|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Студент:** | Еселидзе Д.В. | | **Допуск:** | Муравьёв К.А. |
| **Отчет по лабораторной работе №4**  «Инфраструктура в контейнерах» | | | | |
| Дата выполнения | 11.12.2022 | Дата защиты | | 24.12.2022 |
| Оценка |  | Подпись | |  |

Цель работы:

|  |
| --- |
| Получить навыки по развертыванию приложений в контейнерах. |

Задачи работы:

|  |
| --- |
| Настроить инфраструктуру приложения в контейнерах, настроить автоматизированную инфраструктуру контейнеров. |
|  |

Краткий конспект теоретической части:

|  |
| --- |
| Контейнерная инфраструктура:  Контейнер – автономное программное обеспечение, не зависящее от других приложений, что позволяет запускать приложения изолировано на любой машине.  Docker – программное обеспечение для автоматизации развертывания контейнеров.  Открытие портов в контейнерах:  Указать, на порт хоста можно с помощью ports в docker-compose.yaml.  Монтирование томов в контейнерах – производится указанием volumes в docker-compose.yaml.  Docker Compose: инструмент для развертывания контейнеров. Параметры запуска всех контейнеров будут описаны в одном файле, и все контейнеры запускаются одной командой. Удобен в применении в работе с несколькими контейнерами.  Взаимодействие сервисов в docker compose: сервисы — это самостоятельные службы, работающие независимо в отдельных контейнерах. Docker Compose не только автоматически запускает или останавливает контейнеры, но и поддерживает их жизненный цикл, обеспечивает совместное использование ресурсов, тем самым позволяя сервисам поддерживать друг друга. |

**1 Docker**

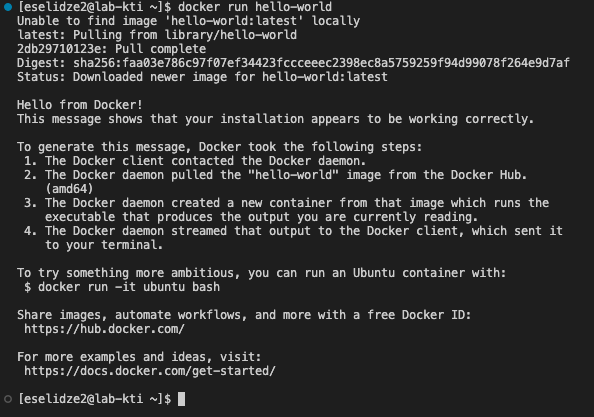
В данной работе мы рассмотрим иной взгляд на виртуализацию приложений, а именно - использование для этих задач контейнеры.

Создадим вторую виртуальную машину, на которой будем разворачивать инфраструктуру. Установим на новую виртуальную машину docker engine для запуска контейнеров (Рисунок 1.1).

Рисунок 1.1 –установка docker engine.

Проверим корректность работы докера командой docker run hello-world

(Рисунок 1.2).

Рисунок 1.2 – Запуск тестового контейнера.

Рассмотрим еще один инструмент для работы с контейнерами – docker compose. Проверим его версию и факт установки (Рисунок 1.3).



Рисунок 1.3 – Версия docker compose.

Создадим для сборки образа flask Dockerfile следующего содержания (Рисунок 1.4).



Рисунок 1.4 – Содержание Dockerfile.

Создадим образ flask командой docker build -t eselidze2/flask:1.0 . (Рисунок 1.5).

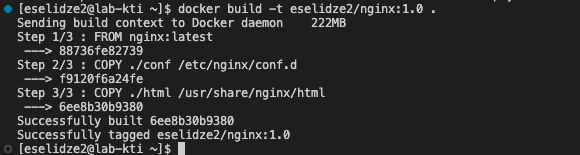


Рисунок 1.5 – Процесс создания образа flask.

Создадим в домашней директории каталог app и файл app.py для нашего проекта (Рисунок 1.6).

