

**Липецкий государственный технический университет**

**Факультет автоматизации и информатики**

**Кафедра автоматизированных систем управления**

**ЛАБОРАТОРНАЯ РАБОТА №5**

**по дисциплине «Операционная система Linux»**

**Контейнеризация**

Студент

Киренский Д.К.

Группа ПИ-19

Руководитель  
Доцент, к.п.н.

Кургасов В.В.

Липецк 2022 г.

## Оглавление

Цель работы.....	3
Задание кафедры .....	4
Ход работы .....	5
Вывод .....	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.

## Ход работы

### 1. Работа с тестовым проектом symfony.

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

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

```
root@ubuntuuser:~# git clone https://github.com/symfony/demo
Cloning into 'demo'...
remote: Enumerating objects: 10570, done.
remote: Counting objects: 100% (695/695), done.
remote: Compressing objects: 100% (425/425), done.
remote: Total 10570 (delta 358), reused 516 (delta 242), pack-reused 9875
Receiving objects: 100% (10570/10570), 19.02 MiB | 409.00 KiB/s, done.
Resolving deltas: 100% (6300/6300), done.
root@ubuntuuser:~# cd demo
root@ubuntuuser:~/demo#
```

Рисунок 1 – Клонирование проекта

Далее установим все сопутствующие зависимости с помощью команды `composer install` и запустим проект с помощью команды `symfony serve`. Результат работы представлен на рисунках 2 и 3.

```
root@ubuntuuser:~/demo# symfony serve

[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

Tailing Web Server log file (/root/.symfony/log/dccf9746bbea6a3c592ea566a426e71379b87e7a.log)
Tailing PHP-FPM log file (/root/.symfony/log/dccf9746bbea6a3c592ea566a426e71379b87e7a/53fb8ec204547646acb3461995e4da5a54cc7575.log)

[OK] Web server listening
The Web server is using PHP FPM 8.1.0
http://127.0.0.1:8000

[Web Server ] Jan 21 19:32:49 |DEBUG| PHP   Reloading PHP versions
[Web Server ] Jan 21 19:32:49 |DEBUG| PHP   Using PHP version 8.1.0 (from default version in $PATH)
[Web Server ] Jan 21 19:32:49 |INFO|  PHP   listening path="/usr/sbin/php-fpm8.1" php="8.1.0" port=35243
[PHP-FPM    ] Jan 21 19:32:49 |NOTICE| FPM   fpm is running, pid 79102
[PHP-FPM    ] Jan 21 19:32:49 |NOTICE| FPM   ready to handle connections
[PHP-FPM    ] Jan 21 19:32:49 |NOTICE| FPM   systemd monitor interval set to 10000ms
[Web Server ] Jan 21 19:32:58 |INFO|  SERVER GET (200) / ip="127.0.0.1"
```

Рисунок 2 – Запуск проекта

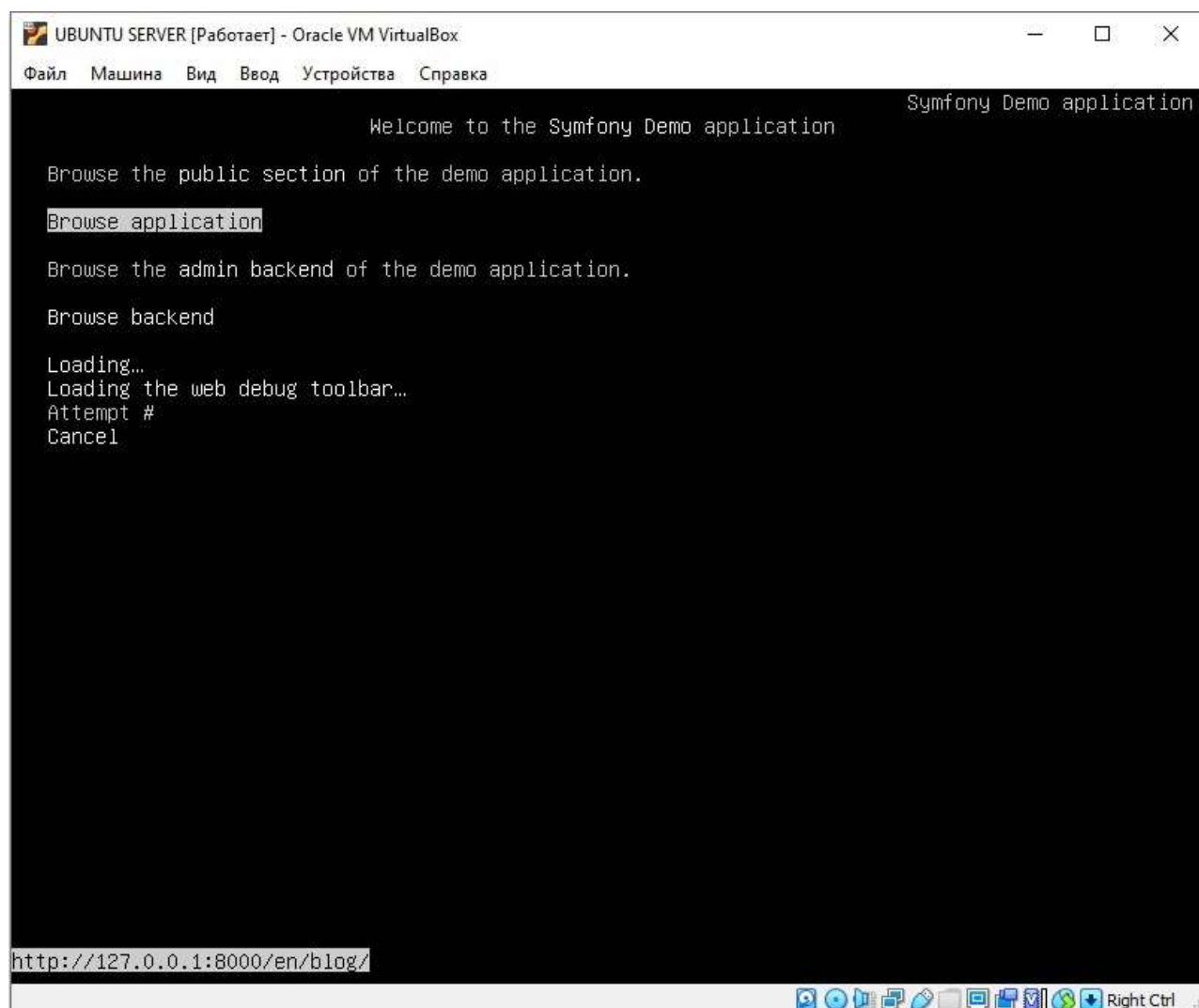
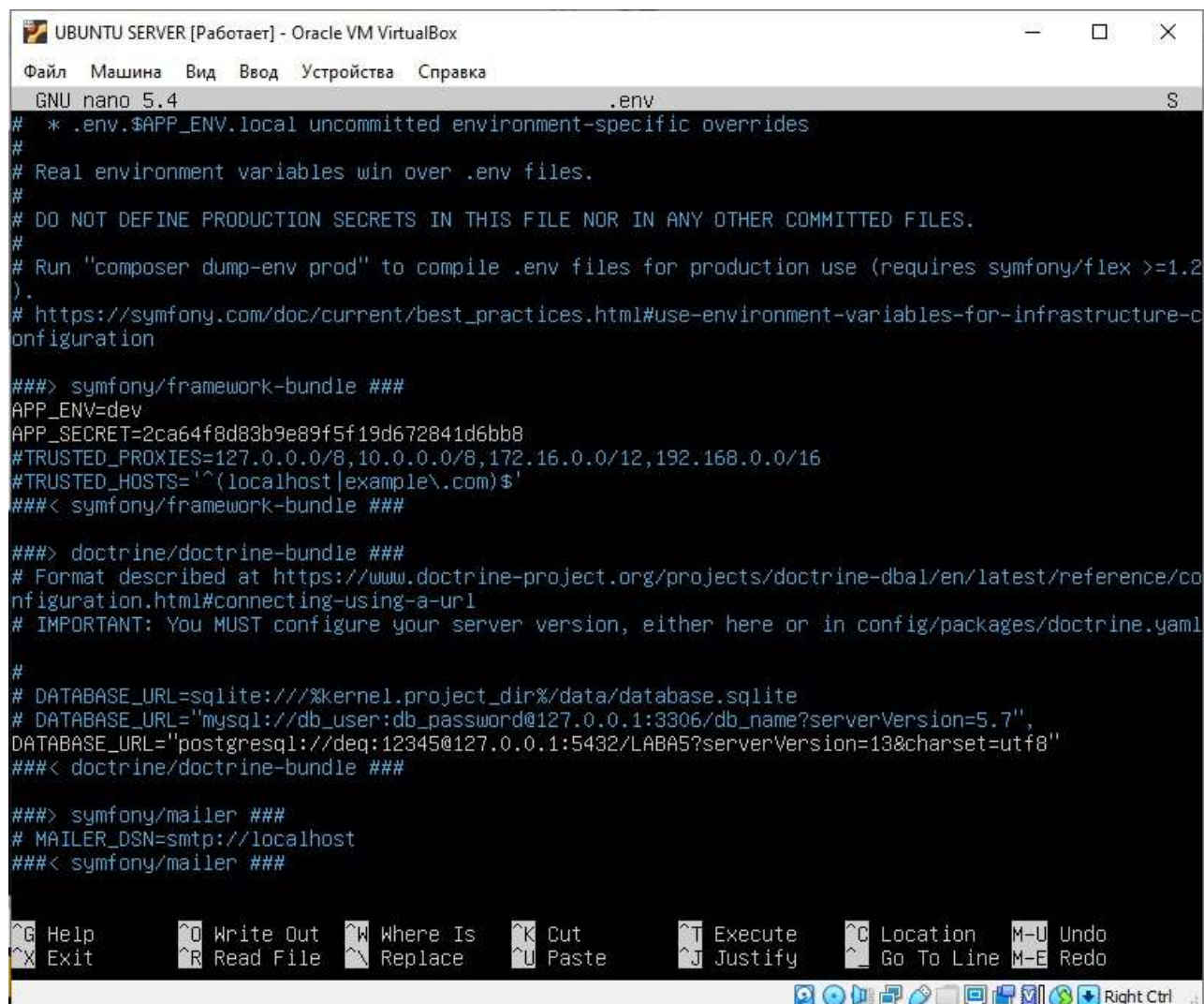


