Скачиваем docker

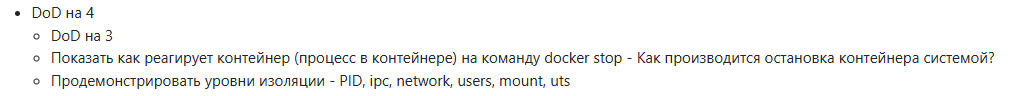


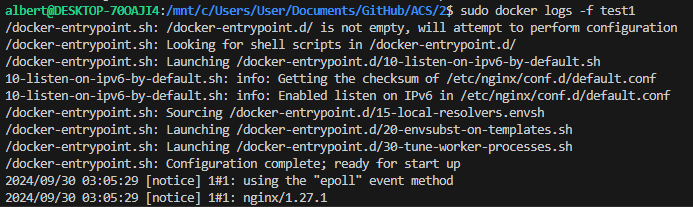
1. Запускаем докер в фоновом режиме (-d), публикуем контейнер на порт хоста (-p 80:80), и присваиваем имя (--name test1) 
2. Старт и остановка контейнера происходит следующим образом:

**Запуск контейнера:**

* Когда выполняется команда docker run, Docker Engine взаимодействует с ядром операционной системы для создания изолированного пространства для процесса — **контейнера**.
* Docker использует **cgroups** и **namespaces** для изоляции контейнера (ограничение по ресурсам и изоляция процессов).
* Образ Nginx скачивается из **Docker Hub** (если его нет локально), и на основе него создается контейнер.
* Когда контейнер стартует, запускается основной процесс Nginx, который начинает слушать запросы на порту 80.

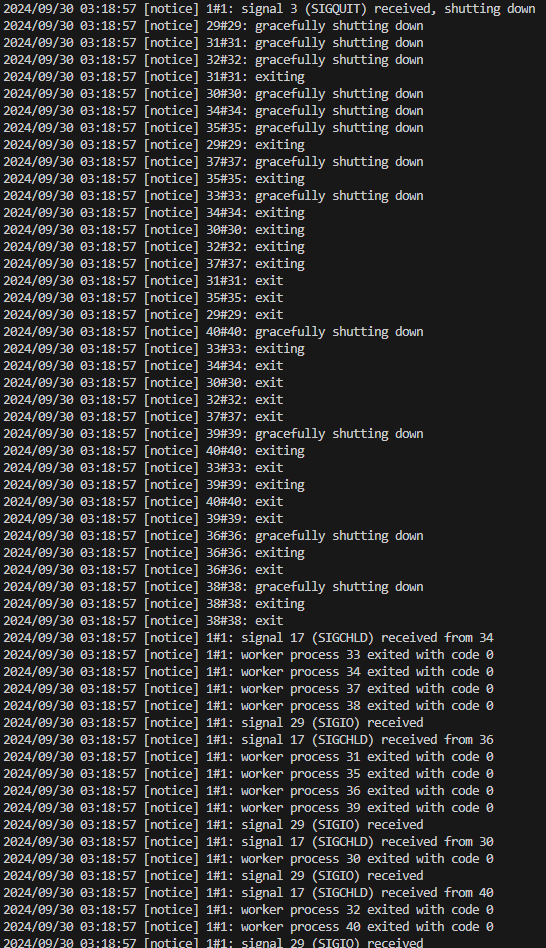
**Остановка контейнера:**

* Команда docker stop отправляет основной процессу внутри контейнера сигнал SIGTERM, давая возможность завершить работу корректно.
* Если процесс не завершится в течение заданного таймаута (по умолчанию 10 секунд), Docker отправит сигнал SIGKILL, чтобы принудительно завершить процесс.
* После завершения всех процессов контейнер останавливается, освобождая все системные ресурсы (например, файловые дескрипторы, сетевые интерфейсы и т.д.).

1.Подпишемся на логи контейнера, чтобы отслеживать его состояние

2. В другой вкладке терминала остановим docker и посмотрим, как это отразится на логах





После сигнала SIGQUIT, система успешно закрыла все свои процессы и остановилась

**2. Уровни изоляции контейнера**

Docker использует несколько типов изоляции для контейнеров: PID, IPC, network, users, mount и UTS. Рассмотрим каждый из них и продемонстрируем, как это работает.

**2.1. PID (Идентификаторы процессов)**

Docker создаёт отдельное пространство процессов для контейнера, изолируя его от процессов на хосте. Это можно проверить следующим образом:

* Запустим контейнер с Bash для проверки:

bash

Копировать код

docker run -it --name testpid --rm alpine sh

* Выполним команду ps внутри контейнера:

bash

Копировать код

ps aux

Здесь вы увидите только процессы, относящиеся к контейнеру.