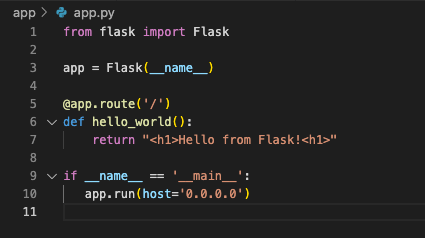


Рисунок 1.6 – Содержимое файла app.py.

Напишем файл docker-compose.yaml в том же каталоге, что и Dockerfile (Рисунок 1.7).

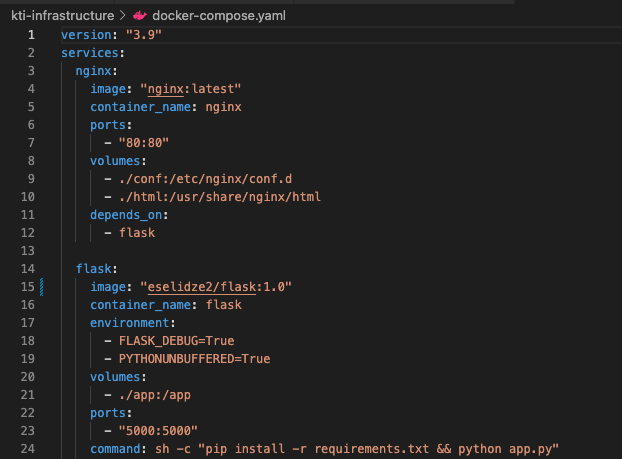


Рисунок 1.7 – Содержание docker-compose.yaml.

Запустим проект командой docker compose up -d (Рисунок 1.8).

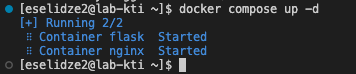


Рисунок 1.8 – Факт запуска контейнеров flask и nginx.

Откроем сайт по IP-адресу нашей виртуальной машины (Рисунок 1.9).



Рисунок 1.9 – Проверка работоспособности проекта.

Изменим Dockerfile для сборки образа с системными пакетами (Рисунок 1.10).

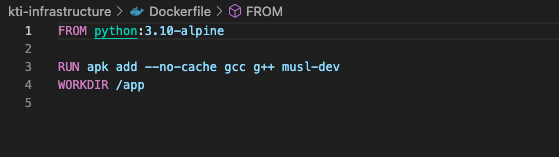


Рисунок 1.10 – Содержимое Dockerfile.

Соберем новый образ flask уже известной нам командой (Рисунок 1.11).

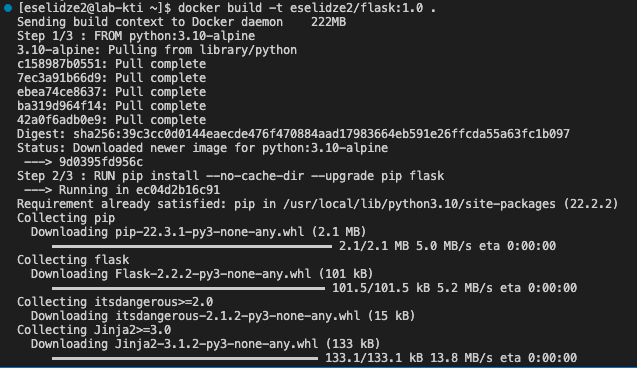


Рисунок 1.11 – Создание образа flask.

Напишем новый docker-compose.yaml, где укажем сервисы nginx, python и postgresql (Рисунок 1.12).

