Kubernetes from Basic to Advanced

kubernetes

Session #01
Container Orchestration

kubernetes

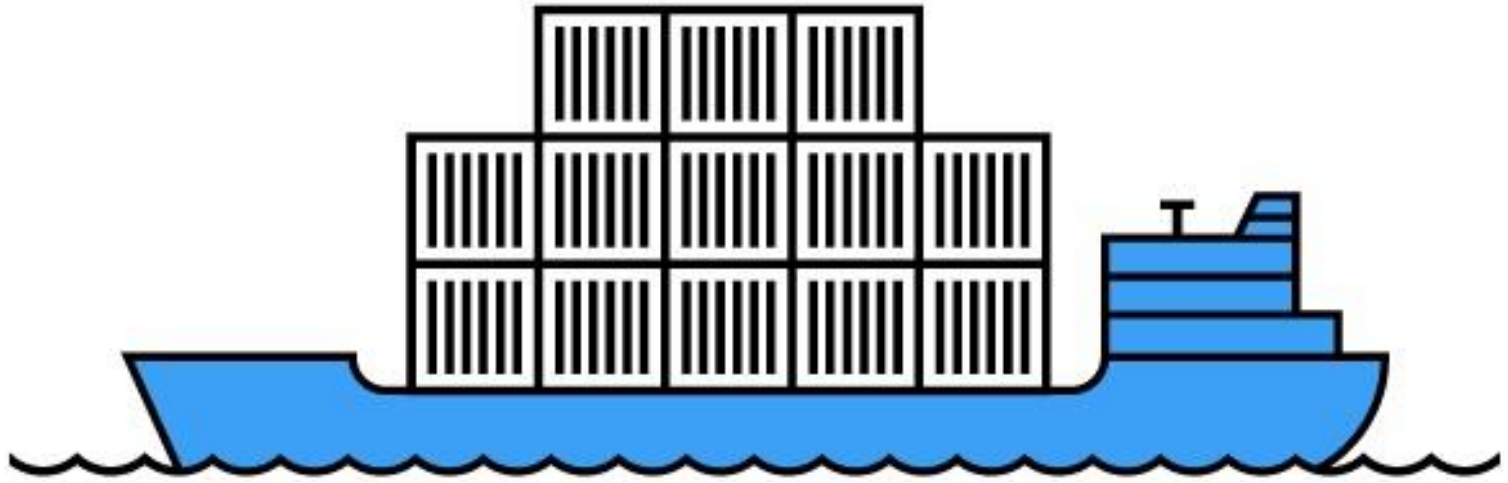
Session Contents

- What is a container?
- Containers and Docker
- Containers Concepts
- Container Orchestration
- Kubernetes Architecture

What is a container?

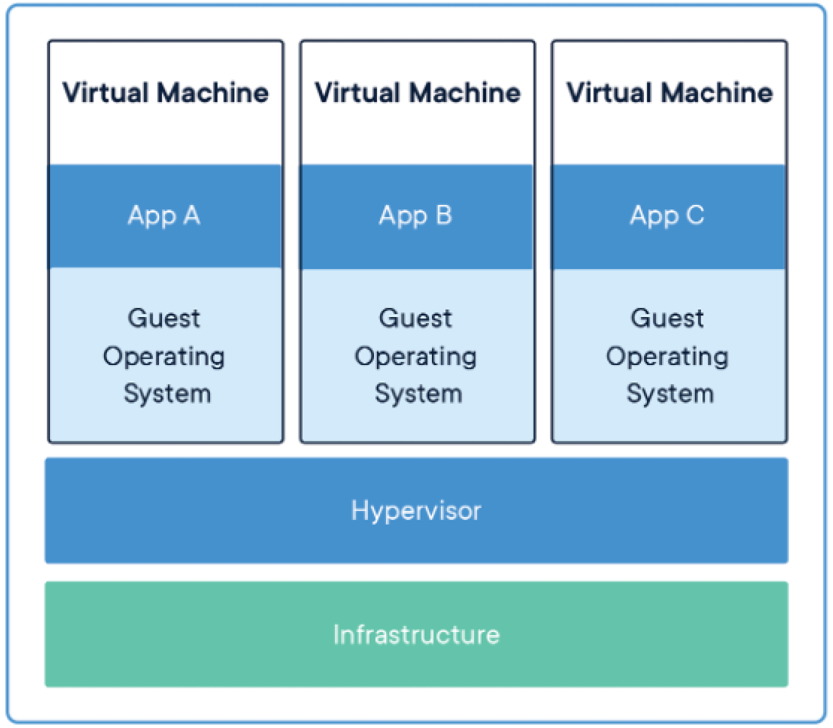
What is a container?

- A method of operation system virtualization
- A way to wrap an application into its own isolated box
- Includes only the binaries needed to support the application
- Isolates an app with its own view of the host from the perspectives of memory, CPU and network

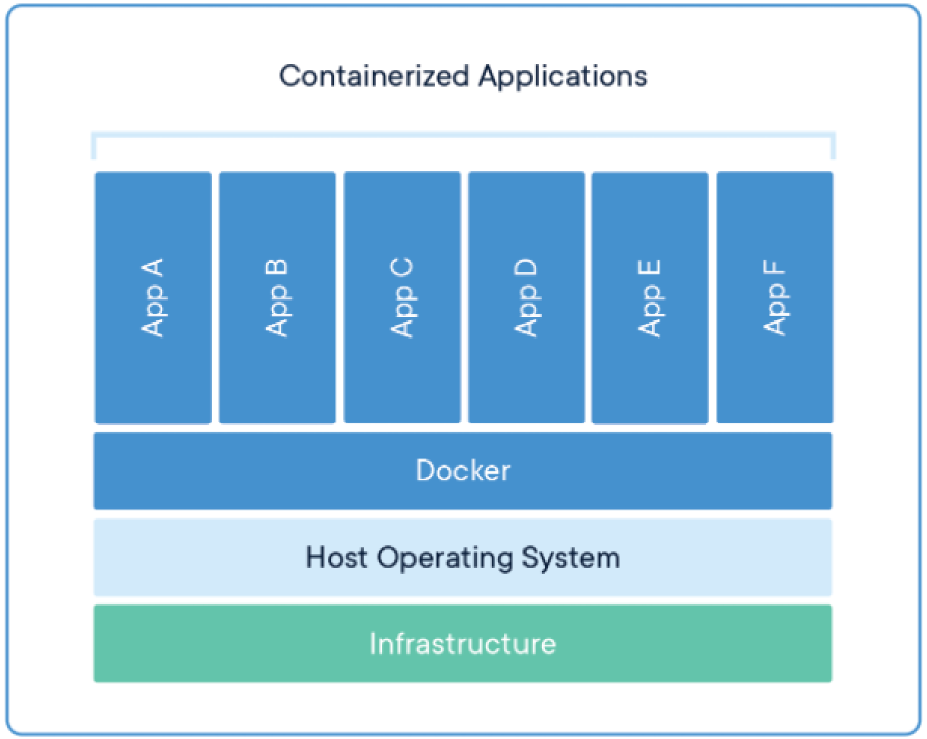


kubernetes

VM vs Containers



Virtual Machines
VMs as units of scaling



Containers
Apps as units of scaling

kubernetes

Benefits of using containers

Any OS	Anywhere	Any app	Any language
Linux	On-premises	Monolith	Java
Windows	Cloud	Microservice	.Net
			Python
			Node

kubernetes

Benefits of using containers

- Agility: Ship apps faster
- Portability: Easily move workloads
- Density: Achieve resource efficiency
- Rapid scale: Scale easily to meet demand

kubernetes

Containers and Docker

Containers and Docker

- Containers are a technology like hypervisor for VMs
- Docker is an open platform like VMWare, VirtualBox, etc.
- Great achievement from Docker was that they make use containers easy
- Now containers and Docker seems to be one and the same thing
- “I’m creating my docker”. “I’m running my docker”. “Let’s dockerize this app”
- Uses a CLI (Command-Line Interface) as standard interface to interact with Docker Engine

kubernetes

Mandatory to run containers?

