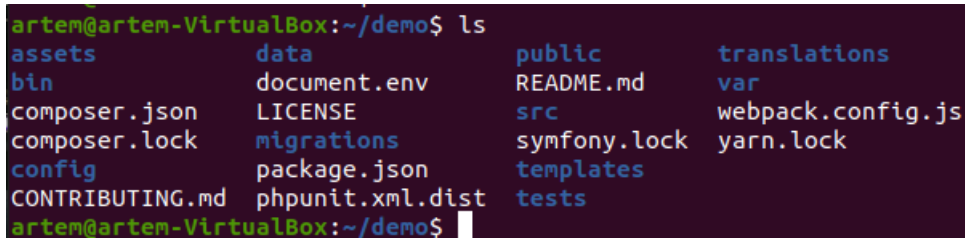
### Задание кафедры

1. С помощью Docker Compose на своем компьютере поднять сборку nginx+php-fpm+postgres, продемонстрировать ее работоспособность, запустив внутри контейнера демо-проект на symfony.
2. По умолчанию проект работает с sqlite-базой. Нужно заменить ее на 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)`.
5. Проект должен открываться по адресу <http://demo-symfony.local/> (Код проекта должен располагаться в папке на локальном хосте)
6. 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.
7. 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.
8. Реализовать подключение проекта к базе данных находящейся на локальной машине для демонстрации обновления данных в реальном времени.

Ход работы

Установим дополнительное ПО, а именно docker, docker-compose, symphony, composer, postgresql

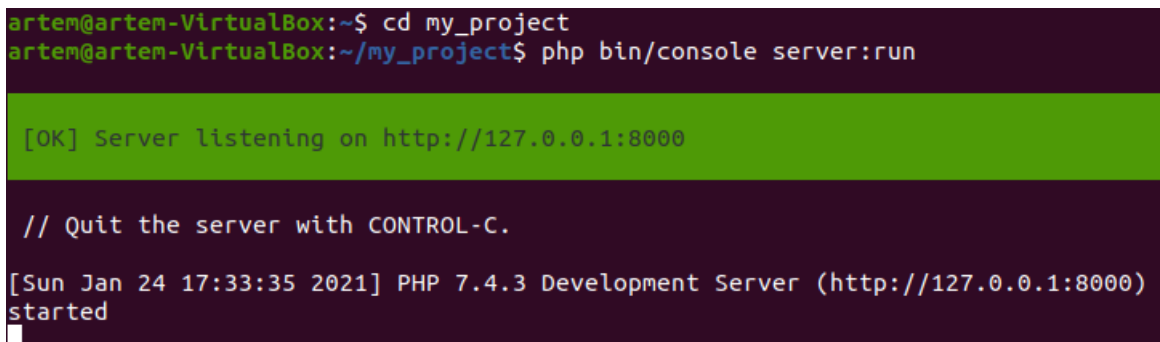
Клонируем проект с помощью команды `git clone https://github.com/symfony/demo.git`. И посмотрим результат.



```
artem@artem-VirtualBox:~/demo$ ls
assets      data        public      translations
bin          document.env README.md    var
composer.json LICENSE      src         webpack.config.js
composer.lock migrations  symfony.lock yarn.lock
config      package.json templates
CONTRIBUTING.md phpunit.xml.dist tests
```

Рисунок 1 – Клонирование проекта

Далее установим все сопутствующие зависимости с помощью команды `composer install` и запустим проект с помощью команды `php bin/console server:run`. Результат работы представлен на рисунках 2 и 3



```
artem@artem-VirtualBox:~$ cd my_project
artem@artem-VirtualBox:~/my_project$ php bin/console server:run

[OK] Server listening on http://127.0.0.1:8000

// Quit the server with CONTROL-C.

[Sun Jan 24 17:33:35 2021] PHP 7.4.3 Development Server (http://127.0.0.1:8000)
started
```

