**Abstract:**

Containers are an operating system virtualization technology used to package applications and their dependencies and run them in isolated environments. They're about bundling up dependencies so we can ship code around in a repeatable, secure way. At a lower level, containers use three features of the Linux kernel, namely Namespaces, Control groups and Layered Filesystems. Containers are widely used due to their increased portability, greater efficiency, better security and less overhead when compared to traditional virtual machines.

**Features:**

1. New namespaces
    a. Mount - isolates the mount points from host
    b. PID - isolates process_id number space
    c. UTS - isolates the host name and the domain name
2. Control groups
    a. Create a new cgroup file for our container
    b. Limit the maximum processes the container can run
3. Layered Filesystems
    a. Provide a new isolated root file system for the container with all the dependencies and binaries from the host
    b. Allows to create and manage directories and files

**TECH STACK:**

- Programming Language : Golang
- Environment: WSL

**IMPLEMENTATION:**

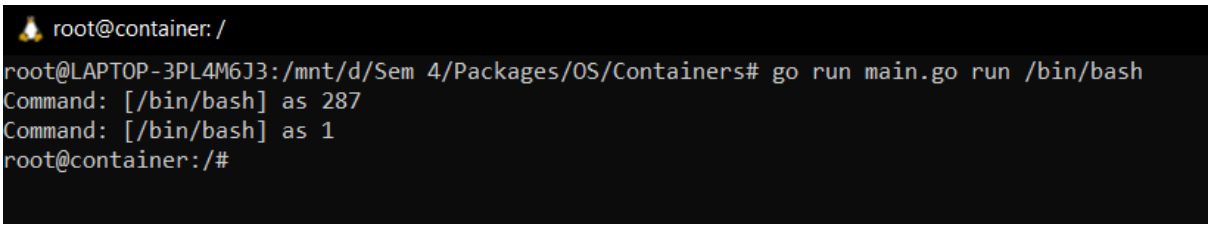
In Linux, there are a set of system calls that enable creating, joining, and discovering namespace.

- clone: create a new process with new namespace
- unshare: create and move the current process to a new namespace

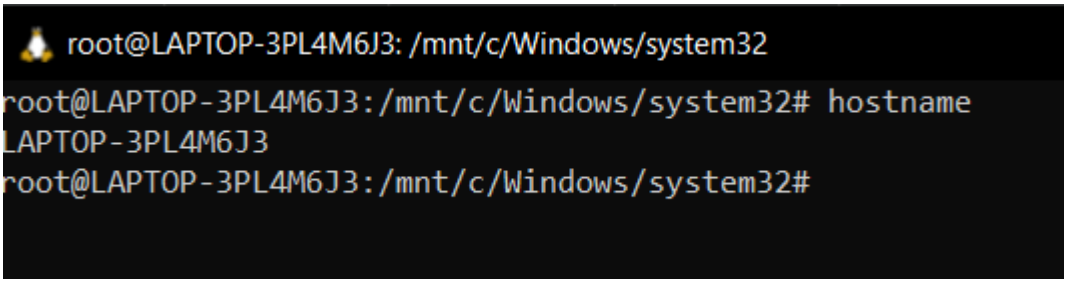
Golang provides a sophisticated package “syscall” to interact with these system calls. Each system call mentioned above needs to be specified with flags to specify the namespace you want.

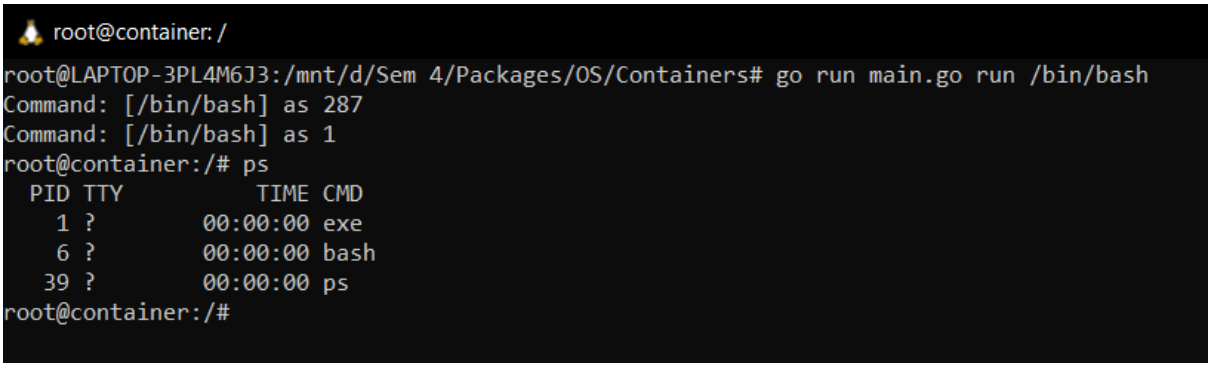
- CLONE_NEWUTS (Unix Time Sharing System - give new hostname
- CLONE_NEWPID (give pid starting from 1)
- CLONE_NEWNS (mount)
- CLONE_NEWIPC (IPC)
- CLONE_NEWNET (Network)

