INTRO TO DOCKER

CONTAINER

- A container is a runtime instance of a docker image.

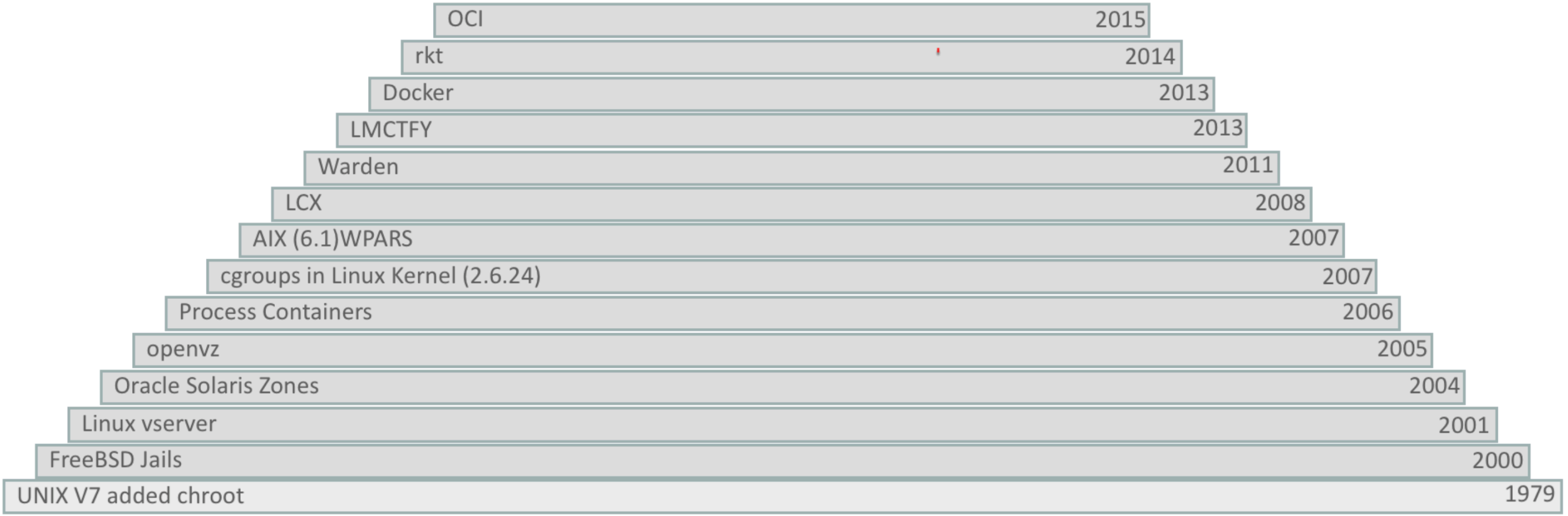
- A Docker container consists of
  - A Docker image
  - An execution environment
  - A standard set of instructions

CONTAINER

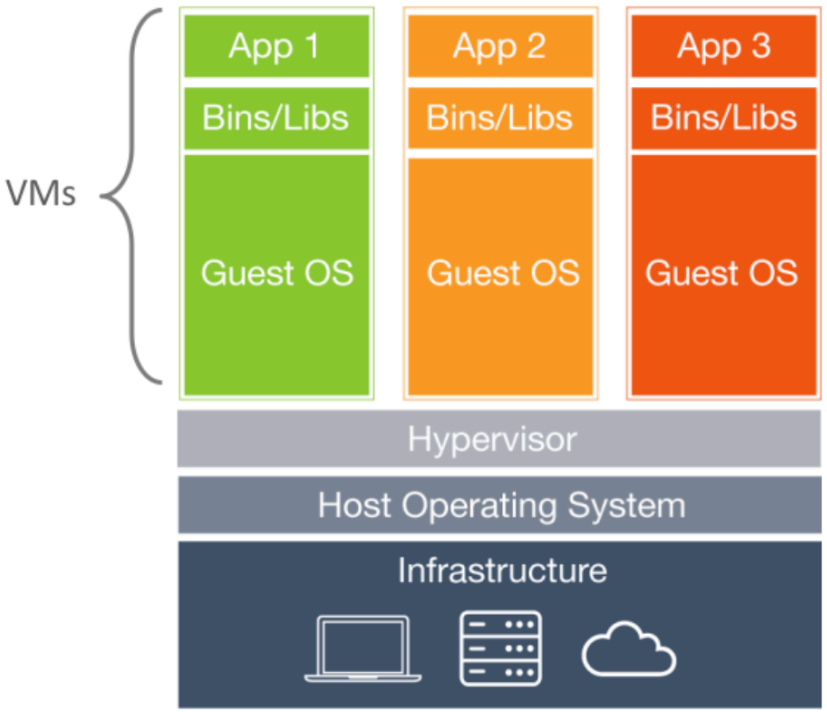
- The concept is borrowed from Shipping Containers, which define a standard to ship goods globally. Docker defines a standard to ship software.