Рисунок 3 – Запуск проекта

Создадим БД для тестирования проекта

Для начала, изменим СУБД в файле .env



```
UBUNTU SERVER [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
GNU nano 5.4 .env
# * .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 >=1.2
).
# https://symfony.com/doc/current/best_practices.html#use-environment-variables-for-infrastructure-c
onfiguration

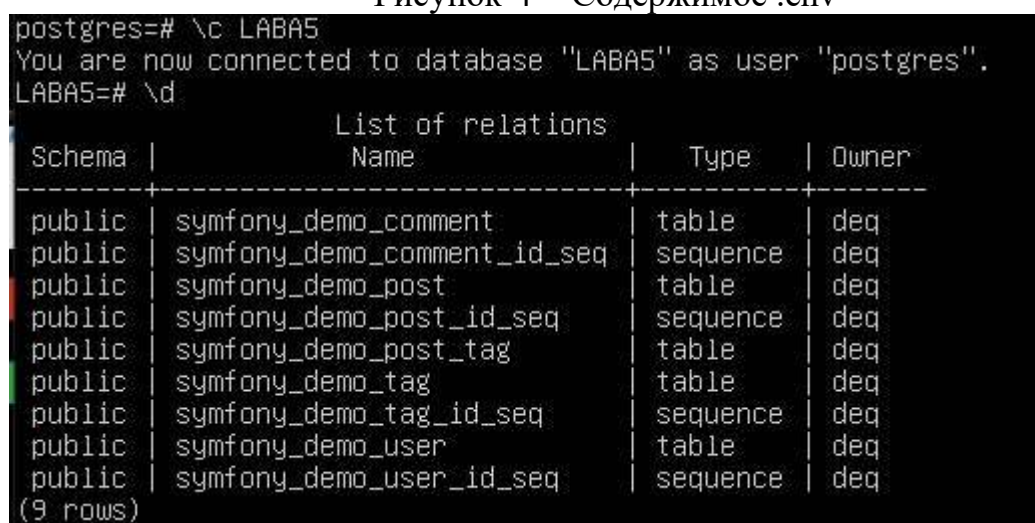
###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=2ca64f8d83b9e89f5f19d672841d6bb8
#TRUSTED_PROXIES=127.0.0.0/8,10.0.0.0/8,172.16.0.0/12,192.168.0.0/16
#TRUSTED_HOSTS='^(localhost|example\.com)$'
###< symfony/framework-bundle ###

###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/co
nfiguration.html#connecting-using-a-url
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
#
# 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://deq:12345@127.0.0.1:5432/LABA5?serverVersion=13&charset=utf8"
###< doctrine/doctrine-bundle ###

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

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit      ^R Read File  ^N Replace    ^U Paste      ^J Justify    ^_ Go To Line  M-E Redo
Right Ctrl
```

Рисунок 4 – Содержимое .env



```
postgres=# \c LABA5
You are now connected to database "LABA5" as user "postgres".
LABA5=# \d

              List of relations
  Schema | Name                                     | Type  | Owner
-----|-----|-----|-----
 public | symfony_demo_comment                    | table  | deq
 public | symfony_demo_comment_id_seq              | sequence | deq
 public | symfony_demo_post                        | table  | deq
 public | symfony_demo_post_id_seq                  | sequence | deq
 public | symfony_demo_post_tag                     | table  | deq
 public | symfony_demo_tag                          | table  | deq
 public | symfony_demo_tag_id_seq                    | sequence | deq
 public | symfony_demo_user                         | table  | deq
 public | symfony_demo_user_id_seq                  | sequence | deq
(9 rows)
```

Рисунок 5 – Создание БД

Далее настроим контейнеры.

Создадим отдельную папку `docker` в которой будут все докер файлы. В ней мы создадим следующие папки и файлы:

`docker-compose.yml`

`nginx/`

`nginx/nginx.conf`

`nginx/Dockerfile`

`nginx/sites/`

`nginx/sites/default.conf`

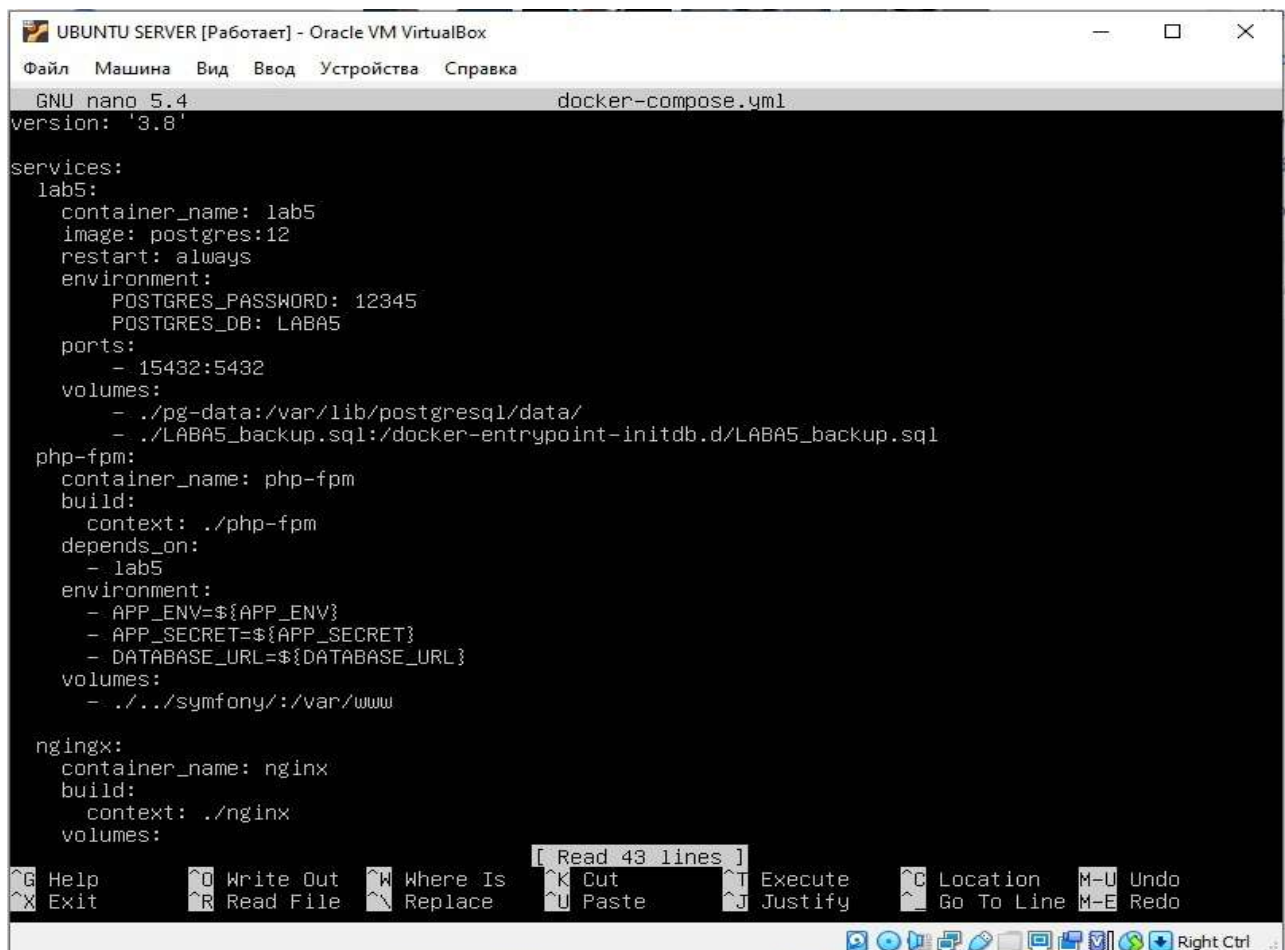
`nginx/conf.d/`

`nginx/conf.d/default.conf`

`php-fpm/`

`php-fpm/Dockerfile`

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



```
UBUNTU SERVER [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
GNU nano 5.4                                docker-compose.yml
version: '3.8'

services:
  lab5:
    container_name: lab5
    image: postgres:12
    restart: always
    environment:
      POSTGRES_PASSWORD: 12345
      POSTGRES_DB: LABA5
    ports:
      - 15432:5432
    volumes:
      - ../pg-data:/var/lib/postgresql/data/
      - ../LABA5_backup.sql:/docker-entrypoint-initdb.d/LABA5_backup.sql
  php-fpm:
    container_name: php-fpm
    build:
      context: ../php-fpm
    depends_on:
      - lab5
    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:
```

