

Липецкий государственный технический университет

Кафедра прикладной математики

Отчет по лабораторной работе № 5 «Контейнерезация» по курсу «ОС Linux»

Студент

подпись, дата

Гришагин Е.Е.

фамилия, инициалы

Группа

ПМ-19-2

Руководитель

ученая степень, ученое звание

подпись, дата

Кургасов В.В.

фамилия, инициалы

Липецк 2021 г.

Содержание

Задание кафедры	3
1. Ход работы	4
1.1. Установка необходимого ПО	4
1.1..1 Установка Docker	4
1.1..2 Установка Docker-Compose	5
1.1..3 Установка PHP	6
1.1..4 Установка Composer	6
1.1..5 Установка Composer	7
1.2. Настройка проекта на сервере	7
1.3. Настройка конфигураций Docker-Compose для запуска мультиконтейнерного приложения	9
1.4. Проект на Wordpress	17
2. Контрольные вопросы	20

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. Создать образ с одним из движков (WordPress, Joomla).

1. Ход работы

1.1. Установка необходимого ПО

Для запуска проекта нужно установить:

1. Docker – для работы с контейнерами
2. Docker Compose – для запуска приложения, состоящего из нескольких контейнеров
3. PHP и его необходимые библиотеки (CLI, FPM и другие)
4. Composer
5. Symfony

1.1.1 Установка Docker

Произведём установку через репозиторий Докера.

Установим пакеты, чтобы разрешить apt использовать репозитории через HTTPS-протокол

```
sudo apt-get update
sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

Добавим официальный GPG-ключ

```
curl -fsSL
https://download.docker.com/linux/ubuntu/gpg |
sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

Добавим официальный репозиторий Докера в sources.list, чтобы установить его через apt

```
echo \  
  "deb [arch=$(dpkg --print-architecture)  
signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]  
https://download.docker.com/linux/ubuntu \  
  $(lsb_release -cs) stable" |  
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Устанавливаем Docker через apt

```
sudo apt-get update  
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

1.1..2 Установка Docker-Compose

Есть несколько вариантов установки, но мы установим через pip. Установим pip через apt.

```
sudo apt install pip
```

Далее установим через pip Docker-Compsoe

```
sudo pip install docker-compose
```

1.1..3 Установка PHP

Добавим репозиторий с PHP8

```
sudo apt install software-properties-common
sudo add-apt-repository ppa:ondrej/php
```

Далее установим сам PHP и его библиотеки

```
sudo apt install php8.0
sudo apt-get install php8.0-sqlite
sudo apt-get install php8.0-xml
sudo apt-get install php8.0-mbstring
sudo apt-get install php8.0-pgsql
sudo apt install postgresql postgresql-contrib
```

1.1..4 Установка Composer

Скачаем установщик с официального сайта

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');
```

Установим Composer, сравнив с хешем

```
php -r
"if (hash_file('sha384', 'composer-setup.php') ===
'906a84df04cea2aa72f40b5f787e49f22d4c2f19492ac
310e8cba5b96ac8b64115ac
402c8cd292b8a03482574915d1a8')
{ echo 'Installer verified'; }
else
```

```
{ echo 'Installer corrupt';  
unlink('composer-setup.php'); }  
echo PHP_EOL;"  
php composer-setup.php  
php -r "unlink('composer-setup.php');"
```

Переместим установленный файл в `/usr/local/bin`, чтобы обращаться к нему в любой директории

```
sudo mv composer.phar /usr/local/bin/composer
```

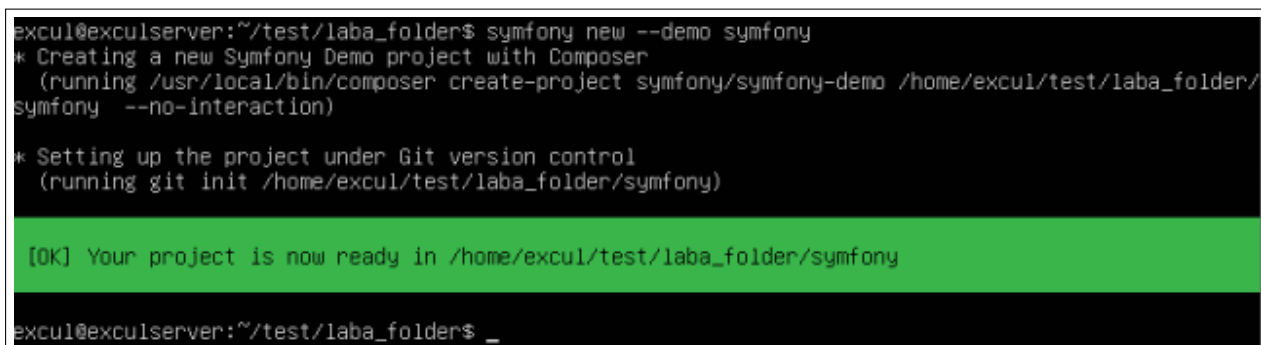
1.1..5 Установка Composer

Проделаем тоже самое, что и с Composer

```
wget https://get.symfony.com/cli/installer -O - | bash  
sudo mv /home/user/.symfony/bin/symfony /usr/local/bin/symfony
```

1.2. Настройка проекта на сервере

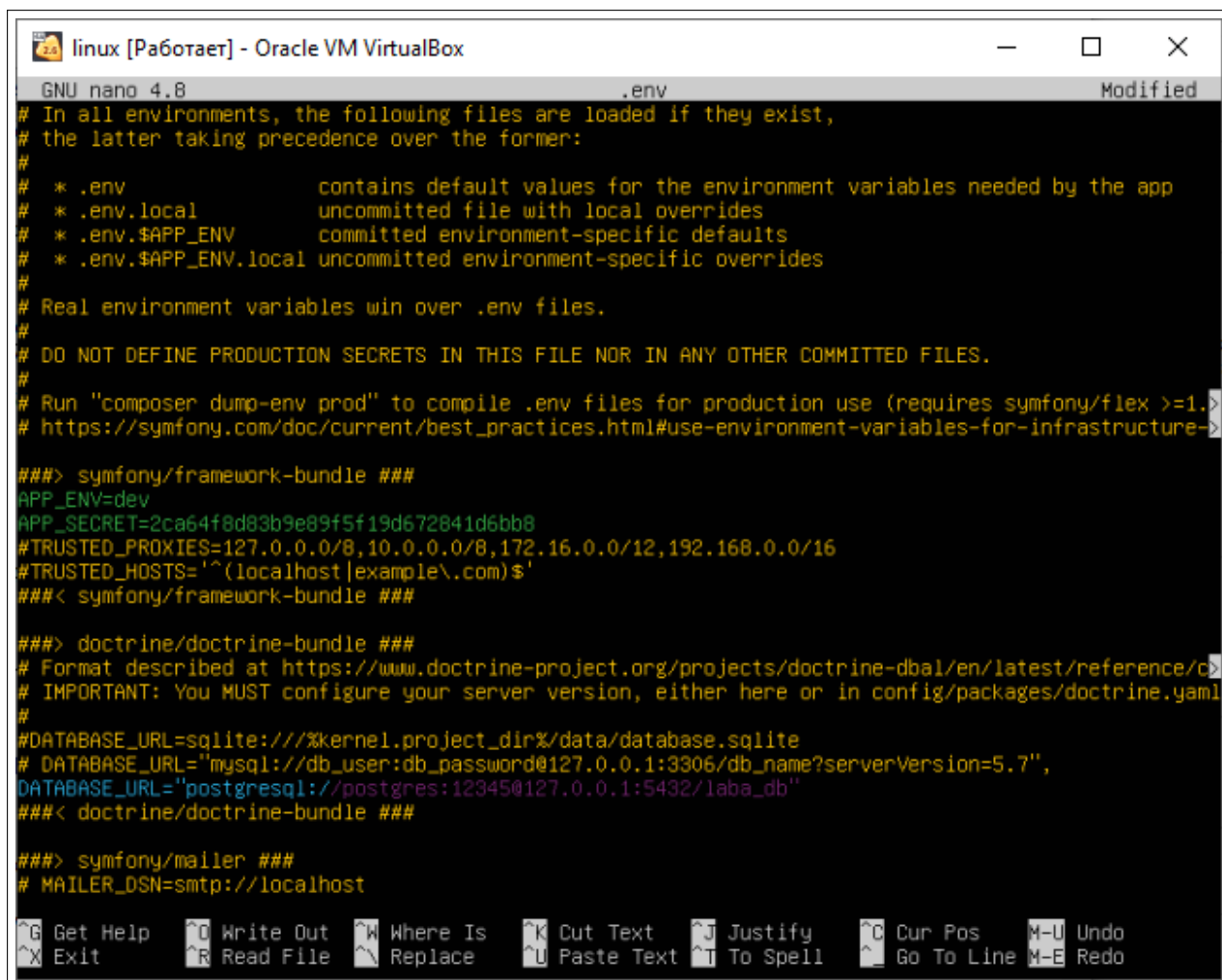
Инициализируем демо проект



```
excul@exculserver:~/test/laba_folder$ symfony new --demo symfony  
* Creating a new Symfony Demo project with Composer  
  (running /usr/local/bin/composer create-project symfony/symfony-demo /home/excul/test/laba_folder/  
symfony --no-interaction)  
  
* Setting up the project under Git version control  
  (running git init /home/excul/test/laba_folder/symfony)  
  
[OK] Your project is now ready in /home/excul/test/laba_folder/symfony  
  
excul@exculserver:~/test/laba_folder$ _
```

Рисунок 1 - Инициализация проекта

Далее меняем конфигурацию в `.env` для базы данных.