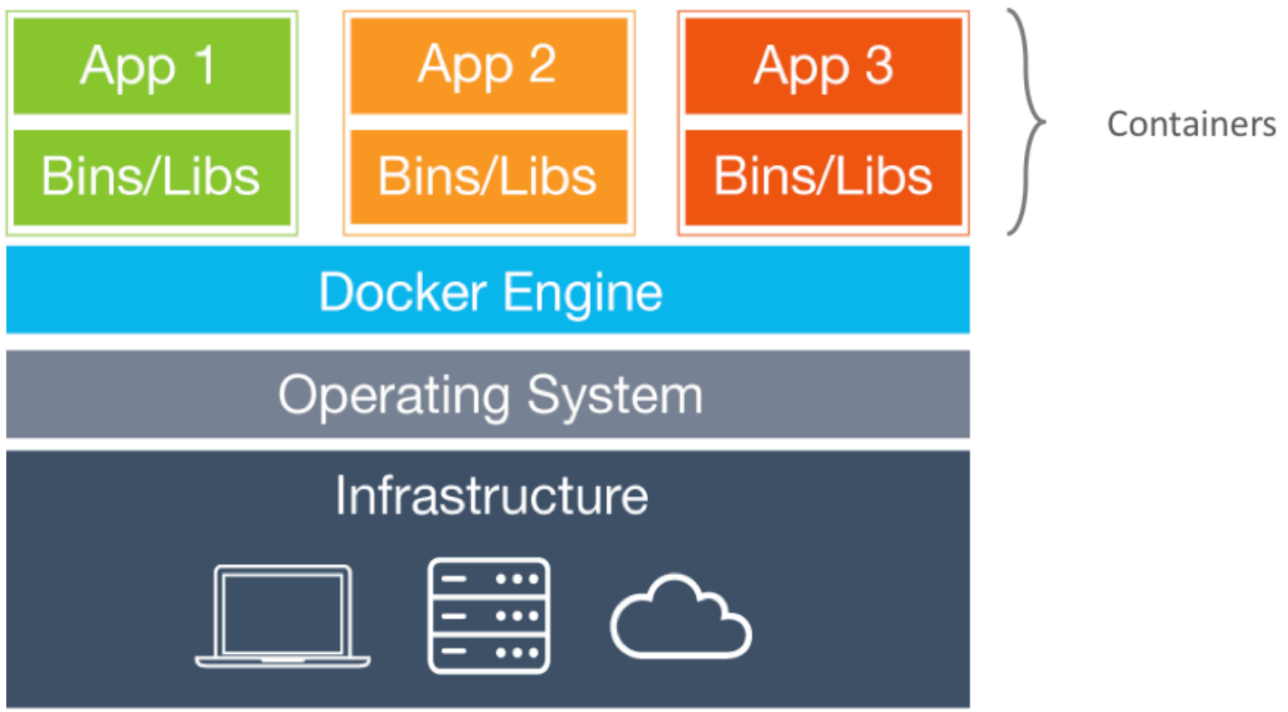
The history of Unix containers

While Docker has been playing a key role in adoption of the Linux container technology, they did not invent the concept of containers



Virtual Machines vs. Containers





Virtual Machines

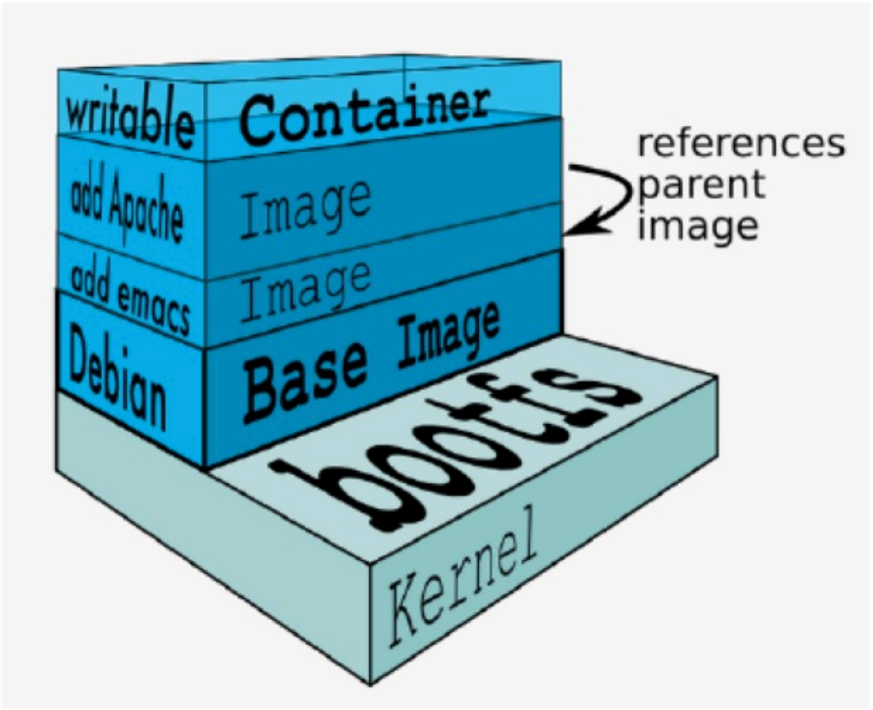
- Each virtual machine (VM) includes the app, the necessary binaries and libraries and an **entire guest operating system**

Containers

- Containers include the app & all of its dependencies, but **share the kernel** with other containers.
- Run as an isolated process in userspace on the host OS
- Not tied to any specific infrastructure – containers run on any computer, infrastructure and cloud.