Рисунок 2 – Запуск проекта

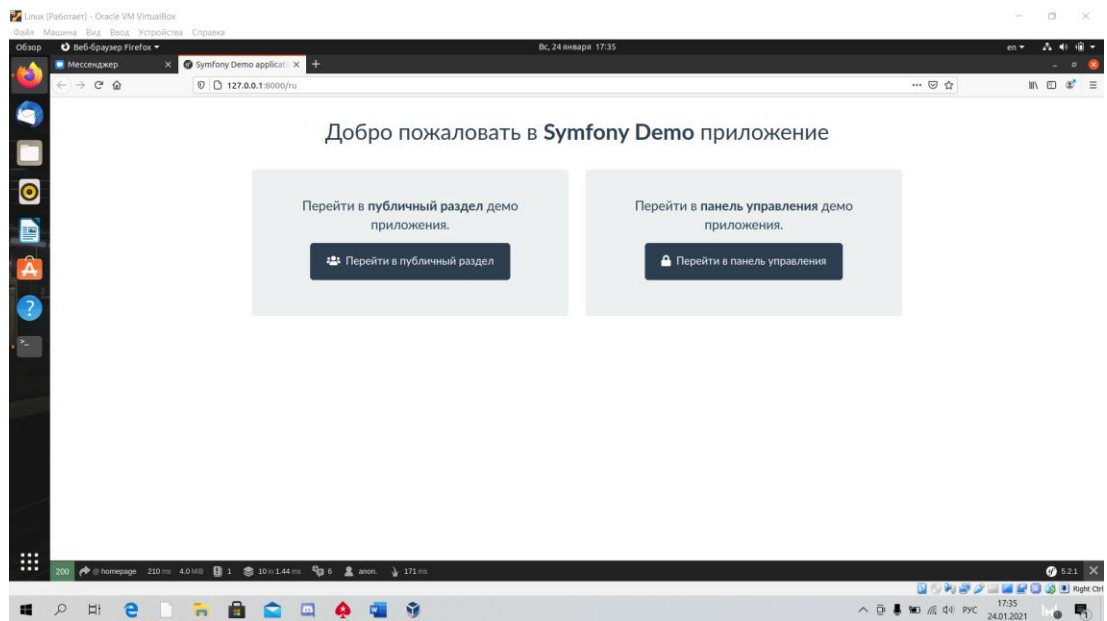


Рисунок 3 – Запуск проекта

Создадим БД для тестирования проекта

`sudo su – postgres`

```
artem@artem-VirtualBox:~/my_project$ psql -U postgres
Password for user postgres:
psql (12.5 (Ubuntu 12.5-0ubuntu0.20.04.1))
Type "help" for help.

postgres=#
```

Рисунок 4 – Создание БД

```
List of databases

```

Name	Owner	Encoding	Collate	Ctype	Access privileges
postgres	postgres	UTF8	ru_RU.UTF-8	ru_RU.UTF-8	
symfonydb	postgres	UTF8	ru_RU.UTF-8	ru_RU.UTF-8	
template0	postgres	UTF8	ru_RU.UTF-8	ru_RU.UTF-8	=c/postgres +
					postgres=CTc/postgres
template1	postgres	UTF8	ru_RU.UTF-8	ru_RU.UTF-8	=c/postgres +
					postgres=CTc/postgres

```
(4 rows)

(END)
```

Рисунок 5 – Создание БД

Далее нужно отредактировать файл окружения `.env` для того чтобы подключить БД к нашему проекту

```
artem@artem-VirtualBox: ~/my_project
GNU nano 4.8 .env
In all environments, the following files are loaded if they exist,
the latter taking precedence over the former:
* .env contains default values for the environment variables needed by the app
* .env.local uncommitted file with local overrides
* .env.SAPP_ENV committed environment-specific defaults
* .env.SAPP_ENV.local uncommitted environment-specific overrides
Real environment variables win over .env files.
DO NOT DEFINE PRODUCTION SECRETS IN THIS FILE NOR IN ANY OTHER COMMITTED FILES.
Run "composer dump-env prod" to compile .env files for production use (requires symfony/flex >=1.2).
https://symfony.com/doc/current/best_practices.html#use-environment-variables-for-infrastructure-configuration

##> symfony/framework-bundle ##
APP_ENV=dev
APP_SECRET=2ceef4fde3b9e95f19d672641d6bb6
TRUSTED_PROXIES=127.0.0.0/8,10.0.0.0/8,172.16.0.0/12,192.168.0.0/16
TRUSTED_HOSTS='(localhost|example\.)com$'
##< symfony/framework-bundle ##

##> doctrine/doctrine-bundle ##
Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html#connecting-using-a-url
For a MySQL database, use: "mysql://db_user:db_password@127.0.0.1:3306/db_name"
For a PostgreSQL database, use: "postgresql://db_user:db_password@127.0.0.1:5432/db_name?serverVersion=11&charset=utf8"
IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
DATABASE_URL=postgresql://artem:qwerty@postgres:5432/symfonydb?serverVersion=11&charset=utf8
##< doctrine/doctrine-bundle ##

##> symfony/mailer ##
MAILER_DSN=smtp://localhost
##< symfony/mailer ##
```

