##### МИНЦИФРЫ

##### «СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

##### ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

##### «Базовая часть контейнеризации»

##### по дисциплине «Архитектура вычислительных систем»

##### Выполнил: студент гр. ИП-212

##### Жеребцов Дмитрий Евгеньевич

##### Новосибирск 2024

# Постановка задачи

##### Задание.

DoD на 3

* + Запустить любой контейнер с сетевым приложением. Обеспечить сетевой доступ к приложению в контейнере. Для примера можно использовать образ nginx
  + Рассказать как происходит старт контейнера и остановка в контексте системы
* DoD на 4
  + DoD на 3
  + Показать как реагирует контейнер (процесс в контейнере) на команду docker stop - Как производится остановка контейнера системой?
  + Продемонстрировать уровни изоляции - PID, ipc, network, users, mount, uts
* DoD на 5
  + DoD на 4
  + Рассказать, для чего используются cgroups
  + Написать приложение, реагирующее на сигналы. Приложение может быть любым, основной функционал - непрерывная работа и обработчик сигналов. Написать Dockerfile для компиляции приложения и создания образа, из которого можно запустить контейнер. В контейнере должно запускаться разработанное приложение.  
    Допускается реализовать демонстрацию изоляции через это приложение (см. DoD на 4)

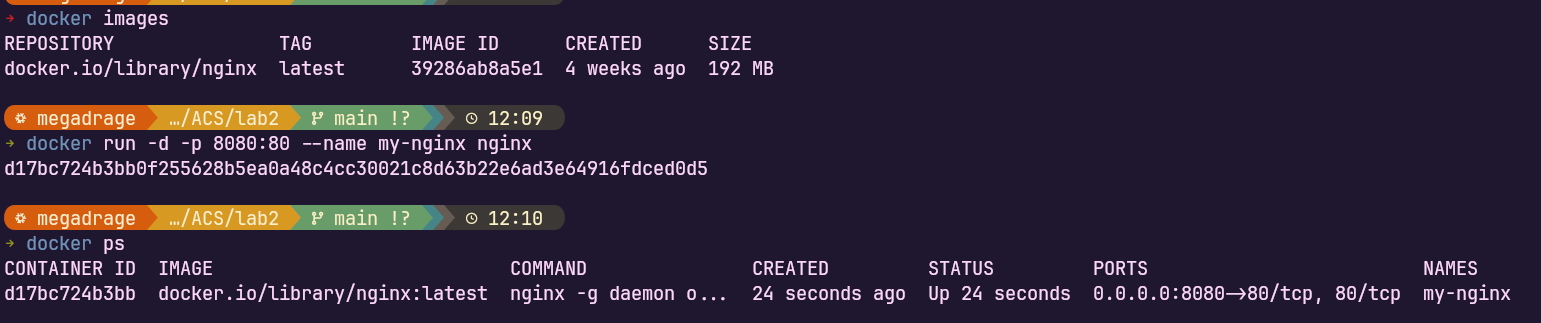
# Выполнение работы

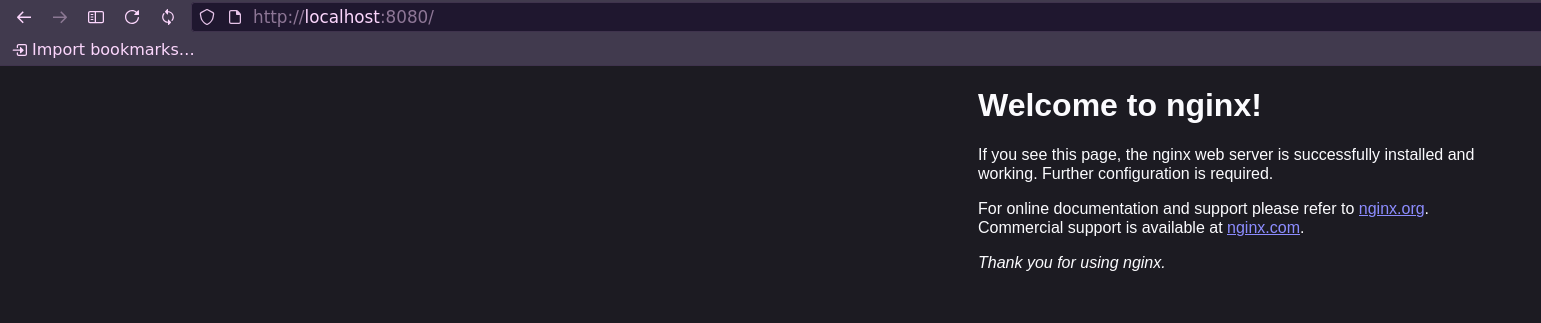
Запуск контейнера

docker run -d -p 8080:80 --name my-nginx nginx

* -d: Запуск контейнера в фоновом режиме (detached mode).
* -p 8080:80: Проброс порта 8080 на хост-машине на порт 80 внутри контейнера.
* --name my-nginx: Присвоение имени контейнеру.
* nginx: Имя образа, который будет использоваться.

docker ps: показывает запущенные контейнеры



Остановка контейнера

1. **Отправка сигнала**:
   * Когда вы выполняетcя команду docker stop, Docker отправляет сигнал SIGTERM основному процессу в контейнере. Этот сигнал уведомляет процесс о том, что ему нужно корректно завершить работу.
2. **Ожидание завершения**:
   * Если процесс в контейнере не завершается в течение определенного времени (по умолчанию 10 секунд), Docker отправляет сигнал SIGKILL, который принудительно завершает процесс.