```
linux [Работает] - Oracle VM VirtualBox
GNU nano 4.8 .env Modified
# 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.$APP_ENV     committed environment-specific defaults
# * .env.$APP_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 >=1.0)
# https://symfony.com/doc/current/best_practices.html#use-environment-variables-for-infrastructure->

###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=2ca64f8d83b9e89f5f19d672841d6bb8
#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 ###

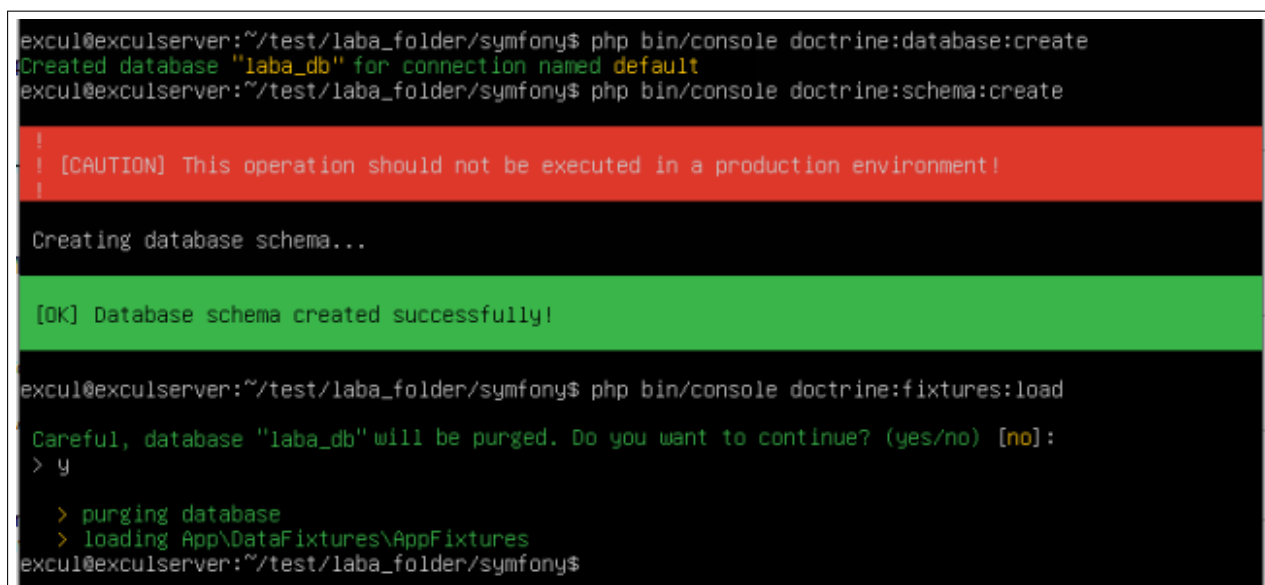
###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/c
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yam
#
#DATABASE_URL=sqlite:///kernel.project_dir%/data/database.sqlite
# DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name?serverVersion=5.7",
DATABASE_URL="postgresql://postgres:12345@127.0.0.1:5432/laba_db"
###< doctrine/doctrine-bundle ###

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

G Get Help  O Write Out  W Where Is  K Cut Text  J Justify  C Cur Pos  M-U Undo
X Exit      R Read File   R Replace  U Paste Text T To Spell  _ Go To Line M-E Redo
```

Рисунок 2 - /symfony/.env

Создадим и заполним базу данных.



```
excul@exculserver:~/test/laba_folder/symfony$ php bin/console doctrine:database:create
Created database "laba_db" for connection named default
excul@exculserver:~/test/laba_folder/symfony$ php bin/console doctrine:schema:create

! [CAUTION] This operation should not be executed in a production environment!

Creating database schema...

[OK] Database schema created successfully!

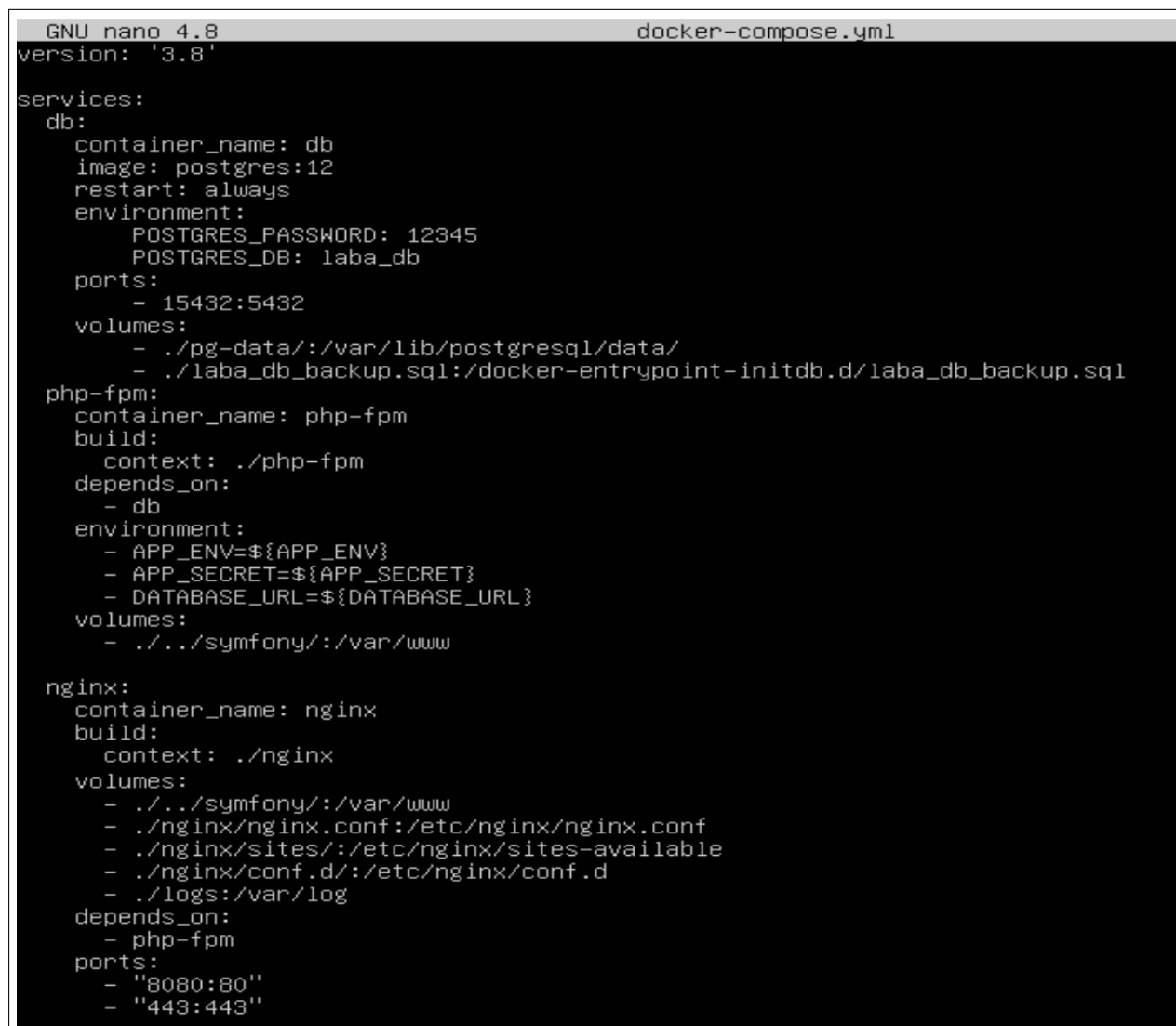
excul@exculserver:~/test/laba_folder/symfony$ php bin/console doctrine:fixtures:load
Careful, database "laba_db" will be purged. Do you want to continue? (yes/no) [no]:
> y

