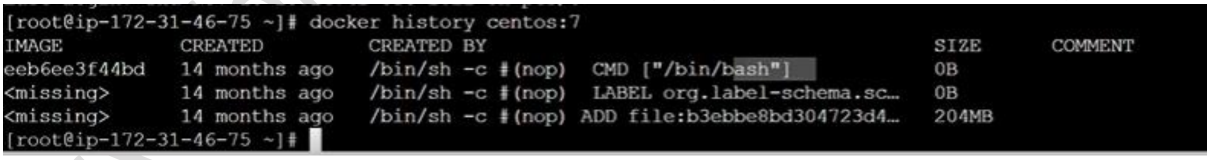
Summary

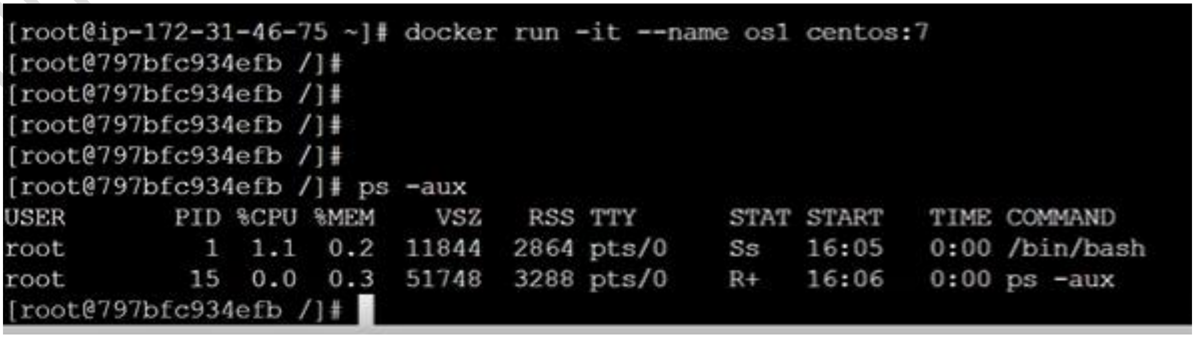
Session No – 14

- A container is just a process & it has its own personal namespace
- The container looks like an operating system internally & from the OS perspective it is just a process
- One of the things the operating system gives is isolation
- Isolation gives us security & a way for organizing our resources
- The container also gives us resources (CPU, RAM, N/C) & processes
- A namespace is the one that will give isolation to the container
- Every operating system requires
  - Hardware (CPU, RAM, HD)
  - Capability to run multiple processes
  - Network i.e. IP address
  - Hostname
  - User name/login
- If we give any process a network card, CPU & RAM, username, and hostname which is also called the process tree then we can say the process is working as an operating system
- When we launch the container from the docker run command first it launches the process of the command which is given in the docker image



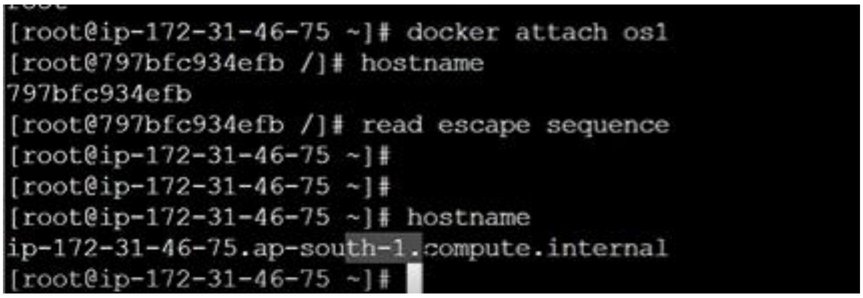
```
[root@ip-172-31-46-75 ~]# docker history centos:7
IMAGE          CREATED        CREATED BY                                      SIZE      COMMENT
eeb6ee3f44bd   14 months ago  /bin/sh -c #(nop)  CMD ["/bin/bash"]             0B
<missing>      14 months ago  /bin/sh -c #(nop)  LABEL org.label-schema.sc…    0B
<missing>      14 months ago  /bin/sh -c #(nop) ADD file:b3ebbe8bd304723d4…    204MB
[root@ip-172-31-46-75 ~]#
```

- Any process that is started by container technology is given a personal namespace