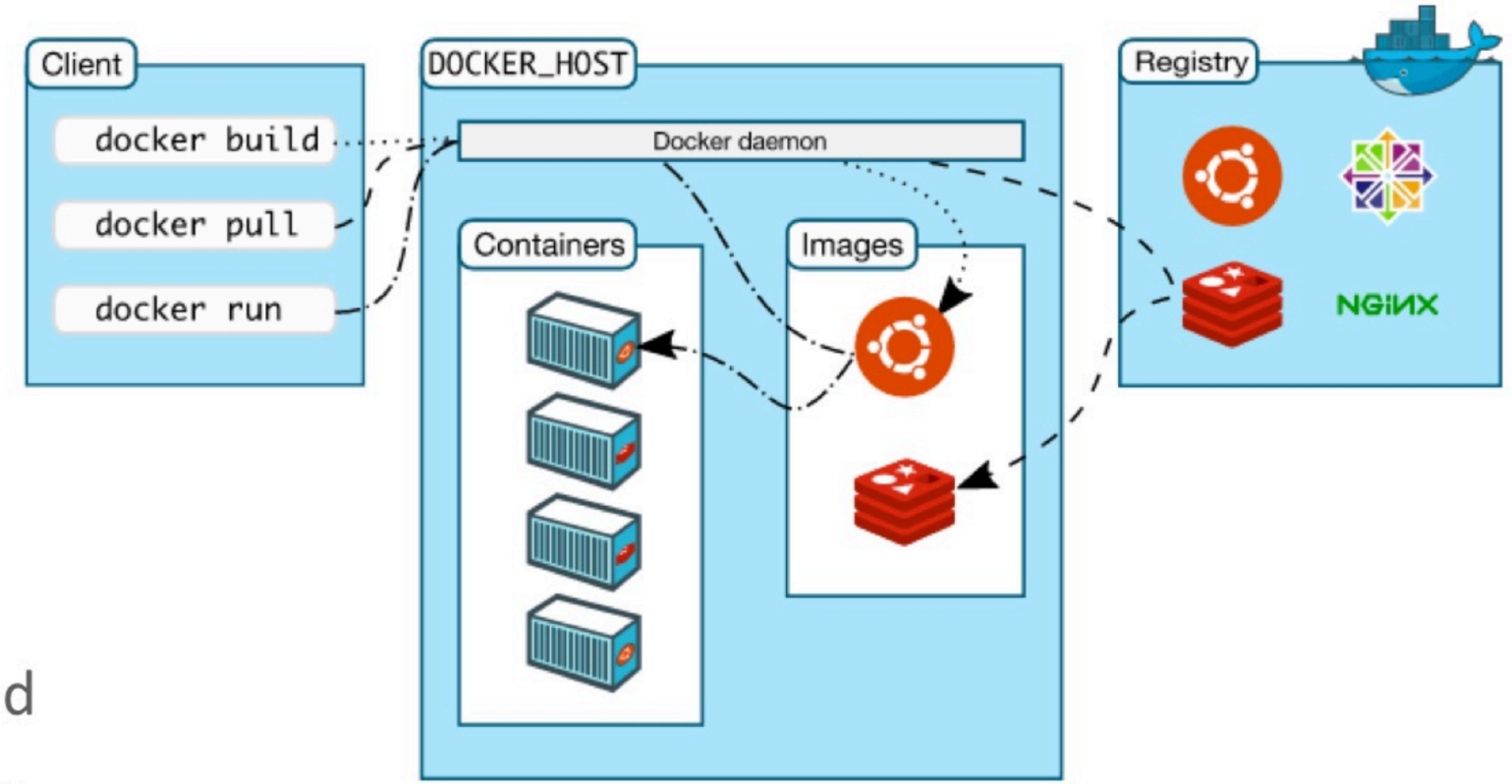
Docker Images

- An image is a collection of files and some meta data
- Images are comprised of multiple layers, multiple layers referencing/based on another image
- Each image contains software you want to run
- Every image contains a base layer
- Docker uses a copy on write file system
- Layers are read only



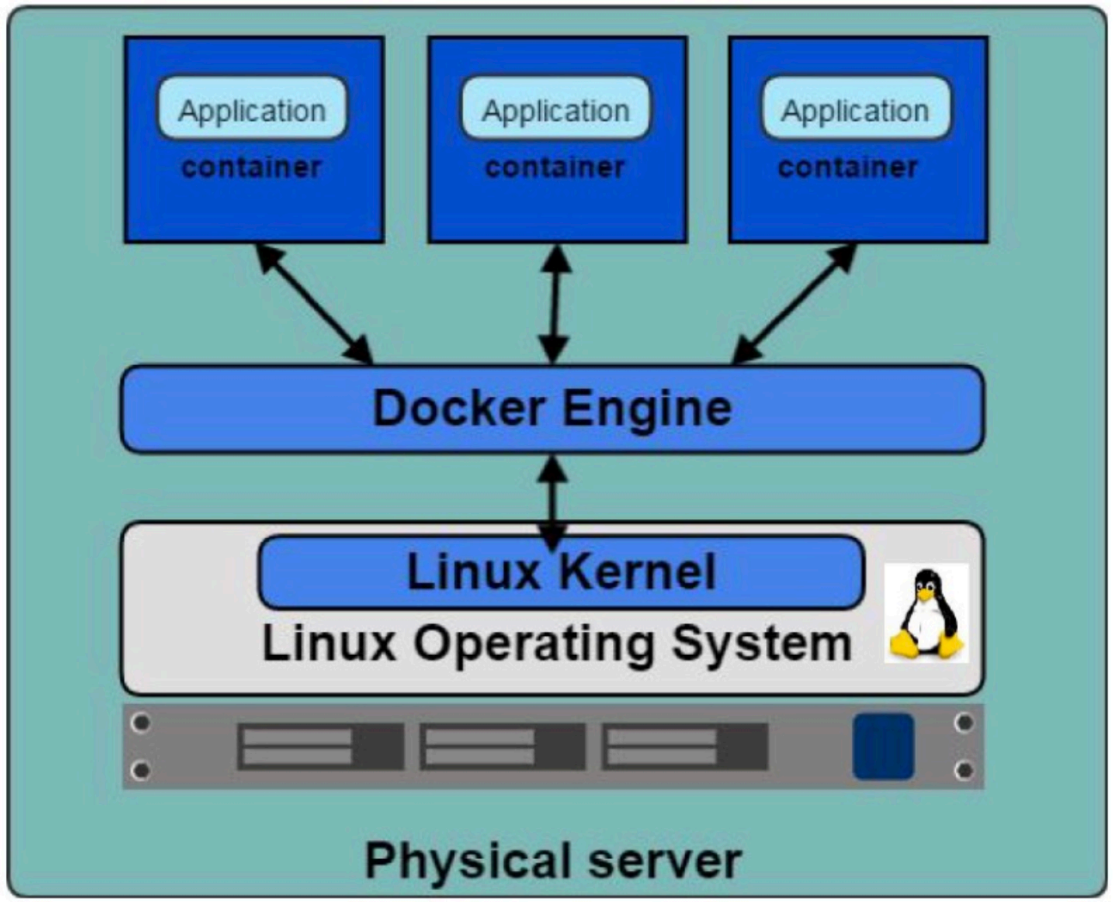
Docker Architecture

- Docker client – Command Line Interface (CLI) for interfacing with the Docker
- Dockerfile – Text file of Docker instructions used to assemble a Docker Image
- Image – Hierarchies of files built from a Dockerfile, the file used as input to the docker build command
- Container – Running instance of an Image using the docker run command
- Registry – Image repository



Docker Engine

- Container execution and admin
- Uses Linux Kernel namespaces and control groups
- Namespaces provide for isolated workspace



Why Containers?

