МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет автоматизации и информатики
Кафедра автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №5

по курсу «ОС Linux»

Студент ПМ-21-2 _____ Шишкина А. Л.

(подпись, дата)

Руководитель _____ Кургасов В.В.

(подпись, дата)

Липецк 2023

Цель работы

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

Задание

I часть

С помощью Docker Compose на своем компьютере поднять сборку nginx+phpfpm+postgres, продемонстрировать ее работоспособность, запустив внутри контейнера демо-проект на symfony (Исходники взять отсюда https://github.com/symfony/demo /ссылка на github/). 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://demosymfony.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

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.

II часть

Шаг №1. Установка Nginx Для начала необходимо установить один лишь Nginx. Что требует создания compose-файла включая директиву ports, иначе порт будет доступен только внутри контейнера и nginx через браузер уже будет недоступен.
Шаг №2. Передача в контейнер html-файлов. В этом нам поможет volumes, которая говорит, что происходит монтирование локальной папки в контейнер по указанному адресу. При монтировании папка по указанному адресу внутри контейнера заменяется папкой с локального компьютера. Необходимо создать папку html на одном уровне с docker-compose.yml и добавить в нее файл index.html с произвольным текстом «Ваш текст», после чего пересоздадим контейнер (docker-compose up -d).
Шаг 3. Web-разработка. Создать папку proxy и в ней сборку dockercompose.yml для обращения по домену и пробросу такого домена на основной контейнер. И сборку nginx, php, mysql и phpmyadmin с использованием proxy сети.
Шаг 4. Имеется работающий Web-сервер. Создайте образ с одним из движков (WordPress, Joomla). Папка для хранения внешних данных с курсами должна быть Вами определена

Ход работы:

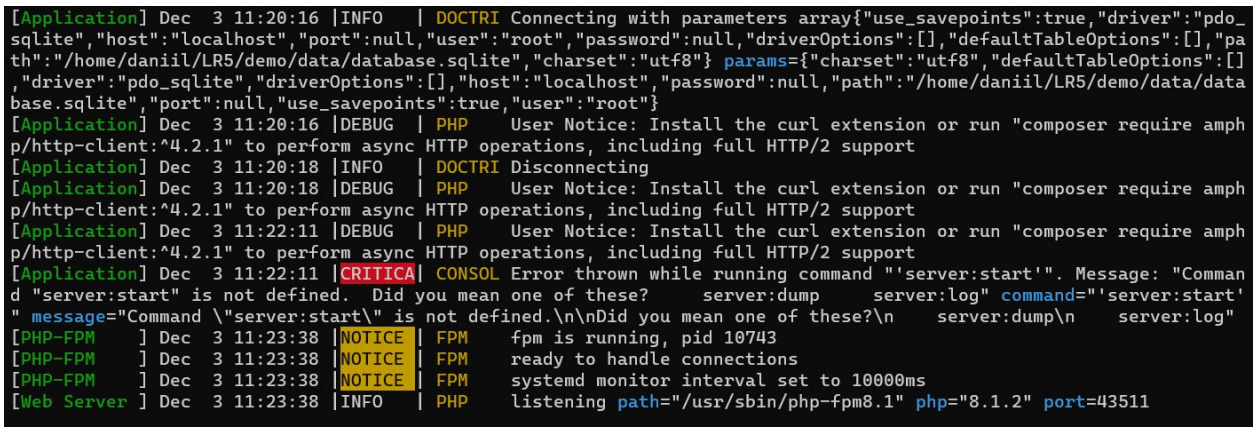
Клонируем проект с помощью git clone, переходим в папку demo и устанавливаем все зависимости:

