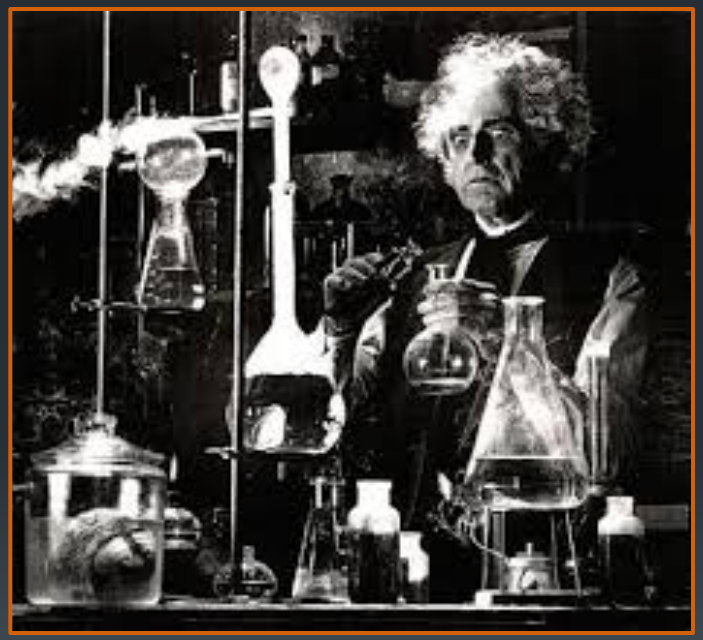
Docker Foundation

An in depth introduction to Docker and Containers

Lecture and Lab

- Our Goals in this class are two fold:
  1. Familiarize you with general concepts and ecosystems
     - This is the primary purpose of the lecture/ discussion sessions
     - The instructor will take you one a tour of the museum
       - Like a museum tour you should listen to and interact with the instructor
       - You will not have time to read the slides during the tour, like a museum the instructor will discuss and point out the highlights of the slides (exhibits) which will be waiting for you to read in depth later should you like to dig deeper
  2. Give you practical experience
     - This is the primary purpose of the labs
     - Classes rarely have time for complete real world projects so think of the labs as thought experiments
       - Like hands on exhibits at the museum



Docker & Containers

1. Container Overview
2. Container Basics
3. Controlling Containers
4. Advanced Container Operations

1: Container Overview

Objectives

- Understand the basic nature of containers
- Explain the container value proposition
- Examine the differences and similarities between Virtual Machines (VMs) and Containers
- Consider the roles of IaaS, PaaS, and Containers
- Explain the purpose and positioning of Docker in the container space
- Define the Docker system requirements
- Learn how to install Docker

What is Docker?

- Docker is an open-source project that automates the deployment of applications inside software containers
  - Originally released in March of 2013
  - Apache 2.0 license
  - Source on GitHub
- Containers provide “operating-system virtualization” on Linux
  - Containers utilize: OS Virtualization
- Containers avoid the overhead of starting virtual machines
  - VMs utilize: Machine Virtualization
- Docker uses resource isolation features of the Linux kernel such as *cgroups* and kernel *namespaces* to allow many diverse containers to run within a single Linux instance
- Docker, Inc. is a venture backed company composed of the original authors and current maintainers of the Docker software

What is a Container?

- Lightweight Linux environment
  - More recently, lightweight Windows environments as well
- Encapsulated and deployable
- Runnable
- A way to package applications for reliable deployment
  - Particularly popular for packaging microservices
- Microservice centric (one atomic service per container)
- Made widely popular by Docker
  - Docker, Inc. provides a container platform including systems for publishing and sharing containers
  - Container technology predates and enables Docker (LXC, OpenVZ, …)
  - Docker has been the dominant mover in this space but competition is growing and standards are evolving
- Relies on integral features of the Linux Kernel
  - CGroups
  - Namespaces
  - Linux Bridge/IPTables/Capabilities/etc.

```
$ time docker container run ubuntu echo "hello world"
hello world

real        0m0.319s
user        0m0.005s
sys         0m0.013s
```

Disk usage: less than 100 kB
Memory usage: less than 1.5 MB

Largest Container Fallacy

- Containers replace virtual machines

  - Virtual Machines aren't going anywhere and are the basis for cloud infrastructure globally
  - Containers can replace VMs in circumstances where:
    - VMs were being used due to lack of alternatives for application packaging
    - VMs were being used for process isolation wherein the constituent processes all run on Linux in a single tenant environment

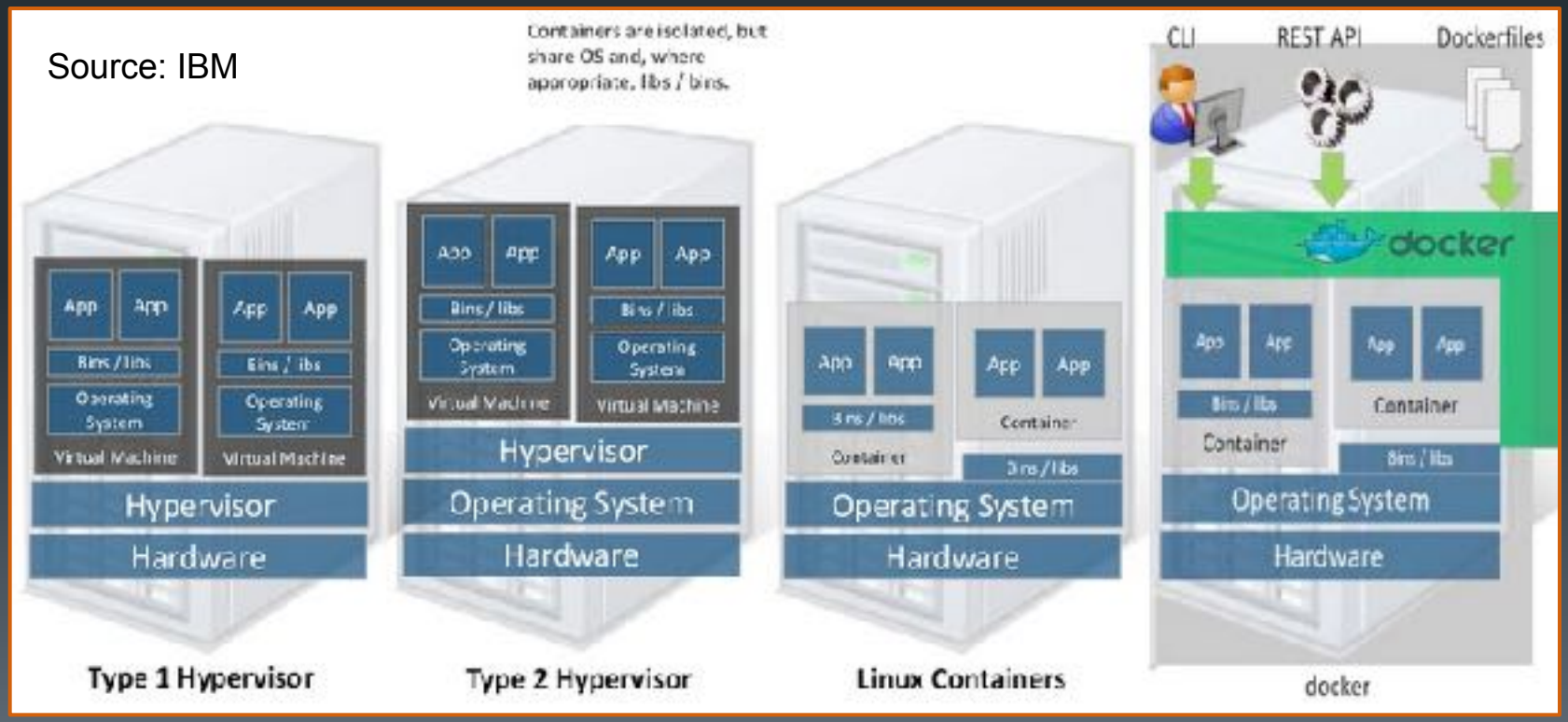
If the only tool you have is a hammer, everything looks like a nail.
-- *Abraham Maslow*

VMs and Containers

- VM
  - A virtual machine for operating systems
- Container
  - A virtual operating system for applications
- These are not mutually exclusive and many environments become optimal when properly combining both

Containers supply only the executables and library interfaces necessary to mimic the application dependent aspects of a Linux distribution (Ubuntu, RHEL, SUSE, etc.), leveraging the fact that all Linux systems use the same underlying kernel

Source: IBM



The Killer Feature of VMs:

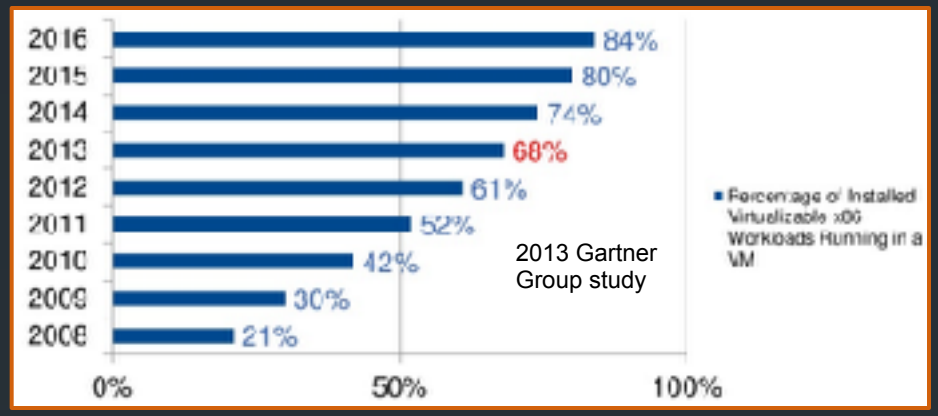
Server Density

VMs allow isolated machine instances to be packed onto physical servers increasing physical server utilization from an estimated 15-30% (per Gartner) to 70-90% (3-5x)

VMs, Why Do I Care?

- VMs allow new machine instances to be deployed in real time (seconds-minutes)
  - Not months, as is typical with physical server purchase/deploy cycles
- VMs enable migration and elasticity
  - Machines can be created, moved and deleted rapidly
- VMs allow physical resources to be fully utilized
  - Physical CPU/RAM use can be maximized without mixing logical machine roles
  - Increased server density
- VM's enable repeatable static system environments to be used for development, testing and deployment
  - The same machine can be launched by Vagrant, OpenStack, Amazon EC2, Microsoft Azure, Google Cloud, etc.
- VMs enable cloud computing
  - Pay as you go
  - Self service

IaaS



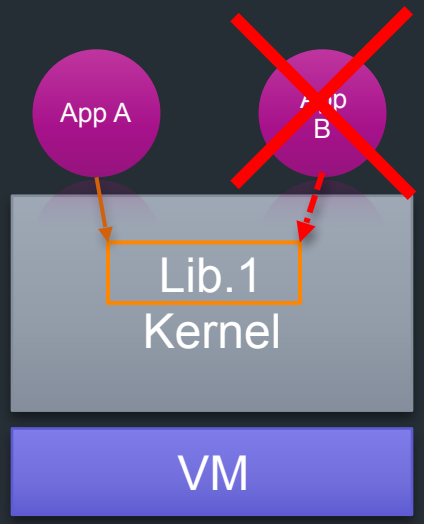
The Killer Feature of Docker:
Reliable Deployment

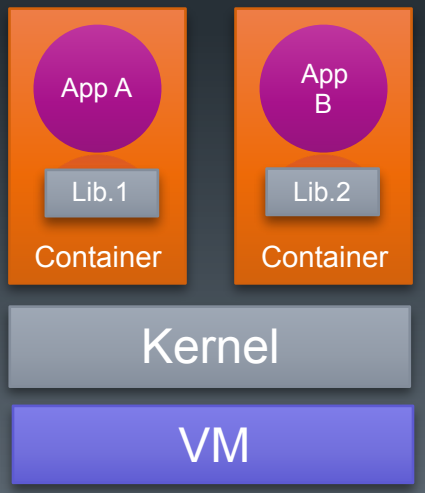
Containers include their own dependencies, isolated from the underlying operating system and other containers

Containers, Why Do I Care?

- Containers provide a static application runtime environment creating reliable deployments
  - Fundamentally changes the approach to operations
  - Removes an entire class of extremely complex operational problems

- VMs are typically used to host infrastructure roles (e.g. Web Server, Application Server, Database Server, …)
  - Applications running on such systems can have complex interactions with system services and other applications
  - This complexity makes it difficult to guarantee identical dev/test/prod environments, making application deployment complex and error prone
- Containers create a new layer of abstraction at the operating system level for application roles (web server, logging system, security tools, monitoring software, etc.)
  - Isolation
    - Allows multiple containerized applications to run on the same VM
  - Encapsulation
    - Each container is encapsulated with its own unique dependencies
  - Portability
    - Repeatable deployment across dev/test/prod environments and clouds





PaaS

Encapsulation

- Containers can encapsulate:
  - Data
  - Code
  - Configuration files
  - Frameworks
  - Libraries
  - System Dependencies
  - Packaging
  - Linux Distributions
  - Environment Variables
- Everything needed by an application can be packaged within the application's container
- We can ignore where and how the container runs
  - The container's internals do not interact with external aspects of the environment, removing an entire class of deployment problems

The Second Killer Feature of Docker:

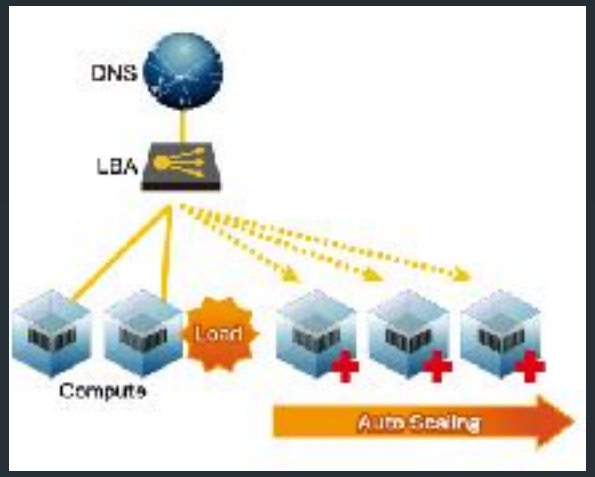
Efficiency

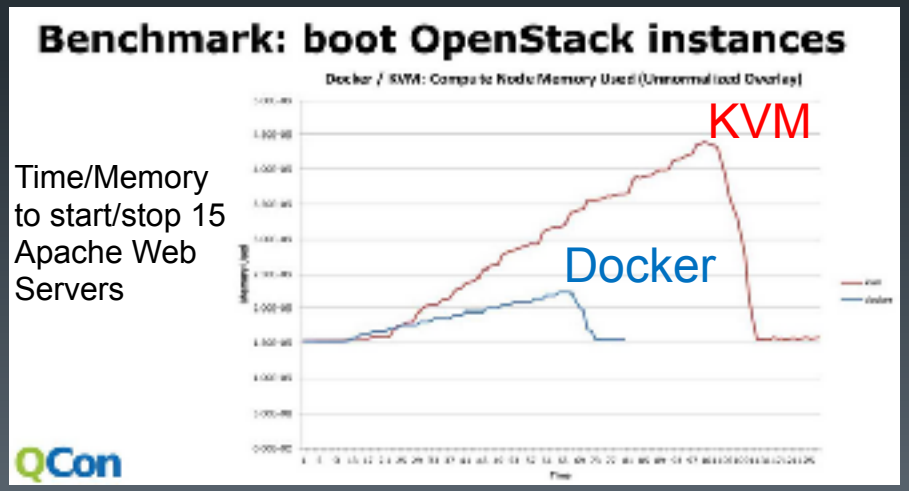
- Time to launch
- Server density
- Performance

