Container / Docker

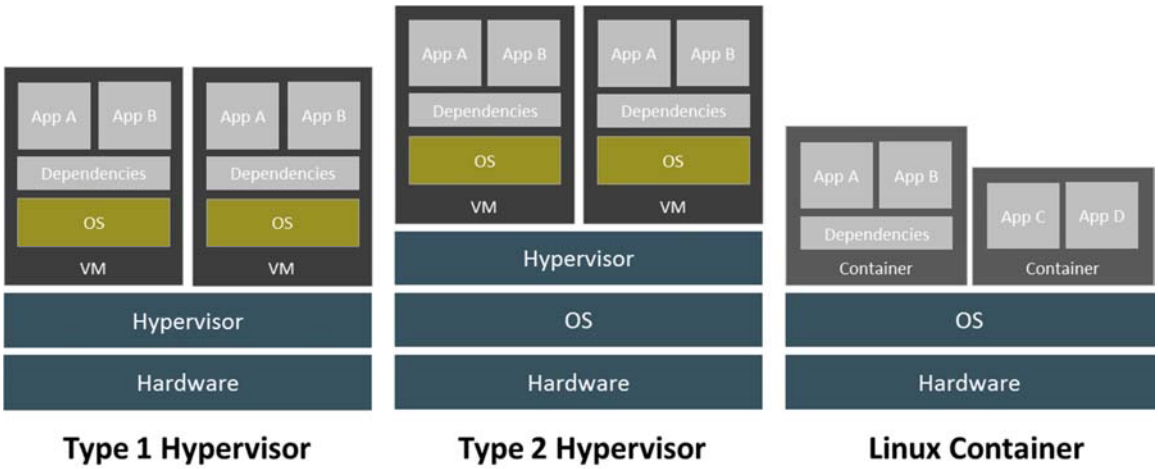
Tien-Fu Chen

Dept. of Computer Science and Information Engineering

National Chiao Tung Univ.

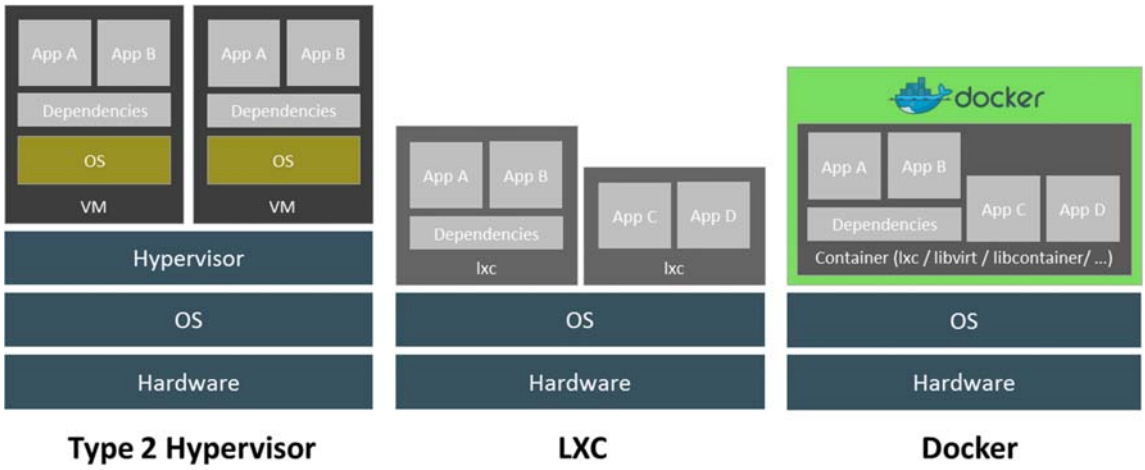
Virtualization Technologies

- **Containers are lightweight:**
  - **share the host OS kernel**
  - **share the host OS root filesystem wherever appropriate**



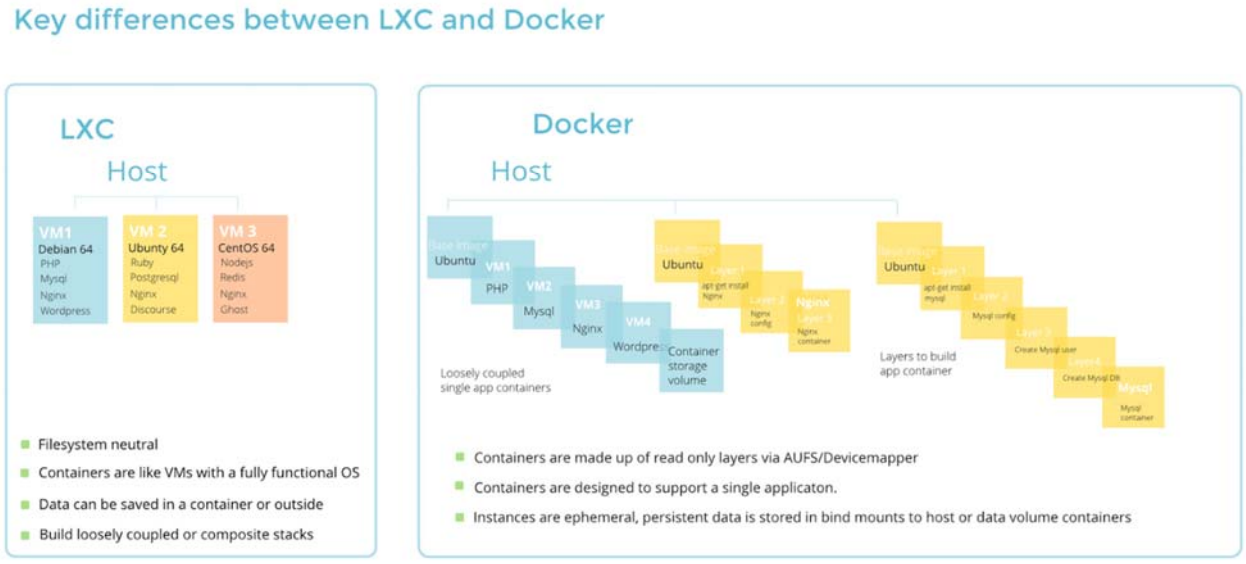
Virtualization Technologies

- **Docker provides a unified access to**
  - **Linux container technology (cgroups, namespaces)**
  - **Various container implementations (lxc, libvirt, libcontainer, etc.)**
- **'libcontainer' is Docker's implementation of container technology**



Key terminology

- Linux containers (LXC) are "lightweight" VMs
- Docker is a commoditized LXC technique that dramatically simplifies the use of LXC



LXC technique

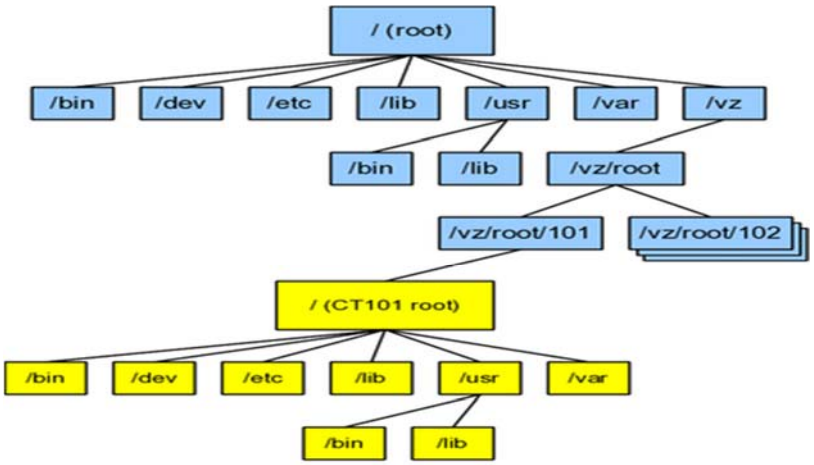
- Linux kernel provides the “control groups” (cgroups) functionality
  - allows limitation and prioritization of resources (CPU, memory, block I/O, network, etc.) without the need for starting any VM
- “namespace isolation” functionality
  - allows complete isolation of an applications' view of the operating environment,including process trees, networking, user IDs and mounted file systems.

