rx-m cloud native training & consulting

Django meets k8s

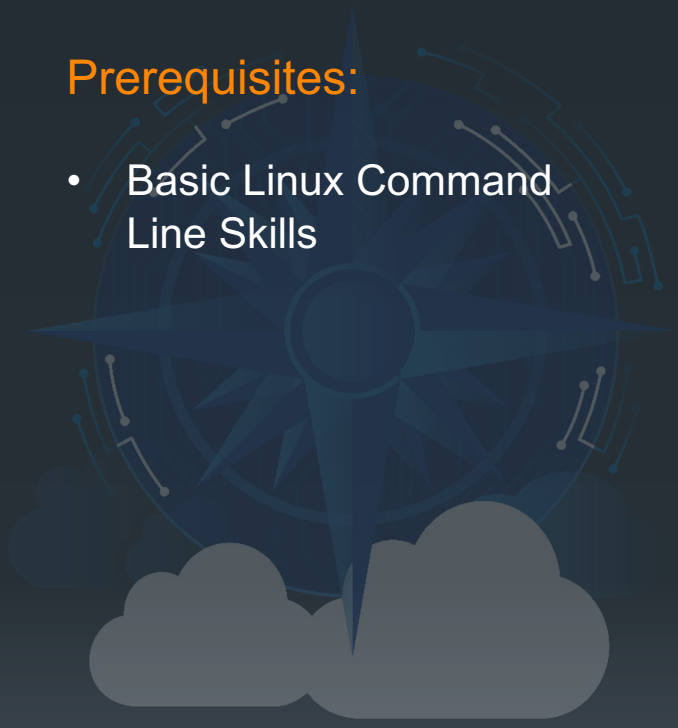
Overview

1. Overview of containers
2. Overview of container images
3. Overview of Kubernetes

4. Demo
   - Create Django app
   - Conteinerizing Django app
   - Docker compose
   - Create multi node Kubernetes cluster
   - Run Django inside Kubernetes
   - Disaster simulation

Prerequisites:

- Basic Linux Command Line Skills



1: Overview of Containers

What is a Container?

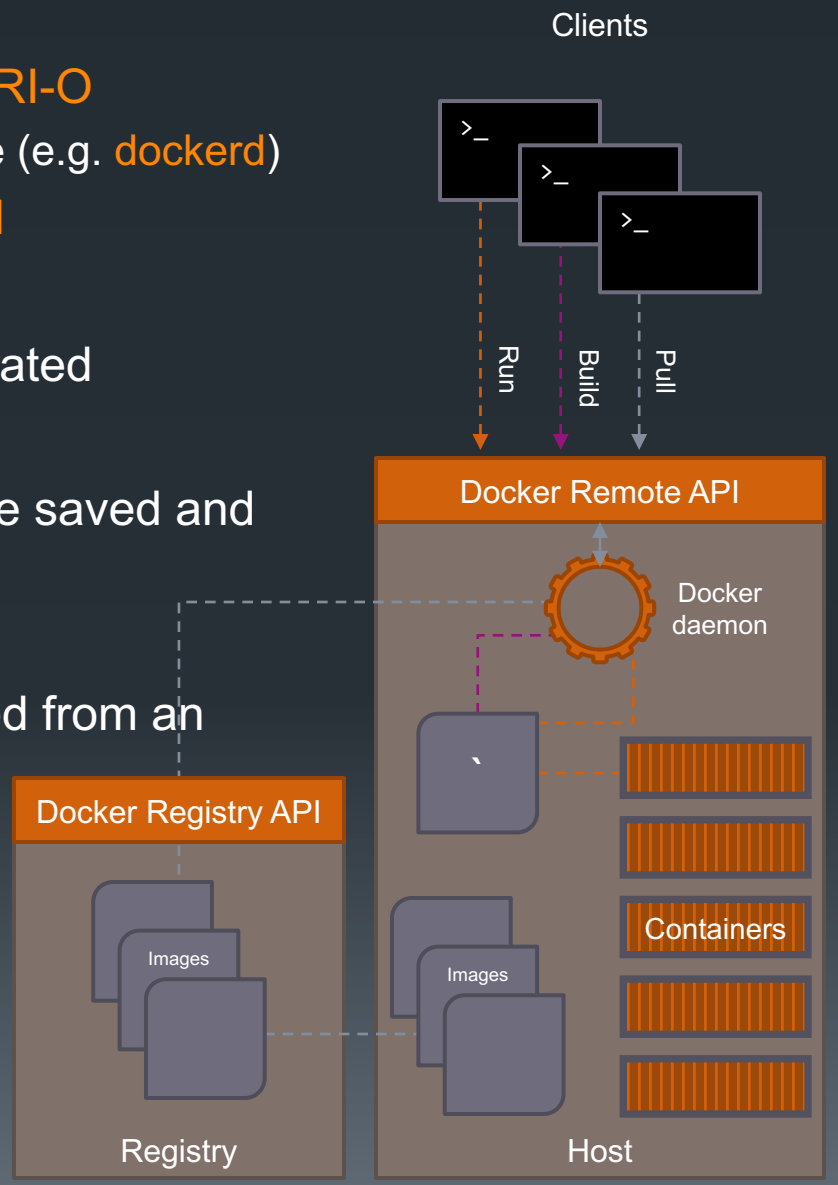
- Lightweight Operating System environment
  - Originally x86 Linux only, now with support for Windows, ARM and other combinations
- Encapsulated, deployable, runnable
  - The new way to package applications
- Microservice-centric
  - one atomic service per container
- Made widely popular by Docker
  - 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



