

Липецкий государственный технический университет  
Факультет автоматизации и информатики  
Кафедра автоматизированных систем управления

Лабораторная работа №6  
по Дисциплине «Операционная система Linux»  
на тему «Контейнеризация»

Студент

Глебов Д.А.

Группа АИ-18

Руководитель

Кургасов В.В.

к.п.н.

Липецк 2020 г.

### Цель работы

Изучить современные методы разработки ПО в динамических и распределённых средах на примере контейнера 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.

```
g4zele@g4zele:~/demo$ git clone https://github.com/symfony/demo.git
Cloning into 'demo'...
remote: Enumerating objects: 33, done.
remote: Counting objects: 100% (33/33), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 9798 (delta 15), reused 18 (delta 6), pack-reused 9765
Receiving objects: 100% (9798/9798), 16.40 MiB | 1.06 MiB/s, done.
Resolving deltas: 100% (5889/5889), done.
g4zele@g4zele:~/demo$
```

Рисунок 1 – Установка проекта

После клонирования демо-проекта необходимо установить все сопутствующие зависимости, для этого используем команду `composer install` находясь в каталоге с демо-проектом.

```
- Installing nikic/php-parser (v4.10.4): Loading from cache
- Installing symfony/maker-bundle (v1.26.1): Loading from cache
- Installing symfony/phpunit-bridge (v5.2.1): Loading from cache
- Installing symfony/web-profiler-bundle (v5.2.1): Loading from cache
Generating autoload files
composer/package-versions-deprecated: Generating version class...
composer/package-versions-deprecated: ...done generating version class
85 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]
g4zele@g4zele:~/demo$
```

Рисунок 2 – Установка зависимостей

После данных манипуляций теперь мы можем запустить данный проект с помощью команды `symfony serve`.

```
g4zele@g4zele:~/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/g4zele/.symfony/log/50c652c6ca4a5cd778045a7135af48f06a71b95f.log)
Tailing PHP-FPM log file (/home/g4zele/.symfony/log/50c652c6ca4a5cd778045a7135af48f06a71b95f/53fb8ec204547646acb3461995e4da5a54cc7575.log)

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

[Web Server ] Jan 8 19:26:05 |DEBUG| PHP Reloading PHP versions
[Web Server ] Jan 8 19:26:06 |DEBUG| PHP Using PHP version 7.4.3 (from default version in $PATH)
[Application] Jan 8 19:25:27 |INFO| REQUEST Matched route "homepage". method="GET" request_uri="http://127.0.0.1:8000/" route="homepage" route_parameters={"_controller":"Symfony\\Bundle\\FrameworkBundle\\Controller\\TemplateController::templateAction", "_locale":"en", "_route":"homepage", "template":"default/homepage.html.twig"}
[Application] Jan 8 19:25:28 |INFO| SECURITY Populated the TokenStorage with an anonymous Token.
[Application] Jan 8 19:25:30 |INFO| REQUEST Matched route "_wdt". method="GET" request_uri="http://127.0.0.1:8000/_wdt/d99c36" route="_wdt" route_parameters={"_controller":"web_profiler.controller.profiler::toolbarAction", "_route":"_wdt", "token":"d99c36"}
[Web Server ] Jan 8 19:26:06 |INFO| PHP listening path="/usr/sbin/php-fpm7.4" php="7.4.3" port=40237
[PHP-FPM ] Jan 8 19:26:07 |NOTICE| FPM fpm is running, pid 7624
[PHP-FPM ] Jan 8 19:26:07 |NOTICE| FPM ready to handle connections
[PHP-FPM ] Jan 8 19:26:07 |NOTICE| FPM systemd monitor interval set to 10000ms
```

Рисунок 3 – Запуск проекта

При открытии браузера по указанному адресу (<http://127.0.0.1:8000>) мы увидим страницу успешно запущенного проекта.

