

Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра Автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №6

По дисциплине «ОС Linux»

Контейнеризация

Студент

Чаплыгин И.С.

Группа ПИ-18

Руководитель

Доцент

Кургасов В.В.

Липецк 2021г

Цель работы

Изучение современных методов разработки ПО в динамических и распределенных средах на примере контейнеров Docker.

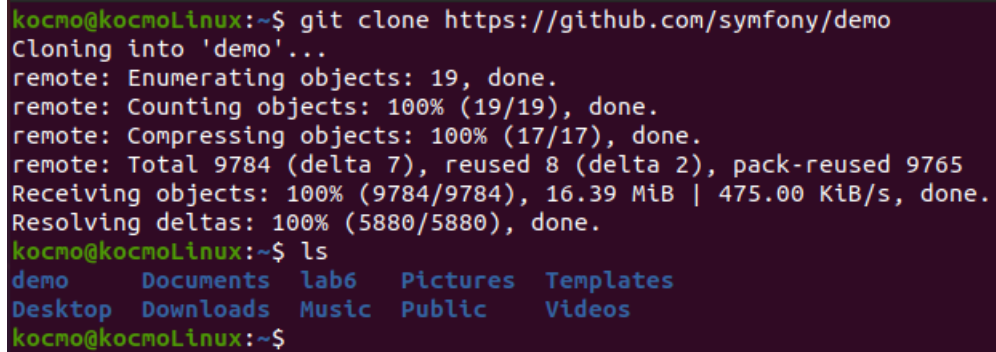
Задание кафедры

1. С помощью Docker Compose на своем компьютере поднять сборку nginx+php-fpm+postgres, продемонстрировать ее работоспособность, запустив внутри контейнера демо-проект на symfony. По умолчанию проект работает с sqlite-базой. Нужно заменить ее на postgres.
2. Заменить DATABASE_URL в .env на строку подключения к postgres.
3. Создать схему БД и заполнить ее данными из фикстур.
4. Создание образа с Wordpress

Выполнение работы

1. Выполнение работы с проектом demo

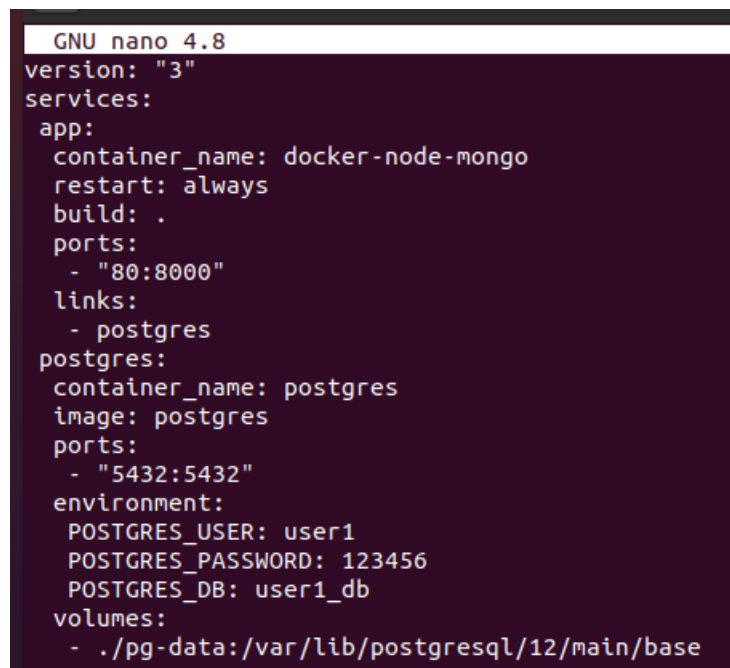
1) С помощью команды `git clone https://github.com/symfony/demo` клонируем проект в папку demo



```
kocmo@kocmoLinux:~$ git clone https://github.com/symfony/demo
Cloning into 'demo'...
remote: Enumerating objects: 19, done.
remote: Counting objects: 100% (19/19), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 9784 (delta 7), reused 8 (delta 2), pack-reused 9765
Receiving objects: 100% (9784/9784), 16.39 MiB | 475.00 KiB/s, done.
Resolving deltas: 100% (5880/5880), done.
kocmo@kocmoLinux:~$ ls
demo    Documents  lab6      Pictures  Templates
Desktop Downloads  Music     Public    Videos
kocmo@kocmoLinux:~$
```