Unique features

- Containers running in the user space
- Each container has
  - Own process space
  - Own network interface
  - Own /sbin/init (coordinates the rest of the boot process and configures the environment for the user)
  - Run stuff as root
- Share kernel with the host
- No device emulation

Namespaces

- **Namespaces** are a Linux kernel feature that isolates and virtualizes resources (PID, hostname, userid, network, ipc, filesystem) of a collection of processes.
- Provide processes with their own view of the system
- Each process is in one namespace of each type



Isolation with namespaces

- Check the results of
  - pid, mnt, net, uts, ipc, user
- Pid namespace
  - Type “ps aux| wc –l” in host and the container
- Mnt namespace
  - Type “wc –l /proc/mounts” in both
- Net namespace
  - Install net-tools
  - Type “ifconfig”

- hostname namespace
  - “hostname”
- ipc namespace
  - Type “ipcs”
- User namespace
  - UID 0-1999 in the first container mapped to UID 10000 – 11999 in host
  - UID 0-1999 in the 2nd container mapped to UID 12000 – 13999 in host

Isolation with cgroups

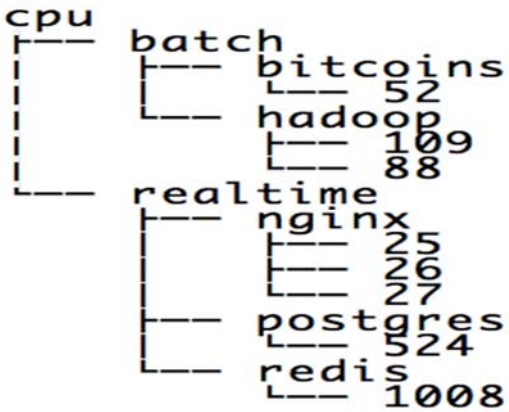
- **cgroups** (**control groups**) is a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes.
- Memory
- CPU
- Blkio
- devices

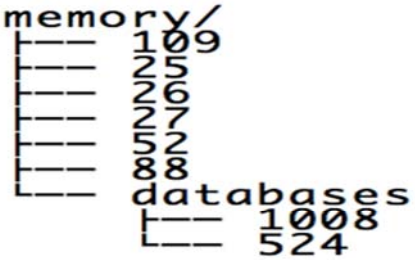
Memory cgroup

- keeps track pages used by each group:
  - file (read/write/mmap from block devices; swap)
  - anonymous (stack, heap, anonymous mmap)
  - active (recently accessed)
  - inactive (candidate for eviction)
- each page is charged to a group
- pages can be shared
- Individual (per-cgroup) limits and out-of-memory killer

CPU cgroup

- keep track of user/system CPU time
- set relative weight per group
- pin groups to specific CPU(s)
  - Can be used to reserve CPUs for some apps





Blkio cgroup

- keep track IOs for each block device
  - read vs write; sync vs async
- set relative weights
- set throttle (limits) for each block device
  - read vs write; bytes/sec vs operations/sec

Devices cgroup

- controls read/write/mknod permissions
- typically:
  - allow: /dev/{tty,zero,random,null}...
  - deny: everything else
  - maybe: /dev/net/tun, /dev/fuse, /dev/kvm, /dev/dri...
- fine-grained control for GPU, virtualization, etc

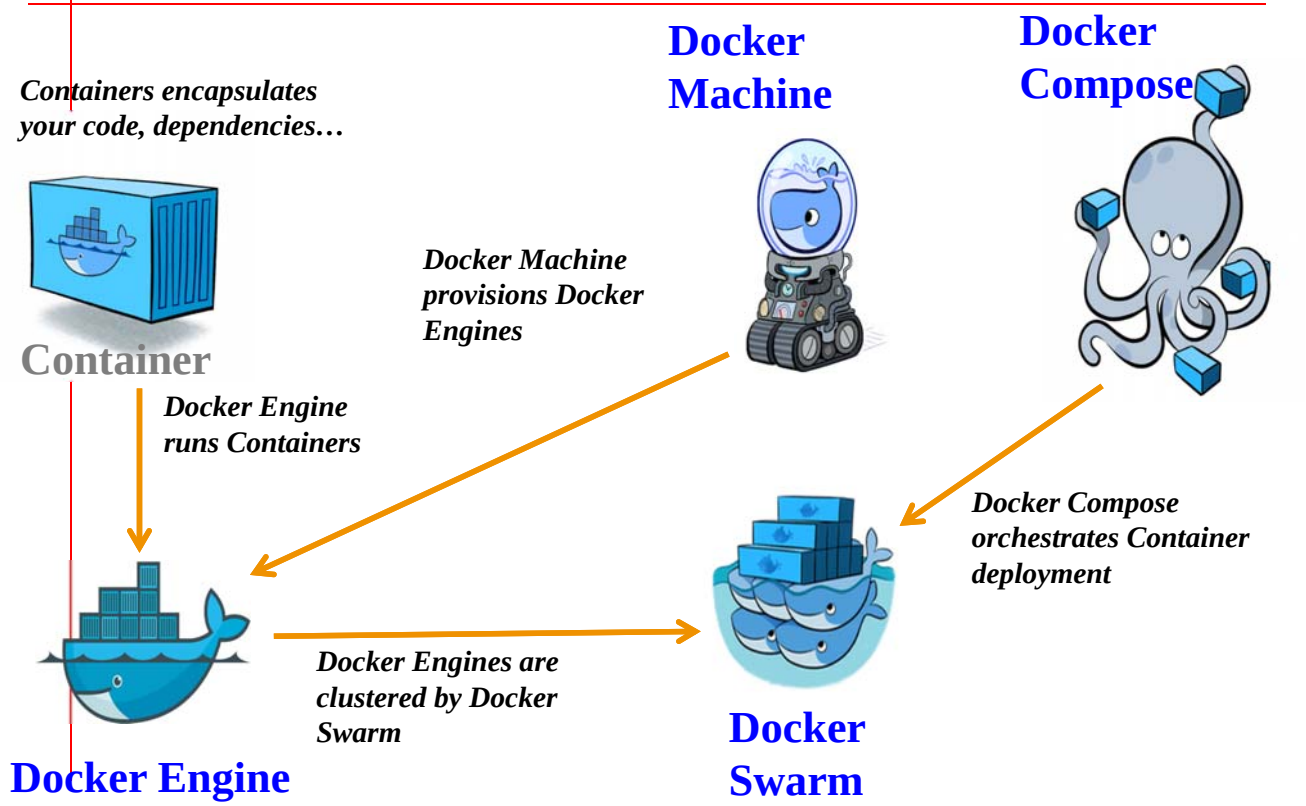
Almost no overhead

- processes are isolated, but run straight on the host
- CPU performance = native performance
- memory performance = a few % shaved off for (optional) accounting
- network performance = small overhead; can be reduced to zero

