Containerized In Action (From Docker to Kubernetes)

Siam Chamnankit Co.,Ltd

Who We Are

Joe

Software Delivery Experience in Test (SDET) at SCK, Specialty (Agile, DDD, Java, Software Design)

Meng

Software Delivery Experience in Test (SDET) at SCK, Specialty (DevOps)

Our Playground

https://miro.com/app/board/uXjVKHU6GgI=/

https://tinyurl.com/sckdojo

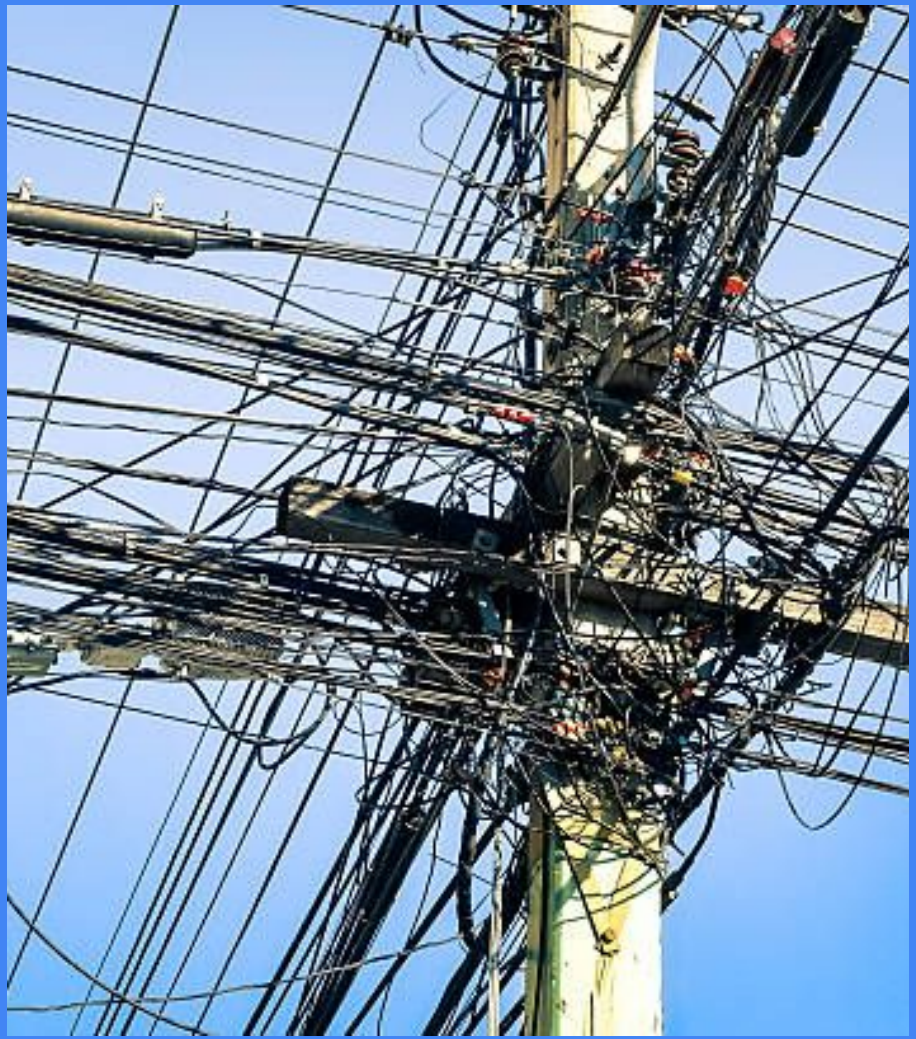
Day 1

Agenda

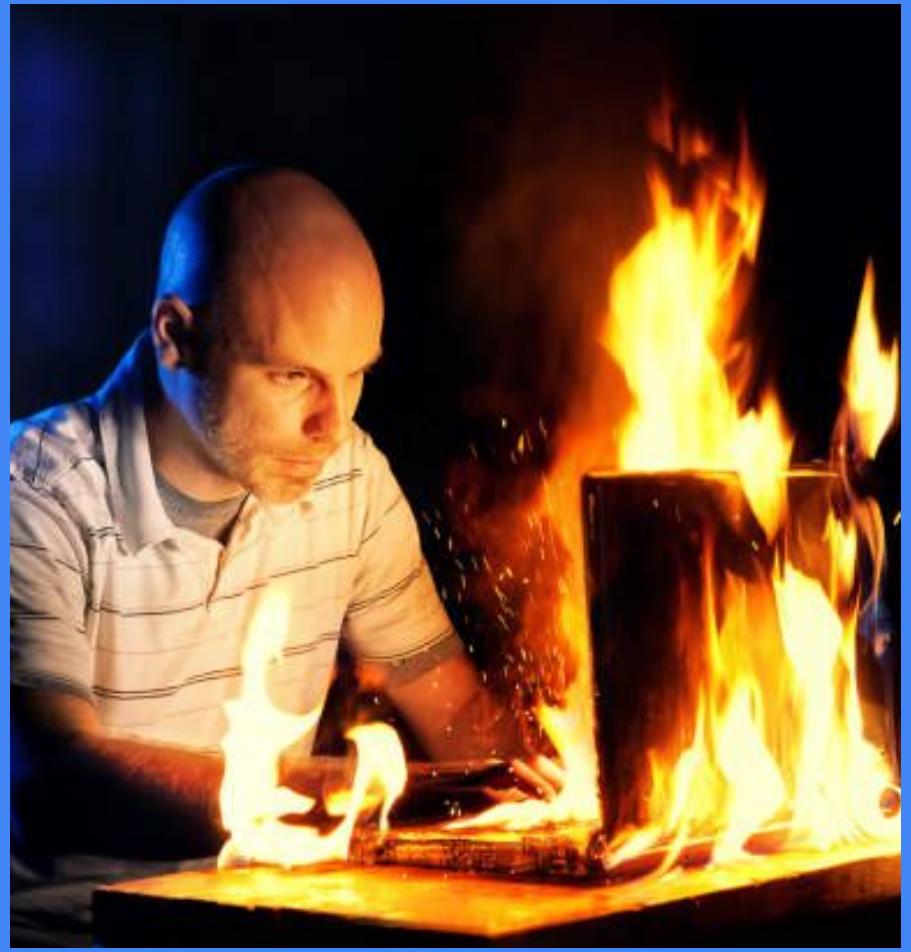
1. Introduce Container with Docker
2. Docker System (a.k.a Engine)
3. Docker Container
4. Docker Image
5. Docker Volume
6. Docker Network
7. Docker Compose
8. Principle for Design Application


