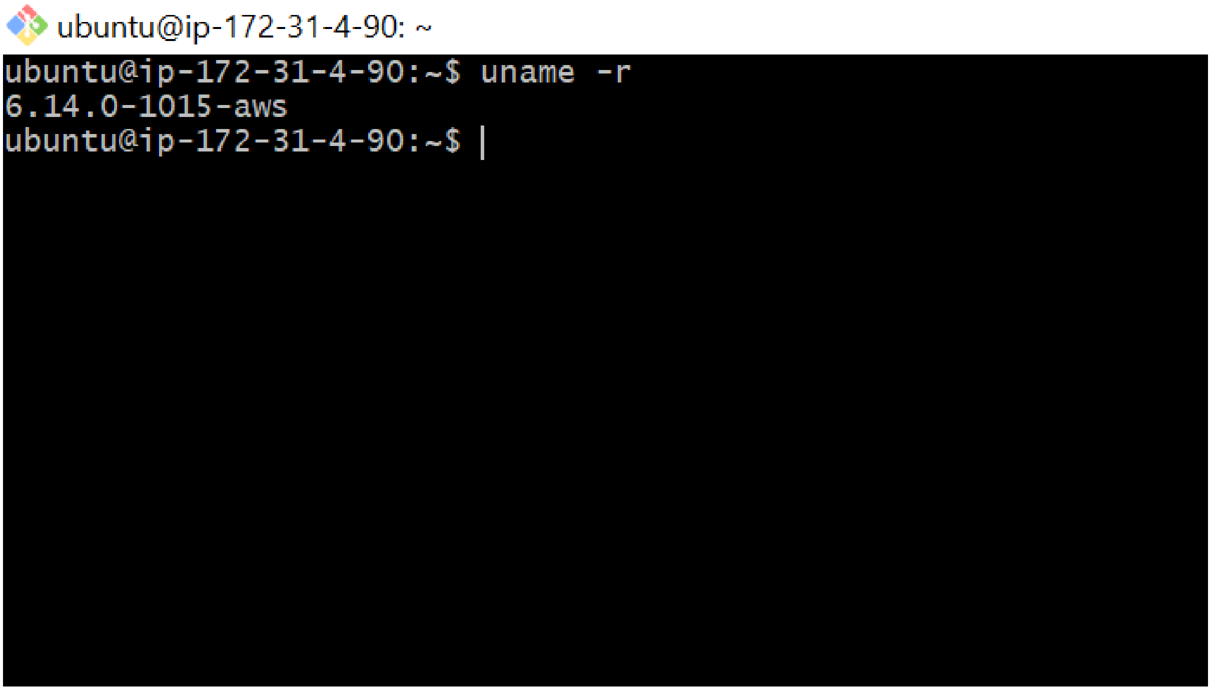
Docker Assignment

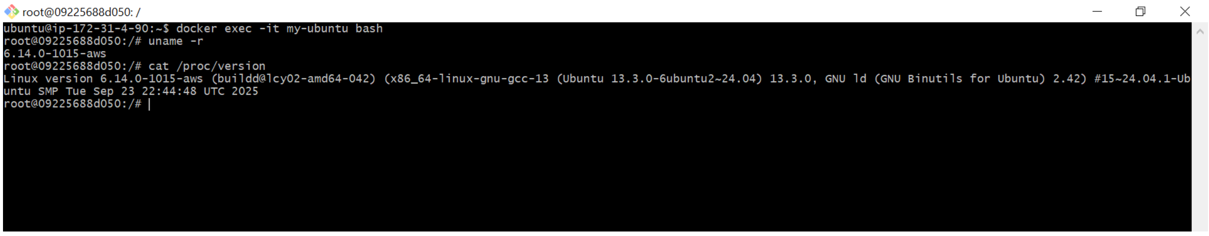
1. Verifying Kernel Sharing

Host Kernel Version: 6.14.0-1015-aws



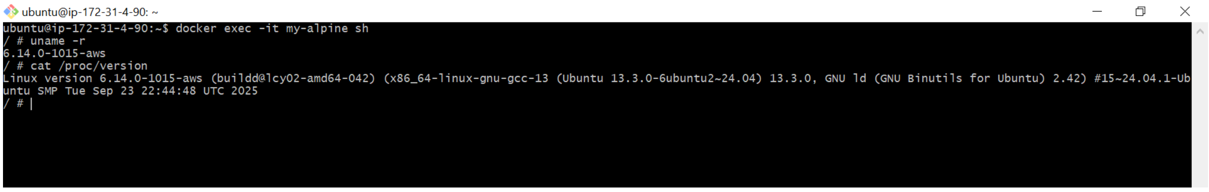
```
ubuntu@ip-172-31-4-90:~$ uname -r
6.14.0-1015-aws
ubuntu@ip-172-31-4-90:~$ |
```

Ubuntu kernel version



```
ubuntu@ip-172-31-4-90:~$ docker exec -it my-ubuntu bash
root@09225688d050:/# uname -r
6.14.0-1015-aws
root@09225688d050:/# 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-Ub
untu SMP Tue Sep 23 22:44:48 UTC 2025
root@09225688d050:/# |
```

Alpine kernel version



```
ubuntu@ip-172-31-4-90:~$ docker exec -it my-alpine sh
/ # 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-Ub
untu SMP Tue Sep 23 22:44:48 UTC 2025
/ # |
```

Observation

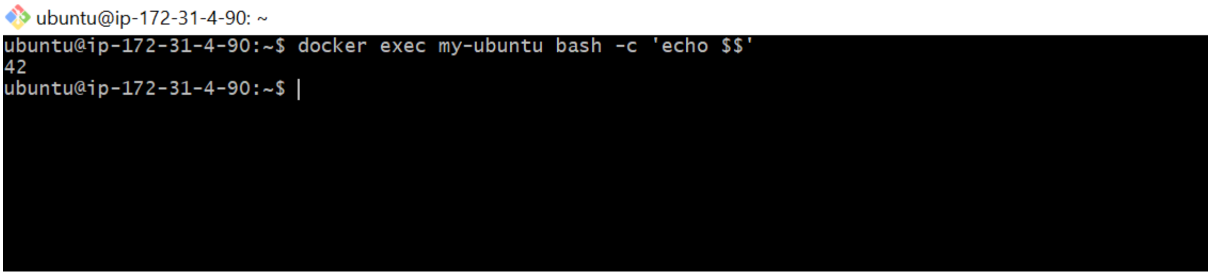
All three environments (host, Ubuntu container, and Alpine container) show identical kernel versions. This proves that:

1. Containers are not work like virtual machines - They don't have their own kernel
2. Kernel is shared - All containers use the host's kernel
3. Isolation is at process level - Containers isolate processes, not the entire OS

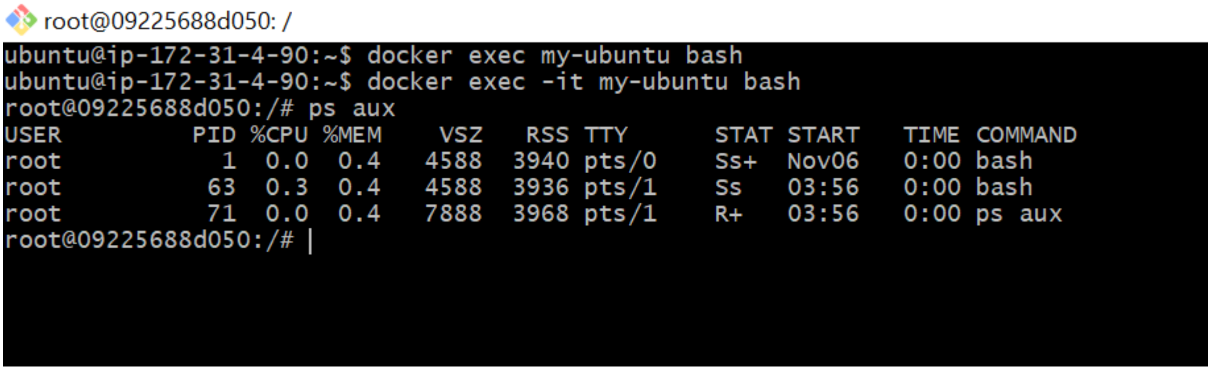
Learnings

- Here I learned that all containers share the same kernel as the host system. When I checked the kernel version inside different containers like `ubuntu` and `alpine`, it was exactly the same as the host's. This proved that containers do not have their own operating system like virtual machines. Instead, they use the host's kernel, which makes them much faster and more lightweight.

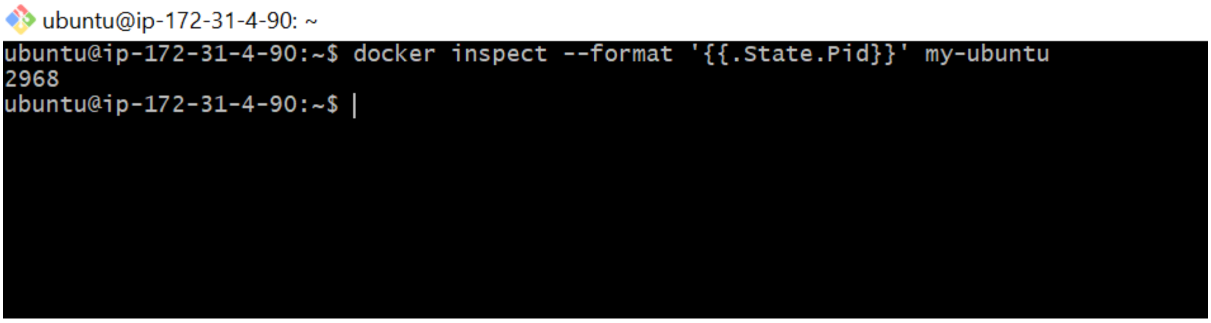
2. Process and PID Mapping



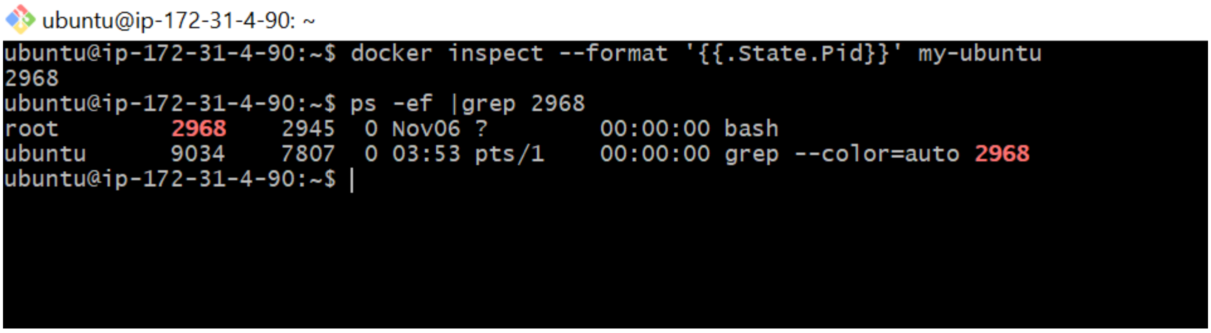
```
ubuntu@ip-172-31-4-90:~$ docker exec my-ubuntu bash -c 'echo $$'
42
ubuntu@ip-172-31-4-90:~$ |
```



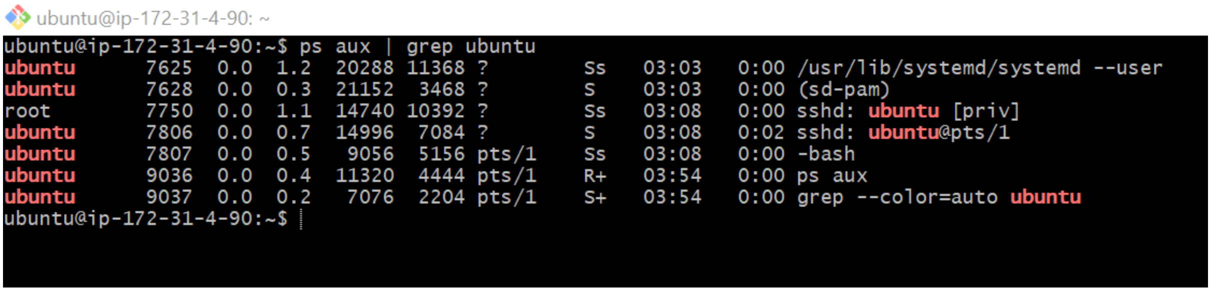
```
ubuntu@ip-172-31-4-90:~$ docker exec my-ubuntu bash
ubuntu@ip-172-31-4-90:~$ docker exec -it my-ubuntu bash
root@09225688d050:/# ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.4   4588  3940 pts/0    Ss+  Nov06   0:00 bash
root          63  0.3  0.4   4588  3936 pts/1    Ss   03:56   0:00 bash
root          71  0.0  0.4   7888  3968 pts/1    R+   03:56   0:00 ps aux
root@09225688d050:/# |
```



```
ubuntu@ip-172-31-4-90:~$ docker inspect --format '{{.State.Pid}}' my-ubuntu
2968
ubuntu@ip-172-31-4-90:~$ |
```



```
ubuntu@ip-172-31-4-90:~$ docker inspect --format '{{.State.Pid}}' my-ubuntu
2968
ubuntu@ip-172-31-4-90:~$ ps -ef |grep 2968
root        2968    2945  0 Nov06 ?        00:00:00 bash
ubuntu      9034    7807  0 03:53 pts/1    00:00:00 grep --color=auto 2968
ubuntu@ip-172-31-4-90:~$ |
```



```
ubuntu@ip-172-31-4-90:~$ ps aux | grep ubuntu
ubuntu      7625  0.0  1.2  20288 11368 ?        Ss   03:03   0:00 /usr/lib/systemd/systemd --user
ubuntu      7628  0.0  0.3  21152  3468 ?        S    03:03   0:00 (sd-pam)
root        7750  0.0  1.1  14740 10392 ?        Ss   03:08   0:00 sshd: ubuntu [priv]
ubuntu      7806  0.0  0.7  14996  7084 ?        S    03:08   0:02 sshd: ubuntu@pts/1
ubuntu      7807  0.0  0.5   9056  5156 pts/1    Ss   03:08   0:00 -bash
ubuntu      9036  0.0  0.4  11320  4444 pts/1    R+   03:54   0:00 ps aux
ubuntu      9037  0.0  0.2   7076  2204 pts/1    S+   03:54   0:00 grep --color=auto ubuntu
ubuntu@ip-172-31-4-90:~$
```

Observations

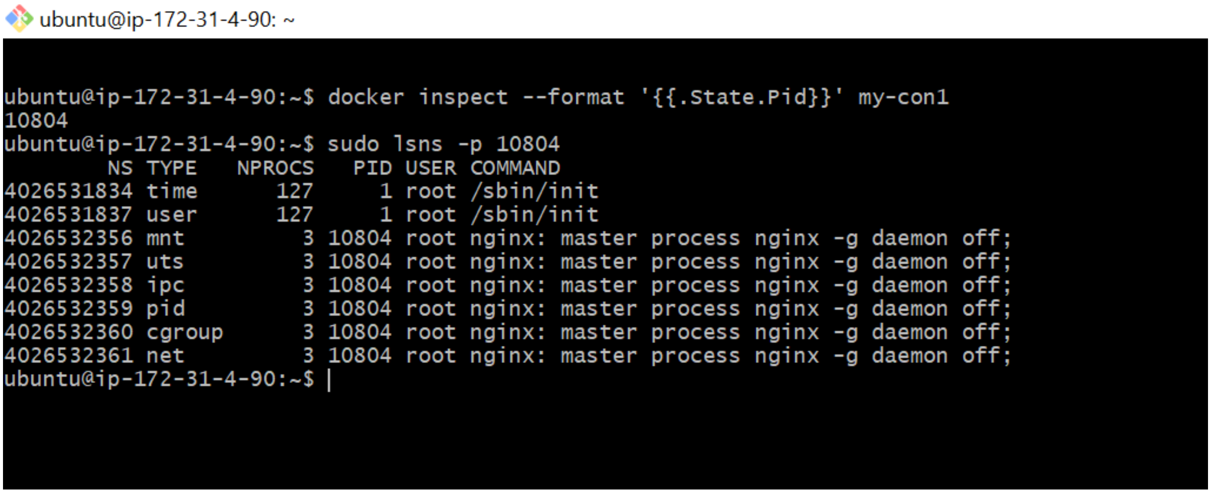
- After Ubuntu container named as my-ubuntu used the command echo $$ inside it, which returned PID 42 — this shows the process ID inside the container namespace.
- Then I hit with docker inspect --format '{{.State.Pid}}' my-ubuntu, which showed 2968 as the corresponding host PID.
- This means that inside the container, process PID 1 (bash) maps to PID 2968 on the host.
- Running ps aux inside the container listed the processes, where PID 1 was bash, confirming that it acts as the init process inside the container.
- On the host, checking with ps -ef | grep 2968 confirmed that bash (PID 2968) exists as a normal Linux process under the parent dockerd/containerd process chain.
- This proves that container processes are actually regular host processes, just running inside isolated namespaces.

