COEN 241
Introduction to Cloud Computing

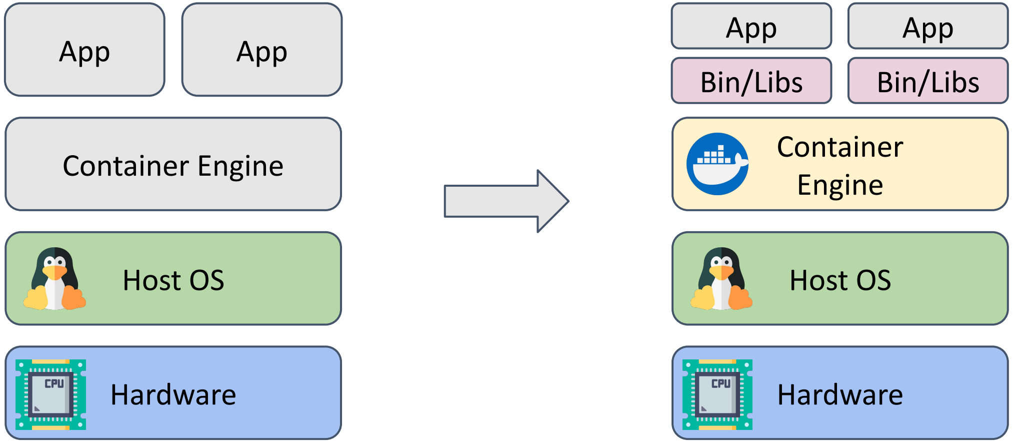
Lecture 7 - Containers II & Orchestration

Lecture 6 Recap

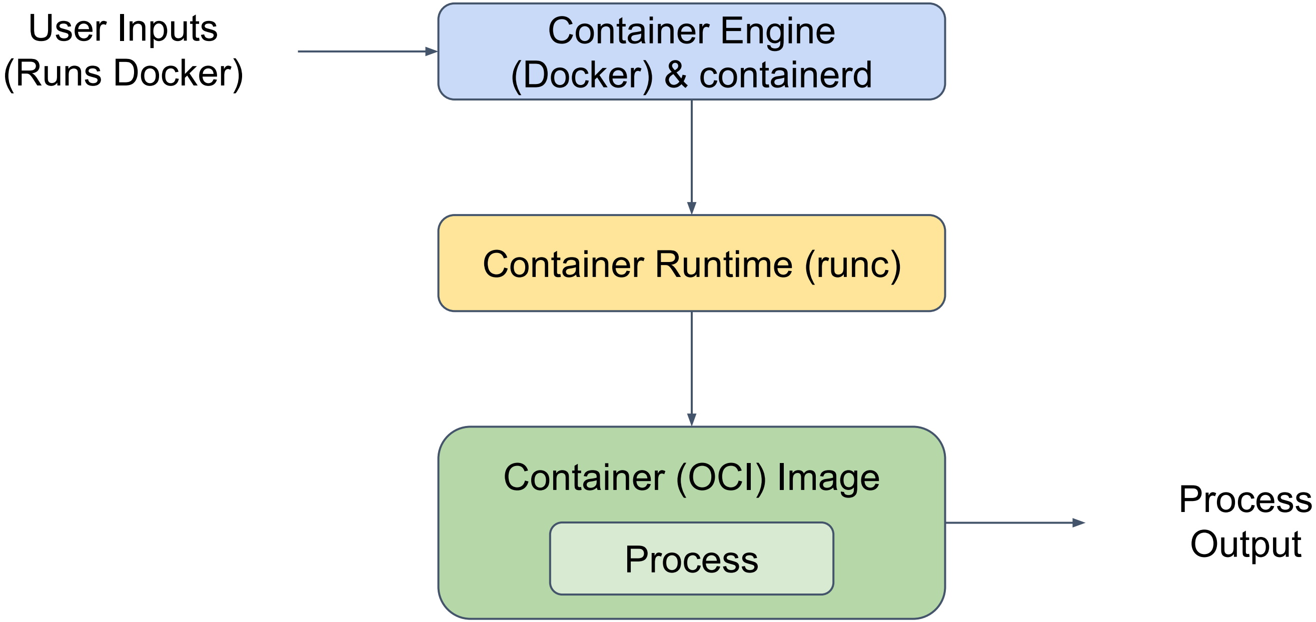
- Containers
  - What is it?
  - Pros and Cons
- Docker
  - Docker architecture
  - Docker Demo

What is a Container?

- A container is a standard unit of software that packages up **code** and **all its dependencies** so the application runs quickly and reliably from one computing environment to another



Container Workflow



User Inputs
(Runs Docker)

Container Engine
(Docker) & containerd

Container Runtime (runc)

Container (OCI) Image

Process

Process
Output

What is Docker?

