

# Липецкий государственный технический университет

Кафедра прикладной математики

## Отчет по лабораторной работе № 5 «Контейнеризация» по курсу «Операционная система Linux»

Студент

\_\_\_\_\_

подпись, дата

Пустовалова И.П.

\_\_\_\_\_

фамилия, инициалы

Группа

Руководитель

Доцент, к. пед. наук  
ученая степень, ученое звание

\_\_\_\_\_

подпись, дата

Кургасов В.В.  
фамилия, инициалы

Липецк 2021 г.

# Содержание

<b>Цель работы</b>	<b>3</b>
<b>Задание кафедры</b>	<b>4</b>
<b>1. Ход работы</b>	<b>5</b>
1.1. Клонирование и запуск тестового проекта с помощью команды «git clone» . . . . .	5
1.2. Установка Docker и Docker-compose . . . . .	6
1.3. Создание БД . . . . .	8
1.4. Сборка контейнера . . . . .	10
<b>Выводы</b>	<b>16</b>

# Цель работы

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

## Задание кафедры

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. (Для этого: 1. Создать новую БД в postgres; 2. Заменить DATABASE\_URL в /.env на строку подключения к postgres; 3. 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) контейнеры с fpm и nginx должны его подхватывать. 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 постгреса 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

# 1. Ход работы

## 1.1. Клонирование и запуск тестового проекта с помощью команды «git clone»

```
rina@rina:~$ git clone https://github.com/symfony/demo
```

```
rina@rina:~$ git clone https://github.com/symfony/demo
Cloning into 'demo'...
remote: Enumerating objects: 10567, done.
remote: Counting objects: 100% (692/692), done.
remote: Compressing objects: 100% (422/422), done.
remote: Total 10567 (delta 358), reused 514 (delta 242), pack-reused 9875
Receiving objects: 100% (10567/10567), 18.92 MiB | 5.23 MiB/s, done.
Resolving deltas: 100% (6299/6299), done.
rina@rina:~$
```

Рисунок 1 – Результат клонирования

```
rina@rina:~/demo$ symfony serve
```

```
rina@rina:~$ symfony server:start -d

INFO A new version is available (4.28.1, currently running v4.26.11).
      Consider upgrading soon using: symfony self:update

[WARNING] run "symfony server:ca:install" first if you want to run the web server with TLS support,
or use "--no-tls" to avoid this warning

[OK] Web server listening
      The Web server is using PHP FPM 8.0.13
      http://127.0.0.1:8000

Stream the logs via symfony server:log
rina@rina:~/demo$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:69:c4:ee brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.102/24 brd 192.168.56.255 scope global dynamic enp0s3
        valid_lft 559sec preferred_lft 559sec
    inet6 fe80::a00:27ff:fe69:c4ee/64 scope link
        valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:92:2d:5d:e8 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
rina@rina:~/demo$ _
```

