CI/CD + Docker

Практические занятия

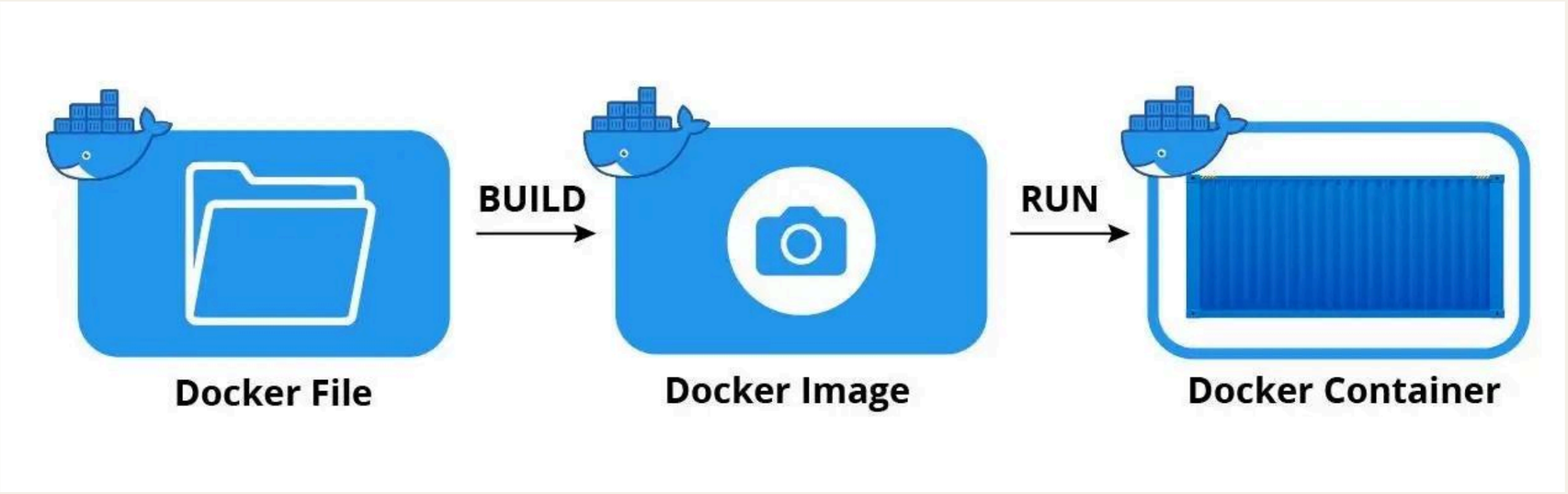
БИВТ-24-17

Надежда Анисимова

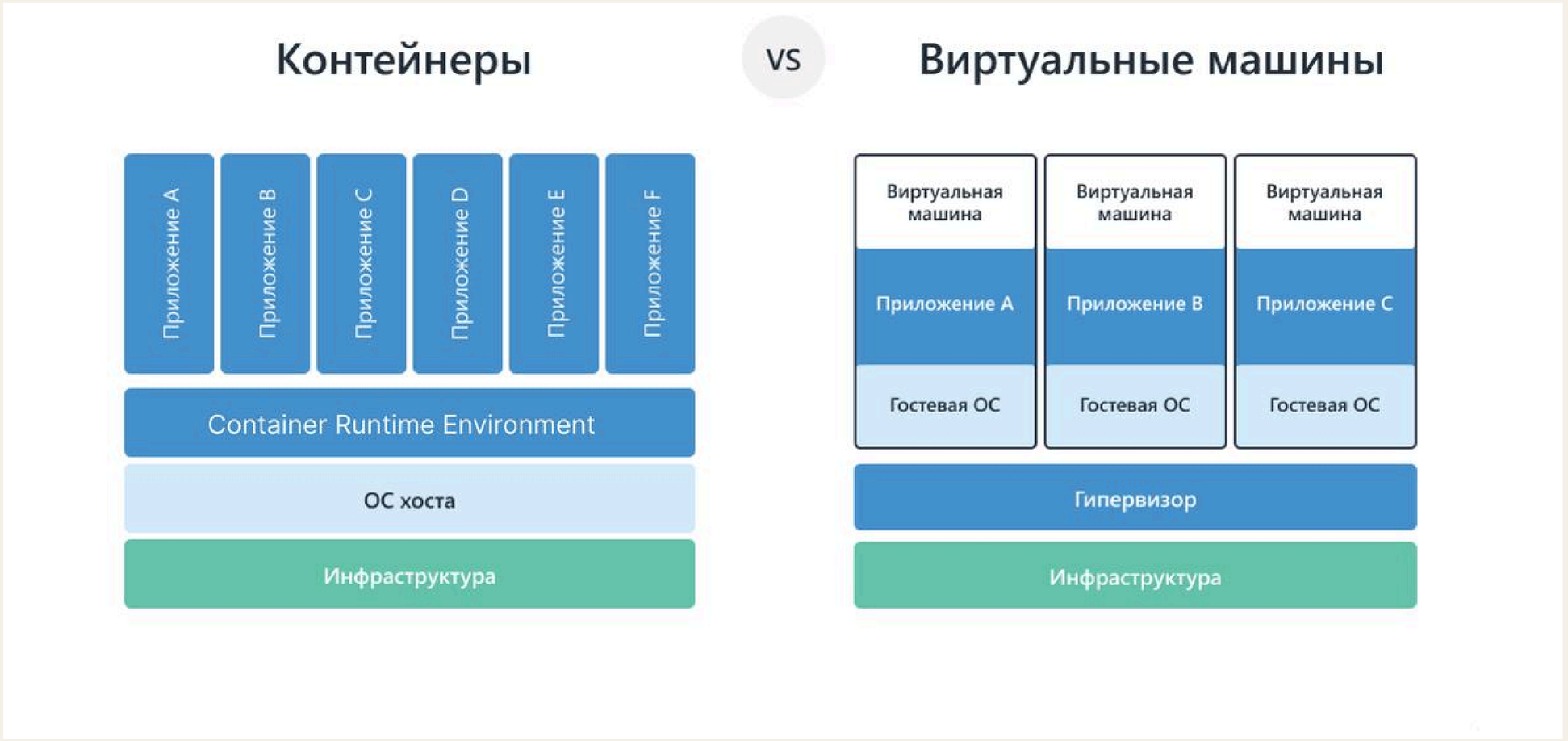
ms teams m2102039@edu.misis.ru

Docker

Docker — это инструмент, который позволяет упаковать приложение вместе со всеми его зависимостями (библиотеками, настройками, окружением) в единый контейнер, который можно запускать где угодно: на сервере, локальной машине, в облаке