What problem is
Docker trying to Solve

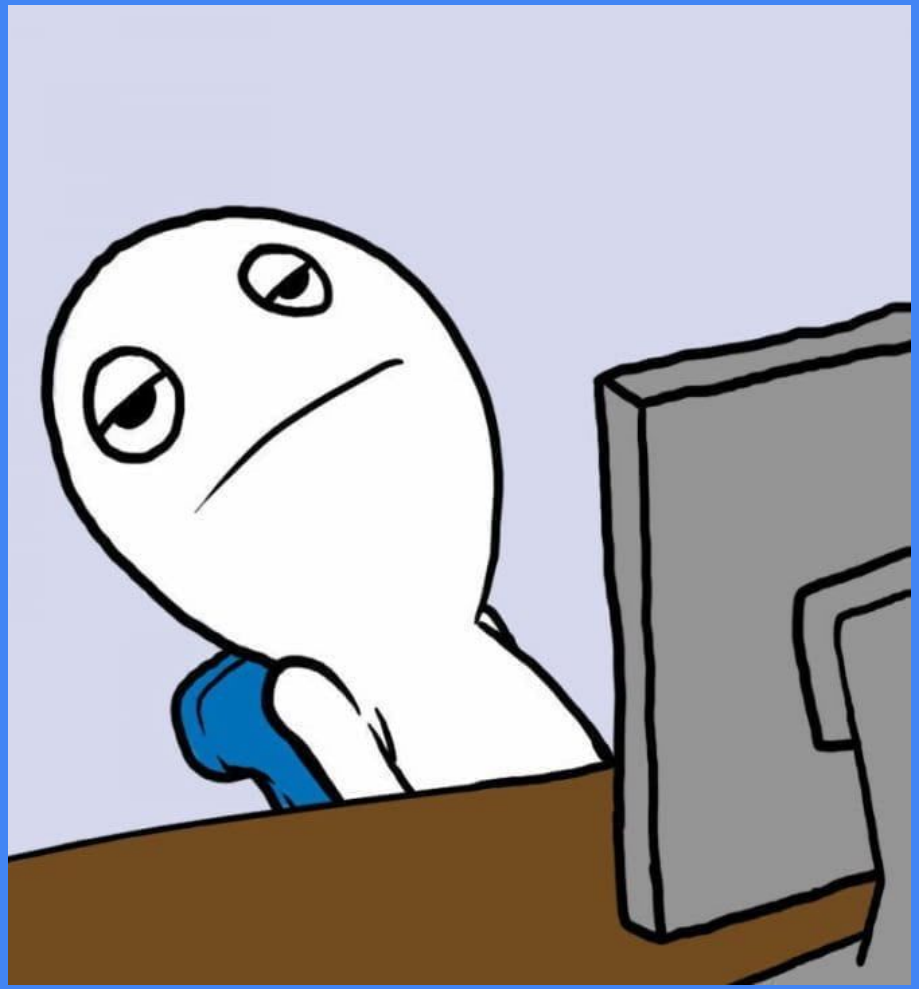
#1 Inconsistency Dependency And Complexity



#2
It works on my machine

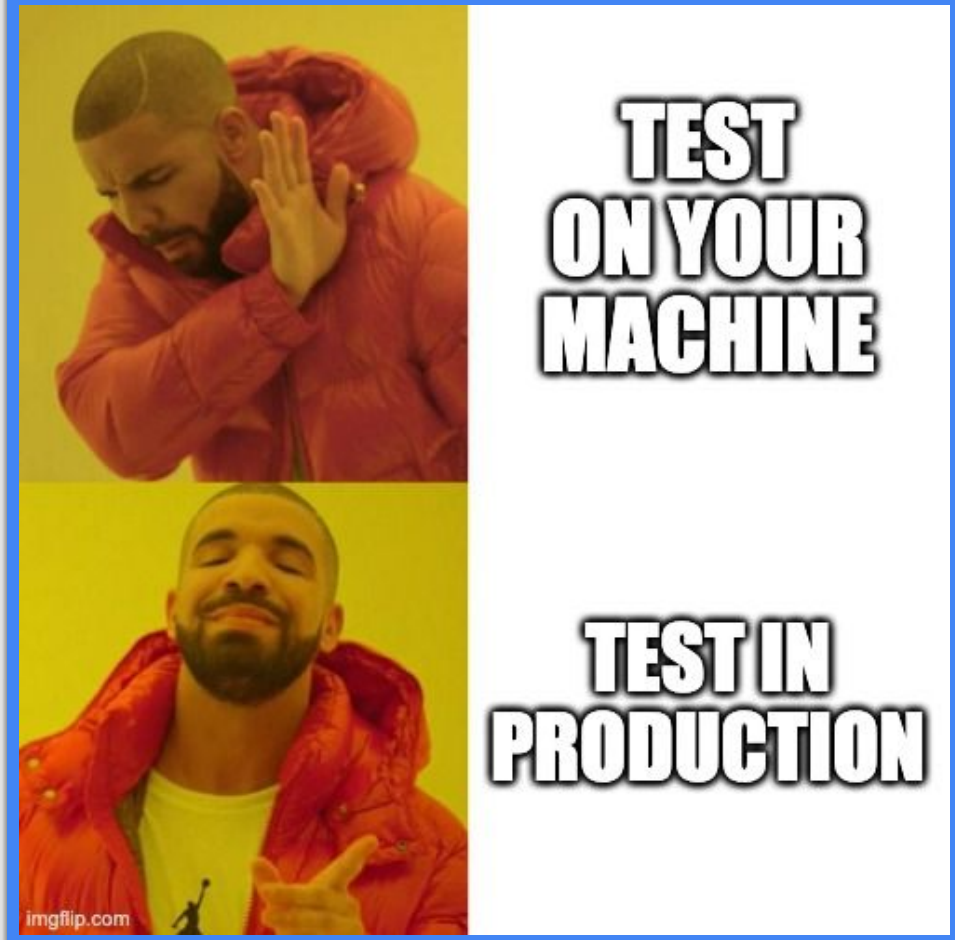


#3
Install,
Upgrade
Remove
and repeat.



#4
Security?

#5
Testing?



