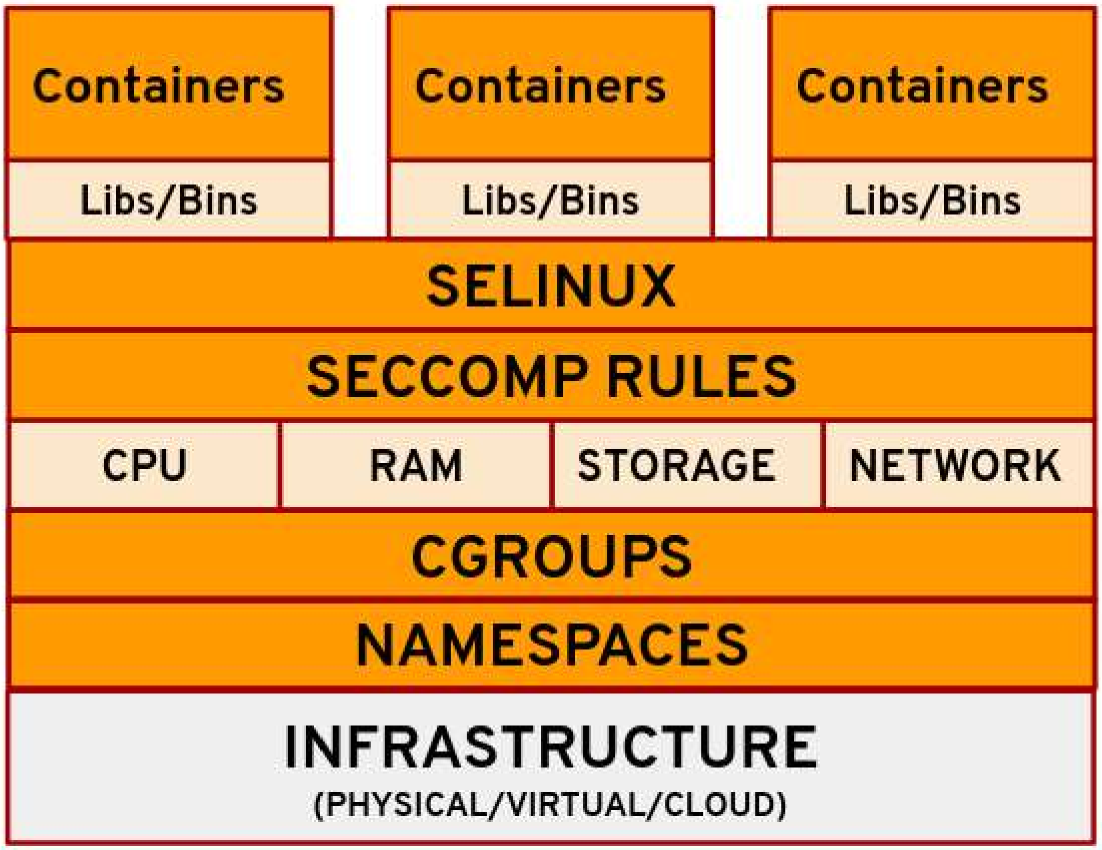
Published in techbeatly

Nived Velayudhan Follow
Jul 13, 2021 · 5 min read · Listen

Save

# CONTAINER INTERNALS — Deep Dive



Linux technologies make up the foundations of building/running a container process in your system. Technologies like:

3. Seccomp

4. SELinux

## Namespaces:

Namespaces provide a layer of isolation f of what appears to be its own Linux filesystem. This would limit as to what a process can see and therefore restrict the amount of resources available to this process.

There are several namespaces in the Linux kernel that are used by docker while creating a container:

```
[nivedv@homelab ~]$ docker container run alpine ping 8.8.8.8
[nivedv@homelab ~]$ sudo lsns -p 29413

        NS TYPE   NPROCS   PID USER COMMAND
4026531835 cgroup    299     1 root /usr/lib/systemd/systemd --
switched...
4026531837 user      278     1 root /usr/lib/systemd/systemd --
switched...
4026533105 mnt         1 29413 root ping 8.8.8.8
4026533106 uts         1 29413 root ping 8.8.8.8
4026533107 ipc         1 29413 root ping 8.8.8.8
4026533108 pid         1 29413 root ping 8.8.8.8
4026533110 net         1 29413 root ping 8.8.8.8
```

