RX-M
Software Consulting

Docker

An in depth introduction to Docker and its ecosystem

Overview

Day One

1. Container Overview
2. Working with Containers
3. Docker Images

Day Two

4. Docker Hub and Registries
5. Dockerfiles
6. Docker in Practice

Administrative Info

- Course: Docker
- Length: 2 Days
- Format: Lecture/Labs/Discussion
- Schedule: 9:30AM – 5:30PM
  15 minute break, AM & PM
  1 hour lunch at noon
  Lab work after each module
- Location: Fire exits, Restrooms, Security, other matters
- Attendees: Name/Role/Experience/Goals for the Course

Lecture and Lab

- Our Goals in this class are two fold:
  1. Familiarize you with the general concepts and ecosystem surrounding Docker
     - This is the primary purpose of the lecture/discussion sessions
  2. Give you practical experience creating and using Docker containers
     - This is the primary purpose of the labs

Day 1

1. Container Overview
2. Working with Containers
3. Creating Docker Images

1: Container Overview

Objectives

- Understand the basic nature of containers
- Examine the differences and similarities between Virtual Machines (VMs) and Containers
- Describe the container value proposition
- Explore the container ecosystem
- Explain the purpose and positioning of Docker in the container space

What is Docker?

- Docker is an open-source project that automates the deployment of applications inside software containers
  - Apache 2.0 license
  - Originally released in March of 2013
- Containers provide “operating-system virtualization” on Linux
- 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 independent containers to run within a single Linux instance
  - Containers utilize: OS Virtualization
- 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
- Encapsulated and deployable
- Runnable
- A new way to package applications for reliable deployment
- Made widely popular by Docker
  - Docker Engine simplifies configuring, constructing and running containers
  - Docker Inc. provides a container platform including systems for publishing and sharing containers
- Container technology predates and enables Docker
  - Docker was the dominant mover in this space during 2013-2014 but competition is appearing

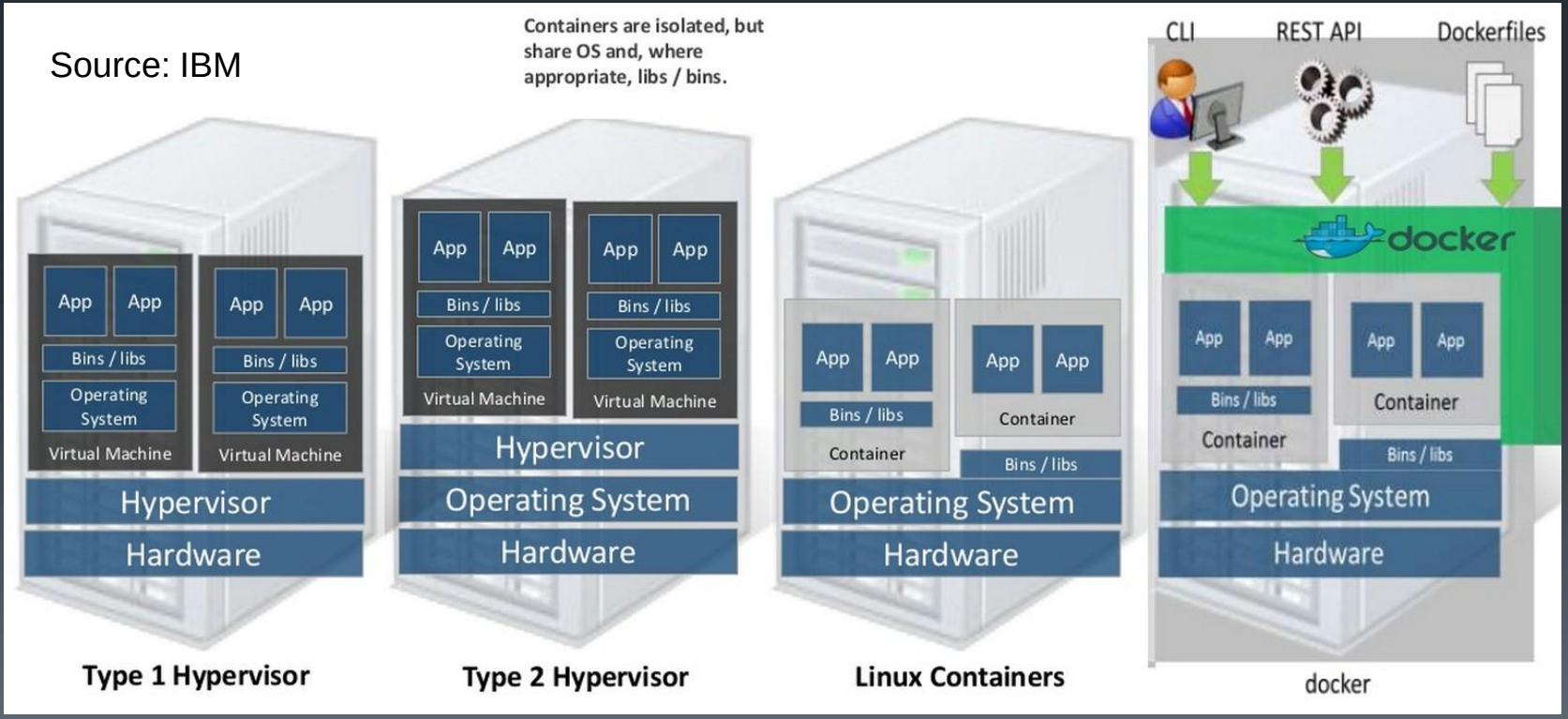
```
$ time docker run ubuntu echo "hello world"
hello world

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

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

VMs and Containers

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

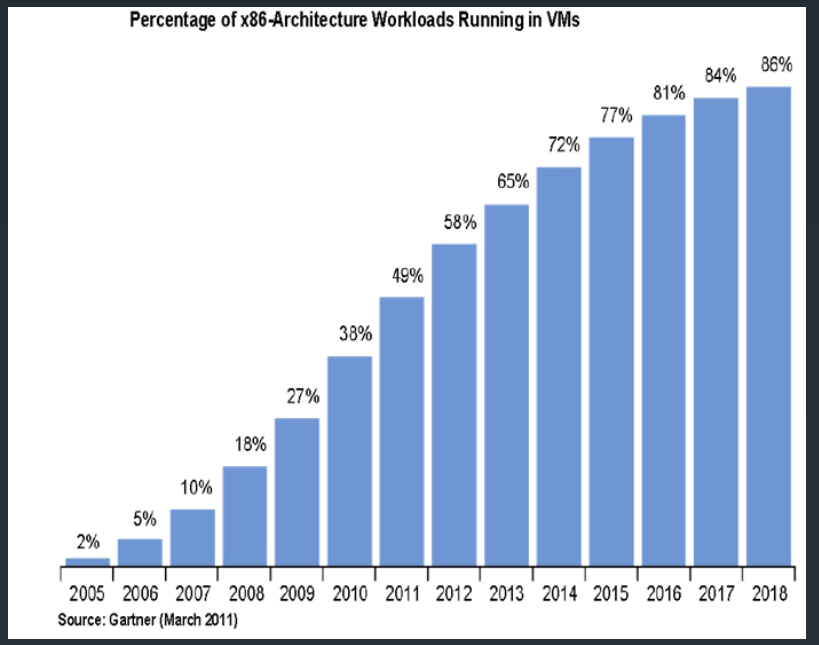
Containers supply only the executables and library interfaces necessary to mimic a Linux distribution (Ubuntu, RHEL, SUSE, etc.), leveraging the fact that all Linux systems use the same underlying kernel.

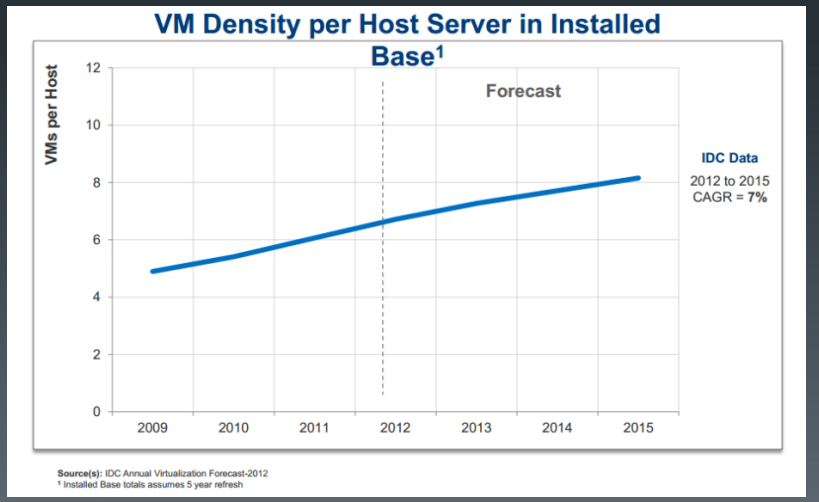


VMs, why do I care?

- VM's allow new machine instances to be deployed in real time (seconds-minutes)
  - Not months, as is typical with physical server purchase/deploy cycles
- VM's enable migration and elasticity
  - Machines can be created, moved and deleted rapidly
- VM's allow physical resources to be fully utilized
  - Physical CPU/RAM use can be maximized without mixing logical machine roles
- 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.
- VM's enable cloud computing
  - Pay as you go
  - Self service

IaaS

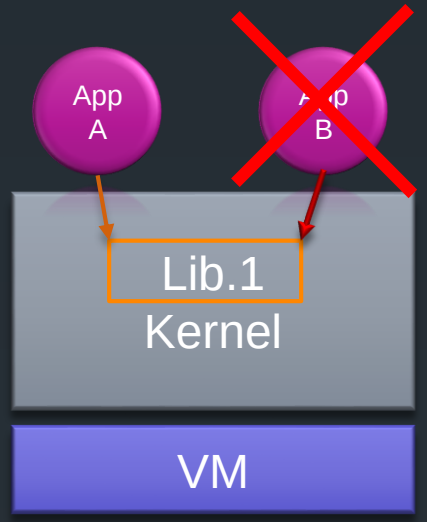


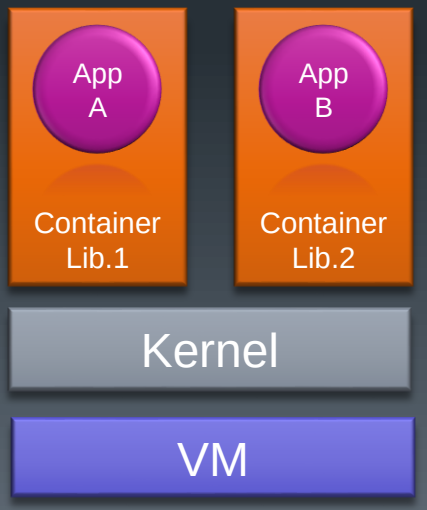


Containers, why do I care?

- 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 has its own unique dependencies directly supported
  - Portability
    - Repeatable deployment across dev/test/prod environments and clouds





PaaS

The killer feature of Docker:

Reliable deployment

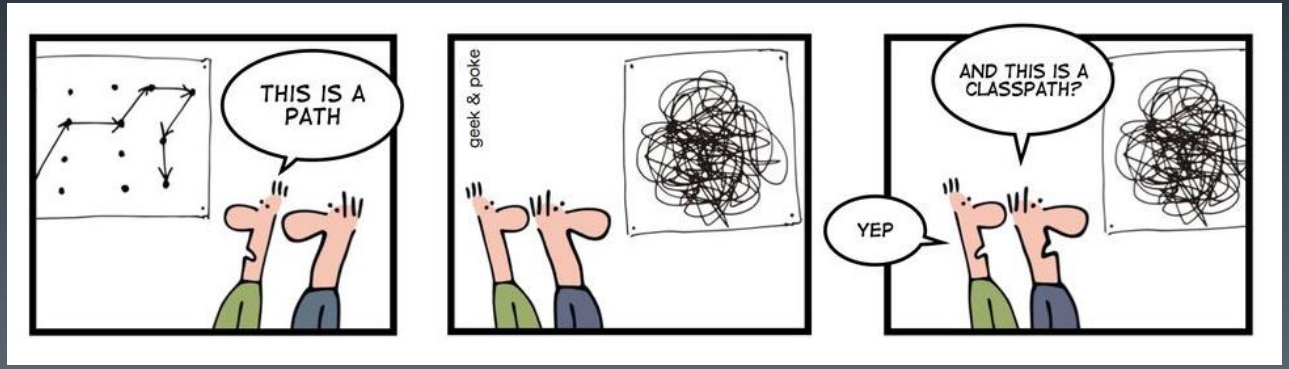
Containers include their own dependencies, isolated from the underlying operating system and other containers

Encapsulation

- Containers can encapsulate:
  - Data
  - Code
  - Frameworks
  - Libraries
  - System Dependencies
  - Packaging
  - Linux Distributions
- 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

Empowering Agile Processes

- SOA and Microservices
  - An approach to software development where many small services are composed into applications
    - Services communicate over well defined interfaces
    - Encourages small teams, each owning a service
  - The more atomic the service:
    1. The more likely it is to be reusable
    2. The more easily it can be encapsulated
    3. The more of them you need to do something useful (!)
- Micro services and VMs have different cardinalities
  - 10:1 Ten services running on a single VM creates an ops integration challenge across all services
- Micro services and Containers have the same cardinality
  - 1:1 One service in one container, requires only ops support for the service encapsulated
- Containers allow each service to be packaged with its own dependencies
  - 10 containerized services map directly to 10 individual, isolated, reliable ops events
  - Gives devops teams autonomy
    - Team owns software and configuration when using containers
  - Empowers CI/CD and incremental application evolution
    - Integration challenges are reduced to inter service communications



The second killer feature of Docker:

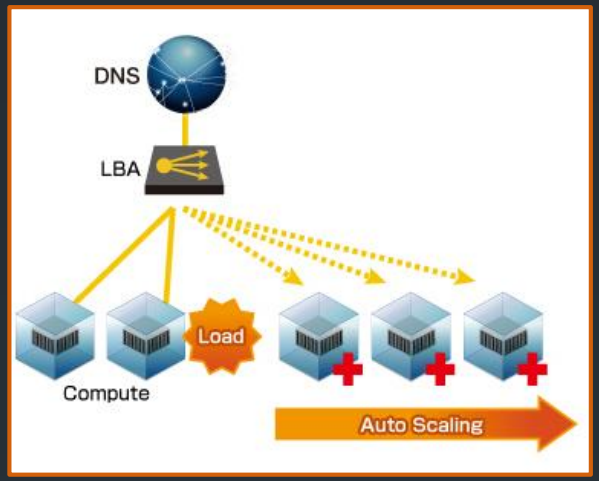
Efficiency

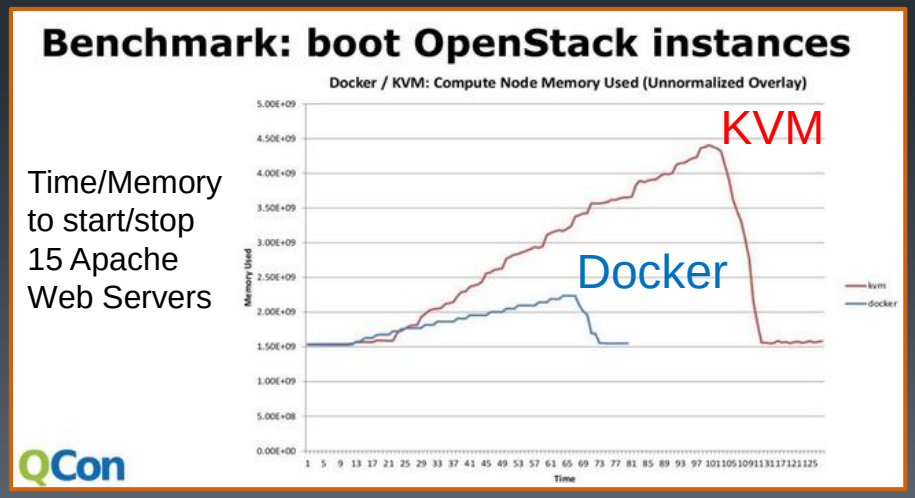
- Time to Launch
- Server Density

Containers can be launched and shutdown at the speed of a process and can directly use/share system resources; Applications Containers are typically 1/10th to 1/100th the size of the equivalent application packaged within a VM leading to increased server density.

Scaling Events & Restart

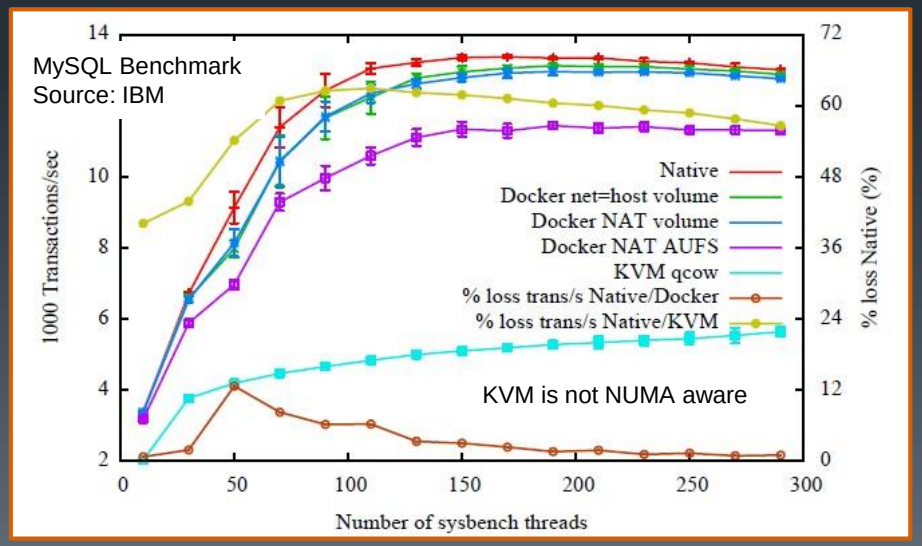
- Containers can be started and stopped in milliseconds
  - Timing like running a program, because it is running a program
    - Run container (dependencies built in)
- VMs are started and stopped in seconds/minutes
  - Timing like booting a computer because it is booting a computer
    - Bootloader
    - OS start
    - Service start
    - CM run (puppet/chef/ansible/salt etc.)
      - Downloading packages
    - Application start
- Container based applications can be scaled in/out quickly
- Failed services within a container can be restarted quickly





Inter Service Communications

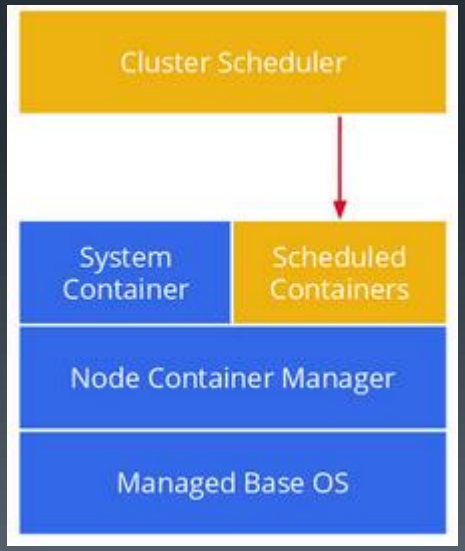
- Inter VM
  - Strong isolation enforced by hypervisor and hardware
  - Forces inter VM process communications to use networking interfaces (Ethernet/IP/[UDP|TCP])
    - Some hypervisors have fast paths but these are often platform specific and some require hardware support
  - The Strength of VMs: Very 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



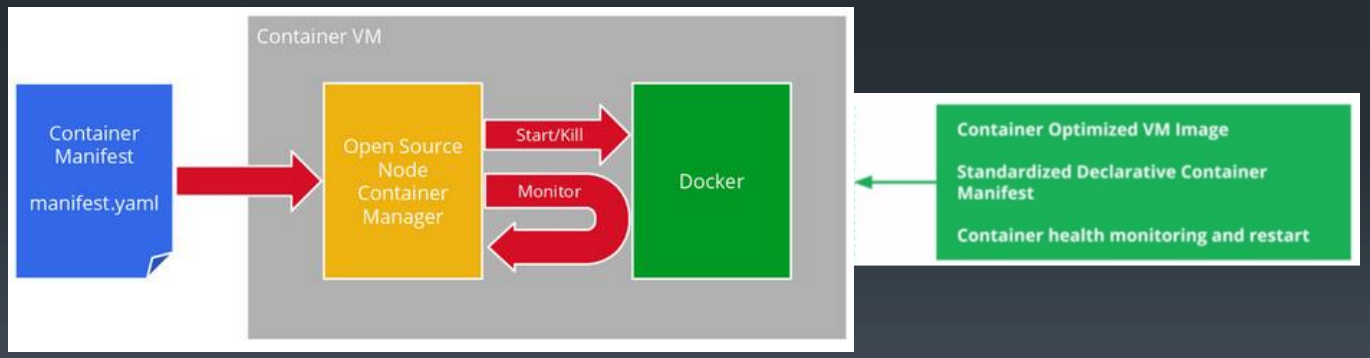
Containers at scale

- Containers - a key enabler of PaaS cloud environments
  - Google App Engine is one of the most visible container based cloud systems
- Everything at Google, from Search to Gmail, is packaged and run in a Linux container
  - Google's Borg system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different containerized applications, across a number of clusters each with up to tens of thousands of machines (Borg is the basis for Kubernetes)
  - Over 2 billion containers are started per week at Google (over 3,000 per second on average)
- lmctfy ("Let Me Contain That For You") open source version of Google's container stack
  - Now collaborating with Docker and porting the core lmctfy concepts and abstractions to libcontainer,
  - libcontainer is the go forward OCI solution

Google Container Infrastructure



Google Cloud Platform
Docker containers in Google Compute Engine



Containers At Scale by Joe Beda
Published May 22, 2014 in Technology

Borg Paper: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43438.pdf

Container & Docker Timeline

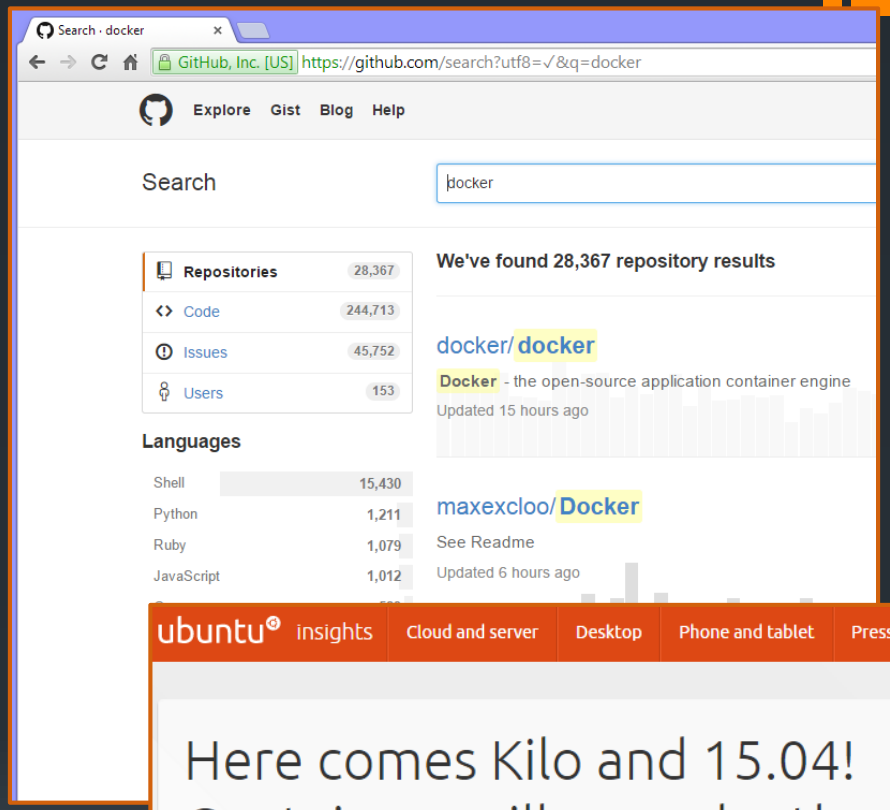
- 2005 OpenVZ (Previously Parallels [SWsoft] Virtuozzo containers) open sourced [Linux 2.6.11]
- 2006 Google (Menage/Seth) add Process Containers to Linux (now called CGroups) [Linux 2.6.15]
- 2007 Google uses cgroups to containerize search [Linux 2.6.20]
- 2008 LXC version 0.1.0 released (basic functions) [Linux 2.6.24]
- 2009 First LTS Linux release with official LXC support [Linux 2.6.32 LTS]
- 2011 Container Unification agreement during kernel summit (In tree unified CGroups and Namespaces resulted)
- 2012 First 3.x LTS kernels [Linux 3.2 & 3.4 LTS]
- 2013 First Linux kernel with full container support for all container systems (LXC, Docker, OpenVZ, etc.) [Linux 3.10 & 3.12 LTS]
  - Canonical provides extended support for Linux 3.13 until April 2016
  - dotCloud pivots, becoming Docker Inc.
  - OpenStack Havana supplies native Docker support (not part of Icehouse or Juno)
  - 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.
  - First DockerCon
- 2014-07 – Docker 1.1 released (Docker Engine + Docker Hub + APIs + expanded ecosystem)
- 2014 Distributions began enabling user namespaces [Linux 3.14 & 3.18 LTS]
- 2014-08 – Docker 1.2 released with many 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 Inc. raises $95mm series D round lead by Insight Venture Partners with new investors Coatue, Goldman Sachs and Northern Trust and existing investors Benchmark, Greylock Partners, Sequoia Capital, Trinity Ventures and AME Cloud Ventures
- 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 1.8.1 released with support for pluggable volume backends
  - 1.8 had a bug which would corrupt images pushed with multiple tags

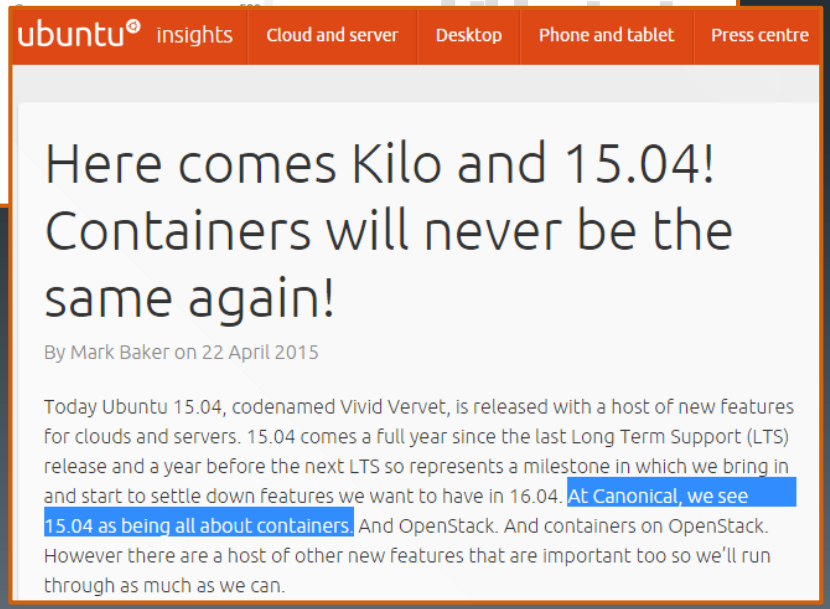
Docker Inc. released a roadmap on 06/2015:

https://github.com/docker/docker/blob/master/ROADMAP.md

Docker EcoSystem

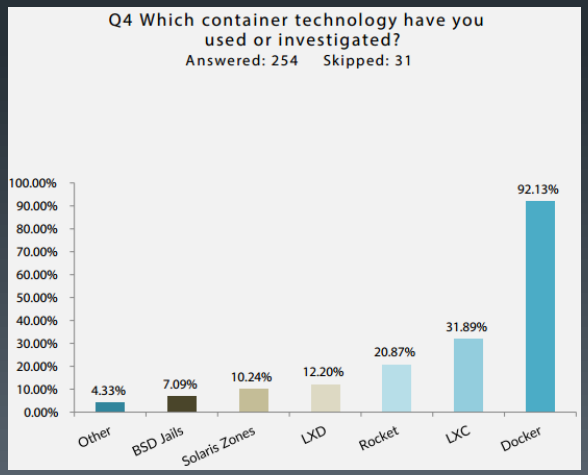
- Github reports over 25,000 Docker projects
- Google Kubernetes (http://kubernetes.io/) is a widely supported open source container orchestration service
  - Supports scheduling and packaging containers (including Docker) onto nodes using Pods
  - Google offers container optimized VMs in their cloud
- Apache Mesos (http://mesos.apache.org/) allows compute clusters to be managed like a single system
  - Mesos kernel runs on every machine providing applications with API's for resource management and scheduling across entire datacenter and cloud environments
  - Marathon enables container deployment on top of Apache Mesos at scale
- Amazon EC2 Container Service allows Docker containers to be deployed on AWS EC2
- November 2014 Microsoft pledged to fully integrate Docker with Windows Server and Azure
  - .Net containers demoed at Build conference 4/2015
- Docker Inc. announced the beta offering of its new suite of orchestration services February 26, 2015
  - Docker Hub Enterprise, on prem repository service
  - Docker Machine configures Docker on cloud instances
  - Docker Swarm allows containers to be deployed in clusters (provides a subset of Kubernetes features)
  - Docker acquired Orchard Laboratories and its Fig container launching service
    - Fig, renamed Docker Compose, enables multiple container application deployment using YAML templates
- OpenStack Havana supports Docker as a Hypervisor
  - Ubuntu adds beta support for lxc containers in Juno/Kilo with their new lxd "lightvisor" (enabling multitenant caliber system containers within which Docker can run)
  - Magnum is an OpenStack incubation project making container orchestration engines (Docker Swarm and Kubernetes) available





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 container inside the logical boundaries of a virtual machine
- CoreOS Inc. Tectonic
  - CoreOS is a minimal Linux OS with direct support for containers (including docker)
    - Similar to RHEL project atomic
  - Rocket (rkt) is a simple and lightweight docker like container platform
  - Jetpack is a FreeBSD implementation of the Rocket App Container spec
  - Tectonic recently announced
    - CoreOS/Rocket/Kubernetes integration
  - Google led a $12mm CoreOS investment round 4/2015
- Joyent SmartOS & Triton
  - Triton uses the docker client but supplies SmartOS (an OpenSolaris/KVM hypervisor) Solaris Zone isolation on the container side for improved security
  - Containers see all SmartOS/Triton machines as a single Docker host, simplifying deployment
  - Built by Joyent the company behind Node.js
- KurmaOS
  - A container based operating system
- LXC
  - Original Linux container library
- BSD Jails
  - Isolation features of Berkeley Software Distribution Unix



https://clusterhq.com/assets/pdfs/state-of-container-usage-june-2015.pdf

Container Foundations and Initiatives

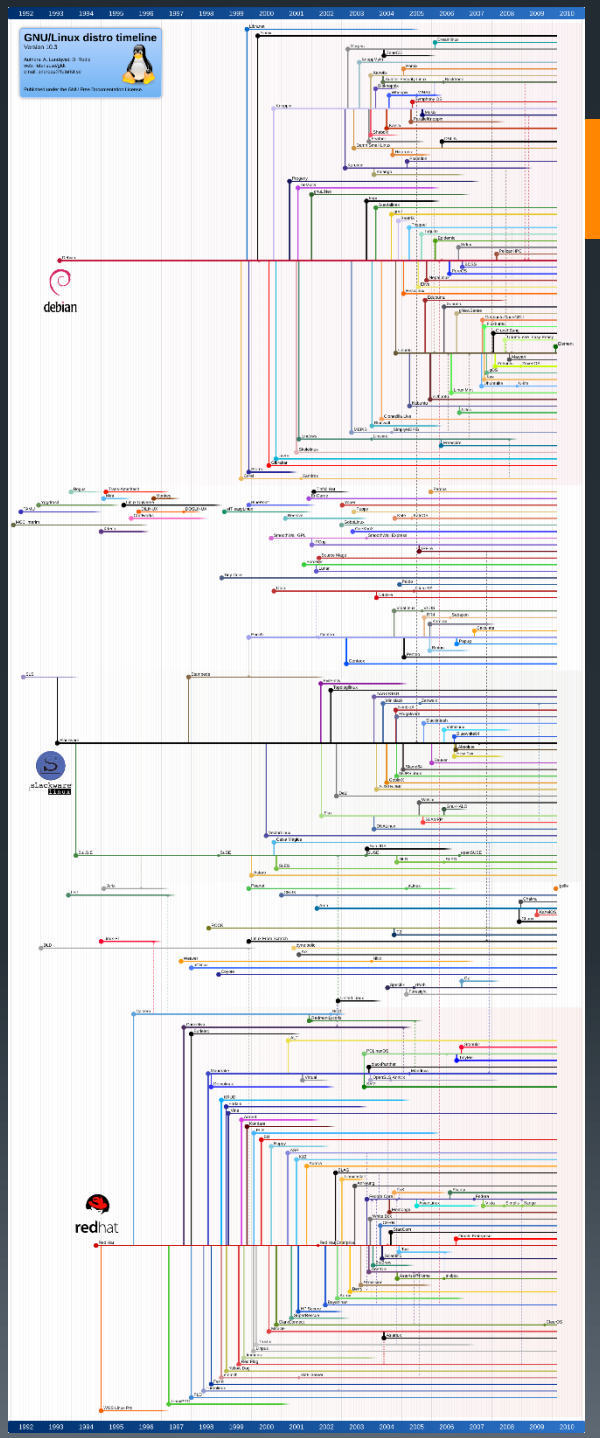
- OCI [circa 6/2015]
  - Open Container Initiative
    - Formerly Open Container project
  - OCI seeks to provide a common container specification
    - A standard that ensures any container can run on any machines or cloud computing service
  - Run by the Linux Foundation
    - Non-profit which oversees the Linux open source operating system
  - Docker donated container format, runtime code and specifications
  - Members:
    - Amazon Web Services, Apcera, Cisco, CoreOS, Docker, EMC, Fujitsu Limited, Goldman Sachs, Google, HP, Huawei, IBM, Intel, Joyent, Linux Foundation, Mesosphere, Microsoft, Pivotal, Rancher Labs, Red Hat and VMware
- CNCF [circa 7/2015]
  - Container Native Computing Foundation
  - Mission: To provide developers with the right set of tools for building, deploying, and managing next-generation, 'cloud-native' applications
  - Aims to govern containers and the stack of technologies around them
    - Google has donated Kubernetes v1.0 to the foundation
  - Run by the Linux Foundation
  - Members include:
    - Cisco, Intel, RedHat, Google, Docker, eBay, Goldman Sachs, CoreOS, IBM, Intel, Joyent, Twitter, VMWare

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 with predefined Docker packages
    - Google Cloud Platform
    - Rackspace Cloud
    - Amazon EC2
    - IBM Softlayer
  - Others (e.g. CoreOS) provide their own direct support
  - Binaries are also supplied for custom installations

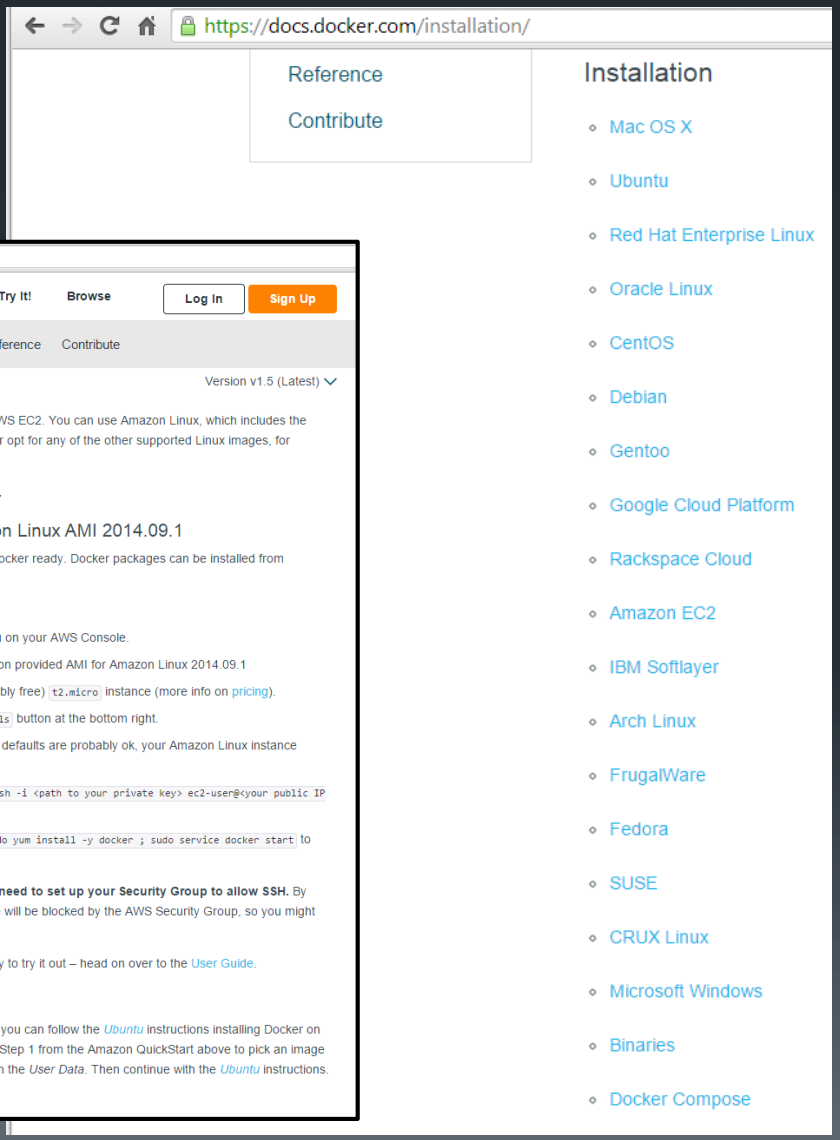
RHEL and Ubuntu are the recommended platforms

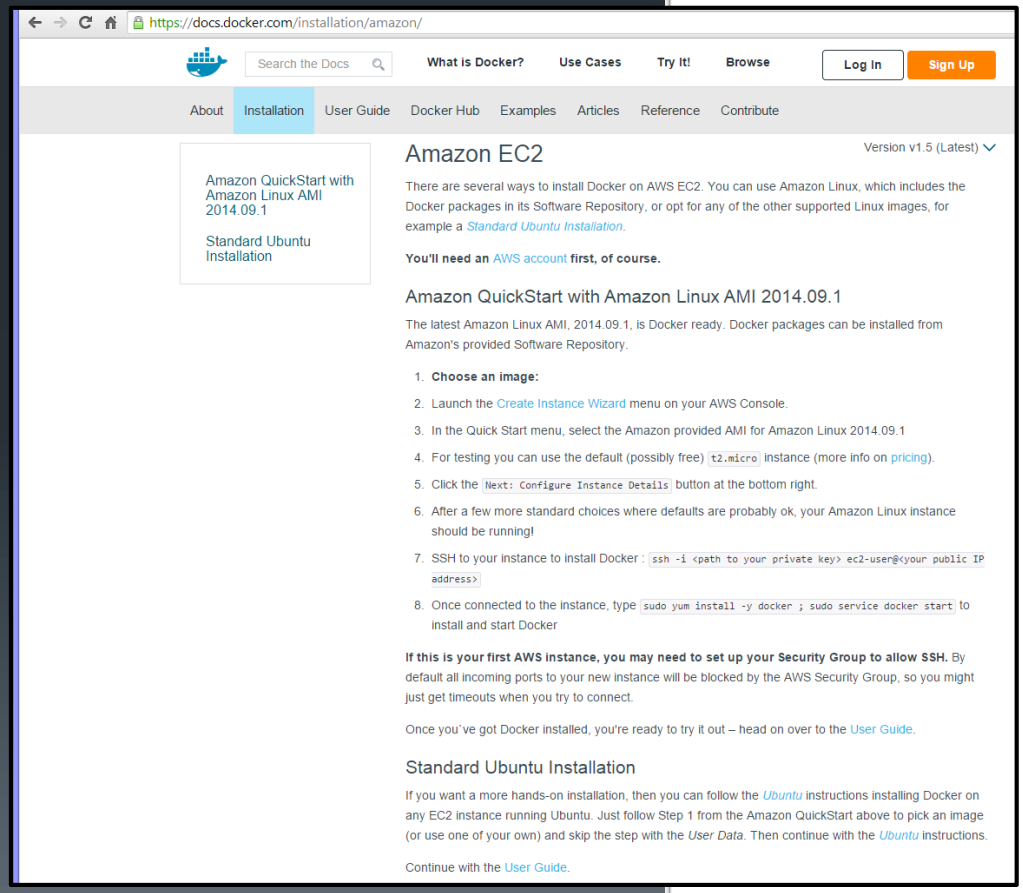
http://futurist.se/gldt



Installation Guides

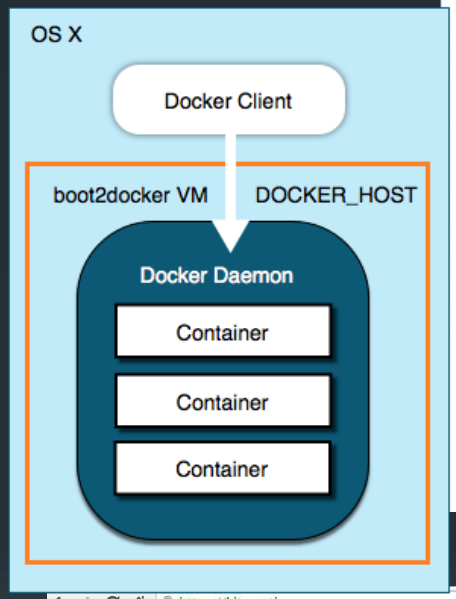
- The latest installation info can be found on the docker web site

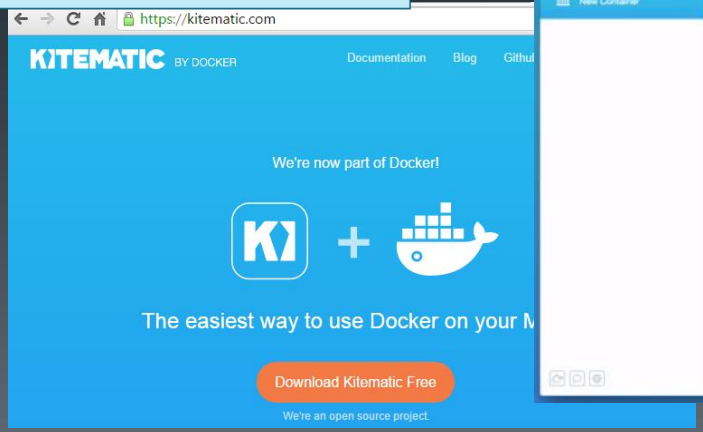


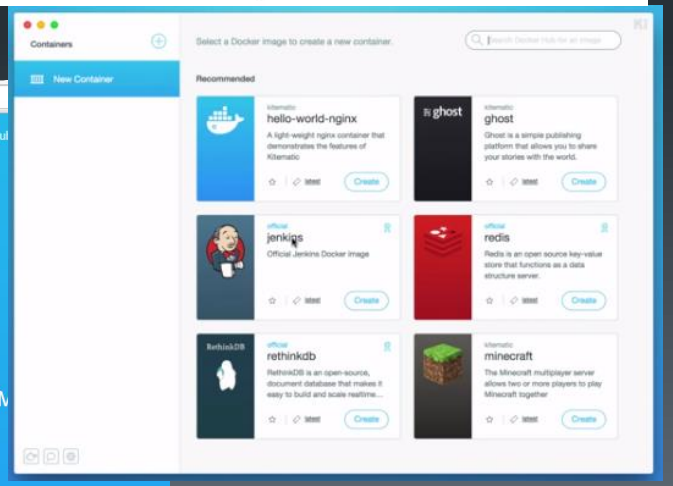


Docker Toolbox

- Docker Toolbox is now the recommended way to get started with Docker
  - Replaces Boot2Docker
- Docker Toolbox has been designed to include the growing set of Docker developer tools like Kitematic, Machine, Swarm, and Compose
  - Boot2Docker installed a command-line tool called boot2docker which was used to manage the Docker VM
  - In Toolbox this has been replaced with Docker Machine
  - Under the hood, Machine still uses the Boot2Docker Linux distribution to run your containers but containers are now managed by Machine instead of the boot2docker







KITEMATIC

- A complete docker solution for Macs with a GUI front end
- Recently acquired by Docker Inc.

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
  - 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
- Currently Docker only runs on x86_64 systems
  - The Docker goal is to run everywhere, including ARM, Joyent SmartOS, and Windows Server
  - The next Microsoft Windows Server will ship with primitives to support container-based process isolation and resource management
- The Docker client can run on virtually any Linux kernel
  - The client also builds on OS X