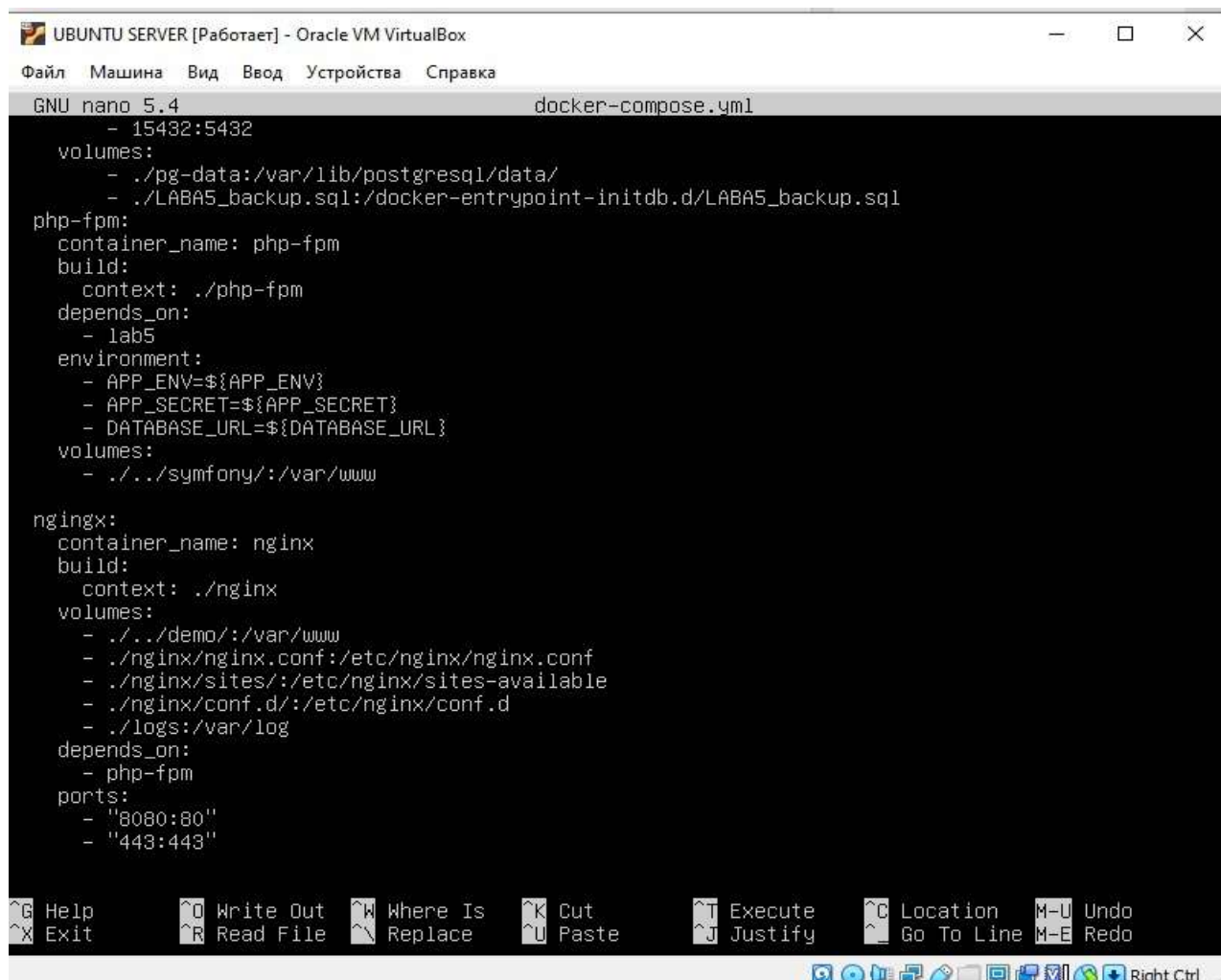
[ Read 43 lines ]

Help Write Out Where Is Cut Execute Location M-U Undo  
Exit Read File Replace Paste Justify Go To Line M-E Redo

Right Ctrl



Рисунок 6 – содержимое файла docker-compose.yml.



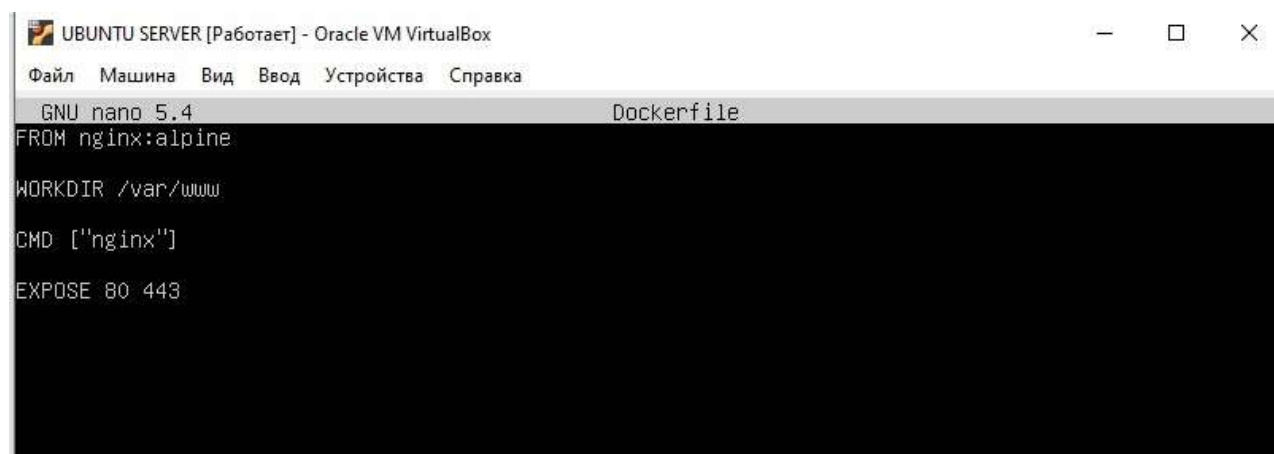
The screenshot shows a terminal window titled "UBUNTU SERVER [Работает] - Oracle VM VirtualBox". The terminal is running the nano text editor, editing a file named "docker-compose.yml". The content of the file is as follows:

```
15432:5432
volumes:
  - ./pg-data:/var/lib/postgresql/data/
  - ./LABA5_backup.sql:/docker-entrypoint-initdb.d/LABA5_backup.sql
php-fpm:
  container_name: php-fpm
  build:
    context: ./php-fpm
  depends_on:
    - lab5
  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:
    - ../../demo/:/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"
```

The bottom of the terminal shows the nano editor's command palette with various shortcuts like ^G Help, ^O Write Out, ^W Where Is, ^K Cut, ^T Execute, ^C Location, M-U Undo, ^X Exit, ^R Read File, ^\_ Replace, ^U Paste, ^J Justify, ^\_ Go To Line, M-E Redo, and a Right Ctrl button.

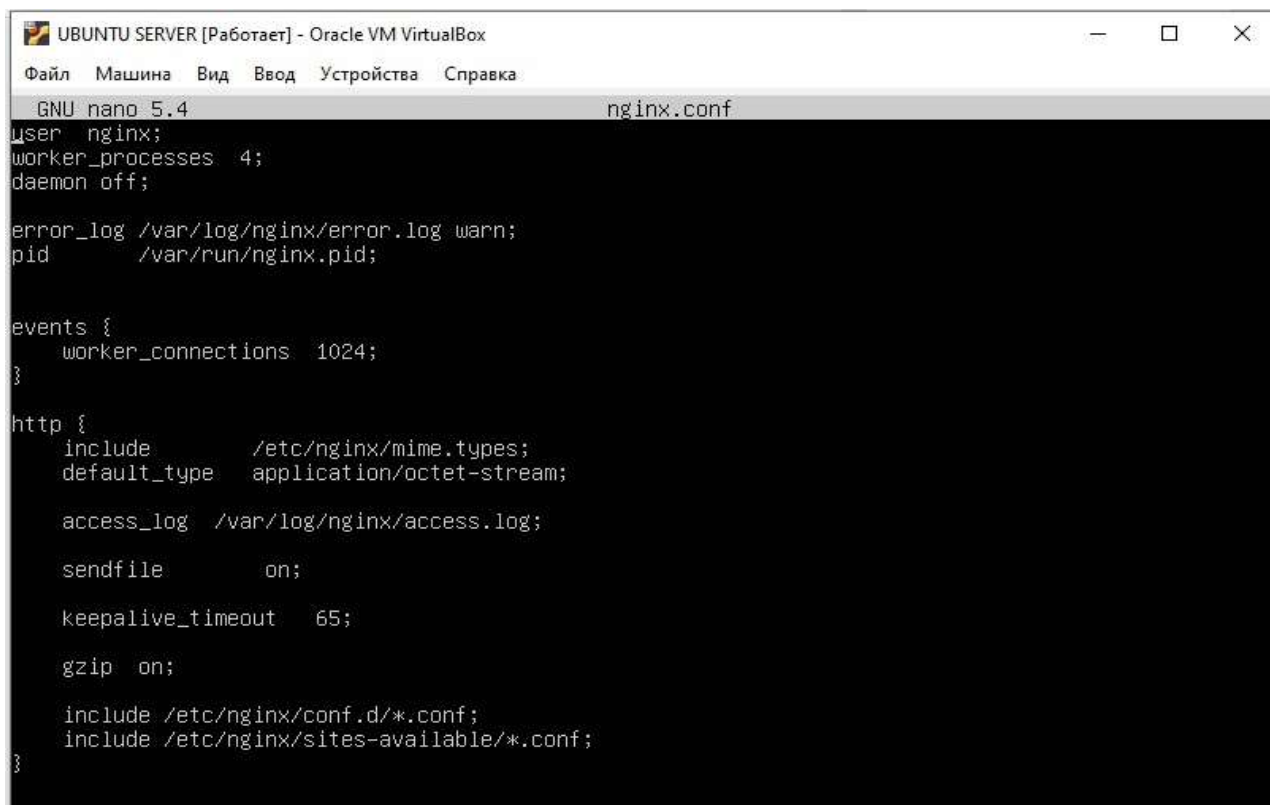
Рисунок 7 – содержимое файла docker-compose.yml.



