

Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №6

по дисциплине «Операционная система Linux»

Контейнеризация

Студент

Посаднев В.В.

Группа АС-18

Руководитель

Кургасов В.В.

Липецк 2020 г.

Цель работы

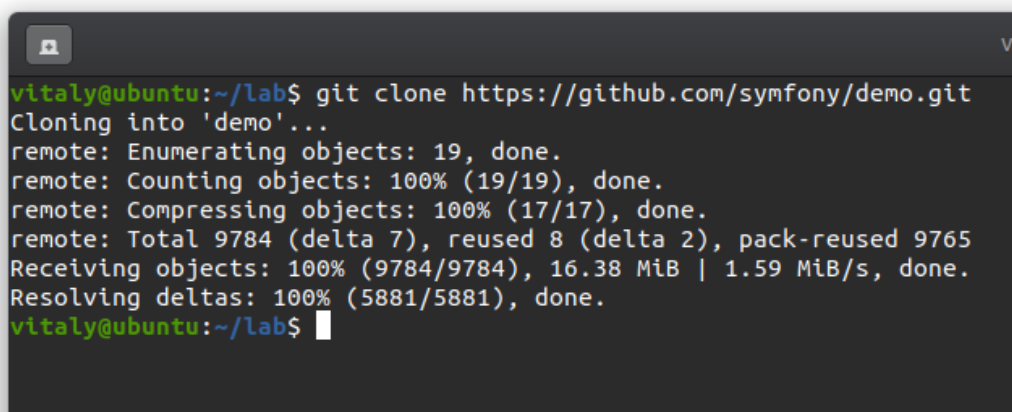
Изучить современные методы разработки ПО в динамических и распределенных средах на примере контейнеров Docker.

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 для демонстрации обновления данных в реальном времени.

Ход работы

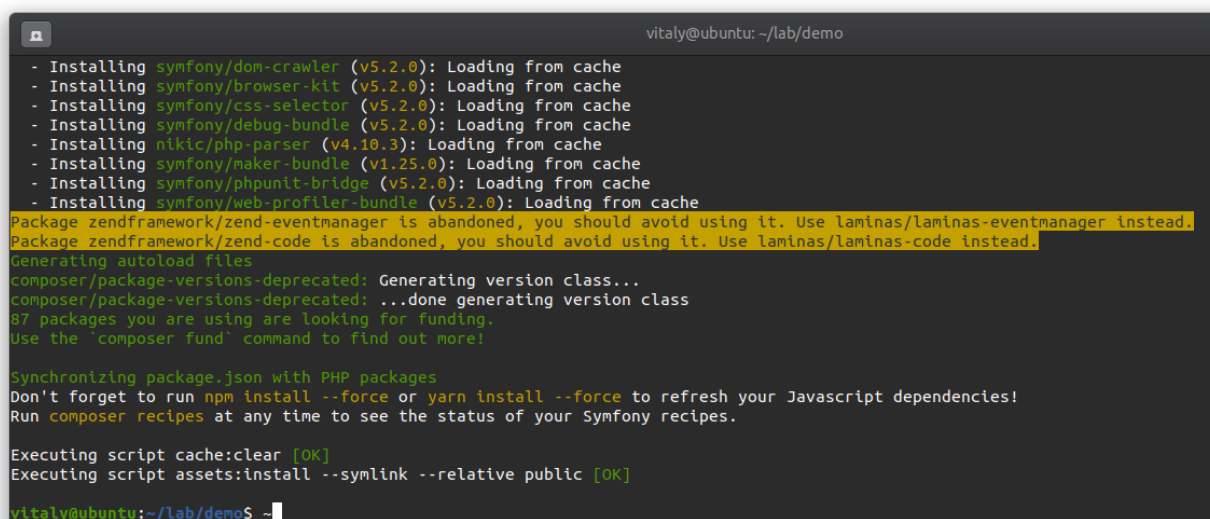
Для выполнения данной работы необходимо установить следующий набор приложений: *docker*, *docker-compose*, *composer*, *symphony*, *pdo_postgresql*. Создадим новый каталог и склонируем репозиторий демо-проекта в него, данное действие изображено на рисунке 1.



```
vitaly@ubuntu:~/lab$ git clone https://github.com/symfony/demo.git
Cloning into 'demo'...
remote: Enumerating objects: 19, done.
remote: Counting objects: 100% (19/19), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 9784 (delta 7), reused 8 (delta 2), pack-reused 9765
Receiving objects: 100% (9784/9784), 16.38 MiB | 1.59 MiB/s, done.
Resolving deltas: 100% (5881/5881), done.
vitaly@ubuntu:~/lab$
```

Рисунок 1 – Клонирование проекта в новый каталог

После клонирования демо-проекта необходимо установить все сопутствующие зависимости, для этого используем команду *composer install* находясь в каталоге с демо-проектом. Пример установки зависимостей для проекта изображено на рисунке 2.



```
vitaly@ubuntu: ~/lab/demo
- Installing symfony/dom-crawler (v5.2.0): Loading from cache
- Installing symfony/browser-kit (v5.2.0): Loading from cache
- Installing symfony/css-selector (v5.2.0): Loading from cache
- Installing symfony/debug-bundle (v5.2.0): Loading from cache
- Installing nikic/php-parser (v4.10.3): Loading from cache
- Installing symfony/maker-bundle (v1.25.0): Loading from cache
- Installing symfony/phpunit-bridge (v5.2.0): Loading from cache
- Installing symfony/web-profiler-bundle (v5.2.0): Loading from cache
Package zendframework/zend-eventmanager is abandoned, you should avoid using it. Use laminas/laminas-eventmanager instead.
Package zendframework/zend-code is abandoned, you should avoid using it. Use laminas/laminas-code instead.
Generating autoload files
composer/package-versions-deprecated: Generating version class...
composer/package-versions-deprecated: ...done generating version class
87 packages you are using are looking for funding.
Use the 'composer fund' command to find out more!

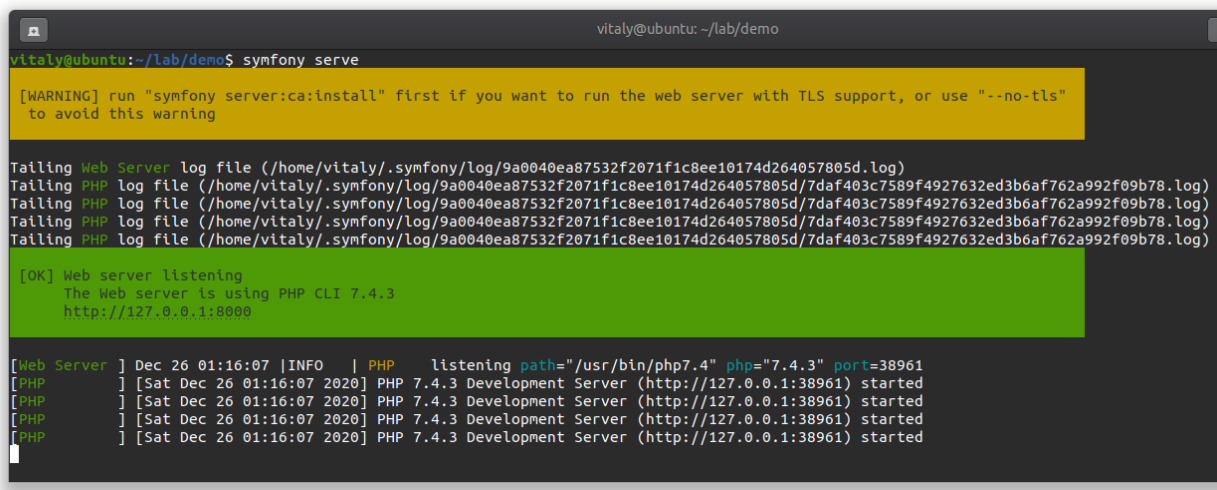
Synchronizing package.json with PHP packages
Don't forget to run npm install --force or yarn install --force to refresh your Javascript dependencies!
Run composer recipes at any time to see the status of your Symfony recipes.

Executing script cache:clear [OK]
Executing script assets:install --symlink --relative public [OK]
vitaly@ubuntu:~/lab/demo$
```

