

Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №5

по дисциплине «Операционная система Linux»

Контейнеризация

Студент

Фетисов В. Д.

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	26

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@vlad:/home/vlad/demo# git clone https://github.com/symfony/demo
Cloning into 'demo'...
remote: Enumerating objects: 10583, done.
remote: Counting objects: 100% (708/708), done.
remote: Compressing objects: 100% (432/432), done.
remote: Total 10583 (delta 366), reused 525 (delta 248), pack-reused 9875
Receiving objects: 100% (10583/10583), 19.06 MiB | 7.06 MiB/s, done.
Resolving deltas: 100% (6307/6307), done.
root@vlad:/home/vlad/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.

```
DockerNew (Тут есть вордпресс :) [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
[Application] Jan 26 18:22:00 |DEBUG | SECURI Checking for authenticator support. authenticators=1
firewall_name="main"
[Application] Jan 26 18:22:00 |DEBUG | SECURI Checking support on authenticator. authenticator="Sym
fony\\Component\\Security\\Http\\Authenticator\\FormLoginAuthenticator"
[Application] Jan 26 18:22:00 |DEBUG | SECURI Authenticator does not support the request.
[Application] Jan 26 18:22:00 |DEBUG | DOCTRI SELECT DISTINCT s0_.id AS id_0, s0_.published_at AS p
ublished_at_1 FROM symfony_demo_post s0_ INNER JOIN symfony_demo_user s1_ ON s0_.author_id = s1_.id
LEFT JOIN symfony_demo_post_tag s3_ ON s0_.id = s3_.post_id LEFT JOIN symfony_demo_tag s2_ ON s2_.id
= s3_.tag_id WHERE s0_.published_at <= ? ORDER BY s0_.published_at DESC LIMIT 10 0="2022-01-26T18:2
2:00+00:00"
[Application] Jan 26 18:22:00 |DEBUG | DOCTRI SELECT s0_.id AS id_0, s0_.title AS title_1, s0_.slug
AS slug_2, s0_.summary AS summary_3, s0_.content AS content_4, s0_.published_at AS published_at_5,
s1_.id AS id_6, s1_.full_name AS full_name_7, s1_.username AS username_8, s1_.email AS email_9, s1_.
password AS password_10, s1_.roles AS roles_11, s2_.id AS id_12, s2_.name AS name_13, s0_.author_id
AS author_id_14 FROM symfony_demo_post s0_ INNER JOIN symfony_demo_user s1_ ON s0_.author_id = s1_.i
d LEFT JOIN symfony_demo_post_tag s3_ ON s0_.id = s3_.post_id LEFT JOIN symfony_demo_tag s2_ ON s2_.
id = s3_.tag_id WHERE s0_.published_at <= ? AND s0_.id IN (?) ORDER BY s0_.published_at DESC, s2_.na
me ASC 1=[1,2,3,4,5,6,7,8,9,10]
[Application] Jan 26 18:22:00 |DEBUG | DOCTRI SELECT count(DISTINCT s0_.id) AS sclr_0 FROM symfony_
demo_post s0_ INNER JOIN symfony_demo_user s1_ ON s0_.author_id = s1_.id LEFT JOIN symfony_demo_post
_tag s3_ ON s0_.id = s3_.post_id LEFT JOIN symfony_demo_tag s2_ ON s2_.id = s3_.tag_id WHERE s0_.pub
lished_at <= ?
[Application] Jan 26 18:22:00 |DEBUG | DOCTRI SELECT quote_ident(table_name) AS table_name,
table_schema AS schema_name FROM information_schema.tables
WHERE table_schema NOT LIKE 'pg\\_%' AND table_schema != 'information_sc
hema' AND table_name != 'geometry_columns' AND table_name != '
spatial_ref_sys' AND table_type != 'VIEW'
[Application] Jan 26 18:22:00 |DEBUG | DOCTRI SELECT schema_name FROM information_schema.schemata
WHERE schema_name NOT LIKE 'pg\\_%' AND schema_name != 'information_schema'
[Application] Jan 26 18:22:00 |DEBUG | DOCTRI SHOW search_path
[Application] Jan 26 18:22:00 |DEBUG | DOCTRI SELECT CURRENT_DATABASE()
[Web Server ] Jan 26 21:34:07 |INFO | PHP listening path="/usr/sbin/php-fpm8.0" php="8.0.14" po
rt=38797
[PHP-FPM ] Jan 26 21:34:07 |NOTICE | FPM fpm is running, pid 52821
[PHP-FPM ] Jan 26 21:34:07 |NOTICE | FPM ready to handle connections
[PHP-FPM ] Jan 26 21:34:07 |NOTICE | FPM systemd monitor interval set to 10000ms
```