The screenshot shows a terminal window titled "UBUNTU SERVER [Работает] - Oracle VM VirtualBox". The terminal is running the nano text editor, editing a file named "Dockerfile". The content of the file is as follows:

```
FROM nginx:alpine
WORKDIR /var/www
CMD ["nginx"]
EXPOSE 80 443
```

Рисунок 8 – содержимое файла nginx/Dockerfile



UBUNTU SERVER [Работает] - Oracle VM VirtualBox

Файл Машина Вид Ввод Устройства Справка

GNU nano 5.4 nginx.conf

```
user nginx;
worker_processes 4;
daemon off;

error_log /var/log/nginx/error.log warn;
pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    access_log /var/log/nginx/access.log;

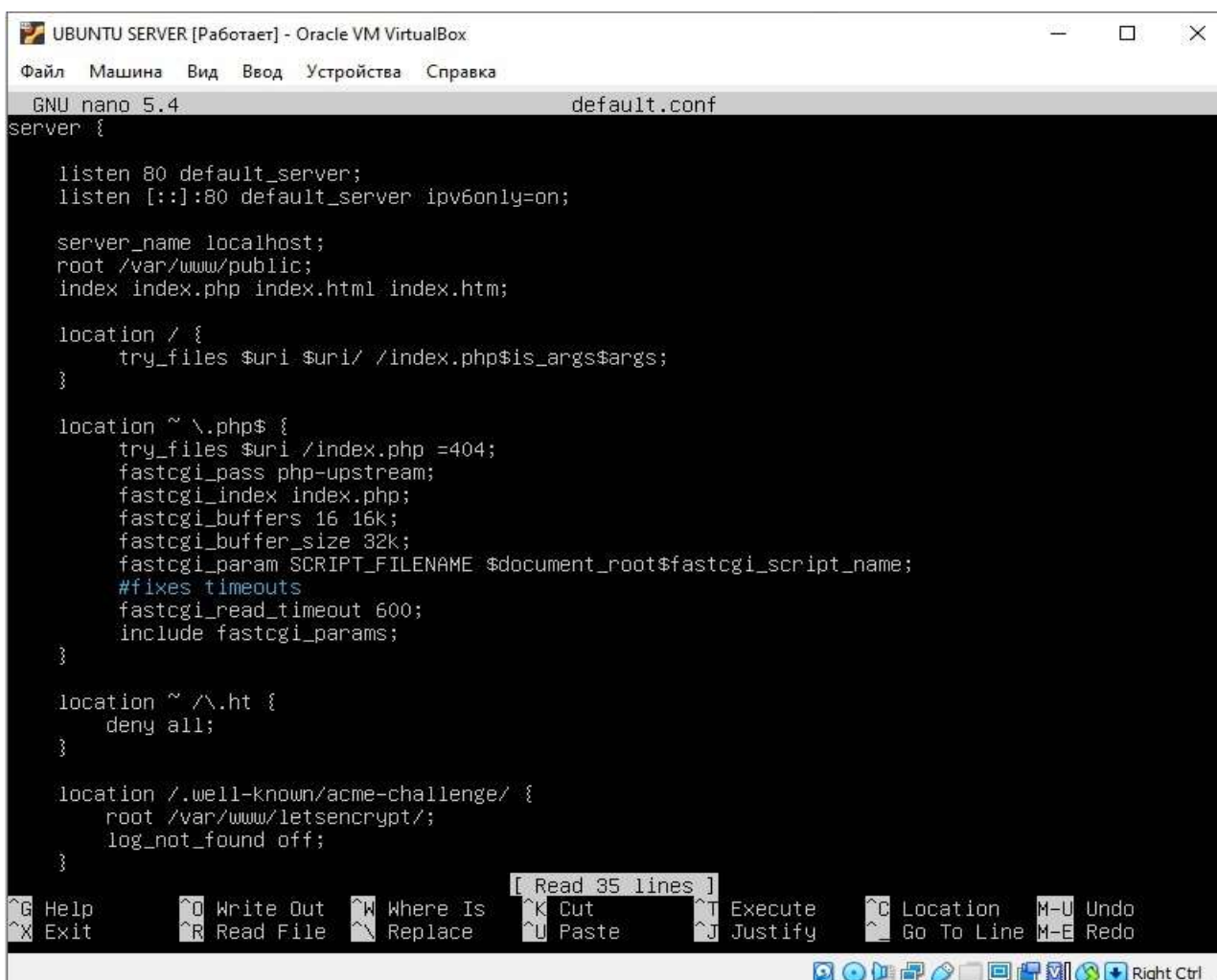
    sendfile on;

    keepalive_timeout 65;

    gzip on;

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-available/*.conf;
}
```

Рисунок 9 – содержимое файла nginx/nginx.conf



UBUNTU SERVER [Работает] - Oracle VM VirtualBox

Файл Машина Вид Ввод Устройства Справка

GNU nano 5.4 default.conf

```
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;
        #fixes timeouts
        fastcgi_read_timeout 600;
        include fastcgi_params;
    }

    location ~ /\.ht {
        deny all;
    }

    location /.well-known/acme-challenge/ {
        root /var/www/letsencrypt/;
        log_not_found off;
    }

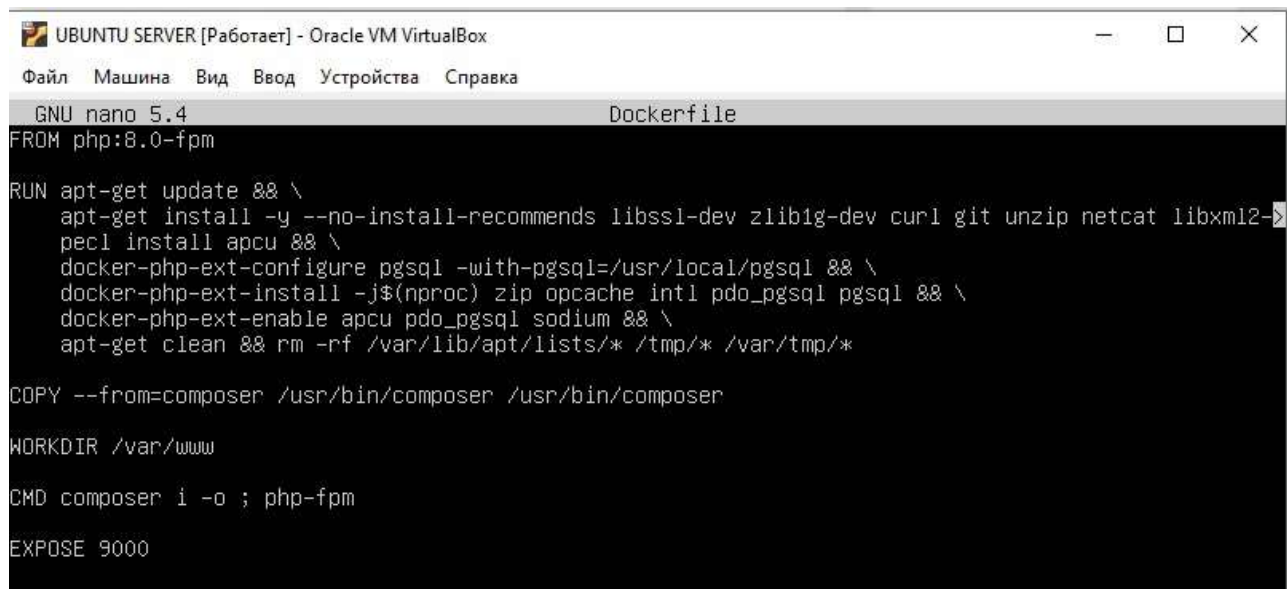
}
```

[ Read 35 lines ]

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo  
^X Exit ^R Read File ^N Replace ^U Paste ^J Justify ^\_ Go To Line M-E Redo

