Команды в linux

Ldd /bin/bash – посмотреть ссылки на необходимые библиотеки

chroot gb – команда для смены корневого каталога системы

ps – покажет PID запущенный процессов

top – покажет PID всех запущенных процессов

gcc exemple1.c -o example1.sh – компилируем файл, т.е. файл exemple1.c компилируем в exemple1.sh

./example1.sh – запускаем файл на исполнение

uname – покажет название операционной системы

uname -n – покажет название образа

uname –r – покажет имя ядра системы

uname –m – покажет архитектуру системы

uname –a – покажет всю информацию о системе

$$ - обозначается PID процесса, который выполняется нашей оболочкой

lxc-create -n test123 -t ubuntu -f /usr/share/doc/lxc/exemples/lxc-veth.conf – установка контейнера lxc

lxc-start -d -n test123 – запуск контейнера lxc

lxc-attach -n test123 – подключение к контейнеру lxc

lxc-ls –f – отобразить состояние запущенных контейнеров

lxc-stop -n test123 – остановка контейнера

lxc-destroy -n test123 – удалить контейнеры

**Seminar01**

mkdir ~/testfolder – создает папку в домашней директории

Для смены корневого каталога необходимо:

1. mkdir ~/testfolder
2. mkdir ~/testfolder/bin
3. cp /bin/bash ~/testfolder/bin
4. mkdir ~/testfolder/lib ~/testfolder/lib64
5. cp /lib/x86\_64-linux-gnu/libtinfo.so.6 ~/testfolder/lib
6. cp /lib/x86\_64-linux-gnu/libc.so.6 ~/testfolder/lib
7. cp /lib64/ld-linux-x86-64.so.2 ~/testfolder/lib64/
8. sudo chroot ~/testfolder – команда смены корневого каталога

Для того, чтобы работала команда ls в новом каталоге, необходимо:

1. Вернутся в основной каталог командой exit
2. cp /bin/ls ~/testfolder/bin/
3. cp /lib/x86\_64-linux-gnu/libselinux.so.1 ~/testfolder/lib/
4. cp /lib/x86\_64-linux-gnu/libpcre2-8.so.0 ~/testfolder/lib/
5. sudo chroot ~/testfolder

Все команда ls работает

ps aux – просмотр запущенных процессов

sudo ip netns add testns – команда для создания пространства имен для сети

sudo ip netns exec testns bash – команда для входа в изолированное пространство сети

sudo unshare --net --pid --fork --mount-proc /bin/bash – команда для создания пространства имен процессов (unshare – команда для создания изолированных пространства имен; --net – для изоляции сетевого пространства имен; --pid – для изоляции пространства по процессам; --fork – для изоляции по памяти; --mount-proc – для изоляции процессов)

*От преподавателя:*

Изолированное выполнение процессов с помощью chroot и пространства имен в Linux Подготовка к использованию chroot:

1. Для начала, создадим каталог "testfolder" в домашнем каталоге пользователя и скопируем необходимые исполняемые файлы и библиотеки в эту папку:
2. mkdir ~/testfolder mkdir ~/testfolder/bin
3. cp /bin/bash ~/testfolder/bin mkdir ~/testfolder/lib ~/testfolder/lib64
4. cp /lib/x86\_64-linux-gnu/libtinfo.so.6 ~/testfolder/lib
5. cp /lib/x86\_64-linux-gnu/libc.so.6 ~/testfolder/lib cp /lib64/ld-linux-x86-64.so.2 ~/testfolder/lib64/
6. Запуск команды chroot: Запустим команду chroot для изменения корневой папки нашей текущей среды: sudo chroot ~/testfolder

Примечание: Убедитесь, что вы являетесь владельцем папки "testfolder" и имеете права на выполнение команды chroot. Изменение корневой папки:

Теперь мы находимся в изолированной среде с корнем, отличным от основной файловой системы. Оболочка интерпретатора Bash запущена в этой изолированной среде. ls В этот момент вы можете заметить ошибку: bash: ls: command not found

Это произошло потому, что необходимо также скопировать исполняемые файлы, такие как "ls", и связанные с ними библиотеки в созданный каталог. Добавление дополнительных файлов: Для решения проблемы с отсутствием команды "ls" и других, продолжим копировать необходимые файлы:

1. cp /bin/ls ~/testfolder/bin/
2. cp /lib/x86\_64-linux-gnu/libselinux.so.1 ~/testfolder/lib/
3. cp /lib/x86\_64-linux-gnu/libpcre2-8.so.0 ~/testfolder/lib/
4. Подготовка для запуска: Теперь мы можем выполнить следующие команды для убедиться, что все необходимые файлы скопированы: ldd ~/testfolder/bin/ls

Использование chroot с дополнительными файлами:

1. Попробуем снова войти в изолированную среду через chroot: chroot ~/testfolder Теперь вы должны видеть изменение приветствия оболочки, что означает успешное выполнение команды chroot.
2. Выполните: \_ ls / Вы увидите, что теперь команда "ls" работает. Пространство имен: Однако стоит отметить, что использование chroot имеет недостатки, такие как необходимость дублирования всех необходимых файлов.