Docker


Контейнеры
VS
Виртуальные машины
Приложение A
Приложение B
Приложение C
Приложение D
Приложение E
Приложение F
Container Runtime Environment
ОС хоста
Инфраструктура
Виртуальная машина
Виртуальная машина
Виртуальная машина
Приложение A
Приложение B
Приложение C
Гостевая ОС
Гостевая ОС
Гостевая ОС
Гипервизор
Инфраструктура

Docker – **зачем нужен**

- Приложение будет работать одинаково на любой машине (у тебя, у коллеги, на сервере).
- Упрощает деплой (выкатывание) приложений.
- Можно тестировать в разных окружениях (например, Python 3.8 и 3.11).
- Нет проблем с конфликтами зависимостей (у каждого контейнера — своё окружение).

Docker

Контейнер:

- Контейнер — это легковесный, изолированный процесс, который выполняет приложение.

Образ (Image):

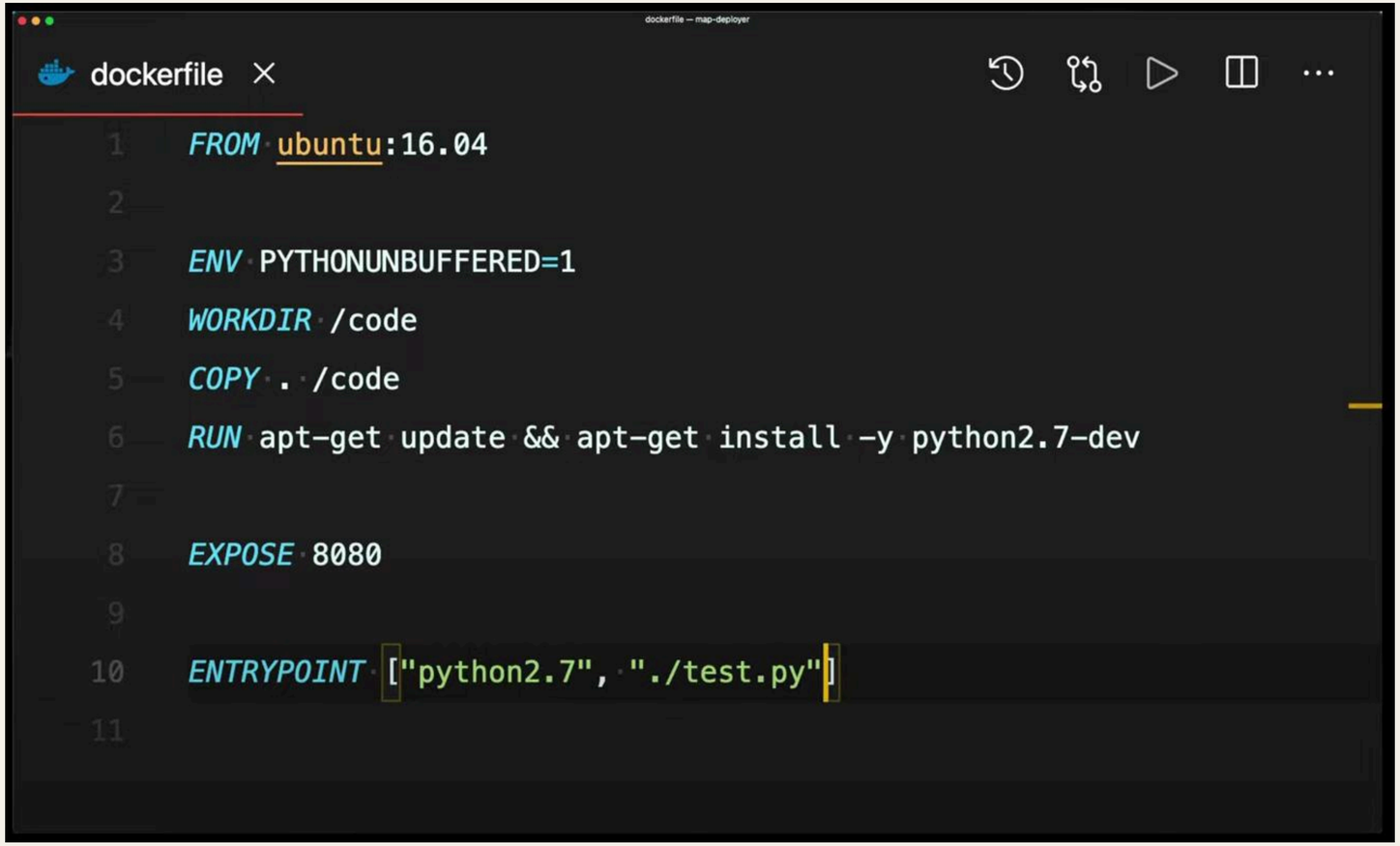
- Образ Docker — это "шаблон", на основе которого создаются контейнеры.

Dockerfile:

- Это текстовый файл, в котором содержатся инструкции по созданию Docker-образа.