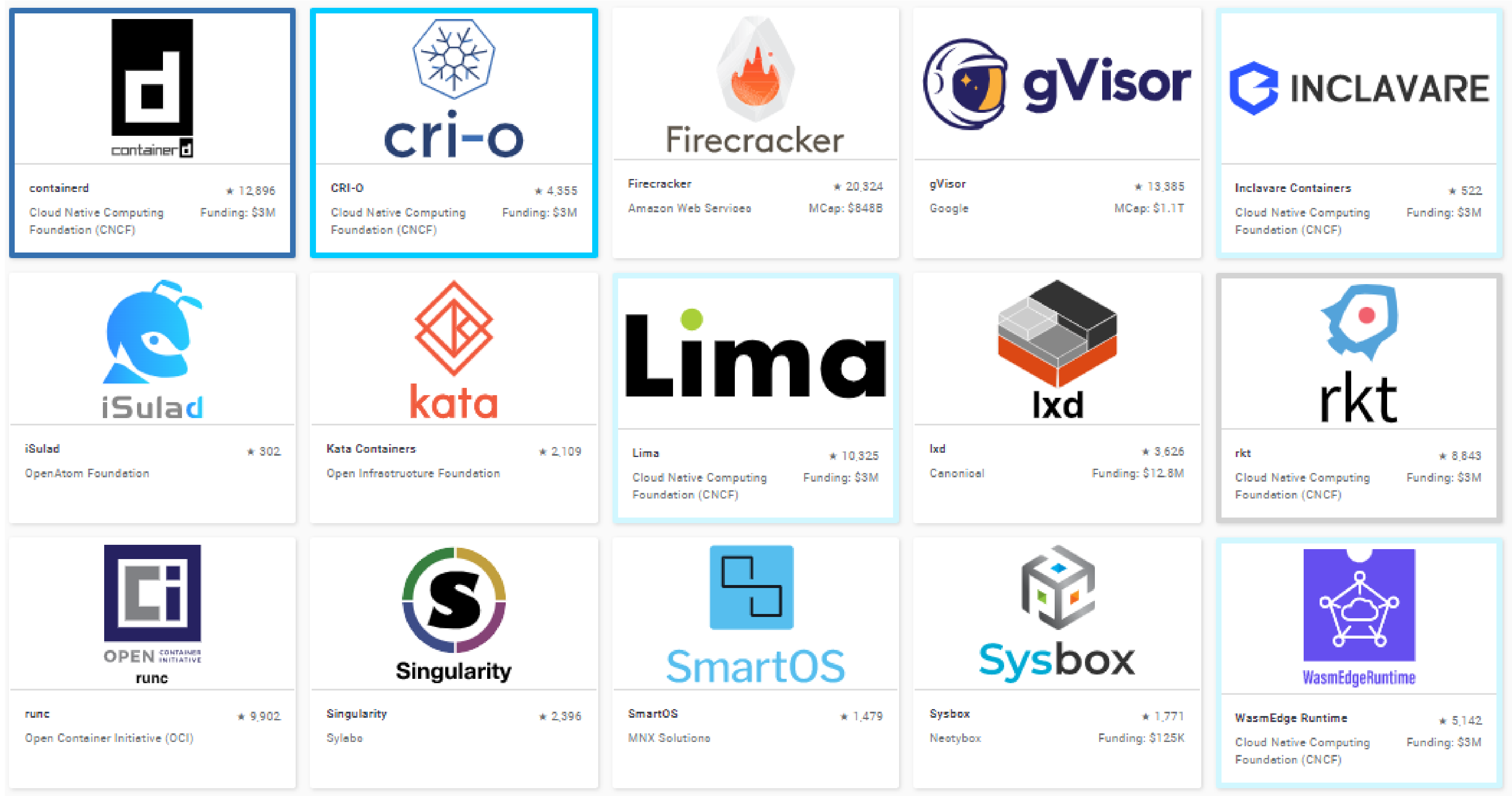
- You don’t need Docker to run your containers
- You only need a Container Runtime
- Docker uses containerd as container runtime
- Docker have several other features like build images, publish images, …

Open Container Initiative (OCI)

- Open governance structure for the express purpose of creating open industry standards around container formats and runtimes
- Created in 2015 by Docker and other leaders on container industry
- Maintains 3 specifications
  - Runtime Specification (runtime-spec)
  - Image Specification (image-spec)
  - Distribution Specification (distribution-spec)
- Several other companies follows this specifications and allow you to create resources that can be use on any of them

kubernetes

Container Runtimes



containerd ★ 12,896
Cloud Native Computing Foundation (CNCF) Funding: $3M

CRI-O ★ 4,355
Cloud Native Computing Foundation (CNCF) Funding: $3M

Firecracker ★ 20,324
Amazon Web Services MCap: $848B

gVisor ★ 13,385
Google MCap: $1.1T

Inclavare Containers ★ 522
Cloud Native Computing Foundation (CNCF) Funding: $3M

iSulad ★ 302
OpenAtom Foundation

Kata Containers ★ 2,109
Open Infrastructure Foundation

Lima ★ 10,325
Cloud Native Computing Foundation (CNCF) Funding: $3M

lxd ★ 3,626
Canonical Funding: $12.8M

rkt ★ 8,843
Cloud Native Computing Foundation (CNCF) Funding: $3M

runc ★ 9,902
Open Container Initiative (OCI)

Singularity ★ 2,396
Sylabs

SmartOS ★ 1,479
MNX Solutions

Sysbox ★ 1,771
Nestybox Funding: $125K

WasmEdge Runtime ★ 5,142
Cloud Native Computing Foundation (CNCF) Funding: $3M

kubernetes

Docker alternatives

buildah

LXD, LXC, LXCFS and distrobuilder

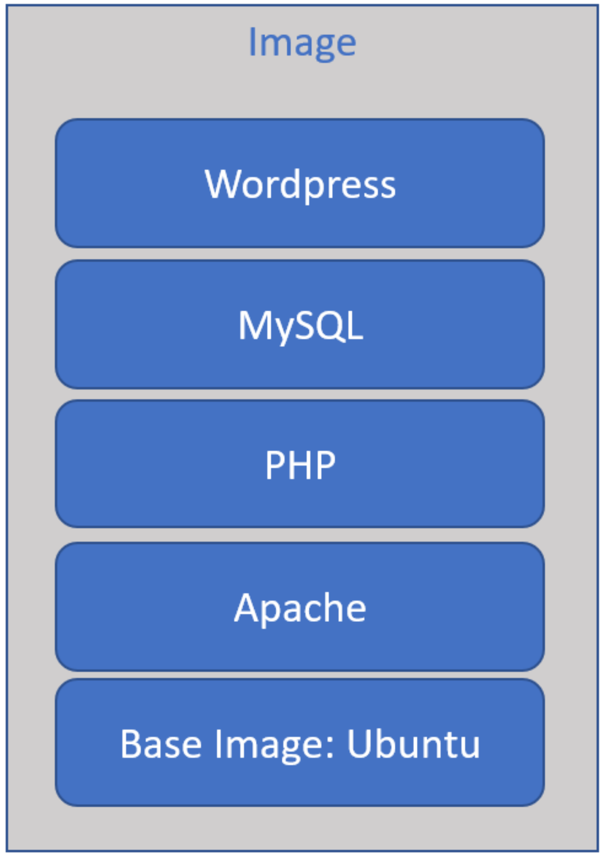
podman

kubernetes

Containers Concepts

Main Concepts: Image

- Read-only templates for containers
- Container is a running instance of the image
  - Like classes and objects on OOP
- Can be reused by several containers
- Built up from a series of layers
- Created recursively using other images
- Initial layer is called base image
- If any change needs to be performed on layers, a new image is created