```
docker@ubuntu:~$ uname -a
Linux ubuntu 3.13.0-46-generic #77-Ubuntu SMP Mon Mar 2 18:23:39 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux
docker@ubuntu:~$ cat /etc/os-release
NAME="Ubuntu"
VERSION="14.04.2 LTS, Trusty Tahr"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 14.04.2 LTS"
VERSION_ID="14.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
docker@ubuntu:~$
```

Summary

- Virtual machines enabled the cloud era IaaS, providing an abstraction layer between operating systems and the underlying hardware
  - Self service compute
  - Pay for use
  - Full hardware utilization
  - Hardware supported isolation
- Containers enable cloud era PaaS, providing an abstraction layer between applications and the kernel
  - Containers are commonly deployed on VM 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
  - Containers can be orchestrated to empower microservice style applications

Lab 1

- Setting up Docker

VMs

- This course includes a preconfigured Ubuntu 14.04 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 14.04 that you have root permissions on

Note that Virtual Box network adapters on the host may interfere with IPv4 DHCP acquisition on a VMWare VM. If you have this problem (VM with IP6 but no IP4 address) remove/disable the virtual box bridge driver.

These virtual machines have a virtual network interface with a generated MAC address. The MAC address must be unique on a given network. NICs are currently set to NAT, which works best in most cases.

If you bridge the NIC you must ensure that your lab VMs do not run into network conflicts with other machines in the lab. You will need to follow the steps below to create a unique MAC address for each Virtual NIC bridged.

- Load the VM into the VMWare library (use “Open a Virtual Machine” from the library window)
- Make sure the machine is not running and open the settings dialog (right or control click the machine and choose settings)
- In the network adapters => advanced dialog choose “generate” to create a new MAC address and choose ok to close all of the dialogs
- Reboot

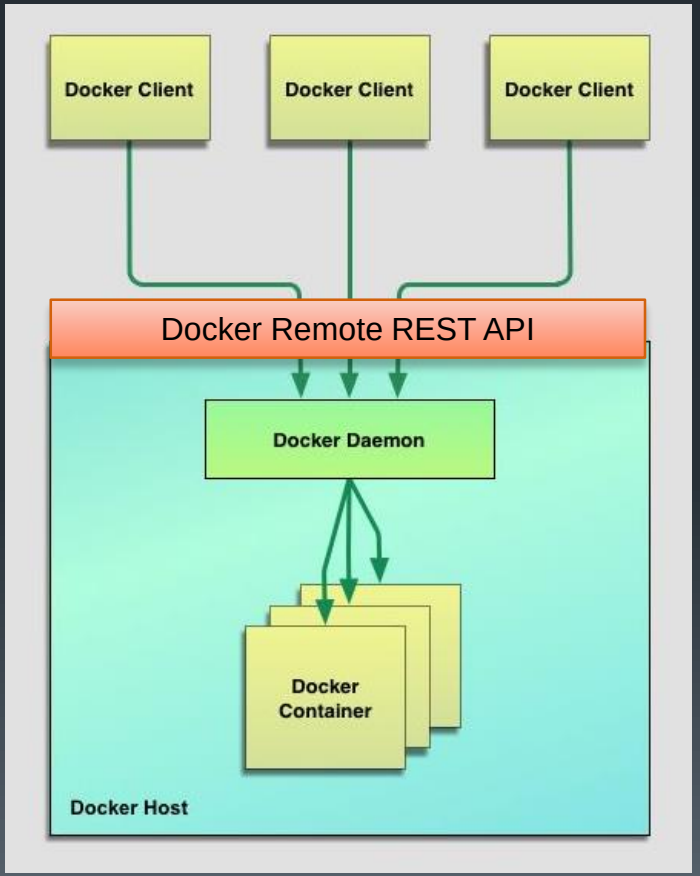
2: Working with Containers

Objectives

- Explain the basic Docker architecture
  - The Docker engine
  - The Docker client
  - Registries
  - Images
  - Containers
- Understand the nature of containers
- Explore docker commands for starting and controlling containers

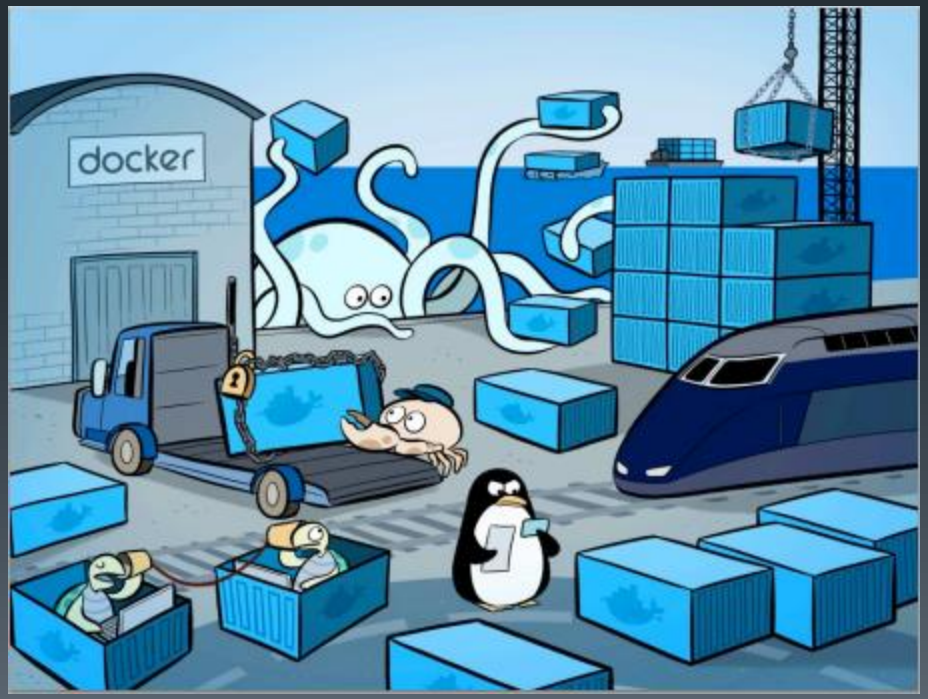
Core Docker Components

- "docker", the Docker client and server
  - The Docker daemon (docker daemon) runs on Linux hosts as a typical daemon supporting docker container execution
    - The Docker daemon is to containers as a hypervisor is to VMs
  - The Docker client (docker) can be used to direct operations of local and remote Docker daemons
  - Docker client processes talk to Docker daemon processes through the Docker daemon RESTful API
- Docker Images
  - Images are used to generate containers just as VM images are used to create VM instances
- Registries
  - Registries are indexes from which Docker Images can be located and retrieved
  - The Docker Hub is an internet based registry with support for public and private images
  - Private registries can be created for an organization's internal use
- Docker Containers
  - A container is a running instance of a Docker image



The Shipping Container Analogy

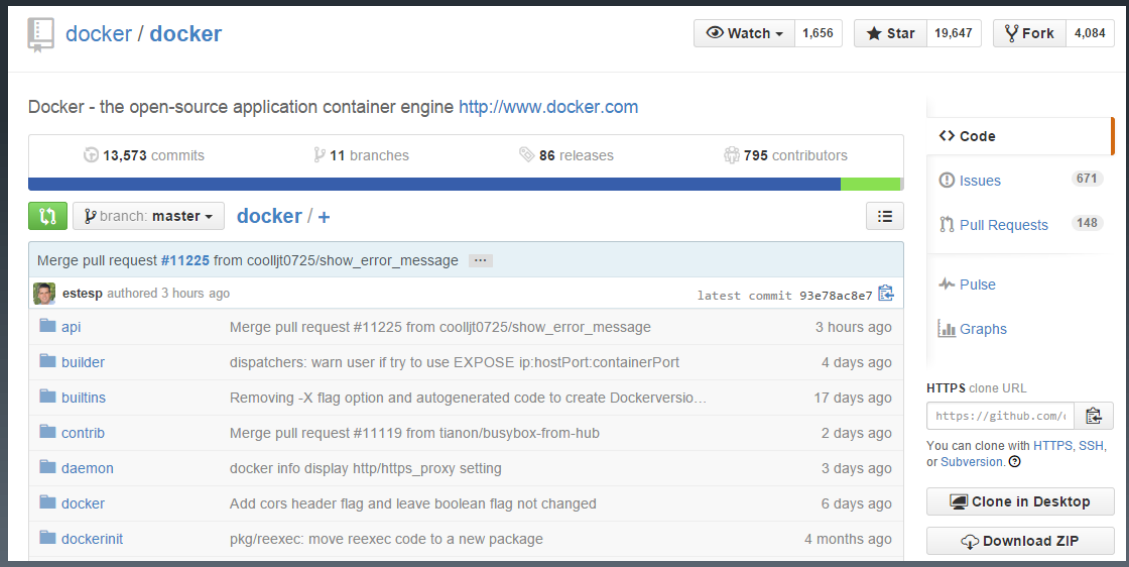
- Like a normal shipping container, Docker containers are:
  - Interchangeable
  - Stackable
  - Portable
  - Generic
- Making it easy to build a range of applications for deployment anywhere
  - CI Test Platform
  - Cloud Deployment
  - Local execution
  - Partner distribution
  - Software as a Service



Source: sd times

Docker Source

- Docker is 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
- The Docker source code can be found on GitHub
  - https://github.com/docker/docker



Container Lifecycle Control

- Docker Containers provide support for typical service and VM control operations
  - ps — List containers
  - create — Create a new container
  - start — Start a container running
  - stop — Stop a running container
  - restart — Restart a running container
  - run — Run a command in a new container (create + start)
  - pause — Pause all processes within a container
  - unpause — Unpause a paused container

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

Start one or more stopped containers
```

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

Stop a running container by sending SIGTERM and then SIGKILL after a
grace period

  -t, --time=10      Seconds to wait for stop before killing it
```

```
Usage: docker pause CONTAINER [CONTAINER...]

Pause all processes within a container
```

```
Usage: docker unpause CONTAINER [CONTAINER...]

Unpause all processes within a container
```

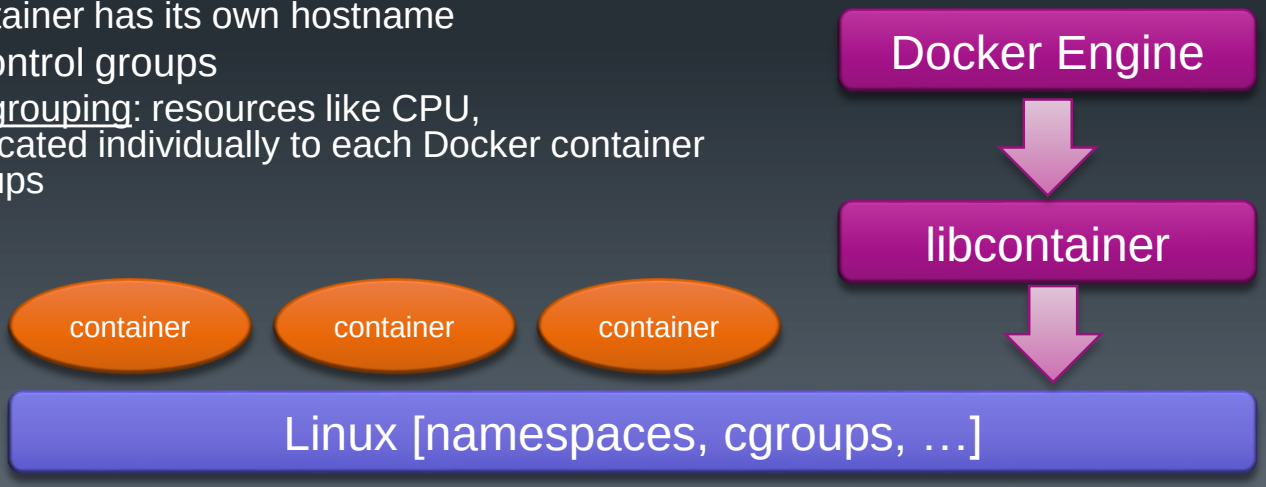
Docker containers work particularly well with applications characterized as:

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

Other applications can also benefit from containerization

Container Enablers

- Docker installations include the following elements which enable container operation and isolation:
  - libcontainer - Docker's native container interface since Docker v0.9
    - https://github.com/docker/libcontainer (source in Go)
    - The older Linux lxc (circa 2009) container interface is also still supported at present
      - Many new commands no longer work with lxc (e.g. docker exec)
      - You must boot the Docker daemon with either libcontainer or lxc (not both)
    - The Parallels (now Odin) libct interface was merged with libcontainer in mid 2014
    - RedHat's project atomic is working toward containerizing Linux with libcontainer
    - Now the standard container library/format of the OCI (subsuming appc/rkt)
  - Linux kernel 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
    - UTS isolation: each container has its own hostname
  - cgroups - Linux kernel control groups
    - Resource isolation and grouping: resources like CPU, memory and I/O are allocated individually to each Docker container using kernel control groups



Docker Daemon

- The Docker Engine and client use the same binary: /usr/bin/docker
- The Docker daemon runs automatically when installed from packages
  - /usr/bin/docker daemon
- The daemon listens on a Unix socket
  - /var/run/docker.sock
- The “docker” group owns the socket, user in that group can run docker without sudo
  - The –H switch can be used to instruct the docker daemon and client to bind to an alternative port
    - You can pass multiple –H switches to bind to several specific interfaces
      - e.g. “$ sudo docker daemon -H tcp://0.0.0.0:4444” would run the docker daemon (-d) on port 4444 on all network interfaces
      - e.g. “$ sudo docker daemon -H tcp://0.0.0.0:4444 -H unix:///home/docker/docker.sock”
- Docker client server communications are not authenticated, precautions should be taken
- Settings are persisted in the server configuration file
  - /etc/default/docker #Ubuntu
  - /usr/lib/systemd/system/docker.service #RHEL/Centos
    - Sometimes: /etc/sysconfig/docker
- 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 primary working directory

```
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 initramfs
-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:~$ sudo ls -l /var/lib/docker
total 44
drwxr-xr-x  5 root root 4096 Mar  1 14:04 aufs
drwx------  3 root root 4096 Mar  1 17:52 containers
drwx------  3 root root 4096 Mar  1 14:04 execdriver
drwx------ 11 root root 4096 Mar  1 16:49 graph
drwx------  2 root root 4096 Mar  1 14:04 init
-rw-r--r--  1 root root 5120 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 14:04 volumes
docker@ubuntu:~$
```

```
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 must 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://0.0.0.0:4444"
```

- Proxy servers can be configured with environment variables:
  - 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.

