Docker

A FRAMEWORK FOR DATA INTENSIVE COMPUTING

Agenda

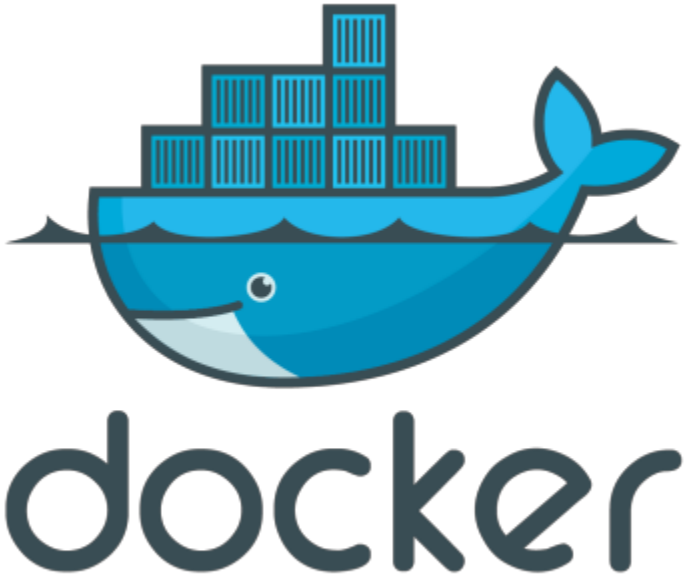
Intro / Prep Environments

Day 1: Docker Deep Dive

Day 2: Kubernetes Deep Dive

Day 3: Advanced Kubernetes: Concepts, Management, Middleware

Day 4: Advanced Kubernetes: CI/CD, open discussions

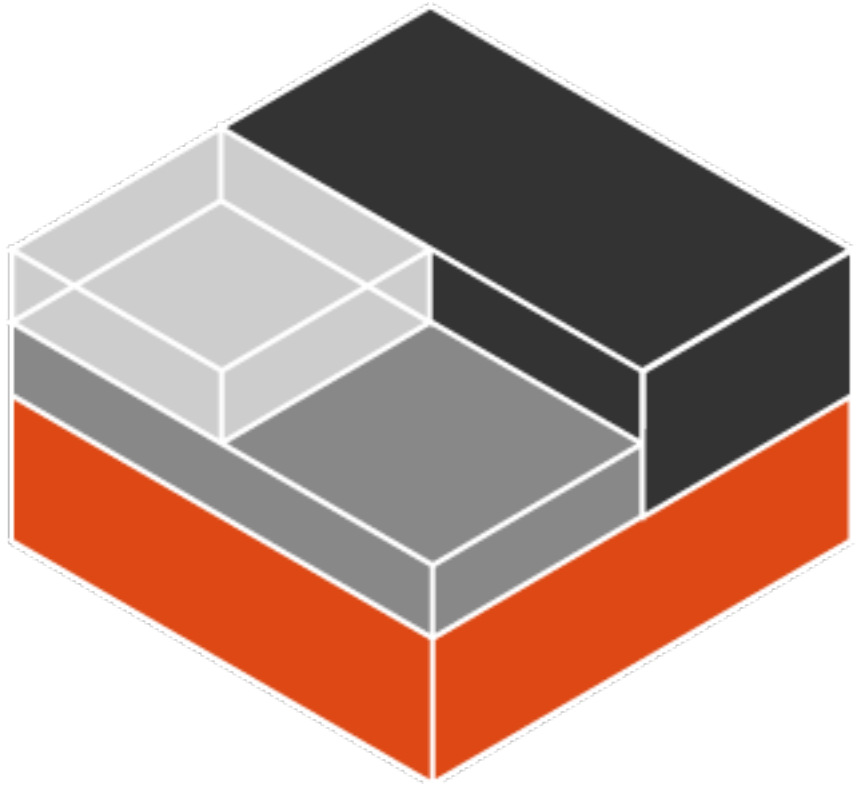


What is This Docker Thing?

- A company?
- A format?
- An API?
- Framework?
- Or, maybe something in the Cloud.
- Everything's in the cloud **seems** like these days!