- **Docker** is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers

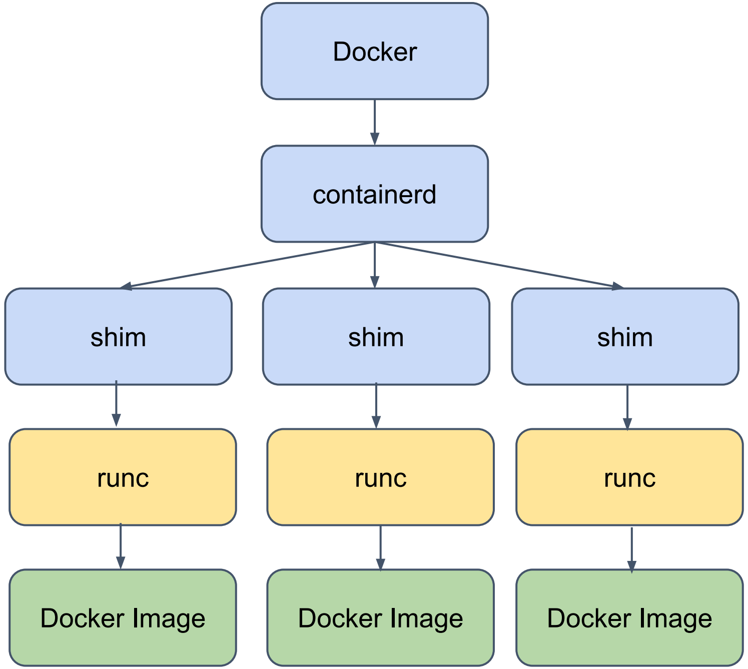
- Founded in 2009, Released in 2013

- It is a software platform consisting of:
    - Docker Engine
    - Docker Hub
    - Docker Trusted Registry
    - Docker Machine
    - Docker Compose
    - Docker for Windows/Mac
    - Docker Datacenter

http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

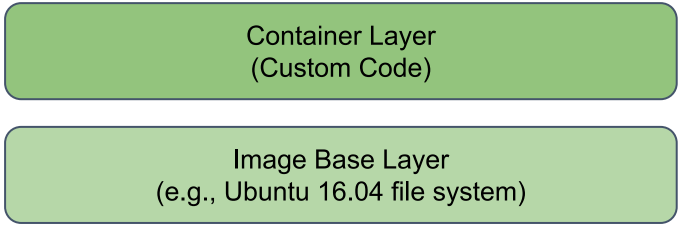
Docker Architecture

- **containerd**: a daemon process that manages and runs containers
- **“shim”**: facilitate communication and integration
  - Enables daemon-less container
- **runc**: low-level container runtime
  - actually creates and runs containers



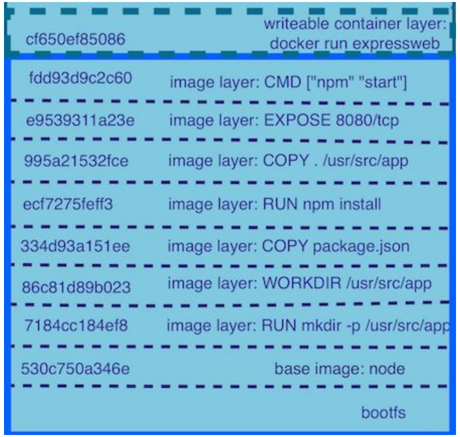
Docker Container Image

- A container originates from a base image layer, including a base file system (and applications)
- When you launch a container, another layer is created on top of the base image layer
- You can stack more container layers!



Container Layer
(Custom Code)

Image Base Layer
(e.g., Ubuntu 16.04 file system)



cf650ef85086	writeable container layer: docker run expressweb
fdd93d9c2c60	image layer: CMD ["npm" "start"]
e9539311a23e	image layer: EXPOSE 8080/tcp
995a21532fce	image layer: COPY . /usr/src/app
ecf7275feff3	image layer: RUN npm install
334d93a151ee	image layer: COPY package.json
86c81d89b023	image layer: WORKDIR /usr/src/app
7184cc184ef8	image layer: RUN mkdir -p /usr/src/app
530c750a346e	base image: node
	bootfs

Read/Write Permissions

- Only the top level (container) layer has both read/write permission
- All base layers are read only
- **Merged view via file systems like AUFS**

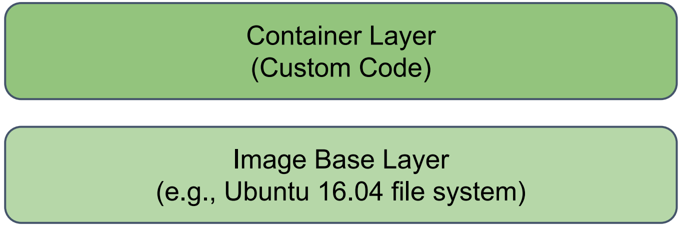
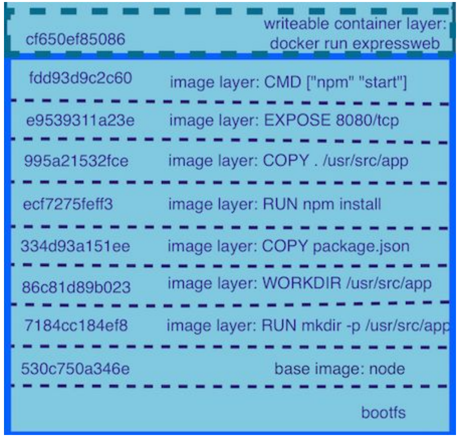