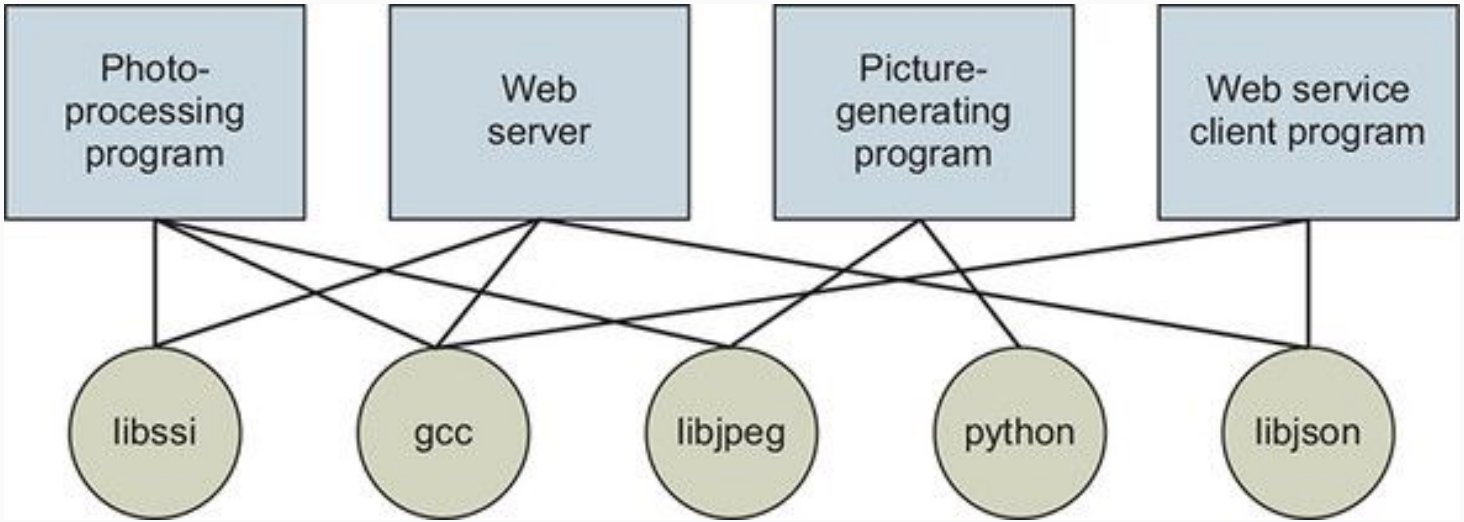
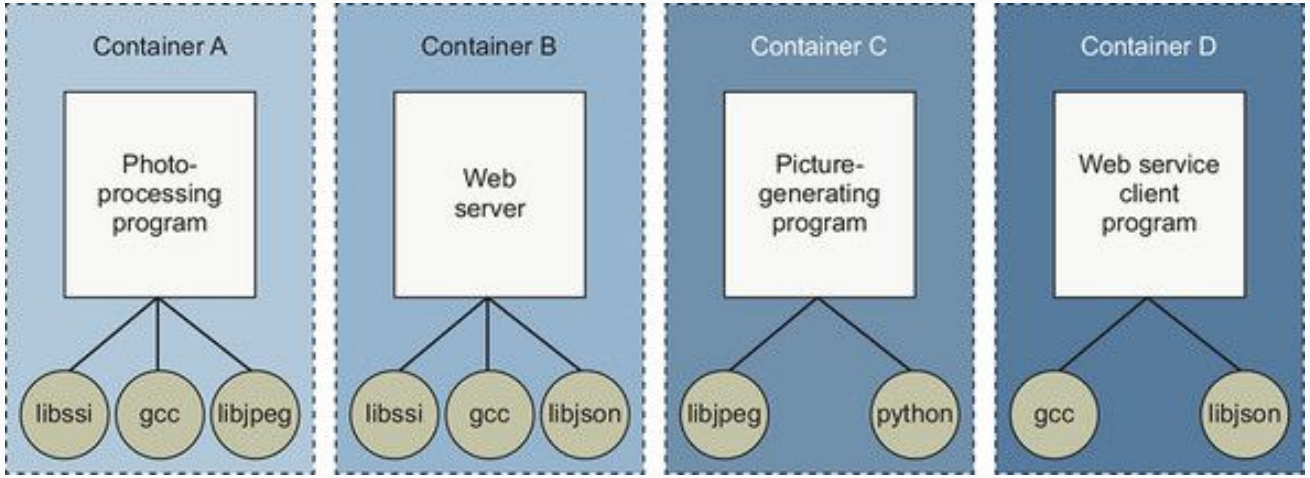
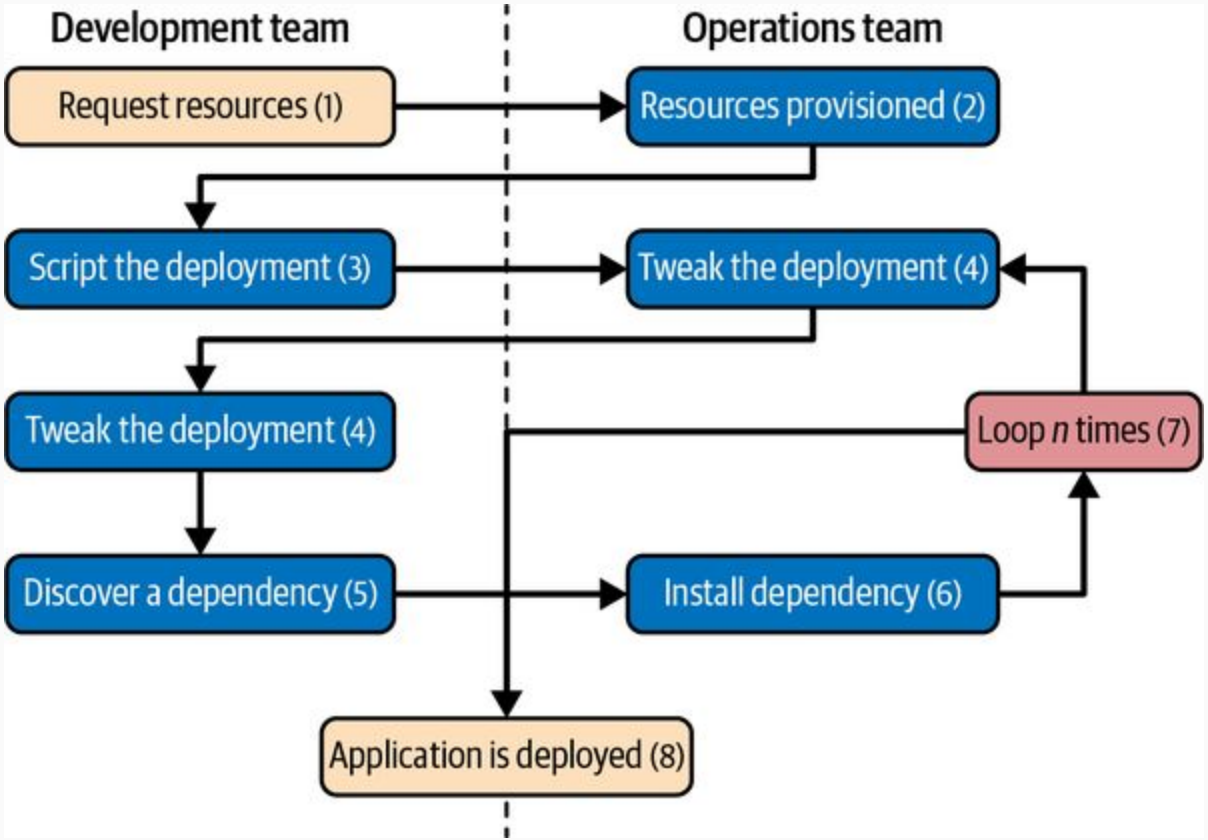
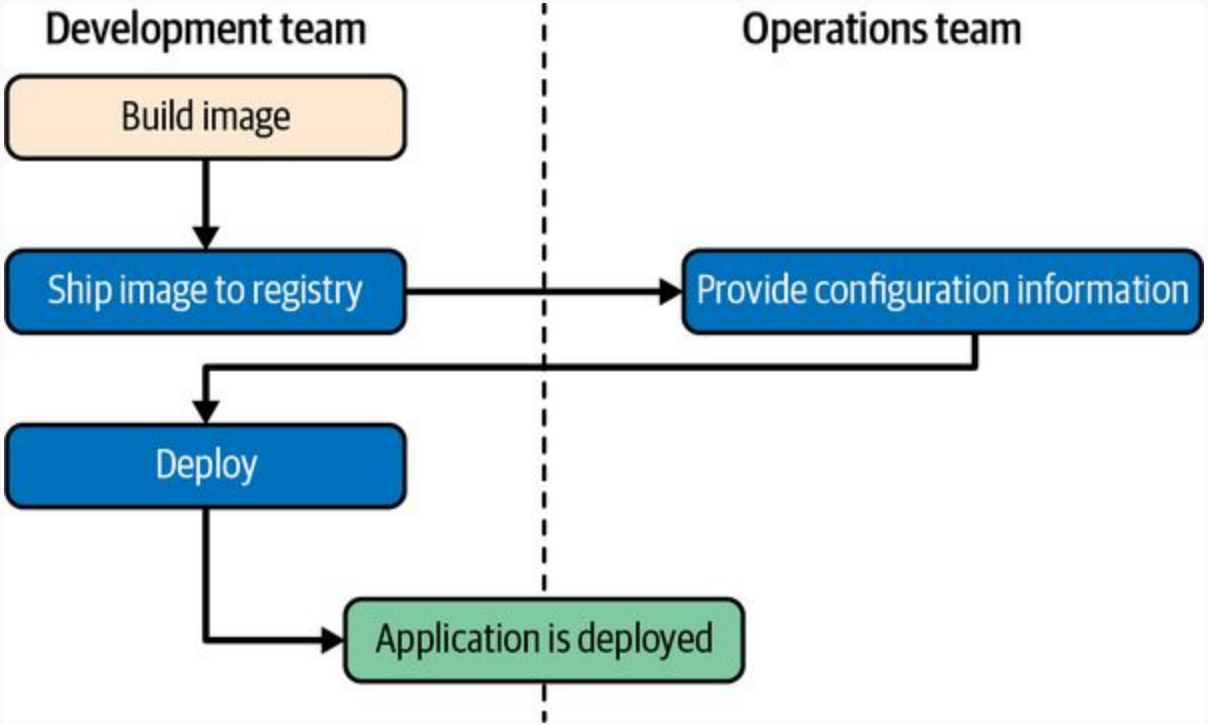


Photo-processing program
Web server
Picture-generating program
Web service client program
libssi
gcc
libjpeg
python
libjson


Container A
Photo-processing program
libssi
gcc
libjpeg
Container B
Web server
libssi
gcc
libjson
Container C
Picture-generating program
libjpeg
python
Container D
Web service client program
gcc
libjson


Development team
Operations team
Request resources (1)
Resources provisioned (2)
Script the deployment (3)
Tweak the deployment (4)
Tweak the deployment (4)
Loop n times (7)
Discover a dependency (5)
Install dependency (6)
Application is deployed (8)