* Для сравнения, на хосте выполните:

bash

Копировать код

ps aux | grep testpid

Процесс контейнера будет виден с другим PID, чем внутри контейнера.

**2.2. IPC (Механизм обмена сообщениями между процессами)**

Контейнеры имеют изолированное пространство для IPC (межпроцессного взаимодействия, например, для использования разделяемой памяти). Проверим это:

* В одном терминале запустим контейнер с доступом к shared memory:

bash

Копировать код

docker run -it --name testipc --rm --ipc=shareable alpine sh

* Внутри контейнера создадим shared memory сегмент:

bash

Копировать код

ipcs -m

Видно, что сегменты памяти изолированы.

**2.3. Network (Сетевое пространство)**

Docker создаёт для каждого контейнера изолированную сеть, но можно использовать различные сетевые режимы.

* Запустим контейнер и проверим сетевые интерфейсы:

bash

Копировать код

docker run -it --rm --name testnetwork alpine sh

* Выполним команду для просмотра интерфейсов:

bash

Копировать код

ifconfig

Здесь вы увидите изолированный сетевой интерфейс контейнера. Он отличается от интерфейсов на хосте.

**2.4. Users (Изоляция пользователей)**

Docker поддерживает изоляцию пользователей, позволяя контейнеру использовать свои UID и GID.

* Запустим контейнер с пользовательской изоляцией:

bash

Копировать код

docker run -it --rm --user 1000:1000 alpine sh

Внутри контейнера пользователь имеет UID и GID, заданные в команде.

**2.5. Mount (Файловая система)**

Контейнеры запускаются с собственными файловыми системами, которые монтируются при запуске контейнера.

* Для проверки смонтируем директорию с хоста:

bash

Копировать код

docker run -it --rm -v /tmp:/hosttmp alpine sh

* Проверим, что директория /hosttmp доступна в контейнере:

bash

Копировать код

ls /hosttmp

Здесь видны файлы из директории /tmp на хосте.

**2.6. UTS (Изоляция хостнейма)**

Изоляция UTS позволяет контейнеру иметь собственное имя хоста.

* Запустим контейнер и проверим его имя хоста:

bash

Копировать код

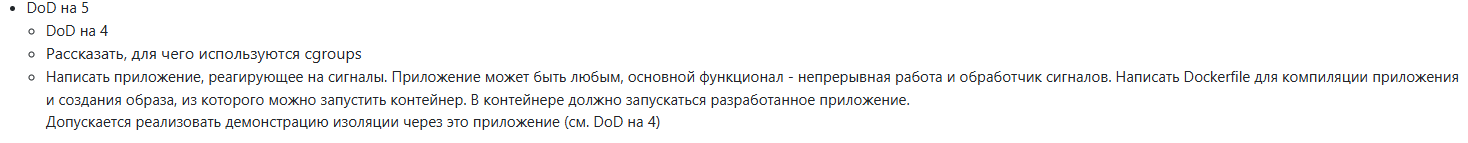
docker run -it --rm alpine sh

Выполним команду:

bash

Копировать код

hostname

У каждого контейнера будет своё имя хоста, отличное от хоста системы.

**1. Что такое cgroups?**

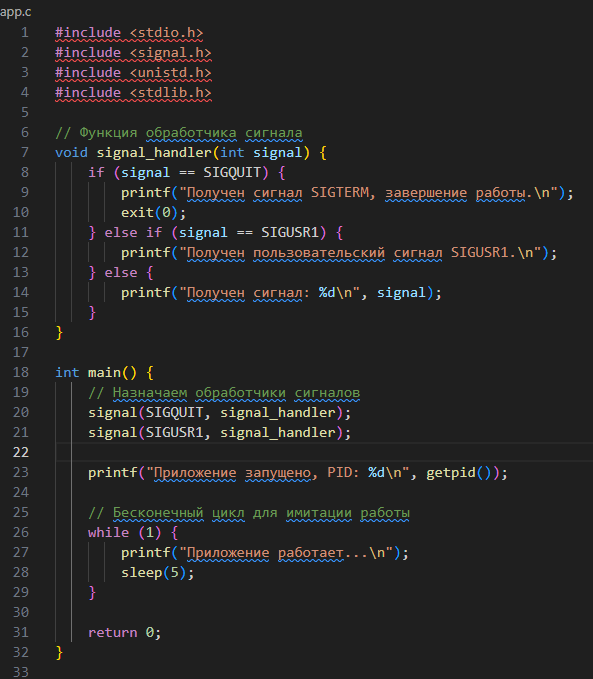
**Control Groups** (cgroups) — это механизм ядра Linux, который позволяет ограничивать, управлять и отслеживать ресурсы (CPU, память, сеть, диск и другие) для групп процессов. Cgroups используются для контроля над ресурсами, которые может использовать процесс, и широко применяются в контейнерах для изоляции и управления ресурсами каждого контейнера.