Container Layer
(Custom Code)

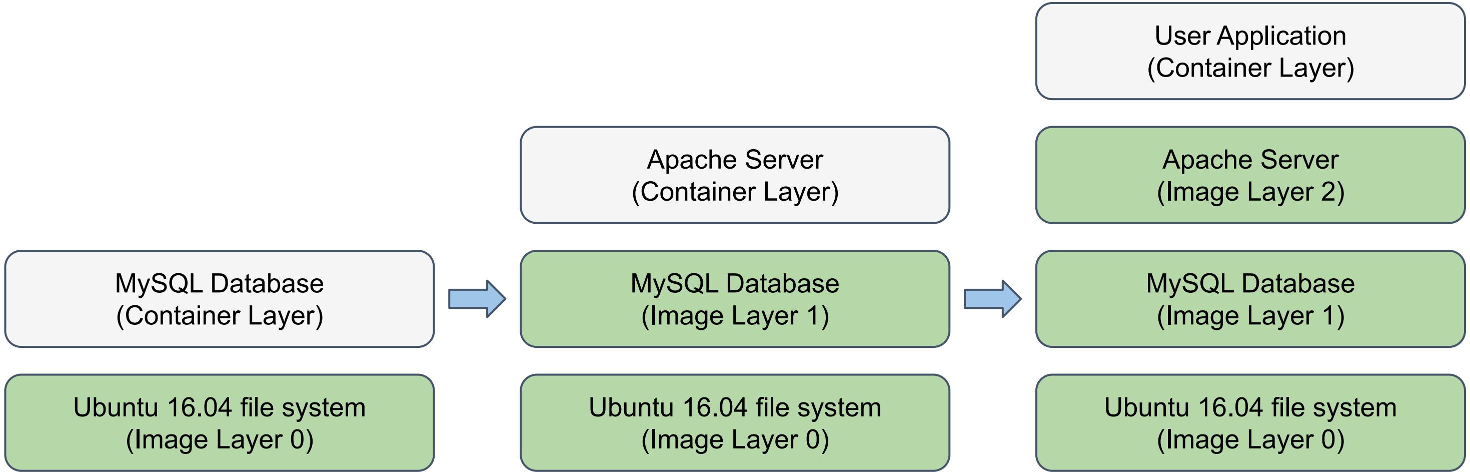
Image Base Layer
(e.g., Ubuntu 16.04 file system)



cf650ef85086	writeable container layer: docker run expressweb
fdd93d9c2c60	image layer: CMD ["npm" "start"]
e9539311a23e	image layer: EXPOSE 8080/tcp
995a21532fce	image layer: COPY . /usr/src/app
ecf7275feff3	image layer: RUN npm install
334d93a151ee	image layer: COPY package.json
86c81d89b023	image layer: WORKDIR /usr/src/app
7184cc184ef8	image layer: RUN mkdir -p /usr/src/app
530c750a346e	base image: node
	bootfs

https://docs.docker.com/storage/storagedriver/aufs-driver/#how-the-aufs-storage-driver-works

Stackable Container Images



Agenda for Today

- DockerFile
- Kata Container
- Orchestration in Cloud
- Infrastructure as Code
- CoreOS
- Readings
  - Recommended: None
  - Optional:
    - https://www.youtube.com/watch?v=4gmLXyMeYWI
    - https://www.stackhpc.com/kata-io-1.html
    - https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9198653
    - https://www.techrepublic.com/article/simplifying-the-mystery-when-to-use-docker-docker-compose-and-kubernetes/

DockerFile

What is a DockerFile?

- Dockerfile is used to automate the Docker image creation.
- Docker builds images by reading instructions from the Dockerfile.

Available Commands in a Dockerfile

- Comments
- FROM
- CMD
- ENTRYPOINT
- WORKDIR
- ENV
- COPY
- LABEL
- RUN
- ADD
- .dockerignore
- ARG
- EXPOSE
- USER
- VOLUME

Example Dockerfile

```
FROM python:alpine

LABEL "about"="This file is just an example to demonstrate the LABEL"
ENV workdirectory /usr/python
WORKDIR $workdirectory
WORKDIR app
COPY requirements.txt .
RUN pip3 install -r requirements.txt
RUN apk update && apk add bash
# command executable and version
CMD ["--version"]
ENTRYPOINT ["python"]
```