```
docker@ubuntu:~$ sudo docker -H "unix:///var/run/docker.sock" info
Containers: 0
Images: 0
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
 Backing Filesystem: extfs
 Dirs: 0
Execution Driver: native-0.2
Kernel Version: 3.13.0-46-generic
Operating System: Ubuntu 14.04.2 LTS
CPUs: 2
Total Memory: 1.948 GiB
Name: ubuntu
ID: F7LK:OBTM:7K5M:VQOI:HKDN:HGLP:ZMEZ:YM5Q:X3PV:5HCV:GRWV:TCJA
WARNING: No swap limit support
```

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 sub command followed by the --help swtich
    - e.g. $ sudo docker run --help
- The run command accepts an image and a command to run
- Run creates a new container from the image to run the command within
  - The -i switch can be used to preserves the container's STDIN stream and the -t switch connects it to the current shell, allowing you to create an interactive session with the process started within the container

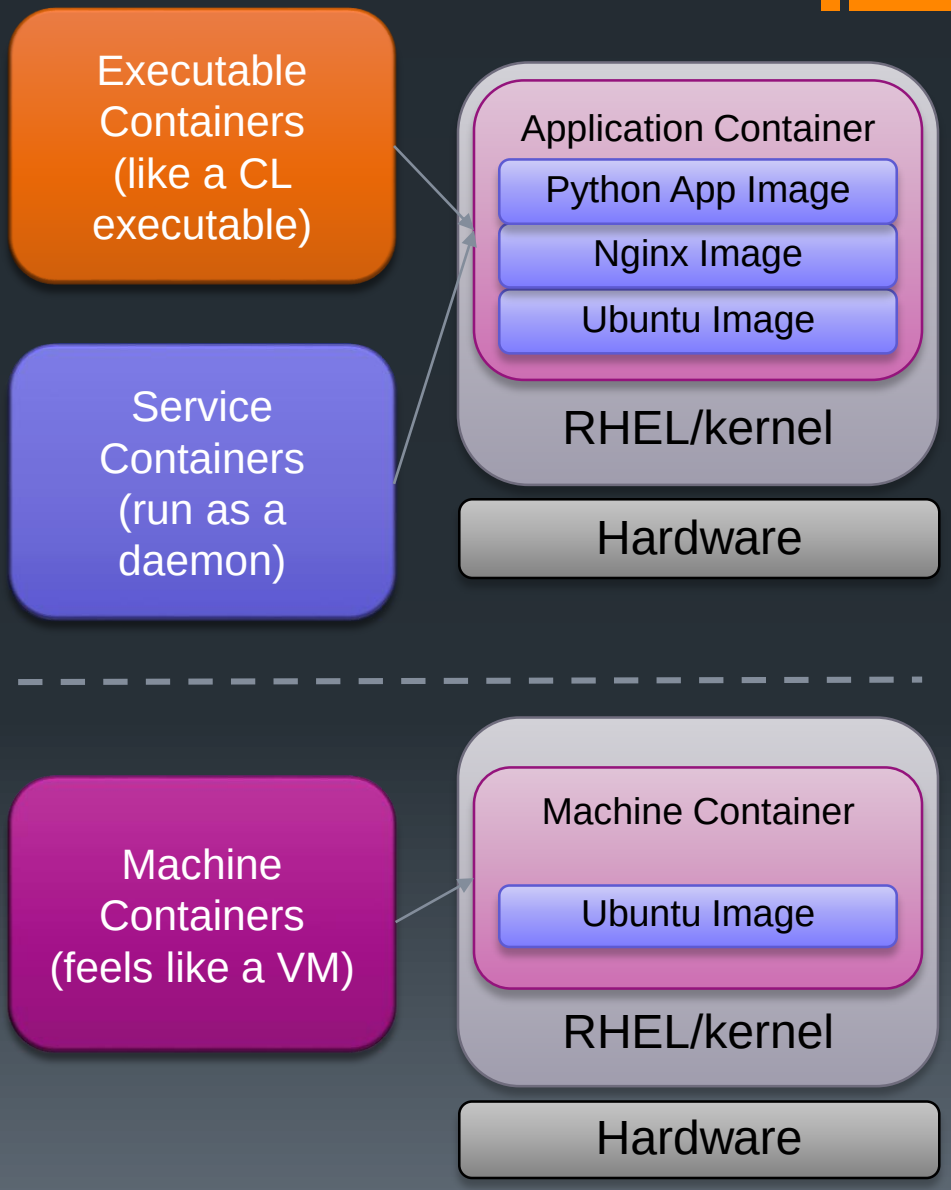
```
docker@ubuntu:~$ sudo docker run -i -t cirros /bin/sh
Unable to find image 'cirros:latest' locally
511136ea3c5a: Pull complete
16a464be5494: Pull complete
56124ee8d2e8: Pull complete
5165258bee80: Pull complete
cc6bc4ab5c0a: Pull complete
8d202478b999: Pull complete
cirros:latest: The image you are pulling has been verified. Important: image
verification is a tech preview feature and should not be relied on to provide
security.
Status: Downloaded newer image for cirros:latest
/ # hostname
8199df37bde6
/ # cat /etc/os-release
NAME=Buildroot
VERSION=2012.05
ID=buildroot
VERSION_ID=2012.05
PRETTY_NAME="Buildroot 2012.05"
/ # exit
docker@ubuntu:~$ cat /etc/os-release
NAME="Ubuntu"
VERSION="14.04.2 LTS, Trusty Tahr"
...
```

Docker uses the specified image to create a new container with an isolated filesystem, and network interface

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

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

Run a command in a new container

  -a, --attach=[]            Attach to STDIN, STDOUT or STDERR.
  --add-host=[]              Add a custom host-to-IP mapping (host:ip)
  -c, --cpu-shares=0         CPU shares (relative weight)
  --cap-add=[]               Add Linux capabilities
  --cap-drop=[]              Drop Linux capabilities
  --cidfile=""               Write the container ID to the file
  --cpuset=""                CPUs in which to allow execution (0-3, 0,1)
```

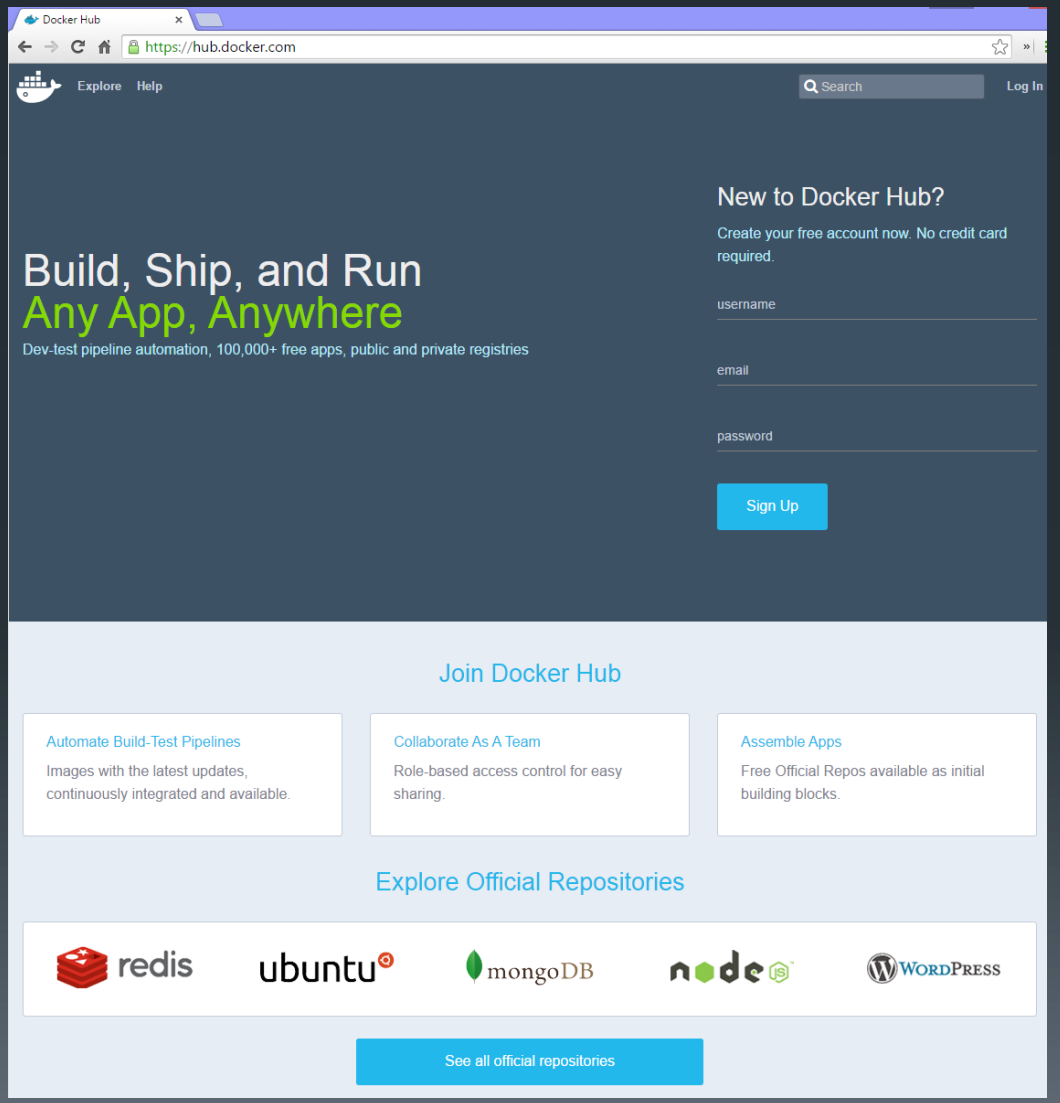
Types of Containers

- Containers are often organized into categories designating their purpose
  - Application Containers
    - Executable containers
      - $ docker 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 run -d my_co_webserver
      - Containers of this type are designed to house application services
      - Simplifies application deployment
  - Machine Containers
    - $ docker run -it centos /bin/bash
    - 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



The Docker Hub

- The Docker Hub is a registry supporting dynamic container image location and download
- The Docker Hub supports a range of base images
  - Cirros
  - Ubuntu
  - Fedora
  - Debian
  - Centos
  - etc .
- These images can be used as the basis for building custom images based on the desired operating system and packaging the desired application components
  - Our prior example automatically downloaded the minimal Cirros Linux cloud image and then ran the Bourne shell within the container
- When docker can not find the requested image locally it looks on docker hub



Exploring a Centos Container

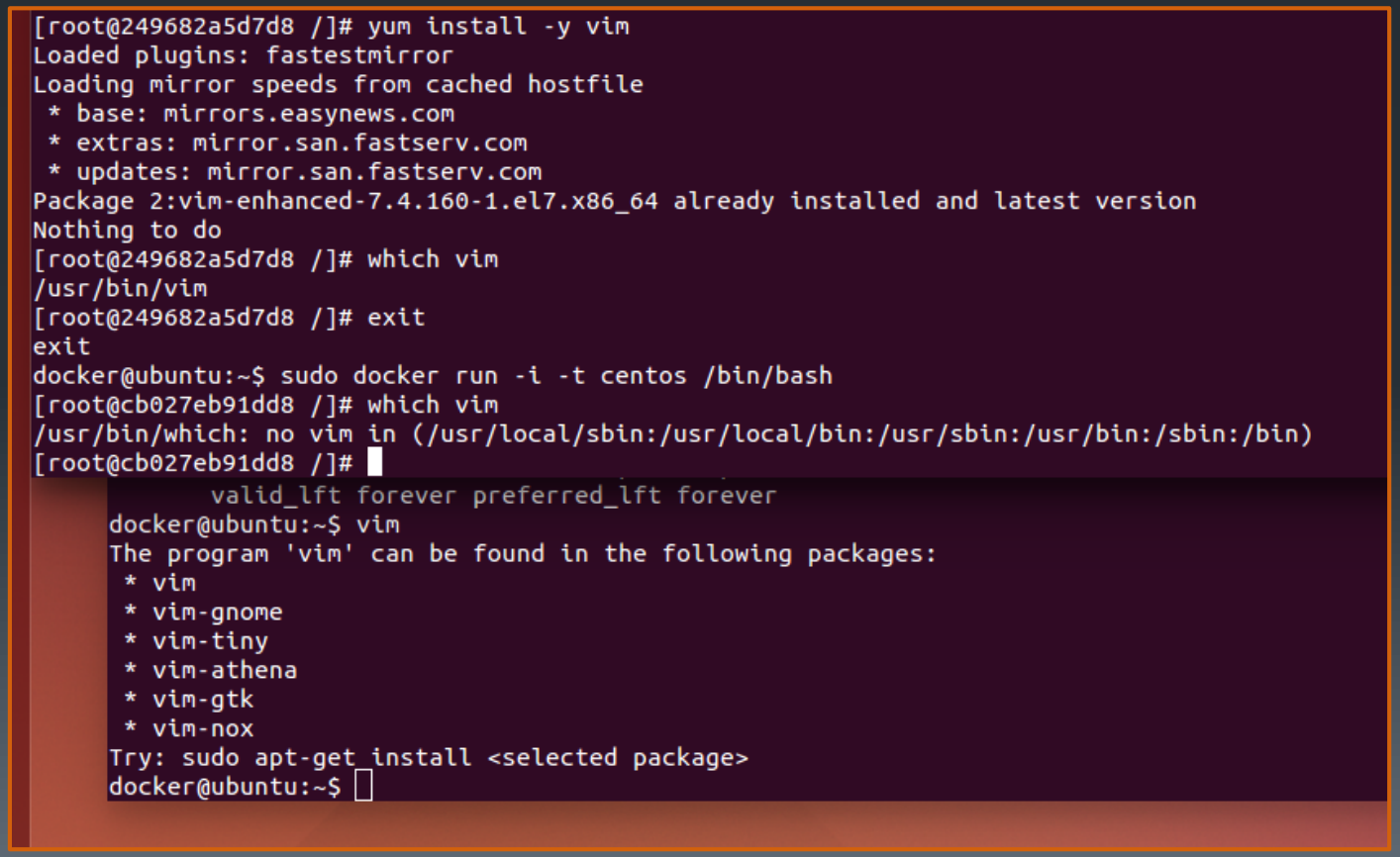
- The base centos image is 218MB, this is less than half the size of a base Centos server VM
- Running containers share the host kernel but supply their own OS personality
  - Libraries, executables, configuration files, etc
- Running containers have their own network interfaces

```
docker@ubuntu:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
docker@ubuntu:~$ sudo docker run -i -t centos /bin/bash
Unable to find image 'centos:latest' locally
5b12ef8fd570: Pull complete
dade6cb4530a: Pull complete
511136ea3c5a: Already exists
centos:latest: The image you are pulling has been verified. Important: image verification is a tech preview feature and should not be rel
ied on to provide security.
Status: Downloaded newer image for centos:latest
[root@84cab6dca071 /]# uname -a
Linux 84cab6dca071 3.13.0-46-generic #76-Ubuntu SMP Thu Feb 26 18:52:13 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux
[root@84cab6dca071 /]# 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/"

[root@84cab6dca071 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UN
    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
14: eth0: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    link/ether 02:42:ac:11:00:07 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.7/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe11:7/64 scope link
       valid_lft forever preferred_lft forever
[root@84cab6dca071 /]#
```

```
docker@ubuntu:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    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: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:a8:1b:76 brd ff:ff:ff:ff:ff:ff
    inet 192.168.235.140/24 brd 192.168.235.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fea8:1b76/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 56:84:7a:fe:97:99 brd ff:ff:ff:ff:ff:ff
    inet 172.17.42.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::5484:7aff:fefe:9799/64 scope link
       valid_lft forever preferred_lft forever
```

Container Isolation

- Images create copy on write containers by default
- 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, repeatable, initial state



Listing Containers

- Containers can be referenced by ID or Name
  - IDs are UUIDs and short versions are displayed by default
    - UUIDs are guaranteed to be unique
  - Container names are generated by default
    - The --name switch allows the name to be set explicitly
    - The “docker rename” command allows you to rename an existing container
    - Names must also be unique within the scope of a particular docker daemon
- By default, containers use their container id as their hostname
  - Hostname can be set with the -h switch
- The ps subcommand displays running containers
  - The -a switch displays all containers (running and stopped)

```
[root@c94802d4a237 /]# cat /etc/hostname
c94802d4a237
[root@c94802d4a237 /]#
```

```
docker@ubuntu:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
c94802d4a237        centos:latest       "/bin/bash"         About a minute ago   Up About a minute                       jolly_mestorf

docker@ubuntu:~$
```

```
docker@ubuntu:~$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS               NAMES
cb027eb91dd8        centos:latest       "/bin/bash"         6 minutes ago       Up 6 minutes                                    compassionate_goldstine
249682a5d7d8        centos:latest       "/bin/bash"         12 minutes ago      Exited (0) 6 minutes ago                        kickass_bell
37fe12b8a425        centos:latest       "/bin/bash"         13 minutes ago      Exited (0) 12 minutes ago                       sad_wilson
c94802d4a237        centos:latest       "/bin/bash"         21 minutes ago      Exited (0) 15 minutes ago                       jolly_mestorf
84cab6dca071        centos:latest       "/bin/bash"         28 minutes ago      Exited (0) 24 minutes ago                       mad_engelbart
8199df37bde6        cirros:latest       "/bin/sh"           32 minutes ago      Exited (0) 30 minutes ago                       gloomy_morse
d624f28e47b6        cirros:latest       "/bin/sh"           About an hour ago   Exited (0) 59 minutes ago                       adoring_ritchie
```

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
- The docker “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 will tail continuously
    - Adding -t will display the log entry timestamp

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

```
docker@ubuntu:~$ sudo docker run --name hellosvc -d cirros /bin/sh -c "while true; do echo hello world; sleep 1; done"
b4cfff7fa00c5adf741f2073be474ca13b5efb1f7dcb67884b2d446d8ee1b7a5
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
b4cfff7fa00c        cirros:latest       "/bin/sh -c 'while t   8 seconds ago       Up 7 seconds                            hellosvc

docker@ubuntu:~$ docker logs hellosvc
hello world
hello world
hello world
hello world
hello world
```

```
docker@ubuntu:~$ docker logs --tail 3 -t hellosvc
2015-03-02T02:03:08.126241861Z hello world
2015-03-02T02:03:09.129319262Z hello world
2015-03-02T02:03:10.130277365Z hello world
```

Controlling Containers

- start
  - You can resume a stopped container using the start sub command
- restart
  - The restart command stops the specified container and then runs it again
  - If the container is not already running, restart simply starts it
- attach
  - You can attach to a running container using the attach subcommand
- stop
  - You can stop a running daemonized container with the stop command

```
docker@ubuntu:~$ which vim
docker@ubuntu:~$ sudo docker start -i kickass_bell
[root@249682a5d7d8 /]# which vim
/usr/bin/vim
[root@249682a5d7d8 /]# exit
exit
docker@ubuntu:~$ sudo docker start -i compassionate_goldstine
[root@cb027eb91dd8 /]# which vim
/usr/bin/which: no vim in (/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin)
[root@cb027eb91dd8 /]# exit
exit
docker@ubuntu:~$
```

```
docker@ubuntu:~$ sudo docker restart kickass_bell
kickass_bell
docker@ubuntu:~$ sudo docker attach kickass_bell

[root@249682a5d7d8 /]# which vim
/usr/bin/vim
[root@249682a5d7d8 /]#
```

Displaying container processes

- top
  - You can examine the processes running within a container using the top subcommand
  - Note that containers provide a passive operating system and library interface with no running processes or services enabled by default
- stats (added in docker 1.5)
  - The stats sub command displays a live status view of the specified containers

```
docker@ubuntu:~$ docker top hellosvc
UID                 PID                 PPID                C                   STIME               TTY                 TIME                CMD
root                9643                6577                0                   17:52               ?                   00:00:00            /bin/sh -c while true;
do echo hello world; sleep 1; done
root                10535               9643                0                   18:05               ?                   00:00:00            sleep 1
docker@ubuntu:~$
```

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

```
docker@ubuntu: ~
CONTAINER           CPU %               MEM USAGE/LIMIT     MEM %               NET I/O
hellosvc            0.05%               324 KiB/1.948 GiB   0.02%               648 B/648 B
```

Adding Processes, Copying & Auto Container Restart

- docker exec <contianer> <cmd> (added in docker 1.3)
  - Runs an arbitrary process within a container
    - -d runs new process detached
    - -I runs new process in interactive mode
- docker cp <container>:<path> <hostdir>
- docker cp <hostdir> <container>:<path>
- docker 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

```
docker@ubuntu:~$ docker run -d centos tail -f /dev/null
14e41498e60a2502662ed4045aa41207e75688d370336984c2f731c44be4e9d2
docker@ubuntu:~$ docker exec -it 14e /bin/bash
[root@14e41498e60a /]# echo "hi docker" > test.dat
[root@14e41498e60a /]# exit
exit
docker@ubuntu:~$ docker cp 14e:/test.dat .
docker@ubuntu:~$ cat test.dat
hi docker
docker@ubuntu:~$
```

```
docker@ubuntu: ~
docker@ubuntu:~$ docker run -d --restart=always ubuntu sleep 20
6f4d40f54caded01273694bb3b6a2fe15f66225f55ed6361ec23a71b4d3c7fc6
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
6f4d40f54cad        ubuntu:latest       "sleep 20"          6 seconds ago       Up 4 seconds                            backstabbing_heisenberg
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
6f4d40f54cad        ubuntu:latest       "sleep 20"          15 seconds ago      Up 13 seconds                           backstabbing_heisenberg
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
6f4d40f54cad        ubuntu:latest       "sleep 20"          20 seconds ago      Up 18 seconds                           backstabbing_heisenberg
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
6f4d40f54cad        ubuntu:latest       "sleep 20"          25 seconds ago      Up 3 seconds                            backstabbing_heisenberg
```

Summary

- The Docker platform includes several parts
  - The Docker daemon
  - The 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

Lab 2

- Running containers

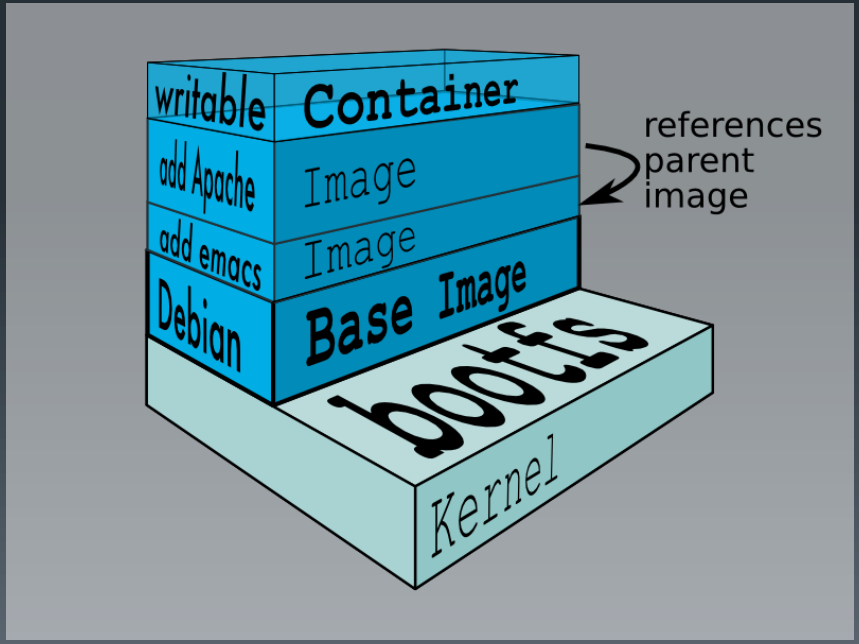
3: Docker Images

Objectives

- Describe Docker image layering
- Understand the nature of layered file system implementation in Docker
- Gain familiarity with the Docker directory structure
- Move images between registries and the local Docker Host
- Explore image creation

Docker Images

- Docker containers are constructed by sequentially mounting a set of file systems from one or more images
- A Docker image is a file system layer with an optional parent image reference
  - Layered images tend to supply one specific feature on top of the parent image
  - Upper layer files mask files at lower layers with the same pathname
- An image with no parent image is called a Base Image
  - Base images tend to be largish and house an operating system with a root file system (e.g. Debian, Ubuntu, Centos, etc.)
- Image file systems are immutable within a Container
  - Allows one Image to support multiple Container instances with repeatable results
  - Reduces the disk and memory footprint of a given set of containers which share the same read only images
- Containers have a writable file system
  - The container file system is initially empty
  - All writes go to this file system and overlay any matching underlying image files
  - In this way, container file systems contain only the delta between their filesystem state and that of their underlying images
  - The view from the top down, including all file systems in the stack, is called the union file system
- The Docker bootfs is a special layer
  - Supplies the in memory file system interface to the kernel
  - Also supplies the kernel library interface



Listing Images

- The docker images subcommand can be used to list local images
- Images are stored as file system layers
  - In our Ubuntu lab system: /var/lib/docker/aufs/layers

```
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
78f9cd309924        cirros:latest       "/bin/sh -c 'while t   19 minutes ago      Up 19 minutes                           hellosvc
docker@ubuntu:~$ sudo ls -l /var/lib/docker/containers
total 4
drwx------ 2 root root 4096 Mar  2 10:02 78f9cd309924608d1ea9bc6c7766a30164f51dc26dfe80ceb255bf0817bc2a64
docker@ubuntu:~$ sudo ls -l /var/lib/docker/containers/78f9cd309924608d1ea9bc6c7766a30164f51dc26dfe80ceb255bf0817bc2a64
total 124
-rw------- 1 root root 96377 Mar  2 10:21 78f9cd309924608d1ea9bc6c7766a30164f51dc26dfe80ceb255bf0817bc2a64-json.log
-rw-r--r-- 1 root root  1862 Mar  2 10:02 config.json
-rw-r--r-- 1 root root   370 Mar  2 10:02 hostconfig.json
-rw-r--r-- 1 root root    13 Mar  2 10:02 hostname
-rw-r--r-- 1 root root   175 Mar  2 10:02 hosts
-rw-r--r-- 1 root root   208 Mar  2 10:02 resolv.conf
-rw-r--r-- 1 root root    71 Mar  2 10:02 resolv.conf.hash
docker@ubuntu:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
centos              latest              dade6cb4530a        3 weeks ago         210.1 MB
cirros              latest              8d202478b999        3 weeks ago         7.698 MB
docker@ubuntu:~$ sudo ls -l /var/lib/docker/aufs/layers
total 36
-rw-r--r-- 1 root root  65 Mar  1 16:11 16a464be5494a73be34a3055b77ae00d072a4f9389897d1a786de0079219aaeb
-rw-r--r-- 1 root root   0 Mar  1 16:11 511136ea3c5a64f264b78b5433614aec563103b4d4702f3ba7d4d2698e22c158
-rw-r--r-- 1 root root 195 Mar  1 16:11 5165258bee80ccf426b1ae475b038bb1b7aebff3dd439101e0b0850674a67aa5
-rw-r--r-- 1 root root 130 Mar  1 16:11 56124ee8d2e899fe47927483be6fa74c311a16079887851dd05dc86de6c7ce27
-rw-r--r-- 1 root root  65 Mar  1 16:47 5b12ef8fd57065237a6833039acc0e7f68e363c15d8abb5cacce7143a1f7de8a
-rw-r--r-- 1 root root 460 Mar  2 10:02 78f9cd309924608d1ea9bc6c7766a30164f51dc26dfe80ceb255bf0817bc2a64
-rw-r--r-- 1 root root 390 Mar  2 10:02 78f9cd309924608d1ea9bc6c7766a30164f51dc26dfe80ceb255bf0817bc2a64-init
-rw-r--r-- 1 root root 325 Mar  1 16:11 8d202478b999318c8ec89c89a808ffd2af76b787072ec02056b10c7ea88a7b9b
-rw-r--r-- 1 root root 260 Mar  1 16:11 cc6bc4ab5c0a9046824fe598f157d9a49b8e197f1661d987f5ba4e1fa5f7e621
-rw-r--r-- 1 root root 130 Mar  1 16:49 dade6cb4530a852b83bc34f6f3904a10879632947c01f1ae0447ff9dbb37e12b
docker@ubuntu:~$
```

containerfs

imagefs

basefs

The containers directory

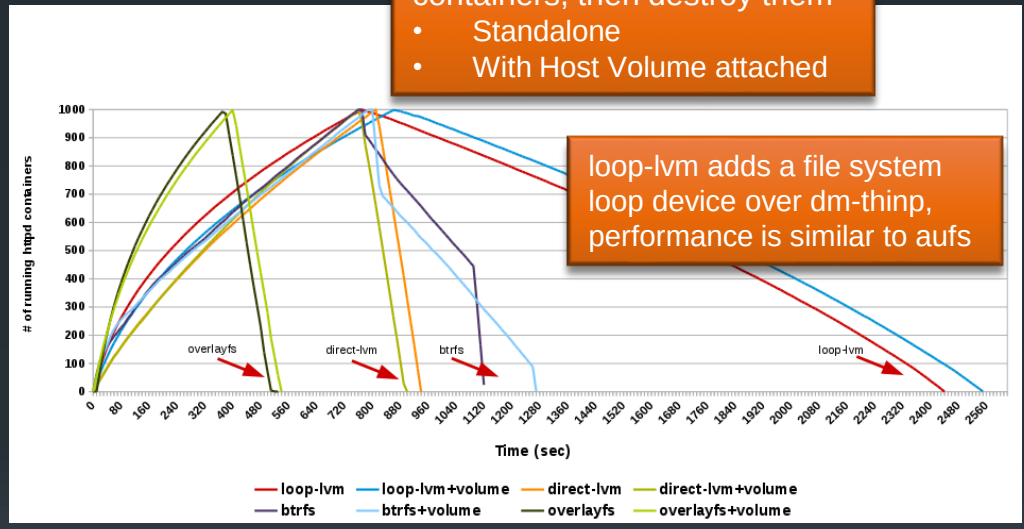
- /var/lib/docker/containers
  - Holds container metadata
    - Log files
    - Configuration files

