DOCKER ASSIGNMENT

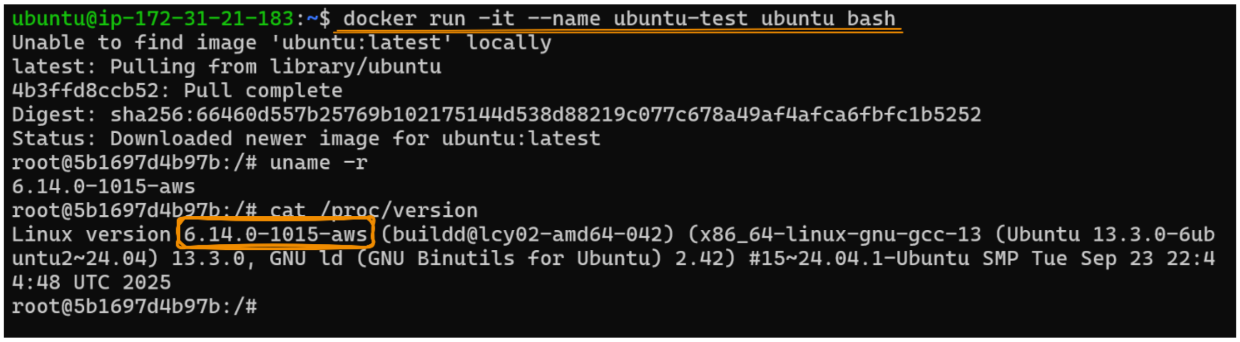
Muhammed Jamzeeth

NOVEMBER 7, 2025

FORTUDE

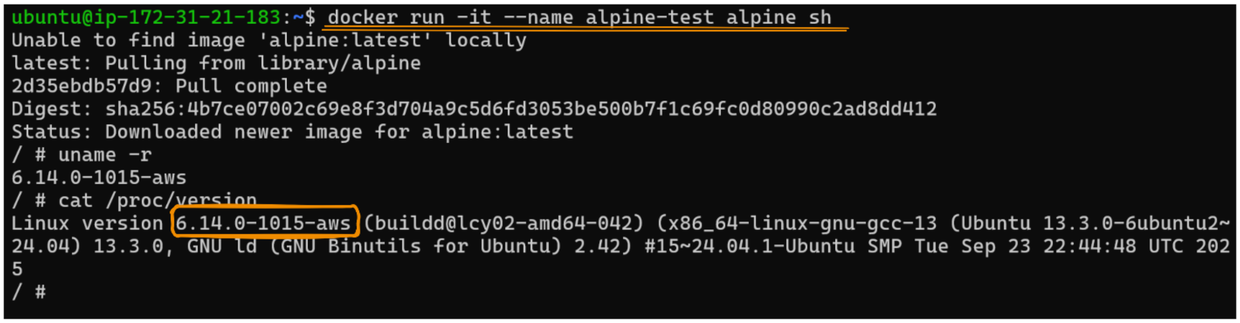
1. Verifying Kernel Sharing

docker run -it --rm ubuntu bash
uname -r
cat /proc/version



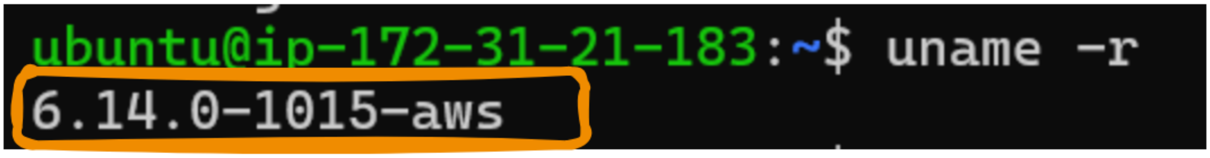
```
ubuntu@ip-172-31-21-183:~$ docker run -it --name ubuntu-test ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
4b3ffd8ccb52: Pull complete
Digest: sha256:66460d557b25769b102175144d538d88219c077c678a49af4afca6fbfc1b5252
Status: Downloaded newer image for ubuntu:latest
root@5b1697d4b97b:/# uname -r
6.14.0-1015-aws
root@5b1697d4b97b:/# cat /proc/version
Linux version 6.14.0-1015-aws (buildd@lcy02-amd64-042) (x86_64-linux-gnu-gcc-13 (Ubuntu 13.3.0-6ub
untu2~24.04) 13.3.0, GNU ld (GNU Binutils for Ubuntu) 2.42) #15~24.04.1-Ubuntu SMP Tue Sep 23 22:4
4:48 UTC 2025
root@5b1697d4b97b:/#
```

docker run -it --rm alpine sh
uname -r
cat /proc/version



```
ubuntu@ip-172-31-21-183:~$ docker run -it --name alpine-test alpine sh
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
2d35ebdb57d9: Pull complete
Digest: sha256:4b7ce07002c69e8f3d704a9c5d6fd3053be500b7f1c69fc0d80990c2ad8dd412
Status: Downloaded newer image for alpine:latest
/ # uname -r
6.14.0-1015-aws
/ # cat /proc/version
Linux version 6.14.0-1015-aws (buildd@lcy02-amd64-042) (x86_64-linux-gnu-gcc-13 (Ubuntu 13.3.0-6ubuntu2~
24.04) 13.3.0, GNU ld (GNU Binutils for Ubuntu) 2.42) #15~24.04.1-Ubuntu SMP Tue Sep 23 22:44:48 UTC 202
5
/ #
```

On host:
Uname -r



```
ubuntu@ip-172-31-21-183:~$ uname -r
6.14.0-1015-aws
```

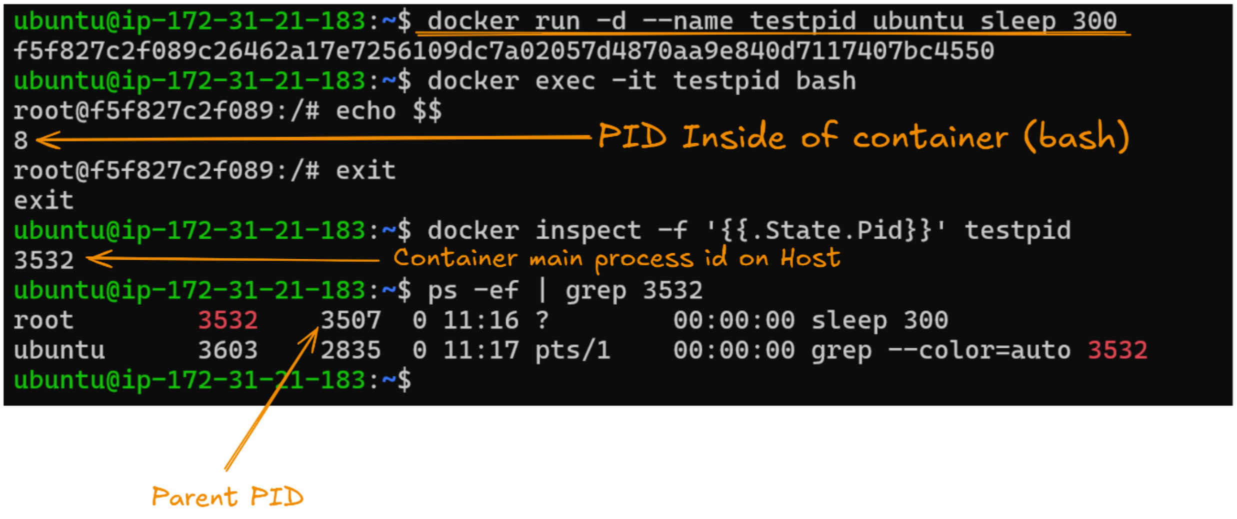
Observation:

- The kernel version is identical across host and containers. (6.14.0-1015-aws)
- This shows containers use the same kernel, not a virtualized one.

Conclusion:
Containers aren’t VMs. They are processes isolated via kernel features, not separate operating systems.

2. Process and PID Mapping

docker run -d --name testpid ubuntu sleep 300
docker exec -it testpid bash
echo $$
On host:
docker inspect -f '{{.State.Pid}}' testpid
ps -ef | grep 3532



```
ubuntu@ip-172-31-21-183:~$ docker run -d --name testpid ubuntu sleep 300
f5f827c2f089c26462a17e7256109dc7a02057d4870aa9e840d7117407bc4550
ubuntu@ip-172-31-21-183:~$ docker exec -it testpid bash
root@f5f827c2f089:/# echo $$
8
root@f5f827c2f089:/# exit
exit
ubuntu@ip-172-31-21-183:~$ docker inspect -f '{{.State.Pid}}' testpid
3532
ubuntu@ip-172-31-21-183:~$ ps -ef | grep 3532
root         3532    3507  0 11:16 ?        00:00:00 sleep 300
ubuntu       3603    2835  0 11:17 pts/1    00:00:00 grep --color=auto 3532
ubuntu@ip-172-31-21-183:~$
```

PID Inside of container (bash)

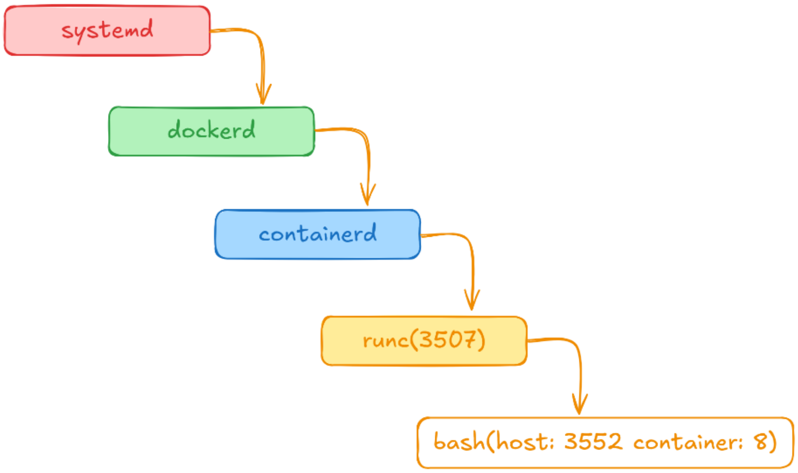
Container main process id on Host

Parent PID

Observation:

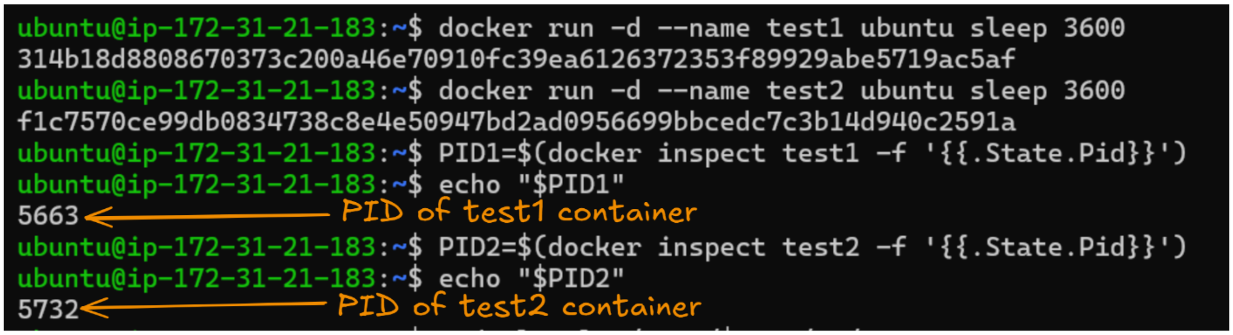
- The container’s PID 8 inside maps to a different PID on the host.
- You’ll see it as a child of containerd-shim or runc.

Process Tree:



3. Exploring Namespace Isolation

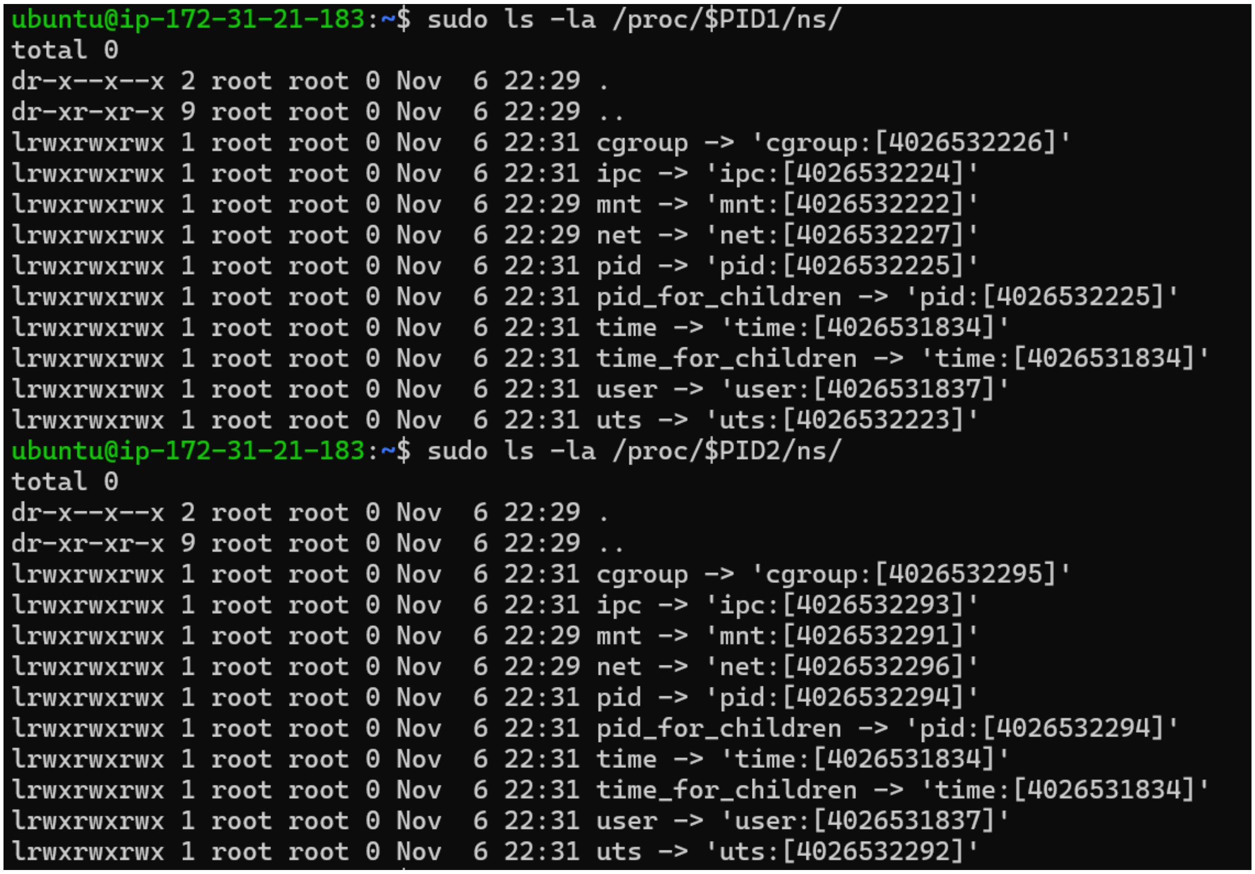
docker run -d --name ns-test1 ubuntu sleep 3600
docker run -d --name ns-test2 ubuntu sleep 3600
PID1=$(docker inspect -f '{{.State.Pid}}' test1)
PID2=$(docker inspect -f '{{.State.Pid}}' test2)



```
ubuntu@ip-172-31-21-183:~$ docker run -d --name test1 ubuntu sleep 3600
314b18d8808670373c200a46e70910fc39ea6126372353f89929abe5719ac5af
ubuntu@ip-172-31-21-183:~$ docker run -d --name test2 ubuntu sleep 3600
f1c7570ce99db0834738c8e4e50947bd2ad0956699bbcedc7c3b14d940c2591a
ubuntu@ip-172-31-21-183:~$ PID1=$(docker inspect test1 -f '{{.State.Pid}}')
ubuntu@ip-172-31-21-183:~$ echo "$PID1"
5663
ubuntu@ip-172-31-21-183:~$ PID2=$(docker inspect test2 -f '{{.State.Pid}}')
ubuntu@ip-172-31-21-183:~$ echo "$PID2"
5732
```

PID of test1 container

PID of test2 container



```
ubuntu@ip-172-31-21-183:~$ sudo ls -la /proc/$PID1/ns/
total 0
dr-x--x--x 2 root root 0 Nov  6 22:29 .
dr-xr-xr-x 9 root root 0 Nov  6 22:29 ..
lrwxrwxrwx 1 root root 0 Nov  6 22:31 cgroup -> 'cgroup:[4026532226]'
lrwxrwxrwx 1 root root 0 Nov  6 22:31 ipc -> 'ipc:[4026532224]'
lrwxrwxrwx 1 root root 0 Nov  6 22:29 mnt -> 'mnt:[4026532222]'
lrwxrwxrwx 1 root root 0 Nov  6 22:29 net -> 'net:[4026532227]'
lrwxrwxrwx 1 root root 0 Nov  6 22:31 pid -> 'pid:[4026532225]'
lrwxrwxrwx 1 root root 0 Nov  6 22:31 pid_for_children -> 'pid:[4026532225]'
lrwxrwxrwx 1 root root 0 Nov  6 22:31 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  6 22:31 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  6 22:31 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Nov  6 22:31 uts -> 'uts:[4026532223]'
ubuntu@ip-172-31-21-183:~$ sudo ls -la /proc/$PID2/ns/
total 0
dr-x--x--x 2 root root 0 Nov  6 22:29 .
dr-xr-xr-x 9 root root 0 Nov  6 22:29 ..
lrwxrwxrwx 1 root root 0 Nov  6 22:31 cgroup -> 'cgroup:[4026532295]'
lrwxrwxrwx 1 root root 0 Nov  6 22:31 ipc -> 'ipc:[4026532293]'
lrwxrwxrwx 1 root root 0 Nov  6 22:29 mnt -> 'mnt:[4026532291]'
lrwxrwxrwx 1 root root 0 Nov  6 22:29 net -> 'net:[4026532296]'
lrwxrwxrwx 1 root root 0 Nov  6 22:31 pid -> 'pid:[4026532294]'
lrwxrwxrwx 1 root root 0 Nov  6 22:31 pid_for_children -> 'pid:[4026532294]'
lrwxrwxrwx 1 root root 0 Nov  6 22:31 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  6 22:31 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  6 22:31 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Nov  6 22:31 uts -> 'uts:[4026532292]'
```

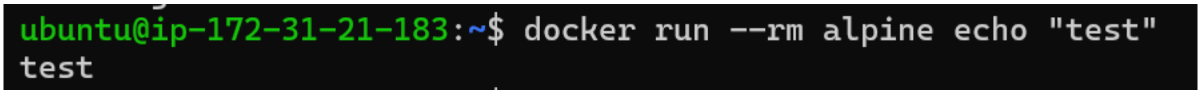
Observation

- The namespace IDs (numbers inside [ ]) differ between $PID1 and $PID2 for most types cgroup, ipc, mnt, net, pid, and uts.
- This indicates that each container has its own isolated namespace for processes, networking, mounts, etc.

- However, the user namespace ID is the same ([4026531837]), meaning both the host and container share the same user namespace typical in default Docker configurations.