> purging database
> loading App\DataFixtures\AppFixtures
excul@exculserver:~/test/laba_folder/symfony$
```

Рисунок 3 - Создание и заполнение базы данных

1.3. Настройка конфигураций Docker-Compose для запуска мультиконтейнерного приложения

Создадим `docker-compose.yml`, по которому будет собираться наше приложение



```
GNU nano 4.8 docker-compose.yml
version: '3.8'

services:
  db:
    container_name: db
    image: postgres:12
    restart: always
    environment:
      POSTGRES_PASSWORD: 12345
      POSTGRES_DB: laba_db
    ports:
      - 15432:5432
    volumes:
      - ./pg-data:/var/lib/postgresql/data/
      - ./laba_db_backup.sql:/docker-entrypoint-initdb.d/laba_db_backup.sql
  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:
      - ../../symfony:/var/www
  nginx:
    container_name: nginx
    build:
      context: ./nginx
    volumes:
      - ../../symfony:/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:
      - "8080:80"
      - "443:443"
```

Рисунок 4 - `docker-compose.yml`

`volumes` отвечают за то, куда будут смонтированы файлы с сервера в контейнер и наоборот. Слева от двоеточия путь к файлам сервера, справа в контейнера.

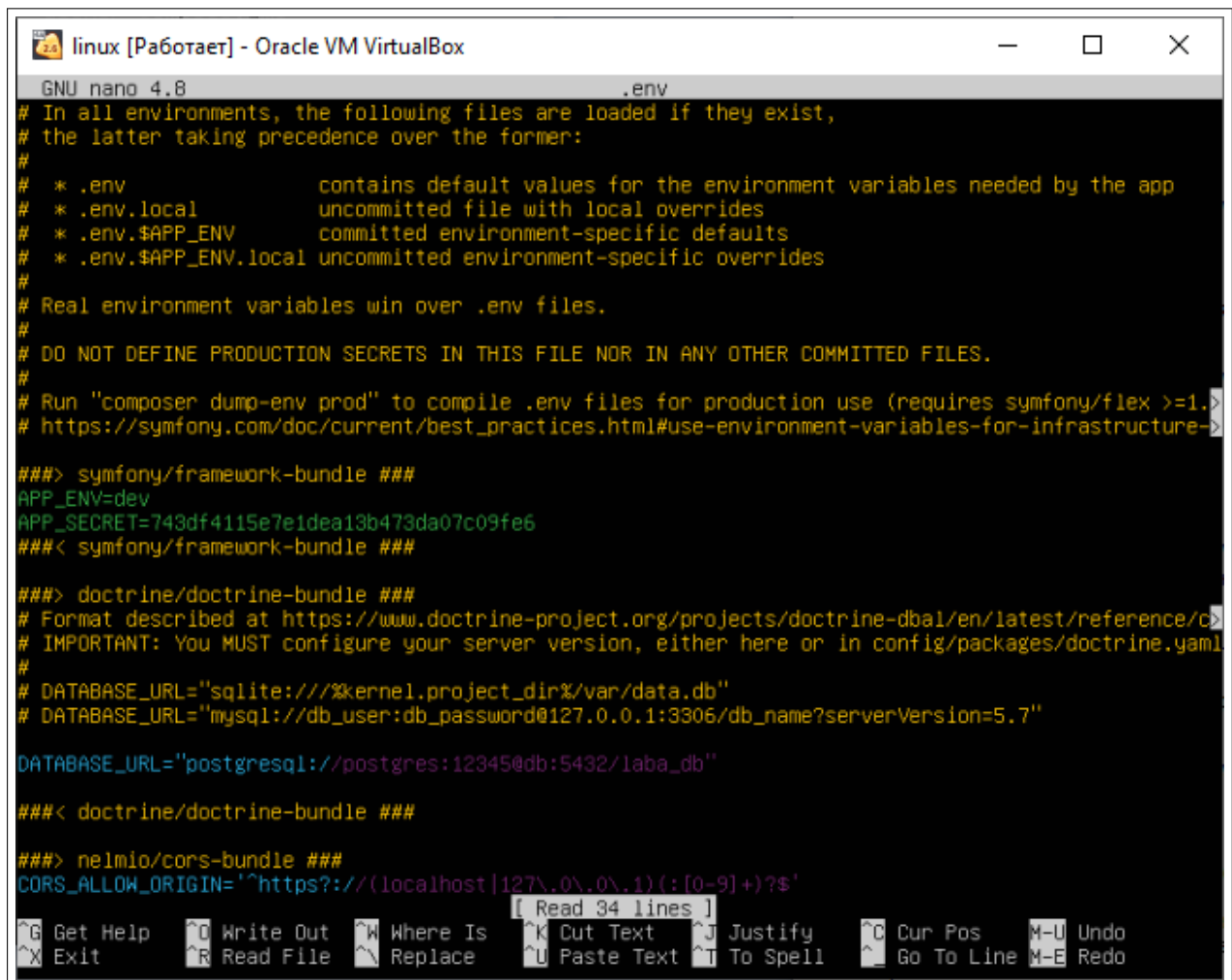
`environments` – это переменные окружения контейнера. Соответственно, для симфони проекта это переменные из `docker/.env` (который будет описан далее), а для базы данных, её переменные окружения для создания базы данных внутри контейнера.

build отвечает за то, в какой папке лежит Dockerfile для каждого контейнера. Dockerfile отвечает за выполнение команд во время создания контейнера.

ports отвечает за занимаемые порты.

depends_on отвечает за то, какой контейнер за каким будет запускаться в Docker-compose сборке. Это нужно, чтобы не возникало ошибок. Например: контейнер с Symfony уже загрузился, а база данных к нему нет.

Далее нужно создать .env для php-fpm контейнера. Можно скопировать .env из папки symfony, но теперь айпи localhost(127.0.0.1) нужно поменять на название нашего контейнера с базой данных.



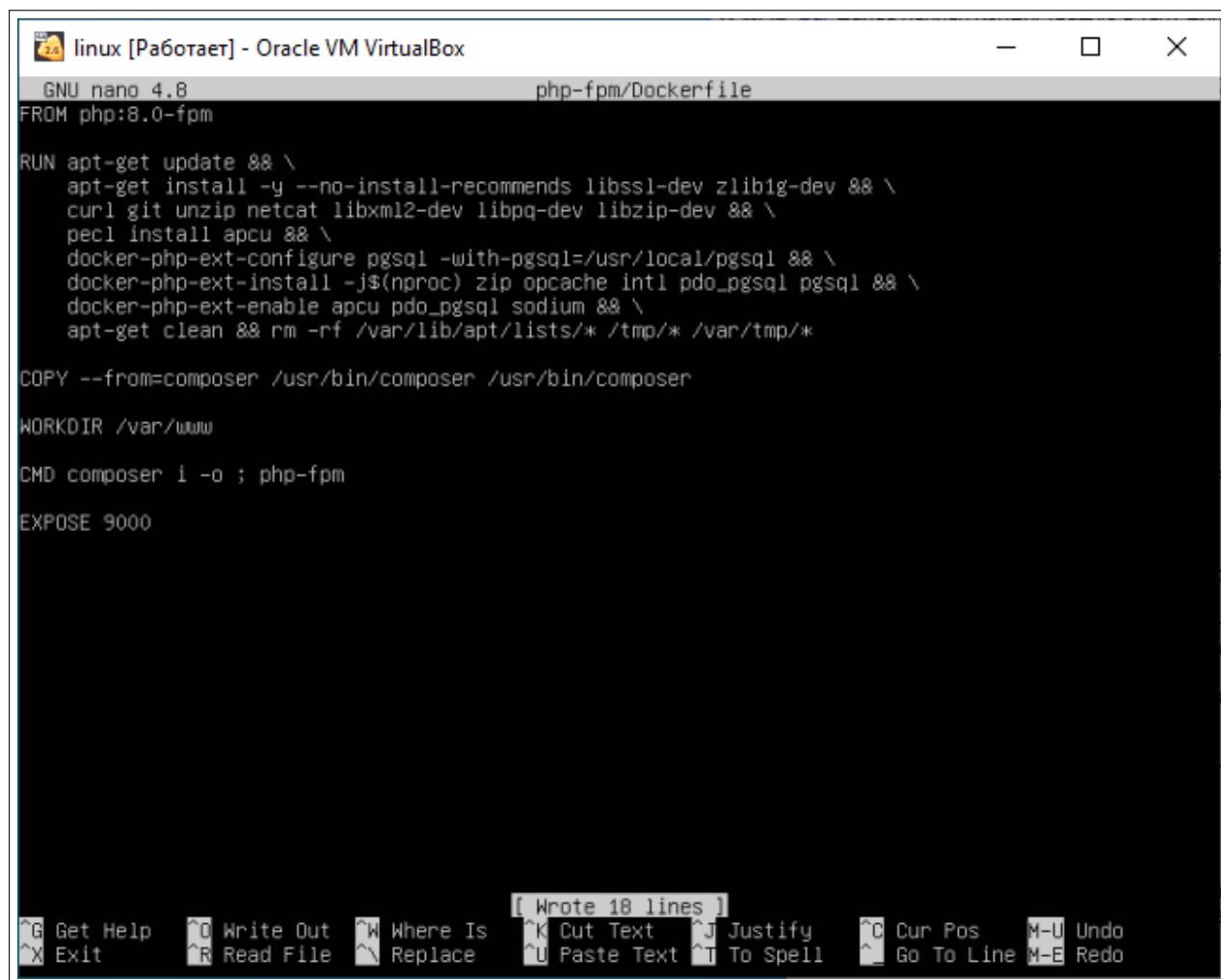
```
linux [Работает] - Oracle VM VirtualBox
GNU nano 4.8 .env
# In all environments, the following files are loaded if they exist,
# the latter taking precedence over the former:
#
# * .env             contains default values for the environment variables needed by the app
# * .env.local       uncommitted file with local overrides
# * .env.$APP_ENV    committed environment-specific defaults
# * .env.$APP_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 >=1.0)
# https://symfony.com/doc/current/best_practices.html#use-environment-variables-for-infrastructure-variables
###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=743df4115e7e1dea13b473da07c09fe6
###< symfony/framework-bundle ###