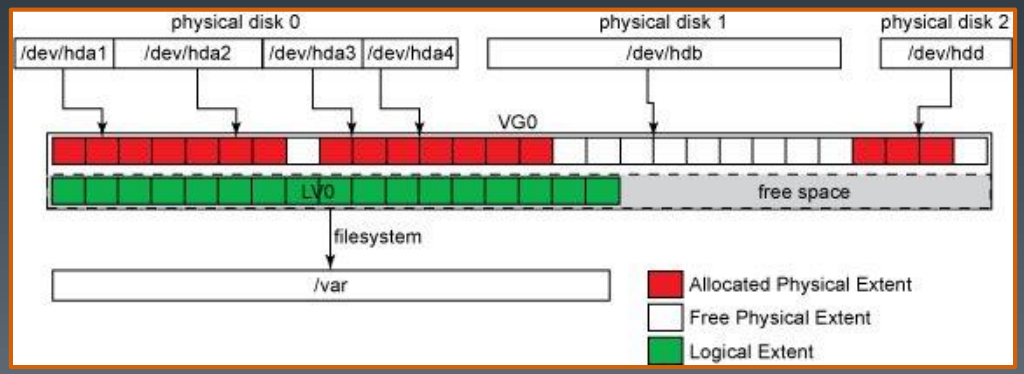
```
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                CREATED             STATUS              PORTS               NAMES
78f9cd309924        cirros:latest       "/bin/sh -c 'while t   28 minutes ago      Up 28 minutes                           hellosvc
docker@ubuntu:~$ sudo ls -l /var/lib/docker/containers
total 4
drwx------ 2 root root 4096 Mar  2 10:02 78f9cd309924608d1ea9bc6c7766a30164f51dc26dfe80ceb255bf0817bc2a64
docker@ubuntu:~$ sudo ls -l /var/lib/docker/containers/78f9cd309924608d1ea9bc6c7766a30164f51dc26dfe80ceb255bf0817bc2a64
total 168
-rw------- 1 root root 142557 Mar  2 10:31 78f9cd309924608d1ea9bc6c7766a30164f51dc26dfe80ceb255bf0817bc2a64-json.log
-rw-r--r-- 1 root root   1862 Mar  2 10:02 config.json
-rw-r--r-- 1 root root    370 Mar  2 10:02 hostconfig.json
-rw-r--r-- 1 root root     13 Mar  2 10:02 hostname
-rw-r--r-- 1 root root    175 Mar  2 10:02 hosts
-rw-r--r-- 1 root root    208 Mar  2 10:02 resolv.conf
-rw-r--r-- 1 root root     71 Mar  2 10:02 resolv.conf.hash
docker@ubuntu:~$ sudo cat /var/lib/docker/containers/78f9cd309924608d1ea9bc6c7766a30164f51dc26dfe80ceb255bf0817bc2a64/78f9cd30992460
0164f51dc26dfe80ceb255bf0817bc2a64-json.log | tail
{"log":"hello world\n","stream":"stdout","time":"2015-03-02T18:31:10.95816233Z"}
{"log":"hello world\n","stream":"stdout","time":"2015-03-02T18:31:11.973632681Z"}
{"log":"hello world\n","stream":"stdout","time":"2015-03-02T18:31:12.989387399Z"}
{"log":"hello world\n","stream":"stdout","time":"2015-03-02T18:31:13.993366868Z"}
{"log":"hello world\n","stream":"stdout","time":"2015-03-02T18:31:14.999398762Z"}
{"log":"hello world\n","stream":"stdout","time":"2015-03-02T18:31:16.015146837Z"}
{"log":"hello world\n","stream":"stdout","time":"2015-03-02T18:31:17.030816919Z"}
{"log":"hello world\n","stream":"stdout","time":"2015-03-02T18:31:18.046440342Z"}
{"log":"hello world\n","stream":"stdout","time":"2015-03-02T18:31:19.050202311Z"}
{"log":"hello world\n","stream":"stdout","time":"2015-03-02T18:31:20.060329939Z"}
docker@ubuntu:~$
```

Filesystem Backends

- Docker depends on the efficient use of layered images
  - Various file system features in the kernel implement file system layering
- Each container has two layers
  - The init layer, whose parent is the container's specified image (RO operational data)
    - Contains .dockerinit and other required Docker files
  - A child of init which contains the container specific content (RW application data)
- Committing a container creates a new image layer based on the image the container was created from
- Docker backends (listed in Docker preference order):
  - OverlayFS - implements union mount for other FS
    - Introduced in kernel 3.18
    - Very fast with SSDs
  - aufs – "Advanced multi layered Unification FileSystem" stores layers as directories with aufs metadata
    - FILE level scheme with shared storage
    - Default Ubuntu but not widely supported on RHEL
    - Originally limited to 42 layers, now 127 (the 127 limit applies to all drivers as of Docker 0.7.2)
  - btrfs – uses file system level snapshotting
    - BLOCK level scheme with shared storage
    - /var/lib/docker must be on a btrfs file system, layers stored as sub-volumes in /var/lib/docker/btrfs/subvolumes
  - devicemapper - uses device-mapper thin provisioning module (dm-thinp) to implement layers
    - Default on RHEL systems
    - BLOCK level scheme with shared storage
    - Device-mapper is the kernel part of the LVM2 system
  - vfs – universally supported but slow and inefficient
    - FILE level scheme with no shared storage
    - Each layer is a separate directory with a deep copy of its parent (no COW)
- Container Independent Volumes
  - Write-heavy I/O (databases, logs, etc.) may be done to a normal host volume eliminating unnecessary storage backend overhead
  - Host volumes can be accessed across multiple concurrent containers
    - Like a SAN without the network overhead

Create 1000 Apache containers, then destroy them
- Standalone
- With Host Volume attached





Container deltas

- /var/lib/docker/aufs/diff
  - Holds copy on write (COW) data
    - Files from any other layer which have been modified
    - e.g. /etc/hostname
  - Holds new files
    - Specific to the container
    - e.g. /myfile & myfile2

```
/ # ls -l
total 6280
drwxrwxr-x    2 root     root          4096 Sep  8 08:07 bin
drwxrwxr-x    3 root     root          4096 Sep  8 08:07 boot
drwxr-xr-x    5 root     root           380 Mar  2 11:54 dev
drwxrwxr-x   14 root     root          4096 Mar  2 11:54 etc
drwxrwxr-x    4 root     root          4096 Sep  8 09:24 home
-rwxr-xr-x    1 root     root          2616 Sep  8 09:24 init
drwxrwxr-x    3 root     root          4096 Sep  8 08:07 lib
lrwxrwxrwx    1 root     root            11 Sep  8 08:52 linuxrc -> bin/busybox
drwxrwxr-x    2 root     root          4096 Sep  8 08:37 media
drwxrwxr-x    2 root     root          4096 Sep  8 08:37 mnt
-rw-r--r--                            227128 Mar  2 11:57 myfile
-rw-r--r--                           6132456 Mar  2 12:00 myfile2
drwxrwxr-x                              4096 Sep  8 08:07 old-root
drwxrwxr-x    2 root     root          4096 Sep  8 08:37 opt
dr-xr-xr-x  336 root     root             0 Mar  2 11:54 proc
drwx------    2 root     root          4096 Mar  2 11:56 root
drwxr-xr-x    2 root     root          4096 Sep  8 09:24 run
drwxrwxr-x    2 root     root          4096 Sep  8 08:07 sbin
dr-xr-xr-x   13 root     root             0 Mar  2 11:54 sys
drwxrwxrwt    3 root     root          4096 Sep  8 09:24 tmp
drwxrwxr-x    6 root     root          4096 Sep  8 08:07 usr
drwxrwxr-x    8 root     root          4096 Sep  8 09:24 var
```

Container

```
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
172a2193b4b4        cirros:latest       "/bin/sh"           12 minutes ago      Up 12 minutes                           angry_rosalind
docker@ubuntu:~$ sudo ls -l /var/lib/docker/aufs/diff/172a2193b4b4b87997441234cb09a3e2107d271c8afb544b1f1443fe5f7dbd98
total 6220
-rw-r--r-- 1 root root  227128 Mar  2 10:57 myfile
-rw-r--r-- 1 root root 6132456 Mar  2 11:00 myfile2
drwx------ 2 root root    4096 Mar  2 10:56 root
docker@ubuntu:~$ sudo ls -l /var/lib/docker/aufs/diff/172a2193b4b4b87997441234cb09a3e2107d271c8afb544b1f1443fe5f7dbd98-init
total 8
drwxr-xr-x 3 root root 4096 Mar  2 10:54 dev
drwxrwxr-x 2 root root 4096 Mar  2 10:54 etc
docker@ubuntu:~$ sudo ls -l /var/lib/docker/aufs/diff/172a2193b4b4b87997441234cb09a3e2107d271c8afb544b1f1443fe5f7dbd98-init/etc
total 0
-rwxr-xr-x 1 root root  0 Mar  2 10:54 hostname
-rwxr-xr-x 1 root root  0 Mar  2 10:54 hosts
lrwxrwxrwx 1 root root 12 Mar  2 10:54 mtab -> /proc/mounts
-rwxr-xr-x 1 root root  0 Mar  2 10:54 resolv.conf
docker@ubuntu:~$
```

Host

docker diff

- The docker diff command can be used to view the changes in the container's filesystem layer
- Three types of deltas are recorded
  - A - Add
  - D - Delete
  - C - Change

```
docker@ubuntu:~$ docker run -it centos
[root@1eced8a123ff /]# echo "hello" > README.md
[root@1eced8a123ff /]#
```

```
docker@ubuntu: ~
docker@ubuntu:~$ docker diff 1eced
A /README.md
C /tmp
docker@ubuntu:~$
```

Container Information

- inspect
  - You can retrieve detailed container information using the inspect subcommand
  - Inspect returns a JSON document with container details
  - -f
    - The -f (format)switch allows you to select specific elements to display

```
docker@ubuntu:~$ docker inspect hellosvc | head -20
[{
    "AppArmorProfile": "",
    "Args": [
        "-c",
        "while true; do echo hello world; sleep 1; done"
    ],
    "Config": {
        "AttachStderr": false,
        "AttachStdin": false,
        "AttachStdout": false,
        "Cmd": [
            "/bin/sh",
            "-c",
            "while true; do echo hello world; sleep 1; done"
        ],
        "CpuShares": 0,
        "Cpuset": "",
        "Domainname": "",
        "Entrypoint": null,
        "Env": [
docker@ubuntu:~$
```

```
docker@ubuntu:~$ docker inspect -f='{{.State}}' hellosvc
map[Paused:false Pid:9643 Restarting:false Error: ExitCode:0 FinishedAt:0001-01-01T00:00:00Z OOMKill
ed:false Running:true StartedAt:2015-03-02T01:52:58.9723728Z]
docker@ubuntu:~$ docker inspect -f='{{.State.StartedAt}}' hellosvc
2015-03-02T01:52:58.9723728Z
docker@ubuntu:~$
```

Image Tags

- Images are given tag names for easy reference
- One image may have several tags
- One repository may contain several images
- You can specify a particular image within a repository using a trailing : and the tag name
  - e.g. “ubuntu:14.04”
  - If no image tag is supplied, Docker assumes the “latest” tag

```
docker@ubuntu:~$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
centos                  latest              dade6cb4530a        3 weeks ago         210.1 MB
cirros                  latest              8d202478b999        3 weeks ago         7.698 MB
bufferoverflow/thrift   latest              5eb5528b2680        4 months ago        4.287 GB
docker@ubuntu:~$ docker search ubuntu
NAME                          DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
ubuntu                        Official Ubuntu base image                      1370      [OK]
dockerfile/ubuntu             Trusted automated Ubuntu (http://www.ubunt...   45                   [OK]
ansible/ubuntu14.04-ansible   Ubuntu 14.04 LTS with ansible                   41                   [OK]
dockerfile/ubuntu-desktop     Trusted automated Ubuntu Desktop (LXDE) (h...   21                   [OK]
ubuntu-upstart                Upstart is an event-based replacement for ...   21        [OK]
tutum/ubuntu                  Ubuntu image with SSH access. For the root...   14                   [OK]
sequenceiq/hadoop-ul
dorowu/ubuntu-deskt
guilhem/vagrant-ubu
ubuntu-debootstrap
tleyden5iwx/ubuntu-
webhippie/cedarish-
maxexcloo/ubuntu
rastasheep/ubuntu-s
nuagebec/ubuntu
nimmis/ubuntu
seetheprogress/ubuntu         Ubuntu image provided by seetheprogress us...   1                    [OK]
alsanium/ubuntu               Ubuntu Core image for Docker                    1                    [OK]
sylvainlasnier/ubuntu         Ubuntu 14.10 root docker images with commo...   1                    [OK]
densuke/ubuntu-jp-remix       Ubuntu Linuxの日本語remix風味です                          1                    [OK]
fike/ubuntu-postgresql        PostgreSQL 9.4 beta until 9.0 version runn...   1                    [OK]
mhutter/ubuntu                Base Ubuntu Image                               0                    [OK]
cpuguy83/ubuntu                                                               0                    [OK]
densuke/ubuntu-supervisor     densuke/ubuntu-jp-remix:trusty 上で supe...      0                    [OK]
partlab/ubuntu                Simple Ubuntu docker images.                    0                    [OK]
docker@ubuntu:~$ docker pull ubuntu
fa4fd76b09ce: Pull complete
fa4fd76b09ce: Download complete
1c8294cc5160: Download complete
117ee323aaa9: Download complete
2d24f826cb16: Download complete
511136ea3c5a: Download complete
Status: Downloaded newer image for ubuntu:latest
docker@ubuntu:~$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
ubuntu                  14.04.2             2d24f826cb16        9 days ago          188.3 MB
ubuntu                  latest              2d24f826cb16        9 days ago          188.3 MB
ubuntu                  trusty              2d24f826cb16        9 days ago          188.3 MB
ubuntu                  trusty-20150218.1   2d24f826cb16        9 days ago          188.3 MB
ubuntu                  14.04               2d24f826cb16        9 days ago          188.3 MB
centos                  latest              dade6cb4530a        3 weeks ago         210.1 MB
cirros                  latest              8d202478b999        3 weeks ago         7.698 MB
bufferoverflow/thrift   latest              5eb5528b2680        4 months ago        4.287 GB
docker@ubuntu:~$
```

```
docker@ubuntu:~$ docker run -i -t ubuntu:trusty /bin/bash
root@7125f5eae5b6:/# grep VERSION /etc/os-release
VERSION="14.04.2 LTS, Trusty Tahr"
VERSION_ID="14.04"
root@7125f5eae5b6:/# exit
exit
```

Displaying Repositories

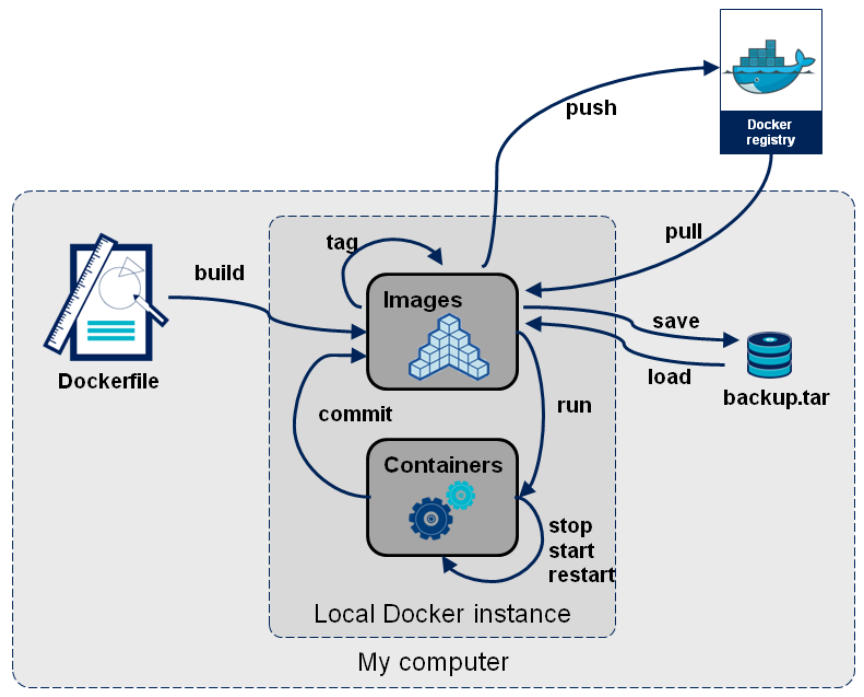
- You can scope the images sub command to examine a particular repository
- You can also specify a particular tagged image to pull
  - Required when you want to pull something other than the latest tag
  - Default is “latest”

```
docker@ubuntu:~$ docker images cirros
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
cirros              latest              8d202478b999        3 weeks ago         7.698 MB
docker@ubuntu:~$ docker images ubuntu
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
ubuntu              14.04.2             2d24f826cb16        9 days ago          188.3 MB
ubuntu              latest              2d24f826cb16        9 days ago          188.3 MB
ubuntu              trusty              2d24f826cb16        9 days ago          188.3 MB
ubuntu              trusty-20150218.1   2d24f826cb16        9 days ago          188.3 MB
ubuntu              14.04               2d24f826cb16        9 days ago          188.3 MB
docker@ubuntu:~$
```

```
docker@ubuntu:~$ docker pull fedora:21
00a0c78eeb6d: Pull complete
834629358fe2: Pull complete
511136ea3c5a: Already exists
fedora:21: The image you are pulling has been verified. Important: image verification is a tech preview feature and should not be relied on to provide
 security.
Status: Downloaded newer image for fedora:21
docker@ubuntu:~$ docker images fedora
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
fedora              21                  834629358fe2        8 weeks ago         241.3 MB
docker@ubuntu:~$ docker pull fedora
511136ea3c5a: Already exists
00a0c78eeb6d: Already exists
834629358fe2: Already exists
fedora:latest: The image you are pulling has been verified. Important: image verification is a tech preview feature and should not be relied on to pro
vide security.
Status: Image is up to date for fedora:latest
docker@ubuntu:~$ docker pull fedora:20
782cf93a8f16: Pull complete
6cece30db4f9: Pull complete
511136ea3c5a: Already exists
fedora:20: The image you are pulling has been verified. Important: image verification is a tech preview feature and should not be relied on to provide
 security.
Status: Downloaded newer image for fedora:20
docker@ubuntu:~$ docker images fedora
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
fedora              20                  6cece30db4f9        8 weeks ago         360.3 MB
fedora              21                  834629358fe2        8 weeks ago         241.3 MB
fedora              latest              834629358fe2        8 weeks ago         241.3 MB
docker@ubuntu:~$
```

Creating Images

- Docker images are the basis of containers
- Docker stores downloaded images on the Docker host
  - If a required image isn't already present on the host it is downloaded from a registry (by default the Docker Hub Registry)
- There are two ways to create new images
  - Update a container created from an image and commit the results to a new image
  - Use a Dockerfile to specify instructions to create an image



Committing a Container

- Committing an existing Container is a simple way to create a new Docker image
  1. Create a container with the desired base
  2. Make the changes necessary to the container layer
  3. Commit the container layer as a new image

1.
```
docker@ubuntu:~$ docker run -t -i ubuntu:14.04 /bin/bash
root@3802a4a2a2e9:/# apt-get install git
...
root@3802a4a2a2e9:/# git clone http://github.com/apache/thrift
…
```

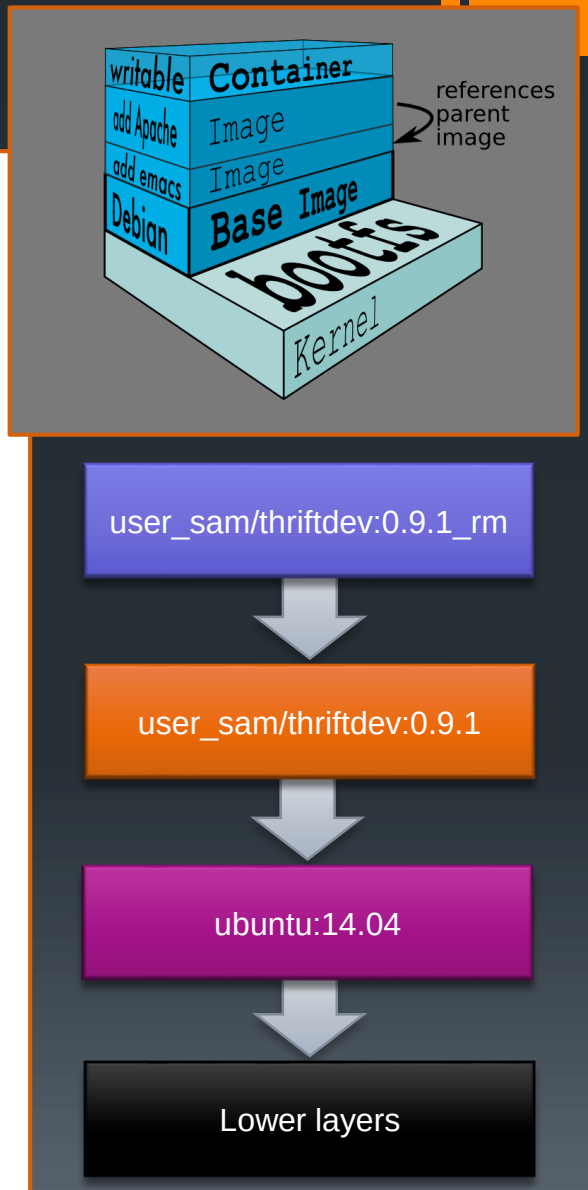
2.
```
root@3802a4a2a2e9:/# cd thrift
root@3802a4a2a2e9:/thrift# git checkout 0.9.1
...
root@3802a4a2a2e9:/thrift# exit
exit
```

3.
```
docker@ubuntu:~$ docker commit -m "Thrift dev" -a "Sam Thrift" 3802a4a2a2e9 user_sam/thriftdev:0.9.1
ca0310547ecb9e71370479f94b5977a358054210d74c06bc0f869e32d51e96ae
docker@ubuntu:~$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
user_sam/thriftdev      0.9.1               ca0310547ecb        8 seconds ago       247.7 MB
ubuntu                  14.04.2             2d24f826cb16        13 days ago         188.3 MB
ubuntu                  14.04               2d24f826cb16        13 days ago         188.3 MB
ubuntu                  latest              2d24f826cb16        13 days ago         188.3 MB
ubuntu                  trusty              2d24f826cb16        13 days ago         188.3 MB
ubuntu                  trusty-20150218.1   2d24f826cb16        13 days ago         188.3 MB
ubuntu                  12.04               1f80e9ca2ac3        13 days ago         131.5 MB
ubuntu                  12.04.5             1f80e9ca2ac3        13 days ago         131.5 MB
ubuntu                  precise             1f80e9ca2ac3        13 days ago         131.5 MB
ubuntu                  precise-20150212    1f80e9ca2ac3        13 days ago         131.5 MB
centos                  latest              dade6cb4530a        4 weeks ago         210.1 MB
cirros                  latest              8d202478b999        4 weeks ago         7.698 MB
fedora                  20                  6cece30db4f9        9 weeks ago         360.3 MB
fedora                  21                  834629358fe2        9 weeks ago         241.3 MB
fedora                  latest              834629358fe2        9 weeks ago         241.3 MB
bufferoverflow/thrift   latest              5eb5528b2680        4 months ago        4.287 GB
```

Examining Ancestry

- Each Docker container has a parent image
- Each Docker image has a parent image, with the exception of base images
- Intermediate layer images have no repository name and no tag
  - These images are used to build up other images from which containers can be based
  - Intermediate images are not designed to be directly used by containers

```
docker@ubuntu:~$ docker run -t -i user_sam/thriftdev:0.9.1 /bin/bash
root@8d75247087b5:/# cd thrift
root@8d75247087b5:/thrift# vi README
root@8d75247087b5:/thrift# exit
exit
docker@ubuntu:~$ docker commit -m "README update" -a "Sam Thrift" 8d75247087b5 user_sam/thriftdev:0.9.1_rm
fc056a4320ce0c439e240b7d56f991cdbd76a4fbc732c180bea4201a7d283cf4
docker@ubuntu:~$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
user_sam/thriftdev      0.9.1_rm            fc056a4320ce        4 seconds ago       247.7 MB
user_sam/thriftdev      0.9.1               ca0310547ecb        11 minutes ago      247.7 MB
ubuntu                  14.04.2             2d24f826cb16        13 days ago         188.3 MB
ubuntu                  latest              2d24f826cb16        13 days ago         188.3 MB
ubuntu                  trusty              2d24f826cb16        13 days ago         188.3 MB
ubuntu                  trusty-20150218.1   2d24f826cb16        13 days ago         188.3 MB
ubuntu                  14.04               2d24f826cb16        13 days ago         188.3 MB
ubuntu                  precise             1f80e9ca2ac3        13 days ago         131.5 MB
ubuntu                  precise-20150212    1f80e9ca2ac3        13 days ago         131.5 MB
ubuntu                  12.04.5             1f80e9ca2ac3        13 days ago         131.5 MB
ubuntu                  12.04               1f80e9ca2ac3        13 days ago         131.5 MB
centos                  latest              dade6cb4530a        4 weeks ago         210.1 MB
cirros                  latest              8d202478b999        4 weeks ago         7.698 MB
fedora                  20                  6cece30db4f9        9 weeks ago         360.3 MB
fedora                  21                  834629358fe2        9 weeks ago         241.3 MB
fedora                  latest              834629358fe2        9 weeks ago         241.3 MB
bufferoverflow/thrift   latest              5eb5528b2680        4 months ago        4.287 GB
docker@ubuntu:~$ docker inspect -f='{{.Parent}}' user_sam/thriftdev:0.9.1_rm
ca0310547ecb9e71370479f94b5977a358054210d74c06bc0f869e32d51e96ae
docker@ubuntu:~$ docker inspect -f='{{.Parent}}' user_sam/thriftdev:0.9.1
2d24f826cb16146e2016ff349a8a33ed5830f3b938d45c0f82943f4ab8c097e7
docker@ubuntu:~$ docker inspect -f='{{.Parent}}' ubuntu:14.04
117ee323aaa9d1b136ea55e4421f4ce413dfc6c0cc6b2186dea6c88d93e1ad7c
docker@ubuntu:~$ sudo docker inspect -f='{{.Parent}}' 117ee323aaa9
1c8294cc516082dfbb731f062806b76b82679ce38864dd87635f08869c993e45
docker@ubuntu:~$ sudo docker inspect -f='{{.Parent}}' 1c8294cc5160
fa4fd76b09ce9b87bfdc96515f9a5dd5121c01cc996cf5379050d8e13d4a864b
docker@ubuntu:~$ sudo docker inspect -f='{{.Parent}}' fa4fd76b09ce
511136ea3c5a64f264b78b5433614aec563103b4d4702f3ba7d4d2698e22c158
docker@ubuntu:~$ sudo docker inspect -f='{{.Parent}}' 511136ea3c5a

docker@ubuntu:~$
```



Intermediate Images

- By default the “docker images” command does not display intermediate images
  - Intermediate images have no tag
  - Intermediate images act as building blocks for higher layer images
- $ docker images -a
  - To display intermediate images
- $ docker history displays an image’s parents
- The --no-trunc can be used to display full UUIDs with most docker commands

```
docker@ubuntu:~$ docker images -a
REPOSITORY              TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
user_sam/thriftdev      0.9.1_rm            fc056a4320ce        About an hour ago   247.7 MB
user_sam/thriftdev      0.9.1               ca0310547ecb        About an hour ago   247.7 MB
ubuntu                  trusty              2d24f826cb16        13 days ago         188.3 MB
ubuntu                  14.04               2d24f826cb16        13 days ago         188.3 MB
ubuntu                  14.04.2             2d24f826cb16        13 days ago         188.3 MB
ubuntu                  latest              2d24f826cb16        13 days ago         188.3 MB
ubuntu                  trusty-20150218.1   2d24f826cb16        13 days ago         188.3 MB
<none>                  <none>              117ee323aaa9        13 days ago         188.3 MB
<none>                  <none>              1c8294cc5160        13 days ago         188.3 MB
<none>                  <none>              fa4fd76b09ce        13 days ago         188.1 MB
ubuntu                  12.04.5             1f80e9ca2ac3        13 days ago         131.5 MB
ubuntu                  precise             1f80e9ca2ac3        13 days ago         131.5 MB
ubuntu                  precise-20150212    1f80e9ca2ac3        13 days ago         131.5 MB
ubuntu                  12.04               1f80e9ca2ac3        13 days ago         131.5 MB
<none>                  <none>              e829ddd93a57        13 days ago         131.5 MB
<none>                  <none>              7bee65a22cab        13 days ago         131.5 MB
<none>                  <none>              edeb8497f5fc        13 days ago         131.4 MB
centos                  latest              dade6cb4530a        4 weeks ago         210.1 MB
cirros                  latest              8d202478b999        4 weeks ago         7.698 MB
<none>                  <none>              cc6bc4ab5c0a        4 weeks ago         7.698 MB
```

```
docker@ubuntu:~$ docker history registry
IMAGE               CREATED             CREATED BY                                      SIZE
a68f5599e08a        5 days ago          /bin/sh -c #(nop) CMD [docker-registry]         0 B
024a18254446        5 days ago          /bin/sh -c #(nop) EXPOSE map[5000/tcp:{}]       0 B
e5a8e33139de        5 days ago          /bin/sh -c #(nop) ENV SETTINGS_FLAVOR=dev       0 B
a4f468439f7f        5 days ago          /bin/sh -c #(nop) ENV DOCKER_REGISTRY_CONFIG=   0 B
30bff528d188        5 days ago          /bin/sh -c patch  $(python -c 'import boto; i   50.8 kB
418dcd975ba2        5 days ago          /bin/sh -c pip install file:///docker-registr   24.82 MB
daa8104aee86        5 days ago          /bin/sh -c pip install /docker-registry/depen   11.55 MB
7eafad9a1f16        5 days ago          /bin/sh -c #(nop) COPY file:fea402efe168b79c0   73 B
f06997673ad7        5 days ago          /bin/sh -c #(nop) COPY dir:a754308ddb81433362   2.429 MB
777c3edddace        5 days ago          /bin/sh -c apt-get update     && apt-get inst   197 MB
2d24f826cb16        2 weeks ago         /bin/sh -c #(nop) CMD [/bin/bash]               0 B
117ee323aaa9        2 weeks ago         /bin/sh -c sed -i 's/^#\s*\(deb.*universe\)$/   1.895 kB
1c8294cc5160        2 weeks ago         /bin/sh -c echo '#!/bin/sh' > /usr/sbin/polic   194.5 kB
fa4fd76b09ce        2 weeks ago         /bin/sh -c #(nop) ADD file:0018ff77d038472f52   188.1 MB
511136ea3c5a        21 months ago                                                       0 B
docker@ubuntu:~$
```

```
docker@ubuntu:~$ docker images -a --no-trunc
REPOSITORY              TAG                 IMAGE ID                                                            CREATED             VIRTUAL SIZE
user_sam/thriftdev      0.9.1_rm            fc056a4320ce0c439e240b7d56f991cdbd76a4fbc732c180bea4201a7d283cf4    About an hour ago   247.7 MB
user_sam/thriftdev      0.9.1               ca0310547ecb9e71370479f94b5977a358054210d74c06bc0f869e32d51e96ae    About an hour ago   247.7 MB
ubuntu                  14.04               2d24f826cb16146e2016ff349a8a33ed5830f3b938d45c0f82943f4ab8c097e7    13 days ago         188.3 MB
ubuntu                  14.04.2             2d24f826cb16146e2016ff349a8a33ed5830f3b938d45c0f82943f4ab8c097e7    13 days ago         188.3 MB
ubuntu                  latest              2d24f826cb16146e2016ff349a8a33ed5830f3b938d45c0f82943f4ab8c097e7    13 days ago         188.3 MB
ubuntu                  trusty-20150218.1   2d24f826cb16146e2016ff349a8a33ed5830f3b938d45c0f82943f4ab8c097e7    13 days ago         188.3 MB
ubuntu                  trusty              2d24f826cb16146e2016ff349a8a33ed5830f3b938d45c0f82943f4ab8c097e7    13 days ago         188.3 MB
```