Development team
Operations team
Build image
Ship image to registry
Provide configuration information
Deploy
Application is deployed


Docker
Container


Workshop
Hello Docker

Exploring Docker Machine

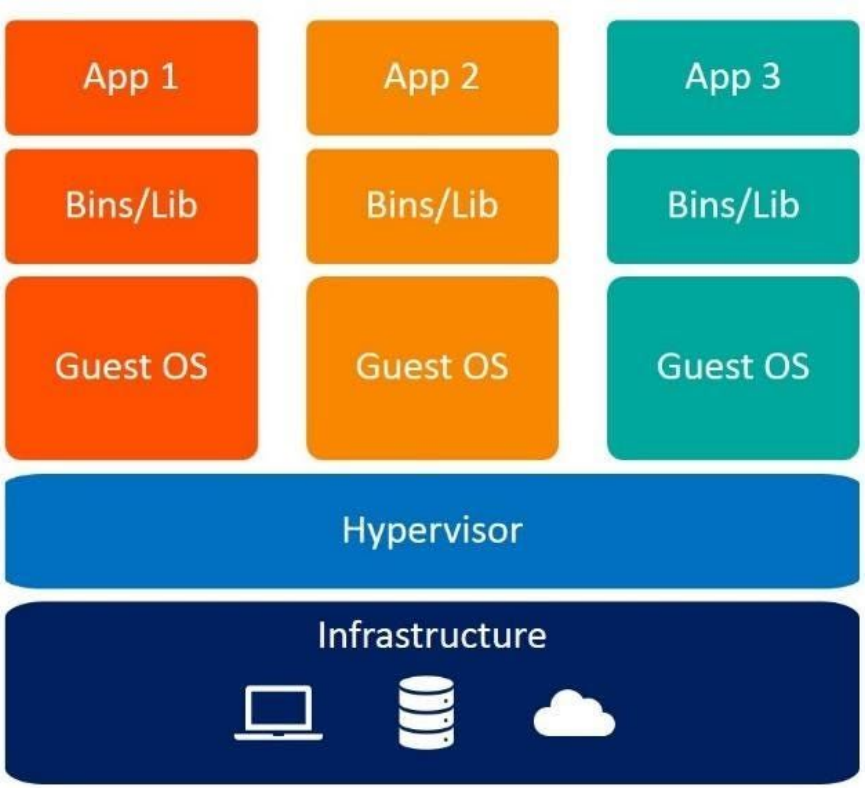
$ docker system info

$ docker context ls

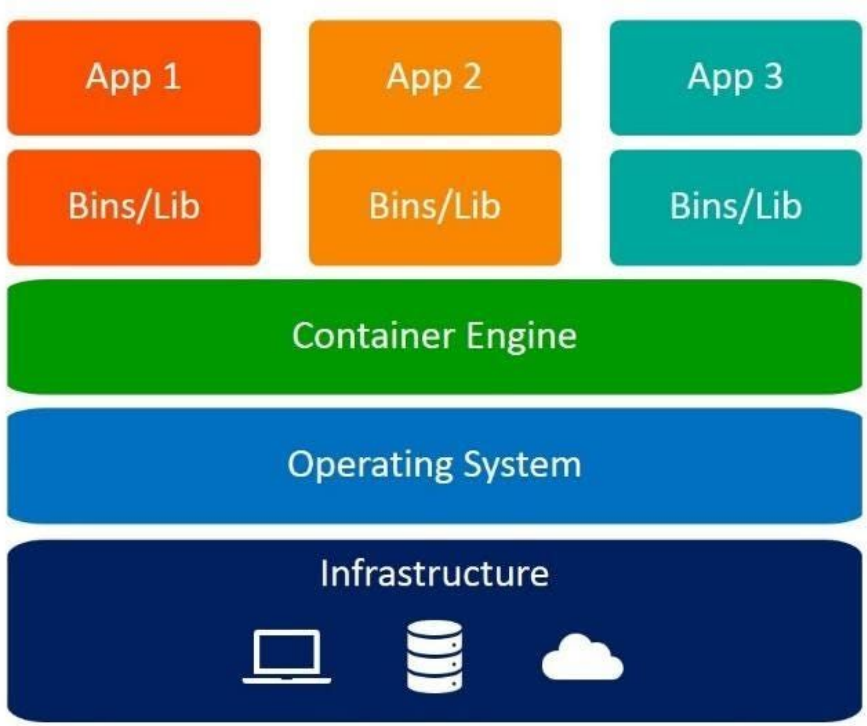
$ docker context desktop-linux

```
# Run a simple docker container

$ docker image pull hello-world

$ docker image ls

$ docker images

$ docker run - -name hello-world hello-world

$ docker container ps
```