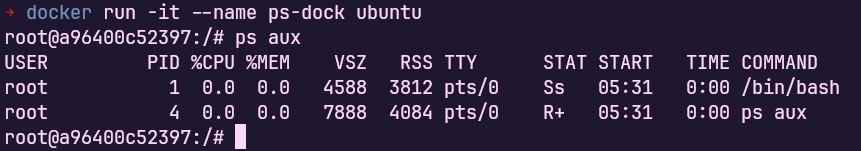
### Уровни изоляции в Docker

#### PID (Process ID)

Каждый контейнер имеет свою собственную таблицу процессов, изолированную от хост-системы и других контейнеров.

docker run -it --name ps-dock ubuntu

ps aux



#### IPC (Inter-Process Communication)

Контейнеры имеют изолированные механизмы межпроцессного взаимодействия, такие как очереди сообщений, семафоры и разделяемая память.

**Пример**:

docker run -it --name ipc-dock ubuntu

ipcs -a

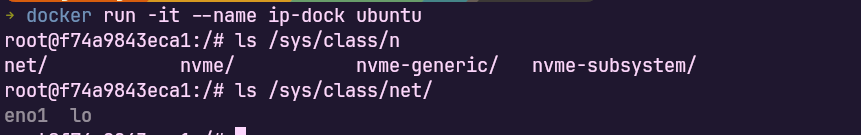
#### Network

Каждый контейнер имеет свою собственную сетевую стек, включая IP-адрес, интерфейсы и таблицы маршрутизации.

**Пример**:

docker run -it --name ip-dock ubuntu

ls /sys/class/net

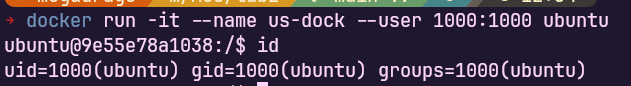


#### Users

Контейнеры могут иметь свои собственные пользовательские пространства, изолированные от хост-системы.

docker run -it --name us-dock --user 1000:1000 ubuntu

id



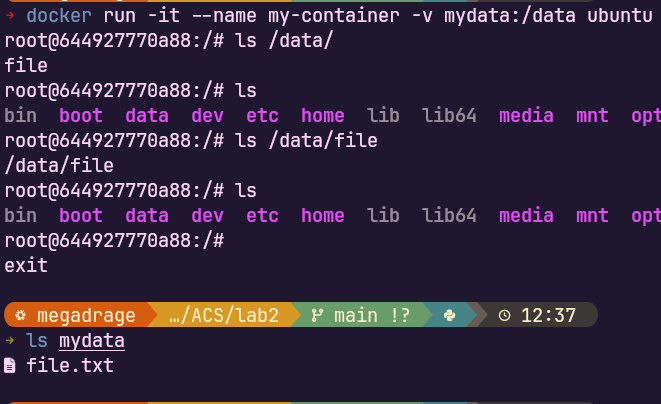
#### Mount

Контейнеры могут иметь свои собственные точки монтирования, изолированные от хост-системы.

**Пример**:

docker run -it --name my-container -v mydata:/data ubuntu

ls /data



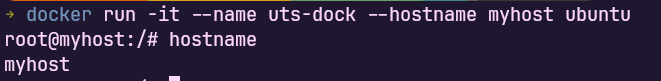
#### UTS (Unix Timesharing System)

Контейнеры могут иметь свои собственные имена хостов и доменные имена, изолированные от хост-системы.

**Пример**:

docker run -it --name uts-dock --hostname myhost ubuntu

hostname



**cgroups (Control Groups)** — это функция ядра Linux, которая позволяет ограничивать, контролировать и изолировать использование ресурсов (CPU, памяти, дискового ввода-вывода, сети и т.д.) процессами. Docker использует cgroups для управления ресурсами контейнеров, чтобы они не могли использовать больше ресурсов, чем разрешено.

**Примеры использования cgroups:**

* **Ограничение CPU**: Ограничить использование CPU для контейнера.
* **Ограничение памяти**: Ограничить использование памяти для контейнера.
* **Ограничение дискового ввода-вывода**: Ограничить скорость дискового ввода-вывода для контейнера.

### Приложение, реагирующее на сигналы

Давайте напишем простое приложение на Python, которое будет работать непрерывно и реагировать на сигналы SIGTERM и SIGINT.

#### Пример приложения (signal\_app.py):

import signal

import time

import sys

def signal\_handler(sig, frame):

print(f"Received signal {sig}. Exiting...")

sys.exit(0)

signal.signal(signal.SIGTERM, signal\_handler)

signal.signal(signal.SIGINT, signal\_handler)

print("Application is running. Press Ctrl+C to exit or send SIGTERM.")

while True:

time.sleep(1)

### Dockerfile для компиляции приложения и создания образа

Создадим Dockerfile для сборки образа, который будет запускать наше приложение.

**Dockerfile (Dockerfile):**

# Используем базовый образ Python

FROM python:3.9-slim

# Устанавливаем рабочую директорию

WORKDIR /app

# Копируем файл приложения в контейнер

COPY signal\_app.py .

# Запускаем приложение при старте контейнера

CMD ["python", "signal\_app.py"]

### Создание и запуск контейнера

### 

### 