Containers

Kernel namespaces

Definitions:

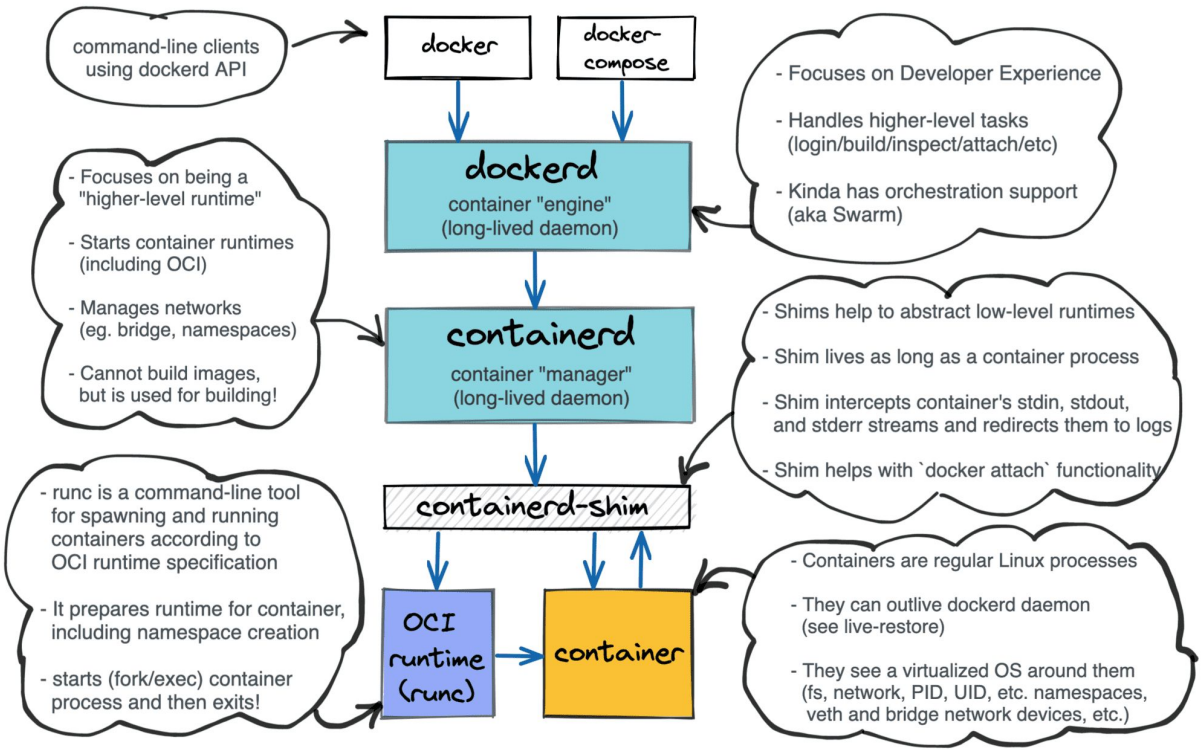
A **container** is the runtime instantiation of a *Container Image*. A container is a standard Linux process typically created through a clone() system call instead of fork() or execvp(). Also, containers are often isolated further through the use of cgroups, SELinux or AppArmor.

A **container image**, in its simplest definition, is a file which is pulled down from a Registry Server and used locally as a mount point when starting Containers.