kubernetes

Main Concepts: Base Image

- Base image needs to be carefully selected
- Images are created using another image
- `scratch` image is the mother of all images ☺
- 3 main approaches
  - Using vendor/known-publisher images, like Mongo, dotnet, python, etc.
  - Using SO only images and install all additional software needed
  - Using slim images (scratch or distroless) and install only needed software
- Distroless images are getting some hype
  - Smaller images means smaller storage on registry and small network usage
  - Smaller images means less attack surface to be used

kubernetes

Main Concepts: Layers

- Layer means filesystem objects (folder and files)
- Images uses read-only layers to be reused
- All layers needs to exists locally on the server where the container will run
- Can be shared by several containers and images
    - Don’t need to be duplicated if used by several images
- Every container creates its own writable layer to make changes on filesystem
- Image layers are never modified

kubernetes

Main Concepts: Registry

- Registry is a stateless, highly scalable server side application that stores and lets you distribute images
- Same concept of a package management solution (maven, nuget, etc.)
- You can push images to the registry and pull images to the registry
- Can be public or private for each or both actions
- Main public registry is Docker Hub (starting to have some caveats…)

kubernetes

Main Concepts: Tags

Image name is composed by

registry-server/repository:tag

registry-server

- Define where the image were pulled: gcr.io, azurecr.io, …
- When omitted means "docker.io"

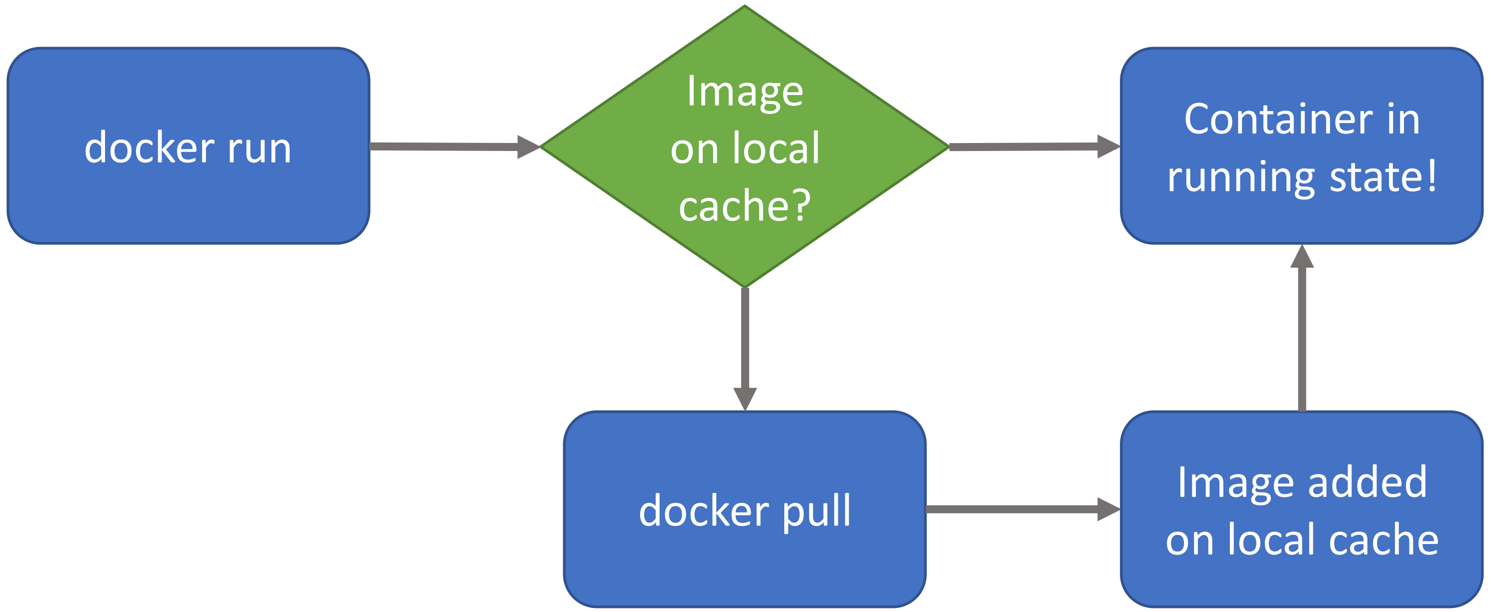
repository

- Define the name of the image
- Can be composed by several levels using '/' as divider

tag

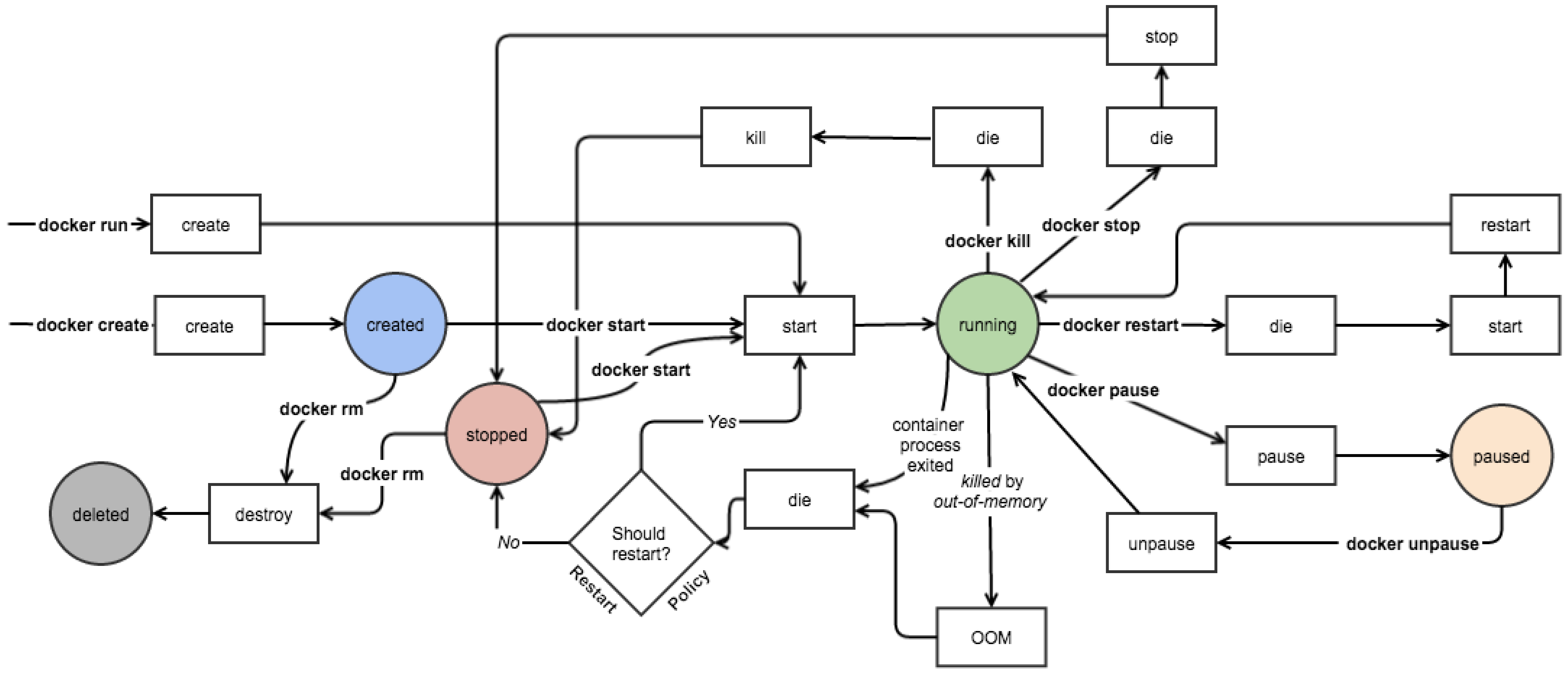
- Define the version of the image
- When omitted gets the value "latest". So "latest" really means "no tag".
- Is a textual and open value
- Can be overwritten on the registry