Рисунок 2 – Запуск симфони

## Добро пожаловать в **Symfony Demo** приложение

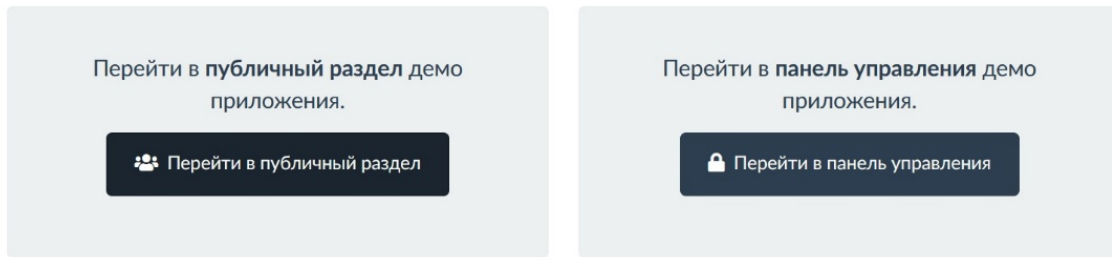


Рисунок 3 – Просмотр результата в браузере

### 1.2. Установка Docker и Docker-compose

```
rina@rina:~/demo$ sudo apt-get install docker-ce
```

```
rina@rina:~/demo$ sudo curl -L https://github.com/docker/
compose/releases/download/1.25.0-rc4/docker-compose-
'uname -s'-'uname -m' -o /usr/local/bin/docker-compose
```

```
rina@rina:~/demo$ sudo chmod +x /usr/local
/bin/docker-compose
```

```
rina@rina:~/demo$ sudo ln -s /usr/local/bin
/docker-compose /usr/bin/docker-compose
```

```
rina@rina:~/demo$ sudo docker info
```

```
Client:
Context:    default
Debug Mode: false
Plugins:
  app: Docker App (Docker Inc., v0.9.1-beta3)
  buildx: Docker Buildx (Docker Inc., v0.7.1-docker)
  scan: Docker Scan (Docker Inc., v0.12.0)

Server:
Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
Images: 0
Server Version: 20.10.12
Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
  userxattr: false
Logging Driver: json-file
Cgroup Driver: cgroupfs
Cgroup Version: 1
Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
Swarm: inactive
Runtimes: io.containerd.runtime.v1.linux runc io.containerd.runc.v2
Default Runtime: runc
Init Binary: docker-init
containerd version: 7b11cfaabd73bb80907dd23182b9347b4245eb5d
runc version: v1.0.2-0-g52b36a2
init version: de40ad0
Security Options:
  apparmor
  seccomp
   Profile: default
Kernel Version: 5.11.0-43-generic
Operating System: Ubuntu 20.04.3 LTS
OSType: linux
Architecture: x86_64
CPUs: 4
Total Memory: 5.685GiB
Name: evgen1067-VivoBook-15-ASUS-Laptop-X540UBR
ID: X32G:IMAW:T6MV:RNKD:TXVE:AITW:TY3S:YT4E:VNS4:NYEV:Z6AH:4VLG
Docker Root Dir: /var/lib/docker
Debug Mode: false
Registry: https://index.docker.io/v1/
Labels:
Experimental: false
Insecure Registries:
  127.0.0.0/8
Live Restore Enabled: false
```

Рисунок 4 – Результат установки

### 1.3. Создание БД

```
In Exception.php line 26:
SQLSTATE[08006] [7] FATAL: database "laba" does not exist

In Driver.php line 28:
SQLSTATE[08006] [7] FATAL: database "laba" does not exist

doctrine:schema:create [--em EM] [--dump-sql]

rina@rina:~/demo$ cd
rina@rina:~$ sudo -i -u postgres
postgres@rina:~$ psql
psql (12.9 (Ubuntu 12.9-0ubuntu0.20.04.1))
Type "help" for help.

postgres=# \q
postgres@rina:~$ createdb lab_5
postgres@rina:~$ psql
psql (12.9 (Ubuntu 12.9-0ubuntu0.20.04.1))
Type "help" for help.

postgres=# \l
          List of databases
  Name      | Owner   | Encoding | Collate | Ctype   | Access privileges
-----+-----+-----+-----+-----+-----
 lab_5      | postgres | UTF8     | ru_RU.UTF-8 | ru_RU.UTF-8 |
 postgres   | 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)

postgres=# _
```

Рисунок 5 – Создание БД

```
rina@rina:~/my_project/src$ php bin
/console doctrine:schema:create

rina@rina:~/my_project
/src$ php bin/console doctrine:fixtures:load
```



```
###> symfony/mailer ###  
#MAILER_URL=null://localhost  
  
rina@rina:~/demo$ bin/console doctrine:schema:load  
  
Command "doctrine:schema:load" is not defined.  
  
Did you mean one of these?  
doctrine:fixtures:load  
doctrine:schema:create  
doctrine:schema:drop  
doctrine:schema:update  
doctrine:schema:validate  
  
rina@rina:~/demo$ bin/console doctrine:schema:create  
  
!  
! [CAUTION] This operation should not be executed in a production environment!  
!  
  
Creating database schema...  
  
[OK] Database schema created successfully!  
  
rina@rina:~/demo$ bin/console doctrine:fixtures:load  
  
Careful, database "laba" will be purged. Do you want to continue? (yes/no) [no]:  
> yes  
  
  > purging database  
  > loading App\DataFixtures\AppFixtures  
rina@rina:~/demo$
```