Рисунок 1 – Скачивание проекта demo

2) В проекте demo создаем файл `docker-compose.yml` и `Dockerfile`



```
GNU nano 4.8
version: "3"
services:
  app:
    container_name: docker-node-mongo
    restart: always
    build: .
    ports:
      - "80:8000"
    links:
      - postgres
  postgres:
    container_name: postgres
    image: postgres
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: user1
      POSTGRES_PASSWORD: 123456
      POSTGRES_DB: user1_db
    volumes:
      - ./pg-data:/var/lib/postgresql/12/main/base
```

Рисунок 2 – Содержимое файла `docker-compose.yml`

```
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 3 – Содержимое файла Dockerfile

3) Изменяем database_url в файле .env для БД, которая будет расположена в будущем контейнере postgres.

```
###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference
# 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://user1:123456@postgres:5432/user1_db?serverVersion=11&charset=utf8"
#DATABASE_URL=sqlite://%kernel.project_dir%/data/database.sqlite
###< doctrine/doctrine-bundle ###
```

Рисунок 4 – Изменение файла .env

4) Введем команду “composer install” для скачивания и установки пакетов.(При первом использовании команды, на консоли будет выводиться информация о скачиваемых файлах)

```
kosmo@kosmoLinux:~/lab6/demo$ composer install
Installing dependencies from lock file (including require-dev)
Verifying lock file contents can be installed on current platform.
Nothing to install, update or remove
Generating autoload files
composer/package-versions-deprecated: Generating version class...
composer/package-versions-deprecated: ...done generating version class
85 packages you are using are looking for funding.
Use the 'composer fund' command to find out more!

Synchronizing package.json with PHP packages
Don't forget to run npm install --force or yarn install --force to refresh your Javascript dependencies.
Run composer recipes at any time to see the status of your Symfony recipes.

Executing script cache:clear [OK]
Executing script assets:install --symlink --relative public [OK]

kosmo@kosmoLinux:~/lab6/demo$
```

Рисунок 5 – Установка пакетов

5) Собираем контейнеры командой docker-compose build

```
kocmo@kocmoLinux:~/lab6/demo$ docker-compose build
postgres uses an image, skipping
Building app
Step 1/7 : FROM richarvey/nginx-php-fpm
--> 5c3ad1891297
Step 2/7 : WORKDIR /var>>/www/html/demo
--> Using cache
--> ff415564ddc0
Step 3/7 : COPY composer.json ./
--> Using cache
--> ebec70038759
Step 4/7 : RUN COMPOSER_MEMORY_LIMIT=-1 composer install
--> Using cache
--> a58544296cbe
Step 5/7 : COPY LibreOffice Writer .
--> Using cache
--> 023310c0c0ce
Step 6/7 : EXPOSE 8000
--> Running in b933bc12186d
Removing intermediate container b933bc12186d
--> 95e9d387317d
Step 7/7 : CMD ["php", "-S", "0.0.0.0:8000", "-t", "public/"]
--> Running in d61748582c58
Removing intermediate container d61748582c58
--> a27c13102892

Successfully built a27c13102892
Successfully tagged demo_app:latest
kocmo@kocmoLinux:~/lab6/demo$
```

Рисунок 6 – Сборка контейнеров

6) Запускаем контейнеры командой docker-compose up -d

```
kocmo@kocmoLinux:~/lab6/demo$ docker-compose up -d
Removing docker-node-mongo
postgres is up-to-date
Recreating df7e5e80d706_docker-node-mongo ... done
kocmo@kocmoLinux:~/lab6/demo$
```

Рисунок 7 – Запуск контейнеров

7) Проверим наличие пользователя и базы данных в контейнере postgres

```
Firefox Web Browser
kocmo@kocmoLinux:~/lab6/demo$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                                                                                               NAME
807f741e192b   demo_app      "docker-php-entrypoi..." 2 minutes ago  Up 2 minutes  80/tcp, 443/tcp, 9000/tcp, 0.0.0.0:80->8000/tcp          dock
er-node-mongo
fb7d6ae9db80   postgres      "docker-entrypoint.s..." 15 hours ago  Up 3 minutes  0.0.0.0:5432->5432/tcp                                   post
gres
2f059a353c58   mysql:5.7     "docker-entrypoint.s..." 46 hours ago  Up About an hour  3306/tcp, 33060/tcp                                       word
press_db_1
kocmo@kocmoLinux:~/lab6/demo$ docker exec -it postgres bash
root@fb7d6ae9db80:/# psql -h localhost -p 5432 -U user1 -d user1_db -W
Password:
psql (13.1 (Debian 13.1-1.pgdg100+1))
Type "help" for help.