Main Concepts: Lifecycle



kubernetes

Main Concepts: Lifecycle



kubernetes

Main Concepts: Dockerfile

- Text file with commands in it to create a new image
- Can be seen as a configuration file with set of instructions needed to assemble a new image.
- A build command parses Dockerfile to build a new container image
- Each command creates a new layer
- Uses caching to only recreate layers if something changed

Main Concepts: Multi-stage Dockerfile

- Containers bring total autonomy on execution
- To create images we need to have access to all needed binaries
- Define all phases of build process inside a container
- Benefits
- Autonomy
  - All needed libraries/frameworks available independently on the host
  - Allow partial execution
  - Multiple tooling for application creation
  - Full images to build phase, slim images to runtime images

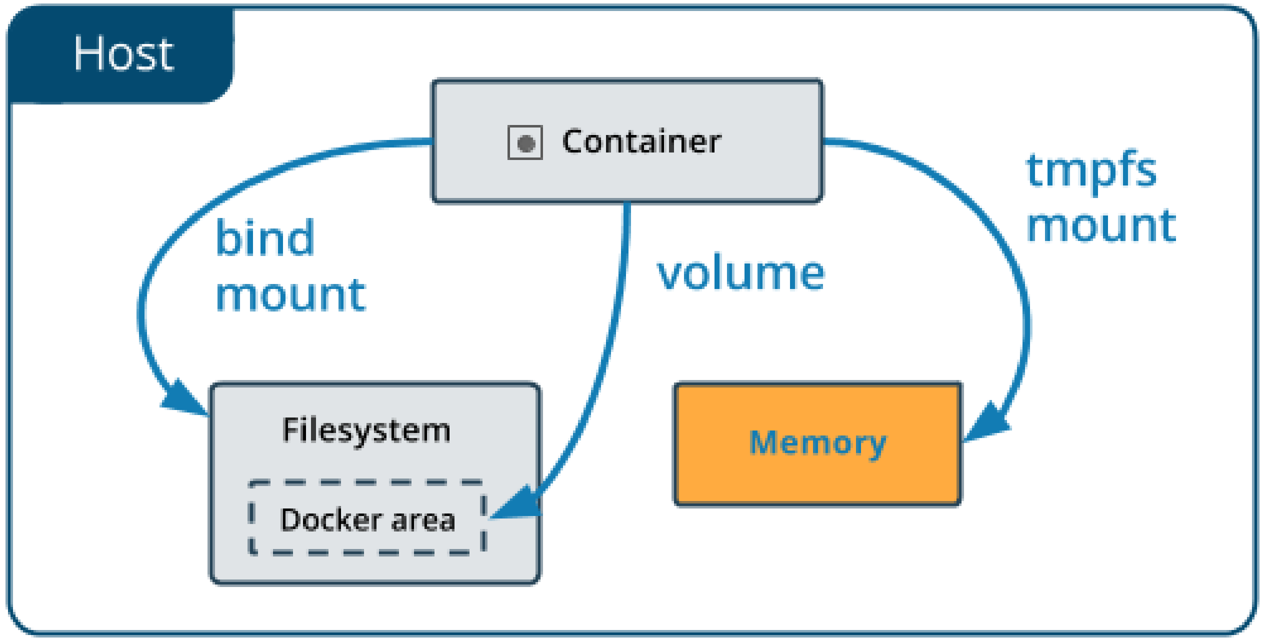
Main Concepts: Dockerfile Best Practices

- Minimize the number of steps in the Dockerfile
- Start your Dockerfile with the steps that are least likely to change
- Clean up your Dockerfile
- Use a .dockerignore file
- Use caching to your benefit
  - Start with commands that handles less changes: install tooling, package management, etc.
  - Group several actions on same command (update distro, install tools, etc.)

kubernetes

Main Concepts: Storage

- If you need persistent data, you need to rely on external storage since containers have an ephemeral writable layer



Main Concepts: Network

- Docker containers can connect them together or connect them to non-Docker workloads
- Docker containers do not even need to be aware that they are deployed on Docker, or whether their peers are also Docker workloads or not
- Docker's networking subsystem is pluggable, using drivers
- Most common drivers:
  - bridge: The default network driver. Bridge networks are usually used when your applications run in standalone containers that need to communicate. Creates a NAT network.
  - host: For standalone containers, remove network isolation between the container and the Docker host, and use the host's networking directly

kubernetes

Main Concepts: Environment Variables

- Container runs on an isolated context
- Environment variables exists on container's scope
- Preferrable way to "send" values to the container allowing different executions
- These variables can be set on docker run command

Main Concepts: Resource Limits

- Host resources are shared between the host and all containers
- By default, any container can consume all host resources
- Not having explicit control may cause resources exhaustion causing impact on containers and the host
- When running a container, you may enforce limits on how much CPU and memory can be used by the container
- Reaching the limit may not break the container, only don’t allow it to get more resources

Main Concepts: Docker Compose

- Docker Compose is a tool for defining and running complex applications with Docker.
- Define a multi-container application in a single file
- Multiple isolated environments on a single host
- Preserve volume data when containers are created
- Only recreate containers that have changed
- Variables and moving a composition between environments
- Multiple compose files
- Allow to handle containers lifecycle
- First try to have container “orchestration”

kubernetes

Container Orchestration

Motivation

- Containers brings benefits on development process, tests and deployment
- Portability between environments
- Higher productivity (less configuration needs)
- Less footprint with bigger density on hardware
- Resources isolation
- But brings several challenges to manage and operate
- All management, maintenance and operation can be done manually but could be a crazy task!