Рисунок 2 – Установка зависимостей для проекта

После данных манипуляций теперь мы можем запустить данный проект с помощью команды *symfony serve*. Пример успешного запуска проекта

изображено на рисунке 3.



```
vitaly@ubuntu: ~/lab/demo
vitaly@ubuntu:~/lab/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 (/home/vitaly/.symfony/log/9a0040ea87532f2071f1c8ee10174d264057805d.log)
Tailing PHP log file (/home/vitaly/.symfony/log/9a0040ea87532f2071f1c8ee10174d264057805d/7daf403c7589f4927632ed3b6af762a992f09b78.log)
Tailing PHP log file (/home/vitaly/.symfony/log/9a0040ea87532f2071f1c8ee10174d264057805d/7daf403c7589f4927632ed3b6af762a992f09b78.log)
Tailing PHP log file (/home/vitaly/.symfony/log/9a0040ea87532f2071f1c8ee10174d264057805d/7daf403c7589f4927632ed3b6af762a992f09b78.log)
Tailing PHP log file (/home/vitaly/.symfony/log/9a0040ea87532f2071f1c8ee10174d264057805d/7daf403c7589f4927632ed3b6af762a992f09b78.log)

[OK] Web server listening
The Web server is using PHP CLI 7.4.3
http://127.0.0.1:8000

[Web Server ] Dec 26 01:16:07 [INFO] | PHP listening path="/usr/bin/php7.4" php="7.4.3" port=38961
[PHP] [Sat Dec 26 01:16:07 2020] PHP 7.4.3 Development Server (http://127.0.0.1:38961) started
[PHP] [Sat Dec 26 01:16:07 2020] PHP 7.4.3 Development Server (http://127.0.0.1:38961) started
[PHP] [Sat Dec 26 01:16:07 2020] PHP 7.4.3 Development Server (http://127.0.0.1:38961) started
[PHP] [Sat Dec 26 01:16:07 2020] PHP 7.4.3 Development Server (http://127.0.0.1:38961) started
```

Рисунок 4 – Состояние консоли при успешном запуске проекта

При открытии браузера по указанному адресу (<http://127.0.0.1:8000>) мы увидим страницу успешно запущенного проекта. Пример данной страницы изображен на рисунке 5.

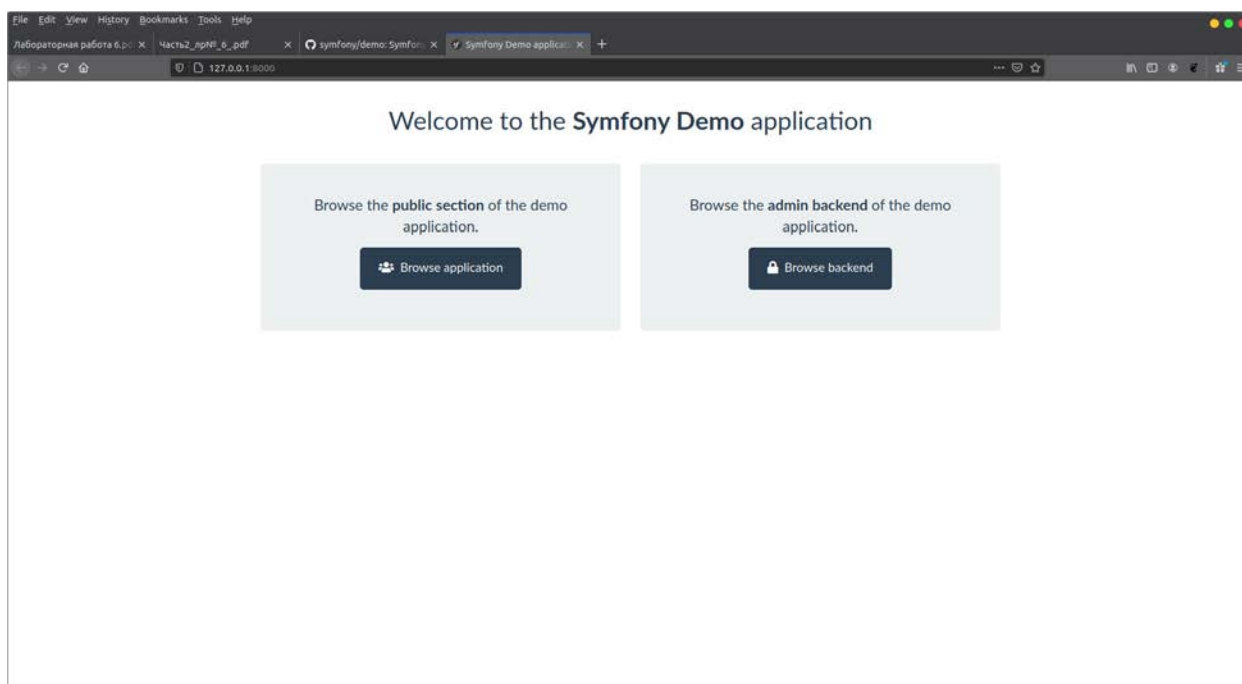


Рисунок 5 – Успешный запуск демо-проекта

Далее необходимо установить и запустить *postgresql*, зайдём в консоль и создадим пользователя *symfony_user* и выдадим ему права для базы *symfony_db*. Пример успешного выполнения данных действий представлено на рисунке 6.

```
vitaly@ubuntu: ~/lab/demo
vitaly@ubuntu:~/lab/demo$ psql -U postgres
Password for user postgres:
psql (12.5 (Ubuntu 12.5-0ubuntu0.20.04.1))
Type "help" for help.

postgres=# \l