Removing Images

- Images can be deleted using the **docker rmi** command
  - The image must not be in use by any running or stopped containers
- Images can be deleted by either
  - repository:tag
  - ID
- Images are not removed until all of the tagged names referencing them are removed
- Intermediate parents (parent images with no tag) are removed by default when no dependencies remain upon them
- Forcing the delete of an image will cause future runs of dependent containers to re-pull the image or fail

```
docker@ubuntu:~$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
user_sam/thriftdev      0.9.1_rm            fc056a4320ce        46 hours ago        247.7 MB
user_sam/thriftdev      0.9.1               ca0310547ecb        47 hours ago        247.7 MB
ubuntu                  trusty              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                  14.04.2             2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                  latest              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                  14.04               2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                  trusty-20150218.1   2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                  precise             1f80e9ca2ac3        2 weeks ago         131.5 MB
ubuntu                  precise-20150212    1f80e9ca2ac3        2 weeks ago         131.5 MB
ubuntu                  12.04               1f80e9ca2ac3        2 weeks ago         131.5 MB
ubuntu                  12.04.5             1f80e9ca2ac3        2 weeks ago         131.5 MB
docker@ubuntu:~$ docker help rmi

Usage: docker rmi [OPTIONS] IMAGE [IMAGE...]

Remove one or more images

  -f, --force=false    Force removal of the image
  --help=false         Print usage
  --no-prune=false     Do not delete untagged parents
docker@ubuntu:~$ docker rmi user_sam/thriftdev:0.9.1_rm
Untagged: user_sam/thriftdev:0.9.1_rm
Deleted: fc056a4320ce0c439e240b7d56f991cdbd76a4fbc732c180bea4201a7d283cf4
docker@ubuntu:~$ docker rmi ca0310547ecb
Error response from daemon: Conflict, cannot delete ca0310547ecb because the container 8d75247087b5
is using it, use -f to force
FATA[0000] Error: failed to remove one or more images
docker@ubuntu:~$ docker stop 8d75247087b5
8d75247087b5
docker@ubuntu:~$ docker rmi ca0310547ecb
Error response from daemon: Conflict, cannot delete ca0310547ecb because the container 8d75247087b5
is using it, use -f to force
FATA[0000] Error: failed to remove one or more images
docker@ubuntu:~$ docker rm 8d75247087b5
8d75247087b5
docker@ubuntu:~$ docker rmi ca0310547ecb
Untagged: user_sam/thriftdev:0.9.1
Deleted: ca0310547ecb9e71370479f94b5977a358054210d74c06bc0f869e32d51e96ae
```

Summary

- Docker Registries provide a central location for storage and retrieval of Docker Repositories
- Docker Repositories are named collections of related Docker images
- Docker Images have:
  - UUID (just a UUID, not a hash of the layer data)
  - Repository [optional]
  - Tag [optional]
  - Filesystem layer
  - Other metadata
- Containers are created from an image
- Containers can be committed to create new images
- Docker images are read-only
  - All changes in a container take place in the container layer overlaying all underlying image layers
  - Image file system layers are overlaid to create a single file system view
- The Docker file system is implemented by a backend
  - Various block and file based solutions are available

Lab 3

- Creating Images

Day 2

4. Docker Hub and Registries
5. Dockerfiles
6. Docker in Practice

4: Docker Hub and Registries

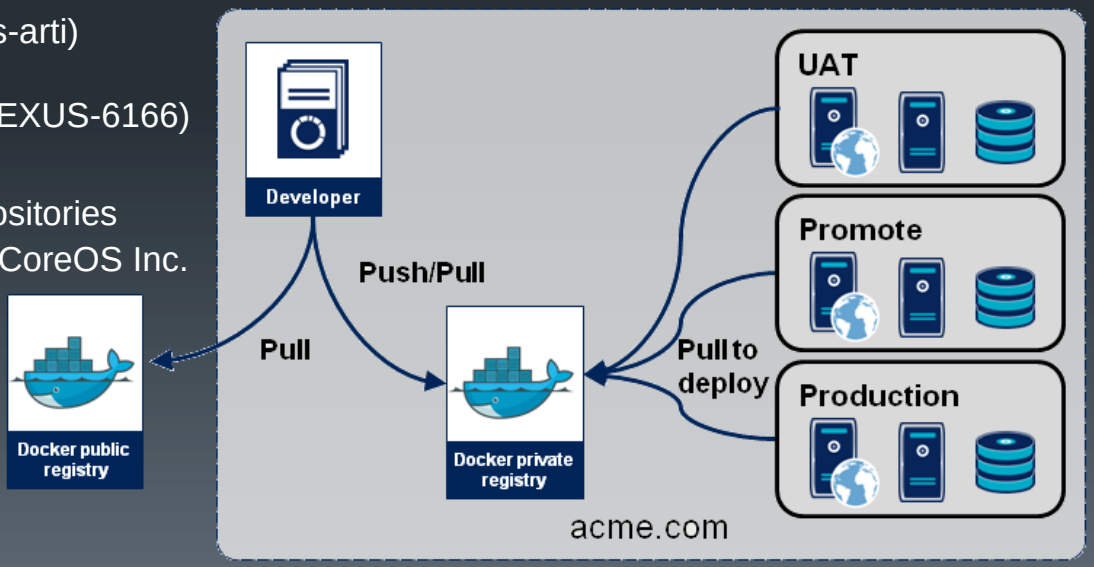
Objectives

- Explain the relationship between Registries and Repositories
- Describe the docker mechanism for locating images via registries
- Contrast Docker Hub with private registries
- Examine the registry login process
- List the steps used to run a private registry
- Explore the explicit and implicit docker registry commands
- Explain the docker tag command and the workings of repository and tag names

Registries, Repos and Tags

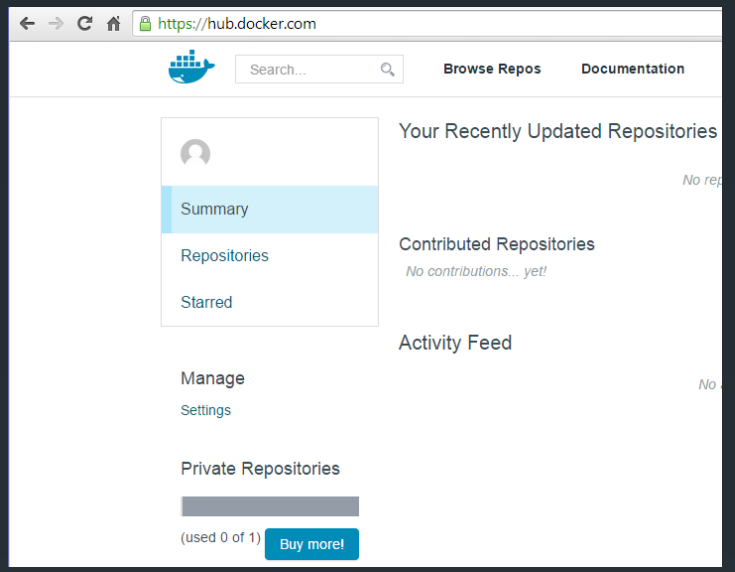
- Images house file system layers and metadata
- Repositories are named collections of images
- Tags are strings used to identify individual images within a repository
- Registries are services that allow you to store and retrieve images by repository:tag name using a REST (HTTP) API
  - Docker Hub is the central public registry
  - Companies can deploy their own registries
    - Using open source and commercial solutions
      - Docker Trusted Registry (formerly known as: Docker Hub Enterprise) from Docker Inc.
        - Released June 22, 2015
      - Docker Registry from Docker Inc. (https://github.com/docker/docker-registry)
      - Artifactory 3.4+ (http://www.jfrog.com/open-source/#os-arti)
      - Nexus 3 Milestone 5 (https://issues.sonatype.org/browse/NEXUS-6166)
    - Using hosted cloud solutions
      - Docker Hub also supports private repositories
      - Quay.io (https://quay.io/), acquired by CoreOS Inc.

The Docker Hub public registry does not do a particularly good job of verifying images and should be used with caution by the security minded



Cloud Registries

- Docker Registries house collections of repositories
- Registries allow users to upload and download images from repositories
- Registries can be private and internal
  - Organizations can run one or more of their own internal registries
- Registries can be public
  - Docker Hub is a public registry run by Docker Inc.
    - Docker Hub is integrated into the Docker ecosystem and is the default registry
  - Other organizations are free to create their own public registries
- Registries can be private and external
  - Companies can offer cloud based registry solutions commercially (or less likely, freely)
  - Docker Hub hosts private registries for paying clients
  - Quay hosts private registries for paying clients
    - Acquired by CoreOS Inc. 2014-08

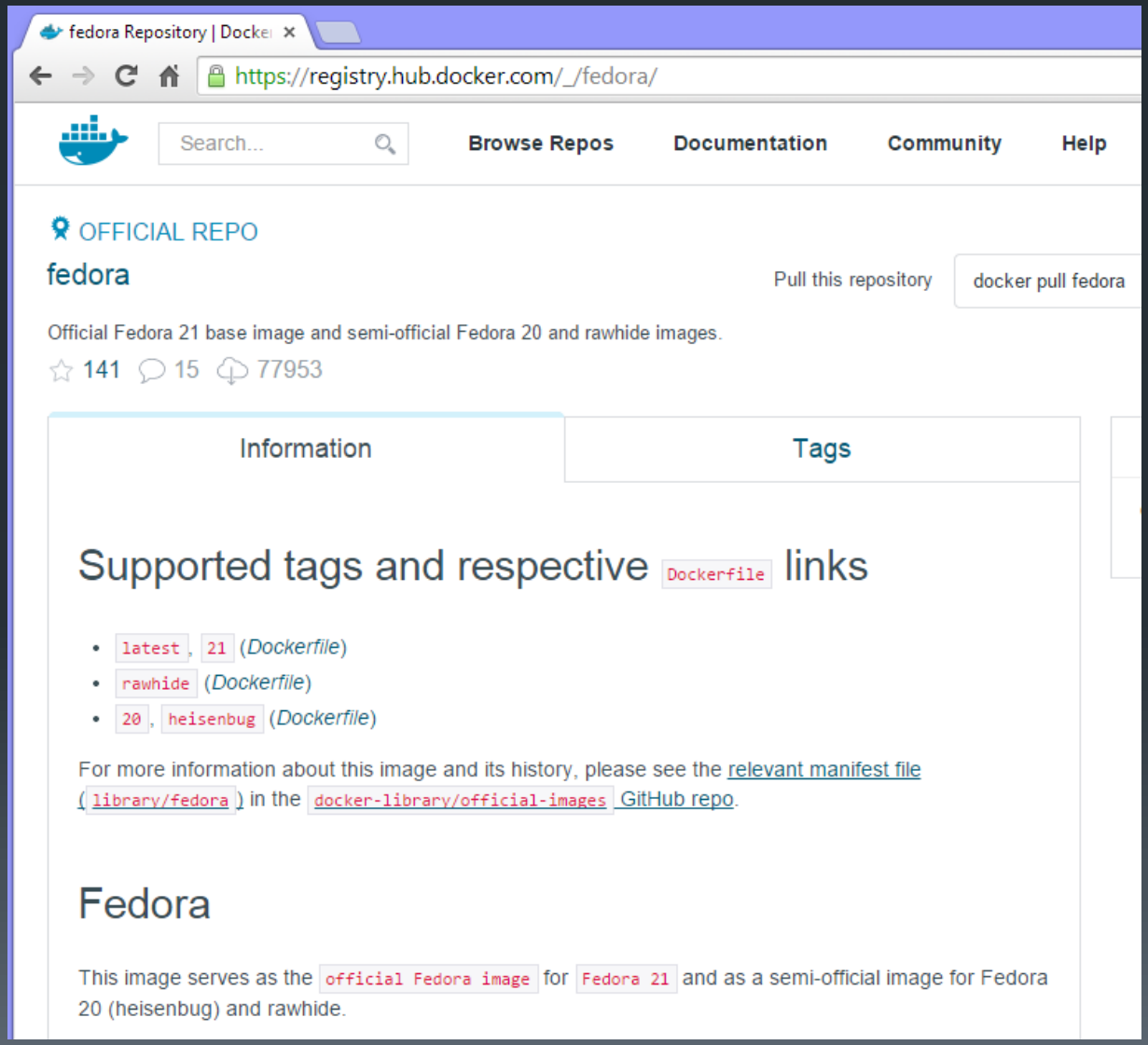


Docker Hub Current Pricing

Free	Micro	Small	Medium	Large
$0/mo	$7/mo	$12/mo	$22/mo	$50/mo
Unlimited public repositories	**Unlimited** public repositories	**Unlimited** public repositories	**Unlimited** public repositories	**Unlimited** public repositories
1 private repositories	**5** private repositories	**10** private repositories	**20** private repositories	**50** private repositories
Sign up! (or login)	Sign up! (or login)	Sign up! (or login)	Sign up! (or login)	Sign up! (or login)

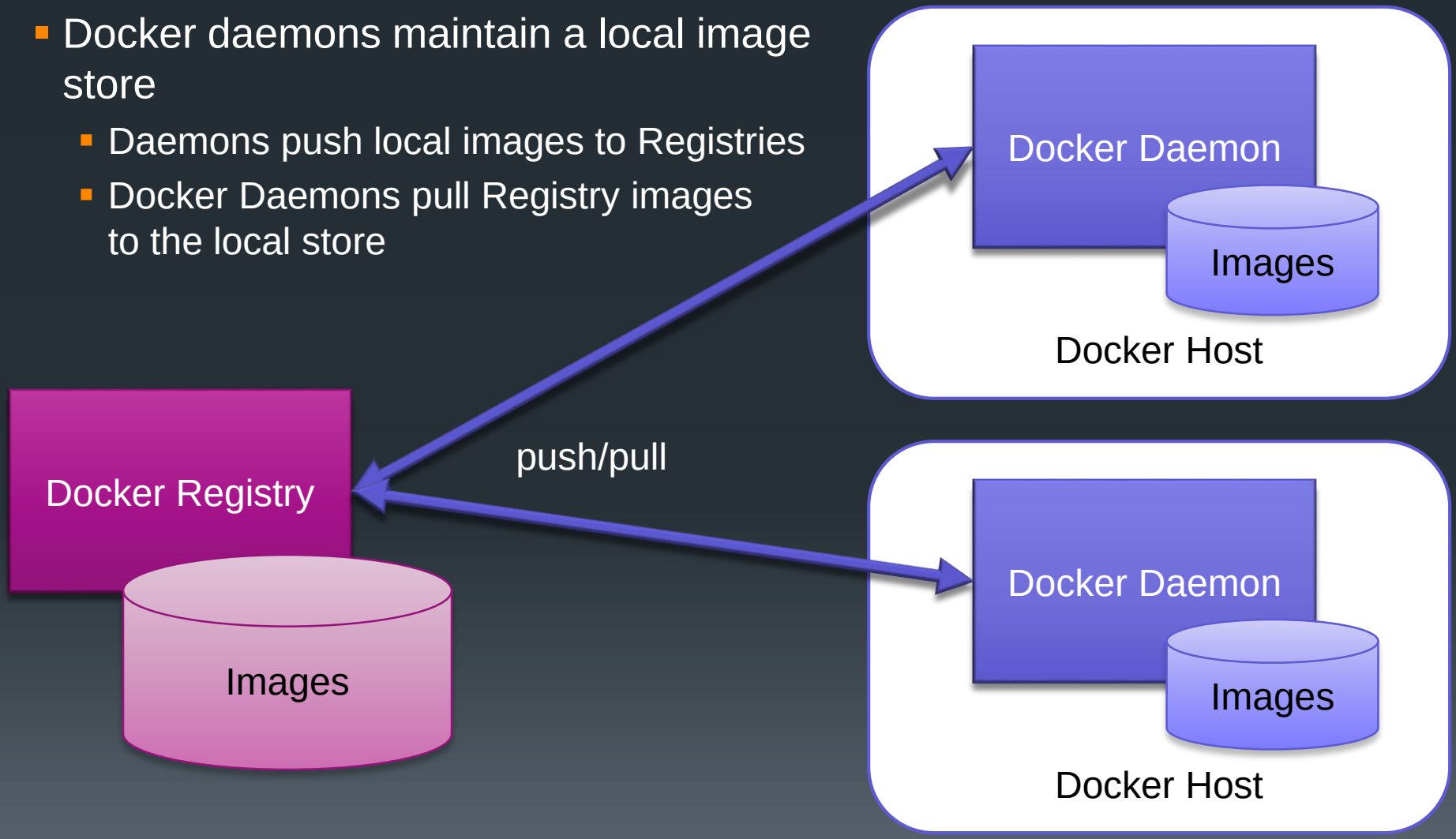
Web Search

- Searching for repositories at the command line may suffice for casual lookups
  - However only basic repository information is returned
- The web interface to docker hub provides detailed information on each repository
  - Including individual tags



push and pull

- Docker daemons maintain a local image store
  - Daemons push local images to Registries
  - Docker Daemons pull Registry images to the local store



Registry Commands

- Docker Registry Commands
  - docker login
    - to login to a registry
  - docker logout
    - To logout of a registry
  - docker search
    - searches a registry for an image
  - docker pull
    - pulls an image from registry to the local machine (also pulls its dependencies)
  - docker push
    - pushes an image to the registry from local machine
- When running a container whose image cannot be found locally, the Docker Engine will try to pull the image from Docker Hub automatically before running

```
docker@ubuntu:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
centos              latest              dade6cb4530a        3 weeks ago         210.1 MB
cirros              latest              8d202478b999        3 weeks ago         7.698 MB
docker@ubuntu:~$ docker search thrift
NAME                            DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
evarga/thrift                   This is a Docker image for Apache Thrift. ...   4                    [OK]
thrift                          Thrift is a framework for generating clien...   3         [OK]
zezuladp/thrift                                                                 0                    [OK]
masgari/thrift                                                                  0                    [OK]
tehcmc/elasticsearch-thrift                                                     0                    [OK]
lins05/elasticsearch-thrift                                                     0                    [OK]
inthecloud247/kdocker-thrift                                                    0                    [OK]
imiell/thrift                                                                   0
cjlarose/thrift                                                                 0
bowery/thrift                                                                   0
mgaut72/coink-thriftserver                                                      0
zezuladp/corenlp-thrift                                                         0                    [OK]
bufferoverflow/thrift           Apache Thrift - *make cross*                    0                    [OK]
docker@ubuntu:~$ docker pull bufferoverflow/thrift
Pulling repository bufferoverflow/thrift
5eb5528b2680: Download complete
511136ea3c5a: Download complete
97fd97495e49: Download complete
2dcbbf65536c: Download complete
6a459d727ebb: Download complete
8f321fc43180: Download complete
03db2b23cf03: Download complete
9cbaf023786c: Download complete
50e7b9a06c7d: Download complete
b3e7763abc23: Download complete
dc0f840983f5: Download complete
eb7e99f9667b: Download complete
6399e5518efe: Download complete
2de42e5ac940: Download complete
5efec72f6d22: Download complete
c35efe5fe594: Download complete
807c7b12fffb: Download complete
c328ff699bc7: Download complete
a3959cbd00b2: Download complete
7e5c48811f37: Download complete
dec7815b184e: Download complete
75bdecba3b21: Download complete
b480f1dd38f7: Download complete
c720ecfc437e: Download complete
Status: Downloaded newer image for bufferoverflow/thrift:latest
docker@ubuntu:~$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
centos                  latest              dade6cb4530a        3 weeks ago         210.1 MB
cirros                  latest              8d202478b999        3 weeks ago         7.698 MB
bufferoverflow/thrift   latest              5eb5528b2680        4 months ago        4.287 GB
docker@ubuntu:~$
```

Accessing Docker Hub

- Docker Hub can be searched and pulled anonymously
- The docker command line tool provides a login command
  - Enables push access to a Docker Hub account from the CLI
  - Stores credentials in:
    - ~/.dockercfg < docker 1.8
    - ~/.docker/config.json >= docker 1.8
  - User, password and email can be supplied on the command line
    - -e, --email="bob@example.com"
    - -p, --password="secret"
    - -u, --username="Bob Smith"
- Private registries can be accessed from the command line also
  - $ docker login localhost:8080

```
docker@ubuntu:~$ docker login
Username: bobsmith
Password:
Email: bob@example.com
Login Succeeded
docker@ubuntu:~$
```

.dockercfg

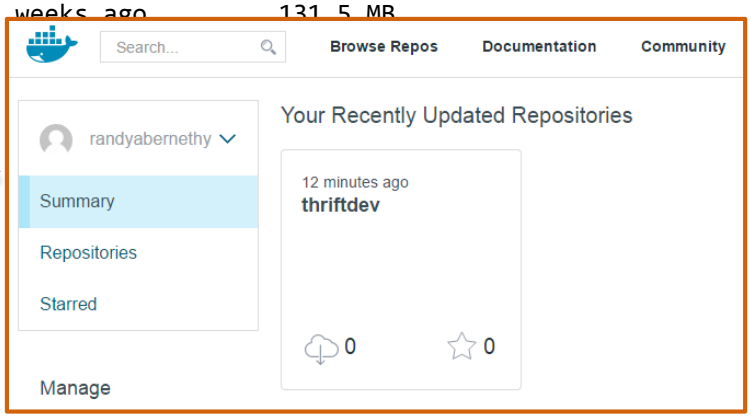
- .dockercfg entries provide a registry URL and an auth token
- The auth token is a base64 encoded string
  - base64(<username>:<password>)
  - This token is usable as long as the account and password on the target host are not changed
  - This representation is not secure and must be protected through file permissions (600)
- The “docker login” command simply populates this file
  - This can be automated with a script and/or predefined values can be copied

```
{
  "https://index.docker.io/v1/": {
    "auth": "assdfljhdsadjhhrjer=",
    "email": "bob@gmail.com"
  },
  "https://index.example.com": {
    "auth": "woeirdjfsudhfvjksds=",
    "email": "bob.smith@example.com"
  }
}
```

```
docker@ubuntu:~$ grep auth .dockercfg | sed 's/.*"auth": "\(.*\)",\?/\1/'
XJfdXNlX3lvdXJlX2RvZ3NfbmFtZV9hc19hX3Bhc3N3b3JkCg=
docker@ubuntu:~$ grep auth .dockercfg | sed 's/.*"auth": "\(.*\)",\?/\1/' | base64 --decode
bobsmith:never_use_your_dogs_name_as_a_password
docker@ubuntu:~$
```

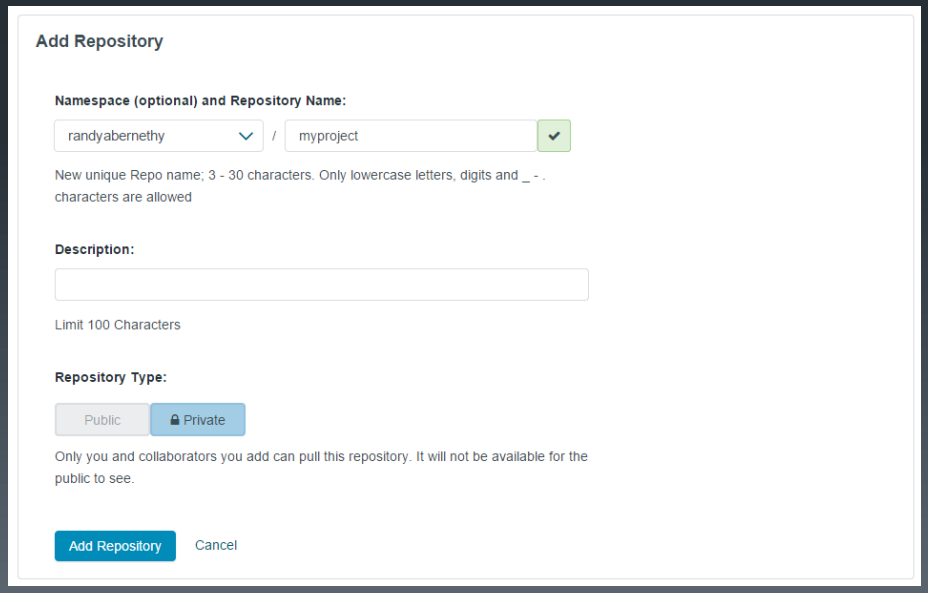
Pushing Repositories

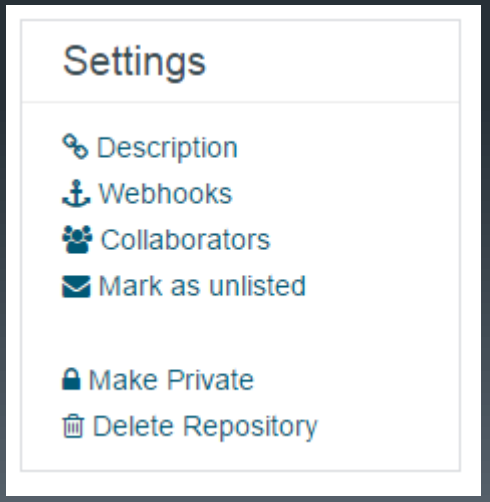
```
docker@ubuntu:~$ docker help push

Usage: docker push [OPTIONS] NAME[:TAG]

Push an image or a repository to the registry

  --help=false       Print usage

docker@ubuntu:~$ docker push thriftdev
FATA[0000] You cannot push a "root" repository. Please rename your repository to <user>/<repo> (ex: bobsmith/thriftdev)

docker@ubuntu:~$ docker commit ff808d882430 bobsmith/thriftdev
85905f19abdd41913aa16b9353b606e5dbbd79287a136cae5484df2ceac4dab0

docker@ubuntu:~$ docker images
REPOSITORY                TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
bobsmith/thriftdev        latest              85905f19abdd        7 minutes ago       226 MB
thriftdev                 latest              a69676e13708        9 minutes ago       226 MB
ubuntu                    trusty              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                    trusty-20150218.1   2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                    14.04.2             2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                    latest              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                    14.04               2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                    precise-20150212    1f80e9ca2ac3        2 weeks ago         131.5 MB
ubuntu                    12.04               1f80e9ca2ac3        2
ubuntu                    12.04.5             1f80e9ca2ac3        2
ubuntu                    precise             1f80e9ca2ac3        2

docker@ubuntu:~$ docker push bobsmith/thriftdev
The push refers to a repository [bobsmith/thriftdev] (len: 1)
Sending image list
Pushing repository bobsmith/thriftdev (1 tags)
511136ea3c5a: Image already pushed, skipping
fa4fd76b09ce: Image already pushed, skipping
1c8294cc5160: Image already pushed, skipping
117ee323aaa9: Image already pushed, skipping
2d24f826cb16: Image already pushed, skipping
85905f19abdd: Image successfully pushed
Pushing tag for rev [85905f19abdd] on {https://cdn-registry-1.docker.io/v1/repositories/bobsmith/thriftdev/tags/latest}
```



Docker Hub – Private Repos

- Private Docker Hub repositories must be added via the Add Repository link on the Docker Hub account page
- Once the private repository is created, you can push and pull images to and from it using Docker as always
  - You must to be signed in and have access to work with a private repository
- It is not possible to browse or search private repositories on the public registry
- Private repositories do not get cached
  - Public repositories do
- It is possible to give access to a private repository to those whom you designate (i.e., collaborators) from its Settings page
- You can also switch repository status (public to private, or vice-versa)

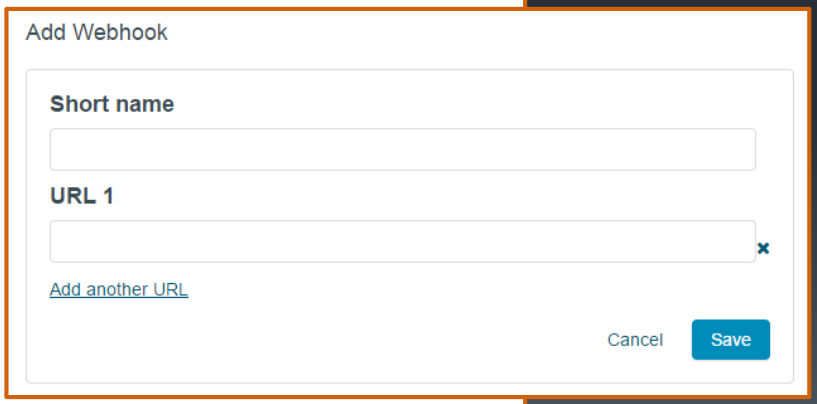




Webhooks

- Webhooks allow you to trigger actions following a successful push to a Docker Hub repository
- Webhooks are configured on the Repository Settings page
- A Webhook makes an HTTP POST requests with a JSON payload describing the push
  - Docker Hub server IP Addresses: 162.242.195.64 - 162.242.195.127
  - You can restrict your service to accept requests from this IP range
- Multiple URLs may be supplied
  - Called a web hook chain
  - Each service called must make a callback POST to the Docker Hub with a JSON body
    - The body must include a state with a value of success, failure or error
    - If the state isn't success, the webhook chain will be interrupted and not further URLs will be called