Рисунок 6 – Результат команды

## 1.4. Сборка контейнера

Разобьем наш проект на папки «docker», «src».

Содержимое файла docker/docker-compose.yml:

```
GNU nano 4.2:
version: '3.8'

services:
  php-fpm:
    container_name: php-fpm
    build:
      context: ./php-fpm
    depends_on:
      - db
    environment:
      - APP_ENV=${APP_ENV}
      - APP_SECRET=${APP_SECRET}
      - DATABASE_URL=${DATABASE_URL}
    volumes:
      - ../../src:/var/www

  nginx:
    container_name: nginx
    build:
      context: ./nginx
    volumes:
      - ../../src:/var/www
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf
      - ./nginx/sites:/etc/nginx/sites-available
      - ./nginx/conf.d:/etc/nginx/conf.d
      - ./logs:/var/log
    depends_on:
      - php-fpm
    ports:
      - "80:80"
      - "443:443"

  db:
    container_name: db
    image: postgres:12
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
      POSTGRES_DB: dbtest
    ports:
      - "15432:5432"
    volumes:
      - ./pg-data:/var/lib/postgresql/data
```

Рисунок 7 – Содержимое файла docker/docker-compose.yml

Содержимое файла docker/nginx/Dockerfile:

```
FROM nginx:alpine

WORKDIR /var/www

CMD ["nginx"]

EXPOSE 80 443
```

Рисунок 8 – Содержимое файла docker/nginx/Dockerfile

Содержимое файла docker/php-fpm/Dockerfile:

```
FROM php:8.0-fpm

COPY wait-for-it.sh /usr/bin/wait-for-it

RUN chmod +x /usr/bin/wait-for-it

RUN apt-get update && \
    apt-get install -y --no-install-recommends libssl-dev zlib1g-dev curl git unzip netcat libxml2-dev libpq-dev libzip-dev && \
    pecl install apcu && \
    docker-php-ext-configure pgsql -with-pgsql=/usr/local/pgsql && \
    docker-php-ext-install -j$(nproc) zip opcache intl pdo_pgsql pgsql && \
    docker-php-ext-enable apcu pdo_pgsql sodium && \
    apt-get clean && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

COPY --from=composer /usr/bin/composer /usr/bin/composer

WORKDIR /var/www

CMD composer i -o ; wait-for-it db:5432; php-fpm

EXPOSE 9000
```

Рисунок 9 – Содержимое файла docker/php-fpm/Dockerfile

```
rina@rina:~/demo$
```

```
sudo docker -compose up -d --build
```

```
-->709db286ed54
Step 7/8 : CMD composer i -o ; wait-for-it db:5432; php-fpm
--> Using cache
--> 98bd55aa4062
Step 8/8 : EXPOSE 9000
--> Using cache
--> 71889290e580
Successfully built 71889290e580
Successfully tagged docker_php-fpm:latest
Building nginx
Sending build context to Docker daemon 7.168kB
Step 1/4 : FROM nginx:alpine
-->cc44224bfe20
Step 2/4 : WORKDIR /var/www
--> Using cache
--> c663063eb9d9
Step 3/4 : CMD ["nginx"]
--> Using cache
--> 3fd249c2a558
Step 4/4 : EXPOSE 80 443
--> Using cache
--> 5d6349aa1a0d
Successfully built 5d6349aa1a0d
Successfully tagged docker_nginx: latest
Creating db ... done
Creating php-fpm ...
done
Creating nginx
done
rina@rina:~/demo$ _
```

