Курс DevOps

Лабораторная работа №4

Docker Compose

1 апреля 2026 г.

Общие инструкции

Рекомендации к выполнению

- Настоятельно рекомендуется выполнять команды из листинга отдельно, одну за другой.
- Рекомендуется копировать команды через промежуточную вставку в текстовый редактор во избежание лишних символов.
- Перед началом работы ознакомтесь с инструкцией по сдаче лабораторных работ.

Требования к выполняемой работе

- Добавьте в репозиторий все необходимые файлы: код приложения, файлы конфигурации и скрипты и т.д.

Внимание: Данная лабораторная работа является продолжением лабораторной работы №3. Для выполнения данной работы необходимо иметь подготовленный репозиторий из третьей лабораторной работы.

1 Цель работы

Получить базовые навыки работы с Docker Compose, научиться управлять мультиконтейнерными приложениями.

2 Теоретические сведения

2.1 Что такое мультиконтейнерное приложение

Мультиконтейнерное приложение — это система, состоящая из нескольких контейнеров, каждый из которых выполняет отдельную функцию.

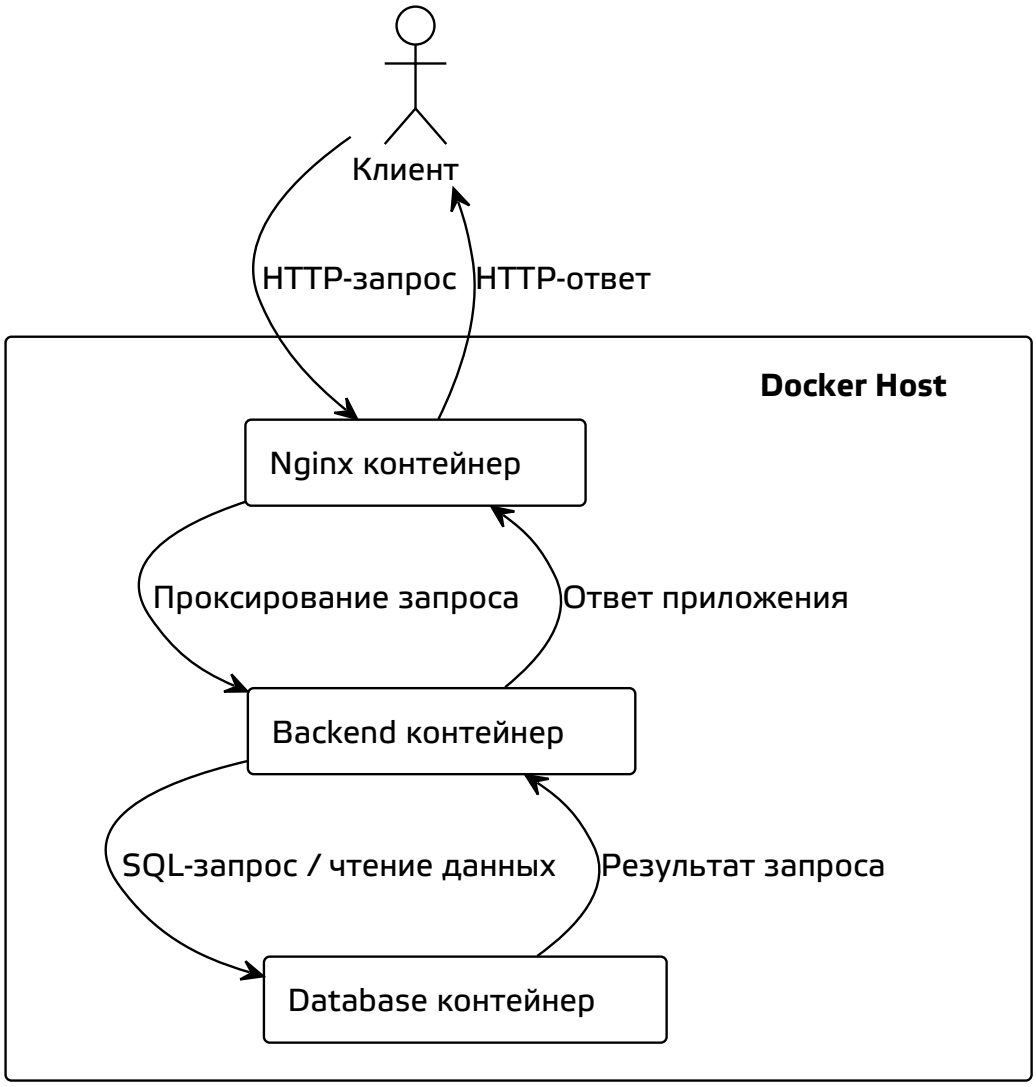

Рис. 1: Типичное мультиконтейнерное приложение

В отличие от монолитного подхода, где все компоненты приложения объединены в одном контейнере, мультиконтейнерный подход позволяет разделить систему на несколько изолированных компонент. Такой подход упрощает обновление, масштабирование и сопровождение приложения, а также повышает его отказоустойчивость.