That's **Not** New About Docker

- Linux-**native** functionality
- Control Groups (**cgroups**)
- Kernel **Namespaces**
- **chroot**
- Linux **Capabilities**
- Security (**SELinux**)



What Are Cgroups

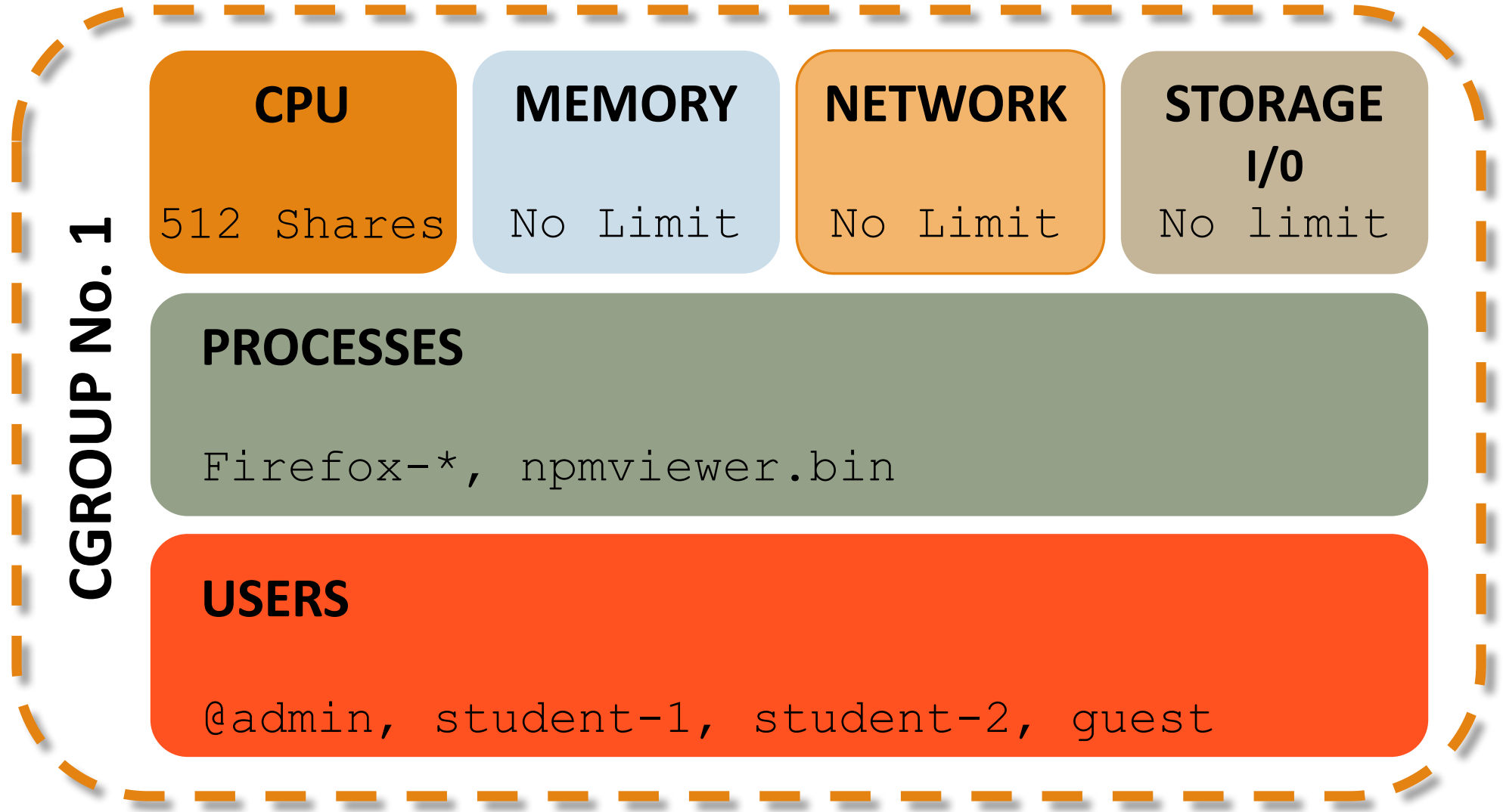
- Built into Kernel (RHEL7/Debian/etc)
- Generically isolates resource usage (CPU, memory, disk, network)
- Guarantee resources to app/set of apps
- Can be adjusted on the fly
- Can monitor the cgroup itself to see utilization

What Are Cgroups

- Wikipedia says:

"**cgroups** (abbreviated from control groups) is a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes."

Diagram of Control Groups



Kernel Namespaces

- Isolating views of the system
- You can make a process think it’s the only process
- Built-in ways to "virtualize" a process

Wikipedia says:

“Namespaces are a feature of the Linux **kernel** that isolates and virtualizes system resources of a collection of processes. Examples of resources that can be virtualized include process IDs, hostnames, user IDs, network access, inter-process communication, and filesystems.”