Рисунок 10 – Сборка образа

```
rina@rina:~/demo$  
sudo docker -compose up -d
```

```
db is up-to-date  
php-fpm is up-to-date  
nginx is up-to-date
```

Рисунок 11 – Инициализация БД


```
rina@rina:~/demo$  
psql postgresql://postgres:password@127.0.0.1:15432/dbtest
```

```
psql (12.9 (Ubuntu 12.9-0ubuntu0.20.04.1))  
Type "help" for help.  
  
dbtest=# \dt  
              List of relations  
 Schema |          Name          | Type  | Owner  
-----+-----+-----+-----  
 public | doctrine_migration_versions | table | postgres  
 public | symfony_demo_comment      | table | postgres  
 public | symfony_demo_post         | table | postgres  
 public | symfony_demo_post_tag     | table | postgres  
 public | symfony_demo_tag          | table | postgres  
 public | symfony_demo_user         | table | postgres  
(6 rows)  
  
dbtest=# \q
```

Рисунок 12 – Просмотр созданной БД

## 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 13 – Просмотр результата в браузере

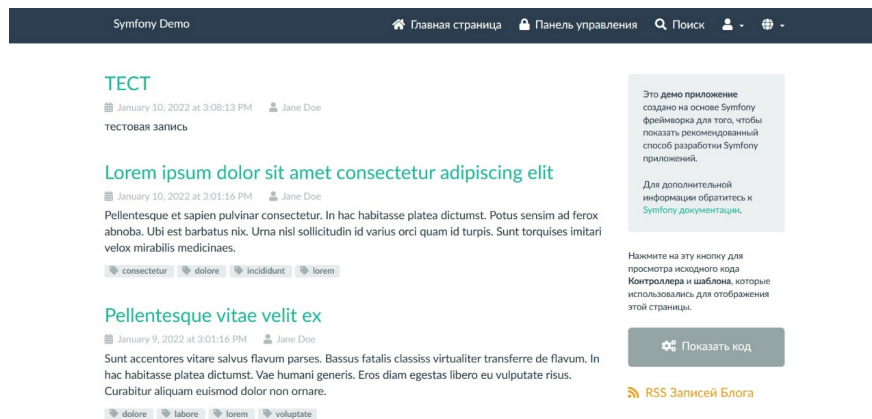


Рисунок 14 – Добавление записи

The screenshot shows the WordPress installation 'Welcome' screen. At the top is the WordPress logo. Below it is the heading 'Добро пожаловать'. The text says: 'Добро пожаловать в знаменитую пятиминутную установку WordPress! Просто заполните поля — и вперед, к использованию самой мощной и гибкой персональной платформы для публикаций в мире!'. Below this is a section titled 'Требуется информация'. The text says: 'Пожалуйста, укажите следующую информацию. Не переживайте, потом вы всегда сможете изменить эти настройки.' The form includes the following fields and options:

- Название сайта**: A text input field.
- Имя пользователя**: A text input field. Below it, a note says: 'Имя пользователя может содержать только латинские буквы, пробелы, подчеркивания, дефисы, точки и символ @.'
- Пароль**: A text input field with a 'Скрыть' button. Below it, a note says: 'Важно: Этот пароль понадобится вам для входа. Сохраните его в надёжном месте.'
- Подтвердите пароль**: A checkbox labeled 'Разрешить использование слабого пароля.'
- Ваш e-mail**: A text input field. Below it, a note says: 'Внимательно проверьте адрес электронной почты, перед тем как продолжить.'
- Видимость для поисковых систем**: A checkbox labeled 'Попросить поисковые системы не индексировать сайт'. Below it, a note says: 'Будет ли учитываться этот запрос — зависит от поисковых систем.'

At the bottom of the form is a button labeled 'Установить WordPress'.