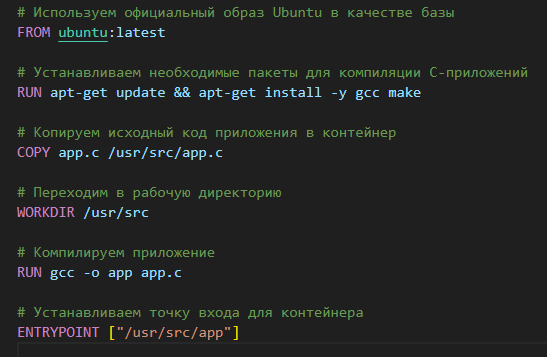
**Основные задачи cgroups:**

1. **Ограничение ресурсов**: Можно установить пределы использования памяти, процессора или сетевых ресурсов для группы процессов.
2. **Приоритеты ресурсов**: Процессам можно присвоить разные приоритеты для использования ресурсов.
3. **Учёт ресурсов**: Отслеживание использования ресурсов (например, сколько памяти или CPU использует процесс).
4. **Изоляция ресурсов**: Обеспечение того, чтобы один процесс не потреблял ресурсы, необходимые другим процессам.
5. **Ограничение количества процессов**: Можно задать ограничение на количество процессов или потоков, которые может создать группа процессов.

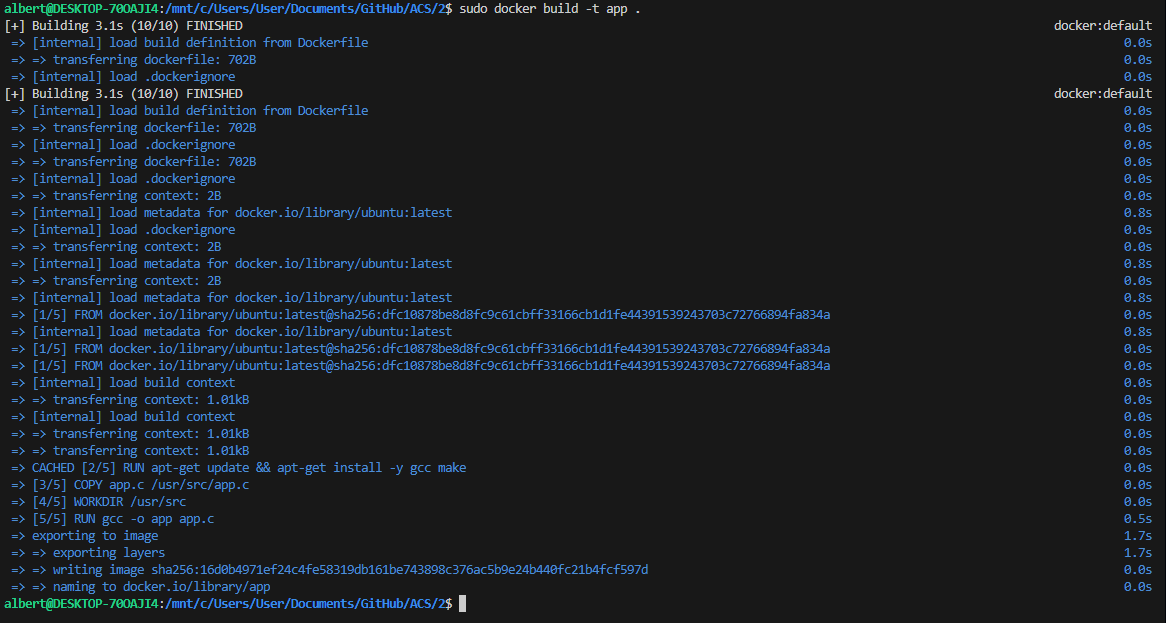
**2. Создание приложения, обрабатывающее сигналы**

Напишем код на C, обрабатывающий сигналы



Напишем DockerFile

Соберем Docker образ



Запустим образ

