

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

```
➤ docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
docker.io/library/nginx  latest         39286ab8a5e1   4 weeks ago    192 MB

➤ megadrage .../ACS/lab2 P main !? 12:09
➤ docker run -d -p 8080:80 --name my-nginx nginx
d17bc724b3bb0f255628b5ea0a48c4cc30021c8d63b22e6ad3e64916fdced0d5

➤ megadrage .../ACS/lab2 P main !? 12:10
➤ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
d17bc724b3bb   docker.io/library/nginx:latest      nginx -g daemon o...    24 seconds ago Up 24 seconds 0.0.0.0:8080→80/tcp, 80/tcp        my-nginx
```



Остановка контейнера

1. Отправка сигнала:

- Когда вы выполняется 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
```

```
→ docker run -it --name ps-dock ubuntu
root@a96400c52397:/# ps aux
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0  4588  3812 pts/0    Ss   05:31   0:00 /bin/bash
root           4  0.0  0.0  7888  4084 pts/0    R+   05:31   0:00 ps aux
root@a96400c52397:/#
```

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
```

```
root@04ef7e3706de:/# ipcs -a

----- Message Queues -----
key          msqid      owner      perms      used-bytes   messages

----- Shared Memory Segments -----
key          shmid      owner      perms      bytes       nattch     status

----- Semaphore Arrays -----
key          semid      owner      perms      nsems
```

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
```

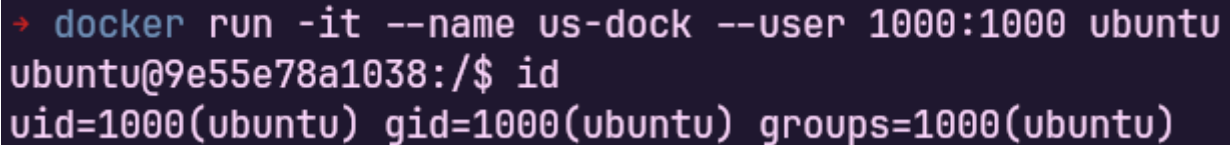
```
→ docker run -it --name ip-dock ubuntu
root@f74a9843eca1:/# ls /sys/class/n
net/          nvme/          nvme-generic/  nvme-subsystem/
root@f74a9843eca1:/# ls /sys/class/net/
eno1  lo
```

Users

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

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

```
id
```



```
→ docker run -it --name us-dock --user 1000:1000 ubuntu
ubuntu@9e55e78a1038:/$ id
uid=1000(ubuntu) gid=1000(ubuntu) groups=1000(ubuntu)
```

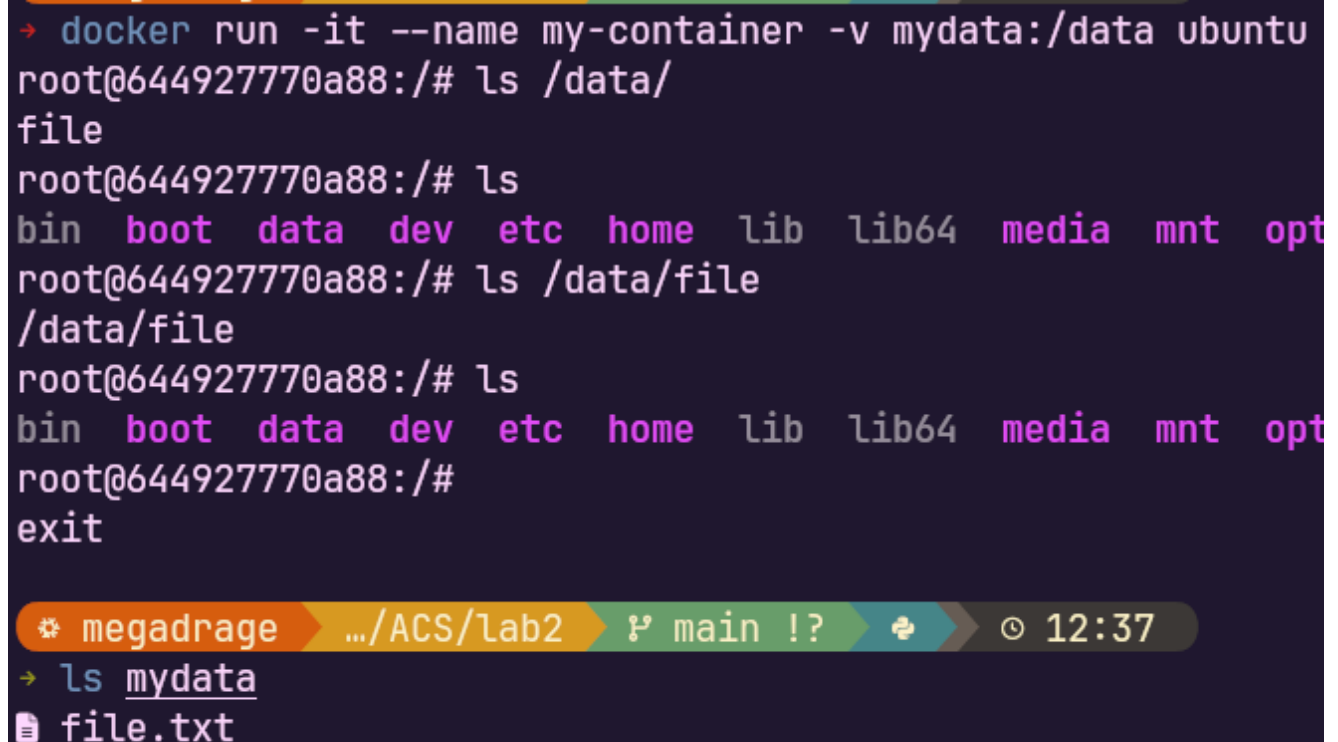
Mount

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

Пример:

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