A container is packaged as an entire runtime environment: the service/app plus all dependencies, libraries, & configuration files needed to run it

Portable across environments & lightweight (share the OS)

Developers care because:

- Quickly create ready-to-run packaged applications, low cost deployment and replay
- Automate testing, integration, packaging
- Reduce / eliminate platform compatibility issues ("It works in dev!")
- Support next gen applications (microservices)

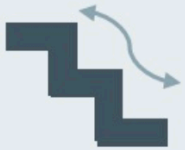
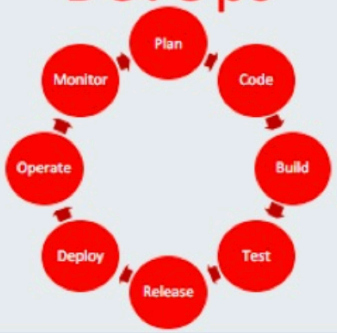
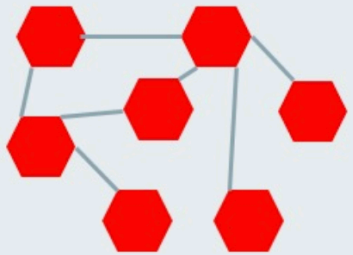
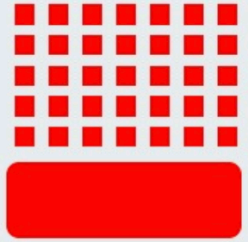
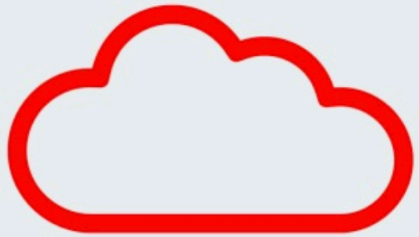
IT cares because:

- Improve **speed** and frequency of releases, reliability of deployments
- Makes app lifecycle efficient, consistent and repeatable – configure once, run many times
- Eliminate environment inconsistencies between development, test, production
- Improve production application resiliency and scale out / in on demand

How We Have Seen Docker Being Used

- Dev/Test of Legacy Apps
- New App Dev (including parts of legacy apps)
- Code Agility, CI/CD Pipeline, DevOps
- Adoption of Open Source
- Microservices & Cloud Native Apps

Multi-Dimensional Evolution of Computing

Development Process	Application Architecture	Deployment and Packaging	Application Infrastructure
Waterfall 	Monolithic 	Physical Server 	Datacenter 
Agile 	N-Tier 	Virtual Servers 	Hosted 
DevOps 	Microservices 	Containers 	Cloud 

LET'S HOP INTO THE HANDS-ON LAB

https://karthequian.github.io/ContainerWorkshop/

Module 1

KUBERNETES INTRO

How do you manage running containers on a single host, and, more importantly, across your whole infrastructure?

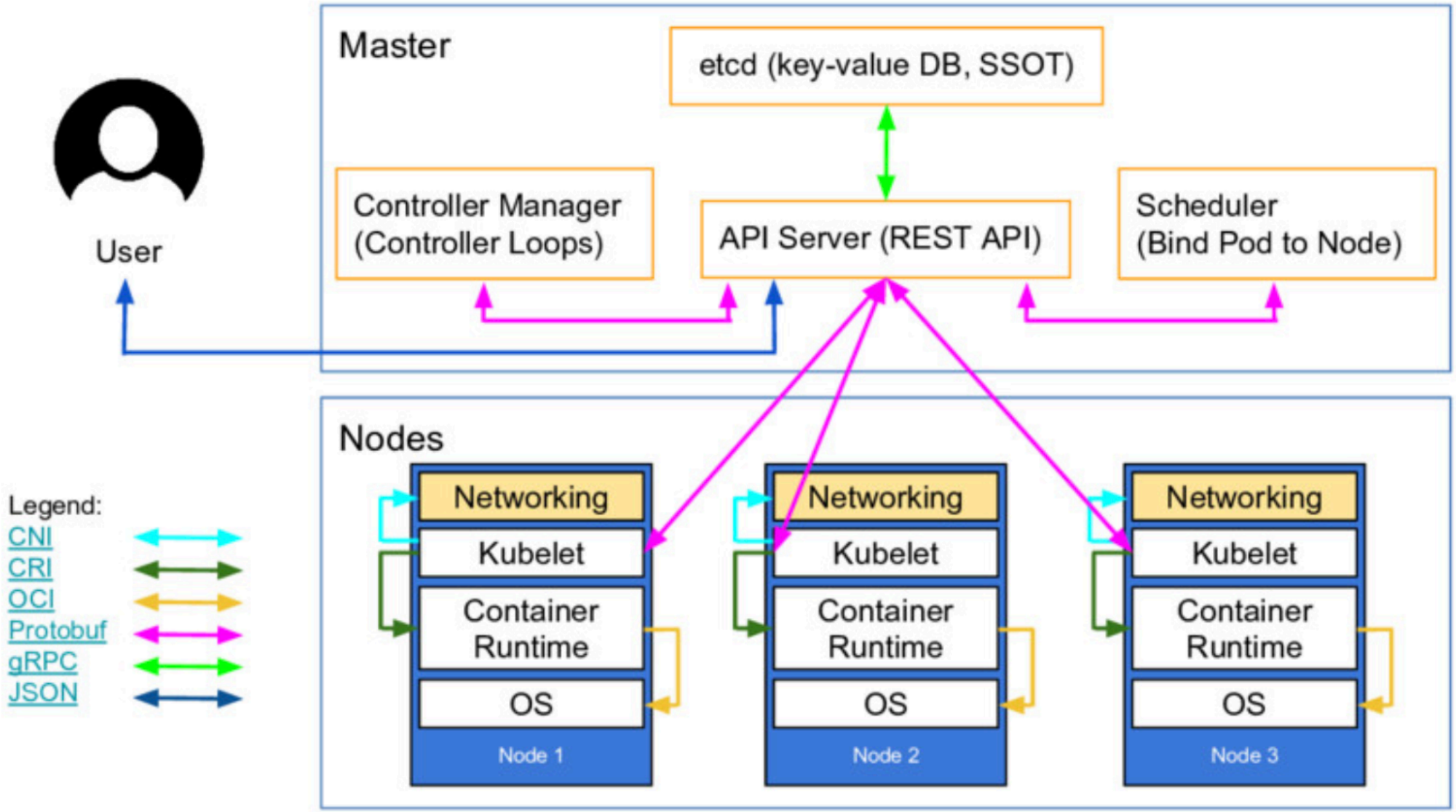
ORCHESTRATOR FEATURES

- Provision hosts
- Instantiate containers on a host
- Restart failing containers
- Expose required containers as services outside the cluster
- Scale up or down the cluster

KUBERNETES (K8S)...