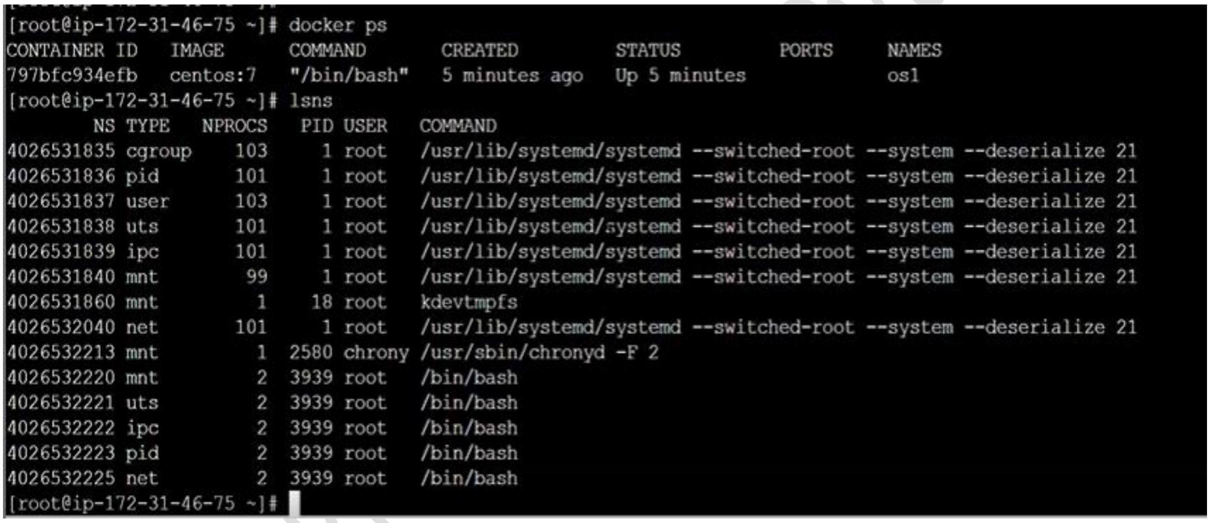
```
[root@ip-172-31-46-75 ~]# docker run -it --name os1 centos:7
[root@797bfc934efb /]#
[root@797bfc934efb /]#
[root@797bfc934efb /]#
[root@797bfc934efb /]#
[root@797bfc934efb /]# ps -aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  1.1  0.2  11844  2864 pts/0    Ss   16:05   0:00 /bin/bash
root          15  0.0  0.3  51748  3288 pts/0    R+   16:06   0:00 ps -aux
[root@797bfc934efb /]#
```

- Every process has a different hostname because on the namespace



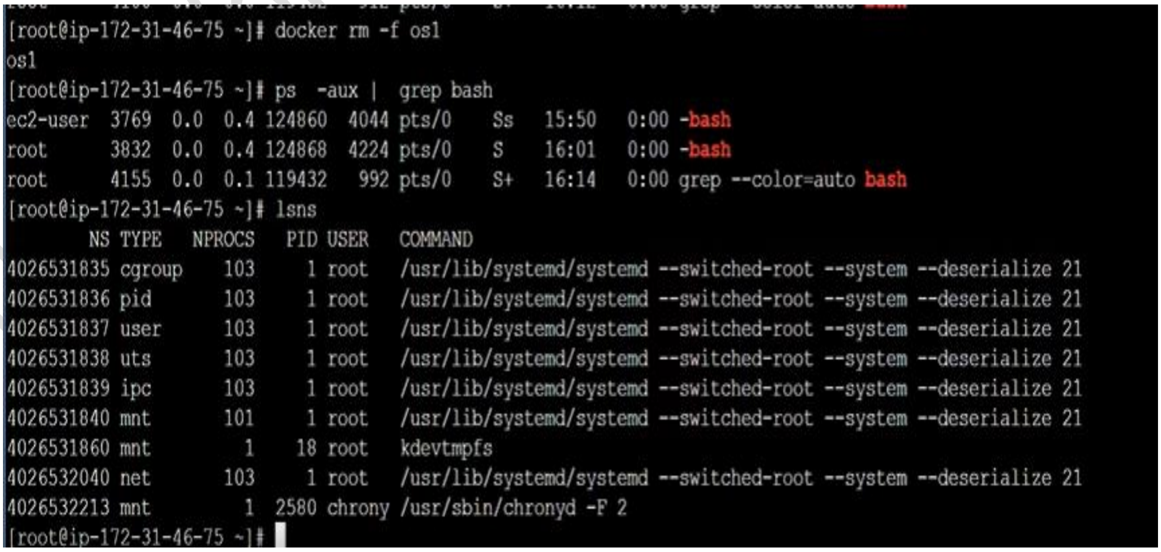
```
[root@ip-172-31-46-75 ~]# docker attach os1
[root@797bfc934efb /]# hostname
797bfc934efb
[root@797bfc934efb /]# read escape sequence
[root@ip-172-31-46-75 ~]#
[root@ip-172-31-46-75 ~]#
[root@ip-172-31-46-75 ~]# hostname
ip-172-31-46-75.ap-south-1.compute.internal
[root@ip-172-31-46-75 ~]#
```