What is Docker

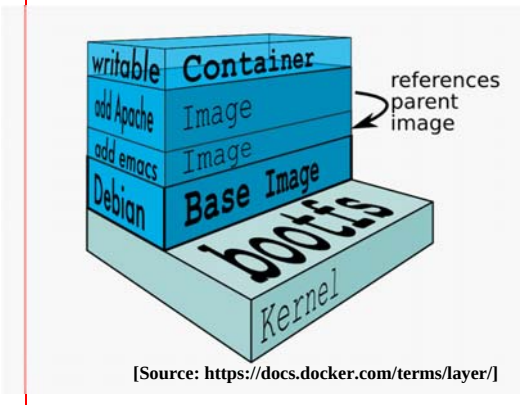
- Open Source engine to commoditize LXC
- using copy-on-write for quick provisioning
  - Every instance of your Docker image uses the same files until one of them needs to change a file.
  - Better utilization of system memory.
  - Higher density of containers for a given resource than other container implementations.
- Container Repository
  - allowing to **create and share** *images*
- **Component reuse**
  - **standard format** for containers
  - standard, *reproducible* way to *easily* build *trusted* images (Dockerfile, Stackbrew...)

Docker Landscape in Pictures

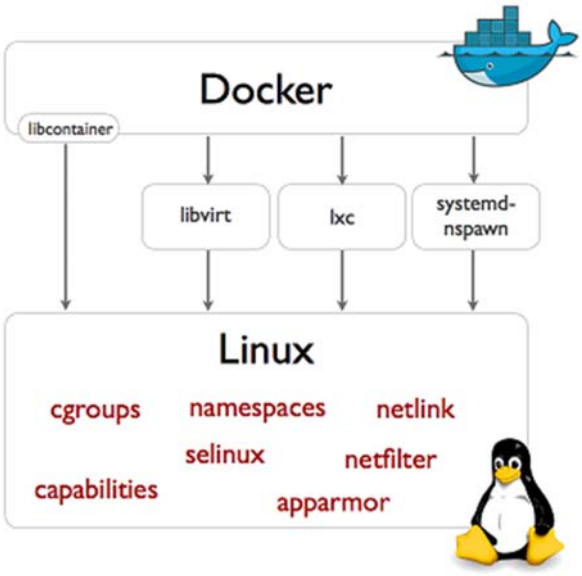


Docker Technology

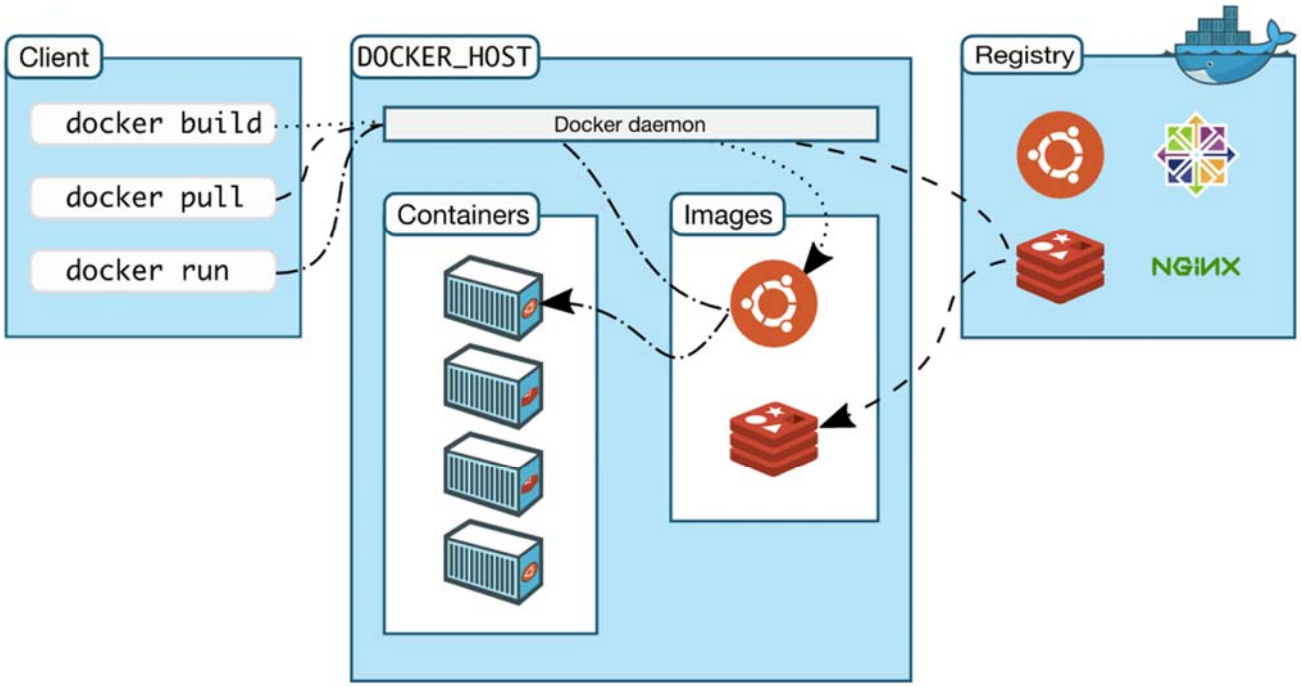
- libvirt: Platform Virtualization
- LXC (LinuX Containers): Multiple isolated Linux systems (containers) on a single host
- Layered File System



[Source: https://docs.docker.com/terms/layer/]

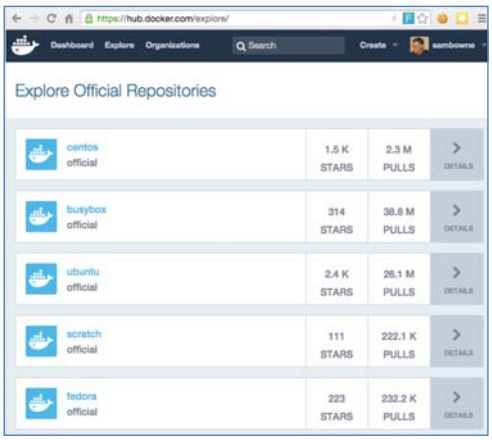


Docker Architecture



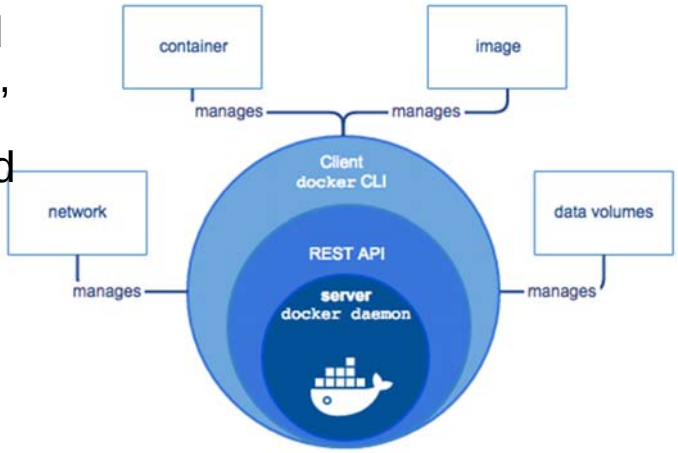
Main Docker Components

- Docker
  - the open source container virtualization platform
- Docker Hub
  - Software-as-a-Service platform for sharing and managing Docker containers
  - The primary public Docker registry
- Features
  - Provides Docker Services
  - Library of public images
  - Storage for your images
  - free for public images
  - cost for private images
  - Automated builds(link github/bitbucket repo; trigger build on commit)