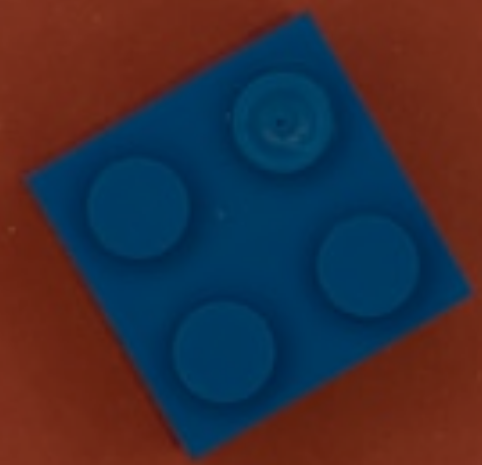
- Defintion: an open-source platform designed to automate deploying, scaling, and operating application containers.
- Goal: foster an ecosystem of components and tools that relieve the burden of running applications in public and private clouds.

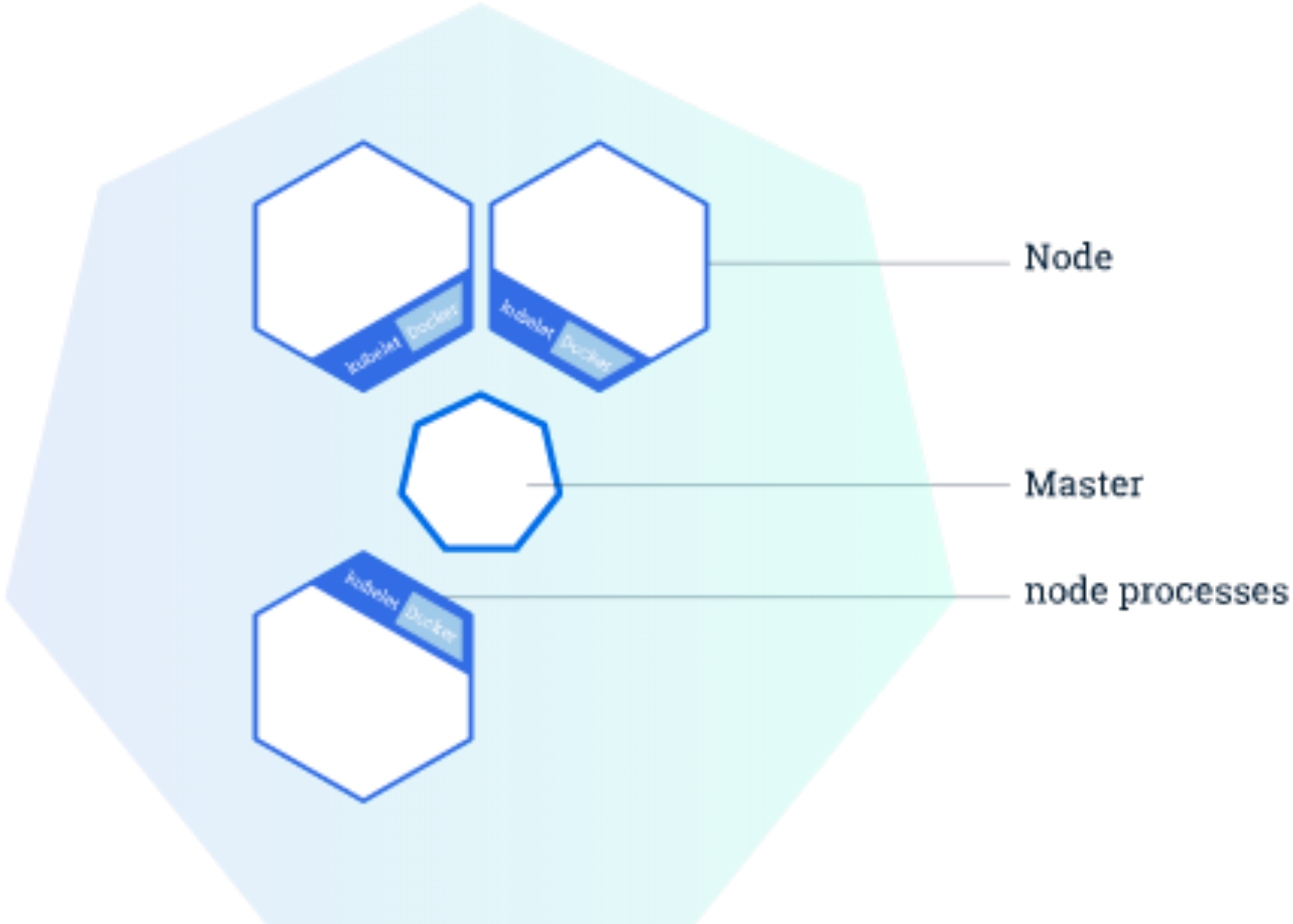
Kubernetes' high-level component architecture



Basic Building Blocks

Nodes and Pods

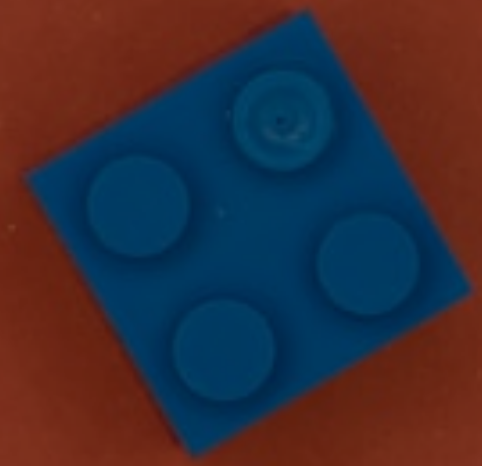




Kubernetes cluster

NODE

The node serves as a worker machine in a k8s cluster. One important thing to note is that the node can be a physical computer, or a virtual machine



NODE REQUIREMENTS

a) a kubelet running.

b) container tooling like Docker

c) a kube-proxy process running

d) supervisord

RECOMMENDATION

- If you're using Kubernetes in production, it is a typically recommended to have at least a 3 node cluster.