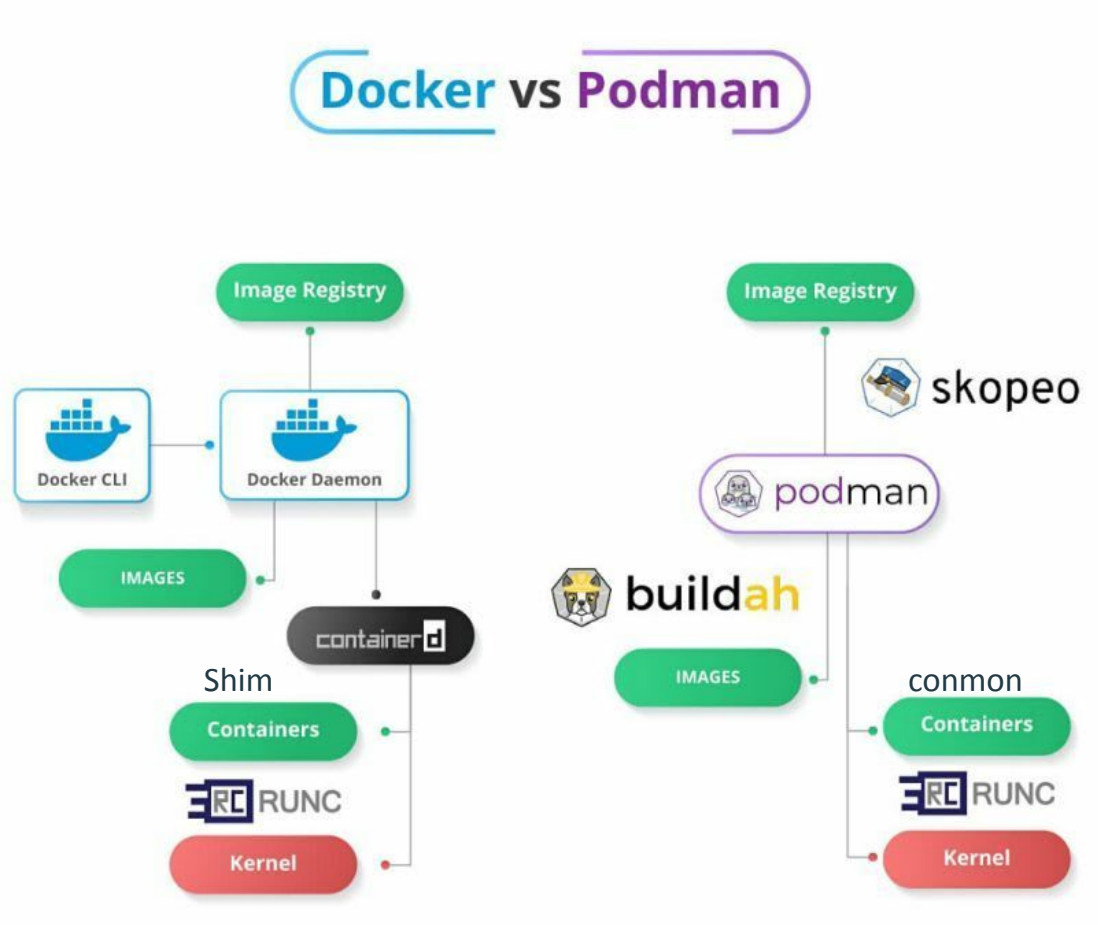
A **container engine** is a piece of software that accepts user requests, including command line options, pullsimages, and from the end user's perspective runs the container. There are many container engines, including docker, **podman**, RKT, **CRI-O[container runtime interface - OCI]**, LXD. Also, many cloud providers, Platforms as a Service (PaaS), and Container Platforms have their own built-in container engines which consume Docker or OCI compliant Container Images.

A **container runtime** a lower level component typically used in a Container Engine but can also be used by hand for testing. The Open Containers Initiative (OCI) Runtime Standard reference implementation is runc. This is the most widely used container runtime, but there are others OCI compliant runtimes, such as crun, railcar, and katacontainers. Docker, CRI-O, and many other Container Engines rely on runc.