Containers can be launched and shutdown at the speed of a process and can directly use/share system resources; typically 1/10th to 1/100th the size of the equivalent application packaged within a VM, increasing server density; containers offer the opportunity for isolation with the performance of bare metal

Scaling Events & Restart

- Containers can be started and stopped in milliseconds
  - Timing like running a program, because it is running a program
    - Run container (dependencies built in)
- VMs are started and stopped in seconds/minutes
  - Timing like booting a computer because it is booting a computer
    - Bootloader
    - OS start
    - Service start
    - CM run (Puppet/Chef/Ansible/Salt etc.)
      - Downloading packages
    - Application start
- Container based applications can be scaled in/out quickly
  - But you must have a cluster to scale them on!
- Failed services within a container can be restarted quickly

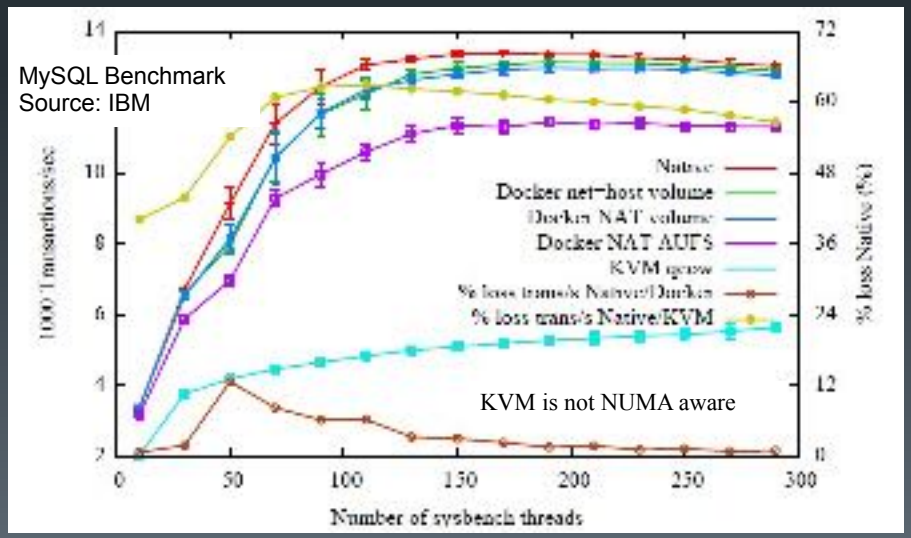




Time/Memory to start/stop 15 Apache Web Servers

Inter Service Communications

- Inter VM
  - Strong isolation enforced by hypervisor and hardware
  - Forces inter VM process communications to use networking interfaces (Ethernet/IP/[UDP|TCP])
    - Some hypervisors have fast paths but these are often platform specific and some require hardware support
  - The Strength of VMs: Very secure
- Inter Container
  - Tunable isolation (namespaces can be isolated or shared)
  - Can communicate over IP loopback (localhost/127.0.0.1)
    - No name lookup
    - No NIC translation
  - Directories can be shared (therefore supporting named pipes, UNIX sockets, memory mapped files)
  - Shared memory
  - Shared kernel structures (semaphores, mutexes, message queues, etc.)
  - The Strength of Containers: Very fast



Docker Alternatives

- Virtual Machines
  - Better isolation, slower performance and more complex deployment
  - VMware, Amazon Web Services, Google, and Rackspace all run Docker-based workloads on behalf of cloud customers in a multi-tenant environment, but do so by putting each customer's Docker containers inside the logical boundaries of virtual machines
- Rocket (rkt)
  - A simple and lightweight Docker-like container platform
    - Jetpack is a FreeBSD implementation of the Rocket App Container spec
  - Used in the Tectonic platform - CoreOS/rkt/Kubernetes integration
- Joyent SmartOS & Triton
  - Triton uses the docker client but supplies SmartOS (an OpenSolaris/KVM hypervisor) with Solaris Zone isolation (using Illumos Zones) for containers offering strong security
  - Containers see all SmartOS/Triton machines as a single Docker host, simplifying deployment
  - Built by Joyent the company behind Node.js
- CRI-O
  - Implementation of the Kubernetes CRI (Container Runtime Interface) to enable using OCI (Open Container Initiative) compatible runtimes
    - Formerly OCID (OCI daemon)
  - Top contributors from Red Hat
    - Used by OpenShift Online 3.7 (tech preview) & 3.9+
- Mesos
  - Distributed OS running "frameworks"
- LXC
  - Original Linux container library (now at v2)
- LXD/OpenVZ
  - Systems in containers (lightvisors)
- BSD Jails
  - Isolation features of BSD Unix
- OpenSolaris Container
  - Zones-based isolation model
- KurmaOS
  - A container-based operating system
- HTCondor
  - High performance parallelization framework

TRITON SmartOS

vmware

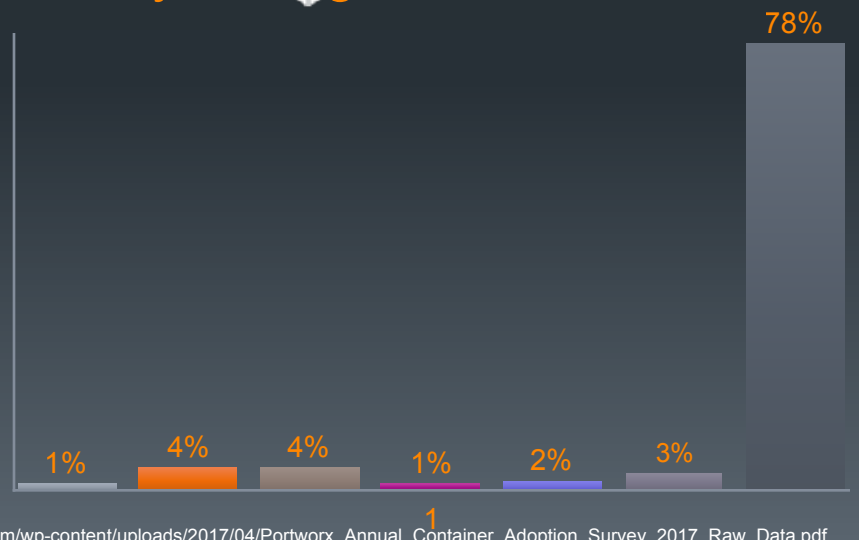
rkt

KVM

cri-o

Which container technology does your organization run?

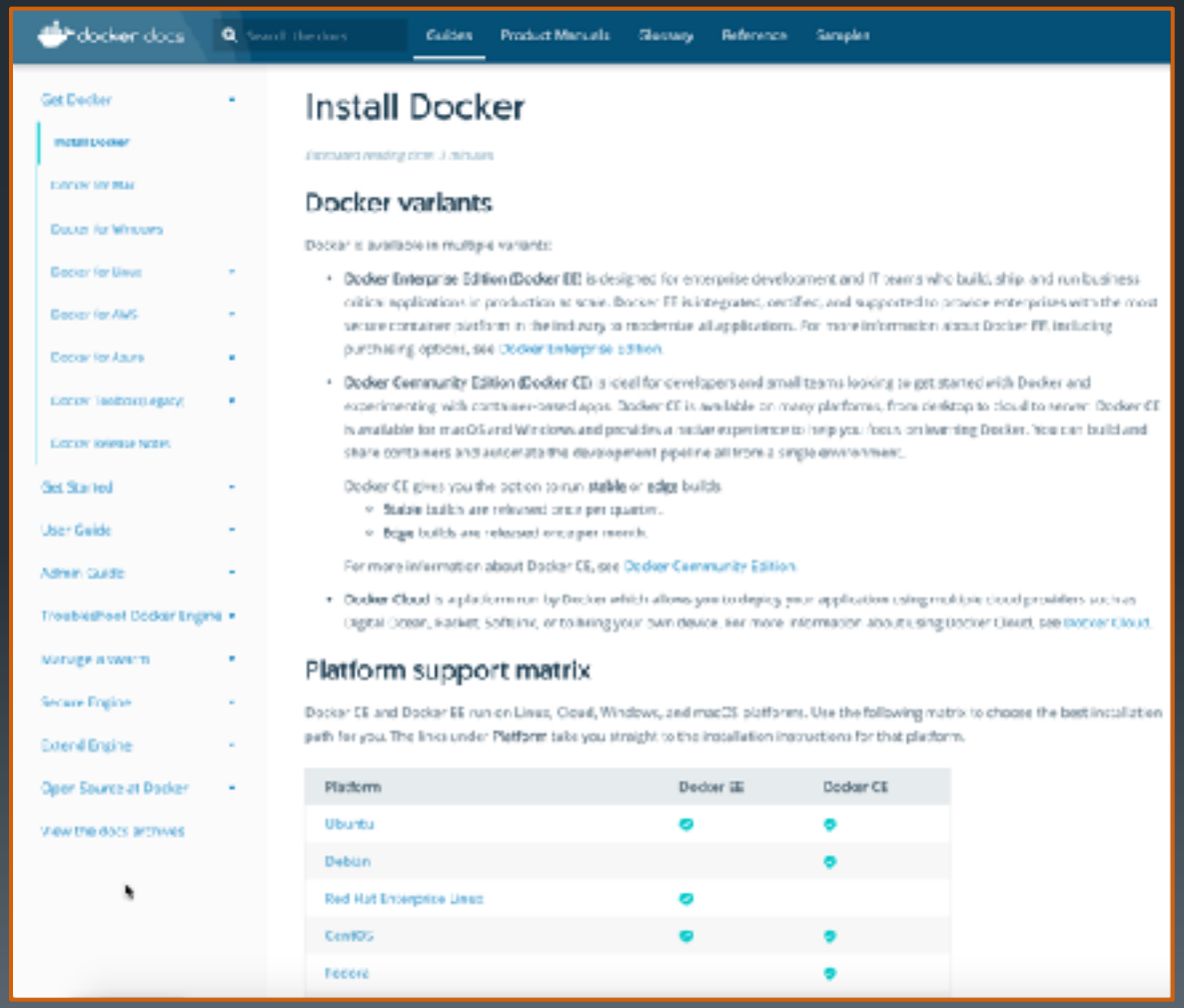
Apache MESOS



https://portworx.com/wp-content/uploads/2017/04/Portworx_Annual_Container_Adoption_Survey_2017_Raw_Data.pdf

Docker Installation

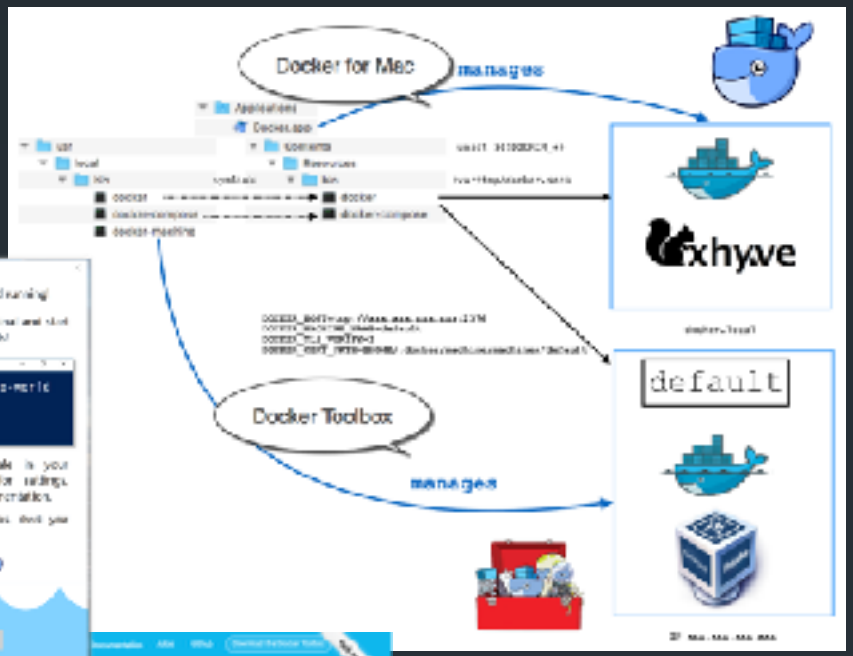
- Docker is easy to install on a range of platforms and cloud systems:
  - OSes with predefined Docker packages
    - Debian/Ubuntu
    - Red Hat EL/CentOS/Oracle EL/Fedora
    - Gentoo
    - Arch Linux
    - FrugalWare
    - OpenSUSE
    - CRUX Linux
    - Mac OS X
    - Microsoft Windows
  - Clouds Docker packages
    - Google Cloud Platform
    - DigitalOcean
    - Microsoft Azure
    - Packet
    - Amazon Web Services
    - IBM Softlayer
  - Others (e.g. CoreOS) provide their own direct support
  - Binaries are also supplied for custom installations



RHEL and Ubuntu are the recommended platforms

Docker on the Desktop

- Docker for Mac/Windows (7/2016)
  - Mac HyperKit (uses xhyve)
    - OS X 10.10.3 Yosemite or newer
    - https://github.com/docker/HyperKit/
    - https://github.com/mist64/xhyve
  - Windows Hyper V
    - 64bit Windows 10 Pro, Enterprise or Education with HyperV enabled
    - https://technet.microsoft.com/library/hh831531.aspx
  - Docker Engine runs in an Alpine Linux VM
  - Installs the Docker CLI, Compose and Machine natively
  - These are new and problems have been reported associated with HyperV and HyperKit interfering with VirtualBox and VMware
- Docker Toolbox (circa 2015)
  - You can still use Docker Toolbox if xhyve or HyperV are not available or problematic
  - Uses the VirtualBox hypervisor
  - Includes Kitematic, Docker Machine, Docker Swarm, and Docker Compose
  - Docker Machine manages the Docker VM
  - Machine still uses the Boot2Docker Linux image to run containers
- Boot2Docker (circa 2014)
  - The original Docker desktop solution
  - Used VirtualBox and the boot2docker launcher