MINIKUBE

Lightweight Kubernetes implementation that creates a VM on your local machine and deploys a simple cluster containing only one node

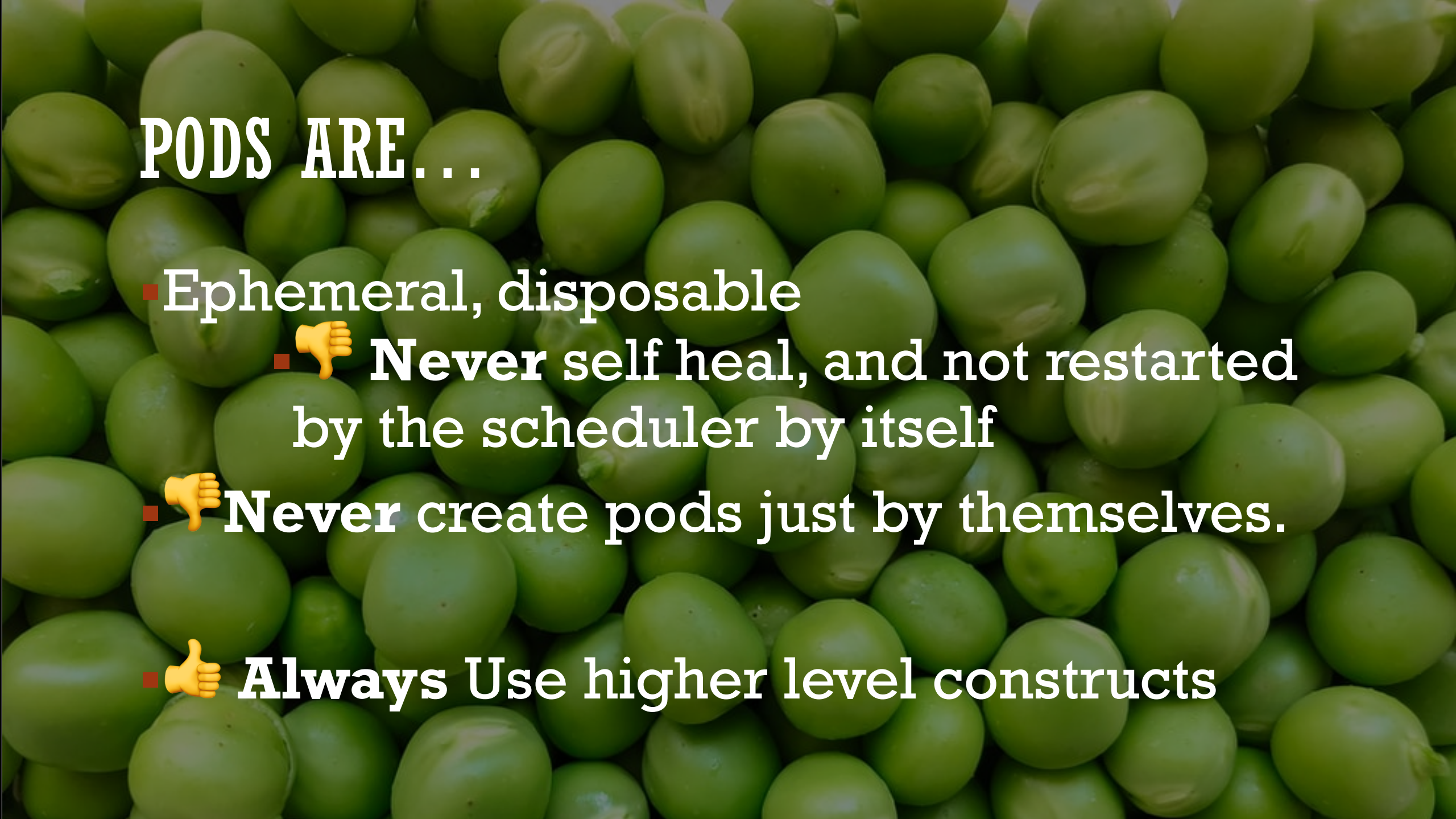

Pods

POD

The simplest unit that you can interact with. You can create, deploy and delete pods and it represents one running process on your cluster.

WHAT'S IN THE POD?

- Your Docker application container
- Storage resources
- Unique network IP
- Options that govern how the container(s) should run



PODS ARE…

- Ephemeral, disposable
  - 👎 **Never** self heal, and not restarted by the scheduler by itself
- 👎 **Never** create pods just by themselves.
- 👍 **Always** Use higher level constructs

RECOMMENDATION

Never use a pod directly; use a controller instead- like a deployment.

POD STATES

Pending

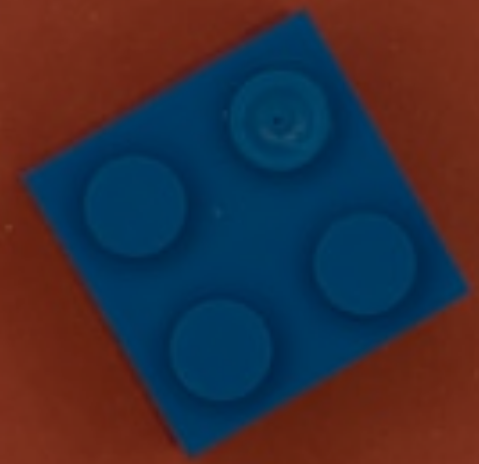
Running

Succeeded

Failed

Crashloopbackoff

Deployment, ReplicaSet, Services



CONTROLLERS

- Pods are building blocks for building applications
- Use controllers to build higher level constructs