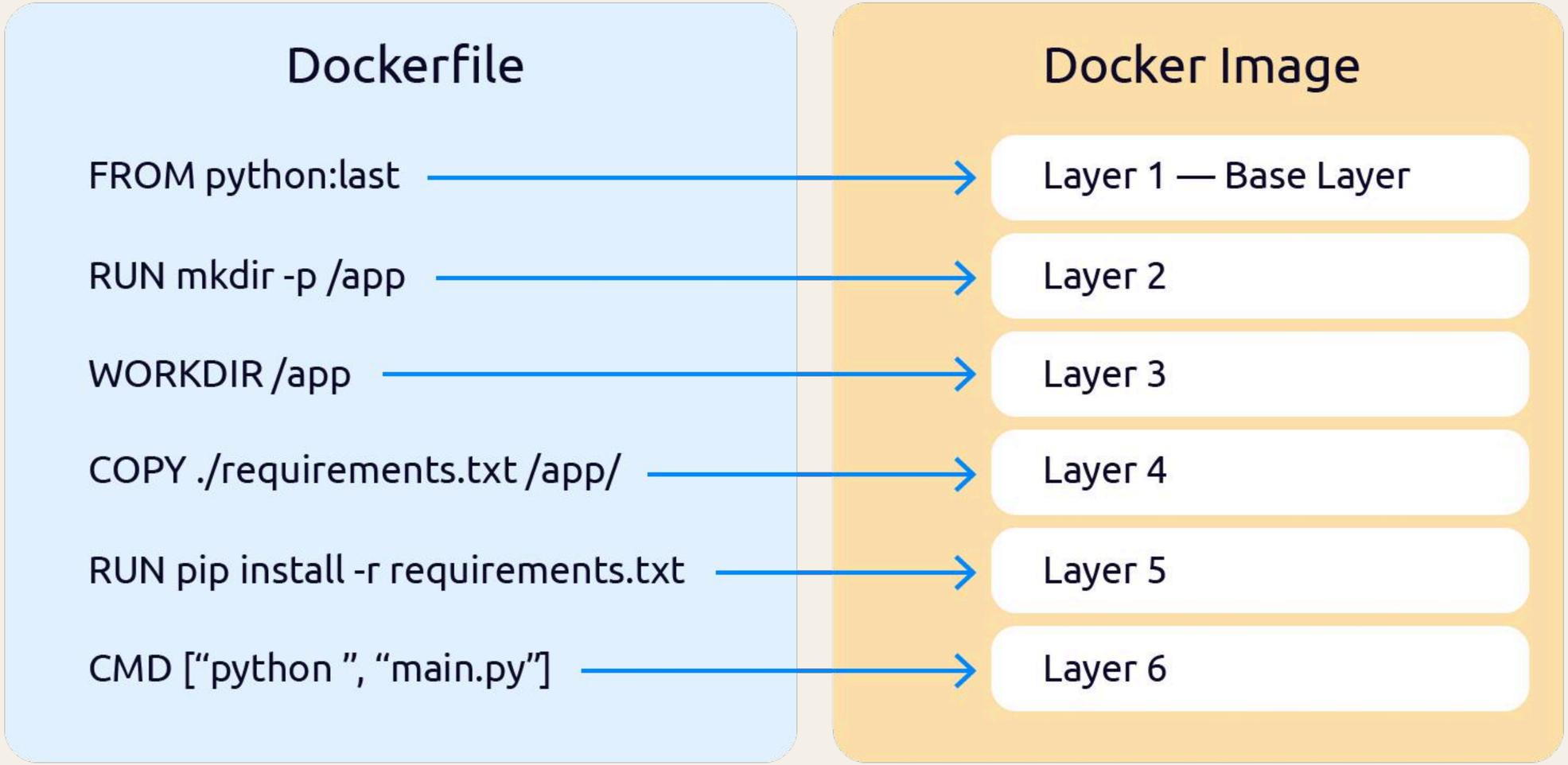
Рецепт создания контейнера через Dockerfile:

- берем базовый образ,
- копируем необходимые файлы
- устанавливаем зависимости,
- настраиваем рабочую директорию
- указываем команду для запуска приложения