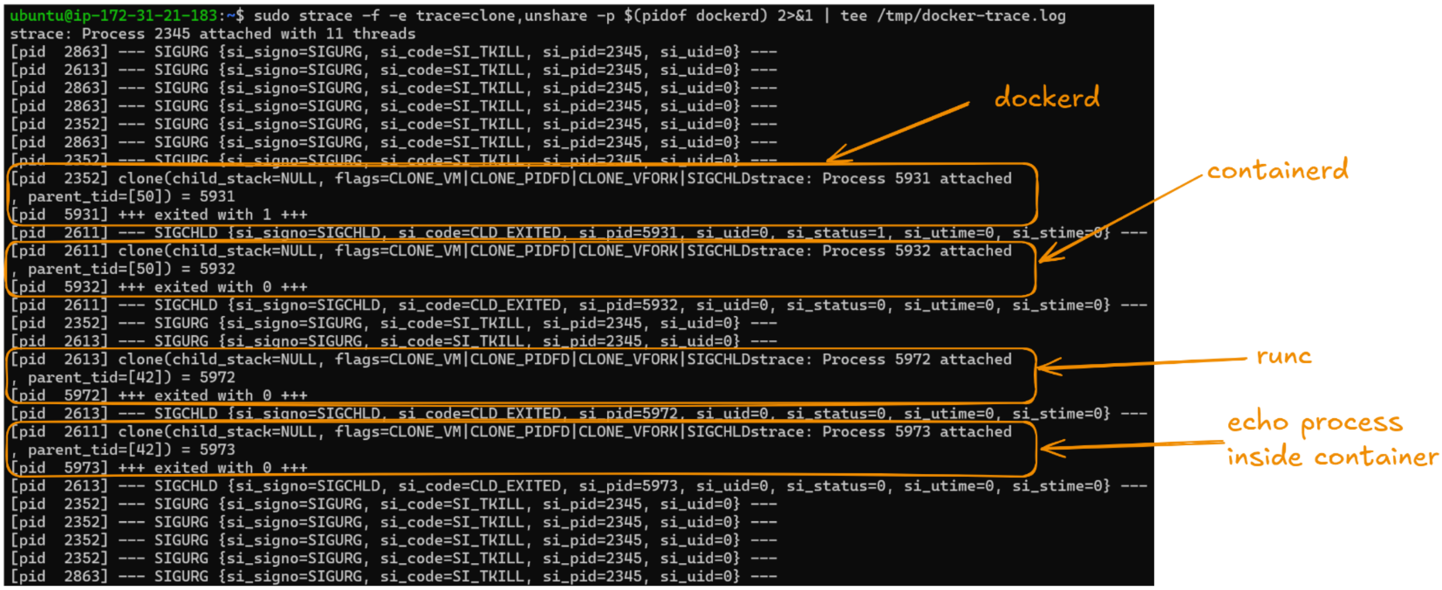
4. Observing New Namespace Creation

# Monitor dockerd for clone() calls
sudo strace -f -e trace=clone,unshare -p $(pidof dockerd) 2>&1 | tee /tmp/docker-trace.log
// After run this command in new terminal start new container to monitor

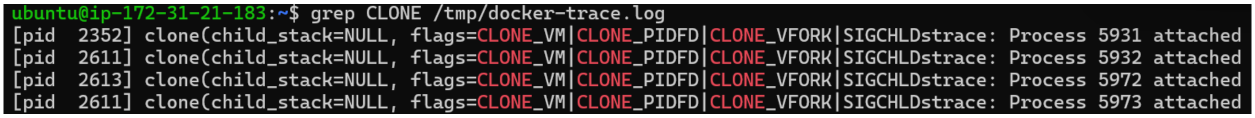


```
ubuntu@ip-172-31-21-183:~$ docker run --rm alpine echo "test"
test
```

// When started go to the trace terminal
Extract clone flags from trace



```
ubuntu@ip-172-31-21-183:~$ sudo strace -f -e trace=clone,unshare -p $(pidof dockerd) 2>&1 | tee /tmp/docker-trace.log
strace: Process 2345 attached with 11 threads
[pid  2863] --- SIGURG {si_signo=SIGURG, si_code=SI_TKILL, si_pid=2345, si_uid=0} ---
[pid  2613] --- SIGURG {si_signo=SIGURG, si_code=SI_TKILL, si_pid=2345, si_uid=0} ---
[pid  2863] --- SIGURG {si_signo=SIGURG, si_code=SI_TKILL, si_pid=2345, si_uid=0} ---
[pid  2863] --- SIGURG {si_signo=SIGURG, si_code=SI_TKILL, si_pid=2345, si_uid=0} ---
[pid  2352] --- SIGURG {si_signo=SIGURG, si_code=SI_TKILL, si_pid=2345, si_uid=0} ---
[pid  2863] --- SIGURG {si_signo=SIGURG, si_code=SI_TKILL, si_pid=2345, si_uid=0} ---
[pid  2352] --- SIGURG {si_signo=SIGURG, si_code=SI_TKILL, si_pid=2345, si_uid=0} ---
[pid  2352] clone(child_stack=NULL, flags=CLONE_VM|CLONE_PIDFD|CLONE_VFORK|SIGCHLDstrace: Process 5931 attached
, parent_tid=[50]) = 5931
[pid  5931] +++ exited with 1 +++
[pid  2611] --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=5931, si_uid=0, si_status=1, si_utime=0, si_stime=0} ---
[pid  2611] clone(child_stack=NULL, flags=CLONE_VM|CLONE_PIDFD|CLONE_VFORK|SIGCHLDstrace: Process 5932 attached
, parent_tid=[50]) = 5932
[pid  5932] +++ exited with 0 +++
[pid  2611] --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=5932, si_uid=0, si_status=0, si_utime=0, si_stime=0} ---
[pid  2352] --- SIGURG {si_signo=SIGURG, si_code=SI_TKILL, si_pid=2345, si_uid=0} ---
[pid  2613] --- SIGURG {si_signo=SIGURG, si_code=SI_TKILL, si_pid=2345, si_uid=0} ---
[pid  2613] clone(child_stack=NULL, flags=CLONE_VM|CLONE_PIDFD|CLONE_VFORK|SIGCHLDstrace: Process 5972 attached
, parent_tid=[42]) = 5972
[pid  5972] +++ exited with 0 +++
[pid  2613] --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=5972, si_uid=0, si_status=0, si_utime=0, si_stime=0} ---
[pid  2611] clone(child_stack=NULL, flags=CLONE_VM|CLONE_PIDFD|CLONE_VFORK|SIGCHLDstrace: Process 5973 attached
, parent_tid=[42]) = 5973
[pid  5973] +++ exited with 0 +++
[pid  2613] --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=5973, si_uid=0, si_status=0, si_utime=0, si_stime=0} ---
[pid  2352] --- SIGURG {si_signo=SIGURG, si_code=SI_TKILL, si_pid=2345, si_uid=0} ---
[pid  2352] --- SIGURG {si_signo=SIGURG, si_code=SI_TKILL, si_pid=2345, si_uid=0} ---
[pid  2352] --- SIGURG {si_signo=SIGURG, si_code=SI_TKILL, si_pid=2345, si_uid=0} ---
[pid  2352] --- SIGURG {si_signo=SIGURG, si_code=SI_TKILL, si_pid=2345, si_uid=0} ---
[pid  2863] --- SIGURG {si_signo=SIGURG, si_code=SI_TKILL, si_pid=2345, si_uid=0} ---
```



```
ubuntu@ip-172-31-21-183:~$ grep CLONE /tmp/docker-trace.log
[pid  2352] clone(child_stack=NULL, flags=CLONE_VM|CLONE_PIDFD|CLONE_VFORK|SIGCHLDstrace: Process 5931 attached
[pid  2611] clone(child_stack=NULL, flags=CLONE_VM|CLONE_PIDFD|CLONE_VFORK|SIGCHLDstrace: Process 5932 attached
[pid  2613] clone(child_stack=NULL, flags=CLONE_VM|CLONE_PIDFD|CLONE_VFORK|SIGCHLDstrace: Process 5972 attached
[pid  2611] clone(child_stack=NULL, flags=CLONE_VM|CLONE_PIDFD|CLONE_VFORK|SIGCHLDstrace: Process 5973 attached
```

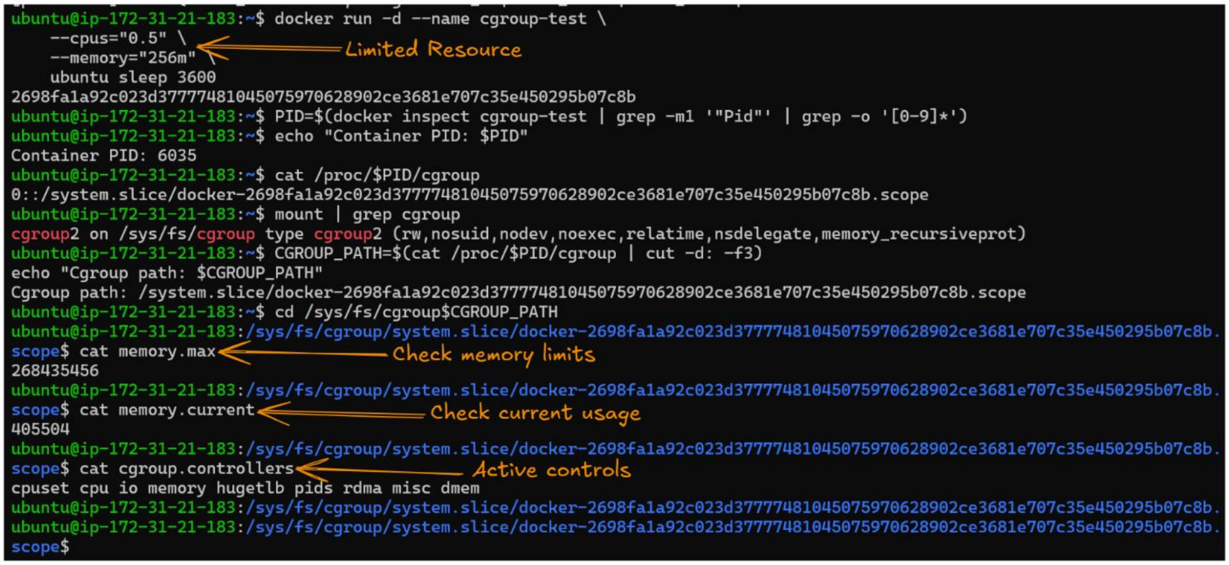
Conclusion
This experiment demonstrates that when a container is created, Docker (through dockerd → containerd → runc) uses the Linux clone() system call with namespace flags to isolate processes, mounts, networks, and hostnames.
Each new container triggers these calls, effectively creating a new namespace context that makes containers appear as independent systems even though they share the same Linux kernel.
Note:

Observation Note:

- In the strace output, namespace creation flags (e.g., CLONE_NEWNS, CLONE_NEWPID, CLONE_NEWNET) were not visible.
- This is because the trace was attached to dockerd, which does not directly create namespaces.
- The actual namespace creation happens inside runc, a short-lived process that runs briefly to set up the container and then exits.
- Therefore, only normal clone() calls (process/thread spawns) were captured, not the namespace-related ones.

5. Investigating cgroup Assignments

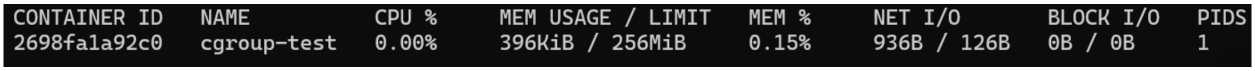
docker run -d --name cgroup-test \ --cpus="0.5" \ --memory="256m" \ ubuntu sleep 3600



```
ubuntu@ip-172-31-21-183:~$ docker run -d --name cgroup-test \
    --cpus="0.5" \
    --memory="256m" \
    ubuntu sleep 3600
2698fa1a92c023d37777481045075970628902ce3681e707c35e450295b07c8b
ubuntu@ip-172-31-21-183:~$ PID=$(docker inspect cgroup-test | grep -m1 '"Pid"' | grep -o '[0-9]*')
ubuntu@ip-172-31-21-183:~$ echo "Container PID: $PID"
Container PID: 6035
ubuntu@ip-172-31-21-183:~$ cat /proc/$PID/cgroup
0::/system.slice/docker-2698fa1a92c023d37777481045075970628902ce3681e707c35e450295b07c8b.scope
ubuntu@ip-172-31-21-183:~$ mount | grep cgroup
cgroup2 on /sys/fs/cgroup type cgroup2 (rw,nosuid,nodev,noexec,relatime,nsdelegate,memory_recursiveprot)
ubuntu@ip-172-31-21-183:~$ CGROUP_PATH=$(cat /proc/$PID/cgroup | cut -d: -f3)
echo "Cgroup path: $CGROUP_PATH"
Cgroup path: /system.slice/docker-2698fa1a92c023d37777481045075970628902ce3681e707c35e450295b07c8b.scope
ubuntu@ip-172-31-21-183:~$ cd /sys/fs/cgroup$CGROUP_PATH
ubuntu@ip-172-31-21-183:/sys/fs/cgroup/system.slice/docker-2698fa1a92c023d37777481045075970628902ce3681e707c35e450295b07c8b.
scope$ cat memory.max
268435456
ubuntu@ip-172-31-21-183:/sys/fs/cgroup/system.slice/docker-2698fa1a92c023d37777481045075970628902ce3681e707c35e450295b07c8b.
scope$ cat memory.current
405504
ubuntu@ip-172-31-21-183:/sys/fs/cgroup/system.slice/docker-2698fa1a92c023d37777481045075970628902ce3681e707c35e450295b07c8b.
scope$ cat cgroup.controllers
cpuset cpu io memory hugetlb pids rdma misc dmem
ubuntu@ip-172-31-21-183:/sys/fs/cgroup/system.slice/docker-2698fa1a92c023d37777481045075970628902ce3681e707c35e450295b07c8b.
ubuntu@ip-172-31-21-183:/sys/fs/cgroup/system.slice/docker-2698fa1a92c023d37777481045075970628902ce3681e707c35e450295b07c8b.
scope$
```

Monitor in real-time:

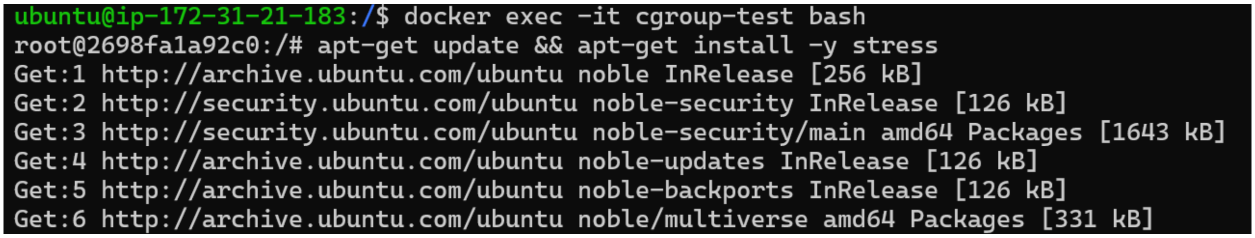
docker stats cgroup-test



```
CONTAINER ID   NAME          CPU %     MEM USAGE / LIMIT   MEM %     NET I/O         BLOCK I/O   PIDS
2698fa1a92c0   cgroup-test   0.00%     396KiB / 256MiB     0.15%     936B / 126B     0B / 0B     1
```

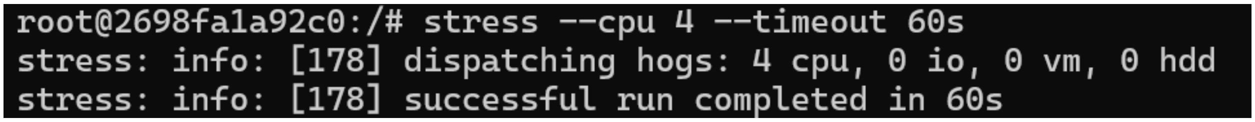
6. Resource Behavior Under Load

Install stress tool:
apt-get update && apt-get install -y stress



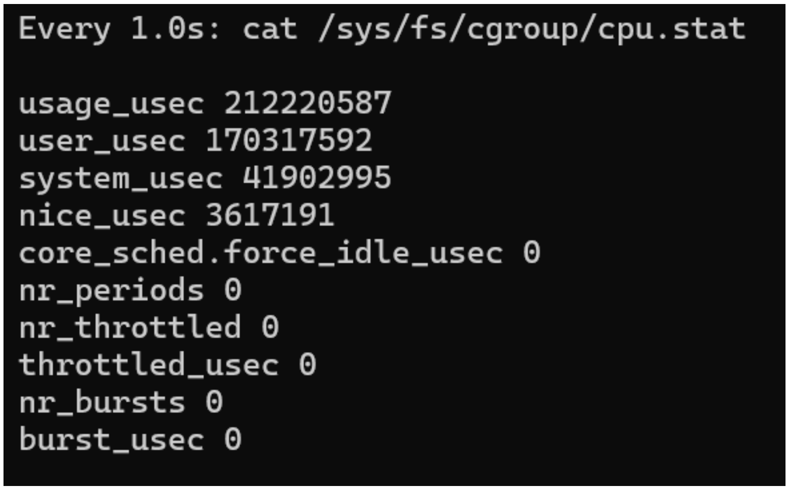
```
ubuntu@ip-172-31-21-183:/$ docker exec -it cgroup-test bash
root@2698fa1a92c0:/# apt-get update && apt-get install -y stress
Get:1 http://archive.ubuntu.com/ubuntu noble InRelease [256 kB]
Get:2 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:3 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [1643 kB]
Get:4 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:5 http://archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:6 http://archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [331 kB]
```

Install stress tool apt-get update && apt-get install -y stress:



```
root@2698fa1a92c0:/# stress --cpu 4 --timeout 60s
stress: info: [178] dispatching hogs: 4 cpu, 0 io, 0 vm, 0 hdd
stress: info: [178] successful run completed in 60s
```

Watch cgroup metrics :
watch -n 1 "cat /sys/fs/cgroup$CGROUP_PATH/cpu.stat"