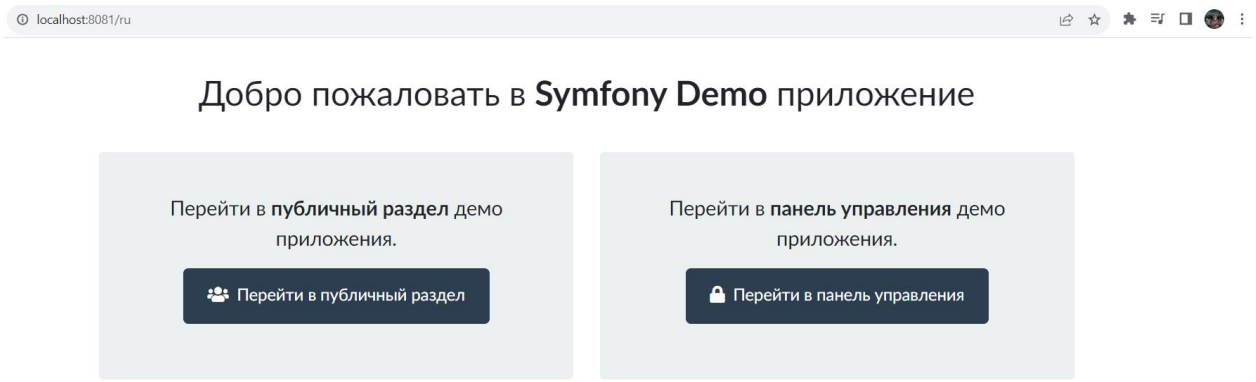
```
Installing dependencies from lock file (including require-dev)
Verifying lock file contents can be installed on current platform.
Package operations: 116 installs, 0 updates, 0 removals
  - Downloading symfony/flex (v2.4.1)
  - Downloading symfony/runtime (v6.4.0)
  - Downloading phpstan/phpstan (1.10.46)
  - Downloading phpstan/extension-installer (1.3.1)
  - Downloading symfony/deprecation-contracts (v3.4.0)
  - Downloading symfony/routing (v6.4.0)
  - Downloading symfony/polyfill-mbstring (v1.28.0)
  - Downloading symfony/polyfill-ctype (v1.28.0)
  - Downloading symfony/polyfill-php83 (v1.28.0)
  - Downloading symfony/http-foundation (v6.4.0)
  - Downloading psr/event-dispatcher (1.0.0)
  - Downloading symfony/event-dispatcher-contracts (v3.4.0)
  - Downloading symfony/event-dispatcher (v6.4.0)
  - Downloading symfony/var-dumper (v6.4.0)
  - Downloading psr/log (3.0.0)
  - Downloading symfony/error-handler (v6.4.0)
  - Downloading symfony/http-kernel (v6.4.0)
```

Запускаем сервер:

symfony server:start

```
 [WARNING] run "symfony server:ca:install" first if you want to run the web server with TLS support, or use "--p12
 " or "--no-tls" to avoid this warning

Following Web Server log file (/home/daniil/.symfony5/log/0a21b154a7a0f8b4cf007c8b1b47d3be8f7241a1.log)
Following PHP-FPM log file (/home/daniil/.symfony5/log/0a21b154a7a0f8b4cf007c8b1b47d3be8f7241a1/53fb8ec204547646acb3
461995e4da5a54cc7575.log)
WARNING the current dir requires PHP 8.1.0 (composer.json from current dir: /home/daniil/LR5/demo/composer.json), bu
t this version is not available: fallback to 8.1

 [WARNING] The local web server is optimized for local development and MUST never be used in a production setup.


 [OK] Web server listening
      The Web server is using PHP FPM 8.1.2
      http://127.0.0.1:8000


[Web Server ] Dec  3 11:23:37 |DEBUG  | PHP    Reloading PHP versions
[Web Server ] Dec  3 11:23:38 |WARN   | PHP    the current dir requires PHP 8.1.0 (composer.json from current dir: /
home/daniil/LR5/demo/composer.json), but this version is not available: fallback to 8.1
[Web Server ] Dec  3 11:23:38 |DEBUG  | PHP    Using PHP version 8.1.2 (from composer.json from current dir: /home/d
aniil/LR5/demo/composer.json)
[Application] Dec  3 11:20:16 |INFO   | DOCTRI Connecting with parameters array{"use_savepoints":true,"driver":"pdo_
sqlite","host":"localhost","port":null,"user":"root","password":null,"driverOptions":[],"defaultTableOptions":[],"pa
th":"/home/daniil/LR5/demo/data/database.sqlite","charset":"utf8"} params={"charset":"utf8","defaultTableOptions":[]
,"driver":"pdo_sqlite","driverOptions":[],"host":"localhost","password":null,"path":"/home/daniil/LR5/demo/data/data
base.sqlite","port":null,"use_savepoints":true,"user":"root"}
[Application] Dec  3 11:20:16 |DEBUG  | PHP    User Notice: Install the curl extension or run "composer require amph
p/http-client:^4.2.1" to perform async HTTP operations, including full HTTP/2 support
[Application] Dec  3 11:20:18 |INFO   | DOCTRI Disconnecting
[Application] Dec  3 11:20:18 |DEBUG  | PHP    User Notice: Install the curl extension or run "composer require amph
p/http-client:^4.2.1" to perform async HTTP operations, including full HTTP/2 support
[Application] Dec  3 11:22:11 |DEBUG  | PHP    User Notice: Install the curl extension or run "composer require amph
```





Создаем базу данных и настраиваем с помощью команд

php bin/console doctrine:schema:create

php bin/console doctrine:fixtures:load

```
 Careful, database "main" will be purged. Do you want to continue? (yes/no)
[no]:
 > yes
```

