Linux Capabilities

Network Namespaces Commands:

- Creating and Listing Network Namespaces: `ip netns add <new namespace name>`
- Network namespace verification: `ip netns list`
- Assign Interfaces to Network Namespaces: `ip link add <veth name> type veth peer name <other veth name>`
- Veth verification: `ip link list`
- Connect interface to namespace: `ip link set <veth name> netns <namespace name>`
- Configure interface in network namespaces: `ip netns exec <network namespace> <command to run against that namespace>`

  `Example: ip netns exec blue ip addr add 10.1.1.1/24 dev veth1`

More information on these commands here:
https://blog.scottlowe.org/2013/09/04/introducing-linux-network-namespaces/

Privileged Containers Commands in Docker:

- Check if container is in privileged mode: `docker inspect --format='{{.HostConfig.Privileged}}' [container_id]`
- Run Docker in privileged mode: `sudo docker run --privileged [image_name]`
- Run an ubuntu Container: `sudo docker run -it --privileged ubuntu`
- Test whether container has access to host: `mount -t tmpfs none /mnt`
- List disk space: `df -h`

Containers:

To add or remove Linux capabilities for a Container, include the capabilities field in the securityContext section of the Container manifest.

Yaml file example of adding capabilities:

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo-4
spec:
  containers:
  - name: sec-ctx-4
    image: gcr.io/google-samples/node-hello:1.0
    securityContext:
      capabilities:
        add: ["NET_ADMIN", "SYS_TIME"]
```

Yaml File Example of dropping capabilities:

```
spec:
  containers:
  - image: mateobur/flask
    name: flask-cap
    securityContext:
      capabilities:
        drop:
          - NET_RAW
          - CHOWN
```

Default Containers Capabilities:

- `CHOWN` - Make arbitrary changes to file UIDs and GIDs
- `DAC_OVERRIDE` - Discretionary access control (DAC) - Bypass file read, write, and execute permission checks.
- `FSETID` - Don't clear set-user-ID and set-group-ID mode bits when a file is modified; set the set-group-ID bit for a file whose GID does not match the file system or any of the supplementary GIDs of the calling process.
- `FOWNER` - Bypass permission checks on operations that normally require the file system UID of the process to match the UID of the file, excluding those operations covered by `CAP_DAC_OVERRIDE` and `CAP_DAC_READ_SEARCH`.
- `MKNOD` - Create special files using mknod(2).
- `NET_RAW` - Use RAW and PACKET sockets; bind to any address for transparent proxying.
- `SETGID` - Make arbitrary manipulations of process GIDs and supplementary GID list; forge GID when passing socket credentials via UNIX domain sockets; write a group ID mapping in a user namespace.
- `SETUID` - Make arbitrary manipulations of process UIDs; forge UID when passing socket credentials via UNIX domain sockets; write a user ID mapping in a user namespace.
- `SETFCAP` - Set file capabilities.
- `SETPCAP` - If file capabilities are not supported: grant or remove any capability in the caller's permitted capability set to or from any other process.
- `NET_BIND_SERVICE` - Bind a socket to Internet domain privileged ports (port numbers less than 1024).
- `SYS_CHROOT` - Use chroot(2) to change to a different root directory.
- `KILL` - Bypass permission checks for sending signals. This includes use of the ioctl(2) `KDSIGACCEPT` operation.
- `AUDIT_WRITE` - Write records to kernel auditing log.

Proc Mount Types:

- The `/proc` directory contains virtual files that are windows into the current state of the running Linux kernel.

SELinux:

To assign SELinux labels to a Container, include the seLinuxOptions field in the securityContext section of your Pod or Container manifest. The seLinuxOptions field is an SELinuxOptions object. Here's an example that applies an SELinux level:

Yaml File Example:

```
...
securityContext:
  seLinuxOptions:
    level: "s0:c123,c456"
```