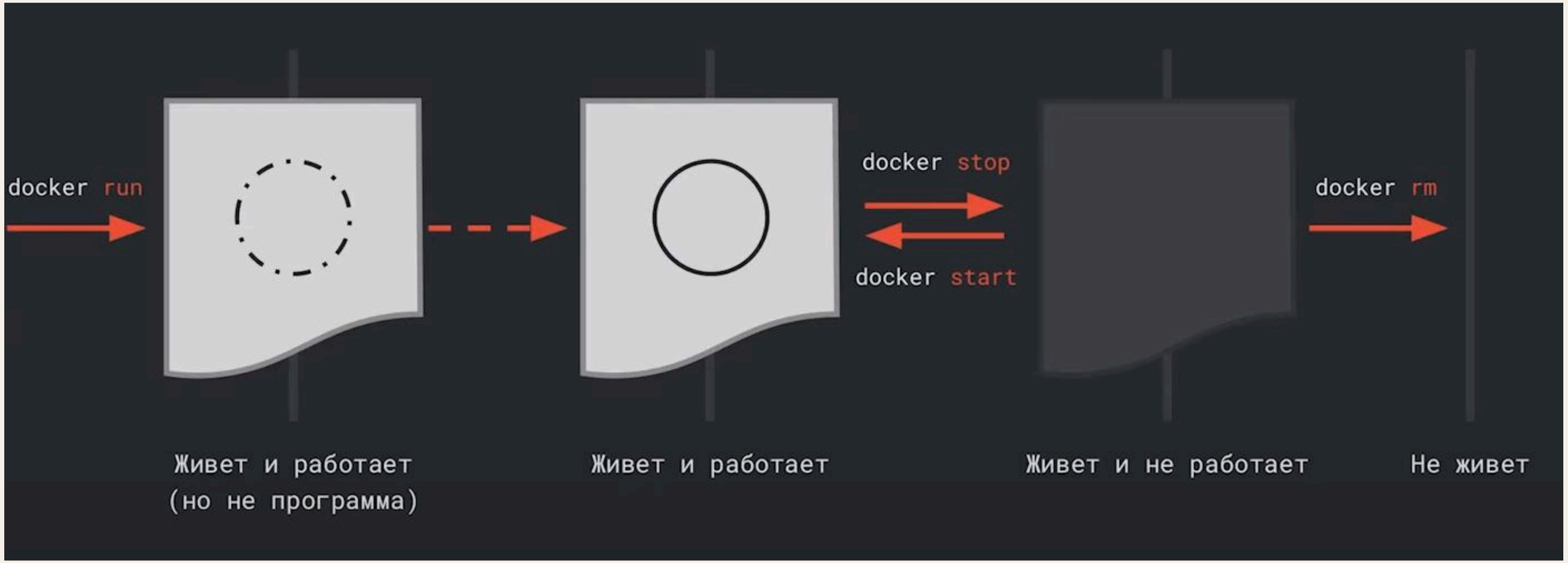
Dockerfile

Создаются слои и слои кешируются



Жизненный цикл контейнера

Остановили != удалили

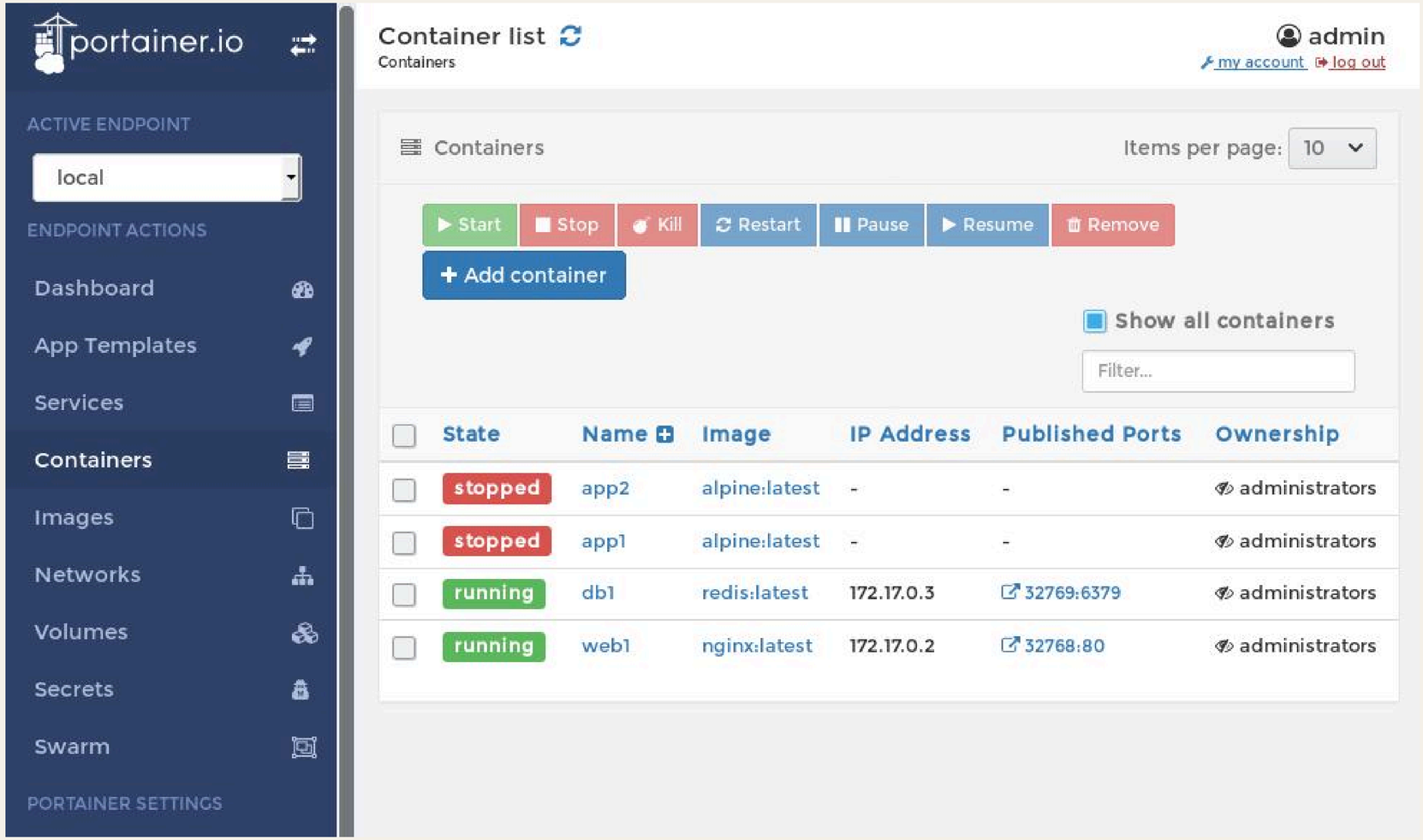


docker system prune

Удаляет все неиспользуемые контейнеры, сети, образы (как висячие, так и неиспользуемые) и, при необходимости, вольюмы

Docker

Отслеживать состояние контейнеров можно через интерфейс



Docker – **топ базовый команд**

docker run <образ> - Запускает контейнер из указанного образа.

docker images - Показывает список всех локальных Docker-образов.

docker ps -a - Показывает все контейнеры (работающие и остановленные).

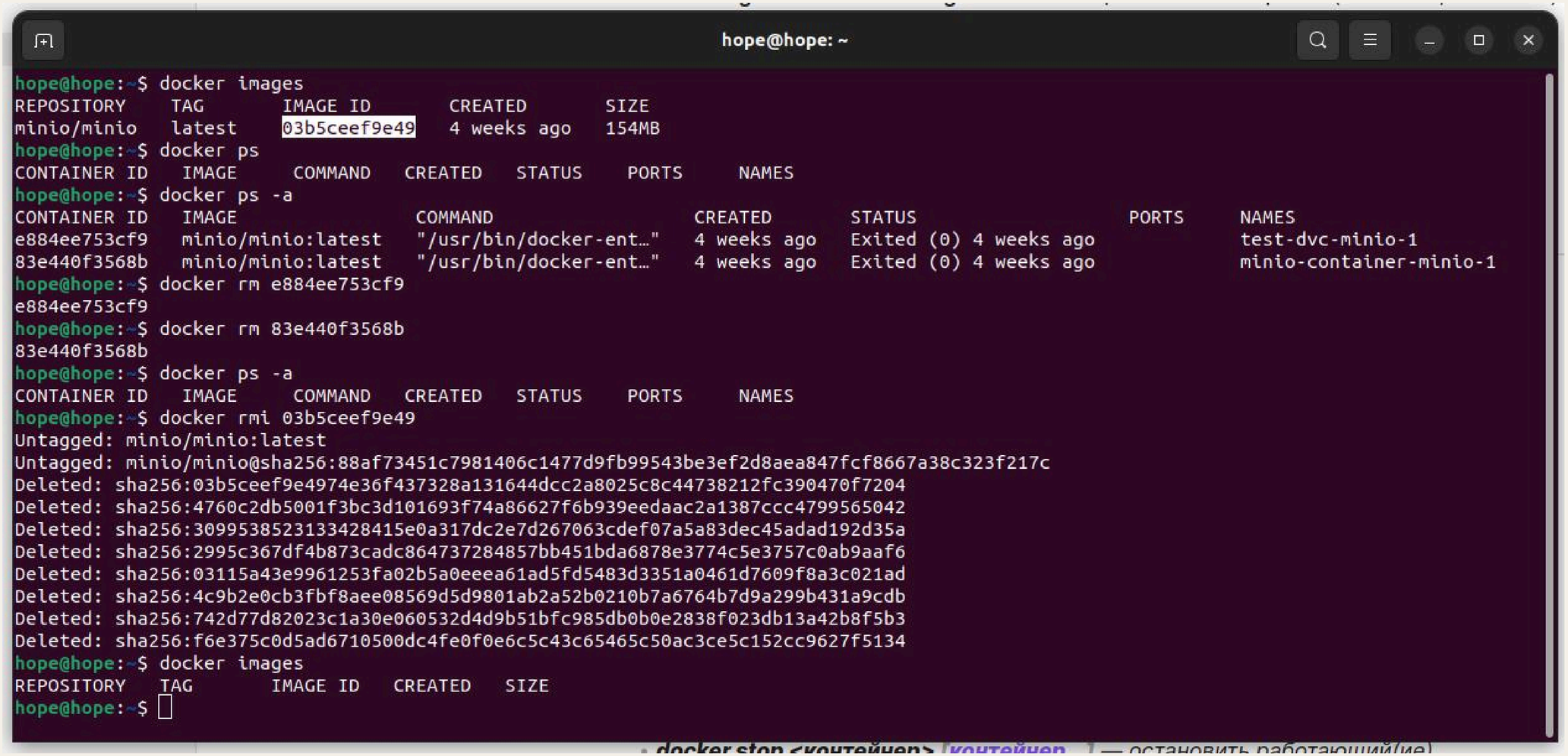
docker rmi $(docker images -q) - Удаляет все локальные Docker-образы.