KITEMATIC

- A Docker GUI for Mac/Windows

Docker Dependencies

- The Docker daemon requires Linux kernel 3.10+
  - Kernels older than 3.10 have bugs which can cause data loss and panic under certain conditions
  - The latest minor version (3.x.y) of the 3.10 or newer Linux kernel is recommended
    - Keeping kernels up to date will ensure critical kernel bugs get fixed
  - Key Distros:
    - RHEL/Centos 7: Kernel 3.10+
    - Ubuntu 14.04: Kernel 3.13+, 16.04: Kernel 4.4+
  - Warning:
    - Custom kernels and kernel packages are not usually supported by Linux distribution vendors
    - Installing a newer kernel may fail with distributions which provide packages which are too old or incompatible with newer kernels
- Docker Inc. supports x86_64 systems and added partial ARM support in v1.10
  - The Docker goal is to run everywhere, some vendors support Docker on other platforms
  - FreeBSD offers a 64 bit Linux compatibility layer supporting Docker
    - Free BSD info: https://wiki.freebsd.org/Docker
  - The Docker Engine API is supported on Joyent SmartOS by Triton, and Windows Server 2016 will ship with a Docker Engine compatible API and primitives to support container-based process isolation and resource management on Windows
    - Only Windows binaries based on Windows images can run on Windows
    - Windows info: https://msdn.microsoft.com/en-us/virtualization/windowscontainers/quick_start/manage_docker
- The Docker client can run on virtually any *nix system
  - The client also builds on OS X and Windows

```
user@ubuntu:~$ uname -a
Linux ubuntu 4.4.0-31-generic #50-Ubuntu SMP Wed Jul 13 00:07:12 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux
user@ubuntu:~$ cat /etc/os-release
NAME="Ubuntu"
VERSION="16.04.1 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04.1 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
UBUNTU_CODENAME=xenial
```

Summary

- Virtual machines enabled the cloud era: IaaS
  - Providing an abstraction layer between OS and the underlying hardware
  - Self service compute
  - Pay for use
  - Full hardware utilization
  - Hardware supported isolation
- Proprietary containers enabled PaaS systems
  - Providing an abstraction layer between applications and the OS
  - Did not provide users with a means to package dependencies
- Standards based containers are enabling Cloud Native Applications
  - Providing an abstraction layer between applications and the kernel
  - Containers can be deployed to PaaS, IaaS/VMs and Bare Metal OSes
  - Containers package applications into repeatable predictable environments, easily deployed consistently in development, test and production settings
  - Containers enable multiple deterministic application environments to run on a single kernel instance
- Docker can run on many platforms
  - Typically runs on x64 Linux and requires a 3.10 or later kernel

Lab 1

- Setting up Docker

VMs

- This course includes a preconfigured Ubuntu 16.04 VM for lab work
- All passwords are “user”
- All non root user accounts are “user”
- You can also complete the labs on any base installation of Ubuntu 16.04 that you have sudo/root permissions on

2: Container Basics

Objectives

- Describe the relationship between Containers and Microservices
- Explain the basic Docker architecture
  - Docker engine
  - Docker Remote API
  - Docker client
  - Registries
  - Images
  - Containers
- Examine Container isolation features (mount, network, UTS, process, IPC)
- Try creating, listing, and removing containers
- Explore docker commands for starting and controlling containers

Microservices and Containers

- Microservices
  - Used to build distributed software systems with processes that communicate with each other over network interfaces in order to perform the tasks required of an application
  - Based on technology-agnostic interface protocols (like REST and Apache Thrift)
- A more concrete and modern interpretation of SOA (Service Oriented Architectures)
  - The first realization of SOA post the introduction of DevOps
  - Becoming the standard for building continuously deployed systems
- In contrast to SOA
  - Microservices answer the question of how big a service should be and how it should communicate
    - In a microservices architecture, services should be small with one crisp responsibility
    - Interface protocols should be lightweight and language agnostic
    - Atomically deployable, dependencies are discovered
- Benefits
  - Enhanced cohesion and reduced coupling
  - Easier to change and add features
  - Allows the architecture of an individual service to emerge through continuous refactoring, reducing the need for a big design up-front and allows for releasing the software early and often

Docker containers work particularly well with microservices characterized as:

- Short-lived
- Immutable
- Disposable
- Service-oriented

Other applications can also benefit from containerization but microservices are the prime target

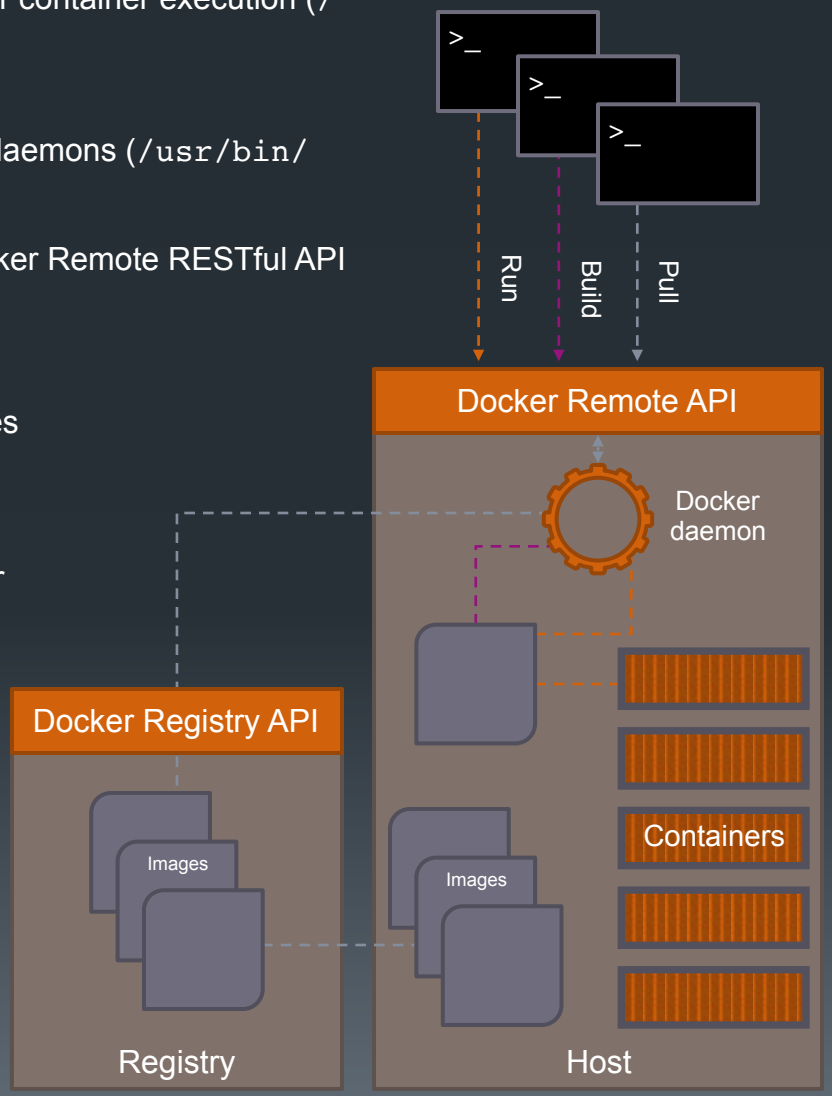
The Shipping Container Analogy

- Like a normal shipping container, Docker containers are:
  - Interchangeable
  - Stackable
  - Portable
  - Generic
- Making it easy to build a range of applications for deployment anywhere
  - CI Test Platform
  - Cloud Deployment
  - Local execution
  - Partner distribution
  - Software as a Service

Image: SD Times

Core Docker Components

- Docker client and server executables
  - $ `dockerd`
    - Runs Docker as a daemon on the Linux host supporting Docker container execution (`/usr/bin/dockerd`)
    - Docker manages containers, the Linux kernel runs containers
  - $ `docker`
    - The Docker client sends requests to local and remote Docker daemons (`/usr/bin/docker` or `docker.exe`)
  - Docker Remote API
    - The Docker client talks to the Docker daemon through the Docker Remote RESTful API
- Docker Images
  - Images are used to generate containers
    - Just as a VM image can create multiple VM instances
    - Just as an executable (/usr/bin/vi) can create multiple processes
- Registries
  - Registries are network services from which Docker Images can be saved and retrieved
  - The Docker Hub is an internet based registry with support for public and private images
  - Private registry servers can be created for an organization's internal use
- Docker Containers
  - A container is a software package generated from an image
  - Said to be running when processes are executing within it
  - Can be stopped and started



Container Lifecycle Control

- Docker Containers provide support for typical service and VM control operations
  - `docker container <command>`
    - `ls` List containers
    - `create` Create a new container
    - `rm` Removes (deletes) a container from the system
    - `start` Start a container running
    - `stop` Stop a running container
    - `restart` Restart a running container or start a stopped container
    - `run` Run a new container (create + start)
    - `pause` Pause all processes within a container
    - `unpause` Unpause a paused container

Docker =< 1.12 (Aug '16)
`docker ps` lists containers
For all other commands simply omit the `container` command

```
user@ubuntu:~$ docker container start --help

Usage:  docker container start [OPTIONS] CONTAINER [CONTAINER...]

Start one or more stopped containers

Options:
  -a, --attach               Attach STDOUT/STDERR and forward signals
      --detach-keys string   Override the key sequence for detaching a container
      --help                 Print usage
  -i, --interactive          Attach container's STDIN
```

```
user@ubuntu:~$ docker container stop --help

Usage:  docker container stop [OPTIONS] CONTAINER [CONTAINER...]

Stop one or more running containers

Options:
      --help       Print usage
  -t, --time int   Seconds to wait for stop before killing it (default 10)
```

Running Containers

- The Docker command line client provides general command help in response to the `--help` switch
  - Specific command help can be acquired by issuing the `docker` subcommand followed by the `--help` swtich
    - e.g. `$ sudo docker container run --help`
- The `run` command accepts an image and a command to run in the generated container
  - If no command is specified the default image command is run
- Run creates a new container from the image to run the command within
  - The `-i` switch connects to the container's STDIN stream
  - The `-t` switch creates an interactive tty within the container

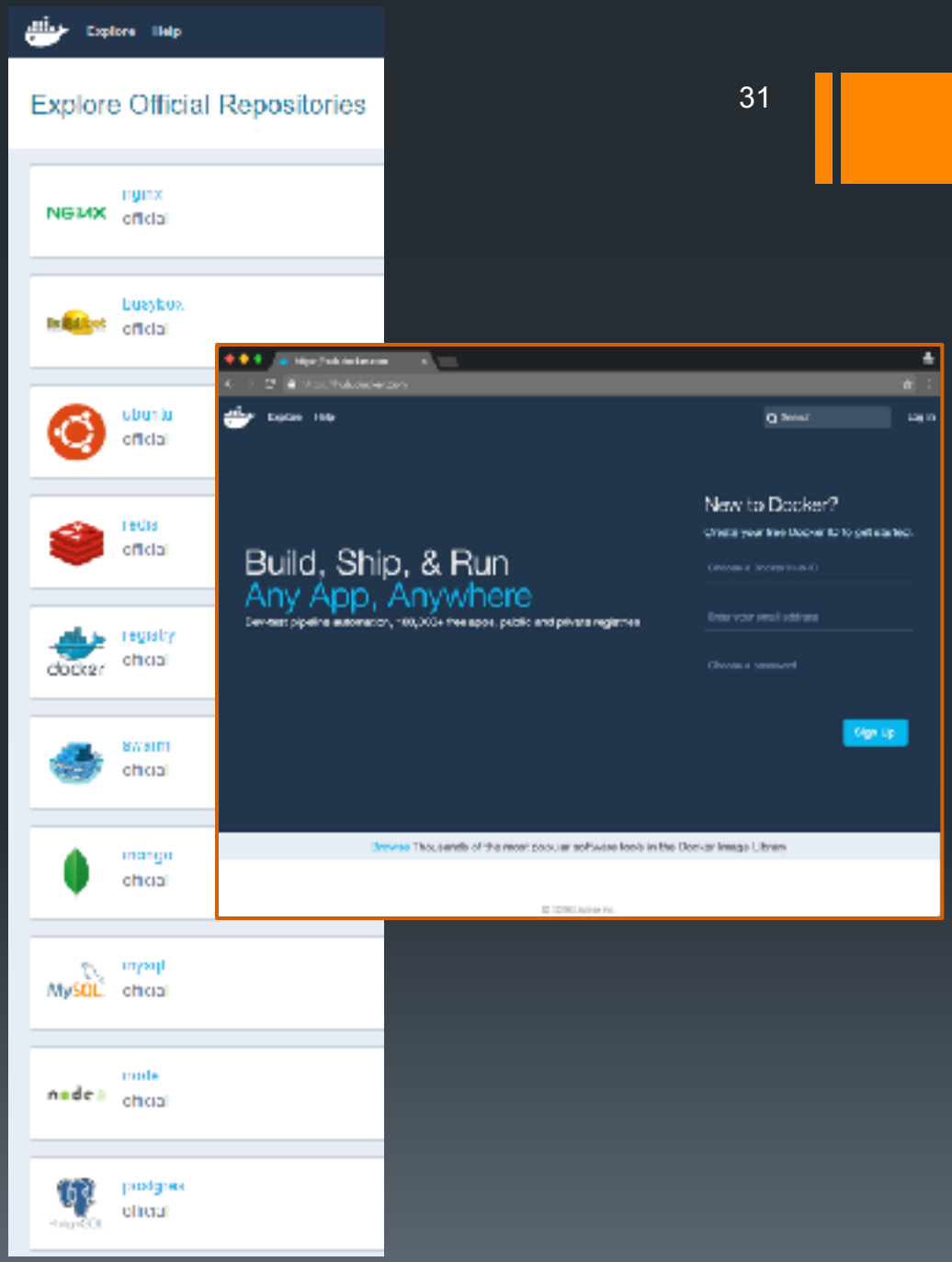
```
user@ubuntu:~$ docker container run -it cirros /bin/sh
Unable to find image 'cirros:latest' locally
latest: Pulling from library/cirros
a3ed95caeb02: Pull complete
8c4568d48636: Pull complete
e6cc72aea3e6: Pull complete
b5a1edf1e876: Pull complete
Digest: sha256:9aa75497b46cc15cccceef625acee6017d7f3e78db9bd5f7b6b933feaa38e3ae
Status: Downloaded newer image for cirros:latest
/ # hostname
57aed463f5b9
/ # cat /etc/os-release
NAME=Buildroot
VERSION=2012.05
ID=buildroot
VERSION_ID=2012.05
PRETTY_NAME="Buildroot 2012.05"
/ # exit
user@ubuntu:~$ cat /etc/os-release
NAME="Ubuntu"
VERSION="16.04.1 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04.1 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
UBUNTU_CODENAME=xenial
```

```
user@ubuntu:~$ docker container run --help

Usage:  docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]

Run a command in a new container

Options:
      --add-host list                    Add a custom host-to-IP mapping (host:ip) (default []) 
  -a, --attach list                      Attach to STDIN, STDOUT or STDERR (default []) 
      --blkio-weight uint16              Block IO (relative weight), between 10 and 1000, or 0 to disable (default 0) 
      --blkio-weight-device weighted-device Block IO weight (relative device weight) (default []) 
      --cap-add list                     Add Linux capabilities (default []) 
      --cap-drop list                    Drop Linux capabilities (default []) 
```

Docker uses the specified image to create a new container with an isolated filesystem, IPC, process environment, and network namespace

The Docker Hub

- Docker Hub is a registry supporting dynamic container image downloads
  - A massive library of open source
- Docker Hub hosts base images
  - Cirros, Ubuntu, Fedora, Alpine, Debian, Centos, ...
  - These images can be used as the basis for building custom application service images
- Docker Hub hosts state stores
  - MongoDB, Cassandra, Redis, Postgres, MySQL, Couchbase, …
- Docker Hub hosts web servers
  - Apache Httpd, Nginx, Tomcat, Tomee, Glassfish, …
- Docker Hub hosts message brokers
  - Nats, Kafka, RabbitMQ, ActiveMQ, …
- Docker Hub hosts dev platforms
  - Java:5, Java:6, Java:7, Java:8, Java:9, NodeJS, Go, Rust, Ruby, Python, Erlang, Haskell, Swift, …
- And more …



