1. Задание 1. Базовые операции с Docker

- Выполните команду `docker run hello-world` и убедитесь, что вывод содержит приветственное сообщение.
- Проверьте список локально сохранённых образов с помощью команды `docker images`.
- Запустите контейнер в фоновом режиме с помощью:

```
docker run -d --name test_container nginx
```

- Проверьте список запущенных контейнеров с помощью команды `docker ps`.
- Посмотрите логи контейнера командой:

```
docker logs test_container
```

- Остановите контейнер командой:

```
docker stop test_container
```

- Удалите остановленный контейнер командой:

```
docker rm test_container
```

---

2. Задание 2. Создание Dockerfile для простого ML-приложения

- Создайте следующую структуру проекта:

```
my_ml_app/
├── app.py
├── requirements.txt
└── Dockerfile
```

- В файле **app.py** реализуйте простое Flask-приложение, которое:
  - Принимает POST-запрос с JSON-данными.
  - Вычисляет сумму чисел (с использованием библиотеки NumPy).
  - Возвращает результат в формате JSON.
- В файле **requirements.txt** укажите зависимости:

```
flask
numpy
```

- Напишите Dockerfile, который:
  - Использует базовый образ `python:3.10-slim`.
  - Устанавливает зависимости из файла `requirements.txt`.
  - Копирует файл `app.py` в образ.
  - Открывает порт 5000.
  - Задаёт команду для запуска приложения (например, `CMD ["python", "app.py"]`).
- Соберите образ командой:

```
docker build -t my_ml_app:latest .
```

- Запустите контейнер командой:

```
docker run -d --name ml_app_container -p 5000:5000 my_ml_app:latest
```

- Проверьте работу приложения, отправив POST-запрос (например, с помощью `curl`).

---

3. Задание 3. Оптимизация Dockerfile с использованием многоступенчатой сборки и безопасности

- Перепишите Dockerfile из задания 2 с использованием многоступенчатой сборки:
  - Первый этап: установите зависимости (копирование `requirements.txt` и установка пакетов).
  - Финальный этап: скопируйте только необходимые файлы (например, `app.py` и установленные библиотеки) в минимальный образ.
- Добавьте в финальном этапе Dockerfile инструкцию для создания нового пользователя (например, `myuser`) и переключитесь на него:

```
RUN useradd -m myuser
USER myuser
```

- Добавьте переменные окружения с помощью инструкции `ENV` (например, `ENV APP_ENV=production`).
- Соберите новый образ и запустите контейнер, убедившись, что приложение работает от непривилегированного пользователя.

---

4. Задание 4. Ограничение ресурсов и мониторинг контейнеров

- Запустите контейнер вашего ML-приложения с ограничениями ресурсов, используя флаги:

```
docker run -d --name ml_app_secure --cpus="1.0" --memory="512m" -p 5000:5000
my_ml_app:latest
```

- Проверьте показатели использования ресурсов контейнера с помощью команды:

```
docker stats ml_app_secure
```

- Просмотрите логи контейнера с помощью команды:

```
docker logs ml_app_secure
```

---

5. Задание 5. Сканирование образа на уязвимости с помощью Trivy

- Установите Trivy, следуя инструкции из официальной документации (например, на GitHub проекта Trivy).
- Проведите сканирование Docker-образа командой:

```
trivy image my_ml_app:latest
```

- Проанализируйте результаты сканирования и, при необходимости, внесите изменения (например, обновите базовый образ или зависимости).