Рисунок 2 – Запуск проекта

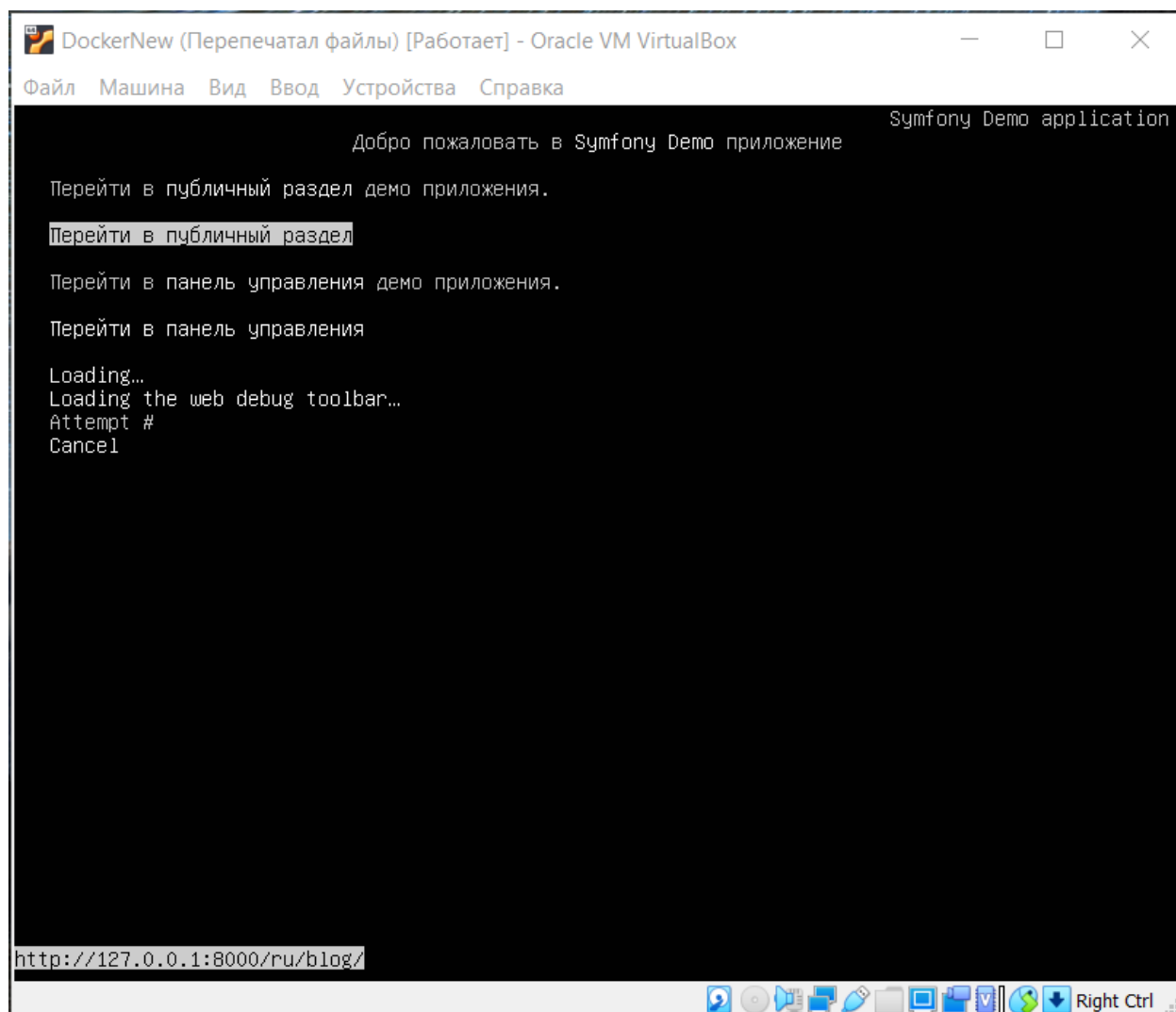
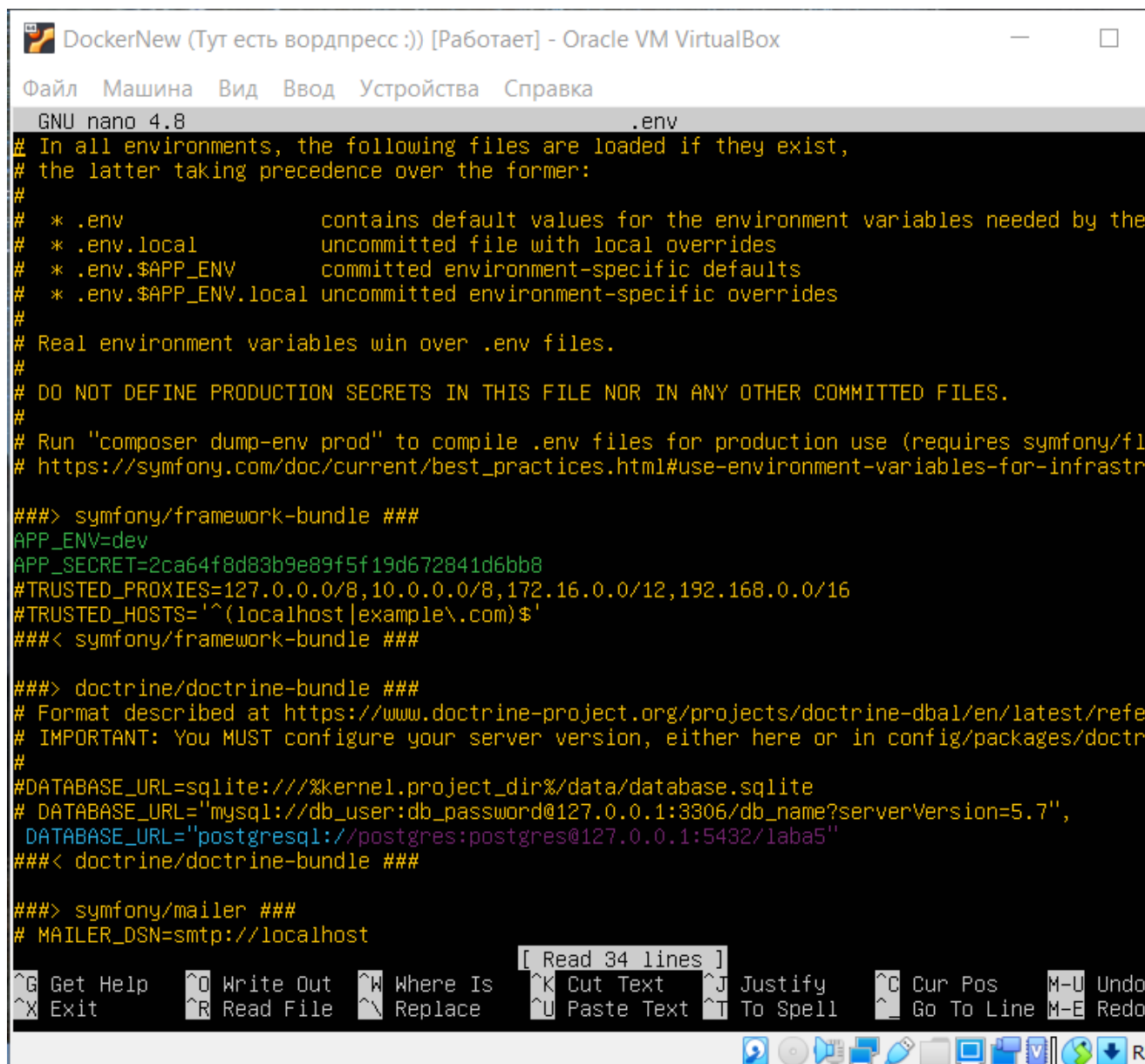


Рисунок 3 – Запуск проекта



```
DockerNew (Тут есть вордпресс :)) [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
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
# * .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/fi
# https://symfony.com/doc/current/best_practices.html#use-environment-variables-for-infrastr

###> 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/refe
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctr
#
#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:postgres@127.0.0.1:5432/laba5"
###< doctrine/doctrine-bundle ###

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

[ Read 34 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
```

Создадим БД для тестирования проекта

Для начала, изменим СУБД в файле .env

Рисунок 4 – Содержимое .env

List of relations			
Schema	Name	Type	Owner
public	symfony_demo_comment	table	postgres
public	symfony_demo_comment_id_seq	sequence	postgres
public	symfony_demo_post	table	postgres
public	symfony_demo_post_id_seq	sequence	postgres
public	symfony_demo_post_tag	table	postgres
public	symfony_demo_tag	table	postgres
public	symfony_demo_tag_id_seq	sequence	postgres
public	symfony_demo_user	table	postgres
public	symfony_demo_user_id_seq	sequence	postgres
(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.

DockerNew (Тут есть вордпресс :)) [Работает] - Oracle VM VirtualBox

Файл Машина Вид Ввод Устройства Справка

GNU nano 4.8 docker-compose.yml

```
version: '3.8'

services:
  db:
    container_name: db
    image: postgres:12
    restart: always
    environment:
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: laba5
    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
```

[Read 42 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

Right Ctrl

Рисунок 6 – содержимое файла docker-compose.yml.

The screenshot shows a window titled "DockerNew (Тут есть вордпресс :)) [Работает] - Oracle VM VirtualBox". Inside, the GNU nano 4.8 editor is open to the file "docker-compose.yml". The file contains the following configuration:

```

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"
  
```

At the bottom, there is a menu bar with options like 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, and a system tray with icons and a "Right Ctrl" button.