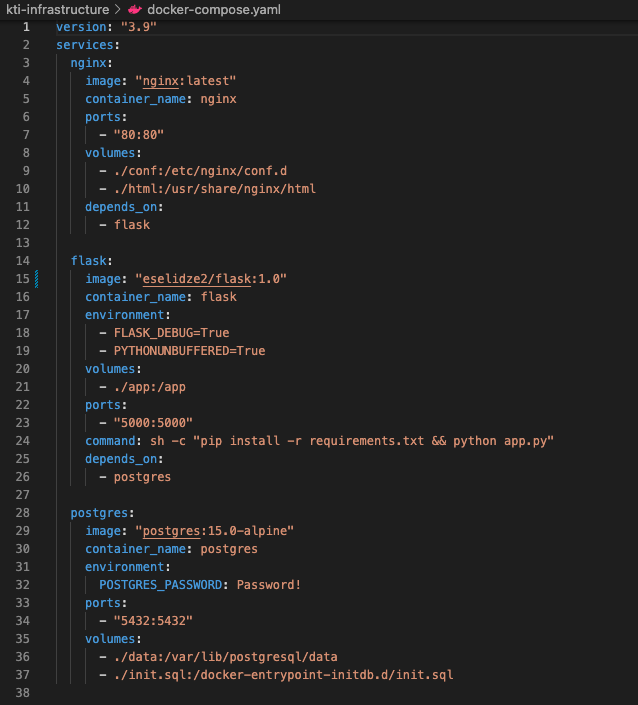


Рисунок 1.12 – Содержимое файла docker-compose.yaml.

Инициализация базы и создание таблиц будет происходить при запуске контейнера postgresql. В момент запуска СУБД считывает файлы из директории /docker-entrypoint-initdb.d, в которую можно смонтировать файл с необходимым скриптом (Рисунок 1.13).

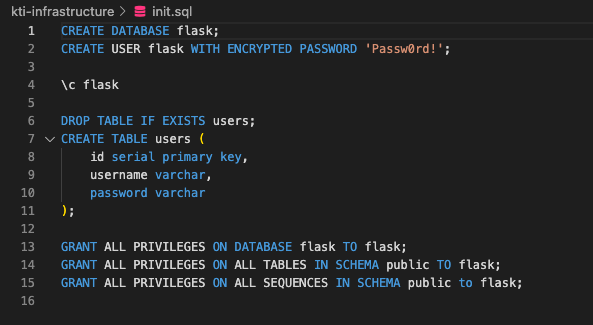


Рисунок 1.13 – Инициализированный скрипт.

Вернемся на первую виртуальную машину и внесем изменения в конфиг nginx, который будет выступать в роли балансировщика при перенаправлении запросов с одной виртуальной машины на другую (Рисунок 1.14).

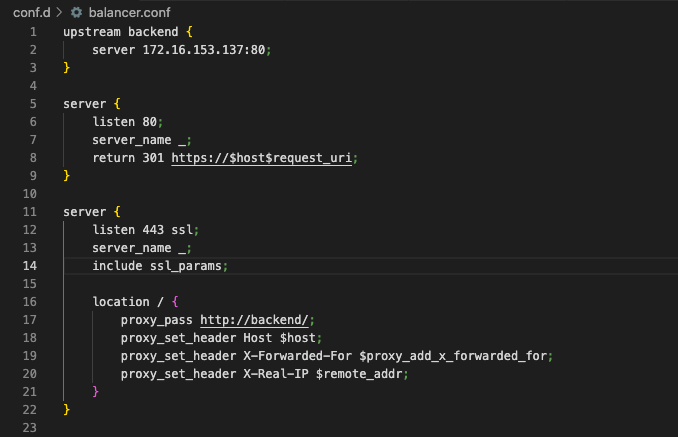


Рисунок 1.14 – Настройка передачи запросов с одной виртуальной машины на другую.

После проверки доступности апстрима проверим правильность написания конфига и перечитаем (Рисунок 1.15).

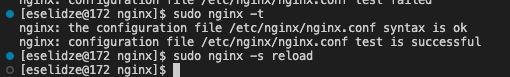


Рисунок 1.15 – Перезапуск сервера nginx.

Проверим работоспособность нашего приложения на обеих виртуальных машин (Рисунок 1.16 и 1.17).