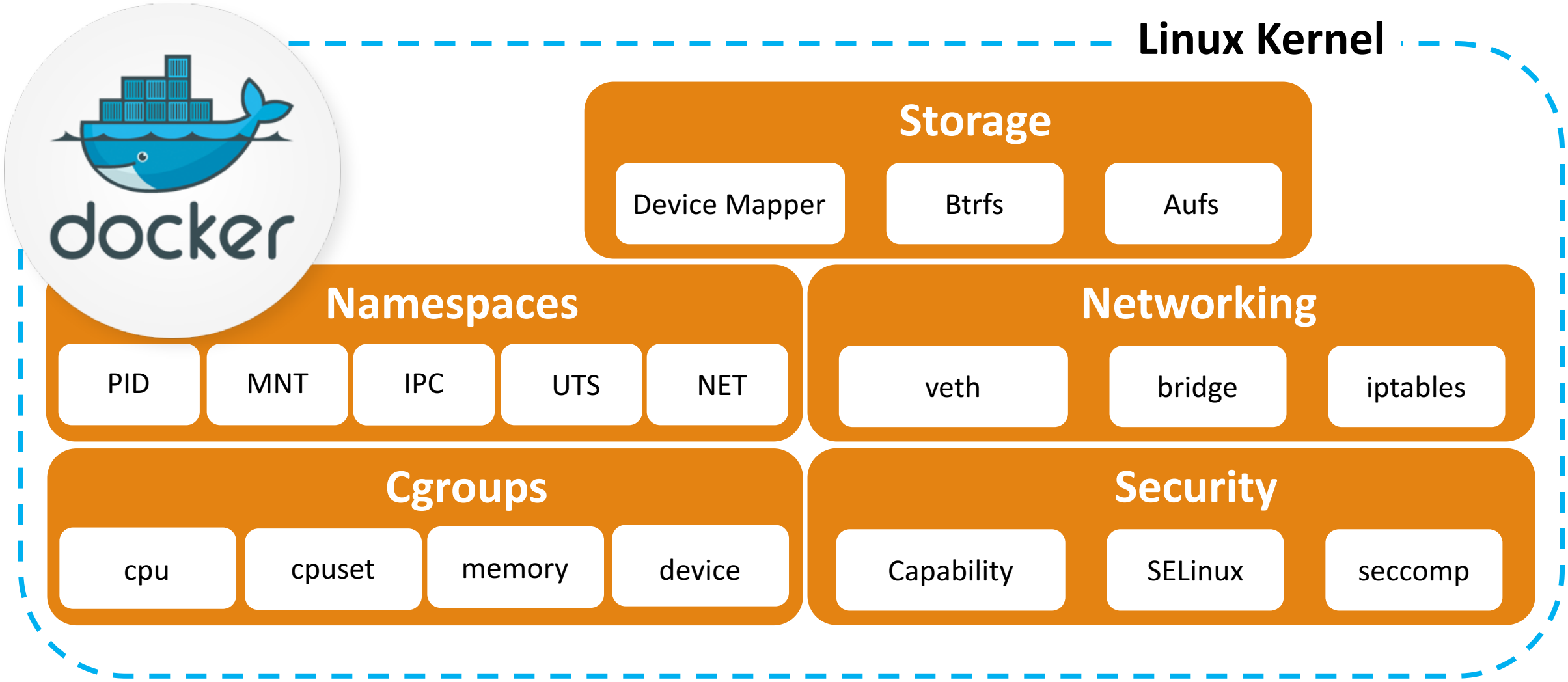
Kernel Namespaces

- **`mnt`** mount points, filesystem
- **`pid`** processes
- **`net`** network stack
- **`ipc`** inter-process comms
- **`uts`** hostname
- **`user`** UIDs

Linux capabilities

- Like Linux, `root` has **all** capabilities.
- Includes a fine-grained division of `root`'s permissions for a processes.
- `CAP_NET_ADMIN` - modify routing tables, firewalling, NAT, etc.
- `CAP_KILL` - bypass any checks for sending the kill signals
- `CAP_SYS_ADMIN` - mount, set hostname, etc.