Exploring a Centos Container

- The base Centos image is 193MB, this is less than half the size of a base Centos server VM
- Running containers share the host kernel but supply their own OS personality
  - Libraries, executables, configuration files, etc
- Running containers also have their own network interfaces

```
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
user@ubuntu:~$ docker container run -it centos /bin/bash
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
785fe1d06b2d: Pull complete
Digest: sha256:be5b4e93f116e57eb3fd454ede72421eec892e3e4925627ec9e44f65fcd69cf8
Status: Downloaded newer image for centos:latest
[root@d8c165c8c75f /]# uname -a
Linux d8c165c8c75f 4.4.0-66-generic #87-Ubuntu SMP Fri Mar 3 15:29:05 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
[root@d8c165c8c75f /]# cat /etc/os-release
NAME="CentOS Linux"
VERSION="7 (Core)"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="7"
PRETTY_NAME="CentOS Linux 7 (Core)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:7"
HOME_URL="https://www.centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"

CENTOS_MANTISBT_PROJECT="CentOS-7"
CENTOS_MANTISBT_PROJECT_VERSION="7"
REDHAT_SUPPORT_PRODUCT="centos"
REDHAT_SUPPORT_PRODUCT_VERSION="7"

[root@d8c165c8c75f /]#
```

```
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
d8c165c8c75f        centos              "/bin/bash"         About a minute ago  Up About a minute                       gallant_saha
user@ubuntu:~$ docker container inspect gallant_saha -f '{{.NetworkSettings.Networks.bridge.IPAddress}}'
172.17.0.2
user@ubuntu:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether [illegible]:3c brd ff:ff:ff:ff:ff:ff
    inet 192.168.225.154/24 brd 192.168.225.255 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe08:948c/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:04:18:37:81 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:64ff:fe10:37e1/64 scope link
       valid_lft forever preferred_lft forever
13: veth369b0ed@if12: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether be:66:9a:47:da:30 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::bc00:9aff:fe47:da30/64 scope link
       valid_lft forever preferred_lft forever
user@ubuntu:~$
```

Container Isolation

- Images create copy-on-write container layers
- Changes made to containers are isolated from the host and other container instances
  - e.g. installing vim in a container makes it available to that container only
  - The host will not have vim installed
  - Other containers generated from the same image will not have vim installed
- This ensures that every container generated from a given image shares the same predictable and repeatable initial state

```
[root@b0c79695c6c6 /]# yum install -y vim
Loaded plugins: fastestmirror, ovl
Loading mirror speeds from cached hostfile
 * base: centos.mirror.ndchost.com
 * extras: mirrors.xmission.com
 * updates: linux.mirrors.es.net
Package 2:vim-enhanced-7.4.160-1.el7_3.1.x86_64 already installed and latest version
Nothing to do
[root@b0c79695c6c6 /]# which vim
/usr/bin/vim
[root@b0c79695c6c6 /]# exit
exit
user@ubuntu:~$ docker container run -it centos /bin/bash
[root@2d1be5e52bf3 /]# vim
bash: vim: command not found
[root@2d1be5e52bf3 /]# exit
exit
user@ubuntu:~$ vim
bash: /usr/bin/vim: No such file or directory
```

Listing Containers

- Containers have (at least) 3 Names
  - ID
  - Container Name
  - Hostname
- Docker commands can reference containers by ID or Container Name
  - Short versions of the IDs are displayed by default
  - Container names:
    - Are generated if not provided (random adjective + _ + random last name)
    - `docker container run --name` switch allows the name to be set explicitly
    - `docker container rename` command allows you to rename an existing container
  - Container names must be unique within the scope of a particular Docker daemon
- Containers use their container ID as their Hostname by default
  - The container Hostname can be set with the `-h` (or `--hostname`) switch
- The `docker container ls` subcommand displays <u>running</u> containers
  - The `-a` switch (or `--all`) displays all containers (running and stopped)

```
[root@7376ca589fc2 /]# cat /etc/hostname
7376ca589fc2
[root@7376ca589fc2 /]#
```

```
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
7376ca589fc2        centos              "/bin/bash"         2 minutes ago       Up 2 minutes                            suspicious_bardeen
user@ubuntu:~$
```

```
user@ubuntu:~$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS               NAMES
7376ca589fc2        centos              "/bin/bash"         4 minutes ago       Up 4 minutes                                    suspicious_bardeen
2d1be5e52bf3        centos              "/bin/bash"         9 minutes ago       Exited (127) 8 minutes ago                      xenodochial_sinoussi
b8c79695c6c6        centos              "/bin/bash"         10 minutes ago      Exited (0) 9 minutes ago                        tender_keller
7edc8d2f9704        centos              "/bin/bash"         13 minutes ago      Exited (127) 11 minutes ago                     lucid_meitner
2baf13131db7        centos              "/bin/bash"         15 minutes ago      Exited (127) 15 minutes ago                     festive_feynman
dc3b8a75d910        centos              "/bin/bash"         16 minutes ago      Exited (0) 15 minutes ago                       fervent_lewin
d8c165c8c75f        centos              "/bin/bash"         35 minutes ago      Exited (137) 16 minutes ago                     gallant_saha
```

Filtering Container Lists

- The `-f` (`--filter`) switch allows you to filter `ls` output based on conditions
  - Conditions are provided as key=value pairs
  - You can pass multiple filter flags (e.g. `--filter "foo=bar" --filter "bif=baz"`)
- Supported filter tags:
  - `id` (container's id)
  - `label` (label=<key> or label=<key>=<value>)
  - `name` (container's name)
  - `exited` (int - the code of exited containers. Only useful with --all)
  - `status` (created|restarting|running|paused|exited|dead)
  - `ancestor` (<image-name>[:<tag>], <image id> or <image@digest>) - filters containers that were created from the given image or a descendant.
  - `before` (container's id or name) - filters containers created before given id or name
  - `since` (container's id or name) - filters containers created since given id or name
  - `isolation` (default|process|hyperv) (Windows daemon only)
  - `volume` (volume name or mount point) - filters containers that mount volumes.
  - `network` (network id or name) - filters containers connected to the provided network

```
user@ubuntu:~$ docker container ls -a
CONTAINER ID   IMAGE    COMMAND                   CREATED             STATUS                      PORTS   NAMES
6b4b4604a141   alpine   "sh"                      3 seconds ago       Created
modest_hawking
d56b7c2999d5   ubuntu   "/bin/bash"               29 seconds ago      Exited (0) 28 seconds ago
pensive_turing
184c60beee53   alpine   "sh -c 'while true; d"    About an hour ago   Up About an hour
Zugspitze

user@ubuntu:~$ docker container ls -f status=created
CONTAINER ID   IMAGE    COMMAND   CREATED          STATUS    PORTS   NAMES
```

Filtering Ancestry

- Filtering by container image is particularly useful
- `-f ancestor=<image name>`
  - The image name must be an exact match of the name used to launch the container (and displayed by `docker container ls`)

```
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE   COMMAND                  CREATED         STATUS          PORTS                   NAMES
f90a992e545c        nginx   "nginx -g 'daemon off"   5 minutes ago   Up 5 minutes    80/tcp, 443/tcp
elated_jepsen
f5df48bbae86        nginx   "nginx -g 'daemon off"   6 minutes ago   Up 6 minutes    80/tcp, 443/tcp
lonely_albattani
d042fea85262        ubuntu  "/bin/bash"              14 minutes ago  Up 14 minutes
prickly_dubinsky
886a27f9b99c        ubuntu  "/bin/bash"              15 minutes ago  Up 15 minutes
infallible_rosalind

user@ubuntu:~$ docker container ls -f ancestor=nginx
CONTAINER ID        IMAGE   COMMAND                  CREATED         STATUS          PORTS                   NAMES
f90a992e545c        nginx   "nginx -g 'daemon off"   6 minutes ago   Up 6 minutes    80/tcp, 443/tcp
elated_jepsen
f5df48bbae86        nginx   "nginx -g 'daemon off"   6 minutes ago   Up 6 minutes    80/tcp, 443/tcp
lonely_albattani
```

Filtering with Labels

- Containers can be assigned arbitrary labels (key/value strings) using the `--label` switch with `docker container run`
- Containers can then be listed with `docker container ls` based on labels
  - You can list containers with a given key
  - You can also list containers with a given key and a given value

```
user@ubuntu:~$ docker container run -itd --label loc=dallas ubuntu
886a27f9b99cb46aadc58b894952731f762f1e2a4017d217c3723a95f120b637

user@ubuntu:~$ docker container run -itd --label loc=portland ubuntu
d042fea85262911863bc0c09f6c97433f612bf3c9c8eef4992737e15976bd169

user@ubuntu:~$ docker container ls -f label=loc
CONTAINER ID   IMAGE    COMMAND       CREATED               STATUS               PORTS     NAMES
d042fea85262   ubuntu   "/bin/bash"   About a minute ago    Up About a minute              prickly_dubinsky
886a27f9b99c   ubuntu   "/bin/bash"   About a minute ago    Up About a minute
infallible_rosalind

user@ubuntu:~$ docker container ls -f="label=loc=dallas"
CONTAINER ID   IMAGE    COMMAND       CREATED               STATUS               PORTS     NAMES
886a27f9b99c   ubuntu   "/bin/bash"   About a minute ago    Up About a minute
infallible_rosalind

user@ubuntu:~$ docker container ls
CONTAINER ID   IMAGE    COMMAND       CREATED               STATUS               PORTS     NAMES
d042fea85262   ubuntu   "/bin/bash"   About a minute ago    Up About a minute              prickly_dubinsky
886a27f9b99c   ubuntu   "/bin/bash"   About a minute ago    Up About a minute
infallible_rosalind
2ec0f1d5e892   ubuntu   "/bin/bash"   22 minutes ago        Up 22 minutes                  mad_blackwell
```

Formatting `ls` Output

- The `docker container ls --format` switch allows you to customize the `ls` output
- The format spec must be a valid Go template
  - Adding the "table" prefix displays column headings
- Fields supported:
  - `.ID` Container ID
  - `.Image` Image ID
  - `.Command` Quoted command
  - `.CreatedAt` Time when the container was created
  - `.RunningFor` Elapsed time since the container was started
  - `.Ports` Exposed ports
  - `.Status` Container status
  - `.Size` Container disk size
  - `.Names` Container names
  - `.Labels` All labels assigned to the container
  - `.Label` Value of a specific label for this container, e.g. '{{.Label "com.docker.swarm.cpu"}}'
  - `.Mounts` Names of the volumes mounted in this container

Go Template Documentation
https://golang.org/pkg/text/template/

```
user@ubuntu:~$ docker container ls --format "table {{.ID}}\t{{.Names}}
\t{{.Labels}}"
CONTAINER ID        NAMES                   LABELS
f90a992e545c        elated_jepsen
f5df48bbae86        lonely_albattani
d042fea85262        prickly_dubinsky        loc=portland
886a27f9b99c        infallible_rosalind     loc=dallas
```

Removing Containers

- The `docker container rm` command removes containers from a host
  - `-f` (`--force`) Force the removal of a running container (uses SIGKILL)

- Prefer `$(docker container ls -aq)` over `` `docker container ls -aq` ``
  - http://mywiki.wooledge.org/BashFAQ/082
  - http://pubs.opengroup.org/onlinepubs/9699919799/xrat/V4_xcu_chap02.html#tag_23_02_06_03

```
user@ubuntu:~$ docker container rm $(docker container ls -aq)
b0c79695c6c6
7edc8d2f9704
2baf13131db7
dc380a75d910
d8c165c0c75f
user@ubuntu:~$
```

```
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
user@ubuntu:~$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS               NAMES
7376ca589fc2        centos              "/bin/bash"         17 minutes ago      Exited (0) 17 seconds ago                       suspicious_bardeen
2d1be5e52bf3        centos              "/bin/bash"         22 minutes ago      Exited (127) 22 minutes ago                     xenodochial_sinoussi
b0c79695c6c6        centos              "/bin/bash"         24 minutes ago      Exited (0) 22 minutes ago                       tender_keller
7edc8d2f9704        centos              "/bin/bash"         27 minutes ago      Exited (127) 25 minutes ago                     lucid_meitner
2baf13131db7        centos              "/bin/bash"         29 minutes ago      Exited (127) 28 minutes ago                     festive_feynman
dc380a75d910        centos              "/bin/bash"         30 minutes ago      Exited (0) 29 minutes ago                       fervent_lewin
d8c165c0c75f        centos              "/bin/bash"         48 minutes ago      Exited (137) 38 minutes ago                     gallant_saha
user@ubuntu:~$ docker container rm xenodochial_sinoussi
xenodochial_sinoussi
user@ubuntu:~$ docker container rm 7376ca589fc2
7376ca589fc2
user@ubuntu:~$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS               NAMES
b0c79695c6c6        centos              "/bin/bash"         25 minutes ago      Exited (0) 24 minutes ago                       tender_keller
7edc8d2f9704        centos              "/bin/bash"         28 minutes ago      Exited (127) 26 minutes ago                     lucid_meitner
2baf13131db7        centos              "/bin/bash"         30 minutes ago      Exited (127) 30 minutes ago                     festive_feynman
dc380a75d910        centos              "/bin/bash"         31 minutes ago      Exited (0) 31 minutes ago                       fervent_lewin
d8c165c0c75f        centos              "/bin/bash"         50 minutes ago      Exited (137) 32 minutes ago                     gallant_saha
user@ubuntu:~$
```

Summary

- The Docker platform includes several parts
  - Docker daemon (docker engine)
  - Docker client
  - Registries
  - Images
  - Containers
- Containers are spawned from images and are isolated from each other and the host operating system
- The docker command line tool provides an array of commands for starting, examining and controlling containers
- You can control the output rows displayed by `docker container ls` with the `--filter` switch
- You can control the output columns displayed by `docker container ls` with the `--format` switch

Lab 2

- Running containers

3: Controlling Containers

Objectives

- Explain the difference between `create` and `run`
- Describe the process used to copy files between a host and a container
- Explore the details of starting, stopping, pausing and unpausing containers
- Learn how to attach and detach from containers
- Gather container information using `ls`, `stats` and `top`
- Execute commands within a container using `nsenter` and `docker container exec`
- Run containers as services and inspect their log output

Creating Containers

- The `docker container create` command creates a container
  - Creating the container filesystem from the specified image
  - Saving the container's metadata (typically in a config.json)
- The container ID is printed to STDOUT
- Similar to *`docker container run`* except the container is never started
  - *`docker container run`* is actually just a wrapper for create and start
- You can use `docker container start <container_id/name>` to start the container
- This is useful when you want to set up a container configuration ahead of time so that it is ready to start when you need it
- The initial status of the new container is created
- All of the switches supported by the `run` command can be applied to the `create` command
  - The metadata configuration file captures all of the (10s of) create/run configuration options
  - The `start` command takes no container configuration switches, it simply runs the container as per the configuration
    - You can specify how you would like your local terminal to be connected to the container with start (e.g. `-i`)

```
user@ubuntu:~$ docker container create --name test99 -h nodea -t nginx
e76f0fdf16a8273e8335855e7fe0bb339306344fad8b3c4f70c1747688738128
user@ubuntu:~$ docker container ls
CONTAINER ID   IMAGE    COMMAND                   CREATED          STATUS         PORTS
NAMES
user@ubuntu:~$ docker container ls -a
CONTAINER ID   IMAGE    COMMAND                   CREATED          STATUS         PORTS
NAMES
e76f0fdf16a8   nginx    "nginx -g 'daemon off"    30 seconds ago   Created
test99
user@ubuntu:~$ docker container start test99
test99
user@ubuntu:~$ docker container ls
```