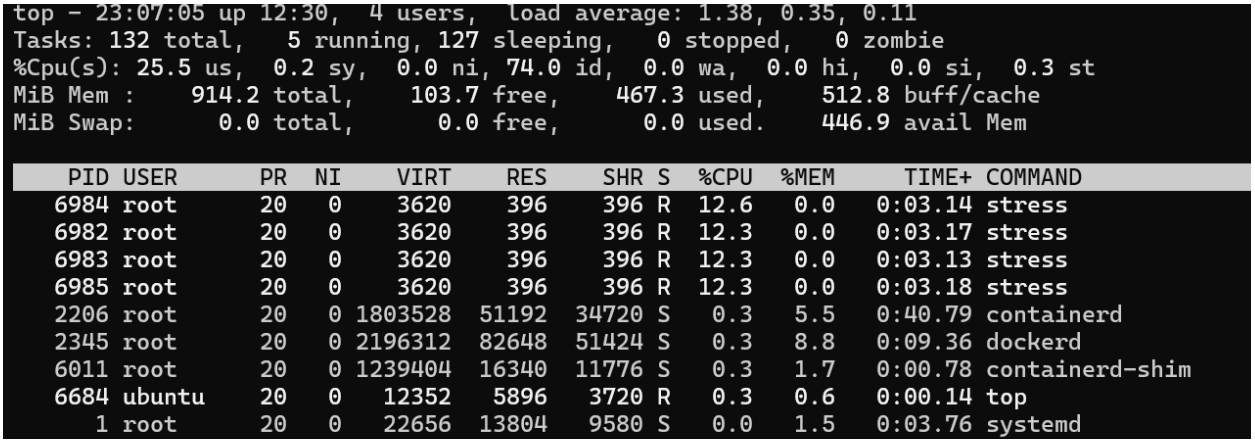
```
Every 1.0s: cat /sys/fs/cgroup/cpu.stat

usage_usec 212220587
user_usec 170317592
system_usec 41902995
nice_usec 3617191
core_sched.force_idle_usec 0
nr_periods 0
nr_throttled 0
throttled_usec 0
nr_bursts 0
burst_usec 0
```

Observation:

- CPU usage capped at ~50% despite 4 stress workers
- Container can't exceed limits even under full load

Watch docker stats:
docker stats cgroup-test



```
top - 23:07:05 up 12:30,  4 users,  load average: 1.38, 0.35, 0.11
Tasks: 132 total,   5 running, 127 sleeping,   0 stopped,   0 zombie
%Cpu(s): 25.5 us,  0.2 sy,  0.0 ni, 74.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.3 st
MiB Mem :    914.2 total,    103.7 free,    467.3 used,    512.8 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.    446.9 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   6984 root      20   0    3620    396    396 R  12.6   0.0   0:03.14 stress
   6982 root      20   0    3620    396    396 R  12.3   0.0   0:03.17 stress
   6983 root      20   0    3620    396    396 R  12.3   0.0   0:03.13 stress
   6985 root      20   0    3620    396    396 R  12.3   0.0   0:03.18 stress
   2206 root      20   0 1803528  51192  34720 S   0.3   5.5   0:40.79 containerd
   2345 root      20   0 2196312  82648  51424 S   0.3   8.8   0:09.36 dockerd
   6011 root      20   0 1239404  16340  11776 S   0.3   1.7   0:00.78 containerd-shim
   6684 ubuntu    20   0   12352   5896   3720 R   0.3   0.6   0:00.14 top
      1 root      20   0   22656  13804   9580 S   0.0   1.5   0:03.76 systemd
```

7. Filesystem Layer Analysis

Navigate to overlay2 directory:
sudo ls -la cd /var/lib/docker/overlay2/

# Find specific container's layers CONTAINER_ID=$(docker ps -qf name=cgroup-test) sudo docker inspect $CONTAINER_ID | grep -E "UpperDir|LowerDir|MergedDir"

# Get the paths UPPER=$(sudo docker inspect $CONTAINER_ID -f '{{.GraphDriver.Data.UpperDir}}')
echo "Upper (writable): $UPPER"

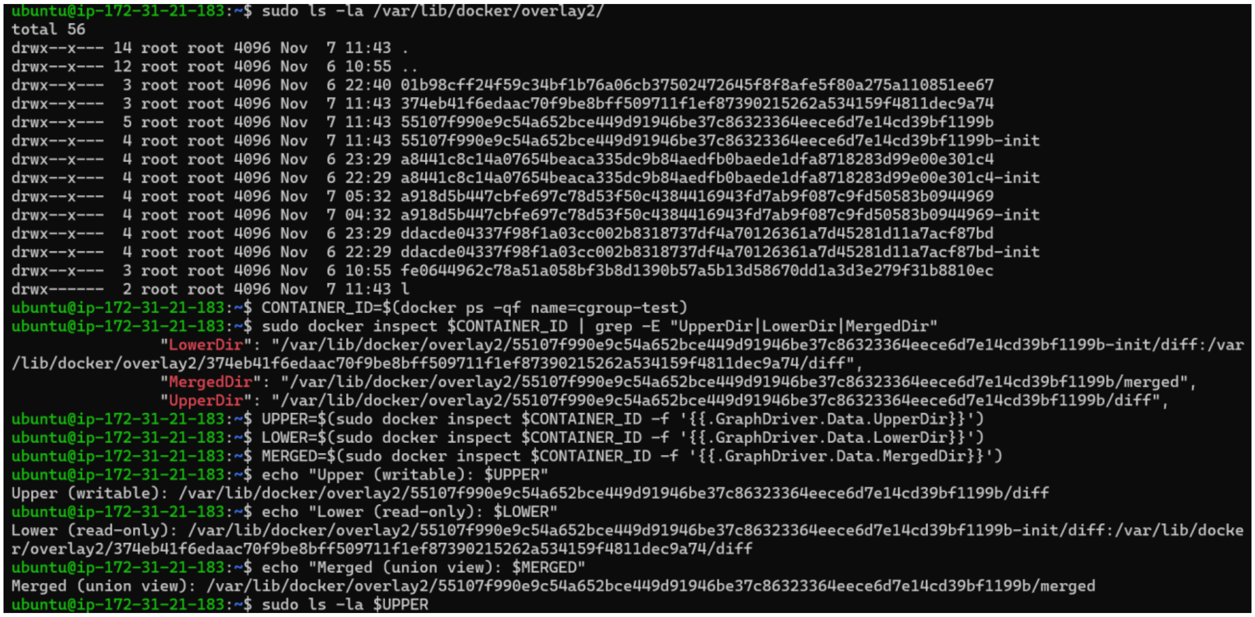
# Check what's in the writable layer
sudo ls -la $UPPER
# Create a file in container
docker exec cgroup-test bash -c "echo 'persistent data' > /testfile.txt"
# Check it appears in UpperDir sudo cat $UPPER/testfile.txt
# Stop container (layer persists)
docker stop cgroup-test

sudo ls $UPPER # Still exists!

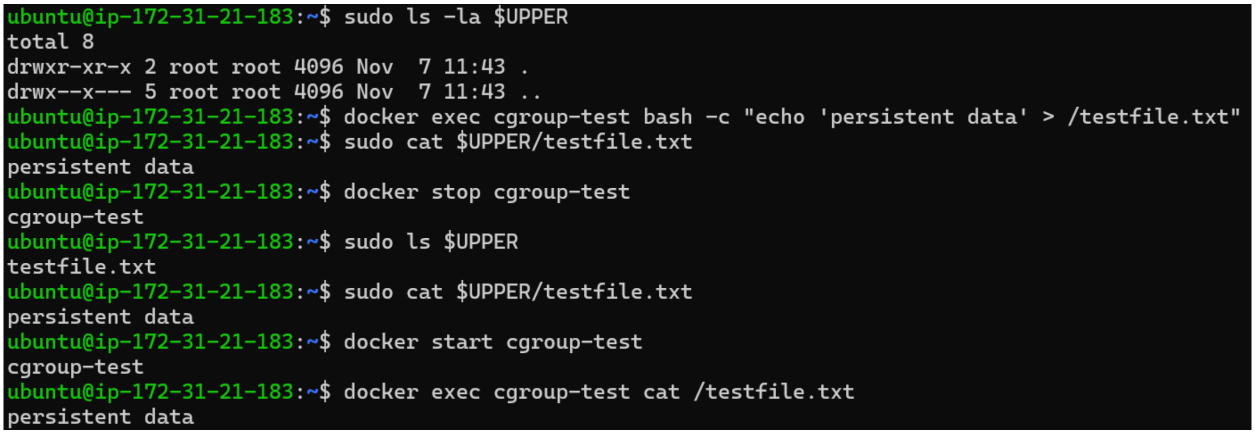
# Start again
docker start cgroup-test
docker exec cgroup-test cat /testfile.txt *# File still there!*

