Namespaces

Namespaces

- namespaces of the current context
  `ls -l /proc/self/ns` or `ls -l /proc/$$/ns`
  - `cgroup` control / limitation of resources
  - `ipc` hmessage queues, semaphores, shared mem for ipc
  - `mnt` mount
  - `net` network
  - `pid, pid_for_children` process id
  - `user` user (UIDs, GIDs)
  - `uts` hostname & domainname
  - `time` time

Network Namespaces

- provide the isolation of networking related resources (see `man network_namespaces.7`)
- network devices,
- IPv4 and IPv6 protocol stacks,
- IP routing tables, firewall rules,
- port numbers (sockets)
- some filesystem entities
  - `/proc/net` directory (a symbolic link to `/proc/PID/net`),
  - `/sys/class/net` directory,
  - various files under `/proc/sys/net`,

Network Namespaces & Device Configuration

- some linux commands in network namespace context
  - `ip netns list`
  - `ip netns add <netns_name>` or
    `ip netns delete <netns_name>`
  - `ip netns exec <netns_name> <command>`

  - `ip link show` (or `ifconfig` or `ifconfig -a`)
  - `ip link add <veth_name1> type veth peer name <veth_name2>`
  - `ip link set <veth_name> netns <netns_name>`
  - `ip link set dev <veth_name> up` (or `down`)
  - `ip addr add <ip_addr>/<net_bits> dev <eth_name>`

Network Namespaces & Device Configuration

- some linux commands in bridging context
  - `brctl show`
  - `brctl addbr <br_name>`
  - `brctl stp <br_name> on`
  - `brctl setageing <br_name> <time>` (0 ↔ off)
  - `ip link set dev <br_name> up`
  - `ip neigh {show|add|del| ...}`
  - (old: `arp -nv , arp -nd <ip_addr>`)

  - `vconfig add <vlan_name> <vlan_no>`
  - `ip link set dev <vlan_name.vlan_no> up` (or `down`)

Namespaces

- namespaces of the current context
  `ls -l /proc/self/ns` or `ls -l /proc/$$/ns`
  - `cgroup` control / limitation of resources
  - `ipc` hmessage queues, semaphores, shared mem for ipc
  - `mnt` mount
  - `net` network
  - `pid, pid_for_children` process id
  - `user` user (UIDs, GIDs)
  - `uts` hostname & domainname
  - `time` time

Mount Namespaces

- some linux commands in mount namespace context
  - `findmnt` display mountpoints
  - `sudo find /proc/*/ns -name 'mnt' -exec readlink {} \; | sort -u` find all mnt namespaces
  - `unshare -m [<prog>]` create a new mnt namespace and run `<prog>` within (or run `$SHELL` by default)
  - `mount --bind d1 d2` make the same content of `d1` available at the mountpoint `d2`.
    Works for files too (file `d2` already has to exist).
    Recursive operation for submounts by `--rbind`.
    Make a directory a mountpoint by `mount --bind dir dir`
    Property change by remounting, e.g. make `d2` read only while `d1` is still writeable by
    `mount -o remount,bind,ro d1 d2`
  - `lsns` display namespaces and processes

Mount Namespaces

- playing around: creating / deleting
  - list all mnt namespaces
  - create a new one by issuing `(sudo) unshare -m`
  - again list all mnt namespaces and observe the newly created
  - exit the shell
  - again list all mnt namespaces and observe that the newly created one has vanished

Mount Namespaces

- playing around: isolated fs
  - create a shell in a new mnt namespace `sudo unshare -m`
  - observe existing mountpoints from within the new ns and from outside using `findmnt` → the same
  - create a new temporary filesystem within the namespace
    ```
    mkdir r1
    mount -t tmpfs tmpfs r1
    ```
  - again observe existing mountpoints from within the new ns and from outside using `findmnt` → the `r1` mountpoint is only available in the the new ns.
  - create a file in the shared filesystem and observe its ownership from within the new ns and from outside
    → both root
  - repeat the experiment creating a new mnt namespace by `unshare -Urm` and observe the file ownership → user / root

Mount Namespaces

- playing around: isolated root fs
  - get the miniroot filesystem from `https://alpinelinux.org/downloads/`
  - create the location for the mnt ns new root fs and prepare it

    ```
    mkdir my_root
    tar -xf alpine...tar.gz -C my_root
    ```

  - create the new mnt ns and make the future root fs a mountpoint (by the way create a user ns `(-U`) and map `root` within the ns to `vagrant` outside `(-r)`)

    ```
    unshare -Urm    (now we are in our new mnt ns)
    mount --bind my_root my_root
    ```

Mount Namespaces

- … playing around: isolated root fs
  - make a directory `old_root` to map the current root fs to
    ```
    cd my_root && mkdir old_root
    ```
  - change the shell to `sh` (alpine does not provide `bash`, so after the root fs change `bash` is no longer available)
    ```
    sh
    ```
  - exchange the old and the new root fs .. that's it!
    ```
    pivot_root . old_root
    cd /
    ```
  - clean up (dissolve the connection to the outer fs)
    ```
    umount -l old_root
    ```
  - now in the mnt ns we have an miniature system without access other parts of the VMs filesystem and with its own executables (based on busybox). From the outer VM we can look into `my_root`, where mnt ns' `/` is living in.

Mount Namespaces

- playing around: isolated overlay fs
  - create the new mnt ns
    ```
    unshare -m
    ```
  - create a directory `over_dir` to hold the contents to lay over lowerdir, and a working directory `over_work`
    ```
    mkdir over_dir && mkdir over_work
    ```
  - create the overlay
    ```
    mount -t overlay -o lowerdir=/etc,\
      upperdir=./over_dir,workdir=./over_work \
      overlayfs /etc
    ```
  - from within the mnt ns writes to `/etc` effectively go to `over_dir`, and vive versa
  - thanks to the mnt ns changes within the mnt ns have no effect on the outside (`/etc` remains unaltered). Outside changes to the (underlaying) `/etc` are visible within the mnt ns too.

Namespaces

- some (more) useful commands
  - `(sudo) lsns -t net`
    display the (network) namespaces and (only) the 1st process under each namespace
  - `(sudo) ps -e -o netns,pid,cmd --sort netns`
    display processes under a (network) namespace
  - `(sudo) nsenter -t <pid> -a <cmd>`
    execute `<cmd>` (default is `$SHELL`) under all (`-a`) the same namespaces as the target process with `<pid>`
  - `(sudo) ip link add veth1` **`netns <name1|pid1>`** `\`
    `type veth peer name veth2` **`netns <name2|pid2>`**
    add a virtual ethernet connection's endpoints directly into the namespaces identified by their names `<name1>` and `<name2>` or the processes' `<pid1>` and `<pid2>` that belong to them.