- As soon as we launch a container from the docker run command it will create a separate namespace for the container or process
    - Command to list namespace:- **lsns**



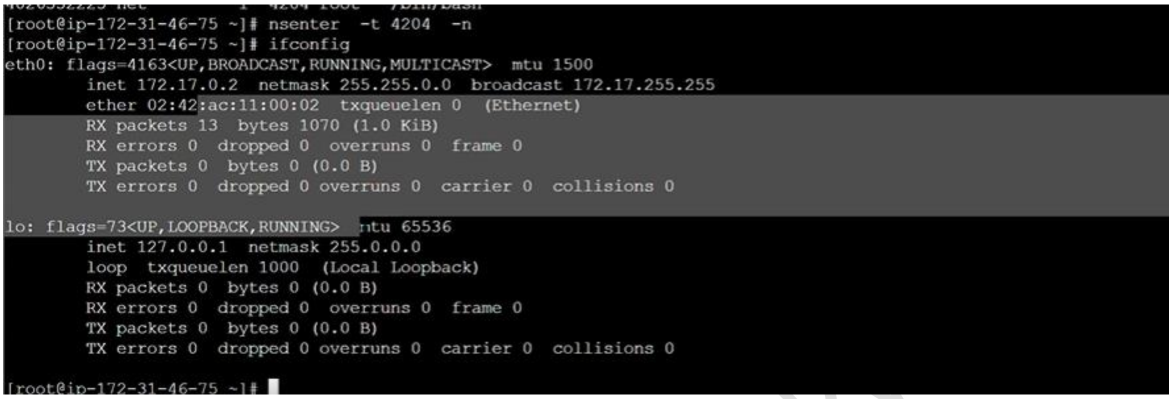
```
[root@ip-172-31-46-75 ~]# docker ps
CONTAINER ID   IMAGE      COMMAND       CREATED         STATUS         PORTS      NAMES
797bfc934efb   centos:7   "/bin/bash"   5 minutes ago   Up 5 minutes              os1
[root@ip-172-31-46-75 ~]# lsns
        NS TYPE   NPROCS   PID USER   COMMAND
4026531835 cgroup    103     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 21
4026531836 pid       101     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 21
4026531837 user      103     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 21
4026531838 uts       101     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 21
4026531839 ipc       101     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 21
4026531840 mnt        99     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 21
4026531860 mnt         1    18 root   kdevtmpfs
4026532040 net       101     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 21
4026532213 mnt         1  2580 chrony /usr/sbin/chronyd -F 2
4026532220 mnt         2  3939 root   /bin/bash
4026532221 uts         2  3939 root   /bin/bash
4026532222 ipc         2  3939 root   /bin/bash
4026532223 pid         2  3939 root   /bin/bash
4026532225 net         2  3939 root   /bin/bash
[root@ip-172-31-46-75 ~]#
```

