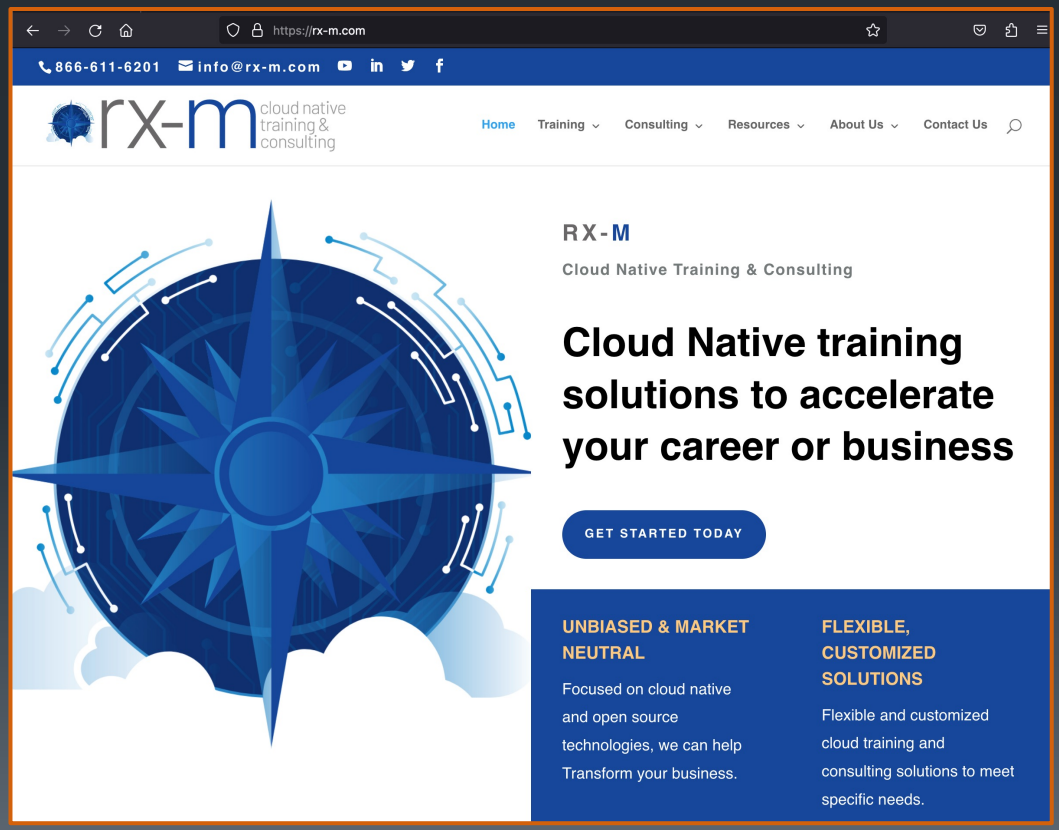
rx-m cloud native training & consulting

Containers and Kubernetes Accelerated

A rapid introduction to the core concepts and features of cloud native systems through an intensive look at container technology and Kubernetes

- Microservice Oriented
  - Microservices Foundation [3 Day]
  - Building Microservices on Kubernetes [3 Day]
  - Building Microservices on AWS/GCP/Azure [3 Day]
  - Building Microservices with Go [3 Day]
  - Building Microservices with Thrift/gRPC [2 Day]
- Container Packaged
  - Docker Foundation [3 Day]
  - Docker Advanced [2 Day]
  - Container Technology [2 Day]
  - Containerd [2 Day]
  - Cri-O [2 Day]
- Dynamically Managed
  - Kubernetes Foundation [2 Day]
  - Kubernetes Advanced [3 Day]
  - Kubernetes CKA boot camp [5 Day]
  - Kubernetes CKAD boot camp [4 Day]
  - Kubernetes Day 2 operations [2 Day]
- Robustly Secured
  - Securing Kubernetes [2 Day]
  - Kubernetes CKS boot camp [5 Day]
  - Secure Development on Kubernetes [2 Day]
  - Zero Trust Architecture [2 Day]
  - SLSA - Securing the Software Supply Chain [3 Day]
  - Monitoring Kubernetes with Prometheus [2 Day]
- Intelligence Enabled
  - Machine Learning Foundation [3 Day]
  - Advanced Topics in Machine Learning [2 Day]
  - MLOps Foundation [2 Day]
  - Python for ML Engineers Foundation [3 Days]
  - ML/AI for Quants [3 Days]
  - LLM Foundation [2 Day]
  - Spark on K8s [2 Day]
  - Machine Learning on K8s [3 Day]
  - Ray Foundation [2 Day]
  - ML/AI on AWS/GCP/Azure [3 Day]

RX-M Cloud Native Advisory, Consulting, and Training



Overview

Day One

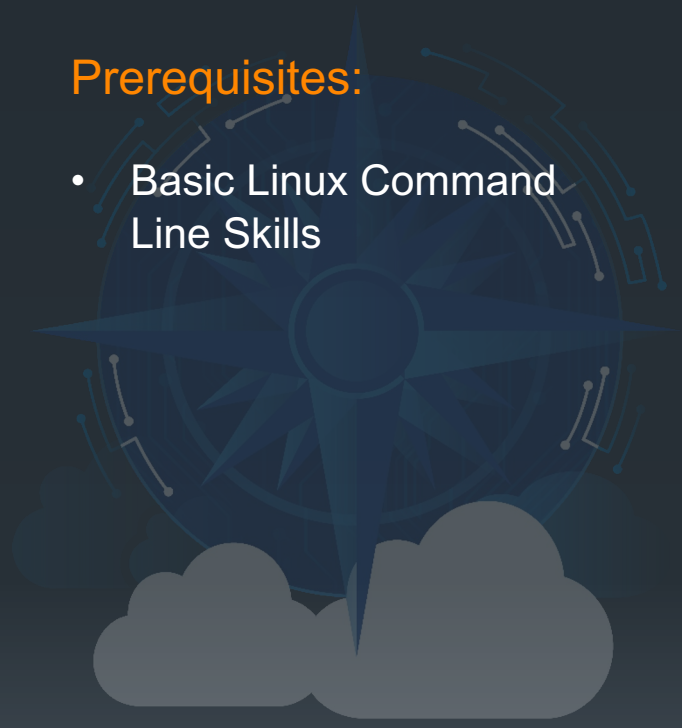
1. Container Overview
2. Image Overview
3. Kubernetes Overview
4. Pod Basics

Day Two

5. Application Configuration
6. Deployments
7. Services
8. Observability Overview

Prerequisites:

- Basic Linux Command Line Skills

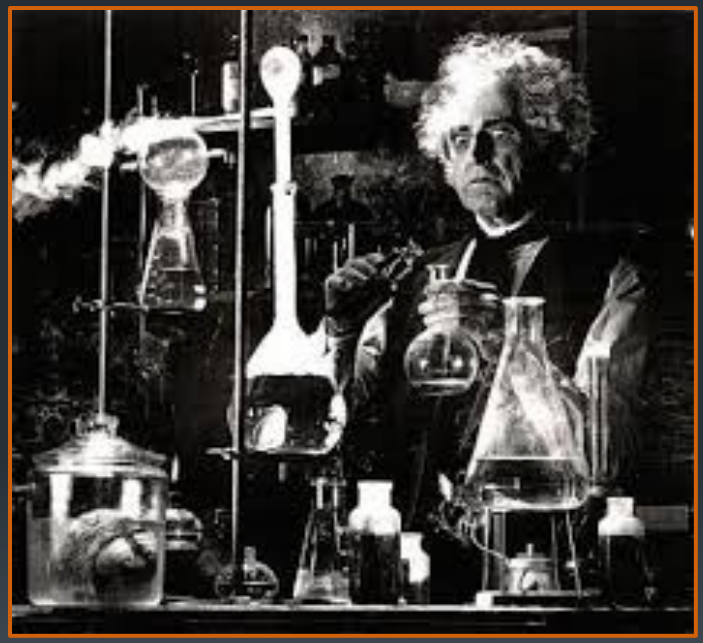


Administrative Info

- Length: 2 Days
- Format: Lecture/Lab/Discussion
- Schedule: 9:00AM – 5:00PM
  15 minute break, AM & PM
  1 hour lunch at noon
  Lab time at the end of each AM and PM session
- Location: Fire exits, Restrooms, Security, other matters
- Attendees: Name/Role/Experience/Goals for the Course
- Sign in: Sign in sheet
- Materials: Links in sign in sheet
- Lab Access: SSH instructions in sign in sheet

Lecture and Lab

- Our Goals in this class are three-fold:
  1. Introduce concepts and ecosystems
     - Covering concepts and where things fit in the world is the primary purpose of the lecture/discussion sessions
     - The instructor will take you on a tour of the museum
       - Like a museum tour, you should interact with the instructor (tour guide), ask questions, discuss
       - Like a museum tour, you will not have time to read the slides during the tour, instead, the instructor will discuss and point out the highlights of the slides (exhibits) which will be waiting for you to read in depth later should you like to dig deeper
  2. Impart practical experience
     - This is a primary purpose of the "walk through" part of the labs
     - Classes rarely have time for complete real-world projects so think of the labs as thought experiments
       - Like hands on exhibits at the museum
  3. Develop problem solving skills
     - This is a primary purpose of the challenge sections in the labs
     - Challenges require you to solve problems on your own using the new skills you have learned in class and in the lab walk throughs
       - Solutions are typically provided so that you can check your work



Day 1

1. Container Overview
2. Image Overview
3. Kubernetes Overview
4. Pod Basics

1: Container Overview

Objectives

- Understand the basic nature of containers
- Explain the container value proposition
- Examine the differences and similarities between Virtual Machines (VMs) and Containers
- Examine Container isolation features (mount, network, UTS, process, IPC)
- Learn how to attach and detach from containers
- Use docker commands to find information about running containers:
  - Logs
  - Stats
  - Top

What is a Container?

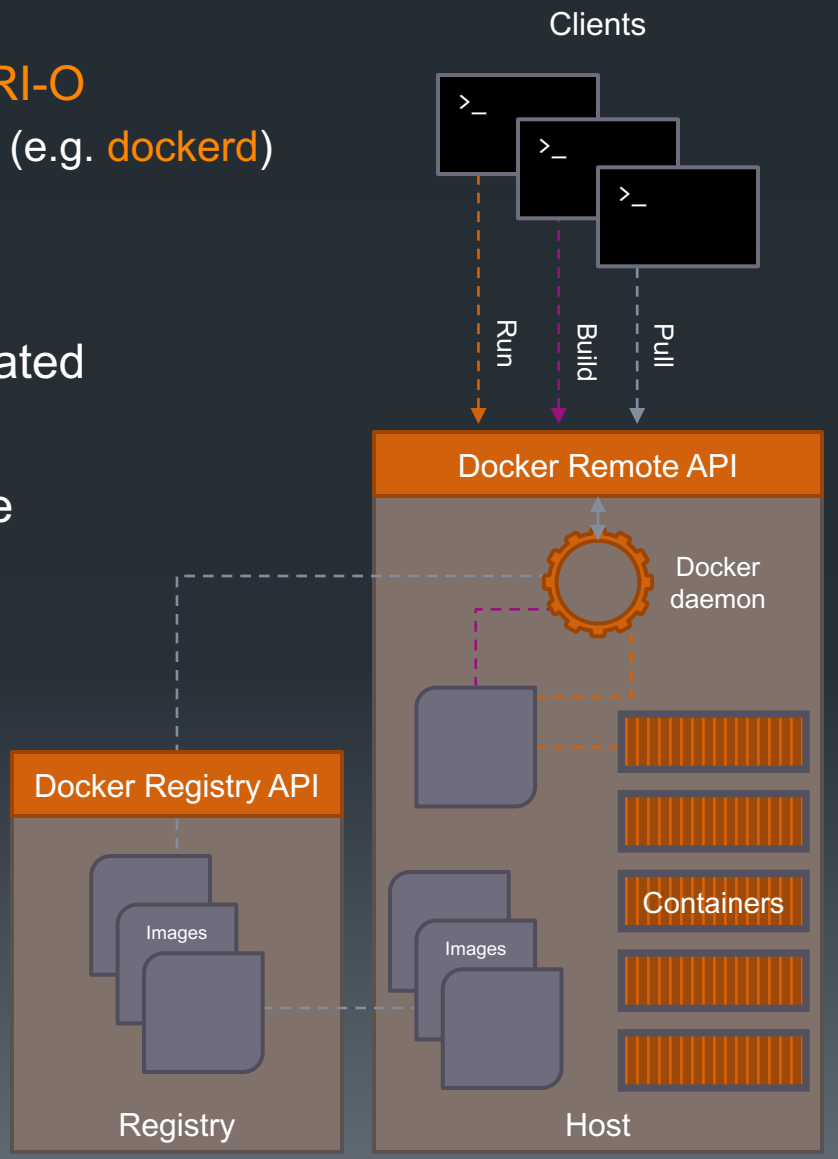
- Lightweight Operating System environment
  - Originally x86 Linux only, now with support for Windows, ARM and other combinations
- Encapsulated, deployable, runnable
  - The new way to package applications
- Microservice-centric
  - one atomic service per container
- Made widely popular by Docker
  - Docker, Inc. provides a container engine including systems for running, publishing, and sharing containers
  - Container technology predates and enables Docker
  - Containers are now standardized through the Open Container Initiative (OCI)
- Containers rely on integral features of the Linux Kernel (now emulated in Windows)
  - CGroups
  - Namespaces
  - Linux Bridge/IPTables/Capabilities/etc.

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

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

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

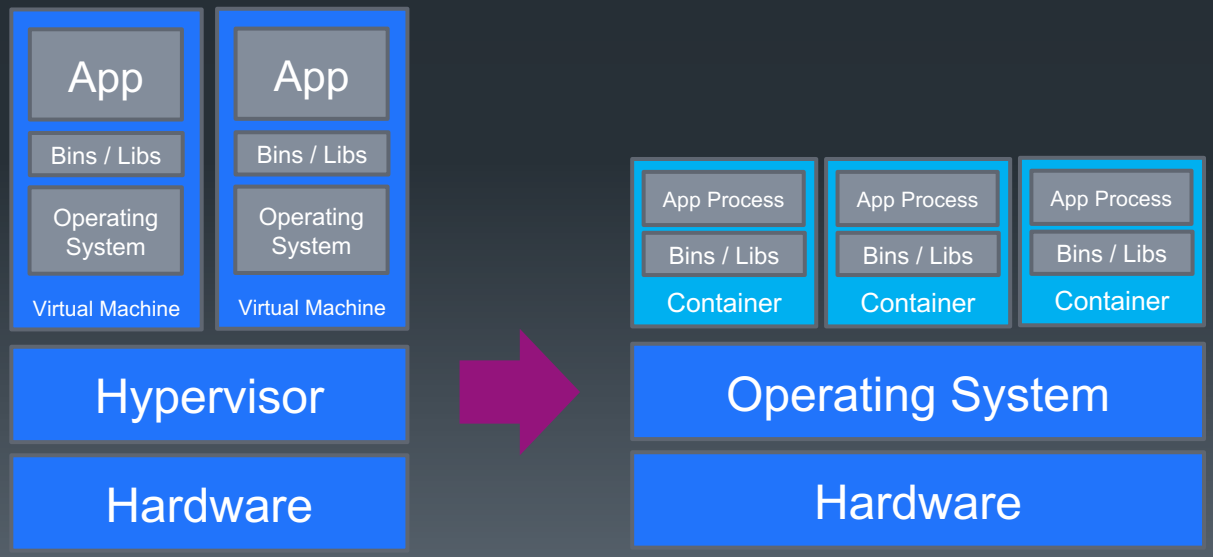
Core Container Concepts

- Container Engine / Runtime
  - Container Runtime Services: containerd, CRI-O
    - Accessed directly or through an engine service (e.g. dockerd)
  - Command line clients: docker, crictl, nerdctl
- Images
  - Templates from which containers are generated
- Registries
  - Network services from which Images can be saved and retrieved
- Containers
  - A container is a software package
    - generated from an image
  - Said to be running when processes are executing within it
  - Can be stopped and started



Contrasting Containers with VMs

- VM
  - A virtual machine for operating systems
- Container
  - A virtual operating system for applications

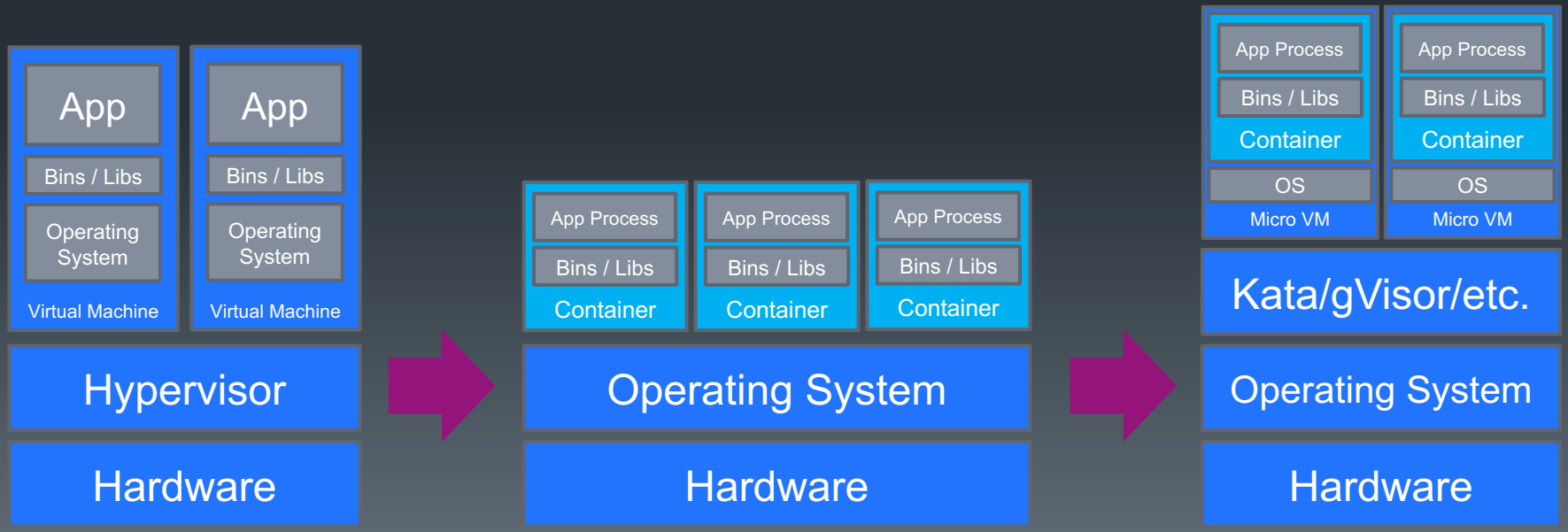


Container Runtimes

- Container Engines - Full suites that run containers, build images, & more
  - Docker
    - Container platform for publishing and sharing containers
  - Podman
    - Like CRI-O but with focus on user (dev or admin) and not platform (i.e. Kubernetes)
- Container Runtimes - Interfaces (e.g. APIs) for running containers only
  - containerd
    - Daemon for Linux and Windows with an emphasis on simplicity
  - CRI-O
    - Implementation of Kubernetes CRI (Container Runtime Interface) used by OpenShift
- Container Runners / Launchers - Run containers in specific ways
  - runc – low level executable OCI runner written in Go
  - crun – RedHat led OCI container runtime written in C, part of the containers project (podman, buildah, libpod, etc.)
  - vSphere/Tanzu
    - Docker API-compatible hypervisor running container images in VMs
  - Kata Containers
    - CRI/OCI compatible lightweight VMs w/ individual kernel instances
  - gVisor
    - OCI runtime (runsc) with user-space kernel, merged guest kernel and VMM
  - Firecracker
    - Runs containers (lambda, etc.) in lightweight virtual machines (MicroVMs)
  - Nabla Containers (also runs in VM)
    - Container isolation via seccomp policy that blocks all but 7 system calls
- Other Container (and container-like) Tech
  - LXC
    - Original Linux container library (now at v2)
  - LXD/OpenVZ
    - Systems in containers (lightvisors, kernel w/ init system per container)
  - BSD Jails
    - Isolation feature of BSD Unix
  - OpenSolaris Container
    - Zones-based isolation model

Sandbox Types

- Containers and VMs are not mutually exclusive and some environments become optimal when properly combining both
- Containers provide OS level application isolation
  - Lightweight multitenancy
- VMs provide hardware level OS isolation
  - Heavyweight multitenancy
- Lightvisors (Kata, gVisor, etc.) apply VM technology to the OCI container ecosystem
  - Better isolation than containers and more responsive than VMs



Container Standardization

- Open Container Initiative (OCI)
  - Circa 6/2015
  - A standard that ensures any container can run on any machine or cloud computing service
- Run by the Linux Foundation
- Three specifications:
  - Runtime Specification (runtime-spec)
    - Specifies the configuration, execution environment, and lifecycle of a container
    - Released v1.0 July 19, 2017
  - Image Specification (image-spec)
    - An OCI implementation downloads an OCI Image, unpacks it, then runs it
    - Released v1.0 July 19, 2017
  - Distribution specification (distribution-spec)
    - Standardize container image distribution based on the specification for the Docker Registry HTTP API V2 protocol
    - Released v1.0 RC 0 February 14, 2019
- Docker donated:
  - container format
  - runtime code (libcontainer)
  - specifications
- runc was created as a libcontainer client
  - represent the lowest-level tool for spawning a containerized process

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

Running Containers

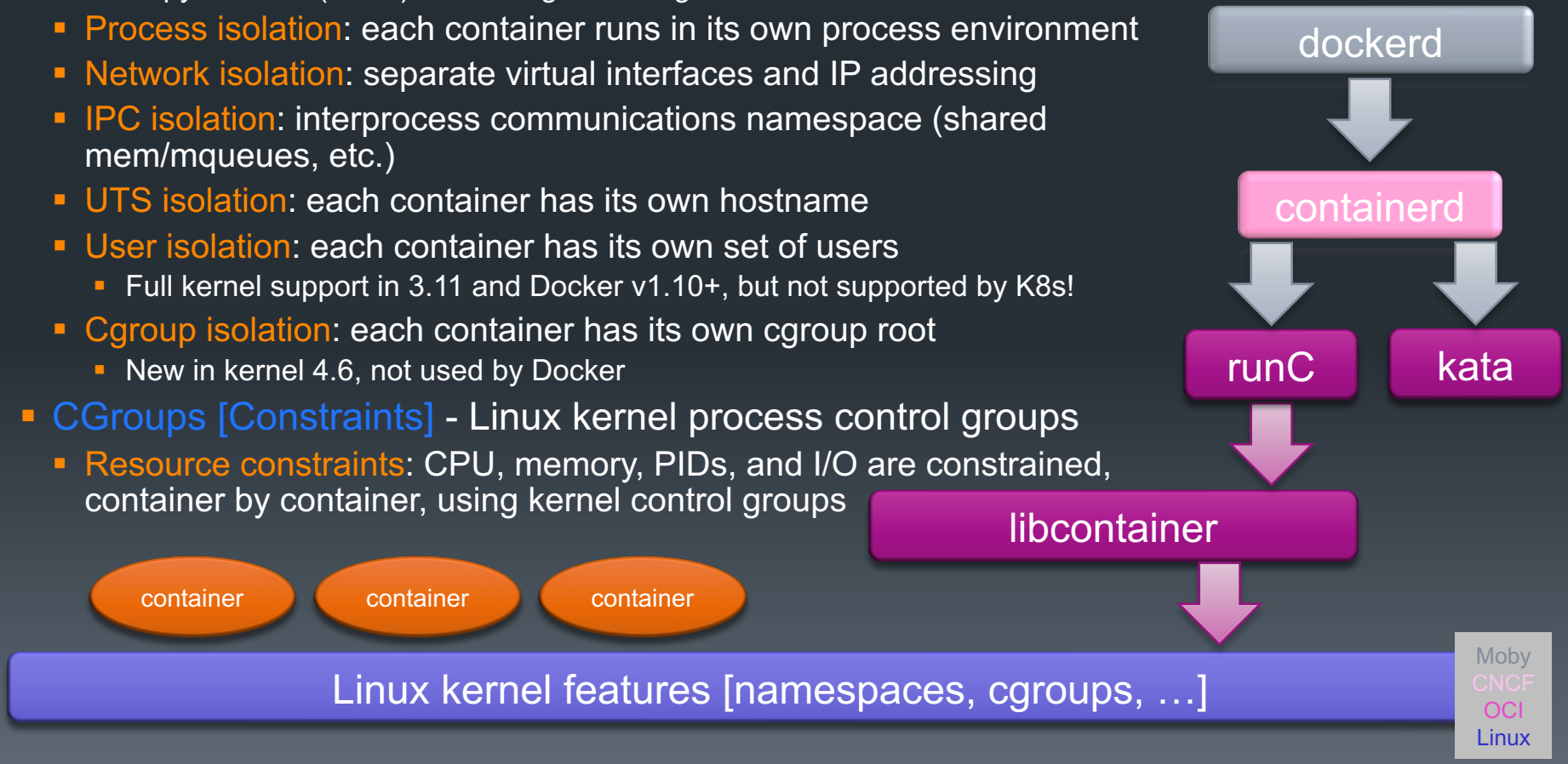
- Clients like `docker`, `podman`, `ctr`, or `crictl` provide commands to pull, create, and start containers
  - e.g. `docker container run`, `crictl run`
- `run` commands typically accept an image and a command to run in the container
  - Some containers allow users to override what program they run
- The resulting container can run
  - in the foreground (directly in the terminal)
  - In the background (like a service)

```
user@ubuntu:~$ docker container run -it cirros /bin/sh
Unable to find image 'cirros:latest' locally
latest: Pulling from library/cirros
a3ed95caeb02: Pull complete
8c4568d40636: Pull complete
e6cc72aea3e6: Pull complete
b5a1edf1e076: Pull complete
Digest: sha256:9aa75497b46cc15cccceef625acee6017d7f3e78db9bd5f7b6b933feaa38e3ae
Status: Downloaded newer image for cirros:latest
/ # hostname
57aed463f5b9
/ # cat /etc/os-release
NAME=Buildroot
VERSION=2012.05
ID=buildroot
VERSION_ID=2012.05
PRETTY_NAME="Buildroot 2012.05"
/ # exit
user@ubuntu:~$ cat /etc/os-release
NAME="Ubuntu"
VERSION="16.04.1 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04.1 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
UBUNTU_CODENAME=xenial
```

