




# SHIPPING CONTAINERS

Sachin Dharmapurikar

---

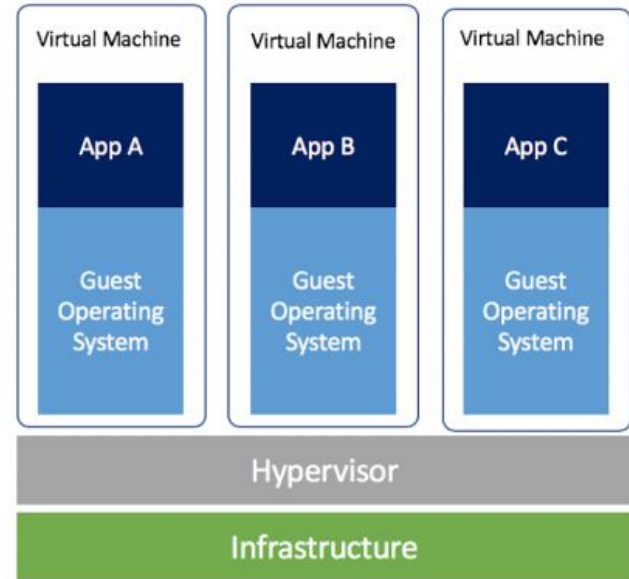
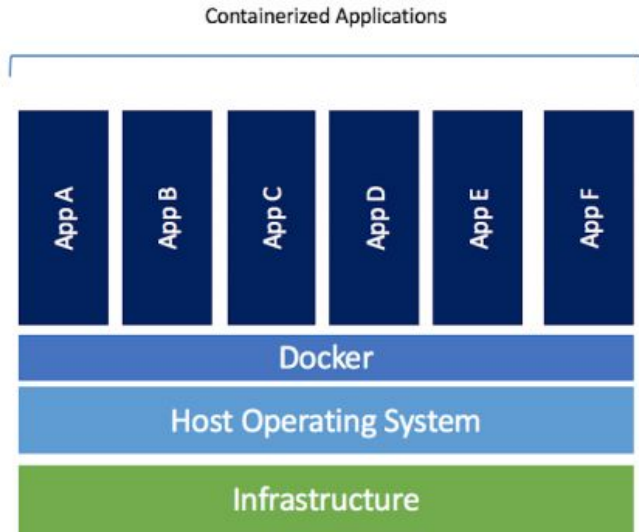
# INTRODUCTION



## WHAT IS A CONTAINER? (EXECUTIVE SUMMARY)

- A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.
- A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

# VM vs Container





# HELLO CONTAINER

- ❏ `docker run hello-world`
- ❏ You just launched a container, it printed text to STDOUT and exited.
- ❏ `docker ps -a`



## SHOW ME A COMPLEX CONTAINER

- ❏ `docker run ubuntu:18.04 echo "Hello from Ubuntu."`
- ❏ `docker run -it ubuntu:18.04`
- ❏ `ls /proc`
- ❏ You just launched an ubuntu container, and launched bash in interactive mode.



## WHAT IS A CONTAINER? (MORE TECHNICAL)

- ❑ Containers share the host Kernel
- ❑ Containers use the kernel ability to group processes for resource control
- ❑ Containers ensure isolation through namespaces
- ❑ Containers feel like lightweight VMs (lower footprint, faster), **but are not Virtual Machines!**



## THREE LINUXes SAME KERNEL

- ❏ `docker run ubuntu:18.04 uname -a`
- ❏ `docker run ubuntu:12.04 uname -a`
- ❏ `docker run alpine:3.8 uname -a`
- ❏ As per <https://wiki.ubuntu.com/SecurityTeam/ESM/12.04> 12.04 never go beyond Linux 3.2.x kernel!





# HISTORY OF CONTAINERS

- Chroot circa 1982
- FreeBSD Jails circa 2000
- Solaris Zones circa 2004
- Linux OpenVZ circa 2005 (not in mainstream Linux)
- LXC circa 2008
- Docker circa 2013
  - built on LXC
  - moved to libcontainer (March 2014)
  - appC (CoreOS) announced (December 2014)
  - Open Containers standard for convergence with Docker Announced (June 2015)
  - moved to runC (OCF compliant) (July 2015)

---

**TELL ME HOW IT'S DONE**



## LET'S BUILD A CONTAINER

- ❏ `[fortune]$ docker build . --tag fortune`
- ❏ `docker run -t fortune`
- ❏ You just built a container from scratch, ran it!