- As soon as we stop the container it will remove all the namespaces



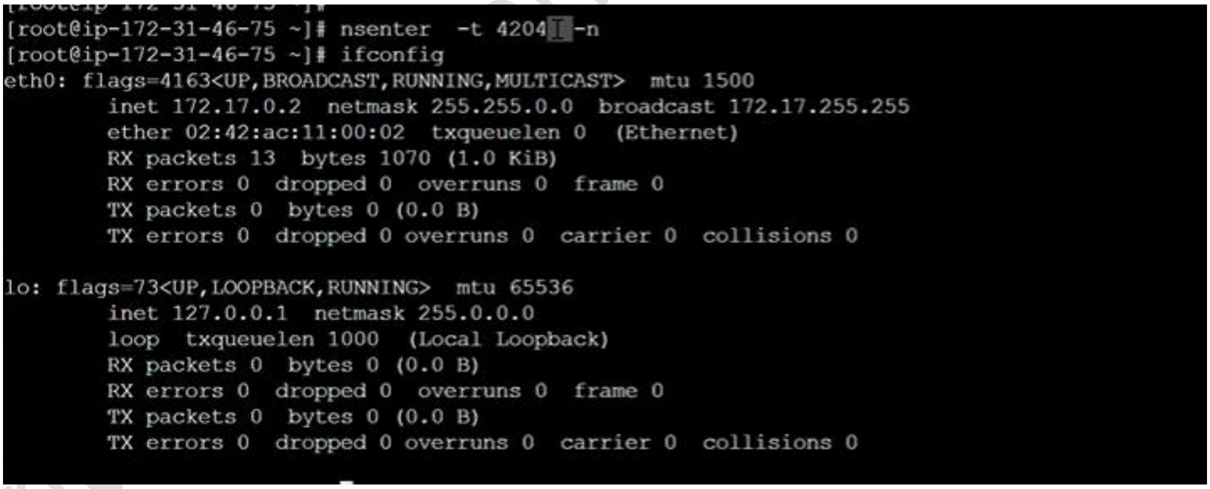
```
[root@ip-172-31-46-75 ~]# docker rm -f os1
os1
[root@ip-172-31-46-75 ~]# ps  -aux |  grep bash
ec2-user  3769  0.0  0.4 124860  4044 pts/0    Ss   15:50   0:00 -bash
root      3832  0.0  0.4 124868  4224 pts/0    S    16:01   0:00 -bash
root      4155  0.0  0.1 119432   992 pts/0    S+   16:14   0:00 grep --color=auto bash
[root@ip-172-31-46-75 ~]# lsns
        NS TYPE   NPROCS   PID USER   COMMAND
4026531835 cgroup    103     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 21
4026531836 pid       103     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 21
4026531837 user      103     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 21
4026531838 uts       103     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 21
4026531839 ipc       103     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 21
4026531840 mnt       101     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 21
4026531860 mnt         1    18 root   kdevtmpfs
4026532040 net       103     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 21
4026532213 mnt         1  2580 chrony /usr/sbin/chronyd -F 2
[root@ip-172-31-46-75 ~]#
```

- **nsenter** command is used for entering the namespace
    - Command:- **nsenter -t (PID) -n**
        - -t = target
        - -n = enter network namespace



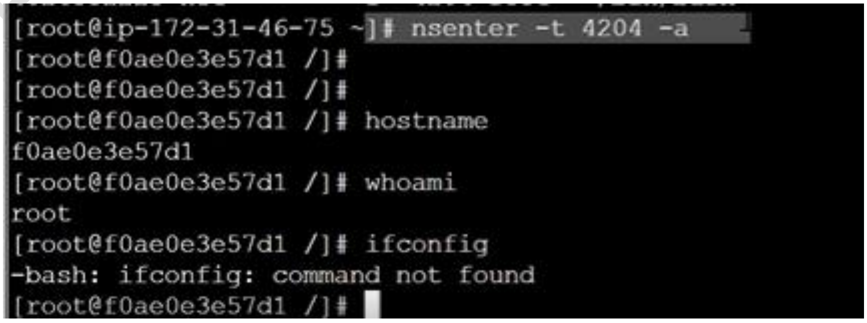
```
[root@ip-172-31-46-75 ~]# nsenter  -t 4204  -n
[root@ip-172-31-46-75 ~]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 13  bytes 1070 (1.0 KiB)
        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

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        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@ip-172-31-46-75 ~]#
```

- Linux operating system has the capability to give a network namespace & every network namespace we can give to different processes & those processes we are using with docker i.e. is the reason every docker container has a different network settings
- After we launch the container and run the ifconfig command same thing we can see the **nsenter** command also



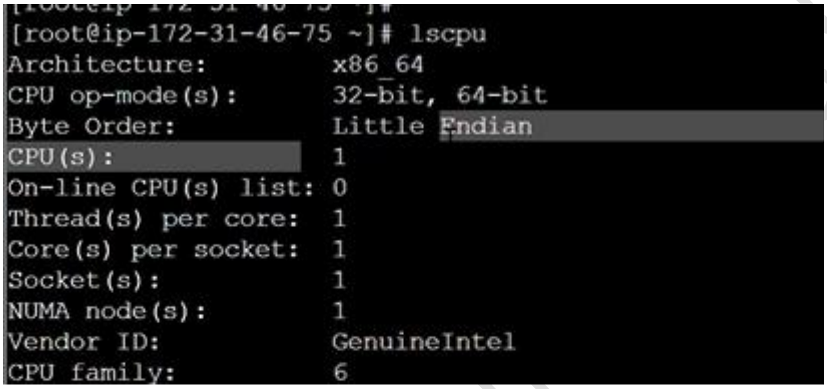
```
[root@ip-172-31-46-75 ~]# nsenter  -t 4204 -n
[root@ip-172-31-46-75 ~]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 13  bytes 1070 (1.0 KiB)
        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

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        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
```