Learnings

- Learned that a container's processes are actually normal processes running on the host system. By checking the container's PID and matching it with the host PID, I could see how Docker uses namespaces to isolate processes while still sharing the same kernel.

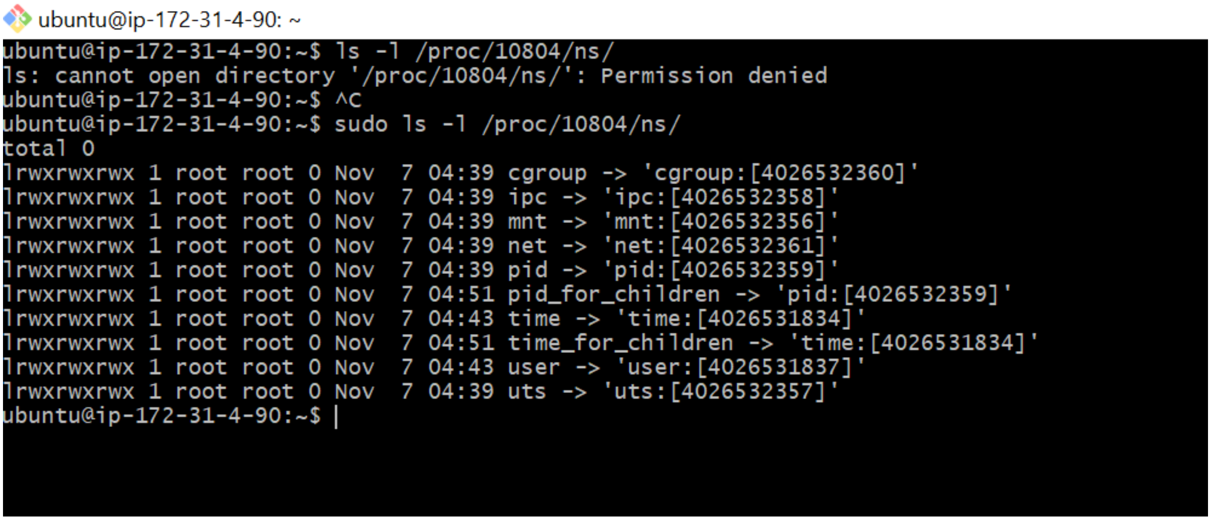
3. Exploring Namespace Isolation

Container1-lsns

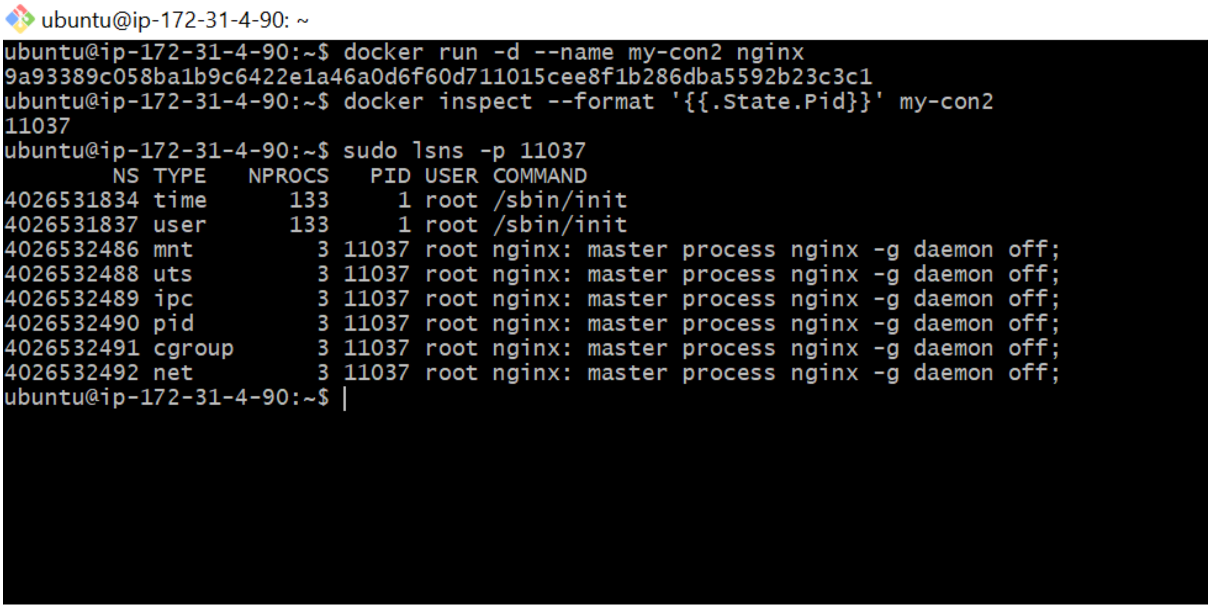


```
ubuntu@ip-172-31-4-90:~$ docker inspect --format '{{.State.Pid}}' my-con1
10804
ubuntu@ip-172-31-4-90:~$ sudo lsns -p 10804
        NS TYPE   NPROCS   PID USER COMMAND
4026531834 time      127     1 root /sbin/init
4026531837 user      127     1 root /sbin/init
4026532356 mnt         3 10804 root nginx: master process nginx -g daemon off;
4026532357 uts         3 10804 root nginx: master process nginx -g daemon off;
4026532358 ipc         3 10804 root nginx: master process nginx -g daemon off;
4026532359 pid         3 10804 root nginx: master process nginx -g daemon off;
4026532360 cgroup      3 10804 root nginx: master process nginx -g daemon off;
4026532361 net         3 10804 root nginx: master process nginx -g daemon off;
ubuntu@ip-172-31-4-90:~$ |
```

Container 1 Namespace Inodes:

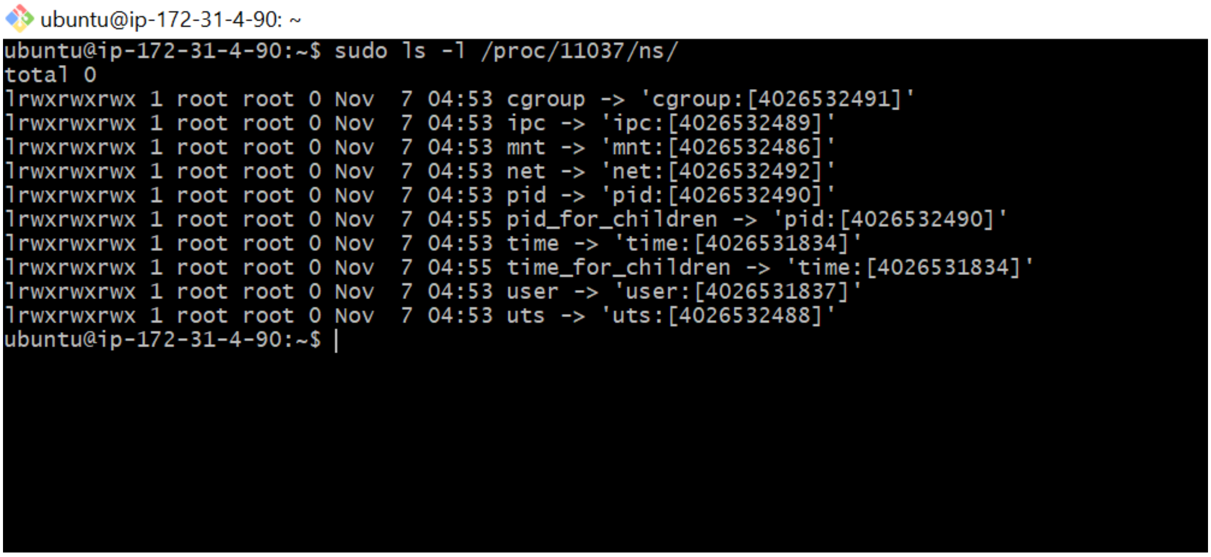


```
ubuntu@ip-172-31-4-90:~$ ls -l /proc/10804/ns/
ls: cannot open directory '/proc/10804/ns/': Permission denied
ubuntu@ip-172-31-4-90:~$ ^C
ubuntu@ip-172-31-4-90:~$ sudo ls -l /proc/10804/ns/
total 0
lrwxrwxrwx 1 root root 0 Nov  7 04:39 cgroup -> 'cgroup:[4026532360]'
lrwxrwxrwx 1 root root 0 Nov  7 04:39 ipc -> 'ipc:[4026532358]'
lrwxrwxrwx 1 root root 0 Nov  7 04:39 mnt -> 'mnt:[4026532356]'
lrwxrwxrwx 1 root root 0 Nov  7 04:39 net -> 'net:[4026532361]'
lrwxrwxrwx 1 root root 0 Nov  7 04:39 pid -> 'pid:[4026532359]'
lrwxrwxrwx 1 root root 0 Nov  7 04:51 pid_for_children -> 'pid:[4026532359]'
lrwxrwxrwx 1 root root 0 Nov  7 04:43 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 04:51 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 04:43 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Nov  7 04:39 uts -> 'uts:[4026532357]'
ubuntu@ip-172-31-4-90:~$ |
```



```
ubuntu@ip-172-31-4-90:~$ docker run -d --name my-con2 nginx
9a93389c058ba1b9c6422e1a46a0d6f60d711015cee8f1b286dba5592b23c3c1
ubuntu@ip-172-31-4-90:~$ docker inspect --format '{{.State.Pid}}' my-con2
11037
ubuntu@ip-172-31-4-90:~$ sudo lsns -p 11037
        NS TYPE   NPROCS   PID USER COMMAND
4026531834 time      133     1 root /sbin/init
4026531837 user      133     1 root /sbin/init
4026532486 mnt         3 11037 root nginx: master process nginx -g daemon off;
4026532488 uts         3 11037 root nginx: master process nginx -g daemon off;
4026532489 ipc         3 11037 root nginx: master process nginx -g daemon off;
4026532490 pid         3 11037 root nginx: master process nginx -g daemon off;
4026532491 cgroup      3 11037 root nginx: master process nginx -g daemon off;
4026532492 net         3 11037 root nginx: master process nginx -g daemon off;
ubuntu@ip-172-31-4-90:~$ |
```

Container 2 Namespace Inodes:



```
ubuntu@ip-172-31-4-90:~$ sudo ls -l /proc/11037/ns/
total 0
lrwxrwxrwx 1 root root 0 Nov  7 04:53 cgroup -> 'cgroup:[4026532491]'
lrwxrwxrwx 1 root root 0 Nov  7 04:53 ipc -> 'ipc:[4026532489]'
lrwxrwxrwx 1 root root 0 Nov  7 04:53 mnt -> 'mnt:[4026532486]'
lrwxrwxrwx 1 root root 0 Nov  7 04:53 net -> 'net:[4026532492]'
lrwxrwxrwx 1 root root 0 Nov  7 04:53 pid -> 'pid:[4026532490]'
lrwxrwxrwx 1 root root 0 Nov  7 04:55 pid_for_children -> 'pid:[4026532490]'
lrwxrwxrwx 1 root root 0 Nov  7 04:53 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 04:55 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 04:53 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Nov  7 04:53 uts -> 'uts:[4026532488]'
ubuntu@ip-172-31-4-90:~$ |
```

Unique Namespaces (Per-Container Isolation)

Namespace Type	**Container 1 Inode**	**Container 2 Inode**	**Status**
PID	4026532359	4026532359	Unique
NET	4026532361	4026532361	Unique
MNT	4026532356	4026532486	Unique
UTS	4026532357	4026532488	Unique
IPC	4026532358	4026532489	Unique
CGROUP	4026532360	4026532491	Unique

Shared Namespaces (Host-Level Sharing)

Namespace Type	**Container 1 Inode**	**Container 2 Inode**	**Status**
TIME	4026531834	4026531834	Shared
USER	4026531837	4026531837	Shared

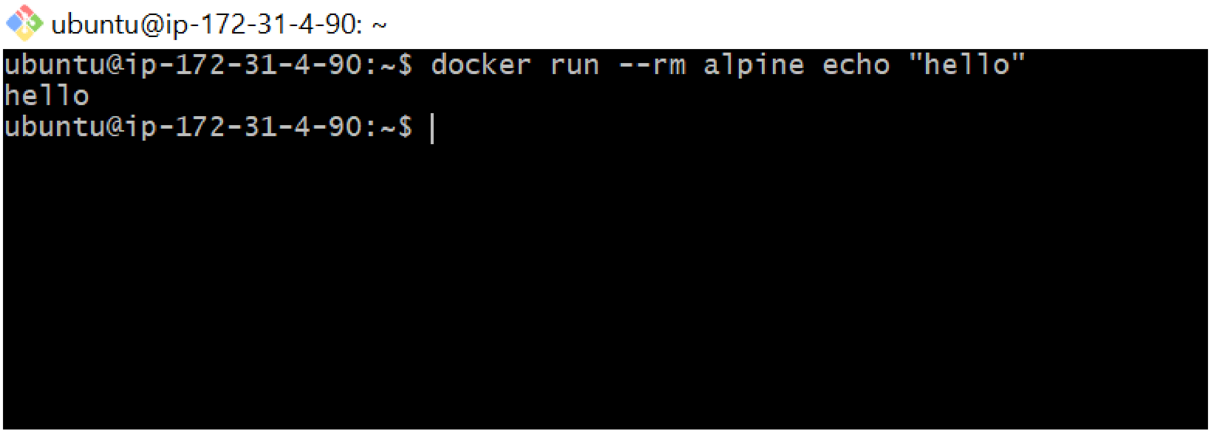
Observation

- When I hit-this “docker inspect --format '{{.State.Pid}}' my-con1” I got the 10804 as Pid of the container.And then listed its namespaces using “lsns -p 10804” and “ls -l /proc/10804/ns/”.
- So the namespace inodes for my-con1 were displayed and i did the same way for the my-con2 which was second container for that i got pid as 11037.
- Then I compared two containers namespace inodes ,PID,NET,MNT,UTS,IPC are unique for each container.

