# ATTACK DEFENSE

by PentesterAcademy

| | |
|---|---|
| **Name** | Cgroups and Namespaces |
| **URL** | https://attackdefense.com/challengedetails?cid=2274 |
| **Type** | Docker Security : Container Basics |

**Important Note:** This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

**Objective:** Follow the manual, learn to use cgroups/namespaces, and create a basic container using basic commands/components!

## Cgroups

Cgroups or Control Groups are a Linux kernel feature to monitor and limit the resource usage of a process or a group of processes.

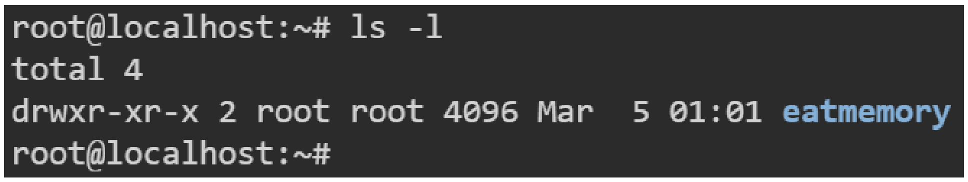
Read more: https://man7.org/linux/man-pages/man7/cgroups.7.html

To test the functionality of cgroup restrictions, we will use eatmemory, a memory eating utility.

**Github:** https://github.com/julman99/eatmemory

**Step 1:** Check the content of the home directory of the root user.

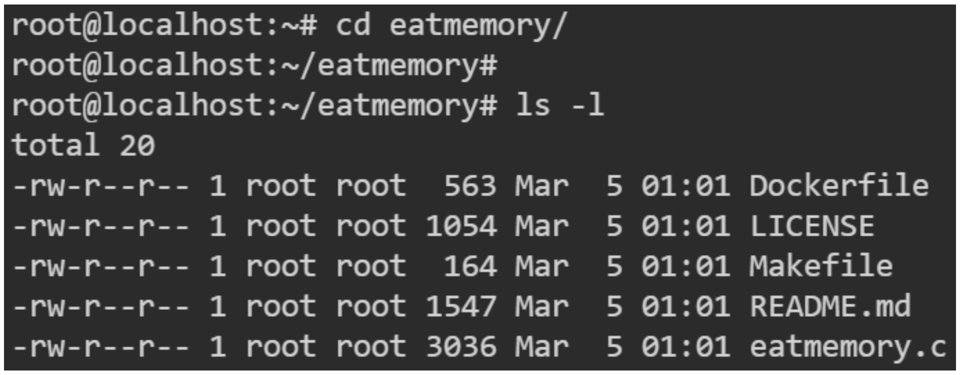
**Command:** ls /root



```
root@localhost:~# ls -l
total 4
drwxr-xr-x 2 root root 4096 Mar  5 01:01 eatmemory
root@localhost:~#
```

**Step 2:** Switch to the eatmemory directory and list the contents.

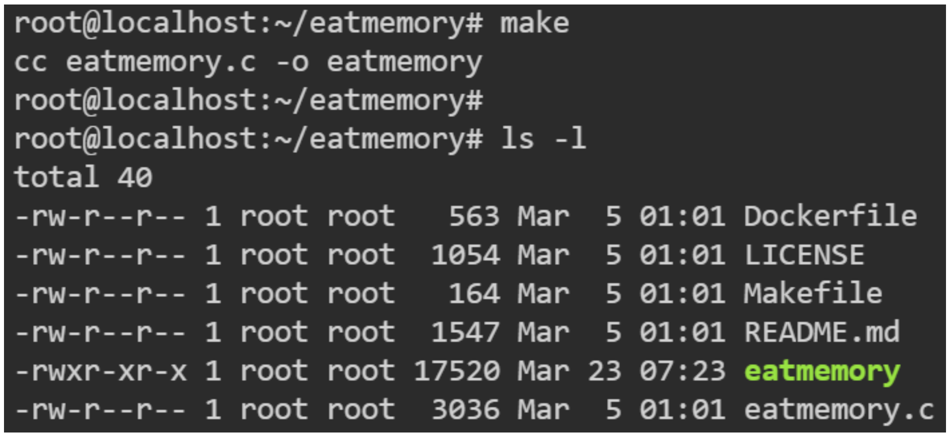
**Commands:**
cd  eatmemory
ls -l



```
root@localhost:~# cd eatmemory/
root@localhost:~/eatmemory#
root@localhost:~/eatmemory# ls -l
total 20
-rw-r--r-- 1 root root  563 Mar  5 01:01 Dockerfile
-rw-r--r-- 1 root root 1054 Mar  5 01:01 LICENSE
-rw-r--r-- 1 root root  164 Mar  5 01:01 Makefile
-rw-r--r-- 1 root root 1547 Mar  5 01:01 README.md
-rw-r--r-- 1 root root 3036 Mar  5 01:01 eatmemory.c
```

**Step 3:** Run make utility to build the binary and check if the binary is created.

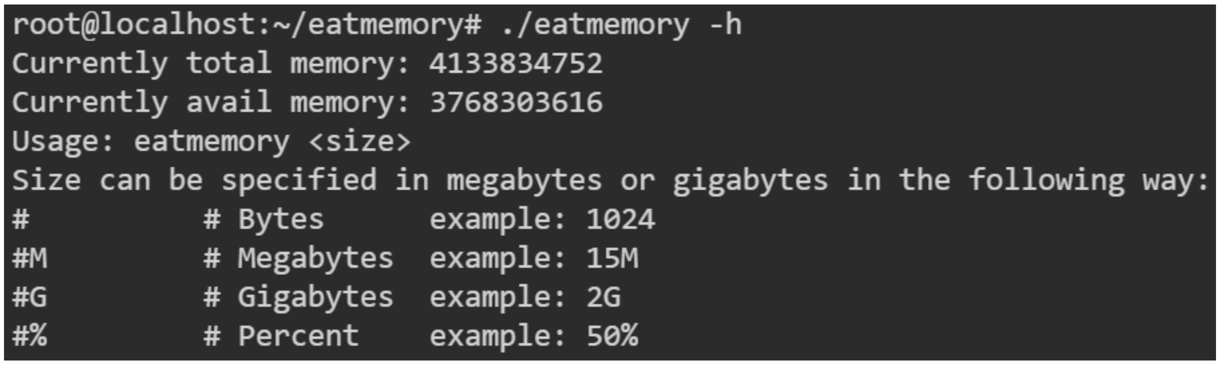
**Commands:**
make
ls -l



```
root@localhost:~/eatmemory# make
cc eatmemory.c -o eatmemory
root@localhost:~/eatmemory#
root@localhost:~/eatmemory# ls -l
total 40
-rw-r--r-- 1 root root   563 Mar  5 01:01 Dockerfile
-rw-r--r-- 1 root root  1054 Mar  5 01:01 LICENSE
-rw-r--r-- 1 root root   164 Mar  5 01:01 Makefile
-rw-r--r-- 1 root root  1547 Mar  5 01:01 README.md
-rwxr-xr-x 1 root root 17520 Mar 23 07:23 eatmemory
-rw-r--r-- 1 root root  3036 Mar  5 01:01 eatmemory.c
```

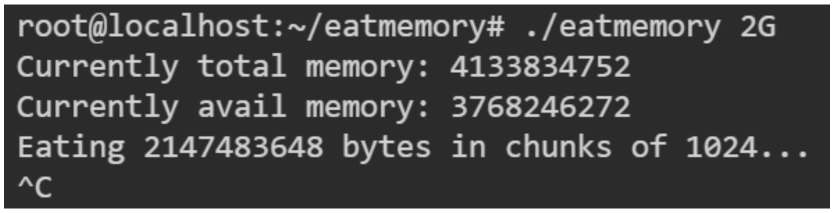
**Step 4:** Execute the binary and check the help menu.