###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html#configuration
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
#
# DATABASE_URL="sqlite://%kernel.project_dir%/var/data.db"
# DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name?serverVersion=5.7"
DATABASE_URL="postgresql://postgres:12345@db:5432/laba_db"
###< doctrine/doctrine-bundle ###

###> nelmio/cors-bundle ###
CORS_ALLOW_ORIGIN='^https?://(localhost|127\.\.0\.\.1)(:[0-9]+)?$'
```

Рисунок 5 - docker/.env

Далее настраиваем php-fpm/Dockerfile



```
linux [Работает] - Oracle VM VirtualBox
GNU nano 4.8                                php-fpm/Dockerfile
FROM php:8.0-fpm

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 ; php-fpm

EXPOSE 9000

[ Wrote 18 lines ]
Get Help  Write Out  Where Is  Cut Text  Justify  Cur Pos  M-U Undo
Exit      Read File  Replace  Paste Text  To Spell  Go To Line M-E Redo
```

Рисунок 6 - php-fpm/Dockerfile

В FROM мы указываем название образа с DockerHub, в нашем случае php:8.0-fpm. Скачиваем все утилиты, которые нужны для корректной работы указываем рабочую папку и обновляем зависимости.

Далее настраиваем nginx/Dockerfile

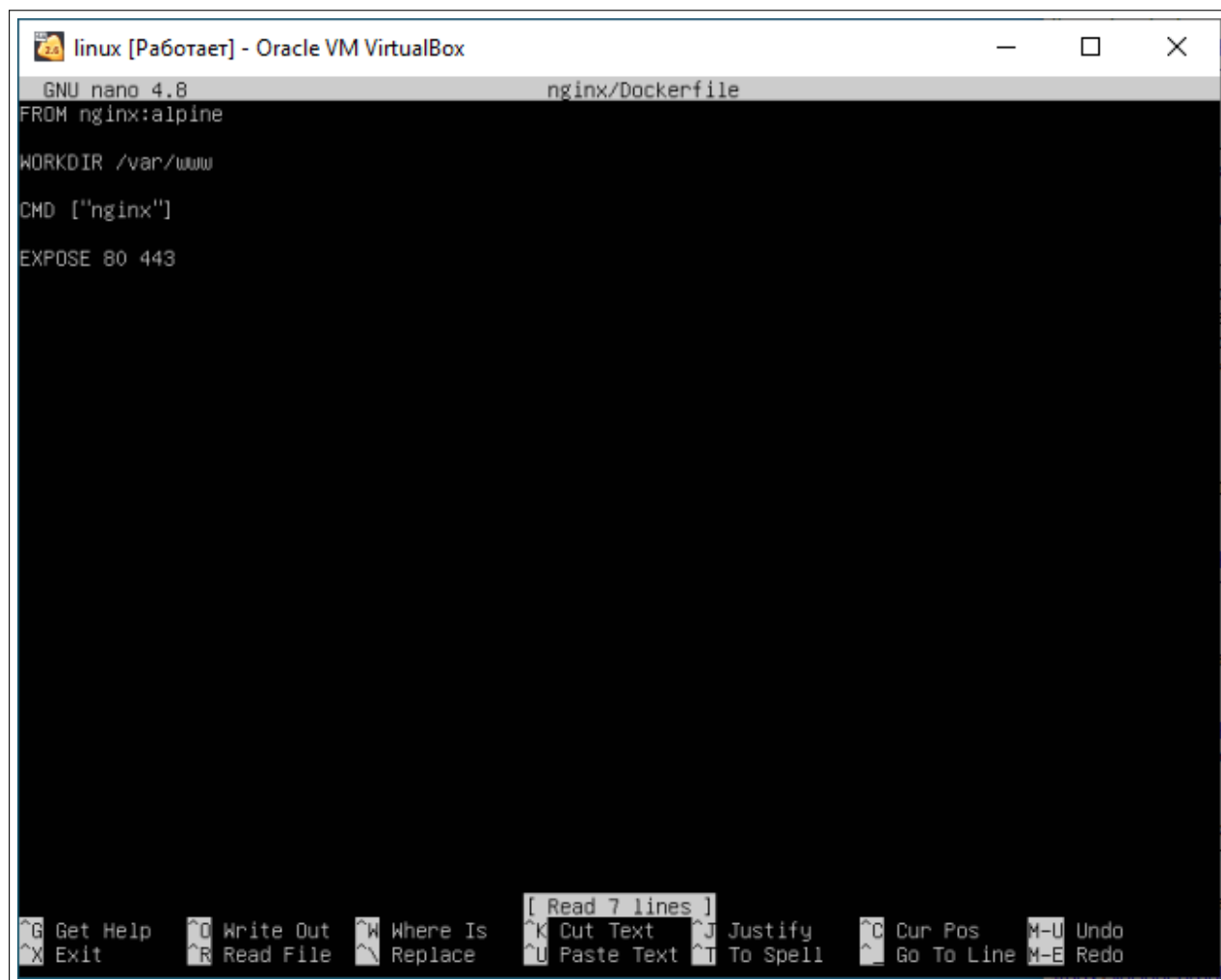
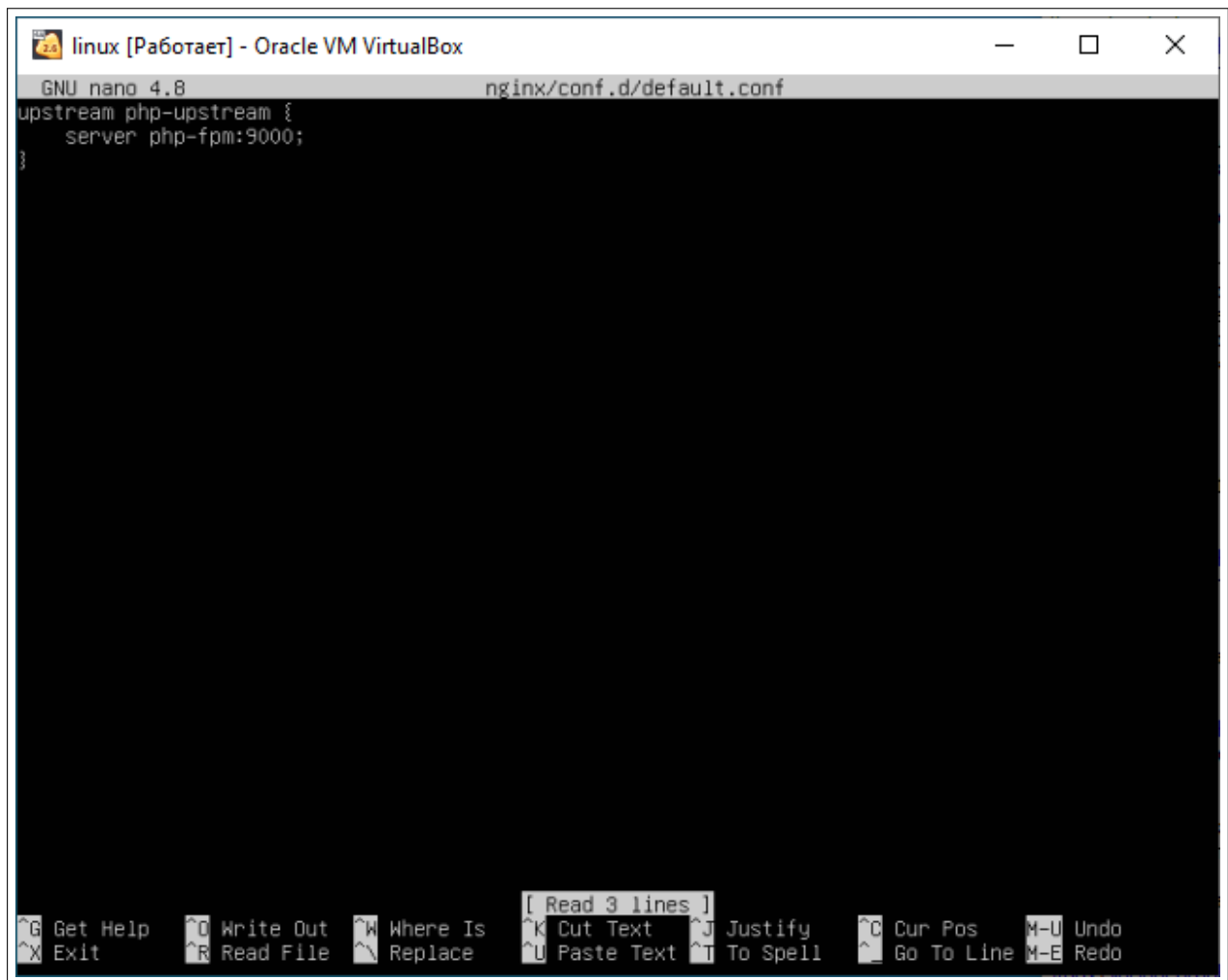


Рисунок 7 - nginx/Dockerfile

Далее настраиваем nginx/conf.d/default.conf



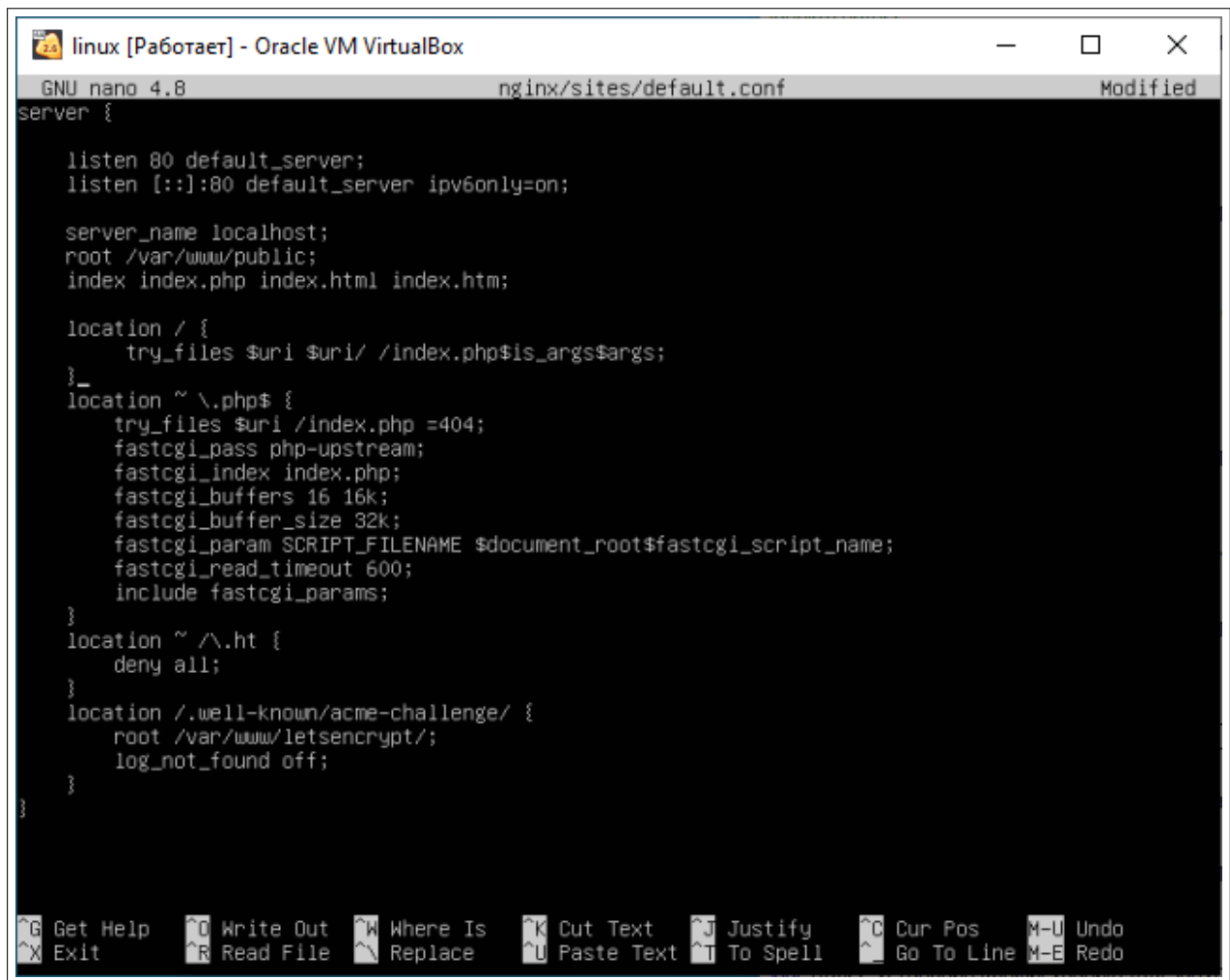
```
linux [Работает] - Oracle VM VirtualBox
GNU nano 4.8 nginx/conf.d/default.conf
upstream php-upstream {
    server php-fpm:9000;
}