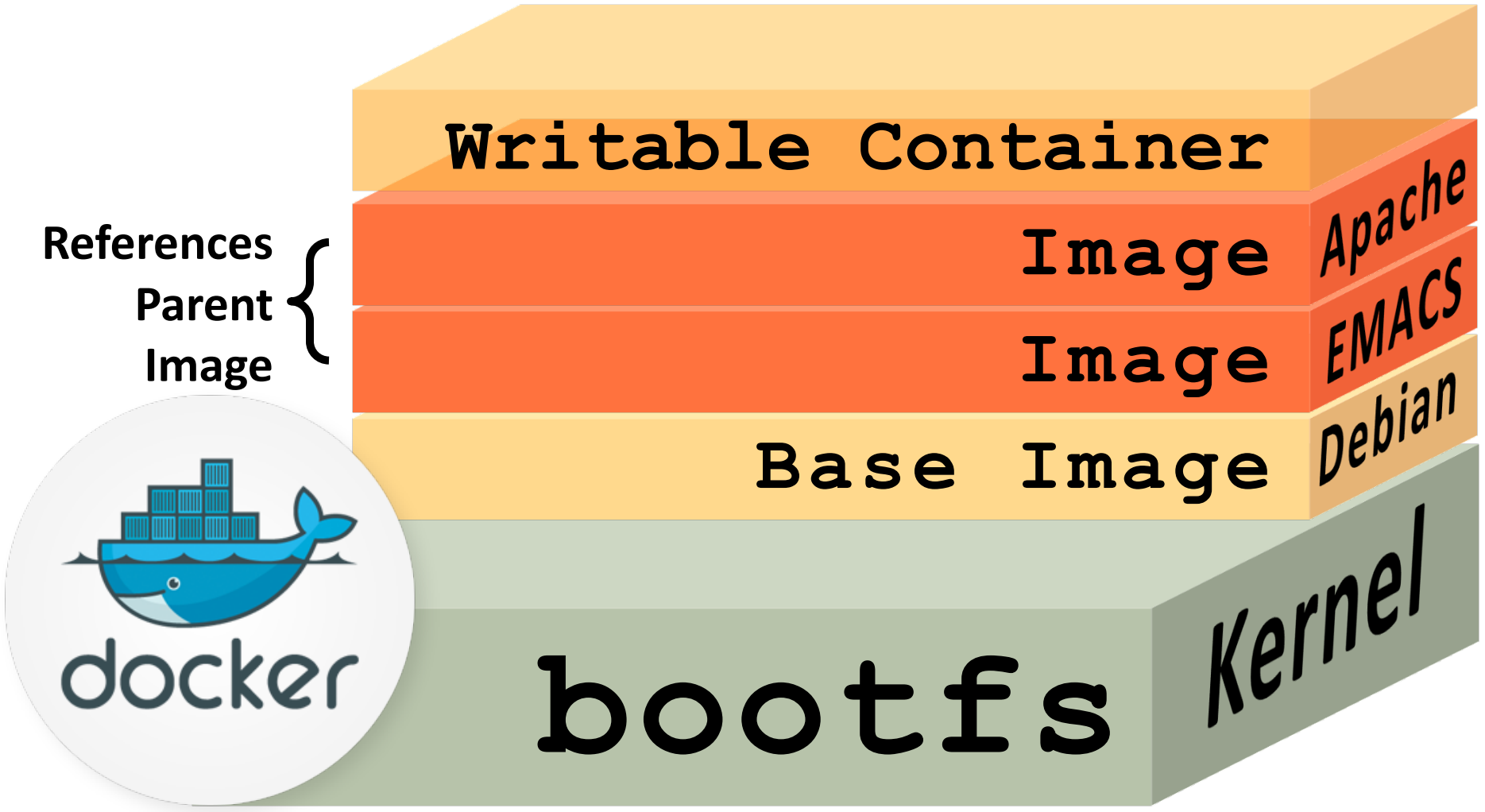
Docker Brings Together



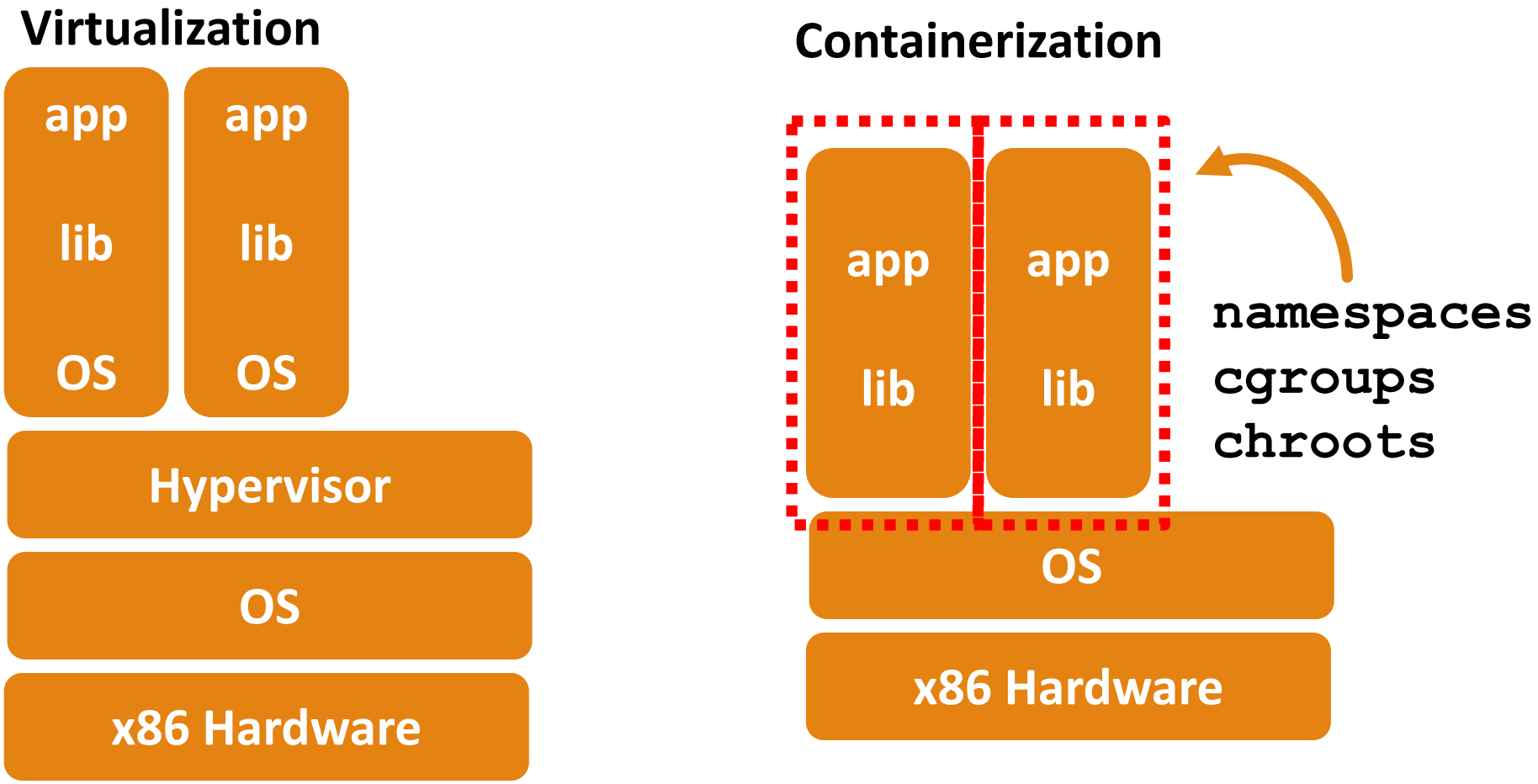
Why is this Important?

- Think about image formatting vs. golden imaging
- When using API calls
- Packaging
- Structural separation of concerns (Devs/Ops)
- Density and infrastructure utilization