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

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

Run a command in a new container

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

Linux Containers

- Container technology predates and enables runtimes like Docker, CRI-O, etc.
- Containers rely on integral features of the Linux Kernel:
  - Namespaces [Isolation] – Linux kernel process namespaces
    - Filesystem isolation: each container has its own root filesystem
      - Copy-on-write (COW) minimizing disk usage
    - Process isolation: each container runs in its own process environment
    - Network isolation: separate virtual interfaces and IP addressing
    - IPC isolation: interprocess communications namespace (shared mem/mqueues, etc.)
    - UTS isolation: each container has its own hostname
    - User isolation: each container has its own set of users
      - Full kernel support in 3.11 and Docker v1.10+, but not supported by K8s!
    - Cgroup isolation: each container has its own cgroup root
      - New in kernel 4.6, not used by Docker
  - CGroups [Constraints] - Linux kernel process control groups
    - Resource constraints: CPU, memory, PIDs, and I/O are constrained, container by container, using kernel control groups



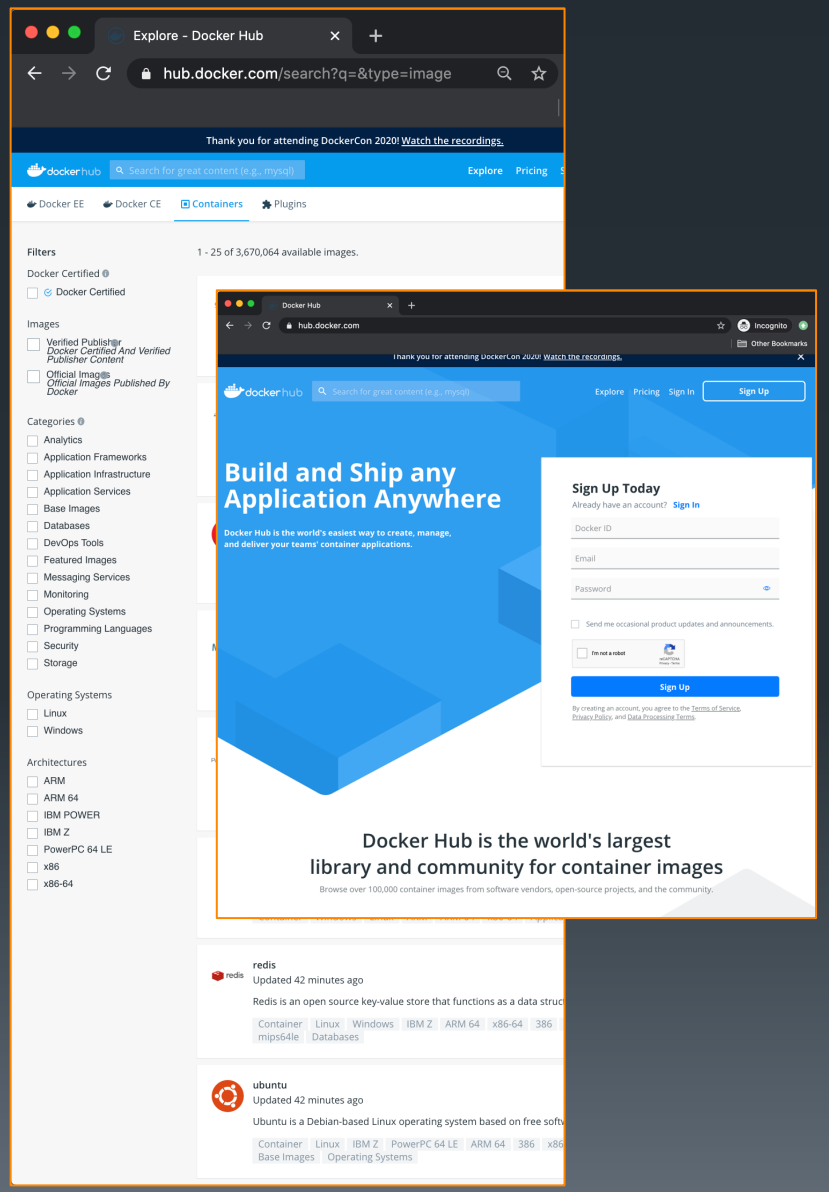
Container Isolation

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

```
$ docker container run -it ubuntu:22.04 /bin/bash

root@c7c250f870e2:/# apt-get update && apt-get install -y vim
...
root@c7c250f870e2:/# which vim
/usr/bin/vim
root@c7c250f870e2:/# exit

$ docker container run -it ubuntu:22.04 /bin/bash

root@2e53ff3bd157:/# vim
bash: vim: command not found
root@2e53ff3bd157:/# which vim
root@2e53ff3bd157:/# exit

$ vim
bash: /usr/bin/vim: No such file or directory
```

Registries and Image Types

- Registries host all kinds of images, including...
- Base images
  - Cirros, Ubuntu, Fedora, Alpine, Debian, Centos, ...
  - These images can be used as the basis for building custom application service images
- Apps like state stores
  - MongoDB, Cassandra, Redis, Postgres, MySQL, Couchbase, …
- Apps like web servers
  - Apache Httpd, Nginx, Tomcat, Tomee, Glassfish, …
- Apps like message brokers
  - Nats, Kafka, RabbitMQ, ActiveMQ, …
- Even pre-packaged dev platforms
  - Java:5, Java:6, Java:7, Java:8, Java:9, NodeJS, Go, Rust, Ruby, Python, Erlang, Haskell, Swift, …



Container Lifecycle Control

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

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

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

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

Start one or more stopped containers

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

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

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

Stop one or more running containers

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

Container Logs

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

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

Usage:  docker container logs [OPTIONS] CONTAINER

Fetch the logs of a container

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

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

```
user@ubuntu:~$ docker container run --name hellosvc -d cirros /bin/sh -c "while true; do echo hello world; sleep 1; done"
016bb141e78acb292a1e9b57ef30a24dbaa718e21eac9031cb3fd52d41becb70
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
016bb141e78a        cirros              "/bin/sh -c 'while..."   6 seconds ago       Up 5 seconds                            hellosvc
user@ubuntu:~$
user@ubuntu:~$ docker container logs hellosvc
hello world
hello world
hello world
hello world
hello world
hello world
hello world
```

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

Looking Up Container PIDs: top

- Examines the processes running within a container using the top subcommand
  - Allows you to identify how the host sees a containerized process
  - Containers supply processes with a PID namespace allowing them to see only the other processes within the container
  - Process IDs within the container start with 1 and are mapped to available PIDs on the host
    - `ps` inside the container shows PID namespace local PIDs and `top` shows host relative PIDs
- Example: `docker container top`
  - Top accepts standard ps switches

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

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

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

Usage:  docker container top CONTAINER [ps OPTIONS]

Display the running processes of a container

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

Looking up Container Resource Use: Stats

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

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

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

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

Container Exec

- Runs an arbitrary process with a new PID within a container (all namespaces and cgroups enforced)
  - `docker container exec <container> <cmd>`
    - `-d` runs new process detached
    - `-i` runs new process in interactive mode
- Exiting an exec session leaves the container running
- Opposite of attach which gives you control of PID 1 in the container
  - If you ctrl C from attach, you exit the main container process and shut down the container!

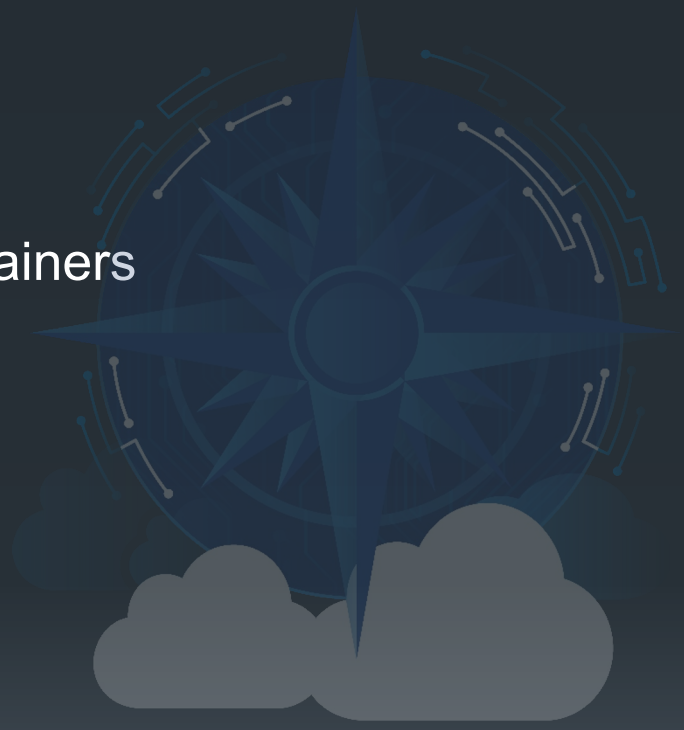
```
$ docker container ls
CONTAINER ID    IMAGE         COMMAND                  CREATED         STATUS         PORTS            NAMES
7f5a4ebf67ac    nginx:1.11    "/docker-entrypoint.…"   4 minutes ago   Up 4 minutes   80/tcp, 443/tcp  silly_carson

$ docker container exec 7f5a cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 8 (jessie)"
NAME="Debian GNU/Linux"
VERSION_ID="8"
VERSION="8 (jessie)"
ID=debian
HOME_URL="http://www.debian.org/"
SUPPORT_URL="http://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"

$ docker container exec -it 7f5a /bin/bash
root@7f5a4ebf67ac:/# ps
   PID TTY          TIME CMD
     7 pts/0    00:00:00 bash
    12 pts/0    00:00:00 ps
root@7f5a4ebf67ac:/# exit
exit

$ docker container ls
CONTAINER ID    IMAGE         COMMAND                  CREATED         STATUS         PORTS            NAMES
7f5a4ebf67ac    nginx:1.11    "/docker-entrypoint.…"   5 minutes ago   Up 5 minutes   80/tcp, 443/tcp  silly_carson
```

Summary

- The Docker platform includes several parts
  - Docker daemon (docker engine)
  - Docker client
  - Registries
  - Images
  - Containers
- Containers are spawned from images and are isolated from each other and the host operating system
- The docker command line tool provides an array of commands for starting, examining and controlling containers

Lab 1

- Setting up Docker and exploring containers



2: Container Images

Objectives

- Understand the differences between Images, Repositories, Tags and Registries
- Describe container image layering
- Explore image creation and Dockerfiles
- Examine commands to display images and image metadata
- Move images between registries and the local Docker host
- Examine Docker CLI commands to interact with registries
- Explore image tagging and using tags to push a locally cached image to a registry

Container Images

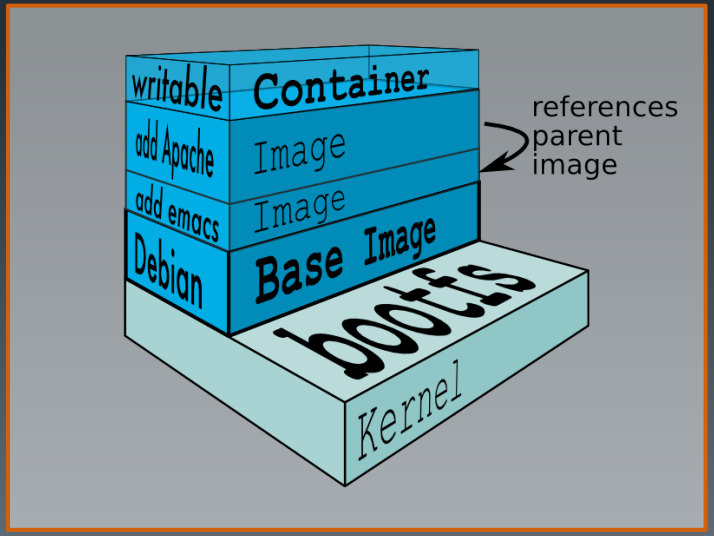
- Images are immutable files that include everything to run an application in a container
  - e.g. code, binaries, dependencies, environment variables, users, etc.
- Containers are running instances of images
  - If a required image isn't already present on the host it is downloaded from an image registry
- Identified using an image name specifying:
  - The address of an image registry, a.k.a. the registry, serving and hosting images over the network
    - If not provided, most container runtimes will assume a default registry (e.g. Docker defaults to DockerHub)
  - The collection in the registry holding all versions of an image known as the repository
  - The individual version of the image, a.k.a. the tag
    - e.g. `docker.io/nginx:1.20.1` specifies version 1.20.1 of the NGINX image found on DockerHub

```
$ docker container run -d -p 80:80 --name apache_webserver httpd
b5aadbe119b3c2f48253b29c9b02feb908f9327fbb3d3653d58fbd9c17519e8a

$ docker container ls
CONTAINER ID      IMAGE      COMMAND              CREATED          STATUS         PORTS                NAMES
b5aadbe119b3      httpd      "httpd-foreground"   6 seconds ago    Up 4 seconds   0.0.0.0:80->80/tcp   apache_webserver

$ curl -L http://localhost
<html><body><h1>It works!</h1></body></html>

$ curl -I http://localhost
HTTP/1.1 200 OK
Date: Mon, 15 Jun 2023 17:48:[illegible] GMT
Server: Apache/2.4.43 (Unix)
Last-Modified: Mon, 11 Jun 2023 18:53:14 GMT
ETag: "2d-432a5e4a73a80"
Accept-Ranges: bytes
Content-Length: 45
Content-Type: text/html
```

```
$ docker image inspect httpd | grep -e HTTPD_VERSION -e Os
                "HTTPD_VERSION=2.4.43",
                "HTTPD_VERSION=2.4.43",
        "Os": "linux",
```

Images

- Containers are constructed by sequentially applying metadata and file system layers from one or more images
- A container image includes metadata and an optional file system layer
  - Image metadata includes the
    - ID of the image's parent - Image IDs are SHA256 hashes of the image (content addressable)
    - default command to run, etc.
  - Layered images tend to supply one specific feature on top of the parent image
  - Upper-layer metadata and files mask metadata and files at lower layers
- An image with no parent image is called a Base Image
  - Base images tend to be larger
  - House operating system libraries and executables (e.g., Debian, Ubuntu, Centos, etc.)
- Image metadata and files are immutable
  - Allows one Image to support multiple container instances with repeatable results
  - Reduces the disk and memory footprint of a given set of containers that share the same read-only images
- Containers have a writable file system layer
  - The container file system layer is initially empty
  - All writes go to this file system layer and overlay any matching underlying image files
  - Container file systems contain only the delta between:
    - Their file system state
    - The underlying images
  - The view from the top down, including all file system layers in the stack is called union file system



Displaying images

- The `docker image ls` command displays the images available on the docker engine
- Images that use the v2 format have a SHA 256 digest and ID
  - Both identify the content
  - The ID is the hash of the image manifest and the uncompressed filesystem layers
  - The digest is the hash of the image manifest and the gzipped filesystem layers
    - Useful for download verification
- `Pull`, `create`, `run`, and `rm` can use the digest
  - `docker image pull hello-world@sha256:0256e8a36e2070f7bf2d0b0763dbabdd67798512411de4cdcf9431a1feb60fd9`
  - Images not pulled will not have a digest

```
user@ubuntu:~$ docker image ls --help

Usage:  docker image ls [OPTIONS] [REPOSITORY[:TAG]]

List images

Aliases:
  ls, images, list

Options:
  -a, --all             Show all images (default hides intermediate images)
      --digests         Show digests
  -f, --filter filter   Filter output based on conditions provided
      --format string   Pretty-print images using a Go template
      --help            Print usage
      --no-trunc        Don't truncate output
  -q, --quiet           Only show numeric IDs
```

```
user@ubuntu:~$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              latest              98d35105a391        6 days ago          193 MB
busybox             latest              00f017a8c2a6        12 days ago         1.11 MB
nginx               latest              6b914bbcb89e        3 weeks ago         182 MB
httpd               latest              f316d5949bb0        3 weeks ago         176 MB
ubuntu              latest              0ef2e08ed3fa        3 weeks ago         130 MB
ubuntu              12.04               b384dd9703db        3 weeks ago         104 MB
hello-world         latest              48b5124b2768        2 months ago        1.84 kB
cirros              latest              f8ce316a37a7        14 months ago       7.74 MB
```

```
user@ubuntu:~$ docker image ls --digests
REPOSITORY          TAG                 DIGEST                                                                    IMAGE ID            CREATED             SIZE
centos              latest              sha256:be5b4a93f116a57ab3fd454ada72421eac892a3a4925627ac9a44f65fcd69cf8   98d35105a391        6 days ago          193 MB
busybox             latest              sha256:32f093055929dbc23dec4d03e09dfe971f5973a9ca5cf059cbfb644c206aa83f   00f017a8c2a6        12 days ago         1.11 MB
nginx               latest              sha256:52a189e49c0c797cfc5cbfe578c68c225d160fb13a42954144b29af3fe4fe335   6b914bbcb89e        3 weeks ago         182 MB
httpd               latest              sha256:4612fba4347bd87eaeecd5c522d844f26cc4065b45eef9291277497946b7a86c   f316d5949bb0        3 weeks ago         176 MB
ubuntu              latest              sha256:dd7808d8792c9841d0b460122f1acf0a2dd1f56404f8d1e56298048885e45535   0ef2e08ed3fa        3 weeks ago         130 MB
ubuntu              12.04               sha256:47effe29d7bed549fee8c63bd60120fa88785e72abfb2aeeccdda21f4eae67e4   b384dd9703db        3 weeks ago         104 MB
hello-world         latest              sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7   48b5124b2768        2 months ago        1.84 kB
cirros              latest              sha256:9aa75497b46cc15cccceef625acee6017d7f3e78db9bd5f7b6b933feaa38e3ae   f8ce316a37a7        14 months ago       7.74 MB
```

Creating Images

- There are two ways to create new images
  - Update a container created from an image
    - Commit the results to a new image
  - Use a Dockerfile
    - Specify instructions to create an image
- Docker caches created or downloaded images on the Docker host
  - If a required image isn't already present on the host, it is downloaded from a registry
  - Created images can subsequently be pushed to a registry

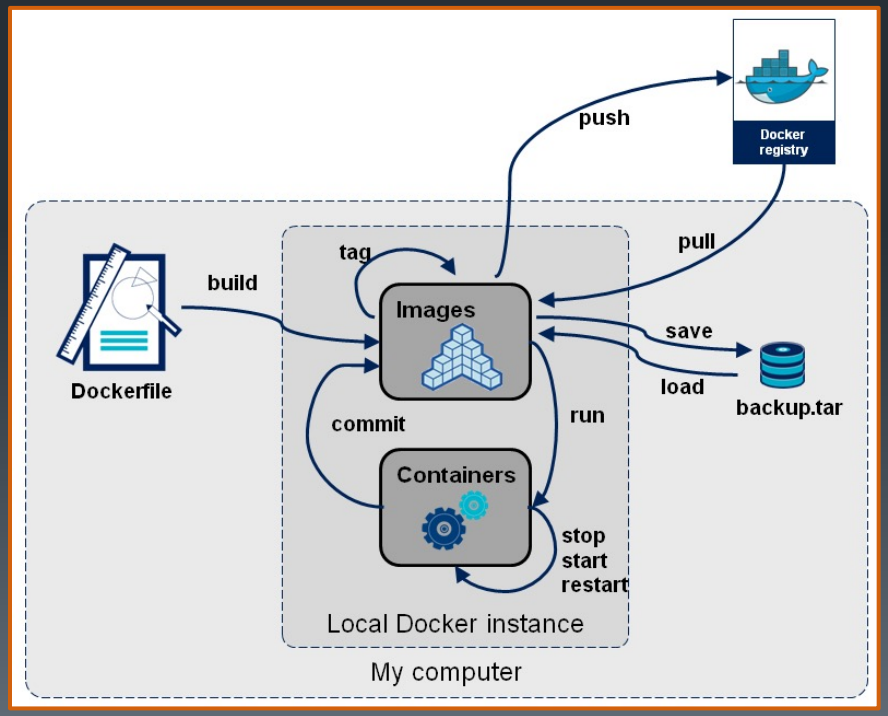


Image Build Tools

- Daemon based Image Builds
  - docker/dockerd: traditional (root) daemon build system https://www.docker.com/
    - buildctl/buildkitd client/daemon used by docker https://github.com/moby/buildkit
- CLI Image Builds
  - oci-image-tool: OCI project image tool https://github.com/opencontainers/image-tools
  - buildah: cli image builder (used by podman) https://buildah.io/
  - kaniko: image builder designed to run in k8s https://github.com/GoogleContainerTools/kaniko
  - img: buildkit based cli https://github.com/genuinetools/img
  - pack: Cloud Native Buildpacks PaaS image builder https://buildpacks.io/

```
apiVersion: v1
kind: Pod
metadata:
  name: kaniko
spec:
  containers:
  - name: kaniko
    image: gcr.io/kaniko-project/executor:latest
    args:
    - "--dockerfile=<path to Dockerfile within the build context>"
    - "--context=gs://<GCS bucket>/<path to .tar.gz>"
    - "--destination=<gcr.io/$PROJECT/$IMAGE:$TAG>"
    volumeMounts:
    - name: kaniko-secret
      mountPath: /secret
    env:
    - name: GOOGLE_APPLICATION_CREDENTIALS
      value: /secret/kaniko-secret.json
  restartPolicy: Never
  volumes:
  - name: kaniko-secret
    secret:
      secretName: kaniko-secret
```

Declarative Image Builds

- Build tools 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
- 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:20.04`
- 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 on the container
  - Commits a new image from the modified container (may be metadata only)
  - Deletes the working container
  - 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   RUN apt-get update && apt-get install -y apache2 && apt-get clean && rm -rf /var/lib/apt/lists/*
5
6   ENV APACHE_RUN_USER www-data
7   ENV APACHE_RUN_GROUP www-data
8   ENV APACHE_LOG_DIR /var/log/apache2
9
10  EXPOSE 80
11
12  CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]
```

Dockerfiles and Context

- By calling `docker image build` from your terminal, you can have Docker construct your image step by step, executing the instructions successively
  - *# docker image build /some/path/*
- The path supplied to docker image build is the context of the build
  - Also known as the source repository
  - The build is run by the Docker engine, 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
  - Therefore, in most cases it is best to put each Dockerfile in an empty directory, adding only the files needed for building that Dockerfile

Dockerfile construction

- `FROM`
  - should (almost) always be the first instruction; specifies a base image that the *Dockerfile* will operate on
- `RUN`
  - Executes commands using tools found in 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`
  - Adds metadata that tells Docker that the application in this container uses a specific port
  - Application code must listen on the exposed port
  - You must map the port to the host using the docker container 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*
  - Overridden if an argument is specified on the docker run command line
    - e.g: `docker container run -it nginx /bin/bash` runs /bin/bash/ instead of the originally intended command described by CMD

```
# 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"]
```

Additional Dockerfile Instructions

- `ENTRYPOINT`
  - Specifies the program that is always run by an image
  - Not overridden if arguments are supplied to a `docker container run` command
  - Arguments stated by a docker container run are passed 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 container run thrift --gen java appcore.thrift`
- `LABEL`
  - Adds arbitrary 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`
  - Operations in the container that do not specify full paths will run from this directory

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

```
LABEL version="1.0"
USER webapp
WORKDIR /opt/webapp/db
RUN bundle install
WORKDIR /home/webapp/
ENTRYPOINT [ "/path/to/bin" ]
```

ADD & COPY

- ADD
  - Adds files and directories from the build environment (a.k.a image build context) 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
      - If the destination ends in a /, docker considers the source a directory
    - ADD automatically unpacks source archives
  - 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
  - Also used in multi-stage builds (covered later)
- 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

Compound Dockerfiles

- Dockerfiles can contain multiple `FROM` statements
  - Each `FROM` defines a new image stack called a stanza
  - Every stanza results in an intermediate image that has their own image ID
  - Only the final `FROM` instruction receives the image tag specified by the user that invoked the build process

- Multi-stage builds
  - Multi-stanza Dockerfile builds that use output from one stanza (e.g. a compiled binary) to build a final image with fewer tools and content than images produced by prior stanzas
  - Ideal for containers that only have the bare minimum contents to run an application (e.g. code, libraries, dependencies, users, envVars)
  - To use artifacts and outputs from prior images in the same build context, use:
    - `COPY --from=<base_image_name/num>`
      - A base *image name* is given in its `FROM` instruction
        - i.e. `FROM golang:1.20 as build-env`

```
# Multiple images example
FROM ubuntu
RUN echo foo > bar
# Will output something like => 907ad6c2736f