[ Read 3 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos  ^M-U Undo
^X Exit      ^R Read File ^_ Replace   ^U Paste Text ^T To Spell  ^_ Go To Line ^M-E Redo
```

Рисунок 8 - nginx/conf.d/default.conf

Директива `upstream` позволяет распределять запросы по прокси на другие серверы, в нашем случае это контейнер `php-fpm`. В настройке самого сервера `php-upstream` будет установлен в `fastcgi_pass`.

Далее установим конфигурации для `nginx` внутри контейнера

A screenshot of a terminal window titled "linux [Работает] - Oracle VM VirtualBox". The terminal shows the contents of the file "nginx/sites/default.conf" using the "nano" editor. The configuration file content is as follows:

```
server {  
    listen 80 default_server;  
    listen [::]:80 default_server ipv6only=on;  
  
    server_name localhost;  
    root /var/www/public;  
    index index.php index.html index.htm;  
  
    location / {  
        try_files $uri $uri/ /index.php$is_args$args;  
    }  
    location ~ \.php$ {  
        try_files $uri /index.php =404;  
        fastcgi_pass php-upstream;  
        fastcgi_index index.php;  
        fastcgi_buffers 16 16k;  
        fastcgi_buffer_size 32k;  
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;  
        fastcgi_read_timeout 600;  
        include fastcgi_params;  
    }  
    location ~ /\.ht {  
        deny all;  
    }  
    location /.well-known/acme-challenge/ {  
        root /var/www/letsencrypt/;  
        log_not_found off;  
    }  
}
```

The terminal window has a status bar at the bottom with various shortcuts: ^G Get Help, ^O Write Out, ^W Where Is, ^K Cut Text, ^J Justify, ^C Cur Pos, M-U Undo, ^X Exit, ^R Read File, ^M Replace, ^U Paste Text, ^T To Spell, ^G Go To Line, M-E Redo.

Рисунок 10 - nginx/sites/default.conf

Далее установим конфигурацию для запуска сервера nginx из контейнера. Указав разрешённые файлы, переменный FastCGI и другие настройки.

В принципе, у нас уже есть рабочий проект, но на нём нет баз данных, т.к. мы не добавили её в наш контейнер. Есть 2 пути решения, собрать проект и добавить её в уже работающий контейнер или смонтировать и добавить её на стадии сборки приложения. В нашем случае используем второй вариант.

Вернёмся к docker-compose.yml и увидим в volumes у db строчку начинающуюся с ./laba_db_backup.sql . Это дамп базы данных, который необходимо смонтировать к нашему контейнеру. Давайте создадим его.

Для этого зайдём под пользователем postgres и запустим утилиту pg_dump и по URL базы данных сделаем дамп в файл laba_db_backup.sql