Dockerfile Demo

```
// Build the image
docker build -t dockerfile -f Dockerfile .

// Inspect the image
docker image inspect dockerfile

// Run the container
docker run -it dockerfile

// Run bash given the container
docker run -it --entrypoint /bin/sh dockerfile

// See the current working directory
pwd
```

Kata Containers

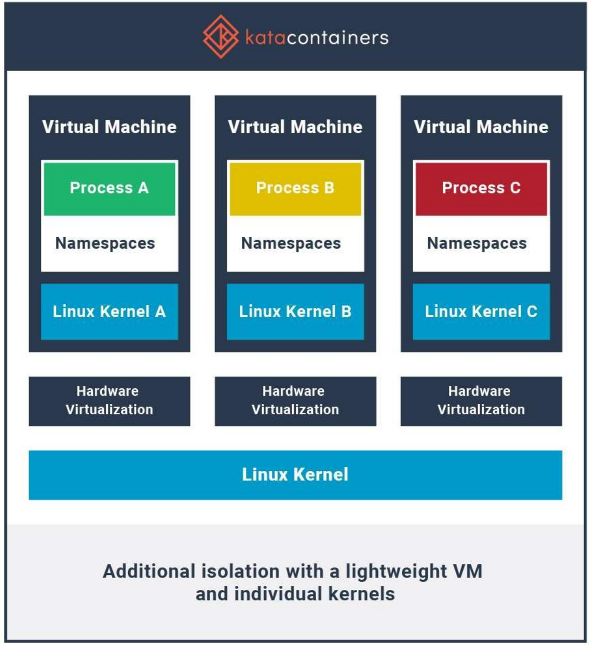
What is Kata Container?

- Released in 2017
  - From the merge of Intel’s Clear Containers and Hyper’s runV
  - “Wraps” containers into dedicated virtual machines
  - OCI runtime implementation can be plugged into the container engine
    - Docker
  - Can consume existing container images
- Kata is a container runtime
  - Can still be coupled with other Docker platforms

kata containers

https://katacontainers.io/

Kata Architecture



katacontainers

Virtual Machine

Process A

Namespaces

Linux Kernel A

Virtual Machine

Process B

Namespaces

Linux Kernel B

Virtual Machine

Process C

Namespaces

Linux Kernel C

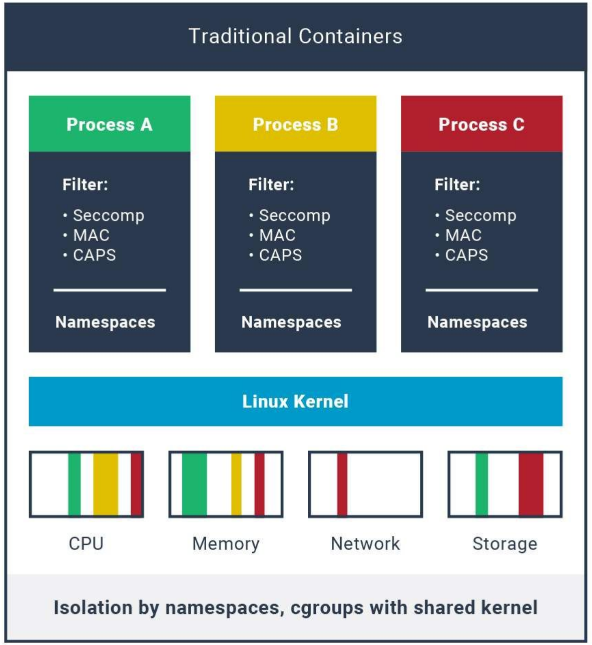
Hardware Virtualization

Hardware Virtualization

Hardware Virtualization

Linux Kernel

Additional isolation with a lightweight VM
and individual kernels



Traditional Containers

Process A

Filter:

- Seccomp
- MAC
- CAPS

Namespaces

Process B

Filter:

- Seccomp
- MAC
- CAPS

Namespaces

Process C

Filter:

- Seccomp
- MAC
- CAPS

Namespaces

Linux Kernel

CPU

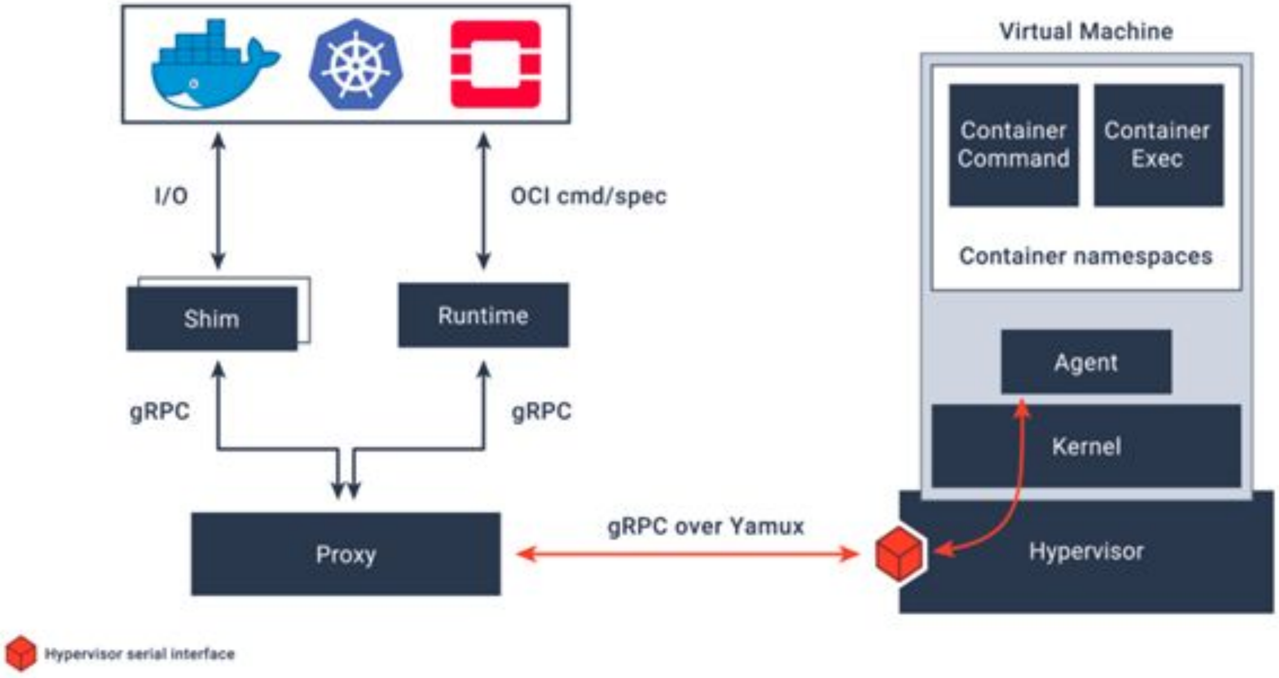
Memory

Network

Storage

Isolation by namespaces, cgroups with shared kernel

Kata Workflow



Pros & Cons of Kata Container

- Pros
  - It’s possible to run containers inside of virtual machines
    - Clouds do this all the time!
  - Each container also gets its own I/O, memory access, and other low-level resources, without having to share them.
  - Introduces another layer of protection provided by the hypervisor
  - Can use security features provided by hardware-level virtualization
  - Lower weight than VMs
- Cons
  - Slower (Obviously)
  - Hypervisor security bugs as well

https://katacontainers.io/

Orchestration

Automating the cloud

Orchestration

- Definition: Automatic management of computer systems
  - Deployment and configuration
  - Interconnection and coordination
  - Monitoring
    - Can also configure to hook up monitoring for management
    - E.g., Deploy more resources as load increases

- Growing set of very good (open-source) solutions:
  - **Machine focused**: e.g., Puppet, Terraform, Ansible, Salt, Chef…
  - **Cloud-based:** e.g., AWS CloudFormation, Terraform
  - **Container clusters:** e.g., Kubernetes

Automation vs Orchestration

- **Automation:** completing a **single task** or function without human intervention

- **Orchestration:** Managing a large-scale virtual environment or network by orchestrating the scheduling and integration of automated tasks between complex distributed systems and services
    - Simplifies interconnected workloads, repeatable processes, and operations.

- To simplify, **Automation** refers to a single task vs **Orchestration** arranges multiple tasks to optimize a workflow

Orchestration in the Cloud

- Need to be able to both **provision** and **configure** cloud resources
  - **Provision:** Setting up VMs from bare-metal machines
  - **Configure:** Install / manage software and other connected services
    - Usually done via APIs calls
    - Example services: storage, virtual networking, load balancing, firewall security
- Various design decisions need to be made for orchestration
  - Should orchestration system build and deploy the VMs?
    - What is the frequency of software change on the VM image?
    - e.g., making app-specific VM vs. use a base Linux VM and configure
  - How to bootstrap various resource for remote access?
    - From where and how without compromising security
    - What should be bootstrapped? VMs only? Also networks?

When to use Orchestration?

- Generally required for a larger system with many automated actions

- Example: Launching a new service which requires:
  - Provisioning hundreds of servers
  - Testing the service
  - Each servers must be provisioned with the correct version of OS and software
  - Addition of new servers during heavy load