Learnings

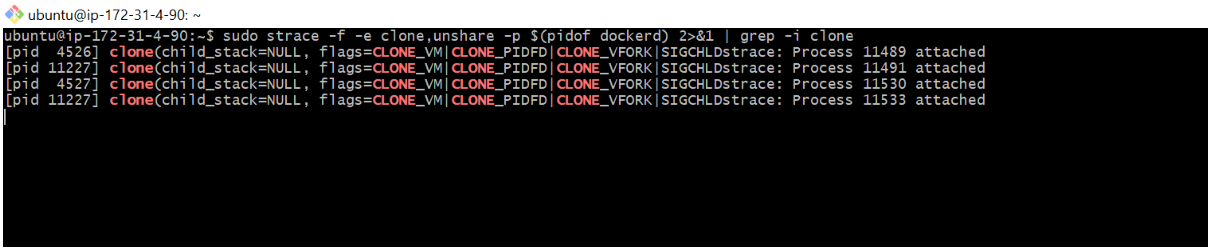
- I learned that Docker containers get isolation through Linux namespaces. Each container gets its own PID, network, mount, UTS, and IPC namespaces, so processes and resources inside one container are not visible to the other. However, some namespaces like USER and TIME can be shared with the host.

4. Observing New Namespace Creation



ubuntu@ip-172-31-4-90: ~

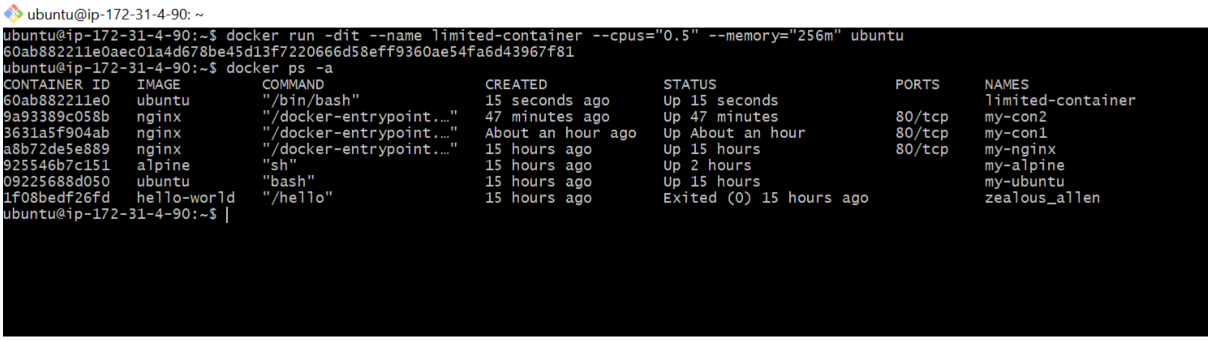
```
ubuntu@ip-172-31-4-90:~$ docker run --rm alpine echo "hello"
hello
ubuntu@ip-172-31-4-90:~$ |
```



ubuntu@ip-172-31-4-90: ~

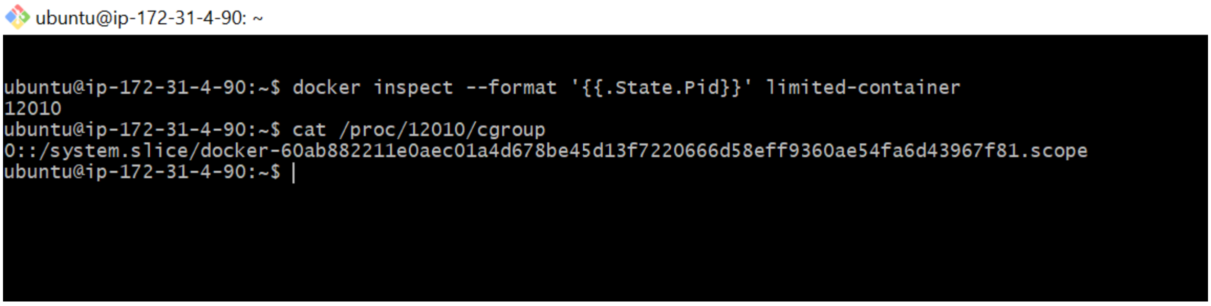
```
ubuntu@ip-172-31-4-90:~$ sudo strace -f -e clone,unshare -p $(pidof dockerd) 2>&1 | grep -i clone
[pid  4526] clone(child_stack=NULL, flags=CLONE_VM|CLONE_PIDFD|CLONE_VFORK|SIGCHLDstrace: Process 11489 attached
[pid 11227] clone(child_stack=NULL, flags=CLONE_VM|CLONE_PIDFD|CLONE_VFORK|SIGCHLDstrace: Process 11491 attached
[pid  4527] clone(child_stack=NULL, flags=CLONE_VM|CLONE_PIDFD|CLONE_VFORK|SIGCHLDstrace: Process 11530 attached
[pid 11227] clone(child_stack=NULL, flags=CLONE_VM|CLONE_PIDFD|CLONE_VFORK|SIGCHLDstrace: Process 11533 attached
```

5. Investigating cgroup Assignments



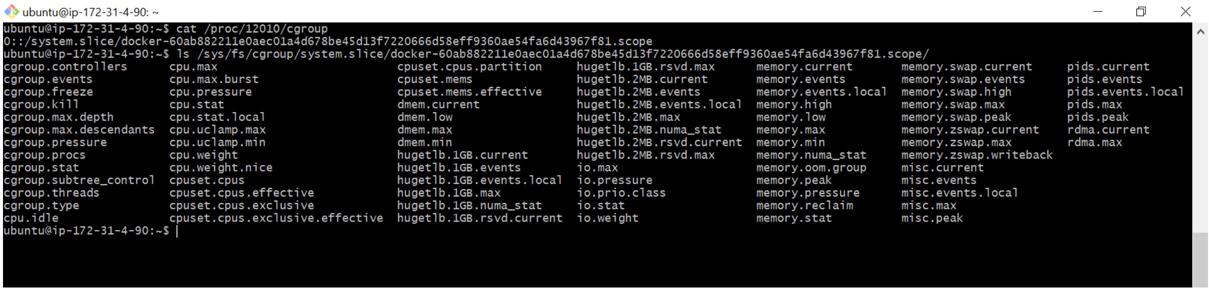
ubuntu@ip-172-31-4-90: ~

```
ubuntu@ip-172-31-4-90:~$ docker run -dit --name limited-container --cpus="0.5" --memory="256m" ubuntu
60ab882211e0aec01a4d678be45d13f7220666d58eff9360ae54fa6d43967f81
ubuntu@ip-172-31-4-90:~$ docker ps -a
CONTAINER ID   IMAGE         COMMAND                  CREATED             STATUS                     PORTS     NAMES
60ab882211e0   ubuntu        "/bin/bash"              15 seconds ago      Up 15 seconds                        limited-container
9a93389c058b   nginx         "/docker-entrypoint.…"   47 minutes ago      Up 47 minutes              80/tcp    my-con2
3631a5f904ab   nginx         "/docker-entrypoint.…"   About an hour ago   Up About an hour           80/tcp    my-con1
a8b72de5e889   nginx         "/docker-entrypoint.…"   15 hours ago        Up 15 hours                80/tcp    my-nginx
925546b7c151   alpine        "sh"                     15 hours ago        Up 2 hours                           my-alpine
09225688d050   ubuntu        "bash"                   15 hours ago        Up 15 hours                          my-ubuntu
1f08bedf26fd   hello-world   "/hello"                 15 hours ago        Exited (0) 15 hours ago              zealous_allen
ubuntu@ip-172-31-4-90:~$ |
```



ubuntu@ip-172-31-4-90: ~

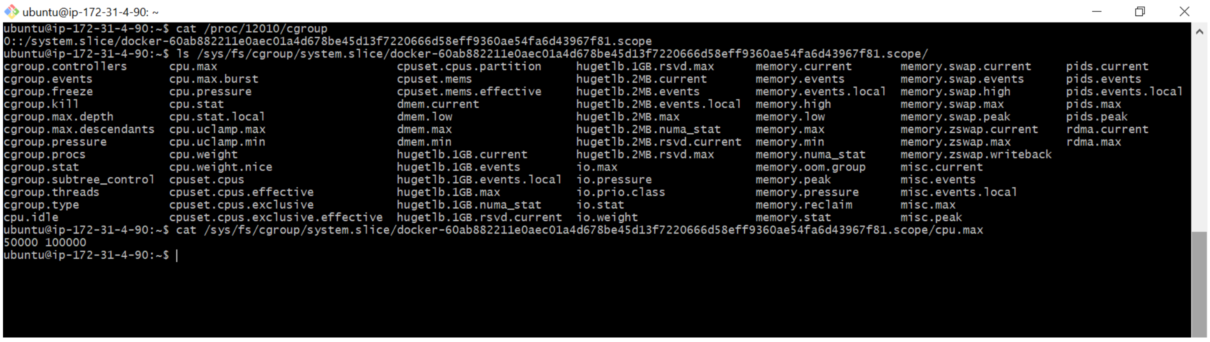
```
ubuntu@ip-172-31-4-90:~$ docker inspect --format '{{.State.Pid}}' limited-container
12010
ubuntu@ip-172-31-4-90:~$ cat /proc/12010/cgroup
0::/system.slice/docker-60ab882211e0aec01a4d678be45d13f7220666d58eff9360ae54fa6d43967f81.scope
ubuntu@ip-172-31-4-90:~$ |
```



ubuntu@ip-172-31-4-90: ~

```
ubuntu@ip-172-31-4-90:~$ cat /proc/12010/cgroup
0::/system.slice/docker-60ab882211e0aec01a4d678be45d13f7220666d58eff9360ae54fa6d43967f81.scope
ubuntu@ip-172-31-4-90:~$ ls /sys/fs/cgroup/system.slice/docker-60ab882211e0aec01a4d678be45d13f7220666d58eff9360ae54fa6d43967f81.scope/
cgroup.controllers      cpu.max                          cpuset.cpus.partition     hugetlb.1GB.rsvd.max     memory.current       memory.swap.current     pids.current
cgroup.events           cpu.max.burst                    cpuset.mems               hugetlb.2MB.current      memory.events        memory.swap.events      pids.events
cgroup.freeze           cpu.pressure                     cpuset.mems.effective     hugetlb.2MB.events       memory.events.local  memory.swap.high        pids.events.local
cgroup.kill             cpu.stat                         dmem.current              hugetlb.2MB.events.local memory.high          memory.swap.max         pids.max
cgroup.max.depth        cpu.stat.local                   dmem.low                  hugetlb.2MB.max          memory.low           memory.swap.peak        pids.peak
cgroup.max.descendants  cpu.uclamp.max                   dmem.max                  hugetlb.2MB.numa_stat    memory.max           memory.zswap.current    rdma.current
cgroup.pressure         cpu.uclamp.min                   dmem.min                  hugetlb.2MB.rsvd.current memory.min           memory.zswap.max        rdma.max
cgroup.procs            cpu.weight                       hugetlb.1GB.current       hugetlb.2MB.rsvd.max     memory.numa_stat     memory.zswap.writeback
cgroup.stat             cpu.weight.nice                  hugetlb.1GB.events        io.max                   memory.oom.group     misc.current
cgroup.subtree_control  cpuset.cpus                      hugetlb.1GB.events.local  io.pressure              memory.peak          misc.events
cgroup.threads          cpuset.cpus.effective            hugetlb.1GB.max           io.prio.class            memory.pressure      misc.events.local
cgroup.type             cpuset.cpus.exclusive            hugetlb.1GB.numa_stat     io.stat                  memory.reclaim       misc.max
cpu.idle                cpuset.cpus.exclusive.effective  hugetlb.1GB.rsvd.current  io.weight                memory.stat          misc.peak
ubuntu@ip-172-31-4-90:~$ |
```

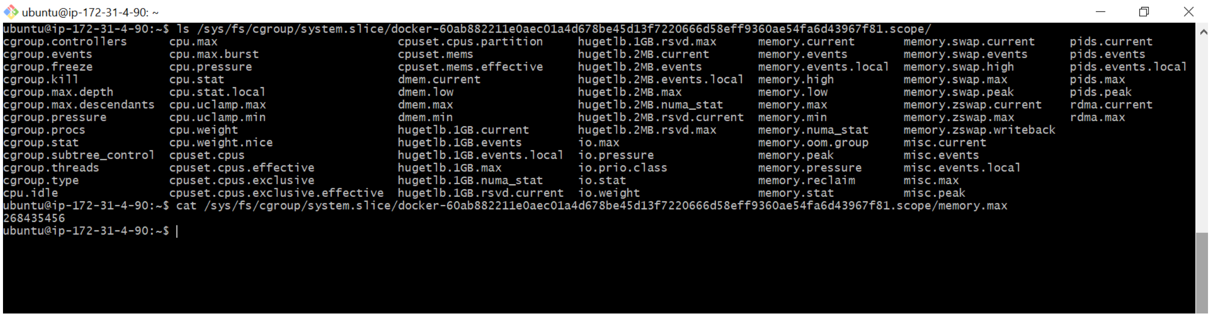
CPU limit



ubuntu@ip-172-31-4-90: ~

```
ubuntu@ip-172-31-4-90:~$ cat /proc/12010/cgroup
0::/system.slice/docker-60ab882211e0aec01a4d678be45d13f7220666d58eff9360ae54fa6d43967f81.scope
ubuntu@ip-172-31-4-90:~$ ls /sys/fs/cgroup/system.slice/docker-60ab882211e0aec01a4d678be45d13f7220666d58eff9360ae54fa6d43967f81.scope/
cgroup.controllers      cpu.max                          cpuset.cpus.partition     hugetlb.1GB.rsvd.max     memory.current       memory.swap.current     pids.current
cgroup.events           cpu.max.burst                    cpuset.mems               hugetlb.2MB.current      memory.events        memory.swap.events      pids.events
cgroup.freeze           cpu.pressure                     cpuset.mems.effective     hugetlb.2MB.events       memory.events.local  memory.swap.high        pids.events.local
cgroup.kill             cpu.stat                         dmem.current              hugetlb.2MB.events.local memory.high          memory.swap.max         pids.max
cgroup.max.depth        cpu.stat.local                   dmem.low                  hugetlb.2MB.max          memory.low           memory.swap.peak        pids.peak
cgroup.max.descendants  cpu.uclamp.max                   dmem.max                  hugetlb.2MB.numa_stat    memory.max           memory.zswap.current    rdma.current
cgroup.pressure         cpu.uclamp.min                   dmem.min                  hugetlb.2MB.rsvd.current memory.min           memory.zswap.max        rdma.max
cgroup.procs            cpu.weight                       hugetlb.1GB.current       hugetlb.2MB.rsvd.max     memory.numa_stat     memory.zswap.writeback
cgroup.stat             cpu.weight.nice                  hugetlb.1GB.events        io.max                   memory.oom.group     misc.current
cgroup.subtree_control  cpuset.cpus                      hugetlb.1GB.events.local  io.pressure              memory.peak          misc.events
cgroup.threads          cpuset.cpus.effective            hugetlb.1GB.max           io.prio.class            memory.pressure      misc.events.local
cgroup.type             cpuset.cpus.exclusive            hugetlb.1GB.numa_stat     io.stat                  memory.reclaim       misc.max
cpu.idle                cpuset.cpus.exclusive.effective  hugetlb.1GB.rsvd.current  io.weight                memory.stat          misc.peak
ubuntu@ip-172-31-4-90:~$ cat /sys/fs/cgroup/system.slice/docker-60ab882211e0aec01a4d678be45d13f7220666d58eff9360ae54fa6d43967f81.scope/cpu.max
50000 100000
ubuntu@ip-172-31-4-90:~$ |
```