Заполняем файлы:

Dockerfile:

```
FROM php:8.2-fpm


RUN apt-get update && apt-get install -y zlib1g-dev g++ git libicu-dev libpq-dev zip libzip-dev \
    && docker-php-ext-install intl opcache pdo pdo_mysql pdo_pgsql \
    && pecl install apcu \
    && docker-php-ext-enable apcu \
    && docker-php-ext-configure zip \
    && docker-php-ext-install zip


# Установка Composer
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer


# Установка Symfony CLI
RUN curl -sS https://get.symfony.com/cli/installer | bash
RUN mv /root/.symfony5/bin/symfony /usr/local/bin/


# Создание директории "logs" и установка прав доступа
RUN mkdir -p /var/www/project/var/log \
    && chown -R www-data:www-data /var/www/project/var/log


# Устанавливаем рабочую директорию
WORKDIR /var/www/project
```

docker-compose.yml

```
version: "3"

services:
  php82-service:
    build:
      context: .
      dockerfile: ./Dockerfile
    container_name: php82-container
    ports:
      - "9000:9000"
    volumes:
      - .:/var/www/project

  pgsql-service:
    image: postgres:14
    container_name: pgsql-container
    ports:
      - "5432:5432"
    volumes:
      - /var/lib/postgresql/data
    environment:
      - POSTGRES_USER=admin
      - POSTGRES_PASSWORD=0000

  nginx-service:
    image: nginx
   container_name: nginx-container
```

```
    ports:
      - "8091:80"
    volumes:
      - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
      - .:/var/www/project
    depends_on:
      - php82-service
```

nginx.conf

```
server {
    listen 80;
    index index.php;
    server_name localhost;
    root /var/www/project/public;

    location / {
        try_files $uri /index.php$is_args$args;
    }

    location ~ ^/index\.php(/|$) {
        fastcgi_pass php82-service:9000;
        fastcgi_split_path_info ^(.+\.php)(/.*)$;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param DOCUMENT_ROOT $document_root;
        fastcgi_buffer_size 128k;
        fastcgi_buffers 4 256k;
        fastcgi_busy_buffers_size 256k;
        internal;
```

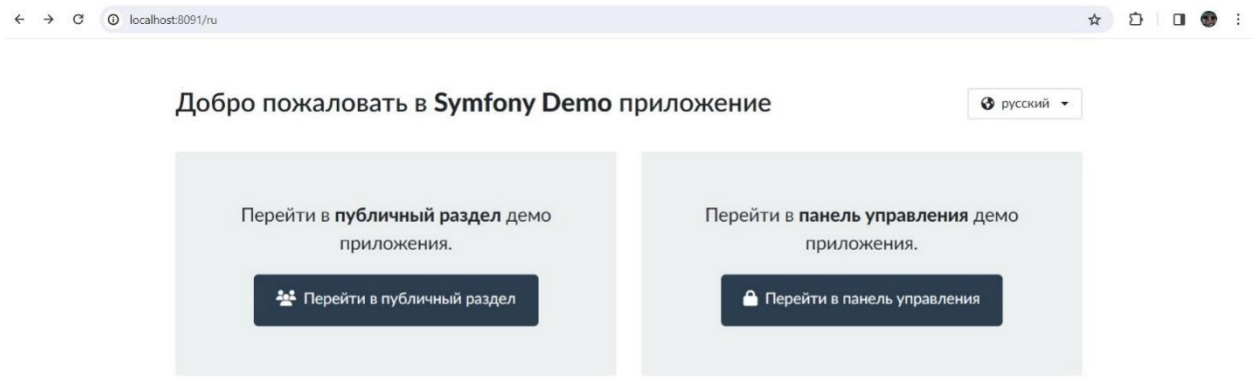
```
    }

    location ~ \.php$ {
        return 404;
    }
    error_log /var/log/nginx/project_error.log;
    access_log /var/log/nginx/project_access.log;
}
```