```
excul@exculserver:~/test/laba_folder/docker$ sudo su - postgres
postgres@exculserver:~$ pg_dump postgresql://postgres:12345@127.0.0.1:laba_db > laba_db_backup.sql
postgres@exculserver:~$ ls
12 laba_db_backup.sql
postgres@exculserver:~$
```

Рисунок 11 - Создание дампа базы данных

Копируем её в /docker.

Проект настроен, осталось только собрать. Используем команду docker-compose build

```
linux [Работает] - Oracle VM VirtualBox
Step 2/6 : 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$(nproc) zip opcache intl pdo_pgsql pgsql && docker-php-ext-enable apcu pdo_pgsql sodium && apt-get clean && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
--> Using cache
--> d47d411160f2
Step 3/6 : COPY --from=composer /usr/bin/composer /usr/bin/composer
--> Using cache
--> 4c5c0ac28470
Step 4/6 : WORKDIR /var/www
--> Using cache
--> 3ad1f9feefd2
Step 5/6 : CMD composer i -o ; php-fpm
--> Using cache
--> fc4aec3bb370
Step 6/6 : EXPOSE 9000
--> Using cache
--> cb605292dc61
Successfully built cb605292dc61
Successfully tagged docker_php:fpm:latest
Building nginx
Sending build context to Docker daemon 6.656kB
Step 1/4 : FROM nginx:alpine
--> cc44224bfe20
Step 2/4 : WORKDIR /var/www
--> Using cache
--> 19cd6dc175db
Step 3/4 : CMD ["nginx"]
--> Using cache
--> 239c7a916d81
Step 4/4 : EXPOSE 80 443
--> Using cache
--> a3dc9b0845d9
Successfully built a3dc9b0845d9
Successfully tagged docker_nginx:latest
excul@exculserver:~/test/laba_folder/docker$ _
```

Рисунок 12 - Сборка проекта

Теперь запускаем, чтобы проверить. Чтобы запустить наше приложение пишем docker-compose up -d.