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

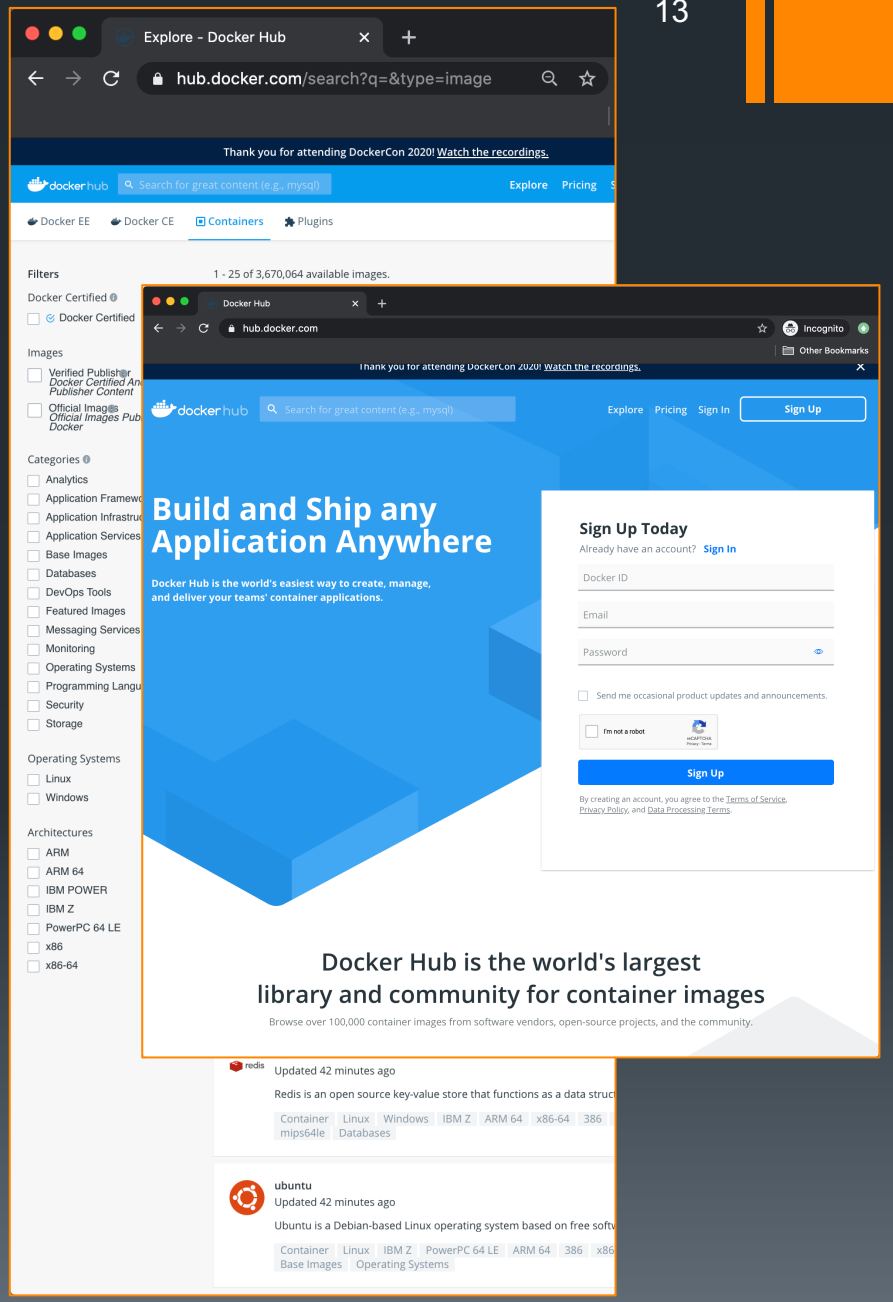
Container Standardization

- 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 the container format, runtime code (libcontainer) and specifications
  - runc was created as a libcontainer client, representing 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

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

- Containers support 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

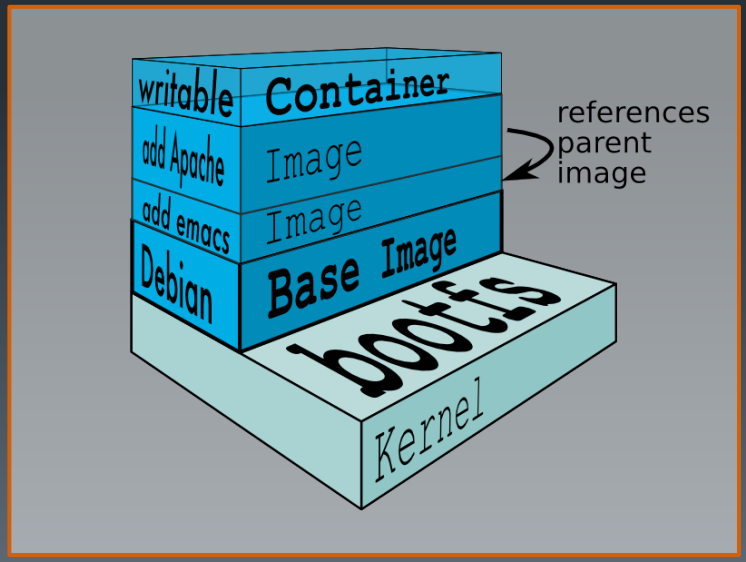
```
user@ubuntu:~$ docker container start --help

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)
```

2: Overview of Container Images

Images

- A container image includes metadata and an optional file system layer
  - Image metadata includes the ID of the image, the default command to run, etc.
  - Each change to the image's filesystem exist within a given layer of an image
  - Each image layer should supply (ideally) one specific feature on top of the base image
  - Upper layer metadata and files mask metadata and files at lower layers
- Data in each layer of an image contributes to the final image's overall size
- An image with no parent image is called a Base Image
  - Base images usually include typical operating system libraries and executables
    - e.g. shell programs, package managers, filesystem tools, etc.
- Image metadata and files are immutable
  - Allows one Image to support multiple container instances with repeatable results
  - Reduces the disk and memory impact of containers created from the same images
  - Images must be rebuilt with the same image tag to incorporate changes
    - Ideally new changes = new container tags
- 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
  - In this way, container file systems contain only the delta between their file system state and that of their underlying images
  - The view from the top down, including all file system layers in the stack, is called the union file system



Creating Images

- Container images are constructed by sequentially applying additional metadata and file system layers on top of existing layers
  - Execute a command, commit the change to a new layer of the image
- There are two ways to create new images
  - Update a container created from an image and commit the results to a new image
  - Use a Dockerfile to specify instructions to create an image and invoking an image build tool
- Docker caches created or downloaded images on the Docker host
  - Created images can subsequently be pushed to a registry