Foundations of Docker



Virtualization vs. Containerization



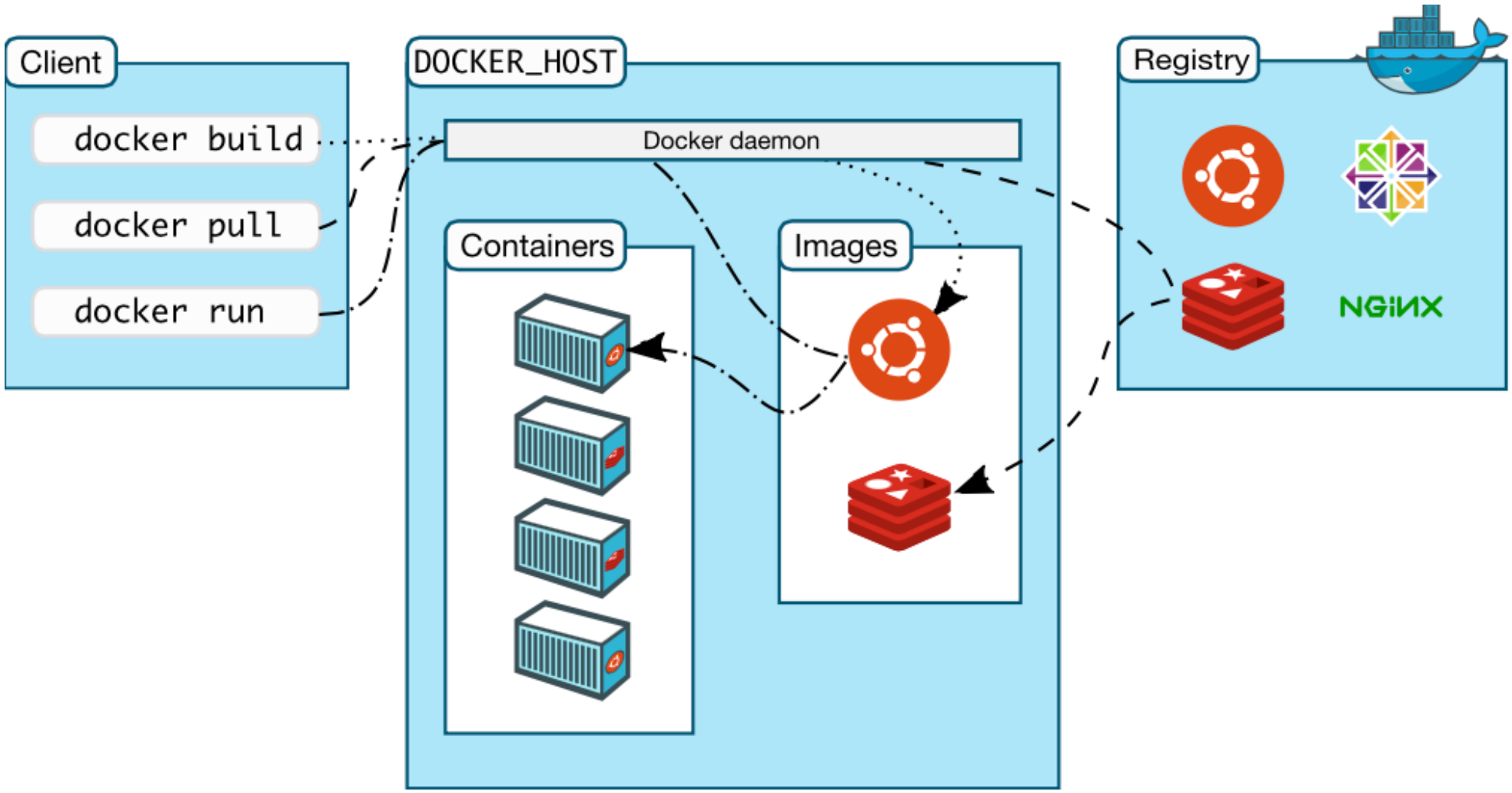
Immutable Infrastructure

- "We'll put it back in **Ansible**"
- Cattle **vs.** Pets
- Don't change it; **replace it**
- System created fully from automation; **avoid drift**
- Manual **intervention** is error prone
- How does **Docker** help?

Basic Docker Components

- Docker Client
- Docker Daemon
- Images
- Registry
- Containers

Basic Docker Components

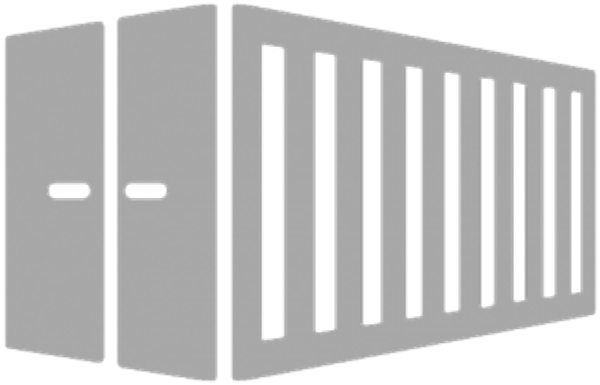


Docker Images

- **Templates** from which containers are created
- **Layered** using union filesystems
- Each change to the system is a **layer** added
- Typically created from **instructions** stored in Dockerfiles
- Stored in a Docker **registry**, locally or remotely

Docker Containers

- **Runtime** instance of a Docker Image
- **Copy** on write file system; changes localized
- **“Virtualized”** with namespaces, cgroups, SELinux, etc.
- Each has **own** IP address, networking, and volumes
- Intended to run **single** process (process virtualization)