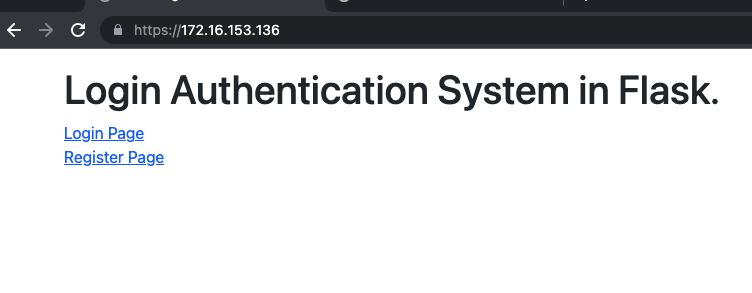


Рисунок 1.16 – Работоспособность приложения по IP-адресу первой виртуальной машины.

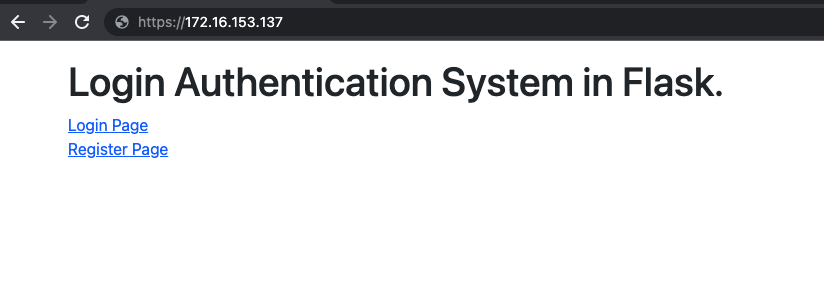


Рисунок 1.17 – Работоспособность приложения по IP-адресу второй виртуальной машины.

Продемонстрируем логи всех трех сервисов (Рисунок 1.18, 1.19, 1.20).

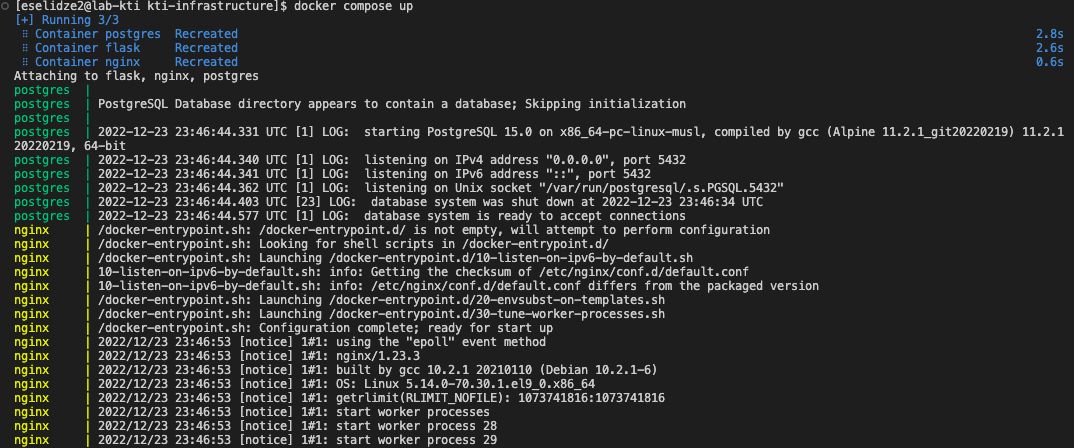


Рисунок 1.18 – Логи сервисов nginx и postgresql.