Рисунок 6 – Подключение БД в файле .env

Загружаем схему БД командой `php bin/console doctrine:schema:create` и заполняем данными с помощью команды `php bin/console doctrine:fixtures:load`. Проверяем работоспособность проекта, запускаем его с помощью команды `php bin/console server:run`.

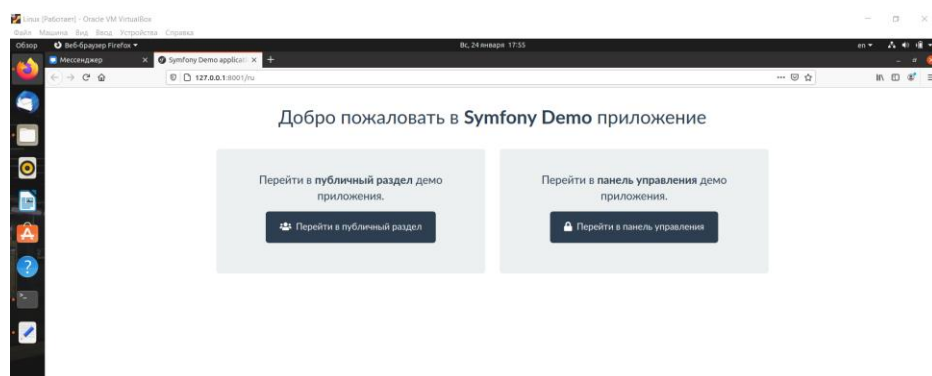


Рисунок 7 – Стартовая страница проекта

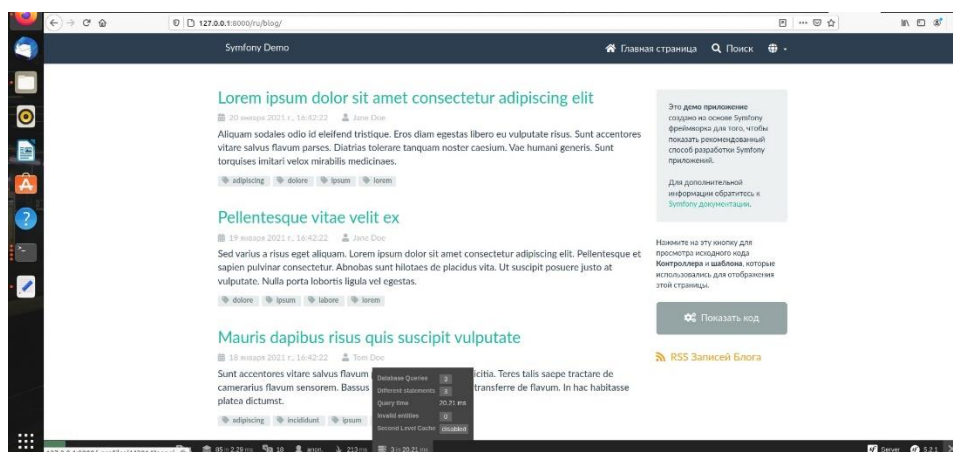


Рисунок 8 – Проверка работоспособности БД

Далее настроим контейнеры. Создадим файл `docker-compose.yml` и заполним его следующим содержимым.

```
GNU nano 4.8 docker-compose.yml
version: "3"

services:
  php-fpm:
    container_name: php-fpm
    build:
      context: .
      dockerfile: docker/php.Dockerfile
    volumes:
      - ./var/www/symfony
      - ./logs/symfony:/var/www/symfony/log
    links:
      - postgres
  nginx:
    container_name: nginx
    build:
      context: .
      dockerfile: docker/nginx.Dockerfile
    ports:
      - "8084:80"
    volumes:
      - ./var/www/symfony
      - ./logs/nginx:/var/log/nginx
    links:
      - php-fpm
  postgres:
    container_name: postgres
    image: postgres
    environment:
      - POSTGRES_DB=symfonydb
```



- POSTGRES\_USER=artem
- POSTGRES\_PASSWORD=459225

volumes:

- ./data/postgresql:/var/lib/postgresql/data

ports:

- 5432:5432

Создадим внутри папки тестового проекта папку docker (mkdir docker) и перейдем в неё (cd docker). Внутри создадим nginx.Dockerfile и php.Dockerfile, а также каталог conf, содержащий файл конфигурации nginx vhost.conf.

1. nginx.Dockerfile

FROM nginx:latest

COPY ./docker/conf/vhost.conf /etc/nginx/conf.d/default.conf

WORKDIR /var/www/symfony

2. php.Dockerfile

FROM php:7.4-fpm

WORKDIR /var/www/symfony

RUN apt-get update && apt-get install -y \

libpq-dev \