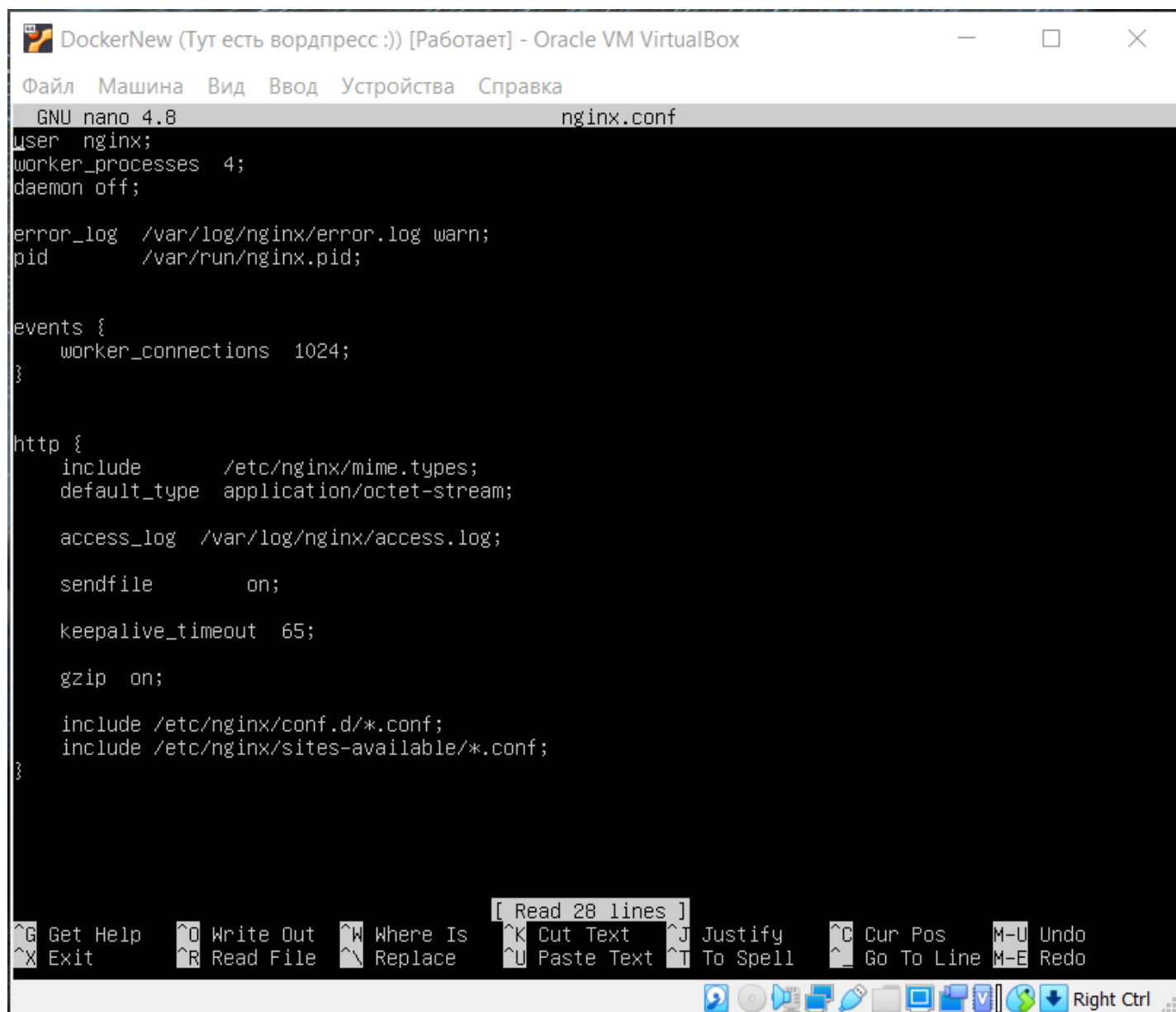
Рисунок 7 – содержимое файла docker-compose.yml.

The screenshot shows a window titled "DockerNew (Тут есть вордпресс :)) [Работает] - Oracle VM VirtualBox". Inside, the GNU nano 4.8 editor is open to the file "Dockerfile". The file contains the following configuration:

```

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

Рисунок 8 – содержимое файла nginx/Dockerfile



The screenshot shows a terminal window titled "DockerNew (Тут есть вордпресс :)) [Работает] - Oracle VM VirtualBox". Inside the terminal, the GNU nano 4.8 editor is open, displaying the contents of the file `nginx.conf`. The configuration includes settings for the `nginx` user, worker processes, daemon mode, error logs, pid file, events (worker connections), and the `http` block with various directives like `include`, `default_type`, `access_log`, `sendfile`, `keepalive_timeout`, `gzip`, and `include` for configuration and site files. The bottom of the terminal shows a status bar with various keyboard shortcuts and a message "[Read 28 lines]".

```
GNU nano 4.8 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;

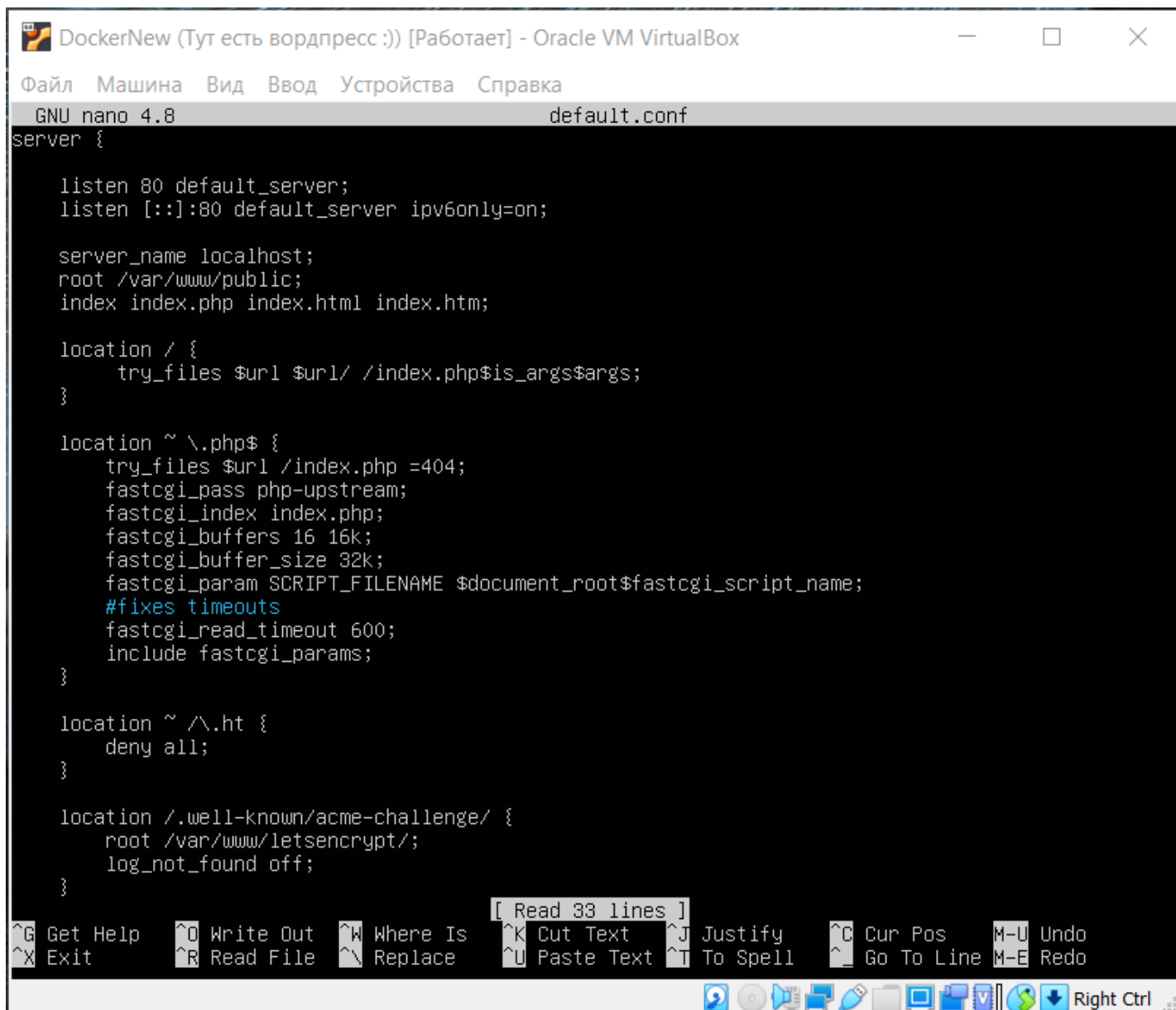
    gzip on;

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

[Read 28 lines]

Get Help Write Out Where Is Cut Text Justify Cur Pos Undo
Exit Read File Replace Paste Text To Spell Go To Line Redo