**Command:** ./eatmemory -h


```
root@localhost:~/eatmemory# ./eatmemory -h
Currently total memory: 4133834752
Currently avail memory: 3768303616
Usage: eatmemory <size>
Size can be specified in megabytes or gigabytes in the following way:
#          # Bytes      example: 1024
#M         # Megabytes  example: 15M
#G         # Gigabytes  example: 2G
#%         # Percent    example: 50%
```

**Step 5:** Run the binary to consume 2 GB of memory.

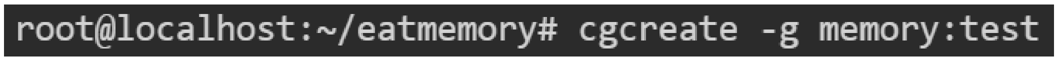
**Command:** ./eatmemory 2G


```
root@localhost:~/eatmemory# ./eatmemory 2G
Currently total memory: 4133834752
Currently avail memory: 3768246272
Eating 2147483648 bytes in chunks of 1024...
^C
```

One can observe that the memory was consumed and eatmemory is still running (before we terminated it with Ctrl+C).

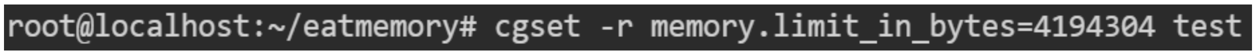
**Step 6:** Create a memory cgroup named “test”.

**Command:** cgcreate -g memory:test


```
root@localhost:~/eatmemory# cgcreate -g memory:test
```

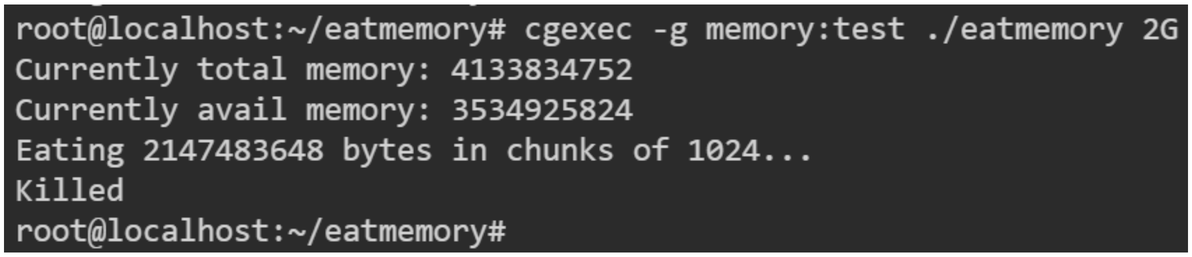
**Step 7:** Set memory limit of 4 MB to this cgroup.

**Command:** cgset -r memory.limit_in_bytes=4194304 test


```
root@localhost:~/eatmemory# cgset -r memory.limit_in_bytes=4194304 test
```

**Step 8:** Execute eatmemory to consume 2 GB memory but this time, confine it with “test” cgroup.

**Command:** cgexec -g memory:test ./eatmemory 2G

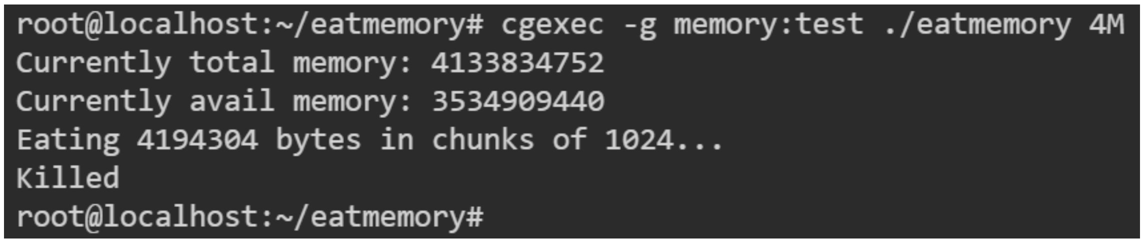


```
root@localhost:~/eatmemory# cgexec -g memory:test ./eatmemory 2G
Currently total memory: 4133834752
Currently avail memory: 3534925824
Eating 2147483648 bytes in chunks of 1024...
Killed
root@localhost:~/eatmemory#
```

One can observe that eatmemory program got killed. This is because it tried to eat more memory  (i.e. 2 GB) than allowed by the “test” cgroup (i.e. 4 MB).

**Step 9:** Try with 4 MB (equals to allowed cgroup memory limit).

**Command:** cgexec -g memory:test ./eatmemory 4M

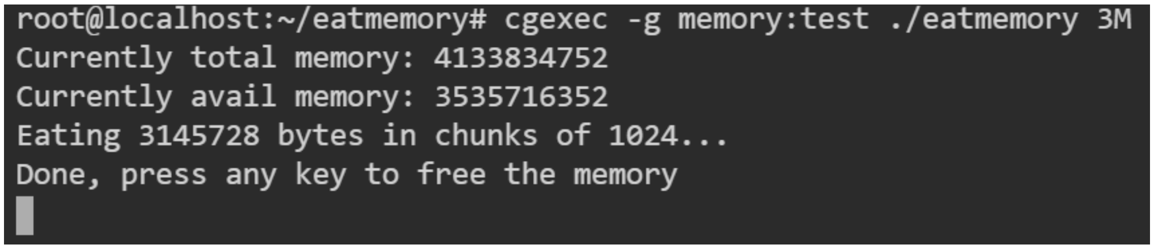


```
root@localhost:~/eatmemory# cgexec -g memory:test ./eatmemory 4M
Currently total memory: 4133834752
Currently avail memory: 3534909440
Eating 4194304 bytes in chunks of 1024...
Killed
root@localhost:~/eatmemory#
```

The program still got killed. Most probably because it tried to eat 4 MB and the program itself must be consuming some memory to run, hence taking total memory consumption by this process beyond 4 MB.