wget \

zlib1g-dev \

libmcrypt-dev \

libzip-dev

RUN docker-php-ext-install pdo pdo\_pgsql pgsql

CMD php-fpm

3. nginx vhost.conf

server {

listen 80;

root /var/www/symfony/public;

server\_name \_;

error\_log /var/log/nginx/symfony\_error.log;

```

access_log /var/log/nginx/symfony_access.log;
location / {
    try_files $uri /$uri /index.php?$query_string;
}
location ~ ^/index\.php(/|$) {
    fastcgi_pass php-fpm:9000;
    fastcgi_split_path_info ^(.+\.php)(/.*)$;
    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_param HTTPS off;
}
}

```

Далее редактируем файл `.env`, строка подключения выглядит следующим образом

```

DATABASE_URL=postgresql://artem:459225@postgres:5432/symfonydb?serverVersion=11&charset=utf8

```

Далее запускаем все контейнеры с помощью команды `docker-compose up -d` и переходим в контейнер с `postgresql`, далее переходим в консоль `psql` и создаём БД, а так же загружаем схему и данные в контейнере с `php`. Редактируем файл `/etc/hosts` добавляем псевдоним адресу `127.0.0.1 demo-symfony.local`.

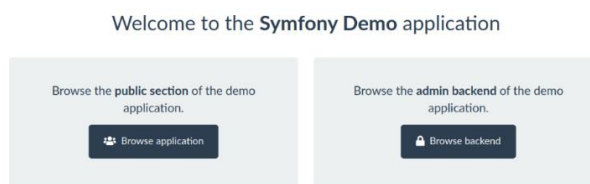


Рисунок 9 – Запуск проекта

## Часть 2

Удалим конфигурацию контейнера с `postgresql` и подключим проект к локальной БД. IP машины добавим под псевдонимом `db` в файле `docker-compose.yml`.

```
version: "3"

services:
  php-fpm:
    container_name: php-fpm
    build:
      context: .
    dockerfile: docker/php.Dockerfile
    volumes:
      - ./:/var/www/symfony
      - ./logs/symfony:/var/www/symfony/log
    extra_hosts:
      - «db: 172.17.0.1»
  nginx:
    container_name: nginx
    build:
      context: .
    dockerfile: docker/nginx.Dockerfile
    ports:
      - "80:80"
    volumes:
      - ./:/var/www/symfony
      - ./logs/nginx:/var/log/nginx
    links:
      - php-fpm
```