Рисунок 9 – содержимое файла `nginx/nginx.conf`



DockerNew (Тут есть вордпресс :)) [Работает] - Oracle VM VirtualBox

Файл Машина Вид Ввод Устройства Справка

GNU nano 4.8 default.conf

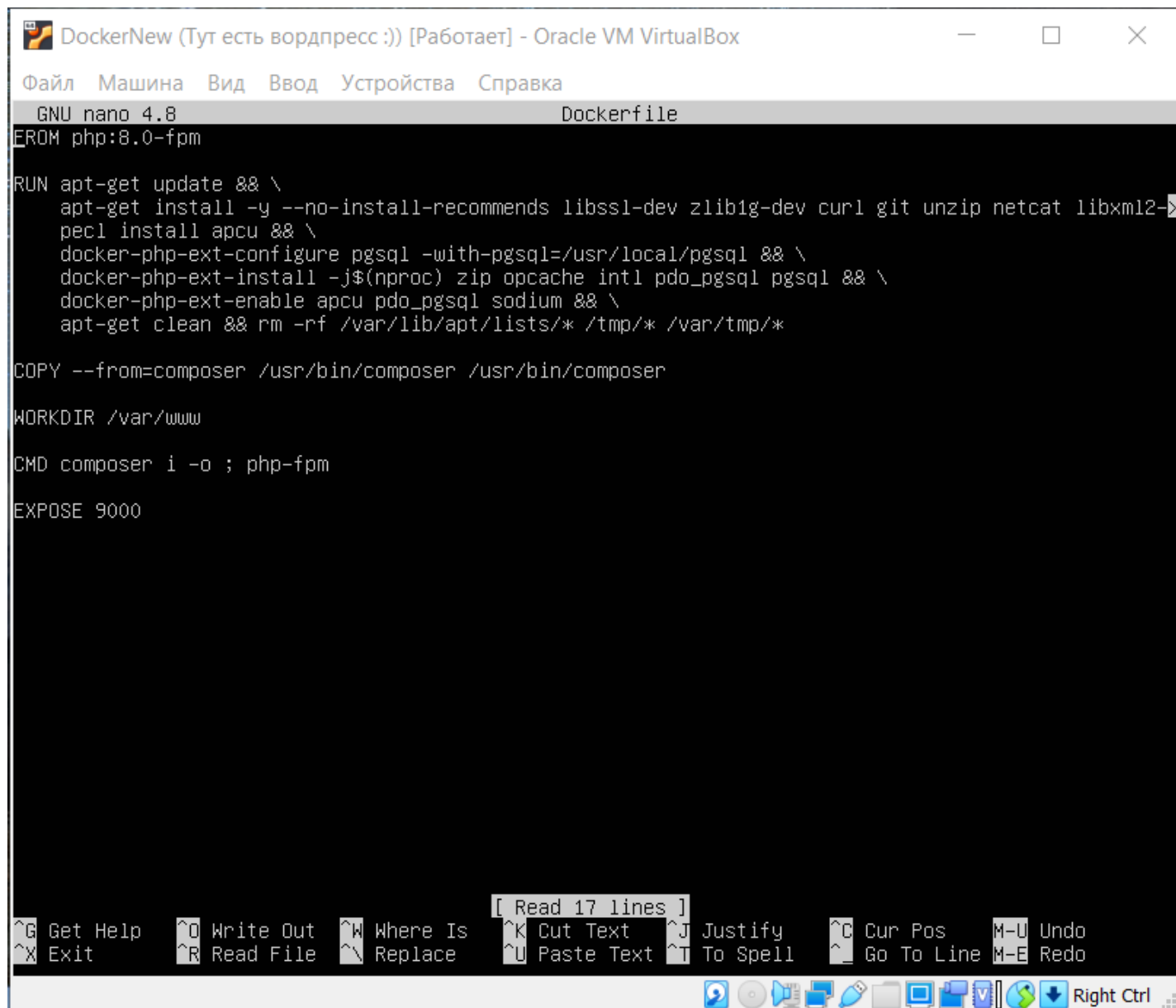
```
server {  
  
    listen 80 default_server;  
    listen [::]:80 default_server ipv6only=on;  
  
    server_name localhost;  
    root /var/www/public;  
    index index.php index.html index.htm;  
  
    location / {  
        try_files $url $url/ /index.php$is_args$args;  
    }  
  
    location ~ \.php$ {  
        try_files $url /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 33 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

Right Ctrl

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



The screenshot shows a nano editor window with the following content:

```
GNU nano 4.8 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
```

The bottom of the window features a nano editor status bar with the following options:

Get Help	Write Out	Where Is	Cut Text	Justify	Cur Pos	Undo
Exit	Read File	Replace	Paste Text	To Spell	Go To Line	Redo

A message "[Read 17 lines]" is displayed above the status bar. The window title is "DockerNew (Тут есть вордпресс :)) [Работает] - Oracle VM VirtualBox".

Рисунок 11 – содержимое файла php-fpm/Dockerfile.

Запустим наш контейнер.