user1_db=#
```

Рисунок 8 – Проверка наличия базы данных в контейнере

8) В каталоге demo создадим схему БД и заполним её командами:

php bin/console doctrine:schema:create

php bin/console doctrine:fixtures:load

После чего зайдём на сервер по адресу localhost:80 и проверим работоспособность



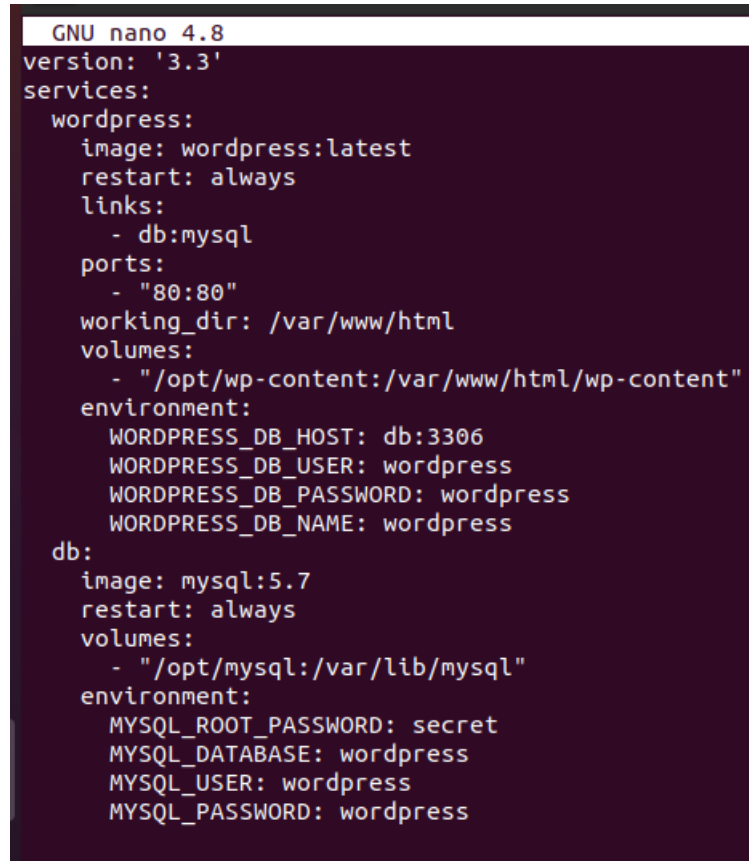
Рисунок 9 – Работоспособность сервера и базы данных.

После закрытия сервера и его повторного включения, загрузилась прежняя база данных с двумя личными записями.

2. Работа с WordPress

WordPress – это система управления контентом. Она позволяет создавать веб приложения для управления сайтами и публиковать контент без знаний программирования. WordPress использует PHP и базу данных MySQL.

1) Создание нового каталога и файла docker-compose.yml



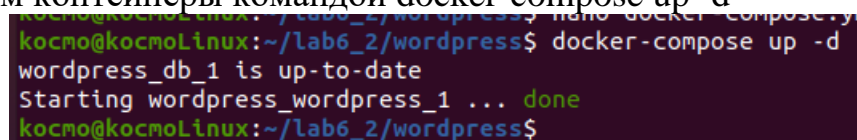
```
GNU nano 4.8
version: '3.3'
services:
  wordpress:
    image: wordpress:latest
    restart: always
    links:
      - db:mysql
    ports:
      - "80:80"
    working_dir: /var/www/html
    volumes:
      - "/opt/wp-content:/var/www/html/wp-content"
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
  db:
    image: mysql:5.7
    restart: always
    volumes:
      - "/opt/mysql:/var/lib/mysql"
    environment:
      MYSQL_ROOT_PASSWORD: secret
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
```

Рисунок 1 – Содержимое файла docker-compose.yml

2) Создадим каталоги для локального хранения файлов плагинов, контента и базы данных MySQL:

```
sudo mkdir /opt/mysql
sudo mkdir /opt/wp-content
sudo chmod 777 /opt/wp-content
```

3) Запустим контейнеры командой docker-compose up -d