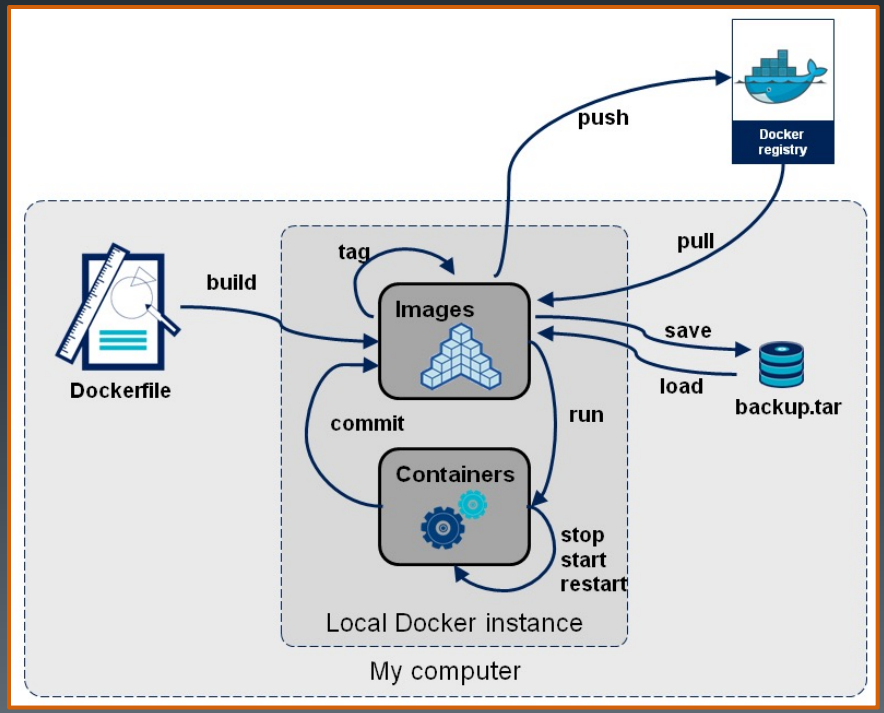


Image Build Tools

- 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/

```yaml
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
```

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

- 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

- `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

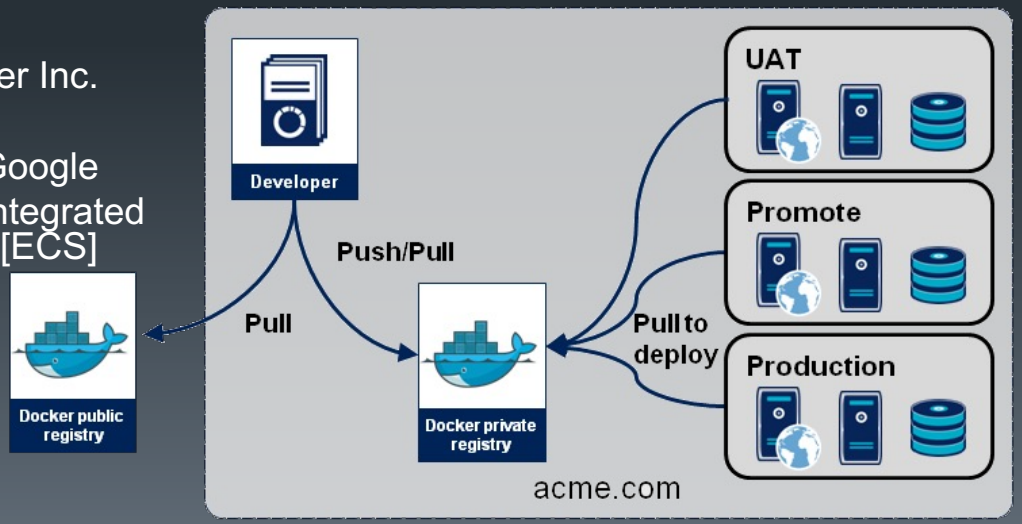
```
LABEL version="1.0"
USER webapp
WORKDIR /opt/webapp/db
RUN bundle install
WORKDIR /home/webapp/
ENTRYPOINT [ "/path/to/bin" ]
```