FROM ubuntu
RUN echo moo > oink
# Will output something like => 695d7793cbe4

# Results in two images 907ad with /bar and
#   695d7 with /oink
```

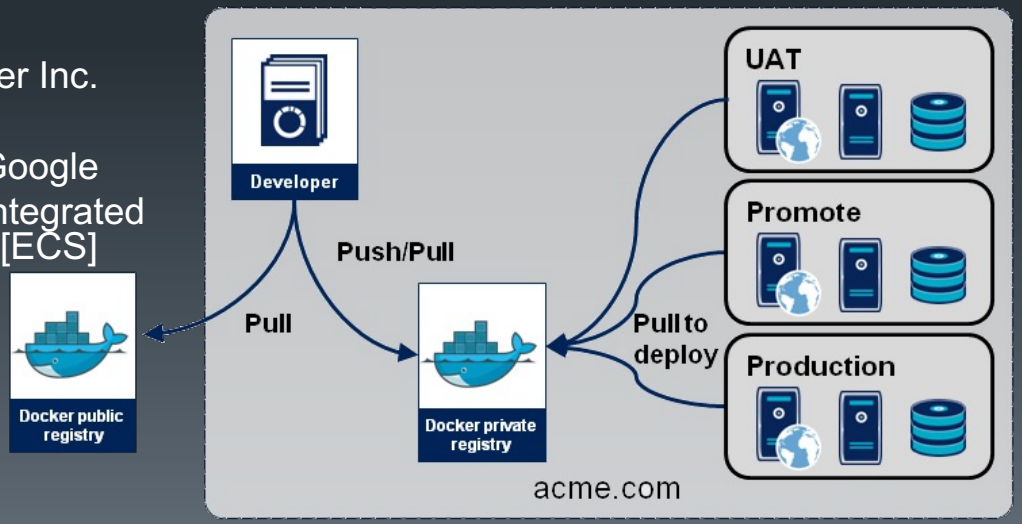
```
# First stage to build the application
FROM maven:3.5.0-jdk-8-alpine as build-env
ADD ./pom.xml pom.xml
ADD ./src src/
RUN mvn clean package

# Final stage to define our minimal runtime
FROM openjdk:8-jre
COPY --from=build-env target/app.jar app.jar
CMD ["java", "-jar", "app.jar"]
```

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
    - Open source and commercial solutions
      - Docker Trusted Registry from Docker Inc. (commercial)
      - Quay Enterprise part of OpenShift platform (commercial)
      - Docker Registry from Docker Inc. (FOSS)
      - Harbor open source enterprise-class container registry, CNCF project
      - GitLab (freemium)
      - Artifactory 3.4+ (commercial)
      - Nexus 3 Milestone 5+ (free)
    - Hosted cloud solutions
      - Docker Hub (hub.docker.com) from Docker Inc.
      - Quay (quay.io)
      - Google Container Registry (gcr.io) from Google
      - Amazon EC2 Container Registry [ECR] integrated with AWS EKS & EC2 Container Service [ECS]
      - Azure Container Registry [ACR] integrated with AKS

Image signing/scanning and **authentication/authorization** support should be considered when choosing a registry

Public registry images should be used with caution by the security minded



Registry Commands

- Docker Registry commands:
  - `docker login`
    - to login to a registry
  - `docker logout`
    - To log out of a registry
  - `docker search`
    - searches a registry for an image
  - `docker image pull`
    - pulls an image from registry to the local machine (also pulls its dependencies)
  - `docker image push`
    - pushes an image to the registry from a local machine

```
❯ docker search thrift
NAME                          DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
thrift                        DEPRECATED; lacking active maintainer           122       [OK]
apache/thrift                 Apache Thrift                                   8                    [OK]
itzg/thrift                   Provides the Apache Thrift generator tool       0                    [OK]
thrift/thrift-build           Apache Thrift build environment.                1                    [OK]
thrift/ubuntu                                                                 0                    [OK]
thrift/thrift-compiler        Apache Thrift Compiler                          1                    [OK]
thrift/debian                 build/docker/debian                             0                    [OK]
cspwizard/thrift              Thrift binary wrapped in docker for generati…   1
jimdo/thrift                  Mirror of Apache Thrift                         0                    [OK]
saltside/thrift               Our thrift fork as docker image used to gene…   0                    [OK]
thrift/thrift-compiler-test   thrift-compiler test run to verify https://i…   0                    [OK]
amd64/thrift                  DEPRECATED; lacking active maintainer           0
alexandersteshenko/thrift     Apache Thrift command line tool version 0.9.0   0
omnisci/thrift                                                                0
```

- These commands (and the run command indirectly) cause the Docker daemon to perform network I/O with a registry independent of the Docker client session
- The IsAutomated field and the “is-automated” filter on the docker search have been deprecated!

Pushing to a private Registry

- To push the image to a registry the registry URL must be the first part of the repository name
- Because repository names and tags are simply aliases, a single image ID may have many names/tags

```
$ docker image build -t bobsmith/thriftdev ~/thriftdev

...

Successfully built ce8e174bab0d
Successfully tagged bobsmith/thriftdev

$ docker image ls
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
bobsmith/thriftdev      latest              84e5bc36d164        10 seconds ago      226 MB
ubuntu                  trusty              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                  14.04               2d24f826cb16        2 weeks ago         188.3 MB

$ docker image tag bobsmith/thriftdev localhost:5000/thriftdev

$ docker image ls
REPOSITORY                  TAG                 IMAGE ID            CREATED             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                      trusty              2d24f826cb16        2 weeks ago         188.3 MB

$ docker image 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/v2/repositories/thriftdev/tags/latest}
$
```

Pulling Images from a Private Registry

- Repository names embed the information used to identify an image's registry
- For Docker Hub:
  - uid/name
- For non Docker Hub:
  - url/name

```
$ docker image ls
REPOSITORY                TAG                 IMAGE ID            CREATED             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

$ docker image rm localhost:5000/thriftdev
Untagged: localhost:5000/thriftdev:latest
Deleted: 84e5bc36d1640aef7cd5be9dba32297ab78d3685312859def2145059a0b0cf32

$ docker image ls
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
ubuntu                  trusty              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                  14.04               2d24f826cb16        2 weeks ago         188.3 MB

$ docker container 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 image ls
REPOSITORY                TAG                 IMAGE ID            CREATED             SIZE
localhost:5000/thriftdev  latest              84e5bc36d164        28 minutes ago      226 MB
ubuntu                    trusty              2d24f826cb16        2 weeks ago         188.3 MB
ubuntu                    14.04               2d24f826cb16        2 weeks ago         188.3 MB
```

Summary

- Container Images have:
  - Metadata
    - ID
    - Parent ID
    - Environment variables
    - etc.
  - Optionally any of the following:
    - One or more Repository:Tag names
    - A filesystem layer
- Container images are read-only
  - Changes in a container occur in the container layer overlaying image layers
  - Image file system layers are unioned to create a single file system view
- Repositories named collections of related Docker images
- Image Registries are a central location for storage and retrieval of Docker images
  - Docker Registry lookups are based on x[/y[/z]][:t] formatted names
    - **/** reserved character separating discrete components of the image path
    - **:** reserved character separating repository path from image tag (also separates hostname from port in x position)
    - **x/y/z** lookups interpret **x** as registry hostname:port, **y** as namespace and **z** as the repository name
    - **t** is the image tag name and defaults to “latest” if not present
  - e.g. “`reg.example.com:5005/fred/specialrepo:latest`” [FQIN!!]

Lab 2

- Registries and Dockerfile builds

3: Kubernetes Overview

Objectives

- Gain a high-level understanding of the Kubernetes container orchestration platform
- Describe the goals of the Kubernetes system
- Explore the Kubernetes architecture
- Discuss Kubernetes control plane components and node agents
- Examine Kubernetes-as-a-service solutions
- Discuss various techniques for installing kubernetes
- Understand how to configure and use the Kubernetes CLI: kubectl

After Containers, what's left?

- Orchestration!
  - Cloud native apps are delivered in 10s of images requiring 1000s of running containers
  - Registries manage image distribution
  - Orchestrators manage containers at scale
- Orchestration features:
  - Container Scheduling
    - Distributing containers to appropriate hosts
    - Host resource leveling
    - Availability zone diversity
    - Related container packaging and co-deployment
    - Affinity/Anti-affinity
  - Container Management
    - Monitoring and recovery
    - Image upgrade rollout
    - Scaling
    - Logging
  - Service Endpoints
    - Discovery/Name-Resolution
    - High Availability
    - Load Balancing
    - Traffic Routing
  - External Services
    - Container Runtimes
      - CRI <--> OCI
    - Network configuration and management
      - CNI
    - Durable volume management
      - CSI

What is Kubernetes

- Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts
- Responds quickly to user demand:
  - Scaling applications on the fly
  - Seamlessly rollout new features
  - Optimized use of hardware using only the resources needed
- Kubernetes is:
  - lean: lightweight, simple, accessible
  - portable: public, private, hybrid, multi-cloud
  - extensible: modular, pluggable, hookable, composable
  - self-healing: auto-placement, auto-restart, auto-replication
- Commonly shortened to K8S (K at the beginning, S at the end, with 8 characters in between)
- Core components Kubernetes ships with:
  - A server that serves **the Kubernetes API**
  - A **base set of API resource types** allowing users to define various desired states
    - e.g. Run a container somewhere in a cluster of computers
  - **Software that consumes API resources** & changes a system per what is described

Kubernetes

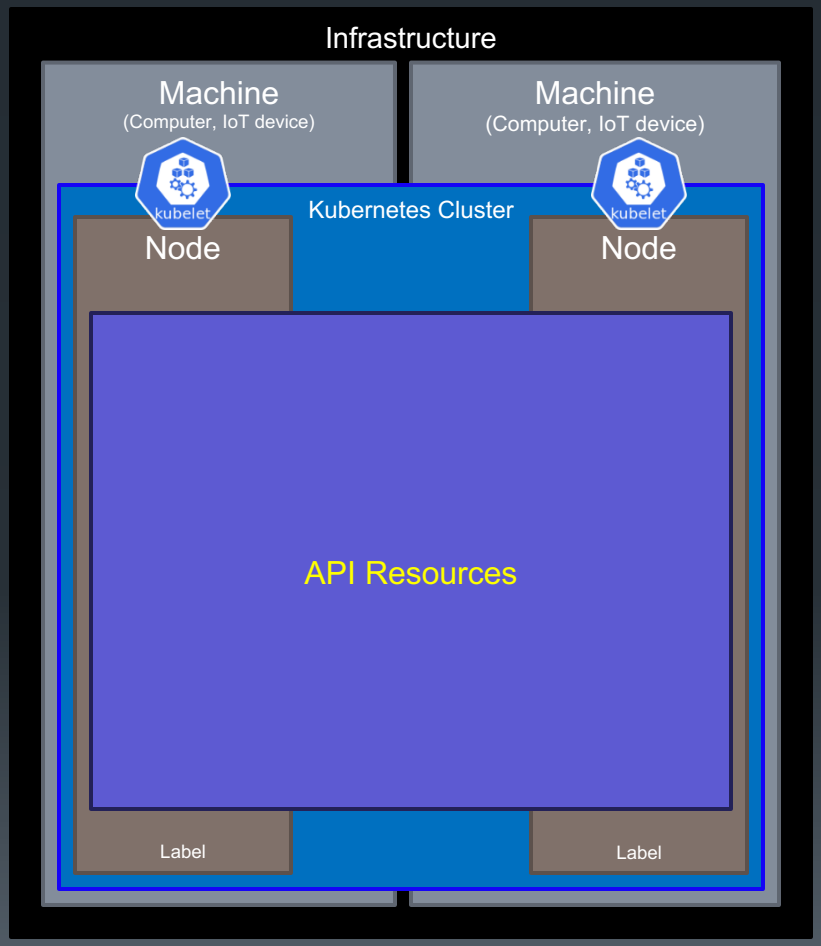
- Greek word (κυβερνήτης) meaning helmsman or captain
- Derived from the Greek word kubernan
- Derivations in other languages include the English cybernetics

- Kubernetes can be deployed directly on private systems in corporate data centers and on various cloud hosting providers, such as Google Compute Engine
- Under active development by many of the same engineers who built Borg=

The Kubernetes project was started by Google in 2014 Kubernetes builds upon a decade and a half of experience that Google has with running production workloads at scale (Borg), combined with best-of-breed ideas and practices from the community

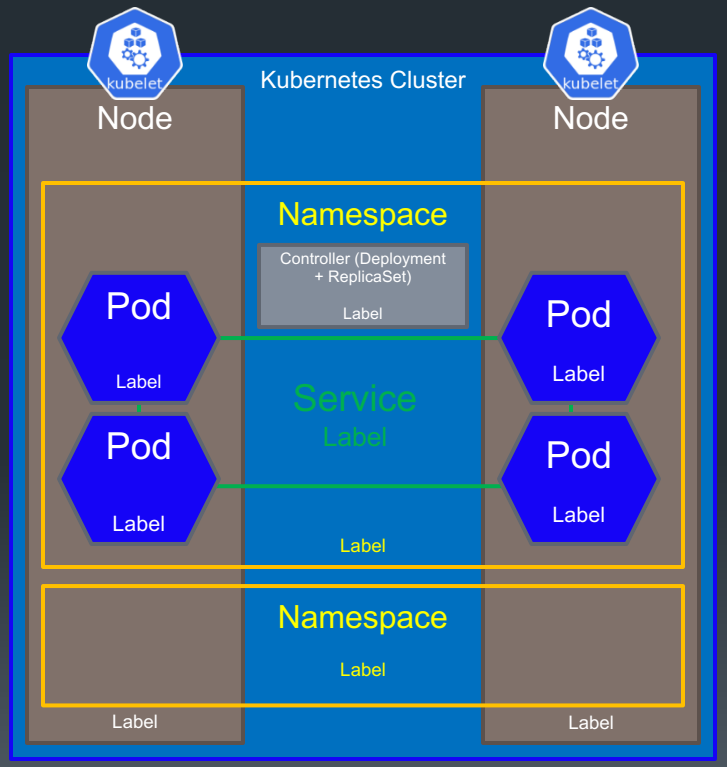
Kubernetes Concepts: Architectural

- Cluster
  - A cluster is a set of physical or virtual machines and other infrastructure resources used by Kubernetes to run your applications
  - Hosts a unique instance of the Kubernetes API
- Node
  - A node is a physical or virtual machine running the Kubernetes agent **`kubelet`**, onto which pods can be scheduled
- Namespace
  - A mechanism to partition resources created by users into a logically named group
  - Allows separate user communities to work in isolation
  - Manifests as separate endpoints in the Kubernetes API
  - Can be used as use, security, & network boundaries
    - Resource quotas & limit ranges to constraint resource use
    - Network Policies to enforce app-to-app communication
- API Resources
  - Specifications of Kubernetes constructs that will provide instructions on how to fulfill a desired state
- Label
  - Labels are optional key:value pairs stored in the metadata of Kubernetes resources used to organize and select groups of objects

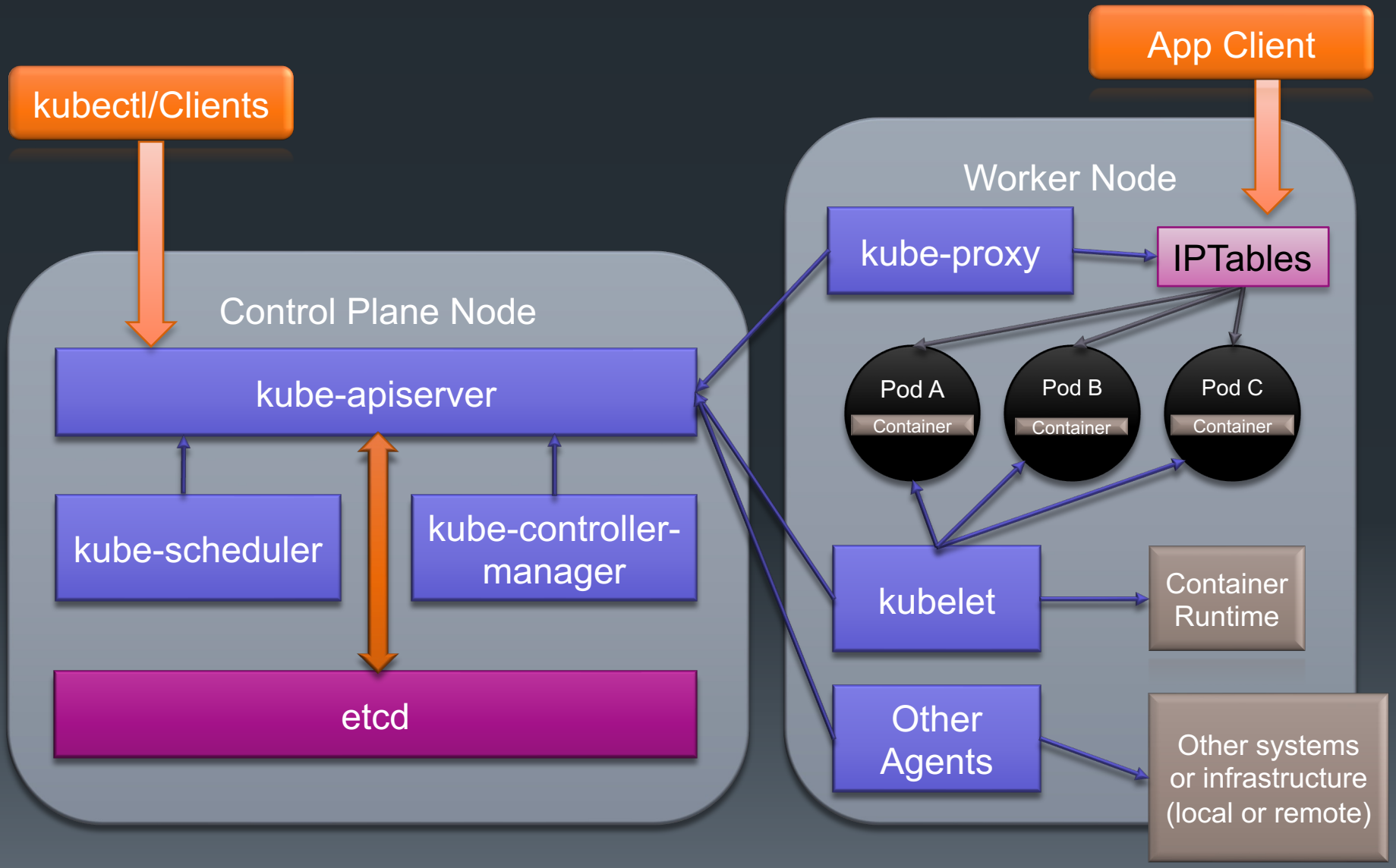


Kubernetes Concepts: Workload-related API Resources

- Pod
  - Pods are co-located groups of application containers
  - The smallest deployable units that can be created, scheduled, and managed with Kubernetes
  - Pods can be created individually, but it's recommended that you use a controller even if creating a single pod
- Controllers
  - Controllers ensure the running state of the cluster reflects the user's specified state
    - e.g. "Run n pods in the cluster"
  - Run control loops that watch for and reconcile differences between the cluster's current state and user's desired state
  - Separated into many types, including (but not limited to):
    - Replica Set
      - Replica Sets manage the lifecycle of pods
      - They ensure that a specified number of pods are running at any given time, by creating or deleting pods as required
    - Deployment
      - Allow initial deployment of pods through replica sets and rolling upgrades
- Service
  - Services provide a single, stable network identity (name and address) for a set of pods
  - Act as basic load balancers

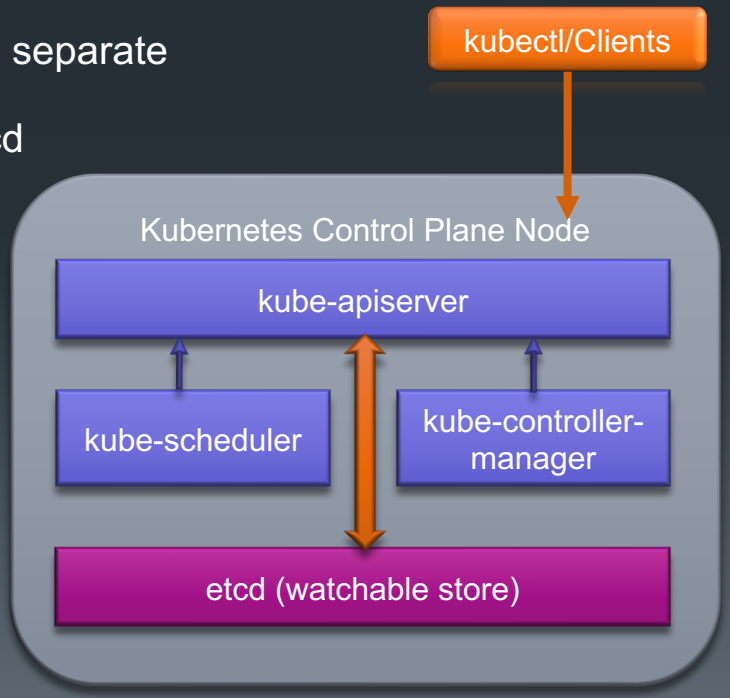


Kubernetes Architecture



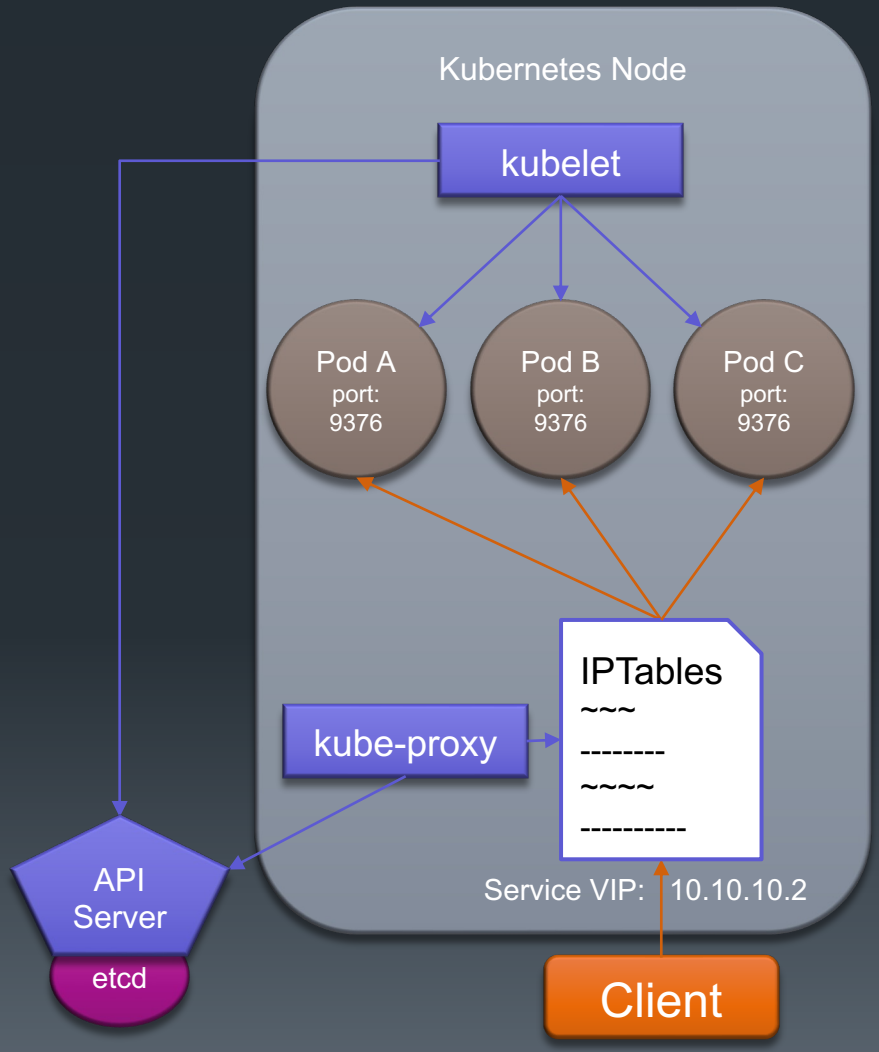
Control Plane

- The Kubernetes Control Plane
  - The Kubernetes control plane is split into a set of microservices
  - These components work together to provide a unified view of the cluster
- etcd
  - All persistent state is stored in an etcd cluster
    - Stores active copies of API resource specifications
    - If it exists in etcd, the cluster will maintain it
  - Watch support allows coordinating components to be notified of changes
- API Server
  - Serves the Kubernetes API
  - A CRUD-y server, most/all business logic implemented in separate components or plug-ins
  - Processes REST operations, validates them, updates etcd
- Scheduler
  - Binds unscheduled pods to nodes via the /binding API
  - The scheduler is customizable and pluggable
- Controller Manager
  - Other cluster-level functions are performed by the Controller Manager
    - Endpoints sync with services; Node discovery and management; Controller (ReplicaSets, Autoscalers, etc.) state mangement



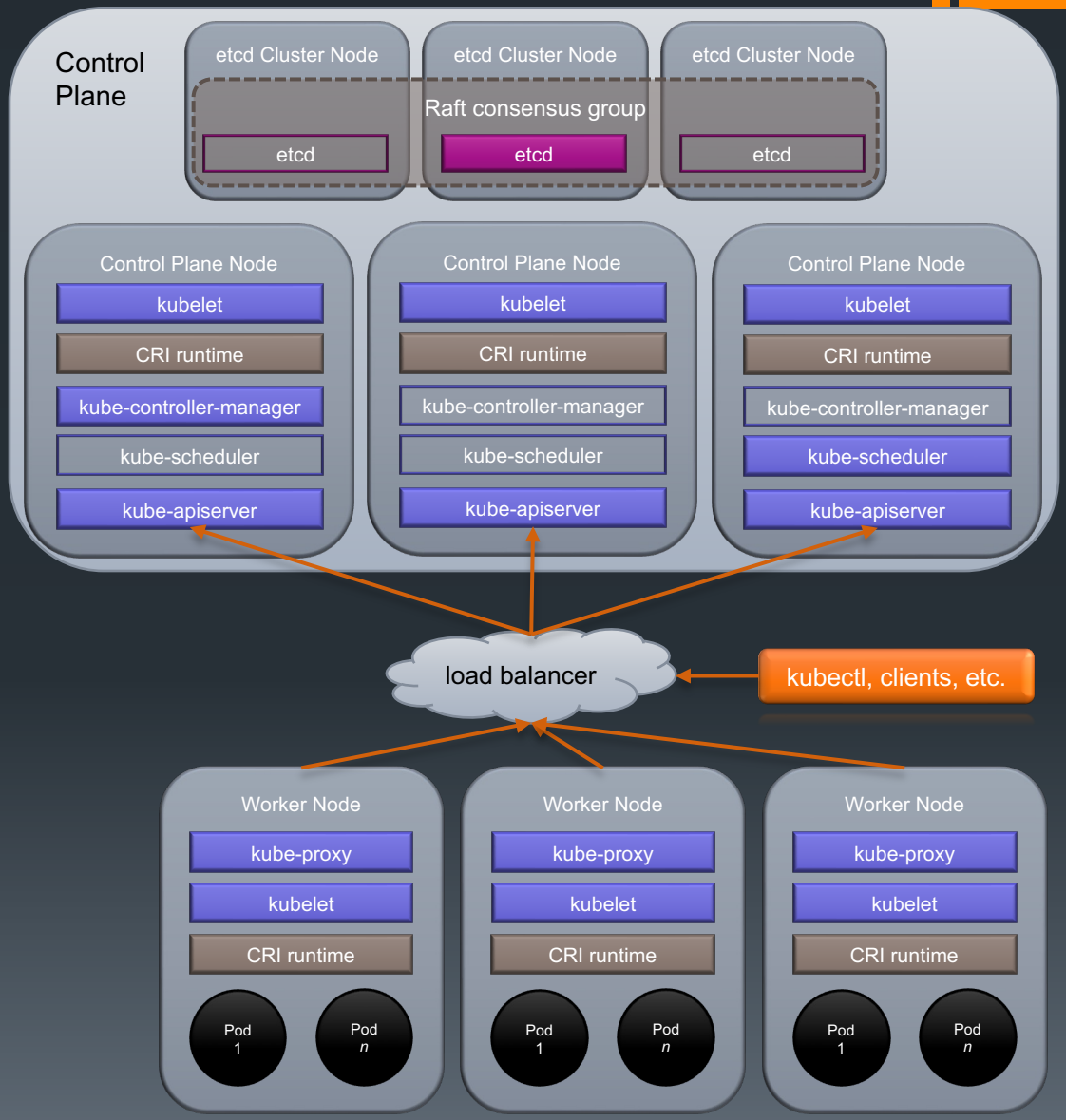
Node Agents

- kubelet
  - The kubelet is the primary “node agent” that runs on each node as a system service
  - The kubelet manages pods and their containers, their images, their volumes, etc.
    - Kubelet contacts the container runtime to create or delete Pod containers
    - The container runtime then assigns Pods IP addresses according to the CNI plugin in use
  - The kubelet consumes Pod specifications (PodSpec) from the API
    - A PodSpec is a YAML or JSON object that describes a pod
    - kubelet PodSpecs are provided through various mechanisms (primarily through the apiserver)
    - Kubelet ensures that the containers described in PodSpecs are running and healthy
  - Responsible for other node agents that run as pods
- kube-proxy
  - The Kubernetes network proxy runs on each node as a pod
  - Manages the iptables service mesh for services defined in the Kubernetes API
    - Ensures that pods can communicate with other pods in the cluster on any node
    - Also ensures that services correctly route to their endpoint pods
  - Can do simple TCP/UDP stream forwarding or round robin TCP/UDP forwarding across a set of backends in proxy mode
- More node agents may be deployed as pods to support additional features
  - e.g. Virtual Routers, Storage Attachers, Metrics/Log Agents



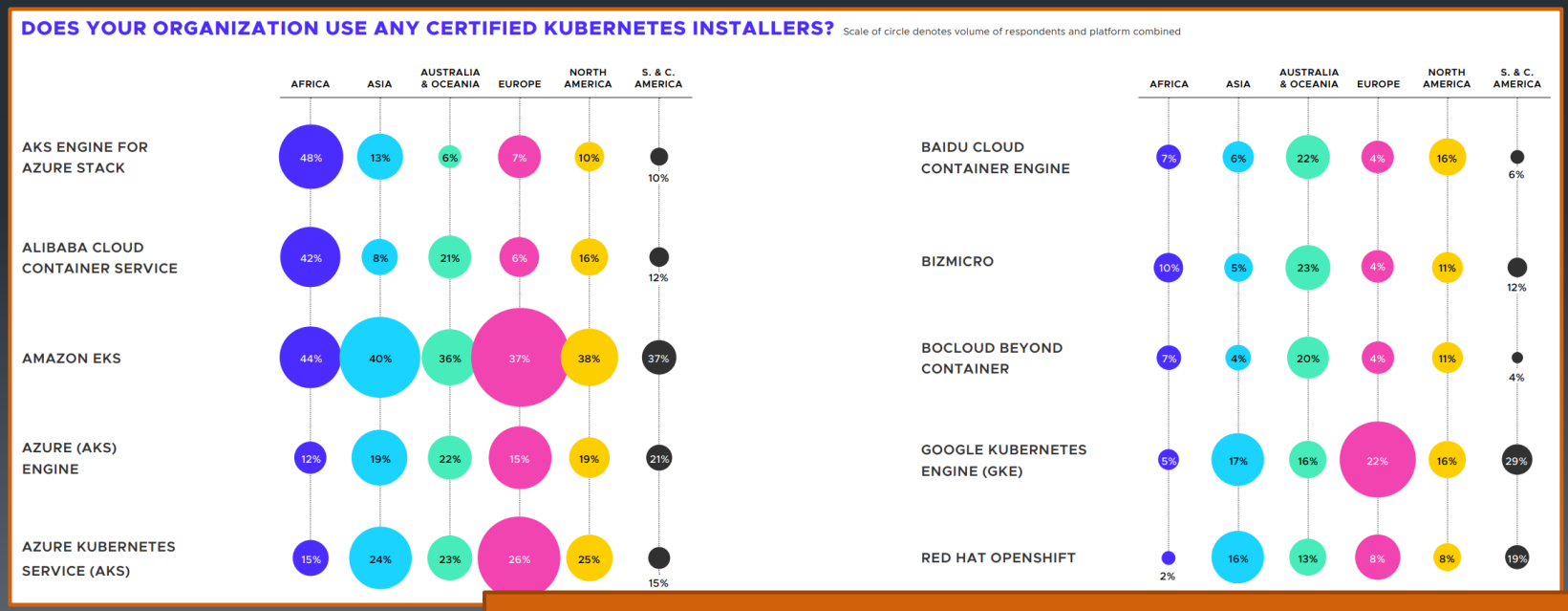
Scaled Architecture & HA Model

- Control Plane components manage the cluster and can be co-located or spread across machines
  - API Server is the central cluster state manager and event distributor
  - Scheduler and controller manager perform their own independent leader elections
    - These components change cluster state so only one instance of each should be active at any given time to prevent conflicting changes
    - Members elect new leader if current leader fails
    - The current leader holds a lease object stored in etcd in the kube-system namespace to indicate who to submit control requests to
- etcd is the distributed watchable key/value store used by the apiServer to store all cluster metadata
- Nodes run services that support pods
  - kubelet manages pods and their containers using plugins:
    - CRI: Container Runtime Interface implemented by a CRI/OCI container manager which downloads images and runs containers [required]
    - CNI: Container Network Interface implemented by a pod networking agent [required]
    - CSI: Container Storage Interface implemented by zero or more storage plugins [optional]
  - kube-proxy configures the pod service mesh



Container Orchestration Platform Adoption

- Kubernetes adoption has grown substantially over the past 3 years
- Some estimate 96% of enterprises use or are evaluating Kubernetes as of 2021
- Self-managed Kubernetes usage appears to be dropping on a percentage basis while aaS Kubernetes solutions appear to be growing
  - 79% of organizations are using some kind of certified installer



https://www.cncf.io/reports/cncf-annual-survey-2021/

K8s as a Service Solutions

- All major clouds presently offer a fairly mature Kubernetes as a Service solution
  - Open cloud console, click some buttons, use Kubernetes
- Certified Kubernetes
  - https://www.cncf.io/certification/software-conformance/
  - Software conformance ensures that every vendor's version of Kubernetes supports the required APIs, as do open source community versions
  - For organizations using Kubernetes, conformance enables interoperability from one Kubernetes installation to the next
  - CNCF runs the Certified Kubernetes Conformance Program
  - Most of the world's leading enterprise software vendors and cloud computing providers have Certified Kubernetes offerings
- KaaS certified offerings:
  - AKS – Microsoft Azure Kubernetes Service
  - EKS – Amazon EKS - Managed Kubernetes Service
  - GKE – Google Kubernetes Engine
  - IKE – IBM Cloud Kubernetes Service
  - OCE – Oracle Container Engine
  - PKS – VMware Cloud PKS
  - PMK – Platform9 Managed Kubernetes
  - RKS – Rackspace Kubernetes as a Service
  - RHOSD – RedHat OpenShift Dedicated
  - Many others

Installers

- Test/Dev Cluster
  - Minikube – the recommended method for creating a single node cluster for testing & development
  - Docker Desktop – single-node K8s cluster for development
  - Minishift – community version of OpenShift for Windows, macOS, Linux
  - MicroK8s – single-command fast install (~30 seconds) w/ plugin support
  - KinD – Kubernetes-in-Docker multi-node container-based cluster
  - Ubuntu on LXD – supports a 9-instance deployment on localhost via LXC
- Single Node Reference Installer
  - kubeadm – the reference installer for Kubernetes
- Multi Node Installers
  - kops – AWS GA, GCE/OpenStack Beta, vSphere Alpha
  - Kubespray – Ansible playbooks to install k8s on GCE, Azure, OpenStack, AWS, or baremetal (uses kubeadm)
  - Cloud Foundry Container Runtime (Kubo)
    - BOSH-based K8s installer

kubeadm

- Works with VMs, physical servers and cloud servers
- Designed to be part of a large provisioning system (used by kops for example)
- Also for easy manual provisioning
- `kubeadm init` bootstraps a K8s cluster:
  - Runs a series of pre-flight checks to validate the system state prior to making changes
  - Generates a token that additional nodes can use to register themselves with the Control Plane Node
  - Generates a self-signed CA using openssl to provision identities for each node in the cluster
    - Also used by the API server to secure communication with clients
  - Outputs a kubeconfig file for the kubelet to use to connect to the API server
    - Creates an additional kubeconfig file for administration
  - Generates Kubernetes resource manifests on the Control Plane Node in `/etc/kubernetes/manifests` for etcd, api-server, controller manager and scheduler
  - Installs any add-on components, such as DNS and kube-proxy
- Configuring kubeadm with a configuration file instead of command line flags is *recommended*
  - Some advanced features (such as using an external etcd cluster) only available as configuration file options