BENEFITS OF CONTROLLERS

- Application reliability
- Scaling
- Load balancing

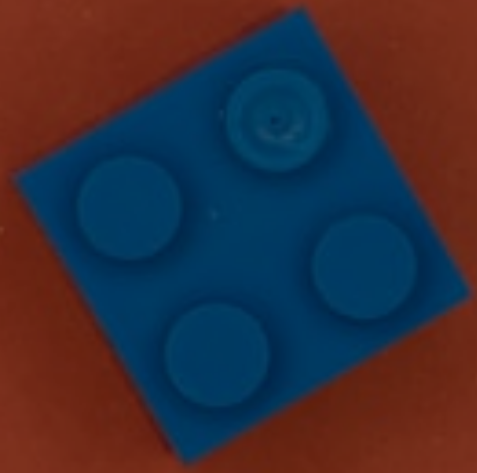
KINDS OF CONTROLLERS

- Replica Sets
- Deployments
- Daemonsets
- Jobs
- Services

REPLICA SETS

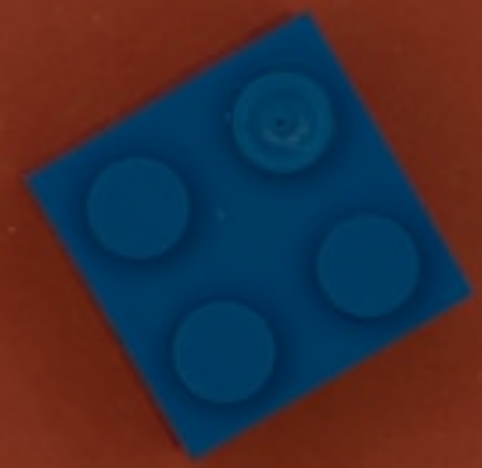
A ReplicaSet has 1 job-

Ensure that a specified number of replicas for a pod are running at all times.



DEPLOYMENT

A Deployment controller provides declarative updates for Pods and ReplicaSets




Pod
Replica Set
Deployment

DEPLOYMENT CONTROLLER USE CASES

- Pod management: Running a replica set allows us to deploy several pods and check their status as a single unit.
- Scaling a replica set, scales out the pods and allows for the deployment to handle more traffic.

DEPLOYMENT CONTROLLER USE CASES

- Pod updates
  - New replicas created for pod updates
- Pod rollbacks
  - Easily rollback to the old replica

DEPLOYMENT CONTROLLER USE CASES

- Pause and Resume:
  - Used with larger changesets
  - Pause deployment, make changes, resume deployment
- ** Seldom used

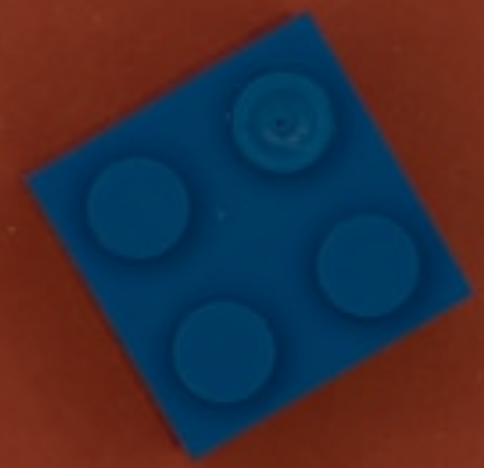
DEPLOYMENT USE CASES

- Status
  - Easy way to check the health of pods, and identify issues

REPLICATION CONTROLLER

Early implementation of deployments and replica sets.

Use deployments + replica sets instead.



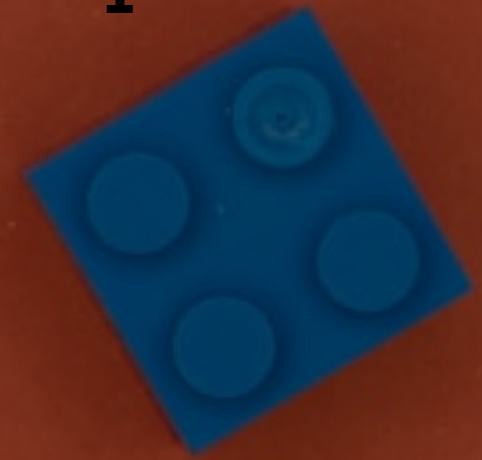
DEPLOYMENTS V/S REPLICATION CONTROLLERS

- Replication Controllers:
  - Initial implementation for controlling pods (1st generation)
- Deployments + Replica Sets
  - Existing implementation

DAEMONSETS

Ensure that all Nodes run a copy of a specific Pod.

As nodes are added or removed from the cluster, a deamonset will add or remove the required pods.



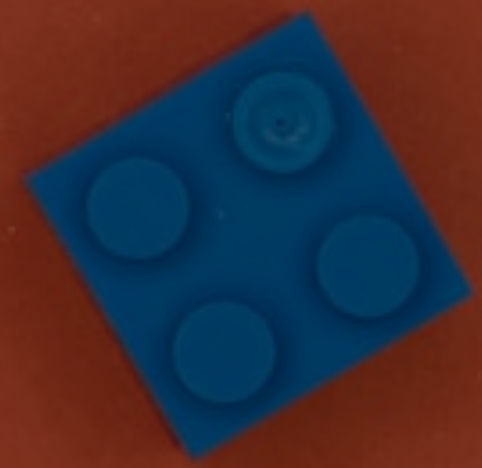
COMMON USE: DAEMONSETS

- Used to run single log aggregator/monitoring agent on nod

JOBS

Supervisor process for pods carrying out batch jobs.

Run individual processes that run once and complete successfully.



COMMON USE: JOBS

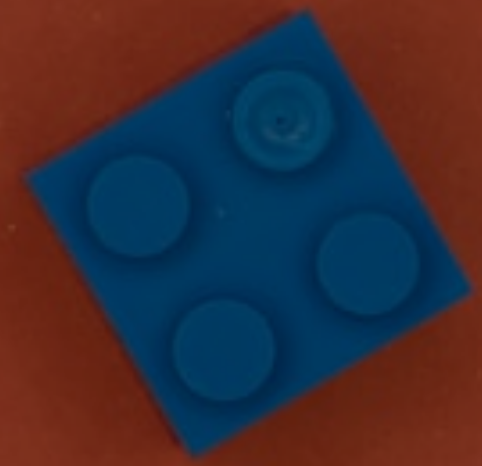
- Typically run as cron jobs to run a specific process at a specific time.