kubernetes

Motivation

- To have more agility, automation and ease on these tasks, orchestration is the key
- Main orchestration features
    - Scheduling
    - Affinity
    - Monitoring
    - Failover
    - Scalability
    - Networking
    - Service Discovery
    - Application upgrades

kubernetes

Orchestration

- Scheduling
  - Container provisioning using nodes metrics and containers requests
- Affinity
  - Specific configuration for provisioning about availability/performance
- Monitoring
  - Detect and fix failures on a reactive/preventive way
- Failover
  - Re-provision faulty instances
  - Re-provision instances to healthy machines

kubernetes

Orchestration

- Scalability
  - Add/remove instances to meet demand
- Networking
  - Networking overlay for container communication
  - Allow inbound/outbound communication with the cluster
- Service Discovery
  - Enable containers to locate each other
- Application Upgrades
  - Avoid downtime and automatically rollback

kubernetes

Orchestration vs. Clustering

- Clustering means to use several resources (nodes) as one
- Orchestration run on top of a cluster (or single node) to implement additional features

kubernetes

Solutions

- Several solutions tried to implement container orchestration
- Docker Swarm was one of the first to use several nodes running Docker Engine
- Apache Mesos bring a different approach with several orchestration types
- Hashicorp Nomad focused on simplicity and performance
- But, Kubernetes take the lead and is considered de-facto container orchestration platform

kubernetes

Kubernetes

- Kubernetes open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation
- The name Kubernetes originates from Greek, meaning helmsman or pilot
- K8s as an abbreviation results from counting the eight letters between the "K" and the "s"
- Created on Google on behalf of project Borg
- Google open-sourced the Kubernetes project in 2014. Donated to Cloud Native Computing Foundation (CNCF) in 2015
- One of the biggest open source project in the world

kubernetes

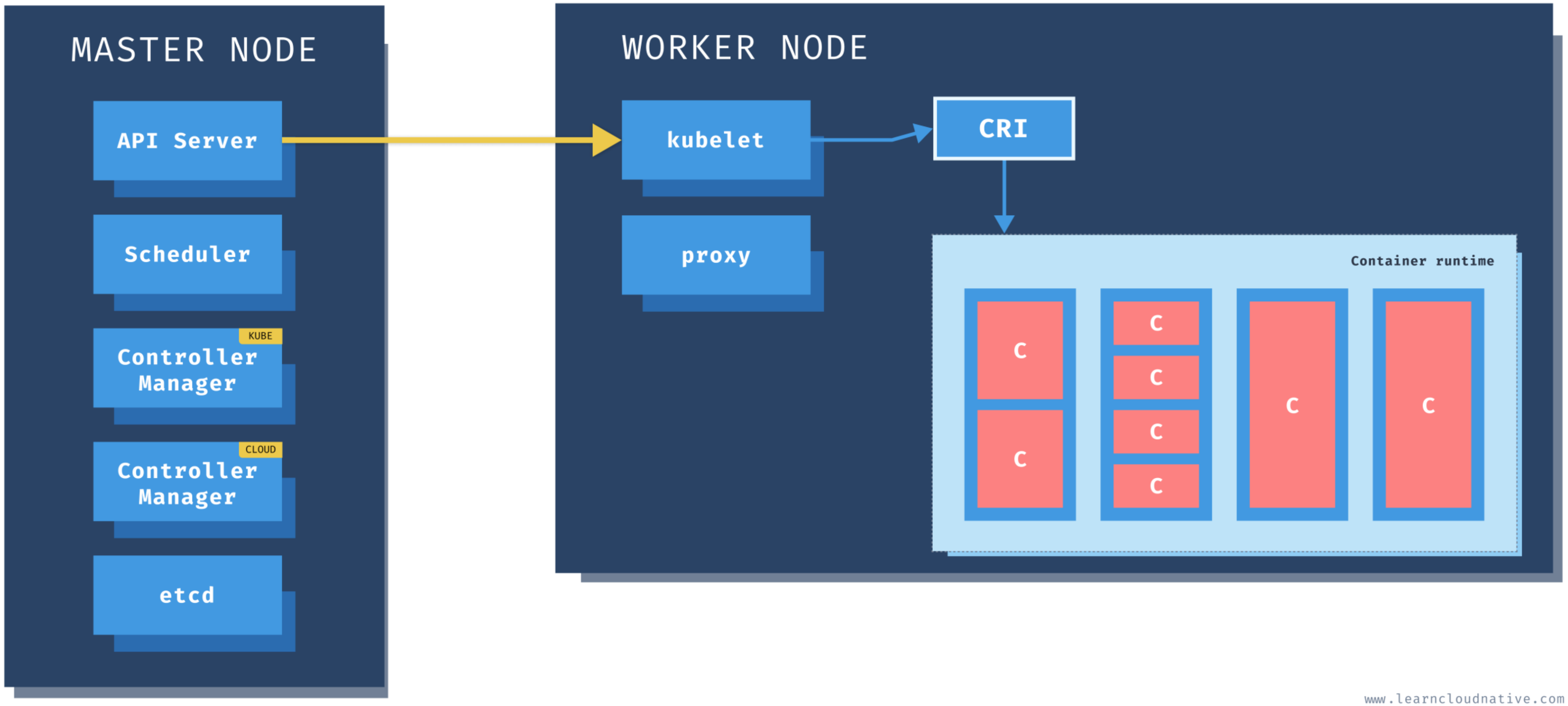
Kubernetes Architecture

Kubernetes Cluster

- A Kubernetes cluster consists of a set of machines (physical or virtual), called nodes
- Control Panel or Master Node(s)manages the cluster and nodes
- Node(s) or Worker Node(s) runs containerized workloads
- Nodes can be heterogeneous (small, large, GPUs, Linux, Windows, etc.)