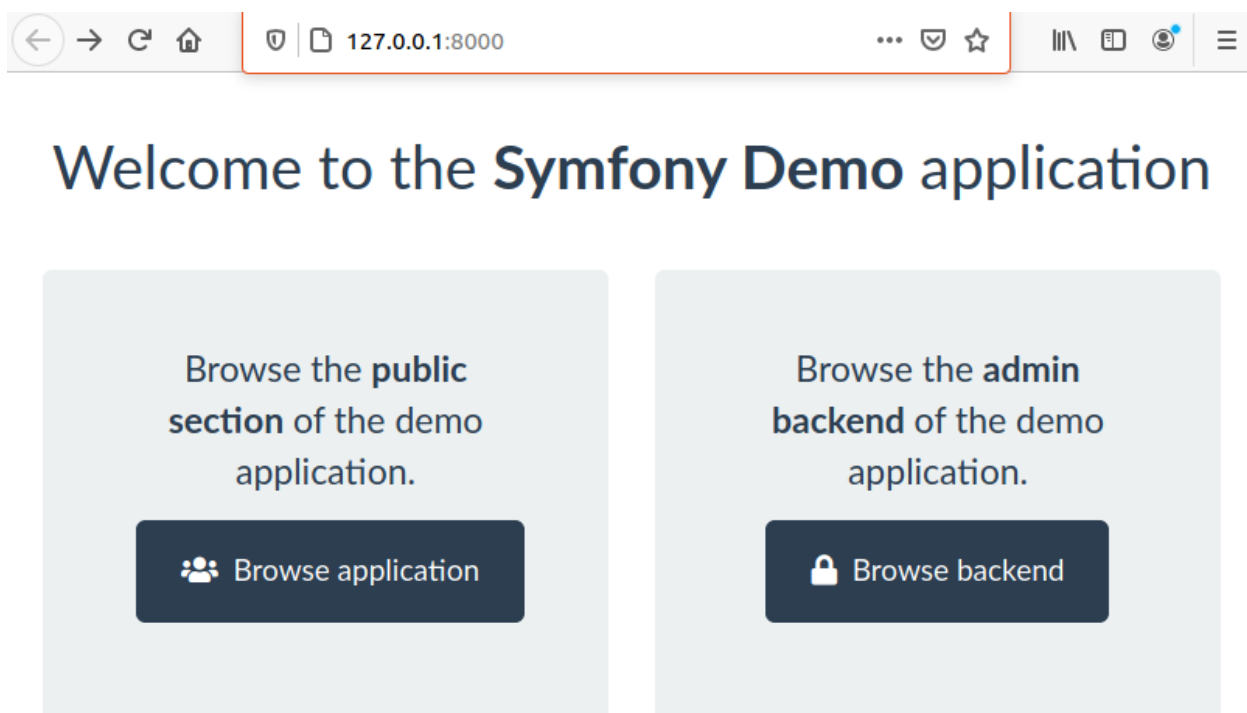


Рисунок 5 – Работа проекта

Далее необходимо установить и запустить postgresql, зайдём в консоль и создадим пользователя `symfony_user` и выдадим ему права для базы `symfony_db`.

```
g4zele@g4zele: ~  
g4zele@g4zele:~$ sudo -u postgres psql  
psql (12.5 (Ubuntu 12.5-0ubuntu0.20.04.1))  
Type "help" for help.  
  
postgres=# \l  
  
          List of databases  
-----  
 Name      | Owner  | Encoding | Collate | Ctype  | Access privileges  
-----  
 g4zele    | postgres | UTF8     | C.UTF-8 | C.UTF-8 |  
 postgres  | postgres | UTF8     | C.UTF-8 | C.UTF-8 |  
 template0 | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres +  
          |         |         |         |         | postgres=CTc/postgres  
 template1 | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres +  
          |         |         |         |         | postgres=CTc/postgres  
 test_db   | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =Tc/postgres +  
          |         |         |         |         | postgres=CTc/postgres +  
          |         |         |         |         | test_user=CTc/postgres  
(5 rows)  
  
postgres=#
```

Рисунок 6 – Настройка базы данных

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.

```
GNU nano 4.8 doctrine.yaml  
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
```

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/test_db?user=test_user&password=123`.

```
# 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/test_db?user=test_user&password=123  
###< doctrine/doctrine-bundle ###
```

Рисунок 8 – Изменение файла .env

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 демопроекта на новой базе данных PostgreSQL.