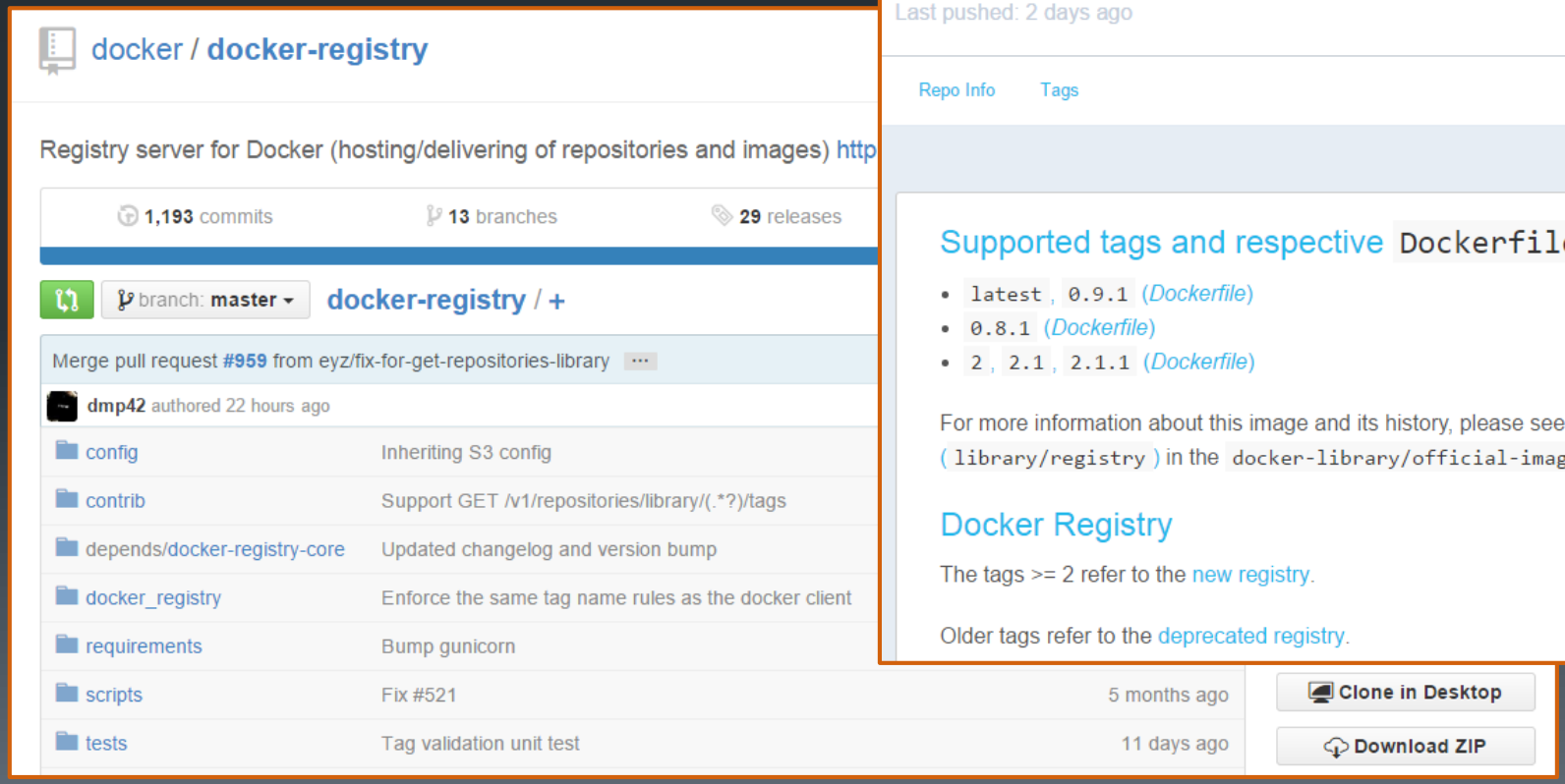
```
{
  "callback_url": "https://registry.hub.docker.com/u/svendowideit/busybox/hook/2141bc0cdec4hebec411i4c1g40242eg110020/",
  "push_data": {
    "images": [
        "27d47432a69bca5f2700e4dff7de0388ed65f9d3fb1ec645e2bc24c223dc1cc3",
        "51a9c7c1f8bb2fa19bcd09789a34e63f35abb80044bc10196e304f6634cc582c",
        ...
    ],
    "pushed_at": 1.417566822e+09,
    "pusher": "svendowideit"
  },
  "repository": {
    "comment_count": 0,
    "date_created": 1.417566665e+09,
    "description": "",
    "full_description": "webhook triggered from a 'docker push'",
    "is_official": false,
    "is_private": false,
    "is_trusted": false,
    "name": "busybox",
    "namespace": "svendowideit",
    "owner": "svendowideit",
    "repo_name": "svendowideit/busybox",
    "repo_url": "https://registry.hub.docker.com/u/svendowideit/busybox/",
    "star_count": 0,
    "status": "Active"
}
```



Private Registries

- Docker, Inc. has open-sourced the Docker registry code
  - This makes it easy to run a private registry
  - The registry offers an API but does not currently have a UI
- Docker Registry for Docker <1.6
  - Written in Python, serial image download
  - https://github.com/docker/docker-registry
- Docker Registry for Docker >=1.6
  - Written in Go, parallel image download
  - https://docs.docker.com/registry/
- The registry can be run from a prebuilt Docker image

Docker Registry 2.0 (released with Docker 1.6) adds support for parallel image download



Running Networked Services in a Container

- The Docker Hub curated repository “registry” supports execution of a registry daemon on any Docker platform
- $ docker run -p 5000:5000 -d registry
  - The example runs the registry repository image tagged “latest” using version 1 of the registry api
    - Be careful because using latest will actually get you 0.9.1, the old v1.0 Python version
  - The –p switch maps a host port to a container port ([hport]:[cport])
  - The registry service runs on port 5000 within the container, but the container registry port can be mapped to any host port (e.g. 80) you like
  - If the port is not mapped the registry will only be accessible from within the container
- V0.9.1 Startup bug
  - There is a known race condition in this version of the registry code which will crash the service on startup occasionally
    - ./registry._setup_database.lock can be referenced before-creation
    - Add the “-e GUNICORN_OPTS=["--preload"]” switch at the end of the command line (temporary work around)
      - Optionally you can just rerun the docker run command until the registry starts successfully
  - This is not a problem with “registry:2” !

```
docker@ubuntu:~$ docker run -p 5000:5000 -d registry
692c9188b1c96a1f97891628d0f348a8415b73cdde302da4b0c1f8404854428e
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
NAMES
692c9188b1c9        registry:latest     "docker-registry"   11 seconds ago      Up 10 seconds       0.0.0.0:5000->5000/tcp
happy_elion
docker@ubuntu:~$ curl http://localhost:5000
"\"docker-registry server\""
docker@ubuntu:~$
```

Setting Container Environment Variables

- You can use environment variables to customize many aspects of the Registry execution
  - The docker run –e switch sets environment variables in the container
  - Configuration files can also be used
- Sample configurations include support for:
  - Local file system repository storage
  - AWS S3
  - Ceph-S3
  - Azureblob
  - Swift, Glance and Glance-Swift
  - Others…
- Further Registry Documentation
  - https://docs.docker.com/registry/
  - v1 code and info:
    - https://github.com/docker/docker-registry/blob/master/README.md
    - https://github.com/docker/docker-registry/blob/master/ADVANCED.md
  - v2 code and info:
    - https://github.com/docker/distribution

You can use:
docker run --env-file <filename>
To load a set of environment variables from a file

```
docker run \
          -e SETTINGS_FLAVOR=s3 \
          -e AWS_BUCKET=acme-docker \
          -e STORAGE_PATH=/registry \
          -e AWS_KEY=AKIAHSHB43HS3J92MXZ \
          -e AWS_SECRET=xdDowwlK7TJajV1Y7EoOZrmuPEJlHYcNP2k4j49T \
          -e SEARCH_BACKEND=sqlalchemy \
          -p 5000:5000 \
          registry
```

docker tag

- The tag command allows you to create a new repository name and/or tag for an existing image
  - Think of this as a hard link (e.g. `$ ln source newlink`)
  - Tag is useful as it allows you to format repository names and tags to suit various registry systems

```
docker@ubuntu:~$ docker images ubuntu
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
ubuntu              14.04.2             2d24f826cb16        2 weeks ago         188.3 MB
ubuntu              trusty-20150218.1   2d24f826cb16        2 weeks ago         188.3 MB
ubuntu              14.04               2d24f826cb16        2 weeks ago         188.3 MB
ubuntu              latest              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu              trusty              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu              12.04               1f80e9ca2ac3        2 weeks ago         131.5 MB
ubuntu              12.04.5             1f80e9ca2ac3        2 weeks ago         131.5 MB
ubuntu              precise             1f80e9ca2ac3        2 weeks ago         131.5 MB
ubuntu              precise-20150212    1f80e9ca2ac3        2 weeks ago         131.5 MB
docker@ubuntu:~$ docker tag ubuntu:12.04.5 bobsmith/ubuntu:prod_2015-02-15
docker@ubuntu:~$ docker images ubuntu
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
ubuntu              14.04               2d24f826cb16        2 weeks ago         188.3 MB
ubuntu              latest              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu              trusty-20150218.1   2d24f826cb16        2 weeks ago         188.3 MB
ubuntu              14.04.2             2d24f826cb16        2 weeks ago         188.3 MB
ubuntu              trusty              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu              12.04.5             1f80e9ca2ac3        2 weeks ago         131.5 MB
ubuntu              precise             1f80e9ca2ac3        2 weeks ago         131.5 MB
ubuntu              precise-20150212    1f80e9ca2ac3        2 weeks ago         131.5 MB
ubuntu              12.04               1f80e9ca2ac3        2 weeks ago         131.5 MB
docker@ubuntu:~$ docker images bobsmith/ubuntu
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
bobsmith/ubuntu     prod_2015-02-15     1f80e9ca2ac3        2 weeks ago         131.5 MB
docker@ubuntu:~$
```

Pushing to a private Registry

- This example creates a new container and then commits an image from it
- To push the image to a registry the registry URL must be the first part of the repository name
  - Or a user name to imply pushing to the default Docker Hub
- Because repository names and tags are simply aliases, a single image ID may have many names/tags

```
docker@ubuntu:~$ docker run -t -i ubuntu:14.04 /bin/bash
root@335a181a7be6:/#
   ### Do some configuring
root@335a181a7be6:/# exit
exit
docker@ubuntu:~$ docker commit 335a181a7be6 bobsmith/thriftdev
84e5bc36d1640aef7cd5be9dba32297ab78d3685312859def2145059a0b0cf32
docker@ubuntu:~$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
bobsmith/thriftdev      latest              84e5bc36d164        10 seconds ago      226 MB
ubuntu                  latest              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                  trusty              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                  trusty-20150218.1   2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                  14.04               2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                  14.04.2             2d24f826cb16        2 weeks ago         188.3 MB
docker@ubuntu:~$ docker tag bobsmith/thriftdev localhost:5000/thriftdev
docker@ubuntu:~$ docker images
REPOSITORY                 TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
bobsmith/thriftdev         latest              84e5bc36d164        2 minutes ago       226 MB
localhost:5000/thriftdev   latest              84e5bc36d164        2 minutes ago       226 MB
ubuntu                     14.04               2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                     latest              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                     trusty              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                     14.04.2             2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                     trusty-20150218.1   2d24f826cb16        2 weeks ago         188.3 MB
docker@ubuntu:~$ docker push localhost:5000/thriftdev
The push refers to a repository [localhost:5000/thriftdev] (len: 1)
Sending image list
Pushing repository localhost:5000/thriftdev (1 tags)
511136ea3c5a: Image successfully pushed
fa4fd76b09ce: Image successfully pushed
1c8294cc5160: Image successfully pushed
117ee323aaa9: Image successfully pushed
2d24f826cb16: Image successfully pushed
84e5bc36d164: Image successfully pushed
Pushing tag for rev [84e5bc36d164] on {http://localhost:5000/v1/repositories/thriftdev/tags/latest}
docker@ubuntu:~$
```

Pulling Images

- Repository names embed the information used to identify an image's registry
- For Docker Hub
  - uid/name
- For non Docker Hub:
  - url/name
- This allows Docker to automatically recover missing images from the appropriate registry
  - Images can be pulled or run by registry name and optional tag

```
docker@ubuntu:~$ docker images
REPOSITORY                TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
localhost:5000/thriftdev  latest              84e5bc36d164        24 minutes ago      226 MB
ubuntu                    trusty              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                    14.04               2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                    latest              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                    trusty-20150218.1   2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                    14.04.2             2d24f826cb16        2 weeks ago         188.3 MB
docker@ubuntu:~$ docker rmi localhost:5000/thriftdev
Untagged: localhost:5000/thriftdev:latest
Deleted: 84e5bc36d1640aef7cd5be9dba32297ab78d3685312859def2145059a0b0cf32
docker@ubuntu:~$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
ubuntu                  trusty              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                  trusty-20150218.1   2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                  14.04               2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                  latest              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                  14.04.2             2d24f826cb16        2 weeks ago         188.3 MB
docker@ubuntu:~$ docker run -t -i localhost:5000/thriftdev:latest /bin/bash
Unable to find image 'localhost:5000/thriftdev:latest' locally
Pulling repository localhost:5000/thriftdev
84e5bc36d164: Download complete
511136ea3c5a: Download complete
fa4fd76b09ce: Download complete
1c8294cc5160: Download complete
117ee323aaa9: Download complete
2d24f826cb16: Download complete
Status: Downloaded newer image for localhost:5000/thriftdev:latest
root@a89e744e7d1d:/# exit
exit
docker@ubuntu:~$ docker images
REPOSITORY                TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
localhost:5000/thriftdev  latest              84e5bc36d164        28 minutes ago      226 MB
ubuntu                    trusty-20150218.1   2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                    trusty              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                    14.04               2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                    14.04.2             2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                    latest              2d24f826cb16        2 weeks ago         188.3 MB
```

Archiving Repositories

- docker save
  - Produces a tarred repository to the standard output stream
    - -o to write to a file instead of STDOUT
  - Contains all parent layers, and all tags + versions, or specified repo:tag, for each argument provided
- docker load
  - Loads a tarred repository from the standard input stream
    - -i to read from a tar archive file
  - Restores both images and tags

```
docker@ubuntu:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
datav               latest              8a9046c1e4cd        43 hours ago        188.3 MB
registry            latest              24dd746e9b9f        45 hours ago        413.9 MB
cirros              latest              d8de71c04044        2 days ago          7.698 MB
centos              latest              fd44297e2ddb        2 days ago          215.7 MB
ubuntu              latest              b7cf8f0d9e82        2 days ago          188.3 MB
docker@ubuntu:~$ docker save datav
FATA[0000] Cowardly refusing to save to a terminal. Use the -o flag or redirect.
docker@ubuntu:~$ docker save -o 'datav.repo'  datav
docker@ubuntu:~$ ls -l
total 192844
-rw-rw-r-- 1 docker docker 197421568 Apr 24 09:45 datav.repo
drwxr-xr-x 2 docker docker      4096 Mar  1 06:57 Desktop
drwxr-xr-x 2 docker docker      4096 Mar  1 06:57 Documents
drwxr-xr-x 2 docker docker      4096 Mar  1 06:57 Downloads
-rw-r--r-- 1 docker docker      8980 Mar  1 06:47 examples.desktop
drwxr-xr-x 2 docker docker      4096 Mar  1 06:57 Music
drwxr-xr-x 2 docker docker      4096 Mar  1 06:57 Pictures
drwxr-xr-x 2 docker docker      4096 Mar  1 06:57 Public
drwxr-xr-x 2 docker docker      4096 Mar  1 06:57 Templates
drwxr-xr-x 2 docker docker      4096 Mar  1 06:57 Videos
drwxrwxr-x 2 docker docker      4096 Apr 21 15:24 websvr
docker@ubuntu:~$ docker rmi datav
Untagged: datav:latest
Deleted: 8a9046c1e4cdd2e47876f6a0f7727ba27c9ee2372bf5bbebe46ece26fe45ca50
docker@ubuntu:~$ docker load -i datav.repo
docker@ubuntu:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
datav               latest              8a9046c1e4cd        43 hours ago        188.3 MB
registry            latest              24dd746e9b9f        45 hours ago        413.9 MB
cirros              latest              d8de71c04044        2 days ago          7.698 MB
centos              latest              fd44297e2ddb        2 days ago          215.7 MB
ubuntu              latest              b7cf8f0d9e82        2 days ago          188.3 MB
docker@ubuntu:~$
```

Archiving Containers

- Docker containers can be exported
  - docker export saves an entire container file system (with all underlying layers) to a tar archive
  - “docker export” applies to containers and is similar to “docker save” which applies to images
  - Unlike save, export flattens the output into a single fileystem
- docker import
  - Creates an empty image and imports the contents of the specified tarball into it

```
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
c2af6e61e924        cirros:latest       "/sbin/init"        4 hours ago         Up 4 hours                              sick_einstein
docker@ubuntu:~$ docker export -o se.tar sick_einstein
docker@ubuntu:~$ ls -l
total 7928
drwxr-xr-x 2 docker docker    4096 Mar  1 06:57 Desktop
drwxr-xr-x 2 docker docker    4096 Mar  1 06:57 Documents
drwxr-xr-x 2 docker docker    4096 Mar  1 06:57 Downloads
-rw-r--r-- 1 docker docker    8980 Mar  1 06:47 examples.desktop
drwxr-xr-x 2 docker docker    4096 Mar  1 06:57 Music
drwxr-xr-x 2 docker docker    4096 Mar  1 06:57 Pictures
drwxr-xr-x 2 docker docker    4096 Mar  1 06:57 Public
-rw-rw-r-- 1 docker docker 8068096 Apr 24 12:30 se.tar
drwxr-xr-x 2 docker docker    4096 Mar  1 06:57 Templates
drwxr-xr-x 2 docker docker    4096 Mar  1 06:57 Videos
drwxrwxr-x 2 docker docker    4096 Apr 21 15:24 websvr
docker@ubuntu:~$ cat se.tar | docker import - newbase:first_cut
2e6275575fe7d30ba5b747e9739f85ab813c38499b0452aab52e9318d4fb721c
docker@ubuntu:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
newbase             first_cut           2e6275575fe7        4 seconds ago       7.698 MB
datav               latest              8a9046c1e4cd        46 hours ago        188.3 MB
registry            latest              24dd746e9b9f        2 days ago          413.9 MB
cirros              latest              d8de71c04044        2 days ago          7.698 MB
centos              latest              fd44297e2ddb        2 days ago          215.7 MB
ubuntu              latest              b7cf8f0d9e82        2 days ago          188.3 MB
docker@ubuntu:~$ docker history newbase:first_cut
IMAGE               CREATED             CREATED BY          SIZE
2e6275575fe7        31 seconds ago                          7.698 MB
docker@ubuntu:~$ docker run -it newbase:first_cut /bin/sh
/ #
```

Summary

- Registries provide centralized lookup and storage facilities for repositories and their images
- Docker registry lookups are based on x/y formatted repository names
  - '/' being a reserved character
  - Docker Hub lookups are based on x being a user id where private registry lookups are based on x being a url
- The Docker Python Registry code can be executed directly from images found on Docker Hub
  - Source is also available on github
- Networked services can be run in containers using the -p switch to map host ports to container ports
- Docker supplies several commands to support registry operations
  - docker login
  - docker logout
  - docker search
  - docker pull
  - docker push
  - docker tag (creates appropriate names for image registry operations)
  - docker run (implicitly pulls missing images)

Lab 4

- Working with Images and Registries

5: Dockerfiles

Objectives

- Explain the purpose of Dockerfiles
- List the main Dockerfile instructions
- Understand the docker build process
- Examine important Dockerfile concepts
  - The Build Cache
  - Build Triggers

Dockerfiles and Context

- Docker can build images automatically by reading the instructions from a Dockerfile
  - A Dockerfile is a text document that contains all the commands you would normally execute manually in order to build a Docker image
- By calling docker build from your terminal, you can have Docker build your image step by step, executing the instructions successively
  - # docker build /some/path
- The path supplied to docker build is the context of the build
  - Also known as the source repository
  - The build is run by the Docker daemon, not by the CLI, so the whole context must be transferred to the daemon
  - The Docker CLI reports "Sending build context to Docker daemon" when the context is sent to the daemon
  - In most cases it is best to put each Dockerfile in an empty directory, adding only the files needed for building that Dockerfile

Dockerfiles

- docker commit
  - A practical way to create images interactively
  - An impractical way to create identical or incrementally improved images
- docker build
  - The docker build command creates images automatically from a Dockerfile
- Dockerfile
  - A text document containing all the commands you would normally execute manually in order to build a Docker image with docker commit
  - To build an image you can place a file called “Dockerfile” at the root of your repository and call “docker build” with the path of your source repository
    - $ docker build .
  - Each command in a Dockerfile creates a new image layer

```
docker@ubuntu:~$ mkdir thriftdev
docker@ubuntu:~$ cd thriftdev/
docker@ubuntu:~/thriftdev$ vi Dockerfile
docker@ubuntu:~/thriftdev$ cat Dockerfile
# Version: 1.0.0
FROM ubuntu:14.04
MAINTAINER Bob Smith "bob@example.com"
RUN apt-get update
RUN apt-get install -y git
RUN echo 'thrift dev image v1.0.0' > /README.md
EXPOSE 9418
docker@ubuntu:~/thriftdev$ docker build .
Sending build context to Docker daemon 2.048 kB
Step 0 : FROM ubuntu:14.04
 ---> 2d24f826cb16
Step 1 : MAINTAINER Bob Smith "bob@example.com"
 ---> Running in 4500320e431e
 ---> 3cb1d6967c66
Removing intermediate container 4500320e431e
Step 2 : RUN apt-get update
 ---> Running in 99e4fa6da78e
...
---> 1c09982ab4ce
Removing intermediate container 99e4fa6da78e
Step 3 : RUN apt-get install -y git
 ---> Running in d95d07597718
...
 ---> 3c0ab41341eb
Removing intermediate container d95d07597718
Step 4 : RUN echo 'thrift dev image v1.0.0' > /README.md
 ---> Running in 4555e0ff156a
 ---> 9fa98b0dc393
Removing intermediate container 4555e0ff156a
Step 5 : EXPOSE 9418
 ---> Running in 9ecb025bb167
 ---> 99fdcb5de9a4
Removing intermediate container 9ecb025bb167
Successfully built 99fdcb5de9a4
docker@ubuntu:~/thriftdev$ ls -l
total 4
-rw-rw-r-- 1 docker docker 185 Mar  8 18:24 Dockerfile
```

```
docker@ubuntu:~/thriftdev$ docker images -a
REPOSITORY                      TAG                 IMAGE ID
<none>                          <none>              9fa98b0dc393
<none>                          <none>              99fdcb5de9a4
<none>                          <none>              3c0ab41341eb
<none>                          <none>              1c09982ab4ce
<none>                          <none>              3cb1d6967c66
ubuntu                          14.04               2d24f826cb16        2 weeks ago         188.3 MB
```

Dockerfile Processing

- A Dockerfile contains a series of instructions paired with arguments
- Each instruction should be in upper-case and be followed by an argument
  - FROM ubuntu:14.04
- Instructions are processed from the top down
  - The ordering of instructions is typically important
- Each instruction commits a new image layer
- To execute an instruction Docker:
  - Runs a container from the previous image
  - Executes the instruction within the container
  - Commits a new layer from the modified container
  - Deletes the working container image
  - Repeats for the next instruction until all instructions have been applied
- If one of the instruction in the Dockerfile build crashes you will still have all of the images leading up to the failed step
  - This is useful for debugging

```
1  # A basic apache server. To use either add or bind mount content under /var/www
2  FROM ubuntu:12.04
3
4  MAINTAINER Kimbro Staken version: 0.1
5
6  RUN apt-get update && apt-get install -y apache2 && apt-get clean && rm -rf /var/lib/apt/lists/*
7
8  ENV APACHE_RUN_USER www-data
9  ENV APACHE_RUN_GROUP www-data
10 ENV APACHE_LOG_DIR /var/log/apache2
11
12 EXPOSE 80
13
14 CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]
```

Source:
https://github.com/kstaken/dockerfile-examples/blob/master/apache/Dockerfile
Several great examples here

Dockerfile construction

- FROM
  - Should always be the first instruction
  - Specifies a base image that the Dockerfile will operate on
- MAINTAINER
  - Tells Docker the author's name and email
- RUN
  - Executes commands on the current image
  - RUN instructions execute in a shell using the command wrapper /bin/sh –c
  - If you wish to execute without a shell (for example, to avoid shell string munging), you can specify the instruction in exec format:
    - RUN ["apt-get", " install", "-y", "nginx"]
    - An array specifies the command to be executed and each parameter to pass to the command
- EXPOSE
  - Tells Docker that the application in this container will use this specific port in the container
  - You must map the port to the host using the docker run command to make it externally available
  - You can specify multiple EXPOSE instructions
- CMD
  - Defines an executable for a container
  - CMD has several forms:
    - CMD ["executable","param1","param2"] (exec form)
    - CMD command param1 param2 (shell form, performs shell processing on the provided command line [e.g. $HOME substitution, etc.])
  - There should only be one CMD instruction in a Dockerfile
  - The CMD will not be executed if an executable is speficied on the docker run command line
    - e.g.: $ docker run –t –i ubuntu **/bin/bash**

```
# Basic install of couchdb
#
# This will move the couchdb http server to port 8101 so adjust the port for your needs.
#
# Currently installs couchdb 1.3.1

FROM ubuntu
MAINTAINER Kimbro Staken

RUN echo "deb http://us.archive.ubuntu.com/ubuntu/ precise universe" >> /etc/apt/sources.list
RUN apt-get -y update
RUN apt-get install -y g++
RUN apt-get install -y erlang-dev erlang-manpages erlang-base-hipe erlang-eunit erlang-nox erlang-xmerl erlang-inets

RUN apt-get install -y libmozjs185-dev libicu-dev libcurl4-gnutls-dev libtool wget

RUN cd /tmp ; wget http://www.bizdirusa.com/mirrors/apache/couchdb/source/1.3.1/apache-couchdb-1.3.1.tar.gz

RUN cd /tmp && tar xvzf apache-couchdb-1.3.1.tar.gz
RUN apt-get install -y make
RUN cd /tmp/apache-couchdb-* ; ./configure && make install

RUN printf "[httpd]\nport = 8101\nbind_address = 0.0.0.0" > /usr/local/etc/couchdb/local.d/docker.ini

EXPOSE 8101

CMD ["/usr/local/bin/couchdb"]
```

Docker build options

- Building from a git repository
  - Rather than specifying a local directory, build can be given a git repo url
  - Clones the repository and uses the cloned repository as context
    - Uses the Dockerfile at the root of the repository
  - You can specify a Git repository using the git:// scheme
  - docker build -t="bobs/thrift:v1" git://github.com/bobs/thriftdev
- -f
  - Allows you to specify a particular file to use as the build input rather than ./Dockerfile
  - Added in Docker v1.5
- -t
  - The tag switch allows you to name repositories ([userid, url]/reponame) and add tags (:0.9.2)
- --no-cache=[true, false]
  - The Docker build cache stores copies of previously created images
  - While building Docker will look for an existing image in its cache that it can reuse
  - Use to turn off caching
  - Instructions are compared against all child images of the base image to see if one was built using the exact same instruction

```
docker@ubuntu:~$ docker build -t bobsmith/thriftdev:0.1.0 thriftdev
Sending build context to Docker daemon 2.048 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:14.04
 ---> 2d24f826cb16
Step 1 : MAINTAINER Bob Smith "bob@example.com"
 ---> Using cache
 ---> 3cb1d6967c66
Step 2 : RUN apt-get update
 ---> Using cache
 ---> 1c09982ab4ce
Step 3 : RUN apt-get install -y git
 ---> Using cache
 ---> 3c0ab41341eb
Step 4 : RUN echo 'thrift dev image v1.0.0' > /README.md
 ---> Using cache
 ---> 9fa98b0dc393
Step 5 : EXPOSE 80
 ---> Using cache
 ---> 99fdcb5de9a4
Successfully built 99fdcb5de9a4
```

Build Cache

- The Build Cache allows multiple Docker files to share the same image layers over a common base
- This optimizes build times and saves disk space
- Maximizing cache use requires careful layout of container commonalities
- There are also operational considerations
  - For example, all commands are run at build time
  - Thus an image's repository index may grow old if the image is not rebuilt
    - The example below uses an ENV instruction to force the follow on images to be rebuilt when the REFRESHED_AT date is updated

```
FROM ubuntu: 14.04
MAINTAINER Bob Smith "bob@example.com"
ENV REFRESHED_AT 2015-02-01
RUN apt-get -qq update
```

Image History

- The history command displays the parents of a given image and the instructions which created them
  - Each line displays:
    - The Image UUID
    - Time created
    - Executed statement creating the image
    - The size added to the overall file system by the command

```
docker@ubuntu:~$ docker history bobsmith/thriftdev:0.1.0
IMAGE               CREATED             CREATED BY                                      SIZE
99fdcb5de9a4        4 hours ago         /bin/sh -c #(nop) EXPOSE 80/tcp                 0 B
9fa98b0dc393        4 hours ago         /bin/sh -c echo 'thrift dev image v1.0.0' > /   24 B
3c0ab41341eb        4 hours ago         /bin/sh -c apt-get install -y git               37.69 MB
1c09982ab4ce        4 hours ago         /bin/sh -c apt-get update                       20.59 MB
3cb1d6967c66        4 hours ago         /bin/sh -c #(nop) MAINTAINER Bob Smith "bob@e   0 B
2d24f826cb16        2 weeks ago         /bin/sh -c #(nop) CMD [/bin/bash]               0 B
117ee323aaa9        2 weeks ago         /bin/sh -c sed -i 's/^#\s*\(deb.*universe\)$/   1.895 kB
1c8294cc5160        2 weeks ago         /bin/sh -c echo '#!/bin/sh' > /usr/sbin/polic   194.5 kB
fa4fd76b09ce        2 weeks ago         /bin/sh -c #(nop) ADD file:0018ff77d038472f52   188.1 MB
511136ea3c5a        21 months ago                                                       0 B
```