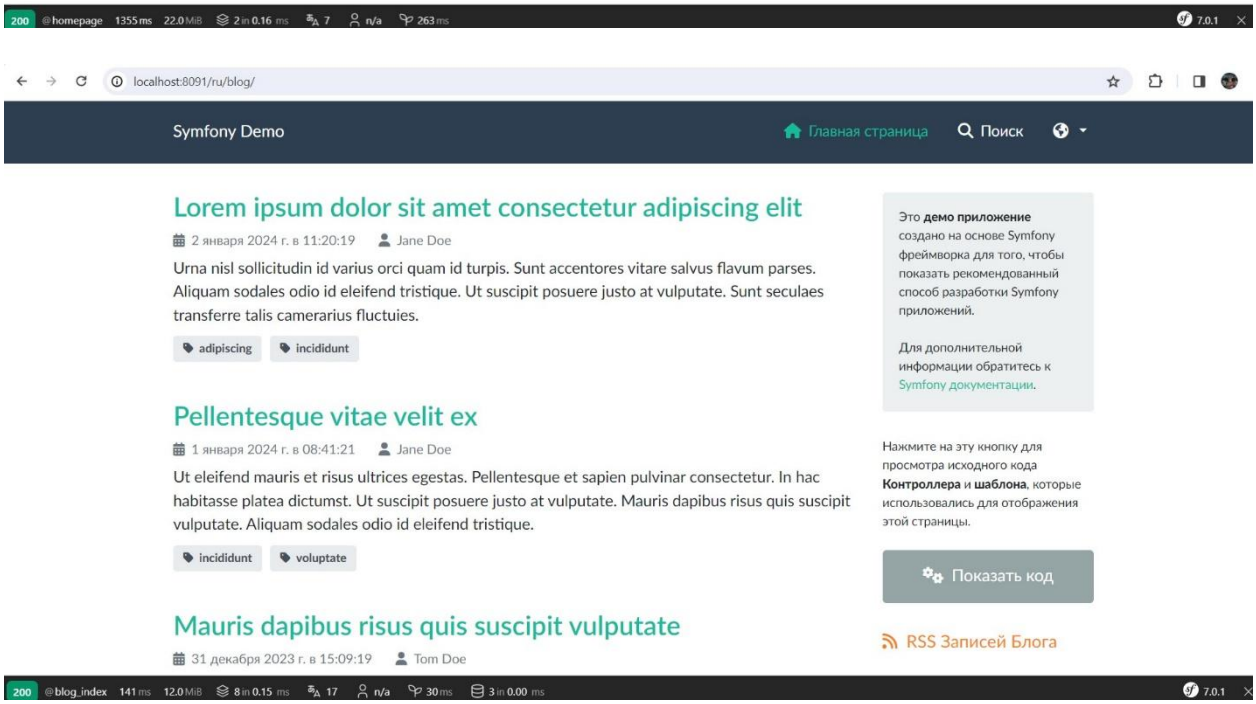
docker-compose build используем для создания Docker-образов:

```
postgres uses an image, skipping
Building app
Step 1/6 : FROM php:8.2-fpm
 ---> 593fe9503690
Step 2/6 : RUN apt-get update && apt-get install -y zlib1g-dev g++ git libicu-dev libpq-dev zip libzip-dev     && docker
-php-ext-install intl opcache pdo pdo_mysql pdo_pgsql     && pecl install apcu     && docker-php-ext-enable apcu     &&
docker-php-ext-configure zip     && docker-php-ext-install zip
 ---> Using cache
 ---> ff1aa051dd12
Step 3/6 : RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer
 ---> Using cache
 ---> 203bf8497004
Step 4/6 : RUN curl -sS https://get.symfony.com/cli/installer | bash
 ---> Using cache
 ---> d3bb6893314d
Step 5/6 : RUN mv /root/.symfony5/bin/symfony /usr/local/bin/
 ---> Running in d6c13c12429c
Removing intermediate container d6c13c12429c
 ---> 186a0bf2165c
Step 6/6 : WORKDIR /var/www/project
 ---> Running in 7403637ba4ca
Removing intermediate container 7403637ba4ca
 ---> ae9050aac725
Successfully built ae9050aac725
Successfully tagged demo_app:latest
```

