Linux Kernel

The Linux kernel is the main component of a Linux operating system (OS) and is the core interface between a computer's hardware and its processes. It communicates between the 2, managing resources as efficiently as possible.

What the kernel does

- Memory management: Keep track of how much memory is used to store what, and where
- Process management: Determine which processes can use the central processing unit (CPU), when, and for how long
- Device drivers: Act as mediator/interpreter between the hardware and processes
- System calls and security: Receive requests for service from the processes
- File system
- TCP/IP Networking Stacks
- It Implements Access Control based on process identity and file permission

Kernel Space/ Kernel Mode (Privileged Mode), Direct access the Hardware.

User Space/ User Mode (Unprivileged Mode)

- SHELL
- Command-Line-Tool
- Application

System calls

- System calls provide an interface to the services made available by an Operating System.
- System call is the programmatic way in which a computer program requests a service from the kernel of the operating system.
- These calls are generally available as routines written in C and C++

Linux Kernel Features

1. namespace: Namespaces are one of the most important methods for organizing and identifying software objects.
A namespace wraps a global system resource (for example a mount point, a network device, or a hostname) in an abstraction that makes it appear to processes within the namespace that they have their own isolated instance of the global resource.

In short limits what you can see (and therefore use)

Namespaces provide processes with their own view of the system

There are 6 types of namespaces:

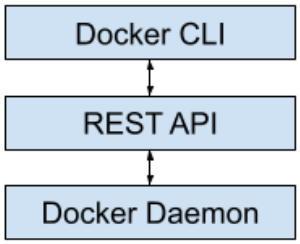
1. **mount ns** - for file system.
2. **UTS(Unique time sharing) ns** - which checks for different hostnames of running containers
3. **IPC ns** - interprocess communication
4. **Network ns**- takes care of different ip allocation to different containers
5. **PID ns** - process id isolation
6. **user ns**- different username(uid)

For more information about namespaces, see the (man `namespaces 7`)

2. **cgroup**: Control Groups provide a mechanism for aggregating/partitioning sets of tasks, and all their future children, into hierarchical groups with specialized behaviour.
In short limits how much you can use.
Cgroups involve resource metering and limiting:

- memory
- CPU
- block I/O
- network

2014 (v0.9)
2016 v1.11)
Docker was first released to the public in 2013.
The Docker Engine Consist of the Docker Daemon the REST API and the Docker CLI.



Docker Daemon

- The Docker daemon is the actual server or process that is responsible for creating and managing objects such as the images, containers, volumes, and networks on a host.

REST API

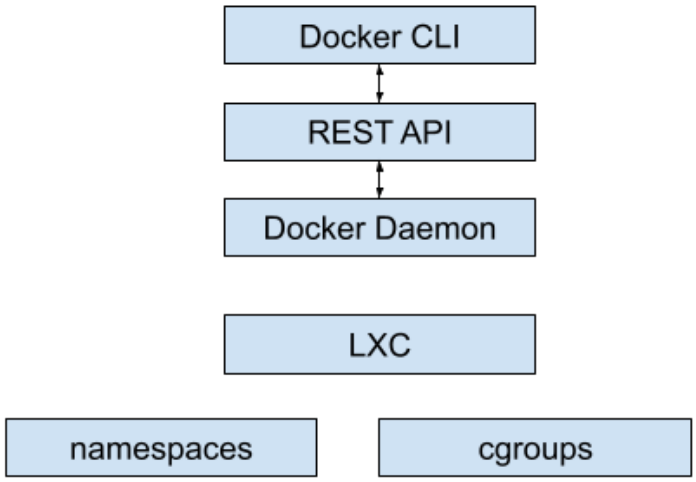
- The REST API provides an interface to manage objects in Docker.

Docker CLI

- The Docker CLI is the command line interface that we use to run commands to manage objects in docker.