```

Name	Owner	Encoding	Collate	Ctype	Access privileges
postgres	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	
symfony_db	symfony_user	UTF8	en_US.UTF-8	en_US.UTF-8	=Tc/symfony_user + symfony_user=CTc/symfony_user +
template0	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	=c/postgres + postgres=CTc/postgres +
template1	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	=c/postgres + postgres=CTc/postgres +
todolist_db	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	

```
(5 rows)
postgres=#
```

Рисунок 6 – Успешный вход в базу данных PostgreSQL

Следующим шагом необходимо привязать только что созданную базу к нашему демо-проекту. Для этого необходимо перейти по следующему пути: *demo/config/packages/doctrine.yaml*. Для успешного подключения проекта к нашей БД необходимо, чтобы файл *doctrine.yaml* имел содержимое эквивалентному на рисунке 7.

```
doctrine:
  dbal:
    url: '%env(resolve:DATABASE_URL)%'
    driver: 'pdo_pgsql'
    # IMPORTANT: You MUST configure your server version,
    # either here or in the DATABASE_URL env var (see .env file)
    #server_version: '5.7'
  orm:
    auto_generate_proxy_classes: true
    naming_strategy: doctrine.orm.naming_strategy.underscore_number_aware
    auto_mapping: true
    mappings:
      App:
        is_bundle: false
        type: annotation
        dir: '%kernel.project_dir%/src/Entity'
        prefix: 'App\Entity'
        alias: App

"demo/config/packages/doctrine.yaml" 18L, 633C
```

Рисунок 7 – Содержимое файла doctrine.yaml

Также необходимо изменить строку подключения к базе данных (DATABASE_URL) в файле находящемся по пути *demo/.env*. В переменной DATABASE_URL должно содержаться следующее: *postgresql://127.0.0.1:5432/db_name?user=user_login&password=user_password*. Конкретный пример содержимого файла *.env* изображен на рисунке 8.

```
vitaly@ubuntu:~/lab6
# 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-configuration
##> 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/configuration.html#connecting-using-a-url
# For a MySQL database, use: "mysql://db_user:db_password@127.0.0.1:3306/db_name"
# For a PostgreSQL database, use: "postgresql://db_user:db_password@127.0.0.1:5432/db_name?serverVersion=11&charset=utf8"
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
DATABASE_URL=postgresql://127.0.0.1:5432/symfony_db?user=symfony_user&password=tst123
##< doctrine/doctrine-bundle ##
##> symfony/mailer ##
# MAILER_DSN=smtp://localhost
##< symfony/mailer ##
```

Рисунок 8 – Содержимое файла настроек окружения проекта

Далее необходимо загрузить схему БД и примеры данных в неё. Для загрузки схемы БД необходимо использовать команду: `php bin/console doctrine:schema:create`, для загрузки данных необходимо использовать команду `php bin/console doctrine:fixtures:load`.

После выполнения данных действий проверим работоспособность демо-проекта на новой базе данных PostgreSQL. Пример успешного запуска проекта изображен на рисунке 9.

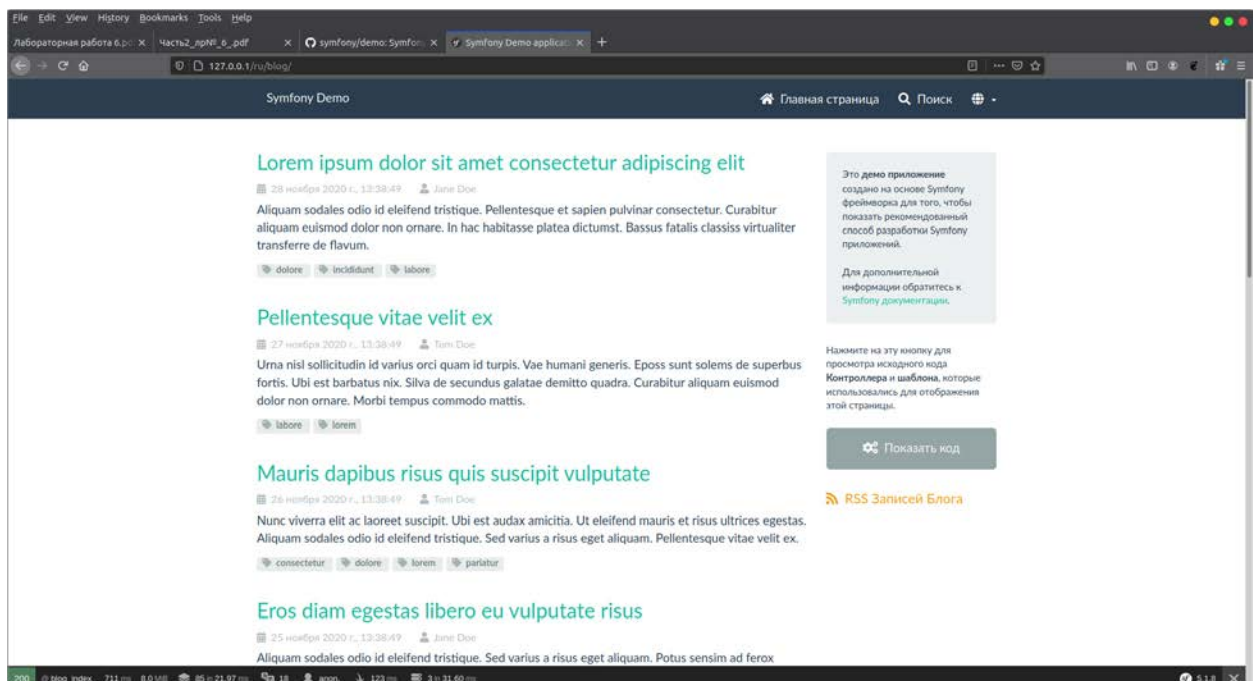
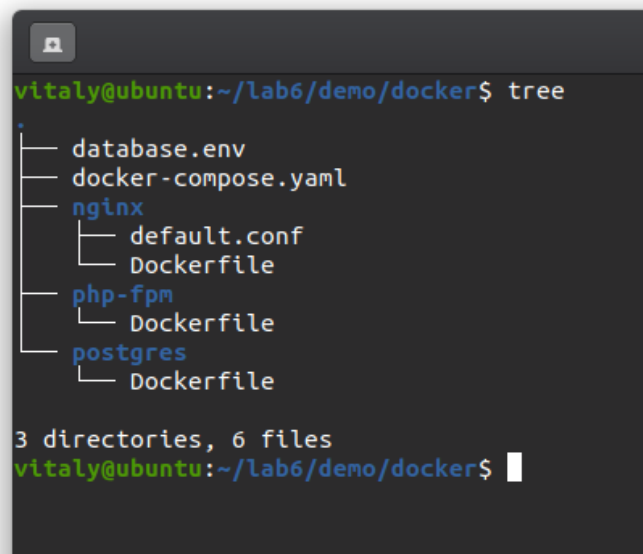


