Лабораторная работа №5
Docker & Docker-compose

Цель лабораторной работы: получить практические навыки развертывания готового продукта с помощью механизма контейнеризации.

Задачи лабораторной работы:

1. Изучить механизм работы Docker
2. Научиться создавать образы и запускать контейнеры на их основе
3. Получить навыки работы с Docker-compose
4. Получить навыки работы с томами Docker volumes
5. Получить навыки работы с DockerHub
6. Научиться настраивать CI/CD для проектов (**опционально**)

Ключевые слова: Docker образ/контейнер (Dockerfile), DockerHub, Docker-compose (docker-compose.yml), nginx, SQLite, Docker volumes, GitHub Actions CI/CD

Теоретическая часть:

Статьи с теорией

1. Официальная документация Docker
2. Хорошая статья на хабре про Docker
3. Статья на хабре про nginx
4. Информация по настройке CI / CD
5. Официальная документация по работе с Docker Hub
6. Гайд по настройке sqlite3 с docker
7. Отличный гайд по docker

Практическая часть:

Описание лабораторной работы:

По ссылке https://github.com/mdn/django-locallibrary-tutorial доступно готовое приложение, состоящее из клиентской и серверной части. Данный веб-сервис представляет собой простой сайт, работающий с базой данных SQLite. Вам необходимо развернуть этот сайт в docker-контейнере.

Задание 1. Подготовительная часть.

1. Установите программы **git**, **docker**, **docker-compose** на вашу виртуальную машину / хостовую систему. Инструкция по установке для CentOS: https://docs.docker.com/engine/install/centos/
2. Проверьте корректность запуска командой **sudo docker run hello-world**
3. Сделайте форк репозитория https://github.com/mdn/django-locallibrary-tutorial и склонируйте его локально на хост \ виртуальную машину.
4. Создайте файл **.env** в корне проекта и пропишите в нем ключи:
   SECRET_KEY=”<любая последовательность символов>”
   DEBUG=True

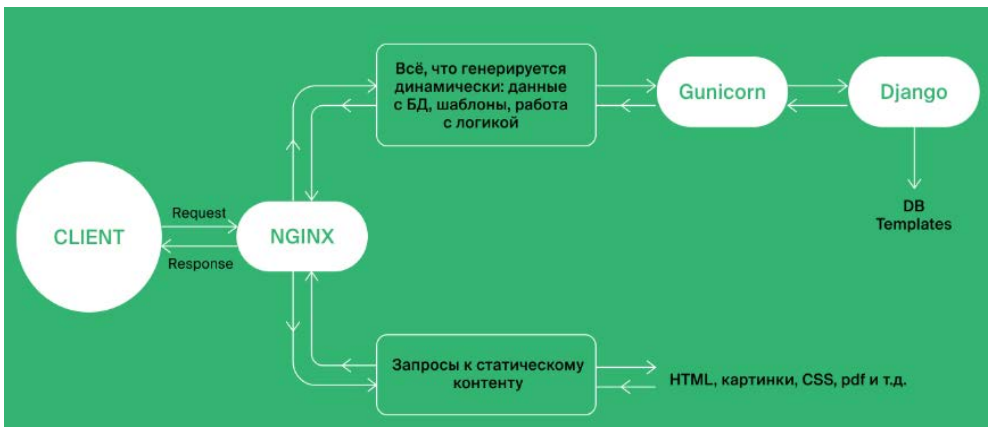
Задание 2. Разверните проект локально.

1. Установите Python версии >= 3.7
2. Создайте внутри локального репозитория виртуальное окружение для используемых библиотек. Хорошим тоном для каждого проекта является наличие изолированной среды со всеми зависимостями, чтобы не засорять файловую систему. Откройте терминал в директории проекта и пропишите команду
   python -m venv venv.
3. Для активации окружения пропишите команду
   source venv/bin/activate.

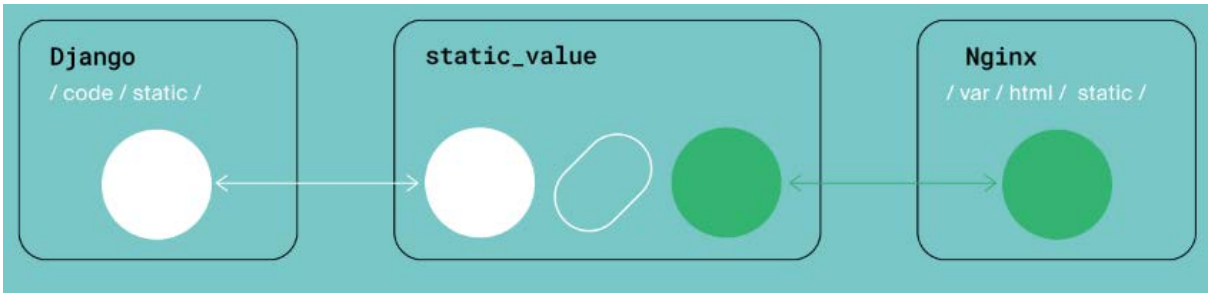
4. Удостоверьтесь, что вы в правильном окружении, посмотрев на надпись в терминале над полем ввода. Она должна иметь значение **venv**
5. Установите необходимые для работы библиотеки зависимости из файла **requirements.txt**. Для установки используйте команду **pip install -r requirements.txt**.
6. Проверьте, что все зависимости установлены с помощью команды **pip freeze**
7. В файле models.py прописаны таблицы, использьемые в базе данных. Загрузите данные миграции в БД, используя команду **python manage.py migrate.** Проверьте, что в корневой директории создался файл с db.sqlite3.
8. Протестируйте проект: запустите команду **python manage.py test.**
9. Разверните проект локально с помощью команды **python manage.py runserver.** Проверьте, что проект доступен по адресу: *http://localhost:8000* или *http://127.0.0.1:8000*