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



3: Overview of Kubernetes

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

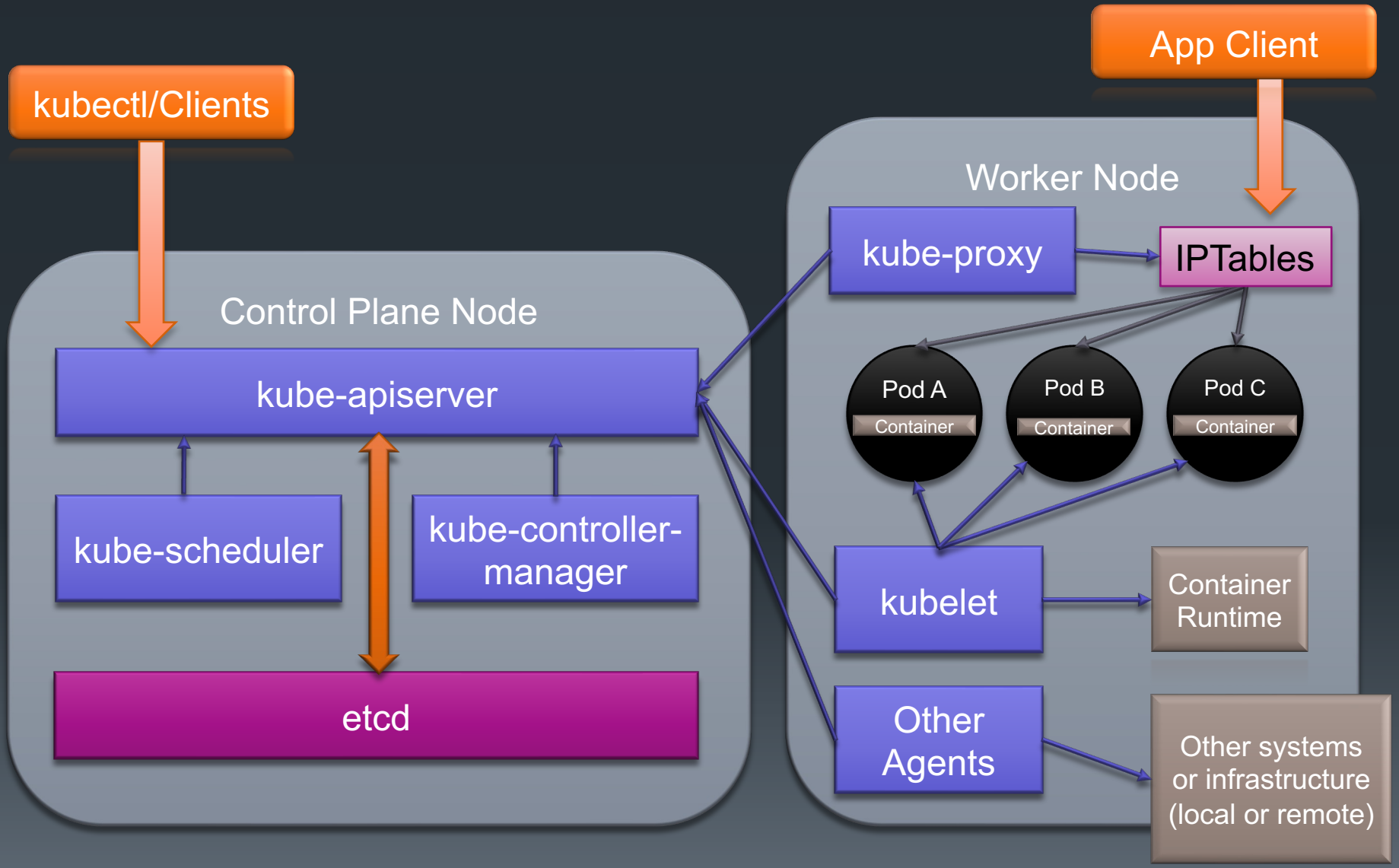
Kubernetes

- 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=

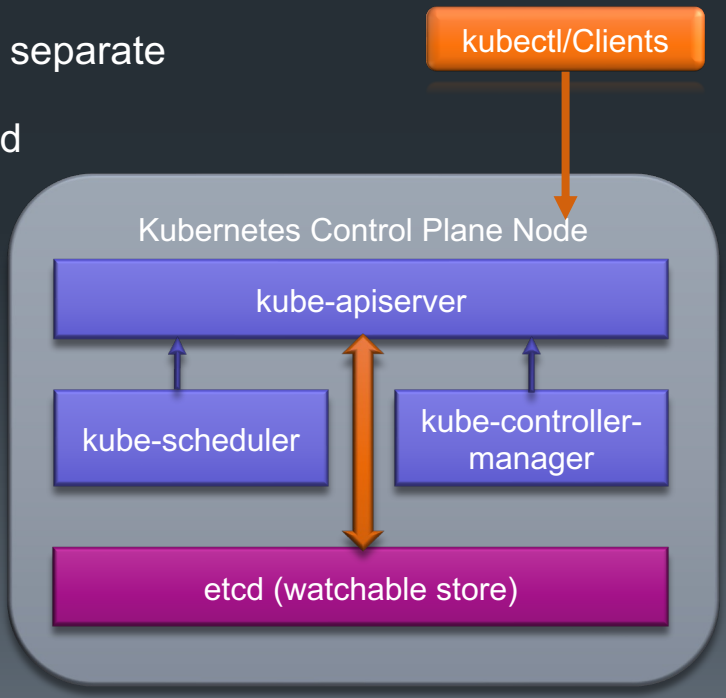
The Kubernetes project was started by Google in 2014
Kubernetes builds upon a decade and a half of experience that Google has with running production workloads at scale (Borg), combined with best-of-breed ideas and practices from the community