**Step 10:** Try with 3 MB.

**Command:** cgexec -g memory:test ./eatmemory 3M



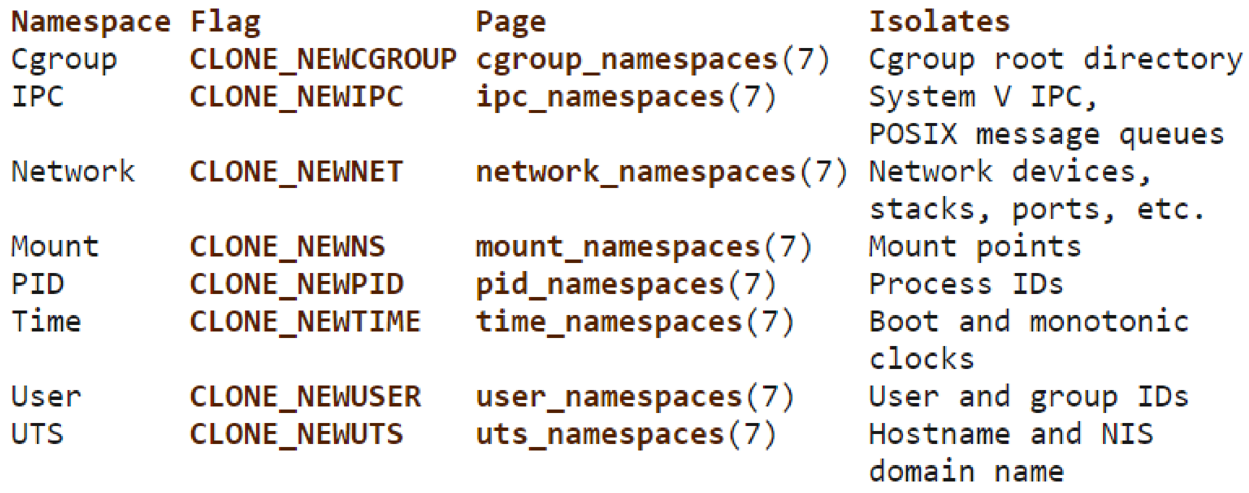
```
root@localhost:~/eatmemory# cgexec -g memory:test ./eatmemory 3M
Currently total memory: 4133834752
Currently avail memory: 3535716352
Eating 3145728 bytes in chunks of 1024...
Done, press any key to free the memory
█
```

This time the process was not killed because it stayed in the limit of 4 MB.

## Namespaces

Namespaces are features of Linux kernel to divide system resources into different logical partitions.

There are different namespaces supported by the kernel.



| Namespace | Flag | Page | Isolates |
|---|---|---|---|
| Cgroup | CLONE_NEWCGROUP | cgroup_namespaces(7) | Cgroup root directory |
| IPC | CLONE_NEWIPC | ipc_namespaces(7) | System V IPC, POSIX message queues |
| Network | CLONE_NEWNET | network_namespaces(7) | Network devices, stacks, ports, etc. |
| Mount | CLONE_NEWNS | mount_namespaces(7) | Mount points |
| PID | CLONE_NEWPID | pid_namespaces(7) | Process IDs |
| Time | CLONE_NEWTIME | time_namespaces(7) | Boot and monotonic clocks |
| User | CLONE_NEWUSER | user_namespaces(7) | User and group IDs |
| UTS | CLONE_NEWUTS | uts_namespaces(7) | Hostname and NIS domain name |

Read more: https://man7.org/linux/man-pages/man7/namespaces.7.html

We will cover two namespaces in this lab.

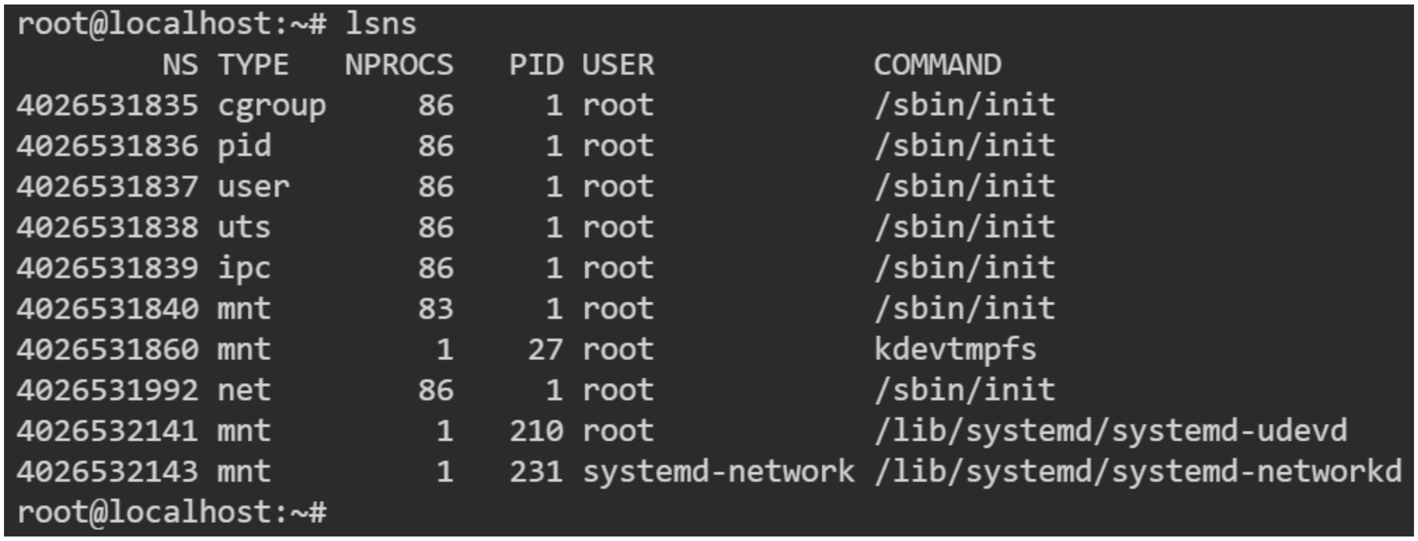
1. Process ID (PID) Namespace
2. Network Namespace

**PID Namespace**

In this exercise, we will create a new PID namespace that is different from the host PID namespace.

**Step 1:** Check existing namespaces.

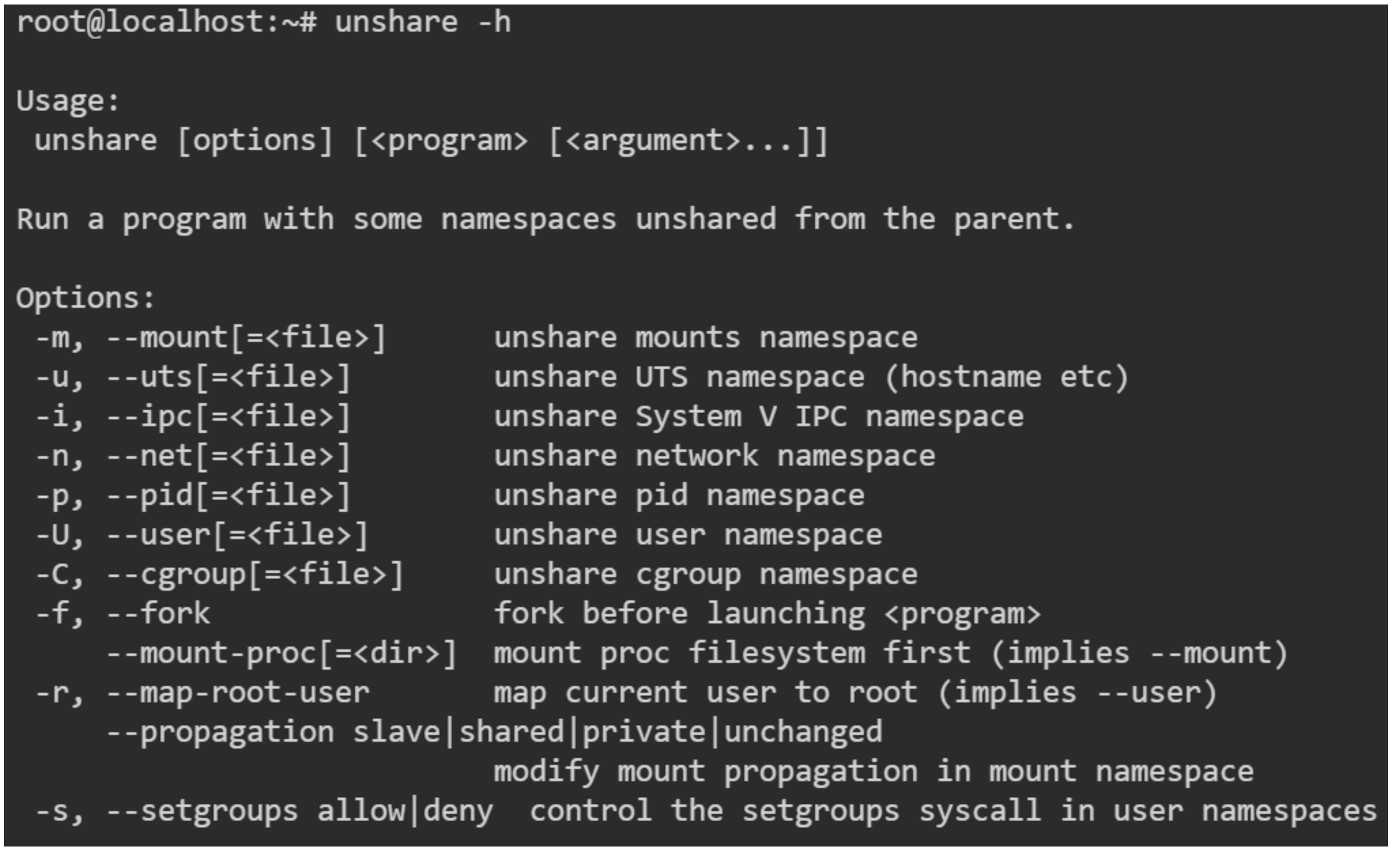
**Command:** lsns



```
root@localhost:~# lsns
        NS TYPE   NPROCS   PID USER            COMMAND
4026531835 cgroup     86     1 root            /sbin/init
4026531836 pid        86     1 root            /sbin/init
4026531837 user       86     1 root            /sbin/init
4026531838 uts        86     1 root            /sbin/init
4026531839 ipc        86     1 root            /sbin/init
4026531840 mnt        83     1 root            /sbin/init
4026531860 mnt         1    27 root            kdevtmpfs
4026531992 net        86     1 root            /sbin/init
4026532141 mnt         1   210 root            /lib/systemd/systemd-udevd
4026532143 mnt         1   231 systemd-network /lib/systemd/systemd-networkd
root@localhost:~#
```