```
ls /data
```



```
→ docker run -it --name my-container -v mydata:/data ubuntu
root@644927770a88:/# ls /data/
file
root@644927770a88:/# ls
bin boot data dev etc home lib lib64 media mnt opt
root@644927770a88:/# ls /data/file
/data/file
root@644927770a88:/# ls
bin boot data dev etc home lib lib64 media mnt opt
root@644927770a88:/#
exit

* megadrage .../ACS/lab2 ? main !? 12:37
→ ls mydata
file.txt
```

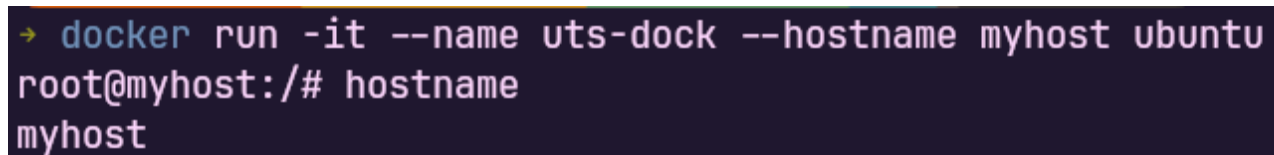
UTS (Unix Timesharing System)

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
```



```
→ docker run -it --name uts-dock --hostname myhost ubuntu
root@myhost:/# hostname
myhost
```

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

```
> docker build -t signal-app .
STEP 1/4: FROM python:3.9-slim
✓ docker.io/library/python:3.9-slim
Trying to pull docker.io/library/python:3.9-slim...
Getting image source signatures
Copying blob b665d04ddefb done |
Copying blob 0fa26e0a6c77 done |
Copying blob a2318d6c47ec skipped: already exists
Copying blob a657783e238b done |
Copying config 397ed8d316 done |
Writing manifest to image destination
STEP 2/4: WORKDIR /app
→ 612bef4a3424
STEP 3/4: COPY signal_app.py .
→ e8dc0c7f3a47
STEP 4/4: CMD ["python", "signal_app.py"]
COMMIT signal-app
→ 9a6fdae7d183
Successfully tagged localhost/signal-app:latest
9a6fdae7d183f5e009b772e96867ee299ee24c30b87705ea1be4a35121889c93
```

```
* megadrage .../ACS/lab2 ? main !? 12:44
→ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
localhost/signal-app	latest	9a6fdae7d183	20 seconds ago	130 MB
docker.io/library/python	3.9-slim	397ed8d31636	6 days ago	130 MB
docker.io/library/nginx	latest	39286ab8a5e1	4 weeks ago	192 MB
docker.io/library/ubuntu	latest	edbf74c41f8	6 weeks ago	80.6 MB