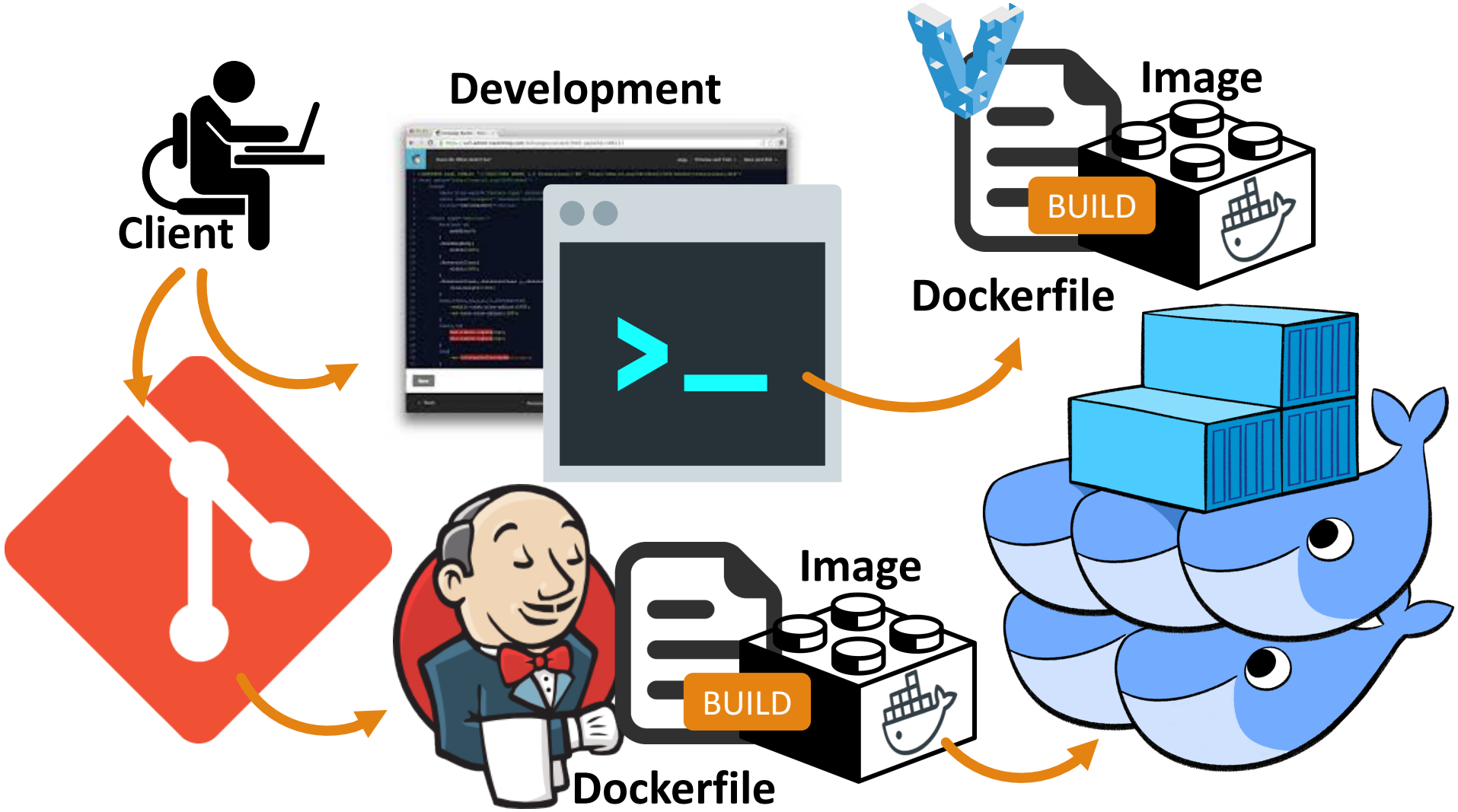


Developer Workflow

- Work from **Vagrant** image
- Can trash and reboot it any time
- **Locally** running Docker client
- Source code in developer IDE
- When ready, use tooling to **generate** Docker image, or hand craft
- Run image locally - possibly with others
- Push code, or image
- CI process kicks in

