📦 Работа с образами и контейнерами

Базовые команды работы с образами и контейнерами

Скачать образ с docker hub

```
docker pull <образ>
```

Вывод информации о образах

```
docker images
```

Запустить образ - создаст контейнер и запустит его

```
docker run --name <name> <имя образа>
```

Вывести все контейнеры

```
docker ps -a
```

Вывод запущенных контейнеров

```
docker ps
```

Через : можно указывать тег образа и теги интерактивного вывода

```
docker run -it python:3.12-alpine
```

Запустить контейнер

```
docker start <имя или идентификартор контейнера>
```

Привязать вывод к стандартому потоку вывода

```
docker start -i <name>
```

Остановить работающий контейнер

```
docker stop <имя или идентификартор контейнера>
docker kill <имя или идентификартор контейнера> # в крайнем случае
```

Удалить контейнер

```
docker rm <имя или идентификартор контейнера>
```

Удалить все неработающие контейнеры

```
docker container prune
```

Удалить образ

```
docker rmi <имя или идентификартор образа>
```

Для помощи по команде

```
docker run --help
```

Dockerfile. Создание собственных образов

Для создания собственного образа необходим файл **Dockerfile** который содержит инструкции для создания образа **docker**'ом.
Общая архитектура:

Структура `Dockerfile`:

Инструкция	Назначение
`FROM`	Базовый образ (например, `python:3.12`)
`COPY`	Копирует файлы из контекста сборки внутрь контейнера
`WORKDIR`	Устанавливает рабочую директорию внутри контейнера
`RUN`	Выполняет команду при сборке образа
`CMD`	Определяет команду по умолчанию при запуске контейнера
`ENTRYPOINT`	Аналогично `CMD`, но не переопределяется так просто
`EXPOSE`	Документирует открытые порты (не делает проброс автоматически)
`ENV`	Устанавливает переменные окружения

Пример образа который запускает python скрипт из папки с проектом.

Структура проекта:

```
my-python-app/
├── Dockerfile
└── script.py
```

Dockerfile:

```
FROM python:3.12-slim

# Отключаем буферизацию, чтобы вывод сразу попадал в stdout
ENV PYTHONUNBUFFERED=1

# Рабочая директория
WORKDIR /app

# Копируем скрипт в контейнер (. просто в папку app)
COPY script.py .

# Команда по умолчанию в JSON формате
CMD ["python", "script.py"]
```

Сборка docker образа:

```
docker build . -t my-python-image
# -t my-python-image — имя образа
# . — текущая директория (путь до Dockerfile)
```

Запуск контейнера с выводом в консоль

```
docker run my-python-image
```

Запуск контейнера с выводом в консоль в интерактивном режиме

```
docker run -it --rm my-python-image
# --rm удалить контейнер после завершения
```

Запуск в фоне

```
docker run -d --name my-container my-python-image
```

Особенности создания образов и запуска контейнеров

Слои Docker образа

- Каждый слой в Docker создаётся на основе инструкций Dockerfile (`FROM`, `COPY`, `RUN` и т.д.).
- Docker кэширует слои — если слой не менялся, он пересобирается из кэша.
- Если верхний слой изменился, все нижележащие кэш слетают.

Именно поэтому стоит разделять `COPY` - COPY . . Docker считает, что всё изменилось даже при малейшем изменении.

```
# Копируем только зависимости
COPY requirements.txt .

# Ставим зависимости
RUN pip install -r requirements.txt

# Потом копируем всё остальное
COPY . .
```

CMD vs ENTRYPOINT

Характеристика	CMD	ENTRYPOINT
Переопределяется	Да	Нет (только через `--entrypoint`)

Основная цель	Команда по умолчанию	Жёстко заданная точка входа
Аргументы при запуске	Полностью заменяют CMD	Передаются как аргументы

Полезные команды

Просмотр логов контейнера:

```
docker logs <container>
```

Сохранить состояние контейнера как образ:

```
docker commit <container> <new_image>
```

Удалить остановленные контейнеры, неиспользуемые тома и сети:

```
docker system prune
```

Удалить только кэш сборки:

```
docker builder prune
```

Удалить весь кэш сборки (слои, unused кеш, всё подряд):

```
docker builder prune --all
```

Удалить **все** неиспользуемые образы:

```
docker image prune -a
```

Удалить **весь** возможный мусор включая **все** неиспользуемые образы:

```
docker system prune -a
```

Также у комманд как **pause**, **ps** и так далее есть множество флагов, например:

```
docker ps -q -f status=restarting
docker stop $(docker ps -q)
```