Рисунок 9 – Успешный запуск проекта на новой базе данных PostgreSQL

Теперь перейдем к созданию контейнеров. Создадим внутри папки с проектом каталог `docker` и перейдем в него, выполнить эти действия можно с

помощью команды `mkdir docker && cd docker`. Внутри данного каталога необходимо создать файл `docker-compose.yaml` для конфигураций всех контейнеров и файл `database.env`, который будет хранить конфигурацию для подключения и создания базы данных. Также создадим 3 каталога для каждого из контейнеров: `nginx`, `php-fpm`, `postgres`. В каждом из вышеперечисленных трех каталогов создадим `Dockerfile` для конфигурации каждого из контейнера. Таким образом у нас получается структура каталога изображенного на рисунке 10.



```
vitaly@ubuntu:~/lab6/demo/docker$ tree
.
├── database.env
├── docker-compose.yaml
├── nginx
│   ├── default.conf
│   └── Dockerfile
├── php-fpm
│   └── Dockerfile
└── postgres
    └── Dockerfile

3 directories, 6 files
vitaly@ubuntu:~/lab6/demo/docker$
```

Рисунок 10 – Структура проекта для работы с Docker`ом

Теперь рассмотрим содержимое каждого из файлов:

Файл `database.env`:

`POSTGRES_USER=symfony_user`

`POSTGRES_PASSWORD=tst123`

`POSTGRES_DB=symfony_db`

Файл `Dockerfile` находящийся в каталоге `nginx`:

`FROM nginx:latest`

`COPY default.conf /etc/nginx/conf.d/`

Файл `default.conf` находящийся в каталоге `nginx`:

`server {`

`listen 80;`

`server_name localhost;`


```
root /var/www/symfony/public;
```

```
location / {  
    try_files $uri @rewriteapp;  
}
```

```
location @rewriteapp {  
    rewrite ^(.*)$ /index.php/$1 last;  
}
```

```
location ~ ^/index\.php(/|$) {  
    fastcgi_pass php:9000;  
    fastcgi_split_path_info ^(.+\.(php|\.php))(/.*)$;  
    include fastcgi_params;  
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;  
    fastcgi_param HTTPS off;  
}
```

```
error_log /var/log/nginx/symfony_error.log;  
access_log /var/log/nginx/symfony_access.log;  
}
```

Файл *Dockerfile* находящийся в каталоге *php-fpm*:

```
FROM php:7.4-fpm
```

```
RUN apt-get update
```

```
RUN apt-get install -y zlib1g-dev libpq-dev git libicu-dev libxml2-dev libzip-dev vim \
```

```
&& docker-php-ext-configure intl \
```

```
&& docker-php-ext-install intl \
```

```
&& docker-php-ext-configure pgsql -with-pgsql=/usr/local/pgsql \
```

```
&& docker-php-ext-install pdo pdo_pgsql pgsql \
```

```
&& docker-php-ext-install zip xml
```

```
WORKDIR /var/www/symfony
```

Файл *Dockerfile* находящийся в каталоге *postgres*:

```
FROM postgres:latest
```

```
RUN apt-get update
```

RUN apt-get install -y vim git

Файл *docker-compose.yaml*:

version: '2'

services:

postgres:

image: postgres

ports:

- '5435:5432'

env_file:

- database.env

volumes:

- ./var/lib/postgresql/data:/var/lib/postgresql/data

php:

build: php-fpm

ports:

- '9002:9000'

volumes:

- ../:/var/www/symfony:cached

- ../logs/symfony:/var/www/symfony/var/logs:cached

links:

- postgres

nginx:

build: nginx

ports:

- '80:80'

links:

- php

volumes_from:

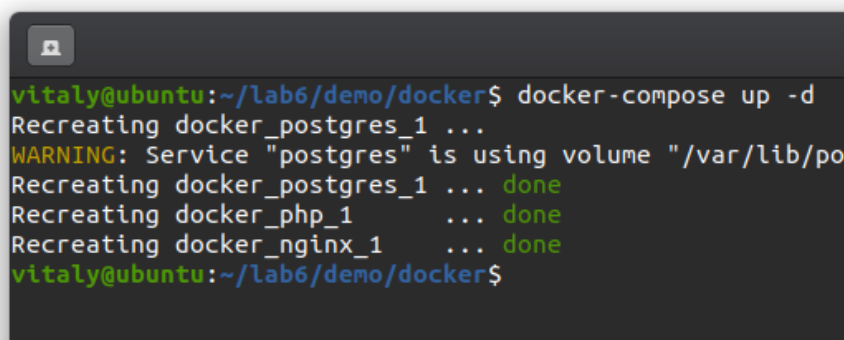
- php

volumes:

- ../logs/nginx:/var/log/nginx:cached

Как видно из содержимого вышеприведенных файлов, код демо-проекта располагается на каталог выше, чем текущая папка (Docker). С помощью команды volumes контейнер php-fpm подхватывает код проекта с локальной машины. Все контейнеры связаны во внутреннюю сеть, для этого необходимо указать версию два в docker-compose.yaml, поэтому контейнеры упомянутые в данном файле могут общаться по имени контейнера вместо ip-адреса. На данном этапе контейнер с проектом не будет видеть базу данных postgres. Для этого необходимо перейти в файл .env проекта и изменить параметр DATABASE_URL на следующее: postgresql://container_name:5432/db_name?user=user_login&password=user_password.