Right Ctrl

Рисунок 10 – содержимое файла nginx/sites/default.conf



```
UBUNTU SERVER [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
GNU nano 5.4                                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 \
    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

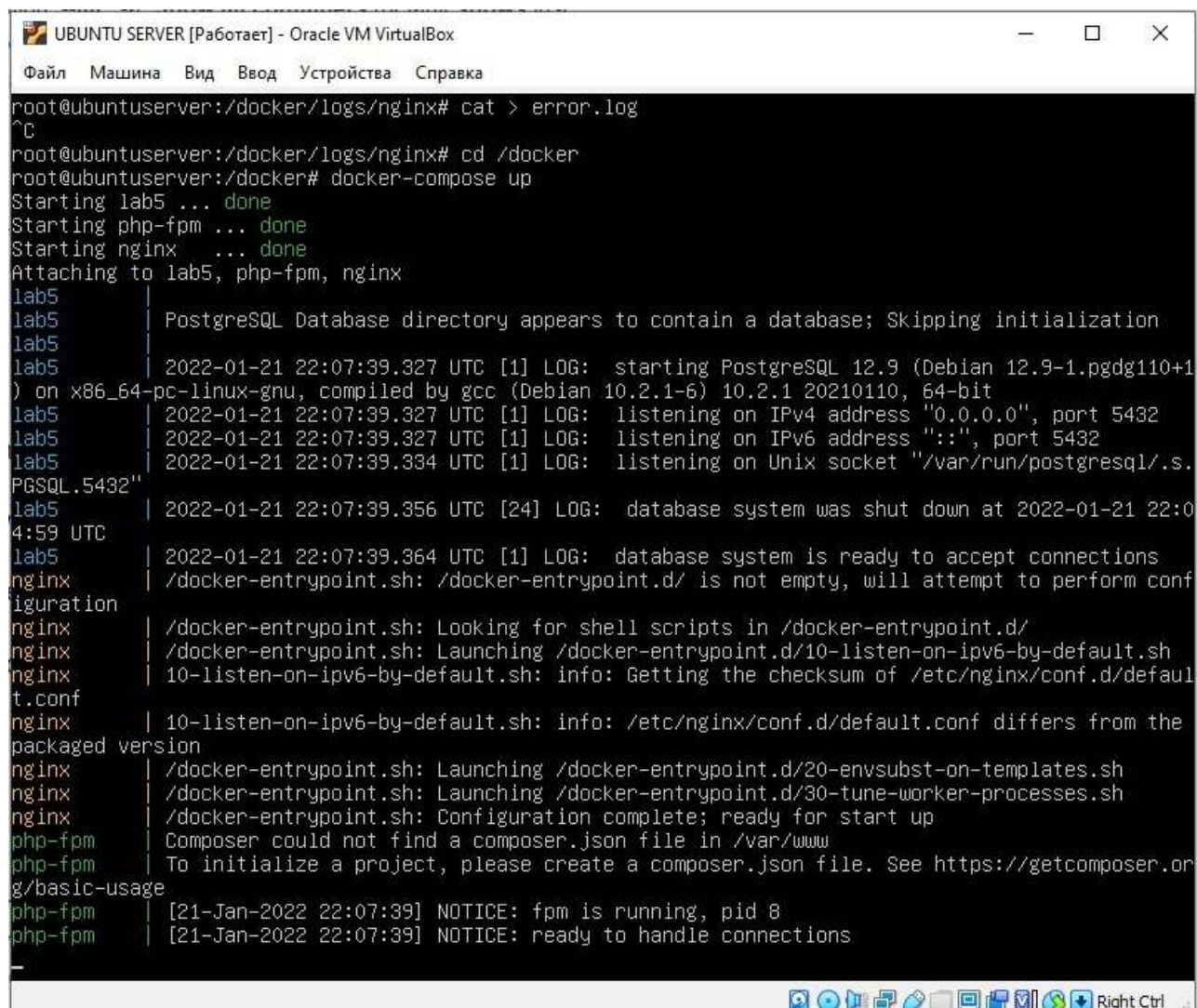
WORKDIR /var/www

CMD composer i -o ; php-fpm

EXPOSE 9000
```

Рисунок 11 – содержимое файла php-fpm/Dockerfile.

Запустим наш контейнер.



```
UBUNTU SERVER [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
root@ubuntuuserver:/docker/logs/nginx# cat > error.log
^C
root@ubuntuuserver:/docker/logs/nginx# cd /docker
root@ubuntuuserver:/docker# docker-compose up
Starting lab5 ... done
Starting php-fpm ... done
Starting nginx ... done
Attaching to lab5, php-fpm, nginx
lab5
lab5      | PostgreSQL Database directory appears to contain a database; Skipping initialization
lab5
lab5      | 2022-01-21 22:07:39.327 UTC [1] LOG:  starting PostgreSQL 12.9 (Debian 12.9-1.pgdg110+1
) on x86_64-pc-linux-gnu, compiled by gcc (Debian 10.2.1-6) 10.2.1 20210110, 64-bit
lab5      | 2022-01-21 22:07:39.327 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
lab5      | 2022-01-21 22:07:39.327 UTC [1] LOG:  listening on IPv6 address ":::", port 5432
lab5      | 2022-01-21 22:07:39.334 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.
PGSQL.5432"
lab5      | 2022-01-21 22:07:39.356 UTC [24] LOG:  database system was shut down at 2022-01-21 22:0
4:59 UTC
lab5      | 2022-01-21 22:07:39.364 UTC [1] LOG:  database system is ready to accept connections
nginx     | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform conf
figuration
nginx     | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
nginx     | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
nginx     | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default
t.conf
nginx     | 10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf differs from the
packaged version
nginx     | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
nginx     | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
nginx     | /docker-entrypoint.sh: Configuration complete; ready for start up
php-fpm   | Composer could not find a composer.json file in /var/www
php-fpm   | To initialize a project, please create a composer.json file. See https://getcomposer.or
g/basic-usage
php-fpm   | [21-Jan-2022 22:07:39] NOTICE: fpm is running, pid 8
php-fpm   | [21-Jan-2022 22:07:39] NOTICE: ready to handle connections
```

Рисунок 12 – запуск контейнера.

Откроем в текстовом браузере результат.

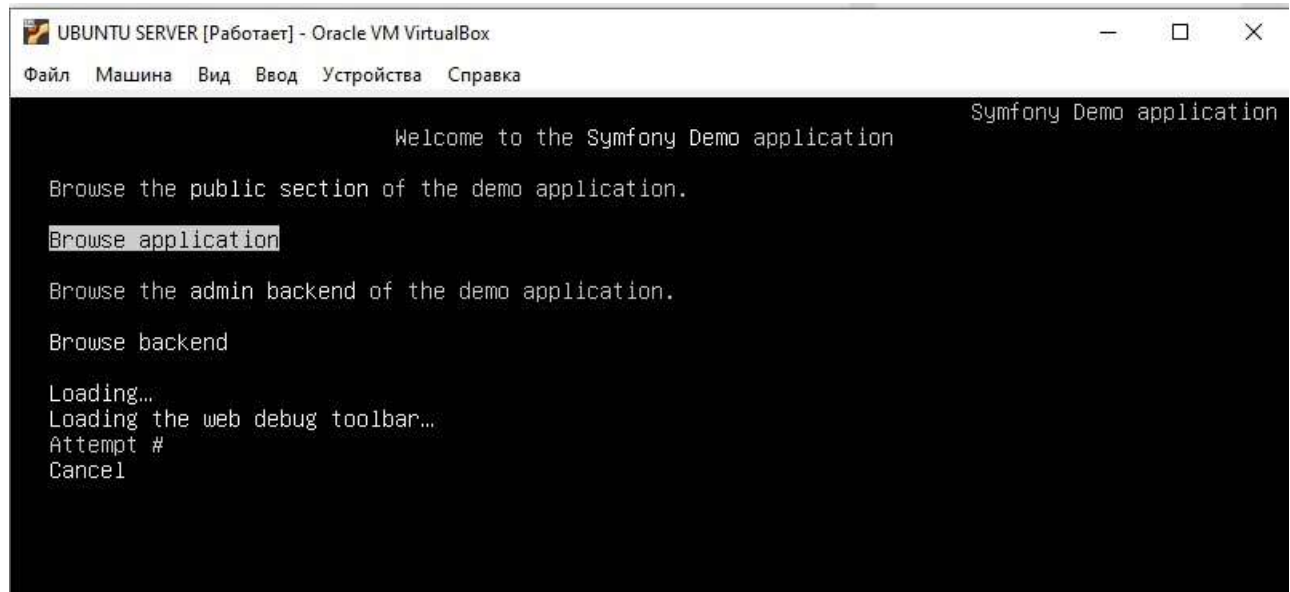


Рисунок 13 – запуск в текстовом браузере.