```
# kubeadm init --help
Run this command in order to set up the Kubernetes control plane

The "init" command executes the following phases:
```
preflight   Run pre-flight checks
certs   Certificate generation
...
kubeconfig   Generate all kubeconfig files necessary to e
...
kubelet-start   Write kubelet settings and (re)start the kub
control-plane   Generate all static Pod manifest files neces
...
etcd   Generate static Pod manifest file for local
...
upload-config   Upload the kubeadm and kubelet configuration
...
upload-certs   Upload certificates to kubeadm-certs
mark-control-plane   Mark a node as a control-plane
bootstrap-token   Generates bootstrap tokens used to join a node to a cluster
kubelet-finalize   Updates settings relevant to the kubelet after TLS bootstrap
...
addon   Install required addons for passing conformance tests
```

**`kubeadm join`**

- Uses the token to talk to the API server and securely get the root CA certificate
- Creates a local key pair; prepares a certificate signing request (CSR) and sends that off to the API server for signing
- Configures the local kubelet to connect to the API server
- Using the `--control-plane` flag enables joining the node as a Control Plane Node (Beta as of 1.15)
```

kubectl

- CLI client for Kubernetes
  - Submits requests to the Kubernetes API Server
  - Request contents can be formulated from the command line (imperatively) or using files (declaratively)
- Syntax
  - `kubectl [command] [TYPE] [NAME] [flags]`
    - `command` – operation to perform on the named resource(s)
    - `TYPE` – specifies the resource type
      - Specify singular, plural or abbreviated forms (`pod`, `pods`, `po`)
    - `NAME` – specifies the name of the resource
    - `flags` – optional flags: (`-o name` for just names, `-o yaml` for yaml output, etc.)
- Basic Resource Commands:
  - `get` – display one or many resources
  - `describe` – show details of a specific resource or group of resources
  - `create` – create a resource by filename or stdin
  - `apply` – apply a configuration change to a resource from a file or stdin
  - `delete` – delete resources by filenames, stdin, resources and names, or by resources and label selector
  - `logs` – print the logs for a container in a pod, (`-p`) for previous container
  - `attach` – attach to a running container
  - `exec` – execute a command in a container
  - `explain` – get documentation of resources
  - `api-resources` – list of all the supported resource types and their abbreviated aliases
- Autocompletion support
  - Add it to your current shell
    - `source <(kubectl completion bash)`
  - To add to your profile so it is automatically loaded in future shells :
    - `echo "source <(kubectl completion bash)" >> ~/.bashrc`

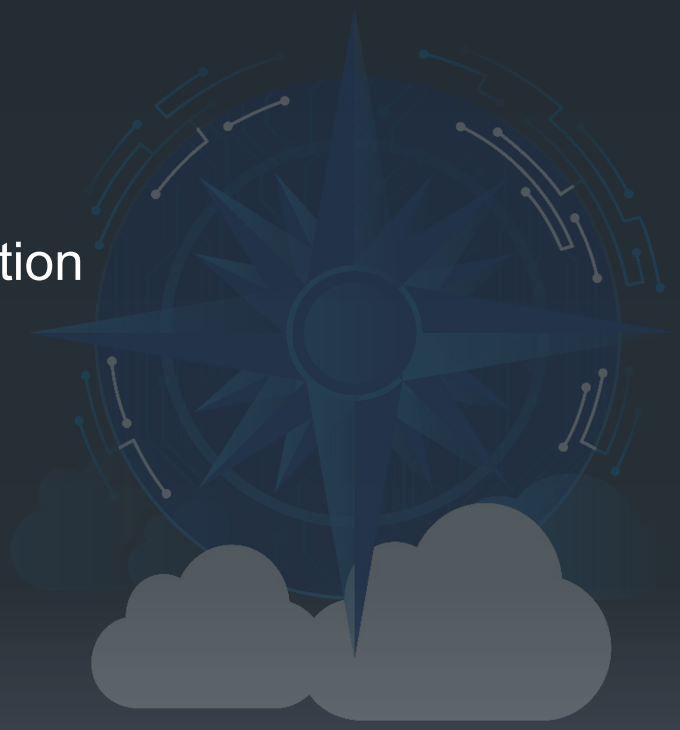
More commands: https://kubernetes.io/docs/reference/kubectl/overview/#operations

Summary

- Kubernetes is a system for managing containerized applications across multiple hosts
- Kubernetes is itself a microservice based system and some or all of it may be executed via containers
- Kubernetes-as-a-service offerings such as EKS, GKE, and others are gaining popularity
- Kubernetes can be installed easily for testing and development using the Kubernetes reference installer: kubeadm
- kubectl offers a number of commands to work with resources:
  - create
  - delete
  - describe
  - get
  - logs

Lab 3

- Kubernetes cluster setup and exploration



4: Pods: Basics

Objectives

- Define Pods and the operation of Pods on Kubernetes
- Examine the interactions between Kubernetes agents during the creation and deployment of an application pod
- Learn how to define resources with YAML files
- Understand the relationship between images, containers and Pods

Running Workloads

- Running workloads in Kubernetes is a matter of defining state with various API Resources
  - Users specify API Resources declaratively using YAML or JSON specifications
    - a.k.a. manifests or "specs"
  - Specs are sent to the API Server for the cluster to fulfill
- e.g. To run containers, users must define pods
  - Pods are atomic resources that define & place containers in a Kubernetes cluster
- `kubectl` supports basic API Object management management:
  - Creating
    - Nonidempotent: `create -f manifest.yaml`
    - Idempotent: `apply -f manifest.yaml`
  - Listing – `get pod`
  - Deleting – `delete pod xxx`
- Many other kubectl commands and options are available
  - `describe` – show object details
  - `get -o yaml` – display the yaml spec for a pod(s)
  - `logs` – display the stdout/err output of the container in the pod
  - `exec` – execute a program inside the pod's container

```
~$ cat pod.yaml

apiVersion: v1
kind: Pod
metadata:
  name: hello
spec:
  restartPolicy: Never
  containers:
  - command:
    - echo
    - hello kubernetes
    image: busybox
    name: hi

~$ kubectl apply -f pod.yaml

pod/hello created

~$ kubectl get pod

NAME    READY   STATUS      RESTARTS   AGE
hello   0/1     Completed   0          7s

~$ kubectl logs hello

hello kubernetes

~$ kubectl delete pod hello

pod "hello" deleted
```

Specification Commonalities

- YAML formatted
- Main parts
  - # Comment
  - apiVersion
    - API Group where a resource type exists
  - kind
    - Kubernetes object type
    - Pod, Deployment, Service, Namespace, etc…
  - metadata
    - Name, labels and other data describing the object
  - spec, data, etc.
    - Instructions for building the object
    - The desired state
    - Field varies depending on what is important to the kind of object being created
- Construction
  - `kubectl apply -f filename`

```yaml
# mypod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
  - name: example-pod
    image: httpd:latest
```

Imperative command equivalent example:

```
kubectl run example-pod \
--image=httpd:latest
```

apiVersions

- Resource groups that organize resource types
- Groups – 3 apiVersion group forms evolved over time
  - v1 – used in the original Kubernetes release without a group name, all GA resources existed in a single group, now referred to as "core"
  - apps/v1 – used in Kubernetes 1.1 and later, resources types are now created in groups, like "apps" or "batch", usually aligned with a SIG
  - storage.k8s.io/v1 – used in current Kubernetes releases, migrating group names to a dotted domain name form under k8s.io, such as: "storage.k8s.io" or "networking.k8s.io" or "rbac.authorization.k8s.io"
- Kubernetes clusters support multiple API versions
  - Kubernetes versions and API versions are indirectly related
- Maturity - API resources come in three levels of maturity
  - Alpha – typically disabled by default, used for features in early stage development
    - apiVersions like: v1alpha1, batch/v1alpha2 and snapshot.storage.k8s.io/v1alpha1
  - Beta – typically enabled by default, used with resources having fairly stable apis (specs) that are still being debugged, tuned, etc.
    - apiVersions like: v1beta1, batch/v1beta3, storage.k8s.io/v1beta1
  - Stable
    - apiVersions like: v1, batch/v1, storage.k8s.io/v1

```
~$ kubectl api-versions
admissionregistration.k8s.io/v1
apiextensions.k8s.io/v1
apiregistration.k8s.io/v1
apps/v1
authentication.k8s.io/v1
authorization.k8s.io/v1
autoscaling/v1
autoscaling/v2
autoscaling/v2beta1
autoscaling/v2beta2
batch/v1
batch/v1beta1
certificates.k8s.io/v1
coordination.k8s.io/v1
discovery.k8s.io/v1
discovery.k8s.io/v1beta1
events.k8s.io/v1
events.k8s.io/v1beta1
flowcontrol.apiserver.k8s.io/v1beta1
flowcontrol.apiserver.k8s.io/v1beta2
metrics.k8s.io/v1beta1
networking.k8s.io/v1
node.k8s.io/v1
node.k8s.io/v1beta1
policy/v1
policy/v1beta1
rbac.authorization.k8s.io/v1
scheduling.k8s.io/v1
storage.k8s.io/v1
storage.k8s.io/v1beta1
v1
~$
```

API Resources

- Prints out supported resources and associated shortnames, groups, and kinds
- Adding `-o wide` shows supported verbs
- Filter by namespaced / non-namespaced: `--namespaced=[true|false]`
- Filter by specific group: `--api-group=<group>ß`

```
~$ kubectl api-resources -o wide
NAME                              SHORTNAMES   APIVERSION                             NAMESPACED   KIND
bindings                                       v1                                     true         Binding
componentstatuses                 cs           v1                                     false        ComponentStatus
configmaps                        cm           v1                                     true         ConfigMap
endpoints                         ep           v1                                     true         Endpoints
events                            ev           v1                                     true         Event
limitranges                       limits       v1                                     true         LimitRange
namespaces                        ns           v1                                     false        Namespace
nodes                             no           v1                                     false        Node
persistentvolumeclaims            pvc          v1                                     true         PersistentVolumeClaim
persistentvolumes                 pv           v1                                     false        PersistentVolume
pods                              po           v1                                     true         Pod
podtemplates                                   v1                                     true         PodTemplate
replicationcontrollers            rc           v1                                     true         ReplicationController
resourcequotas                    quota        v1                                     true         ResourceQuota
secrets                                        v1                                     true         Secret
serviceaccounts                   sa           v1                                     true         ServiceAccount
services                          svc          v1                                     true         Service
mutatingwebhookconfigurations                  admissionregistration.k8s.io/v1        false        MutatingWebhookConfiguration
validatingwebhookconfigurations                admissionregistration.k8s.io/v1        false        ValidatingWebhookConfiguration
customresourcedefinitions         crd,crds     apiextensions.k8s.io/v1                false        CustomResourceDefinition
apiservices                                    apiregistration.k8s.io/v1              false        APIService
controllerrevisions                            apps/v1                                true         ControllerRevision
daemonsets                        ds           apps/v1                                true         DaemonSet
deployments                       deploy       apps/v1                                true         Deployment
replicasets                       rs           apps/v1                                true         ReplicaSet
statefulsets                      sts          apps/v1                                true         StatefulSet
```

What is YAML

- Human-readable data serialization format
- Designed to be easily mapped to data types common to most high-level languages:
  - List [array/set]
  - Associative array [hash/object/map]
  - Scalar
- Friendly to grep/Python/Perl/Ruby operations
- Used by Kubernetes and many other cloud native projects for configuration and manifest formatting
- Eschews enclosures (quotation marks, brackets, braces, open/close-tags, etc.)
- Data structure hierarchy is maintained by outline indentation
  - Spaces only, no tabs!
  - 2 spaces is convention
- All legal JSON is legal YAML (!)
  - Anywhere you can use YAML you can use JSON
  - Documents can be all YAML, all JSON or a mix
- Rhymes with camel

```
---
receipt: Invoice3344
date: 2012-08-06
total: 17.95
customer:
  given: Dorothy
  family: Gale
items:
- name: snickers
  price: 4.00
  qty: 4
- price: 1.95
  name: tax
  qty: 1
```

Using YAML and JSON


```
ubuntu@ip-172-31-46-20:~$ ls -l
total 12
-rw-rw-r-- 1 ubuntu ubuntu 317 Oct 23 04:30 pod.json
-rw-rw-r-- 1 ubuntu ubuntu 220 Oct 23 04:30 pod.yaml
-rw-rw-r-- 1 ubuntu ubuntu 219 Oct 23 04:31 pod.yamson
ubuntu@ip-172-31-46-20:~$ cat *
{
  "apiVersion": "v1",
  "kind": "Pod",
  "metadata": {
    "name":"myapp-json",
    "labels": { "app": "myapp" }
  },
  "spec": {
    "containers": [
      {
        "name":"myapp-container",
        "image":"busybox",
        "command": ["sh", "-c", "echo Hello Kubernetes! && sleep 3600"]
      }
    ]
  }
}
```

JSON

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-yaml
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command:
    - sh
    - -c
    - echo Hello Kubernetes! && sleep 3600
```

YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-yamson
  labels: { "app": "myapp" }
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: [ "sh", "-c", "echo Hello Kubernetes! && sleep 3600" ]

ubuntu@ip-172-31-46-20:~$ kubectl create -f .
pod/myapp-json created
pod/myapp-yaml created
ubuntu@ip-172-31-46-20:~$ kubectl create -f pod.yamson
pod/myapp-yamson created
ubuntu@ip-172-31-46-20:~$ kubectl get pod
NAME           READY   STATUS    RESTARTS   AGE
myapp-json     1/1     Running   0          20s
myapp-yaml     1/1     Running   0          20s
myapp-yamson   1/1     Running   0          8s
ubuntu@ip-172-31-46-20:~$ kubectl logs -l app=myapp
Hello Kubernetes!
Hello Kubernetes!
Hello Kubernetes!
ubuntu@ip-172-31-46-20:~$
```

YAML / JSON

- Kubectl accepts yaml and/or json
  - Then converts it all to json before POSTing the data to the api-server