Docker Engine

- A client-server application with these major components
  - A server which is a type of long-running program called a daemon process.
  - A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
  - A command line interface (CLI) client.
- The daemon creates and manages Docker objects, such as images, containers, networks, and data volumes



Images, Registries, and Containers

- Docker images
  - Read-only templates with OS and installed software
  - Used to **build** servers
- Docker registries
  - Stores that hold images. May be public or private.
  - You upload or download images from stores
  - Used to **distribute** servers
- Docker containers
  - Like virtual machines
  - Can be run, started, stopped, moved, and deleted
  - Used to **run** servers

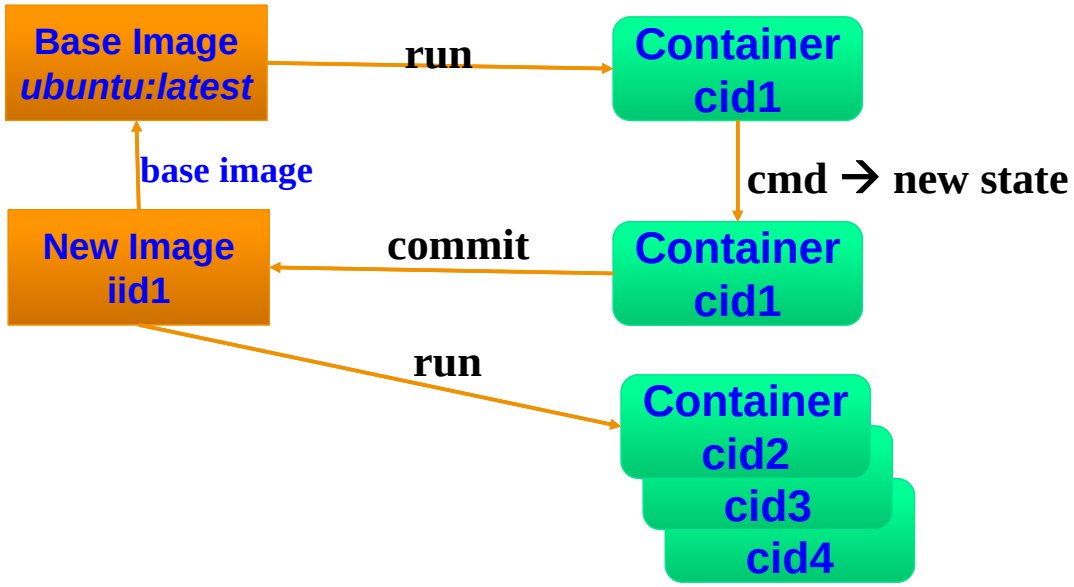
Terminology - Image

- Persisted snapshot that can be run
  - *images:* List all local images
  - *run*: Create a container from an image and execute a command in it
    - **docker run -it** <*image*> **bash**
      - Creates a container based on <*image*> , starts it, and shows a Bash shell
  - *tag*: Tag an image
  - *pull*: Download image from repository
  - *rmi*: Delete a local image
    - This will also remove intermediate images if no longer used

Terminology - Container

- Runnable instance of an image
  - *ps:* List all running containers
  - *ps –a*: List all containers (incl. stopped)
  - *top*: Display processes of a container
  - *start*: Start a stopped container
  - *stop*: Stop a running container
  - *pause*: Pause all processes within a container
  - *rm*: Delete a container
  - *commit*: Create an image from a container
- **Ctrl+P**, **Ctrl+Q**
  - Detaches from a running container without stopping it
- **docker run -p 8080:80 -it** <*image-name*> **bash**
  - Creates a container, forwarding port 8080 on the host to port 80 inside the container, showing a bash shell

Image vs. Container



$ docker run -i -t ubuntu /bin/bash

1. **Pulls the ubuntu image:** Docker Engine checks for the presence of the ubuntu image. If exists locally, uses it for the new container. Otherwise, then Docker Engine pulls it from Docker Hub.
2. **Creates a new container:**
3. **Allocates a filesystem and mounts a read-write *layer*:** container is created in file and a read-write layer is added.
4. **Allocates a network / bridge interface:** Creates a network interface that allows the Docker container to talk to the local host.
5. **Sets up an IP address:** Finds and attaches an IP from a pool.
6. **Executes a process that you specify:** Executes the /bin/bash.
7. **Captures and provides application output:** Connects and logs standard input, outputs and errors for your application, because of interactive mode.

Advanced Docker-ing

- Interactive containers:
  - `$ sudo docker run -t -i ubuntu:14.04 /bin/bash`
    - -t creates a pseudo-terminal
    - -i captures STDIN
- Results (example):
  -
```
root@af8bae53bdd3:/# pwd
/
root@af8bae53bdd3:/# ls
bin boot dev etc home lib lib64 media
mnt opt proc root run sbin srv sys
tmp usr var
```


Advanced Docker-ing

- Hello world daemon
  - `$ sudo docker run -d ubuntu:14.04 /bin/sh -c "while true; do echo hello world; sleep 1; done“`
    - -d runs containers in the background (daemonizes them)
  - This example returns a container ID, which you can use to interact with the container running that command
  - To stop a container, use `docker stop $container_name`

Running a webapp in Docker

- Docker's example runs a Python Flask app:
  - `$ sudo docker run -d -P training/webapp python app.py`
    - -P: map required network ports inside the container to the host
  - Check results with `sudo docker ps –l`:
    - `CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES`
      `bc533791f3f5 training/webapp:latest python app.py 5 seconds ago Up 2 seconds 0.0.0.0:49155->5000/tcp nostalgic_morse`
    - -l: tells `docker ps` to return the last container started

Running a webapp in Docker

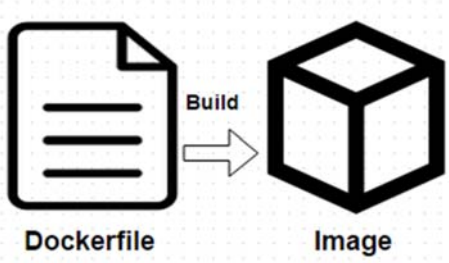
- Ports: super malleable
  - -P is a shortcut for –p 5000
  - Docker generally picks a high external port to map to
  - Can run 1:1 on your port mappings, but this way you can have multiple apps thinking they're on the same port
  - Lookup: use `docker port $container_name $port`

Building docker image

- With run/commit commands
  1) docker run ubuntu bash
  2) apt-get install this and that
  3) docker commit <containerid> <imagename>
  4) docker run <imagename> bash
  5) git clone git://.../mycode
  6) pip install -r requirements.txt
  7) docker commit <containerid> <imagename>
  8) repeat steps 4-7 as necessary
  9) docker tag <imagename> <user/image>
  10) docker push <user/image>

Dockerfile

- Each Dockerfile is a script,
  - composed of various commands (instructions) and arguments listed successively
  - to automatically perform actions on a base image in order to create (or form) a new one.
- Using docker build users can create an automated build that executes several command-line instructions in succession.



