Midterm Presentation Schedule

- October 18th
  - Omni, Aurora, Aviato, Bash, CodeRing
- October 20th
  - Flash, NPComplete, Omega, Sangam

Mid Term Presentation Format

- 15 minutes
  - Be prepared to use the class PC
  - Slides in PDF uploaded to Canvas
- Give a demo of your system
  - Show how to run end-to-end through Travis, Code Deploy
- Describe your system implementation
  - Which programming languages did you use?
  - What tools did you use (Web servers, etc)
  - How did you implement inter-service communication
- Discuss the problems you faced and how you solved them

Containers, Docker, and Microservices

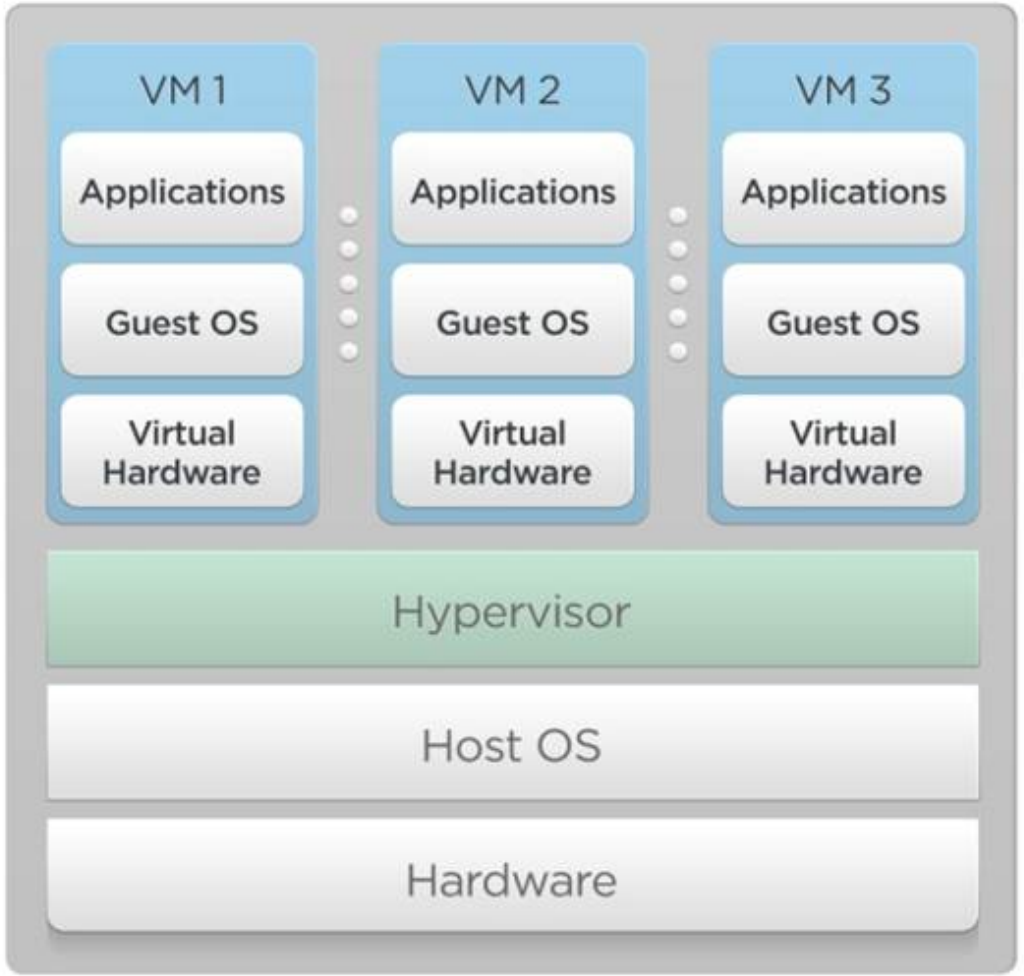
Efficient deployment of self-contained services

Project Milestone 2 Requirements

- Wrap all of your microservices in Docker containers.
  - Use Dockerfile
  - Decide when to use Docker Compose and when not
- Integrate your dockerized services into Travis-CI.
- Use Amazon Code Deploy to deploy your containers to Amazon
- Optional now, but required for Project Milestone 3: integrate Code Deploy with the Amazon Elastic Container Service

Hypervisor Virtualization

- Hypervisors provide software emulated hardware and support multiple OS tenants.
- Type 1 Hypervisors run directly on the hardware
  - Kernel-based Virtual Machine (KVM)
  - Xen: Amazon uses (or used) this
- Type 2 Hypervisors run as another program in the HostOS
  - Virtual Box
  - VMWare