```
kocmo@kocmoLinux: ~/lab6_2/wordpress$ nano docker-compose.yml
kocmo@kocmoLinux:~/lab6_2/wordpress$ docker-compose up -d
wordpress_db_1 is up-to-date
Starting wordpress_wordpress_1 ... done
kocmo@kocmoLinux:~/lab6_2/wordpress$
```

Рисунок 2 – Запуск контейнеров

4) Переходим в браузере по адресу localhost:80 и выполняем настройку WordPress.

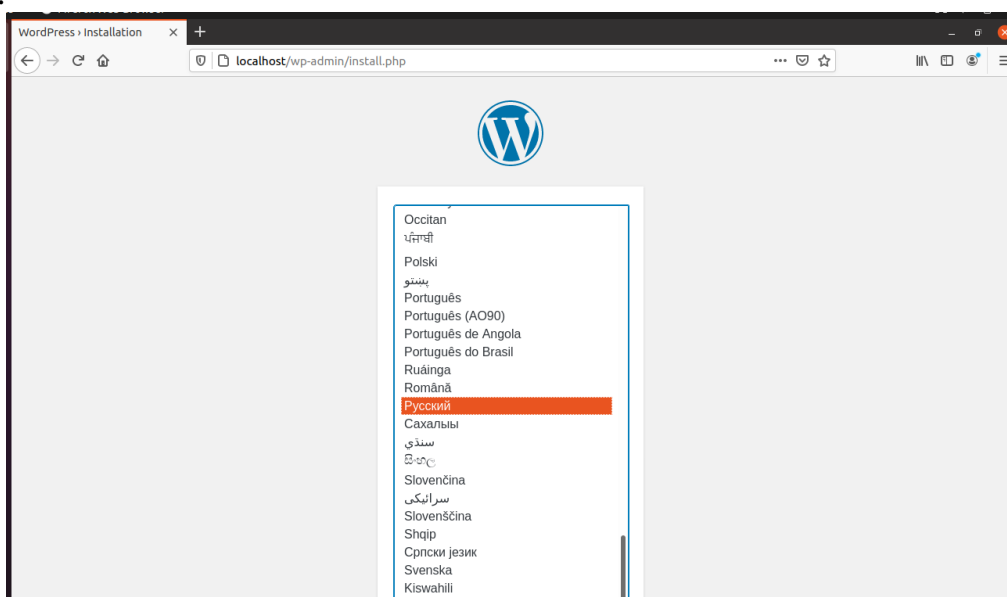


Рисунок 3 – Настройка языка

A screenshot of the WordPress installation 'Welcome' screen in Russian. The WordPress logo is at the top. The main heading is 'Добро пожаловать' (Welcome). Below it, a message 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!' (Welcome to the famous five-minute WordPress installation! Just fill in the fields — and forward, to using the most powerful and flexible personal platform for publishing in the world!). The section 'Требуется информация' (Information required) asks the user to provide the following information, noting that it can be changed later. The form includes: 'Название сайта' (Site name) with the value 'TestWordPress'; 'Имя пользователя' (Username) with the value 'Космо', with a note that it can only contain Latin letters, spaces, underscores, dashes, and the @ symbol; 'Пароль' (Password) with a masked input, a 'Show' button, and a 'Надёжный' (Strong) indicator, with a note that it's needed for login; 'Ваш e-mail' (Your email) with the value 'chaplogin141299@gmail.com', with a note to check the email address carefully; and a checkbox for 'Видимость для поисковых систем' (Search engine visibility), which is checked, with the text 'Попросить поисковые системы не индексировать сайт' (Ask search engines not to index the site) and a note that it depends on the search engines. At the bottom is a button 'Установить WordPress' (Install WordPress).