Docker

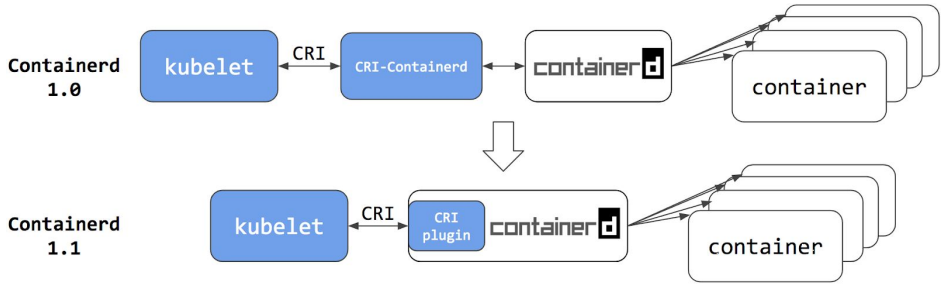

command-line clients using dockerd API
docker
docker-compose
- Focuses on Developer Experience
- Handles higher-level tasks (login/build/inspect/attach/etc)
- Kinda has orchestration support (aka Swarm)
dockerd
container "engine" (long-lived daemon)
- Focuses on being a "higher-level runtime"
- Starts container runtimes (including OCI)
- Manages networks (eg. bridge, namespaces)
- Cannot build images, but is used for building!
containerd
container "manager" (long-lived daemon)
- Shims help to abstract low-level runtimes
- Shim lives as long as a container process
- Shim intercepts container's stdin, stdout, and stderr streams and redirects them to logs
- Shim helps with `docker attach` functionality
containerd-shim
- runc is a command-line tool for spawning and running containers according to OCI runtime specification
- It prepares runtime for container, including namespace creation
- starts (fork/exec) container process and then exits!
OCI runtime (runc)
container
- Containers are regular Linux processes
- They can outlive dockerd daemon (see live-restore)
- They see a virtualized OS around them (fs, network, PID, UID, etc. namespaces, veth and bridge network devices, etc.)

Podman


Docker vs Podman
Image Registry
Docker CLI
Docker Daemon
IMAGES
containerd
Shim
Containers
RUNC
Kernel
Image Registry
skopeo
podman
buildah
IMAGES
conmon
Containers
RUNC
Kernel

Kubernetes?

OR



Today we look at containers

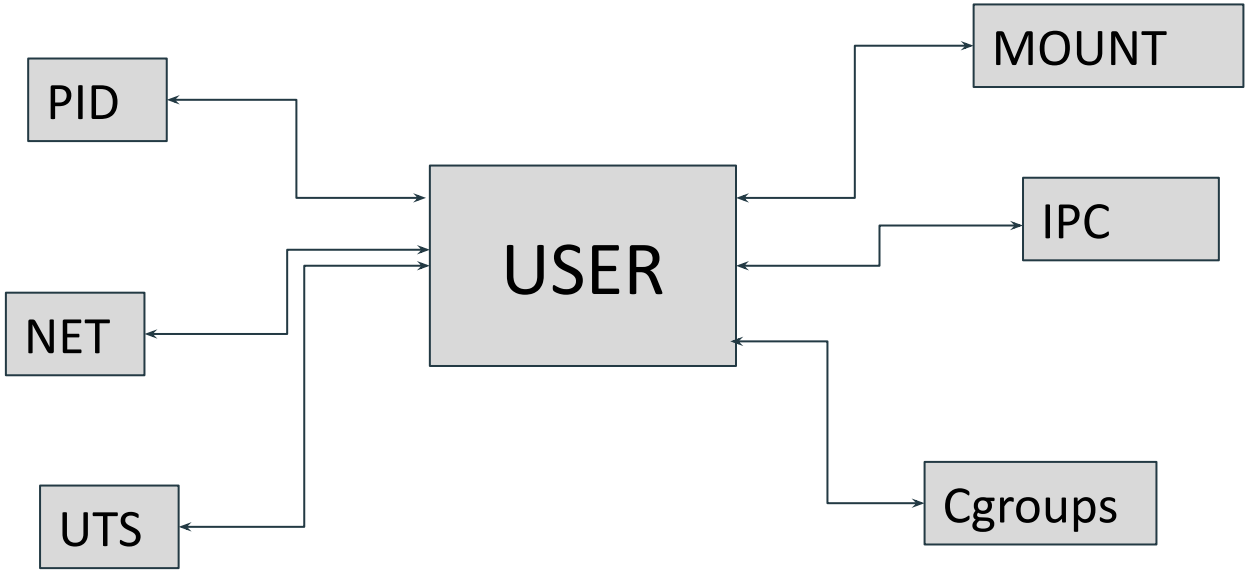
A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource. Changes to the global resource are visible to other processes that are members of the namespace, but are invisible to other processes. One use of namespaces is to implement containers.

man 7 namespaces

- resource compartmentalization / one set of process sees a set of resources

  IDEA! kind of resources are grouped in kernel namespace!!

Namespaces



Every process has of a set of namespaces!
But what can it do on them depends on the capabilities it has on that namespace

Capabilities

For the purpose of performing permission checks, traditional UNIX implementations distinguish two categories of processes: privileged processes (whose effective user ID is 0, referred to as superuser or root), and unprivileged processes (whose effective UID is nonzero). Privileged processes bypass all kernel permission checks, while unprivileged processes are subject to full permission checking based on the process's credentials (usually: effective UID, effective GID, and supplementary group list).