Теперь мы можем запустить контейнеры с помощью команды docker-compose up -d. Пример запуска контейнеров изображен на рисунке 11.



```
vitaly@ubuntu:~/lab6/demo/docker$ docker-compose up -d
Recreating docker_postgres_1 ...
WARNING: Service "postgres" is using volume "/var/lib/postgresql/data"
Recreating docker_postgres_1 ... done
Recreating docker_php_1 ... done
Recreating docker_nginx_1 ... done
vitaly@ubuntu:~/lab6/demo/docker$
```

Рисунок 11 – Запуск контейнеров

Однако после перехода на страницу с демо-проектом мы увидим, что появилась ошибка от отсутствия таблиц в базе данных. Пример такой ошибки изображен на рисунке 12. Также это можно проверить, для этого посмотрим на список запущенных контейнеров с помощью команды docker ps. И выберем ID контейнера с базой данных, далее перейдем в данный контейнер с помощью команды docker exec -it ID_CONTAINER bash. После перейдем в базу данных и посмотрим на содержимое БД. Пример выполнения данных действий изображено на рисунке 13.

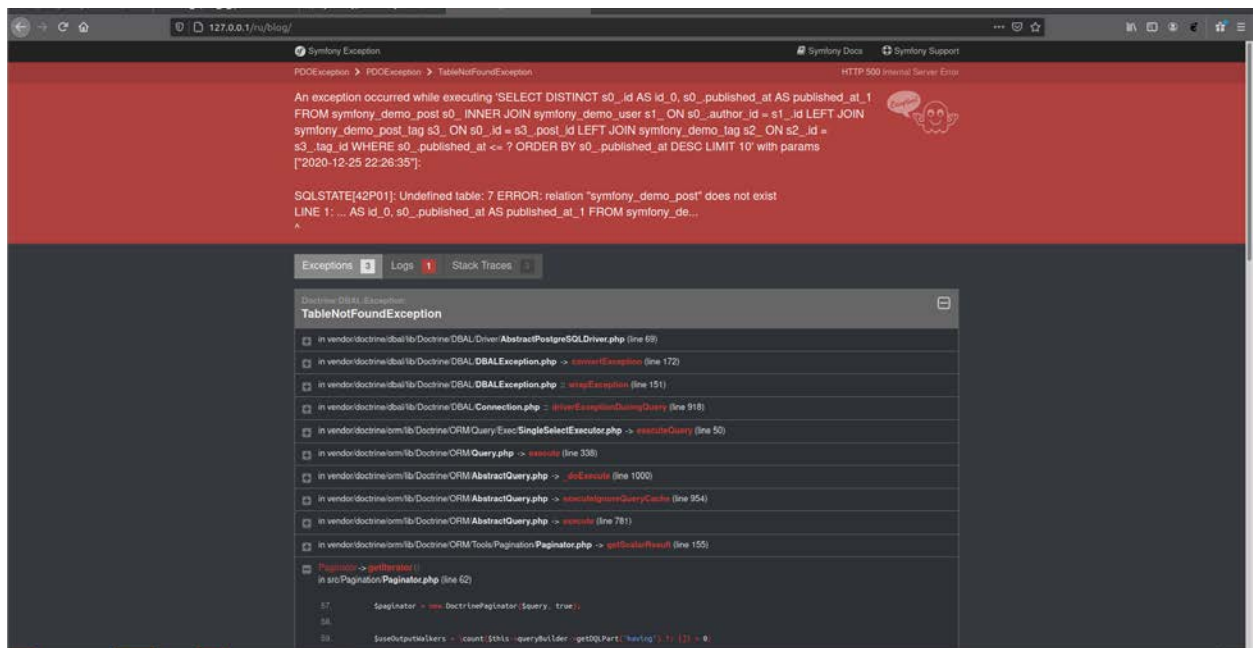


Рисунок 12 – Ошибка от отсутствия таблиц в базе данных

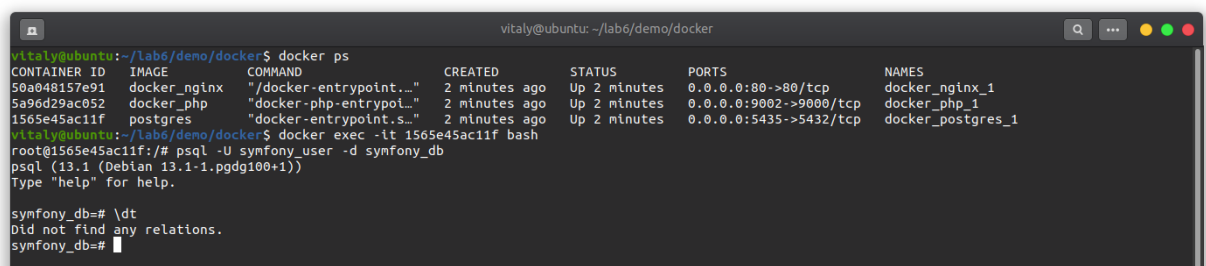


Рисунок 13 – Отсутствие таблиц в базе данных контейнера

Теперь нам необходимо подключиться к контейнеру с демо-проектом и выполним команды для инициализации таблиц базы данных с загрузкой в них тестовых данных. Пример выполнения данных команд изображено на рисунке 14.