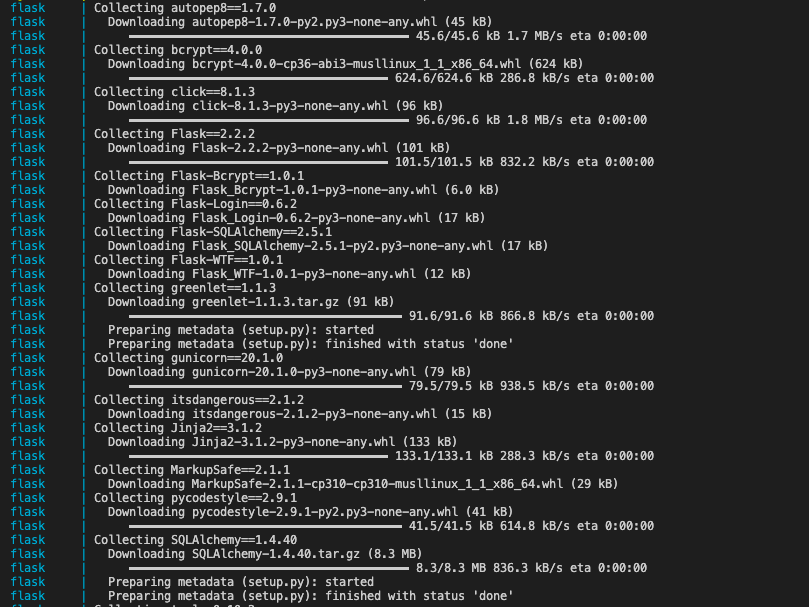


Рисунок 1.19 – Логи сервиса flask.

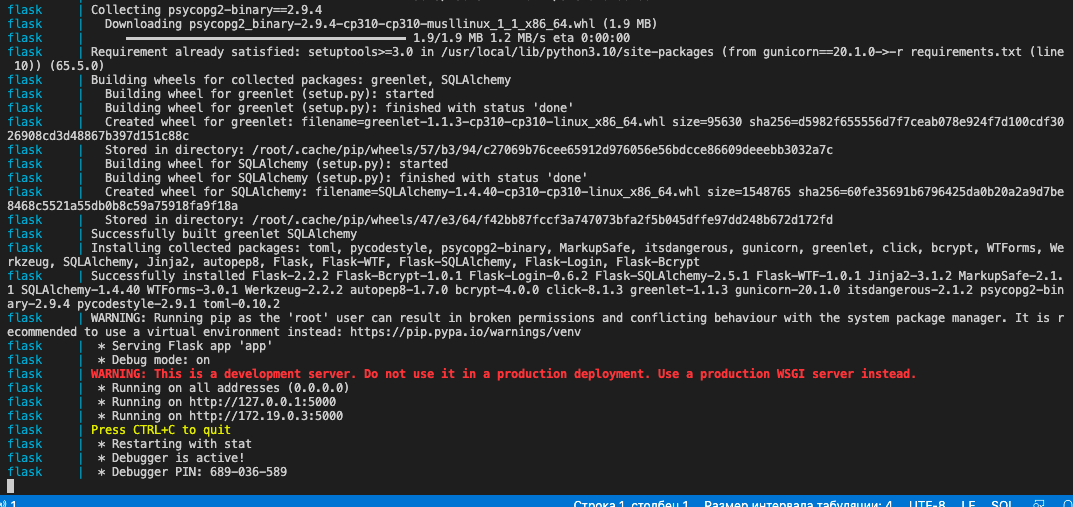


Рисунок 1.20 – Логи сервиса flask.

Продемонстрируем запросы с первой виртуальной машины (Рисунок 1.21).

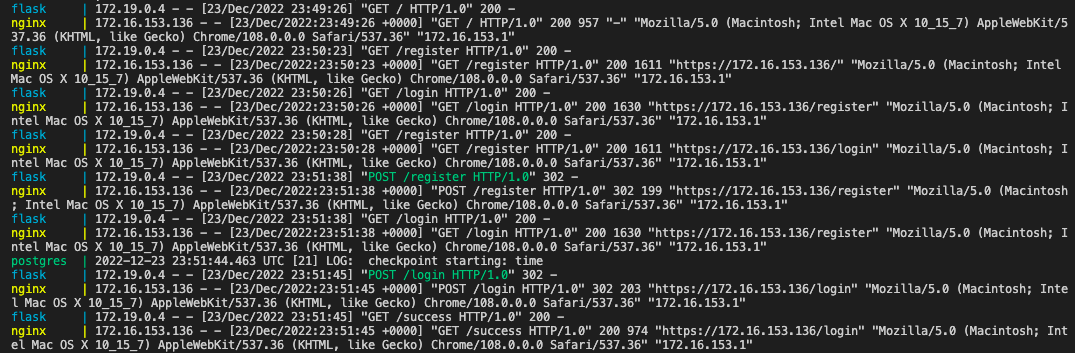


Рисунок 1.21 – Перенаправление запросов между виртуальными машинами.