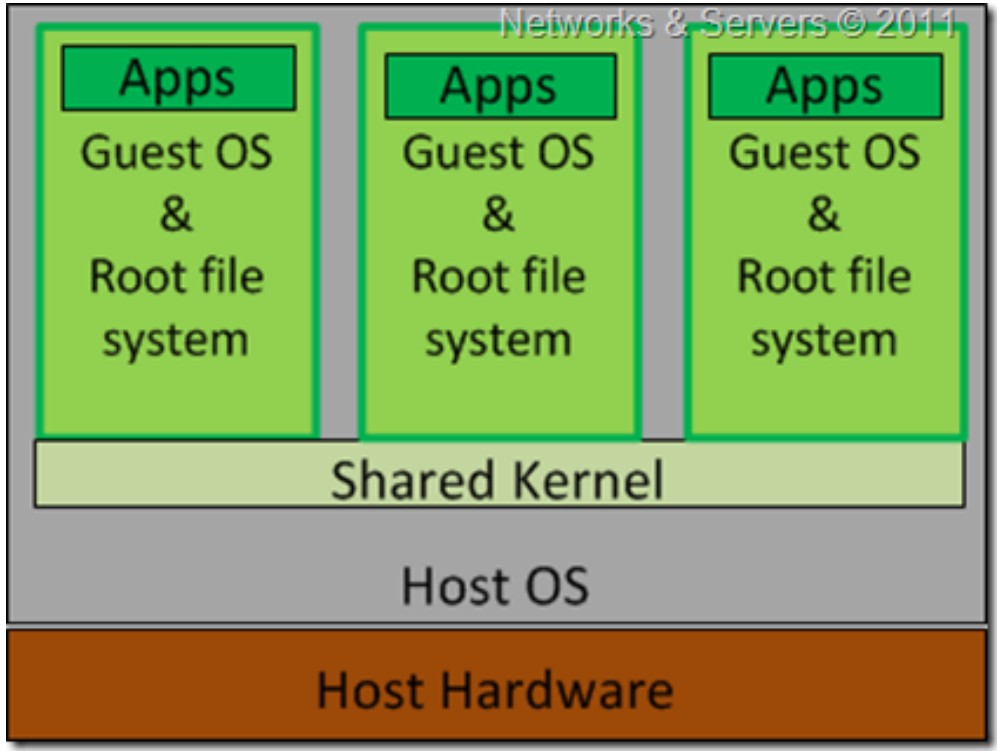


Drawbacks to Using Hypervisors

- Virtual Machines using hypervisors access **virtual hardware**.
- This allows VMs to run a wide range of guest operating systems.
- But it adds overhead
  - VMs take a long time to startup.
  - Performance is worse: software emulated hardware
  - VMs take up a lot of resources. Limited number of VMs can run on the real hardware

Operating System (OS) Virtualization

- The OS can run other OS instances as separate processes.
- No hardware virtualization or emulation.
- Tenants share the OS kernel with the host OS.
- The guest (tenant) OS instances are shielded from each other.
- This limits the OS options for the tenants
- But performances is much greater
- And you can fit a lot more OS VMs onto the host.
- But not as secure as Hypervisor Virtualization



Linux Containers (LXC)

- **Cgroups**: 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.
  - Cgroups lets you carve up the hardware into continuous, resizable pieces
- **Namespace Isolation:** groups of processes are separated such that they cannot "see" resources in other groups.
  - For example, a PID namespace provides a separate enumeration of process identifiers within each namespace
  - Shield container processes from the host and other containers' processes.

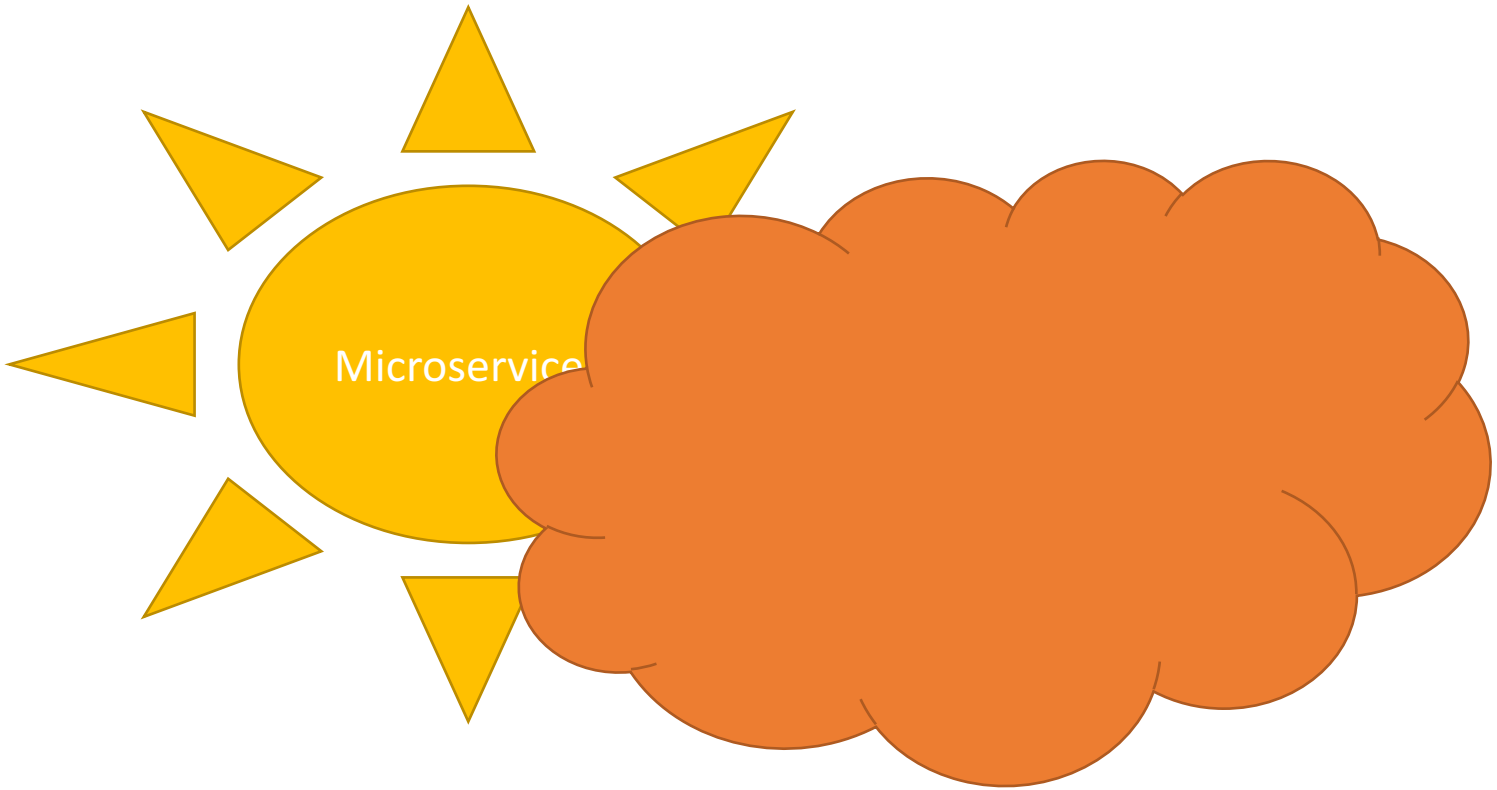
https://en.wikipedia.org/wiki/Cgroups

Container Takeaways

- When combined, cgroups and namespace isolation let you run much lighter weight VMs.
  - Less resource usage overhead for running VMs
  - VMs can start and stop much more quickly
- You can do further tricks to optimize containers.

Results: you can run dozens or even hundreds of containers on a single host server.

Hmm, when would I need to run lots of little VMs…?



Docker 101

Wrapping your Microservices in Containers

What is Docker?