Starting with Linux 2.2 (1999), Linux divides the privileges traditionally associated with superuser into distinct units, known as capabilities, which can be independently enabled and disabled.

Capabilities are a per-thread attribute.

man capabilities

Setuid/Setgid

Capabilities

Why all this fuss about capabilities?

User namespace

User namespaces isolate security-related identifiers and attributes, in particular, user IDs and group IDs, the root directory, .., and capabilities. A process's user and group IDs can be different inside and outside a user namespace. In particular, a process can have a normal unprivileged user ID outside a user namespace while at the same time having a user ID of 0 inside the namespace; in other words, the process has full privileges for operations inside the user namespace, but is unprivileged for operations outside the namespace.

- User namespaces can be nested

man user_namespace

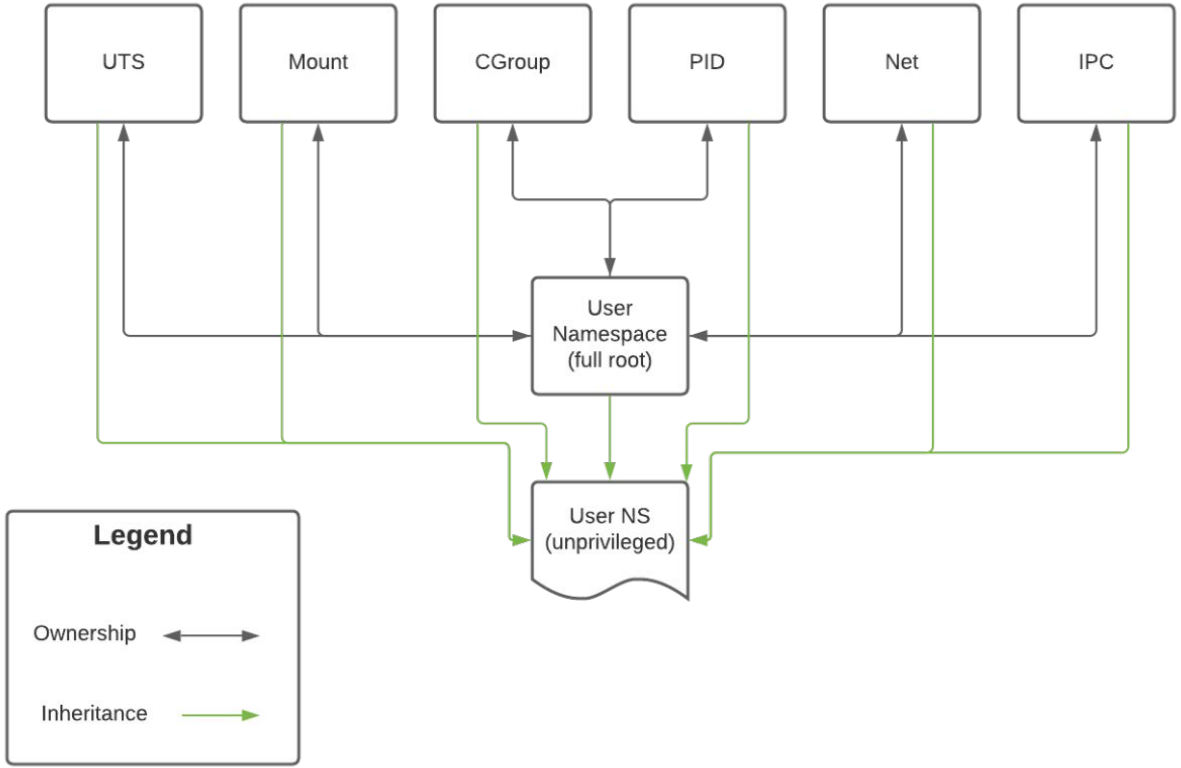
User namespace and capabilities

The child process created ...in a new namespace... starts out with a complete set of capabilities in the new user namespace.

Having a **capability inside** a user namespace permits a process to perform operations (that require privilege) only on resources governed by that namespace. In other words, having a capability in a user namespace permits a process to perform privileged operations on resources that are governed by (nonuser) namespaces owned by (associated with) the user namespace (see the next subsection).