```
ubuntu@ip-172-31-46-20:~$ kubectl get pod myapp-json -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2019-10-23T04:36:06Z"
  labels:
    app: myapp
  name: myapp-json
  namespace: default
  resourceVersion: "93071"
  selfLink: /api/v1/namespaces/default/pods/myapp-json
  uid: 8c30126b-4300-4d73-919f-be0e8bec903f
spec:
  containers:
  - command:
    - sh
    - -c
    - echo Hello Kubernetes! && sleep 3600
    image: busybox
    imagePullPolicy: Always
    name: myapp-container
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-gssng
      readOnly: true
  dnsPolicy: ClusterFirst
  enableServiceLinks: true
  nodeName: ip-172-31-46-20
  priority: 0
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext: {}
  serviceAccount: default
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
  tolerations:
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
  volumes:
  - name: default-token-gssng
    secret:
      defaultMode: 420
      secretName: default-token-gssng
```

- Kubectl can output yaml or json
  - The -o yaml switch outputs manifests in yaml
  - The -o json switch outputs manifests in json
  - The -o wide switch display normal extended output

Pod Configuration Files

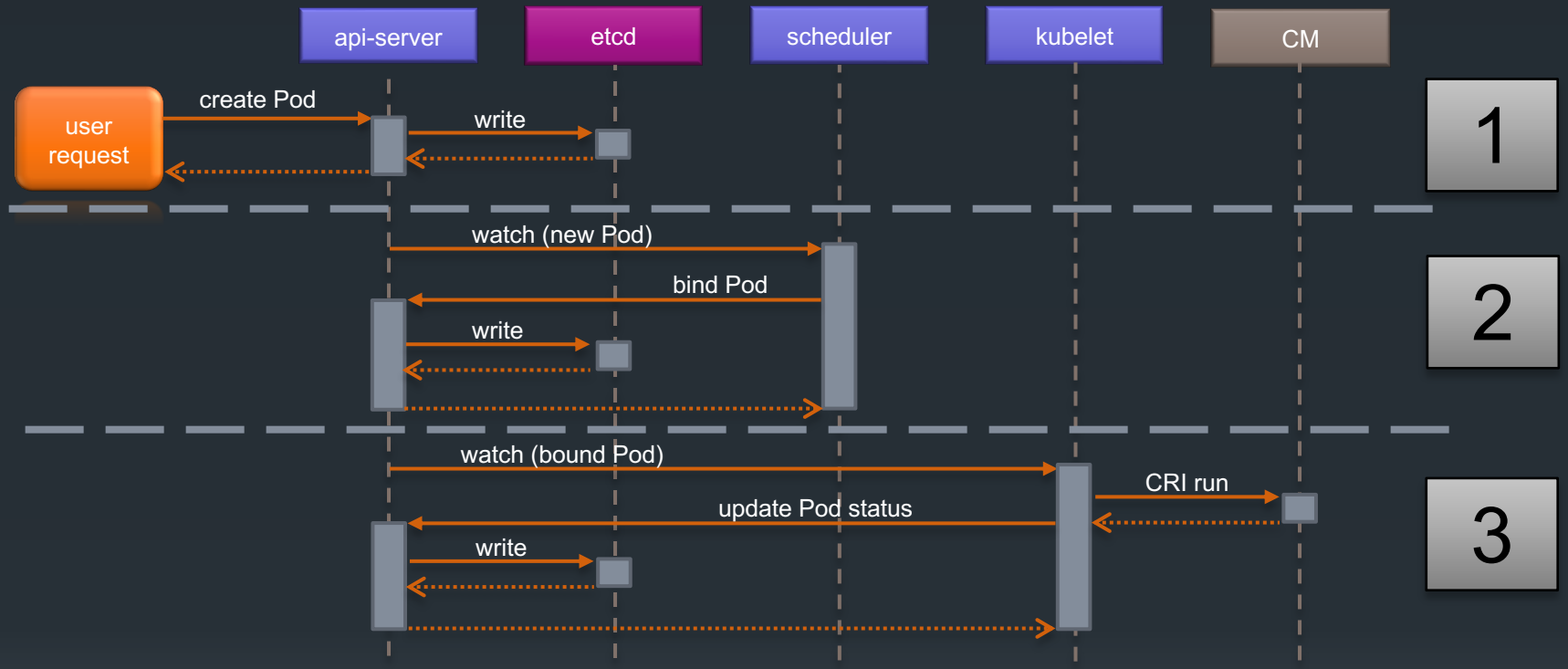
- Configured using YAML or JSON files
- The more verbose the manifest, the more effective Kubernetes becomes
- Fields for Pod configuration include (but not limited to):
  - `spec`: the pod specification
    - `volumes`[]: List of volumes that can be mounted by containers in the pod
    - `restartPolicy`: applies to all containers in pod
    - `terminationGracePeriodSeconds`: number of seconds to shutdown before kill
    - `imagePullSecrets`: secrets that provide credentials for pulling images from secure registries
    - `containers`[]: containers to run in the pod
      - `name`: Name of the container
      - `image`: Docker image name
      - `command`[]: command line (like Docker ENTRYPOINT)
      - `args`[]: entrypoint arguments (like Docker CMD)
      - `env`[]: environment vars
      - `ports`[]: port exposures
        - `containerPort`: port to expose
        - `protocol`: port protocol
      - `volumeMounts`[]: paths to mount volumes
      - `securityContext`[]: security controls (users, privileges, etc.)
      - `livenessProbe`[]: user-defined condition to determine container health (and if it needs to be restarted)
      - `readinessProbe`[]: user-defined condition to determine container readiness (when it can start taking user traffic)
      - `lifecycle`[]: actions to run after the container starts and before the container stops

```
$ cat website.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: apache
  name: website
  namespace: default
spec:
  containers:
  - image: httpd:2.2
    imagePullPolicy: IfNotPresent
    command:
    - /usr/local/apache2/bin/apachectl
    args:
    - -f
    - /usr/local/apache2/conf/httpd.conf
    name: apache
    volumeMounts:
    - mountPath: /tmp
      name: scratch-disk-space
  restartPolicy: Always
  volumes:
  - name: scratch-disk-space
    emptyDir: {}

$ kubectl apply -f website.yaml
pod/website created

$ kubectl get pod
NAME       READY     STATUS    RESTARTS   AGE
website    0/1       Pending   0          15s

$ kubectl logs website
...

$ kubectl get pod
NAME       READY     STATUS    RESTARTS   AGE
website    1/1       Running   0          30s
```

Multi-Container Pods

- Pods Model application-specific "logical hosts", enabling multiple containers (unrelated process trees) to run in a shared context
  - Therefore, a *pod* (as in a pod of whales) is a group of containers, the shared storage for those containers, and options about when and how to run the containers
  - The shared context is defined by Linux kernel namespaces and is known as the "pod sandbox"
  - Enables sharing and fast communication, e.g.:
    - An application with no git functionality consuming a directory synchronized by a separate git agent
    - A webserver with a proxy communicating over localhost
    - An application with logs handled by a logging agent
- Containers in a pod are placed and scheduled together as a unit on a Kubernetes Node, making a pod the atomic unit of scheduling
  - When the pod is replicated, all the containers are replicated by the same factor
    - e.g.: replicating a 2-container pod 3 times results in 6 total containers

Pod Creation Flow



1. user creates a Pod via the API Server and the API server writes it to etcd
2. api-server sends an asynchronous event to the scheduler in response to the scheduler's "watch" for unbound pods
   - scheduler then binds the pod to an appropriate node and saves the binding to the api-server
3. api-server sends an asynchronous event to the Kubelet bound to the pod
   - kubelet runs the container via the container manager
   - kubelet updates the status of the pod to "Running" in the API Server

Summary

- A workload is a microservice deployed as a Kubernetes Pod
- Pods are the unit of application deployment in Kubernetes
  - Pods are atomic and scheduled on a single node
- YAML files are used to specify resources and the metadata associated with them
  - Well-written manifests are key to deploying application pods
- All Kubernetes configs require a declaration of:
  - apiVersion
  - kind
  - metadata
  - spec

Lab 4

- Pods: Basics

Day 2

5. Application Configuration
6. Deployments
7. Services
8. Observability Overview

5: Application Configuration

Objectives

- Explore common configuration paradigms in modern systems
- Explore the use of ConfigMaps for configuration files & data
- Discuss creating and leveraging Secrets in configs
- Discuss creating immutable ConfigMaps and Secrets
- Examine using pod metadata in volumes and environment variables through the Downward API

Application Configuration Paradigms

- Possible configuration schemes:
  - Default Settings
    - Runs the program with the built-in options as they are coded in the binary
  - Command Line Options
    - Runs the program with modified options based on user input
  - Environment Variables
    - Runs the program with options set by the environment that is running them
  - Configuration Files
    - Runs the program using a pre-defined document containing various runtime options
    - Usually stored in some accessible location

```
ubuntu@ip-172-31-16-136:~$ docker run fluent/fluent-bit  /fluent-bit/bin/fluent-bit
Fluent Bit v1.6.10
* Copyright (C) 2019-2020 The Fluent Bit Authors
* Copyright (C) 2015-2018 Treasure Data
* Fluent Bit is a CNCF sub-project under the umbrella of Fluentd
* https://fluentbit.io

Error: No Input(s) have been defined. Aborting

ubuntu@ip-172-31-16-136:~$
```

```
ubuntu@ip-172-31-16-136:~$ docker run fluent/fluent-bit  /fluent-bit/bin/fluent-bit -i stdin -o stdout
Fluent Bit v1.6.10
* Copyright (C) 2019-2020 The Fluent Bit Authors
* Copyright (C) 2015-2018 Treasure Data
* Fluent Bit is a CNCF sub-project under the umbrella of Fluentd
* https://fluentbit.io

[2021/01/09 05:41:08] [ info] [engine] started (pid=1)
[2021/01/09 05:41:08] [ info] [storage] version=1.0.6, initializing...
[2021/01/09 05:41:08] [ info] [storage] in-memory
[2021/01/09 05:41:08] [ info] [storage] normal synchronization mode, checksum disabled, max_chunks_up=128
[2021/01/09 05:41:08] [error] [input collector] COLLECT_EVENT registration failed
[2021/01/09 05:41:08] [ info] [sp] stream processor started
```

```
1  [SERVICE]
2      Flush     5
3      Daemon    off
4      Log_Level debug
5
6  [INPUT]
7      Name  cpu
8      Tag   my_cpu
9
10 [OUTPUT]
11     Name  stdout
12     Match my*cpu
```

Application Configuration in Kubernetes

- Configurable Settings
  - Program to run
  - Program CL Options
- Environmental Settings
  - Environment Variables
  - Resource Settings
  - Volume Mounts

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: configurable
  name: configurable
spec:
  containers:
  - command:
    - ls
    - -l
    - -r
    env:
    - name: SHELL
      value: /bin/sh
    image: busybox
    name: configurable
    volumeMounts:
    - name: certs
      mountPath: /etc/ssl/certs
  volumes:
  - name: certs
    hostPath:
      path: /etc/ssl/certs
```

ConfigMaps

```
ubuntu@ip-172-31-0-249:~$ cat config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mlparams
data:
  k: "9"
  q: "42"
ubuntu@ip-172-31-0-249:~$ kubectl apply -f config.yaml
configmap/mlparams created
ubuntu@ip-172-31-0-249:~$ kubectl get cm
NAME       DATA   AGE
mlparams   2      7s
ubuntu@ip-172-31-0-249:~$ kubectl describe cm mlparams
Name:         mlparams
Namespace:    default
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"v1","data":{"k":"9","q":"42"},"k
ams","namespace":"default"}}

Data
====
q:
----
42
k:
----
9
Events:  <none>
ubuntu@ip-172-31-0-249:~$
```

```
ubuntu@ip-172-31-0-249:~$ cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: configme
spec:
  restartPolicy: Never
  containers:
  - name: test
    image: k8s.gcr.io/busybox
    command: [ "/bin/sh", "-c", "env" ]
    env:
    - name: ML_PARAM_K
      valueFrom:
        configMapKeyRef:
          name: mlparams
          key: k
ubuntu@ip-172-31-0-249:~$ kubectl apply -f pod.yaml
pod/configme created
ubuntu@ip-172-31-0-249:~$ kubectl get pod
NAME       READY   STATUS      RESTARTS   AGE
configme   0/1     Completed   0          5s
ubuntu@ip-172-31-0-249:~$ kubectl logs configme
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_SERVICE_PORT=443
HOSTNAME=configme
SHLVL=1
HOME=/root
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP_PROTO=tcp
ML_PARAM_K=9
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
PWD=/
KUBERNETES_SERVICE_HOST=10.96.0.1
ubuntu@ip-172-31-0-249:~$
```

- Enables the Kubernetes API as a (configuration) data source!
- ConfigMap resources
  - meant to house configuration data in key/value pairs
- Keyed ConfigMap data can be mounted into a pod
  - Files
    - The key is the file name / the value is the contents of the file
  - Environment variables
    - The key can be the name of the environment variable (or you can set your own name) / the value is the value of the environment variable
    - Environment variables can be used as command line args

ConfigMaps with complex data

- These configuration artifacts should be decoupled from image content in order to keep containerized applications portable
- The ConfigMap resource provides mechanisms to inject containers with configuration data while keeping containers agnostic of Kubernetes
- ConfigMap can be used to store fine-grained information like individual properties or coarse-grained information like entire config files or JSON blobs

```
# ConfigMap of prometheus.yml config file
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |
    scrape_configs:
    - job_name: 'kubernetes-apiservers'
      kubernetes_sd_configs:
      - role: endpoints
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels: [__meta_kubernetes_namespace, __meta_kubernetes_service_name]
        action: keep
        regex: default;kubernetes;https
    - job_name: 'kubernetes-nodes'
...
```

```
scrape_configs:
- job_name: 'kubernetes-apiservers'
  kubernetes_sd_configs:
  - role: endpoints
  scheme: https
  tls_config:
    ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
    #    insecure_skip_verify: true
  bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
  relabel_configs:
  - source_labels: [__meta_kubernetes_namespace, __meta_kubernetes_service_name]
    action: keep
    regex: default;kubernetes;https
-  job_name: 'kubernetes-nodes'
...
```

Secrets

- Kubernetes objects used to hold sensitive information
  - Ex. passwords, OAuth tokens, SSH keys
- Two ways to store data:
  - data: you base64 encode data
  - stringData: base64 encoded for you
- Ideally encrypted in etcd through api-server settings
- Secrets type is used for programmatic secret processing (implies key names and contents expected):
  - Opaque - Normal arbitrary data (most ConfigMap-like)
  - Docker-Registry (kubernetes.io/dockerconfigjson) – docker/oci registry token
  - TLS (kubernetes.io/tls) - public key certificate and private key pair
  - Other Secret types:
    - kubernetes.io/service-account-token - In-cluster credential for containers
    - kubernetes.io/dockercfg - Serialized ~/.dockercfg file
    - kubernetes.io/dockerconfigjson - Serialized ~/.docker/config.json file
    - kubernetes.io/basic-auth - Stores basic authentication information
    - kubernetes.io/ssh-auth - Stores SSH Keys
    - bootstrap.kubernetes.io/token - Authentication token used to allow new nodes to join the cluster)

```
$ echo -n 'admin' | base64
YWRtaW4=
$ echo -n 'password' | base64
cGFzc3dvcmQ=

$ vi dev-db-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: dev-db-secret-v1
type: Opaque
data:
  password: cGFzc3dvcmQ=
  username: YWRtaW4
```

```
$ kubectl create secret --help
Create a secret using specified subcommand.

Available Commands:
  docker-registry Create a secret for use with a Docker registry (kubernetes.io/dockerconfigjson)
  generic         Create a secret from a local file, directory or literal value (Opaque)
  tls             Create a TLS secret (kubernetes.io/tls)
```

Protecting Secrets

- Contents of secrets are only encoded in base64
  - Not encrypted by default in etcd
- Access to secrets should be restricted using RBAC
- Secrets can be further protected by encrypting them in etcd
  - With encryption at rest enabled, secrets are encrypted on write within etcd
  - `kube-apiserver` must be configured for encryption at rest with `--encryption-provider-config`
  - KMS-based systems that transmit secret data at startup and store secrets in memory are ideal
    - i.e. AWS Secrets Manager or Hashicorp Vault

```
~$ kubectl get secret secret1 -o yaml

apiVersion: v1
data:
  mykey: bXlkYXRh
kind: Secret
metadata:
  name: secret1
  namespace: default
type: Opaque
```

Secret stored in etcd without encryption enabled

```
~$ etcdctl get /registry/secrets/default/secret1 --cacert /etc/kubernetes/pki/etcd/ca.crt
--cert /etc/kubernetes/pki/etcd/server.crt --key /etc/kubernetes/pki/etcd/server.key

/registry/secrets/default/secret1
k8s


v1Secret
secret1default"*$02833b1c-fc3d-413a-a9b0-8c9ff4b9b4ef2ňz
kubectl-createUpdatevňFieldsV1:.
,{"f:data":{".":{},"f:mykey":{}},"f:type":{}}B
mykeymydataOpaque"
```

```
~$ etcdctl get /registry/secrets/default/secret1 --cacert /etc/kubernetes/pki/etcd/ca.crt
--cert /etc/kubernetes/pki/etcd/server.crt --key /etc/kubernetes/pki/etcd/server.key

/registry/secrets/default/secret1
k8s:enc:aescbc:v1:key1:
!*j.BjO^3QoTT:AvU1J36(K$L¤|Ku.N
_                    Fdp
._rx?$n
```

Secret stored in etcd with encryption enabled

Creating ConfigMaps/Secrets

- Can be created from directories, files, or key-value pairs supplied from the command line
  - `--from-file`
    - Creates a data key from the supplied file
      - File name becomes the key, contents become the value
      - Key can be overridden with: `--from-file=<my-key-name>=<path-to-file>`
    - If a directory is supplied, each file creates a new key in the resulting ConfigMap/Secret
  - `--from-env-file`
    - Parses every line in a file as separate data keys in the resulting ConfigMap/Secret
  - `--from-literal=`
    - Creates a data key for each key-value supplied by the user in the command line
  - Each `--from-file` or `--from-literal` arguments creates a new key
- As a yaml file using `kubectl apply -f`
  - Secrets must have the values base64 encoded

```
$ cat app.conf
key=value1
property=config
another.property=configValue

$ cat ui.conf
color.one=blue
color.two=white
font.headings=sans-serif
font.body=serif

$ kubectl create configmap config-v1 \
--from-file=/home/user/configmap/

configmap/config created

$ kubectl describe configmap config

Name:         config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
app.conf:
----
key=value1
property=config
another.property=configValue

ui.conf:
----
color.one=blue
color.two=white
font.headings=sans-serif
font.body=serif

Events:  <none>
```

ConfigMap/Secret Volumes

- The referenced Secret/ConfigMap must exist in the same namespace as the pod
  - If the kubelet can not fetch the Secret/ConfigMap:
    - It will report an event
    - The kubelet will periodically retry
    - Once the secret/configMap is created and fetched by the kubelet, it will create the volume and start the associated pod
- When a Secret/ConfigMap being consumed in a volume is updated, projected keys are eventually updated as well
- `imagePullSecret` pass a secret that contains an image registry credential to Kubelet so it can pull a private image on behalf of your pod
  - Never mounted inside pod/container filesystem
- You can specify the permission mode bits of all Secret/ConfigMap files (or part of the files a Secret/ConfigMap will have)
  - Otherwise 0644 is used by default

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secret-file-pod
  labels:
    secretpod: db-client
spec:
  volumes:
  - name: dev-db-secret
    secret:
      secretName: dev-db-secret-v1
      defaultMode: 400
  - name: app-config
    configMap:
      name: config-v1
  containers:
  - name: secret-file-container
    image: redis
    imagePullSecrets:
    - name: myregistrykey
    volumeMounts:
    - name: dev-db-secret
      readOnly: true
      mountPath: "/etc/secret-volume"
    - name: app-config
      mountPath: "/etc/app/config"
```

Secrets/ConfigMaps as Environment Variables

- Secrets appear as normal environment variables containing the base-64 decoded values of the secret data
  - ConfigMaps are always plain text
- Add an environment variable to each container for each key you wish to consume
- When a Secret/ConfigMap being consumed in ENVs is updated, projected values are NOT updated!
  - Requires a pod restart or rolling update
- Use `envFrom` to define all of a Secret/ConfigMap's data as container environment variables
  - *Keys that are considered invalid environment variable names will be skipped!*
    - The pod will be allowed to start
    - An `InvalidVariableNames` event will contain the list of invalid keys that were skipped

```
apiVersion: v1
kind: Pod
metadata:
  name: env-pod
spec:
  containers:
  - name: env-container
    image: redis
    env:
    - name: SECRET_USERNAME
      valueFrom:
        secretKeyRef:
          name: dev-db-secret-v1
          key: username
    - name: SECRET_PASSWORD
      valueFrom:
        secretKeyRef:
          name: dev-db-secret-v1
          key: password
    - name: CONFIG_VAR
      valueFrom:
        configMapKeyRef:
          name: config-v1
          key: property
    envFrom:
    - configMapRef:
        name: multi-config
```

Immutable ConfigMaps/Secrets

- Set individual Secrets and ConfigMaps as immutable
  - Prevents updates to ConfigMaps and Secrets that may break applications running in pods
  - Improves cluster performance
    - Reduces load on kube-apiserver by closing watches for Secrets and ConfigMaps
  - Requires `immutable: true` in manifest to set a ConfigMap or Secret to be immutable
- Update strategy for Immutable ConfigMaps and Secrets:
  - Create a new ConfigMap with updated contents
  - Update any pods or their controllers that are using the old ConfigMap with the new version
    - This approach retains the old version (for a time) of the ConfigMap/Secret so it is available in case a rollback is necessary

```
$ kubectl get cm config-v1
apiVersion: v1
kind: ConfigMap
metadata:
  name: config-v1
data:
  app.conf: |
    key: value1
    property: config
    another.property: configValue
  ui.conf: |
    color.one: blue
    color.two: white
    font.headings: sans-serif
    font.body: serif
immutable: true

$ kubectl get cm config-v2
apiVersion: v1
kind: ConfigMap
metadata:
  name: config-v2
data:
  app.conf: |
    key: value1
    property: config
    another.property: configValue
  ui.conf: |
    color.one: red
    color.two: white
    font.headings: serif
    font.body: serif
immutable: true
```

Downward API

- Allows containers to consume information about themselves or the cluster without using the kubectl client or API server
- Ways of exposing Pod and Container fields through:
  - Environment variables
  - DownwardAPIVolumeFiles
- Several basic data fields like names, labels, and annotations are available via the `fieldRef` key:
  - `spec.nodeName`
  - `status.hostIP`
  - `metadata.name`
  - `metadata.namespace`
  - `status.podIP`
  - `spec.serviceAccountName`
  - `metadata.uid`
  - `metadata.labels`
    - When appended with `['<key>']` a single value of a pod's label defined by <key>
  - `metadata.annotations`
    - When appended with `['<key>']` a single value of a pod's annotation defined by <key>
- Container resource Information available via `resourceFieldRef`:
  - Container CPU limits and request
  - Memory limits and memory request

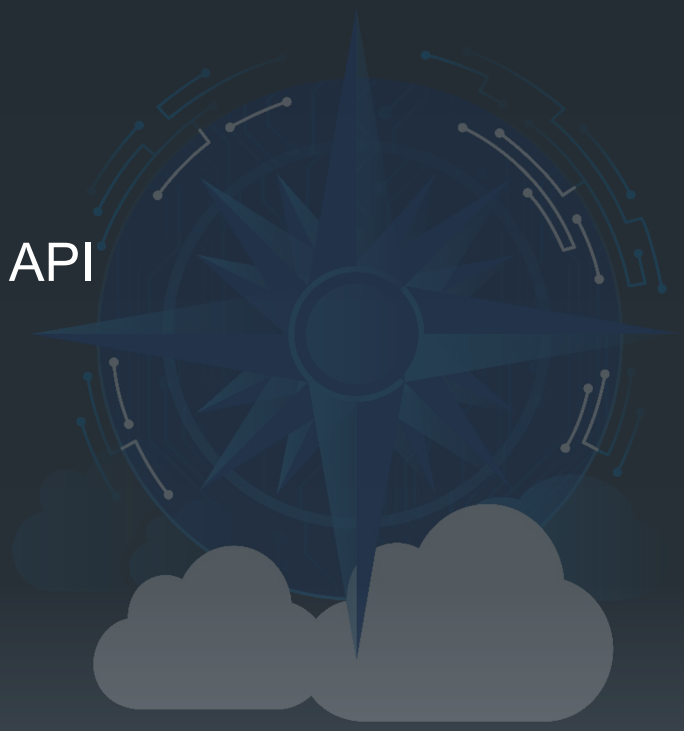
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: downward-api-pod
  labels:
    cluster: test-cluster1
    rack: rack-22
spec:
  containers:
  - name: downward-api-container
    image: busybox
    env:
    - name: MY_POD_IP
      valueFrom:
        fieldRef:
          fieldPath: status.podIP
    volumeMounts:
    - name: dapi-info
      mountPath: /etc/dapinfo
  volumes:
  - name: dapi-info
    downwardAPI:
      items:
      - path: "labels"
        fieldRef:
          fieldPath: metadata.labels
      - path: "mem_limit"
        resourceFieldRef:
          containerName: downward-api-container
          resource: limits.memory
          divisor: 1Mi
```

Summary

- Applications in modern systems are configurable using command-line arguments, environment variables, and configuration files
- ConfigMaps are used for the following:
  - Populate the value of environment variables
  - Set command-line arguments in a container
  - Populate config files in a volume
- Secrets can be used to pass sensitive data to Pods
  - kubelet can use secrets for logging into registries
- Downward API can be used to deliver dynamic metadata to applications running in Pods as files or environment variables

Lab 5

- ConfigMaps, Secrets, and Downward API



6: Deployments

Objectives

- Understand labels and selectors and how they are used to identify resources in a Kubernetes cluster
- Explore the types, roles, and workings of Controllers
  - Deployments, Replica Sets
- Understand how to perform application rollouts

Controllers

- Controllers are control loops that watch the cluster's state
  - States are defined by the user using API objects the controllers are designed to watch
- A controller watches and changes a resources' current state to match the desired state
  - e.g. A Pod is deleted, a new pod is created to satisfy current state == desired state
  - e.g. A resource is scaled up or down, pods will be created or terminated to match the desired state

```
$ cat redis-deploy.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis
spec:
  replicas: 2
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
      - image: redis
        name: redis
```