Вместо этого, можно рассмотреть использование механизма пространства имен. Пространство имен в Linux обеспечивает изоляцию процессов друг от друга, и может быть более эффективным способом управления изоляцией и ресурсами. Вышеупомянутые инструменты, такие как Docker, также предоставляют более надежные и гибкие средства для изоляции процессов и управления ресурсами. Пример использования сетевого пространства имен: Создание Пространства Имен для Сети:

1. Воспользуемся командой ip для создания сетевого пространства имен. Давайте создадим пространство имен с именем "testns": \_ ip netns add testns Это создаст изолированное сетевое окружение, похожее на свитч, к которому можно подключить процессы.
2. Запуск Процесса в Пространстве Имен: Используя команду ip, мы можем выполнить процесс в созданном пространстве имен: \_ ip netns exec testns bash Это подобно подключению процесса к изолированному свитчу, где процесс работает в собственной виртуальной сетевой среде.
3. Изоляция и Проверка: Внутри изолированной среды мы можем выполнить команды, такие как ip a, чтобы увидеть сетевые настройки. Однако, поскольку в этой среде нет реальных сетевых ресурсов, мы можем увидеть только виртуальные настройки. Даже в этой изолированной среде, мы по-прежнему можем использовать команду ps aux, чтобы увидеть процессы, ограниченные только пространством имен.
4. Просмотр Процессов: Выполнив команду ps aux, мы можем увидеть список всех процессов в текущем пространстве имен. Однако они будут ограничены только к процессам, которые работают в данной изолированной области.

Часть 2: Более Глубокая Изоляция Применяя дополнительные параметры, мы можем углубить уровень изоляции:

1. Изоляция по Процессам и Файловой Системе: \_\_ unshare --net --pid --fork --mount-proc /bin/bash

ps aux unshare Утилита которая позволяет это разграничивать - --net — ограничевает сетевое пространство имен ip a -mount-proc — разграничивает процессы ps aux --fork — изолирует память --pid — изолирует дерево процессов Формально мы внутри контейнера ls ls / ps aux``

**Lesson03**

Установка Docker выполняется по инструкции на сайте <https://docs.docker.com/engine/install/ubuntu/>

docker --help - выведет все доступные команды

docker --version – выведет версию docker

docker info – выведет всю информацию о docker

docker pull ubuntu – получит образ, но не установит его. Для выполнение команды нужно быть root

docker images – покажет полученные образы

docker rmi ubuntu:22.04 – удалит указанный образ

docker rmi $(docker images -aq) – удалит все полученные образы. Не получится удалить образ, который запущен.

docker images –aq – выводит id образов

docker rmi $(docker images -aq) --force – в случае если система не может удалить какой либо образ, его можно удалить добавив --force.

docker run -it ubuntu bash – запустит образ ubuntu и в нем выполнить команду bash, т.е. запустит оболочку (командный интерпритатор). –it говорит о запуске контейнера в интерактивном режиме

docker run ubuntu date – пакажет дату запущенную из контейнера

docker run --publish 8080:80 nginx – запуск nginx

docker run -d --publish 8080:80 nginx – запуск nginx в фоновом режиме

docker rm имя контейнера – удаление контейнера

docker rm $(docker ps -a -q) – удалит не запущенные контейнеры

ps -a – выведет все доступные в системе контейнеры

ps -q – выведет id запущенные контейнеры

ps -a -q – выведет id всех доступных контейнеров

docker ps –a – покажет id запущенного контейнера

docker stop id процесса – останавливает контейнер с указанным id. После повторяем команду docker rm $(docker ps -a -q).

docker system df – покажет место занимаемое контейнерами

docker system prune –af – удалит контейнеры и образы, которые остановленные и не используются

docker run -d ubuntu sleep 15 – запустит контейнер, который будет работать указанное время sleep 15 (15 секунд)

**Lesson04**

apt install cowsay – установка приложения cowsay

/usr/games/cowsay "GeekBrains" – запуск приложения

cat > Dockerfile – создаем Dockerfile с содержанием:

FROM ubuntu:24.04

RUN apt-get update

RUN apt-get install -y cowsay

RUN ln -s /usr/games/cowsay /usr/bin/cowsay

CMD ["cowsay"]

docker build -t cowsaytest . – запускаем файл Dockerfile на исполнение

docker run -it cowsaytest bash – запускаем созданный контейнер

hostname – покажет hostname

cowsay "Hi" – запускаем приложение в контейнере



docker run cowsaytest cowsay "GeekBrains" – запускаем приложение без захода в контейнер