Изменим ip адрес в файле .env

Изменим конфигурацию локальной БД, так чтобы она допускала подключение из контейнера, изменим файлы конфигурации

```
# - Connection Settings -

#listen_addresses = '*'          # what IP address(es) to listen on;
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for all
                                # (change requires restart)
port = 5432                      # (change requires restart)
max_connections = 100           # (change requires restart)
#superuser_reserved_connections = 3 # (change requires restart)
unix_socket_directories = '/var/run/postgresql' # comma-separated list of directories
                                # (change requires restart)
#unix_socket_group = ''         # (change requires restart)
#unix_socket_permissions = 0777 # begin with 0 to use octal notation
                                # (change requires restart)
#bonjour = off                  # advertise server via Bonjour
                                # (change requires restart)
#bonjour_name = ''              # defaults to the computer name
                                # (change requires restart)

# - TCP settings -
# see "man 7 tcp" for details

#tcp_keepalives_idle = 0         # TCP_KEEPIDL, in seconds;
                                # 0 selects the system default
#tcp_keepalives_interval = 0    # TCP_KEEPINTVL, in seconds;
                                # 0 selects the system default
#tcp_keepalives_count = 0       # TCP_KEEPCNT;
                                # 0 selects the system default
#tcp_user_timeout = 0           # TCP_USER_TIMEOUT, in milliseconds;
                                # 0 selects the system default
```

Рисунок 10 – Изменение файла postgresql.conf

```
GNU nano 4.8          pg_hba.conf          Изменён
# maintenance (custom daily cronjobs, replication, and similar tasks).
#
# Database administrative login by Unix domain socket
local    all             postgres                                peer

# TYPE      DATABASE      USER      ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local    all             all                                peer
# IPv4 local connections:
host     all             all             127.0.0.1/32            md5
host     all             all             0.0.0.0/0               md5
# IPv6 local connections:
host     all             all             ::1/128                 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
local    replication     all                                peer

^G Помощь      ^O Записать    ^M Поиск      ^K Вырезать   ^J Выворнять  ^C ТекПозиц
^X Выход      ^R ЧитФайл    ^\ Замена     ^U Paste Text ^T Словарь    ^_ К строке
```

Рисунок 11 – Изменение файла pg\_hba.conf

Далее подключимся к локальной БД

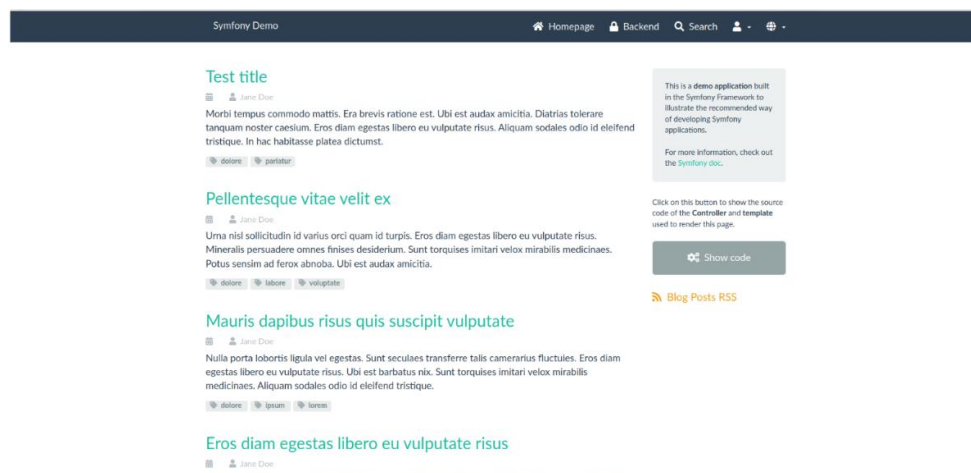


Рисунок 12 – Подключение к локальной БД

## Вывод

Во время выполнения лабораторной работы я изучила современные методы разработки ПО в динамических и распределенных средах на примере контейнеров Docker.

## Контрольные вопросы

1. Назовите отличия использования контейнеров по сравнению с виртуализацией.

- A. Меньшие накладные расходы на инфраструктуру
- B. Время старта приложений больше
- C. Невозможность запуска GNU/Linux- и Windows-приложений на одном хосте
- D. Обязательное использование гипервизора KVM

Ответ: A;

2. Назовите основные компоненты Docker.