docker rm $(docker ps -a -q) - Удаляет все контейнеры (работающие и остановленные)

docker exec -it <контейнер> **bash –** Запустить bash процесс и «войти» в контейнер

Docker

Просмотр/удаление контейнеров/образов



Docker-compose

позволяет определять и запускать несколько контейнеров с помощью одного конфигурационного файла

docker-compose up - Запускает контейнеры, указанные в docker-compose.yml, без пересборки образов.

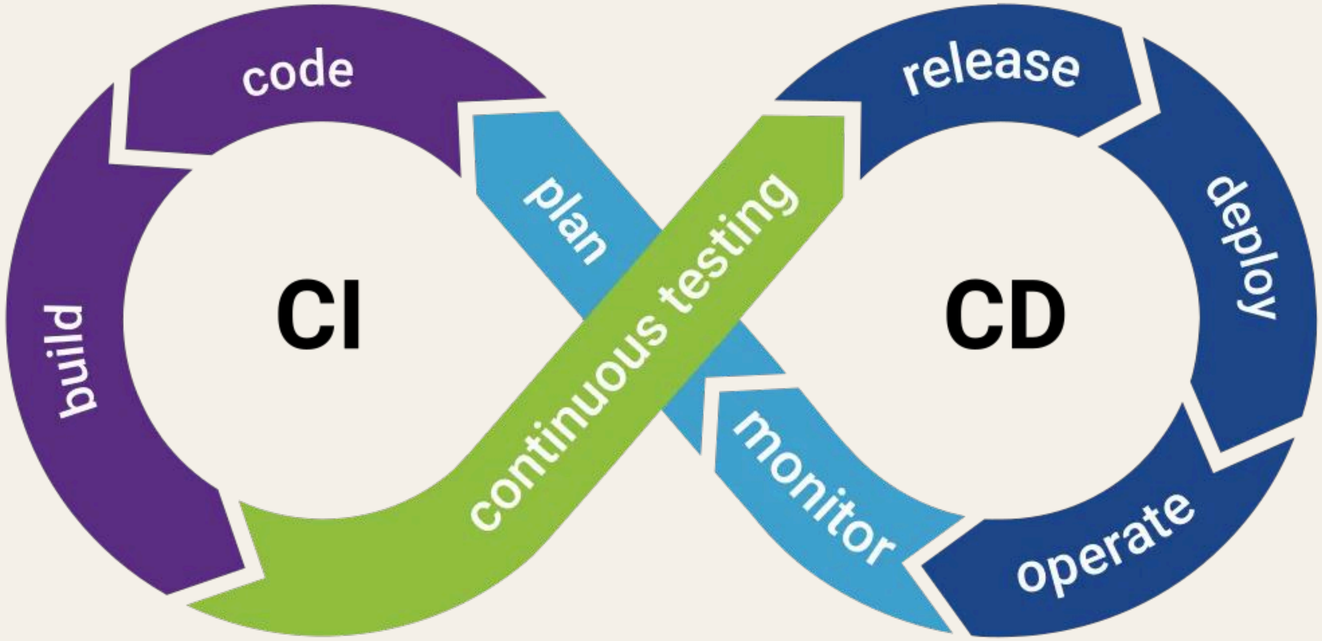
docker-compose up --build - Запускает контейнеры, пересобирая образы перед запуском, если они изменились

```yaml
version: '3.8'

services:
  python-app:
    build: .
    container_name: python-hello-world
    command: python /app/print_hello.py
    depends_on:
      - db

  db:
    image: postgres:latest
    container_name: postgres-db
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: example_db
```

CI/CD

CI = continuous integration = непрерывная интеграция

CD = continuous deployment = непрерывное развертывание



CI/CD

Автоматизация и ускорение процесса поставки кода от разработчиков до продакшн-среды

Continuous Integration (CI)

После каждого коммита запускаются автоматические тесты, линтеры и сборка проекта

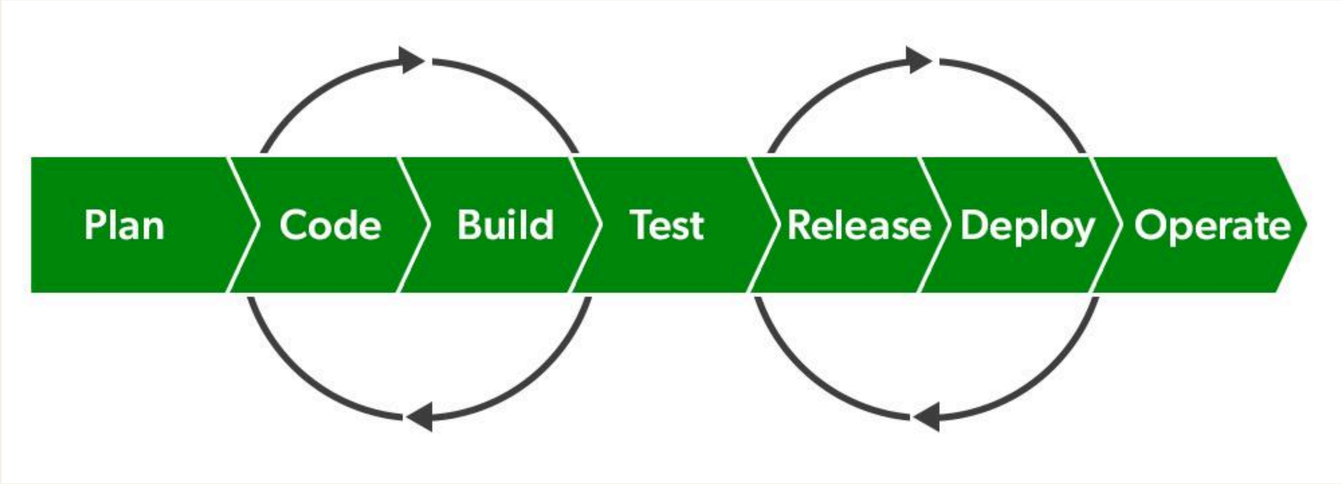
Continuous Delivery (CD)
Continuous Deployment (CD)