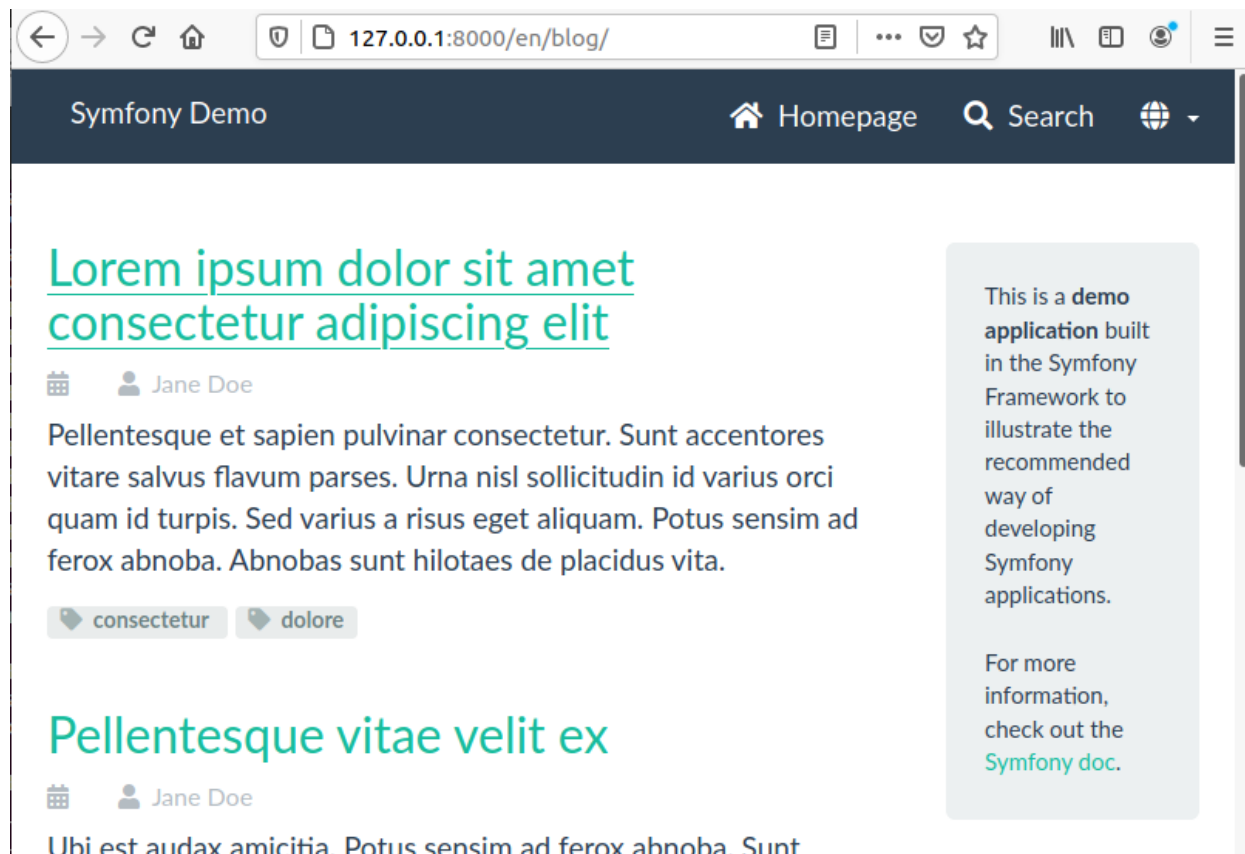


Рисунок 9 – Работа проекта на базе данных PostgreSQL

Перейдем к настройке контейнеров. Первым делом создадим папку и уже в ней файл `docker-compose.yml` и заполним его следующим содержимым:

```
version: '3'

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
```

В данном файле мы определяем структуру и связи наших контейнеров, перенаправляем некоторые порты, а так же связываем файлы и каталоги на локальном компьютере с контейнерами. В частности с помощью данного механизма исключается возможность потери базы данных при остановке контейнеров.