- Examples: Nightly reports, database backups

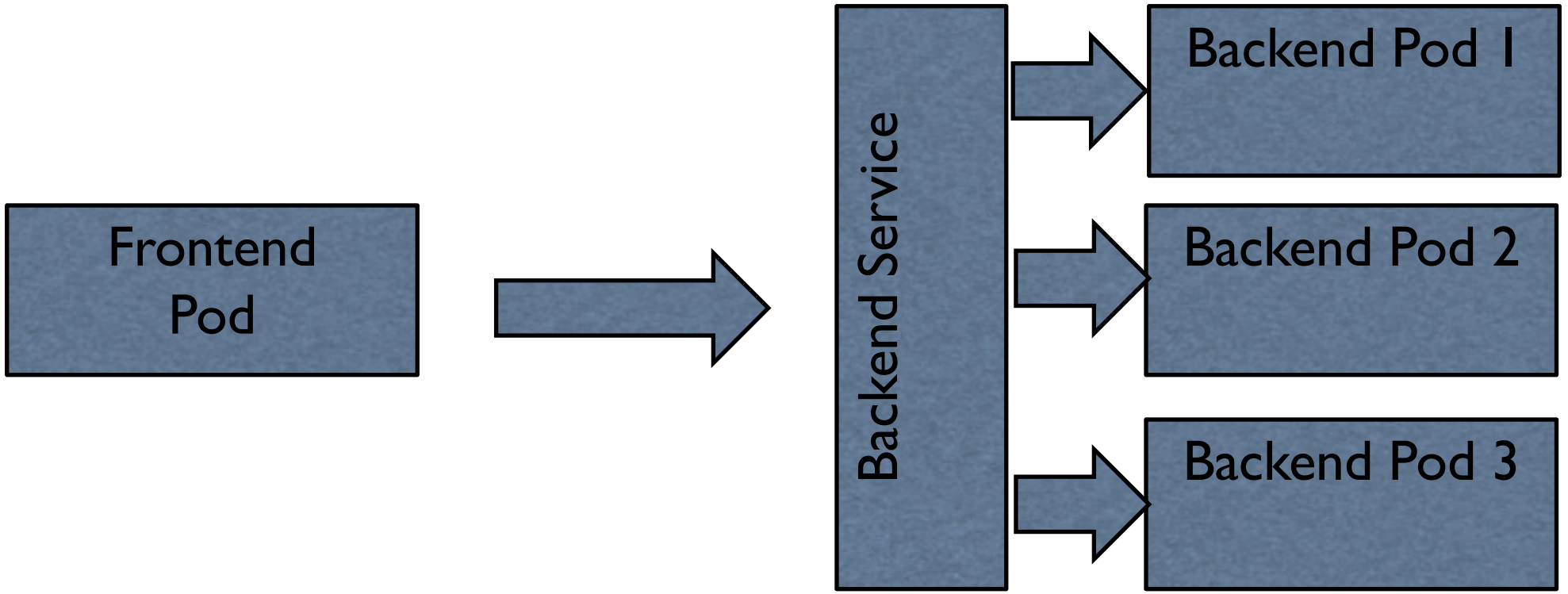
SERVICES

Allow the communication between one set of deployments with another.

Use a service to get pods in 2 deployments to talk to each other.



USING SERVICES



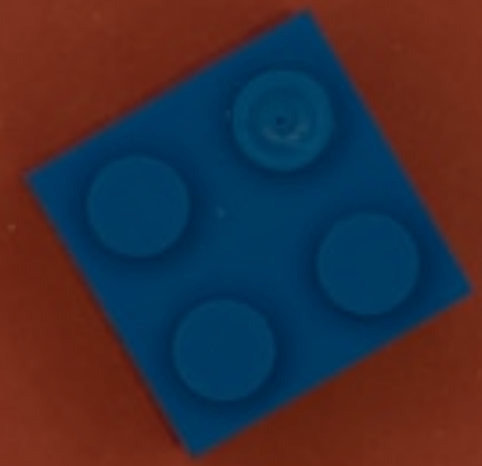
KINDS OF SERVICES

- Internal: IP is only reachable within the cluster
- External: Endpoint available through node ip: port (called nodeport)
- Loadbalancer: Exposes application to the internet with a loadbalancer (available with a cloud provider)


Labels,
selectors,
namespaces

LABELS

Labels are key/value pairs that are attached to objects like pods, services and deployments. Labels are for users of kubernetes to identify attributes for objects.



LABELS

Labels are used to organize clusters in some meaningful way

Can be added at deployment time or later on.

EXAMPLES OF LABELS

- "release" : "stable", "release" : "canary"
- "environment" : "dev", "environment" : "qa", "environment" : "production"
- "tier" : "frontend", "tier" : "backend", "tier" : "cache"

SELECTORS

- Used synonymously with labels
- 2 Kinds:
  - Equality Based
  - Set Based

EQUALITY BASED SELECTORS

=	!=
two labels or values of labels should be equal	the values of the labels should not be equal

SET BASED SELECTORS

- 3 Kinds:
- IN
- NOTIN
- EXISTS

SET BASED SELECTORS

- IN: A value should be inside a set of defined values.
- NOTIN: A value should not be in a set of defined values.
- EXISTS: Determines whether a label exists or not.

LABELS + SELECTORS

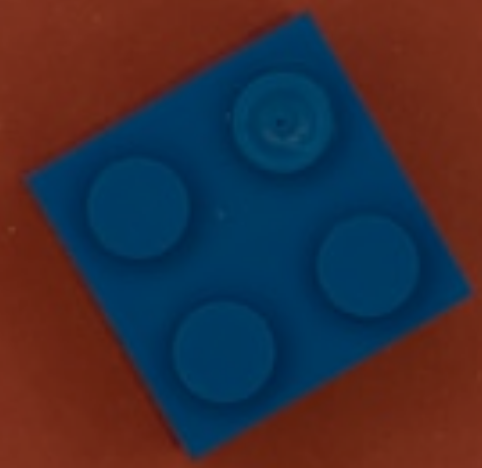
- Labels used with selectors gives you a powerful feature!

- Label + selectors allow you to identify a set of objects


NAMESPACES

NAMESPACES

Allow you to have multiple virtual clusters backed by the same physical cluster



NAMESPACES

- Great for large enterprises.
- Allows teams to access resources, with accountability.
- Great way to divide cluster resources between users.
- Provides scope for names- must be unique in the namespace

NAMESPACES

- “Default” namespace created when you launch Kubernetes.
- Objects placed in “default” namespace at start.
- Newer applications install their resources in a different namespace.

LET'S HOP INTO THE HANDS-ON LAB

https://karthequian.github.io/ContainerWorkshop/

Module2