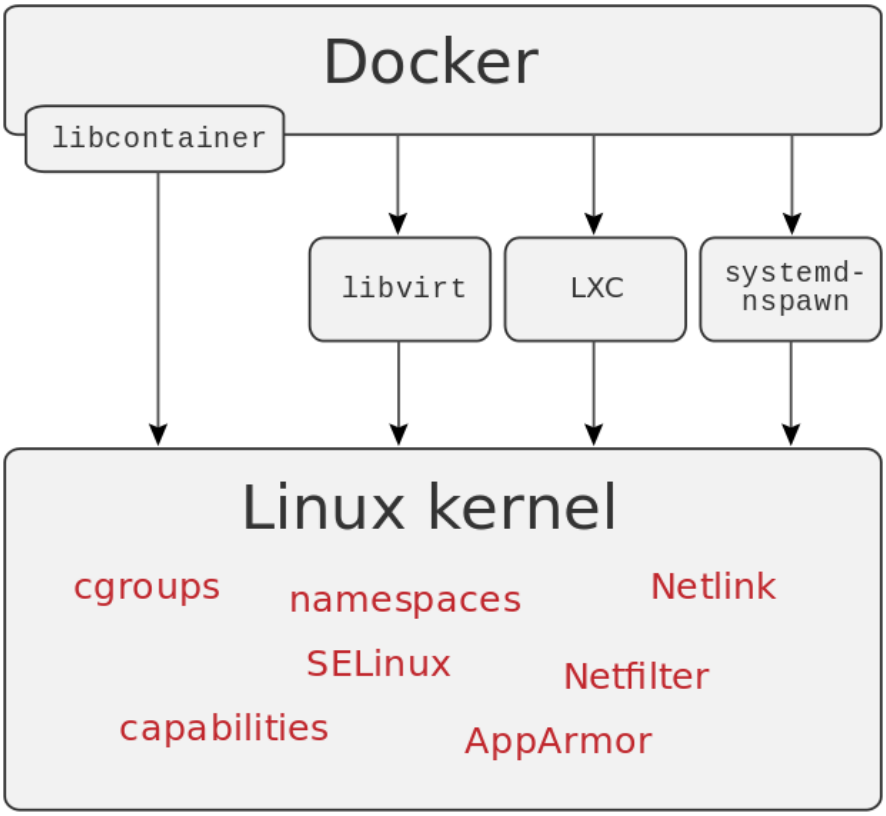
Nitin Agarwal Follow

All Things DevOps Strategist, Architect and Enabler. Containers & Docker Extraordinaire

yesterday · 3 min read

Understanding the Docker Internals



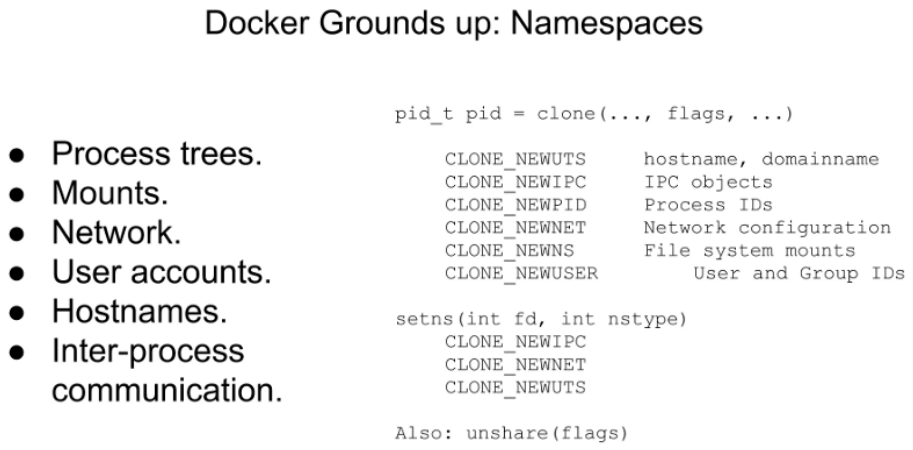
Docker takes advantage of several features of the Linux kernel to deliver its functionality.

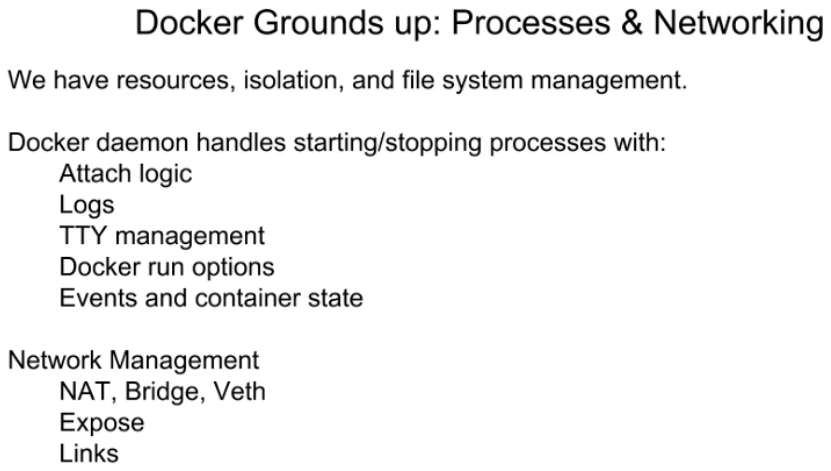
Namespaces

Docker makes use of kernel namespaces to provide the isolated workspace called the *container*. When you run a container, Docker creates a set of *namespaces* for that container. These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

Docker Engine uses the following namespaces on Linux:

- **PID namespace** for process isolation.
- **NET namespace** for managing network interfaces.
- **IPC namespace** for managing access to IPC resources.
- **MNT namespace** for managing filesystem mount points.
- **UTS namespace** for isolating kernel and version identifiers.