После CI артефакт (например, Docker-образ) автоматически готовится к релизу

Github Actions

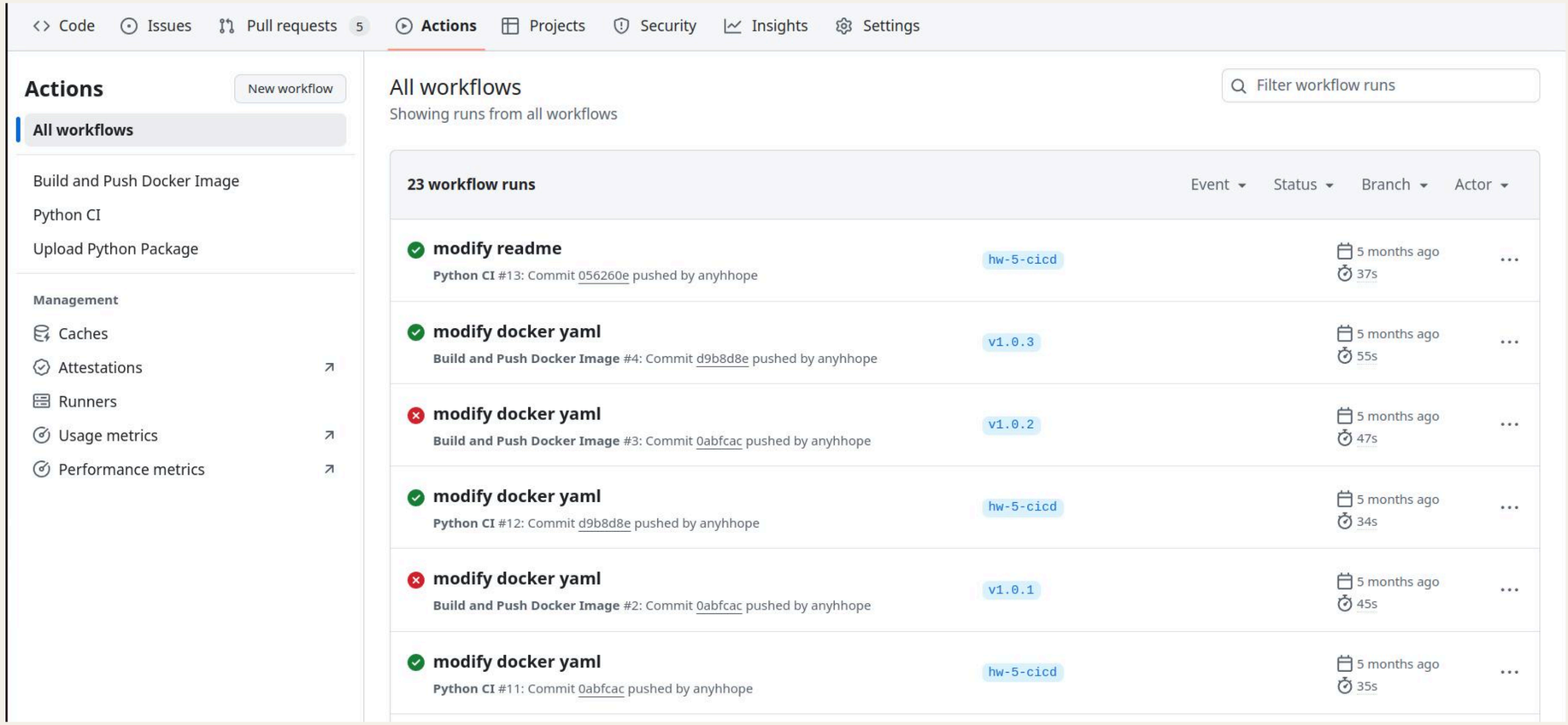
На гитхабе есть инструмент **Github Actions** позволяющий исполнить ci пайплайн для вашего кода

Например, автоматически запускать unit-тесты, каждый раз когда вы пушите свой код