# Remove container (layer deleted)
docker rm cgroup-test

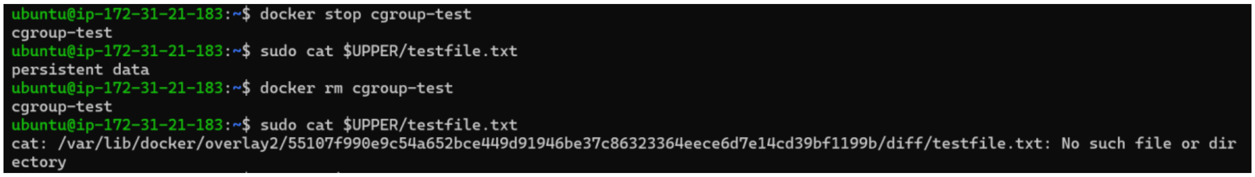
# Check again
sudo cat $UPPER/testfile.txt # You cannot find the file



```
ubuntu@ip-172-31-21-183:~$ sudo ls -la /var/lib/docker/overlay2/
total 56
drwx--x--- 14 root root 4096 Nov  7 11:43 .
drwx--x--- 12 root root 4096 Nov  6 10:55 ..
drwx--x---  3 root root 4096 Nov  6 22:40 01b98cff24f59c34bf1b76a06cb37502472645f8f8afe5f80a275a110851ee67
drwx--x---  3 root root 4096 Nov  7 11:43 374eb41f6edaac70f9be8bff509711f1ef87390215262a534159f4811dec9a74
drwx--x---  5 root root 4096 Nov  7 11:43 55107f990e9c54a652bce449d91946be37c86323364eece6d7e14cd39bf1199b
drwx--x---  4 root root 4096 Nov  7 11:43 55107f990e9c54a652bce449d91946be37c86323364eece6d7e14cd39bf1199b-init
drwx--x---  4 root root 4096 Nov  6 23:29 a8441c8c14a07654beaca335dc9b84aedfb0baede1dfa8718283d99e00e301c4
drwx--x---  4 root root 4096 Nov  6 22:29 a8441c8c14a07654beaca335dc9b84aedfb0baede1dfa8718283d99e00e301c4-init
drwx--x---  4 root root 4096 Nov  7 05:32 a918d5b447cbfe697c78d53f50c4384416943fd7ab9f087c9fd50583b0944969
drwx--x---  4 root root 4096 Nov  7 04:32 a918d5b447cbfe697c78d53f50c4384416943fd7ab9f087c9fd50583b0944969-init
drwx--x---  4 root root 4096 Nov  6 23:29 ddacde04337f98f1a03cc002b8318737df4a70126361a7d45281d11a7acf87bd
drwx--x---  4 root root 4096 Nov  6 22:29 ddacde04337f98f1a03cc002b8318737df4a70126361a7d45281d11a7acf87bd-init
drwx--x---  3 root root 4096 Nov  6 10:55 fe0644962c78a51a058bf3b8d1390b57a5b13d58670dd1a3d3e279f31b8810ec
drwx------  2 root root 4096 Nov  7 11:43 l
ubuntu@ip-172-31-21-183:~$ CONTAINER_ID=$(docker ps -qf name=cgroup-test)
ubuntu@ip-172-31-21-183:~$ sudo docker inspect $CONTAINER_ID | grep -E "UpperDir|LowerDir|MergedDir"
                "LowerDir": "/var/lib/docker/overlay2/55107f990e9c54a652bce449d91946be37c86323364eece6d7e14cd39bf1199b-init/diff:/var
/lib/docker/overlay2/374eb41f6edaac70f9be8bff509711f1ef87390215262a534159f4811dec9a74/diff",
                "MergedDir": "/var/lib/docker/overlay2/55107f990e9c54a652bce449d91946be37c86323364eece6d7e14cd39bf1199b/merged",
                "UpperDir": "/var/lib/docker/overlay2/55107f990e9c54a652bce449d91946be37c86323364eece6d7e14cd39bf1199b/diff",
ubuntu@ip-172-31-21-183:~$ UPPER=$(sudo docker inspect $CONTAINER_ID -f '{{.GraphDriver.Data.UpperDir}}')
ubuntu@ip-172-31-21-183:~$ LOWER=$(sudo docker inspect $CONTAINER_ID -f '{{.GraphDriver.Data.LowerDir}}')
ubuntu@ip-172-31-21-183:~$ MERGED=$(sudo docker inspect $CONTAINER_ID -f '{{.GraphDriver.Data.MergedDir}}')
ubuntu@ip-172-31-21-183:~$ echo "Upper (writable): $UPPER"
Upper (writable): /var/lib/docker/overlay2/55107f990e9c54a652bce449d91946be37c86323364eece6d7e14cd39bf1199b/diff
ubuntu@ip-172-31-21-183:~$ echo "Lower (read-only): $LOWER"
Lower (read-only): /var/lib/docker/overlay2/55107f990e9c54a652bce449d91946be37c86323364eece6d7e14cd39bf1199b-init/diff:/var/lib/docke
r/overlay2/374eb41f6edaac70f9be8bff509711f1ef87390215262a534159f4811dec9a74/diff
ubuntu@ip-172-31-21-183:~$ echo "Merged (union view): $MERGED"
Merged (union view): /var/lib/docker/overlay2/55107f990e9c54a652bce449d91946be37c86323364eece6d7e14cd39bf1199b/merged
ubuntu@ip-172-31-21-183:~$ sudo ls -la $UPPER
```



```
ubuntu@ip-172-31-21-183:~$ sudo ls -la $UPPER
total 8
drwxr-xr-x 2 root root 4096 Nov  7 11:43 .
drwx--x--- 5 root root 4096 Nov  7 11:43 ..
ubuntu@ip-172-31-21-183:~$ docker exec cgroup-test bash -c "echo 'persistent data' > /testfile.txt"
ubuntu@ip-172-31-21-183:~$ sudo cat $UPPER/testfile.txt
persistent data
ubuntu@ip-172-31-21-183:~$ docker stop cgroup-test
cgroup-test
ubuntu@ip-172-31-21-183:~$ sudo ls $UPPER
testfile.txt
ubuntu@ip-172-31-21-183:~$ sudo cat $UPPER/testfile.txt
persistent data
ubuntu@ip-172-31-21-183:~$ docker start cgroup-test
cgroup-test
ubuntu@ip-172-31-21-183:~$ docker exec cgroup-test cat /testfile.txt
persistent data
```

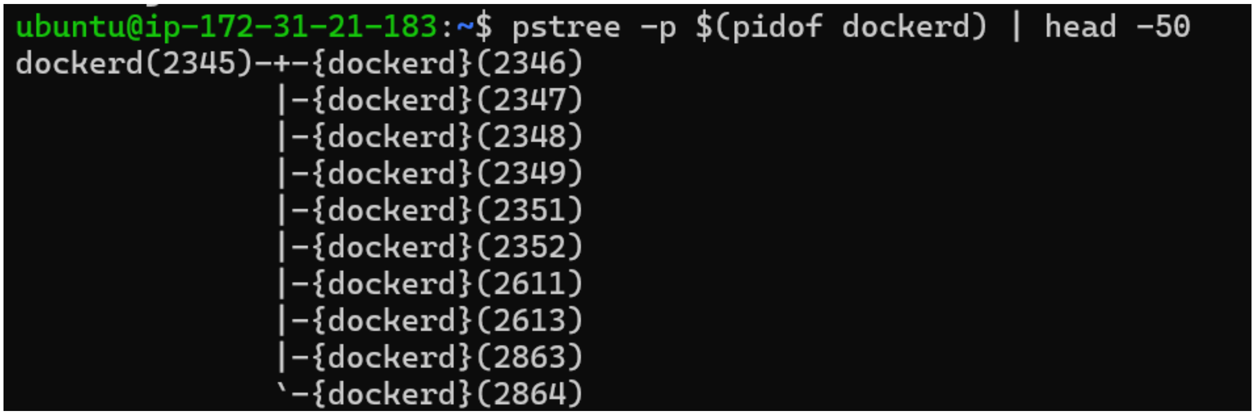


```
ubuntu@ip-172-31-21-183:~$ docker stop cgroup-test
cgroup-test
ubuntu@ip-172-31-21-183:~$ sudo cat $UPPER/testfile.txt
persistent data
ubuntu@ip-172-31-21-183:~$ docker rm cgroup-test
cgroup-test
ubuntu@ip-172-31-21-183:~$ sudo cat $UPPER/testfile.txt
cat: /var/lib/docker/overlay2/55107f990e9c54a652bce449d91946be37c86323364eece6d7e14cd39bf1199b/diff/testfile.txt: No such file or dir
ectory
```

8. Process Creation Flow Examination

# Show complete Docker process hierarchy

pstree -p $(pidof dockerd) | head -50

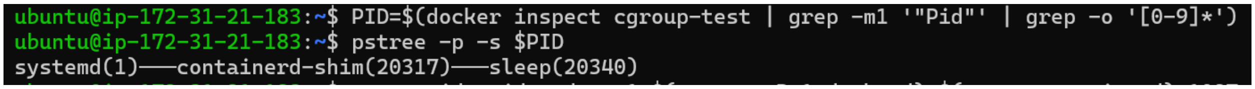


```
ubuntu@ip-172-31-21-183:~$ pstree -p $(pidof dockerd) | head -50
dockerd(2345)-+-{dockerd}(2346)
              |-{dockerd}(2347)
              |-{dockerd}(2348)
              |-{dockerd}(2349)
              |-{dockerd}(2351)
              |-{dockerd}(2352)
              |-{dockerd}(2611)
              |-{dockerd}(2613)
              |-{dockerd}(2863)
              `-{dockerd}(2864)
```

# Trace specific container

PID=$(docker inspect cgroup-test | grep -m1 '"Pid"' | grep -o '[0-9]*')

pstree -p -s $PID



```
ubuntu@ip-172-31-21-183:~$ PID=$(docker inspect cgroup-test | grep -m1 '"Pid"' | grep -o '[0-9]*')
ubuntu@ip-172-31-21-183:~$ pstree -p -s $PID
systemd(1)──containerd-shim(20317)──sleep(20340)
```

1. systemd (PID 1): Ubuntu's init system
2. dockerd: Docker daemon (manages containers)
3. containerd: Container runtime (manages container lifecycle)
4. containerd-shim: Shim process (keeps STDIO open, reaps zombies)
5. runc init: Actually creates namespaces/cgroups (short-lived)
6. Container process : Your bash/application (PID 1 in container)

9. Comparative Namespace Experiment

# Run a fully isolated container

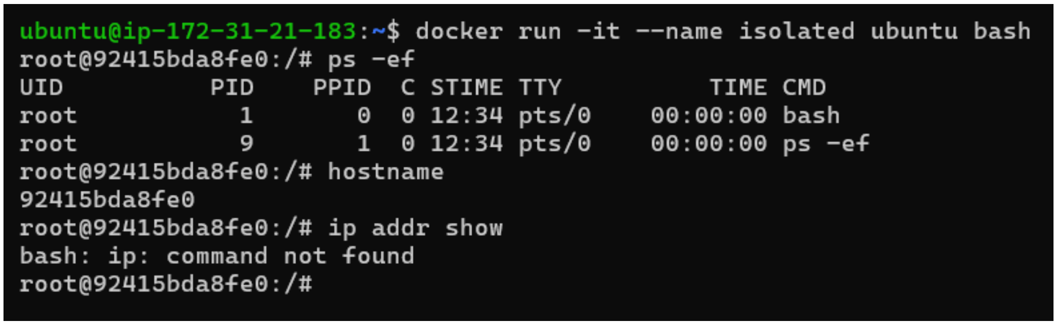
docker run -it --name isolated ubuntu bash