```
DockerNew (Перепечатал файлы) [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
db |
db | waiting for server to start....2022-01-26 17:49:28.510 UTC [48] 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
db | 2022-01-26 17:49:28.513 UTC [48] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
db | 2022-01-26 17:49:28.525 UTC [49] LOG:  database system was shut down at 2022-01-26 17:49:28 UTC
db | 2022-01-26 17:49:28.529 UTC [48] LOG:  database system is ready to accept connections
db | done
db | server started
db | CREATE DATABASE
db |
db | /usr/local/bin/docker-entrypoint.sh: running /docker-entrypoint-initdb.d/laba_db_backup.sql
db | psql:/docker-entrypoint-initdb.d/laba_db_backup.sql: error: could not read from input file: Is a directory
db |
db | PostgreSQL Database directory appears to contain a database; Skipping initialization
db |
db | 2022-01-26 17:49:29.236 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
db | 2022-01-26 17:49:29.239 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
db | 2022-01-26 17:49:29.240 UTC [1] LOG:  listening on IPv6 address ":::", port 5432
db | 2022-01-26 17:49:29.243 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
db | 2022-01-26 17:49:29.255 UTC [25] LOG:  database system was interrupted; last known up at 2022-01-26 17:49:28 UTC
db | 2022-01-26 17:49:29.495 UTC [25] LOG:  database system was not properly shut down; automatic recovery in progress
db | 2022-01-26 17:49:29.498 UTC [25] LOG:  redo starts at 0/164E4E8
db | 2022-01-26 17:49:29.499 UTC [25] LOG:  invalid record length at 0/164E5E0: wanted 24, got 0
db | 2022-01-26 17:49:29.500 UTC [25] LOG:  redo done at 0/164E598
db | 2022-01-26 17:49:29.509 UTC [1] LOG:  database system is ready to accept connections
-
```

Рисунок 12 – запуск контейнера.

Откроем в текстовом браузере результат.

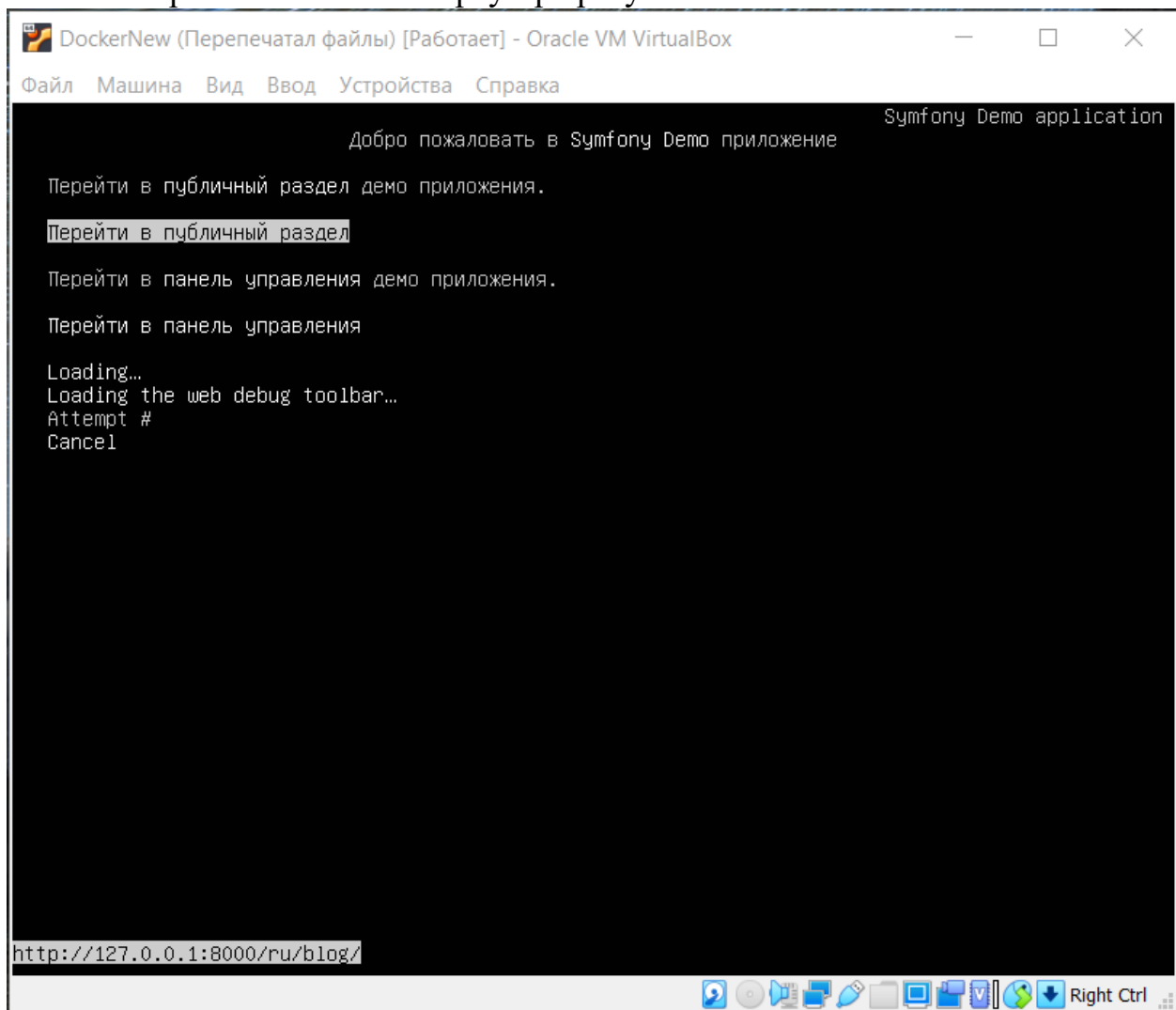


Рисунок 13 – запуск в текстовом браузере.

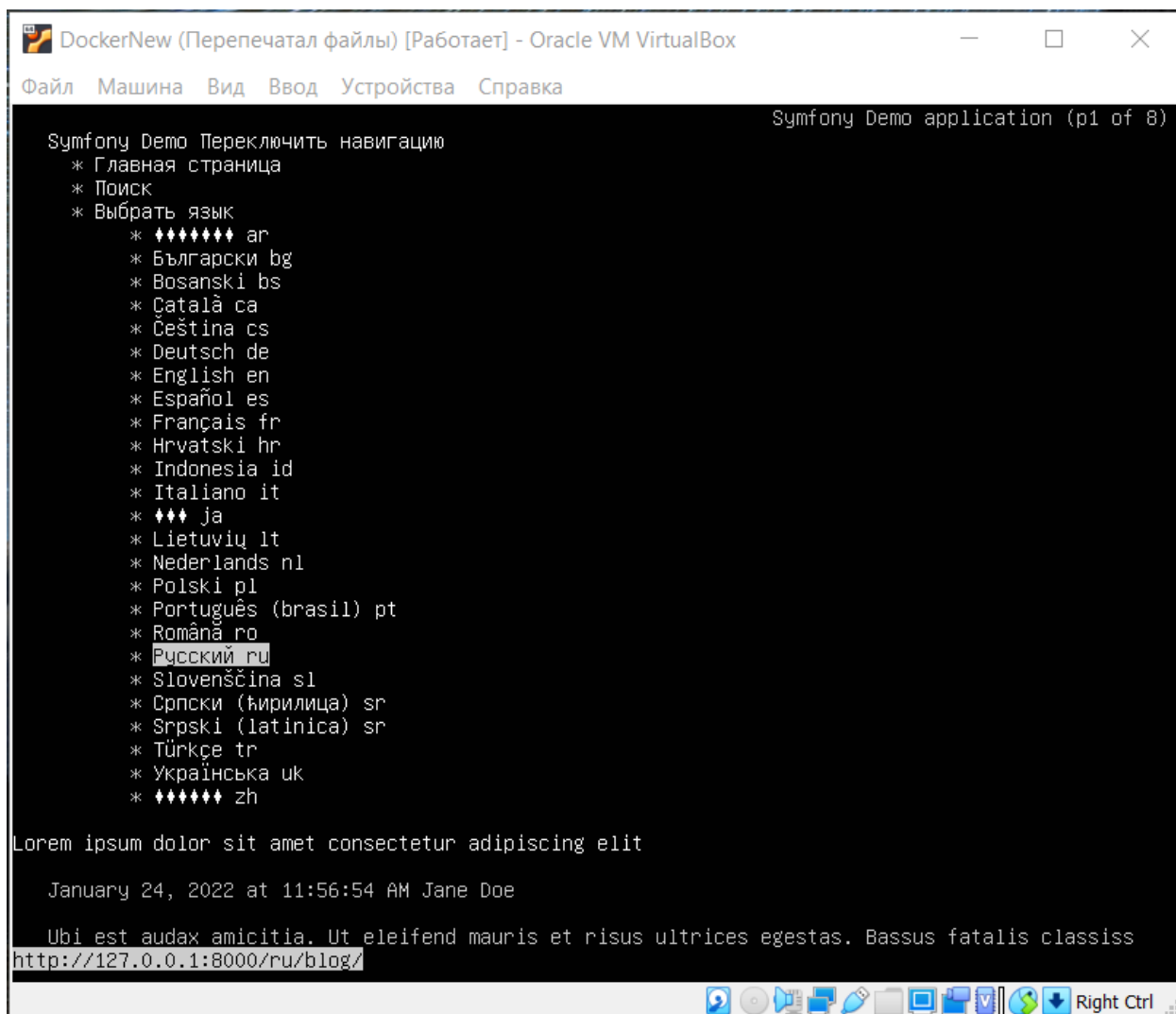


Рисунок 14 – запуск в текстовом браузере.

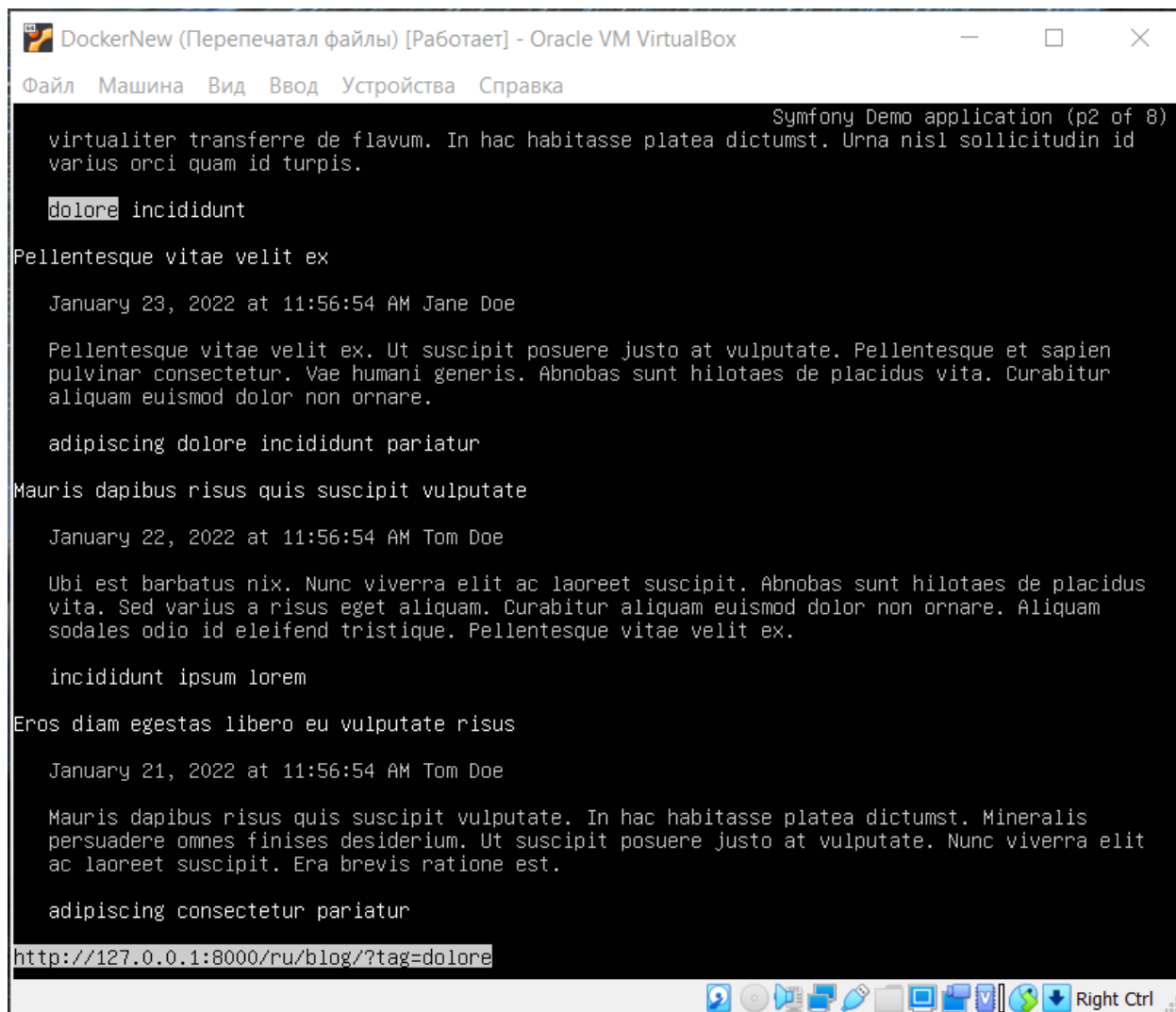
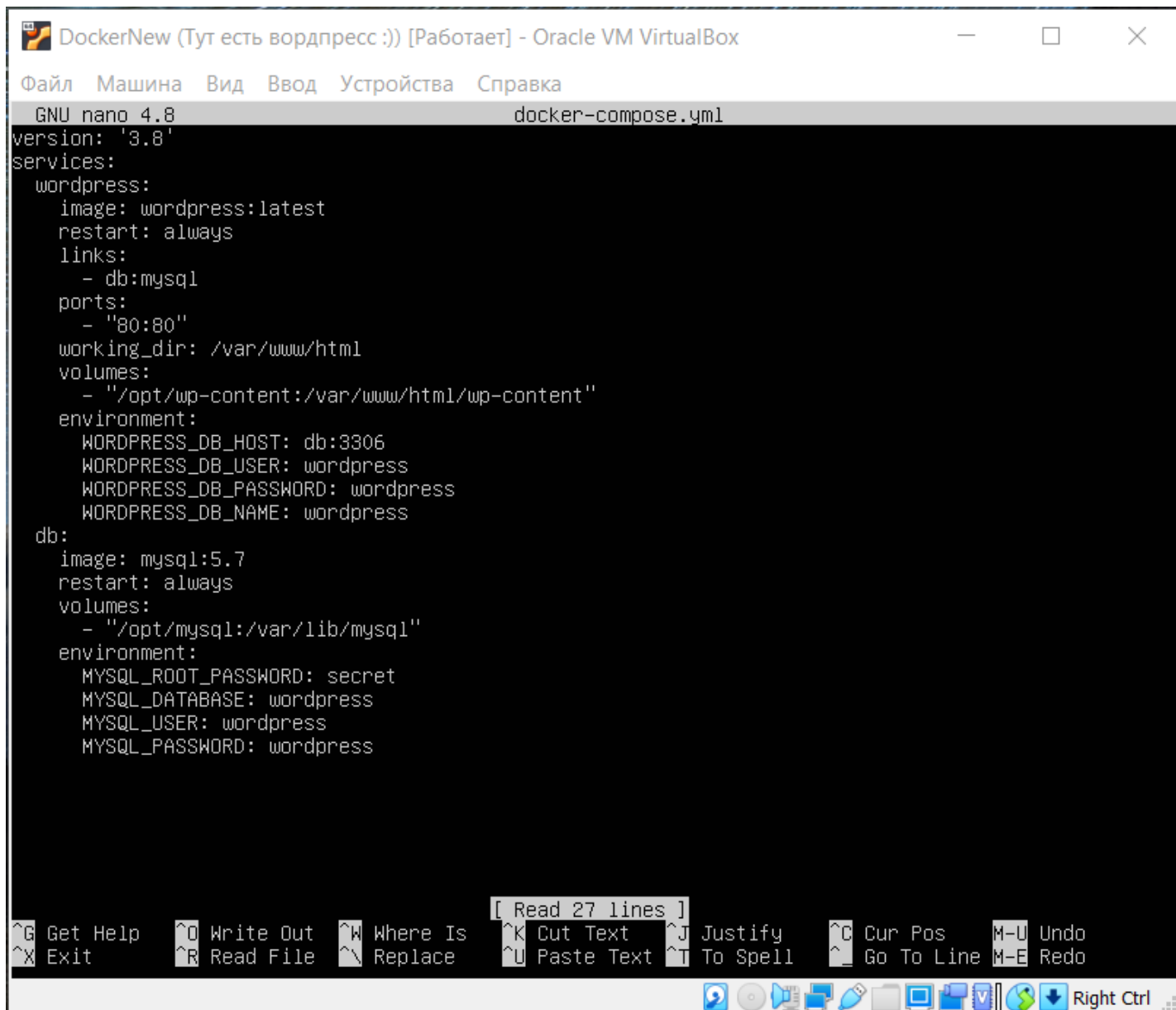


Рисунок 15 – запуск в текстовом браузере и проверка работы БД.

2. Работа с Wordpress

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