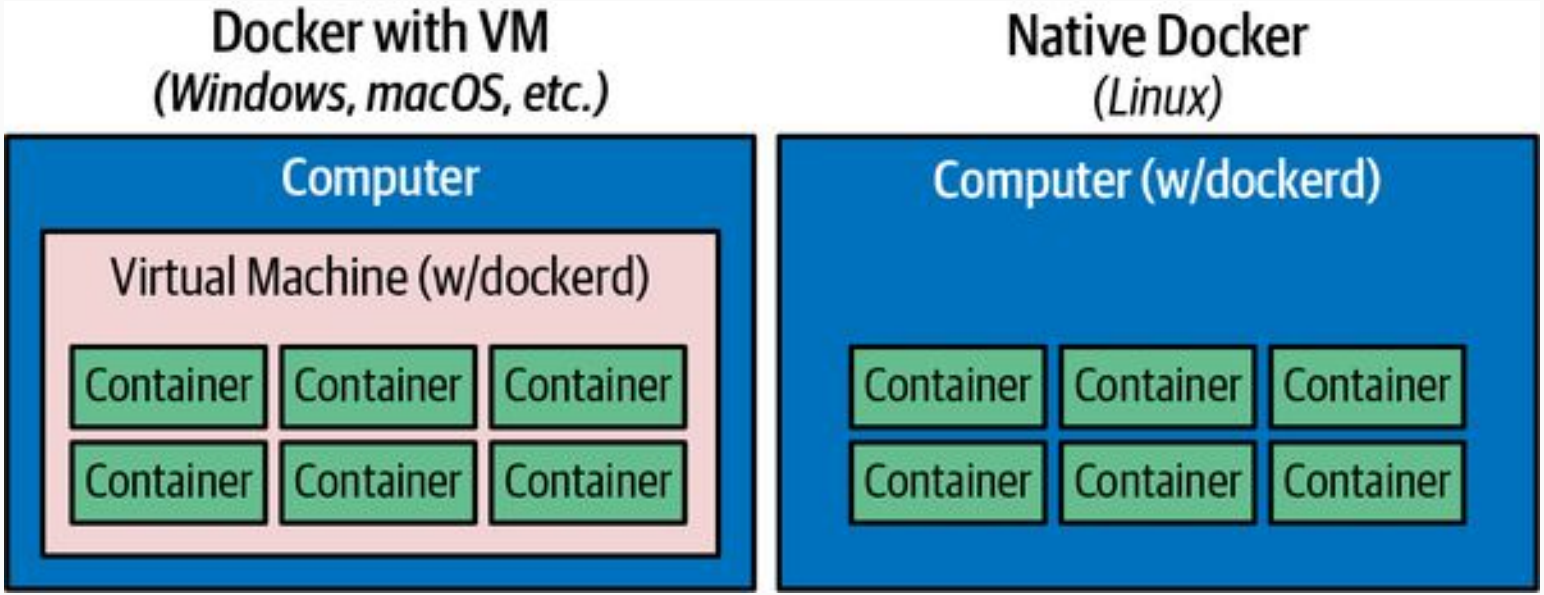

Docker Architecture
Under the Hood

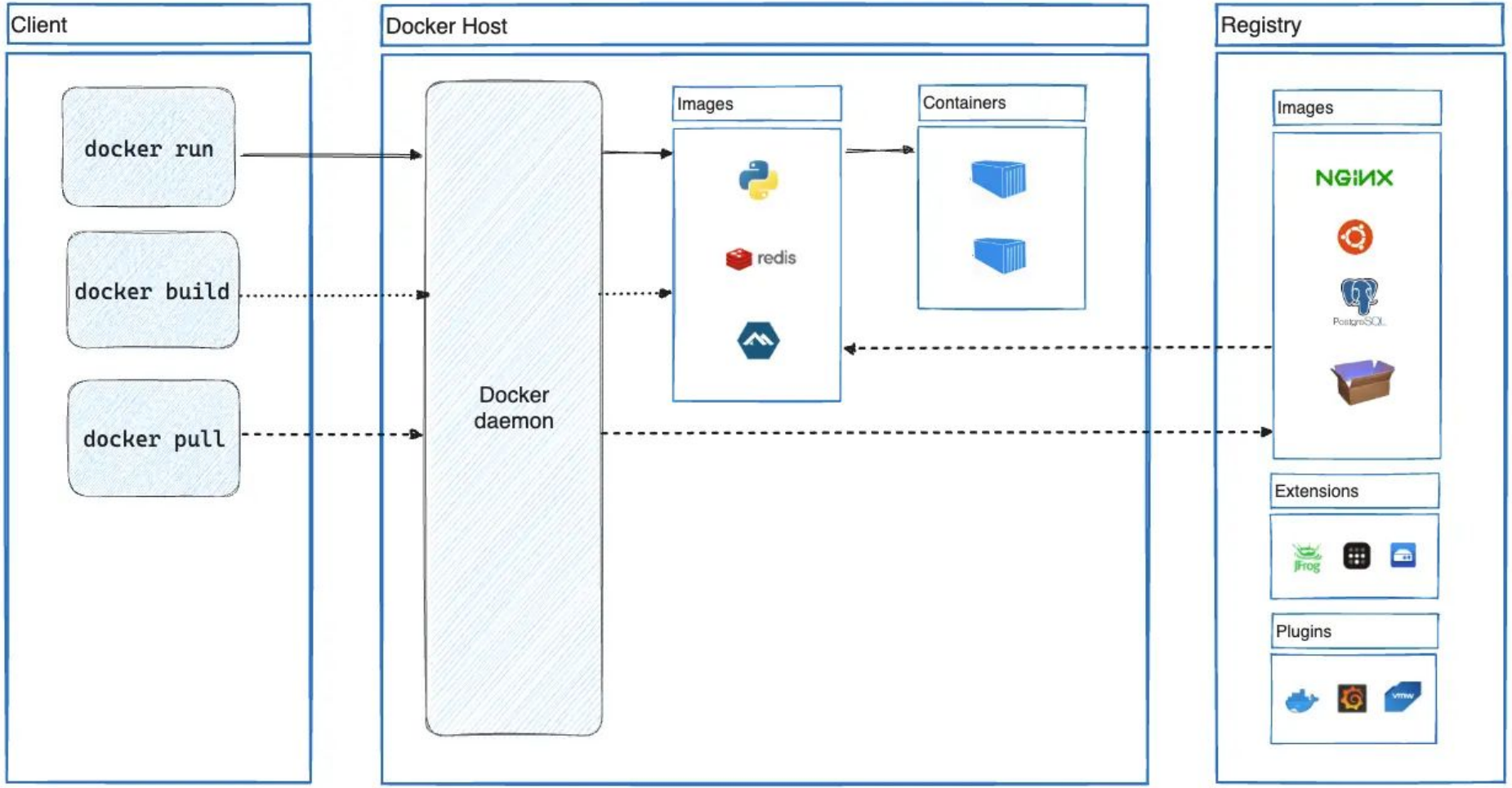


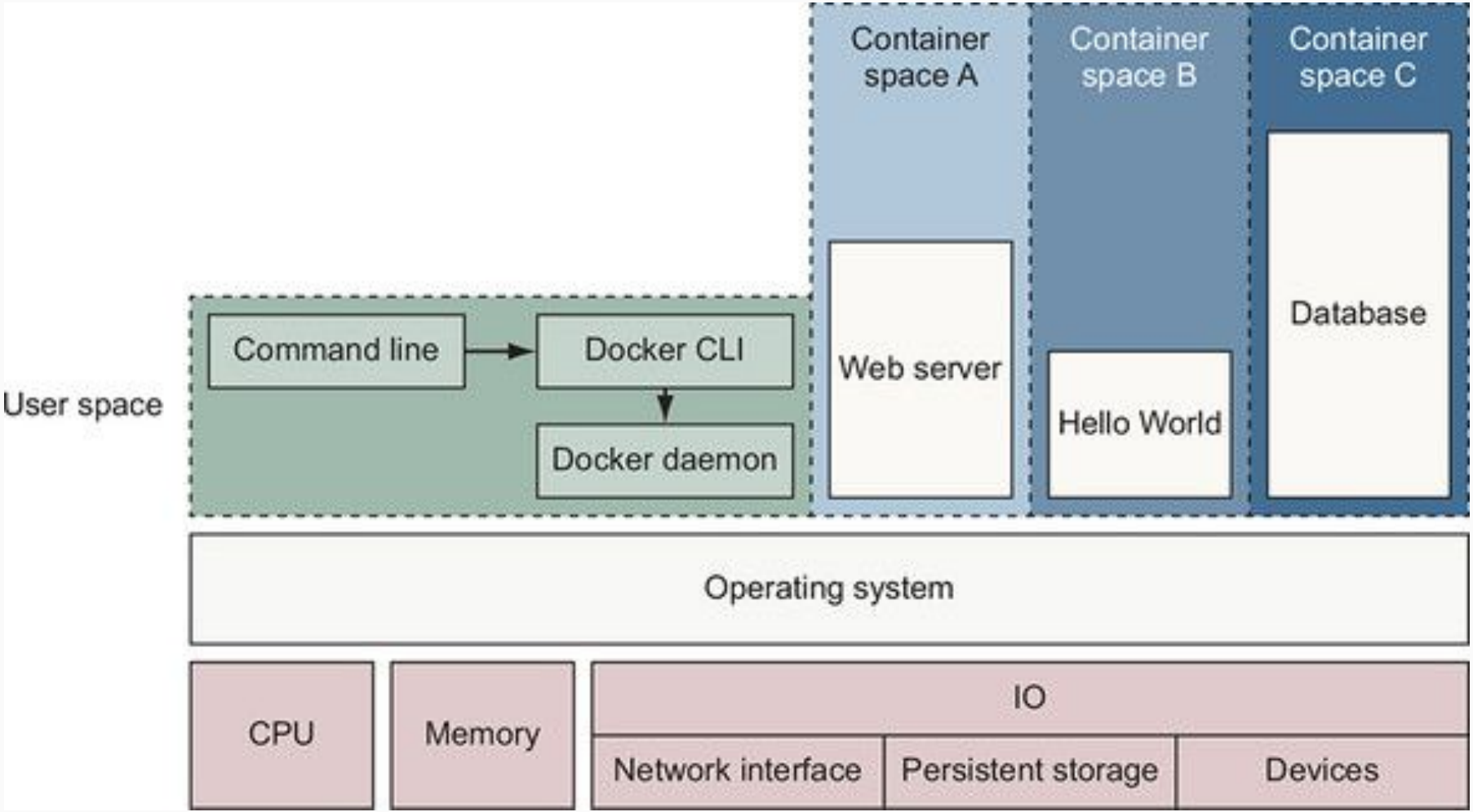
Virtual Machines

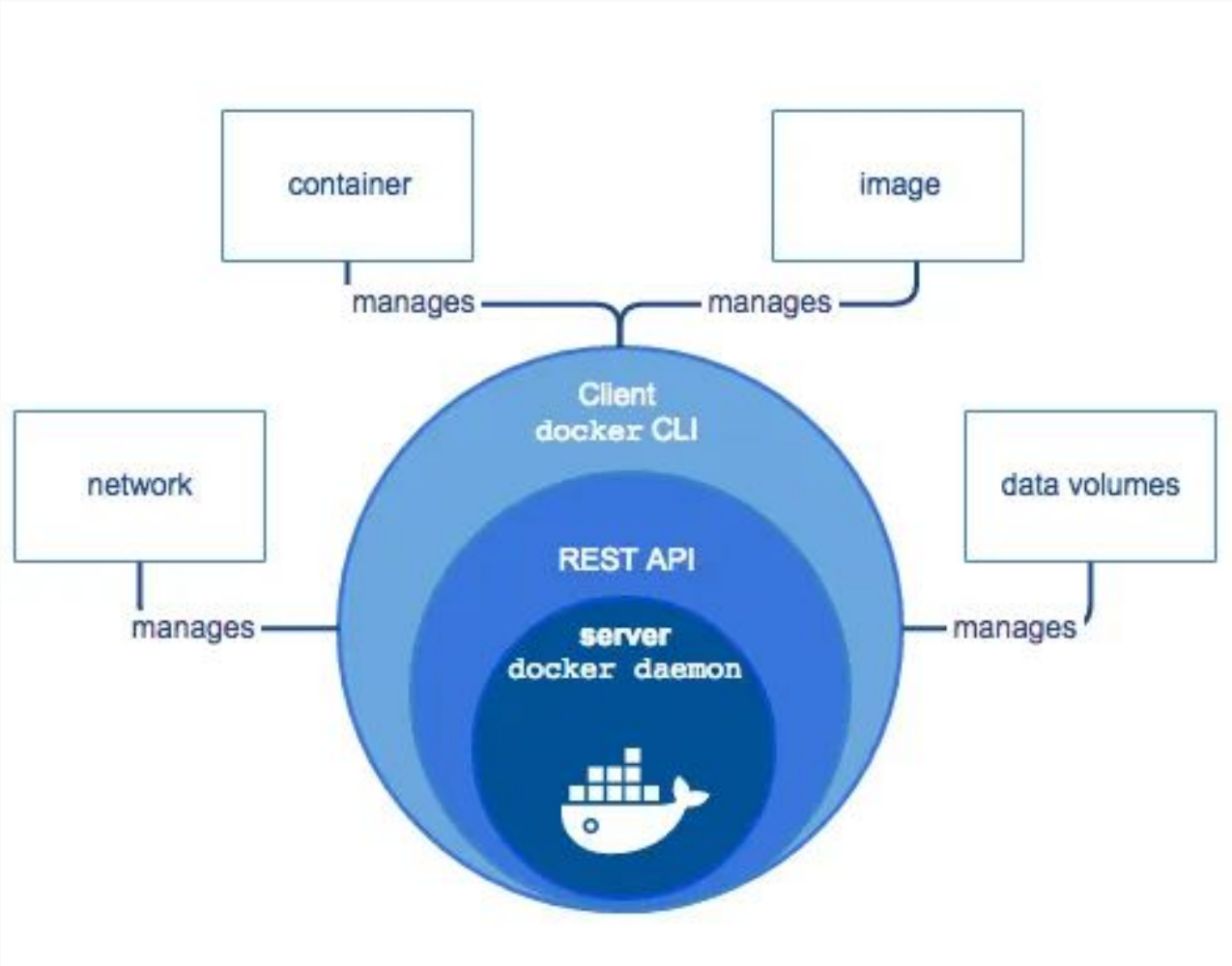


Containers


Docker with VM
(Windows, macOS, etc.)
Computer
Virtual Machine (w/dockerd)
Container
Container
Container
Container
Container
Container
Native Docker
(Linux)
Computer (w/dockerd)
Container
Container
Container
Container
Container
Container


Client
docker run
docker build
docker pull
Docker Host
Docker daemon
Images
redis
Containers
Registry
Images
NGINX
PostgreSQL
Extensions
Plugins


Container space A
Container space B
Container space C
Command line
Docker CLI
Docker daemon
Web server
Hello World
Database
User space
Operating system
CPU
Memory
IO
Network interface
Persistent storage
Devices


container
image
manages
manages
Client