Kubernetes Architecture



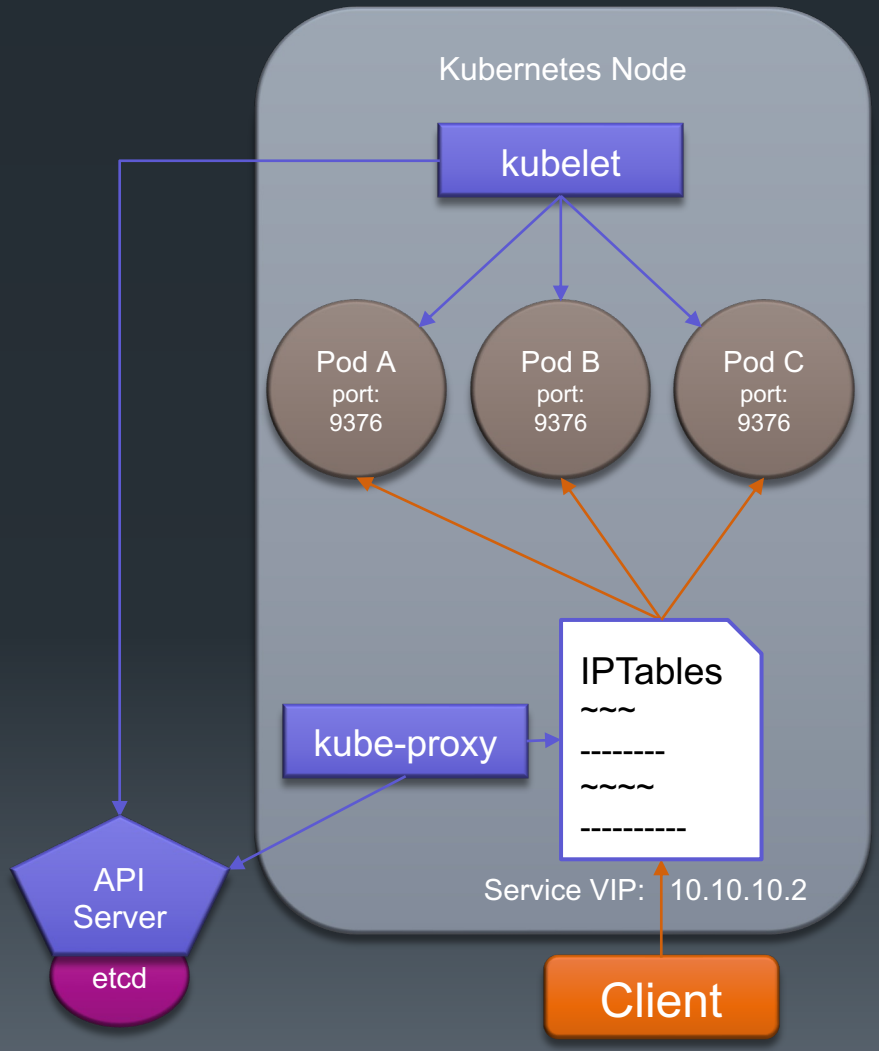
Control Plane

- 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



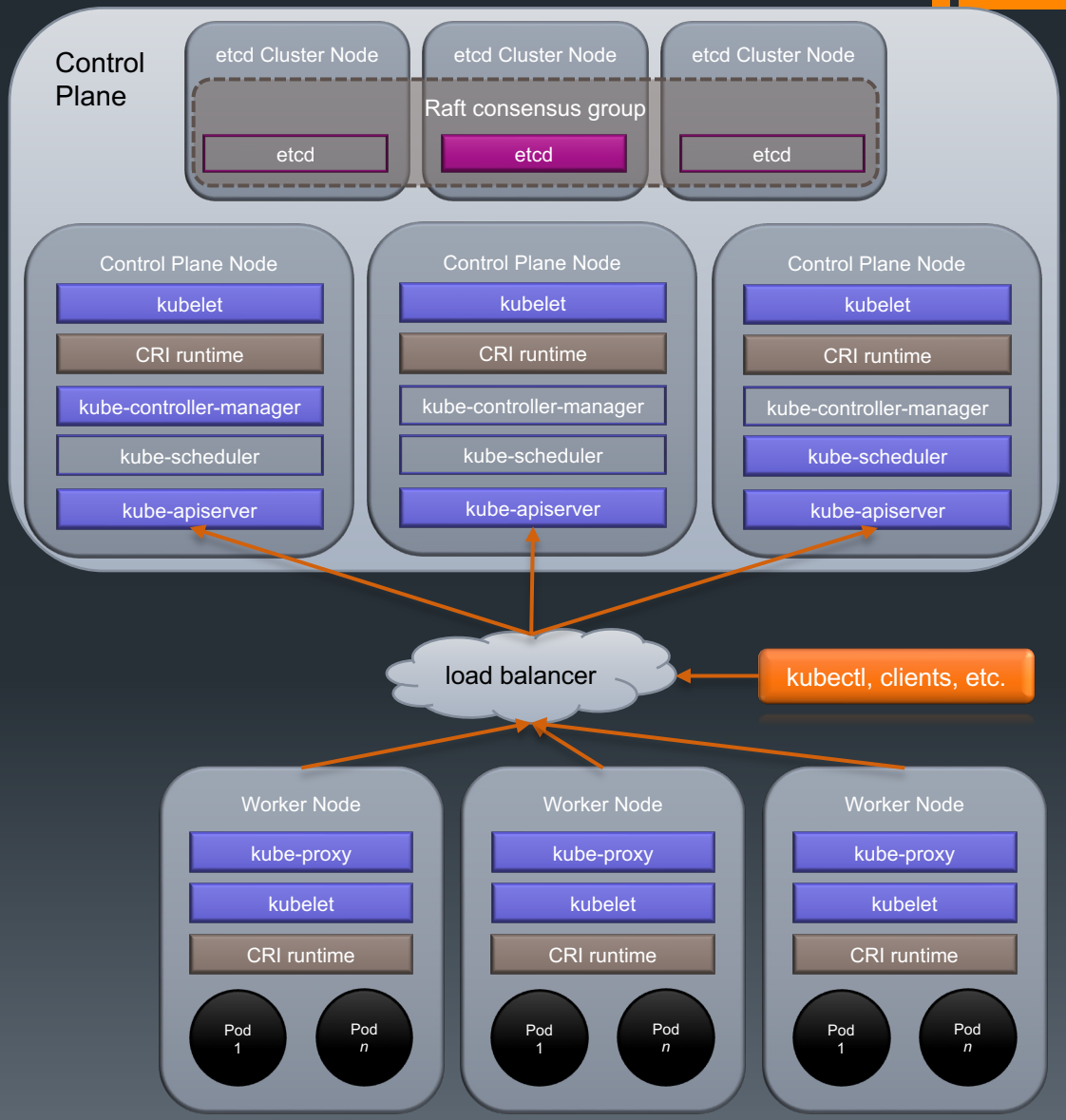
Node Agents

- 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



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

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

Demo