Оптимизируем контейнер для уменьшения веса. Объединим команды и теперь Dockerfile выглядит:

FROM ubuntu:24.04

RUN apt-get update && \

apt-get install -y cowsay && \

ln -s /usr/games/cowsay /usr/bin/cowsay && \

rm -rf /var/lib/apt/lists/\*

CMD ["cowsay"]

Уменьшился его вес

**Seminar02**

sudo apt-get install lxc debootstrap bridge-utils lxc-templates – установка контейнера lxc

sudo apt-get install lxd-installer – установка lxd

sudo lxd initt –инициирование lxd

sudo apt install snapd

sudo snap install core

sudo lxc storage list

sudo apt update

sudo lxc-create -n test123 -t Ubuntu – создания контейнера

sudo lxc-start -n test123 – запускаем контейнер

sudo lxc-attach -n test123 – заходим в контейнер

free –m – проверяем память

exit – вход из контейнера

sudo lxc-stop -n test123 –останавливаем контейнер

sudo nano /var/lib/lxc/test123/config – открываем файл конфигурации

Добавляем в конец файла строку lxc.cgroup2.memory.max = 256M и снова запускаем контейнер, заходим в контейнер и вводим команду free –m и видим ограничение памяти



* У кого не меняется память, попробуйте запустить контейнер с указанием пути к конфигу:  
  sudo lxc-start -n test123 -f /var/lib/lxc/test123/config  
  Мне помогло

Начало формы

Конец формы

udo apt **update**

apt-**get** install lxc debootstrap bridge-utils lxc-templates

apt-**get** install lxd-installer

lxd init(Здесь просто нажимаем на Enter что уствновились значения по умолчанию)

Проверяем

lxc storage list

sudo apt **update**

Создаем

lxc-**create** -n test123 -t ubuntu *--создаем контейнер*

lxc-**start** -n test123 *-- запускаем*

lxc-attach -n teat123 *-- войдем в него*

**free** -m —проверяем пямаять

exit *-- выходим*

lxc-stop -n test123 - -закрываем

nano /var/lib/lxc/test123/config-открываем

lxc.rootfs.path = dir:/var/lib/lxc/test1234/rootfs — путь

lxc.uts.name = test1234 *-- имя*

Network configuration — Конфегурация сети

.

.

.

lxc.cgroup2.memory.max = 256M *-- ограничиваем(В режиме Вставка делаем запись)*

lxc-**start** -n test123 *-- запускаем*

lxc-attach -n teat123 *-- войдем в него*

**free** -m *-- проверяем пямаять*

!!!Видим что наше ограничение работает!!

sudo lxc-stop -n HW02 – останавливаем контейнер

sudo lxc-destroy -n HW02 – удаляем контейнер

**Lesson05**

docker run --name some-mysql -e MYSQL\_ROOT\_PASSWORD=test123 -d mysql:8.0.31 - запускаем docker

docker run --name myphp -d --link some-mysql:db -p 8081:80 phpmyadmin/phpmyadmin

Создадим файл nano compose.yaml со следующим содержанием:

version: ‘3.9’

services:

db:

image: mariadb:10.10.2

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: 12345

adminer:

image: adminer:4.8.1

restart: always

ports:

- 6080:8080

docker-compose build – собирает сервисы, которые прописаны в конфигурационных файлах

docker-compose up – разворачивает и запускает проекта в фоновом режиме

docker-compose up -d – запуск проекта в режиме демон

docker-compose start – запускает ранее остановленные проекты

docker-compose down – останавливает проект и удаляет все сущности проекта

docker-compose stop – просто останавливает проект

docker-compose logs –f <service\_name> - позволяет собрать логи сервиса

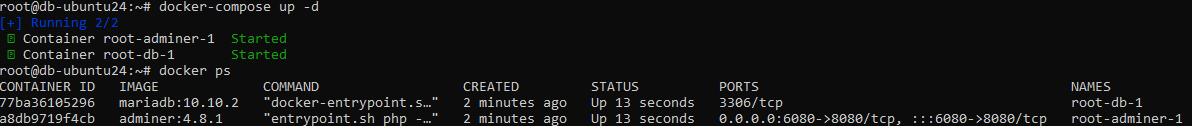
docker-compose exec – выполняет команду в сервисе не заходя в контейнер

docker-compose images – выводит список образов

Для выполнения команды docker-compose up сначала sudo apt install -y docker.io, потом sudo snap install docker и только после этого выполняем docker-compose up.

docker-compose up -d – запускаем в режиме демон

Проверяем по процессам



docker container inspect root-adminer-1 – проверяем работу контейнера

docker container inspect root-db-1 – также проверяем второй контейнер