Docker Compose

29 октября 2025 г.

Развитие приложения: новое требование

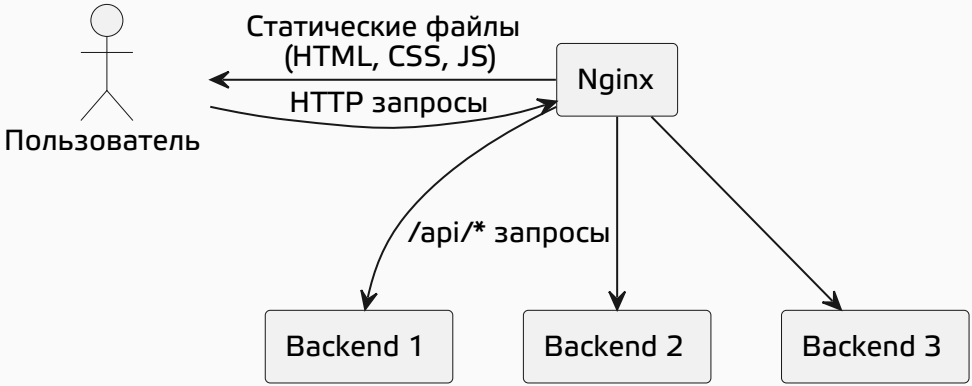
Текущее состояние нашего приложения:

- nginx работает в контейнере и раздает html
- Сервер настраивается через ansible
- Настроен автоматический деплой через CI/CD pipeline

Новое требование:

Добавить вызов backend API для работы с динамическими данными.
Важный критерий — высокая доступность.

Целевая архитектура



Что нам нужно запустить?

Для реализации нашей архитектуры потребуется:

- **1 nginx контейнер** — веб-сервер и load balancer
- **3 backend контейнера** — Python Flask API
- **Docker сеть** — для связи между контейнерами

Как это запустить?

Попробуем запустить вручную

Последовательность команд:

```
docker network create demo-net
docker run -d --rm --network demo-net --name backend-1 busybox:stable sleep 250
docker run -d --rm --network demo-net --name backend-2 busybox:stable sleep 250
docker run -d --rm --network demo-net --name backend-3 busybox:stable sleep 250
docker run -d --rm --network demo-net -p 8080:80 --name nginx nginx:alpine
```

Проблемы: много команд, можно ошибиться

Docker Compose

Docker Compose — инструмент для определения и запуска многоконтейнерных приложений

Ключевые возможности:

- **Декларативное описание** — вся архитектура в одном YAML файле
- **Простое управление** — одна команда для запуска всего стека
- **Автоматическое масштабирование** — легко увеличить количество экземпляров
- **Управление зависимостями** — контейнеры запускаются в правильном порядке

Результат: вместо N команд docker — одна команда

Команды жизненного цикла

Основные команды Docker Compose:

- **docker compose up** — запустить все сервисы
- **docker compose up -d** — запустить в фоновом режиме
- **docker compose down** — остановить и удалить контейнеры
- **docker compose build** — пересобрать образы
- **docker compose pull** — загрузить новые версии образов

Команды мониторинга:

- **docker compose ps** — статус всех сервисов
- **docker compose logs** — логи всех сервисов

Масштабирование в Compose

Имя контейнера: <project-name>-<service>-#

docker compose up -p app --scale backend=3

Будут созданы три контейнера:

app-backend-1
app-backend-2
app-backend-3

Много других возможностей

- Сети — изоляция и связь между сервисами
- Секреты — безопасное управление паролями и ключами
- Volumes — постоянное хранение данных
- Health checks — мониторинг состояния сервисов
- Зависимости — контроль порядка запуска сервисов

Полная документация: docs.docker.com/compose/

Несколько Docker Compose файлов

```yaml
# compose.yml
services:
  web:
    image: example/my_web_app:latest
    depends_on:
      - db
      - cache
  db:
    image: postgres:latest
  cache:
    image: redis:latest
```

```yaml
# compose.prod.yml
services:
  web:
    ports:
      - 80:80
    environment:
      PRODUCTION: 'true'
  cache:
    image: redis:prod
    environment:
      TTL: '500'
```

```yaml
# merged
services:
  web:
    image: example/my_web_app:latest
    ports:
      - 80:80
    depends_on:
      - db
      - cache
    environment:
      PRODUCTION: 'true'
  db:
    image: postgres:latest
  cache:
    image: redis:prod
    environment:
      TTL: '500'
```

https://docs.docker.com/compose/how-tos/multiple-compose-files/

Демонстрация: Docker Compose

Что покажем:

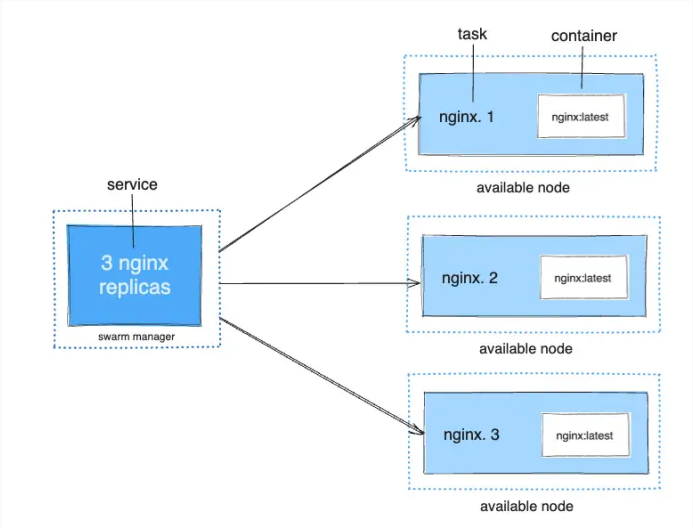
1. Настройка docker compose файла для схемы из начала лекции
2. Работа с несколькими файлами docker compose

Демо-репозиторий:
https://github.com/prafdin/devops-demo-website/tree/compose-demo

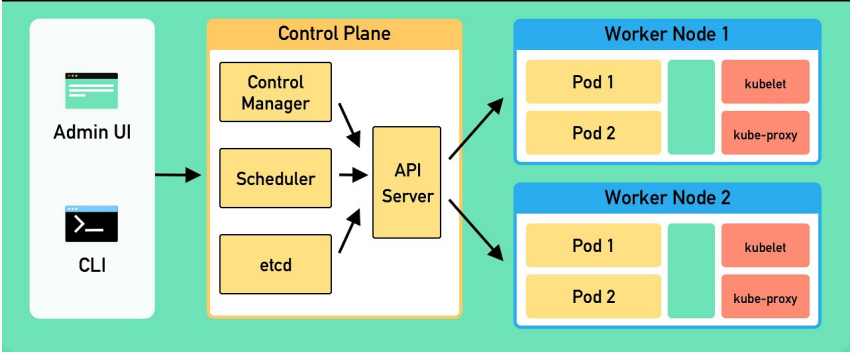
Альтернативы Docker Compose

- **Docker Swarm** — расширение функционала docker compose
- **Nomad** — от HashiCorp, поддерживает не только контейнеры
- **Kubernetes** — индустриальный стандарт оркестрации
- **OpenShift** — enterprise платформа на базе Kubernetes

Docker Swarm



What is Kubernetes?



Что запомнить

- Docker Compose решает проблему сложности управления множеством связанных контейнеров
- Декларативный подход лучше императивного — надёжнее и проще в поддержке

Вопросы?