- Orchestration tools provide templates to achieve the above steps

- Also provides monitoring, backup and security services for repeatability

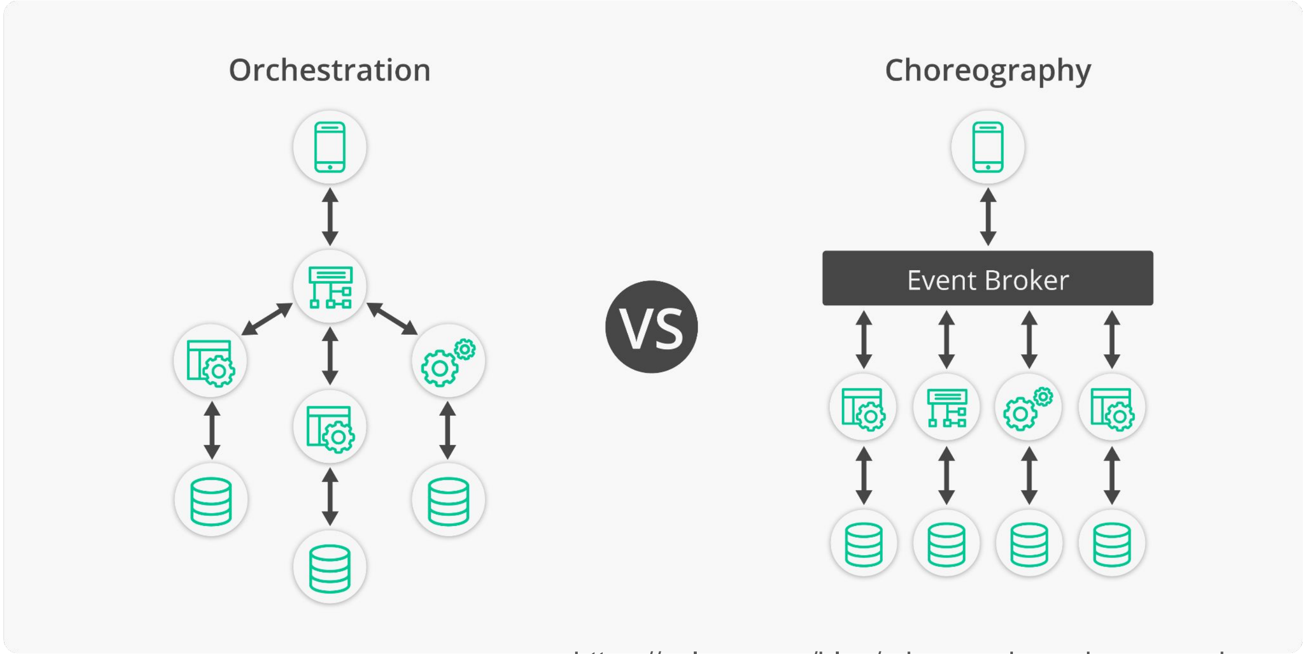
Multi-Cloud Orchestration

- Most large organizations use many cloud providers
  - Protect against vendor lock-in
  - Achieve resilience to failures within one cloud provider
  - Consequence of non-coordinated decisions in large organizations
- Services to manage multiple + hybrid clouds are emerging
  - E.g., Scalr applies policy controls across all cloud resources
- Some concepts are common across cloud providers: VMs & containers
- Specifics of security and network configuration will be different:
  - E.g., Amazon IAM (Identity & Access Management)

Choreography

- Microservices work independently but coordinate with each other using cues or events
  - E.g., via protocols and rules between specific services
  - Could describe the logic on each service about interactions
- Relatively a newer term (although the technologies existed)

Orchestration vs. Choreography



https://solace.com/blog/microservices-choreography-vs-orchestration/

Infrastructure As Code

In Detail

Recap: Infrastructure As Code (IAC)

- IAC covers configuration management & provisioning
  - Also involves avoiding hardware configuration (e.g., switches)
  - Goal: complete automation from machine readable files
  - For cloud, cluster of servers or single server management
- Cost reduction
  - Focus on business needs rather than device management
  - Continuous integration pipelines often integrated
- Now a requirement for most businesses running on cloud
- Declarative & Imperative IAC

Declarative Configuration Management

- Declarative tools specify the desired target state
  - E.g., Can I have a coffee on my desk at 9AM on Monday morning?
- The means to reach target state is up to the configurations
  - Can take corrective action to react to drift in machine's state
- State specification will be a domain specific language (DSL)
- Some example FOSS systems with large user communities
  - Puppet, Terraform, SaltStack