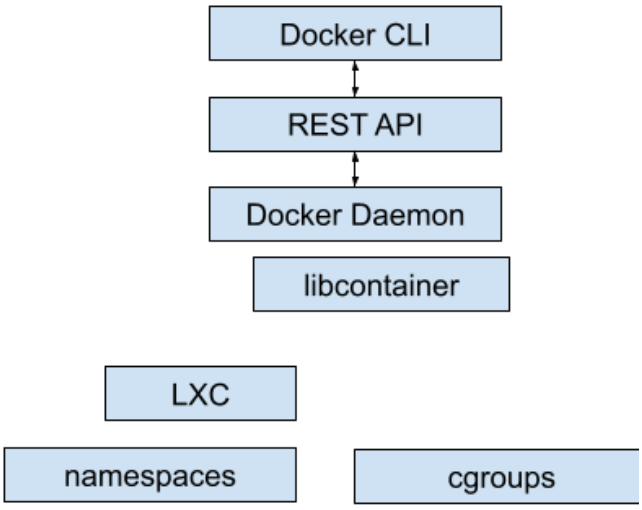
How does Docker manage containers on a host?

In the past, Docker used a technology called Linux Containers or LXC to manage containers on Linux.



LXC used capabilities of the Linux kernel such as namespaces and cgroups to create isolated env on linux known as Container.
It was hard to work with LXC directly for a normal user.

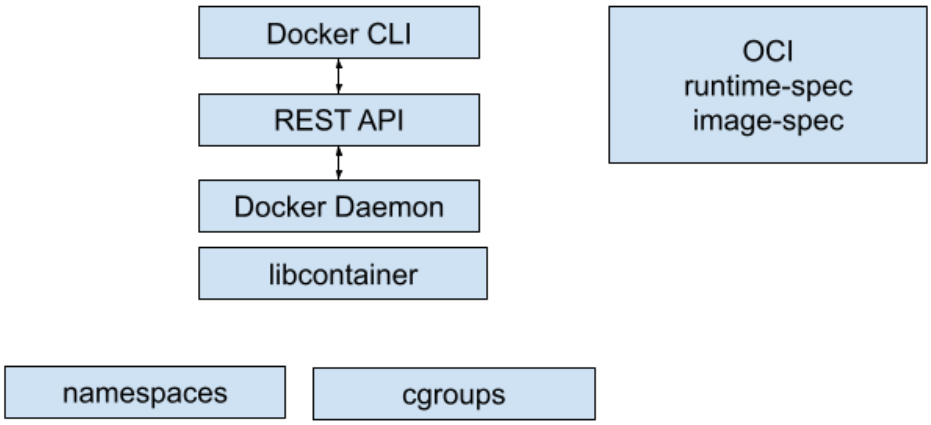
That's where docker provided a set of tools that made managing containers simpler.
With the release of version 0.9, Docker introduced its o**wn execution environment** known as **libcontainer**.



Libcontainer is written in GO, the same programming language that docker is written in and reduces docker's dependency on the kernel's LXC technology.
With libcontainer, docker could now directly interact with the Linux kernel features such as namespaces and cgroups and thus, libcontainer replaced LXC as the default execution environment of docker.

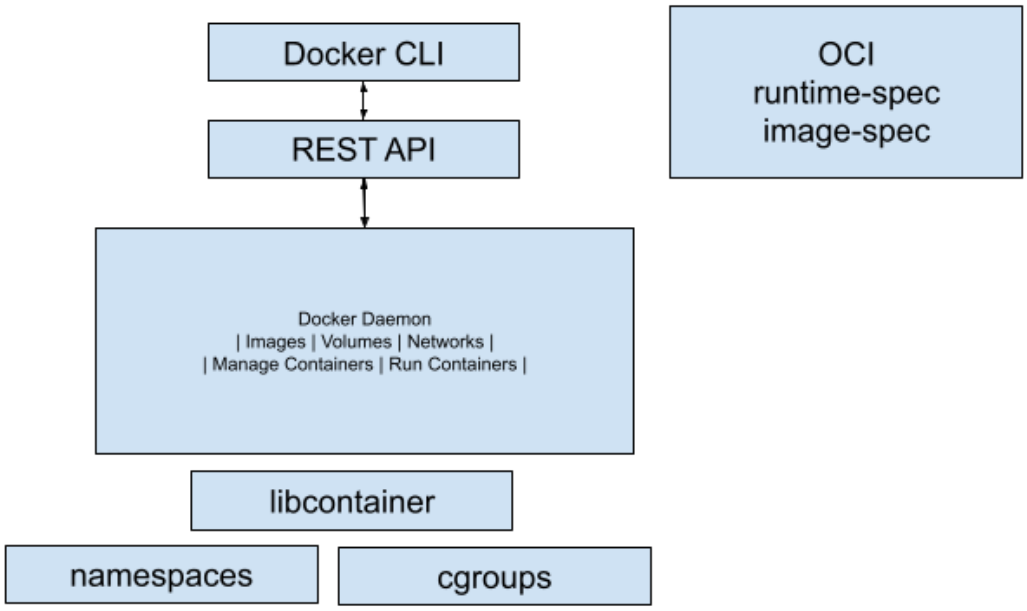
Until early 2015, there were no standards or guidelines for container softwares.
The open container initiative or OCI, was formed by Docker coreOS and other leaders in the industry for creating a set of open industry standards around container formats and runtime.
To streamline the creation of standards for containers, OCI created two specifications.