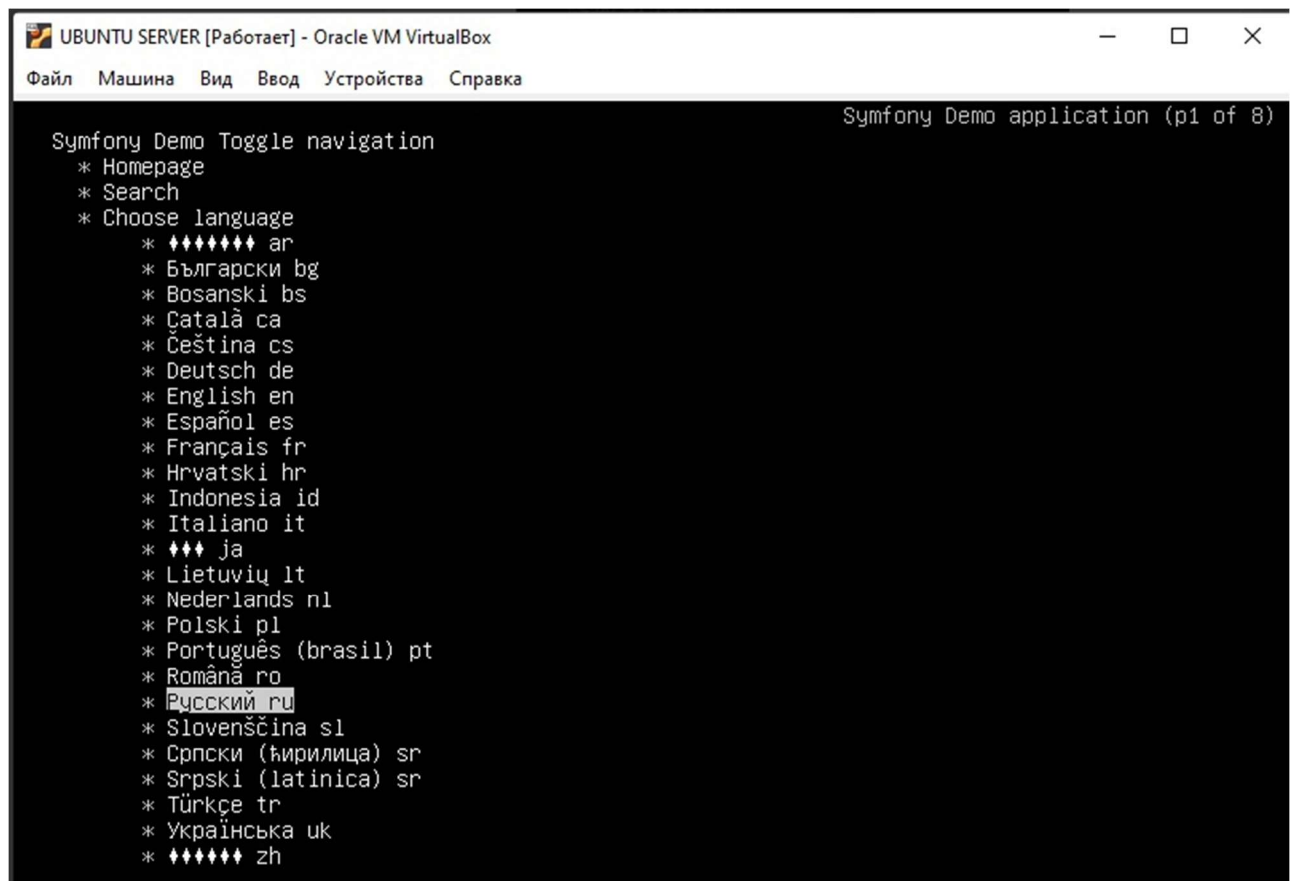


Рисунок 14 – запуск в текстовом браузере.

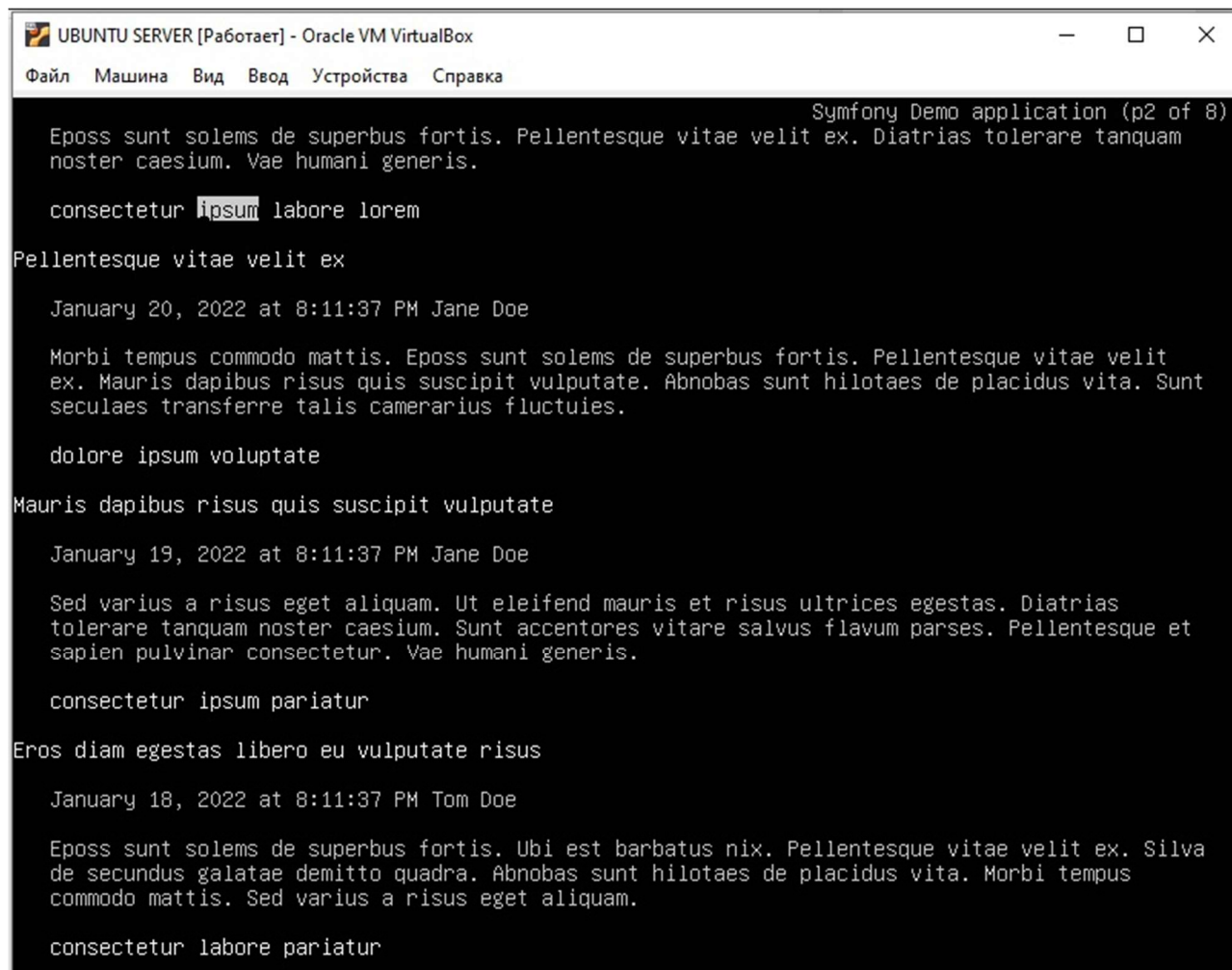
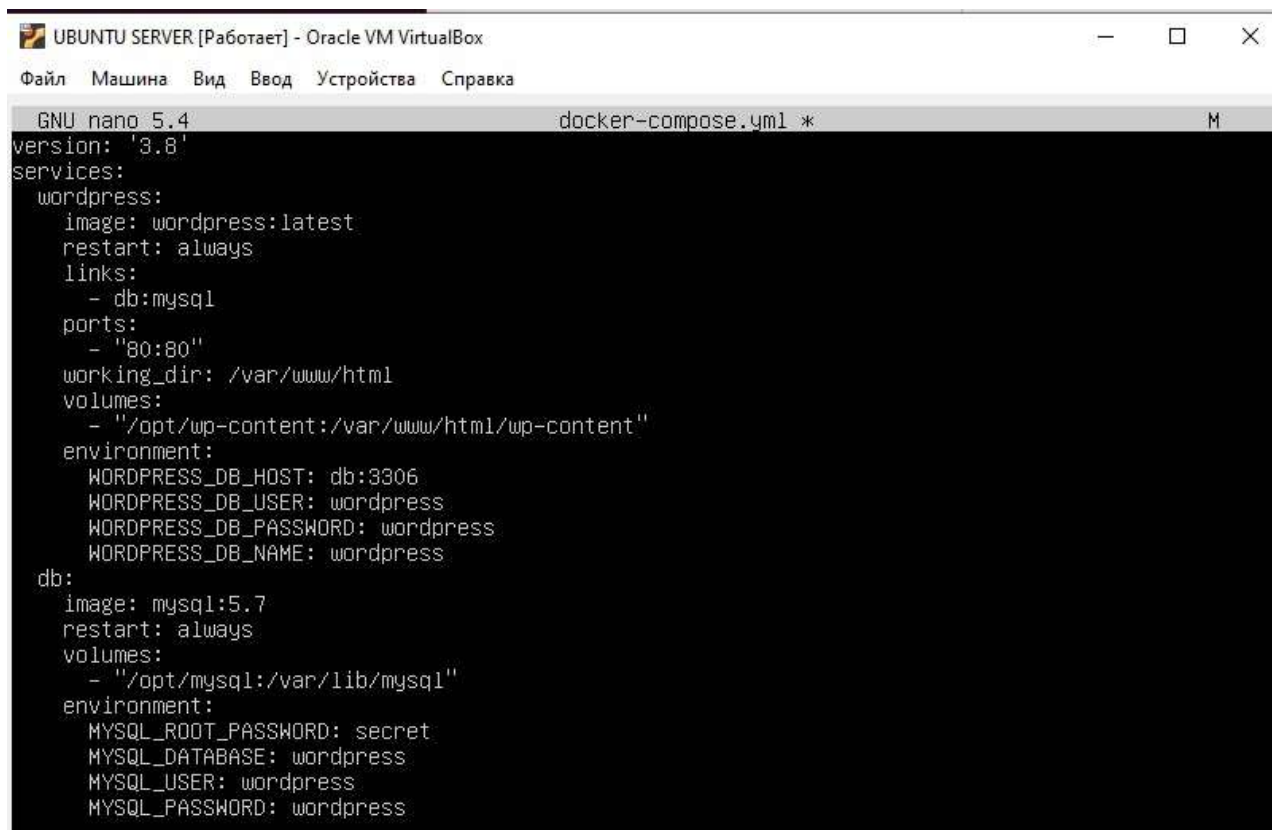


Рисунок 15 – запуск в текстовом браузере и проверка работы БД.