```
excul@exculserver:~/test/laba_folder/docker$ sudo docker-compose up -d
Creating db ... done
Creating php-fpm ... done
Creating nginx ... done
excul@exculserver:~/test/laba_folder/docker$ _
```

Рисунок 13 - Сборка проекта

Далее зайдём по айпи nginx и порту, указанному в docker-compose.yml(8080) на наш сайт.

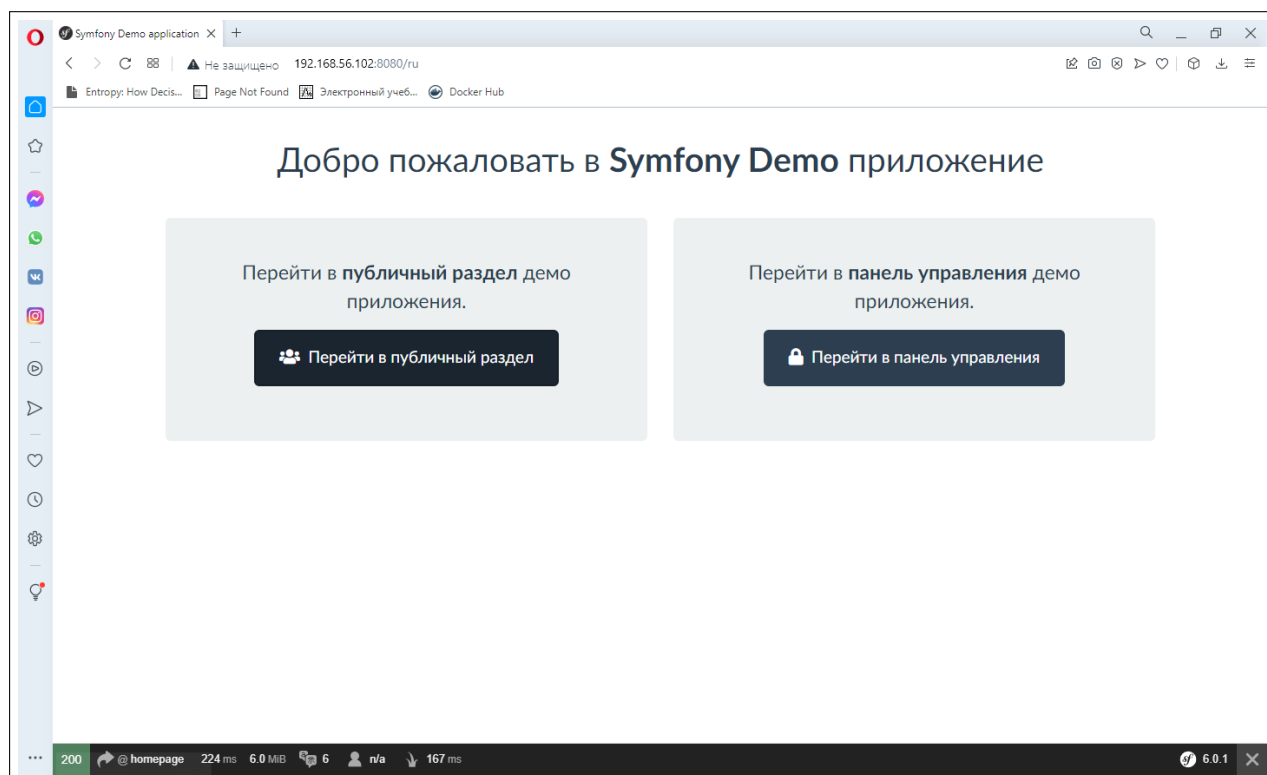


Рисунок 14 - Результат 1

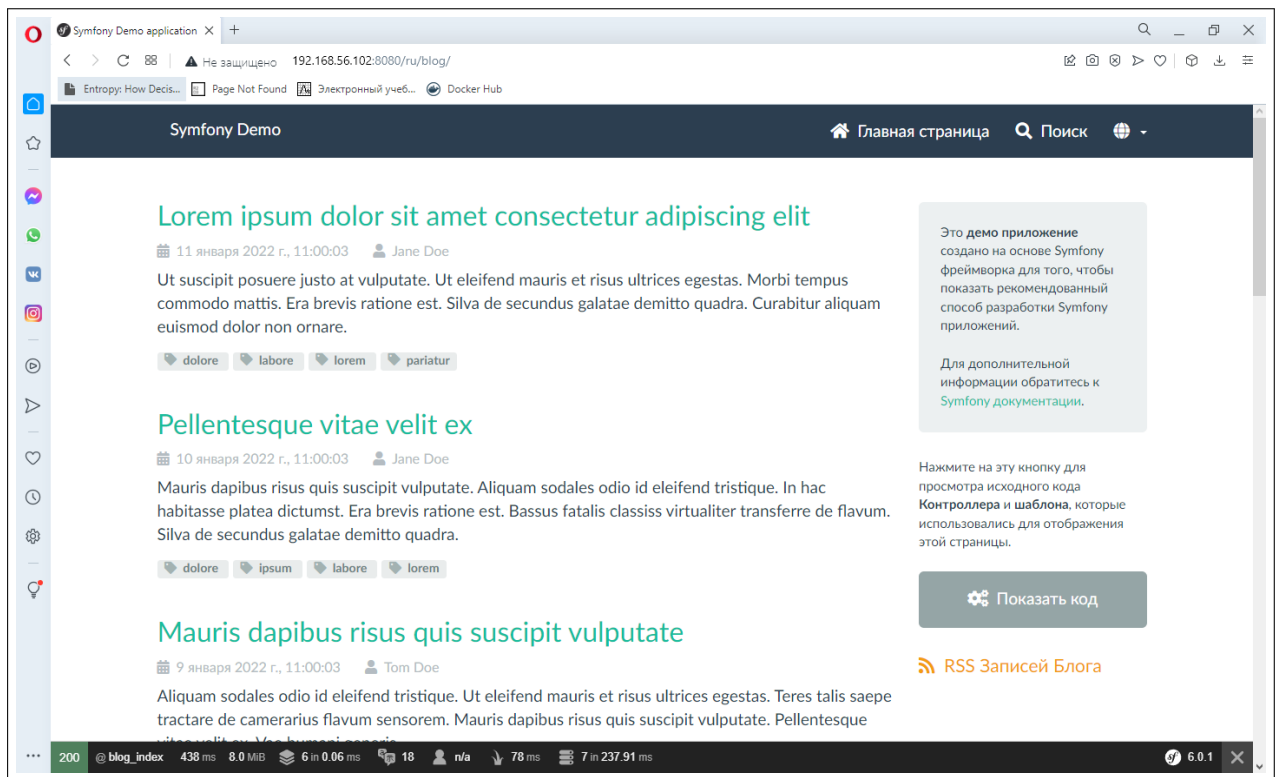
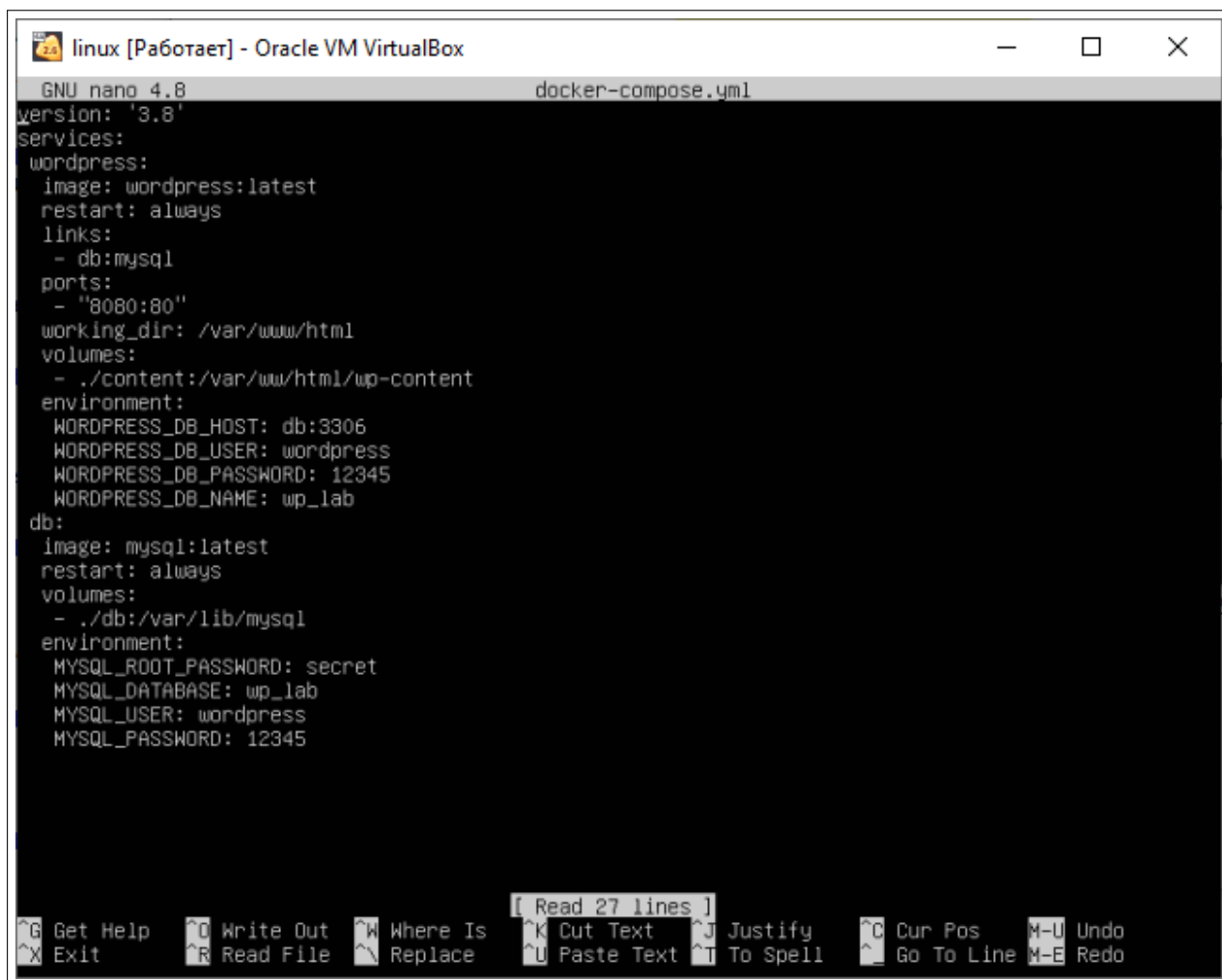


Рисунок 15 - Результат 2

1.4. Проект на Wordpress

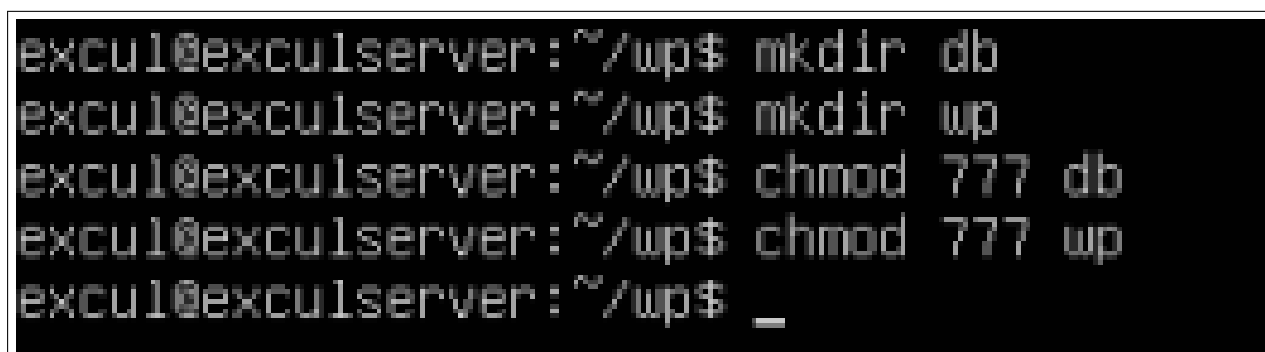
Делаем аналогично предыдущим шагам

A screenshot of a terminal window titled "linux [Работает] - Oracle VM VirtualBox". The terminal shows the nano 4.8 editor editing a file named "docker-compose.yml". The file content is a Docker Compose configuration for a WordPress service and a database service. The WordPress service uses the "wordpress:latest" image, restarts "always", and links to the "db:mysql" service. It maps port "8080:80" and uses the working directory "/var/www/html". It mounts the local directory "./content" to "/var/www/html/wp-content". The database service uses the "mysql:latest" image, restarts "always", and mounts the local directory "./db" to "/var/lib/mysql". Both services have environment variables for database connection details. The terminal window has a menu bar at the bottom with options like "Get Help", "Write Out", "Where Is", "Cut Text", "Justify", "Cur Pos", "Exit", "Read File", "Replace", "Paste Text", "To Spell", "Go To Line", "Undo", and "Redo".