docker CLI
network
data volumes
REST API
manages
server
docker daemon
manages


Docker
Image

Workshop
Create Your Own Docker Image

```
# Create an image from cli

$ docker image pull nginx

$ docker images

$ docker run --detach --name my-nginx-container nginx

$ docker container ps

$ docker container exec -it my-nginx-container bash
```

Update index.html in Container

$ docker container exec -it my-nginx-container bash

#root@d87900c0fa56: cd /usr/share/nginx/html

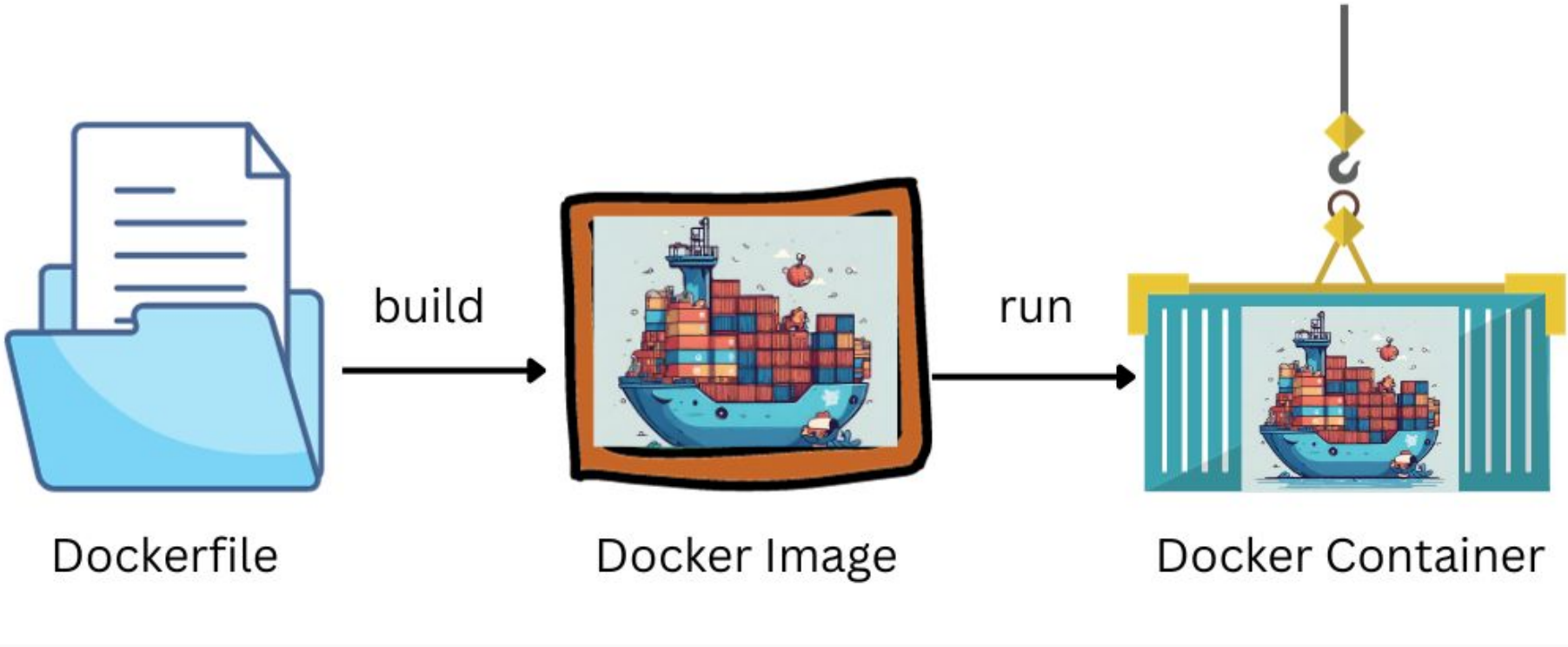
ls -l

vim index.html

apt-get update && apt-get install vim

exit

```
# Create an image from cli

$ docker container commit \
      -a jostars -m "make nginx to my welcome me" \
      my-nginx-container \
      my-nginx-image:v1

$ docker image ls

$ docker run -it --name my-nginx-container-v1 my-nginx-image:v1 bash

$ cat /usr/share/nginx/html/index.html
```