## LET'S BUILD A COMPLEX CONTAINER

- ❏ `[webserver]$ docker build . --tag demo-server`
- ❏ `docker run --name demoserver -d demo-server`
- ❏ `docker ps`
- ❏ `docker inspect demoserver|grep IPAddress`
- ❏ `docker run --name demoserver_2 -p 80:80 -d demo-server`

---

# DOCKER COMPOSE TO THE RESCUE

## DEMO: SAP

---

**WHAT HAPPENS UNDER THE HOOD?**



## CGROUPS - SECRET BEHIND DOCKER

- **cgroups** (Control Groups) is a feature of Kernel that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes.
- Two engineers at Google started developing this in 2006. It was first merged in Linux mainline kernel in 2008.
- Second revision of **cgroups** was merged in Linux mainline in 2016. It is focused on process discrimination rather than threads.



## CGROUPS CONT.

- **Memory**
  - Hard Limit - If a process requests more memory than this limit, the entire group is killed.
  - Soft limit - If a process requests more memory than this limit, it is allowed but eventually all memory is exhausted.
- **CPU**
  - Allows to set weights. Not limits.
  - On idle host with low shares, a process is allowed to use 100% CPU!
- Other limits are **Disk, I/O, Network** etc.





# LIFECYCLE OF A CONTAINER

```
t0=$(date "+%Y-%m-%dT%H:%M:%S")
```

```
docker run --name=ephemeral -t dharmapurikar/cowsay 'Hello  
Kodelounge!'
```

```
t1=$(date "+%Y-%m-%dT%H:%M:%S")
```

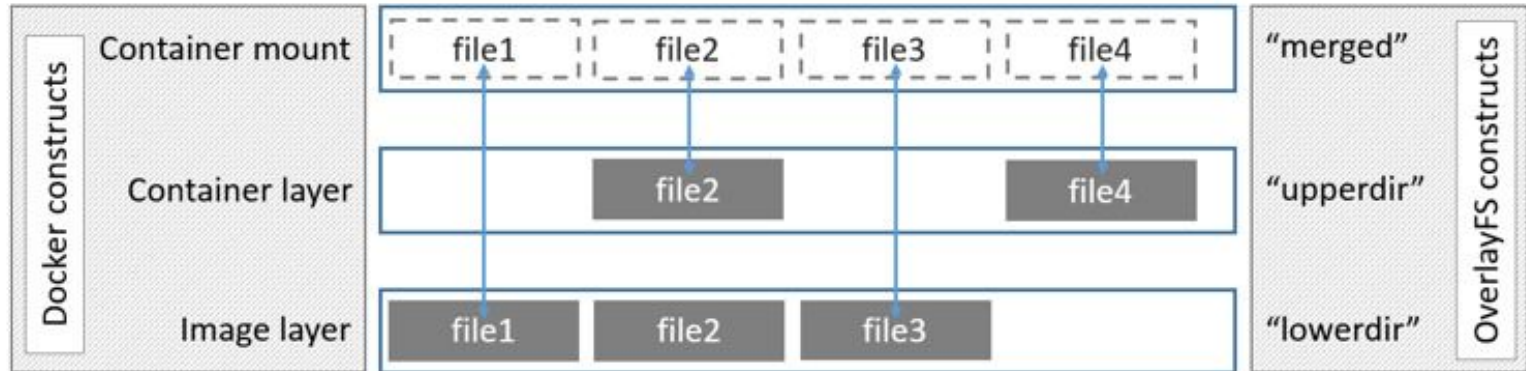
```
docker events --since $t0 --until $t1
```



# WHAT HAPPENS TO MY FILESYSTEM

- Docker creates the filesystem in the form of layers. Every layer contains set of files added, removed or modified. Previous layers are unaffected by the changes in current layer.
- Each container starts from a layer and continues creating new layers.
- Docker uses **copy-on-write** method to form layers. A thin RW layer is created when container is launched. Every file which is written or modified from layers below is copied and modified.
- Top most layer of the docker is ephemeral if container is not saved or exported. Upon removing container, it will lose all of its data.

# LAYERED FILE SYSTEMS





# LAYERED FILE SYSTEMS

```
[original]$ docker build . --tag layer:1
```

```
[extended]$ docker build . --tag layer:2
```

```
docker history layer:2
```

```
docker history layer:1
```

---

# CLOSING THOUGHTS



## FEW THINGS TO REMEMBER

- ❑ Docker is now Xerox. There are other container technologies available.
- ❑ Virtual Machines are more secure by design.
- ❑ Use containers because they are light and can package a piece of software component well.
- ❑ Containers are only native to Linux.

---

**FIN.**

**<https://github.com/dharmapurikar/docker-talk>**