# Inside the container:

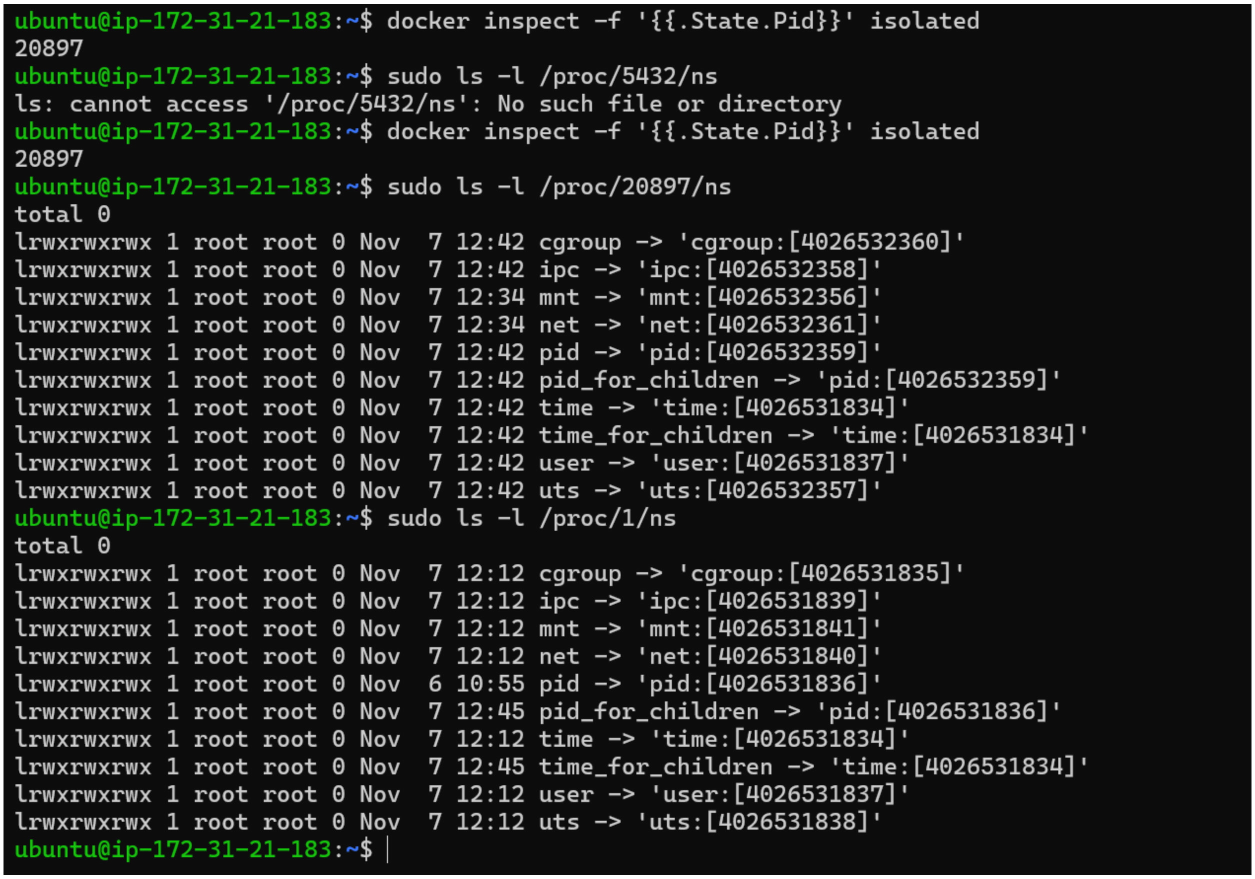
ps -ef

hostname

ip addr show



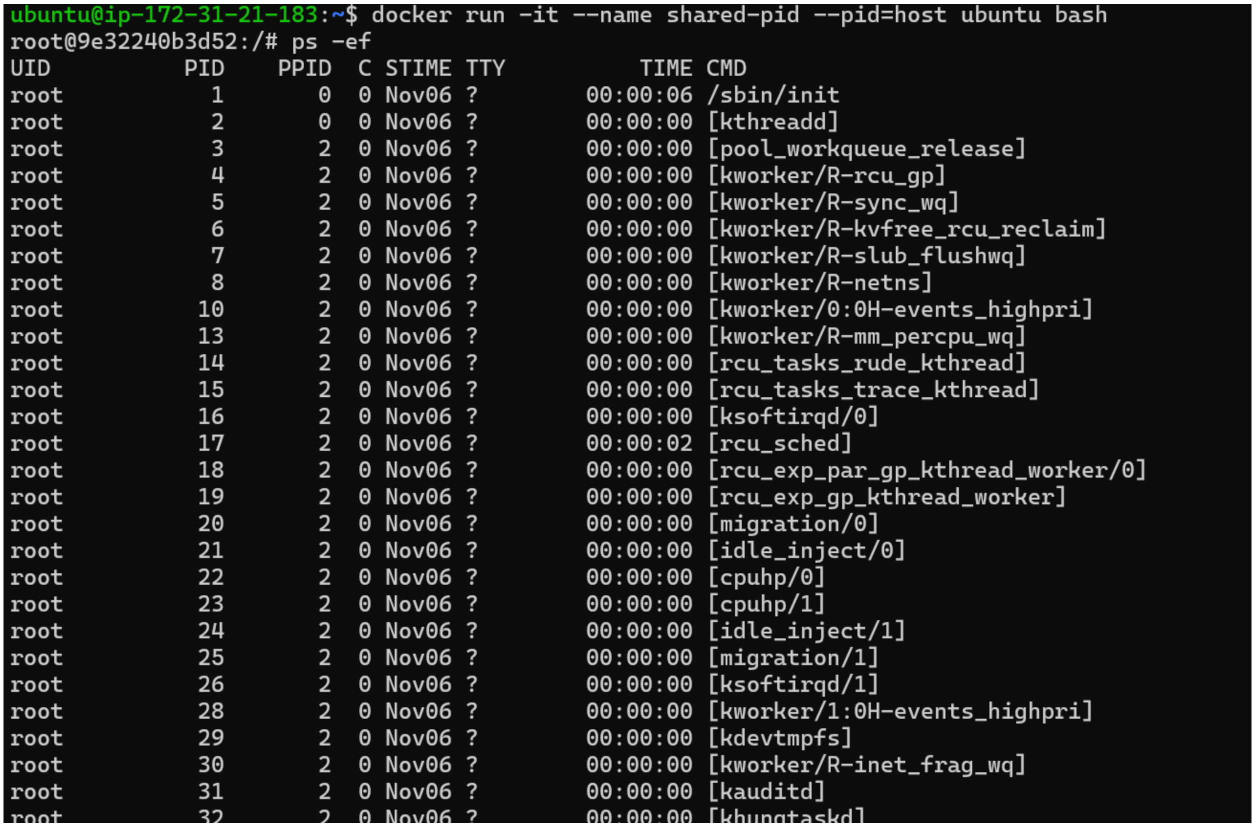
```
ubuntu@ip-172-31-21-183:~$ docker run -it --name isolated ubuntu bash
root@92415bda8fe0:/# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 12:34 pts/0    00:00:00 bash
root           9       1  0 12:34 pts/0    00:00:00 ps -ef
root@92415bda8fe0:/# hostname
92415bda8fe0
root@92415bda8fe0:/# ip addr show
bash: ip: command not found
root@92415bda8fe0:/#
```



```
ubuntu@ip-172-31-21-183:~$ docker inspect -f '{{.State.Pid}}' isolated
20897
ubuntu@ip-172-31-21-183:~$ sudo ls -l /proc/5432/ns
ls: cannot access '/proc/5432/ns': No such file or directory
ubuntu@ip-172-31-21-183:~$ docker inspect -f '{{.State.Pid}}' isolated
20897
ubuntu@ip-172-31-21-183:~$ sudo ls -l /proc/20897/ns
total 0
lrwxrwxrwx 1 root root 0 Nov  7 12:42 cgroup -> 'cgroup:[4026532360]'
lrwxrwxrwx 1 root root 0 Nov  7 12:42 ipc -> 'ipc:[4026532358]'
lrwxrwxrwx 1 root root 0 Nov  7 12:34 mnt -> 'mnt:[4026532356]'
lrwxrwxrwx 1 root root 0 Nov  7 12:34 net -> 'net:[4026532361]'
lrwxrwxrwx 1 root root 0 Nov  7 12:42 pid -> 'pid:[4026532359]'
lrwxrwxrwx 1 root root 0 Nov  7 12:42 pid_for_children -> 'pid:[4026532359]'
lrwxrwxrwx 1 root root 0 Nov  7 12:42 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 12:42 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 12:42 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Nov  7 12:42 uts -> 'uts:[4026532357]'
ubuntu@ip-172-31-21-183:~$ sudo ls -l /proc/1/ns
total 0
lrwxrwxrwx 1 root root 0 Nov  7 12:12 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Nov  7 12:12 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 Nov  7 12:12 mnt -> 'mnt:[4026531841]'
lrwxrwxrwx 1 root root 0 Nov  7 12:12 net -> 'net:[4026531840]'
lrwxrwxrwx 1 root root 0 Nov  6 10:55 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Nov  7 12:45 pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Nov  7 12:12 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 12:45 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 12:12 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Nov  7 12:12 uts -> 'uts:[4026531838]'
ubuntu@ip-172-31-21-183:~$
```

We can see different inode numbers for PID, NET, MNT, etc. compared to host processes — proving the container is isolated.