**Step 2:** Unshare command can be used to create a separate PID namespace. Check the help options for this command.

**Command:** unshare -h

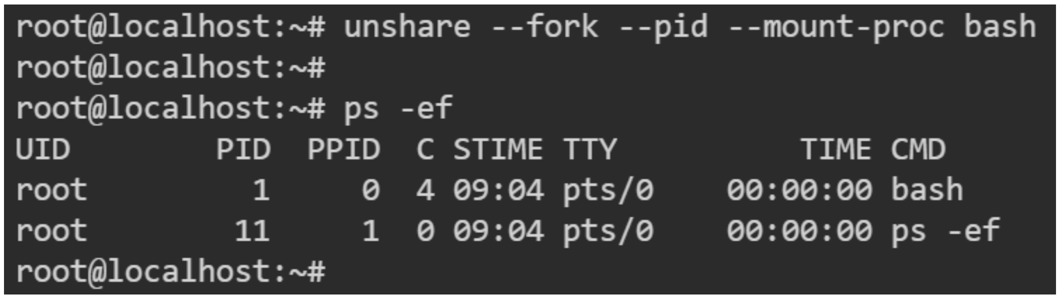

```
root@localhost:~# unshare -h

Usage:
 unshare [options] [<program> [<argument>...]]

Run a program with some namespaces unshared from the parent.

Options:
 -m, --mount[=<file>]      unshare mounts namespace
 -u, --uts[=<file>]        unshare UTS namespace (hostname etc)
 -i, --ipc[=<file>]        unshare System V IPC namespace
 -n, --net[=<file>]        unshare network namespace
 -p, --pid[=<file>]        unshare pid namespace
 -U, --user[=<file>]       unshare user namespace
 -C, --cgroup[=<file>]     unshare cgroup namespace
 -f, --fork                fork before launching <program>
     --mount-proc[=<dir>]  mount proc filesystem first (implies --mount)
 -r, --map-root-user       map current user to root (implies --user)
     --propagation slave|shared|private|unchanged
                           modify mount propagation in mount namespace
 -s, --setgroups allow|deny  control the setgroups syscall in user namespaces
```

**Step 3:** Create a new PID namespace with unshare command.

**Command:** unshare --fork --pid --mount-proc bash

And check the process IDs of processes running in this namespace.

**Command:** ps -ef


```
root@localhost:~# unshare --fork --pid --mount-proc bash
root@localhost:~#
root@localhost:~# ps -ef
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  4 09:04 pts/0    00:00:00 bash
root          11     1  0 09:04 pts/0    00:00:00 ps -ef
root@localhost:~#
```

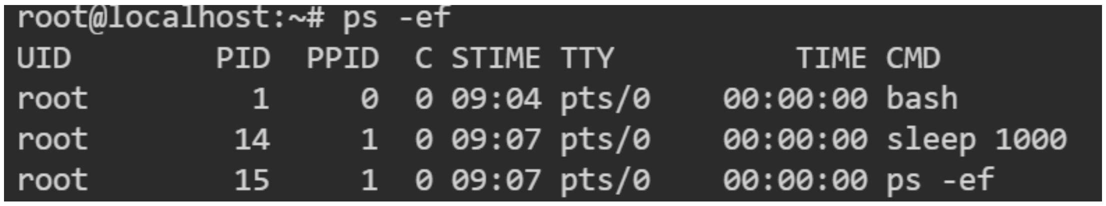
Observe that the bash command that was executed with unshare, is running with PID 1. That clearly shows that it is in a new PID namespace.

**Step 4:** Check the difference in PIDs of a process inside the newly created PID namespace and host machine. To do this, run sleep 1000 command and background it.

**Command:** sleep 1000 &

Check the PID of sleep command, from inside the newly created namespace

**Command:** ps -ef

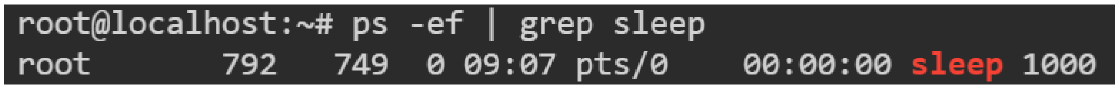


```
root@localhost:~# ps -ef
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 09:04 pts/0    00:00:00 bash
root          14     1  0 09:07 pts/0    00:00:00 sleep 1000
root          15     1  0 09:07 pts/0    00:00:00 ps -ef
```

The PID of sleep process is 14

Open a new terminal by clicking the “Lab Link” or copying the existing terminal link in a new tab. Check the process ID for sleep 1000 process on the host Linux machine.