Copying to/from Containers

- `docker container cp <container>:<path> <hostdir>`
- `docker container cp <hostdir> <container>:<path>`
  - Host to container direction added in Docker 1.8
- Note: `cp` can copy to/from any container (it does not have to be running)
- A dash ("–") can be used to `cp` a tar archive to STDOUT or from STDIN

```
user@ubuntu:~$ docker container create --name=zugspitze alpine /usr/bin/tail –f /dev/null
user@ubuntu:~$ echo "From the host" > host.md

user@ubuntu:~$ docker container cp ./host.md zugspitze:/test.md

user@ubuntu:~$ docker container cp zugspitze:/test.md ./cont.md
user@ubuntu:~$ cat cont.md
From the host

user@ubuntu:~$ docker container cp zugspitze:/etc - > zsetcdir.tar
user@ubuntu:~$ tar -tf zsetcdir.tar | grep host
etc/hostname
etc/hosts
```

Container Size

- The `docker container ls -s` switch can be used to display the size of the container's writable layer
  - The virtual size includes the size of the image stack the container is based on

```
user@ubuntu:~$ docker container ls -s
CONTAINER ID   IMAGE    COMMAND       CREATED          STATUS          PORTS   NAMES               SIZE
2ec0f1d5e892   ubuntu   "/bin/bash"   14 minutes ago   Up 14 minutes           mad_blackwell       29 B (virtual
126.4 MB)
8faa3d0adf53   ubuntu   "/bin/bash"   14 minutes ago   Up 14 minutes           prickly_wright      0 B (virtual
126.4 MB)

user@ubuntu:~$ docker container exec mad_blackwell sh -c 'echo Hello > readme'

user@ubuntu:~$ docker container ls -s
CONTAINER ID   IMAGE    COMMAND       CREATED          STATUS          PORTS   NAMES               SIZE
2ec0f1d5e892   ubuntu   "/bin/bash"   14 minutes ago   Up 14 minutes           mad_blackwell       35 B (virtual
126.4 MB)
8faa3d0adf53   ubuntu   "/bin/bash"   14 minutes ago   Up 14 minutes           prickly_wright      0 B (virtual
126.4 MB)
```

Controlling Containers

- `docker container start`
  - You can resume a stopped container using the start sub command
  - To reconnect your local shell input stream to the container use the `-i` switch
- `docker container restart`
  - The restart command stops the specified container and then runs it again
  - If the container is not already running, restart simply starts it
- `docker container stop`
  - You can stop a running container with the stop command
  - Sends SIGINT, waits 10 seconds, sends SIGKILL
  - You can set the delay in seconds between the two SIGs with the `-t` switch (also works with restart)

```
user@ubuntu:~$ docker container create --name=zugspitze alpine /usr/bin/tail -f /dev/null
3d232d3591cf900a5d6d6040e61bc46840b07785e001ed25d9c3f958cb9bf164
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
NAMES
user@ubuntu:~$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
NAMES
3d232d3591cf        alpine              "/usr/bin/tail -f /de"   7 seconds ago       Created
zugspitze
user@ubuntu:~$ docker container start zugspitze
zugspitze
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
NAMES
3d232d3591cf        alpine              "/usr/bin/tail -f /de"   22 seconds ago      Up 6 seconds
zugspitze
user@ubuntu:~$ time docker container stop zugspitze
zugspitze

real           0m10.201s
user           0m0.004s
sys            0m0.004s
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
NAMES
user@ubuntu:~$ docker container restart zugspitze
zugspitze
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
NAMES
3d232d3591cf        alpine              "/usr/bin/tail -f /de"   2 minutes ago       Up 4 seconds
zugspitze
user@ubuntu:~$ docker container restart zugspitze
dockerzugspitze
user@ubuntu:~$ docker container ls
```

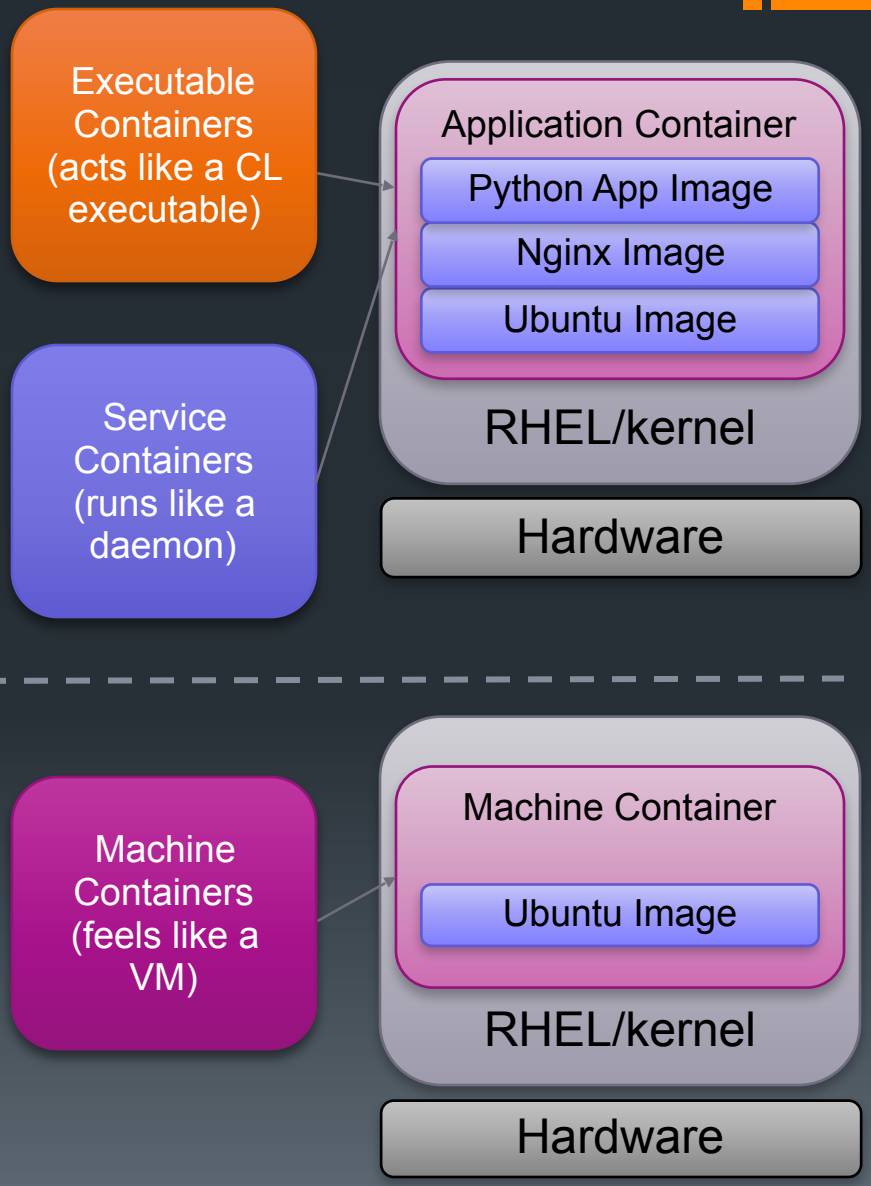
Attach

- `docker container attach`
  - You can attach to a running container using the attach subcommand
    - This attaches your shell to PID 1 in the container
    - Not all programs running in a container offer a useful console
  - The `^p ^q` sequence detaches from a container (if you are attached to a shell in the container)
    - The `--detach-keys` switch sets a different detach sequence

```
user@ubuntu:~$ docker container run -itd ubuntu:12.04
7f5a4ebf67ac551f2ce197b2cd493217b193231f707a1a1bfefa803cd37e8f36

user@ubuntu:~$ docker container attach 7f5a
root@7f5a4ebf67ac:/# ps
   PID TTY          TIME CMD
     1 ?        00:00:00 bash
    10 ?        00:00:00 ps

root@7f5a4ebf67ac:/# read escape sequence (CTRL-p CTRL-q pressed)

user@ubuntu:~$ docker container attach --detach-keys="ctrl-a,ctrl-a" 7f5a
root@7f5a4ebf67ac:/# ps
   PID TTY          TIME CMD
     1 ?        00:00:00 bash
    11 ?        00:00:00 ps
root@7f5a4ebf67ac:/# read escape sequence (CTRL-a CTRL-a pressed)

user@ubuntu:~$
```

Types of Containers

- Containers can be organized into categories designating their purpose
  - Application Containers
    - Executable containers
      - `$ docker container run thrift --gen cpp myidl.thrift`
      - Containers of this type are designed to distribute command line tools
      - The advantage is that the container allows the binary to run on any flavor on Linux with Docker installed without building from source or installing an interpreter
    - Service Containers
      - `$ docker container run -d nginx`
      - Containers of this type are designed to house application services
      - Simplifies application deployment
  - Machine Containers
    - `$ docker container run -it centos`
    - Containers of this type are designed to house the non kernel elements of a Linux distro
    - Allows you to run and test any Linux distro on a single Linux base OS, also used as a base for service images
    - Can be deployed to support admin tasks



Containers as Services

- Containers are commonly executed in the background as daemons on servers
- The `-d` (`--detach`) switch runs a container in daemon mode
  - daemonized containers do not have an interactive session

```
user@ubuntu:~$ docker container run --name hellosvc -d cirros /bin/sh -c "while true; do echo hello world; sleep 1;
done"
01bcd51a204dba0358c4b6f6d08a9173f5dce5c82bf95516904ce786272ba769

user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE       COMMAND                  CREATED             STATUS              PORTS
NAMES
01bcd51a204d        cirros      "/bin/sh -c 'while..."   4 seconds ago       Up 3 seconds
hellosvc
```

Container Logs

- The `docker container logs` subcommand displays the specified container's STDOUT/STDERR log
  - The `logs` command supports the `--tail` switch
    - Tail will display the last n lines of the log
    - Adding `-f` (`--follow`) will tail continuously
    - Adding `-t` (`--timestamps`) will display the log entry timestamp
      - `--since` can be used to display events more recent than a given timestamp
        - e.g. 2013-01-02T13:23:37 or relative: 42m for 42 minutes
    - The `--details` switch will display log-options set by the user at run time

```
user@ubuntu:~$ docker container logs --help

Usage:  docker container logs [OPTIONS] CONTAINER

Fetch the logs of a container

Options:
      --details        Show extra details provided to logs
  -f, --follow         Follow log output
      --help           Print usage
      --since string   Show logs since timestamp
      --tail string    Number of lines to show from the end of the logs (default "all")
  -t, --timestamps     Show timestamps
```

```
user@ubuntu:~$ docker container logs --tail 3 hellosvc
hello world
hello world
hello world
user@ubuntu:~$ docker container logs --tail 0 -f hellosvc
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
```

```
user@ubuntu:~$ docker container run --name hellosvc -d cirros /bin/sh -c "while true; do echo hello world; sleep 1; done"
016bb141e7Bacb292a1e9b57ef30a24dbaa71Be21eac9031cb3fd52d41becb70
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
016bb141e7Ba        cirros              "/bin/sh -c 'while..."   6 seconds ago       Up 5 seconds                            hellosvc
user@ubuntu:~$
user@ubuntu:~$ docker container logs hellosvc
hello world
hello world
hello world
hello world
hello world
hello world
hello world
```

```
user@ubuntu:~$ docker container logs --tail 3 -t hellosvc
2017-03-22T02:39:41.484952508Z hello world
2017-03-22T02:39:42.486703716Z hello world
2017-03-22T02:39:43.488530360Z hello world
```

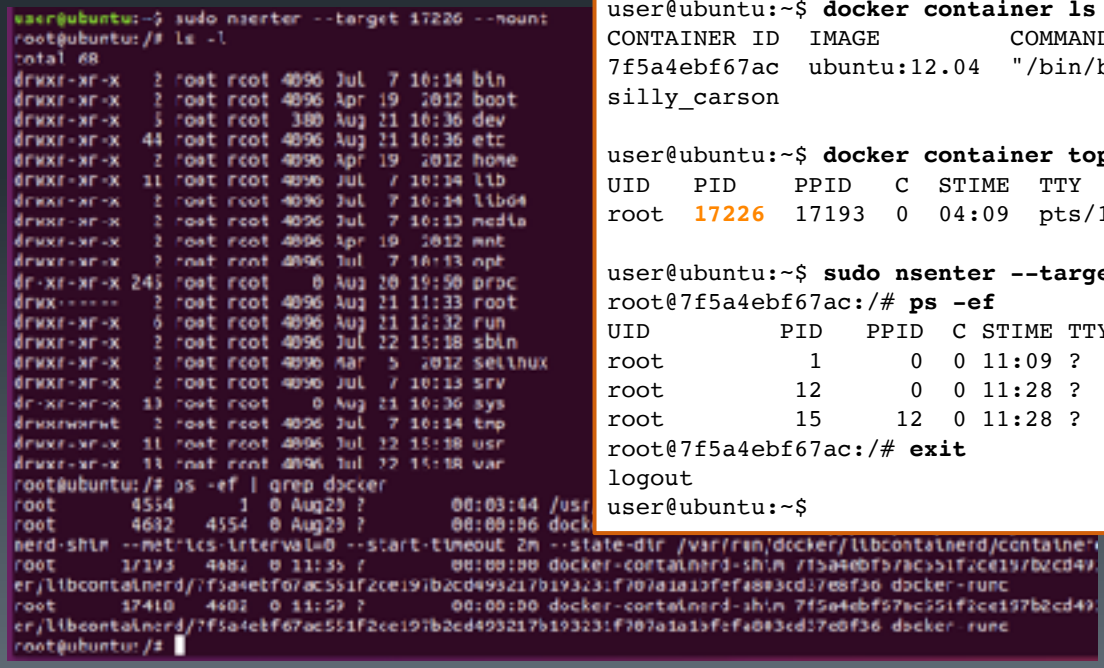
Pause and Unpause

- Pause suspends a container's execution but leaves it resident in memory
  - This stops the execution of all processes and threads within the container
- Unpause releases a container from the pause state

```
user@ubuntu:~$ docker container run -d --name=zugspitze alpine sh -c 'while true; do echo "hello world"; sleep
3; done'
184c60beee53016c990be5b6a580610a16d1e619b97048047bb960b17934064e
user@ubuntu:~$ docker container ls
CONTAINER ID   IMAGE    COMMAND                   CREATED         STATUS              PORTS      NAMES
184c60beee53   alpine   "sh -c 'while true; d"    3 seconds ago   Up 2 seconds                   zugspitze
user@ubuntu:~$ docker container logs zugspitze
hello world
hello world
hello world
hello world
user@ubuntu:~$ docker container pause zugspitze
zugspitze
user@ubuntu:~$ docker container logs zugspitze | wc -l
8
user@ubuntu:~$ sleep 5
user@ubuntu:~$ docker container logs zugspitze | wc -l
8
user@ubuntu:~$ docker container ls
CONTAINER ID   IMAGE    COMMAND                   CREATED         STATUS                   PORTS   NAMES
184c60beee53   alpine   "sh -c 'while true; d"    52 seconds ago  Up 51 seconds (Paused)           zugspitze
user@ubuntu:~$ docker container unpause zugspitze
zugspitze
user@ubuntu:~$ docker container ls
CONTAINER ID   IMAGE    COMMAND                   CREATED               STATUS             PORTS   NAMES
184c60beee53   alpine   "sh -c 'while true; d"    About a minute ago    Up About a minute          zugspitze
user@ubuntu:~$ docker container logs zugspitze | wc -l
12
user@ubuntu:~$
```