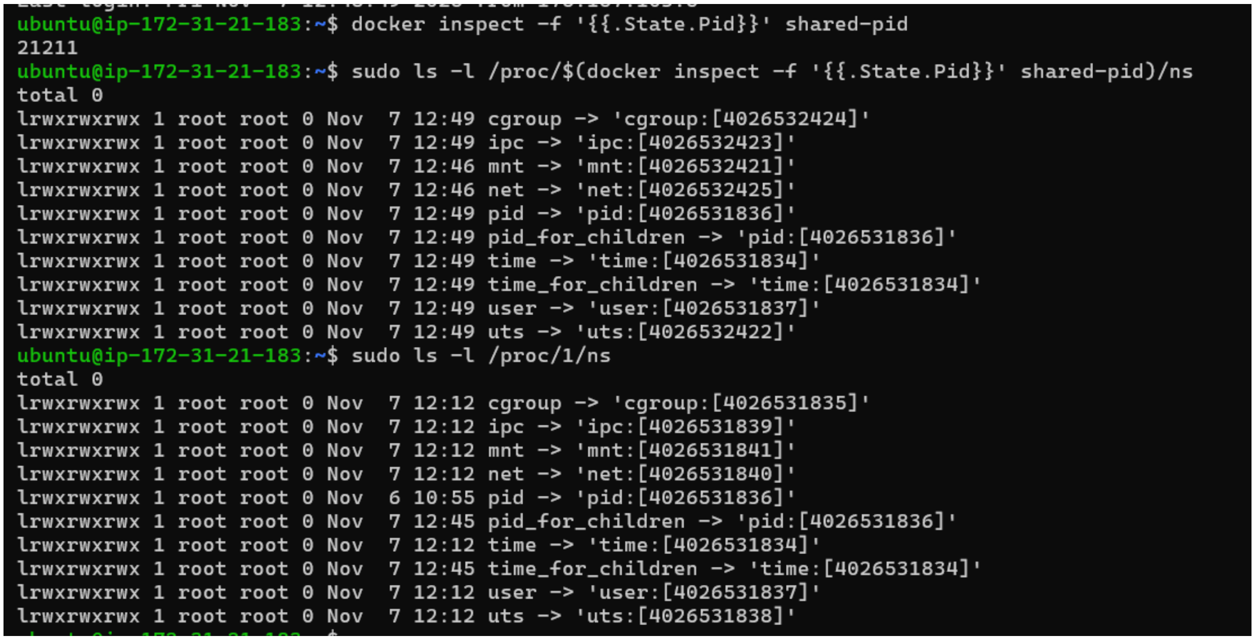
Run a Container Sharing the Host PID Namespace:



```
ubuntu@ip-172-31-21-183:~$ docker run -it --name shared-pid --pid=host ubuntu bash
root@9e32240b3d52:/# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 Nov06 ?        00:00:06 /sbin/init
root           2       0  0 Nov06 ?        00:00:00 [kthreadd]
root           3       2  0 Nov06 ?        00:00:00 [pool_workqueue_release]
root           4       2  0 Nov06 ?        00:00:00 [kworker/R-rcu_gp]
root           5       2  0 Nov06 ?        00:00:00 [kworker/R-sync_wq]
root           6       2  0 Nov06 ?        00:00:00 [kworker/R-kvfree_rcu_reclaim]
root           7       2  0 Nov06 ?        00:00:00 [kworker/R-slub_flushwq]
root           8       2  0 Nov06 ?        00:00:00 [kworker/R-netns]
root          10       2  0 Nov06 ?        00:00:00 [kworker/0:0H-events_highpri]
root          13       2  0 Nov06 ?        00:00:00 [kworker/R-mm_percpu_wq]
root          14       2  0 Nov06 ?        00:00:00 [rcu_tasks_rude_kthread]
root          15       2  0 Nov06 ?        00:00:00 [rcu_tasks_trace_kthread]
root          16       2  0 Nov06 ?        00:00:00 [ksoftirqd/0]
root          17       2  0 Nov06 ?        00:00:02 [rcu_sched]
root          18       2  0 Nov06 ?        00:00:00 [rcu_exp_par_gp_kthread_worker/0]
root          19       2  0 Nov06 ?        00:00:00 [rcu_exp_gp_kthread_worker]
root          20       2  0 Nov06 ?        00:00:00 [migration/0]
root          21       2  0 Nov06 ?        00:00:00 [idle_inject/0]
root          22       2  0 Nov06 ?        00:00:00 [cpuhp/0]
root          23       2  0 Nov06 ?        00:00:00 [cpuhp/1]
root          24       2  0 Nov06 ?        00:00:00 [idle_inject/1]
root          25       2  0 Nov06 ?        00:00:00 [migration/1]
root          26       2  0 Nov06 ?        00:00:00 [ksoftirqd/1]
root          28       2  0 Nov06 ?        00:00:00 [kworker/1:0H-events_highpri]
root          29       2  0 Nov06 ?        00:00:00 [kdevtmpfs]
root          30       2  0 Nov06 ?        00:00:00 [kworker/R-inet_frag_wq]
root          31       2  0 Nov06 ?        00:00:00 [kauditd]
root          32       2  0 Nov06 ?        00:00:00 [khungtaskd]
```

We can now see all host processes, not just the container ones because the container is using the host PID namespace.

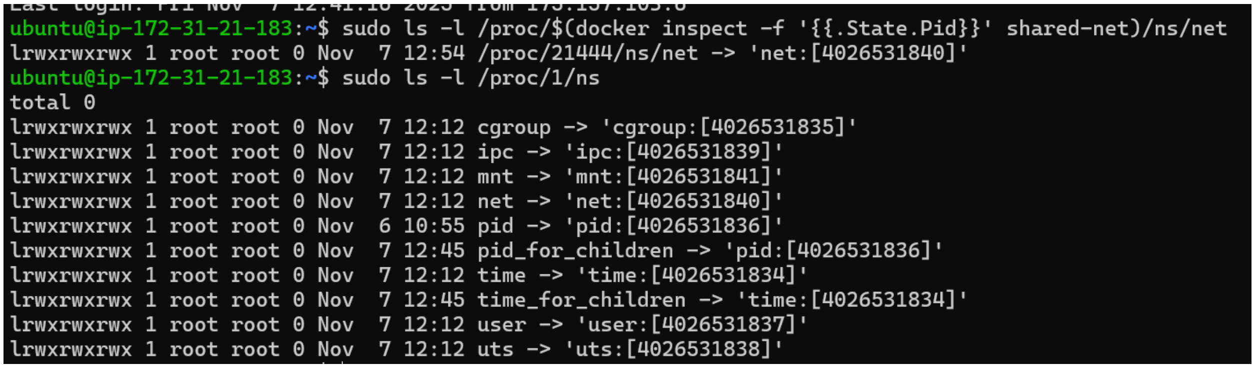
On the host:



```
ubuntu@ip-172-31-21-183:~$ docker inspect -f '{{.State.Pid}}' shared-pid
21211
ubuntu@ip-172-31-21-183:~$ sudo ls -l /proc/$(docker inspect -f '{{.State.Pid}}' shared-pid)/ns
total 0
lrwxrwxrwx 1 root root 0 Nov  7 12:49 cgroup -> 'cgroup:[4026532424]'
lrwxrwxrwx 1 root root 0 Nov  7 12:49 ipc -> 'ipc:[4026532423]'
lrwxrwxrwx 1 root root 0 Nov  7 12:46 mnt -> 'mnt:[4026532421]'
lrwxrwxrwx 1 root root 0 Nov  7 12:46 net -> 'net:[4026532425]'
lrwxrwxrwx 1 root root 0 Nov  7 12:49 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Nov  7 12:49 pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Nov  7 12:49 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 12:49 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 12:49 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Nov  7 12:49 uts -> 'uts:[4026532422]'
ubuntu@ip-172-31-21-183:~$ sudo ls -l /proc/1/ns
total 0
lrwxrwxrwx 1 root root 0 Nov  7 12:12 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Nov  7 12:12 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 Nov  7 12:12 mnt -> 'mnt:[4026531841]'
lrwxrwxrwx 1 root root 0 Nov  7 12:12 net -> 'net:[4026531840]'
lrwxrwxrwx 1 root root 0 Nov  6 10:55 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Nov  7 12:45 pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Nov  7 12:12 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 12:45 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 12:12 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Nov  7 12:12 uts -> 'uts:[4026531838]'
```

Compare namespace inode numbers with your host’s /proc/1/ns/pid.
We will see they are identical meaning the container shares the host’s PID namespace.

Run a Container Sharing the Host Network Namespace:



```
Last login: Fri Nov  7 12:41:18 2025 from 173.137.163.8
ubuntu@ip-172-31-21-183:~$ sudo ls -l /proc/$(docker inspect -f '{{.State.Pid}}' shared-net)/ns/net
lrwxrwxrwx 1 root root 0 Nov  7 12:54 /proc/21444/ns/net -> 'net:[4026531840]'
ubuntu@ip-172-31-21-183:~$ sudo ls -l /proc/1/ns
total 0
lrwxrwxrwx 1 root root 0 Nov  7 12:12 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Nov  7 12:12 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 Nov  7 12:12 mnt -> 'mnt:[4026531841]'
lrwxrwxrwx 1 root root 0 Nov  7 12:12 net -> 'net:[4026531840]'
lrwxrwxrwx 1 root root 0 Nov  6 10:55 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Nov  7 12:45 pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Nov  7 12:12 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 12:45 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 12:12 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Nov  7 12:12 uts -> 'uts:[4026531838]'
```

You can see here both host and shared-net using same net ID

Understanding setns():

When Docker runs a container with --pid=host or --network=host,
instead of creating new namespaces, it uses the setns() system call.

- clone(): creates a new namespace
- setns(): joins an existing namespace