Важно отметить, что сама архитектура Docker подразумевает принцип «один контейнер — один компонент». Каждый контейнер должен выполнять строго определённую задачу, что обеспечивает изоляцию, предсказуемость поведения и упрощает управление зависимостями между сервисами.

Мультиконтейнерные приложения реализуют этот принцип на практике, объединяя несколько контейнеров в единую систему через инструменты оркестрации, такие как Docker Compose.

2.2 Что такое Docker Compose

Docker Compose - это инструмент для определения и запуска мультиконтейнерных приложений Docker. С его помощью все компоненты приложения описываются в одном файле docker-compose.yml, где указываются образы, зависимости между контейнерами, сети и volumes для каждого контейнера.

Современные версии Docker Compose поставляются в качестве плагина для CLI утилиты docker. После установки плагина, docker compose может быть вызван командой `docker compose`.

2.3 Основные понятия Docker Compose

- Стек (Stack) — описание мультиконтейнерного приложения: используемые сервисы, сети, volumes и прочее
- Сервис (Service) — компонент мультиконтейнерного приложения. Сервис состоит из описания создаваемого контейнера и метаинформации о связи с другими сервисами (при её наличии)
- Сеть (Network) — сетевое пространство для создаваемых контейнеров
- Volumes — описание используемых volumes для создаваемых контейнеров
- Compose файл — файл docker-compose.yaml, где описывается структура мультиконтейнерного приложения

2.4 Основные команды docker compose

Docker compose позволяет управлять мультиконтейнерным приложением следующим образом:

- docker compose up — запуск всего приложения (всех сервисов)
- docker compose stop — остановка всего приложения (всех сервисов)
- docker compose down — остановка всего приложения с последующим удалением контейнеров, сетей и volumes

- docker compose ps — просмотр запущенных сервисов

Для более подробного описания всех команд и их опций можно обратиться к официальной документации Docker.

2.5 Директивы compose файла

В файле docker-compose.yml существуют директивы верхнего уровня, которые определяют структуру всего приложения и его общие ресурсы.

Основные директивы верхнего уровня:

- **name** - название приложения
- **services** - основной раздел, в котором описываются все сервисы приложения
- **volumes** - определение volumes, используемых сервисами
- **networks** - определение сетей для взаимодействия сервисов

Весь список директив верхнего уровня можно найти в документации docker compose.

```
services:
  frontend:
image: example/webapp
networks:
  - front-tier
  - back-tier
backend:
image: example/backend
volumes:
  - db-data:/etc/data
networks:
  - back-tier
volumes:
  db-data:
networks:
  front-tier:
  back-tier:
```

Листинг 1: Пример определения мультиконтейнерного приложения в docker-compose.yaml

2.5.1 Service раздел

Раздел service является единственным обязательным разделом docker-compose.yaml файла и не может быть пустым.

Основные директивы раздела service:

- image - имедж, из которого будет создан контейнер
- ports - определение port mappings между хостом и контейнером

- environment - определение переменных окружения, которые будут установлены в окружении контейнера
- volumes - список точек монтирования данных в контейнерах. Данные могут быть смонтированы как из Docker volumes, так и из директорий хоста

Весь список директив раздела service можно найти в в документации docker compose.

2.5.2 Networks раздел

Раздел networks не является обязательным и может отсутствовать в docker-compose.yaml файле. Если раздел network не указан, то Docker Compose автоматически создаст сеть с названием default типа bridge и все создаваемые контейнеры автоматически будут подключены в эту сеть.

Весь список директив раздела networks можно найти в в документации docker compose.

2.5.3 Volumes раздел

Раздел volumes не является обязательным и может отсутствовать в docker-compose.yaml файле.

Весь список директив раздела volumes можно найти в в документации docker compose.