When a nonuser namespace is created, it is owned by the user namespace in which the creating process was a member at the time of the creation of the namespace. Privileged operations on resources governed by the nonuser namespace require that the process has the necessary capabilities in the user namespace that owns the nonuser namespace.

End result hands on



Mount Namespace

Mount namespaces provide isolation of the list of mounts seen by the processes in each namespace instance. Thus, the processes in each of the mount namespace instances will see distinct single-directory hierarchies.

What is a mount point?
- mount point is a directory to access a filesystem.

All files in Linux are organized hierarchically and are rooted at slash in a tree. ’/’ is mounted at boot time and is called root new line.

- mount points are listed in /proc/self/mountinfo (findmnt)

Theory of mount points

Mount namespaces are never created empty!!
They inherit a copy of the namespace they are created from.

Mounts propagate by default because of a feature in the kernel called the **shared subtree**.

This allows every mount point to have its own propagation type associated with it.

peer group is defined as a group of vfsmounts that propagate events to each other.
Events are things such as mounting a network share or unmounting an optical device.

peer groups are often the deciding factor as to whether or not a mount is visible and can be interacted with. A **mount state** determines whether a member in a peer group can receive the event.

The Virtual File System (aka Virtual Filesystem Switch, VFS) is the software layer in the kernel that provides the filesystem interface to userspace programs.

Theory of mount points

There are five mount states:

1. **shared** - A mount that belongs to a peer group. Any changes that occur will propagate through all members of the peer group.
2. **slave** - One-way propagation. The master mount point will propagate events to a slave, but the master will not see any actions the slave takes.
3. **shared and slave** - Indicates that the mount point has a master, but it also has its own peer group. The master will not be notified of changes to a mount point, but any peer group members downstream will.
4. **private** - Does not receive or forward any propagation events.
5. **unbindable** - Does not receive or forward any propagation events and *cannot* be bind mounted.

What should I care?

1. originally because you wanted more user to use a CD-ROM
2. That it might happen that you need to change root folder and you want to bring some files with you?
3. You might want to mount a version file system, for example snapshot.
4. Or you have been hired by a tree data agency and you will need to provide all the users without custom user namespace or mount namespace to avoid them to see other people. And in that case Pam can help you. For example you can drop you when you open a new session into a new namespace dedicated for that. You can achieve whatever you want.

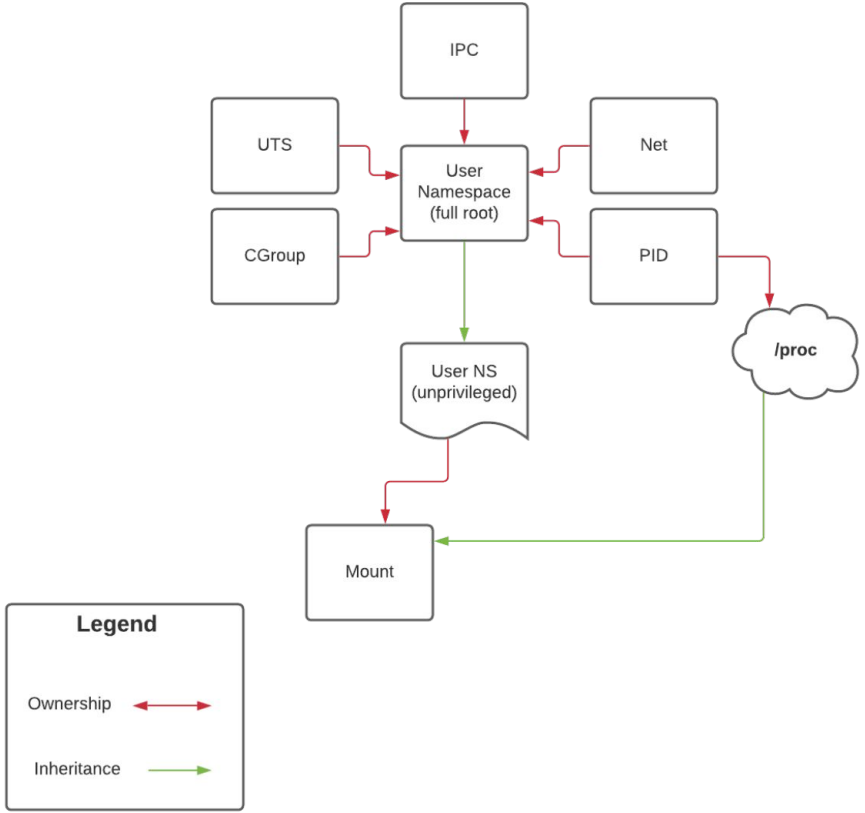
To our scope the most important is: **Share mount points**:
share some file in more points
mount -bind A B

Pid Namespace

PID namespaces isolate the process ID number space, meaning that processes in different PIDnamespaces can have the same PID. PID namespaces allow containers to provide functionality such as suspending/resuming the set of processes in the container and migrating the container to a new host while the processes inside the container maintain the same PIDs.