- A. Гипервизор
- B. Контейнеры
- C. Образы виртуальных машин
- D. Реестры

Ответ: B;

3. Какие технологии используются для работы с контейнерами?

- A. Пространства имен (Linux Namespaces)
- B. Подключаемые модули аутентификации (PAM)
- C. Контрольные группы (cgroups)
- D. Аппаратная поддержка виртуализации

Ответ: C;

4. Найдите соответствие между компонентом и его описанием:

- контейнеры - доступные только для чтения шаблоны приложений.
- образы - изолированные при помощи технологий операционной системы пользовательские окружения, в которых выполняются приложения.
- реестры - сетевые хранилища образов (репозитории)

Ответ:

- 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.

#### 5. В чем отличие контейнеров от виртуализации?

Ответ: Виртуальная машина содержит как приложения, так и все, что нужно для его запуска (системные исполняемые файлы и библиотеки, и т.д.). Она также несет в себе весь аппаратный стек, включая виртуальные сетевые адаптеры, файловое хранилище и центральный процессор, и свою собственную полноценную гостевую операционную систему.

Ответ: Контейнер обеспечивает виртуализацию на уровне операционной системы с помощью абстрагирования «пользовательского пространства», в отличие от виртуальной машины, использующей аппаратную виртуализацию.

#### 6. Перечислите основные команды утилиты Docker с их кратким описанием.

Ответ:

- `docker ps` – отобразить все запущенные контейнеры;
- `docker ps -a` – отобразить остановленные контейнеры;
- `docker build` – сборка образа по настройкам в `Dockerfile`’е;
- `docker run ...` – запуск контейнера;
- `docker stop...` – остановка контейнера;
- `docker images` – отобразить образы в локальном репозитории;
- `docker exec -it...` - выполнить команду в определенном контейнере.

#### 7. Каким образом осуществляется поиск образов контейнеров?

Ответ: Docker проверяет в начале локальный репозиторий на наличие нужного образа, если образ не был найден, то docker проверяет удаленный репозиторий Docker Hub.

#### 8. Каким образом осуществляется запуск контейнера?

Ответ: Docker выполняет инициализацию и запуск ранее созданного по образу контейнера по имени.

#### 9. Что значит управлять состоянием контейнеров?

Ответ: Означает контролирование хода выполнения контейнера, и перевод его в любой момент времени в остановленный/запущенный режим и выполнять команды внутри контейнера.

10. Как изолировать контейнер?

Ответ: Для изоляции контейнера достаточно правильно сконфигурировать файлы Dockerfile и docker-compose.yaml (если есть).

11. Опишите последовательность создания новых образов, назначение Dockerfile?

Ответ: Для создания нового образа выбирается основа образа (любой подходящий пакет из репозитория Docker Hub), добавляются необходимые слои, выполняются нужные операции и разворачивается рабочее окружение внутри контейнера с необходимыми зависимостями. После чего происходит сборка образа.

12. Возможно ли работать с контейнерами Docker без одноименного движка?

Ответ: Да, можно, в среде другой виртуализации Kubernetes

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

Ответ: Kubernetes – это открытое программное обеспечение для автоматизации развёртывания, масштабирования контейнеризированных приложений и управления ими. Поддерживает основные технологии контейнеризации, включая Docker.

Основным объектами являются: узел, под, том.

Узел — это отдельная физическая или виртуальная машина, на которой развёрнуты и выполняются контейнеры приложений. Каждый узел в кластере содержит сервисы для запуска приложений в контейнерах (например, Docker), а также компоненты, предназначенные для централизованного управления узлом.

Под — базовая единица для управления и запуска приложений, один или несколько контейнеров, которым гарантирован запуск на одном узле,



обеспечивается разделение ресурсов, межпроцессное взаимодействие и предоставляется уникальный в пределах кластера IP-адрес.

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

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.

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

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.

Операторы — специализированный вид программного обеспечения Kubernetes, предназначенный для включения в кластер сервисов, сохраняющих своё состояние между выполнениями, таких как СУБД, системы мониторинга или кэширования.