Рисунок 15 – Настройка Wordpress

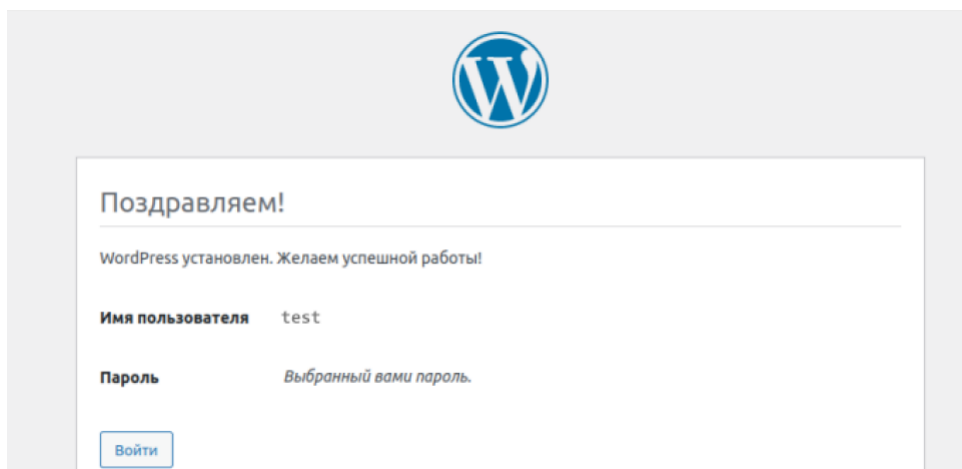


Рисунок 16 – Завершение настройки Wordpress

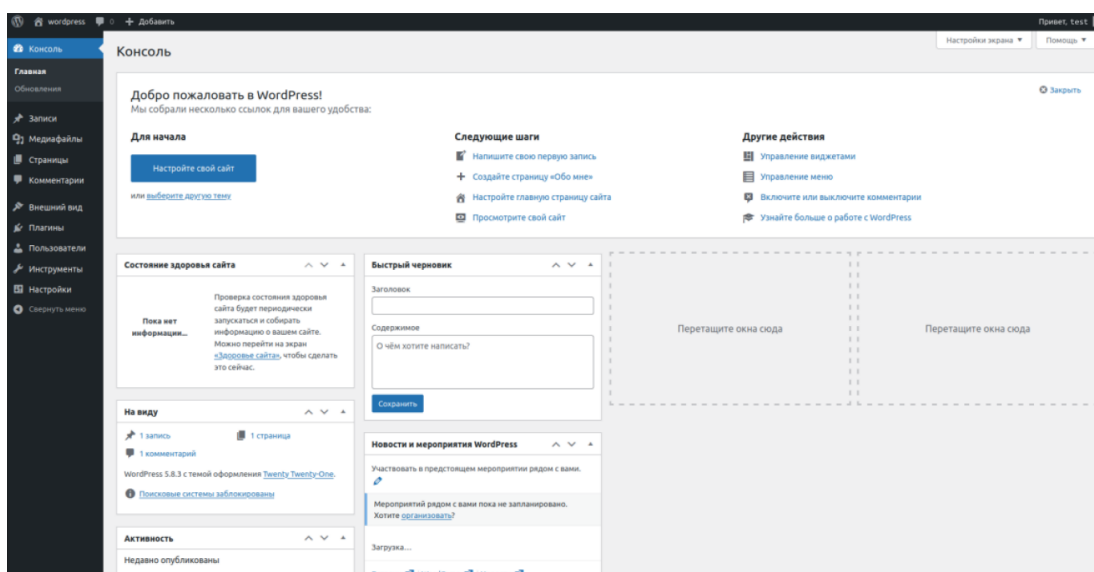


Рисунок 17 – Главная страница Wordpress

## Выводы

В ходе выполнения данной лабораторной работы мной были получены знания о контейнеризации, работе с git-ом, PostgreSQL и MySQL. Получен опыт создания таких файлов, как «docker-compose.yml», «Dockerfile», «.env».