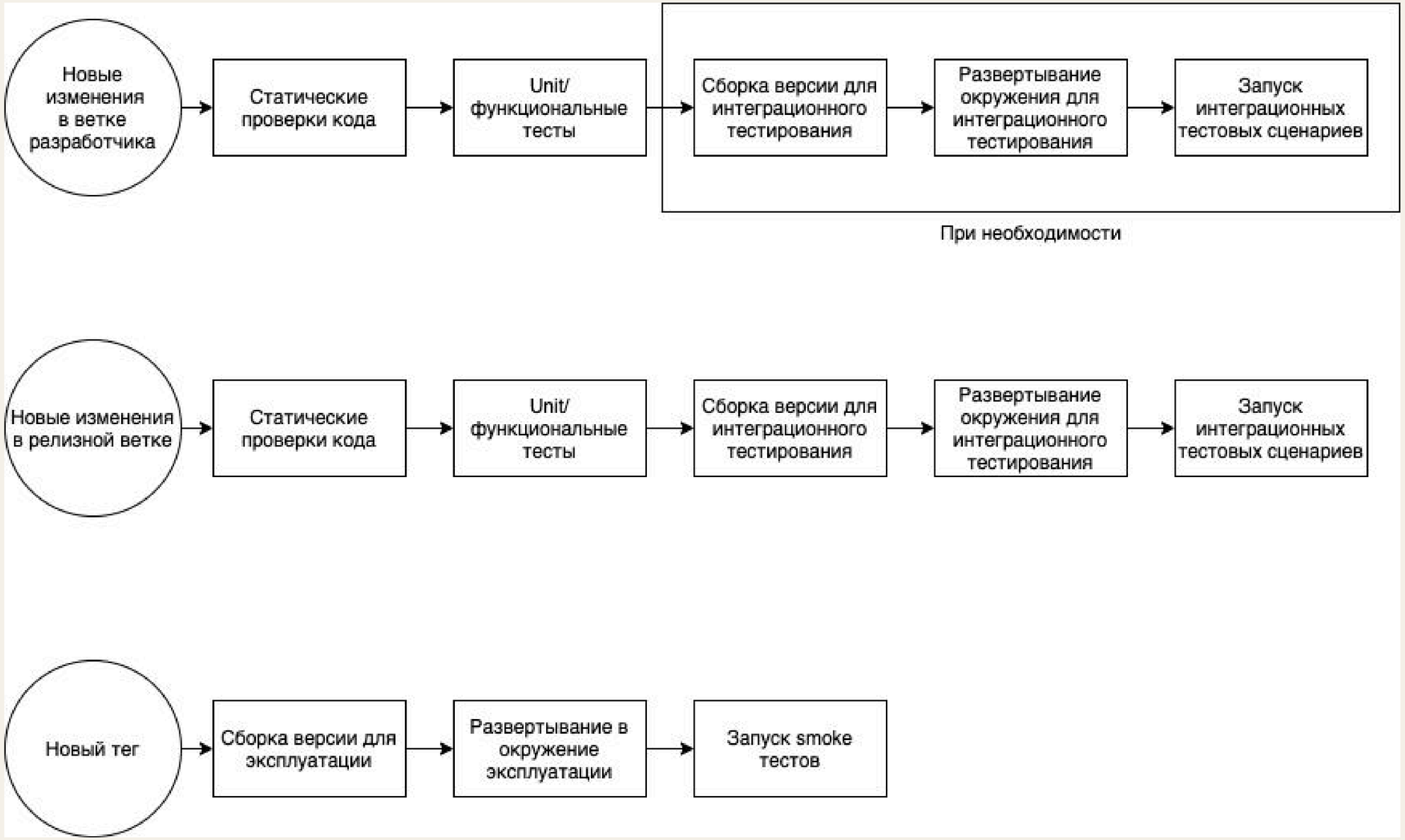


Github Actions

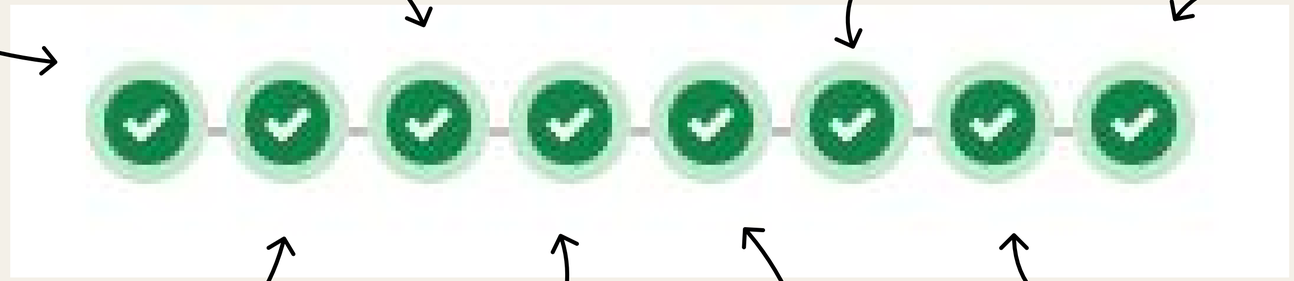
GitHub Actions позволяет автоматизировать различные действия, которые могут выполняться перед релизом, а также в процессе разработки



Пример этапов ci пайплайна



Пример этапов ci пайплайна из практики



Линтеры

Сборка питон-пакета

Сборка докер-образов

Пуш докер-образов

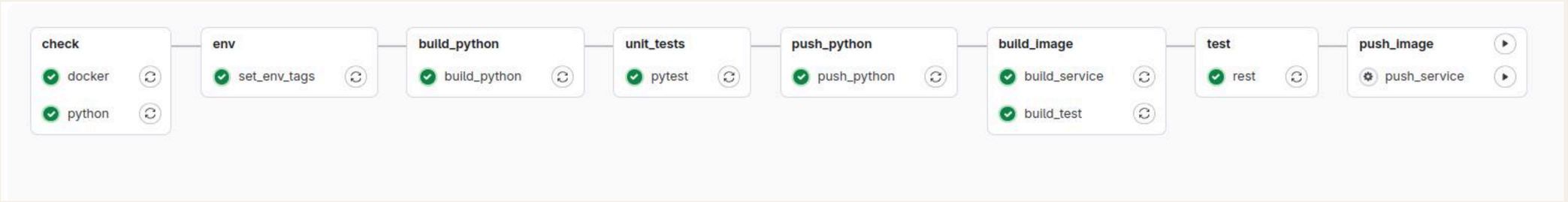
Подготовка переменных окружения

Юнит-тесты

Пуш питон-пакета

Интеграционные тесты для образов

Пример этапов сі пайплайна из практики



Pipeline Jobs 10 **Tests 48**

Summary

48 tests | 0 failures | 0 errors | 100% success rate | 27.88s

Jobs

Job	Duration	Failed	Errors	Skipped	Passed	Total
pytest	27.88s	0	0	0	48	48

Синтаксис

JSON vs YAML

JSON ✓

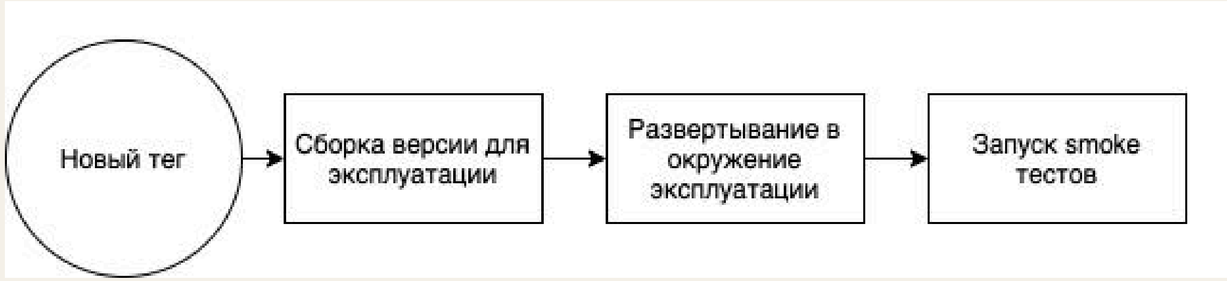
```json
{ "apps":
[
  {
    "course": "TTIS",
    "name": "Анна",
    "email": "anna@email.ru",
    "phone": "123456789",
    "corp": true,
    "wishes": {
      "city": "Казань",
      "online": true,
      "group ": true,
      "number_of_students": 20
    }
  },
  {
    "course": "MODP",
    "name": "Борис",
    "email": "boris@email.ru",
    "corp": false,
    "wishes": {
      "online": true,
      "group ": false
    }
  },
  {
```