- Docker packages LXC and other tools to make creating and using images easy.
  - No more DIY containers
- You can run Docker on Linux desktops and servers.
- Other OS's can run Docker inside a VM
  - MacOS: Use Virtual Box to run a Linux flavor

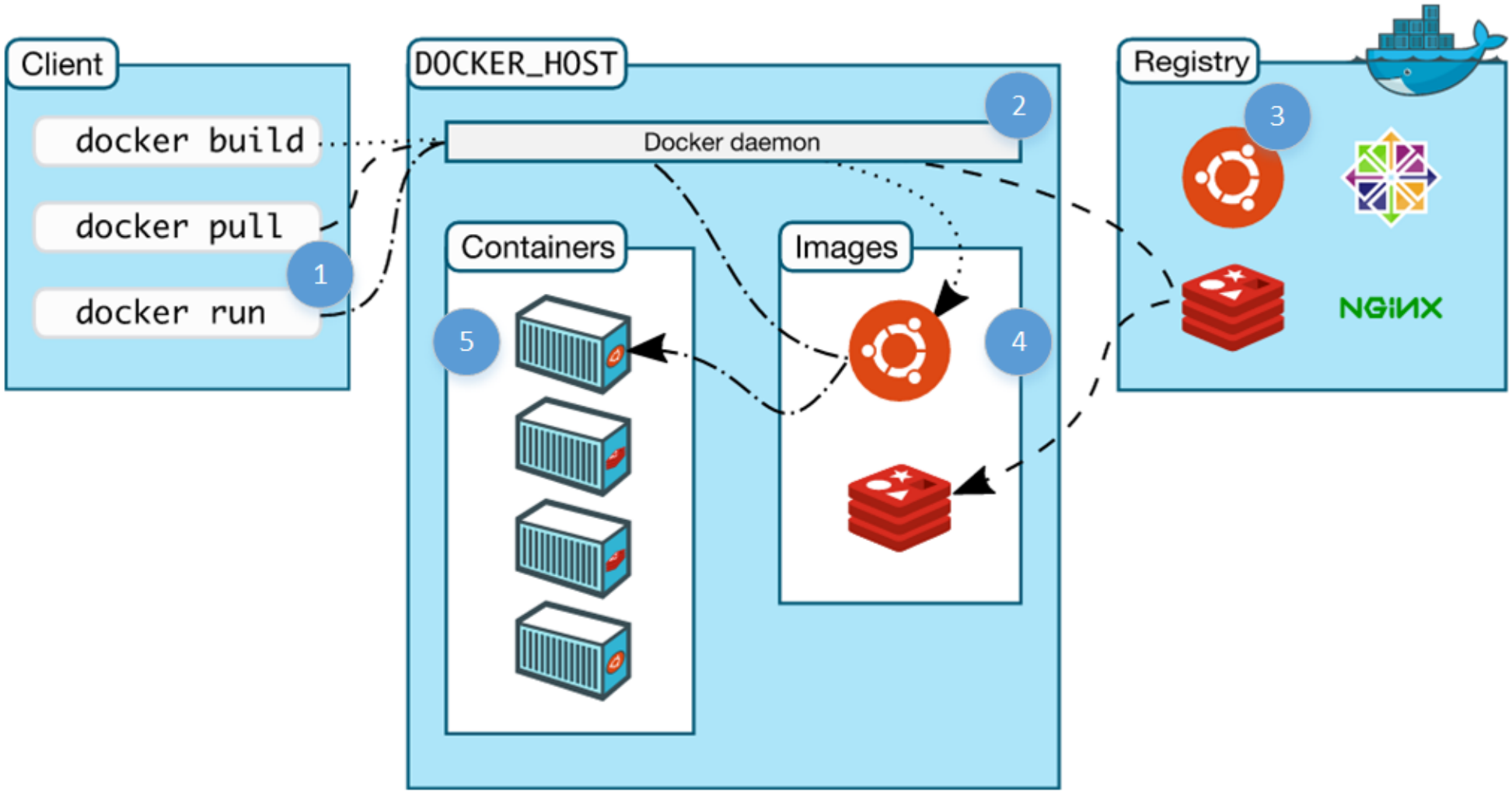
Example: Set up a MySql Database

```
docker run --name db --detach --env MYSQL_ROOT_PASSWORD=123 --publish 4407: 3306 mysql:latest
```

- This creates a container named “db”
- The container runs in the background (--detach)
- We pass the environment variable MYSQL_ROOT_PASSWORD to the container.
- The container maps the host’s port 4407 to its internal port 3306 (--publish)
  - 3306 is the default MySQL port.
- We run the image mysql:latest

The command returns with a UID for our image.

What Happens (1/2)?



What Happens (2/2)?

- The client command contacts the Docker daemon on your local host.
- Docker daemon looks for an image named “mysql-latest”.
  - Your local repository
  - DockerHub
- If necessary, the image is copied to your local repository.
- The image is used to create an running container instance.

What Happens if I Run the Command Again

- Give the container a different name (“db2”)
- Change the --publish external port to use something besides 4407.
- You’ll get another MySQL DB identical to the first.
- Do it again. You can quickly bring up dozens or hundreds of MySQL DBs on your laptop.

What's Next?

- You can now connect to your image and run MySQL startup scripts through an interactive shell.
  - docker exec -it db /bin/bash: you get a command prompt
  - Remember: "db" is the name of the instance.
- But this is not what we want to do
  - Why?

Make Your Own Images

- Use the “docker build” command to create your own images.
- Use a *Dockerfile* to specify exactly how you want your image to be created.
  - See next slide
- Save your image to a repository after you build it with “docker push”
  - You don’t need to run “docker build” every time
  - See “Docker Hub” in a couple of slides.

```
# A basic apache server. To use either add or bind mount content under /var/www
FROM ubuntu:12.04
MAINTAINER Kimbro Staken version: 0.1

RUN apt-get update && apt-get install -y apache2 && apt-get clean && rm -rf
/var/lib/apt/lists/*

ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2

ADD ./index.html /var/www/html/

EXPOSE 80

CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]
```

https://github.com/kstaken/dockerfile-examples

Keyword	Description
FROM	The base image to use
RUN	Execute these commands on the base image and make a new image.
ENV	Set image-wide environment variables; these can also be used by other elements of the Dockerfile
CMD	Executes the specified command. Use this to start services.
ADD	Copies all specified files and directories from the host to the image.
EXPOSE	Specifies the ports that the container will listen to

Some Dockerfile commands

Docker Hub

- Docker Hub is a repository for Docker images.
- You interact with it a lot like GitHub
  - *docker push* command pushes a local image to Docker Hub
  - Ex: *docker push yourUserName/gateway-service-X* will push a local image named "yourUserName/gateway-service-X" to Docker Hub.
  - You need to use Docker tags and namespaces to do this correctly.
- You can check out your own images anywhere
  - Run *docker run yourUserName/gateway-service-X* on Travis-CI, for example
- Use Docker Hub to also get common images from trusted providers
  - "docker search"
- You can connect your images to GitHub repos to trigger automatic builds if you have a build file.