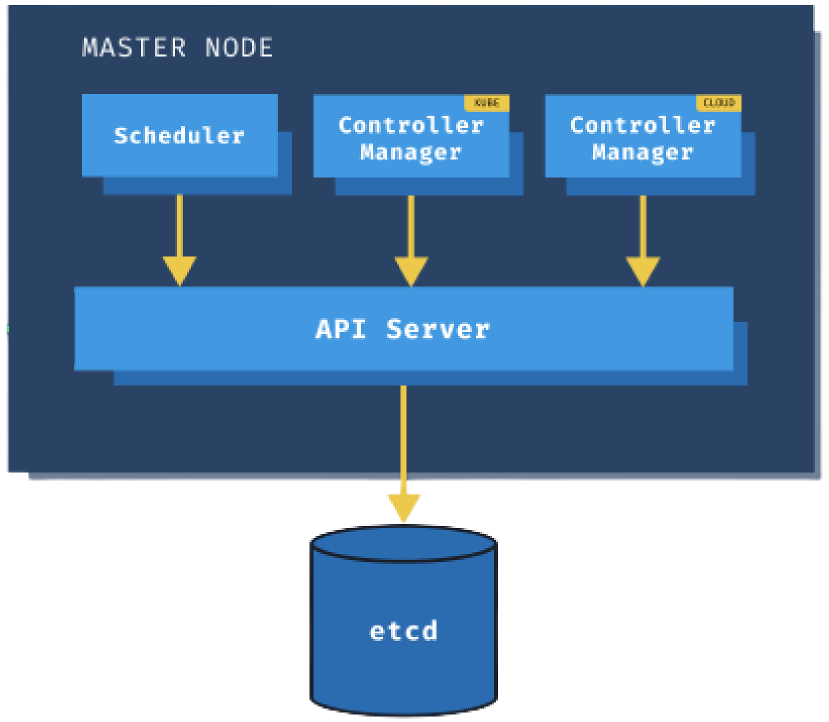
kubernetes

Kubernetes Cluster



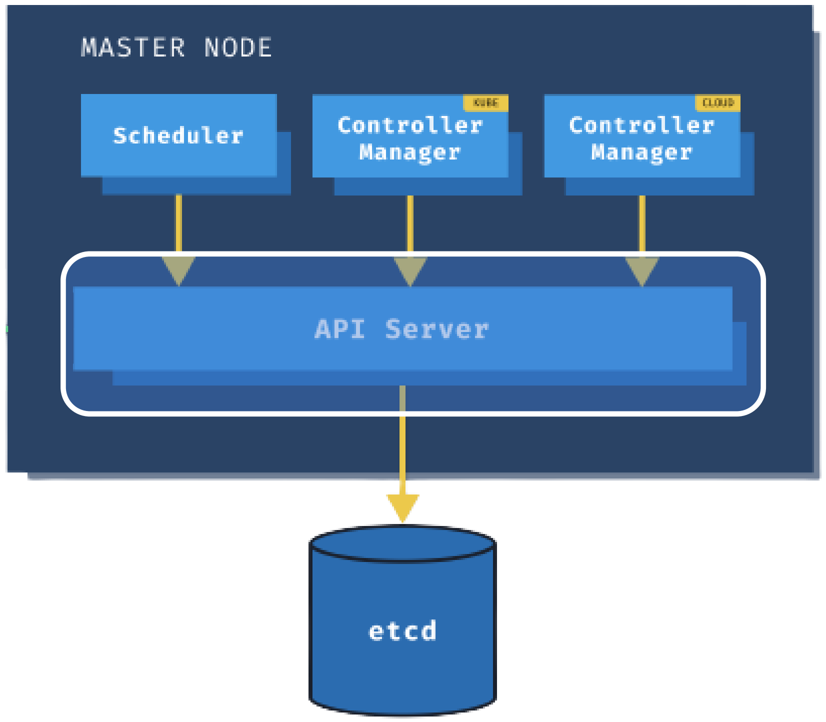
Control Plane Components

- Make global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events (for example, starting up a new pod when a deployment's replicas field is unsatisfied)
- Control plane components can be run on any machine in the cluster. For simplicity, start all control plane components on the same machine, and do not run user containers on this machine.



Control Plane: API Server

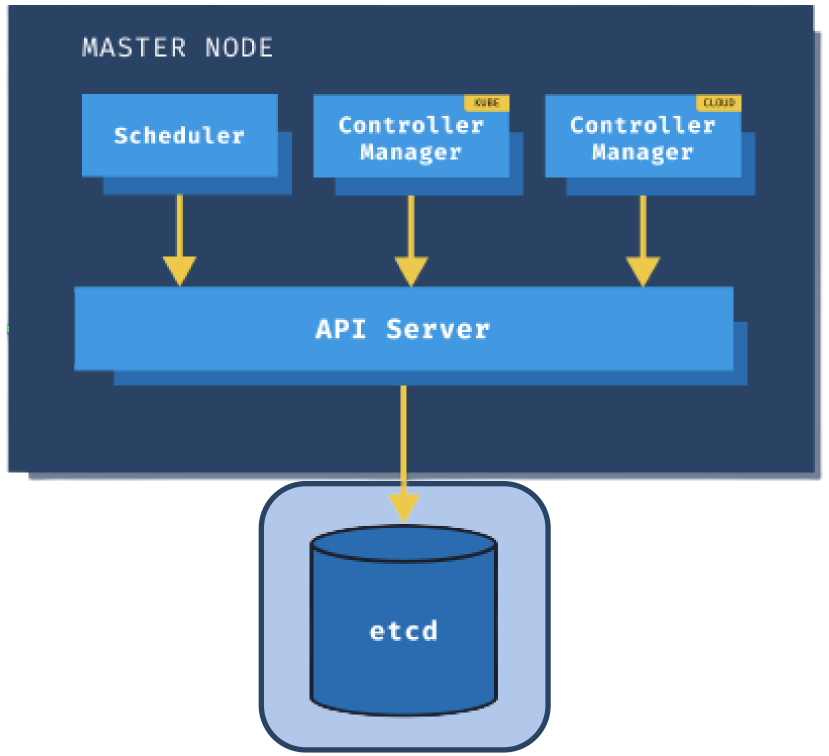
- Component that exposes the Kubernetes API
- The API server is the front end for the Kubernetes control plane
- Every request to manage your cluster should use API Server
- Main implementation of a Kubernetes API server is kube-apiserver.
- Scale horizontally, deploying more instances and balance traffic between those instances



kubernetes

Control Plane: etcd

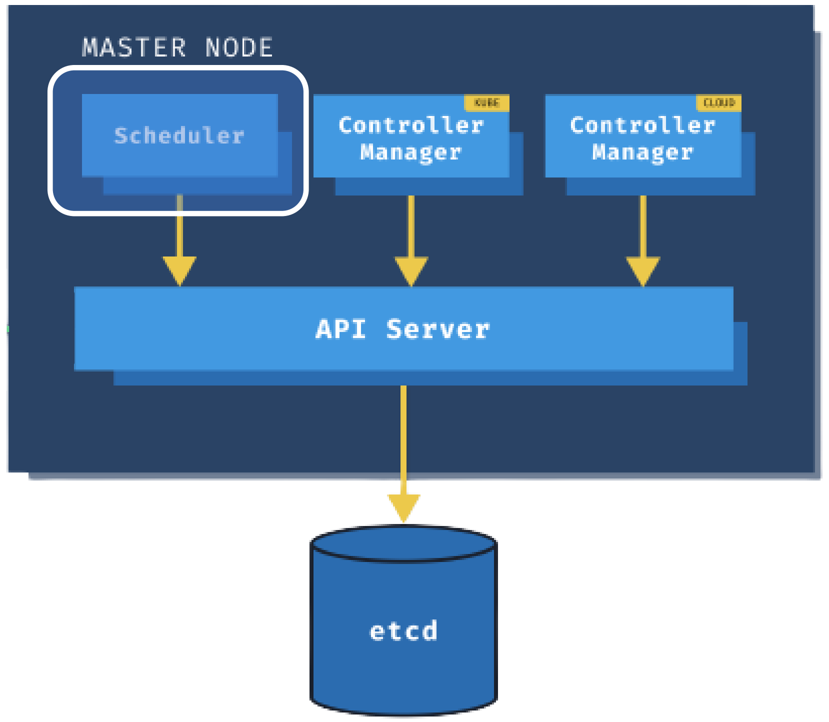
- Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data
- Used as cluster backing store because of simplicity and performance
- Don’t have a backup plan implementation on the cluster, you need to do it manually