- What are Process Ids?

The end result of the hands-on.



Steve Ovens

How to stop a process?

kill -s signal pid

Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup (*)
SIGINT	2	Term	Interrupt from keyboard
SIGKILL	9	Term	Kill signal
SIGTERM	15	Term	Termination signal
SIGCONT	19,18,25	Cont	Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process

UTS namespace

UTS namespaces provide isolation of two system identifiers: the hostname and the NIS domain name. These identifiers are set using sethostname(2) and setdomainname(2), and can be retrieved using uname(2), gethostname(2), and getdomainname(2). Changes made to these identifiers are visible to all other processes in the same UTS namespace, but are not visible to processes in other UTS namespaces.

IPC namespace

IPC namespaces isolate certain IPC resources.

Time namespace

Is to play with time, the only usage I see is to prank people about your uptime:

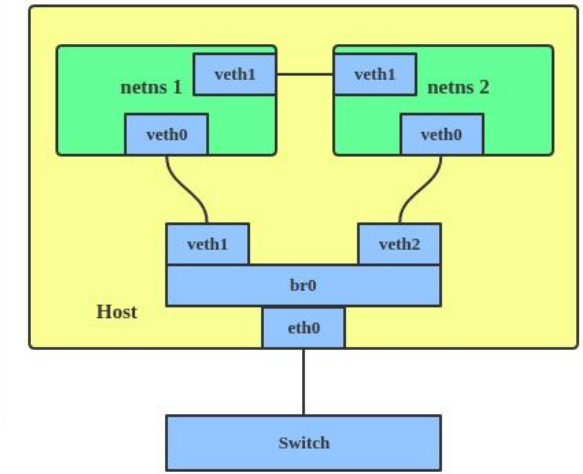
Cgroup namespace

Add a cgroup fs, very useful when live migrating containers, or having to convince a process that he is pid1 (systemd), or for security.

Network namespace

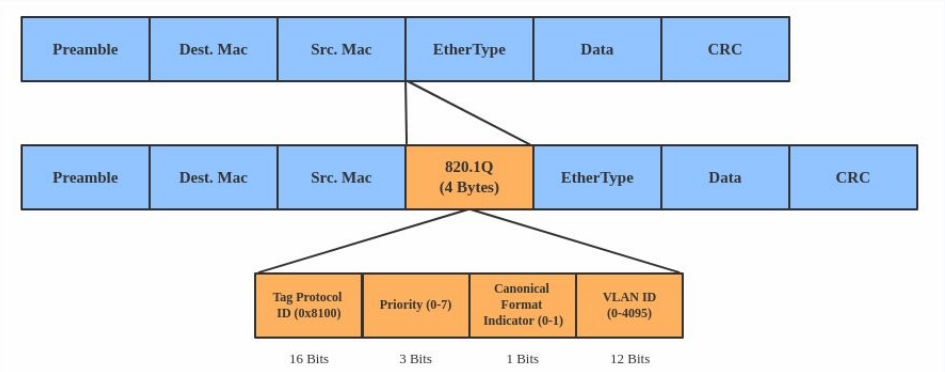
Network namespaces provide isolation of the system resources associated with networking: network devices, IPv4 and IPv6 protocol stacks, IP routing tables, firewall rules, the /proc/net directory (which is a symbolic link to /proc/pid/net), the /sys/class/net directory, various files under /proc/sys/net, port numbers (sockets), and so on. In addition, network namespaces isolate the UNIX domain abstract socket namespace (see unix(7)).