**Command:** ps -ef | grep sleep



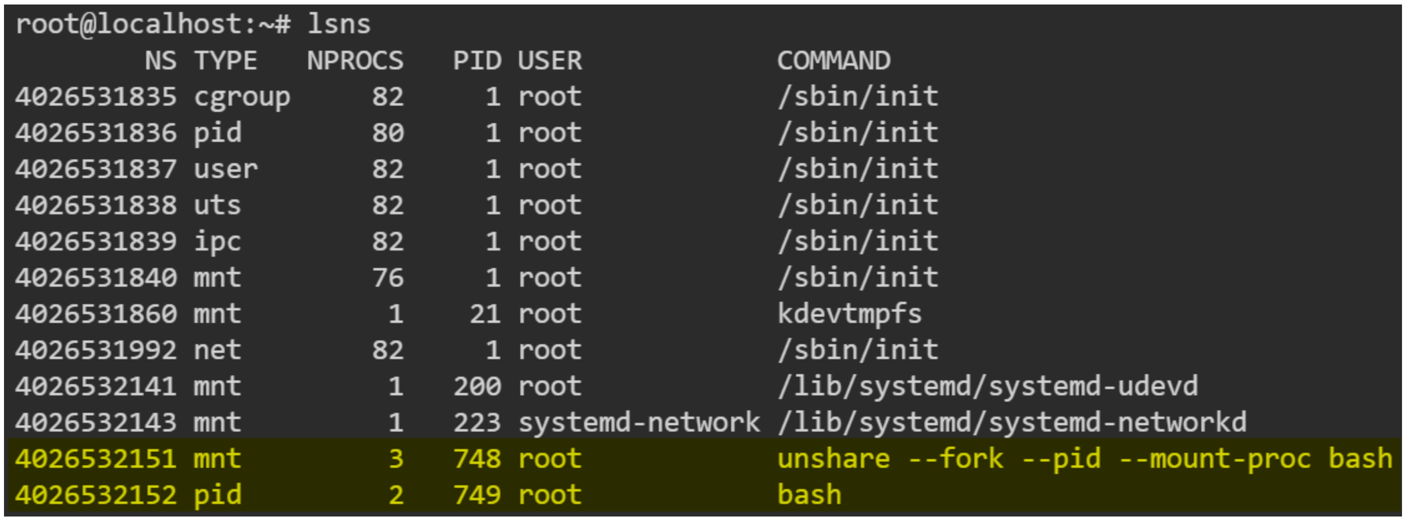
```
root@localhost:~# ps -ef | grep sleep
root        792    749  0 09:07 pts/0    00:00:00 sleep 1000
```

The PID of sleep process is 792

One can observe the difference between the PIDs.

**Step 5:** List the namespaces again (on the host Linux machine) to observe the newly created namespaces.

**Command:** lsns



```
root@localhost:~# lsns
        NS TYPE   NPROCS   PID USER            COMMAND
4026531835 cgroup     82     1 root            /sbin/init
4026531836 pid        80     1 root            /sbin/init
4026531837 user       82     1 root            /sbin/init
4026531838 uts        82     1 root            /sbin/init
4026531839 ipc        82     1 root            /sbin/init
4026531840 mnt        76     1 root            /sbin/init
4026531860 mnt         1    21 root            kdevtmpfs
4026531992 net        82     1 root            /sbin/init
4026532141 mnt         1   200 root            /lib/systemd/systemd-udevd
4026532143 mnt         1   223 systemd-network /lib/systemd/systemd-networkd
4026532151 mnt         3   748 root            unshare --fork --pid --mount-proc bash
4026532152 pid         2   749 root            bash
```

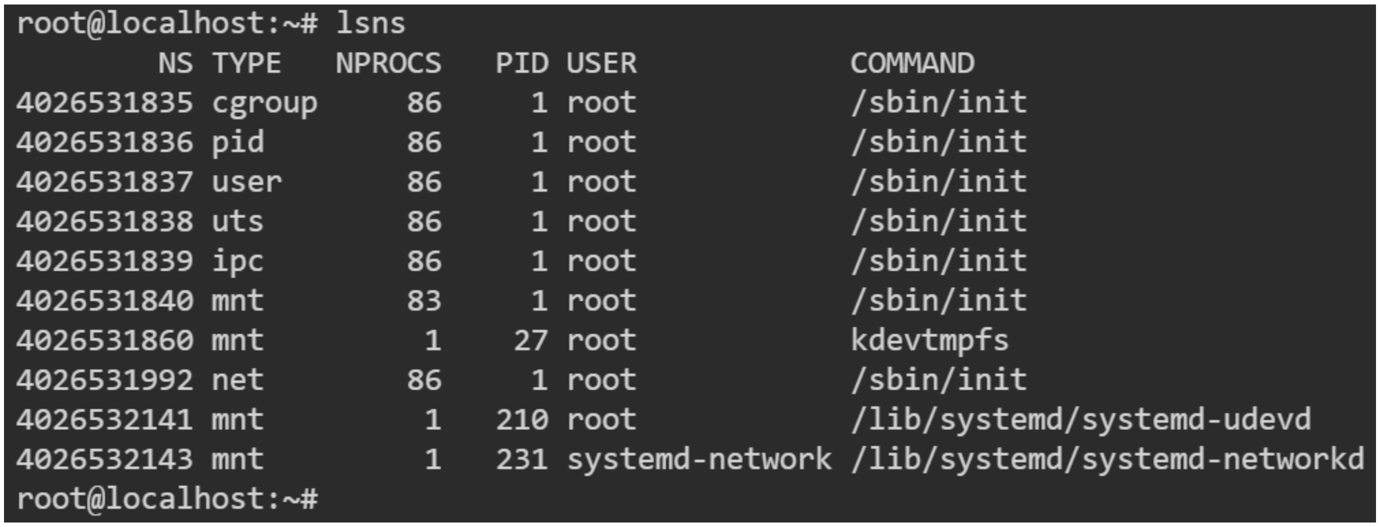
Two more namespaces are available. One of which is a PID namespace.

## Network Namespace

In this exercise, we will create a new network namespace that is different from the host network namespace and add a dummy interface to it.

**Step 1:** Check existing namespaces.

**Command:** lsns

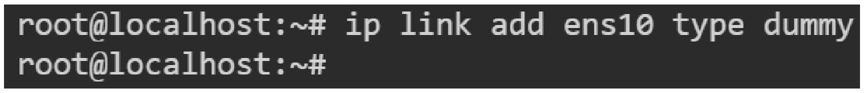


```
root@localhost:~# lsns
        NS TYPE   NPROCS   PID USER            COMMAND
4026531835 cgroup     86     1 root            /sbin/init
4026531836 pid        86     1 root            /sbin/init
4026531837 user       86     1 root            /sbin/init
4026531838 uts        86     1 root            /sbin/init
4026531839 ipc        86     1 root            /sbin/init
4026531840 mnt        83     1 root            /sbin/init
4026531860 mnt         1    27 root            kdevtmpfs
4026531992 net        86     1 root            /sbin/init
4026532141 mnt         1   210 root            /lib/systemd/systemd-udevd
4026532143 mnt         1   231 systemd-network /lib/systemd/systemd-networkd
root@localhost:~#
```

One network namespace (TYPE net)  is present.

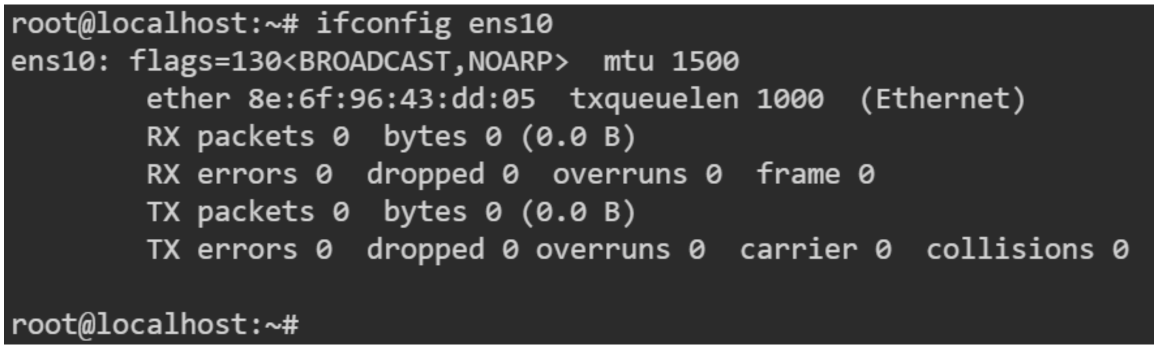
**Step 2:** Add a dummy interface.