Создадим 2 файла: nginx.Dockerfile и php.Dockerfile, а также каталог conf, содержащий файл конфигурации nginx default.conf.

Файл nginx.Dockerfile:

```
FROM nginx:latest
COPY default.conf /etc/nginx/conf.d/
```

Файл vhost.conf:

```
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 php.Dockerfile:

```

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/symphony

```

После этого также редактируем файл .env, заменяем строку подключения к БД следующей строкой:

```

DATABASE_URL=postgresql://db_user:db_pass@postgres:5432/
db_name?serverVersion=11&charset=utf8

```

Следующим шагом запускаем все контейнеры в фоне, с помощью команды `docker-compose up -d`, переходим в контейнер с postgresql (команда `docker exec -it postgres bash`), внутри контейнера переходим в консоль psql и создаем БД demo\_bd (команда `create database demo_db;`). Далее переходим в контейнер с php и загрузить схему БД (команда `php bin/console doctrine:schema:create`) и данные для БД (команда `php bin/console doctrine:fixtures:load`).

После этого редактируем файл `/etc/hosts` и добавляем псевдоним адресу `127.0.0.1 demo-symfony.local`.

Запускаем контейнеры командой `docker-compose up -d` и проверяем работу проекта.

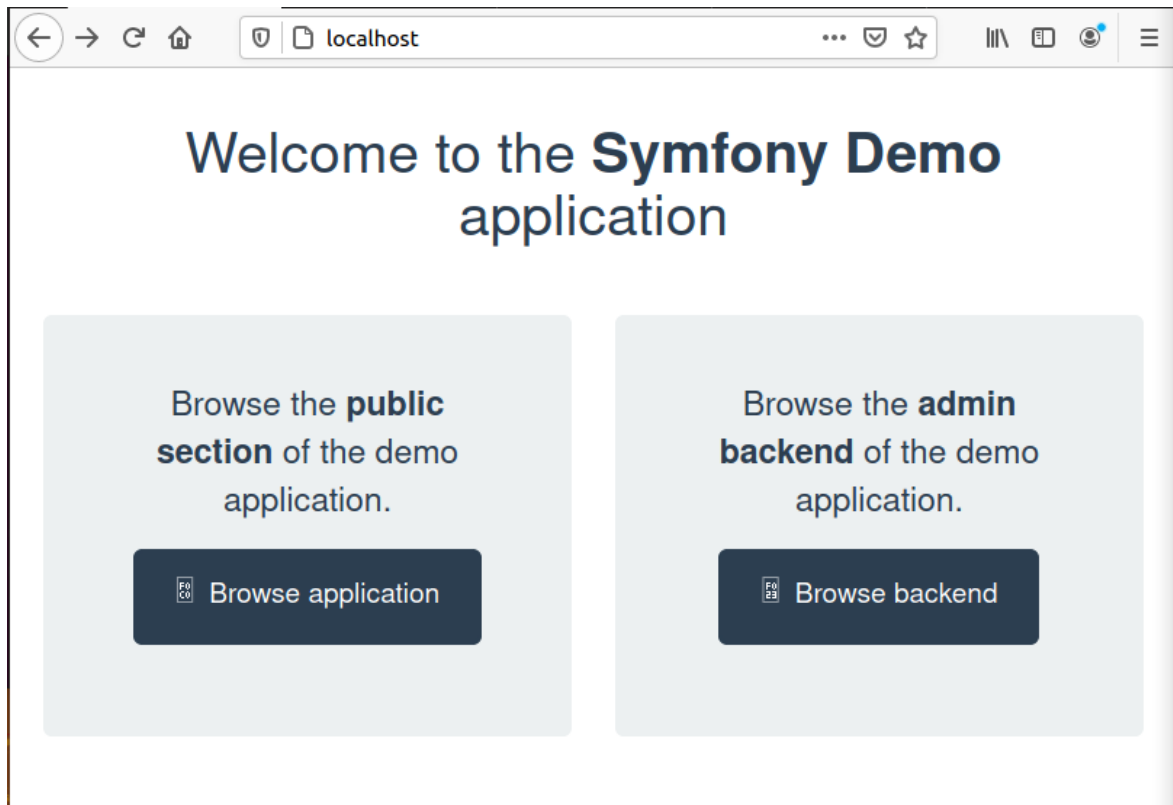


Рисунок 10 - Запуск проекта в контейнерах

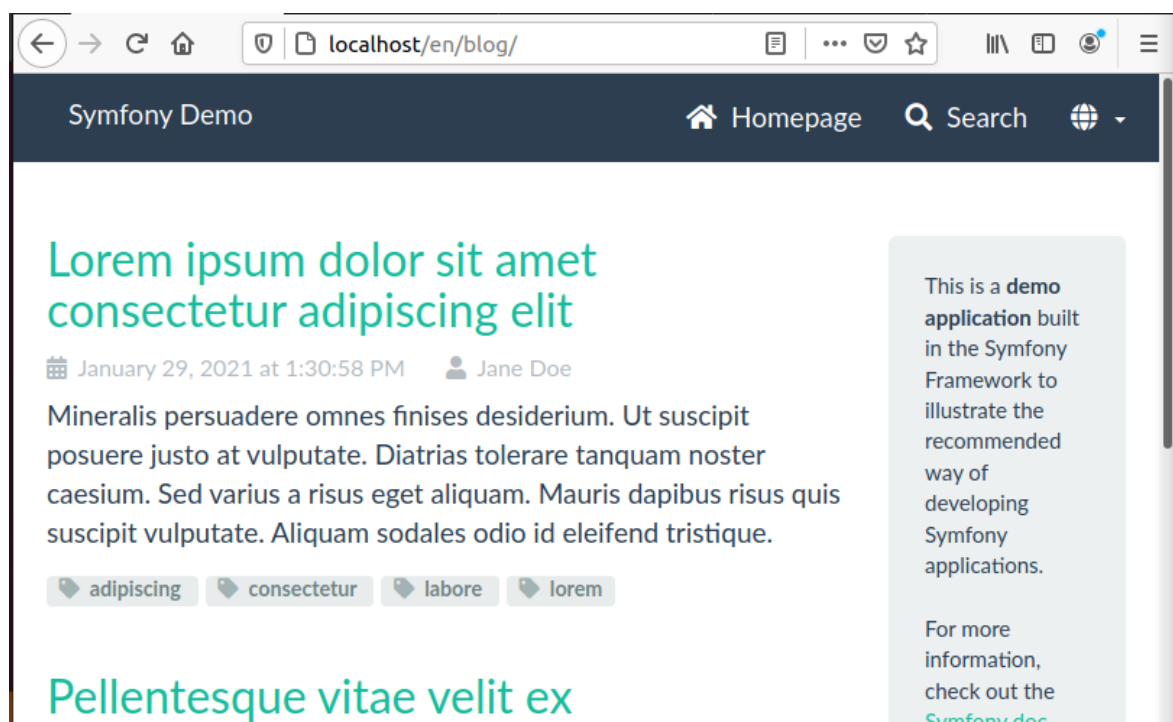


Рисунок 11 - Проверка, что БД не удаляется после перезагрузки проекта

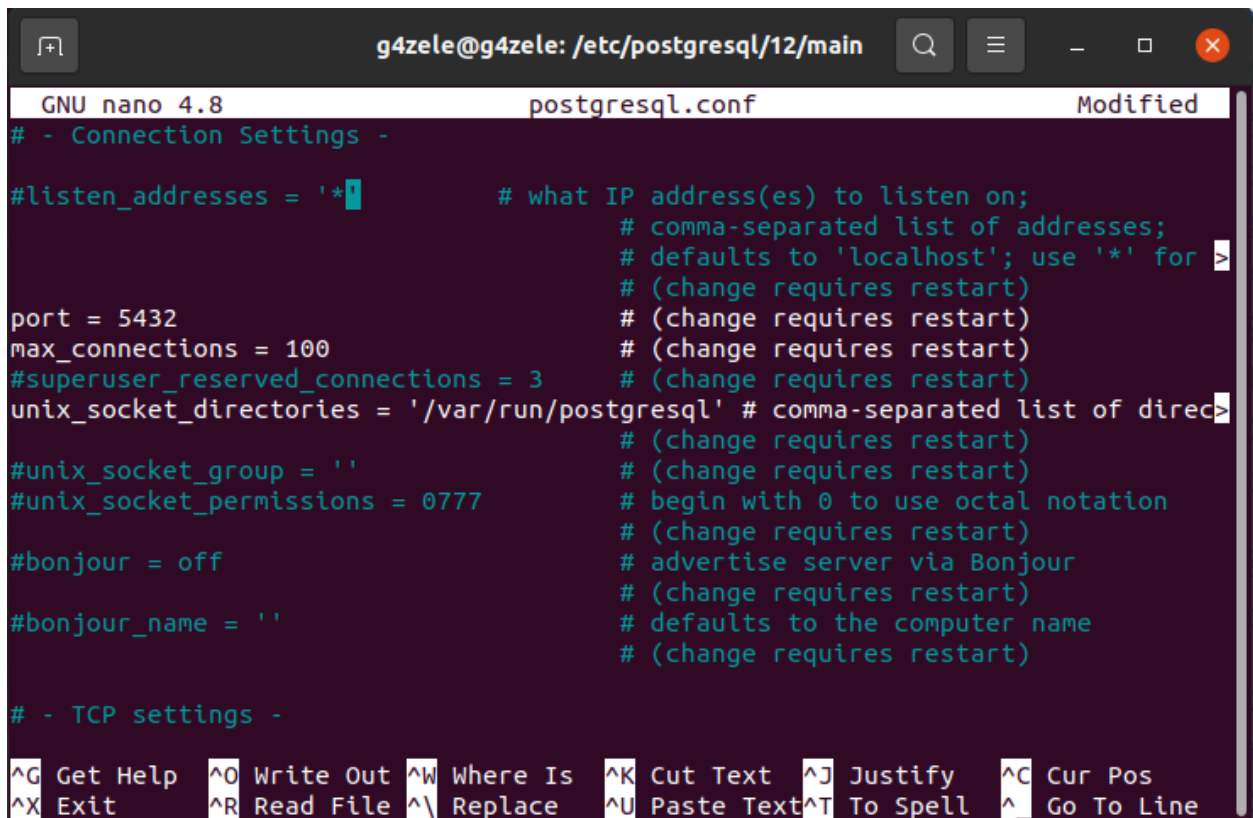
## Часть 2

Теперь удаляем конфигурацию контейнера с postgresql и подключим проект к локальной базе данных. Для этого узнаем ip локальной машины с помощью команды `hostname -I | cut -d ' ' -f1` и добавим этот ip под псевдонимом `bd` в наш файл `docker-compose.yml`, получим следующее содержимое:

```
version: '3'
services:
  php:
    build: php-fpm
    ports:
      - '9002:9000'
    volumes:
      - ../:/var/www/symfony:cached
      - ../logs/symfony:/var/www/symfony/var/logs:cached
    links:
      - postgres
    extra_hosts:
      - "db: 127.0.1.1"

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

Также изменим этот адрес в файле `.env`. Теперь изменим конфигурацию локальной базы данных, так, чтобы она допускала подключение из контейнера. Для этого изменим файлы конфигурации `/etc/postgresql/10/main/postgresql.conf` и `pg_hba.conf`:

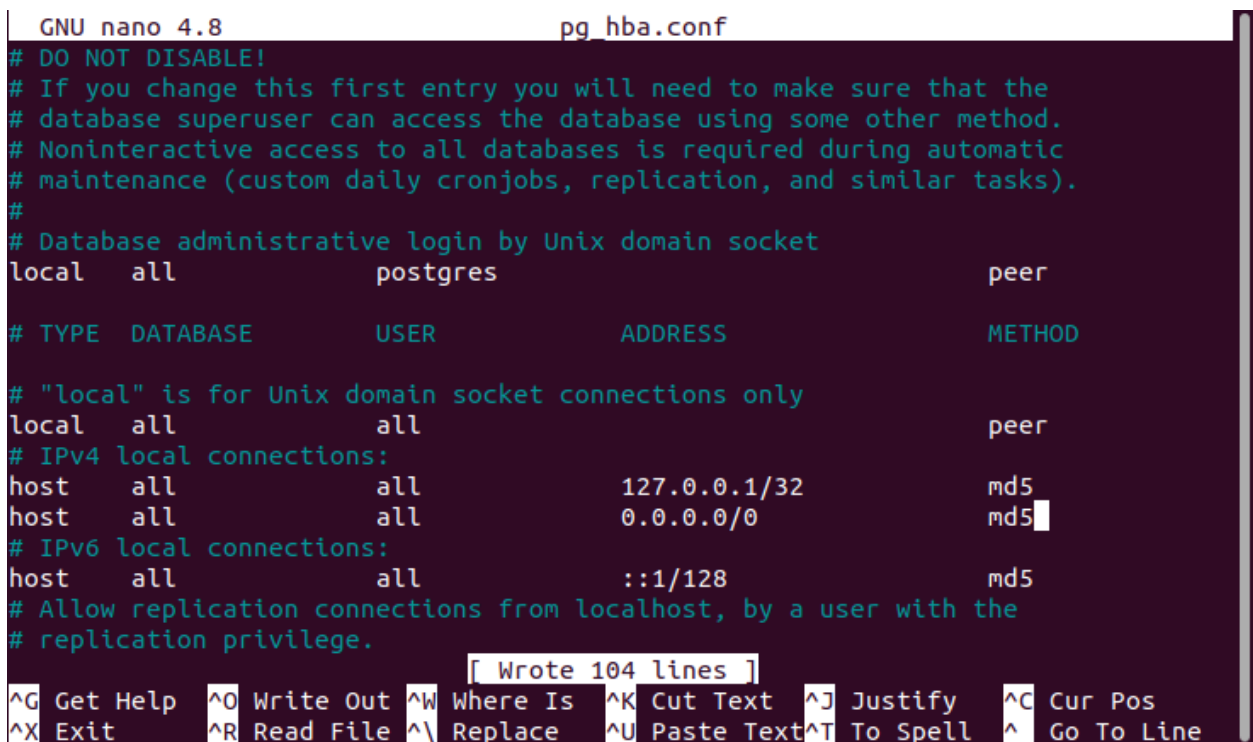


```
g4zele@g4zele: /etc/postgresql/12/main
GNU nano 4.8 postgresql.conf Modified
# - Connection Settings -
#listen_addresses = '*'          # what IP address(es) to listen on;
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for
                                # (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 direc>
                                # (change requires restart)
#unix_socket_group = ''         # (change requires restart)
#unix_socket_permissions = 0777 # begin with 0 to use octal notation
                                # (change requires restart)
#bonjour = off                  # advertise server via Bonjour
                                # (change requires restart)
#bonjour_name = ''              # defaults to the computer name
                                # (change requires restart)

# - TCP settings -

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

Рисунок 12 - Изменение файла postgresql.conf



```
GNU nano 4.8 pg_hba.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 peer

# TYPE DATABASE USER ADDRESS METHOD
# "local" is for Unix domain socket connections only
local all all peer
# 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.