The runtime specification
The image specification;



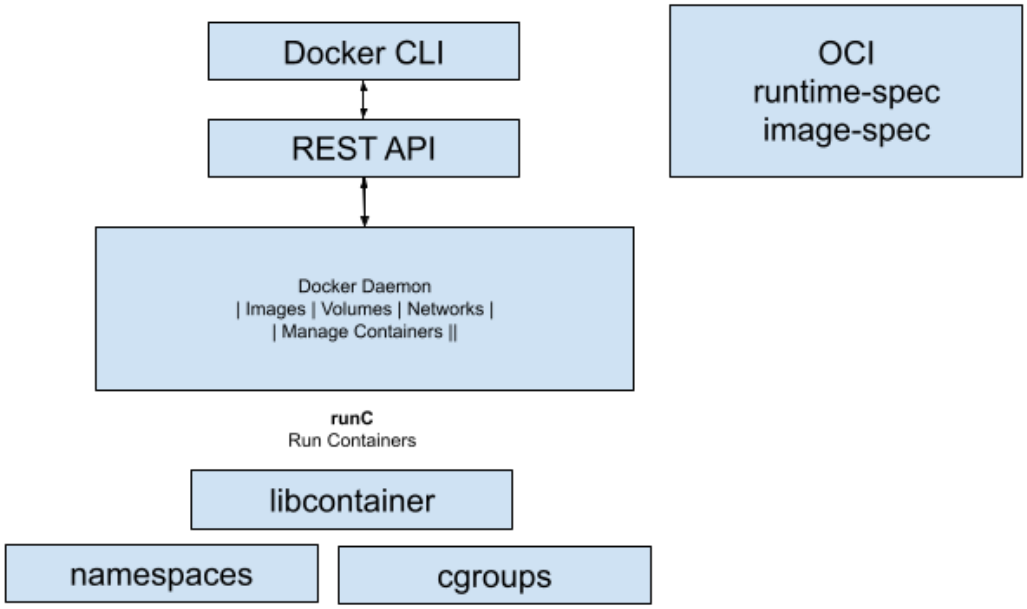
The runtime specification defines the life cycle of a container technology such as create command create a command start command a start a container.

Until then , the docker daemon was a single large monolithic code base that performed multiple functions such as running containers and managing containers and managing networks of volumes and images on the docker host as well as the mechanisms required for pushing and pulling images from docker repositories.



With the oct standards in places, the architecture of the docker engine was refactored and broken down into smaller reusable components, in version 1.11

The part that run containers became an independent component known as runC.
runC was the first OCI based technology and was donated by docker to the Open Compute Project Foundation.



Now you could run container simpleng by installing runC.
Without installing docker at all but without many of the dokcers, core features
The daemon that managed containers became a seprate component kowns as containerd
it now manages runC which in run uses the libcontainer to create containers on a host.
Now what happenes whene the deamon itself goes down or is restrated what happens to the containers, and who takes care of the containers.

to handle this situation, a new compotent named containerd-shim was added to make containers daemon less.

instead of containerd creating container it;s the containerd-shim that does it and monitors the state of the contianer

even if the docker daemon shuts down or is restarted

the containers run in the background and is attached back when the daemon comes back up,

that's the high level arch of docker