- `docker container exec <container> <cmd>` (added in docker 1.3)
  - Runs an arbitrary process within a container (all namespaces and cgroups in force)
    - `-d` runs new process detached
    - `-i` runs new process in interactive mode
- Linux also offers the *nsenter* command
  - Enters the namespaces of another process and then executes the specified program
  - Allows you to choose the namespace(s) to enter and ignores cgroup constraints of the target process

```
user@ubuntu:~$ docker container exec 7f5a cat /etc/os-release
NAME="Ubuntu"
VERSION="12.04.5 LTS, Precise Pangolin"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu precise (12.04.5 LTS)"
VERSION_ID="12.04"

user@ubuntu:~$ docker container exec -it 7f5a /bin/bash
root@7f5a4ebf67ac:/# ps
   PID TTY          TIME CMD
    27 ?        00:00:00 bash
    37 ?        00:00:00 ps
root@7f5a4ebf67ac:/# exit
exit

user@ubuntu:~$ docker container ls
CONTAINER ID   IMAGE          COMMAND       CREATED          STATUS          PORTS   NAMES
7f5a4ebf67ac   ubuntu:12.04   "/bin/bash"   50 minutes ago   Up 50 minutes
silly_carson

user@ubuntu:~$ docker container top 7f5a
UID    PID     PPID    C   STIME   TTY      TIME       CMD
root   17226   17193   0   04:09   pts/12   00:00:00   /bin/bash

user@ubuntu:~$ sudo nsenter --target 17226 --mount --uts --ipc --net --pid
root@7f5a4ebf67ac:/# ps -ef
UID          PID   PPID  C STIME TTY          TIME CMD
root           1      0  0 11:09 ?        00:00:00 /bin/bash
root          12      0  0 11:28 ?        00:00:00 -bash
root          15     12  0 11:28 ?        00:00:00 ps -ef
root@7f5a4ebf67ac:/# exit
logout
user@ubuntu:~$
```



Container Exec

Top

- Just like the old `docker ps` command should have been called `docker ls` … (now *`docker container ls`* is the go-forward command)
- `docker container top` (should have been called `docker ps`)
  - You can examine the processes running within a container using the top subcommand
    - Top accepts standard ps switches
  - Note that containers provide a passive operating system and library interface with no running processes or services enabled by default
  - Docker containers supply processes with a PID namespace allowing them to see only the other processes within the container
  - Process IDs within the container start with 1 and are mapped to available PIDs on the host
    - `ps` inside the container shows PID namespace local PIDs and `top` shows host relative PIDs

```
user@ubuntu:~$ docker container top hellosvc
UID                 PID                 PPID                C                   STIME               TTY                 TIME                CMD
root                8854                8827                0                   19:42               ?                   00:00:00            /bin/sh -c w
hile true; do echo hello world; sleep 1; done
root                9552                8854                0                   19:52               ?                   00:00:00            sleep 1
user@ubuntu:~$
```

```
user@ubuntu:~$ docker container exec 29b7 ps -eo user,pid,ppid
USER         PID    PPID
root           1       0
root           9       0
root          19       0
user@ubuntu:~$
```

```
user@ubuntu:~$ docker container top --help

Usage:  docker container top CONTAINER [ps OPTIONS]

Display the running processes of a container

Options:
      --help   Print usage
user@ubuntu:~$ docker container top 29b7
UID                 PID                 PPID                C                   STIME               TTY                 TIME                CMD
root                10531               10494               0                   19:57               pts/1               00:00:00            /bin/bash
root                10987               10970               0                   20:02               pts/13              00:00:00            /bin/bash
user@ubuntu:~$ docker container top 29b7 -eo user,pid,netns
USER                PID                 NETNS
root                10531               4026532532
root                10987               4026532532
user@ubuntu:~$
```

Stats

- `docker container stats` (added in docker 1.5)
  - The `stats` subcommand displays a live status view of the specified containers
  - The `--no-stream` switch executes a one shot stats
  - The `-a` switch displays states for all containers
    - Without arguments stats displays all running containers
  - Stats shows summary data for the entire container (not by process)

```
user@ubuntu:~$ docker container stats hellosvc

CONTAINER           CPU %               MEM USAGE / LIMIT     MEM %               NET I/O             BLOCK I/O           PIDS
hellosvc            0.05%               128 KiB / 1.936 GiB   0.01%               2.98 kB / 648 B     0 B / 0 B           2
```

```
user@ubuntu:~$ docker container stats hellosvc --no-stream
CONTAINER           CPU %               MEM USAGE / LIMIT     MEM %               NET I/O             BLOCK I/O           PIDS
hellosvc            0.07%               128 KiB / 1.936 GiB   0.01%               2.98 kB / 648 B     0 B / 0 B           2
user@ubuntu:~$
```

Summary

- Containers can be created prior to starting them
- A container need not be running to copy files in and out of it
- Containers can be started multiple times
- *docker container stop* sends signals to the container's PID 1 process
- You can *attach* to and *detach* from a containers PID 1 process
- *docker container ls* and *top* tools allow you to display custom data describing containers
- *docker container exec* allows you to execute arbitrary processes within a container
- *nseneter* allows you to execute processes that use only the specified namespaces of the container
- *docker container logs* allows you to view the log output of containers

Lab 3

- Controlling containers

4: Advanced Container Operations

Objectives

- Become familiar with the evolution of Docker and container technology
- Describe the relationship between Docker and the OCI
- List the OCI components of the Docker daemon stack
- Describe the Docker daemon and client configuration options
- Explore Docker container metadata
- Work with Docker environment variables, signals and events
- Use container restart policies

Container & Docker Timeline

- 2005 OpenVZ (Previously Parallels [SWsoft] Virtuozzo containers) open sourced [Linux 2.6.11]
- 2006 Google (Menage/Seth) add Process Containers to Linux (now called CGroups) [Linux 2.6.15]
- 2007 Google uses cgroups to containerize search [Linux 2.6.20]
- 2008 LXC version 0.1.0 released (basic functions) [Linux 2.6.24]
- 2011 Container Unification agreement during kernel summit (In tree unified CGroups and Namespaces resulted)
  - dotCloud raises $12mm Series A to create a PaaS framework
- 2013 First Linux kernel with full container support for all container systems (LXC, Docker, OpenVZ, etc.) [Linux 3.10 & 3.12 LTS]
  - dotCloud pivots, becoming Docker Inc. and adds Docker support to OpenStack Havana (not part of Icehouse+); Google ComputeEngine adds Docker support
- 2014-01 – Docker Inc. raises $15M series B
- 2014-04 – Docker Governance Advisory Board (DGAB), a step toward Docker open governance
  - Amazon integrates Docker with Elastic Beanstalk
  - Ubuntu 14.04 ships with native Docker packages
  - RHEL 7 offers Docker support and Red Hat certified containers
- 2014-06 – Docker 1.0 released, commercial support by Docker Inc. and 1st DockerCon
- 2014-07 – Docker 1.1 released (Docker Engine + Docker Hub + APIs + expanded ecosystem)
  - Docker Inc. acquires Orchard (Fig renamed Compose)
- 2014 Linux Distros began enabling user namespaces [Linux 3.14 & 3.18 LTS]
- 2014-08 – Docker 1.2 general improvements & Docker VMware partnership announced
- 2014-09 – Docker Inc. raises $40M series C
- 2014-10 – Docker 1.3 released with support for signed images & Docker Microsoft partnership announced
- 2014-11 – Amazon announces Docker container support in EC2 Container Service
- 2014-12 – Docker 1.4 released to improve stability
- 2015-02 – Docker 1.5 released with support for IPv6 and read-only containers
  - Docker releases Orchestration tools: Machine, Swarm and Compose
- 2015-04 – Docker 1.6 with Registry 2.0 and Windows Client Preview
  - Docker raises $95mm series D [> $1bn valuation] - Insight, Coatue, Goldman Sachs, Northern Trust, Benchmark, Greylock, Sequoia, Trinity & AME
  - Docker Acquires Kitematic (Mac GUI) and SocketPlane (Container Networking)
- 2015-06 – Docker 1.7 released with many performance tweaks
  - Linux Foundation - Open Container Initiative [OCI] with libcontainer as base
- 2015-07 – Docker 1.7.1 released with several important bug fixes
  - FreeBSD announces Docker support via 64bit Linux compatibility layer
  - Linux Foundation - Container Native Computing Foundation [CNCF] with Kubernetes as base
- 2015-08 – Docker 1.8 released with support for pluggable volume backends
  - 1.8 bug corrupts images pushed with multiple tags, 1.8.1 fix a week later, other fixes/security patches lead to 1
- 2015-10 – Docker Inc. acquires DevOps/Container orchestration and management platform TuTum
- 2015-11 – Docker 1.9 released with support for multi-host networking and named volumes
- 2016-01 – Docker acquires Unikernel Systems (folks from the Xen Project)
- 2016-02 – Docker 1.10 released with support for user namespaces, SecComp and DNS container name resolution
- 2016-03 – Docker Cloud released with automated deployment to AWS, DigitalOcean, SoftLayer and Azure
- 2016-04 – Docker v1.11 released with OCI support (built on containerd and runC)
- 2016-08 – Docker v1.12 released with integrated cluster management (Swarm Mode)
- 2017-02 – Docker v1.13 released docker plugin, docker secret, docker stacks with support for Compose on Swarm Mode
- 2017-03 – Docker v17.03-ce monthly release cycle, via two release channels (monthly and quarterly, security/bug-fixes vs features)
- 2017-06 – Docker v17.06-ce support for swarm-mode services with node-local networks, isolates Swarm control-plane traffic from data-plane traffic
- 2017-09 – Docker v17.09-ce promotes overlay2 graphdriver over aufs

Docker Inc. roadmap on GitHub (now Moby project) updated 2017-10 https://github.com/docker/docker/blob/master/ROADMAP.md

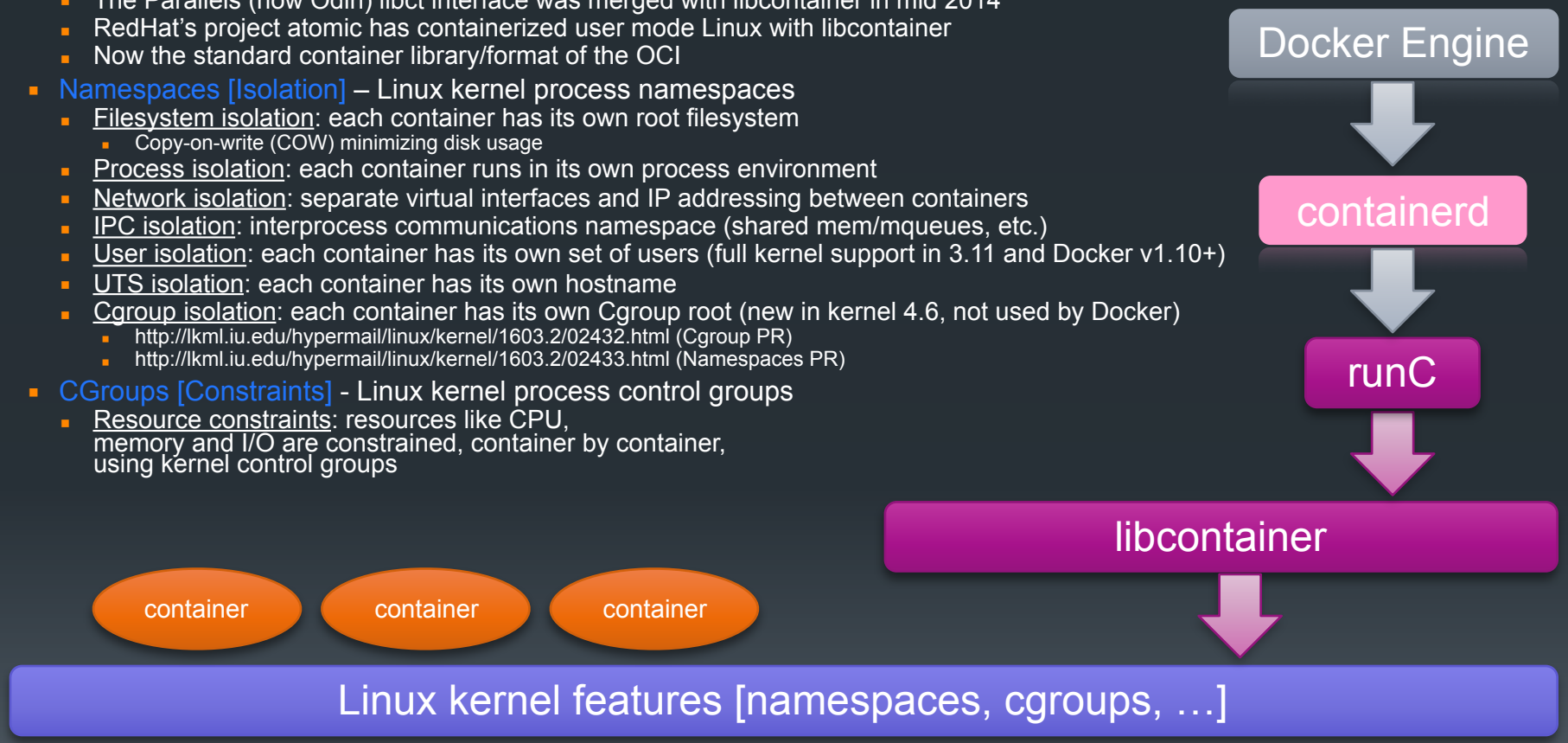
Container Standardization

- OCI [circa 6/2015]
  - Open Container Initiative
    - Formerly Open Container Project
    - A standard that ensures any container can run on any machines or cloud computing service
    - v1.0 released July 19, 2017
  - Run by the Linux Foundation
    - Non-profit which oversees the Linux open source operating system
  - Includes two specifications:
    - Runtime Specification (runtime-spec)
      - Outlines how to run a “filesystem bundle” that is unpacked on disk
    - Image Specification (image-spec)
      - An OCI implementation downloads an OCI Image, unpacks it, then runs it
  - Docker donated the container format, runtime code (libcontainer) and specifications
  - Members (All of the tech companies in the Fortune 100 except Apple, all of the key container startups):
    - Amazon, Anchore, Apcera, AT&T, Cisco, ContainerShip, CoreOS, DellEMC, Docker, EasyStack, Facebook, Fujitsu Limited, Goldman Sachs, Google, Hewlett Packard Enterprise, Huawei, IBM, Infoblox, Intel, Joyent, Kontena, Mesosphere, Microsoft, Midokura, Nutanix, Oracle, Pivotal, Polyverse, Portworx, Rancher Labs, Red Hat, Replicated, Resin.io, Robin Systems, SUSE, Sysdig, Twistlock, Twitter, Univa, Verizon Labs, Virtuozzo, VMware, Weaveworks, Wercker and WD