1. Назовите отличия использования контейнеров по сравнению с виртуализацией.
  - Меньшие накладные расходы на инфраструктуру.
2. Назовите основные компоненты Docker.
  - Контейнеры.
3. Какие технологии используются для работы с контейнерами?
  - Контрольные группы (cgroups)
4. Найдите соответствие между компонентом и его описанием:
  - 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?
  - Главное отличие – способ работы. При виртуализации создается полностью отдельная операционная система. При контейнеризации используется ядро операционной системы той машины, на которой открывается контейнер.
  - Ещё одно значимое отличие – размер и скорость работы. Размер виртуальной машины может составлять несколько гигабайт. Также для загрузки операционной системы и запуска приложений, которые в них размещены, требуется много времени. Контейнеры более лёгкие — их размер измеряется в мегабайтах. По сравнению с виртуальными машинами, контейнеры могут запускаться намного быстрее.
6. Перечислите основные команды утилиты Docker с их кратким описанием.

- Контейнеры (`docker container my_command`):
  - `create` – Создать контейнер из изображения.
  - `start` – Запустите существующий контейнер.
  - `run` – Создайте новый контейнер и запустите его.
  - `ls` – Список работает контейнеры.
  - `inspect` – Смотрите много информации о контейнере.
  - `logs` – Печать журналов.
  - `stop` – Изящно прекратить запуск контейнера.
  - `kill` – внезапно остановить основной процесс в контейнере.
  - `rm` – Удалить остановленный контейнер.
- Изображения (`docker image my_command`):
  - `build` – Построить образ.
  - `push` – Нажмите на изображение в удаленном реестре.
  - `ls` – Список изображений.
  - `history` – Смотрите промежуточную информацию изображения.
  - `inspect` – Смотрите много информации об изображении, в том числе слоев.
  - `rm` – Удалить изображение.

7. Каким образом осуществляется поиск образов контейнеров?

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

8. Каким образом осуществляется запуск контейнера?

- Docker выполняет инициализацию и запуск ранее созданного по образу контейнера по его имени.

9. Что значит управлять состоянием контейнеров?

- Это значит иметь возможность взаимодействовать с контролирующим его процессом.

10. Как изолировать контейнер?

- Сконфигурировать необходимые для этого файлы «docker-compose.yml» и «Dockerfile».

11. Опишите последовательность создания новых образов, назначение Dockerfile?

- 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. Dockerfile – это простой текстовый файл с инструкциями по созданию образа Docker. Он содержит все команды, которые пользователь может вызвать в командной строке для создания образа.

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

- Да, возможно при использовании среды другой виртуализации.

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

- Назначение Kubernetes состоит в выстраивании эффективной системы распределения контейнеров по узлам кластера в зависимости от текущей нагрузки и имеющихся потребностей при работе сервисов. Kubernetes способен обслуживать сразу большое количество хостов, запускать на них многочисленные контейнеры Docker или Rocket, отслеживать их состояние, контролировать совместную работу и репликацию, проводить масштабирование и балансировку нагрузки.

- Основные объекты:
  - Kubectl Command Line Interface (kubectl.md): kubectl интерфейс командной строки для управления Kubernetes.
  - Volumes (volumes.md): Volume(раздел) это директория, возможно, с данными в ней, которая доступна в контейнере.
  - Labels (labels.md): Label'ы это пары ключ/значение которые прикрепляются к объектам, например pod'ам. Label'ы могут быть использованы для создания и выбора наборов объектов
  - Replication Controllers (replication-controller.md): replication controller гарантирует, что определенное количество «реплик» pod'ы будут запущены в любой момент времени.
  - Services (services.md): Сервис в Kubernetes это абстракция которая определяет логический объединённый набор pod и политику доступа к ним.
  - Pods (pods.md): Pod это группа контейнеров с общими разделами, запускаемых как единое целое.
  - Nodes (node.md): Нода это машина в кластере Kubernetes.