Port Assignment

- The -p flag manages container network ports forwarding
  - Containers may be configured in various ways
    - The container ports may be internal to the container (unavailable from the host)
    - Docker can randomly assign a host port (49000 – 49900) to container ports
    - You can specify a host port for each container port
    - Docker can map the same port used by the container on the host interface using -P (capital P w/ no params)
  - The host port can be prefixed with an interface
  - UDP ports can be exposed by adding the /udp suffix
- The port command displays port details (similar to the ps command)
- For example the docker “registry” image specifies the EXPOSE 5000 instruction
  - The session below demonstrates three possible port mapping styles

```
docker@ubuntu:~$ docker run -d registry
913120c8f3c3f40968079e70f4b758618ecf95a9d234337841ff09cf688a7494
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
913120c8f3c3        registry:latest     "docker-registry"   4 seconds ago       Up 3 seconds        5000/tcp
berserk_carson
docker@ubuntu:~$ docker stop 913120c8f3c3
913120c8f3c3
docker@ubuntu:~$ docker run -d -p 5000 --name="regtest" registry
55af8926ce3ea511b2357d4dc9e750018ff72a177457fab5991650f1dc73b777
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                     NAMES
55af8926ce3e        registry:latest     "docker-registry"   3 seconds ago       Up 2 seconds        0.0.0.0:49154->5000/tcp   regtest
docker@ubuntu:~$ docker stop regtest
regtest
docker@ubuntu:~$ docker rm regtest
regtest
docker@ubuntu:~$ docker run -d -p 127.0.0.1:8585:5000 --name="regtest" registry
9ad870687ebf53381da6ae51c1c4ae09cddb03a8a33d6d17c0ef817bdc4dc36e
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                       NAMES
9ad870687ebf        registry:latest     "docker-registry"   14 seconds ago      Up 13 seconds       127.0.0.1:8585->5000/tcp
regtest
docker@ubuntu:~$ docker port regtest
5000/tcp -> 127.0.0.1:8585
```

Metadata created by EXPOSE

Docker port mapping reference
http://docs.docker.com/userguide/dockerlinks/#network-port-mapping-refresher

Additional Dockerfile Instructions

- ENTRYPOINT
  - Runs the specified program
    - ENTRYPOINT ["/usr/ sbin/nginx"]
  - Unlike CMD, additional arguments supplied to a docker run of an image with an ENTRYPOINT are supplied directly to the ENTRYPOINT program
    - If CMD and ENTRYPOINT are present the CMD array is passed to the ENTRYPOINT program as parameters and overridden if command line args are supplied during docker run
    - This allows for natural "command style" execution of containers:
      - Assuming thrift is an image with an entry point that runs the thrift compiler:
        - $ docker run thrift --gen java appcore.thrift
- ENV
  - Sets environment variables
  - Environment variables can be referenced in many docker instructions using a $ prefix
  - Environment variables can also be set on the docker run command line using the –e switch
  - The following Dockerfile instructions can handle environment variables as arguments: ENV, ADD, COPY, WORKDIR, EXPOSE, VOLUME, USER

```
ENV APP_DIR /home/webapp/
WORKDIR $APP_DIR
```

- LABEL
  - Adds metadata to an image (e.g. LABEL version="1.0")
- USER
  - specifies a user:group that the image should be run as (by name or id)
  - Can specify a user ("bob") or a user and group ("bob:emp")
- WORKDIR
  - Sets the working directory for the container and the ENTRYPOINT/CMD

- ❖ CMD sets a command/arguments to run in the image if no arguments are passed to docker run
- ❖ ENTRYPOINT makes your image behave like a binary

```
LABEL version="1.0"
USER webapp
ENV APP_DIR /home/webapp/
WORKDIR /opt/webapp/db
RUN bundle install
WORKDIR $APP_DIR
ENTRYPOINT [ "rackup" ]
```

ADD & COPY

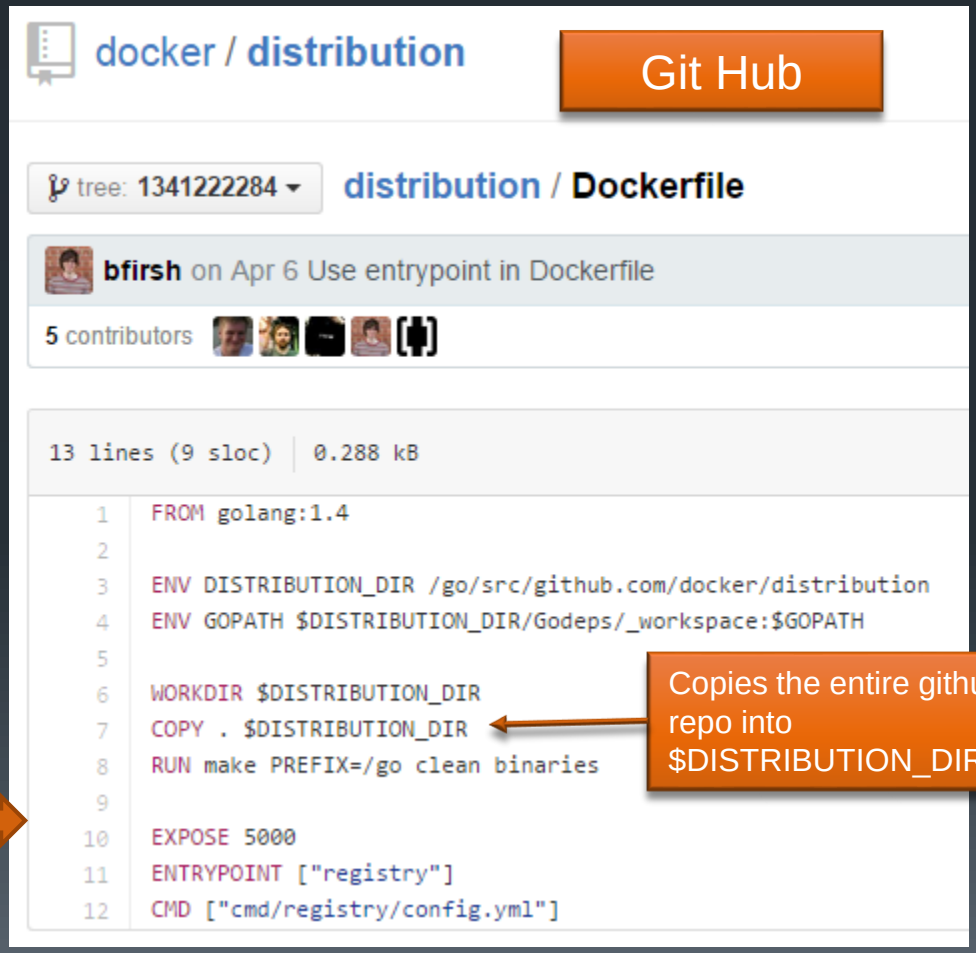
- ADD
  - Adds files and directories from the build environment into the image
  - Specifies a source and a destination for files:
    - ADD software.lic /opt/application/software.lic
    - Copies software.lic from the build directory to /opt/application/software.lic in the image
    - The source of the file can be a URL, filename, or directory
    - The source must be inside the build context
    - If the destination ends in a /, docker considers the source a directory
    - ADD will automatically unpack source archives
      - gzip
      - bzip2
      - xz
  - New files and directories will be created with a mode of 0755 and a UID and GID of 0
- COPY
  - Does is almost exactly like ADD but without URL and Archive support
- To speed the build and keep images lean you can exclude files and directories when COPYing or ADDing to the image by adding a .dockerignore file to the context directory

Docker registry:2.0 Dockerfile case study

Docker/Git Hub

- **Docker Hub** images are frequently sourced from Dockerfiles linked in from **github** or other URLs

Docker Hub



Copies the entire github repo into $DISTRIBUTION_DIR

VOLUME

- Volumes
  - Hold data
  - Can be shared/reused across containers
  - Store data in the host file system not the container layer
- A volume can be created in two ways:
  - `VOLUME /some/dir  #within a Dockerfile`
  - `$ docker run -v /some/dir ...`
- Volumes can decouple the life of the data from the life of the container(s)
  - You can docker rm some_container without affecting the volumes it uses
  - The Data Container Pattern involves creating containers that never run and only exist to provide volumes for other running containers
  - This allows running containers to be ephemeral and data containers to be persistent
- Volumes bypass the layered file system
  - This makes volumes as fast as the host filesystem
  - Changes to a volume will not be included in the image when you commit the container

```
docker@ubuntu:~$ docker run -it -v /some/dir ubuntu
root@efdae4210fb5:/# echo "hello volumes" > /some/dir/test.md
root@efdae4210fb5:/# ls -l /some/dir
total 4
-rw-r--r-- 1 root root 14 Apr 22 20:36 test.md
```

```
docker@ubuntu:~$ docker run -it --volumes-from=efdae4210fb5 ubuntu
root@7456cafa8740:/# cat /some/dir/test.md
hello volumes
```

More on volumes in the next chapter

```
# Dockerfile to create a data volume container for Oracle Linux
FROM oraclelinux:6.6
# Add starter data as needed
RUN mkdir -p /var/www/html
RUN echo "This is the content for file1.html" > /var/www/html/file1.html
RUN echo "This is the content for file2.html" > /var/www/html/file2.html
RUN echo "This is the content for index.html" > /var/www/html/index.html
# Export volume to share
VOLUME /var/www/html
```

ONBUILD

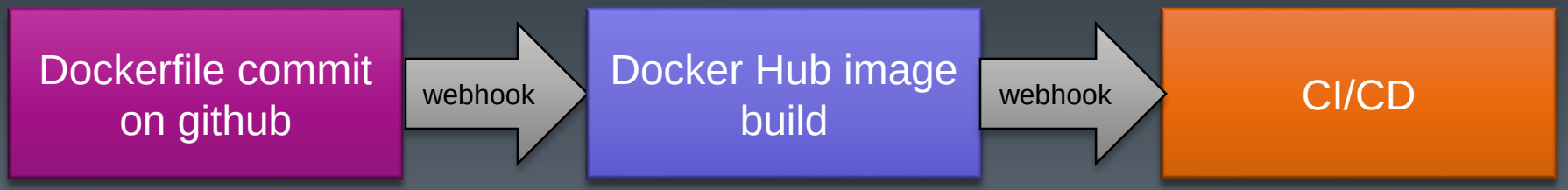
- ONBUILD adds triggers to images
- A trigger is executed when the image is used as the basis of another image
- Uses:
  - Creating an image that needs source code from a specific location that is not yet available
  - Execute a build script specific to the environment in which the image is built
- Triggers insert a new instruction in the build process, as if it were specified right after the FROM instruction
- The trigger can be any build instruction except FROM, MAINTAINER, and ONBUILD
- ONBUILD triggers are executed in the order specified in the parent image and are only inherited once
  - by children, not grandchildren

```
# bobs/apache
FROM ubuntu: 14.04
MAINTAINER Bob Smith "bob@example.com"
RUN apt-get update
RUN apt-get install -y apache2
ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2
ONBUILD ADD . /var/www/
EXPOSE 80
ENTRYPOINT ["/usr/sbin/apache2"]
CMD ["-D", "FOREGROUND"]
```

```
# bobs/webapp
#   The important parts of this image
#   are the static files in the build
#   dir.
FROM bobs/apache
### BUILD TRIGGER happens here
MAINTAINER Bob Smith "bob@example.com"
ENV APPLICATION_NAME webapp
ENV ENVIRONMENT development
```

Automated Builds

- Web Hooks can be used in response to new image push operations to run CI/CD tasks
- New containers can be automatically generated
- DVCS platforms like GitHub and BitBucket support WebHooks which can invoke a DockerHub build against a DVCS repo and its Dockerfile
- Docker Hub WebHooks can in turn invoke Jenkins or other CI platforms
- This requires your cloud DVCS to be linked to your Docker Hub account
- When creating a repository on Docker Hub you can select Repository or "Automated Build"
  - This provides a dialog allowing you to configure the target GitHub/BitBucket repo and Dockerfile to use for the build
- Once configured you can not push to an automated build, rather you push the sources and Dockerfile to the DVCS to trigger the build



Dockerfile/Image Tips

- Docker image authors may ask:
  - Is my image easy to use?
  - Is my image easy to base another image on?
  - Does my image behave in a performant manner?
- Use the MAINTAINER instruction to set the Author field of the image
  - Provides contact information for users
- Know the Differences Between CMD and ENTRYPOINT
  - CMD sets a command/arguments to run in the image if no arguments are passed to docker run
  - ENTRYPOINT makes your image behave like a binary
- Know the Difference Between Array and String Forms of CMD and ENTRYPOINT
  - Passing an array will result in the exact command being run
  - Passing a string will run the command in a shell (/bin/sh –c)
- Always exec in Wrapper Scripts
  - Images using a CMD/ENTRYPOINT script target will cause docker to send signals to the script, use exec in the script to run programs so that the program will receive the signals (otherwise only the script will)
- Always EXPOSE Important Ports
  - Exposed ports show up in ps listings, are visible in metadata and are automatically linked when using container linking
- Containers should be ephemeral
  - Stop and destroy the container then create a new one with an absolute minimum of set-up and configuration
- Use a .dockerignore file
  - For faster uploading and smaller images, use a .dockerignore to exclude files/directories from the build context
  - E.g. unless .git is needed you should add it to .dockerignore, saving many megabytes of upload time
- Avoid installing unnecessary packages
  - Reduces complexity, dependencies, file sizes, and build times
  - E.g. you don't need to include a text editor in a database image
- Run only one process per container
  - In almost all cases, you should only run a single process in a single container
  - Decoupling apps into multiple containers makes it easier to scale horizontally and reuse containers
  - If a service depends on another service use container linking
- Minimize the number of layers
  - Balance Dockerfile readability (and thus long-term maintainability) and layer minimization
  - Be strategic and cautious about the number of layers you use
- Sort multi-line arguments
  - Ease later changes by sorting multi-line arguments alphanumerically
  - Helps avoid duplication and make the list easier to update
  - Also makes Pull Requests easier to read and review
- Build cache
  - Use the build cache wisely

```
RUN \
    apt-get update && \
    apt-get install -y \
        bzr \
        git \
        mercurial
```

Summary

- Dockerfiles allow docker images to be scripted
  - This makes image construction highly reapeatable
- There are several key Dockerfile instructions
  - ADD, CMD, COPY, ENTRYPOINT, ENV, EXPOSE, FROM, LABEL, MAINTAINER, ONBUILD, RUN, USER, VOLUME, WORKDIR
- The docker build cache saves copies of each instruction's image, potentially speeding up future builds
- WebHooks can be used to automate docker builds from cloud based DVCS systems

Dockerfile Documentation:
https://docs.docker.com/reference/builder/

Lab 5

- Creating Dockerfiles

6: Docker in Practice

Objectives

- Describe the features of Docker networking
- Use docker and host based commands to work with containers and networks
- Describe the Docker container linking feature
- Examine the nature of shared volumes
- Contrast host and container based volume sharing
- Explore Docker security features
- List some Docker and Dockerfile best practices

Container Networking

- Docker containers expose ports which can be mapped to host interfaces so that container services are exposed on the host's external network
- Docker also supports internal networking
  - Docker assigns containers an IP address on an isolated host based virtual network (172.17.x.x, though docker will choose an alternate subnet if there is a conflict)
  - Docker defines a host interface as a gateway
    - Docker0, usually 172.17.42.1
- Docker 1.5+ also supports IPv6 addresses
  - To enable ipv6 run the Docker daemon with the --ipv6 flag
- A container gets a new IP address each time it is started

```
docker@ubuntu:~$ ip a show docker0
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 56:84:7a:fe:97:99 brd ff:ff:ff:ff:ff:ff
    inet 172.17.42.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::5484:7aff:fefe:9799/64 scope link
       valid_lft forever preferred_lft forever
docker@ubuntu:~$
```

Container interfaces


```
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
100e64bb676c        ubuntu:14.04        "/bin/bash"         About a minute ago   Up About a minute                       desperate_banach

7a676cb55ca9        ubuntu:14.04        "/bin/bash"         2 minutes ago        Up 2 minutes                            admiring_colden

docker@ubuntu:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    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: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:a8:1b:76 brd ff:ff:ff:ff:ff:ff
    inet 192.168.235.140/24 brd 192.168.235.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fea8:1b76/64 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 56:84:7a:fe:97:99 brd ff:ff:ff:ff:ff:ff
    inet 172.17.42.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::5484:7aff:fefe:9799/64 scope link
       valid_lft forever preferred_lft forever
45: vethd9d2b7c: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue m
    link/ether 8e:05:b2:f1:63:98 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::8c05:b2ff:fef1:6398/64 scope link
       valid_lft forever preferred_lft forever
47: veth4433108: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue m
    link/ether d6:92:6c:86:00:dc brd ff:ff:ff:ff:ff:ff
    inet6 fe80::d492:6cff:fe86:dc/64 scope link
       valid_lft forever preferred_lft forever
docker@ubuntu:~$
```

- Each container receives a veth (virtual Ethernet) connection to the Docker virtual network

```
docker@ubuntu:~$ docker run -t -i ubuntu /bin/bash
root@100e64bb676c:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
t
    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
46: eth0: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue sta
    link/ether 02:42:ac:11:00:18 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.24/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe11:18/64 scope link
       valid_lft forever preferred_lft forever
root@100e64bb676c:/#
```

```
docker@ubuntu:~$ docker run -t -i ubuntu /bin/bash
root@7a676cb55ca9:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    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
44: eth0: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:17 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.23/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe11:17/64 scope link
       valid_lft forever preferred_lft forever
root@7a676cb55ca9:/# ping -c 2 172.17.0.24
PING 172.17.0.24 (172.17.0.24) 56(84) bytes of data.
64 bytes from 172.17.0.24: icmp_seq=1 ttl=64 time=0.107 ms
64 bytes from 172.17.0.24: icmp_seq=2 ttl=64 time=0.062 ms

--- 172.17.0.24 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1012ms
rtt min/avg/max/mdev = 0.062/0.084/0.107/0.024 ms
root@7a676cb55ca9:/#
```

The Docker Host Interface

- Docker containers access the outside world through the Host interface
- The 172 network is not externally routable and must be translated using NAT/PAT
  - Firewall rules and NAT configuration allow Docker to route between containers and the host network
  - DNAT (destination based network address translation) is used to map host ports to containers

```
docker@ubuntu:~$ docker run -d -p 8585:5000 registry
c4a56fdeee31ab3616efdcdbc401ab880982b7fe45bdc1856c8072167a556d80
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                    NAMES
c4a56fdeee31        registry:latest     "docker-registry"   12 seconds ago      Up 11 seconds       0.0.0.0:8585->5000/tcp   insane_poinca
re
100e64bb676c        ubuntu:14.04        "/bin/bash"         14 minutes ago      Up 14 minutes                                desperate_ban
ach
7a676cb55ca9        ubuntu:14.04        "/bin/bash"         15 minutes ago      Up 15 minutes                                admiring_cold
en
docker@ubuntu:~$ sudo iptables -t nat -L -n
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DOCKER     all  --  0.0.0.0/0            0.0.0.0/0            ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
DOCKER     all  --  0.0.0.0/0           !127.0.0.0/8          ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  172.17.0.0/16        0.0.0.0/0
MASQUERADE  tcp  --  172.17.0.25          172.17.0.25          tcp dpt:5000

Chain DOCKER (2 references)
target     prot opt source               destination
DNAT       tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:8585 to:172.17.0.25:5000
docker@ubuntu:~$
```

Linking Containers

- Inter-Container Communications
  - Docker containers can use external services via SNAT
  - Docker containers can provide services externally using DNAT
  - Docker containers can also be linked directly using the Docker virtual network
- When containers are linked a source container can send information to a recipient container describing its service endpoint
- Container naming
  - Docker relies on the names of containers to share link information
- The docker run --link flag is used to create container links:
  - For example:
    - $ sudo docker run -d --name dbsvr train/postgres
    - $ sudo docker run -d -P --name web **--link dbsvr:db** train/webapp python app.py
      - "-P" (uppercase P) switch publishes the container EXPOSE port(s) directly on the host
  - The --link switch takes the form: --link <name or id>:alias
    - Where name is the name of the container to link to and alias is an alias for the link name used within the client container
- Inter-container connections do not require externally exposed ports
  - Note the db container was started without -P or -p

```
docker@ubuntu:~$ sudo docker inspect -f '{{ .NetworkSettings.IPAddress }}' c4a56fdeee31
172.17.0.25
docker@ubuntu:~$
```

Link /etc/hosts and ENV updates

- Host files
  - When two containers are linked Docker adds a host entry for the source container to the /etc/hosts file
    - $ sudo docker run -it --rm --link dbsvr:db training/webapp /bin/bash
    - root@aed84ee21bde:/opt/webapp# cat /etc/hosts
    - 172.17.0.7 aed84ee21bde #the web app itself
    - . . .
    - 172.17.0.5 db #the linked db container
- Environment Variables
  - When two containers are linked Docker adds environment variables in the client container also
    - $ sudo docker run -it --rm --link dbsvr:db training/webapp /bin/bash
    - root@aed84ee21bde:/opt/webapp# env
    - . . .
    - DB_NAME=/web2/db
    - DB_PORT=tcp://172.17.0.5:5432
    - DB_PORT_5432_TCP=tcp://172.17.0.5:5432
    - DB_PORT_5432_TCP_PROTO=tcp
    - DB_PORT_5432_TCP_PORT=5432
    - DB_PORT_5432_TCP_ADDR=172.17.0.5

Linking Details

- To enhance security, you can force Docker to only allow communication between containers if a link exists
  - To do this start the Docker daemon with the --icc = false flag
- Container linking is good for creating small clusters of co-deployed containers
  - You can't link between containers on separate Docker hosts
- If the target container of a link is restarted the IP address in the link client's /etc/hosts file will be updated with the new IP address (since Docker 1.3)
- You can use normal network discovery to connect containers running on different hosts
  - The docker run command allows DNS to be configured with the --dns and --dns-search flags
- Much like virtual machines can run within virtual machines, Docker can run within a Docker container
  - This allows you to realize complex isolation models

Volumes

- Volumes:
  - Hold data
  - Can be shared/reused across containers
  - Store data in the host file system not the container layer
- A volume can map to a preexisting host directory or a Docker created directory
  - Any part of the host file system can be mapped into a container as a volume
  - Any part of a container's file system can be configured as a volume
    - In this case the data is still stored directly on the host filesystem but docker creates a directory on the host to mount (typically in /var/lib/docker)
- Volumes can decouple the life of the data from the life of the container(s)
  - You can docker rm some_container without affecting the volumes it uses
  - The Data Container Pattern involves creating containers that never run and only exist to provide a volume for other running containers
  - This allows running containers to be ephemeral and data containers to be persistent
  - Docker never automatically deletes volumes when you remove a container, nor will it "garbage collect" volumes that are no longer referenced by a container
    - To delete a volume you must explicitly call docker rm -v against the last container with a reference to the volume
- Volumes bypass the layered file system
  - This makes volumes as fast as the host based filesystem
  - Changes to a volume will not be included when you update an image by committing the container
    - The volume is outside the container filesystem layer
- A volume can be created in two ways:
  - `VOLUME /some/dir  # Within a Dockerfile`
  - `$ docker run -v /some/dir ...  # At the command line`

Mounting Host Paths

- A host path volume can be automatically mounted within a docker container using the docker run -v switch
- Volumes default to read/write
  - To mount a volume read only use the :ro suffix
    - $ sudo docker run -d -P --name web -v /src/wapp:/opt/wapp:ro cisco/wapp python app.py
- The -v flag can also be used to mount a single file
  - $ sudo docker run --rm -it -v ~/.bash_history:/.bash_history ubuntu /bin/bash

```
docker@ubuntu:~$ mkdir data
docker@ubuntu:~$ cd data
docker@ubuntu:~/data$ vim README.md
docker@ubuntu:~/data$ cd ..
docker@ubuntu:~$ docker run –i -t -v ~/data:/webdata --name="web01" ubuntu:14.04 /bin/bash
root@2e5cf445a1b5:/# cat /webdata/README.md
shared Docker data

root@2e5cf445a1b5:/# exit
exit
docker@ubuntu:~$
```

Volume Architecture

- Volumes map container paths to host paths
  - When no host path is supplied Docker creates one
    - Docker managed volume
  - Volumes listed in container's config.json
- No current way to list volumes ☹
  - "docker volumes ls" proposed 10/2014
  - https://github.com/docker/docker/issues/8363
- Only host path volumes can be Read Only
  - For obvious reasons!
- Docker managed Volumes have a UUID
- Docker managed Volumes are marked as IsBindMount=false
  - Bind mount implies a host path is mounted
  - Allows docker to delete them when instructed to (docker rm -v)
  - Volumes are never automatically deleted

```
docker@ubuntu:~$ docker run -it -v /some/dir ubuntu
root@26a6ef088746:~# echo "hello volumes" > /some/dir/test.md
root@26a6ef088746:~# ls -l /some/dir
total 4
-rw-r--r-- 1 root root 14 Apr 22 21:25 test.md
```

```
docker@ubuntu:~$ docker run -it --volumes-from=26a6ef088746 ubuntu
root@7456cafa8740:/# cat /some/dir/test.md
hello volumes
```

```
root@ubuntu:~# docker inspect -f "{{.Volumes}}" 26a6ef0887460403
map[/some/dir:/var/lib/docker/vfs/dir/35e965ad6155928990328c2c2d1edf4951a3a2719146d9573a46d940abb61955]
```

```
root@ubuntu:~# ls -l /var/lib/docker/vfs/dir/35e965ad6155928990328c2c2d1edf4951a3a2719146d9573a46d940abb61955
-rw-r--r-- 1 root root 3 Apr 22 14:10 test.md
```

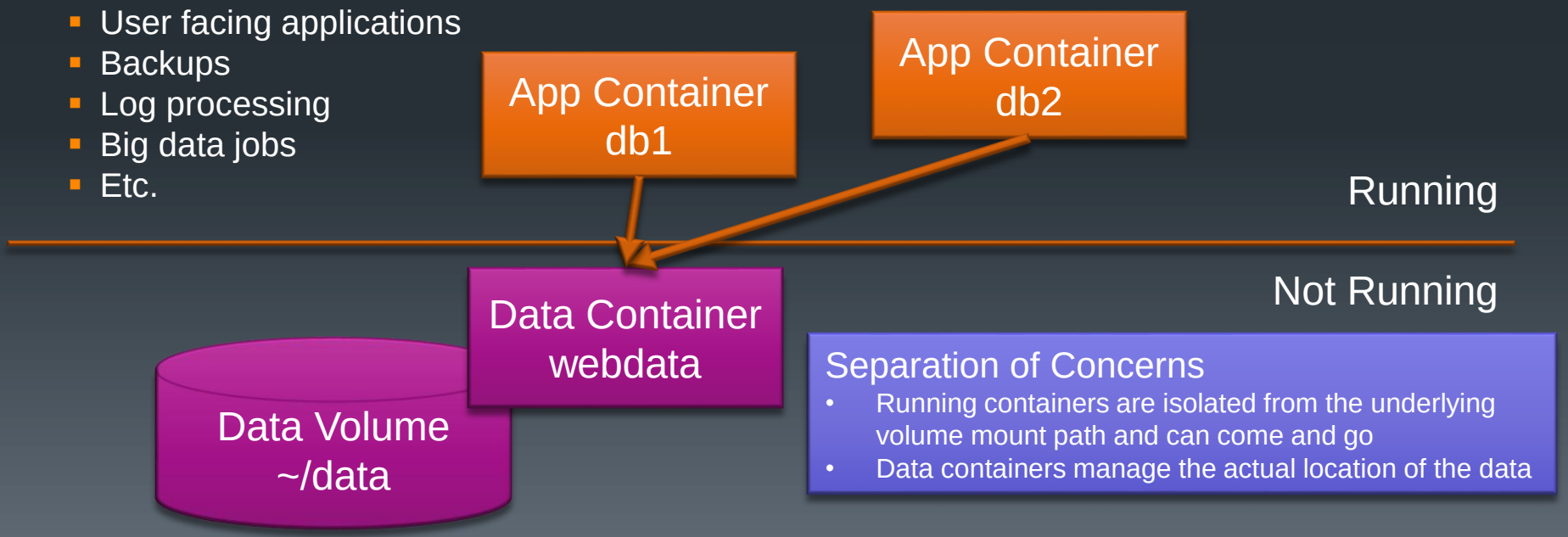
```
root@ubuntu:~# grep -r '35e965ad6155928990328c2c2d1edf4951a3a2719146d9573a46d940abb61955' /var/lib/docker/
/var/lib/docker/volumes/35e965ad6155928990328c2c2d1edf4951a3a2719146d9573a46d940abb61955/config.json:{"ID":"35e965ad6155928990328c2c2d1edf4951a3a2719146d9573a46d940abb61955","Path":"/var/lib/docker/vfs/dir/35e965ad6155928990328c2c2d1edf4951a3a2719146d9573a46d940abb61955","IsBindMount":false,"Writable":true}
/var/lib/docker/containers/26a6ef08874604035f3ce221fc3c7a004b7d8bd7c3a9ef254400be1753da3797/config.json:{"State":{"Running":true,"Paused":false,"Restarting":false,"OOMKilled":false,"Dead":false,"Pid":3928,"ExitCode":0,"Error":"","StartedAt":"2015-04-22T21:09:36.957255878Z","FinishedAt":"0001-01-01T00:00:00Z"},"ID":"26a6ef08874604035f3ce221fc3c7a004b7d8bd7c3a9ef254400be1753da3797","Created":"2015-04-22T21:09:36.832153988Z","Path":"/bin/bash","Args":[],"Config":{"Hostname":"26a6ef088746","Domainname":"","User":"","Memory":0,"MemorySwap":0,"CpuShares":0,"Cpuset":"","AttachStdin":true,"AttachStdout":true,"AttachStderr":true,"PortSpecs":null,"ExposedPorts":null,"Tty":true,"OpenStdin":true,"StdinOnce":true,"Env":null,"Cmd":["/bin/bash"],"Image":"ubuntu","Volumes":{"/some/dir":{}},"WorkingDir":"","Entrypoint":null,"NetworkDisabled":false,"MacAddress":"","OnBuild":null,"Labels":{}},"Image":"b7cf8f0d9e82c9d96bd7afd22c600bfdb86b8d66c50d29164e5ad2fb02f7187b","NetworkSettings":{"IPAddress":"172.17.0.6","IPPrefixLen":16,"MacAddress":"02:42:ac:11:00:06","LinkLocalIPv6Address":"fe80::42:acff:fe11:6","LinkLocalIPv6PrefixLen":64,"GlobalIPv6Address":"","GlobalIPv6PrefixLen":0,"Gateway":"172.17.42.1","IPv6Gateway":"","Bridge":"docker0","PortMapping":null,"Ports":{}},"ResolvConfPath":"/var/lib/docker/containers/26a6ef08874604035f3ce221fc3c7a004b7d8bd7c3a9ef254400be1753da3797/resolv.conf","HostnamePath":"/var/lib/docker/containers/26a6ef08874604035f3ce221fc3c7a004b7d8bd7c3a9ef254400be1753da3797/hostname","HostsPath":"/var/lib/docker/containers/26a6ef08874604035f3ce221fc3c7a004b7d8bd7c3a9ef254400be1753da3797/hosts","LogPath":"/var/lib/docker/containers/26a6ef08874604035f3ce221fc3c7a004b7d8bd7c3a9ef254400be1753da3797/26a6ef08874604035f3ce221fc3c7a004b7d8bd7c3a9ef254400be1753da3797-json.log","Name":"/cocky_banach","Driver":"aufs","ExecDriver":"native-0.2","MountLabel":"","ProcessLabel":"","AppArmorProfile":"","RestartCount":0,"UpdateDns":false,"Volumes":{"/some/dir":"/var/lib/docker/vfs/dir/35e965ad6155928990328c2c2d1edf4951a3a2719146d9573a46d940abb61955"},"VolumesRW":{"/some/dir":true},"AppliedVolumesFrom":null}
```

Container Volumes

- Containers can also expose Docker managed host paths to other containers
- This allows containers to depend directly on other containers, not the underlying volume
  - If you have some persistent data that you want to share between containers, or want to use from non-persistent containers, it's best to create a named Data Volume Container, and then to mount the data from it
- Data volume containers do not typically run applications
  - $ sudo docker create -v /dbdata --name dbdata bobs/postgres
- --volumes-from
  - Used to mount a volume from one container in another container
  - $ sudo docker run -d --volumes-from dbdata --name db1 bobs/postgres
  - $ sudo docker run -d --volumes-from dbdata --name db2 bobs/postgres
  - You can use multiple --volumes-from parameters to bring together multiple data volumes from multiple containers
- You can also mount a volume from a container, which in turn was mounted from another container
  - $ sudo docker run -d --name db3 --volumes-from db1 bobs/postgres
- If you remove containers that mount volumes, including the initial dbdata container, or the subsequent containers db1 and db2, the volumes will not be deleted
  - To delete the volume from disk, you must explicitly call docker rm -v against the last container with a reference to the volume
  - This allows you to upgrade, or effectively migrate data volumes between containers

Using Container Volumes

- Data Containers are a design pattern wherein data volumes are owned by data containers
- Data containers run no process
  - $ docker create -v ~/data:/webdata --name="webdata“ ubuntu:14.04
  - $ docker run -d --volumes-from webdata --name db1 bobs/postgres
- Data Containers exist to own volumes
- Related containers then use “--volumes-from” to mount volumes held by the data container
- A range of operational containers can perform independent tasks in this way
  - User facing applications
  - Backups
  - Log processing
  - Big data jobs
  - Etc.