The Open Container Initiative (OCI) is a lightweight, open governance structure (project), formed under the auspices of the Linux Foundation, for the express purpose of creating open industry standards around container formats and runtime. The OCI was launched on June 22nd 2015 by Docker, CoreOS and other leaders in the container industry

Container Enablers

- Docker installations include the following elements:
  - **dockerd** [Docker daemon] – service performing high level container management
    - https://github.com/docker/docker-ce
  - **containerd** [runc manager] – daemon directly managing OCI containers (used by Docker v1.11+)
    - https://github.com/containerd/containerd
  - **runc** [launcher] – OCI container launcher (used by Docker v1.11+)
    - https://github.com/opencontainers/runc
  - **libcontainer** [system interface] – OCI kernel container interface (used by Docker v0.9+)
    - https://github.com/opencontainers/runc/tree/master/libcontainer
    - The Parallels (now Odin) libct interface was merged with libcontainer in mid 2014
    - RedHat's project atomic has containerized user mode Linux with libcontainer
    - Now the standard container library/format of the OCI
  - Namespaces [Isolation] – Linux kernel process namespaces
    - Filesystem isolation: each container has its own root filesystem
      - Copy-on-write (COW) minimizing disk usage
    - Process isolation: each container runs in its own process environment
    - Network isolation: separate virtual interfaces and IP addressing between containers
    - IPC isolation: interprocess communications namespace (shared mem/mqueues, etc.)
    - User isolation: each container has its own set of users (full kernel support in 3.11 and Docker v1.10+)
    - UTS isolation: each container has its own hostname
    - Cgroup isolation: each container has its own Cgroup root (new in kernel 4.6, not used by Docker)
      - http://lkml.iu.edu/hypermail/linux/kernel/1603.2/02432.html (Cgroup PR)
      - http://lkml.iu.edu/hypermail/linux/kernel/1603.2/02433.html (Namespaces PR)
  - CGroups [Constraints] - Linux kernel process control groups
    - Resource constraints: resources like CPU, memory and I/O are constrained, container by container, using kernel control groups

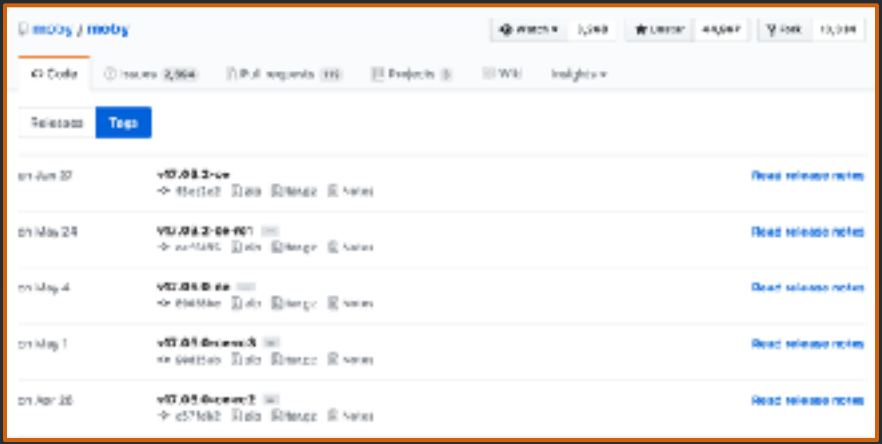


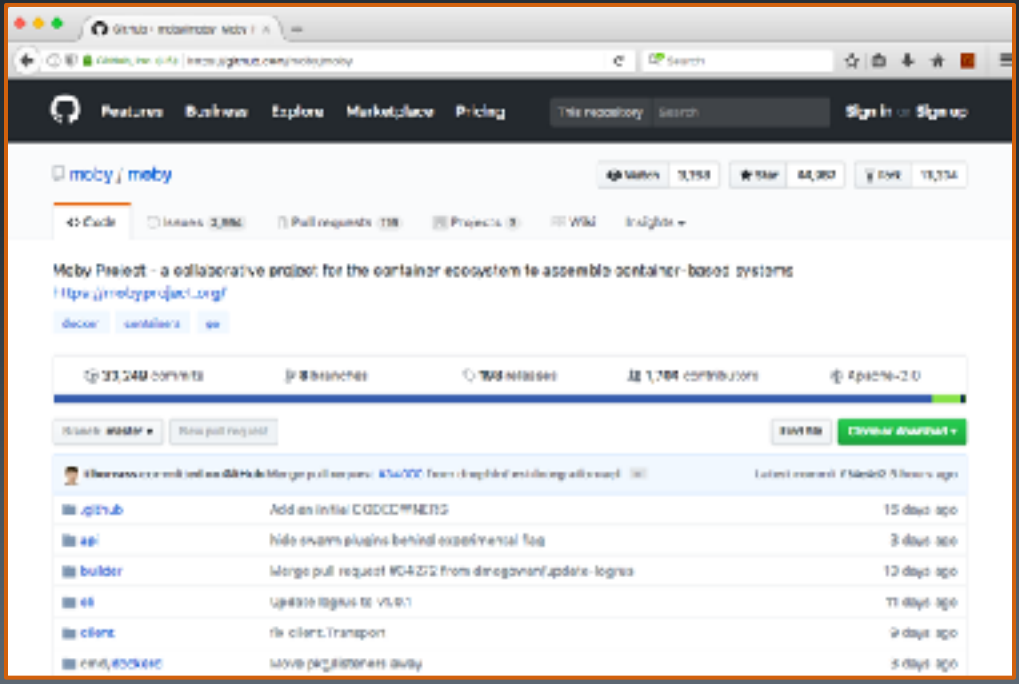

```
user@ubuntu:~$ ps -f $(pgrep docker)
UID          PID    PPID  C STIME TTY      STAT   TIME CMD
root        1388       1  0 Feb28 ?        Ssl    5:06 /usr/bin/docker
root        1497    1388  0 Feb28 ?        Ssl    2:46 unix:///var/run/docker/libcontainerd/docker-containerd.sock --metrics-interval=0 --start-timeout 2m --
state-dir /var/run/docker/libcontainerd/containerd --shim docker-containerd-shim --runtime docker-runc
```

```
user@ubuntu:~$ ls -l $(which docker-runc)
-rwxr-xr-x 1 root root 8199616 Apr  7 22:49 /usr/bin/docker-
runc
```

Container Tools Source

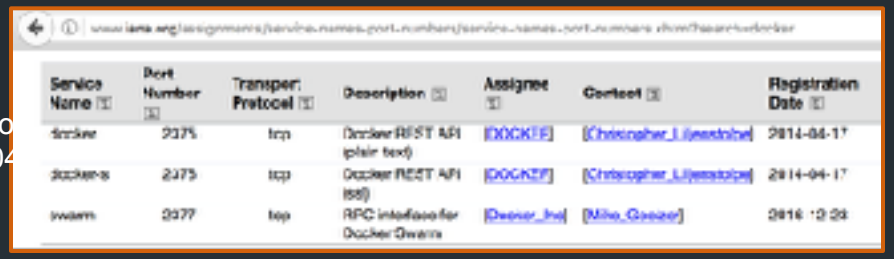
- Docker, Kubernetes, etcd, and most other container tools are written in Go
  - Go (aka. golang) is a programming language developed at Google in 2007 by Robert Griesemer, Rob Pike, and Ken Thompson
    - Statically-typed
    - Loosely derived from C
    - Garbage collection
    - Type safety
    - Some dynamic-typing capabilities
    - Built-in types including variable-length arrays and key-value maps
    - A large standard library
  - Now used in some of Google's production systems
  - Go's primary "gc" compiler targets Linux, Mac OS X, FreeBSD, NetBSD, OpenBSD, Plan 9, and Windows
    - i386, amd64, ARM and IBM POWER processors
  - Go's secondary "gccgo" compiler is a GCC frontend
  - Go has become the language of containers; container systems written in Go include:
    - Docker, Rocket, Kubernetes, etcd, Deis (based on Kubernetes), Flynn, Consul kv, runC and libcontainer
- The Docker source code can be found on GitHub
  - https://github.com/moby/moby
  - https://github.com/docker/docker-ce





Docker Daemon

- The Docker daemon runs automatically when installed from packages
  - `/usr/bin/dockerd`
- The daemon listens on a Unix socket
  - `/var/run/docker.sock`
- The "docker" group owns the socket, users in that group can communicate with the daemon without sudo
  - The `-H` switch instructs the Docker daemon to bind to a port
    - You can pass multiple `-H` switches
      - `sudo dockerd -H tcp://0.0.0.0:2375`
        - Listen on port 2375 on all interfaces
      - `sudo dockerd -H tcp://0.0.0.0:2375 \ -H unix:///home/docker/docker.sock`
        - Listen on port 2375 on all interfaces and a unix socket
- Settings are persisted in the server configuration file
  - `/etc/default/docker` #Ubuntu 14.04
  - `/usr/lib/systemd/system/docker.service` #RHEL/Cento
  - `/lib/systemd/system/docker.service` #Ubuntu 16.04
  - `/etc/sysconfig/docker` #others
- The Docker daemon logs to /var/log
  - `/var/log/upstart/docker.log` #Ubuntu
  - `/var/log/daemon.log` #Centos/Debian (grep docker)
  - `/var/log/messages` #RHEL (grep docker)
- Docker uses the `/var/lib/docker` directory as its default operating directory



Service Name	Port Number	Transport Protocol	Description	Assignee	Contact	Registration Date
docker	2375	tcp	Docker REST API (plain text)	[DOCKER]	[Christopher Li (petstolpe)]	2014-04-17
docker-s	2376	tcp	Docker REST API (ssl)	[DOCKER]	[Christopher Li (petstolpe)]	2014-04-17
swarm	2377	tcp	RPC interface for Docker Swarm	[Dweomer_Inc]	[Mike Gaston]	2016-12-28

```
docker@ubuntu:~$ sudo ls -l /var/lib/docker
total 44
drwxr-xr-x  5 root root 4096 Mar  1 14:84 aufs
drwx------  3 root root 4096 Mar  1 17:52 containers
drwx------  3 root root 4096 Mar  1 14:84 execdriver
drwx------ 11 root root 4096 Mar  1 16:49 graph
drwx------  2 root root 4096 Mar  1 14:84 init
-rw-r--r--  1 root root 5128 Mar  1 17:52 linkgraph.db
-rw-------  1 root root  192 Mar  1 16:49 repositories-aufs
drwx------  2 root root 4096 Mar  1 16:49 tmp
drwx------  2 root root 4096 Mar  1 16:11 trust
drwx------  2 root root 4096 Mar  1 14:84 volumes
docker@ubuntu:~$
```

```
docker@ubuntu:~$ ls -lL /var/run
total 52
-rw-r--r-- 1 root       root          5 Mar  1 11:57 acpid.pid
srw-rw-rw- 1 root       root          0 Mar  1 11:57 acpid.socket
drwxr-xr-x 2 root       root         40 Mar  1 11:57 alsa
drwxr-xr-x 2 avahi      avahi        80 Mar  1 11:57 avahi-daemon
-rw-r--r-- 1 root       root          5 Mar  1 11:57 crond.pid
---------- 1 root       root          0 Mar  1 11:57 crond.reboot
drwxr-xr-x 3 root       lp          120 Mar  1 11:57 cups
drwxr-xr-x 2 messagebus messagebus   80 Mar  1 11:57 dbus
-rw-r--r-- 1 root       root          4 Mar  1 14:04 docker.pid
srw-rw---- 1 root       docker        0 Mar  1 14:04 docker.sock
drwxr-xr-x 2 root       root         40 Mar  1 11:57 initrams
-rw-r--r-- 1 root       root          5 Mar  1 11:57 kerneloops.pid
drwx--x--x 3 root       root         60 Mar  1 11:57 lightdm
-rw-r--r-- 1 root       root          5 Mar  1 11:57 lightdm.pid
drwxrwxrwt 3 root       root         60 Mar  1 15:17 lock
-rw-r--r-- 1 root       root        113 Mar  1 11:57 motd.dynamic
drwxr-xr-x 2 root       root         60 Mar  1 11:57 mount
drwxr-xr-x 3 root       root        140 Mar  1 11:57 network
```

```
docker@ubuntu:~$ cat /etc/default/docker
# Docker Upstart and SysVinit configuration file

# Customize location of Docker binary (especially for development testing).
#DOCKER="/usr/local/bin/docker"

# Use DOCKER_OPTS to modify the daemon startup options.
#DOCKER_OPTS="--dns 8.8.8.8 --dns 8.8.4.4"

# If you need Docker to use an HTTP proxy, it can also be specified here.
#export http_proxy="http://127.0.0.1:3128/"

# This is also a handy place to tweak where Docker's temporary files go.
#export TMPDIR="/mnt/bigdrive/docker-tmp"
docker@ubuntu:~$
```

Docker Client

- Client requests specify non default or remote ports with the `-H` switch
  - The `DOCKER_HOST` environment variable can be set as an alternative

```
export DOCKER_HOST="tcp://
10.0.13.20:2375"
```

- Proxy s[illegible]s:
  - `HTTPS_PROXY`
  - `HTTP_PROXY`
  - `NO_PROXY`
- Command line interface reference and help:
  - http://docs.docker.com/reference/commandline/cli/
  - Shell help:
    - `$ docker --help`
    - `$ docker help start`
    - `$ docker help run  # etc.`

```
user@ubuntu:~$ sudo docker -H "unix:///var/run/docker.sock" info
Containers: 4
 Running: 0
 Paused: 0
 Stopped: 4
Images: 4
Server Version: 17.03.0-ce
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
 Backing Filesystem: extfs
 Dirs: 20
 Dirperm1 Supported: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 977c511eda0925a723debdc94d09459af49d082a
runc version: a01dafd4Bbc1c7cc12bdb0120Gf9fea7dd6feb70
```

CLI Config File

- You can modify the `docker` command behavior using:
  - Command-line options, which take priority over:
  - Environment variables, which take priority over
  - config.json options
- Docker CLI options are stored in `~/.docker/config.json`
  - You can specify an alternate location via:
    - `DOCKER_CONFIG` environment variable
    - `--config` command line option: `docker --config ~/testconfigs/ container ls`
  - Docker manages the files in the configuration directory, config.json is the only one you should edit
- config.json properties:
  - HttpHeaders specifies a set of headers to include in all messages sent from the Docker client to the daemon
    - Docker does not try to interpret or understand these headers
    - You can not change any headers Docker sets for itself
  - psFormat specifies the default format for `docker container ls` output
    - When the `--format` flag is not provided with the `docker container ls` command, Docker's client uses this property
  - detachKeys property sets the detach key sequence
  - imagesFormat specifies the default format for `docker image ls` output

```
{
  "HttpHeaders": {
    "MyHeader": "MyValue"
  },
  "psFormat": "table {{.ID}}\\t{{.Image}}\\t{{.Command}}\\t{{.Labels}}",
  "imagesFormat": "table {{.ID}}\\t{{.Repository}}\\t{{.Tag}}\\t{{.CreatedAt}
  "detachKeys": "ctrl-e,e"
}
```

```
user@nodea:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND                  LABELS
7a5c503a3954        nginx               "nginx -g 'daemon ..."   loc=seattle,sys=x1275
8cb0fd2e080b        nginx               "nginx -g 'daemon ..."   loc=portland,sys=x1275
0c01a885eff4        nginx               "nginx -g 'daemon ..."
af8a03a26819        ubuntu              "/bin/bash"
```