kubernetes

Control Plane : Scheduler

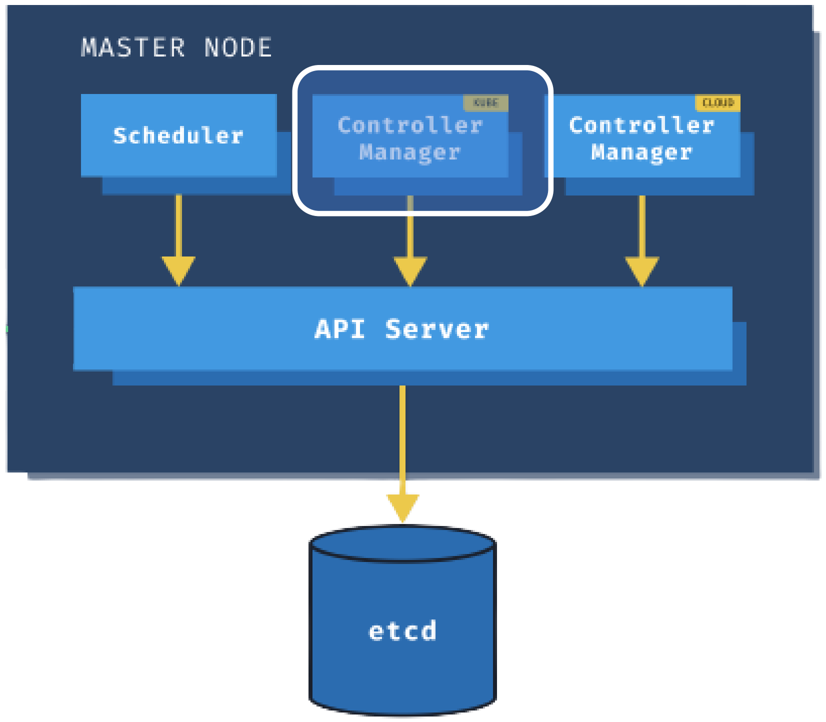
- Component that watches for newly created Pods with no assigned node, and selects a node for them to run on
- Factors taken into account for scheduling decisions include: individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications



kubernetes

Control Plane : Kube Controller Manager

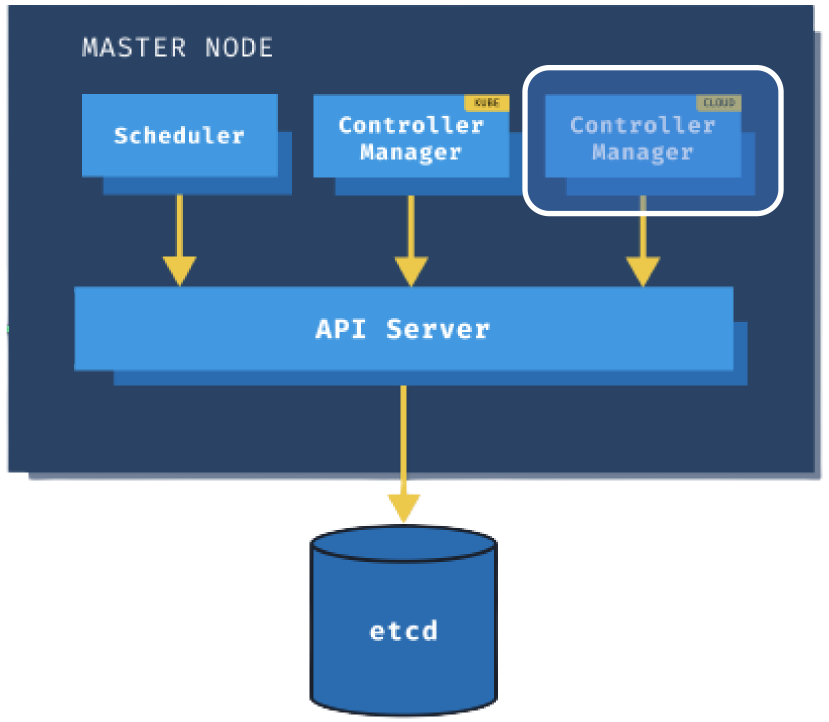
- Component that runs controller processes
- Each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process
- Some types of these controllers are:
  - Node controller: Responsible for noticing and responding when nodes go down.
  - Replication controller: Responsible to run the number of Pods defined on each ReplicaSet
  - ServiceAccount controller: Create default ServiceAccounts for new namespaces



kubernetes

Control Plane : Cloud Controller Manager

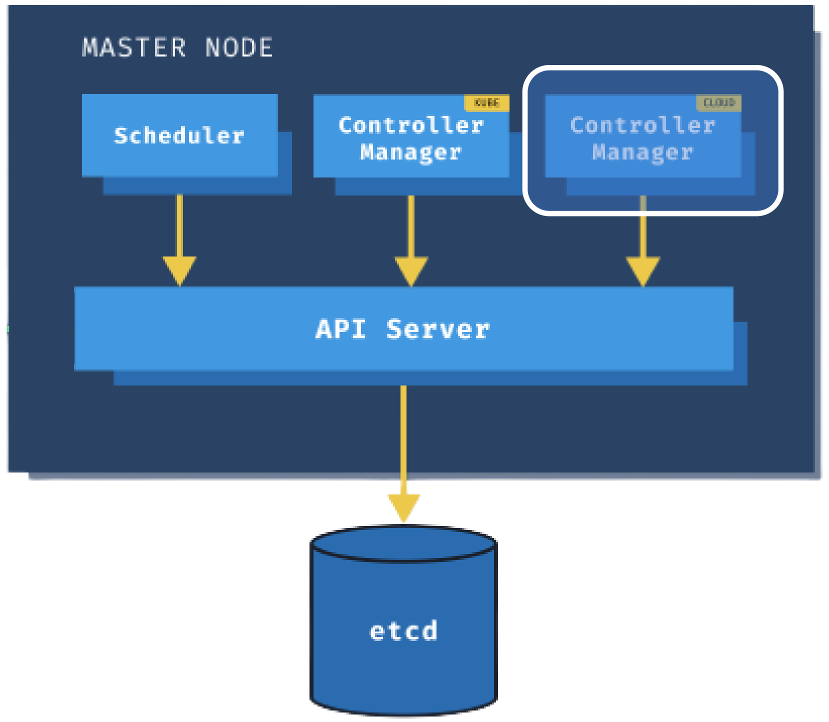
- Component that embeds cloud-specific control logic
- Lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that only interact with your cluster
- If you are running Kubernetes on your own premises, or in a learning environment inside your own PC, the cluster does not have a cloud controller manager.



kubernetes

Control Plane : Cloud Controller Manager

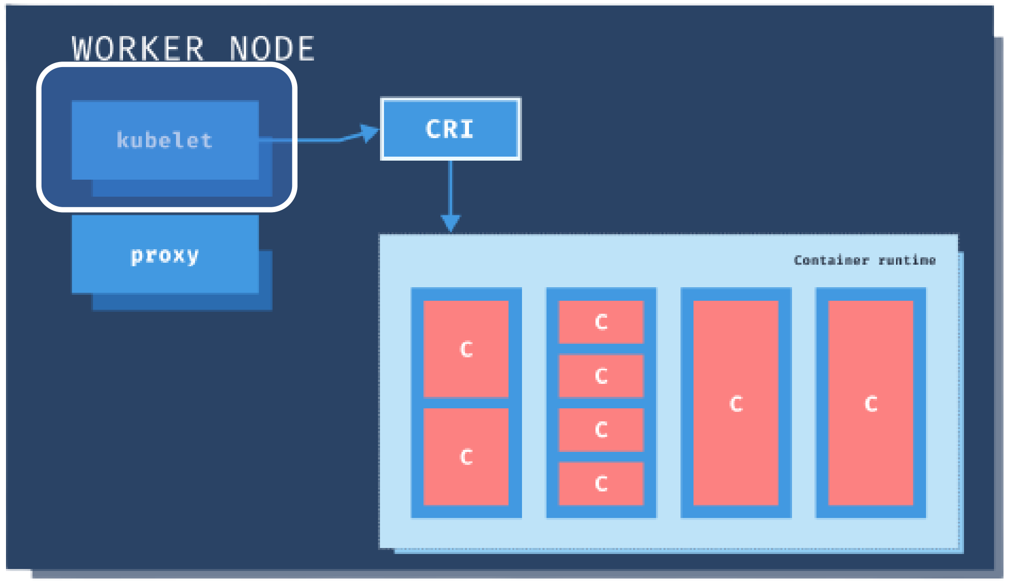
- The following controllers can have cloud provider dependencies:
  - Node controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
  - Route controller: For setting up routes in the underlying cloud infrastructure
  - Service controller: For creating, updating and deleting cloud provider load balancers