This project creates a new process with UTS, MNT, NS, PID namespaces.



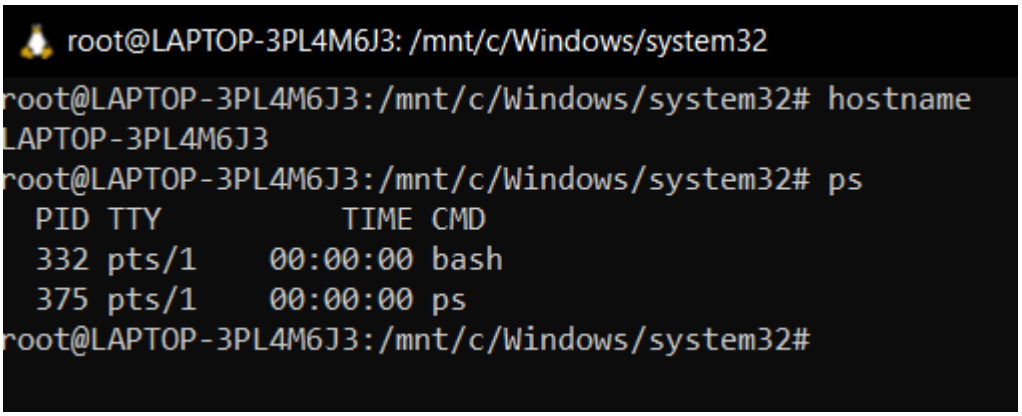

We can see that the hostname of the container is “container”, while that of host is



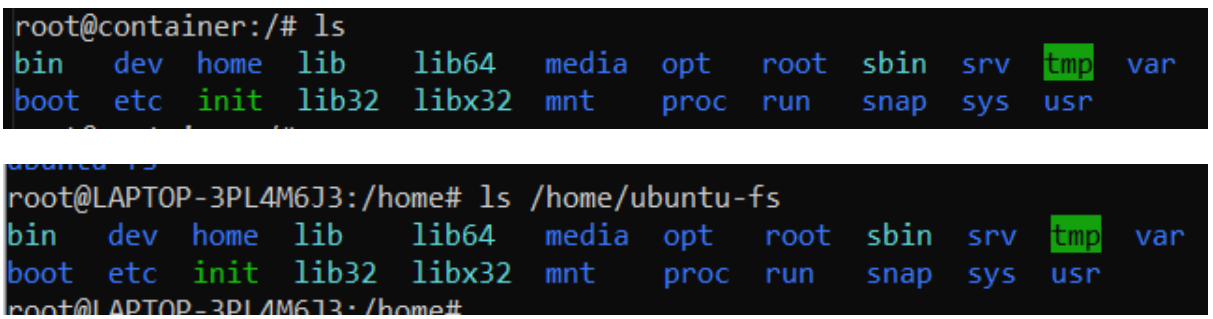



The pids of processes running inside the container are isolated from the host processes. The /proc directory has been mounted to the container so as to give an illusion the processes running inside the containers that they are isolated.

ps command in host:




The container has been provided with a copy of the host filesystem to implement the Layered FileSystems.




To check whether kernel has assigned the root to the process, make the shell to sleep and find the respective pid from host,

```
root@LAPTOP-3PL4M6J3:/mnt/c/Windows/system32# ps
  PID TTY          TIME CMD
  332 pts/1    00:00:00 bash
  375 pts/1    00:00:00 ps
root@LAPTOP-3PL4M6J3:/mnt/c/Windows/system32# ps -C sleep
  PID TTY          TIME CMD
  377 pts/0    00:00:00 sleep
root@LAPTOP-3PL4M6J3:/mnt/c/Windows/system32# ls-l/proc/377/root
-bash: ls-l/proc/377/root: No such file or directory
root@LAPTOP-3PL4M6J3:/mnt/c/Windows/system32# ls -l /proc/377/root
lrwxrwxrwx 1 root root 0 Jun 21 01:41 /proc/377/root -> /home/ubuntu-fs
```

To limit the resource your container can use, we manipulate the cgroup file in /sys/fs/cgroup.