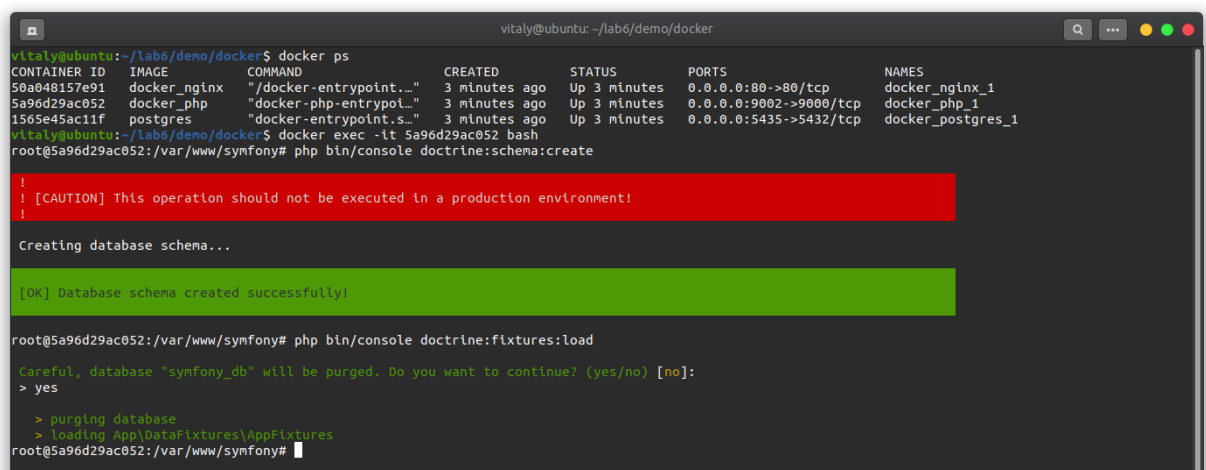


Рисунок 14 – Инициализации таблиц и данных в БД

После инициализации данных в БД мы можем перейти обратно на сайт и увидеть корректность его работы, что приведено на рисунке 15.

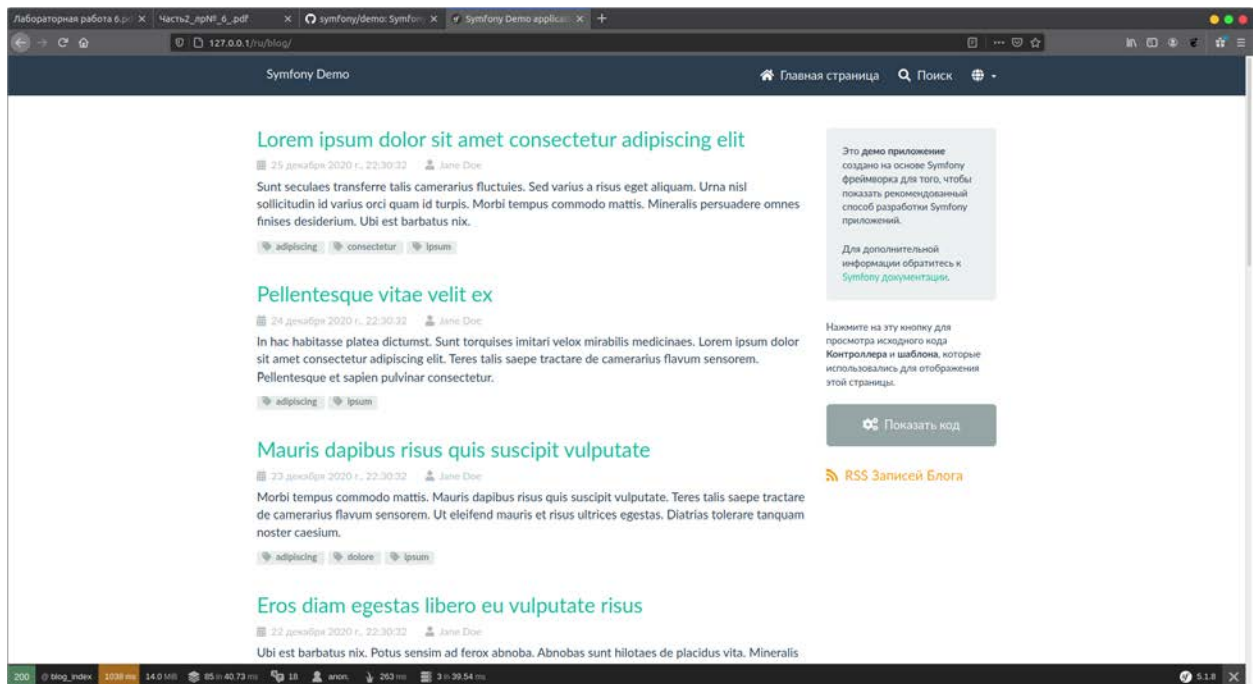


Рисунок 15 – Корректность работы демо-проекта в контейнере

Последним шагом осталось назначить имя данному демо-проекту. Для этого необходимо отредактировать файл `/etc/hosts` и дать новый псевдоним адресу `127.0.0.1` также, как это изображено на рисунке 16.

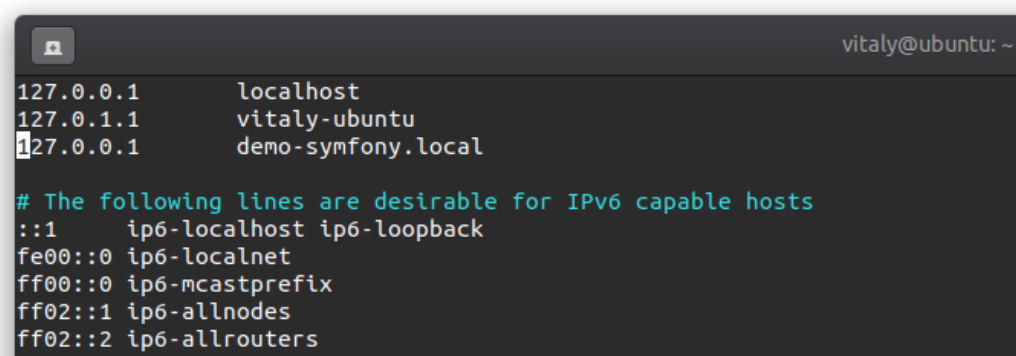


Рисунок 16 – Выдача нового псевдонима локальному адресу

Теперь попробуем убрать контейнер с базой данных postgres и подключаться к нашей локальной копии БД. Для этого необходимо изменить файл `docker-compose.yml` на следующее:

Содержимое файла `docker-compose.yml`:

version: '2'

services:

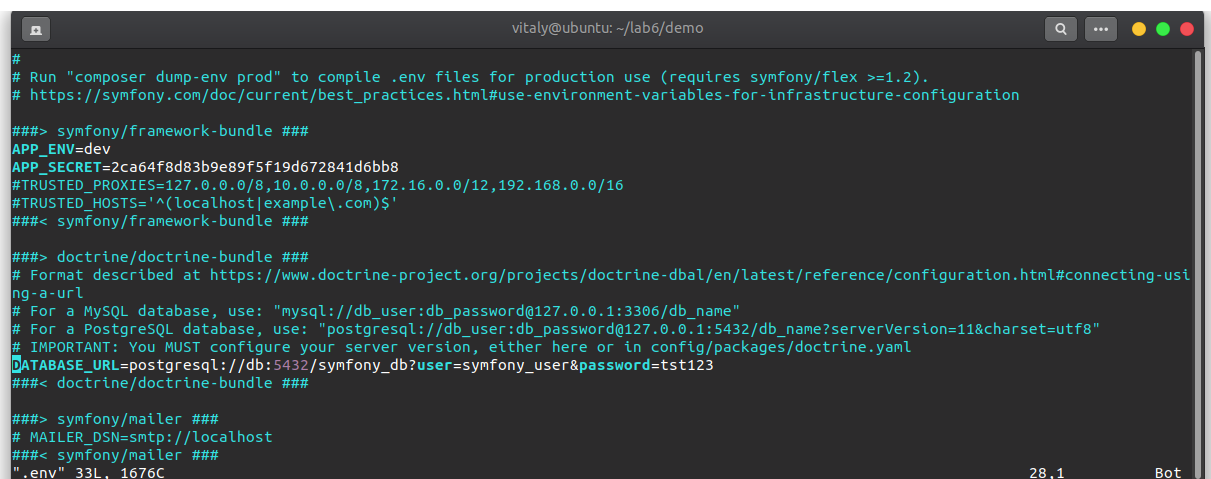
```

php:
  build: php-fpm
  ports:
    - '9002:9000'
  volumes:
    - ../:/var/www/symfony:cached
    - ../logs/symfony:/var/www/symfony/var/logs:cached
  extra_hosts:
    - "db:192.168.0.103"

nginx:
  build: nginx
  ports:
    - '80:80'
  links:
    - php
  volumes_from:
    - php
  volumes:
    - ../logs/nginx:/var/log/nginx:cached

```

В данном файле мы убрали контейнер с базой данных и прописали, что под переменной db будет определяться IP адрес нашей локальной машины к которой необходимо подключаться в проекте. Также изменим соответствующий адрес в настройках проекта, аналогично рисунку 17.



```

#
# 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-configuration

###> 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/configuration.html#connecting-usi
ng-a-url
# For a MySQL database, use: "mysql://db_user:db_password@127.0.0.1:3306/db_name"
# For a PostgreSQL database, use: "postgresql://db_user:db_password@127.0.0.1:5432/db_name?serverVersion=11&charset=utf8"
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
DATABASE_URL=postgresql://db:5432/symfony_db?user=symfony_user&password=tst123
###< doctrine/doctrine-bundle ###

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

".env" 33L, 1676C
28,1 Bot

```

Рисунок 17 – Настройка виртуального окружения для работы с локальной БД

Еще необходимо разрешить локальной базе данных подключение из вне и доступ с любого IP адреса. Для этого необходимо изменить конфигурацию файлов `/etc/postgresql/12/main/postgresql.conf` и `pg_hba.conf` аналогично рисункам 18 и 19.

```

external_pid_file = '/var/run/postgresql/12-main.pid'           # write
an extra PID file                                              # (change requires restart)

#-----
# CONNECTIONS AND AUTHENTICATION
#-----

# - Connection Settings -

listen_addresses = '*'          # what IP address(es) to listen on;
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for a
ll                                # (change requires restart)
port = 5432                     # (change requires restart)
max_connections = 100           # (change requires restart)
#superuser_reserved_connections = 3 # (change requires restart)
unix_socket_directories = '/var/run/postgresql' # comma-separated list of direct
ories                                # (change requires restart)
#unix_socket_group = ''         # (change requires restart)
"postgresql.conf" 750L, 26804C          59,19          6%

```

Рисунок 18 – Изменение файла postgresql.conf

```

# DO NOT DISABLE!
# If you change this first entry you will need to make sure that the
# database superuser can access the database using some other method.
# Noninteractive access to all databases is required during automatic
# maintenance (custom daily cronjobs, replication, and similar tasks).
#
# Database administrative login by Unix domain socket
local    all             postgres                                md5
# TYPE  DATABASE  USER  ADDRESS  METHOD

# "local" is for Unix domain socket connections only
local    all             all                                md5
# IPv4 local connections:
host     all             all             127.0.0.1/32      md5
host     all             all             0.0.0.0/0         md5
# IPv6 local connections:
host     all             all             ::1/128          md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
local    replication     all                                peer
host     replication     all             127.0.0.1/32      md5
host     replication     all             ::1/128          md5
"pg_hba.conf" 99L, 4713C          92,24-49          Bot

```

Рисунок 19 – Изменение файла pg_hba.conf

После перезагрузки postgres с помощью команды: `sudo service postgresql restart` попробуем добавить новую запись в локальную базу данных и посмотреть изменение на сайте с демо-проектом. Демонстрация добавления новой записи и отображения её на сайте показано на рисунке 20. Также посмотрим на запущенные контейнеры с помощью команды `docker ps` и отобразим полученный результат на рисунке 21.