Authoring image with a dockerfile

- A sample dockerfile

```
FROM ubuntu
RUN apt-get -y update
RUN apt-get install -y g++
RUN apt-get install -y erlang-dev erlang-manpages erlang-base-hipe ...
RUN apt-get install -y libmozjs185-dev libicu-dev libtool ...
RUN apt-get install -y make wget
RUN wget http://.../apache-couchdb-1.3.1.tar.gz | tar -C /tmp -zxf-
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"]
```

Run the command to build:
docker build -t your_account/couchdb .

Docker Hub

- Public repository of Docker images
  - https://hub.docker.com/
  - docker search [term]
- Automated: Has been automatically built from Dockerfile
  - Source for build is available on GitHub

Dev-> test->production

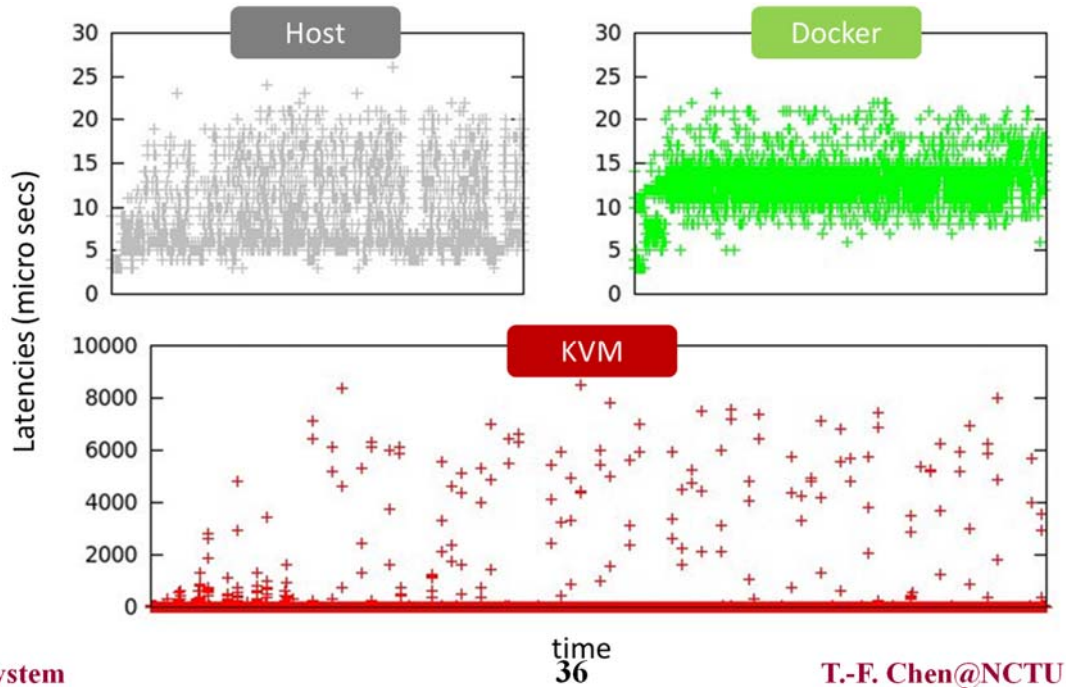
- code in local environment
  (« dockerized » or not)
- each push to the git repo triggers a hook
- the hook tells a build server to clone the code and run

« docker build » (using the Dockerfile)

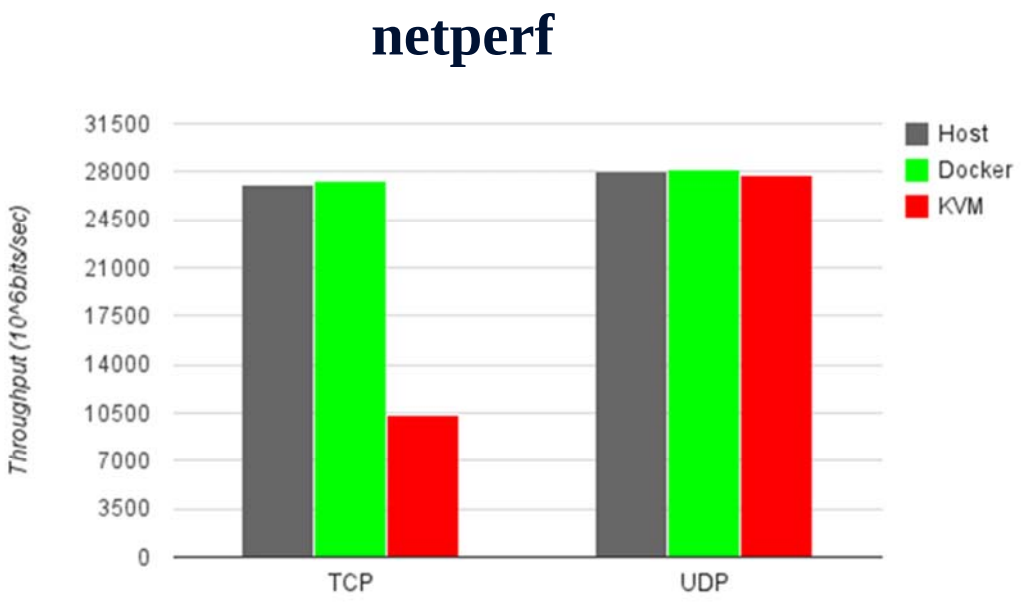
- the containers are tested (nosetests, Jenkins...),
  and if the tests pass, pushed to the registry
- production servers pull the containers and run them
- for network services, load balancers are updated