Developer Works Locally

Developer Pushes Code



Hello Docker Container

LET'S CREATE OUR FIRST CONTAINER

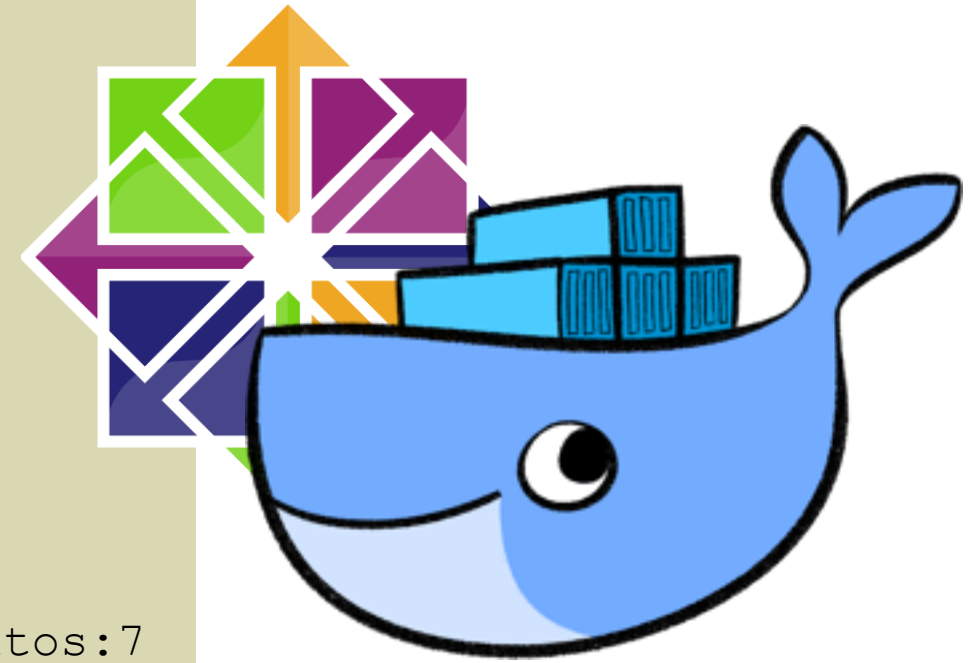
Pull a Docker Image

- Let's pull a Centos 7 image from DockerHub, like this:

```
docker pull centos:7
```

- Output:

```
Pulling from library/centos
fa5be2806d4c: Pull complete
0cd86ce0a197: Pull complete
e9407f1d4b65: Pull complete
c9853740aa05: Pull complete
e9fa5d3a0d0e: Pull complete
Digest: sha256:xxxxxxxxxxxxx...
Status: Downloaded newer image for centos:7
```



List Docker Images

- Now, let's list all our locally installed images, like:

```
docker images
```

- Output:

```
REPOSITORY TAG IMAGE ID      CREATED     VIRTUAL SIZE

centos     7   e9fa5d3a0d0e  2 days ago  172.3 MB
```

List Docker Images

- We can pass the images command (-a) option to see all images, past and present:

```
docker images -a
```

- Output:

```
REPOSITORY TAG      IMAGE ID      CREATED       VIRTUAL SIZE
centos     7        e9fa5d3a0d0e  2 days ago    172.3 MB
<none>     <none>     c9853740aa05       2 days ago    172.3 MB
<none>     <none>     e9407f1d4b65       2 days ago    172.3 MB
<none>     <none>     0cd86ce0a197       2 days ago      172.3 MB
<none>     <none>     fa5be2806d4c       5 weeks ago     0 B
```

Running Our First Docker Container

- Here's how we can then run a Linux command inside a Docker container:

```
docker run –rm centos:7 echo “hello world”
```

- Output:

```
hello world
```

- Woah, what happened? It just printed "hello, world"? So what?

Running Our First Docker Container

- Let's run a Linux shell **inside** a Docker container, from the inside:

```
docker run --it --rm centos:7 bash
```

- Output:

```
[root@d7dfcc490cbe /]# _
```

Inside Our Container

- Now from inside the Docker container itself, we'll issue commands, like this:

```
# ll /etc/*-release
```

- Output:

```
-rw-r--r-- 1 root root  38 Mar 31  2015 /etc/centos-release

-rw-r--r-- 1 root root 393 Mar 31  2015 /etc/os-release

lrwxrwxrwx 1 root root  14 Aug 14 21:00 /etc/redhat-release ->
centos-release

lrwxrwxrwx 1 root root  14 Aug 14 21:00 /etc/system-release ->
centos-release
```

Inside Our Container

- We can now play around with Linux commands within the container:

```
# hostname -f
# cat /etc/hosts
# ps aux
# yum -y install vim
# ip a
```

- If it looks like Linux and it works like Linux...

Let's Make a Mess

- Now, lets pretend we do some destructive stuff, on purpose:

```
# rm -fr /usr/sbin
```

- And now say we delete this, too:

```
# rm -fr /usr/bin
```

- Now, if we tried to issue any commands, like `ls`, `ps`, or `cd` we'd get:

```
bash: /usr/bin/ls: No such file or directory
```

- Whoops… cannot `ls`, `cd` or do anything useful anymore. What'd we'd do?

How Easy Is Starting Over

- In the last slide we pretended to destroy all our hard work.
- Now, how easy is it to start over?
- Just exit the container and fire up a new one:

```
docker run --it --rm centos:7 bash
```

- And everything is back! Wow… right?

Lab

End of Chapter