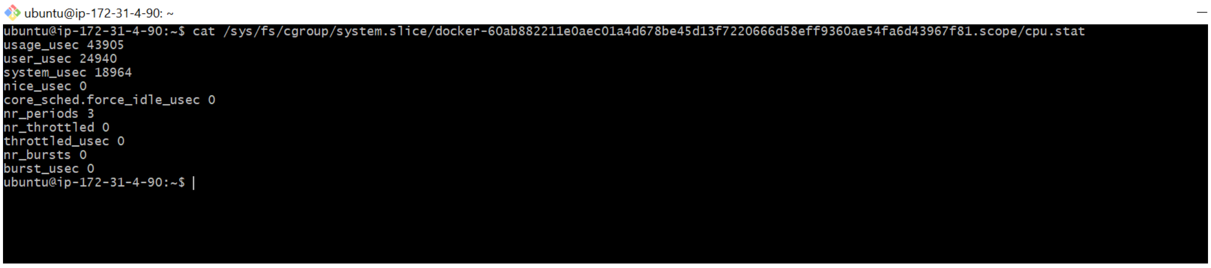
Memory limit



ubuntu@ip-172-31-4-90: ~

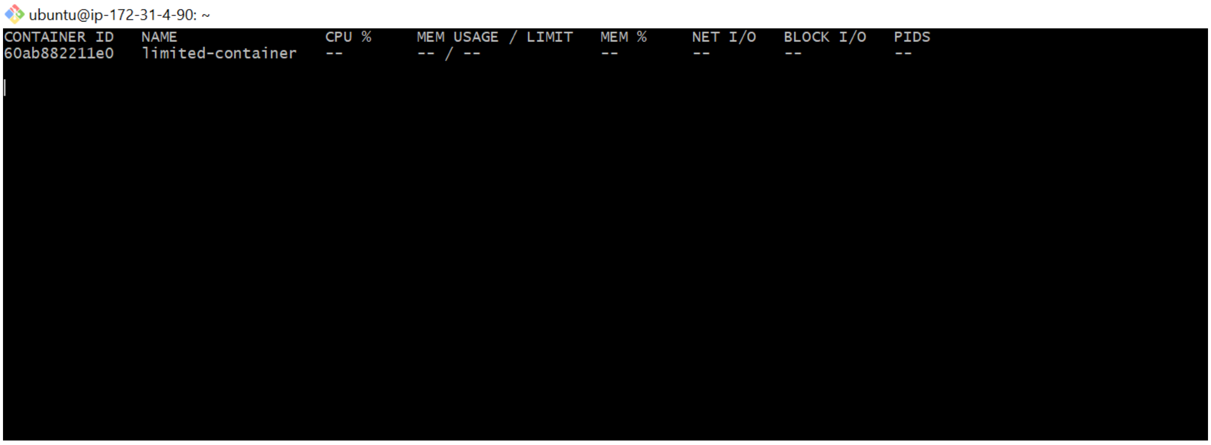
```
ubuntu@ip-172-31-4-90:~$ ls /sys/fs/cgroup/system.slice/docker-60ab882211e0aec01a4d678be45d13f7220666d58eff9360ae54fa6d43967f81.scope/
cgroup.controllers      cpu.max                          cpuset.cpus.partition     hugetlb.1GB.rsvd.max     memory.current       memory.swap.current     pids.current
cgroup.events           cpu.max.burst                    cpuset.mems               hugetlb.2MB.current      memory.events        memory.swap.events      pids.events
cgroup.freeze           cpu.pressure                     cpuset.mems.effective     hugetlb.2MB.events       memory.events.local  memory.swap.high        pids.events.local
cgroup.kill             cpu.stat                         dmem.current              hugetlb.2MB.events.local memory.high          memory.swap.max         pids.max
cgroup.max.depth        cpu.stat.local                   dmem.low                  hugetlb.2MB.max          memory.low           memory.swap.peak        pids.peak
cgroup.max.descendants  cpu.uclamp.max                   dmem.max                  hugetlb.2MB.numa_stat    memory.max           memory.zswap.current    rdma.current
cgroup.pressure         cpu.uclamp.min                   dmem.min                  hugetlb.2MB.rsvd.current memory.min           memory.zswap.max        rdma.max
cgroup.procs            cpu.weight                       hugetlb.1GB.current       hugetlb.2MB.rsvd.max     memory.numa_stat     memory.zswap.writeback
cgroup.stat             cpu.weight.nice                  hugetlb.1GB.events        io.max                   memory.oom.group     misc.current
cgroup.subtree_control  cpuset.cpus                      hugetlb.1GB.events.local  io.pressure              memory.peak          misc.events
cgroup.threads          cpuset.cpus.effective            hugetlb.1GB.max           io.prio.class            memory.pressure      misc.events.local
cgroup.type             cpuset.cpus.exclusive            hugetlb.1GB.numa_stat     io.stat                  memory.reclaim       misc.max
cpu.idle                cpuset.cpus.exclusive.effective  hugetlb.1GB.rsvd.current  io.weight                memory.stat          misc.peak
ubuntu@ip-172-31-4-90:~$ cat /sys/fs/cgroup/system.slice/docker-60ab882211e0aec01a4d678be45d13f7220666d58eff9360ae54fa6d43967f81.scope/memory.max
268435456
ubuntu@ip-172-31-4-90:~$ |
```

Throttling



ubuntu@ip-172-31-4-90: ~

```
ubuntu@ip-172-31-4-90:~$ cat /sys/fs/cgroup/system.slice/docker-60ab882211e0aec01a4d678be45d13f7220666d58eff9360ae54fa6d43967f81.scope/cpu.stat
usage_usec 43905
user_usec 24940
system_usec 18964
nice_usec 0
core_sched.force_idle_usec 0
nr_periods 3
nr_throttled 0
throttled_usec 0
nr_bursts 0
burst_usec 0
ubuntu@ip-172-31-4-90:~$ |
```



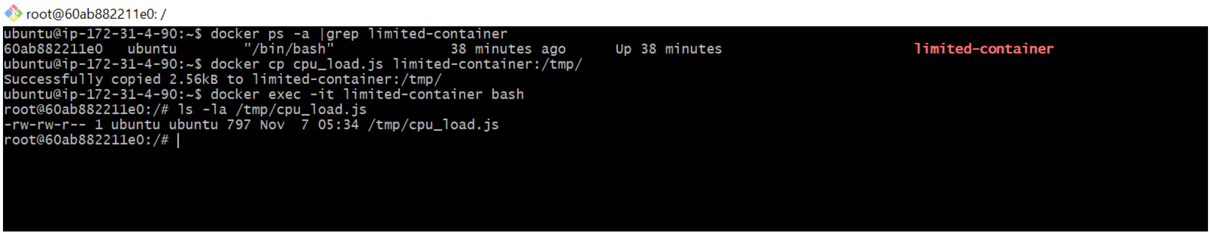
ubuntu@ip-172-31-4-90: ~

```
CONTAINER ID   NAME                CPU %     MEM USAGE / LIMIT     MEM %     NET I/O   BLOCK I/O   PIDS
60ab882211e0   limited-container   --        -- / --               --        --        --          --
```

Observations

- I created a container using `docker run -dit --name limited-container --cpus="0.5" --memory="256m" ubuntu`, which limits it to half a CPU core and 256 MB of memory.
- Using `docker inspect`, I found its main process ID was 12010. Checking `/proc/12010/cgroup` showed it's managed under `/system.slice/docker-...scope`, confirming Docker uses cgroups v2. The CPU limit (`cat cpu.max`) was `50000 100000`, meaning 50% CPU usage, and the memory limit (`cat memory.max`) was `268435456` bytes (256 MB). The CPU stats showed no throttling, and `mount | grep cgroup` confirmed cgroup v2 is active.
- Finally, `docker stats limited-container` showed that CPU and memory usage stayed within the set limits.

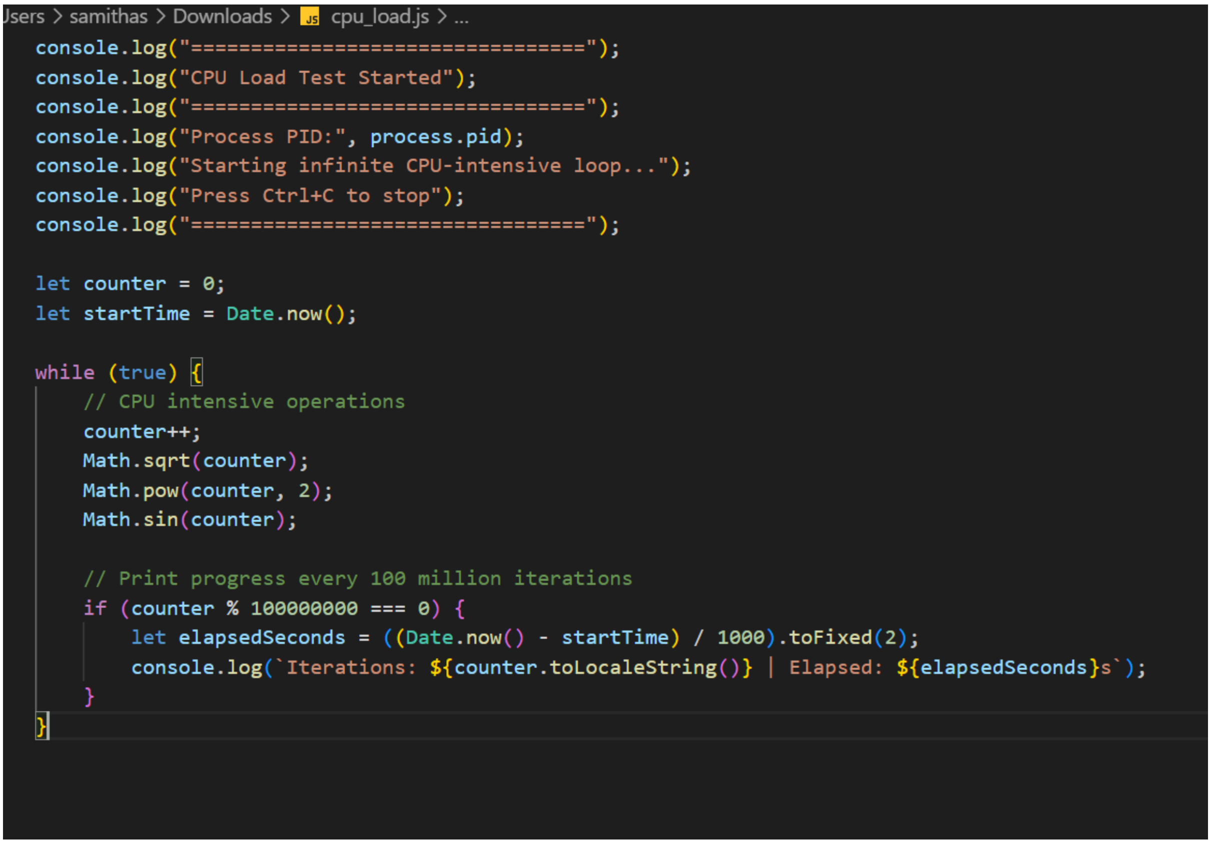
6. Resource Behavior Under Load



root@60ab882211e0: /

```
ubuntu@ip-172-31-4-90:~$ docker ps -a |grep limited-container
60ab882211e0   ubuntu         "/bin/bash"                38 minutes ago      Up 38 minutes                           limited-container
ubuntu@ip-172-31-4-90:~$ docker cp cpu_load.js limited-container:/tmp/
Successfully copied 2.56kB to limited-container:/tmp/
ubuntu@ip-172-31-4-90:~$ docker exec -it limited-container bash
root@60ab882211e0:/# ls -la /tmp/cpu_load.js
-rw-rw-r-- 1 ubuntu ubuntu 797 Nov  7 05:34 /tmp/cpu_load.js
root@60ab882211e0:/# |
```

Cpu_load.js



Jsers > samithas > Downloads > cpu_load.js > ...

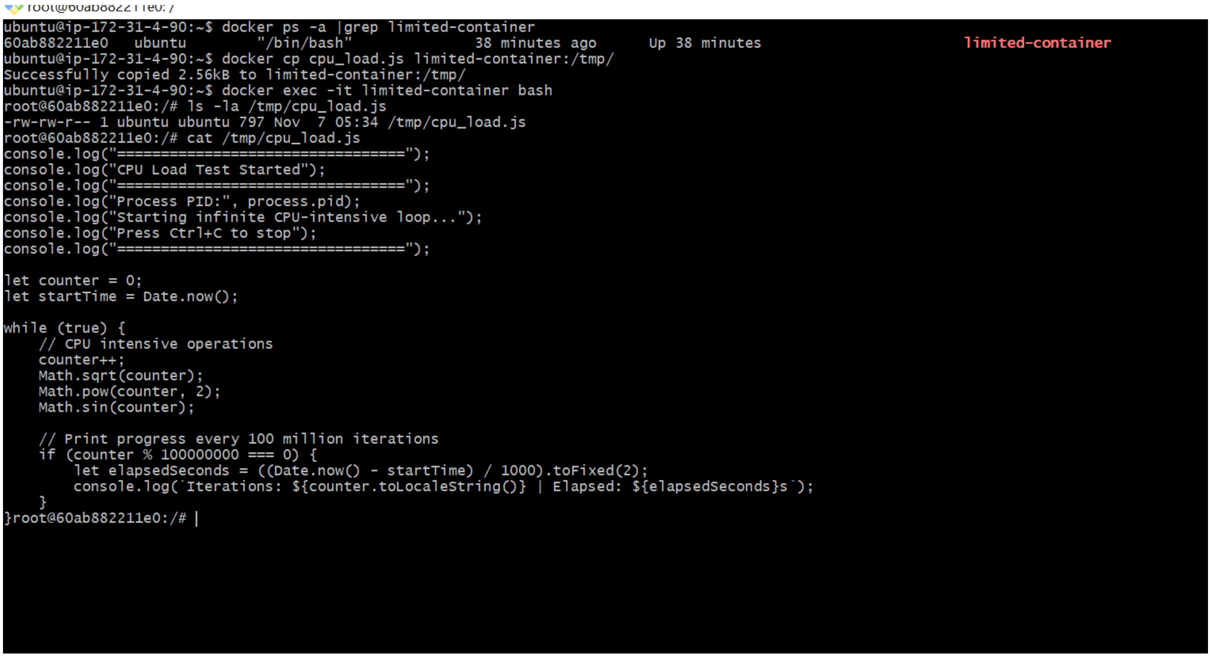
```
console.log("=================================");
console.log("CPU Load Test Started");
console.log("=================================");
console.log("Process PID:", process.pid);
console.log("Starting infinite CPU-intensive loop...");
console.log("Press Ctrl+C to stop");
console.log("=================================");

let counter = 0;
let startTime = Date.now();

while (true) {
    // CPU intensive operations
    counter++;
    Math.sqrt(counter);
    Math.pow(counter, 2);
    Math.sin(counter);

    // Print progress every 100 million iterations
    if (counter % 100000000 === 0) {
        let elapsedSeconds = ((Date.now() - startTime) / 1000).toFixed(2);
        console.log(`Iterations: ${counter.toLocaleString()} | Elapsed: ${elapsedSeconds}s`);
    }
}
```