[ Wrote 104 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Paste Text ^T To Spell  ^_ Go To Line
```

Рисунок 13 - Изменение файла pg\_hba.conf

Перезапустим postgresql с помощью команды `service postgresql restart` и попробуем запустить наш проект.

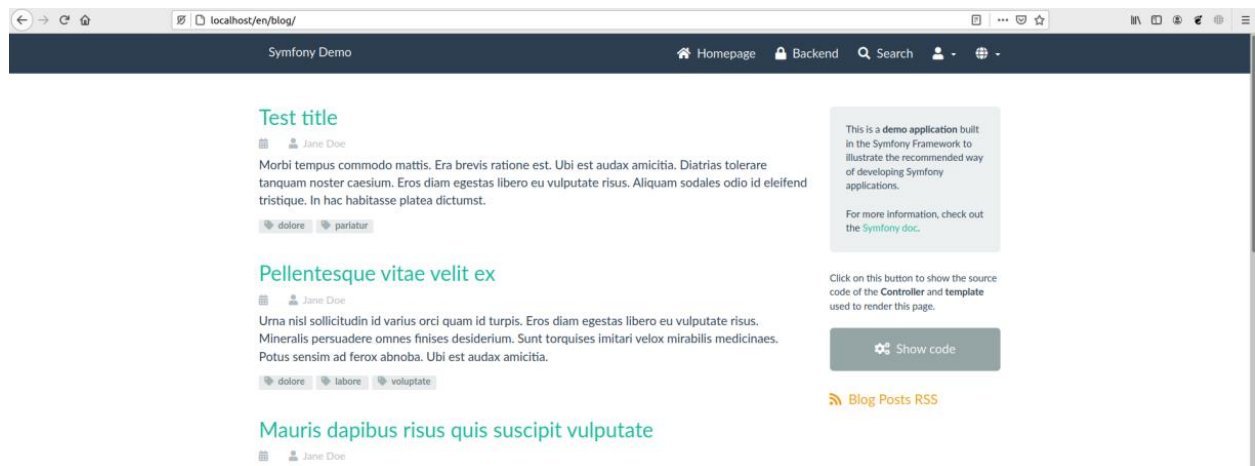


Рисунок 13 - Подключение к локальной БД

## Контрольные вопросы

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?

Виртуальная машина – программная и/или аппаратная система, эмулирующая аппаратное обеспечение некоторой целевой и исполняющая программы для гостевой платформы на платформе-хозяине (хосте) или виртуализирующая некоторую платформу и создающая на ней среды, изолирующие друг от друга программы и даже операционные системы. Виртуальные машины запускают на физических машинах, используя гипервизор.

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

В целом контейнеры выглядят как виртуальные машины. Например, у них есть изолированное пространство для запуска приложений, они позволяют выполнять команды с правами суперпользователя, имеют частный сетевой интерфейс и IP-адрес, пользовательские маршруты и правила межсетевого экрана и т. д.

Одна большая разница между контейнерами и виртуальными машинами в том, что контейнеры разделяют ядро хоста с другими контейнерами.

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

- `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. Каким образом осуществляется поиск образов контейнеров?

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

8. Каким образом осуществляется запуск контейнера?

Для запуска контейнера его необходимо изначально создать из образа, поэтому изначально контейнер собирается с помощью команды `docker build`, а уже затем запускается с помощью команды `docker run`.

9. Что значит управлять состоянием контейнеров?

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

## 10. Как изолировать контейнер?

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

## 11. Опишите последовательность создания новых образов, назначение `Dockerfile`?

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

Да, если использовать 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.

— 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 «реплик» 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.