build
run
Dockerfile
Docker Image
Docker Container

Create an image from Dockerfile

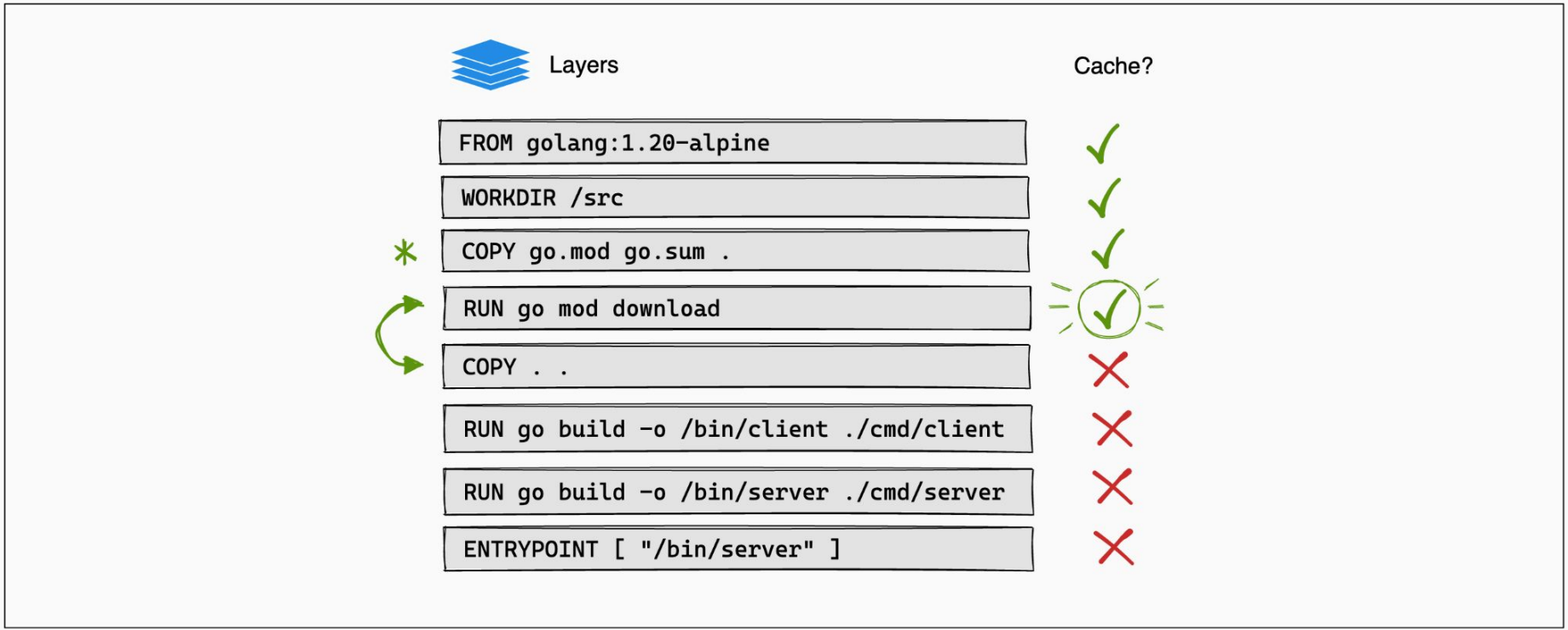
$ cat <<EOF >>index.html
<h1> HELLO, My NGINX V2 </h1>
EOF

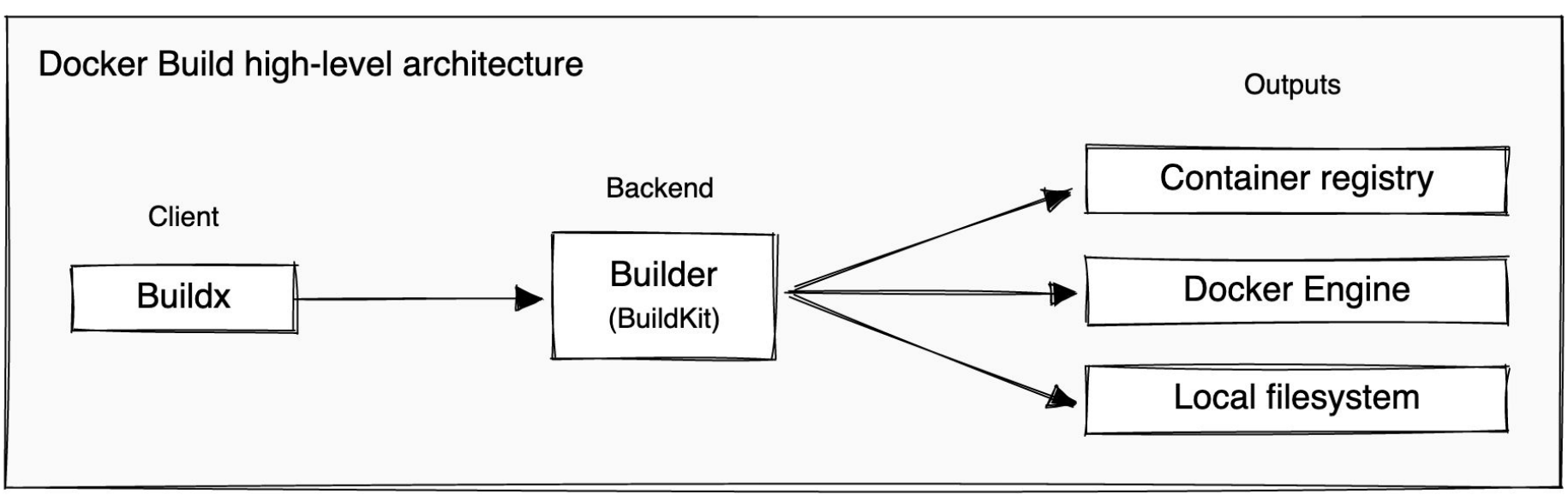
```Dockerfile
FROM nginx:latest
COPY index.html /usr/share/nginx/html/index.html
```

$ docker build -t my-nginx-image:v2 .

$ docker run -d --rm -p 8080:80 my-nginx-image:v2

$ curl http://localhost:8080


Layers
Cache?
FROM golang:1.20-alpine
WORKDIR /src
COPY go.mod go.sum .
RUN go mod download
COPY . .
RUN go build -o /bin/client ./cmd/client
RUN go build -o /bin/server ./cmd/server
ENTRYPOINT ["/bin/server"]


Docker Build high-level architecture
Outputs
Client
Backend
Buildx
Builder
(BuildKit)
Container registry
Docker Engine
Local filesystem