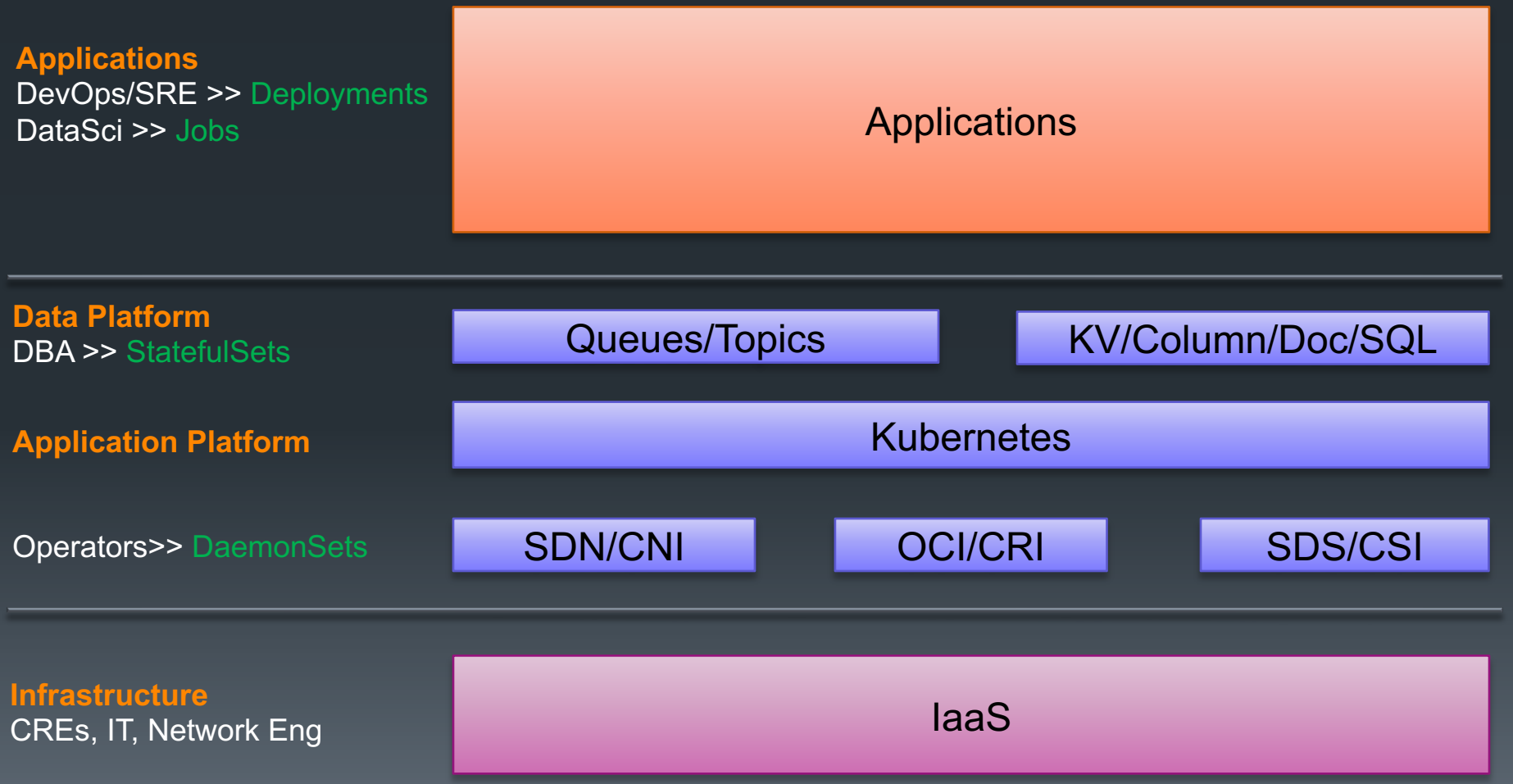
```
$ kubectl apply -f redis-deploy.yaml
deployment.apps/redis created

$ kubectl get deploy,rs,pod
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/redis   2/2     2            2           9s

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/redis-85d47694f4     2         2         2       9s

NAME                         READY   STATUS    RESTARTS   AGE
pod/redis-85d47694f4-h8l4w   1/1     Running   0          9s
pod/redis-85d47694f4-n7lgv   1/1     Running   0          9s

$ kubectl delete pod redis-85d47694f4-h8l4w
pod "redis-85d47694f4-h8l4w" deleted

$ kubectl get deploy,rs,pod
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/redis   2/2     2            2           35s

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/redis-85d47694f4     2         2         2       35s

NAME                         READY   STATUS    RESTARTS   AGE
pod/redis-85d47694f4-hmx78   1/1     Running   0          10s
pod/redis-85d47694f4-n7lgv   1/1     Running   0          35s

$ kubectl scale deploy redis --replicas=3
deployment.apps/redis scaled

$ kubectl get deploy,rs,pod
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/redis   3/3     3            3           54s

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/redis-85d47694f4     3         3         3       54s

NAME                         READY   STATUS    RESTARTS   AGE
pod/redis-85d47694f4-hmx78   1/1     Running   0          29s
pod/redis-85d47694f4-lkh65   1/1     Running   0          4s
pod/redis-85d47694f4-n7lgv   1/1     Running   0          54s
```

Deployment created

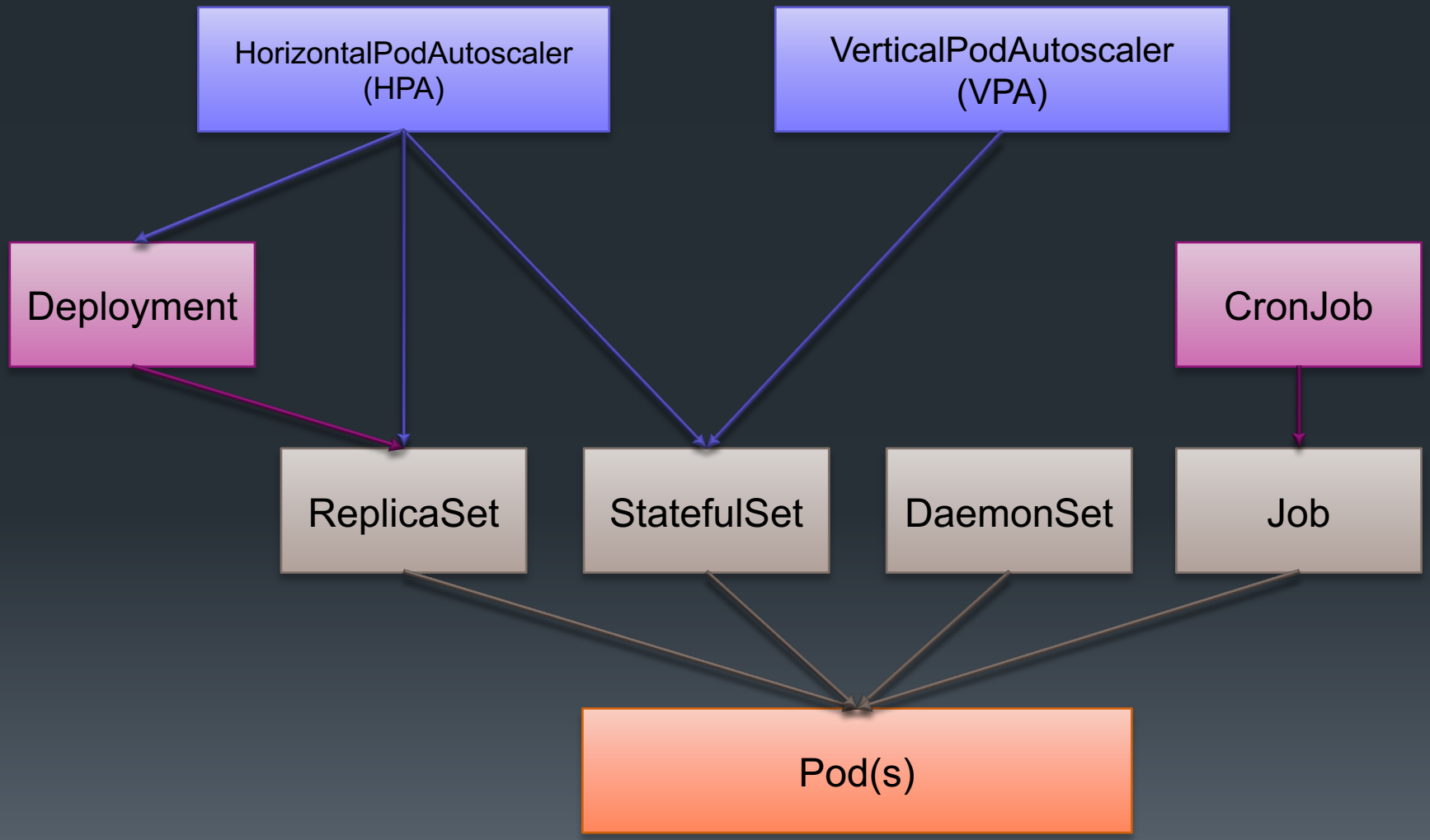
Pod deleted
Pod created to satisfy number of replicas desired

Deployment scaled

Division of Labor in a Cloud Native World



Workload Resource Types



Labels and Annotations

- Labels are key value pairs associated with a resource
  - Any resource can have label metadata
    - pods, nodes, services, namespaces, etc.
  - Resources can be “selected” using labels
  - Label keys and values must consist of letters, numbers, dashes and underbars
- Annotations can contain arbitrary key value data
  - Like labels, annotations are metadata and can be associated with any resource
  - Unlike labels annotations can store values containing any character sequence
  - Not used by Kubernetes label selectors
  - Third party tools often read resource annotations to parameterize their behavior
    - e.g. a CNI plugin may read a pod annotation to decide which network to place the pod on

```
$ kubectl get pod --show-labels
NAME   READY   STATUS    RESTARTS   AGE     LABELS
web    1/1     Running   0          8m35s   team=marketing
$ kubectl label pod web app=configurator
pod/web labeled
$ kubectl get pod --show-labels
NAME   READY   STATUS    RESTARTS   AGE     LABELS
web    1/1     Running   0          9m20s   app=configurator,team=marketing
$ kubectl label pod web team=sales --overwrite=true
pod/web labeled
$ kubectl get pod --show-labels
NAME   READY   STATUS    RESTARTS   AGE     LABELS
web    1/1     Running   0          9m46s   app=configurator,team=sales
$ kubectl label pod web app-
pod/web labeled
$ kubectl get pod --show-labels
NAME   READY   STATUS    RESTARTS   AGE   LABELS
web    1/1     Running   0          10m   team=sales
$ kubectl get pod -l team=sales
NAME   READY   STATUS    RESTARTS   AGE
web    1/1     Running   0          10m
$ kubectl get pod -l team=marketing
No resources found in default namespace.
$

$ kubectl get pods -o custom-columns=NAME:metadata.name,ANNO:.metadata.annotations
NAME   ANNO
web    <none>
$ kubectl annotate pod web web-fwd="using:ingress1##@delivery==priv:repeat^ save"
pod/web annotated
$ kubectl get pods -o custom-columns=NAME:metadata.name,ANNO:.metadata.annotations
NAME   ANNO
web    map[web-fwd:using:ingress1##@delivery==priv:repeat^ save]
$ kubectl annotate pod web web-fwd-
pod/web annotated
$ kubectl get pods -o custom-columns=NAME:metadata.name,ANNO:.metadata.annotations
NAME   ANNO
web    <none>
$
```

Selectors

- There are several options you can use to select on labels:
  - `=` or `==` –select keys with values equal to the string on the right: `name=apache`
  - `!=` – select keys with values that do not equal the string on the right: `environment!=test`
  - `in` – select resources whose labels have keys with values in this set: `'tier in (web,app)'`
  - `notin` – select resources whose labels have keys with values not in this set: `'tier notin (lb,app)'`
  - <Key name> – use a key name only to select resources whose labels contain this key: `tier`
- Multiple selectors can be separated by commas and are logically “AND”ed
  - There is no logical OR operator
- Many kubectl subcommands offer the `-l` switch to specify label selectors (`get`, `delete`, `logs`, etc.)

```
$ kubectl get deploy -l 'team in (scarif,bespin)' --show-labels
NAME                     READY   UP-TO-DATE   AVAILABLE   AGE    LABELS
configurator-web         3/3     3            3           50s    app=configurator,release=stable,team=scarif,tier=frontend
configurator-web-canary  3/3     3            3           84s    app=configurator,release=canary,team=scarif,tier=frontend
rideshare-web            3/3     3            3           119s   app=rideshare,release=stable,team=bespin,tier=frontend
$ kubectl get deploy -l 'team in (scarif,bespin),app=rideshare' --show-labels
rideshare-web   3/3     3            3           2m39s   app=rideshare,release=stable,team=bespin,tier=frontend
$ kubectl logs -l app=rideshare,tier=frontend
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 10.36.0.3. Set the 'ServerName'
directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 10.40.0.2. Set the 'ServerName'
directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 10.40.0.3. Set the 'ServerName'
directive globally to suppress this message
$ kubectl delete deploy -l 'team in (scarif,bespin),release=canary'
deployment.apps "configurator-web-canary" deleted
```

Deployment Config

- A Deployment controller provides declarative updates for Pods and ReplicaSets
- You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments
- Keys related to the Deployment kind
  - **`replicas`** - defines the desired number of pods
  - **`selector`** - specifies a label selector for the Pods targeted by this deployment
    - Tells the Replica Set which pods to watch
  - **`strategy`**
    - `type`: Recreate or RollingUpdate (default)
    - `maxSurge`: max number of pods that can be scheduled above the desired number of pods
    - `maxUnavailable`: max number of pods that can be unavailable during the update
  - **`minReadySeconds`** - optional field that specifies the minimum number of seconds for which a newly created Pod should be ready without any of its containers crashing, for it to be considered available
  - **`revisionHistoryLimit`** - optional field that specifies the number of old ReplicaSets to retain to allow rollback
  - **`template`** - defines a template to launch a new pod
    - Contains the same elements defined for pod configs
- Selector values need to match the labels values specified in the Pod template
  - A Deployment's label selector is **immutable** after it gets created

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8sapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: k8sapp
      deployment: development
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  template:
    metadata:
      labels:
        app: k8sapp
        deployment: development
    spec:
      containers:
      - name: k8sapp-container
        image: k8sapp:v3.3
        ports:
        - containerPort: 80
```

Simple Deployment / RS Walk Through

```
$ kubectl apply -f mydep.yaml
deployment.apps/myapp created

$ $ kubectl get deploy,rs,po
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/myapp   3/3     3            3           76s

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/myapp-1072152842   3         3         3       76s

NAME                         READY   STATUS    RESTARTS   AGE
pod/myapp-1072152842-fn5cg   1/1     Running   0          72s
pod/myapp-1072152842-k4pwl   1/1     Running   0          72s
pod/myapp-1072152842-v5wm2   1/1     Running   0          72s

$ kubectl describe deployment myapp
Name:                   myapp
Namespace:              default
CreationTimestamp:      Wed, 01 Aug 2019 12:48:13 -0800
Labels:                 team=kamino,app=myapp,deployment=dev
Annotations:            deployment.kubernetes.io/revision=1
Selector:               app=myapp,deployment=dev
Replicas:               3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:               app=myapp
                        deployment=dev
  Containers:
   myapp-container :
    Image:              logagg
    Port:               80/TCP
    Environment:        <none>
    Mounts:             <none>
  Volumes:              <none>
Conditions:
  Type                  Status        Reason
  ----                  ------        ------
  Available             True          MinimumReplicasAvailable
 Progressing            True          NewReplicaSetAvailable
OldReplicaSets:         <none>
NewReplicaSet: myapp-1072152842 (3/3 replicas created)
Events:
  FirstSeen     LastSeen   Count  From                    SubObjectPath  Type     Reason            Message
  ---------     --------   -----  ----                    -------------  -------- ------            -------
  3m            3m         1      {deployment-controller}                Normal   ScalingReplicaSet  Scaled up replica set myapp-1072152842 to 3
```

```
$ cat mydep.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  labels:
    team: kamino
    app: myapp
    deployment: dev
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
      deployment: dev
  template:
    metadata:
      labels:
        app: myapp
        deployment: dev
    spec:
      containers:
      - name: myapp-container
        image: myapp:v2.8
        ports:
        - containerPort: 80
```

Updating Resources with kubectl

- `kubectl apply` - Applies a configuration from a spec file or STDIN
  - Creates resources from spec files if they do not exist
  - Applies changes to existing resources from spec files
  - Takes no action if no update was provided for an existing resource
- `kubectl patch` - Uses a jsonpath or a file to change a specific portion of a select resource spec
- `kubectl edit` - Opens the selected resource's spec in a local editor
- `kubectl set` - Make specific changes to certain application settings, including:
  - Deployment pod template Environment variables, container images, resources, or selectors
  - The serviceAccounts used by a resource
  - The subject in a RoleBinding or ClusterRolebinding
- `kubectl label` - Adds or removes labels from the target resource's metadata
  - Existing labels can be updated using the --override option
  - Labels can be removed by selecting the key followed trailing minus
    - `kubectl label pod mypod mylabel-`

```
user@ubuntu:~$ kubectl run nginx-front --image nginx
pod/nginx-front created
user@ubuntu:~$ kubectl describe pod nginx-front | head
Name:         nginx-front
Namespace:    default
Priority:     0
Node:         labsys/192.168.229.143
Start Time:   Thu, 23 Jul 2020 15:45:16 -0700
Labels:       run=nginx-front
Annotations:  <none>
Status:       Running
IP:           10.36.0.4
IPs:
user@ubuntu:~$ kubectl patch pod nginx-front -p '{"metadata": {"labels": {"run":
"nginx-front", "purpose": "test"}}}'
pod/nginx-front patched
user@ubuntu:~$ kubectl describe pod nginx-front | head
Name:         nginx-front
Namespace:    default
Priority:     0
Node:         labsys/192.168.229.143
Start Time:   Thu, 23 Jul 2020 15:45:16 -0700
Labels:       purpose=test
              run=nginx-front
Annotations:  <none>
Status:       Running
IP:           10.36.0.4
```

Rollouts

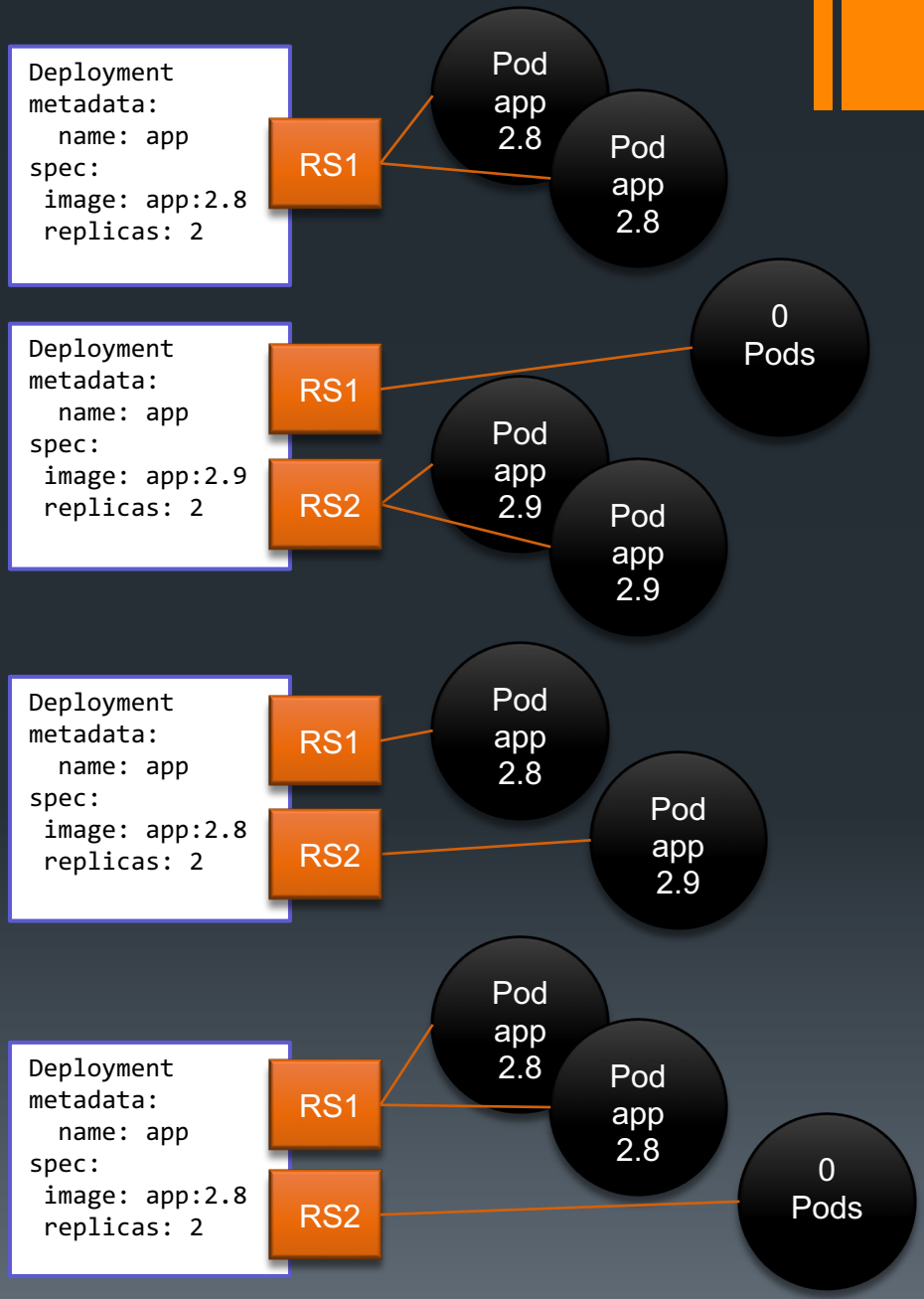
- `kubectl apply -f appdeploy.yaml`

- `kubectl set image deploy/app \ mypod=app:2.9 --record`
  (can also use `edit`, `patch`, or `apply`)

- `kubectl rollout undo deploy/app`
- `kubectl rollout pause deploy/app`

- `kubectl rollout resume deploy/app`

- `kubectl rollout history deploy/app`
- `kubectl rollout status deploy/app`



Summary

- Controllers define and enforce state in the cluster, e.g.
  - Run a number of pods somewhere in the cluster
  - Run a number of pods to some completion target
- Kubernetes uses sets of key/value pairs called labels to identify resources
  - Some Kubernetes resources use labels to target other resources
- Deployments/ReplicaSets are key components in the orchestration of horizontally-scaled microservices
- RSs use kubelets to perform on-node monitoring and restart tasks

Lab 6

- Deployments and ReplicaSets

7. Services

Objectives

- Explore the nature of Kubernetes services
- Discuss the various types of Services:
  - ClusterIPs, NodePorts, LoadBalancers
- Understand the role of kube-proxy in relation to Services
- Review the IPTables manipulations made by the proxy
- Explain the use of external load balancers with K8s
- Examine Kubernetes DNS
- Learn about using DNS policies in pod configs

Services

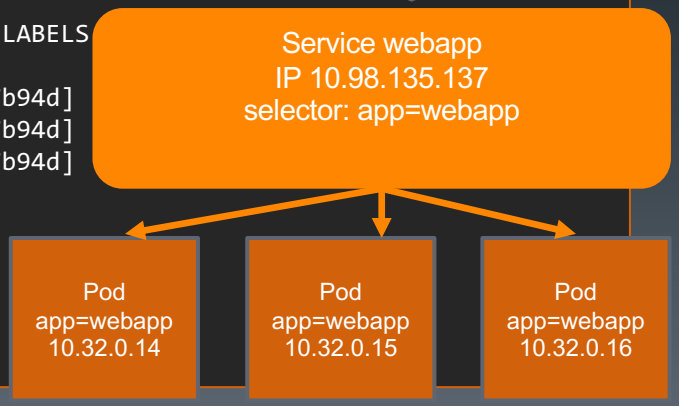
- A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them
  - Pods targeted by a service are identified by a label that matches service's Selector
- Provides a single, stable network endpoint clients use to access groups of pods
  - Pods IPs are unreliable (changing depending on which node they are assigned in a cluster)
  - Services provide a virtual IP and/or cluster-internal DNS name to route traffic to pods

```
$ kubectl get svc -o wide
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE   SELECTOR
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP   11h   <none>
webapp       ClusterIP   10.98.135.137   <none>        80/TCP    22s   app=webapp

$ kubectl get deploy,rs -o wide
NAME                     READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES         SELECTOR
deployment.apps/webapp   3/3     3            3           12s   nginx        nginx:latest   app=webapp

NAME                                DESIRED   CURRENT   READY   AGE   CONTAINERS   IMAGES         SEL
replicaset.apps/webapp-fdcf7b94d    3         3         3       12s   nginx        nginx:latest   app
template-hash=fdcf7b94d

$ kubectl get pod -o=custom-columns='NAME:.metadata.name, IP:.status.podIP, LABELS
NAME                      IP           LABELS
webapp-fdcf7b94d-lnqw5    10.32.0.16   map[app:webapp pod-template-hash:fdcf7b94d]
webapp-fdcf7b94d-lws5m    10.32.0.14   map[app:webapp pod-template-hash:fdcf7b94d]
webapp-fdcf7b94d-xfgpv    10.32.0.15   map[app:webapp pod-template-hash:fdcf7b94d]

$ curl -L http://10.98.135.137
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

webapp service abstracts pods with label app=webapp

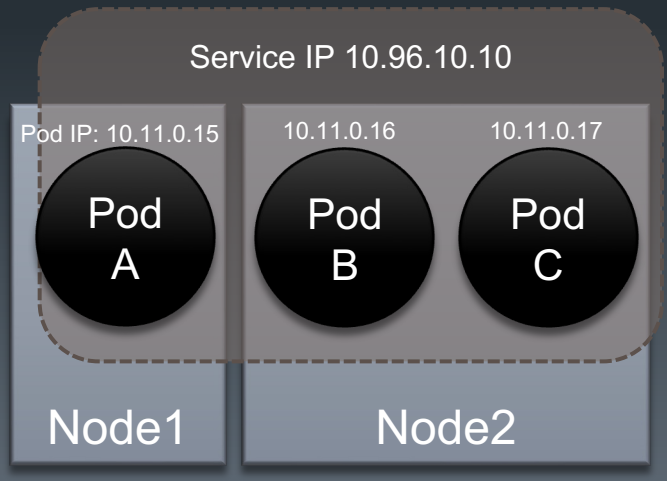
traffic

curl the service IP reaches a webapp pod



Service Types

- Services can be exposed in different ways by specifying a type in the ServiceSpec
  - Each method offers a different level of exposure
- `ClusterIP` – exposes the Service on an internal IP in the cluster
  - Only accessible by containers in the cluster and cluster nodes
- `NodePort` – exposes the Service on the same port of each selected Node in the cluster using NAT
  - Accessible by anybody who has the node address and node port
- `LoadBalancer` – assigns an external address to the Service and LB
  - External address provided by a cloud backend (e.g. AWS ELB) or advertising agent (e.g. MetalLB)
  - Accessible by clients who have the external address
- `ExternalName` – maps a Service to an external DNS name
  - Backs a service IP with an external DNS name rather than Pod IPs
  - Enables apps to use internal Kubernetes DNS names while taking advantage of non-containerized backends and services