kubernetes

Node: Kubelet

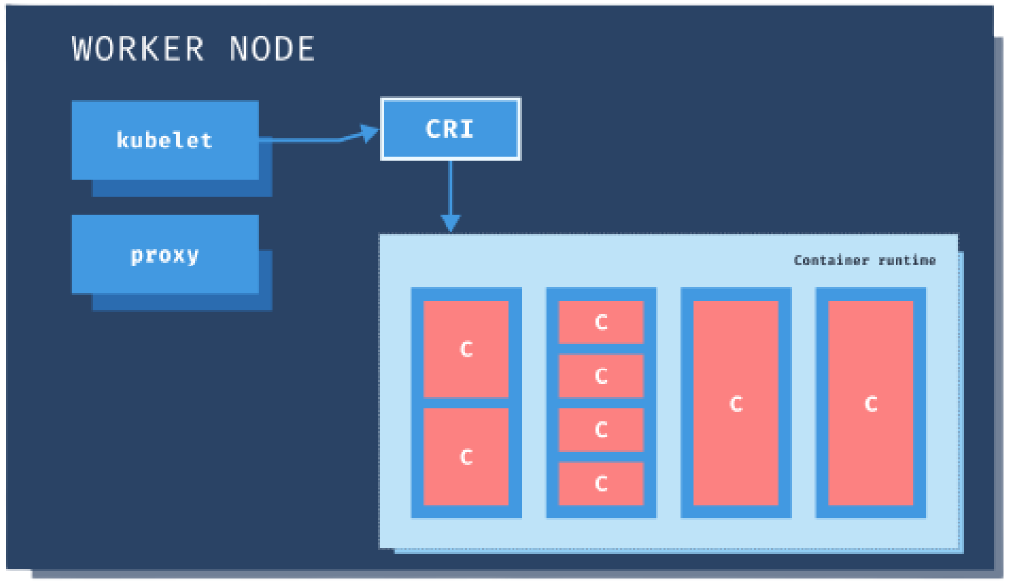
- An agent that runs on each node in the cluster and makes sure that containers are running in a Pod
- Takes a set of PodSpecs and ensures that the containers described in those PodSpecs are running and healthy
- Doesn't manage containers which were not created by Kubernetes



kubernetes

Node Components

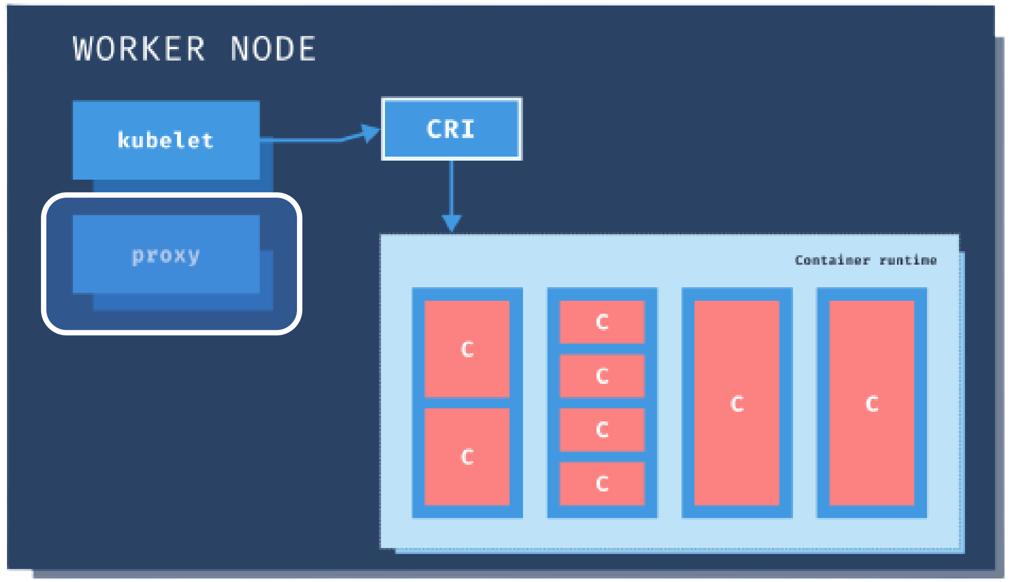
- Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment
- These components needs to run on a machine to be considered part of the Kubernetes cluster



kubernetes

Node: Kube Proxy

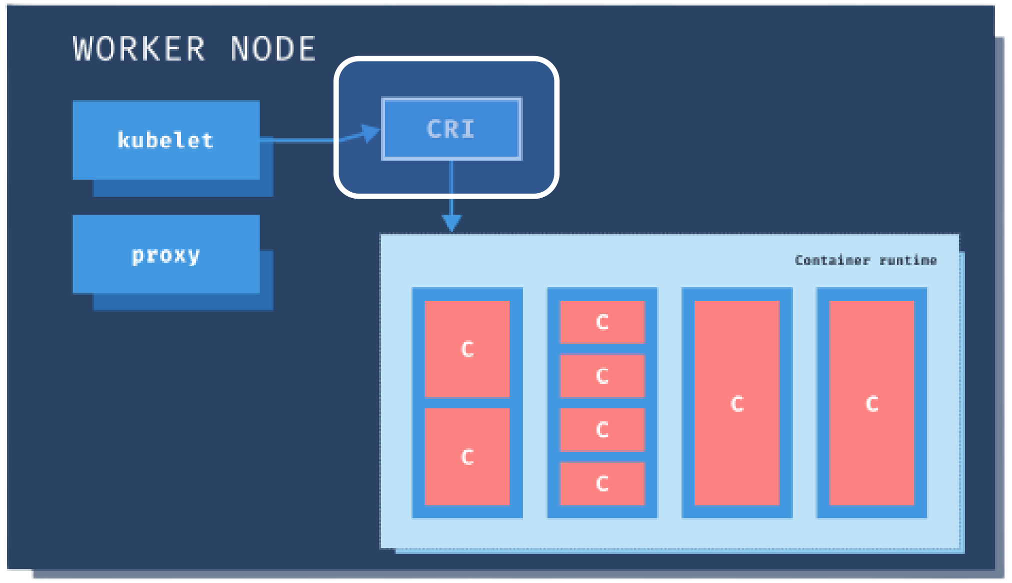
- Network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept
- Maintains network rules on nodes to allow network communication to your Pods from network sessions inside or outside of your cluster
- kube-proxy uses the operating system packet filtering layer



kubernetes

Node: CRI

- The container runtime is the software that is responsible for running containers
- Kubernetes supports container runtimes such as containerd, CRI-O, and any other implementation of the Kubernetes CRI (Container Runtime Interface)



kubernetes

Questions?

kubernetes