Make Multiple Containers

- You can now create another container to run a Web server and connect to your DB.
- Use your favorite technology
  - Node.js, Tomcat + JSP, PHP, etc.
- You could also choose to run both the web application and the DB in the same container.
  - Which is better?

Connecting Containers

- Use linking to allow containers to communicate with each other securely without exposing ports.
- A linked container shares its environment variables

```
$ docker run -d --name db training/postgres
```

Link: the db doesn't need to expose a port #

```
$ docker run -d -P --name web --link db training/webapp python app.py
```

Composing Containers

- Docker Compose is a tool for defining and running multi-container Docker applications.
- Docker Compose lets you specify in a single YAML file how to create all the containers you need on a single host.
- *docker-compose build:* creates instances for all the images specified in docker-compose.yml

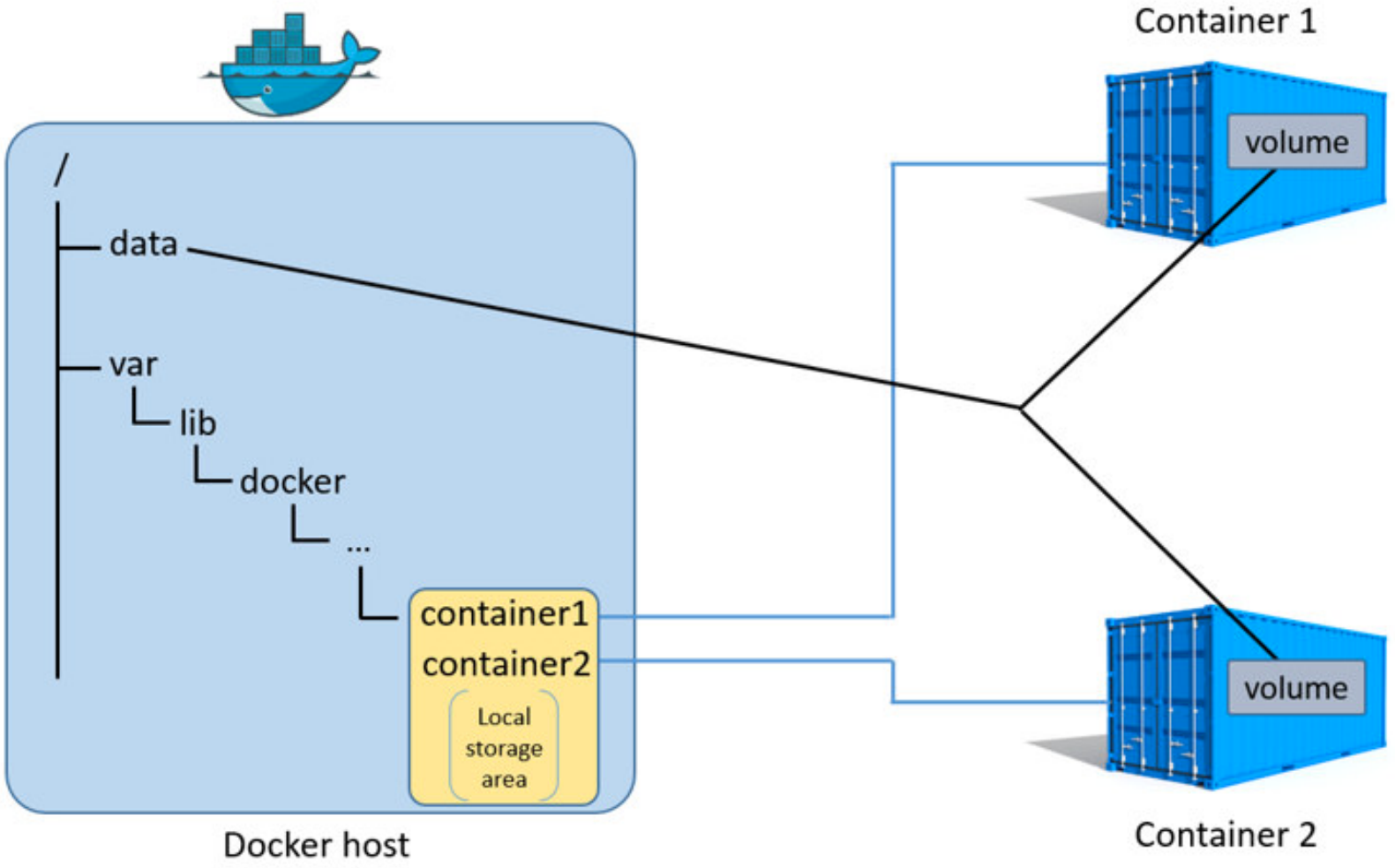
```
version: '2'
services:
  users-service:
    build: ./users-service
    ports:
     - "8123:8123"
    depends_on:
     - db
    environment:
     - DATABASE_HOST=db
  db:
    build: ./test-database
```

Example docker-compose.yml

- This example creates 2 services
  - users-service
  - db
- *build:* this tells Docker Compose where to find the Dockerfile.
- Note users-service depends on db
- *docker-compose up* will start all the services.