- **-a** keyword in the **nsenter** command is used to enter all the namespaces
    - Command:- **nsenter -t (PID) -a**



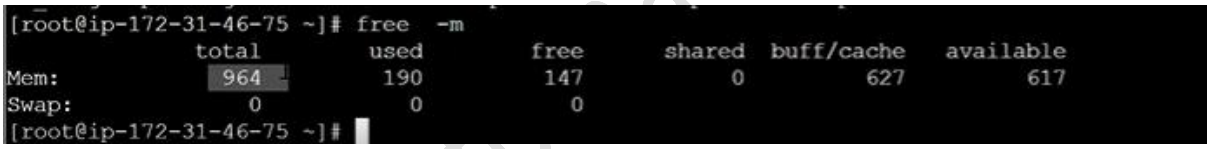
```
[root@ip-172-31-46-75 ~]# nsenter -t 4204 -a
[root@f0ae0e3e57d1 /]#
[root@f0ae0e3e57d1 /]#
[root@f0ae0e3e57d1 /]# hostname
f0ae0e3e57d1
[root@f0ae0e3e57d1 /]# whoami
root
[root@f0ae0e3e57d1 /]# ifconfig
-bash: ifconfig: command not found
[root@f0ae0e3e57d1 /]#
```

- It means behind the scene in the docker attach command it is running **nsenter -t (PID) -a** command only
- If in one operating system we launch multiple processes all the processes by default share underline hardware resources
- Whenever we run the container whatever hardware resources we have on the base system are shared with the container
  - On the base system
    - Command to list CPU:- **lscpu**



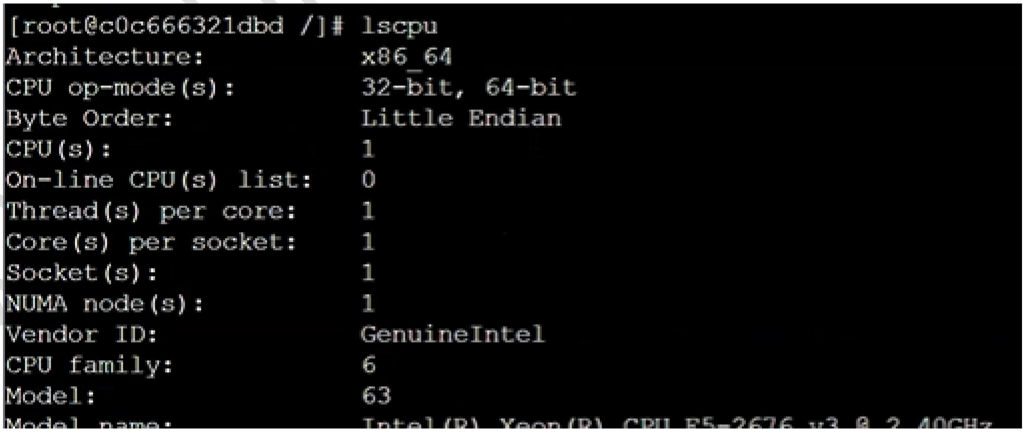
```
[root@ip-172-31-46-75 ~]# lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                1
On-line CPU(s) list:   0
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
```

    - Command to see ram:- **free -m**



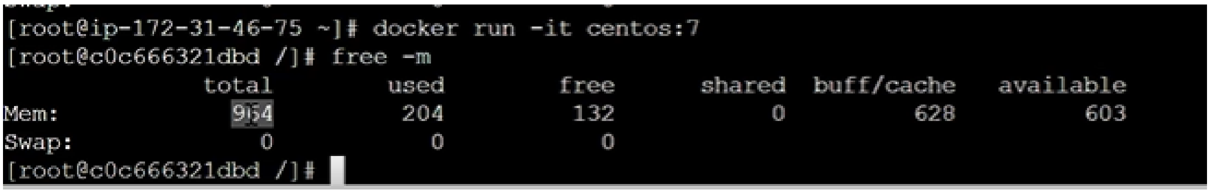
```
[root@ip-172-31-46-75 ~]# free  -m
              total        used        free      shared  buff/cache   available
Mem:            964         190         147           0         627         617
Swap:             0           0           0
[root@ip-172-31-46-75 ~]# 
```