Запускаем контейнер:

```
WARNING: Found orphan containers (node-mongo, demo_composer_1, docker-node-mongo, postgres) for this project. If you rem
oved or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it u
p.
php82-container is up-to-date
Starting pgsql-container ... done
Starting nginx-container ... done
Attaching to php82-container, pgsql-container, nginx-container
pgsql-container  |
pgsql-container  | PostgreSQL Database directory appears to contain a database; Skipping initialization
pgsql-container  |
nginx-container  | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
nginx-container  | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
nginx-container  | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
nginx-container  | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
nginx-container  | 10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf differs from the packaged versi
on
nginx-container  | /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
php82-container  | [02-Jan-2024 13:49:10] NOTICE: fpm is running, pid 1
php82-container  | [02-Jan-2024 13:49:10] NOTICE: ready to handle connections
php82-container  | 172.19.0.4 -  02/Jan/2024:13:49:17 +0000 "GET /index.php" 500
php82-container  | NOTICE: PHP message: 2024-01-02T13:49:17+00:00 [critical] Uncaught Exception: Unable to write in the
"logs" directory (/var/www/project/var/log).
php82-container  | [02-Jan-2024 13:52:29] NOTICE: Finishing ...
php82-container  | [02-Jan-2024 13:52:29] NOTICE: exiting, bye-bye!
php82-container  | [02-Jan-2024 13:52:39] NOTICE: fpm is running, pid 1
php82-container  | [02-Jan-2024 13:52:39] NOTICE: ready to handle connections
nginx-container  | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
nginx-container  | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
nginx-container  | /docker-entrypoint.sh: Configuration complete; ready for start up
pgsql-container  | 2024-01-02 13:52:43.086 UTC [1] LOG:  starting PostgreSQL 14.10 (Debian 14.10-1.pgdg120+1) on x86_64-
pc-linux-gnu, compiled by gcc (Debian 12.2.0-14) 12.2.0, 64-bit
pgsql-container  | 2024-01-02 13:52:43.087 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
pgsql-container  | 2024-01-02 13:52:43.087 UTC [1] LOG:  listening on IPv6 address "::", port 5432
nginx-container  | 2024/01/02 13:52:43 [notice] 1#1: using the "epoll" event method
```





2 часть

Шаг 1-2

Создаем файлы и заполняем:

docker-compose.yml

```
version: '3'

services:

  nginx:

    image: nginx:latest
```

```
    ports:
      - "8052:80"
    volumes:
      - ../nginx/nginx.conf:/etc/nginx/nginx.conf:ro
      - ../html:/usr/share/nginx/html
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Моя веб-страница</title>
</head>
<body>
    <h1>Welcome to nginx</h1>



</body>
</html>
```

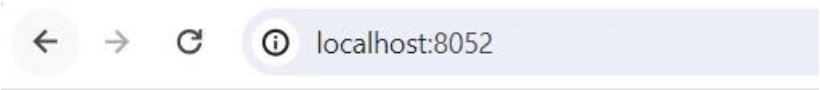
nginx.conf

```
events {
    worker_connections 1024;
}

http {
    server {
        listen 80;
```

```
        location / {
            root   /usr/share/nginx/html;
            index  index.html index.htm;
        }
    }
}
```

После чего запускаем:

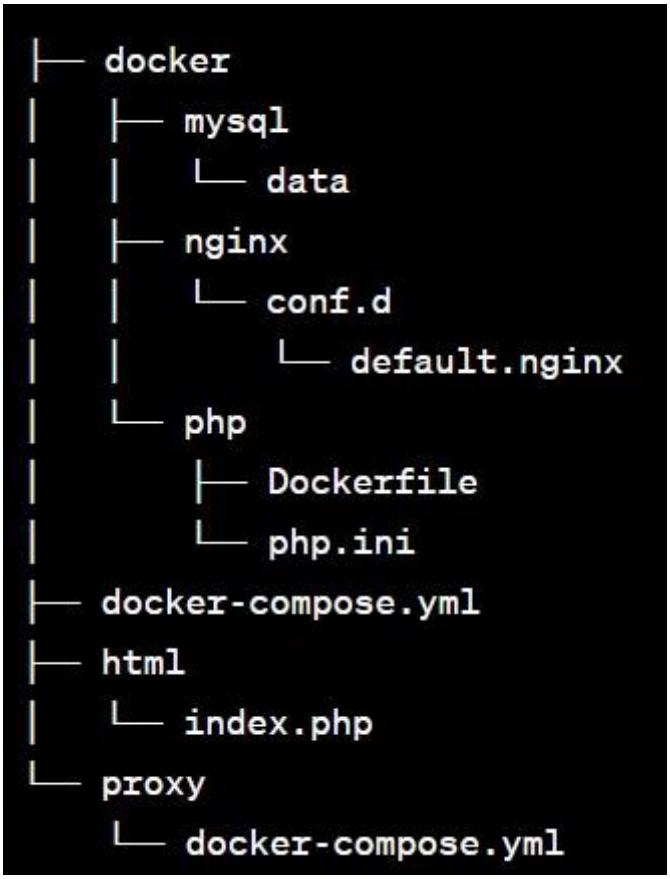
docker-compose up -d



Welcome to nginx

Шаг 3

Создаем структуру проекта:



Заполняем файлы:

docker-compose.yml

```
version: '3.0'
services:
  nginx:
    image: nginx
    environment:
      VIRTUAL_HOST: site.local
    depends_on:
      - php
    volumes:
      - ./docker/nginx/conf.d/default.nginx:/etc/nginx/conf.d/default.conf
      - ./html:/var/www/html/
    ports:
      - "8057:80"  # Прямое привязывание порта 8052 на хосте к порту 80 в контейнере
    networks:
      - frontend
      - backend


  php:
    build:
      context: ./docker/php
    volumes:
      - ./docker/php/php.ini:/usr/local/etc/php/php.ini
      - ./html:/var/www/html/
    networks:
      - backend
```

```
  mysql:
    image: mysql:5.7
    volumes:
      - ./docker/mysql/data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: root
    networks:
      - backend

  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    environment:
      VIRTUAL_HOST: phpmyadmin.local
      PMA_HOST: mysql
      PMA_USER: root
      PMA_PASSWORD: root
    networks:
      - frontend
      - backend

networks:
  frontend:
    external:
      name: proxy_proxy
  backend:
```

index.php

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My Dockerized PHP App</title>
</head>
<body>

<?php

$link = mysqli_connect('mysql', 'root', 'root');

if (!$link) {
    die('Ошибка соединения: ' . mysqli_error());
}

echo '<h1>Успешно соединились с базой данных!</h1>';

mysqli_close($link);

?>

</body>
</html>
```

proxy/docker-compose.yml

```
version: '3.0'
services:
  proxy:
```

```
    image: jwilder/nginx-proxy
    ports:
      - "80:80"
    volumes:
      - /var/run/docker.sock:/tmp/docker.sock:ro
    networks:
      - proxy

networks:
  proxy:
    driver: bridge
```

default.nginx

```
server {
    listen 80;
    server_name_in_redirect off;
    access_log /var/log/nginx/host.access.log main;
    root /var/www/html/;

    location / {
        try_files $uri /index.php$is_args$args;
    }

    location ~ \.php$ {
        try_files $uri =404;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        fastcgi_pass php:9000;
        fastcgi_index index.php;
        include fastcgi_params;
```

```
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }

    location ~ /\.ht {
        deny all;
    }
}
```

Dockerfile

```
FROM php:8.2-fpm

RUN apt-get update && apt-get install -y \
    libzip-dev \
    zip \
    && docker-php-ext-install zip mysqli

COPY --from=composer:latest /usr/bin/composer /usr/bin/composer
WORKDIR /var/www/html
```

После чего запускаем:

```
Building php
Step 1/4 : FROM php:8.2-fpm
 ---> 593fe9503690
Step 2/4 : RUN apt-get update && apt-get install -y     libzip-dev     zip     && docker-php-ext-install zip mysqli
 ---> Running in 0b498584d848
Get:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
Get:2 http://deb.debian.org/debian bookworm-updates InRelease [52.1 kB]
Get:3 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB]
Get:4 http://deb.debian.org/debian bookworm/main amd64 Packages [8787 kB]
Get:5 http://deb.debian.org/debian bookworm-updates/main amd64 Packages [12.7 kB]
Get:6 http://deb.debian.org/debian-security bookworm-security/main amd64 Packages [134 kB]
Fetched 9185 kB in 3s (3651 kB/s)
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  libzip4 unzip zlib1g-dev
The following NEW packages will be installed:
  libzip-dev libzip4 unzip zip zlib1g-dev
0 upgraded, 5 newly installed, 0 to remove and 1 not upgraded.
Need to get 1531 kB of archives.
After this operation, 2842 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian bookworm/main amd64 libzip4 amd64 1.7.3-1+b1 [55.5 kB]
Get:2 http://deb.debian.org/debian bookworm/main amd64 zlib1g-dev amd64 1:1.2.13.dfsg-1 [916 kB]
Get:3 http://deb.debian.org/debian bookworm/main amd64 libzip-dev amd64 1.7.3-1+b1 [163 kB]
Get:4 http://deb.debian.org/debian bookworm/main amd64 unzip amd64 6.0-28 [166 kB]
Get:5 http://deb.debian.org/debian bookworm/main amd64 zip amd64 3.0-13 [230 kB]
```

Успешно соединились с базой данных!

Шаг 4

Изменяем **docker-compose.yml:**

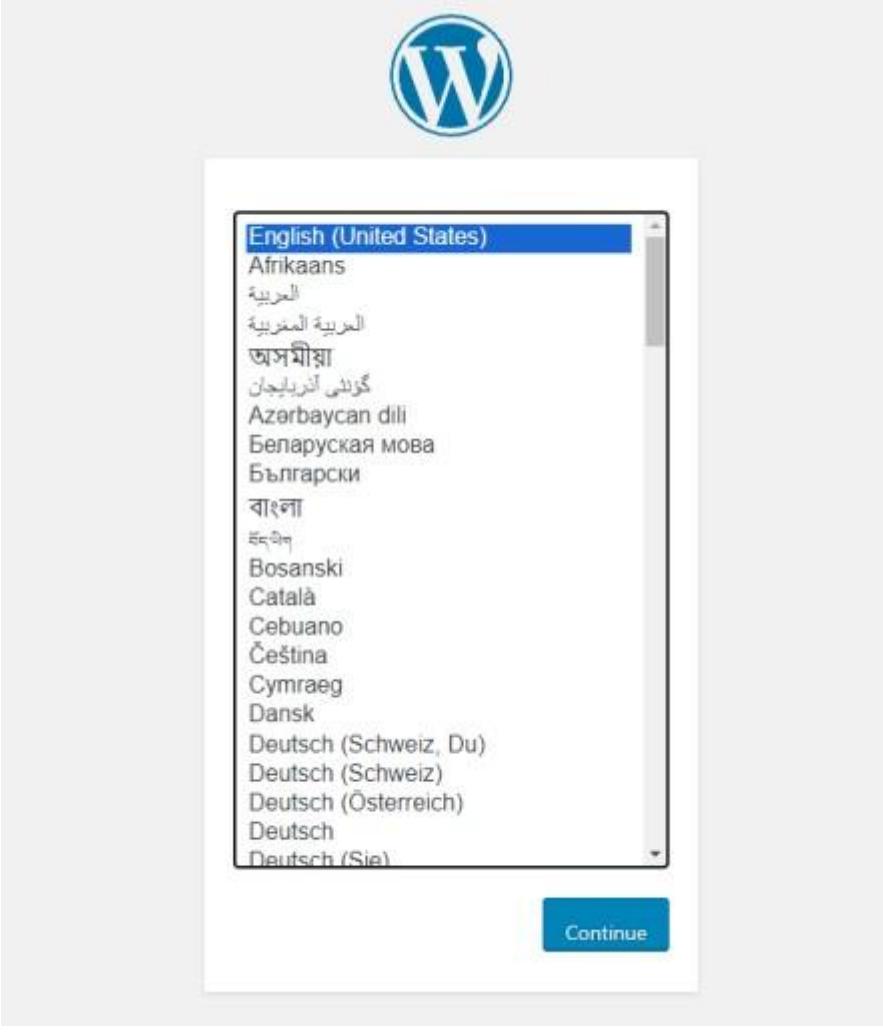
```
version: '3.0'