A view of the cgroup folder from host:

```
root@LAPTOP-3PL4M6J3:/home# cd /sys/fs/cgroup
root@LAPTOP-3PL4M6J3:/sys/fs/cgroup# ls
blkio  cpu  cpuacct  cpuset  devices  freezer  hugetlb  memory  net_cls  net_prio  perf_event  pids  rdma  unified
root@LAPTOP-3PL4M6J3:/sys/fs/cgroup# ls memory
ash                     memory.kmem.limit_in_bytes          memory.max_usage_in_bytes        memory.soft_limit_in_bytes
cgroup.clone_children   memory.kmem.max_usage_in_bytes      memory.memsw.failcnt             memory.stat
cgroup.event_control    memory.kmem.tcp.failcnt             memory.memsw.limit_in_bytes      memory.swappiness
cgroup.procs            memory.kmem.tcp.limit_in_bytes      memory.memsw.max_usage_in_bytes  memory.usage_in_bytes
cgroup.sane_behavior    memory.kmem.tcp.max_usage_in_bytes  memory.memsw.usage_in_bytes      memory.use_hierarchy
memory.failcnt          memory.kmem.tcp.usage_in_bytes      memory.move_charge_at_immigrate  notify_on_release
memory.force_empty      memory.kmem.usage_in_bytes          memory.oom_control               release_agent
memory.kmem.failcnt     memory.limit_in_bytes               memory.pressure_level            tasks
root@LAPTOP-3PL4M6J3:/sys/fs/cgroup# ls cpu
ash                     cgroup.procs          cpu.cfs_period_us  cpu.rt_period_us   cpu.shares  notify_on_release  tasks
cgroup.clone_children   cgroup.sane_behavior  cpu.cfs_quota_us   cpu.rt_runtime_us  cpu.stat    release_agent
root@LAPTOP-3PL4M6J3:/sys/fs/cgroup#
```

The “ash” folder is set aside for the container. It contains the flags that were set by us to specify the kernel to limit specific resources such as:

- Limiting maximum number of process (pids.max)
- Limit memory limit (memory.limit_in_bytes)
- Limit memory swap limit (memory.memsw.limit_in_bytes)
- Limit cpu access time (cpu.cfs_quota_us) and how regularly a cgroup's access to CPU resources should be reallocated (cpu.cfs_period_us)

```
root@LAPTOP-3PL4M6J3:/sys/fs/cgroup/pids# cd ash
root@LAPTOP-3PL4M6J3:/sys/fs/cgroup/pids/ash# cat pids.max
20
```

Here, the maximum number of processes is set to 20.

To check this, we fork bomb our container and ensure the container cannot create more than 20 processes.

```
root@container:/# :() { : | : & }; :
[1] 43
root@container:/# bash: fork: retry: Resource temporarily unavailable
bash: fork: retry: Resource temporarily unavailable
bash: fork: retry: Resource temporarily unavailable
bash: fork: retry: Resource temporarily unavailable
bash: fork: retry: Resource temporarily unavailable
bash: fork: retry: Resource temporarily unavailable
bash: fork: retry: Resource temporarily unavailable
```

As only 20 processes were assigned, recursive call of the :() function cannot be done.

```
root@LAPTOP-3PL4M6J3:/sys/fs/cgroup/memory# cd ash
root@LAPTOP-3PL4M6J3:/sys/fs/cgroup/memory/ash# cat memory.limit_in_bytes
524288000
root@LAPTOP-3PL4M6J3:/sys/fs/cgroup/memory/ash# cat memory.memsw.usage_in_bytes
20602880
root@LAPTOP-3PL4M6J3:/sys/fs/cgroup/memory/ash# cat memory.memsw.limit_in_bytes
524288000
```

```
root@LAPTOP-3PL4M6J3:/sys/fs/cgroup/cpu/ash# cat cpu.cfs_quota_us
2000000
root@LAPTOP-3PL4M6J3:/sys/fs/cgroup/cpu/ash# cat cpu.cfs_period_us
1000000
```