```
GNU nano 4.8 docker-compose.yml
version: '3.8'
services:
  wordpress:
    image: wordpress:latest
    restart: always
    links:
      - db:mysql
    ports:
      - "8080:80"
    working_dir: /var/www/html
    volumes:
      - ./content:/var/www/html/wp-content
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: 12345
      WORDPRESS_DB_NAME: wp_lab
  db:
    image: mysql:latest
    restart: always
    volumes:
      - ./db:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: secret
      MYSQL_DATABASE: wp_lab
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: 12345
```

Рисунок 16 - docker-compose.yml

Создаём папки для монтирования

A screenshot of a terminal window showing a series of commands being executed to create and set permissions for two directories, 'db' and 'wp'. The prompt is 'excul@exculserver: ~/wp\$'. The commands are: 'mkdir db', 'mkdir wp', 'chmod 777 db', 'chmod 777 wp', and an empty prompt line indicating the end of the sequence.

```
excul@exculserver: ~/wp$ mkdir db
excul@exculserver: ~/wp$ mkdir wp
excul@exculserver: ~/wp$ chmod 777 db
excul@exculserver: ~/wp$ chmod 777 wp
excul@exculserver: ~/wp$ _
```

Рисунок 17 - Папки для монтирования

Далее собираем и запускаем проект командой `docker-compose up -d --build`

Результат:

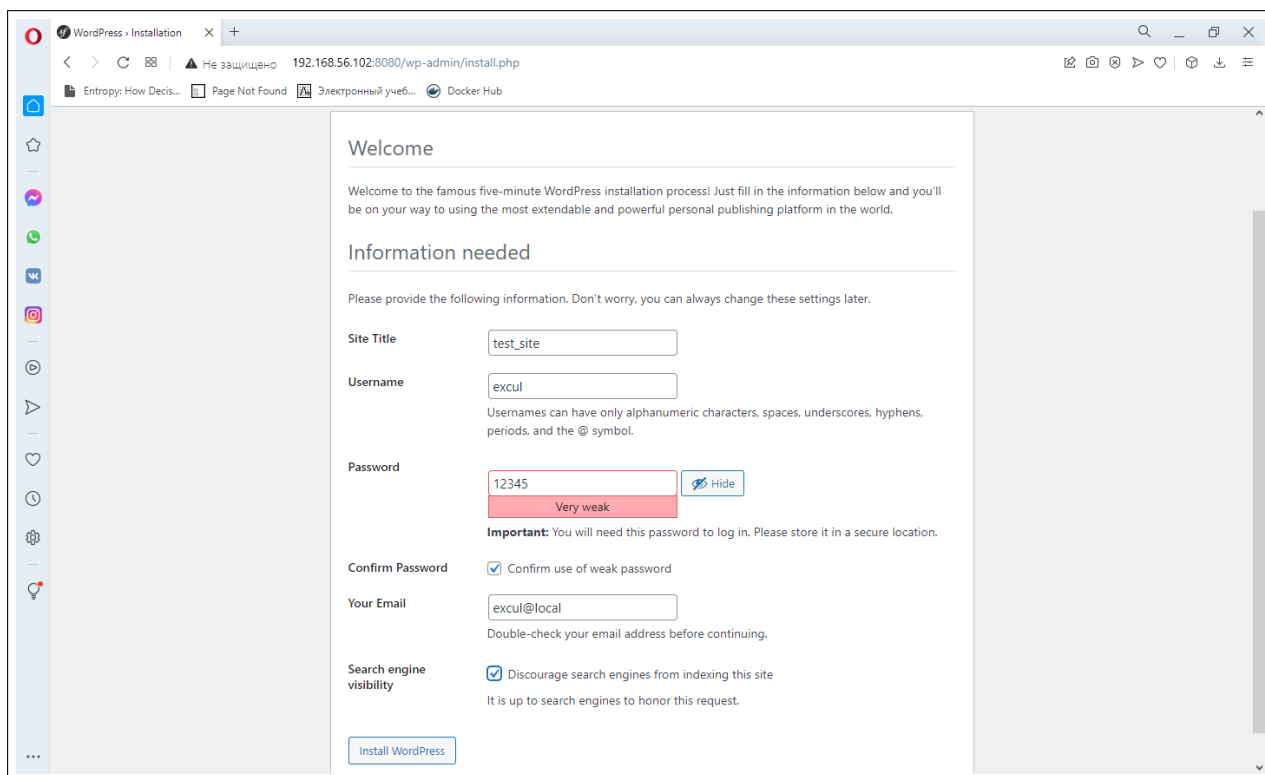


Рисунок 18 - Результат 1

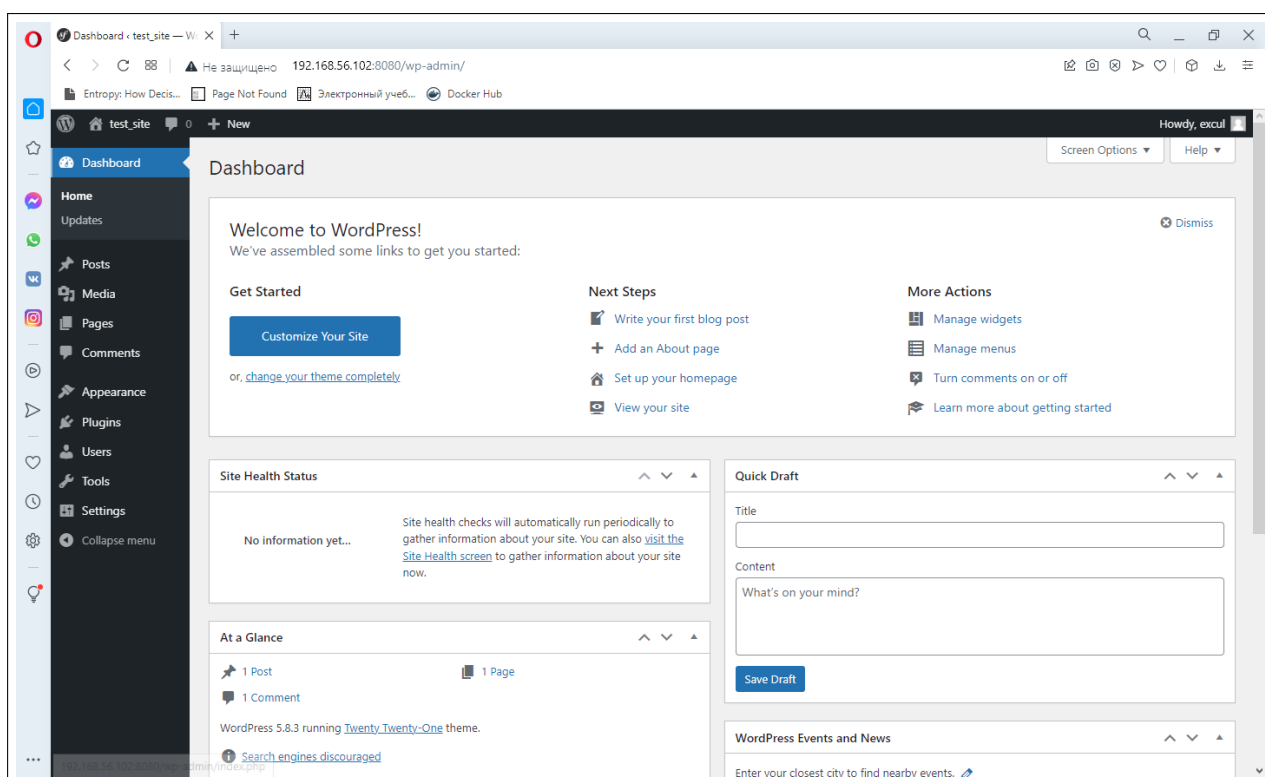


Рисунок 19 - Результат 2

2. Контрольные вопросы

1. Вопрос: Назовите отличия использования контейнеров по сравнению с виртуализацией.

Ответ: Меньшие накладные расходы на инфраструктуру

2. Вопрос: Назовите основные компоненты Docker.

Ответ: Контейнеры

3. Вопрос: Какие технологии используются для работы с контейнерами?

Ответ: Контрольные группы (cgroups)

4. Вопрос: Найдите соответствие между компонентом и его описанием

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. Вопрос: В чем отличие контейнеров от виртуализации?

Ответ: Контейнер обеспечивает виртуализацию на уровне операционной системы, а аппаратная виртуализация обеспечивает виртуализацию на уровне машины.

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

Ответ: — `docker ps` — показывает список запущенных контейнеров;

— `docker pull` — скачать определённый образ или набор образов (репозиторий);

— `docker build` — эта команда собирает образ Docker из Dockerfile и «контекста»;

— `docker run` — запускает контейнер, на основе указанного образа;

— `docker logs` — эта команда используется для просмотра логов указанного контейнера;

- `docker volume ls` — показывает список томов, которые являются предпочитаемым механизмом для сохранения данных, генерируемых и используемых контейнерами Docker;
- `docker rm` — удаляет один и более контейнеров;
- `docker rmi` — удаляет один и более образов;
- `docker stop` — останавливает один и более контейнеров;
- `docker exec -it ...` - выполняет команду в определенном контейнере

7. Вопрос: Каким образом осуществляется поиск образов контейнеров?

Ответ: Сначала проверяется локальный репозиторий на наличия нужного контейнера, если он не найден локально, то поиск производится в репозитории Docker Hub.

8. Вопрос: Каким образом осуществляется запуск контейнера?

Ответ: Команда `docker run (container_name)`

9. Вопрос: Что значит управлять состоянием контейнеров?

Ответ: Это означает, что в любой момент времени есть возможность запустить, остановить или выполнить команды внутри контейнера.

10. Вопрос: Как изолировать контейнер?

Ответ: Контейнеры уже по сути своей являются изолированными единицами, поэтому достаточно без ошибок сконфигурировать файлы Dockerfile и/или docker-compose.yml.

11. Вопрос: Опишите последовательность создания новых образов, назначение Dockerfile?

Ответ: Производится выбор основы для нового образа на Docker Hub, далее производится конфигурация Dockerfile, где описываются все необходимые пакеты, файлы, команды и т.п.

Dockerfile — это текстовый файл с инструкциями, необходимыми для создания образа контейнера. Эти инструкции включают идентифи-

кацию существующего образа, используемого в качестве основы, команды, выполняемые в процессе создания образа, и команду, которая будет выполняться при развертывании новых экземпляров этого образа контейнера.

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

Ответ: Да. Существует Kubernetes.

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

Ответ: 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.

— Nodes: Нода это машина в кластере Kubernetes.

— Pods: Pod это группа контейнеров с общими разделами, запускаемых как единое целое.

— Replication Controllers: replication controller гарантирует, что определенное количество «реplik» pod’ы будут запущены в любой момент времени.

— Services: Сервис в Kubernetes – это абстракция, которая определяет логический объединённый набор pod и политику доступа к ним.

— Volumes: Volume(раздел) это директория, возможно, с данными в ней, которая доступна в контейнере.

— Labels: Label’ы это пары ключ/значение которые прикрепляются к объектам, например pod’ам. Label’ы могут быть использованы для создания и выбора наборов объектов.

— Kubectl Command Line Interface: kubectl интерфейс командной строки для управления Kubernetes.