Data Volumes and the Storage Driver

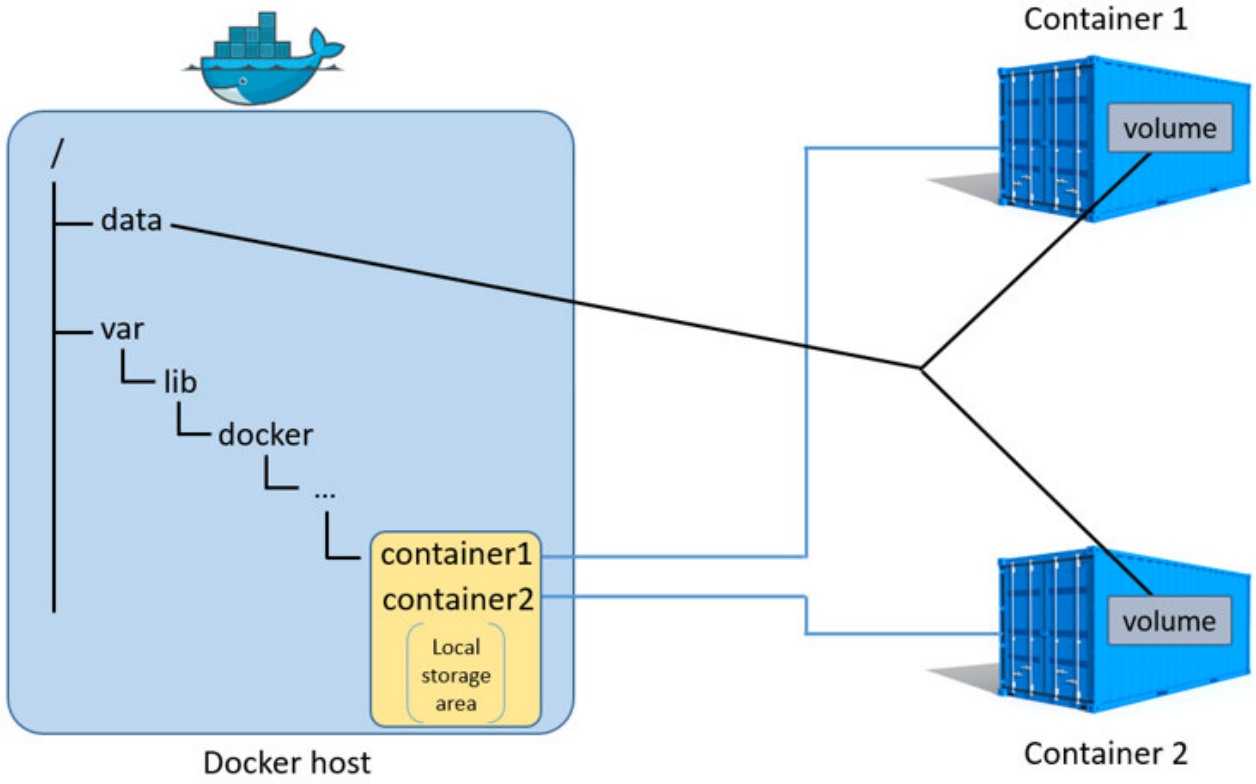
- When a container is deleted, any data written to the container that is not stored in a *data volume* is deleted along with the container.
- A data volume is a directory or file in the Docker host's filesystem that is mounted directly into a container.



Docker volumes allow containers to mount the host's file system.

```
$ docker run -d -P --name web -v /data:/data training/webapp python app.py
```

This will mount the host's /data directory to the /data directory in the container.



Container Overboard!

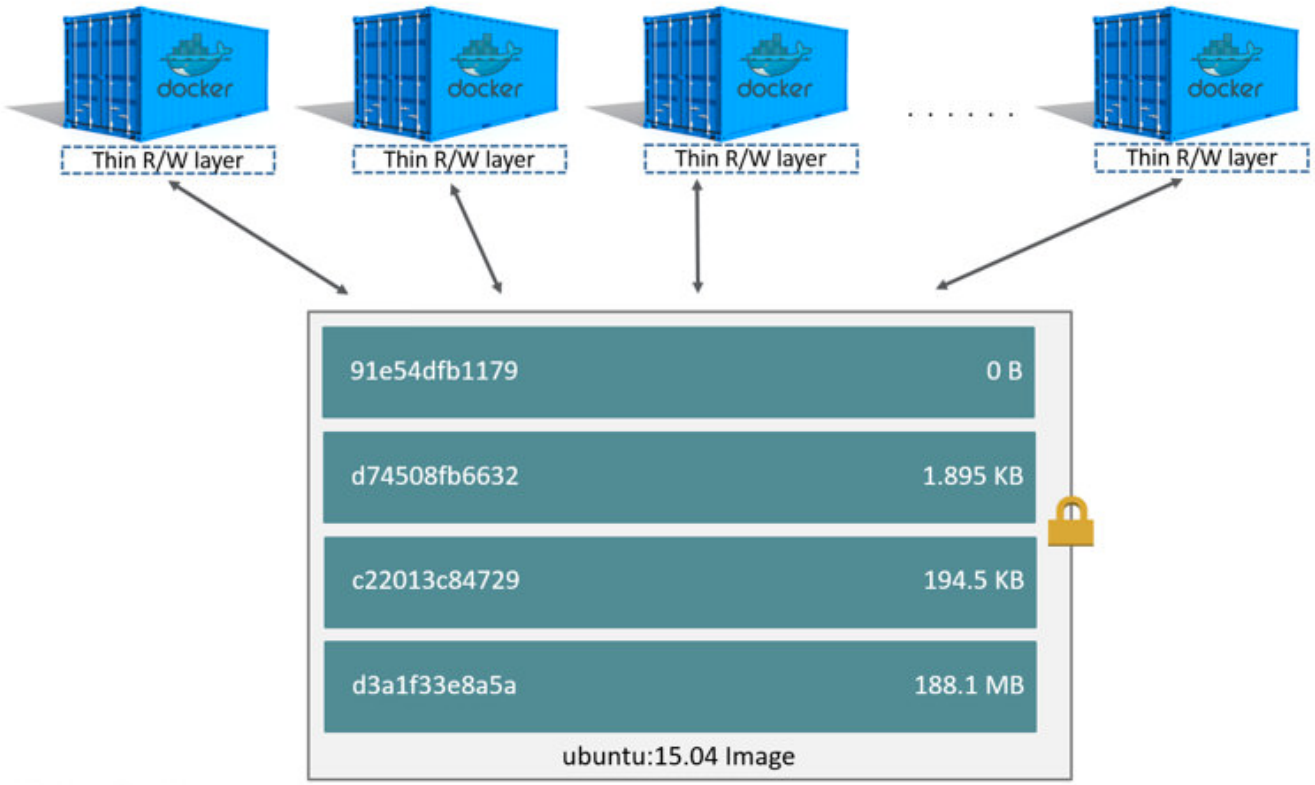
- Mounting the host's file system should be done with care.
- Containers owned by different users can access each other's data.
- Accidentally giving the container too many privileges on the host's file system.
- State gets bound to a specific host.
- This is where VMs can help