Declarative Configuration Management Example

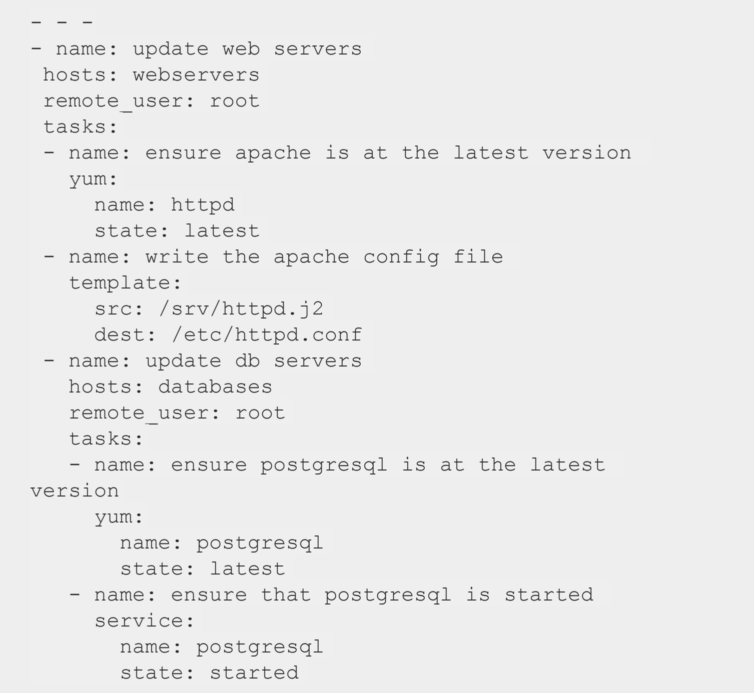
```
terraform {
 version = "0.11.13"
}
provider "aws" {
 region = "us-east-2"
}
resource "aws_s3_bucket" "your_new_bucket" {
 bucket = "my-first-website-cloud-native-website"
 acl    = "public-read"
 website {
   index_document = "index.html"
 }
}
```

- Terraform developed by Hashicorp
  - Used by companies like Zendesk
- Declaring target state without knowing how it is done

Imperative Configuration Management

- Also called ‘procedural’, i.e., specifying steps to run:
    - Usually written in chunks of code in configuration system authors’ favorite PL
- Some example systems:
    - Ansible (Py), Chef (Ruby), Saltstack
- Can write imperative code to have declarative effect

Imperative Configuration Management Example



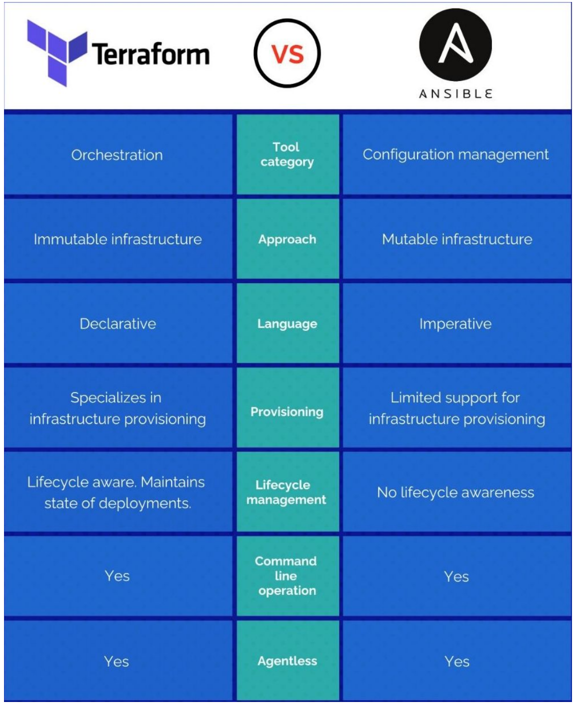
```
- - -
- name: update web servers
 hosts: webservers
 remote_user: root
 tasks:
 - name: ensure apache is at the latest version
   yum:
     name: httpd
     state: latest
 - name: write the apache config file
   template:
     src: /srv/httpd.j2
     dest: /etc/httpd.conf
 - name: update db servers
   hosts: databases
   remote_user: root
   tasks:
   - name: ensure postgresql is at the latest
version
     yum:
       name: postgresql
       state: latest
   - name: ensure that postgresql is started
     service:
       name: postgresql
       state: started
```

- Ansible developed by Redhat
  - Used by companies like Udemy
- Imperative specifies every single step to reach the final state

Declarative vs Imperative

- Dealing with “Configuration Drift”: Infra changes slowly over time
    - Declarative is easier to adapt, imperative is harder to adapt
- Ease of Repeatability
    - Declarative is easier to repeat, imperative may have different outcome
- Idempotency: Repeated run has no additional effect
    - Declarative is idempotent, imperative is not
- State Management
    - Declarative needs to manage states, Imperative does not

Declarative vs Imperative



Terraform	VS	ANSIBLE
Orchestration	Tool category	Configuration management
Immutable infrastructure	Approach	Mutable infrastructure
Declarative	Language	Imperative
Specializes in infrastructure provisioning	Provisioning	Limited support for infrastructure provisioning
Lifecycle aware. Maintains state of deployments.	Lifecycle management	No lifecycle awareness
Yes	Command line operation	Yes
Yes	Agentless	Yes