- USER: This is used to isolate users and groups within a container. This is done by allowing containers to have a different view of UID and GID ranges as compared to the host system. This allows the software to run inside the container as the root user, but if a hacker is able to attack the container and then escape to the host machine, it will only have a non-root identity.
- MNT: This namespace allows the containers to have their own view of its file system hierarchy on the system. You can find the mount points for each container process in the /proc/<PID>/mounts location in your Linux system.

random ID is used as the hostname e
the unshare command to get an idea

```
nivedv@homelab ~]$ docker contai

/ # hostname
9c9a5edabdd6
/ #

nivedv@homelab ~]$ sudo unshare -u sh

sh-5.0# hostname isolated.hostname
sh-5.0# hostname
isolated.hostname
sh-5.0#
sh-5.0# exit
exit
[nivedv@homelab ~]$ hostname
homelab.redhat.com
```

- IPC: Inter-Process Communication namespace makes it possible for different container processes to communicate with each other by giving them access to a shared range of memory or by using a shared message queue.

```
[root@demo /]# ipcmk -M 10M
Shared memory id: 0
[root@demo /]# ipcmk -M 20M
Shared memory id: 1
[root@demo /]#
[root@demo /]# ipcs

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch
status
0xd1df416a 0          root       644        10485760   0
```

- PID: The process ID namespace is res inside a container are isolated from t inside a container, you only see the p the host machine because of this namespace.
- NET: The network namespace allows the container to have its own view of network interface, IP addresses, routing tables, port numbers, etc. How does a container able to communicate to the external world? All containers you create get attached to the master — docker0 interface.

```
[nivedv@homelab ~]$ docker container run --rm -it alpine sh
/ # ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=119 time=21.643 ms
64 bytes from 8.8.8.8: seq=1 ttl=119 time=20.940 ms
^C

[root@homelab ~]# ip link show veth84ea6fc

veth84ea6fc@if22: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
noqueue

master docker0 state UP mode DEFAULT group default
```

Control groups ( cgroups ):

Cgroups are fundamental blocks of making a container. It is responsible to allocate and limit the resources, such as CPU, memory, Network I/O, that are used by containers. The Container Engine automatically creates cgroup filesystem of each type.

```
[root@homelab ~]# lscgroup | grep docker

cpuset:/docker
```

```
memory:/docker
perf_event:/docker
blkio:/docker
pids:/docker
```

The Container Runtime sets up the cgroups values for each container when the container is run and all information is stored in /sys/fs/cgroup/*/docker. The following command will ensure that the container can use 50,000 microseconds of CPU time, and set up the soft and hard limits of memory to 500M and 1G respectively.

```
[root@homelab ~]# docker container run -d --name test-cgroups --cpus
0.5 --memory 1G --memory-reservation 500M httpd

[root@homelab ~]# lscgroup cpu,cpuacct:/docker memory:/docker
cpu,cpuacct:/docker/
cpu,cpuacct:/docker/c3503ac704dafea3522d3bb82c77faff840018e857a2a7f669
065f05c8b2cc84
memory:/docker/
memory:/docker/c3503ac704dafea3522d3bb82c77faff840018e857a2a7f669065f0
5c8b2cc84

[root@homelab c....c84]# cat cpu.cfs_period_us
100000
[root@homelab c....c84]# cat cpu.cfs_quota_us
50000

[root@homelab c....c84]# cat memory.soft_limit_in_bytes
524288000
[root@homelab c....c84]# cat memory.limit_in_bytes
1073741824
```

## SECCOMP:

Seccomp basically stands for Secure computing. It is a Linux feature that is used to restrict the set of system calls that an application is allowed to make. The default seccomp profile of docker disables around 44 syscalls out of the 300+.

to make changes to the kernel modules s

*delete_module* syscalls.

## SELINUX:

SELinux stands for security-enhanced Lin

your hosts, then SELinux is enabled by default. SELinux lets you limit an application to have access only to its own files and prevent any other processes from being able to access them. So, if an application is compromised, it would limit the number of files that it can affect or control. It does this by setting up contexts for files and processes and by defining policies that would enforce what a process is able to see and make changes to.

SELinux policies for containers are defined by the container-selinux package. By default, containers are run with the container_t label and are allowed to read & execute under the */usr* directory and read most content from */etc* directory. The files under */var/lib/docker* and */var/lib/containers* have the label container_var_lib_t.