services:
  nginx:
    image: nginx
    environment:
      VIRTUAL_HOST: site.local
    depends_on:
      - php
    volumes:
      - ./docker/nginx/conf.d/default.nginx:/etc/nginx/conf.d/default.conf
      - ./wordpress:/var/www/html/
    ports:
```

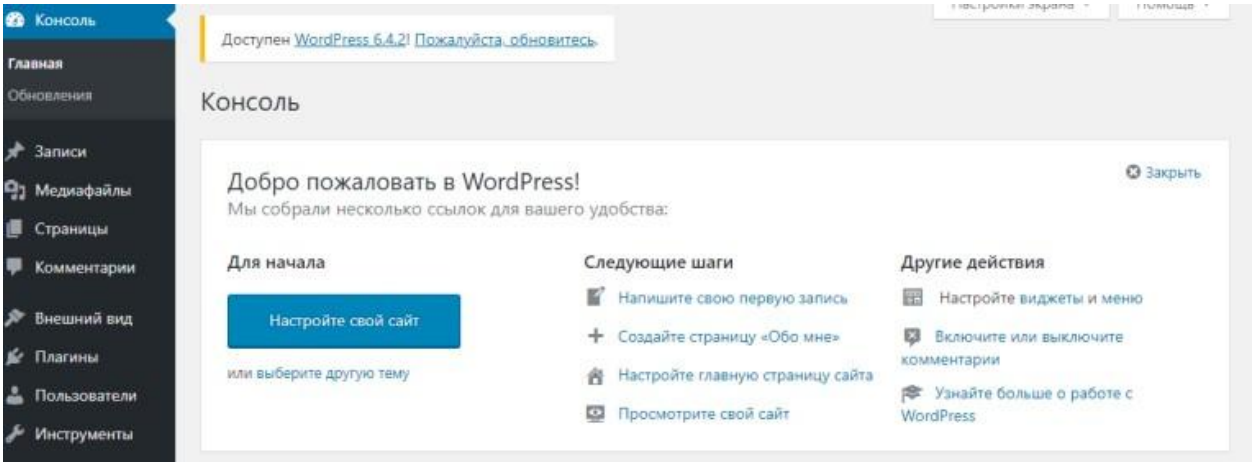
```
    - "8057:80"  # Прямое привязывание порта 8052 на хосте к порту 80 в контейнере
    networks:
      - frontend
      - backend

  php:
    build:
      context: ./docker/php
    volumes:
      - ./docker/php/php.ini:/usr/local/etc/php/php.ini
      - ./wordpress:/var/www/html/
    networks:
      - backend

  mysql:
    image: mysql:5.7
    volumes:
      - ./docker/mysql/data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: root
    networks:
      - backend