Service Config spec

- `type` - type of exposed service
  - `ClusterIP` – [Default] uses kube-proxy and cluster based virtual IP (available only inside the cluster)
  - `NodePort` – exposes the service on the same port on each node of the cluster (for external access)
  - `LoadBalancer` – uses an external load balancer, typically with a public IP and port (AWS ELB, etc.)
- `port` - The list of ports that are exposed by this service
  - Need not be the same as the port of the actual pods
- `targetPort` – The port exposed by the backend Pods
  - By default the targetPort will be set to the same value as the port field
  - Can be a string, referring to the name of a port in the backend Pods
- `selector` - This service will route traffic to pods having labels matching this selector
  - If empty, all pods are selected (!)
  - Pods that qualify for a service's selector are called endpoints
    - Stored inside an `endpoint` resource that shares the name of the service
    - Pods must be in a `Ready` state in order to be included as endpoints for a service
- `clusterIP` - IP is usually assigned by the master and is the IP address of the service
  - If specified, it will be allocated to the service if it is unused or else creation of the service will fail
  - Valid values are `None` (headless service), empty string (""), or a valid IP address
- `sessionAffinity` - Supports "ClientIP" and "None"
  - Used to maintain session affinity
  - Defaults to None

```
apiVersion: v1
kind: Service
metadata:
  name: testweb
  labels:
    app: websvc
spec:
  type: ClusterIP
  ports:
  - protocol: TCP
    port: 8080
    targetPort: 80
  selector:
    run: testweb
```

```
$ kubectl get endpoints testweb

NAME     ENDPOINTS                                      AGE
testweb  10.32.0.2:80,10.46.0.3:80,10.44.0.6:80  1h
```

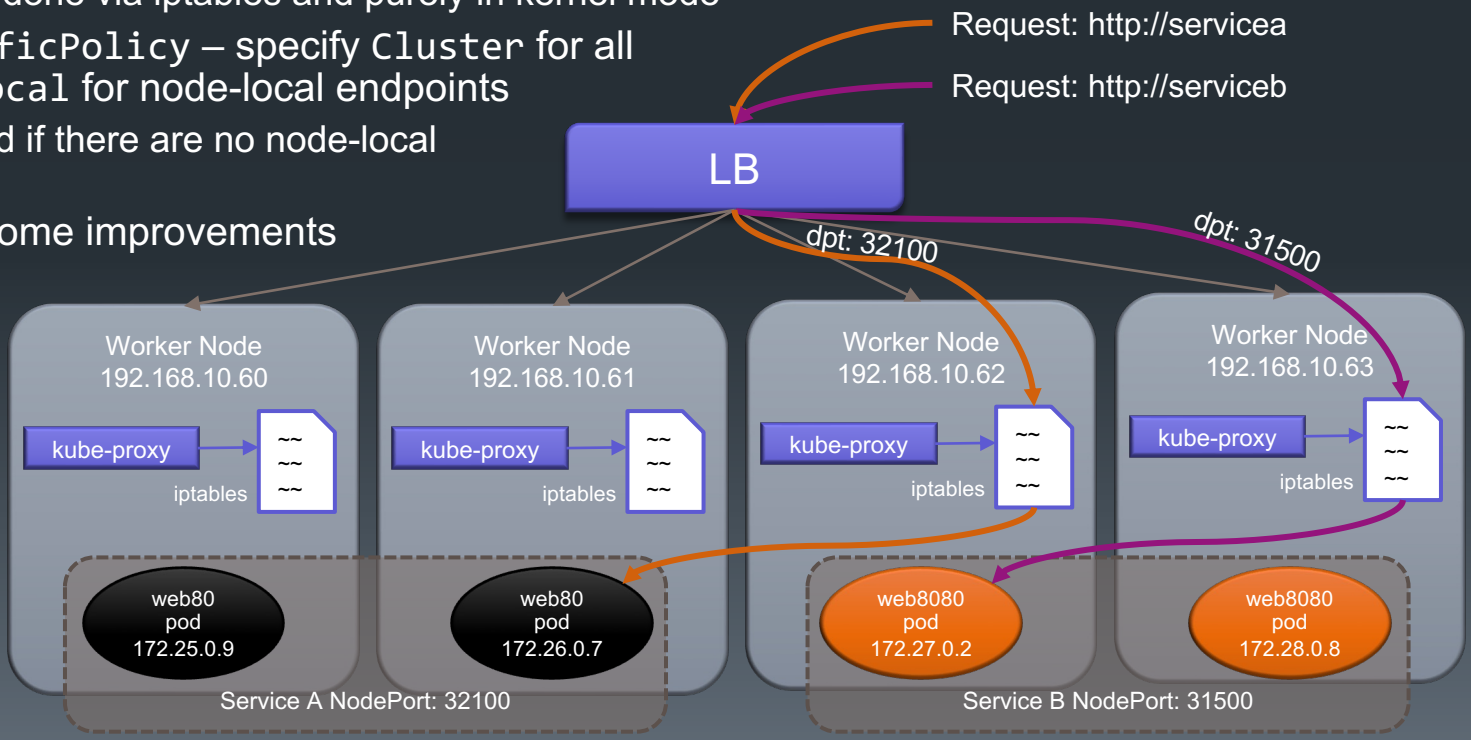
Node Port Details

- The Kubernetes master allocates a port from a flag-configured range
  - Default: 30000-32767
- Each Node will proxy that port (the same port number on every Node)
- If you want a specific port number, you can specify a value in the `nodePort` field
  - The value you specify must be in the configured range for node ports
    - May not conflict with existing ports
- This gives developers the freedom to set up their own load balancers, to configure cloud environments that are not fully supported by Kubernetes
  - Or even to just expose one or more nodes' IPs directly
- Node Port Services also have all of the features of ClusterIP

```
$ cat nodeport.yaml

apiVersion: v1
kind: Service
metadata:
  name: testweb-np
spec:
  type: NodePort
  ports:
    - port: 80
      protocol: TCP
  selector:
    run: testweb

$ kubectl apply -f nodeport.yaml

$ kubectl get service
NAME         CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes   10.96.0.1        <none>        443/TCP        1d
testweb-np   10.111.101.166   <nodes>       80:30143/TCP   32s

$ kubectl describe service testweb-np
Name:                     testweb-np
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 run=testweb
Type:                     NodePort
IP:                       10.111.101.166
Port:                     <unset>      80/TCP
NodePort:                 <unset>      30143/TCP
Endpoints:                10.32.0.4:80,10.32.0.5:80,10.32.0.6:80
Session Affinity:         None
Events:                   <none>
```

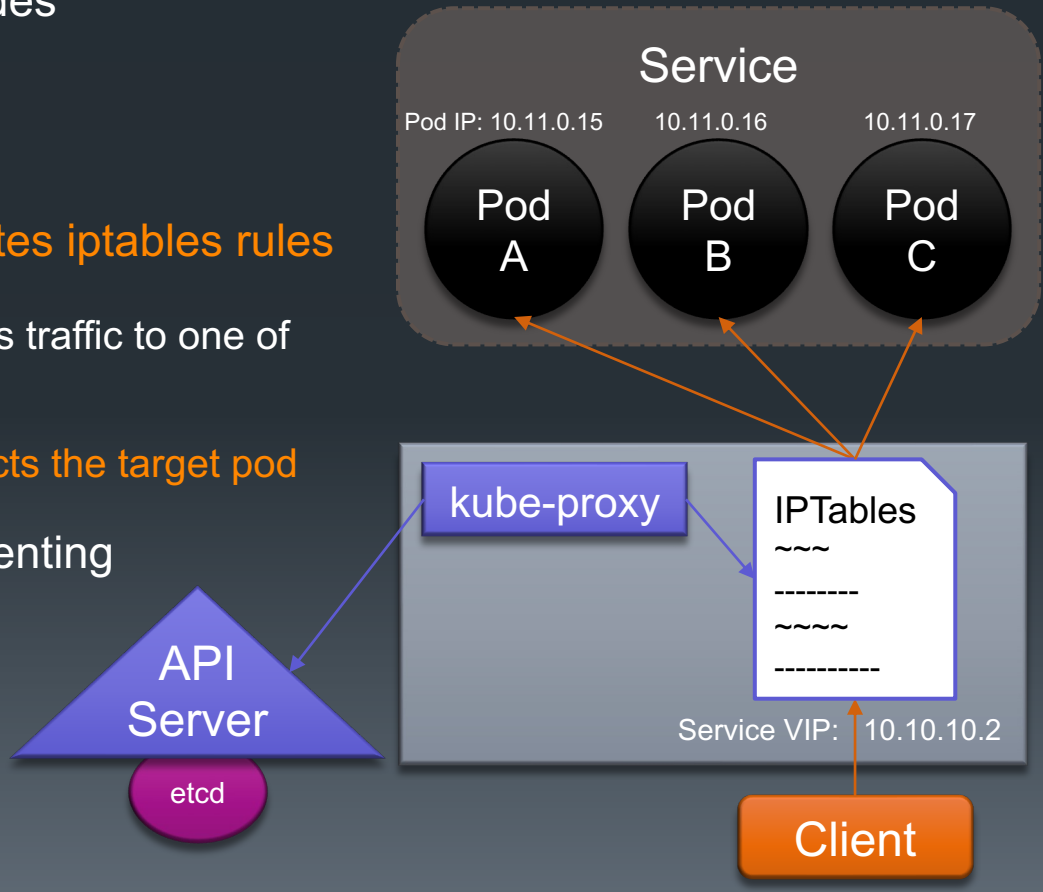
Load Balancers

- LoadBalancer Services wire up cloud load balancers:
  - GCE's ForwardingRules, AWS's ELB (NLB beta as of v1.15), Azure Load Balancer
- LoadBalancer Services outside the cloud can use alpha `loadBalancerClass`
- NodePort Services in combination with an **external load balancer**
- Pod IPs are determined by the SDN (and are thus not real), so the load balancer must rely on Node IP addresses (which are real) and their IP tables for DNAT to route traffic to the pods
  - If IPs are real, `spec.allocateLoadBalancerNodePorts` can be set to `false`
- Most traffic will get double-bounced on the network
  - Nodes essentially act as routers
  - All forwarding is done via iptables and purely in kernel mode
  - `internalTrafficPolicy` – specify `Cluster` for all endpoints or `Local` for node-local endpoints
    - Traffic dropped if there are no node-local endpoints!
- Ingress provides some improvements



Kube-Proxy

- Routing Mesh
  - Pods frequently need to communicate with other services
  - The Kube-Proxy runs on each node in the cluster providing a routing mesh to map services to implementation pods
    - Service IPs work everywhere kube-proxy is running
  - The proxy operates in one of 3 modes (`--proxy-mode`):
    - User Mode Proxy [old]
    - IPTables [default]
    - IPVS [new, eventual default]
  - In IPTables mode Kube-Proxy creates iptables rules for every service
    - Produces a virtual endpoint that routes traffic to one of the service's target pods
      - Typically a ClusterIP, aka Virtual IP (VIP)
    - Iptables loadbalancing randomly selects the target pod to forward to
  - Kube-Proxy identifies pods implementing the service via a selector
  - All Kube-Proxies watch the API Server for service changes, updating their iptables as needed



NodePort & ClusterIP Service


```
$ kubectl create deploy testweb --image=nginx --replicas=3
deployment.apps/testweb created

$ kubectl apply -f svc.yaml
service/testweb created

$ kubectl get deploy,po,rs
NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/testweb   3/3     3            3           48s

NAME                               READY   STATUS    RESTARTS   AGE
pod/testweb-58d5cbcdcf-9f7qz       1/1     Running   0          40s
pod/testweb-58d5cbcdcf-9xctw       1/1     Running   0          40s
pod/testweb-58d5cbcdcf-vkq2w       1/1     Running   0          48s

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/testweb-58d5cbcdcf    3         3         3       48s

$ kubectl get service
NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes           ClusterIP   10.96.0.1       <none>        443/TCP        7h6m
testweb              NodePort    10.97.67.39     <none>        80:30320/TCP   49s

$ kubectl get endpoints testweb
NAME      ENDPOINTS                                   AGE
testweb   10.32.0.4:80,10.44.0.5:80,10.39.0.9:80      53s

$ curl -s http://10.97.67.39 | head -4
...
<title>Welcome to nginx!</title>

$ curl -s http://10.32.0.4 | head -4
...
<title>Welcome to nginx!</title>
```

$ vim svc.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: testweb
  labels:
    svc: testweb
spec:
  type: NodePort
  sessionAffinity: ClientIP
  sessionAffinityConfig:
    clientIP:
      timeoutSeconds: 600
  ports:
  - port: 80
  selector:
    app: testweb
```

NodePort & ClusterIP Service Nat

```
user@ubuntu:~/svc$ sudo iptables -nL -t nat

Chain PREROUTING (policy ACCEPT)
target          prot opt source              destination
KUBE-SERVICES   all  --  0.0.0.0/0           0.0.0.0/0       /* kubernetes service portals */
DOCKER          all  --  0.0.0.0/0           0.0.0.0/0       ADDRTYPE match dst-type LOCAL


Chain KUBE-NODEPORTS (1 references)
target                    prot opt source              destination
KUBE-MARK-MASQ            tcp  --  0.0.0.0/0           0.0.0.0/0     /* default/testweb: */ tcp dpt:30320
KUBE-SVC-EG66NKXDU7X2QZTA tcp  --  0.0.0.0/0           0.0.0.0/0     /* default/testweb: */ tcp dpt:30320


Chain KUBE-SERVICES (2 references)
target                         prot opt source         destination
KUBE-SVC-XGLOHA7QRQ3V22RZ      tcp  --  0.0.0.0/0      10.106.125.172  /* kube-system/kubernetes-dashboard: cluster IP */ tcp dpt:80
KUBE-SVC-NPX46M4PTMTKRN6Y      tcp  --  0.0.0.0/0      10.96.0.1       /* default/kubernetes:https cluster IP */ tcp dpt:443
KUBE-SVC-TCOU7JCQXEZGVUNU      udp  --  0.0.0.0/0      10.96.0.10      /* kube-system/kube-dns:dns cluster IP */ udp dpt:53
KUBE-SVC-ERIFXISQEP7F7OF4      tcp  --  0.0.0.0/0      10.96.0.10      /* kube-system/kube-dns:dns-tcp cluster IP */ tcp dpt:53
KUBE-SVC-EG66NKXDU7X2QZTA      tcp  --  0.0.0.0/0      10.97.67.39     /* default/testweb: cluster IP */ tcp dpt:80
KUBE-NODEPORTS                 all  --  0.0.0.0/0      0.0.0.0/0       /* kubernetes service nodeports; NOTE: this must be the last rule in this chain */
ADDRTYPE match dst-type LOCAL


Chain KUBE-SVC-EG66NKXDU7X2QZTA (1 references)
target                    prot opt source      destination
KUBE-SEP-7KX2LIG5LIB5UVVO  all  --  0.0.0.0/0   0.0.0.0/0   /* default/testweb: */ recent: CHECK seconds: 600 reap name: KUBE-SEP-7KX2LIG5LIB5UVVO side: source mask: 255.255.255.255
KUBE-SEP-TSJSFJG2GUXROSTI  all  --  0.0.0.0/0   0.0.0.0/0   /* default/testweb: */ recent: CHECK seconds: 600 reap name: KUBE-SEP-TSJSFJG2GUXROSTI side: source mask: 255.255.255.255
KUBE-SEP-RY3VYSR3AYX2BBO2  all  --  0.0.0.0/0   0.0.0.0/0   /* default/testweb: */ recent: CHECK seconds: 600 reap name: KUBE-SEP-RY3VYSR3AYX2BBO2 side: source mask: 255.255.255.255
KUBE-SEP-7KX2LIG5LIB5UVVO  all  --  0.0.0.0/0   0.0.0.0/0       /* default/testweb: */ statistic mode random probability 0.33332999982
KUBE-SEP-TSJSFJG2GUXROSTI  all  --  0.0.0.0/0   0.0.0.0/0       /* default/testweb: */ statistic mode random probability 0.50000000000
KUBE-SEP-RY3VYSR3AYX2BBO2  all  --  0.0.0.0/0   0.0.0.0/0       /* default/testweb: */


Chain KUBE-SEP-7KX2LIG5LIB5UVVO (1 references)
target          prot opt source      destination
KUBE-MARK-MASQ  all  --  10.32.0.4   0.0.0.0/0   /* default/testweb: */
DNAT            tcp  --  0.0.0.0/0   0.0.0.0/0   /* default/testweb: */ recent: SET name: KUBE-SEP-7KX2LIG5LIB5UVVO side: source mask: 255.255.255.255 tcp to:10.32.0.4:80


Chain KUBE-SEP-TSJSFJG2GUXROSTI (1 references)
target          prot opt source      destination
KUBE-MARK-MASQ  all  --  10.32.0.5   0.0.0.0/0   /* default/testweb: */
DNAT            tcp  --  0.0.0.0/0   0.0.0.0/0   /* default/testweb: */ recent: SET name: KUBE-SEP-TSJSFJG2GUXROSTI side: source mask: 255.255.255.255 tcp to:10.44.0.5:80


Chain KUBE-SEP-RY3VYSR3AYX2BBO2 (1 references)
target          prot opt source      destination
KUBE-MARK-MASQ  all  --  10.32.0.6   0.0.0.0/0   /* default/testweb: */
DNAT            tcp  --  0.0.0.0/0   0.0.0.0/0   /* default/testweb: */ recent: SET name: KUBE-SEP-RY3VYSR3AYX2BBO2 side: source mask: 255.255.255.255 tcp to:10.39.0.9:80
```

Statistic IPTables module: apply a rule with a probability

IPTables module "recent" enables session affinity

Reverse NAT for hairpin traffic

Services & DNS

- Services defined in a cluster (including the DNS server itself) are resolvable in DNS by name
  - Creating a service named “web” in namespace “area51” in a cluster with the suffix “groomlake.local” would produce the FQDN:
    - `web.area51.svc.groomlake.local`
- Pod receive a tailored /etc/resolv.conf configured to use the Cluster DNS with a search list including the Pod’s namespace and cluster’s domain suffix
- Services within the pod’s name space can be resolved directly:
  - In the area51 ns, web would resolve to web.area51.svc.groomlake.local
  - web.area77 (in any ns) would resolve to web.area77.svc.groomlake.local

```
ubuntu@ip-172-31-29-148:~$ kubectl attach -n default -it client
Defaulting container name to client.
Use 'kubectl describe pod/client -n default' to see all of the containers in this pod.
If you don't see a command prompt, try pressing enter.
/ # cat /etc/resolv.conf
nameserver 10.96.0.10
search default.svc.cluster.local svc.cluster.local cluster.local us-west-1.compute.internal
options ndots:5
/ # cat /etc/hostname
client
/ # cat /etc/hosts
# Kubernetes-managed hosts file.
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
fe00::0 ip6-mcastprefix
fe00::1 ip6-allnodes
fe00::2 ip6-allrouters
10.44.0.6       client
/ # ping -c 1 kube-dns.kube-system
PING kube-dns.kube-system (10.96.0.10): 56 data bytes
```

```
ubuntu@ip-172-31-29-148:~$ kubectl get service -n kube-system
NAME       TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
kube-dns   ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   46h
ubuntu@ip-172-31-29-148:~$
```

```
~$ kubectl get pods,svc,deploy -o wide ### The Testing Environment

NAME                          READY   STATUS    RESTARTS   AGE   IP           NODE              NOMINATED NODE   READINESS GATES
pod/dns-test                  1/1     Running   0          47m   10.32.0.7    ip-172-31-3-67    <none>           <none>
pod/website                   1/1     Running   0          48m   10.32.0.6    ip-172-31-3-67    <none>           <none>
pod/website-6c475f8d5c-5pcnp  1/1     Running   0          79s   10.32.0.9    ip-172-31-3-67    <none>           <none>

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE   SELECTOR
service/kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        4d    <none>
service/mysvc        NodePort    10.102.43.253   <none>        80:31741/TCP   44m   run=website

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES   SELECTOR
deployment.apps/website   1/1     1            1           100s  nginx        nginx    run=website
```

DNS availability

Type	curl	200 OK from a Pod In the same Namespace	200 OK from a Pod in a different Namespace	200 OK from a cluster Node	200 OK From external nodes that can reach the cluster node network
Pod IP	10.32.0.6	Yes	Yes	Yes	No
Standalone Pod DNS (Type A)	10-32-0-6.default.pod.cluster.local	Yes	Yes	No	No
Standalone Pod DNS (Type A)	10-32-0-6.default.pod	Yes	Yes	No	No
Service DNS (Type A)	mysvc.default.svc.cluster.local	Yes	Yes	No	No
Service DNS (Type A)	mysvc.default.svc	Yes	Yes	No	No
Service DNS (Type A)	mysvc.default	Yes	Yes	No	No
Service DNS (Type A)	mysvc	Yes	No	No	No
Service ClusterIP	10.102.43.253	Yes	Yes	Yes	No
Pod created by Deployment exposed by service DNS (Type A)	10-32-0-9.mysvc.default.svc.cluster.local	Yes	Yes	No	No
Pod created by Deployment exposed by service DNS (Type A)	10-32-0-9.mysvc.default.svc	Yes	Yes	No	No
NodePort Service via Node IP	172.31.3.67:31741	Yes	Yes	Yes	Yes
NodePort Service via localhost IP	127.0.0.1:31741	No	No	Yes	No
NodePort Service via Node DNS (Type A)	ip-172-31-3-67.ec2.internal:31741	Yes	Yes	Yes	Yes

Summary

- Services provide an abstraction for delivering traffic to Pods
  - ClusterIP services expose Pods to internal clients only
  - NodePort services expose Pods to external clients and internal client Pods
  - LoadBalancer services expose Pods to Internet, external and internal clients
    - External load balancers, whether LoadBalancer services or manually-configured, typically double-bounce application traffic
- Kubernetes implements a simple Linux native networking model using standards-driven technologies
  - Virtual IPs
  - iptables
  - DNS
  - Etc.
- Kubernetes provides DNS for services and pods
- PodSpecs can be modified to use specific DNS configs independent of cluster settings

Lab 7

- Services and Service Selectors

8. Observability Overview

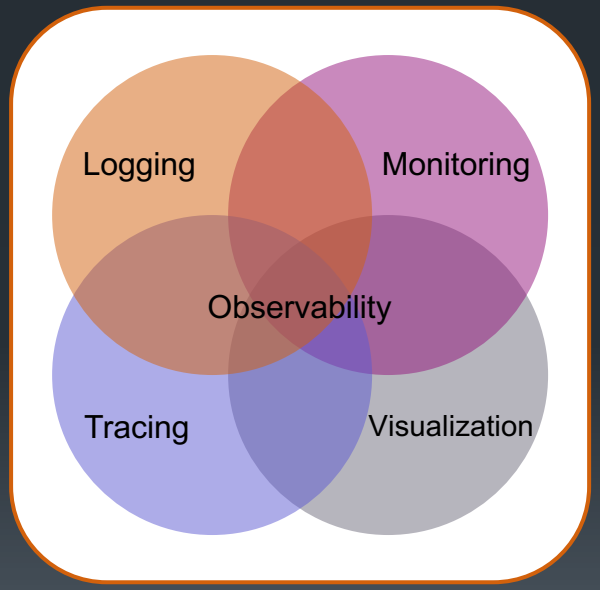
Objectives

- Define the aspects of observability
- Contrast metrics, tracing and logging
- Discuss distributed tracing
- Understand container logging
- Examine methods to monitor and aggregate metrics about Nodes and Pods
- Describe the Kubernetes metrics implementation
- Discuss the Metrics APIs
- Discover ways to autoscale applications:
  - Horizontal Pod Autoscaler (HPA)
  - Vertical Pod Autoscaler (VPA)

Observability

- Observability
  - What is required in order to "see" the behavior of applications and infrastructure
- Distributed Systems
  - In distributed systems, to "see" your application running you need data from many places
  - "Seeing" behavior in distributed systems requires the collection of data from many physical systems and logical layers
  - It hard to "see" what is happening in distributed systems with traditional, single host oriented solutions
- Cloud native observability is based on three independent sets of data:
  - Metrics
  - Logs
  - Traces
- Visualization
  - Tools that display data with charts, graphs, tables and other means are called visualization systems

Metrics, logging and tracing are all cross cutting concerns (best implemented in one way cluster wide)



Weaveworks
https://www.weave.works/technologies/monitoring-kubernetes-with-prometheus/

Tracing

- A trace tracks an execution path through a system from component to component
  - Tracing is critical in distributed systems (distributed tracing) as application features used by clients typically require multiple services interacting with each other
- Tracing can be considered a specialized use of logging to record information about a program's execution
  - There is not always a clear distinction between tracing and other forms of logging
  - However, specific “tracing” systems have become prevalent in cloud native systems
- This information is used by operators, developers, SREs and devops staff for:
  - Introspection (examining internal processes for the purpose of understanding)
  - Diagnostics (understanding system problems)
  - Debugging (repairing application errors)
  - Activity Monitoring (tracking customer activity)
- Tracing is a cross-cutting concern
  - All cloud native applications need tracing, so best to build once and implement everywhere
- Tracing can be implemented in many ways:
  - Custom application logging code or custom code written to a specific tracer tools
  - Pluggable Libraries allow developers to code tracing abstractly: OpenTracing with plug ins:
    - Jaeger
    - Open Zipkin
    - Lightstep
  - Proxies can be used to intercept, report on, and instrument networ interactions without app support
    - Linkerd
    - Envoy
    - Conduit