```
GNU nano 4.8 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
```

[Read 27 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 16 – содержимое docker-compose.yml

```

root@vlad:/home/vlad/wordpress# docker-compose up -d --build
Starting wordpress_db_1 ... done
Starting wordpress_wordpress_1 ... done
root@vlad:/home/vlad/wordpress# docker ps
CONTAINER ID   IMAGE          NAMES                  COMMAND                  CREATED        STATUS        PORTS
7a277c5f6660   wordpress:latest   "docker-entrypoint.s..."   22 minutes ago   Up 2 seconds   0.0.0.0:80->80/tcp, :::80->80/tcp   wordpress_wordpress_1
a734eb1867b6   mysql:5.7         "docker-entrypoint.s..."   31 minutes ago   Up 2 seconds   3306/tcp, 33060/tcp               wordpress_db_1
root@vlad:/home/vlad/wordpress#

```

Рисунок 17 – сборка контейнера.

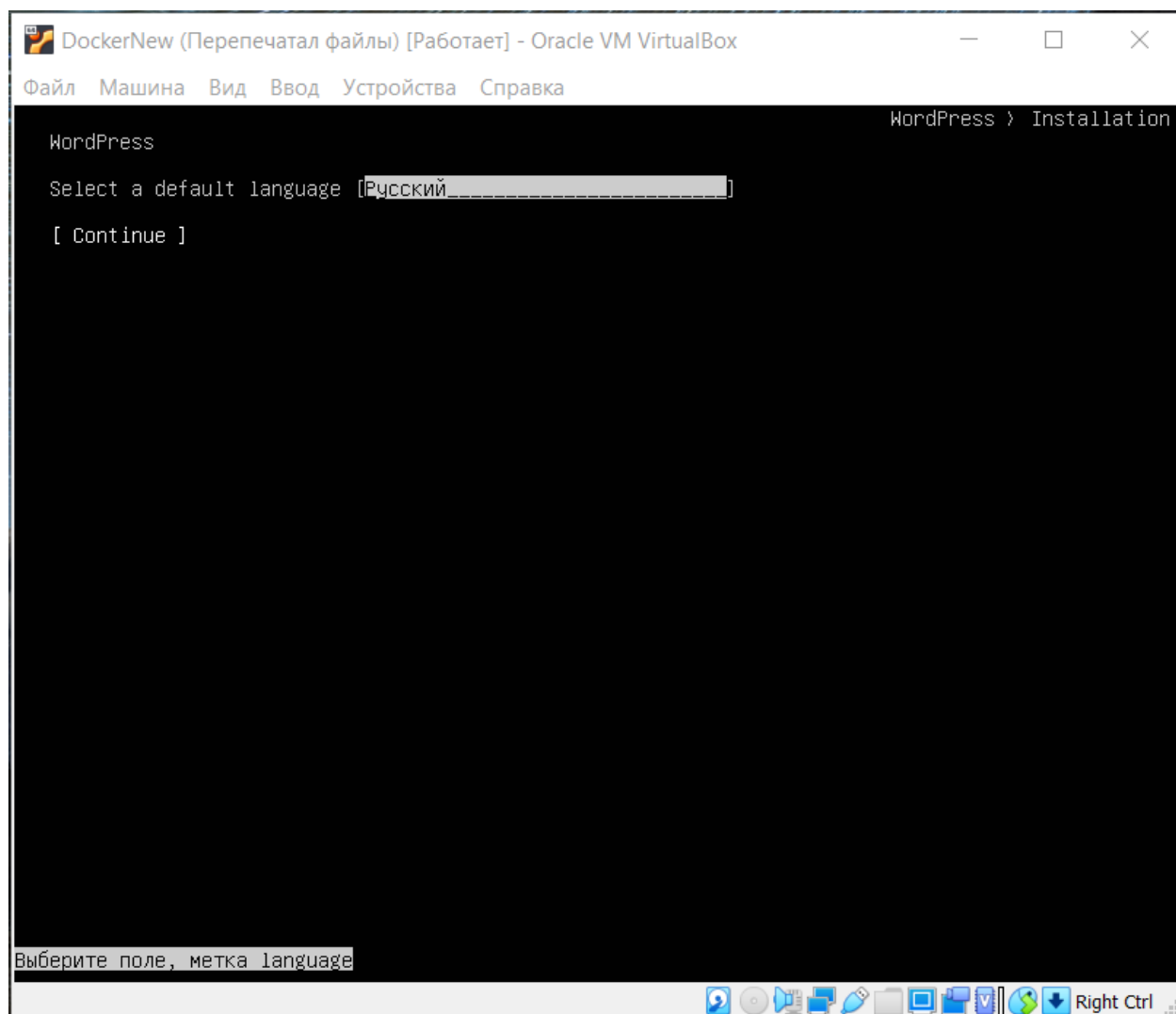


Рисунок 18 – пример работы wordpress.

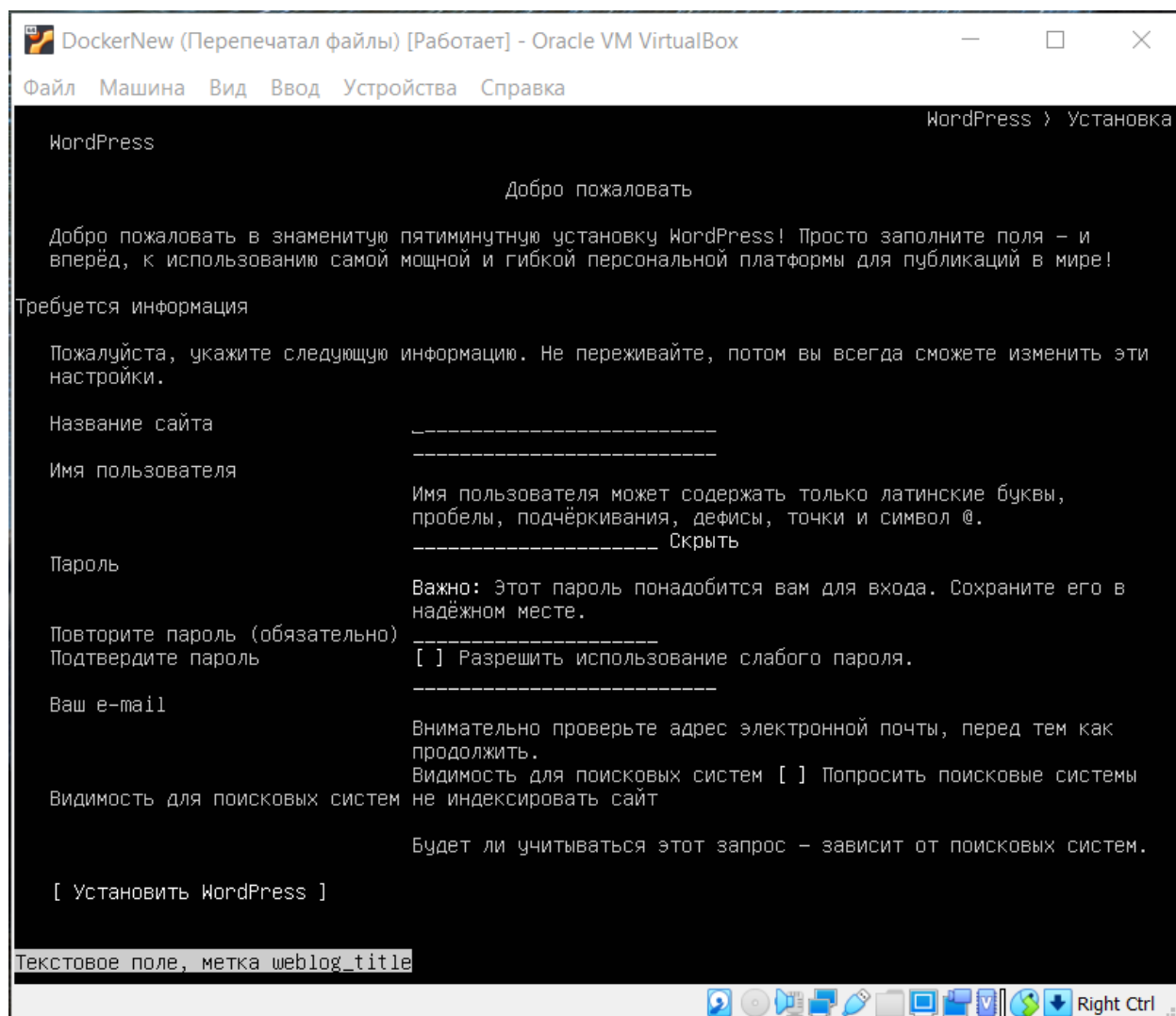


Рисунок 19 – пример работы wordpress.

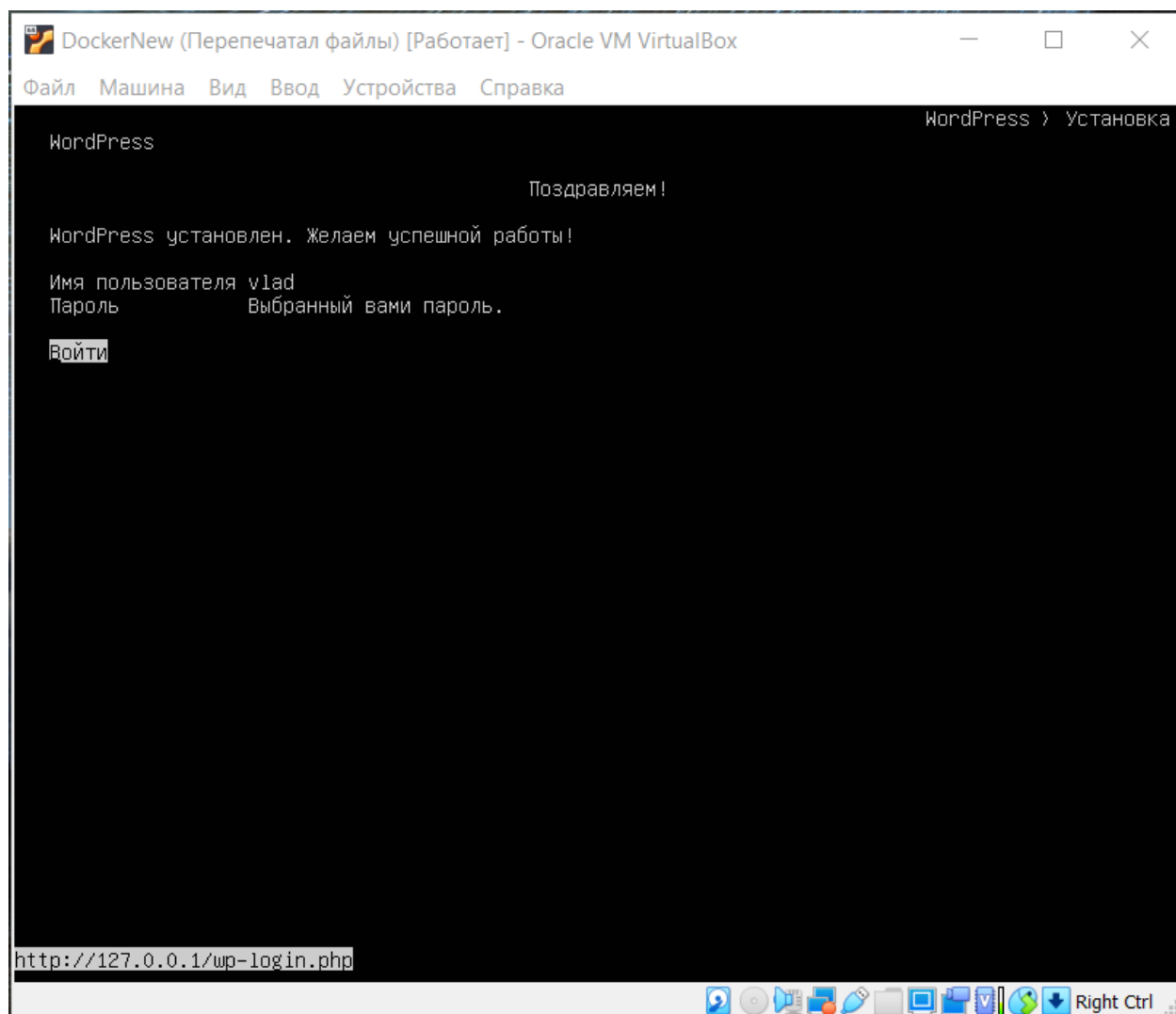


Рисунок 20 – пример работы wordpress.

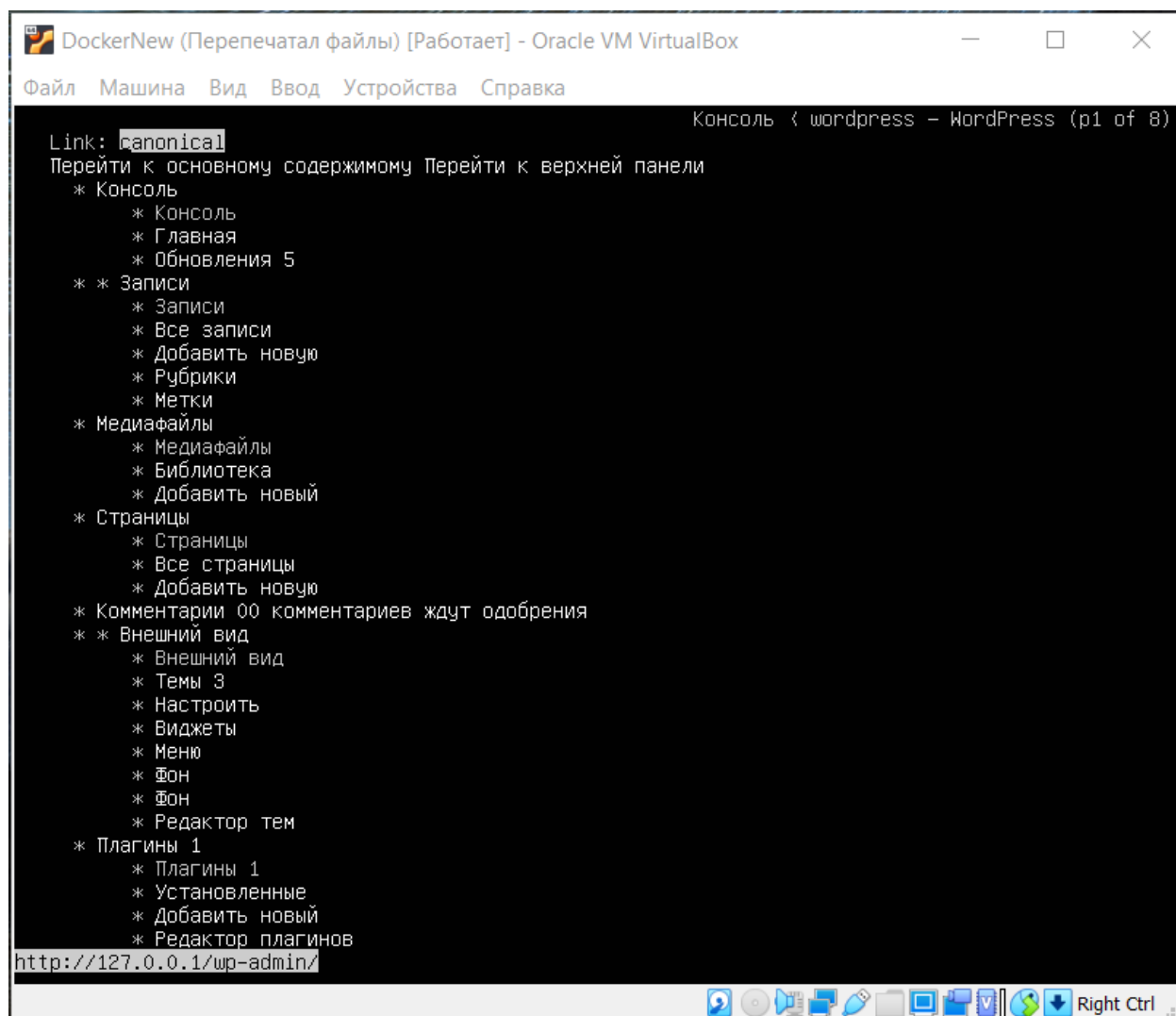


Рисунок 21 – пример работы wordpress.

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