**Command:** ip link add ens10 type dummy


```
root@localhost:~# ip link add ens10 type dummy
root@localhost:~#
```

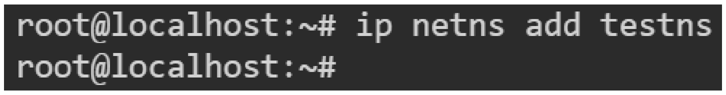
**Step 3:** Verify that the interface is created

**Command:** ifconfig ens10


```
root@localhost:~# ifconfig ens10
ens10: flags=130<BROADCAST,NOARP>  mtu 1500
        ether 8e:6f:96:43:dd:05  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@localhost:~#
```

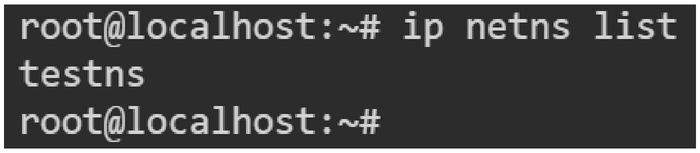
**Step 4:** Create a new network namespace named “testns”

**Command:** ip netns add testns


```
root@localhost:~# ip netns add testns
root@localhost:~#
```

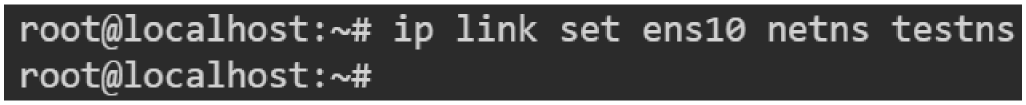
**Step 5:** List network namespaces

**Command:** ip netns list


```
root@localhost:~# ip netns list
testns
root@localhost:~#
```

**Step 6:** Add the dummy interface to “testns” namespace.

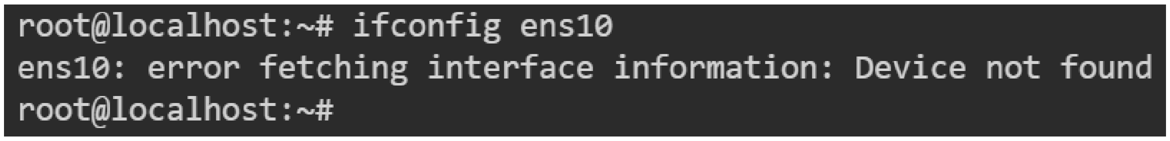
**Command:** ip link set ens10 netns testns



```
root@localhost:~# ip link set ens10 netns testns
root@localhost:~#
```

**Step 7:** Try to list the details of the dummy interface.

**Command:** ifconfig ens10

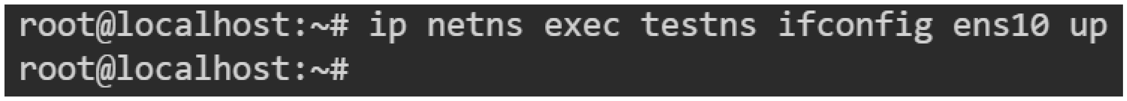


```
root@localhost:~# ifconfig ens10
ens10: error fetching interface information: Device not found
root@localhost:~#
```

The listing failed because the dummy interface is moved to “testns”

**Step 8:** Turn the dummy interface while selecting “testns” namespace.

**Command:** ip netns exec testns ifconfig ens10 up

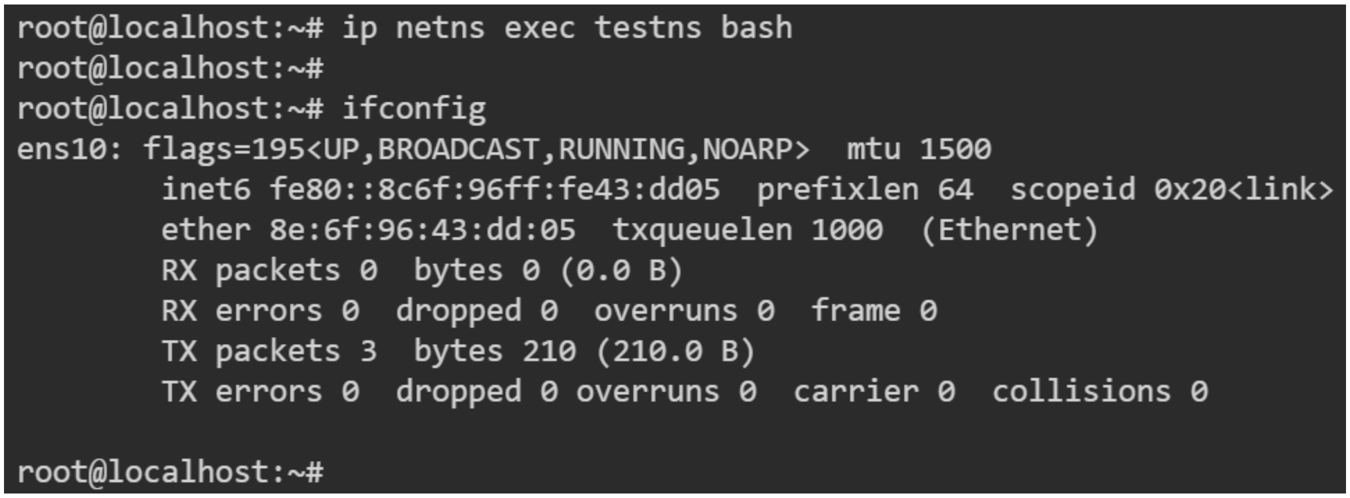


```
root@localhost:~# ip netns exec testns ifconfig ens10 up
root@localhost:~#
```

Command executed normally.

**Step 9:** Start a bash in “testns” namespace and list the interfaces.

**Commands:**
ip netns exec testns bash
ifconfig



```
root@localhost:~# ip netns exec testns bash
root@localhost:~#
root@localhost:~# ifconfig
ens10: flags=195<UP,BROADCAST,RUNNING,NOARP>  mtu 1500
        inet6 fe80::8c6f:96ff:fe43:dd05  prefixlen 64  scopeid 0x20<link>
        ether 8e:6f:96:43:dd:05  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3  bytes 210 (210.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@localhost:~#
```

The dummy interface can be observed in the listing.