Docker Big Picture

Some slides courtesy of http://www.slideshare.net/dotCloud/docker-intro-november

Docker

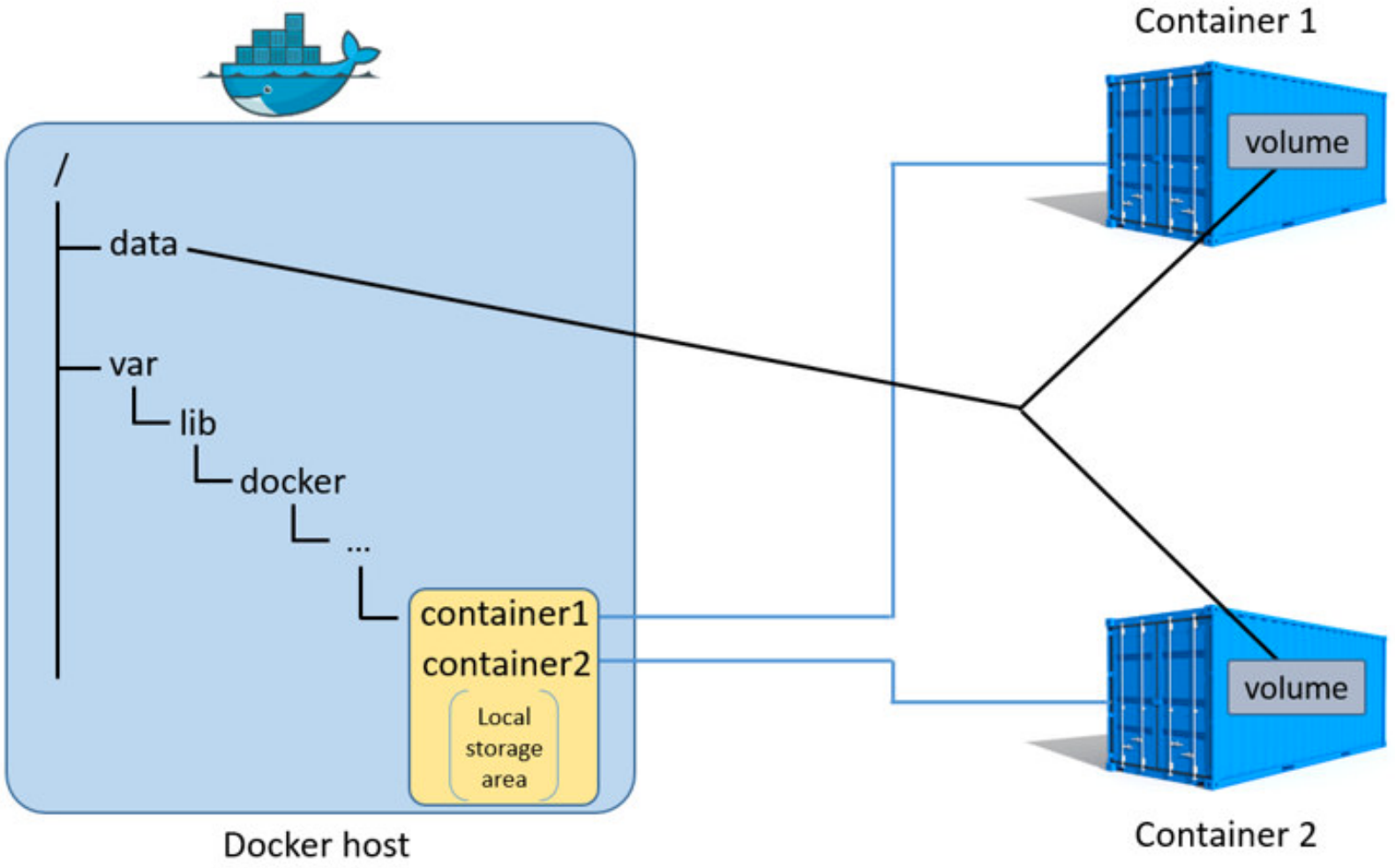
- Docker replaces the need for DIY containers
- Docker uses LXC containerization + AuFS
  - AuFS lets Docker containers share common files
  - AuFS also allows you to have efficient version control of container images



Because each container has its own thin writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state.

Data Volumes and the Storage Driver

- When a container is deleted, any data written to the container that is not stored in a *data volume* is deleted along with the container.
- A data volume is a directory or file in the Docker host's filesystem that is mounted directly into a container.



Docker volumes allow containers to mount the host's file system.

Docker Goals

- Package an application and all its dependencies in a single container.
- Shield applications from host OS dependencies
- Shield applications from other applications' conflicting dependencies.
- Provide standard docker images for common applications.
- Allow you to assemble containers into composite applications

Docker and Microservices

- You can pack a lot of docker containers into a host computer without adding a lot of virtualization overhead.
- Sounds like a good fit for microservices.
  - One microservice per container

The Challenge

Multiplicity of Stacks

Static website

nginx 1.5 + modsecurity + openssl + bootstrap 2

User DB

postgresql + pgv8 + v8

Queue

Redis + redis-sentinel

Analytics DB

hadoop + hive + thrift + OpenJDK

Background workers

Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs

Web frontend

Ruby + Rails + sass + Unicorn

API endpoint

Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client

Do services and apps interact appropriately?

Multiplicity of hardware environments

Development VM

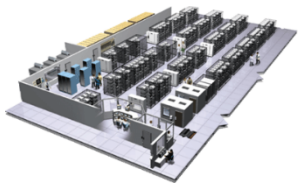
QA server

Customer Data Center

Public Cloud

Disaster recovery

Production Servers



Production Cluster

Contributor's laptop

Can I migrate smoothly and quickly?

The Matrix From Hell

	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

Cargo Transport Pre-1960

Multiplicity of Goods



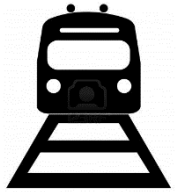


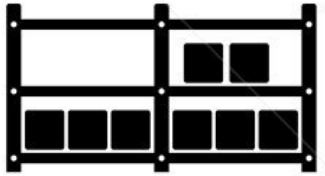


Do I worry about how goods interact (e.g. coffee beans next to spices)

Multipilicity of methods for transporting/storing

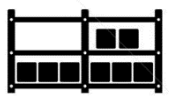





Can I transport quickly and smoothly (e.g. from boat to train to truck)

A matrix from hell

	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

It is a combinatorial problem

Combinatorial problems can be solved in a classic fashion

Solution: Intermodal Shipping Container

Multiplicity of Goods



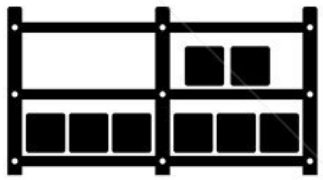


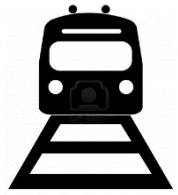
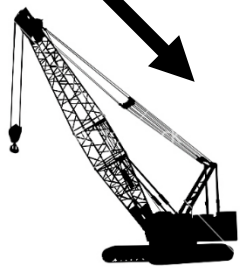

Do I worry about how goods interact (e.g. coffee beans next to spices)

A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.