Performance Benchmarks
Real-time Latency of event handling

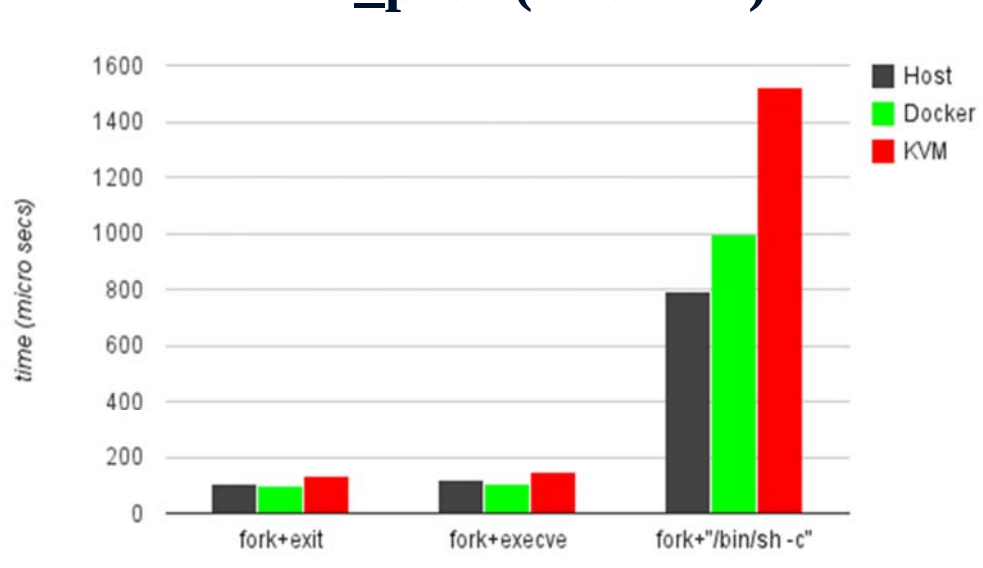
Cyclictest



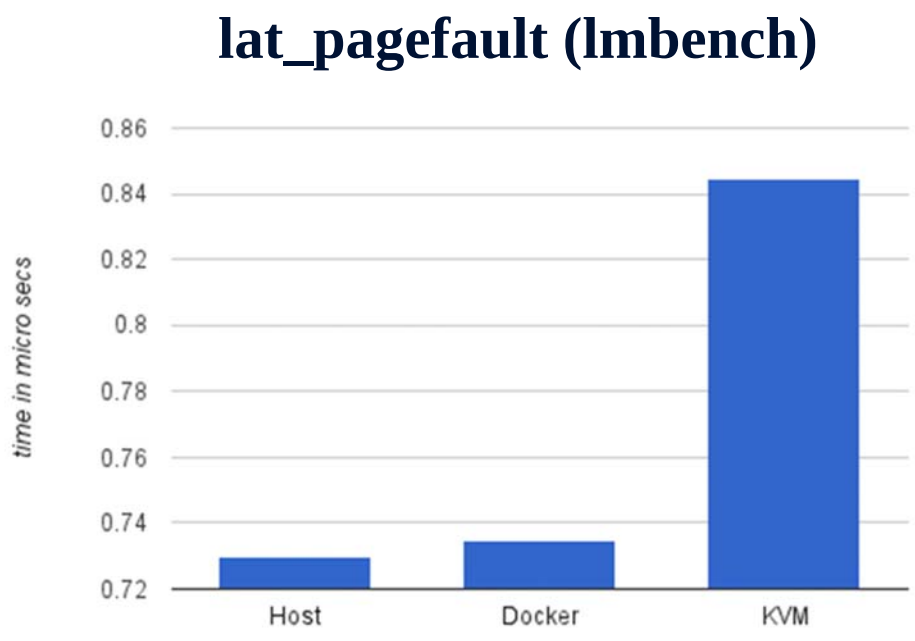
Network Performance



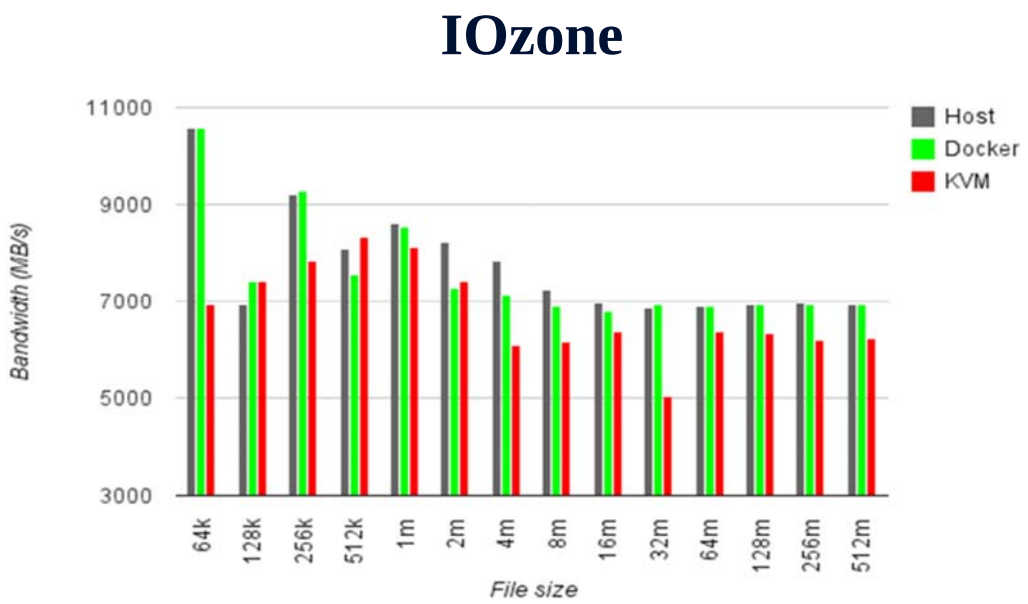
Process Creation



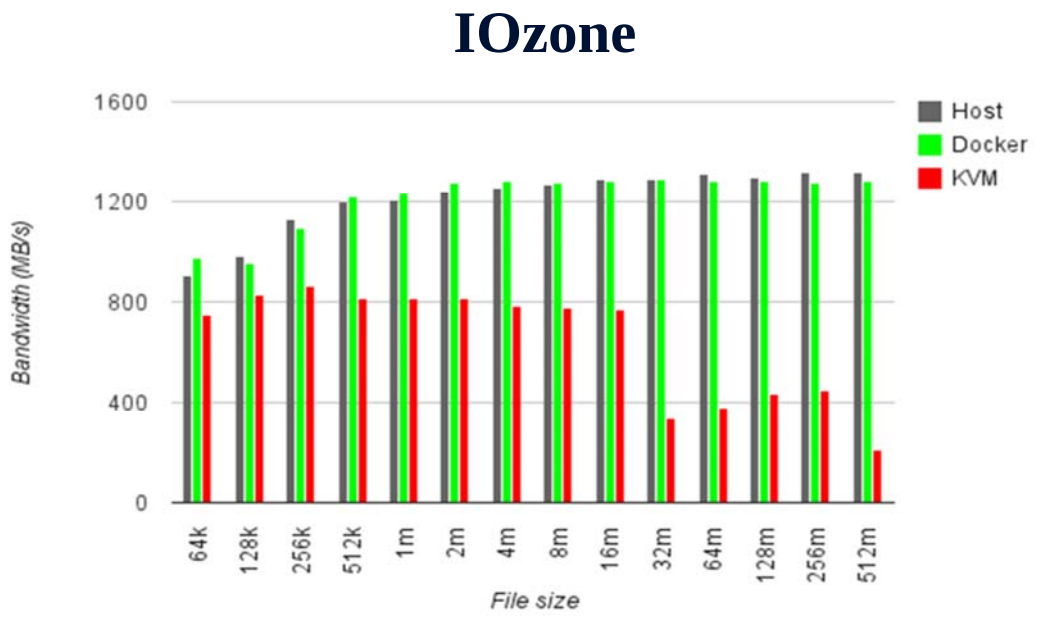
Page Fault



File-system Read Performance



File-system Write Performance



Orchestration

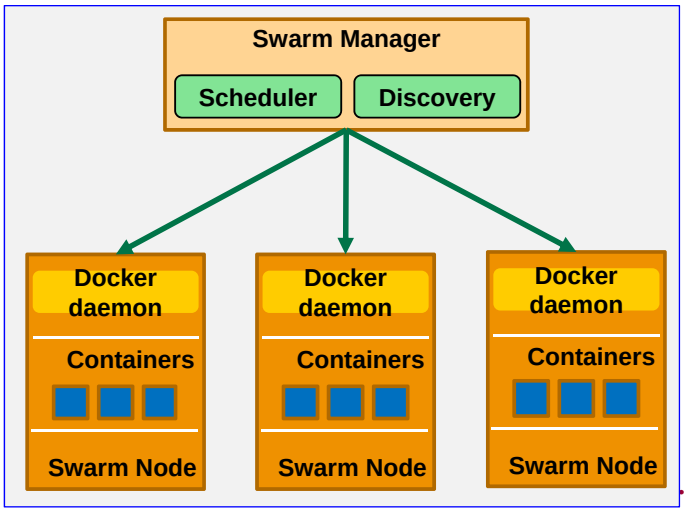
Main goal: To provide automated container management as well as guarantees for multi-container services and container engines.

- Marathon with Apache Mesosphere(Scheduler)
  - Multiple physical node scaling (treated as one machine)