Check the js file



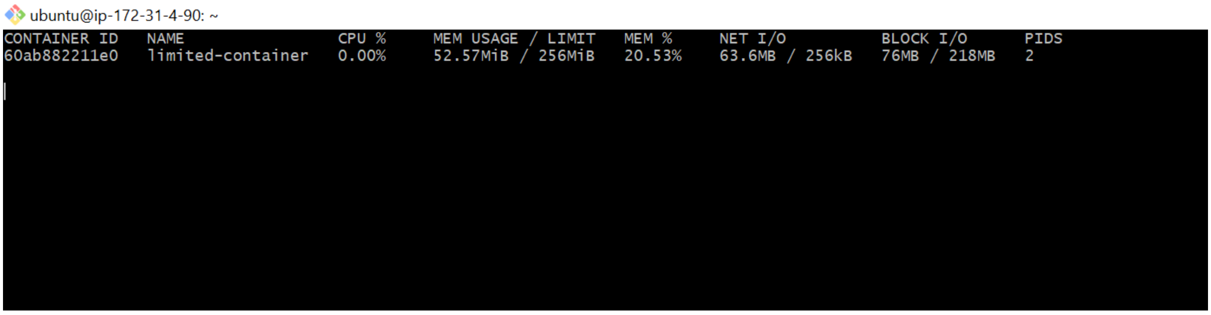
```
ubuntu@ip-172-31-4-90:~$ docker ps -a |grep limited-container
60ab882211e0   ubuntu         "/bin/bash"               38 minutes ago      Up 38 minutes                            limited-container
ubuntu@ip-172-31-4-90:~$ docker cp cpu_load.js limited-container:/tmp/
Successfully copied 2.56kB to limited-container:/tmp/
ubuntu@ip-172-31-4-90:~$ docker exec -it limited-container bash
root@60ab882211e0:/# ls -la /tmp/cpu_load.js
-rw-rw-r-- 1 ubuntu ubuntu 797 Nov  7 05:34 /tmp/cpu_load.js
root@60ab882211e0:/# cat /tmp/cpu_load.js
console.log("================================");
console.log("CPU Load Test Started");
console.log("================================");
console.log("Process PID:", process.pid);
console.log("Starting infinite CPU-intensive loop...");
console.log("Press Ctrl+C to stop");
console.log("================================");

let counter = 0;
let startTime = Date.now();

while (true) {
    // CPU intensive operations
    counter++;
    Math.sqrt(counter);
    Math.pow(counter, 2);
    Math.sin(counter);

    // Print progress every 100 million iterations
    if (counter % 100000000 === 0) {
        let elapsedSeconds = ((Date.now() - startTime) / 1000).toFixed(2);
        console.log(`Iterations: ${counter.toLocaleString()} | Elapsed: ${elapsedSeconds}s`);
    }
}root@60ab882211e0:/# |
```

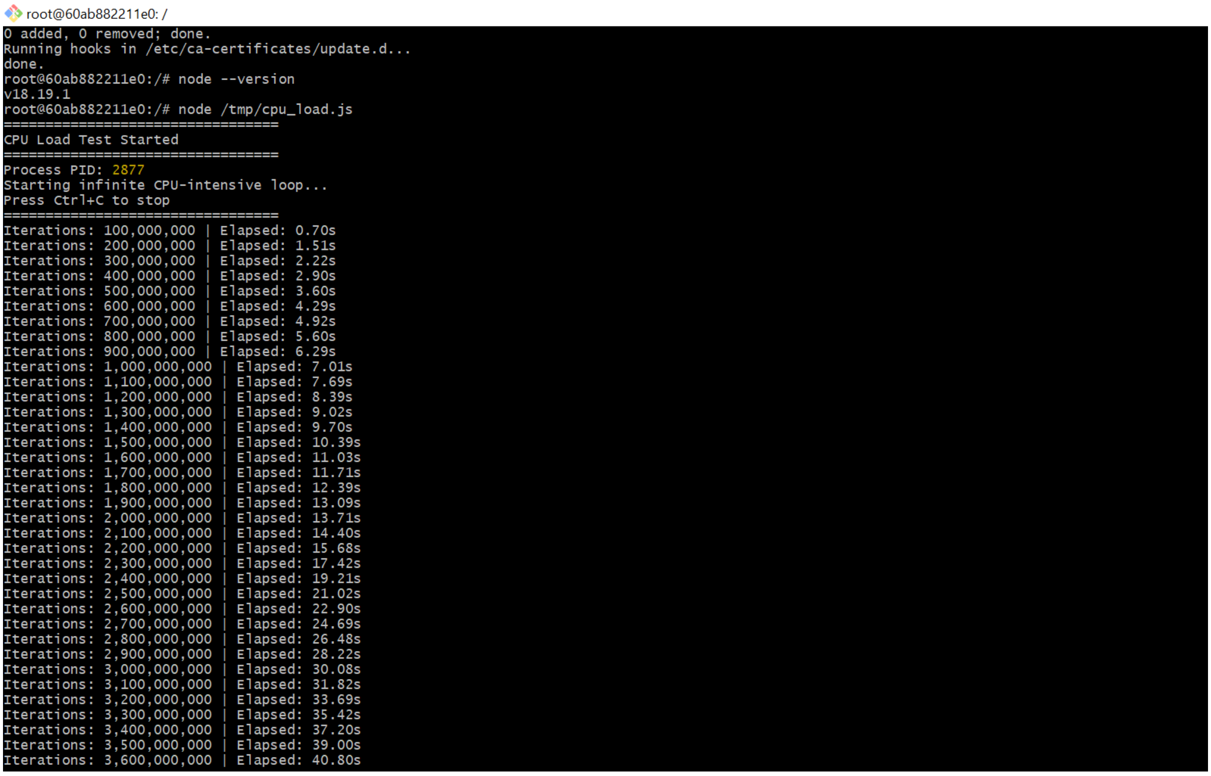
Monitor terminal (before)



ubuntu@ip-172-31-4-90: ~

```
CONTAINER ID   NAME                CPU %     MEM USAGE / LIMIT     MEM %     NET I/O          BLOCK I/O       PIDS
60ab882211e0   limited-container   0.00%     52.57MiB / 256MiB     20.53%    63.6MB / 256kB   76MB / 218MB    2
```

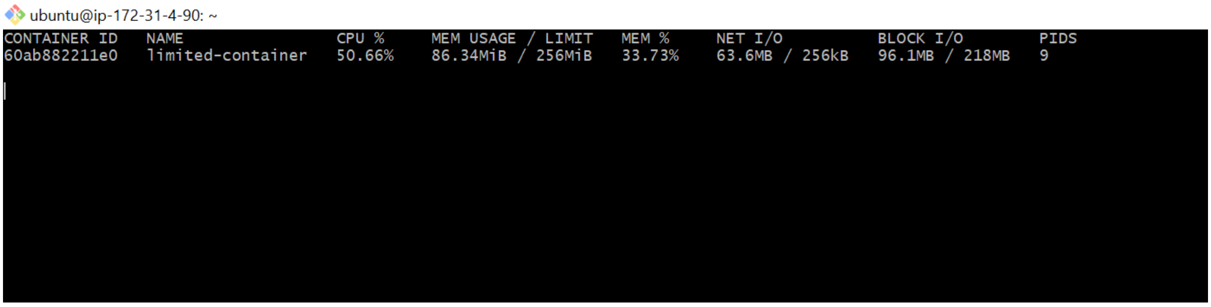
Cpu_laod.js file run



root@60ab882211e0: /

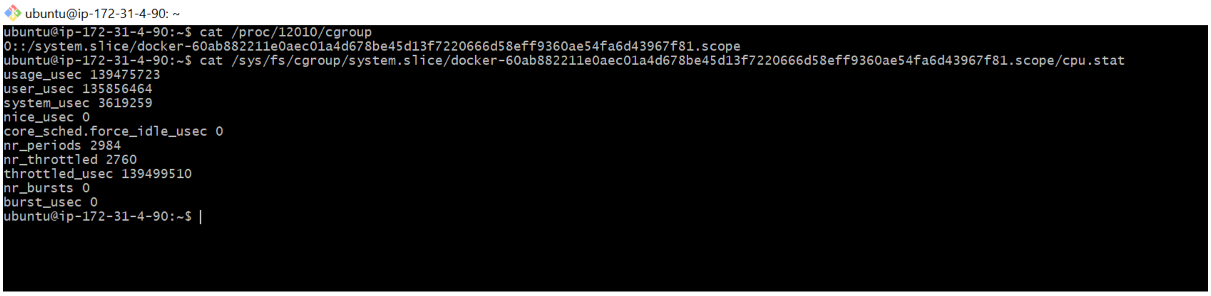
```
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
root@60ab882211e0:/# node --version
v18.19.1
root@60ab882211e0:/# node /tmp/cpu_load.js
================================
CPU Load Test Started
================================
Process PID: 2877
Starting infinite CPU-intensive loop...
Press Ctrl+C to stop
================================
Iterations: 100,000,000 | Elapsed: 0.70s
Iterations: 200,000,000 | Elapsed: 1.51s
Iterations: 300,000,000 | Elapsed: 2.22s
Iterations: 400,000,000 | Elapsed: 2.90s
Iterations: 500,000,000 | Elapsed: 3.60s
Iterations: 600,000,000 | Elapsed: 4.29s
Iterations: 700,000,000 | Elapsed: 4.92s
Iterations: 800,000,000 | Elapsed: 5.60s
Iterations: 900,000,000 | Elapsed: 6.29s
Iterations: 1,000,000,000 | Elapsed: 7.01s
Iterations: 1,100,000,000 | Elapsed: 7.69s
Iterations: 1,200,000,000 | Elapsed: 8.39s
Iterations: 1,300,000,000 | Elapsed: 9.02s
Iterations: 1,400,000,000 | Elapsed: 9.70s
Iterations: 1,500,000,000 | Elapsed: 10.39s
Iterations: 1,600,000,000 | Elapsed: 11.03s
Iterations: 1,700,000,000 | Elapsed: 11.71s
Iterations: 1,800,000,000 | Elapsed: 12.39s
Iterations: 1,900,000,000 | Elapsed: 13.09s
Iterations: 2,000,000,000 | Elapsed: 13.71s
Iterations: 2,100,000,000 | Elapsed: 14.40s
Iterations: 2,200,000,000 | Elapsed: 15.68s
Iterations: 2,300,000,000 | Elapsed: 17.42s
Iterations: 2,400,000,000 | Elapsed: 19.21s
Iterations: 2,500,000,000 | Elapsed: 21.02s
Iterations: 2,600,000,000 | Elapsed: 22.90s
Iterations: 2,700,000,000 | Elapsed: 24.69s
Iterations: 2,800,000,000 | Elapsed: 26.48s
Iterations: 2,900,000,000 | Elapsed: 28.22s
Iterations: 3,000,000,000 | Elapsed: 30.08s
Iterations: 3,100,000,000 | Elapsed: 31.82s
Iterations: 3,200,000,000 | Elapsed: 33.69s
Iterations: 3,300,000,000 | Elapsed: 35.42s
Iterations: 3,400,000,000 | Elapsed: 37.20s
Iterations: 3,500,000,000 | Elapsed: 39.00s
Iterations: 3,600,000,000 | Elapsed: 40.80s
```

Monitor terminal (After)



```
CONTAINER ID   NAME                CPU %     MEM USAGE / LIMIT    MEM %     NET I/O          BLOCK I/O        PIDS
60ab882211e0   limited-container   50.66%    86.34MiB / 256MiB    33.73%    63.6MB / 256kB   96.1MB / 218MB   9
```

Throttling proof



```
ubuntu@ip-172-31-4-90:~$ cat /proc/12010/cgroup
0::/system.slice/docker-60ab882211e0aec01a4d678be45d13f7220666d58eff9360ae54fa6d43967f81.scope
ubuntu@ip-172-31-4-90:~$ cat /sys/fs/cgroup/system.slice/docker-60ab882211e0aec01a4d678be45d13f7220666d58eff9360ae54fa6d43967f81.scope/cpu.stat
usage_usec 139475723
user_usec 135856464
system_usec 3619259
nice_usec 0
core_sched.force_idle_usec 0
nr_periods 2984
nr_throttled 2760
throttled_usec 139499510
nr_bursts 0
burst_usec 0
ubuntu@ip-172-31-4-90:~$ |
```

Observations

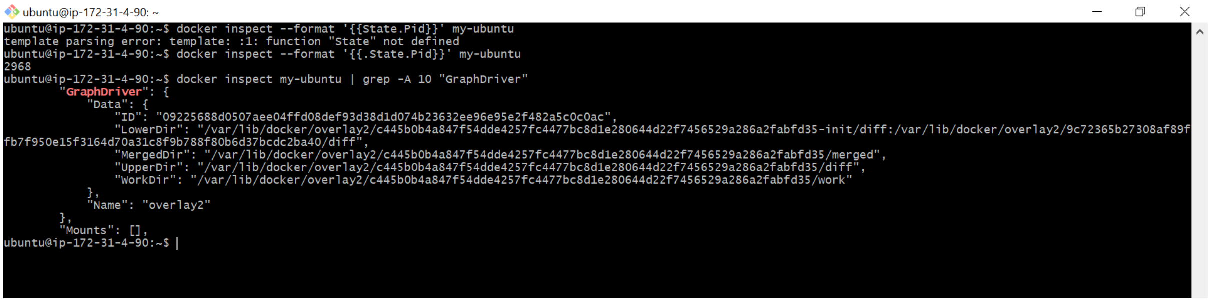
- First CPU Limit enforcement,since CPU limited to 0.5(50%) in limited container,when the js ran in a infinite loop thats get 100% CPU usage.docker stats showed exactly 50.66% CPU (limit enforced)cgroup stats showed 2,760 throttle events in 2,984 periods.

Learnings

- Understand how Docker controls system resources using cgroups. By running processes inside a limited container, I saw how Docker can restrict CPU and memory usage to prevent a single container from overusing system resources. It showed that even when a process tries to use more power, Docker enforces the set limits and keeps the host system stable.