When to use Declarative or Imperative

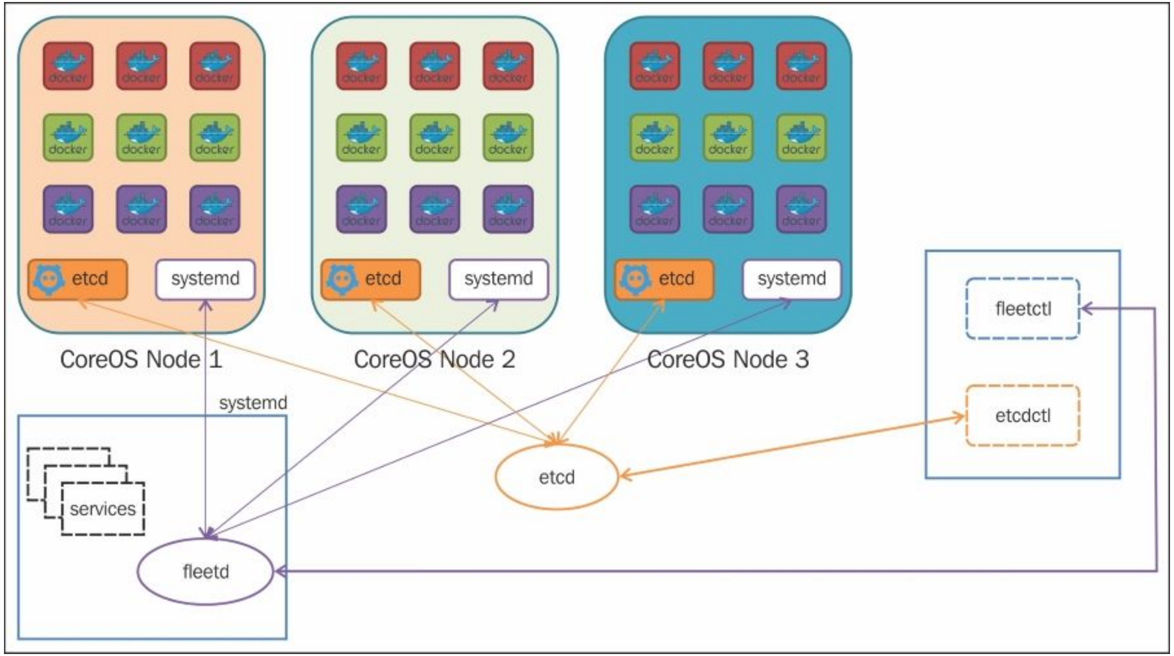
- No correct answer!
- A food for thought
  - If you need quick simple update: Use imperative configuration management.
  - If you are configuring a larger infrastructure that evolves over time:
    Use declarative configuration management
- Still depends heavily on context!

Core OS

Core OS

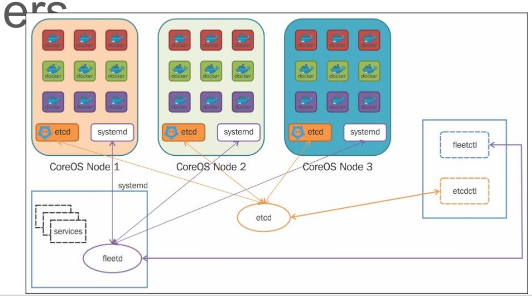
- From both RHEL (Red Hat Enterprise Linux) and Fedora
  - Founded on 2013
- Automatically-updating, minimal operating system for running containerized workloads securely and at scale
  - I.E., Linux distribution intended to run Containers
  - No software package manager: /usr is read-only
  - Can be started using network boot
  - Security updates are applied monolithically
  - Can schedule rolling reboot of cluster machines
- Supports both Docker and rtk containers

Core OS Architecture



Core OS Architecture

- **etcd: Distributed Key-Value store for state management**
  - Super useful!
- fleetd: Distributed runtime scheduler
- systemd: Mechanisms that allow fleetd to execute the runtimes
- Containers: Docker and/or rkt containers



Core OS Advantages

- Core OS is a lightweight Linux distribution
  - Can easily and quickly install into VMs or into cloud
  - Only ~300MBs
  - https://docs.fedoraproject.org/en-US/fedora-coreos/getting-started/
- Easily integrates with public clouds (& QEMU too!)
- Provides distributed / container based operating system
- Adding and removing nodes to it is pretty easy
- Enables High Availability at low cost

Core OS Disadvantages

- Not very popular unfortunately
- Often difficult to configure when network changes
- Have to manage lot of unit files for systemd
- Open Source

Container Orchestration via IAC

- Similar to VMs, Containers also need to be managed via IAC
- Various container orchestration systems are available
    - Docker swarm: Docker built-in simple cluster manager
    - Docker compose: Used to specify multi-container environment
    - Apache Mesos: Supports both container and non-container workloads
    - OpenShift: Container orchestration tool by RedHat
    - **Kubernetes: Will talk about it next lecture**

Agenda for Today

- Dockerfile
- Kata Container
- Orchestration in Cloud
- Infrastructure as Code
- CoreOS
- Readings
  - Recommended: None
  - Optional:
    - https://www.youtube.com/watch?v=4gmLXyMeYWI
    - https://www.stackhpc.com/kata-io-1.html
    - https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9198653
    - https://www.techrepublic.com/article/simplifying-the-mystery-when-to-use-docker-docker-compose-and-kubernetes/

TODOs!

- HW 1
- HW 2 will be out next class
- Midterm coming soon!

Questions?