## Create Custom Container

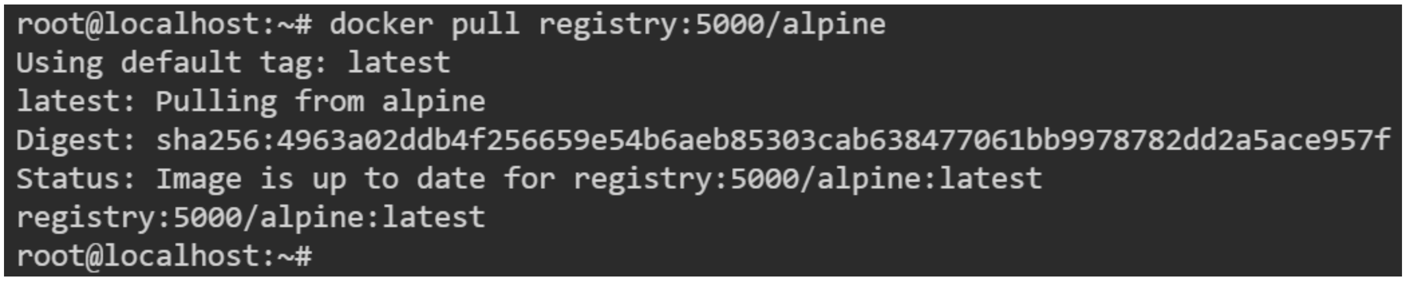
One will require the following components to create a Container.

- Filesystem
- Cgroup
- Namespaces

**Filesystem:** Use the filesystem of Alpine Linux image.

**Step 1:** Pull alpine image from Docker registry

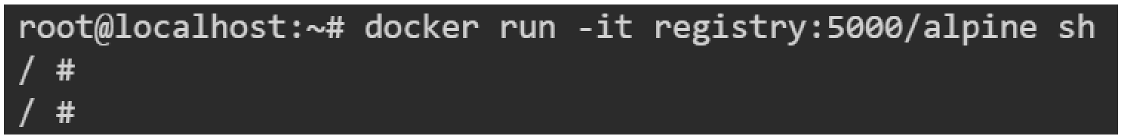
**Command:** docker pull registry:5000/alpine



```
root@localhost:~# docker pull registry:5000/alpine
Using default tag: latest
latest: Pulling from alpine
Digest: sha256:4963a02ddb4f256659e54b6aeb85303cab638477061bb9978782dd2a5ace957f
Status: Image is up to date for registry:5000/alpine:latest
registry:5000/alpine:latest
root@localhost:~#
```

**Step 2:** Pull alpine image from Docker registry

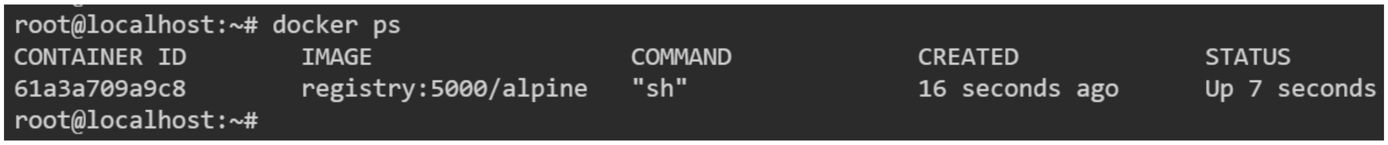
**Command:** docker run -it registry:5000/alpine sh



```
root@localhost:~# docker run -it registry:5000/alpine sh
/ #
/ #
```

**Step 3:** Open a new terminal and check running containers.

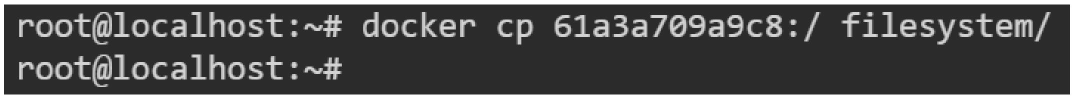
**Command:** docker ps



```
root@localhost:~# docker ps
CONTAINER ID        IMAGE                   COMMAND             CREATED             STATUS
61a3a709a9c8        registry:5000/alpine    "sh"                16 seconds ago      Up 7 seconds
root@localhost:~#
```

**Step 4:** Create a folder /root/filesystem and copy all files from the container.

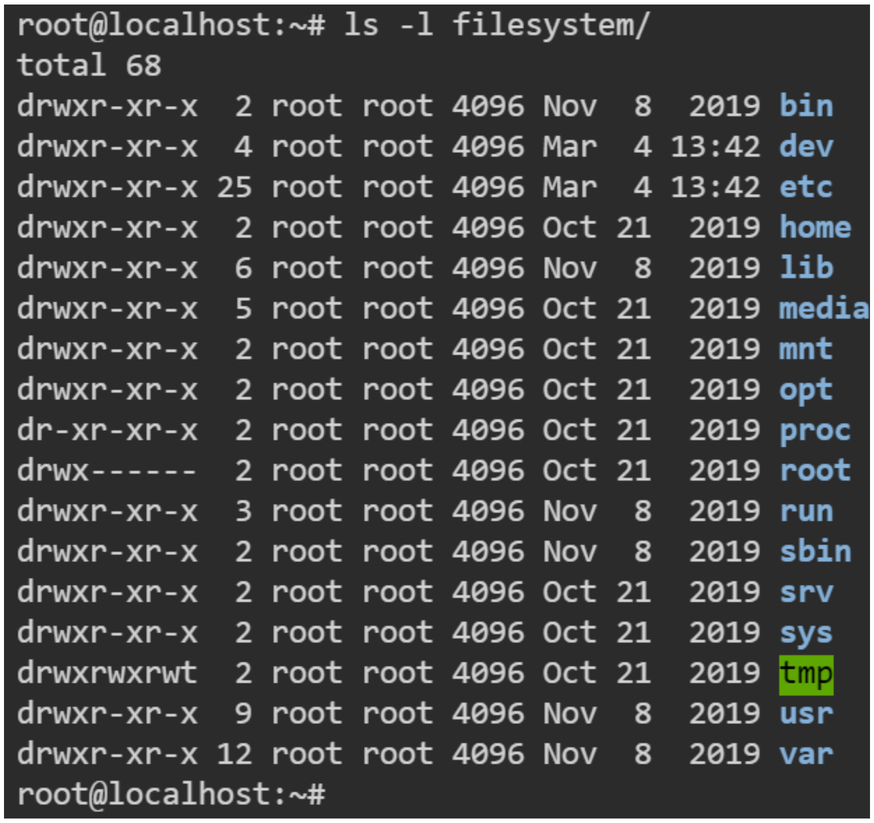
**Commands:**
mkdir /root/filesystem
docker cp 61a3a709a9c8:/ filesystem/



```
root@localhost:~# docker cp 61a3a709a9c8:/ filesystem/
root@localhost:~#
```

**Step 5:** Check the copied filesystem.

**Command:** ls -l filesystem



```
root@localhost:~# ls -l filesystem/
total 68
drwxr-xr-x  2 root root 4096 Nov  8  2019 bin
drwxr-xr-x  4 root root 4096 Mar  4 13:42 dev
drwxr-xr-x 25 root root 4096 Mar  4 13:42 etc
drwxr-xr-x  2 root root 4096 Oct 21  2019 home
drwxr-xr-x  6 root root 4096 Nov  8  2019 lib
drwxr-xr-x  5 root root 4096 Oct 21  2019 media
drwxr-xr-x  2 root root 4096 Oct 21  2019 mnt
drwxr-xr-x  2 root root 4096 Oct 21  2019 opt
dr-xr-xr-x  2 root root 4096 Oct 21  2019 proc
drwx------  2 root root 4096 Oct 21  2019 root
drwxr-xr-x  3 root root 4096 Nov  8  2019 run
drwxr-xr-x  2 root root 4096 Nov  8  2019 sbin
drwxr-xr-x  2 root root 4096 Oct 21  2019 srv
drwxr-xr-x  2 root root 4096 Oct 21  2019 sys
drwxrwxrwt  2 root root 4096 Oct 21  2019 tmp
drwxr-xr-x  9 root root 4096 Nov  8  2019 usr
drwxr-xr-x 12 root root 4096 Nov  8  2019 var
root@localhost:~#
```