…in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another

Multiplicity of methods for transporting/storing

Can I transport quickly and smoothly (e.g. from boat to train to truck)

Docker is a shipping container system for code

Multiplicity of Stacks

Static website

User DB

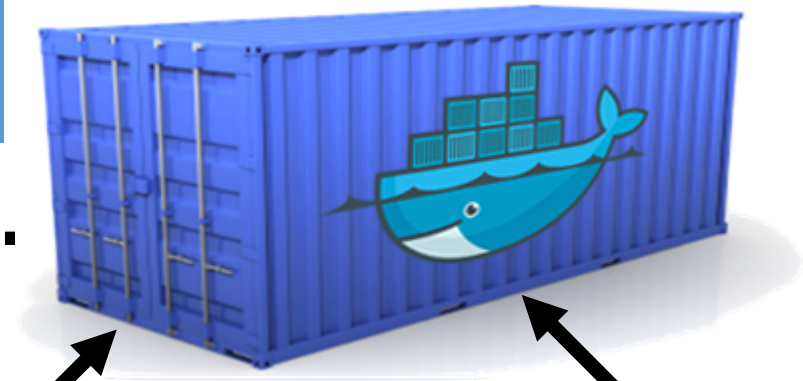
Web frontend

Queue

Analytics DB

Do services and apps interact appropriately?

An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container…



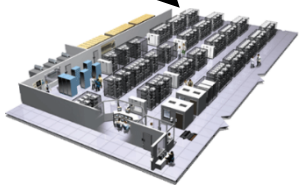
…that can be manipulated using standard operations and run consistently on virtually any hardware platform

Multiplicity of hardware environments

Development VM

QA server

Customer Data Center



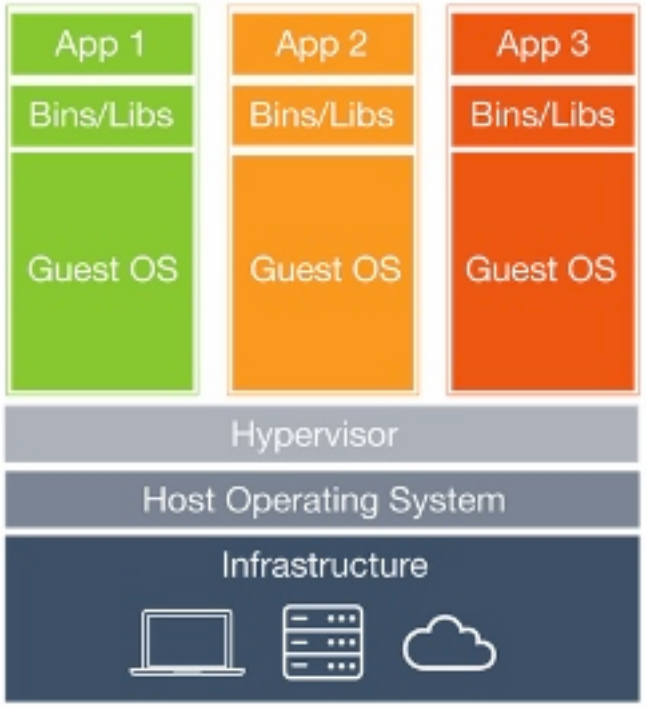
Public Cloud

Production Cluster

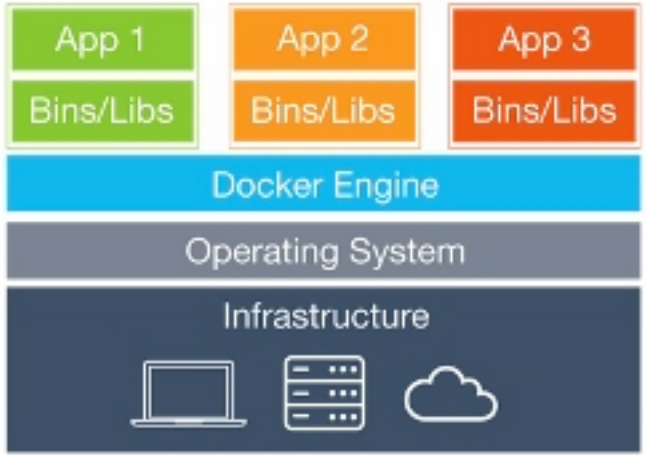
Contributor's laptop

Can I migrate smoothly and quickly

Containers vs. Virtual Machines



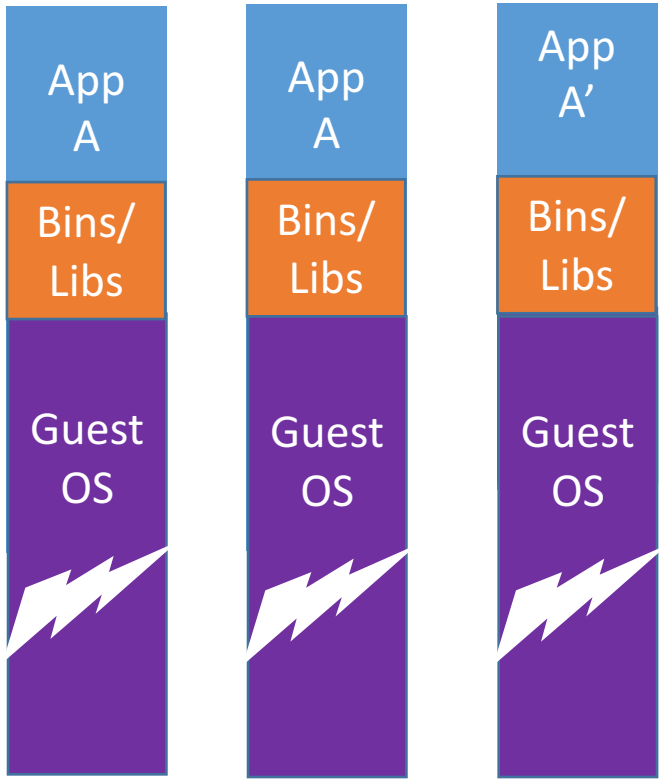
Virtual Machines



Containers

Why are Docker containers lightweight?