## 2. Работа с Wordpress

Для создания контейнера wordpress создадим папку wordpress, а в ней создадим docker-compose.yml.



```
GNU nano 5.4 docker-compose.yml *
version: '3.8'
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 16 – содержимое docker-compose.yml

```
UBUNTU SERVER [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка

a2abf6c4d29d: Already exists
c5608244554d: Already exists
2d07066487a0: Already exists
1b6dfaf1958c: Already exists
32c5e6a60073: Pull complete
90cf855b27cc: Pull complete
8b0f1068c586: Pull complete
5355461305e8: Pull complete
ad1eec592342: Pull complete
e03fbc76cb78: Pull complete
1f5796e48b39: Pull complete
72fbe8e1d4e7: Pull complete
96edece66175: Pull complete
5f46f0743de2: Pull complete
c9f9671a5e1f: Pull complete
3f543dcd35b1: Pull complete
c88e21a0c2a0: Pull complete
964b4457a910: Pull complete
1a025de137e1: Pull complete
6c7917957677: Pull complete
271be7ae0cc8: Pull complete
Digest: sha256:7c33c72f1aba217fd5554d5dbc7dbad3274962c5602da353a6196db1241cd05a
Status: Downloaded newer image for wordpress:latest
Creating wordpress_db_1 ... done
Creating wordpress_wordpress_1 ... done
root@ubuntuserver:/home/degrun/wordpress# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS
PORTS         NAMES
b1c696841eee   wordpress:latest "docker-entrypoint.s..." 15 seconds ago Up 12 seconds
0.0.0.0:80->80/tcp, :::80->80/tcp   wordpress_wordpress_1
80d2b812c756   mysql:5.7      "docker-entrypoint.s..." 16 seconds ago Up 14 seconds
3306/tcp, 33060/tcp                wordpress_db_1
71e378e5b65b   postgres:12    "docker-entrypoint.s..." 4 days ago     Up About an hour
0.0.0.0:15432->5432/tcp, :::15432->5432/tcp   lab5
634a24e71512   demo_app       "docker-php-entrypoi..." 12 days ago    Restarting (255) 43 seconds ago
                                docker-node-mongo
root@ubuntuserver:/home/degrun/wordpress#
```

Рисунок 17 – сборка контейнера.

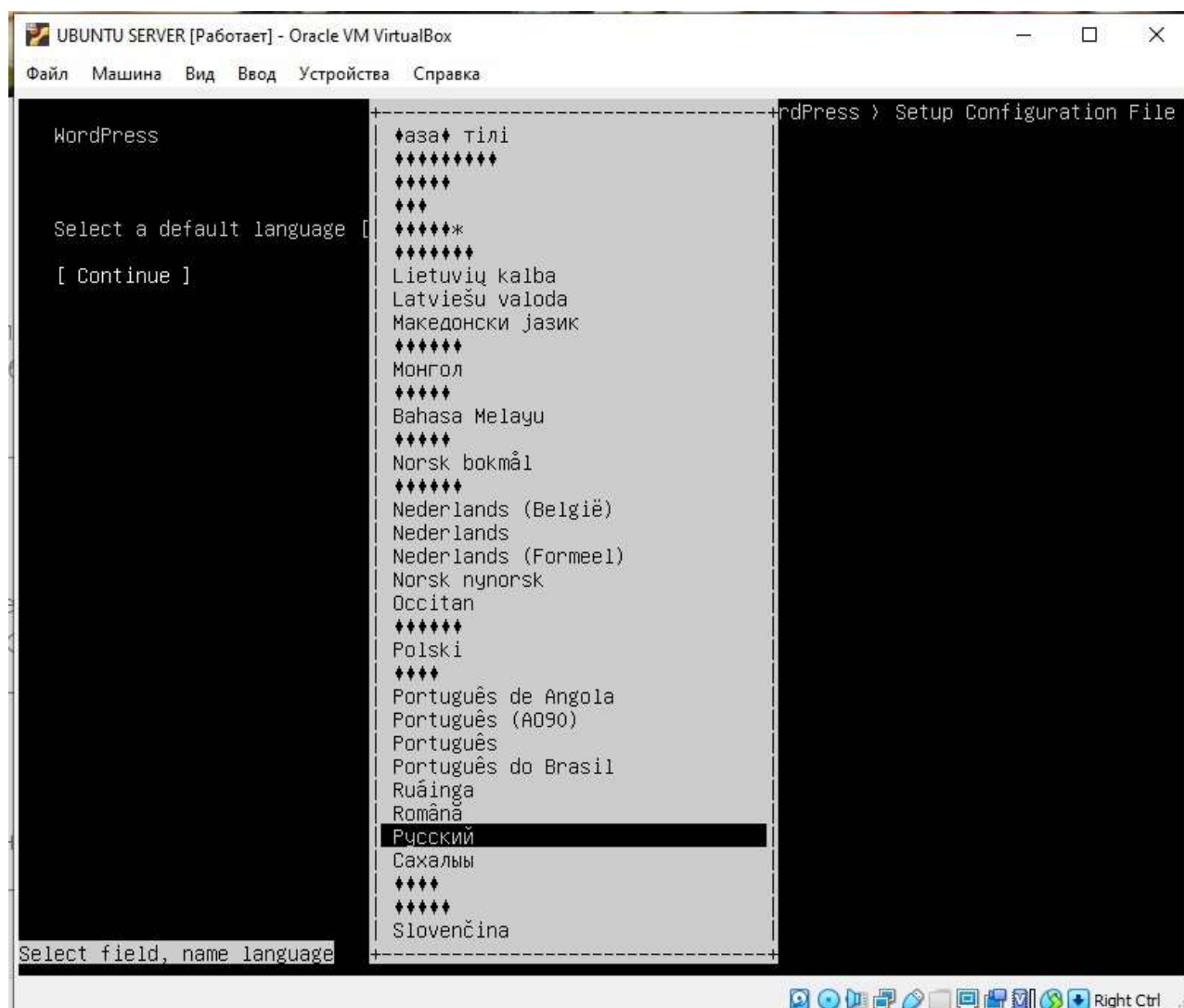


Рисунок 18 – пример работы wordpress.



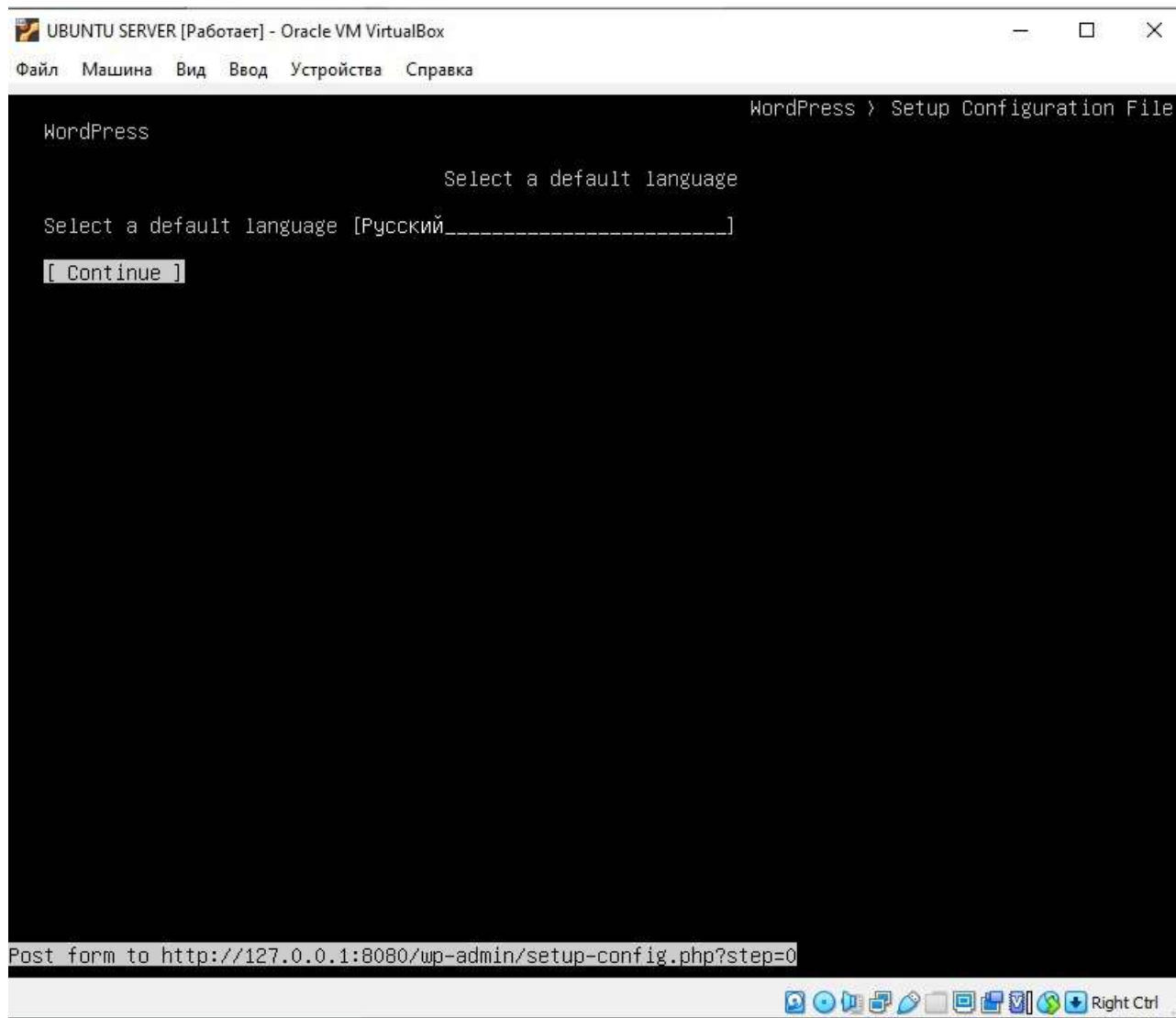


Рисунок 19 – пример работы wordpress.

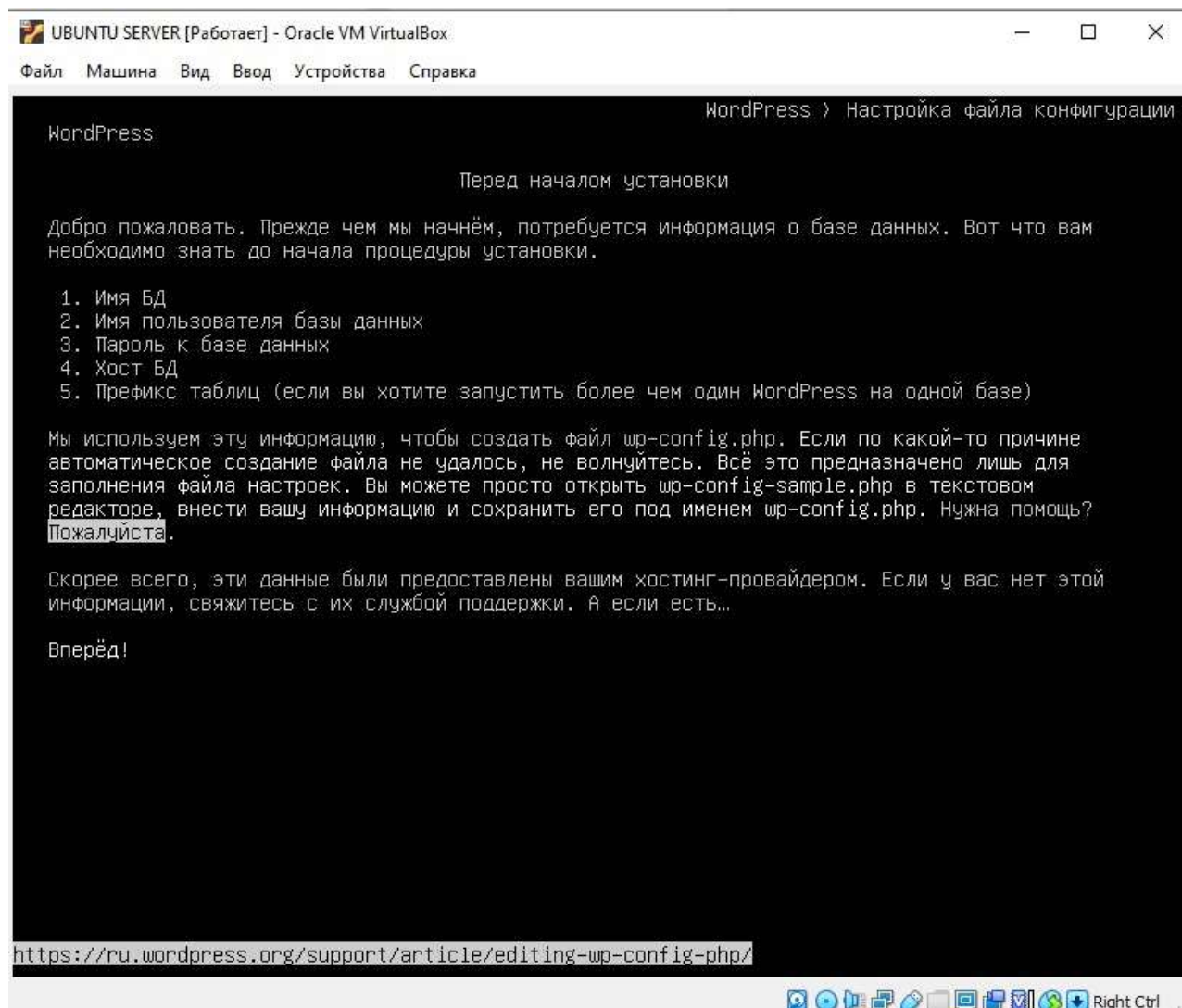


Рисунок 20 – пример работы wordpress.

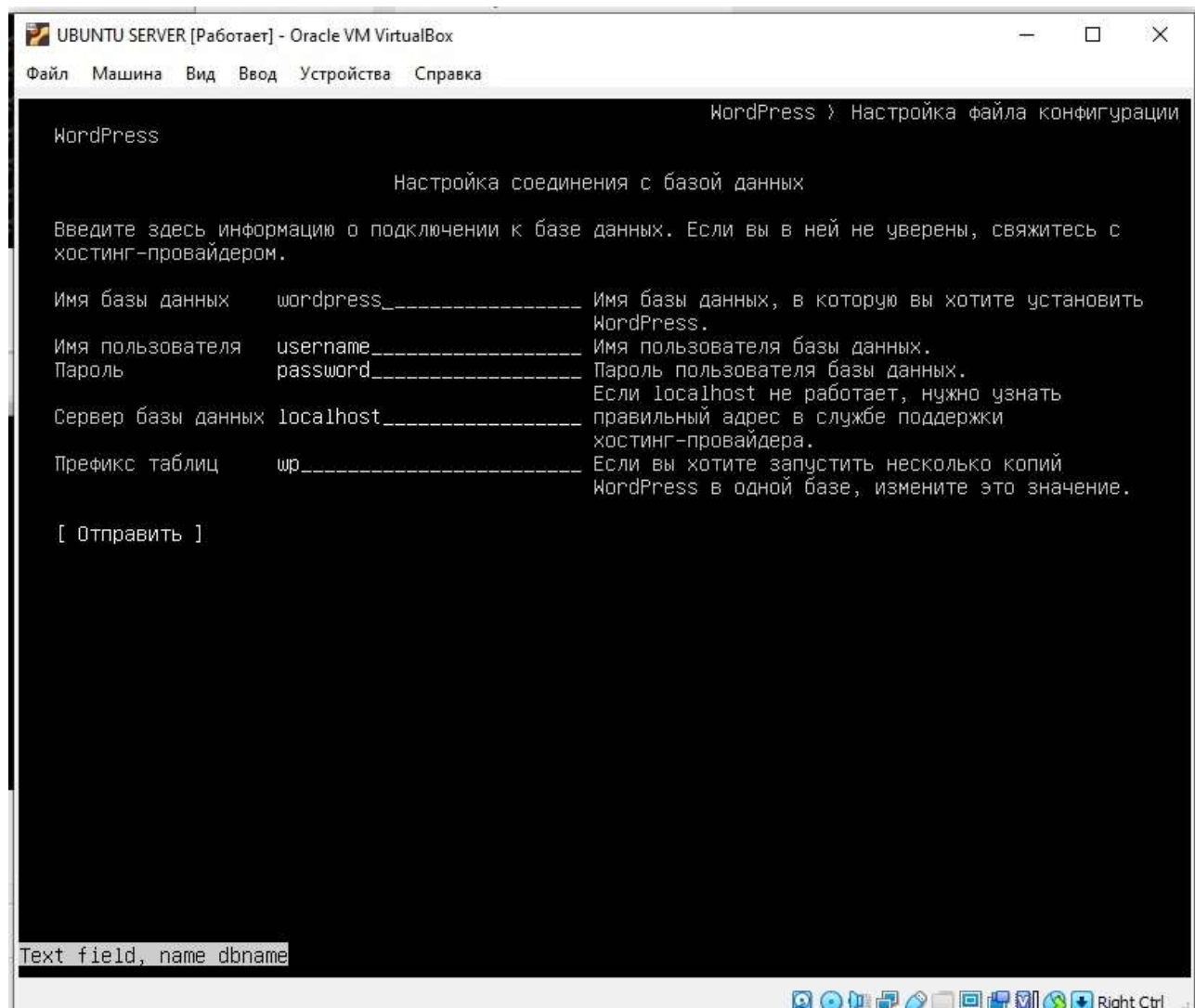


Рисунок 21 – пример работы wordpress.

## Вывод

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

## Контрольные вопросы

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.

- 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.
- 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