Рисунок 4 – Ввод необходимой информации и регистрация пользователя

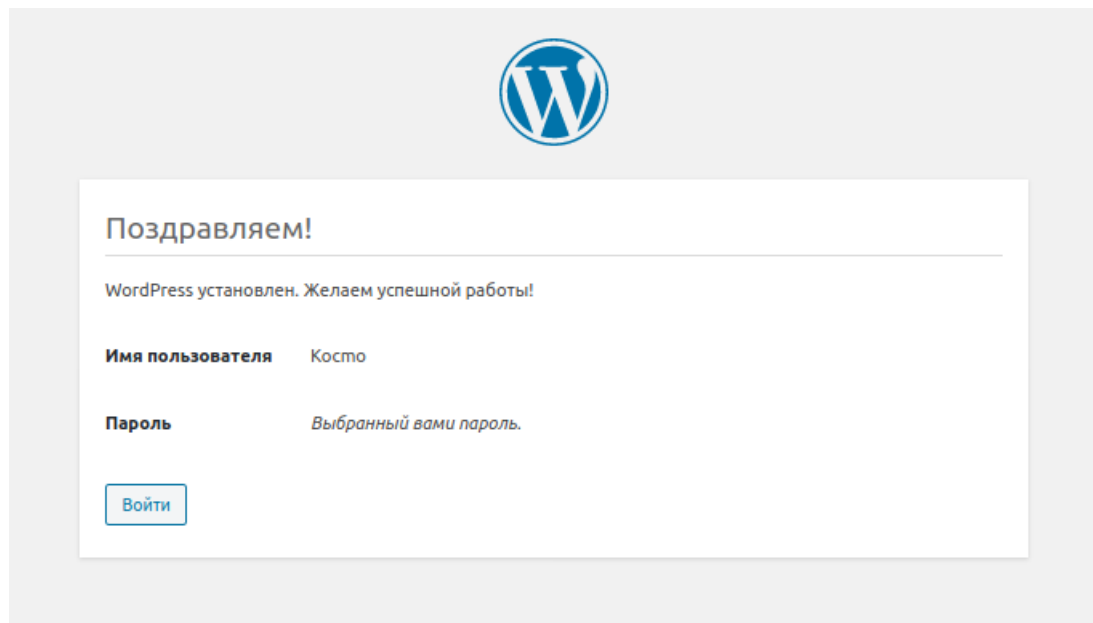


Рисунок 5 – Успешная установка WordPress

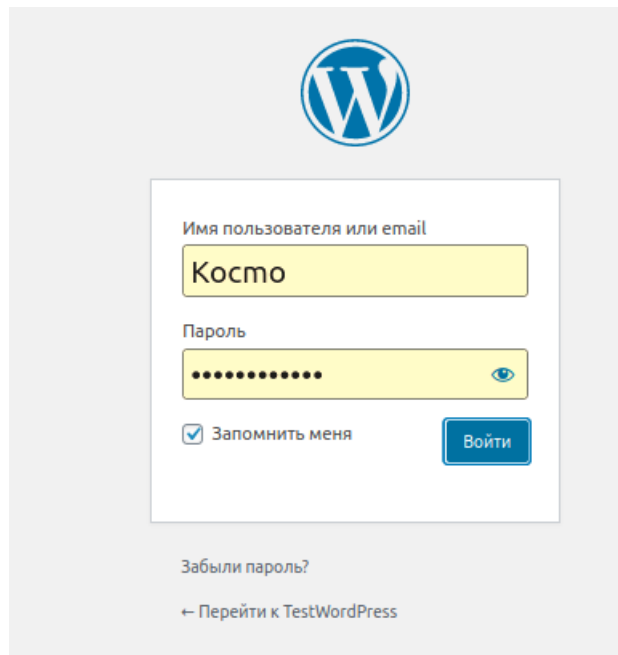


Рисунок 6 - Авторизация

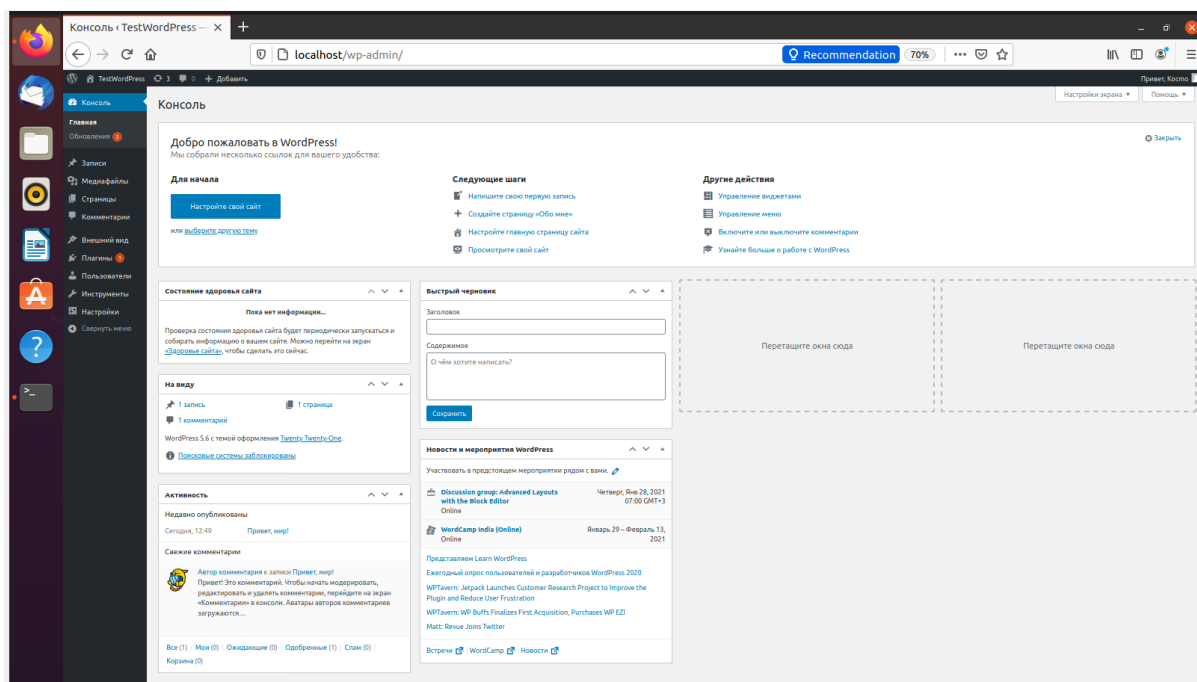


Рисунок 7 – Вход окно настроек сайта

Вывод

В ходе выполнения лабораторной работы были изучены методы разработки ПО в динамических и распределённых средах на примере контейнеров Docker. Были изучены основы работы с системой управления контентом WordPress.

Вопросы для самопроверки:

1. Назовите отличия использования контейнеров по сравнению с виртуализацией:

- А) меньшие накладные расходы на инфраструктуру;
- В) время старта приложения меньше.

2. Назовите основные компоненты Docker.

- С) Образы виртуальных машин
- Д) Реестры

3. Какие технологии используются для работы с контейнерами?

- А) Пространства имен (Linux Namespaces)
- С) Контрольные группы (cgroups)

4. Найдите соответствие между компонентом и его описанием:

Контейнеры – изолированные при помощи технологий операционной системы пользовательские окружения, в которых выполняются приложения.

Образы – доступные только для чтения шаблоны приложений .

Реестры – сетевые хранилища образов.

5. Для виртуализации требуется гипервизор, а также полная копия ОС, запускаемое приложение и все библиотеки поддержки. В случае контейнеров ядро-хоста совместно используется работающими контейнерами (это означает, что контейнеры всегда ограничиваются использованием того же ядра, которое функционирует на хосте), а так же процессы внутри контейнеров равнозначны собственными процессами ОС хоста и не влекут за собой дополнительные накладные расходы, связанные с выполнением гипервизора.

6. Основные команды утилиты docker:

`docker images` (можно посмотреть список образов);