YAML

```yaml
---
apps:
- course: TTIS
  name: Анна
  email: anna@email.ru
  phone: '123456789'
  corp: true
  wishes:
    city: Казань
    online: true
    'group ': true
    number_of_students: 20
- course: MODP
  name: Борис
  email: boris@email.ru
  corp: false
  wishes:
    online: true
    'group ': false
- course: OAIS
  name: Лиза
  phone: '987654321'
  corp: false
  wishes:
    country: Россия
    online: true
```

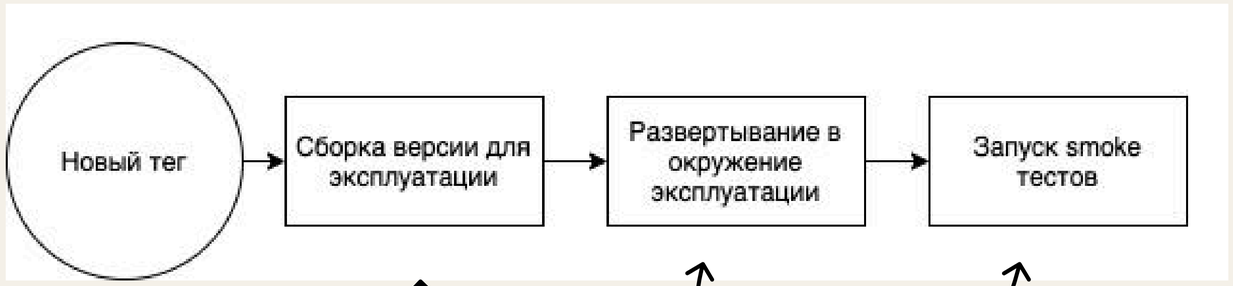
Github Actions **конфигурация**

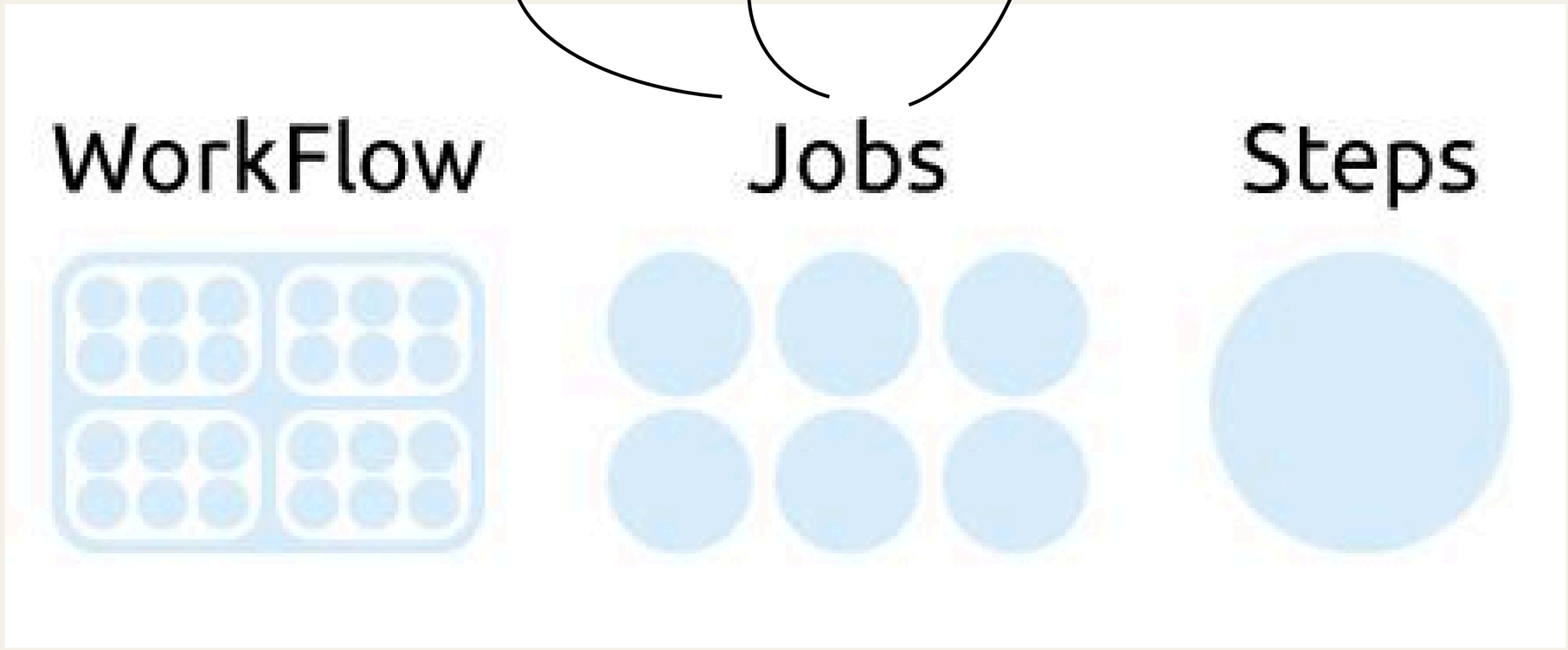
В yaml файле прописываются шаги пайплана, которые должны быть выполнены




```
name: Print workflow #Название workflow
on: workflow_dispatch #Триггер, который будет вызывать workflow

jobs: #Начало секции джобов
  print_hello: #Название джоба
    runs-on: ubuntu-latest #Среда выполнения джоба

    steps: #Начало секции шагов
      - name: Print hello #Название шага
        run: echo "Hello world!" #Команда, которая будет выполнена в терминале
```

Github Actions **конфигурация**





Попробуйте на ваших проектах