- Kubernetes (Google Inc.)
  - Large scale service oriented design
  - "Self-Healing" services and Load balancing
- Swarm (Docker)
  - Standard basic clustering tool
  - Native Docker API for third party tool integration

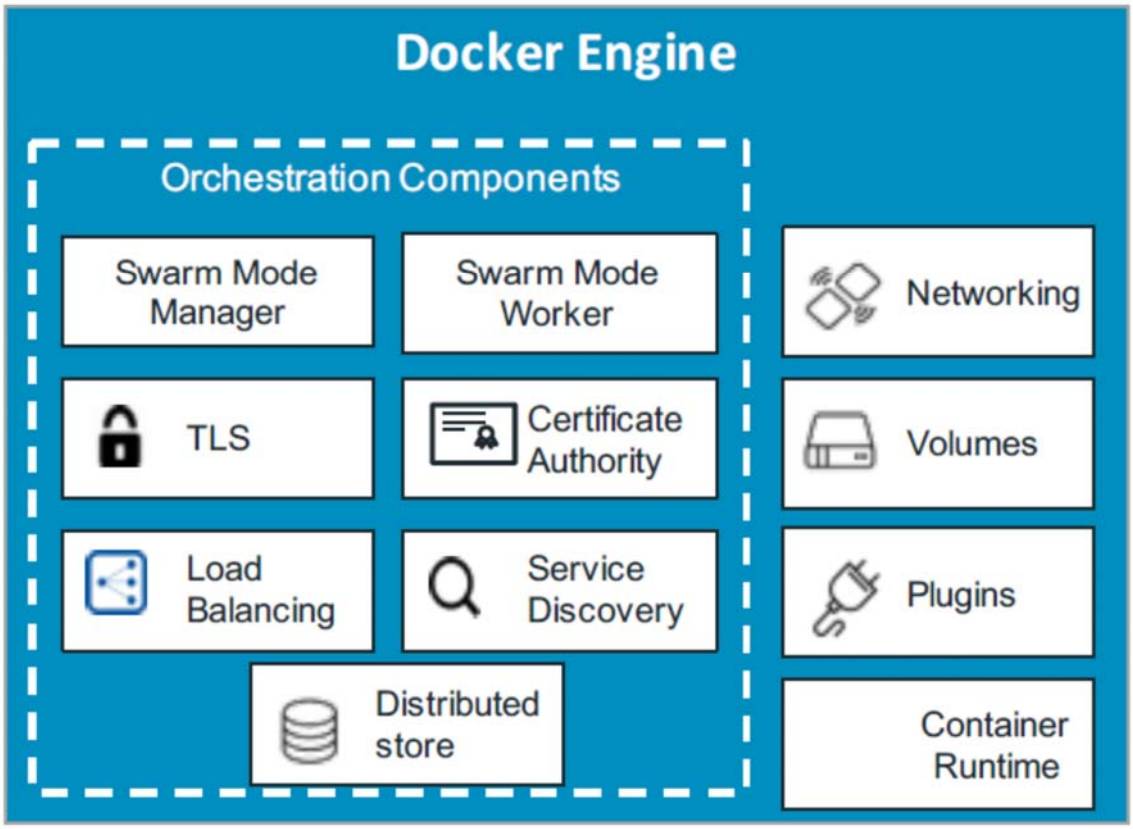
https://docs.mesosphere.com/overview/
https://github.com/coreos/fleet/blob/master/Documentation/fleet-k8s-compared.md

Swarm in a nutshell

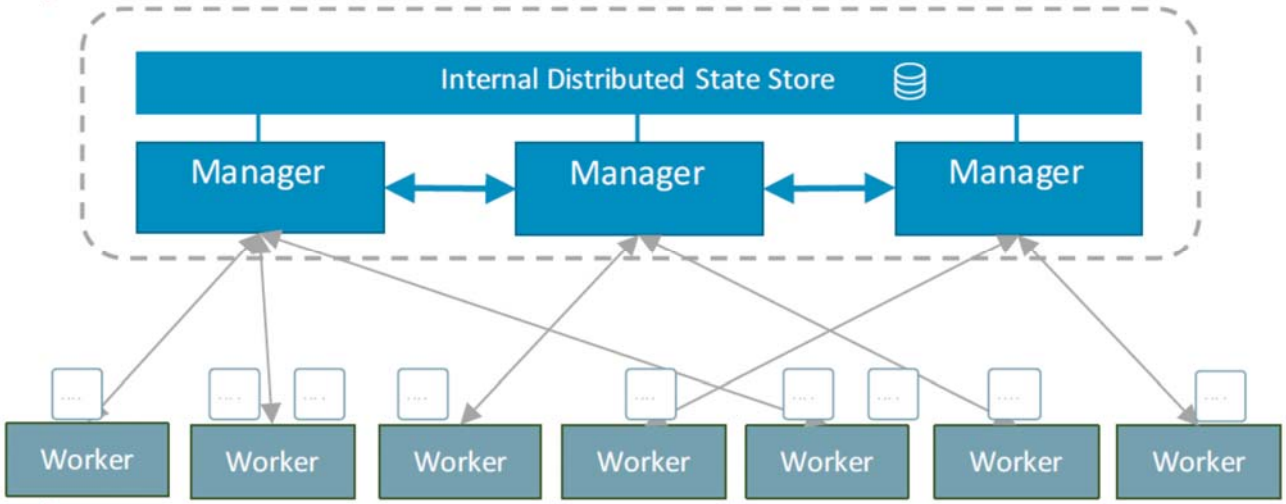
- Exposes several Docker Engines as a single virtual Engine
- Exposes standard Docker API
- Tools that work with Docker can use Swarm to transparently scale to multiple hosts. Eg., Compose, Jenkins, Docker client