На данном этапе построена масштабируемая платформа для развертывания веб-приложений с легко заменяемыми компонентами (всегда возможно использовать иную СУБД или другой язык программирования), поддерживающая балансировку нагрузки и подключение по безопасным протоколам извне.

**2 Автоматизация**

Приступим к написанию скрипта для автоматического развертывания инфраструктуры в контейнерах.

После создания двух репозиториев на GitHub с кодом приложения и инфраструктуры удалим все, что было создано за время работы (Рисунок 2.1 и 2.2).

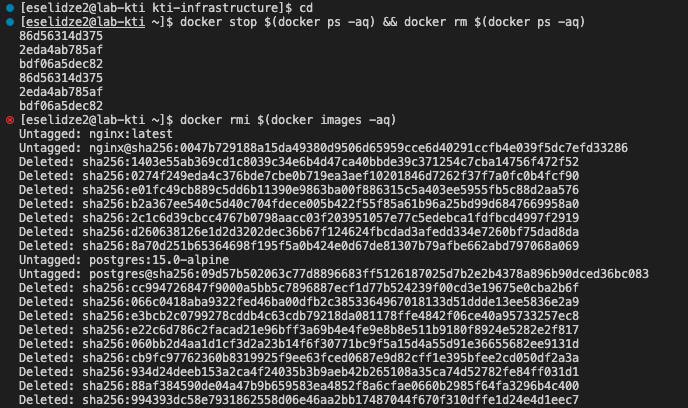


Рисунок 2.1 – Удаление контейнеров и их образов.

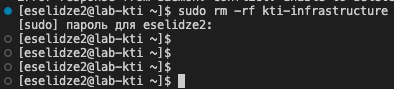


Рисунок 2.1 – Удаление инфраструктуры.

Создадим вышеупомянутый скрипт следующего содержания (Рисунок 2.3).

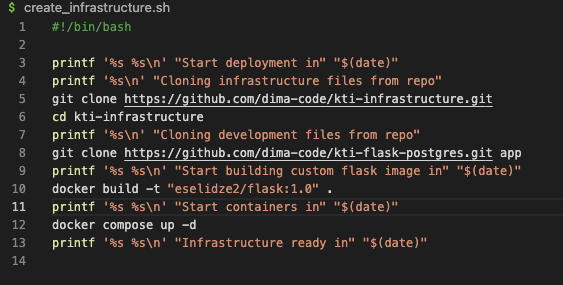


Рисунок 2.2 – Содержание скрипта для автоматизации.

Предварительно выдав скрипту права на выполнение, запустим его (Рисунок 2.3 и 2.4).



Рисунок 2.3 – Развертывание инфраструктуры.

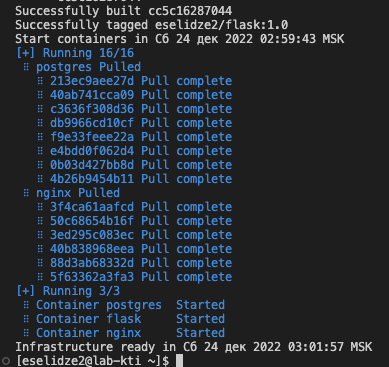


Рисунок 2.4 – Собирание образов и запуск контейнеров

Готово! Инфраструктура была развернута за ~5 минут.

Составим структурную схему развернутой на двух виртуальных машинах инфраструктуры (Рисунок 2.5).