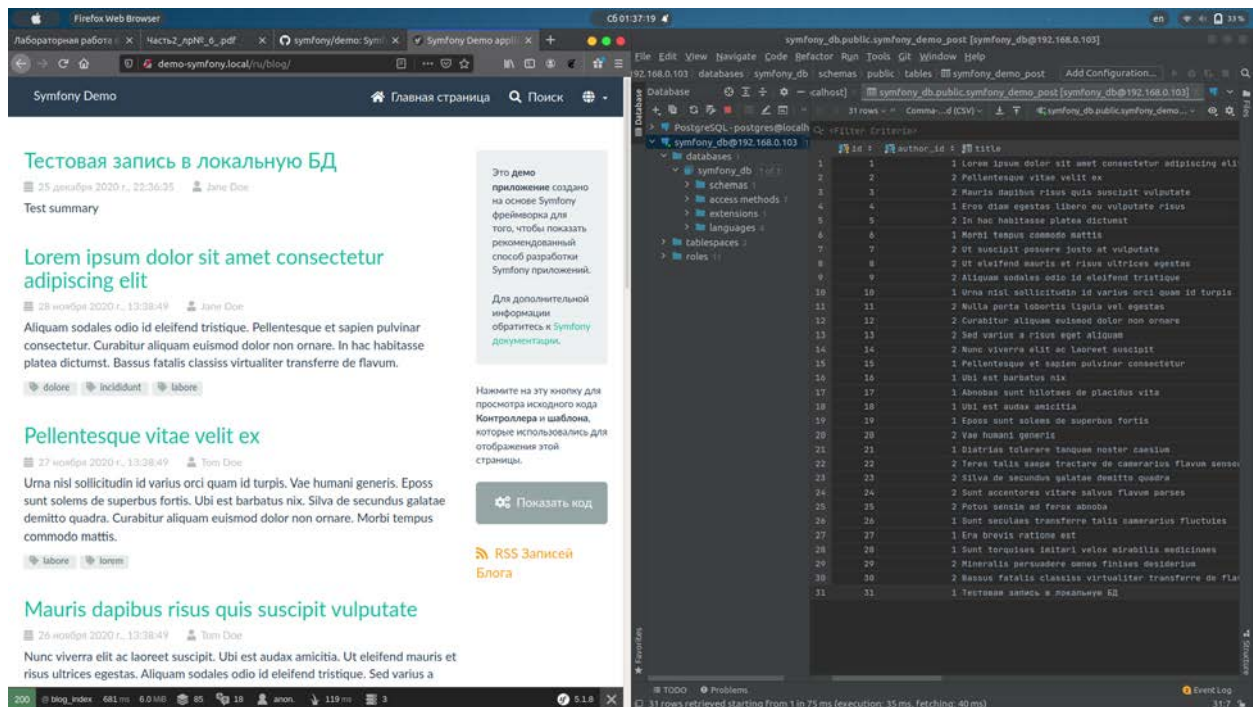


Рисунок 20 – Добавление и отображение новой записи на сайте

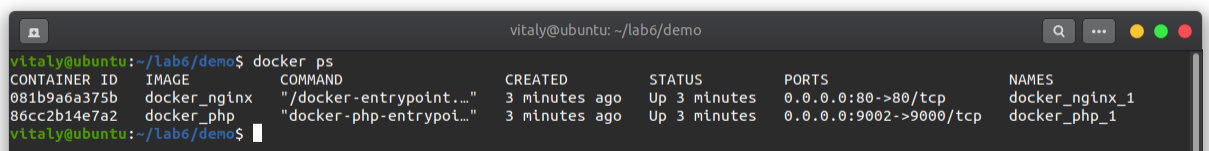


Рисунок 21 – Запущенные docker контейнеры

Вопросы для самопроверки

1. Назовите отличия использования контейнеров по сравнению с виртуализацией.

А. Меньшие накладные расходы на инфраструктуру

2. Назовите основные компоненты Docker.

В. Контейнеры

3. Какие технологии используются для работы с контейнерами?

С. 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?

Виртуальную машину, запускаемую на хосте, также часто называют «гостевой машиной». Гостевая машина содержит как приложение, так и все, что нужно для его запуска (например, системные исполняемые файлы и библиотеки). Она также несет в себе весь аппаратный стек, включая виртуальные сетевые адаптеры, файловое хранилище и центральный процессор, и свою собственную полноценную гостевую операционную систему.

В отличие от виртуальной машины, обеспечивающей аппаратную виртуализацию, контейнер обеспечивает виртуализацию на уровне операционной системы с помощью абстрагирования «пользовательского пространства».

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

docker build – сборка образа по настройкам в *Dockerfile*'е

docker run ... – запуск контейнера

docker stop... – остановка контейнера

docker images – отобразить образы в локальном репозитории

docker rmi ... - удалить образ

docker ps – отобразить все запущенные контейнеры

docker ps -a – отобразить остановленные контейнеры

docker exec -it... - выполнить команду в определенном контейнере

7. Каким образом осуществляется поиск образов контейнеров?

Изначально docker проверяет локальный репозиторий на наличие нужного образа. Если образ не найден, docker проверяет удаленный репозиторий Docker Hub.

8. Каким образом осуществляется запуск контейнера?

Docker выполняет инициализацию и запуск ранее созданного по образу контейнера по имени.

9. Что значит управлять состоянием контейнеров?

Это означает контролировать ход выполнения контейнера и в любой момент времени переводить его в остановленный/запущенный режим и выполнять команды внутри контейнера.

10. Как изолировать контейнер?

Для изоляции контейнера достаточно правильно сконфигурировать файлы *Dockerfile* и *docker-compose.yaml* (если есть). По умолчанию контейнеры запускаются от *root* прав, поэтому стоит быть осторожным с монтированием томов на хост машину.

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

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

Да, в среде другой виртуализации Kubernetes

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

Kubernetes открытое программное обеспечение для автоматизации развёртывания, масштабирования контейнеризированных приложений и управления ими. Поддерживает основные технологии контейнеризации, включая Docker, rkt, также возможна поддержка технологий аппаратной виртуализации.

Основные объекты:

Узел — это отдельная физическая или виртуальная машина, на которой развёрнуты и выполняются контейнеры приложений. Каждый узел в кластере содержит сервисы для запуска приложений в контейнерах (например, Docker), а также компоненты, предназначенные для централизованного управления узлом.

Под — базовая единица для управления и запуска приложений, один или несколько контейнеров, которым гарантирован запуск на одном узле, обеспечивается разделение ресурсов, межпроцессное взаимодействие и предоставляется уникальный в пределах кластера IP-адрес.

Том — общий ресурс хранения для совместного использования из контейнеров, развёрнутых в пределах одного пода.

Все объекты управления (узлы, поды, контейнеры) в Kubernetes помечаются метками, селекторы меток — это запросы, которые позволяют получить ссылку на объекты, соответствующие какой-то из меток; метки и селекторы — это главный механизм Kubernetes, который позволяет выбрать, какой из объектов следует использовать для запрашиваемой операции.

Сервисом в Kubernetes называют совокупность логически связанных наборов подов и политик доступа к ним.

Контроллер — это процесс, который управляет состоянием кластера, пытаясь привести его от фактического к желаемому; он делает это, оперируя набором подов, который определяется с помощью селекторов меток, являющихся частью определения контроллера.

Операторы — специализированный вид программного обеспечения Kubernetes, предназначенный для включения в кластер сервисов, сохраняющих своё состояние между выполнениями, таких как СУБД, системы мониторинга или кэширования. Назначение операторов — предоставить возможность управления stateful-приложениями в кластере Kubernetes прозрачным способом и скрыть подробности их настроек от основного процесса управления кластером Kubernetes.

Вывод

При выполнении данной лабораторной работы были изучены современные методы разработки ПО в динамических и распределенных средах на примерах контейнеров Docker и получены базовые навыки работы с СУБД PostgreSQL.