Inspect

- The `docker container inspect` subcommand displays the meta data associated with a container
- All container configuration and/or descriptive data which is not part of the container's filesystem can be found in the metadata produced by inspect

```
user@ubuntu:~$ docker container inspect 7f5a
[
    {
        "Id":
"7f5a4ebf67ac551f2ce197b2cd493217b193231f707a1a1bfefa803cd37e8f36",
        "Created": "2016-08-21T10:36:01.189415017Z",
        "Path": "/bin/bash",
        "Args": [],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 17226,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2016-08-21T10:36:01.782333381Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:60df6678b2556867…df4b6a2e3d1053ce86ea63b0efd7",
        "ResolvConfPath": "/var/lib/docker/containers/7f5a…cd37e8f36/
resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/7f5a4eb…3cd37e8f36/
hostname",
        "HostsPath": "/var/lib/docker/containers/7f5a…f36/hosts",
        "LogPath": "/var/lib/docker/containers/7f5a…f36/7f5a4…e8f36-json.log",
        "Name": "/silly_carson",
        "RestartCount": 0,
        "Driver": "aufs",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "",
        "ExecIDs": [
            "60fb427aacbd9148244b594c51070043c4896926603d3d455e263561777da80b"
        ], }
```

Top Level Metadata Keys

- Id – container ID
- Created – timestamp of the container's creation (not necessarily the start time)
- Path – path of the executable launched as PID 1
- Args – command line arguments passed to the executable
- State – general run data (status, start and finish times, exit code, host relative PID, etc.)
- Image – image used to launch the container (SHA hash ID)
- ResolvConfPath – /etc/resolv.conf used in container
- HostnamePath – /etc/hostname used in container
- HostsPath – /etc/hosts used in container
- LogPath – file used to capture container stdout/stderr
- Name – container name
- RestartCount – number of times container has been restarted by the restart policy
- Driver – unioning file system driver used (aka. Graph driver)
- MountLabel – the selinux context (label) for the mounts in the container
- ProcessLabel – the selinux label to apply to processes running in the container
- AppArmorProfile – the AppArmour profile to associate with the container
- ExecIDs – IDs of any tasks executed with "docker exec" within the container
- HostConfig – the host specific (non-portable) configuration settings
- GraphDriver – configuration for the unioning driver
- Mounts – array of volume mounts in use
- Config – the portable configuration settings
- NetworkSettings – network configuration

Particularly Useful Metadata


```
//IDENTITY
user@ubuntu:~$ docker container inspect -f '{{.State.Pid}}' websvr
18263
user@ubuntu:~$ docker container inspect -f '{{.Id}}' websvr
63b55d8befb430359f8727726138d530ef198aae50ef8a3d129c5812c4212351
user@ubuntu:~$ docker container inspect -f '{{.Path}}' websvr
nginx
user@ubuntu:~$ docker container inspect -f '{{.Args}}' websvr
[-g daemon off;]
user@ubuntu:~$ docker container inspect -f '{{.Config.Image}}' websvr
nginx
//NETWORK
user@ubuntu:~$ docker container inspect -f '{{.Config.Hostname}}' websvr
63b55d8befb4
user@ubuntu:~$ docker container inspect -f '{{.NetworkSettings.IPAddress}}' websvr
172.17.0.3
user@ubuntu:~$ docker container inspect -f '{{.NetworkSettings.MacAddress}}' websvr
02:42:ac:11:00:03
//VOLUMES
user@ubuntu:~$ docker container inspect -f '{{.Mounts}}' test
[{ /home/user /myuser   true rprivate}]
//STATUS
user@ubuntu:~$ docker container inspect -f '{{.State.Status}}' websvr
running
user@ubuntu:~$ docker container inspect -f '{{.State.ExitCode}}' websvr
0
user@ubuntu:~$ docker container inspect -f '{{.Created}}' websvr
2016-08-21T16:29:52.091890205Z
user@ubuntu:~$ docker container inspect -f '{{.State.StartedAt}}' websvr
2016-08-21T16:29:52.625209698Z
user@ubuntu:~$ docker container inspect -f '{{.State.FinishedAt}}' websvr
0001-01-01T00:00:00Z
```

Environment Variables

- When a container is started Docker sets several environment variables
  - `HOME` Set based on the value of `USER`
  - `HOSTNAME` The hostname associated with the container
  - `PATH` Includes std directories: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
  - `TERM` xterm if the container is allocated a pseudo-TTY
- Containers can be configured with explicit environment variables
  - `-e`, (`--env`) value Sets an environment variable
    - can be used multiple times on the same command line
  - `--env-file` value Read in a file of environment variables
    - File is k=v format with one environment variable per line
- The environment the Docker CLI is operated from has no interaction with containers launched:

```
user@ubuntu:~$ export TESTSUITE=REGRESSION
user@ubuntu:~$ docker container run -e="TESTCASE=A8" -e=LIMIT=600 ubuntu sh -c 'echo $TESTSUITE
$TESTCASE $LIMIT'
A8 600
user@ubuntu:~$ cat vars.env
TESTCASE=A8
LIMIT=600
user@ubuntu:~$ docker container run --env-file=/home/user/vars.env ubuntu sh -c 'echo $TESTCASE
$LIMIT'
A8 600
user@ubuntu:~$
```

Sending Signals and Waiting for Containers

- `docker container kill` limits the need for SSH
  - `$ docker container kill -s <signal> <container>`
- `docker container wait` allows you to synchronize with containers
  - `$ docker container wait <container>`

Sending SIGUSR1 to a daemon will dump all goroutines stacks without exiting

```
user@ubuntu:~$ docker container run -d centos:7 /usr/bin/tail -f /dev/null
39df6e0adcc023322a6f9e3c318ab64ba33ff5c8ed421a1548f6c9d721465074
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
39df6e0adcc0        centos:7            "/usr/bin/tail -f ..."   4 seconds ago       Up 4 seconds                            inspiring_varahamihira
user@ubuntu:~$ docker container stop 39d
39d
user@ubuntu:~$ docker container run -d --name="longproc" centos:7 /usr/bin/tail -f /dev/null
ca1515cb4ecceb16e3d258f79b90e7123d8eae427392b561d339ee04ea17eb18
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
ca1515cb4ece        centos:7            "/usr/bin/tail -f ..."   5 seconds ago       Up 5 seconds                            longproc
user@ubuntu:~$ docker container exec -it longproc /bin/bash
[root@ca1515cb4ece /]# ps
   PID TTY          TIME CMD
     5 ?        00:00:00 bash
    18 ?        00:00:00 ps
[root@ca1515cb4ece /]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 03:54 ?        00:00:00 /usr/bin/tail -f /dev/null
root         5     0  0 03:54 ?        00:00:00 /bin/bash
root        19     5  0 03:54 ?        00:00:00 ps -ef
[root@ca1515cb4ece /]# exit
exit
user@ubuntu:~$ docker container kill --signal="HUP" longproc
longproc
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
ca1515cb4ece        centos:7            "/usr/bin/tail -f ..."   About a minute ago  Up About a minute                       longproc
user@ubuntu:~$ docker container kill --signal="TERM" longproc
longproc
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
ca1515cb4ece        centos:7            "/usr/bin/tail -f ..."   About a minute ago  Up About a minute                       longproc
user@ubuntu:~$ docker container kill --signal="KILL" longproc
longproc
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
user@ubuntu:~$
```

```
user@ubuntu:~$ docker container wait longproc ; echo "Long Running Process Complete"
137
Long Running Process Complete
user@ubuntu:~$
```

Signals

- Processes can ignore, block, or catch all signals except SIGSTOP and SIGKILL
- For a process to catch a signal it must include code that will take action when the signal is received
- If the signal is not caught by the process, the kernel will take default "action" for the signal
- A process running as PID 1 inside a container is treated specially by Linux
  - Default actions for signals other than SIGKILL and SIGSTOP are ignored
  - e.g. PID 1 will not terminate on SIGINT or SIGTERM unless it is coded to do so
- Signal numbers are architecture dependent
  - Col1: alpha and sparc
  - Col2: x86, arm, and most other architectures
  - Col3: mips

Signal	Value	Action	Comment POSIX.1-1990
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal instruction
SIGABRT	6	Core	Abort signal from abort(3)
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers
SIGALRM	14	Term	Timer signal from alarm(2)
SIGTERM	15	Term	Termination signal
SIGUSR1	30,10,16	Term	User defined signal 1
SIGUSR2	31,12,17	Term	User-defined signal 2
SIGCHLD	20,17,18	Ign	Child stopped or terminated
SIGCONT	19,18,25	Cont	Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process
SIGTSTP	18,20,24	Stop	Stop typed at terminal
SIGTTIN	21,21,26	Stop	Terminal input for background process
SIGTTOU	22,22,27	Stop	Terminal output for background process

http://man7.org/linux/man-pages/man7/signal.7.html

Signal	Value	Action	Comment SUSv2 and POSIX.1-2001
SIGBUS	10,7,10	Core	Bus error (bad memory access)
SIGPOLL		Term	Pollable event (Sys V). Synonym for SIGIO
SIGPROF	27,27,29	Term	Profiling timer expired
SIGSYS	12,31,12	Core	Bad argument to routine (SVr4)
SIGTRAP	5	Core	Trace/breakpoint trap
SIGURG	16,23,21	Ign	Urgent condition on socket (4.2BSD)
SIGVTALRM	26,26,28	Term	Virtual alarm clock (4.2BSD)
SIGXCPU	24,24,30	Core	CPU time limit exceeded (4.2BSD)
SIGXFSZ	25,25,31	Core	File size limit exceeded (4.2BSD)

Restart Policies

- `docker container run --restart <repo>[:<tag>]` (added in docker 1.2)
  - Causes Docker to restart the container if it stops
    - 100ms delay, doubling each restart to prevent flooding the server
  - Respected when the Docker daemon starts (e.g. after a system crash/reboot)
  - Three policies can be set:
    - `no` the default, never restart
    - `always` restarts the container no matter what
    - `on-failure[:max-retry]`
      - Restart the container every time it exits with a non 0 exit code If max-retry is set the engine will try up to max-retry times to restart the container before giving up

```
user@ubuntu:~$ docker container run -d --restart=always ubuntu sleep 20
61a4420ffa718e13ea2fd528e43b8268fa86c77d6c28bb26bee932b3e7143b04
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
61a4420ffa71        ubuntu              "sleep 20"          4 seconds ago       Up 4 seconds                            friendly_golick
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
61a4420ffa71        ubuntu              "sleep 20"          11 seconds ago      Up 11 seconds                           friendly_golick
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
61a4420ffa71        ubuntu              "sleep 20"          22 seconds ago      Up 1 second                             friendly_golick
```

Docker Engine Events

- The `docker events` subcommand allows you to monitor Docker engine events
  - Greatly expanded in Docker Engine v1.10 (new events in orange) (Docker v1.12 additions in green)
- Container events reported:
  - attach, commit, copy, create, destroy (rm), detach, die (container processes exited), exec_create, exec_start, export, health_status, kill, oom (container out of memory), pause, rename, resize, restart, start, stop, top, unpause, update
- Image events reported:
  - delete, import, load, pull, push, tag, untag
- Volume events reported:
  - create, mount, unmount, destroy
- Network events reported:
  - create, connect, disconnect, destroy

```
user@ubuntu:~$ docker container run -it cirros /bin/sh
/ # exit
user@ubuntu:~$
```

```
user@ubuntu:~$ docker events
2017-03-21T21:08:28.210925030-07:00 container create 0c478f036b3c9c6fb36a4a2a43a4a654d929f30aee81372fc378bde4b3b29e7d (image=cirros, name=distracted_franklin)
2017-03-21T21:08:28.211757033-07:00 container attach 0c478f036b3c9c6fb36a4a2a43a4a654d929f30aee81372fc378bde4b3b29e7d (image=cirros, name=distracted_franklin)
2017-03-21T21:08:28.227509338-07:00 network connect 9036ad93c6fa19d8a2832ebea3e03cdcb5967bd5bedc59e4ead0ee4a78f3754f (container=0c478f036b3c9c6fb36a4a2a43a4a554
d929f30aee81372fc378bde4b3b29e7d, name=bridge, type=bridge)
2017-03-21T21:08:28.430192357-07:00 container start 0c478f036b3c9c6fb36a4a2a43a4a654d929f30aee81372fc378bde4b3b29e7d (image=cirros, name=distracted_franklin)
2017-03-21T21:08:28.431310651-07:00 container resize 0c478f036b3c9c6fb36a4a2a43a4a654d929f30aee81372fc378bde4b3b29e7d (height=26, image=cirros, name=distracted_
franklin, width=119)
2017-03-21T21:08:32.348356755-07:00 container die 0c478f036b3c9c6fb36a4a2a43a4a654d929f30aee81372fc378bde4b3b29e7d (exitCode=0, image=cirros, name=distracted_fr
anklin)
2017-03-21T21:08:32.434934103-07:00 network disconnect 9036ad93c6fa19d8a2832ebea3e03cdcb5967bd5bedc59e4ead0ee4a78f3754f (container=0c478f036b3c9c6fb36a4a2a43a4a
654d929f30aee81372fc378bde4b3b29e7d, name=bridge, type=bridge)
2017-03-21T21:08:34.064378105-07:00 container die 61a4420ffa718e13ea2fd528e43b8268fa86c77d6c28bb26bee932b3e7143004 (exitCode=0, image=ubuntu, name=friendly_goli
ck)
2017-03-21T21:08:34.150178809-07:00 network disconnect 9036ad93c6fa19d8a2832ebea3e03cdcb5967bd5bedc59e4ead0ee4a78f3754f (container=61a4420ffa718e13ea2fd528e43b8
268fa86c77d6c28bb26bee932b3e7143004, name=bridge, type=bridge)
2017-03-21T21:08:34.204694610-07:00 network connect 9036ad93c6fa19d8a2832ebea3e03cdcb5967bd5bedc59e4ead0ee4a78f3754f (container=61a4420ffa718e13ea2fd528e43b8268
fa86c77d6c28bb26bee932b3e7143004, name=bridge, type=bridge)
2017-03-21T21:08:34.407191116-07:00 container start 61a4420ffa718e13ea2fd528e43b8268fa86c77d6c28bb26bee932b3e7143004 (image=ubuntu, name=friendly_golick)
^C
user@ubuntu:~$
```

Summary

- Linux container technology has been evolving for over a decade
- Docker is a recent entry into the Linux container space but has become the dominant force
- The OCI standardizes the format of containers making it possible to run an OCI compliant container in many environments (with or without Docker)
- The Docker daemon and the Docker client can be configured via configuration files and command line options
- Docker containers are described by metadata displayed by *docker container inspect*
- Containers can be passed environment variables on the command line or via a file
- Docker containers process signals and the *docker container kill* command can be used to send signals to a container's PID 1
- Docker produces events corresponding to most container state changes
- Users can establish container restart policies to ensure failed containers are restarted

Lab 4

- Containers, under the hood