3 Содержание работы

В данной работе необходимо перейти к работе с приложением через Docker Compose. Для этого следует описать приложение в файле docker-compose.yaml, а его развертывание в CD пайплайне должно выполняться командой docker compose up.

3.1 Задание 1

Необходимо убедиться, что на виртуальной машине установлен Docker Compose с помощью команды docker compose version. В случае отсутствия Docker Compose - установите его по инструкции.

3.2 Задание 2

Опишите используемое приложение в файле docker-compose.yaml и поместите файл в репозиторий.

3.3 Задание 3

В CD пайплайне перейдите на развертывание приложения через docker compose.

Примечание:

- Приложение, которые вы используете в данной работе, должно являться мультиконтейнерным.
- В используемом приложении должен быть персистентный слой, например, база данных.
- При пересоздании стека (например, в случае обновления имеджа базы данных) вы не должны терять данные приложения. Для решения этой задачи рекомендуется использовать docker volumes.
- Контейнеры с приложением должны запускаться из заранее собранного образа. Не монтируйте исходный код приложения напрямую с хоста в контейнер (bind mount) для запуска. Это нарушает принцип «один контейнер — один компонент» и усложняет повторяемость окружения.
- Если ваше приложение имеет интеграционные тесты (тесты, которые требуют запущенное приложение — E2E, UI, API), в данной работе допустимо не запускать их. Для желающих можно настроить их запуск через service containers или через запуск тестов после делпоя новой версии на работающем приложении.

Мультиконтейнерные версии рекомендуемых приложений:[1]

- **Статический сайт:** prafdin/website-example
- **C# приложение:** danlla/Csharp-example/tree/multicontainers
- **Python веб приложение:** prafdin/catty-reminders-app/tree/multicontainers

4 Контрольные вопросы

1. Как Docker Compose позволяет управлять мультиконтейнерными приложениями?
2. Как определить порядок запуска сервисов в Docker Compose? В какой ситуации это может потребоваться?
3. Чем отличается Docker Compose сервис и docker контейнер?
4. В чем разница между volumes и bind mounts?
5. Как масштабировать Docker Compose сервис? Например, увеличить количество контейнеров сервиса frontend до 3.

[1] Инструкция по переходу к мультиконтейнерной версии рекомендуемых приложений

6. Что такое pull_policy в определении Docker Compose сервиса и для чего это нужно?

7. Благодаря чему контейнеры из Docker Compose стека могут обращаться к другим контейнерам по имени?

5 Инструкция по переходу к мультиконтейнерной версии рекомендуемых приложений

Для перехода к мультиконтейнерной версии приложения необходимо настроить удалённый репозиторий с помощью механизма git remotes и слить вашу и удаленную версию репозитория с помощью механизма git merge или git rebase.

```
# Подключение удаленного репозитория
git remote add upstream git@github.com:prafdin/devops-course-labs.git

# Получаем обновления из удаленного репозитория
git fetch upstream

# Переходим в нужную ветку
git checkout lab4

# Сливаем новую и старую версию, с учетом текущих изменений в вашей ветке.
# Замените ветку main в catty-reminders-app-multicontainers на необходимую,
# согласно названию вашего приложения <app_name>-multicontainers.

git rebase upstream/catty-reminders-app-multicontainers

# При слиянии возможно возникновение конфликтов: после ввода команды git rebase
# в ответ вы получите "... CONFLICT (content): ...".
# Вывод команды git status покажет "... interactive rebase in progress ...".
# Рекомендуется осознано решить конфликты. Например, можно открыть репозиторий в
# IDE (VS code, Intellij idea другие) и использовать интерактивное решение конфликтов.
# Также вы можете решить конфликты в пользу удаленного репозитория
# выполняя следующую группу команд, пока не получите в ответ "... Successfully rebased ..."
git checkout --ours .
git add .
git rebase --continue

# Учтите, что если ветка lab4 уже существовала в вашем удаленном репозитории,
# то выполнить пуш после git rebase не получится.
# Для пуша новой версии воспользуйтесь флагом --force. Используйте её с особой
# осторожностью и только после проверки локальной копии репозитория.
git push origin lab4 --force-with-lease
```

Также вы можете перейти к новой версии приложения путем копирования всех файлов из необходимой ветки (предварительно сохраните локальную копию!). Однако, данный метод не является рекомендуемой практикой из-за большого количества ручных манипуляций.