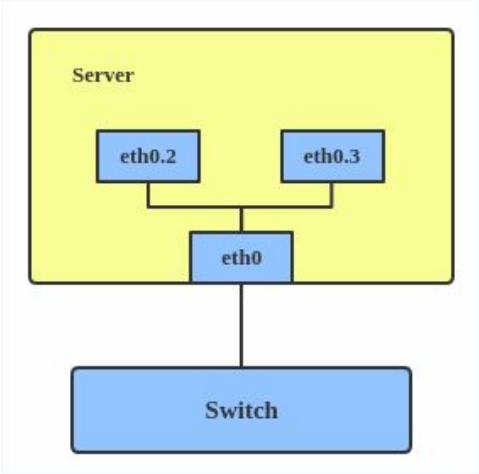
The VETH (virtual Ethernet) device is a local Ethernet tunnel. Devices are created in pairs, as shown in the diagram below. Packets transmitted on one device in the pair are immediately received on the other device. When either device is down, the link state of the pair is down.
[[./support



VLAN

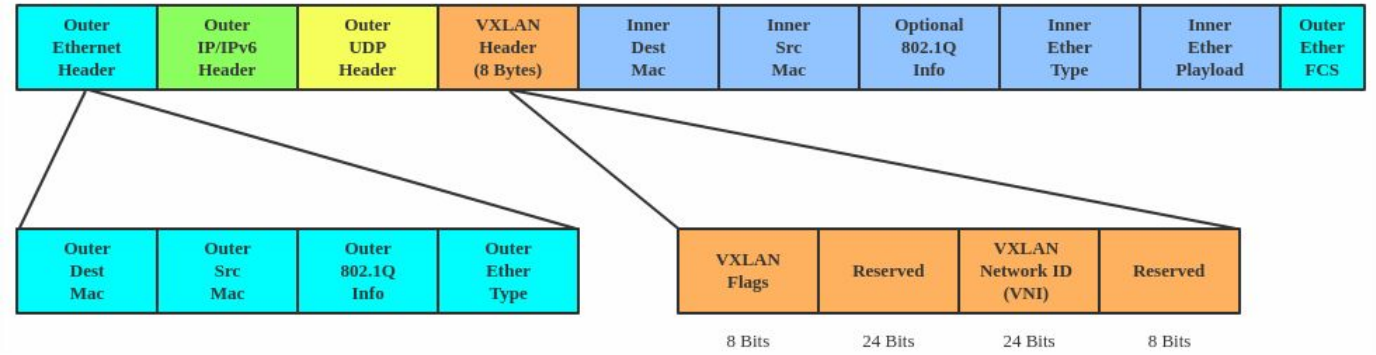
A VLAN, aka virtual LAN, separates broadcast domains by adding tags to network packets. VLANs allow network administrators to group hosts under the same switch or between different switches.

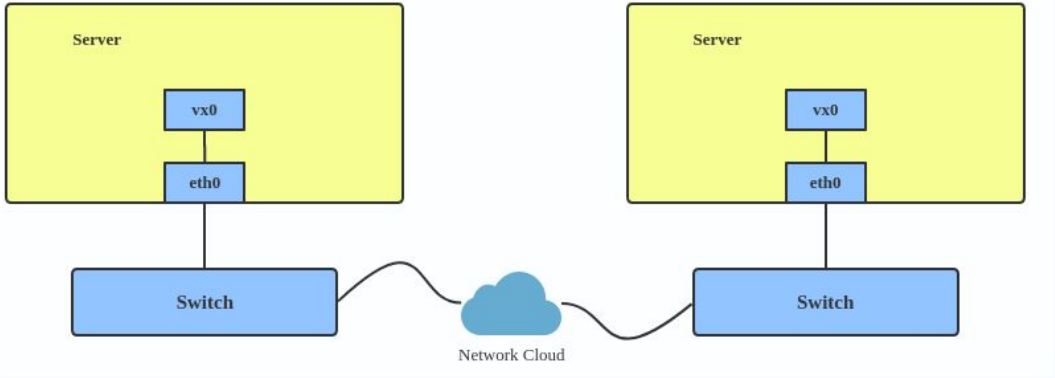




VXLAN

VXLAN encapsulates Layer 2 frames with a VXLAN header into a UDP-IP packet, which looks like this:





TUN/TAP

In computer networking, TUN and TAP are kernel virtual network devices. Being network devices supported entirely in software, they differ from ordinary network devices which are backed by physical network adapters. On the other side of a tun/tap interface there is a software usually, not another port!!