  - Inside container
    - Command to list CPU:- **lscpu**



```
[root@c0c666321dbd /]# lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                1
On-line CPU(s) list:   0
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 63
Model name:            Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz
```

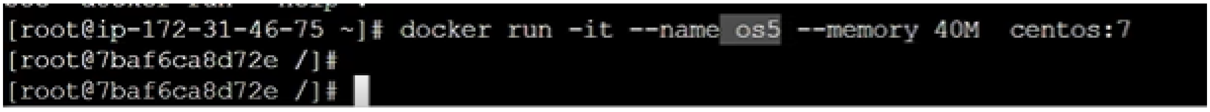
    - Command to see ram:- **free -m**



```
[root@ip-172-31-46-75 ~]# docker run -it centos:7
[root@c0c666321dbd /]# free -m
              total        used        free      shared  buff/cache   available
Mem:            964         204         132           0         628         603
Swap:             0           0           0
[root@c0c666321dbd /]# 
```

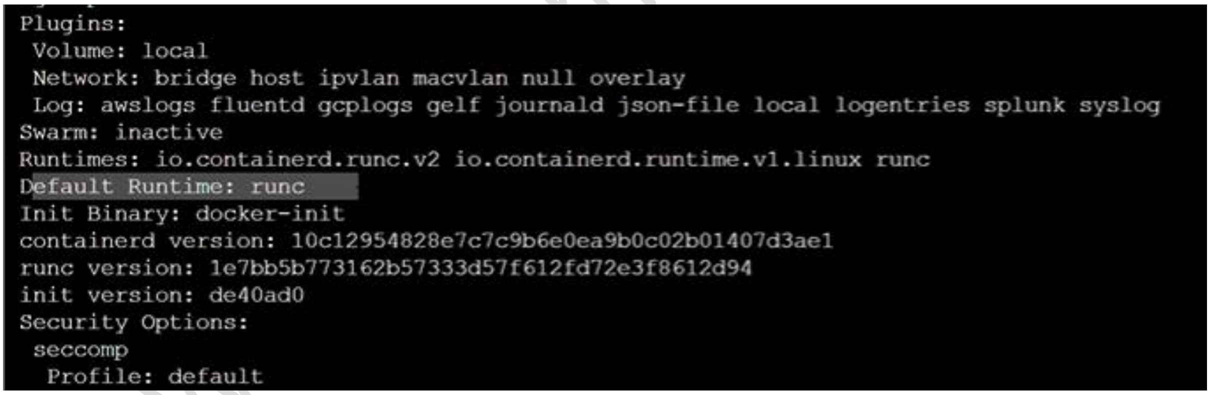
- Namespace does not work for hardware resources like CPU, RAM & HD
- With the help of **cgroup,** we can restrict our resources with the process

- With the help of the **memory** keyword in the run command, we can limit the memory for the docker container



```
[root@ip-172-31-46-75 ~]# docker run -it --name os5 --memory 40M  centos:7
[root@7baf6ca8d72e /]#
[root@7baf6ca8d72e /]#
```

- Whenever we run the **docker run -it ubuntu14.04** command
  - It will start the process bash and give a personal namespace & we can see the namespace with the **lsns** command
  - -it means after the process start take us inside the namespace there we can use the **nsenter** command
  - Because docker has its own network namespace we can see different Ip addresses inside it
  - The container has its own / drive and the data inside is coming from the image
  - Entire hardware or resources inside the container is coming from a base operating system but we can restrict it with the help of **cgroup**
- Behind the scene, the containers is launched by the container run time program called **runc**
  - Command:- **docker info**



```
Plugins:
 Volume: local
 Network: bridge host ipvlan macvlan null overlay
 Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
Swarm: inactive
Runtimes: io.containerd.runc.v2 io.containerd.runtime.v1.linux runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 10c12954828e7c7c9b6e0ea9b0c02b01407d3ae1
runc version: 1e7bb5b773162b57333d57f612fd72e3f8612d94
init version: de40ad0
Security Options:
 seccomp
  Profile: default
```