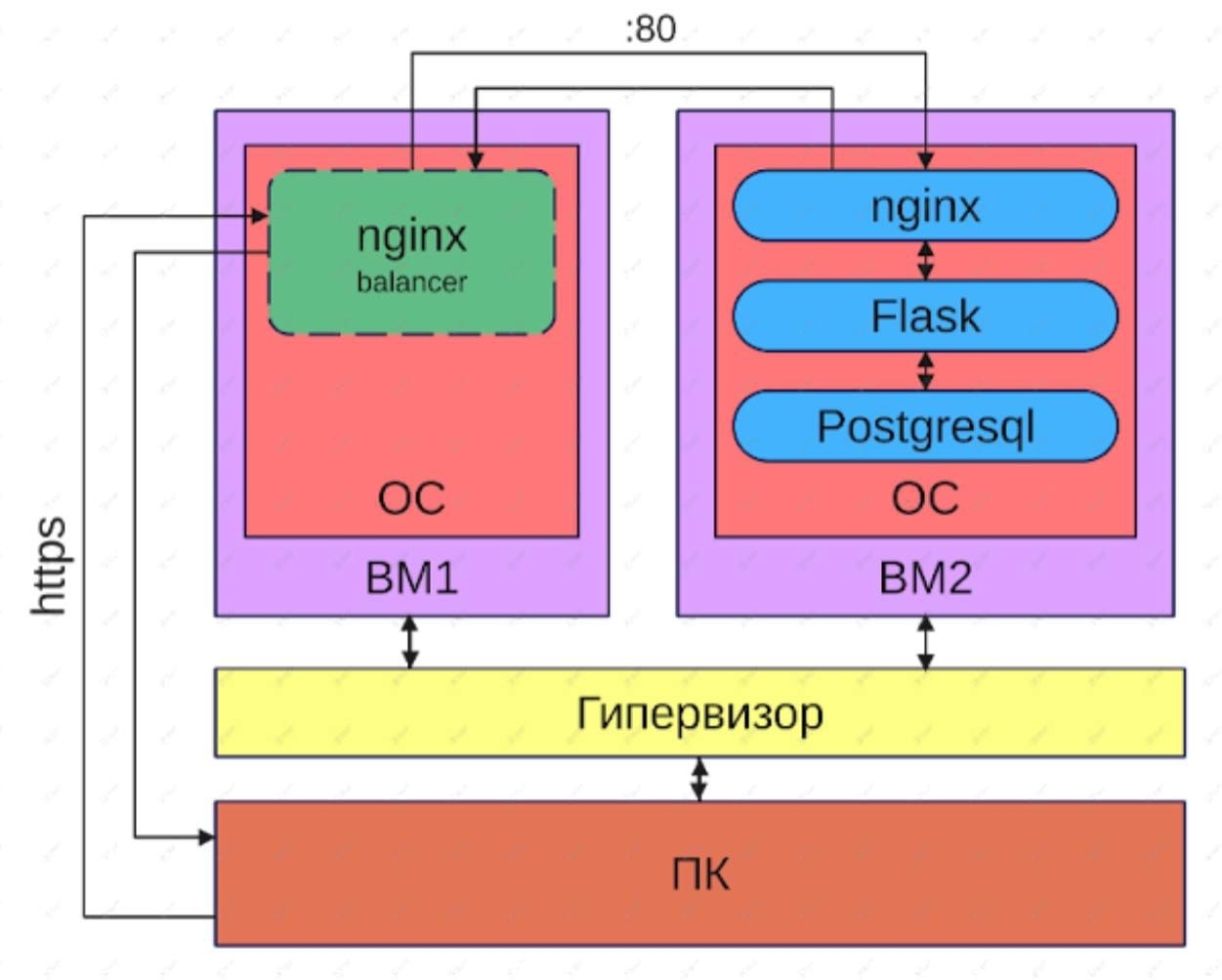


Рисунок 2.5 – Структурная схема.

Вывод:

В ходе работы мы получили навыки по развертыванию инфраструктуры в контейнерах, а также создали масштабируемую платформу для развертывания веб-приложений, поддерживающую балансировку нагрузки и подключение по безопасным протоколам извне с автозапуском.