Задание 3. Запустите тот же самый проект в докере

1. Создайте **Dockerfile**, **docker-compose.yml** и **nginx.conf** файлы в корневой директории.
2. Пропишите необходимые команды в файлах конфигурации docker и nginx.
3. В Dockerfile последним пунктом добавьте настройку **gunicorn**. Команда: **CMD gunicorn locallibrary.wsgi:application --bind 0.0.0.0:8000**. **Gunicorn** - это **wsgi** сервис, который помогает



связывать nginx и django, по факту посредник между ними.

4. Запустите все контейнеры через docker-compose. Проверьте, что у вас есть доступ к сайту по адресу http://127.0.0.1:8000. Убедитесь в том, что сайт некрасивый (можно например зайти на /admin)
5. Исправьте стили и верстку, загрузив файлы статики в **docker volume**, таким образом чтобы nginx видел файлы и раздавал их. Убедитесь в правильности настроек и подключении docker volumes.
   Как работают **volumes**:
   

6. Выполните команду **collectstatic** внутри контейнера**.** Для этого, находясь в корневой директории, выполните команду: **docker-compose exec <CONTAINER NAME> python manage.py collectstatic --noinput**.
7. Импортируйте миграции в базу данных с помощью команды **docker-compose exec <CONTAINER NAME> python manage.py migrate --noinput**
8. Создайте суперпользователя внутри контейнера: **docker-compose exec <CONTAINER NAME> python manage.py createsuperuser**
9. Посмотрите содержимое контейнера с помощью команды **docker-compose exec <CONTAINER NAME> bash**

Задание 4. Настройка CI/CD.

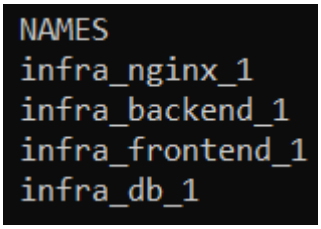
1. Создайте файл .github/workflows/main.yml
2. Настройте CI/CD в файле .github/workflows/main.yml для осуществления доставки docker образов на **Docker Hub** при каждой операции push в репозитории на github
3. Измените файл .github/workflows/main.yml для прохождения автотестов при каждой операции push в репозитории на github
4. Сделайте push в репозиторий. Проверьте, что ваши тесты запускаются

Предоставьте преподавателю отчет, содержащий следующие пункты:

1. Скриншот запуска контейнера hello-world из пункта 1.2
2. Содержимое файла .env
3. Скриншот работающего сайта, развернутого локально
4. Итоговое содержимое файлов Dockerfile, docker-compose.yml и nginx.conf
5. Скриншот работающего сайта, развернутого в docker
6. Содержимое файла .github/workflows/main.yml
7. Отчет о прохождении тестировании при операции push в репозитории на github
8. Ссылку на форк репозитория в вашем аккаунте на гитхабе с финальной версией проекта

Возможные проблемы и рекомендации

- Если после миграций произошли какие-то проблемы, попробуйте выполнить команду **docker-compose exec backend python manage.py makemigrations --noinput**. Данная команда генерирует правила таблиц для БД (по факту порядок sql запросов к бд). После этого выполните команду **migrate** снова.
- Если сайт выглядит некрасиво, удостоверьтесь, что сгенерировали статику (collectstatic) и правильно настроили docker volumes.
- Если сайт выдает ошибку с кодом 500 (Server error), удостоверьтесь, что правильно настроили **nginx**.
  P.S. можно посмотреть логи контейнера
- Зайти внутрь контейнера можно не только через docker-compose, но и с помощью самого docker. Различие только в названии.
  Пример:
  Обратимся к контейнеру nginx:
  docker: docker exec infra_nginx_1 bash
  docker-compose: docker-compose exec nginx_1 bash



Разница в том, что docker-compose запускает контейнеры связано => с общим именем (в данном случае infra - это папка, в которой лежат файлы для конфигурации docker-compose и nginx)