  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    environment:
      VIRTUAL_HOST: phpmyadmin.local
      PMA_HOST: mysql
```

```
      PMA_USER: root
      PMA_PASSWORD: root
    networks:
      - frontend
      - backend

  wordpress:
    depends_on:
      - mysql
    image: wordpress:5.1.1-fpm-alpine
    container_name: wordpress
    restart: unless-stopped
    env_file: .env
    environment:
      - WORDPRESS_DB_HOST=mysql:3306
      - WORDPRESS_DB_USER=root
      - WORDPRESS_DB_PASSWORD=root
      - WORDPRESS_DB_NAME=wordpress
    volumes:
      - ./wordpress:/var/www/html/
    networks:
      - frontend
      - backend

networks:
  frontend:
    external:
      name: proxy_proxy
  backend:
```

Запускаем контейнер:

```
Creating network "lr52_backend" with the default driver
Creating lr52_php_1        ... done
Creating lr52_phpmyadmin_1 ... done
Creating lr52_mysql_1      ... done
Creating lr52_nginx_1      ... done
Creating wordpress         ... done
```





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

2. Назовите основные компоненты Docker. A. Гипервизор B. Контейнеры C. Образы виртуальных машин D. Реестры

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

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 (репозитории) - сетевые хранилища образов

5. В чем отличие контейнеров от виртуализации?

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

7. Каким образом осуществляется поиск образов контейнеров?

8. Каким образом осуществляется запуск контейнера?

9. Что значит управлять состоянием контейнеров?

10. Как изолировать контейнер?

11. Опишите последовательность создания новых образов, назначение Dockerfile?

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

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

1. Ответы:
A. Меньшие накладные расходы на инфраструктуру

- Верно. Контейнеры обладают меньшими накладными расходами, так как они используют общий ядро операционной системы и не требуют полной виртуализации.

B. Время старта приложений больше

- Неверно. Время старта контейнеров обычно меньше, чем у виртуальных машин, благодаря более легковесному подходу.

C. Невозможность запуска GNU/Linux- и Windows-приложений на одном хосте

- Неверно. Контейнеры обеспечивают изолированное выполнение приложений разных типов, включая GNU/Linux и Windows, на одном хосте.

D. Обязательное использование гипервизора KVM

- Неверно. Контейнеры не требуют гипервизора, они работают непосредственно на уровне операционной системы.

2. Ответы: B. Контейнеры D. Реестры
3. Ответы: A. Пространства имен (Linux Namespaces) C. Контрольные группы (cgroups)
4. Соответствие:
   - контейнеры - изолированные при помощи технологий операционной системы пользовательские окружения, в которых выполняются приложения.
   - образы - доступные только для чтения шаблоны приложений.
   - реестры (репозитории) - сетевые хранилища образов.
5. Различие между контейнерами и виртуализацией:
   - Виртуализация использует гипервизор для запуска отдельных виртуальных машин с собственной операционной системой, в то время как контейнеры разделяют общее ядро ОС и изолируют приложения.
6. Основные команды Docker:
   - **docker run**: Запуск контейнера.
   - **docker build**: Создание образа из Dockerfile.
   - **docker images**: Просмотр списка доступных образов.
   - **docker ps**: Просмотр активных контейнеров.
   - **docker exec**: Запуск команды внутри работающего контейнера.
   - **docker stop** и **docker start**: Остановка и запуск контейнера.
7. Поиск образов контейнеров осуществляется с помощью команды:
   - **docker search <имя_образа>**.
8. Запуск контейнера осуществляется с использованием команды:
   - **docker run <опции> <имя_образа>**.

9. Управление состоянием контейнеров означает контроль и изменение их текущего состояния, включая запуск, остановку и масштабирование.
10. Изоляция контейнера включает в себя использование пространств имен и контрольных групп для обеспечения изолированного выполнения и ограничения ресурсов.
11. Последовательность создания новых образов с использованием Dockerfile:

- Создание Dockerfile с инструкциями для сборки образа.
- Запуск команды **docker build -t <имя_образа> <путь_к_Dockerfile>** для создания образа.

12. Да, возможно. Другие инструменты, такие как Podman, также могут работать с контейнерами без использования Docker Engine.
13. Kubernetes - система оркестрации контейнеров для автоматизации развертывания, масштабирования и управления контейнеризированными приложениями. Основные объекты Kubernetes включают Pod, Deployment, Service и ConfigMap, среди других.