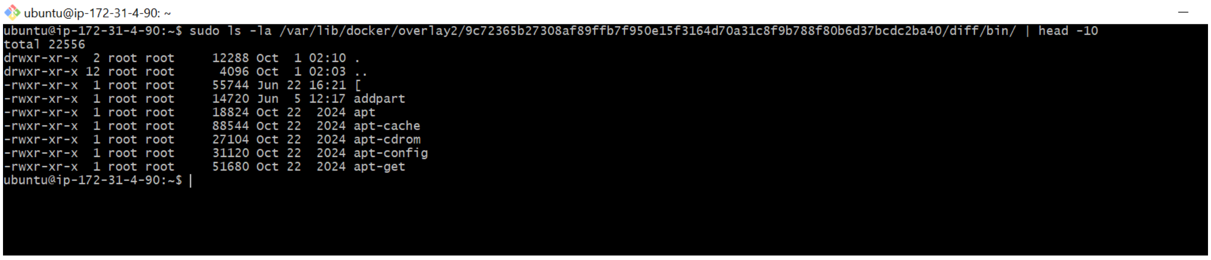
7. Filesystem Layer Analysis

Add test.txt and show in upperdirectory



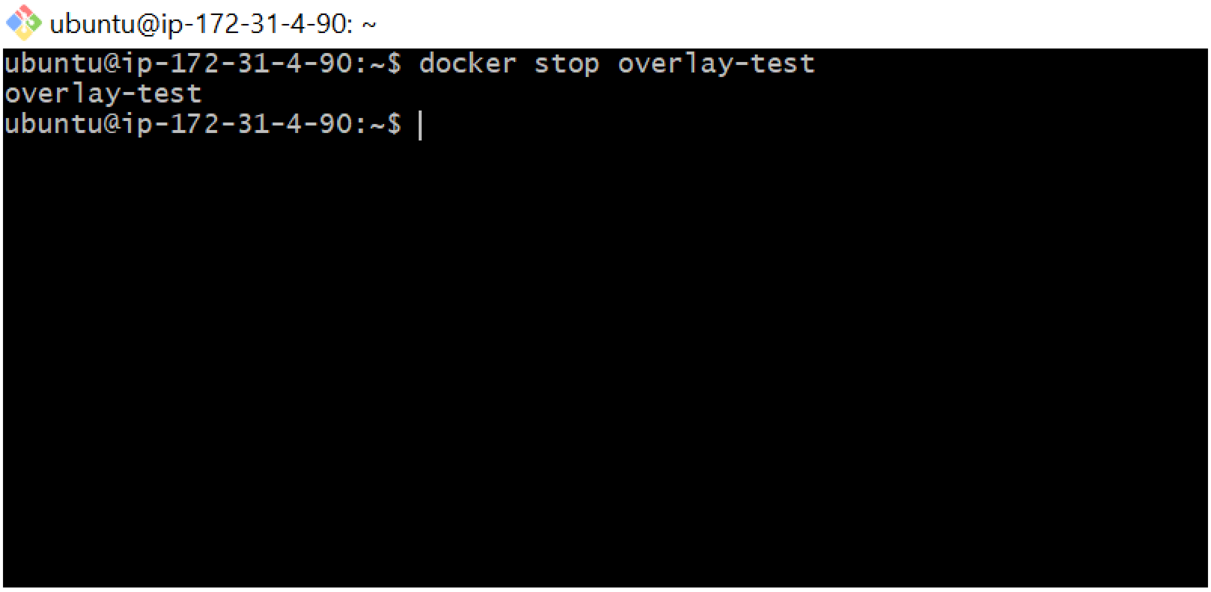
```
ubuntu@ip-172-31-4-90:~$ docker inspect --format '{{State.Pid}}' my-ubuntu
template parsing error: template: :1: function "State" not defined
ubuntu@ip-172-31-4-90:~$ docker inspect --format '{{.State.Pid}}' my-ubuntu
2968
ubuntu@ip-172-31-4-90:~$ docker inspect my-ubuntu | grep -A 10 "GraphDriver"
        "GraphDriver": {
            "Data": {
                "ID": "09225688d0507aee04ffd08def93d38d1d074b23632ee96e95e2f482a5c0c0ac",
                "LowerDir": "/var/lib/docker/overlay2/c445b0b4a847f54dde4257fc4477bc8d1e280644d22f7456529a286a2fabfd35-init/diff:/var/lib/docker/overlay2/9c72365b27308af89f
fb7f950e15f3164d70a31c8f9b788f80b6d37bcdc2ba40/diff",
                "MergedDir": "/var/lib/docker/overlay2/c445b0b4a847f54dde4257fc4477bc8d1e280644d22f7456529a286a2fabfd35/merged",
                "UpperDir": "/var/lib/docker/overlay2/c445b0b4a847f54dde4257fc4477bc8d1e280644d22f7456529a286a2fabfd35/diff",
                "WorkDir": "/var/lib/docker/overlay2/c445b0b4a847f54dde4257fc4477bc8d1e280644d22f7456529a286a2fabfd35/work"
            },
            "Name": "overlay2"
        },
        "Mounts": [],
ubuntu@ip-172-31-4-90:~$ |
```

Lower directory



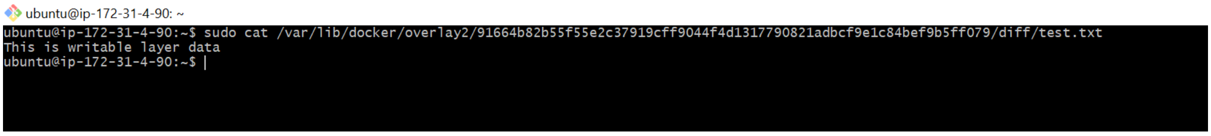
```
ubuntu@ip-172-31-4-90:~$ sudo ls -la /var/lib/docker/overlay2/9c72365b27308af89ffb7f950e15f3164d70a31c8f9b788f80b6d37bcdc2ba40/diff/bin/ | head -10
total 22556
drwxr-xr-x  2 root root     12288 Oct  1 02:10 .
drwxr-xr-x 12 root root      4096 Oct  1 02:03 ..
-rwxr-xr-x  1 root root     55744 Jun 22 16:21 [
-rwxr-xr-x  1 root root     14720 Jun  5 12:17 addpart
-rwxr-xr-x  1 root root     18824 Oct 22  2024 apt
-rwxr-xr-x  1 root root     88544 Oct 22  2024 apt-cache
-rwxr-xr-x  1 root root     27104 Oct 22  2024 apt-cdrom
-rwxr-xr-x  1 root root     31120 Oct 22  2024 apt-config
-rwxr-xr-x  1 root root     51680 Oct 22  2024 apt-get
ubuntu@ip-172-31-4-90:~$ |
```

Docker stop



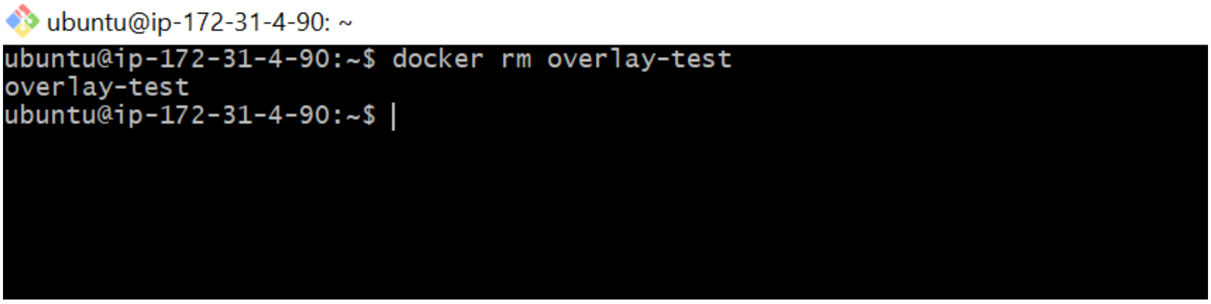
```
ubuntu@ip-172-31-4-90:~$ docker stop overlay-test
overlay-test
ubuntu@ip-172-31-4-90:~$ |
```

Still file in UpperDir



```
ubuntu@ip-172-31-4-90:~$ sudo cat /var/lib/docker/overlay2/91664b82b55f55e2c37919cff9044f4d1317790821adbcf9e1c84bef9b5ff079/diff/test.txt
This is writable layer data
ubuntu@ip-172-31-4-90:~$ |
```

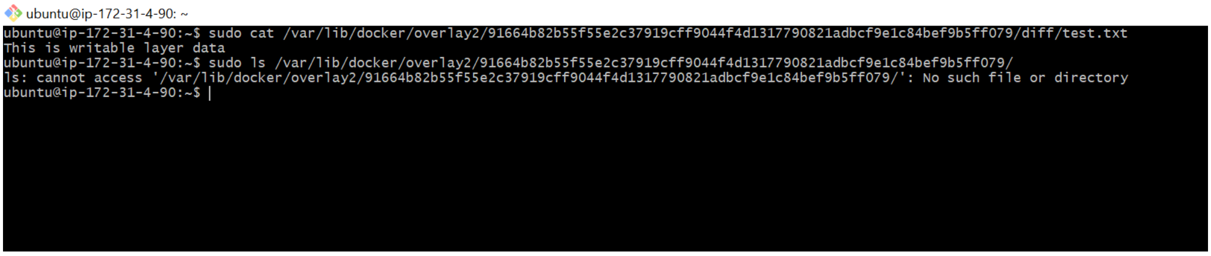
Docker remove



ubuntu@ip-172-31-4-90: ~

```
ubuntu@ip-172-31-4-90:~$ docker rm overlay-test
overlay-test
ubuntu@ip-172-31-4-90:~$ |
```

Remove from upperdirectory



ubuntu@ip-172-31-4-90: ~

```
ubuntu@ip-172-31-4-90:~$ sudo cat /var/lib/docker/overlay2/91664b82b55f55e2c37919cff9044f4d1317790821adbcf9e1c84bef9b5ff079/diff/test.txt
This is writable layer data
ubuntu@ip-172-31-4-90:~$ sudo ls /var/lib/docker/overlay2/91664b82b55f55e2c37919cff9044f4d1317790821adbcf9e1c84bef9b5ff079/
ls: cannot access '/var/lib/docker/overlay2/91664b82b55f55e2c37919cff9044f4d1317790821adbcf9e1c84bef9b5ff079/': No such file or directory
ubuntu@ip-172-31-4-90:~$ |
```

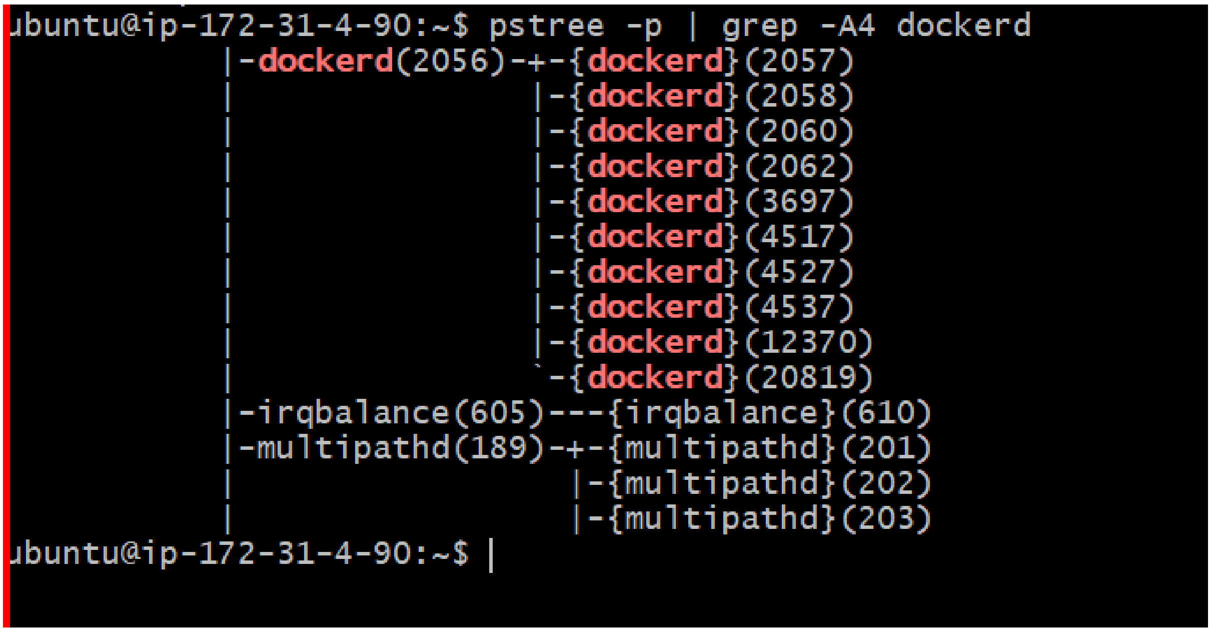
Observation

- I noticed that Docker creates different layer directories inside /var/lib/docker/overlay2/. The test.txt file I made was saved in the writable UpperDir, which shows that new files inside a container are stored there.
- The LowerDir had the normal Ubuntu system files that come from the image and are read-only. After stopping the container, the test.txt file was still there, meaning the writable layer remains until the container is deleted. When the container was removed, the writable layer also disappeared.

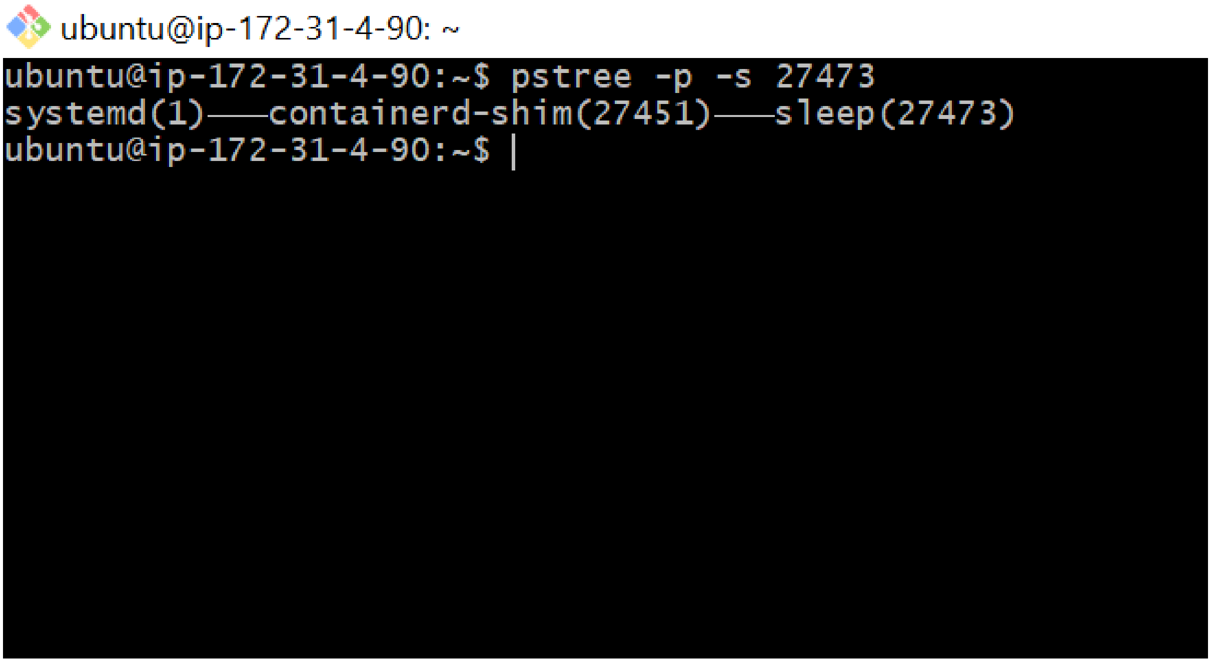
Learning

- This experiment helped me understand how Docker's overlay2 storage works. Each container combines read-only image layers with its own writable layer, so any new files or changes are stored separately.

8. Process Creation Flow Examination



```
ubuntu@ip-172-31-4-90:~$ pstree -p | grep -A4 dockerd
           |-dockerd(2056)-+-{dockerd}(2057)
           |               |-{dockerd}(2058)
           |               |-{dockerd}(2060)
           |               |-{dockerd}(2062)
           |               |-{dockerd}(3697)
           |               |-{dockerd}(4517)
           |               |-{dockerd}(4527)
           |               |-{dockerd}(4537)
           |               |-{dockerd}(12370)
           |               `-{dockerd}(20819)
           |-irqbalance(605)---{irqbalance}(610)
           |-multipathd(189)-+-{multipathd}(201)
           |                 |-{multipathd}(202)
           |                 |-{multipathd}(203)
ubuntu@ip-172-31-4-90:~$ |
```



ubuntu@ip-172-31-4-90: ~

```
ubuntu@ip-172-31-4-90:~$ pstree -p -s 27473
systemd(1)───containerd-shim(27451)───sleep(27473)
ubuntu@ip-172-31-4-90:~$ |
```

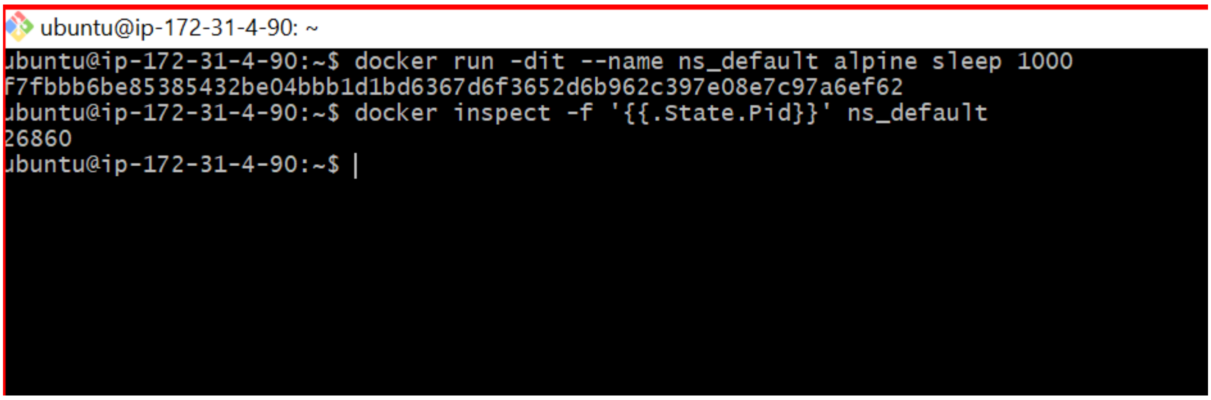
Observation

- When we run the pstree command with the container’s PID, it shows the process structure of the container.
- Can see that Docker creates several layers of processes as first dockerd, then containerd, followed by containerd-shim, runc, and finally the actual process running inside the container. (But with current versions dockerd and containerd and not attached)

- This shows how Docker isolates container processes while managing them through the parent processes on the host.