```
// 2) Demonstrate simple OpenTracing instrumentation
parent := tracer.StartSpan("Parent")
for i := 0; i < 20; i++ {
        parent.LogEvent(fmt.Sprintf("Starting child #%d", i))
        child := tracer.StartSpan("Child", opentracing.ChildOf(parent.Context()))
        time.Sleep(50 * time.Millisecond)
        child.Finish()
}
parent.LogEvent("A Log")
parent.Finish()
```

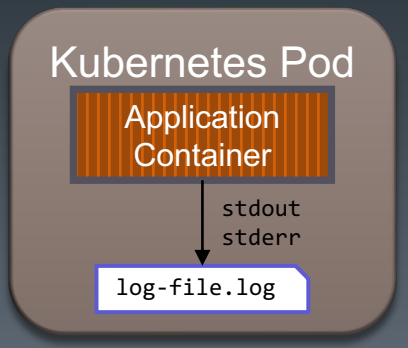
Logging

- A log is a record of discrete events reported by a system component
- Software systems may write log data in a structured or informal way
  - User place a new order
  - NewOrder, UID 56749, BUY 800 GSCO MKT
  - 7,56749,1,800,GSCO,8
  - 0x26 0xFA 0xA4 0x11 0x3D …
- Log data allows applications and other system elements to report rich information associated with a wide range of problems and activities
- Log data is typically timestamped

```
-- Logs begin at Fri 2018-08-24 21:50:18 PDT, end at Wed 2018-08-29 12:06:41 PDT. --
Aug 24 21:50:18 ubuntu systemd-journald[65140]: Runtime journal (/run/log/journal/) is 22.3M, max 19.8M, 0B free.
Aug 24 21:50:18 ubuntu systemd-journald[65140]: Journal started
Aug 24 21:50:17 ubuntu systemd[1]: systemd-logind.service: Watchdog timeout (limit 3min)!
Aug 24 21:50:17 ubuntu dhclient[4288]: DHCPREQUEST of 192.168.91.132 on ens33 to 192.168.91.254 port 67 (xid=0x4ed41778)
Aug 24 21:50:17 ubuntu systemd[1]: systemd-journald.service: Watchdog timeout (limit 3min)!
Aug 24 21:50:17 ubuntu dhclient[4288]: DHCPACK of 192.168.91.132 from 192.168.91.254
Aug 24 21:50:17 ubuntu systemd[4854]: Time has been changed
Aug 24 21:50:17 ubuntu dhclient[4288]: bound to 192.168.91.132 -- renewal in 893 seconds.
Aug 24 21:50:18 ubuntu systemd[1]: systemd-logind.service: Service has no hold-off time, scheduling restart.
Aug 24 21:50:18 ubuntu systemd[1]: systemd-timesyncd.service: Service has no hold-off time, scheduling restart.
Aug 24 21:50:18 ubuntu systemd[1]: Stopped Network Time Synchronization.
Aug 24 21:50:18 ubuntu systemd[1]: Starting Network Time Synchronization...
Aug 24 21:50:18 ubuntu systemd[1]: Stopped Login Service.
Aug 24 21:50:18 ubuntu systemd[1]: Starting Login Service...
Aug 24 21:50:18 ubuntu systemd[1]: Starting Flush Journal to Persistent Storage...
Aug 24 21:50:18 ubuntu systemd[1]: Started udev Kernel Device Manager.
Aug 24 21:50:18 ubuntu systemd[1]: Started Network Time Synchronization.
Aug 24 21:50:18 ubuntu systemd[1]: Started Flush Journal to Persistent Storage.
Aug 24 21:50:18 ubuntu systemd-logind[65153]: New seat seat0.
Aug 24 21:50:18 ubuntu systemd[1]: Started Login Service.
Aug 24 21:50:18 ubuntu systemd-logind[65153]: Watching system buttons on /dev/input/event0 (Power Button)
Aug 24 21:50:18 ubuntu systemd[1]: Started User Manager for UID 1000.
Aug 24 21:50:18 ubuntu systemd-logind[65153]: New session c2 of user user.
Aug 24 21:50:16 ubuntu systemd-timesyncd[65150]: Synchronized to time server 91.189.94.4:123 (ntp.ubuntu.com).
Aug 24 21:50:16 ubuntu systemd[1]: Time has been changed
Aug 24 21:50:16 ubuntu systemd[4854]: Time has been changed
```

Container Logging

- Everything a containerized application writes to stdout and stderr is handled and redirected somewhere by a container engine
  - Docker's default config sends logs to the container's ephemeral file system layer
- If a container restarts, the kubelet keeps one terminated container with its logs (`kubectl logs -c myapp --previous=true mypod`)
- If a pod is evicted from a Node, all corresponding containers are also evicted, along with their logs
- `kubectl logs` - Retrieves the select pod's logs from its host container runtime
  - Entirely dependent on the container runtime logging engine
  - Application must also be configured to send logs to the container logging engine
  - Users can adjust log output according to their needs
    - `--since` scopes returned logs to a certain time period (Minutes or Hours)
    - `--max-log-requests` limits how many lines of logs are returned
    - `--previous` or `-p` displays the logs from any previous versions of the container
    - `--container` or `-c` displays logs from a specific container in a pod
    - `--all-containers` displays logs from all containers in a pod



Kubernetes Pod Logs

- Kubernetes accesses container stdout as pod logs

```
$ kubectl run webserver --image=httpd:2
pod/webserver created

$ kubectl logs webserver
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 10.32.0.14. Set the
'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 10.32.0.14. Set the
'ServerName' directive globally to suppress this message
[Fri Jun 12 00:06:55.022741 2020] [mpm_event:notice] [pid 1:tid 140240995370112] AH00489: Apache/2.4.43 (Unix)
configured -- resuming normal operations
[Fri Jun 12 00:06:55.023033 2020] [core:notice] [pid 1:tid 140240995370112] AH00094: Command line: 'httpd -D
FOREGROUND'

$ kubectl get pod -o wide
NAME         READY   STATUS    RESTARTS   AGE   IP           NODE         NOMINATED NODE   READINESS GATES
webserver    1/1     Running   0          29s   10.32.0.14   k8s-master   <none>           <none>

$ curl -L http://10.32.0.14
<html><body><h1>It works!</h1></body></html>

$ kubectl logs webserver
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 10.32.0.14. Set the
'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 10.32.0.14. Set the
'ServerName' directive globally to suppress this message
[Fri Jun 12 00:06:55.022741 2020] [mpm_event:notice] [pid 1:tid 140240995370112] AH00489: Apache/2.4.43 (Unix)
configured -- resuming normal operations
[Fri Jun 12 00:06:55.023033 2020] [core:notice] [pid 1:tid 140240995370112] AH00094: Command line: 'httpd -D
FOREGROUND'
10.32.0.1 - - [12/Jun/2020:00:07:21 +0000] "GET / HTTP/1.1" 200 45
```

Run Apache Webserver pod

Retrieve pod logs from container stdout

Get pod IP

curl pod IP

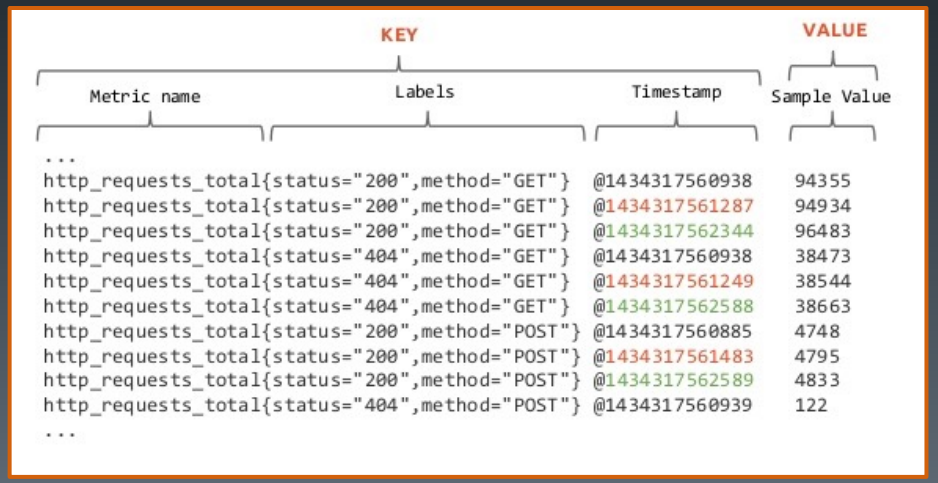
Retrieve pods logs after curl

New entry from curl

Metrics

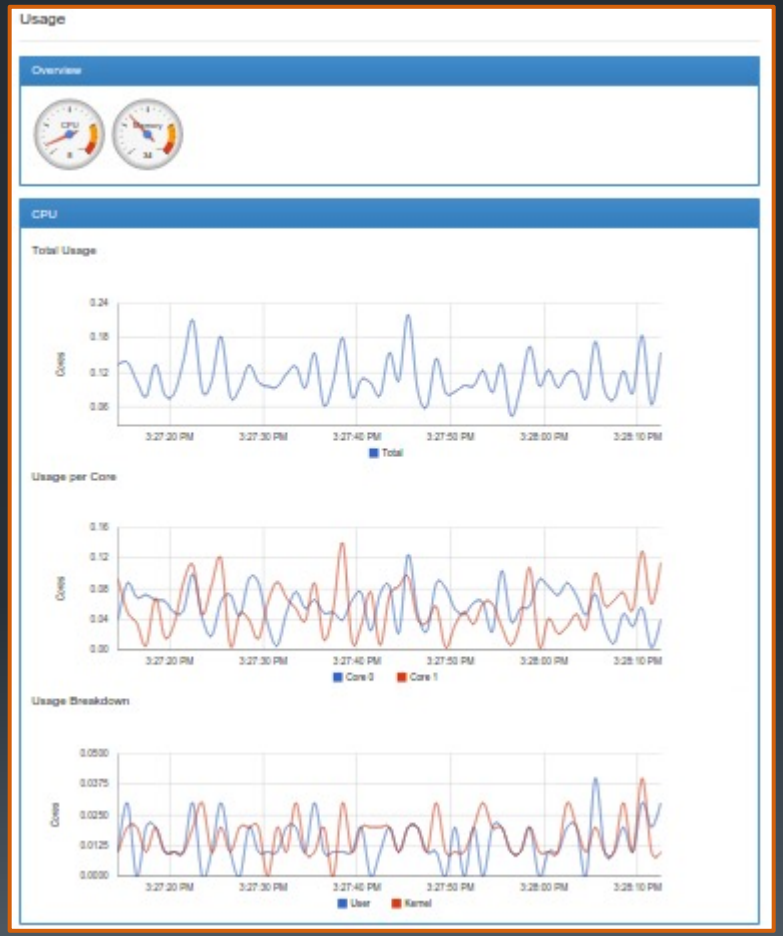
- Metrics are sets of numbers that give information about a particular process or activity
- Generally metrics come in the form of a timeseries
  - Meter – the thing being measured
  - Meter Type – percentage (% cpu), rate (packets per second), count (# of files), …
- Each event/sample in a timeseries has
  - Timestamp (2018-04-22,21:08:15.261)
  - Number (57.9%)
- Timeseries are usually sampled at regular intervals
  - Useful for predicting trends which may lead to outages, resource exhaustion, …
- Timeseries are easy to compress, store, plot, process, and retrieve
  - Extremely long periods can be maintained by removing intermediate metrics
    - e,.g. replacing a days 86,400 per second samples with 4: begin, high, low, end
- Prometheus
  - The leading metric monitoring solution for Kubernetes
  - Prometheus metrics add an arbitrary set of labels to the sample data enriching the information provided with each sample
    - Adds multidimensionality
      - Search by http REQ **or** http REQ w/ status 404 **or** http REQ w/ status 404 and method POST
  - Prometheus metrics format spun out to form Open Metrics Project

Resource ID	Name	Type	Volume	Unit	Timestamp
instance-00000002-bf8cf0d1-e4b5-46f1-8df2-83ad603bf9d2-tap76c94c76-43	network.outgoing.bytes	cumulative	2156.0	B	2014-05-12T19:59:15
instance-00000002-bf8cf0d1-e4b5-46f1-8df2-83ad603bf9d2-tap76c94c76-43	network.outgoing.bytes	cumulative	2433.0	B	2014-05-12T16:47:59
instance-00000002-bf8cf0d1-e4b5-46f1-8df2-83ad603bf9d2-tap76c94c76-43	network.outgoing.bytes	cumulative	2498.0	B	2014-05-12T17:41:19
instance-00000002-bf8cf0d1-e4b5-46f1-8df2-83ad603bf9d2-tap76c94c76-43	network.outgoing.bytes	cumulative	2498.0	B	2014-05-12T17:51:19
instance-00000002-bf8cf0d1-e4b5-46f1-8df2-83ad603bf9d2-tap76c94c76-43	network.outgoing.bytes	cumulative	2498.0	B	2014-05-12T18:01:19
instance-00000002-bf8cf0d1-e4b5-46f1-8df2-83ad603bf9d2-tap76c94c76-43	network.outgoing.bytes	cumulative	2924.0	B	2014-05-12T18:11:19
instance-00000002-bf8cf0d1-e4b5-46f1-8df2-83ad603bf9d2-tap76c94c76-43	network.outgoing.bytes	cumulative	3243.0	B	2014-05-12T16:57:59
instance-00000002-bf8cf0d1-e4b5-46f1-8df2-83ad603bf9d2-tap76c94c76-43	network.outgoing.bytes	cumulative	3257.0	B	2014-05-23T10:58:30
instance-00000002-bf8cf0d1-e4b5-46f1-8df2-83ad603bf9d2-tap76c94c76-43	network.outgoing.bytes	cumulative	3322.0	B	2014-05-14T11:06:31
instance-00000002-bf8cf0d1-e4b5-46f1-8df2-83ad603bf9d2-tap76c94c76-43	network.outgoing.bytes	cumulative	3350.0	B	2014-05-12T18:21:19
instance-00000002-bf8cf0d1-e4b5-46f1-8df2-83ad603bf9d2-tap76c94c76-43	network.outgoing.bytes	cumulative	3421.0	B	2014-05-12T16:03:12
instance-00000002-bf8cf0d1-e4b5-46f1-8df2-83ad603bf9d2-tap76c94c76-43	network.outgoing.bytes	cumulative	3669.0	B	2014-05-12T17:08:00



cAdvisor

- An open source container resource usage and performance analysis agent
- Purpose-built for containers and supports Docker containers natively
- In Kubernetes cAdvisor is integrated into the Kubelet binary
  - cAdvisor auto-discovers all containers in the machine and collects CPU, memory, filesystem, and network usage statistics
  - cAdvisor also provides the overall machine usage by analyzing the 'root' container on the machine
- Kubelet exposed the cAdvisor UI up to version v1.11
  - Starting from v1.13, you can deploy cAdvisor as a DaemonSet to reenable the cAdvisor UI



Kubernetes Metrics API

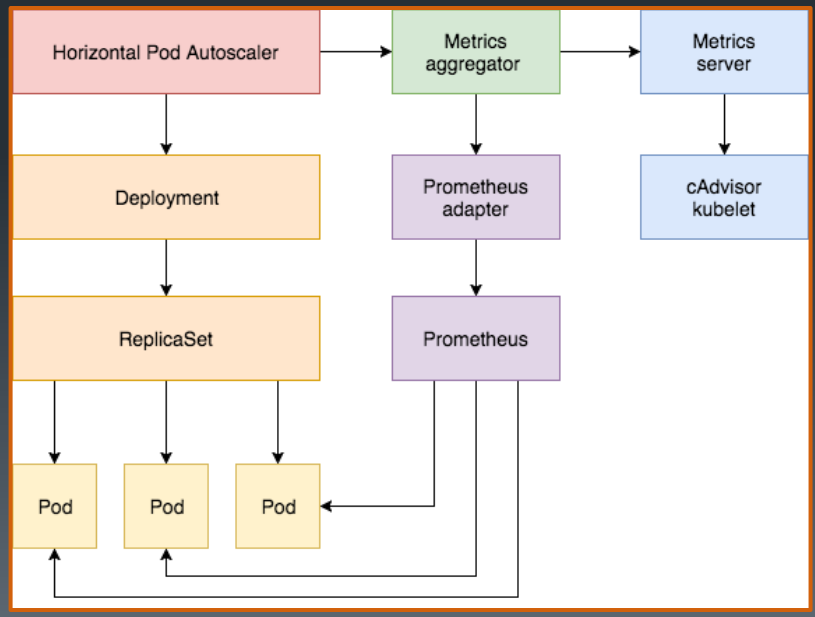

```
floreks [master|✚4…1]$ kubectl top pod
NAME                                CPU(cores)   MEMORY(bytes)
grafana-3867175031-pgcsn            0m           7Mi
nginx-2371676037-ngw6v              0m           2Mi
nginx-2371676037-b16bp              0m           2Mi
nginx-deployment-3646295028-z3rd7   0m           2Mi
frontend-51sbd                      0m           10Mi
frontend-rx5z7                      0m           10Mi
nginx-2371676037-q4gk9              0m           2Mi
nginx-deployment-3646295028-qcv76   0m           2Mi
```

- /apis/metrics.k8s.io/
- Exposes resource usage metrics
  - Container/Pod/Node CPU usage
  - Container/Pod/Node memory usage
  - …
- The Metrics API offers only the current sample (not a storage interface)
- The Metrics API requires the Kubernetes Metrics Server to be deployed in the cluster

- Kubernetes Metrics Server
  - Metrics Server is a cluster-wide aggregator of resource usage data
    - Tracks Container/Pod/Node CPU and Memory metrics
  - Metric server collects metrics from the Summary API exposed by the Kubelet on each node
  - The Metrics Server stores data in memory and only holds the most recent sample
    - Crash/restart eliminates all data
    - Early versions of the Metrics Server support only 30 pods per cluster node (running more may cause an OOM failure)
- Metrics API clients
  - kubectl top
  - Horizontal Pod Autoscaler

Custom Metrics API

- Defined to allow any system to implement custom Kubernetes metrics
  - Intentionally just an API definition and not an implementation
  - The metrics aggregator allows multiple systems to supply metrics through the K8s metrics API
- Implementations are installed into a Kubernetes cluster as Aggregated APIs
  - Aggregated APIs were introduced in K8s 1.7
  - Aggregated APIs allows services running anywhere in the cluster to extend the Master API
  - Metrics Servers are registered in the main API through the Kubernetes API Aggregation interface
- Implementations can be switched-out/added but the API stays the same
- The custom metrics API allows requesting arbitrary metrics
  - Custom metrics API implementations are specific to the respective backing monitoring system
- Implementations:
  - Prometheus Adapter – an implementation of the custom metrics API that attempts to support arbitrary metrics following a set label and naming scheme
  - Microsoft Azure Adapter – an implementation of the custom metrics API that allows you to retrieve arbitrary metrics from Azure Monitor
  - Datadog Cluster Agent – implementation of the external metrics provider, using Datadog as a backend for the metrics
  - Kube Metrics Adapter – general purpose metrics adapter for Kubernetes that can collect and serve custom and external metrics for Horizontal Pod Autoscaling
    - Provides the ability to scrape pods directly or from Prometheus through user defined queries
    - Also capable of serving external metrics from a number of sources including AWS' SQS and ZMON monitoring



Metrics Scraping

- Applications with metric endpoints can be scraped

```
$ kubectl run trash-levels --image=rxmllc/trash-levels
pod/trash-levels created

$ kubectl logs trash-levels
2020/06/12 00:20:23 Listening on: 8080

$ kubectl get pod -o wide
NAME           READY   STATUS    RESTARTS   AGE   IP           NODE         NOMINATED NODE   READINESS GATES
trash-levels   1/1     Running   0          45s   10.32.0.14   k8s-master   <none>           <none>

$ curl -L http://10.32.0.14:8080/metrics | grep process_cpu_seconds_total
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  6247    0  6247    0     0  4732k      0 --:--:-- --:--:-- --:--:-- 6100k
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.04

$ curl -L http://10.32.0.14:8080/metrics | grep go_memstats_mspan_inuse_bytes
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  6249    0  6249    0     0  4306k      0 --:--:-- --:--:-- --:--:-- 6102k
# HELP go_memstats_mspan_inuse_bytes Number of bytes in use by mspan structures.
# TYPE go_memstats_mspan_inuse_bytes gauge
go_memstats_mspan_inuse_bytes 41344
```

Run app with a metrics endpoint

Retrieve pod logs; listening on 8080

Get pod IP

curl metrics endpoint and grep for a cpu metric

curl metrics endpoint and grep for allocation statistics of in-use pages managed by the mheap

Horizontal Pod Autoscaler

- Allows the number of pods in a replication controller or deployment to scale automatically based on metrics
- Metrics include:
  - Observed CPU utilization – scales the number of pods in a Deployment, Replica Set or Replication Controller based on observed CPU utilization
    - Autoscaler periodically queries CPU utilization for the pods it targets
    - Threshold expressed as: `targetCPUUtilizationPercentage`
    - Requires metrics aggregation in your cluster for autoscaling to work
  - Custom metrics
    - You can add custom metrics for the Horizontal Pod Autoscaler to use in the `autoscaling/v2beta1` API
      - `targetCPUUtilizationPercentage` field replaced with an array called `metrics`
    - For example, queries per second or average request latency

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: example-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: mydep
  maxReplicas: 6
  minReplicas: 3
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 80
  - type: Pods
    pods:
      metricName: requests-per-second
      targetAverageValue: 20
```

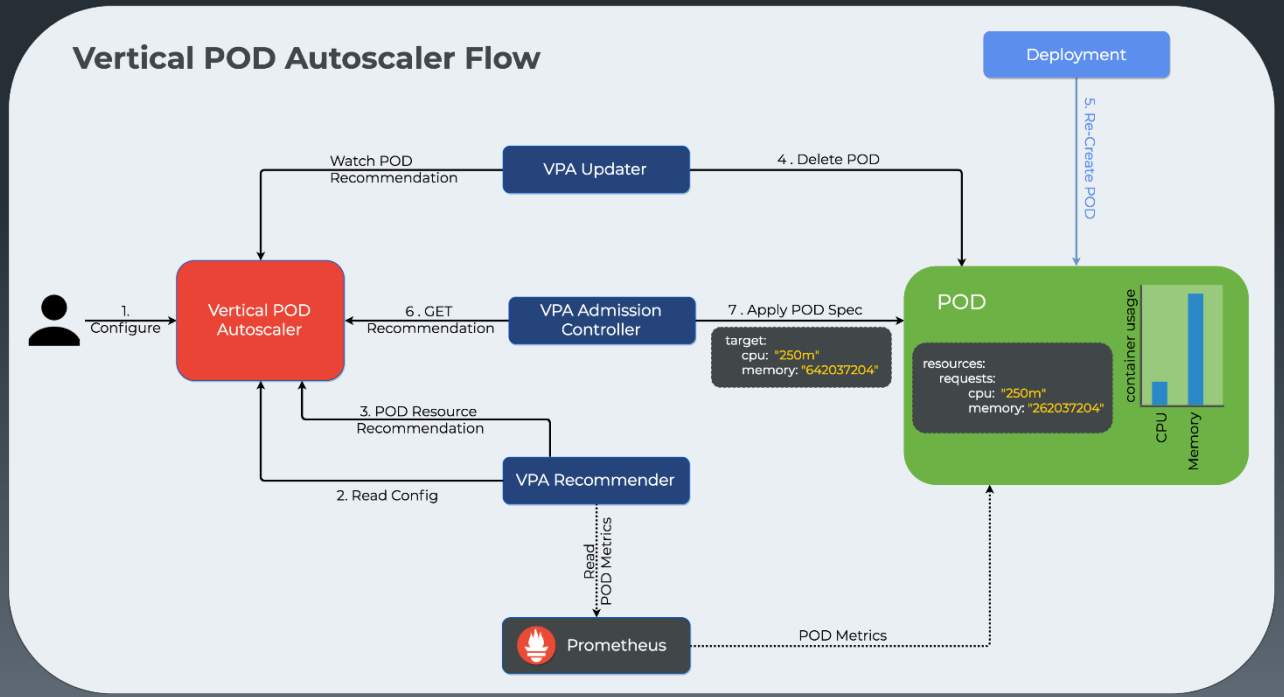
Imperative command equivalent example:

```
kubectl autoscale deploy mydep \
--min=3 --max=6 --cpu-percent=80
```

Vertical Pod Autoscaler

- Updates resource limits and requests for pod containers based on usage over time
- Up-scale pods that are under-requesting resources
- Down-scale pods that are over-requesting resources
- Independent install from the kubernetes/autoscaler GitHub repo

```yaml
apiVersion: autoscaling.k8s.io/v1beta2
kind: VerticalPodAutoscaler
metadata:
  name: my-sts-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: StatefulSet
    name: my-sts
  updatePolicy:
    updateMode: "Auto"
```



Summary

- Observability is critical in distributed systems like Kubernetes
- The three pillars of observability:
  - Logging
    - Capturing logs begins with the container runtime and requires planning to ensure logs are saved beyond the life of a given container
  - Metrics
    - cAdvisor, built into kubelet is a composable container aware monitoring tool
    - The Kubernetes metrics-server presents current resource metrics from kubelet cAdvisors to the Kubernetes metrics API
    - Custom metrics adapters enable tools like Prometheus to present custom metrics to the Kubernetes metrics API
    - Kubernetes Pod Autoscalers use the metrics API to automatically scale controllers like Deployments and StatefulSets according to resource use
  - Tracing
    - Tracing can be implemented in many ways, as custom application logging code, pluggable Libraries that allow developers to code tracing abstractly or with telemetry reported by proxies

Lab 8

- Kubernetes Metrics and HPA

The End

Many thanks for attending!

rx-m cloud native training & consulting