Sending signals and waiting for Containers

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

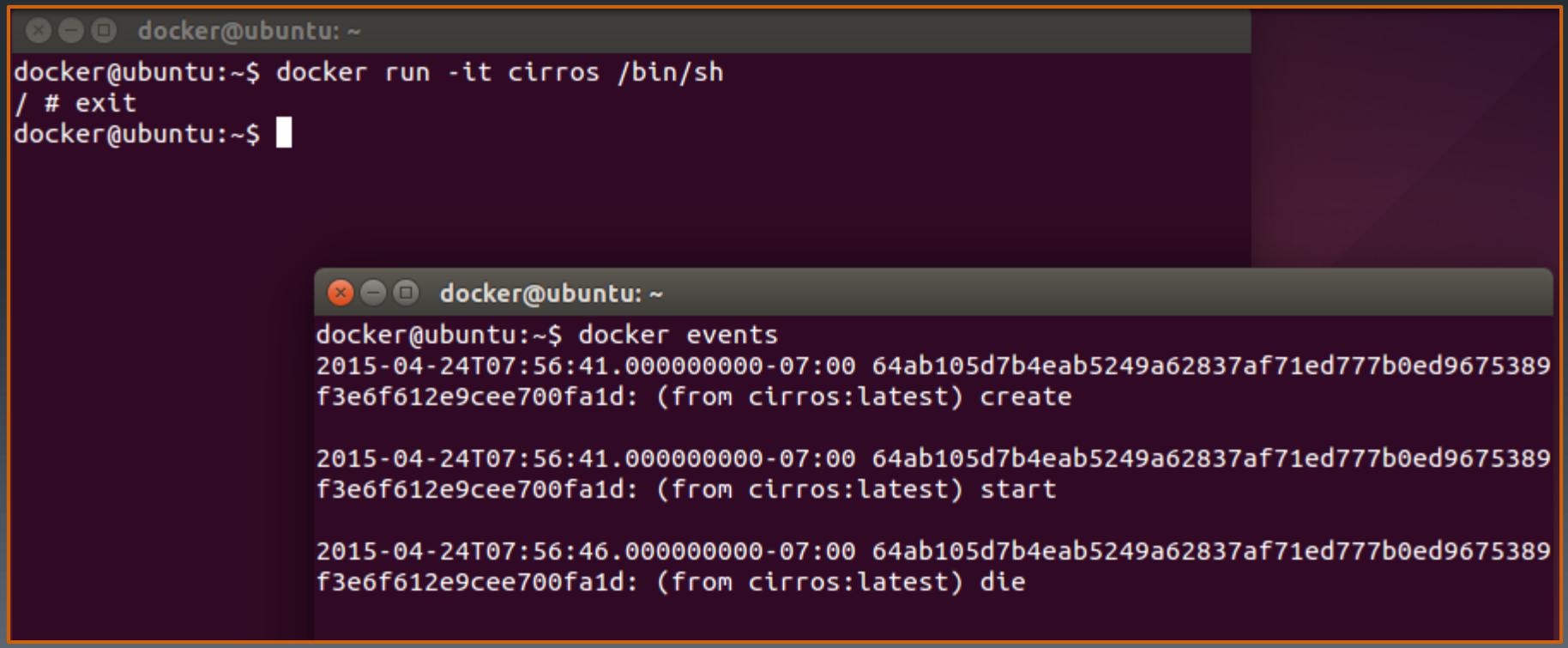
Sending SIGUSR1 to a daemon will dump all goroutines stacks without exiting

```
docker@ubuntu: ~
docker@ubuntu:~$ docker run -d centos:7 /usr/bin/tail -f /dev/null
1b2b83c12a83460bda23e476bedbe829945670c0753978faa97c7631e5756596
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                CREATED             STATUS              PORTS               NAMES
1b2b83c12a83        centos:7            "/usr/bin/tail -f /d   4 seconds ago       Up 3 seconds                            hopeful_carson
docker@ubuntu:~$ docker stop 1b
1b
docker@ubuntu:~$ docker run -d --name="longproc" centos:7 /usr/bin/tail -f /dev/null
6829b9797624c06cc1a24841b8f41f754c37ad1fac1e30a3bc8f99fec02855c7
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                CREATED             STATUS              PORTS               NAMES
6829b9797624        centos:7            "/usr/bin/tail -f /d   5 seconds ago       Up 4 seconds                            longproc
docker@ubuntu:~$ docker exec -it longproc /bin/bash
[root@6829b9797624 /]# ps
  PID TTY          TIME CMD
    7 ?        00:00:00 bash
   22 ?        00:00:00 ps
[root@6829b9797624 /]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 23:09 ?        00:00:00 /usr/bin/tail -f /dev/null
root         7     0  0 23:09 ?        00:00:00 /bin/bash
root        23     7  0 23:09 ?        00:00:00 ps -ef
[root@6829b9797624 /]# exit
exit
docker@ubuntu:~$ docker kill --signal="HUP" longproc
longproc
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                CREATED             STATUS              PORTS               NAMES
6829b9797624        centos:7            "/usr/bin/tail -f /d   5 minutes ago       Up 5 minutes                            longproc
docker@ubuntu:~$ docker kill --signal="TERM" longproc
longproc
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                CREATED             STATUS              PORTS               NAMES
6829b9797624        centos:7            "/usr/bin/tail -f /d   5 minutes ago       Up 5 minutes                            longproc
docker@ubuntu:~$ docker kill --signal="KILL" longproc
longproc
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
docker@ubuntu:~$
```

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

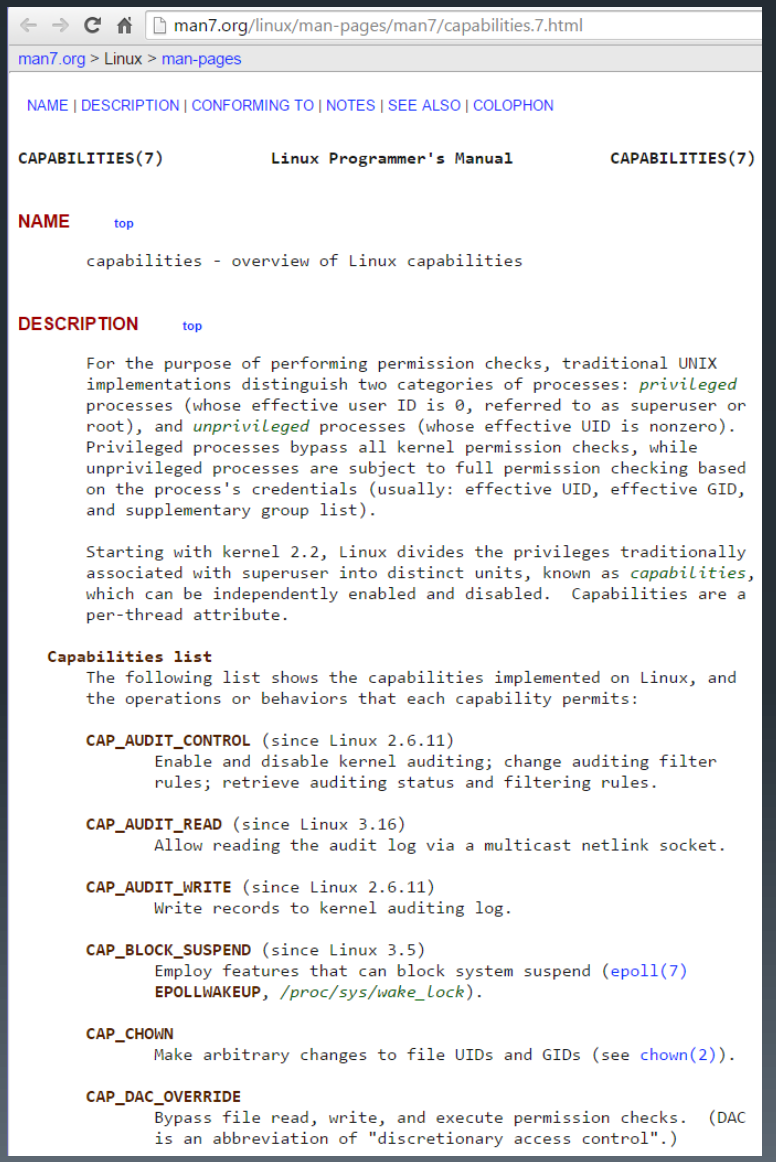
Container events

- The Docker events sub command allows you to monitor container events on a given Docker engine
- Container events reported:
  - create, destroy (rm), die (container processes exited), export, kill, oom (container died because it was out of memory), pause, restart, start, stop, unpause
- Image events reported:
  - untag, delete



Docker Security Features

- The Docker command offers several security related switches
  - --privileged
    - Grants all capabilities inside a container (no whitelist)
    - Not recommended for production (container processes run as if directly on the host)
  - --cap-add & --cap-drop (v1.2)
    - Linux Capabilities turn the binary "root/non-root" dichotomy into a fine-grained access control system
    - Containers can be given complete capabilities or they can follow a whitelist of allowed capabilities
    - --cap-add/drop give you fine grain control over Linux Capabilities granted to a particular container
    - Examples:
      - docker run --cap-add=NET_ADMIN ubuntu sh -c "ip link eth0 down"
      - docker run --cap-drop=CHOWN ...
      - docker run --cap-add=ALL --cap-drop=MKNOD ...
    - Linux Capabilities documentation: http://man7.org/linux/man-pages/man7/capabilities.7.html
      - There are currently 38 seperate capabilities
  - --device (v1.2)
    - You can use devices in privileged containers by bind mounting them (with `-v`)
    - The --device flag lets you use a device without requiring a --privileged container
    - Examples:
      - docker run --device=/dev/snd:/dev/snd ...
  - --security-opt (v1.3)
    - Sets custom SELinux and AppArmor labels and profiles
    - A policy (“svirt_apache”) allowing a container process to listen only on Apache ports can be applied as follows:
      - docker run --security-opt label:type:svirt_apache -i -t centos \ bash
    - Enables docker-in-docker without use of --privileged on kernels supporting SELinux or AppArmor



man7.org/linux/man-pages/man7/capabilities.7.html

man7.org > Linux > man-pages

NAME | DESCRIPTION | CONFORMING TO | NOTES | SEE ALSO | COLOPHON

CAPABILITIES(7) Linux Programmer's Manual CAPABILITIES(7)

NAME top

capabilities - overview of Linux capabilities

DESCRIPTION top

For the purpose of performing permission checks, traditional UNIX implementations distinguish two categories of processes: *privileged* processes (whose effective user ID is 0, referred to as superuser or root), and *unprivileged* processes (whose effective UID is nonzero). Privileged processes bypass all kernel permission checks, while unprivileged processes are subject to full permission checking based on the process's credentials (usually: effective UID, effective GID, and supplementary group list).

Starting with kernel 2.2, Linux divides the privileges traditionally associated with superuser into distinct units, known as *capabilities*, which can be independently enabled and disabled. Capabilities are a per-thread attribute.

Capabilities list

The following list shows the capabilities implemented on Linux, and the operations or behaviors that each capability permits:

CAP_AUDIT_CONTROL (since Linux 2.6.11)
Enable and disable kernel auditing; change auditing filter rules; retrieve auditing status and filtering rules.

CAP_AUDIT_READ (since Linux 3.16)
Allow reading the audit log via a multicast netlink socket.

CAP_AUDIT_WRITE (since Linux 2.6.11)
Write records to kernel auditing log.

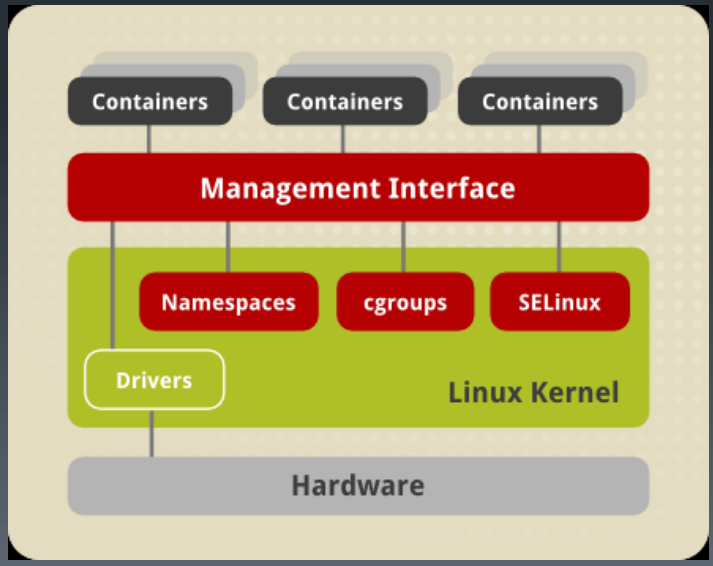
CAP_BLOCK_SUSPEND (since Linux 3.5)
Employ features that can block system suspend (epoll(7) **EPOLLWAKEUP**, */proc/sys/wake_lock*).

CAP_CHOWN
Make arbitrary changes to file UIDs and GIDs (see chown(2)).

CAP_DAC_OVERRIDE
Bypass file read, write, and execute permission checks. (DAC is an abbreviation of "discretionary access control".)

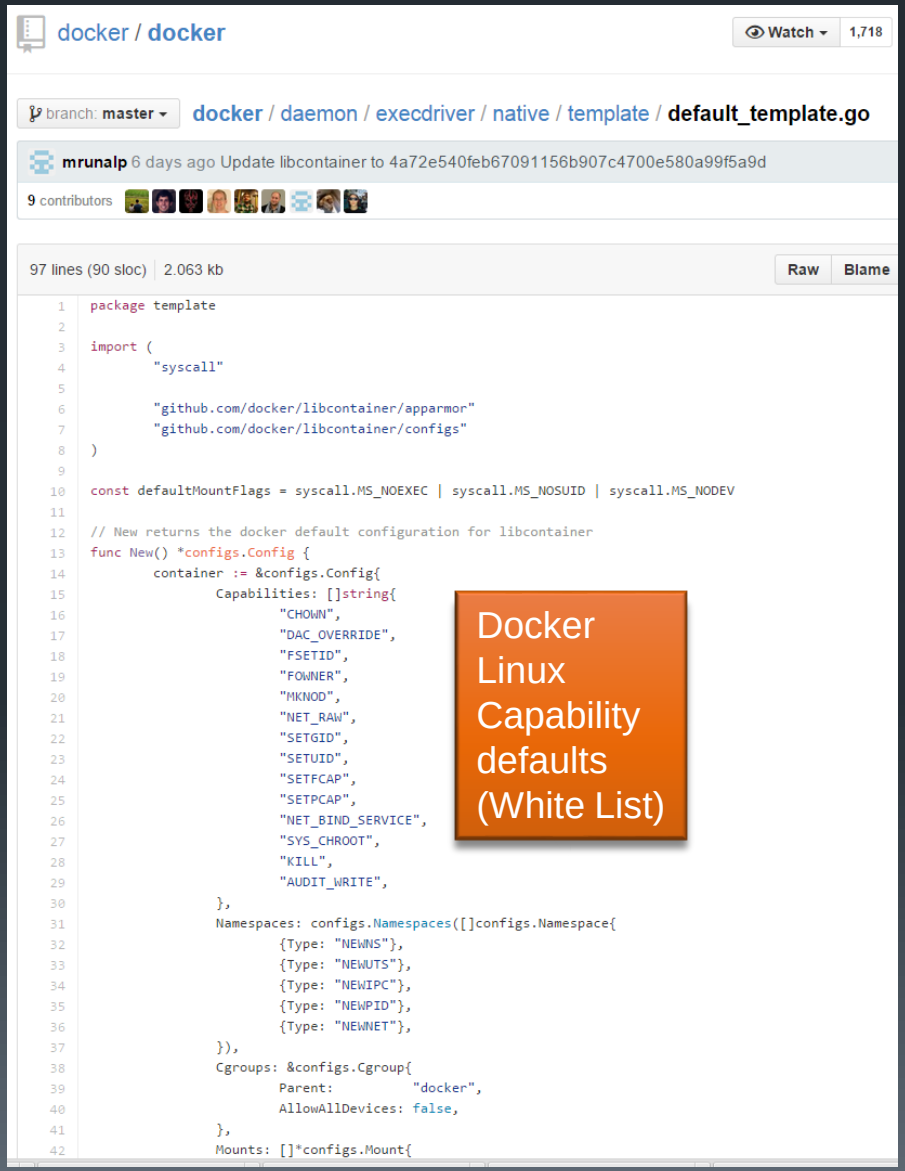
Docker Security

- Bottom Line (as of August 2015):
  - Assume privileged processes in a Docker container can do anything a privileged processes on the host can do
- Today (August 2015)
  - Docker containers can not credibly impede malicious privileged code seeking to access the host
  - Docker uses five namespaces with containers: Process, Network, Mount, Hostname, Shared Memory
    - Namespaces were added to kernel 2.6 in 2008
    - Many kernel subsystems are not namespaced and are accessible to containers e.g. SELinux, Kernel Modules and some paths for /sys, /proc, and /dev
  - Docker on a single-tenant system using good security practices for services in containers is not a problem
  - Docker on multitenant systems and/or systems seeking to use containers to solve security issues is problematic
- Many future security improvements are under consideration
  - User namespaces (only enabled in common distros in 2014)
    - A kernel namespace allowing separation between UIDs on the host and the container
    - A range of container UID/GIDs (e.g. 70,000-71,000) can map into the host user namespace as 0-1000
    - The kernel treats UID 0 (root) inside the container as UID 70,000 outside the container (or nobody, UID 65534, or whatever)
    - Any UID on a file or a process that is not in the mapped range would be treated as UID=-1 and not be accessible in the container
    - Challenges associated with volume mounts remain, this is a work in progress and will likely require more kernel support as well as libcontainer and docker enhancements
  - Seccomp
    - There are 600 syscalls and a bug in any one could enable privilege escalation
    - Seccomp was developed by Google to remove system calls from a process (used with Chrome plugins)
    - Red Hat's Paul Moore created libseccomp to simplify the management of the syscall tree, now used in tools like qemu, systemd, lxc tools and others
    - Go bindings are underway for libseccomp to allow libcontainer to drop system calls from containers
  - Authentication and Logging
    - If you can talk to Docker's socket you can do anything
    - Adding authentication to the Docker socket host and RBAC is under consideration
    - Adding robust Docker admin activity logging is under consideration
- References:
  - https://docs.docker.com/articles/security/
  - https://opensource.com/business/15/3/docker-security-future
  - https://github.com/GDSSecurity/Docker-Secure-Deployment-Guidelines



Security Best Practices

- Run Docker Engine with AppArmor or SELinux to provide better process containment
  - Use --security-opt
- Map groups of mutually-trusted containers to separate machines
- Do not run untrusted applications with root privileges
- Follow the “Principle of least privilege”
  - In a particular abstraction layer of a computing environment, every module (such as a process, a user or a container) must be able to access only the information and resources that are necessary for its legitimate purpose
  - Use --cap-add/drop and –device, not --privileged



Docker Best Practices

- Use Dockerfiles to build images
  - Docker commit is great for experimentation and debugging but does not build an "Infrastructure as Code" style repeatable process
- Use Volumes Appropriately
  - Volumes can be shared between containers using --volumes-from and changes to large files are faster
  - Use volumes to share a directory between containers and when writing large amounts of data to a directory
- Use the Data Container Pattern with Volumes
  - Volumes are reference counted and deleted when no longer used by a container when "docker rm –v" is used, a dedicated (empty) Data Container associated with the volume allows the volume to have a life span independent of the other ephemeral containers using the volume
  - The Data Container associated with the disk volume need not ever be run for other containers to use the volume
  - The Data Container allows the underlying volume to be encapsulated, hiding details from other containers (which use --volumes-from to access the Data Container's volumes)
  - Base Data Containers on the same image the running containers which access the VOLUME use
- Choose a minimum permissions User
  - By default docker containers run as root with full control of the host system
  - Use the USER instruction to specify a non-root user for containers to run as
  - You can create the users and groups you need in the Dockerfile
  - Also know that images inherit their parent image's USER, override the parent image USER with a local USER statement when appropriate

Much borrowed from Project Atomic:
http://www.projectatomic.io/docs/docker-image-author-guidance/

Summary

- Docker supplies a virtual network on each Docker host
- Standard networking commands can be used to interact with Docker virtual networks
- Docker also supplies commands like “inspect”, “port” and “ps” to acquire additional information
- The Docker container linking feature allows containers to discover service targets at runtime
- Containers can mount host volumes independent of the normal image layered file system
- Containers can mount volumes in use by other containers

Lab 6

- Building application fabrics with Docker

Links

- Docker homepage - https://www.docker.com/
- Docker Hub - https://hub.docker.com
- Docker blog - http://blog.docker.com/
- Docker documentation - https://docs.docker.com/
- Docker code on GitHub - https://github.com/docker/docker
- Docker Forge - https://github.com/dockerforge
  - Collection of Docker tools, utilities, and services
- Docker mailing list - https://groups.google.com/forum/#!forum/docker-dev
- Docker on IRC: irc.freenode.net channel #docker
- Docker on Twitter - #docker
- StackOverflow - tag docker
- Docker Cheat Sheet - https://github.com/wsargent/docker-cheat-sheet

Books

- The Docker Book, Turnbull, 2014
- O'Reilly Books due in 2015:
  - Docker Up and Running (July 3, 2015)
  - Docker Cookbook (eta October)
  - Using Docker (eta October)
- Manning Books due in 2015
  - Docker in Action (available now in MEAP, eta December)
  - Docker in Practice (available now in MEAP, eta December)

The End

Many thanks for attending!

Appendix

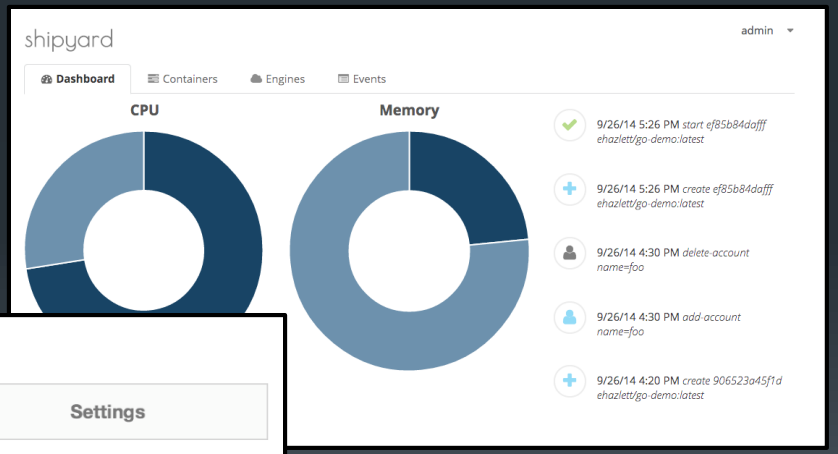
- Additional topics of interest for your reference

Docker APIs

- There are three important Docker Ecosystem REST APIs
  - Docker Remote API - the Docker daemon interface
    - https://docs.docker.com/reference/api/docker_remote_api/
  - Registry API - the Docker registry interface
    - https://docs.docker.com/reference/api/registry_api/
  - Docker Hub API - the Docker Hub interface
    - Extends the Registry API
    - Always available in the cloud: https://index.docker.io
    - https://docs.docker.com/reference/api/docker-io_api/

Docker GUIs

- There are several web UIs available for Docker
  - Shipyard – supports management of Docker hosts and containers (built on the Citadel cluster management toolkit)
    - Provides web and CLI interfaces
    - http://shipyard-project.com/
  - DockerUI - a web interface for the Docker Remote API
    - https://github.com/crosbymichael/dockerui
  - maDocker – a web based Docker Manager written with NodeJS & Backbone
    - Very basic
    - https://github.com/izifortune/maDocker

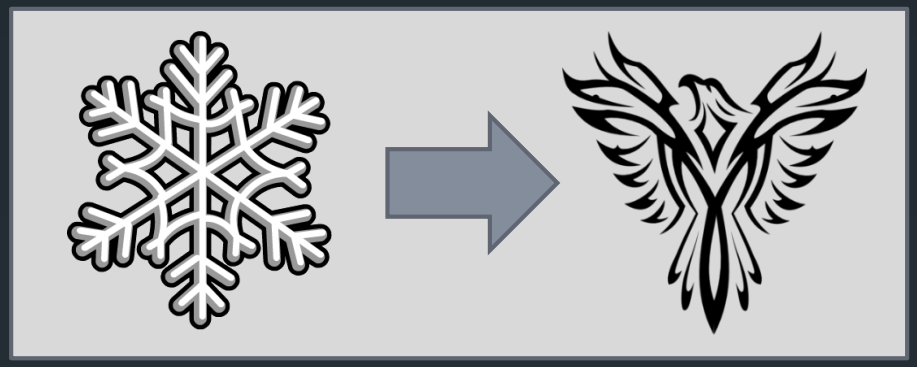


DockerUI

Home | Containers | Images | Settings

Containers:

Id	Image	Command	Created	Status
9c8a34d...	5886995bfd18	/bin/sh -c /usr/local/bin/sentry --config=/sentry.conf.py start	1370720983	Up 4 hours
d60b3f6...	26d8f45fc1d3	/bin/sh -c /usr/bin/redis-server /etc/redis/redis.conf	1370716229	Ghost



Turning a Snowflake into a Phoenix

- Snowflake servers have one of a kind configurations
  - Modified directly by admins over time
  - The server's configuration is not easy to reproduce
  - Not a good basis for modern IaC (Infrastructure as Code)
- Phoenix servers can be destroyed and restarted from a base over and over again
  - Like containers launched from images/Dockerfiles or VMs launched from a snapshot or nodes built from a Puppet manifest or a Chef recipe
- There are several ways you can generate a Docker image from a running snowflake system
  - Tar the entire file system and use docker import to create an image from the tarball
  - Use blueprint or other similar tools
    - https://zwischenzugs.wordpress.com/2015/05/24/convert-any-server-to-a-docker-container/

Changing Filesystem Drivers

- Ubuntu example changing the Docker Engine FS Driver to Device Mapper
  - First save (docker save) all repositories and then delete them from the docker host
  - Change the FS driver
    - $ service docker stop
    - $ vim /etc/default/docker
    - $ cat /etc/default/docker
      - ...
      - ### Set DOCKER_OPTS to use devicemapper:
      - DOCKER_OPTS="-s devicemapper"
      - ...
    - $ service docker start
  - Reload all require repositories (docker load)

Base Ubuntu servers need to add packages to use devicemapper:

- lvm2
- thin-provisioning-tools

aufs3 has a known bug which causes cpio cap_set_file to fail. The Apache httpd install on Centos attempts this. You can ignore this error if you see it during your lab. In production you can work around this issue by building the image on another file system driver or updating aufs. Aufs is unsupported on Linux kernel 3.13 as of 1/2015, requiring kernel 3.14-3.19. More info here: https://github.com/docker/docker/issues/6980