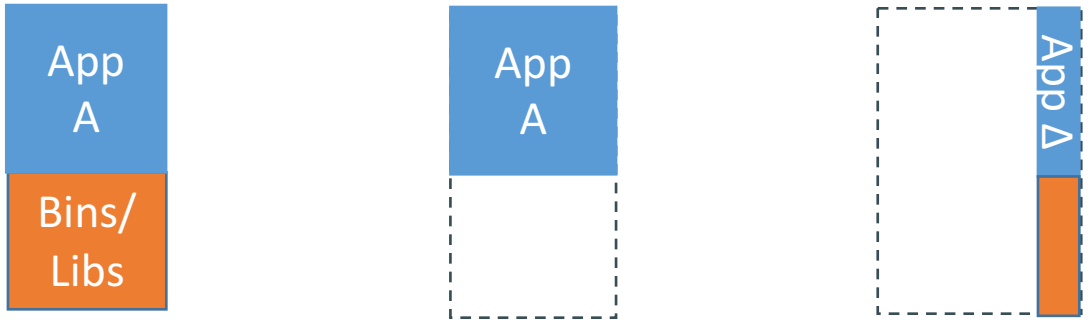
VMs



VMs

Every app, every copy of an app, and every slight modification of the app requires a new virtual server

Containers



Original App

(No OS to take up space, resources, or require restart)

LXC enables this

Copy of App

No OS. Can Share bins/libs

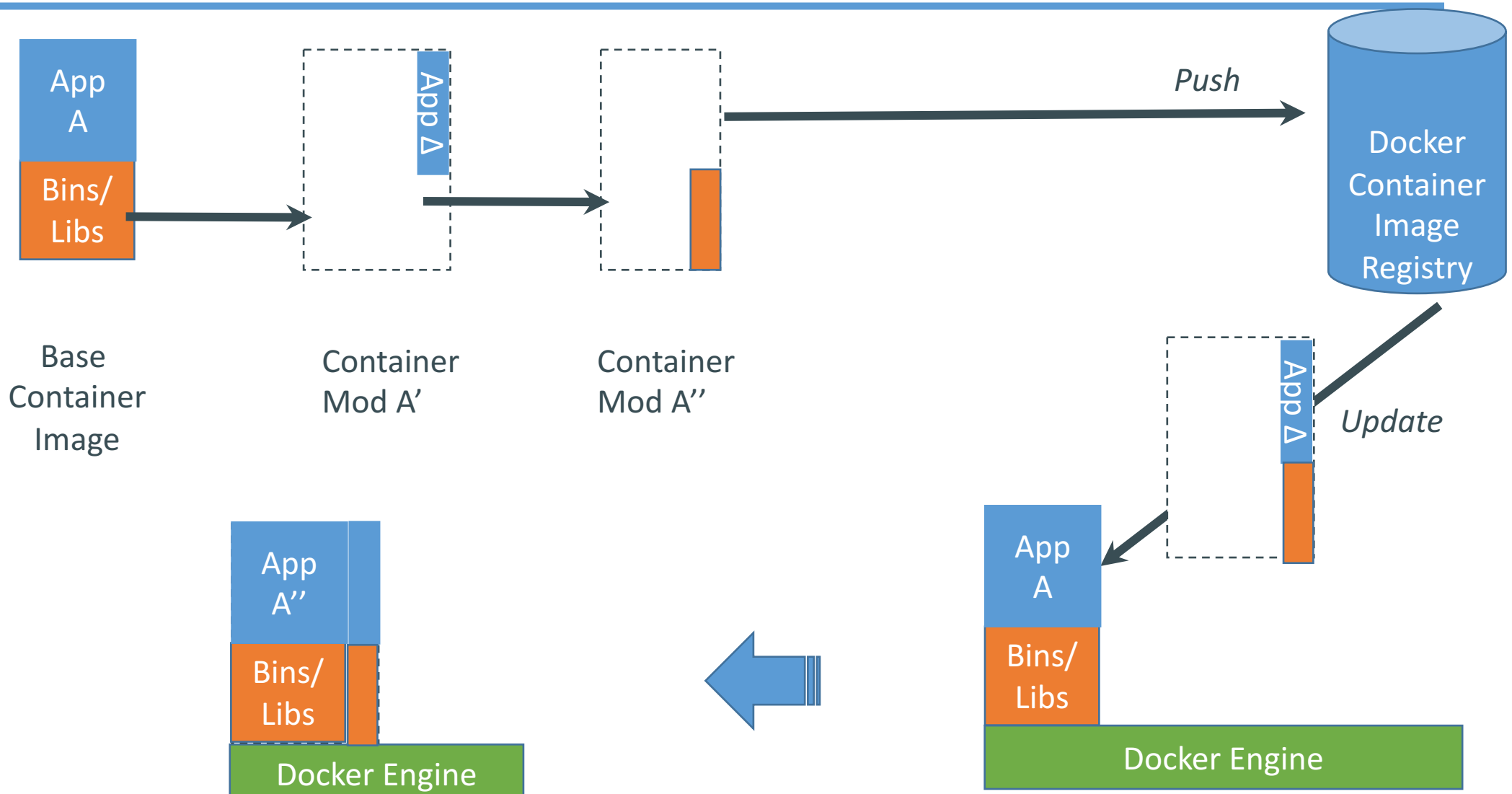
AuFS enables these

Modified App

Copy on write capabilities allow us to only save the diffs Between container A and container A'

docker

Changes and Updates



Host is now running A''

Host running A wants to upgrade to A''.
Requests update. Gets only diffs

docker

Docker: Infrastructure as Code

- In short, Docker lets you define in script files everything about each of your microservices.
- Combine this with CI/CD systems to deploy EACH microservice.
  - Your development to test to production environments should be identical and reproducible.
  - Testing and production deployments for each service should be infinitely clone-able.
- This is not elasticity, but it is a prerequisite.
- Docker and other containers have much less overhead