9. Comparative Namespace Experiment

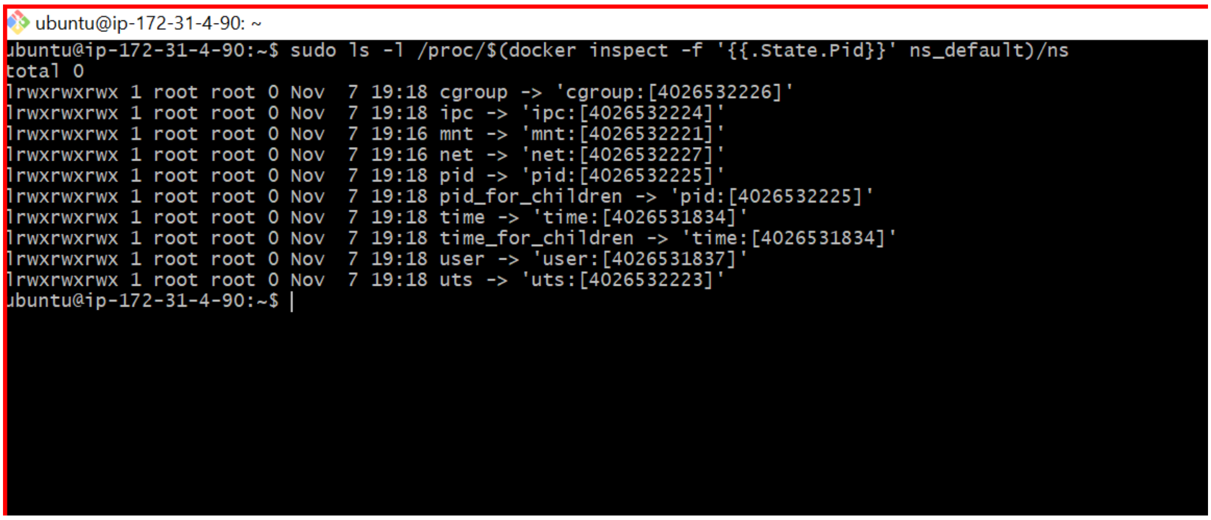
Normal Container with Default Namespaces



ubuntu@ip-172-31-4-90: ~

```
ubuntu@ip-172-31-4-90:~$ docker run -dit --name ns_default alpine sleep 1000
f7fbbb6be85385432be04bbb1d1bd6367d6f3652d6b962c397e08e7c97a6ef62
ubuntu@ip-172-31-4-90:~$ docker inspect -f '{{.State.Pid}}' ns_default
26860
ubuntu@ip-172-31-4-90:~$ |
```

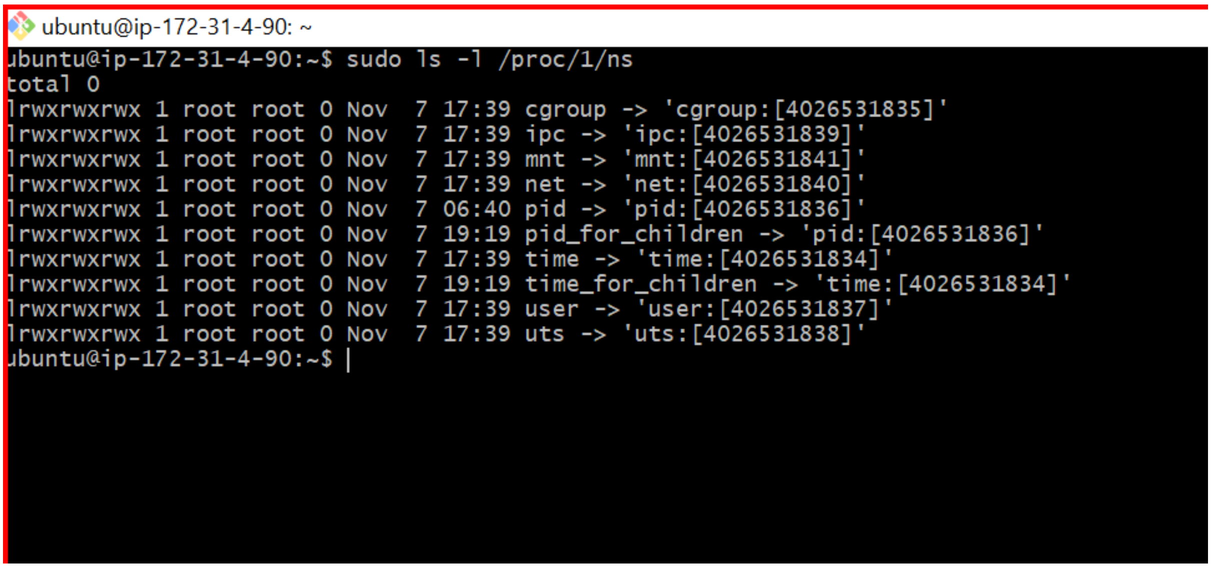
Check its namespaces:



ubuntu@ip-172-31-4-90: ~

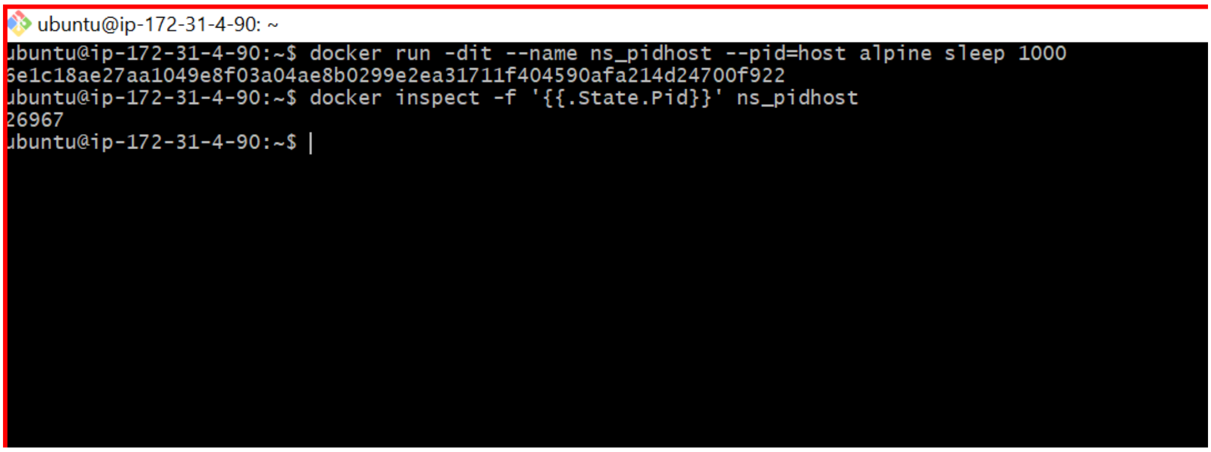
```
ubuntu@ip-172-31-4-90:~$ sudo ls -l /proc/$(docker inspect -f '{{.State.Pid}}' ns_default)/ns
total 0
lrwxrwxrwx 1 root root 0 Nov  7 19:18 cgroup -> 'cgroup:[4026532226]'
lrwxrwxrwx 1 root root 0 Nov  7 19:18 ipc -> 'ipc:[4026532224]'
lrwxrwxrwx 1 root root 0 Nov  7 19:16 mnt -> 'mnt:[4026532221]'
lrwxrwxrwx 1 root root 0 Nov  7 19:16 net -> 'net:[4026532227]'
lrwxrwxrwx 1 root root 0 Nov  7 19:18 pid -> 'pid:[4026532225]'
lrwxrwxrwx 1 root root 0 Nov  7 19:18 pid_for_children -> 'pid:[4026532225]'
lrwxrwxrwx 1 root root 0 Nov  7 19:18 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 19:18 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 19:18 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Nov  7 19:18 uts -> 'uts:[4026532223]'
ubuntu@ip-172-31-4-90:~$ |
```

host’s namespaces (PID 1 usually belongs to systemd or init)



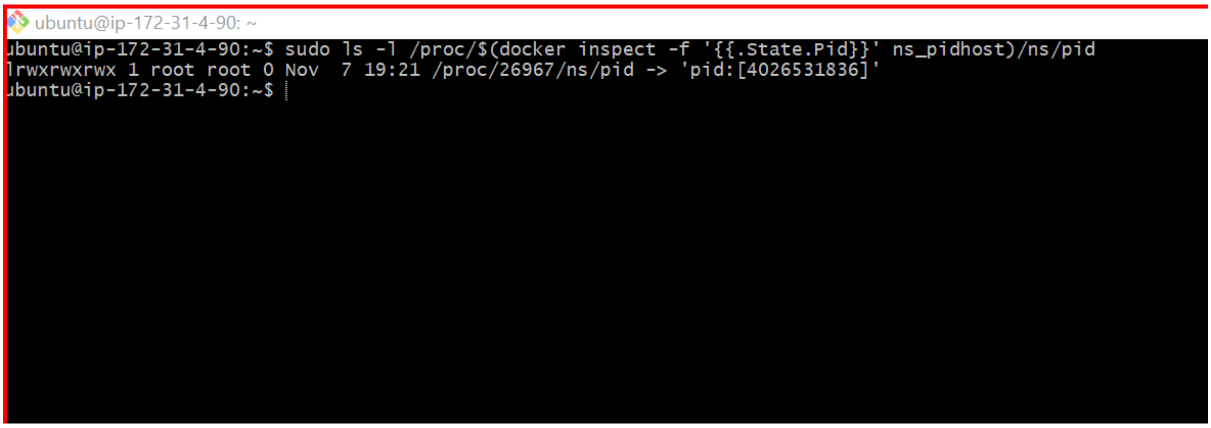
```
ubuntu@ip-172-31-4-90:~$ sudo ls -l /proc/1/ns
total 0
lrwxrwxrwx 1 root root 0 Nov  7 17:39 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Nov  7 17:39 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 Nov  7 17:39 mnt -> 'mnt:[4026531841]'
lrwxrwxrwx 1 root root 0 Nov  7 17:39 net -> 'net:[4026531840]'
lrwxrwxrwx 1 root root 0 Nov  7 06:40 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Nov  7 19:19 pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Nov  7 17:39 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 19:19 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 17:39 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Nov  7 17:39 uts -> 'uts:[4026531838]'
ubuntu@ip-172-31-4-90:~$ |
```

Container with Shared PID Namespace (--pid=host)

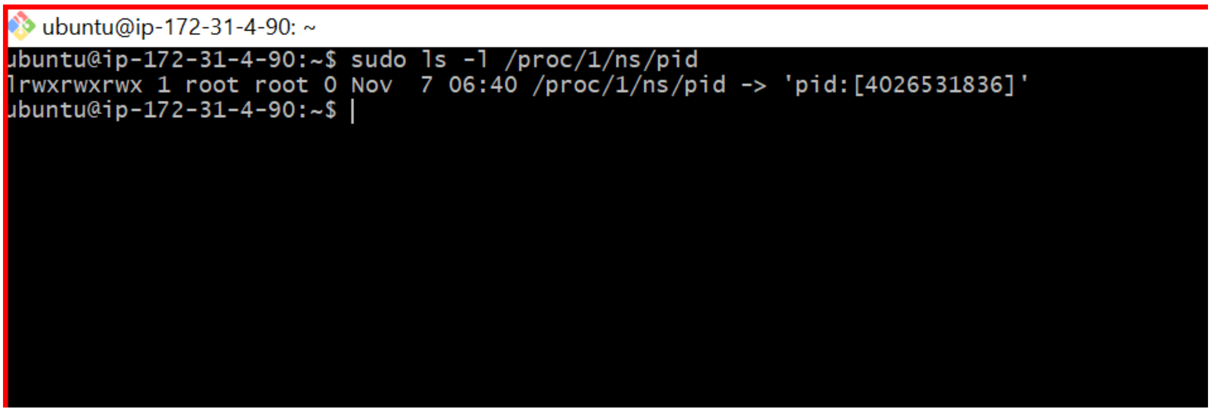


```
ubuntu@ip-172-31-4-90:~$ docker run -dit --name ns_pidhost --pid=host alpine sleep 1000
6e1c18ae27aa1049e8f03a04ae8b0299e2ea31711f404590afa214d24700f922
ubuntu@ip-172-31-4-90:~$ docker inspect -f '{{.State.Pid}}' ns_pidhost
26967
ubuntu@ip-172-31-4-90:~$ |
```

Check the namespaces



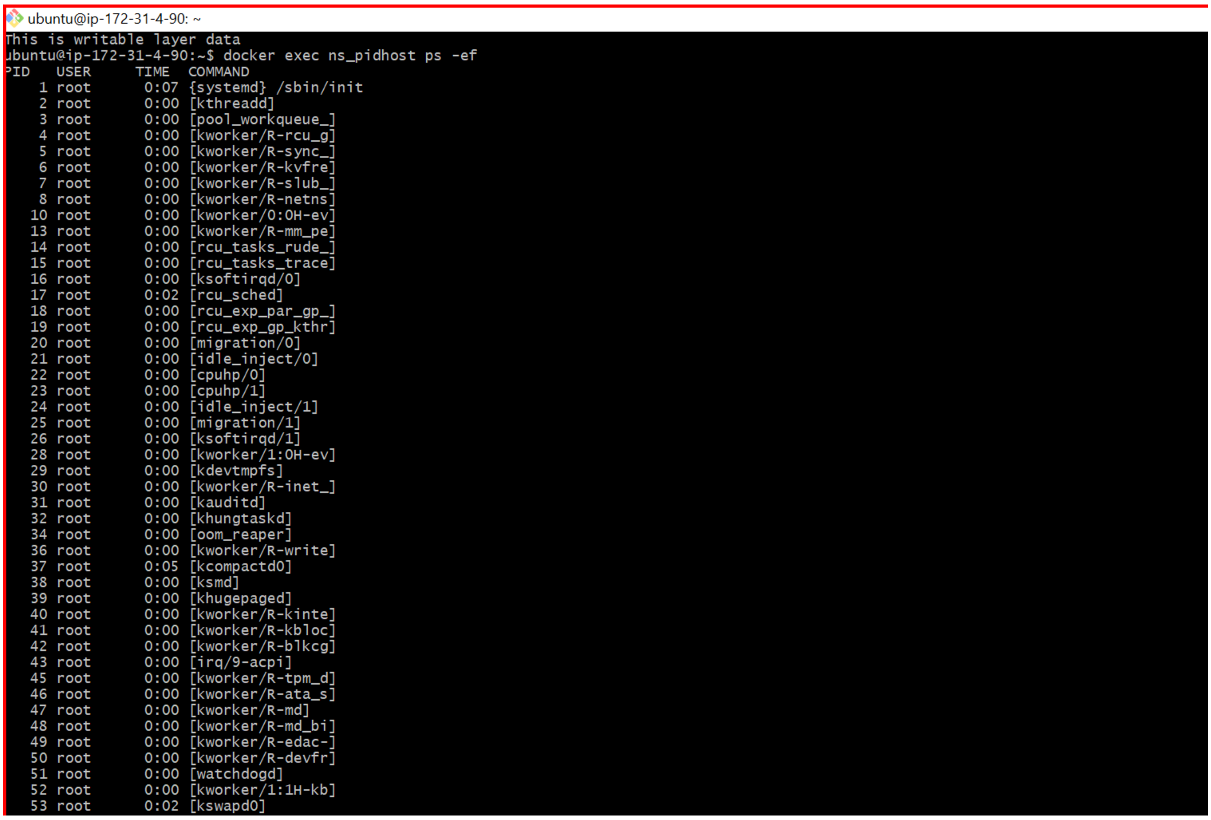
```
ubuntu@ip-172-31-4-90:~$ sudo ls -l /proc/$(docker inspect -f '{{.State.Pid}}' ns_pidhost)/ns/pid
lrwxrwxrwx 1 root root 0 Nov  7 19:21 /proc/26967/ns/pid -> 'pid:[4026531836]'
ubuntu@ip-172-31-4-90:~$
```



ubuntu@ip-172-31-4-90: ~

```
ubuntu@ip-172-31-4-90:~$ sudo ls -l /proc/1/ns/pid
lrwxrwxrwx 1 root root 0 Nov  7 06:40 /proc/1/ns/pid -> 'pid:[4026531836]'
ubuntu@ip-172-31-4-90:~$ |
```