`docker ps` (вывести список контейнеров);

`docker tag` (изменение имени образу);

`docker rmi` (удаление образа);

`docker rm` (удаление контейнера).

7. Командой `docker search <image name>` можно запустить поиск Docker образов на сервере регистра с терминала.

8. Командой `docker run <image name>` осуществляет запуск выбранного образа в новом контейнере.

9. Контейнер можно быть в 3 состояниях: а) контейнер работает б) контейнер создан, но в настоящий момент не выполняется в) контейнер завершил исполнение

10. Для изоляции контейнера достаточно правильно сконфигурировать файлы `Dockerfile` и `docker-compose.yml`. По умолчанию контейнеры запускаются от `root` прав, поэтому стоит быть осторожным с монтированием томов на хост машину.

11. Последовательность создания нового образа:

Создается файл `Dockerfile` в корне проекта. Внутри описывается процесс создания образа;

Выполняется сборка образа командой `docker build`;

Выполняется публикация образа в Registry командой `docker push`.

`Dockerfile` — содержит инструкции по созданию образа.

12. Без `docker-engine` невозможно работать (запускать, изменять и т.п.) с контейнерами `docker`.

13. Оркестрация — обеспечение совместной работы всех элементов системы. Запуск контейнеров на соответствующих хостах и установление соединений между ними. Организационная система также может включать поддержку масштабирования, автоматического восстановления после критических сбоев и инструменты изменения балансировки нагрузки на узлы. Kubernetes – это высокоуровневое решение оркестровки, в которое по умолчанию встроены функции восстановления после критических сбоев и масштабирования и которое может работать поверх других решений кластеризации. Основные объекты в Kubernetes: pods, flat networking space, labels, services