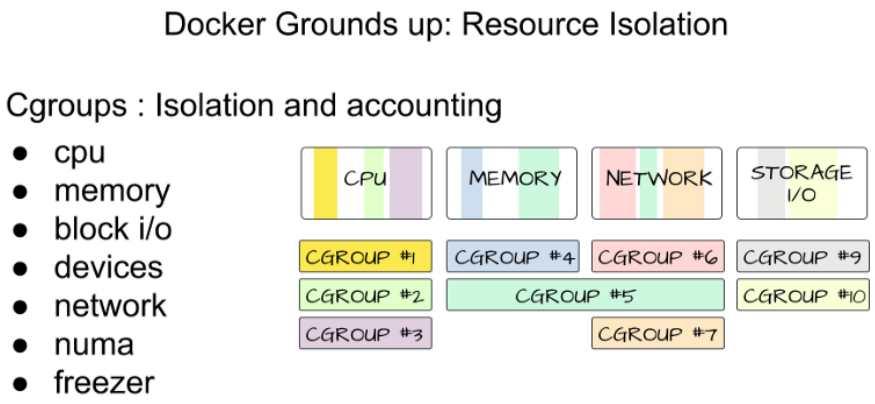
Cgroups

Docker also makes use of kernel control groups for resource allocation and isolation. A cgroup limits an application to a specific set of re-

sources. Control groups allow Docker Engine to share available hardware resources to containers and optionally enforce limits and constraints.

Docker Engine uses the following cgroups:

- **Memory cgroup** for managing accounting, limits and notifications.
- **HugeTBL cgroup** for accounting usage of huge pages by process group.
- **CPU group** for managing user / system CPU time and usage.
- **CPUSet cgroup** for binding a group to specific CPU. Useful for real time applications and NUMA systems with localized memory per CPU.
- **BlkIO cgroup** for measuring & limiting amount of blckIO by group.
- **net_cls** and **net_prio cgroup** for tagging the traffic control.
- **Devices cgroup** for reading / writing access devices.
- **Freezer cgroup** for freezing a group. Useful for cluster batch scheduling, process migration and debugging without affecting prtrace.



Union File Systems

Union file systems operate by creating layers, making them very light-weight and fast. Docker Engine uses UnionFS to provide the building blocks for containers. Docker Engine can use multiple UnionFS variants, including AUFS, btrfs, vfs, and devicemapper.

Docker Grounds up: Filesystem

File-system Isolation:

Building a rootfs dir and chroot into it.

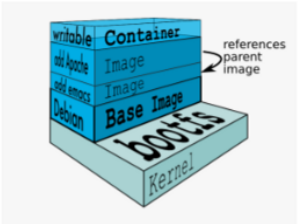
With mount namespace, use pivot-root.

Features:

Layering, CoW, Caching, Diffing

Solutions:

UnionFS, Snapshotting FS, CoW block devices



Docker Grounds up: Filesystem

	Union Filesystems	Snapshotting Filesystems	Copy-on-write block devices
Provisioning	Superfast Supercheap	Fast Cheap	Fast Cheap
Changing small files	Superfast Supercheap	Fast Cheap	Fast Costly
Changing large files	Slow (first time) Inefficient (copy-up!)	Fast Cheap	Fast Cheap
Diffing	Superfast	Superfast	Slow
Memory usage	Efficient	Efficient	Inefficient (at high densities)
Drawbacks	Random quirks AUFS not mainline !AUFS more quirks	ZFS not mainline BTRFS not as nice	Higher disk usage Great performance (except diffing)
Bottom line	**Ideal for PAAS and high density things**	**This is the Future (probably)**	**Dodge Ram 3500**

Container Format

Docker Engine combines the namespaces, control groups and UnionFS into a wrapper called a container format. The default container format is libcontainer.

Security

Docker Engine makes use of AppArmor, Seccomp, Capabilities kernel features for security purposes.

- **AppArmor** allows to restrict programs capabilities with per-program profiles.
- **Seccomp** used for filtering syscalls issued by a program.
- **Capabilties** for performing permission checks.

Docker Grounds up: Add Security

- Linux Capabilities
  - Drops most capabilities.
  - Enable what a task needs.
- GRSEC and PAX
- SELinux
- AppArmor

. . .

Cheers, Nitin Agarwal

Disclaimer: Images for this post has been taken from the web. Credit for these images goes to concerned folk.