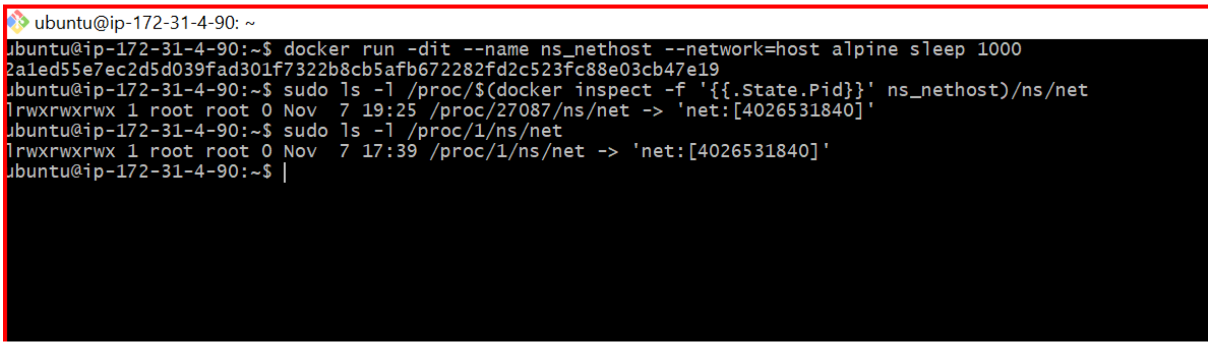
all host processes



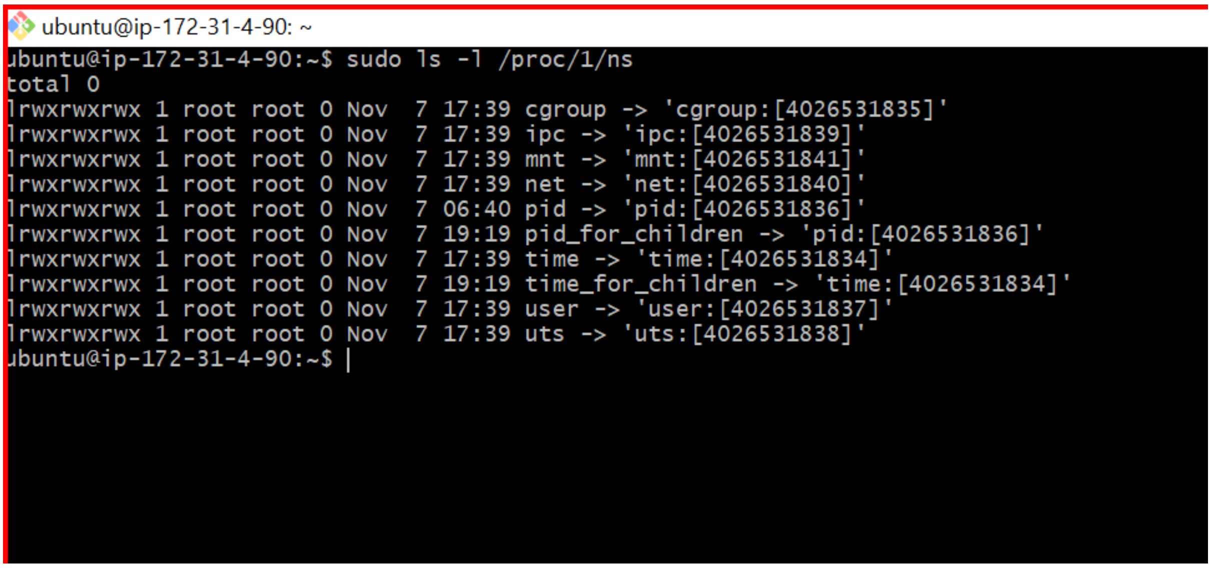
ubuntu@ip-172-31-4-90: ~

```
This is writable layer data
ubuntu@ip-172-31-4-90:~$ docker exec ns_pidhost ps -ef
PID   USER     TIME  COMMAND
    1 root      0:07 {systemd} /sbin/init
    2 root      0:00 [kthreadd]
    3 root      0:00 [pool_workqueue_]
    4 root      0:00 [kworker/R-rcu_g]
    5 root      0:00 [kworker/R-sync_]
    6 root      0:00 [kworker/R-kvfre]
    7 root      0:00 [kworker/R-slub_]
    8 root      0:00 [kworker/R-netns]
   10 root      0:00 [kworker/0:0H-ev]
   13 root      0:00 [kworker/R-mm_pe]
   14 root      0:00 [rcu_tasks_rude_]
   15 root      0:00 [rcu_tasks_trace]
   16 root      0:00 [ksoftirqd/0]
   17 root      0:02 [rcu_sched]
   18 root      0:00 [rcu_exp_par_gp_]
   19 root      0:00 [rcu_exp_gp_kthr]
   20 root      0:00 [migration/0]
   21 root      0:00 [idle_inject/0]
   22 root      0:00 [cpuhp/0]
   23 root      0:00 [cpuhp/1]
   24 root      0:00 [idle_inject/1]
   25 root      0:00 [migration/1]
   26 root      0:00 [ksoftirqd/1]
   28 root      0:00 [kworker/1:0H-ev]
   29 root      0:00 [kdevtmpfs]
   30 root      0:00 [kworker/R-inet_]
   31 root      0:00 [kauditd]
   32 root      0:00 [khungtaskd]
   34 root      0:00 [oom_reaper]
   36 root      0:00 [kworker/R-write]
   37 root      0:05 [kcompactd0]
   38 root      0:00 [ksmd]
   39 root      0:00 [khugepaged]
   40 root      0:00 [kworker/R-kinte]
   41 root      0:00 [kworker/R-kbloc]
   42 root      0:00 [kworker/R-blkcg]
   43 root      0:00 [irq/9-acpi]
   45 root      0:00 [kworker/R-tpm_d]
   46 root      0:00 [kworker/R-ata_s]
   47 root      0:00 [kworker/R-md]
   48 root      0:00 [kworker/R-md_bi]
   49 root      0:00 [kworker/R-edac-]
   50 root      0:00 [kworker/R-devfr]
   51 root      0:00 [watchdogd]
   52 root      0:00 [kworker/1:1H-kb]
   53 root      0:02 [kswapd0]
```

Container with Shared Network Namespace (--network=host)

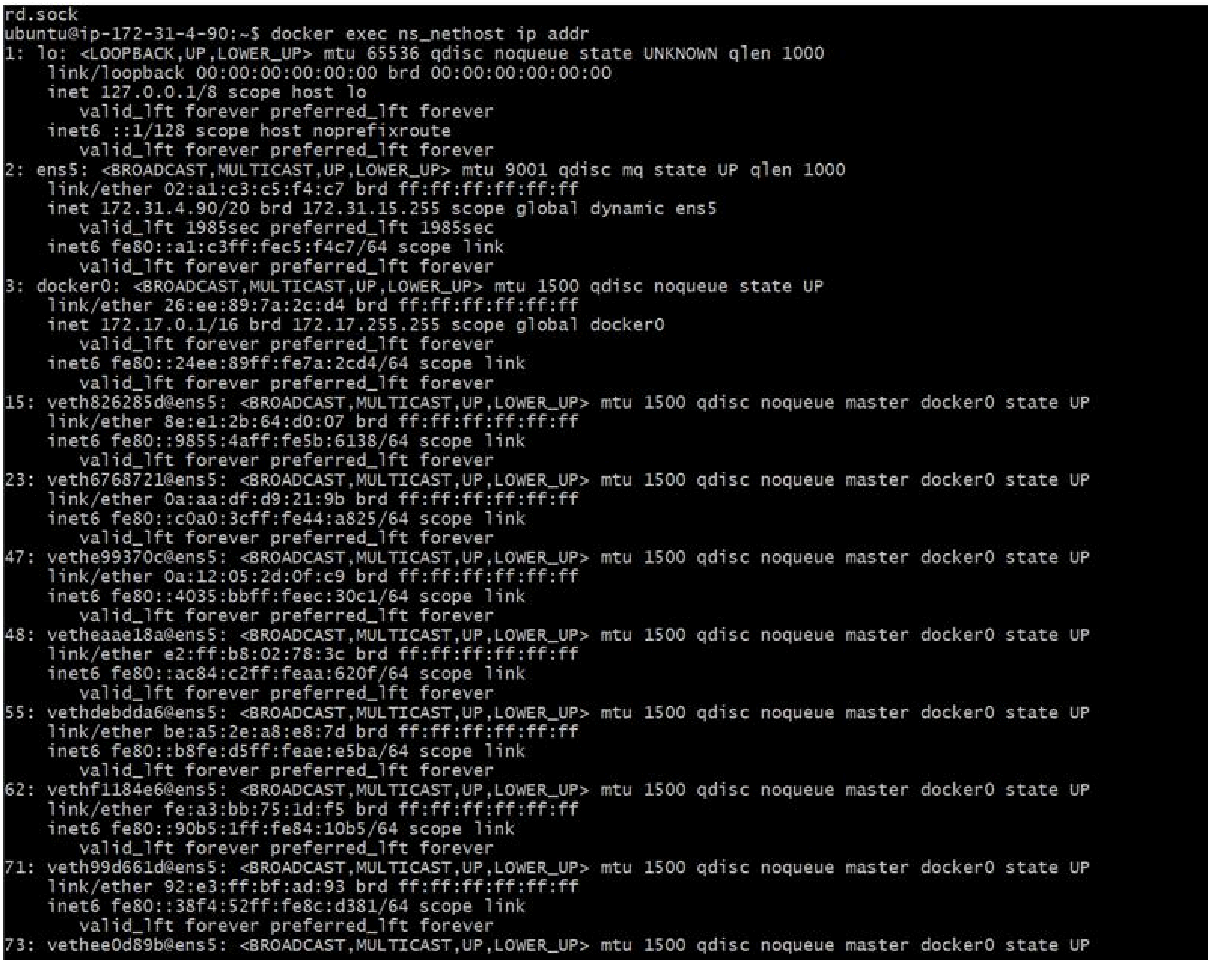


```
ubuntu@ip-172-31-4-90:~$ docker run -dit --name ns_nethost --network=host alpine sleep 1000
2a1ed55e7ec2d5d039fad301f7322b8cb5afb672282fd2c523fc88e03cb47e19
ubuntu@ip-172-31-4-90:~$ sudo ls -l /proc/$(docker inspect -f '{{.State.Pid}}' ns_nethost)/ns/net
lrwxrwxrwx 1 root root 0 Nov  7 19:25 /proc/27087/ns/net -> 'net:[4026531840]'
ubuntu@ip-172-31-4-90:~$ sudo ls -l /proc/1/ns/net
lrwxrwxrwx 1 root root 0 Nov  7 17:39 /proc/1/ns/net -> 'net:[4026531840]'
ubuntu@ip-172-31-4-90:~$ |
```



```
ubuntu@ip-172-31-4-90:~$ sudo ls -l /proc/1/ns
total 0
lrwxrwxrwx 1 root root 0 Nov  7 17:39 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Nov  7 17:39 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 Nov  7 17:39 mnt -> 'mnt:[4026531841]'
lrwxrwxrwx 1 root root 0 Nov  7 17:39 net -> 'net:[4026531840]'
lrwxrwxrwx 1 root root 0 Nov  7 06:40 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Nov  7 19:19 pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Nov  7 17:39 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 19:19 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Nov  7 17:39 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Nov  7 17:39 uts -> 'uts:[4026531838]'
ubuntu@ip-172-31-4-90:~$ |
```

Check the IP inside the container



```
rd.sock
ubuntu@ip-172-31-4-90:~$ docker exec ns_nethost ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN 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: ens5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP qlen 1000
    link/ether 02:a1:c3:c5:f4:c7 brd ff:ff:ff:ff:ff:ff
    inet 172.31.4.90/20 brd 172.31.15.255 scope global dynamic ens5
       valid_lft 1985sec preferred_lft 1985sec
    inet6 fe80::a1:c3ff:fec5:f4c7/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 26:ee:89:7a:2c:d4 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::24ee:89ff:fe7a:2cd4/64 scope link
       valid_lft forever preferred_lft forever
15: veth826285d@ens5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP
    link/ether 8e:e1:2b:64:d0:07 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::9855:4aff:fe5b:6138/64 scope link
       valid_lft forever preferred_lft forever
23: veth6768721@ens5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP
    link/ether 0a:aa:df:d9:21:9b brd ff:ff:ff:ff:ff:ff
    inet6 fe80::c0a0:3cff:fe44:a825/64 scope link
       valid_lft forever preferred_lft forever
47: vethe99370c@ens5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP
    link/ether 0a:12:05:2d:0f:c9 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::4035:bbff:feec:30c1/64 scope link
       valid_lft forever preferred_lft forever
48: vetheaae18a@ens5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP
    link/ether e2:ff:b8:02:78:3c brd ff:ff:ff:ff:ff:ff
    inet6 fe80::ac84:c2ff:feaa:620f/64 scope link
       valid_lft forever preferred_lft forever
55: vethdebdda6@ens5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP
    link/ether be:a5:2e:a8:e8:7d brd ff:ff:ff:ff:ff:ff
    inet6 fe80::b8fe:d5ff:feae:e5ba/64 scope link
       valid_lft forever preferred_lft forever
62: vethf1184e6@ens5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP
    link/ether fe:a3:bb:75:1d:f5 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::90b5:1ff:fe84:10b5/64 scope link
       valid_lft forever preferred_lft forever
71: veth99d661d@ens5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP
    link/ether 92:e3:ff:bf:ad:93 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::38f4:52ff:fe8c:d381/64 scope link
       valid_lft forever preferred_lft forever
73: vethee0d89b@ens5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP
```

Observation

- By default, each container runs in its own namespace, isolating processes and doing their networking.
- When we run the container with --pid=host, it shares the host’s process namespace, meaning it can see all host processes.
- When we run the container with --network=host, it shares the host’s network namespace, so it uses the same IP and network interfaces as the host.
- Checking the namespace files in /proc/<pid>/ns showed that containers using these options have the same namespace IDs as the host.
- This means namespace isolation is removed for those options, and the container becomes connected to the host system for that particular namespace.