Swarm mode enabled Docker Engine



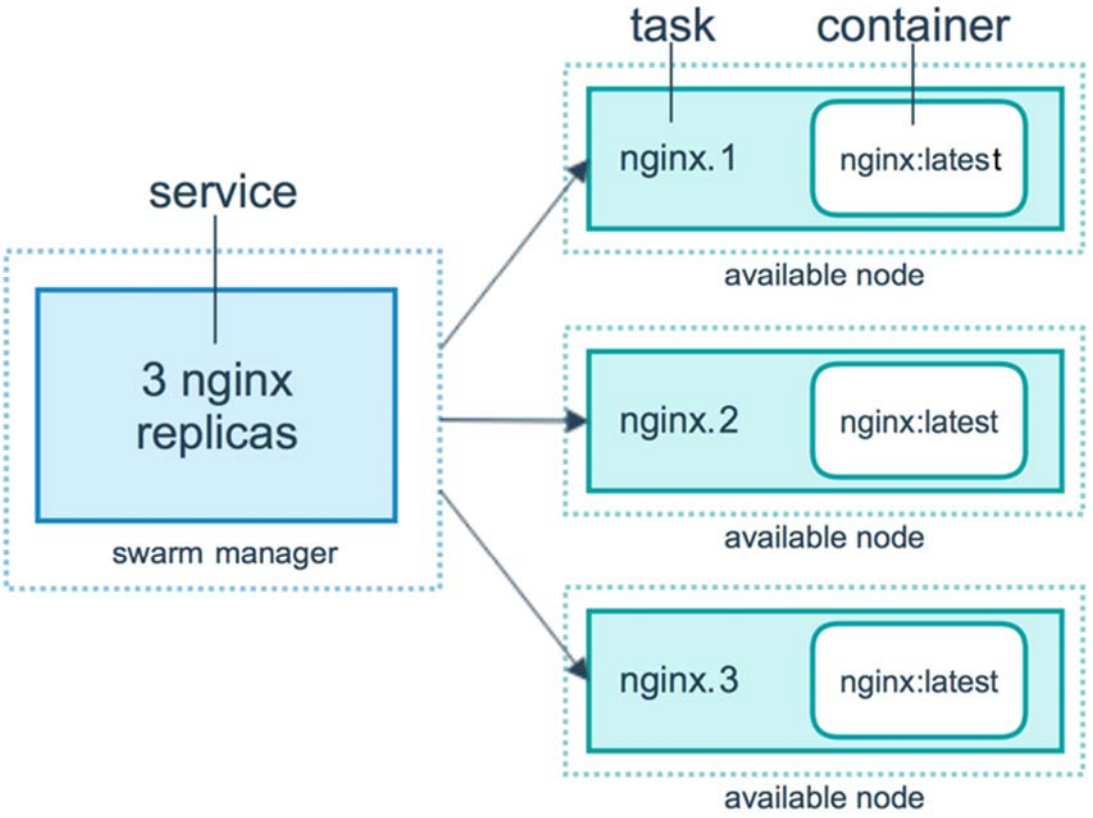
Docker Swarm mode cluster architecture Manager Manager



Docker Swarm mode services

- Services
  - the definition of the tasks to execute on the worker nodes.
  - the central structure of the swarm system and the primary root of user interaction with the swarm.
- Declarative definition
  - Replicas
  - Global vs replicated
  - Deploy / restart strategies
  - Health checks
- Tasks
  - Created from services
  - Scheduled to nodes
- docker service create [OPTIONS] IMAGE [COMMAND] [ARG...]

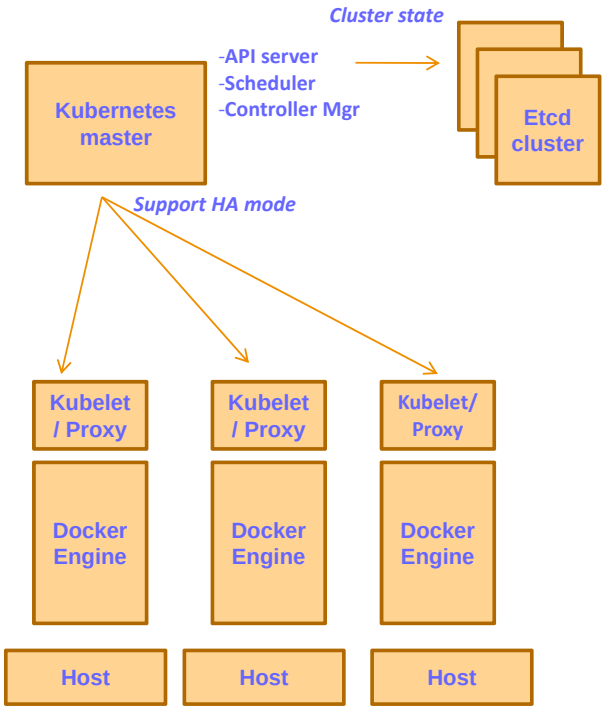
Services, tasks, and containers



Kubernetes in a nutshell

Open source orchestration system for Docker containers

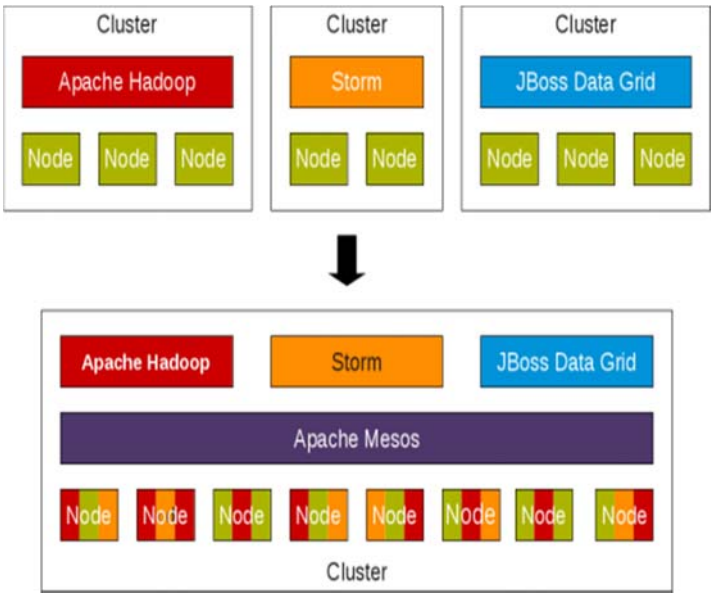
- Handles scheduling onto nodes in a compute cluster
- Actively manages workloads to ensure that their state matches the users declared intentions
- "labels" and "pods" to group into logical units for easy management and discovery
- Replication controllers, services
- Model is quite different from native docker API / Swarm, cannot leverage Docker tools



Mesos in a nutshell

Open-source cluster manager

- Efficient resource isolation and sharing across distributed applications, or frameworks
- Enables siloed applications to be consolidated on a shared pool of resources, delivering:
  - **Higher utilization**
  - **Better application performance**
- Rich framework ecosystem
  - **Hadoop, Spark, Kubernetes, Marathon, Docker, Rocket, MongoDB, Elastic Search …**



Using Swarm & Kubernetes with Mesos

- Mesos manages the actual resources on the cluster
- Incoming API/CLI are stored in a queue, waiting for offers from Mesos
- The framework's scheduler is used to choose the target host from the Mesos offers
- The framework sends a "task" to Mesos slave to create the container