**For CGroup and Namespaces use the following script:**

```
1	#! /bin/bash
2	cd filesystem
3	cgcreate -g cpu,cpuacct,memory:testc
4	cgset -r cpu.shares=512 testc
5	cgset -r memory.limit_in_bytes=1000000000 testc
6	cgexec -g "cpu,cpuacct,memory:testc" unshare -fmuipn --mount-proc chroot
/root/filesystem /bin/sh -c "/bin/mount -t proc proc /proc && /bin/sh"
```

**Description:**

**Line 1 :** Bash shebang line

**Line 2:** Making /root/filesystem as present working directory (pwd)

**Line 3:** Creating a Cgroup “testc” to control memory and CPU usage

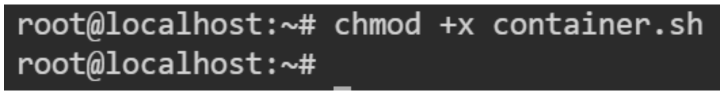
**Line 4:** Setting CPU usage limit for “testc”

**Line 5:** Setting memory limit for “testc”

**Line 6:** Run unshare to create new namespace while using the /root/filesystem as the filesystem and containing it with “testc” cgroup. /bin/sh is the default shell for Alpine Linux, it will be executed in this environment to give us command execution capability.

**Step 6:** Save the provided script as container.sh and make it executable.

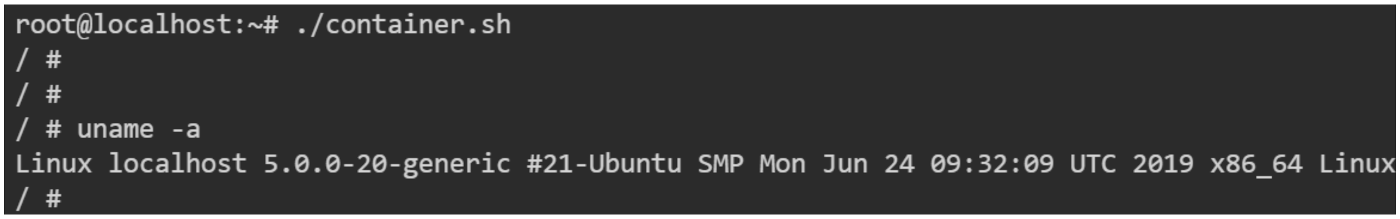
**Command:** chmod +x container.sh



```
root@localhost:~# chmod +x container.sh
root@localhost:~# _
```

**Step 7:** Execute the script and run uname -a from the sh prompt.

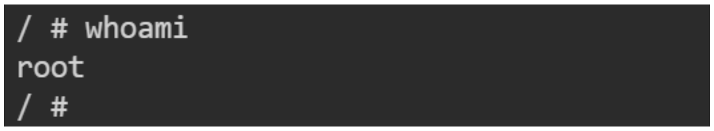
**Commands:**
./container.sh
uname -a



```
root@localhost:~# ./container.sh
/ #
/ #
/ # uname -a
Linux localhost 5.0.0-20-generic #21-Ubuntu SMP Mon Jun 24 09:32:09 UTC 2019 x86_64 Linux
/ #
```

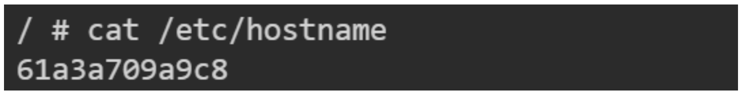
**Step 8:** Execute whoami command to check the current user and check the hostname.

**Command:** whoami



```
/ # whoami
root
/ #
```

**Command:** cat /etc/hostname

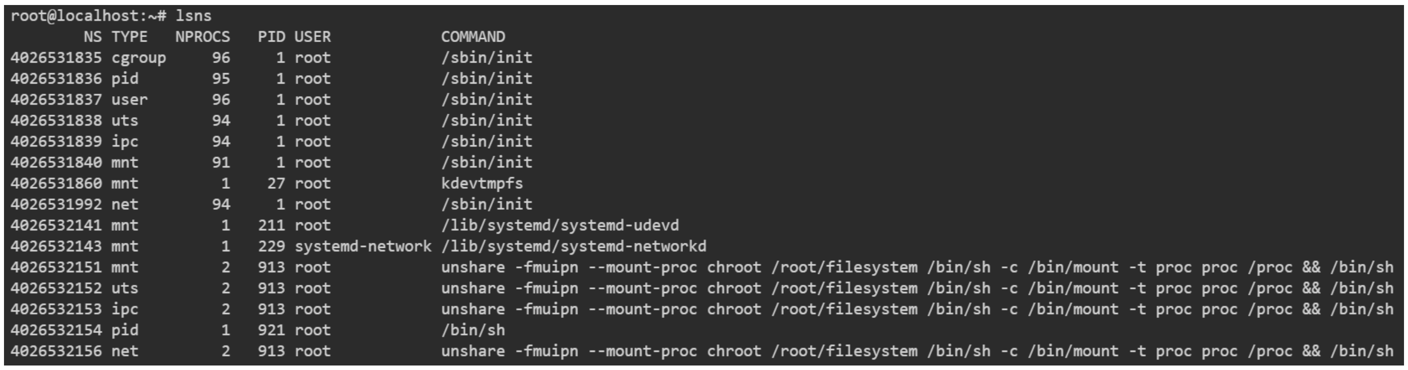


```
/ # cat /etc/hostname
61a3a709a9c8
```

This is a basic container created by using an alpine filesystem and features of the host machine's kernel (Cgroups and Namespaces).

**Step 9:** Open a new terminal and list the namespaces.

**Command:** lsns



```
root@localhost:~# lsns
        NS TYPE   NPROCS   PID USER            COMMAND
4026531835 cgroup     96     1 root            /sbin/init
4026531836 pid        95     1 root            /sbin/init
4026531837 user       96     1 root            /sbin/init
4026531838 uts        94     1 root            /sbin/init
4026531839 ipc        94     1 root            /sbin/init
4026531840 mnt        91     1 root            /sbin/init
4026531860 mnt         1    27 root            kdevtmpfs
4026531992 net        94     1 root            /sbin/init
4026532141 mnt         1   211 root            /lib/systemd/systemd-udevd
4026532143 mnt         1   229 systemd-network /lib/systemd/systemd-networkd
4026532151 mnt         2   913 root            unshare -fmuipn --mount-proc chroot /root/filesystem /bin/sh -c /bin/mount -t proc proc /proc && /bin/sh
4026532152 uts         2   913 root            unshare -fmuipn --mount-proc chroot /root/filesystem /bin/sh -c /bin/mount -t proc proc /proc && /bin/sh
4026532153 ipc         2   913 root            unshare -fmuipn --mount-proc chroot /root/filesystem /bin/sh -c /bin/mount -t proc proc /proc && /bin/sh
4026532154 pid         1   921 root            /bin/sh
4026532156 net         2   913 root            unshare -fmuipn --mount-proc chroot /root/filesystem /bin/sh -c /bin/mount -t proc proc /proc && /bin/sh
```

One can see the newly created namespaces.

**References:**

1. Cgroups (https://man7.org/linux/man-pages/man7/cgroups.7.html)
2. Namespaces ( https://man7.org/linux/man-pages/man7/namespaces.7.html)