

Летняя школа ВШЭ. День 1.

Лабораторная работа. Создание контейнера с нуля своими руками.

Цель работы - научиться создавать и использовать контейнеры в Linux для изоляции процессов и файловой системы. В этой лабораторной работе вы создадите простую файловую систему контейнера, сконфигурируете её для работы с командной оболочкой и протестируете изоляцию процессов.

Для выполнения работы вам потребуется виртуальная машина или контейнер с ОС Linux (лабораторная работа тестировалась на Ubuntu 24.04).

Шаг 1. Создание структуры контейнера

1. Создайте основной каталог для контейнера:

```
sudo mkdir /home/container
```

Мы создаем основной каталог, в котором будет располагаться наша контейнерная файловая система

2. Создайте подкаталог box:

```
sudo mkdir /home/container/box
```

Этот каталог будет использоваться как корневая файловая система нашего контейнера

3. Создайте подкаталоги для различных типов файлов:

```
sudo mkdir /home/container/box/bin
sudo mkdir /home/container/box/lib
sudo mkdir /home/container/box/lib64
sudo mkdir /home/container/box/proc
sudo mkdir /home/container/box/data
```

bin содержит исполняемые файлы, lib и lib64 — библиотеки, необходимые для этих исполняемых файлов, а proc будет использоваться для виртуальной файловой системы, которая отображает информацию о процессах.

Шаг 2. Копирование необходимых бинарных файлов.

1. Скопируйте исполняемые файлы для работы в контейнере:

```
sudo cp -v /usr/bin/kill /home/container/box/bin/
sudo cp -v /usr/bin/ps /home/container/box/bin
sudo cp -v /bin/bash /home/container/box/bin
sudo cp -v /bin/ls /home/container/box/bin
```

Команды kill, ps, bash, и ls будут использоваться внутри контейнера. Мы копируем их в каталог bin контейнера, чтобы они были доступны для использования внутри контейнера.

2. Скопируйте библиотеки, необходимые для работы этих команд:

```
sudo cp -r /lib/* /home/container/box/lib/
sudo cp -r /lib64/* /home/container/box/lib64/
```

Библиотеки из каталогов /lib и /lib64 необходимы для правильной работы скопированных команд, так как они зависят от этих библиотек. В зависимости от

версии вашего дистрибутива у вас может и не быть директории lib64 (в этом случае копируйте только lib).

3. Примонтируйте виртуальную файловую систему proc в контейнере:

```
sudo mount -t proc proc /home/container/box/proc
```

Файловая система proc предоставляет информацию о текущих процессах и состоянии системы. Это необходимо для правильного функционирования команд внутри контейнера.

Шаг 3. Вход в контейнер и иллюстрация изоляции файловой системы.

1. Войдите в контейнер с помощью chroot:

```
sudo chroot /home/container/box /bin/bash
```

Команда chroot изменяет корневую директорию для текущего процесса и его потомков. Мы используем её, чтобы запустить оболочку bash внутри нашего контейнера.

2. Проверьте содержимое корневого каталога контейнера:

```
ls
```

В качестве корня команда должна показать содержимое папки /box.

3. Выйдите из контейнера

```
exit
```

4. Примонтируйте каталог /tmp на хосте в каталог data внутри контейнера

```
sudo mount --bind /tmp/ /home/container/box/data
```

Команда mount --bind позволяет подключить один каталог к другому, что позволяет контейнеру видеть содержимое каталога /tmp на хосте как data.

5. Создайте файлы в каталоге /tmp на хосте

```
sudo touch /tmp/a
sudo touch /tmp/b
sudo touch /tmp/c
```

Мы создаем несколько файлов в каталоге /tmp на хостовой системе для проверки доступности этих файлов из контейнера

6. Войдите снова в контейнер:

```
sudo chroot /home/container/box /bin/bash
```

7. Проверьте содержимое каталога data:

```
ls data
```

Теперь вы должны увидеть файлы a, b и c в каталоге data, что подтверждает успешное подключение каталога /tmp из хостовой системы и показывает условность изоляции с помощью chroot.

8. Выйдите из контейнера:

```
exit
```

9. Размонтируйте каталог data:

```
sudo umount /tmp/ /home/container/box/data
```

Шаг 4. Усиливаем изоляцию процессов в контейнере с помощью unshare.

1. Скопируйте дополнительный бинарный файл утилиты `ip` в контейнер, чтобы мы могли проверить состояние сетевых интерфейсов.

```
sudo cp -v /usr/bin/ip /home/container/box/bin
```

2. Запустите контейнер в изолированном пространстве с помощью `unshare`:

```
sudo unshare -p -f --mount-proc=/home/container/box/proc/ chroot  
/home/container/box/ /bin/bash
```

Команда `unshare` позволяет создать изолированное пространство для процессов, файловых систем и сетевых настроек. Мы используем её для изоляции процесса контейнера, а также указываем, где находится файловая система `proc`.

3. Проверьте список процессов (`ps`) и сетевых интерфейсов (`ip a`) и выйдите с помощью (`exit`):

```
ps  
ip a
```

```
[bash-5.2# ps
```

PID	TTY	TIME	CMD
1	?	00:00:00	bash
2	?	00:00:00	ps

```
[bash-5.2# ip a
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000  
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
    inet 127.0.0.1/8 scope host lo  
        valid_lft forever preferred_lft forever  
    inet6 ::1/128 scope host noprefixroute  
        valid_lft forever preferred_lft forever  
2: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000  
    link/ether 00:0c:29:73:a6:8c brd ff:ff:ff:ff:ff:ff  
    altname enp2s0  
    inet 192.168.59.128/24 metric 100 brd 192.168.59.255 scope global dynamic ens160  
        valid_lft 1370sec preferred_lft 1370sec  
    inet6 fe80::20c:29ff:fe73:a68c/64 scope link  
        valid_lft forever preferred_lft forever  
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
```

Здесь мы можем увидеть, что namespace PID изолирован, поскольку `unshare` создал для процесса `bash` новый неймспейс PID с помощью опции `-p`.

А вот пространство сетевых имен пока не изолировано, поэтому из изолированного процесса мы можем видеть сетевые интерфейсы ОС. Изолируем и это с опцией `-n`

4. Повторим `unshare` для того чтобы изолировать сеть.

```
sudo unshare -p -f -n --mount-proc=/home/container/box/proc/ chroot  
/home/container/box/ /bin/bash
```

5. Проверьте IP-адреса внутри контейнера:

```
ip a
```

```
bash-5.2# ip a
```

```
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000  
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

Теперь, благодаря дополнительной изоляции сети, контейнер будет иметь свои собственные сетевые настройки. Это поможет вам увидеть, как изоляция влияет на сетевые параметры.

6. Выйдите из контейнера.

```
exit
```

Заключение: в этой лабораторной работе вы создали и настроили контейнер, проверили его изоляцию, взаимодействие с хостовой системой и возможность работы с сетевыми настройками. Вы также ознакомились с основными командами и методами, используемыми для управления контейнерами в Linux. Конечно, этот контейнер не полностью идентичен тому, что использует для примера Docker, но показывает примитивы благодаря которым возможно создание так называемых контейнеров.