Structure of Docker Image Name

{Organization}/{Publisher}/{ImageName}:{Tag}

Ex. scrum123.com/kkosittaruk/nginx:1.0.0

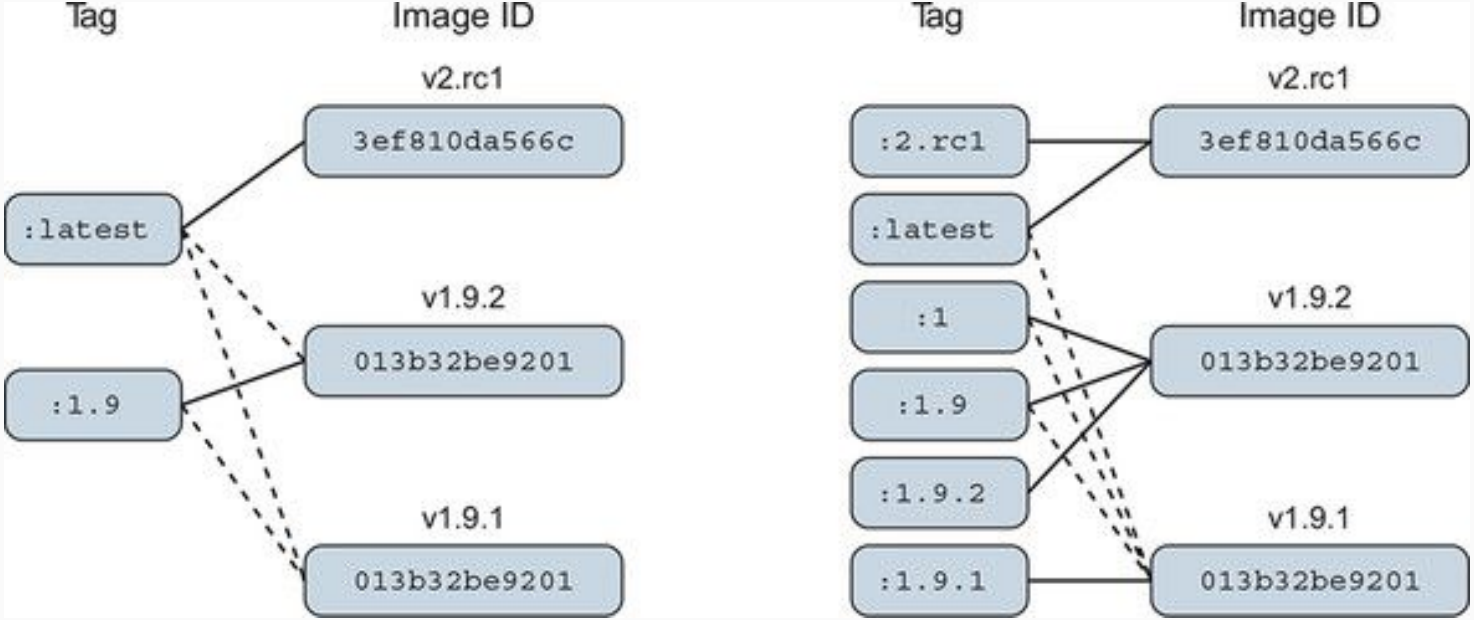
Docker Image Tag Pattern

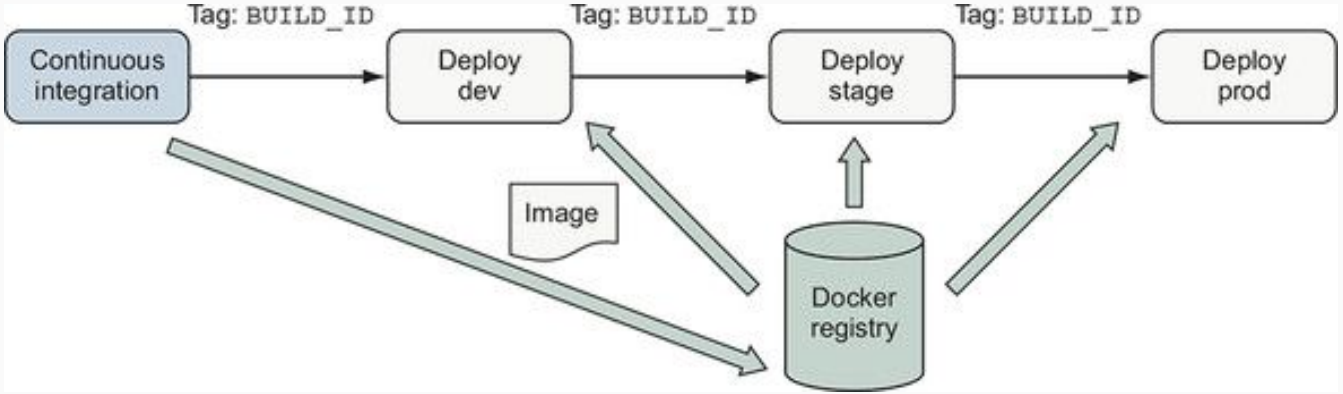
Patterns

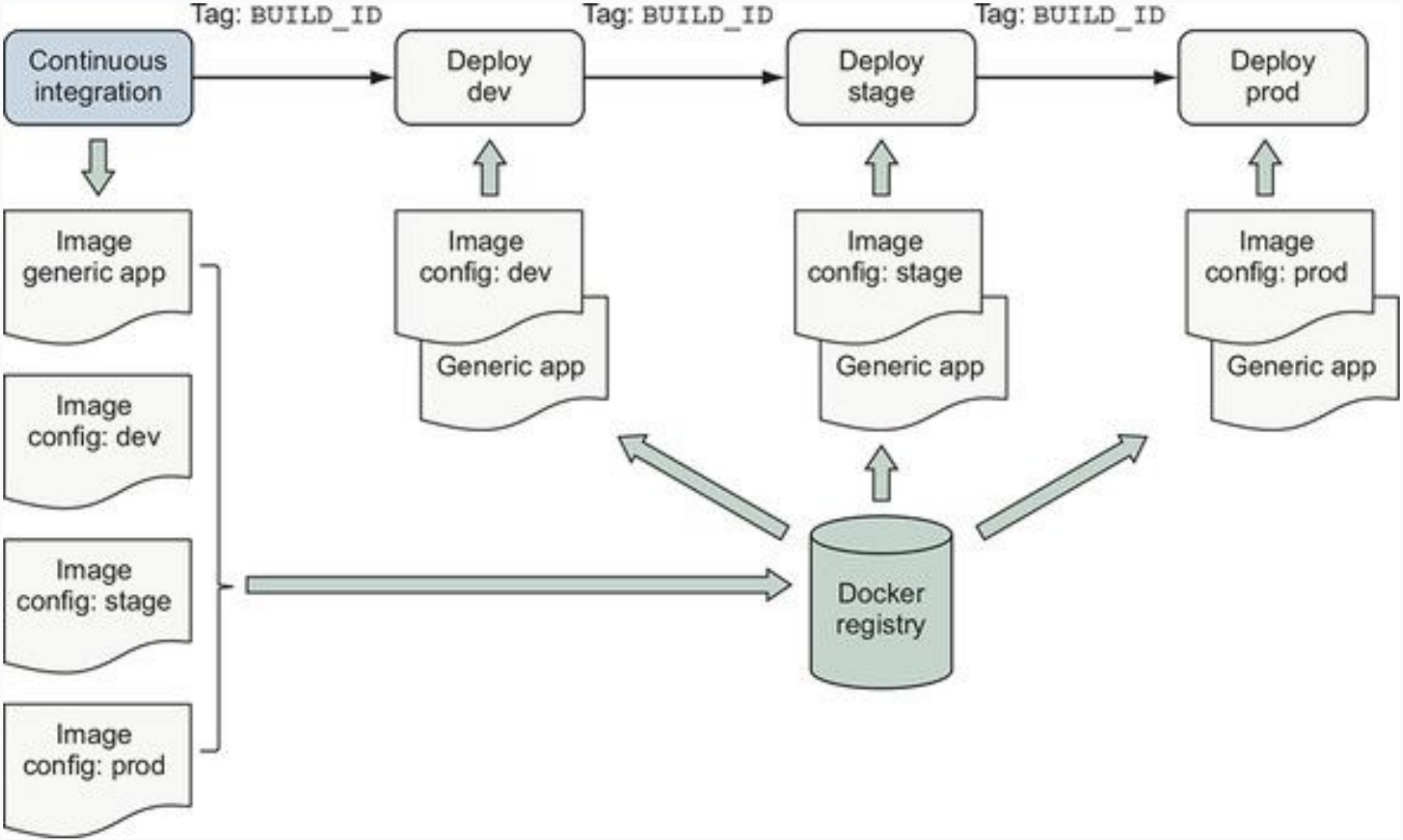
1. Semantic versioning (ex. 1.2.1)
2. Date versioning (ex. 2024.03.25)
3. Environment
   (ex. nginx:dev, nginx:uat, nginx:prod)
4. Hybrid versioning
   (ex. 1.2.1-dev, 1.2.0-uat, 1.2.0-prod)
5. Hash versioning (ex. nginx:cfe4ff9)

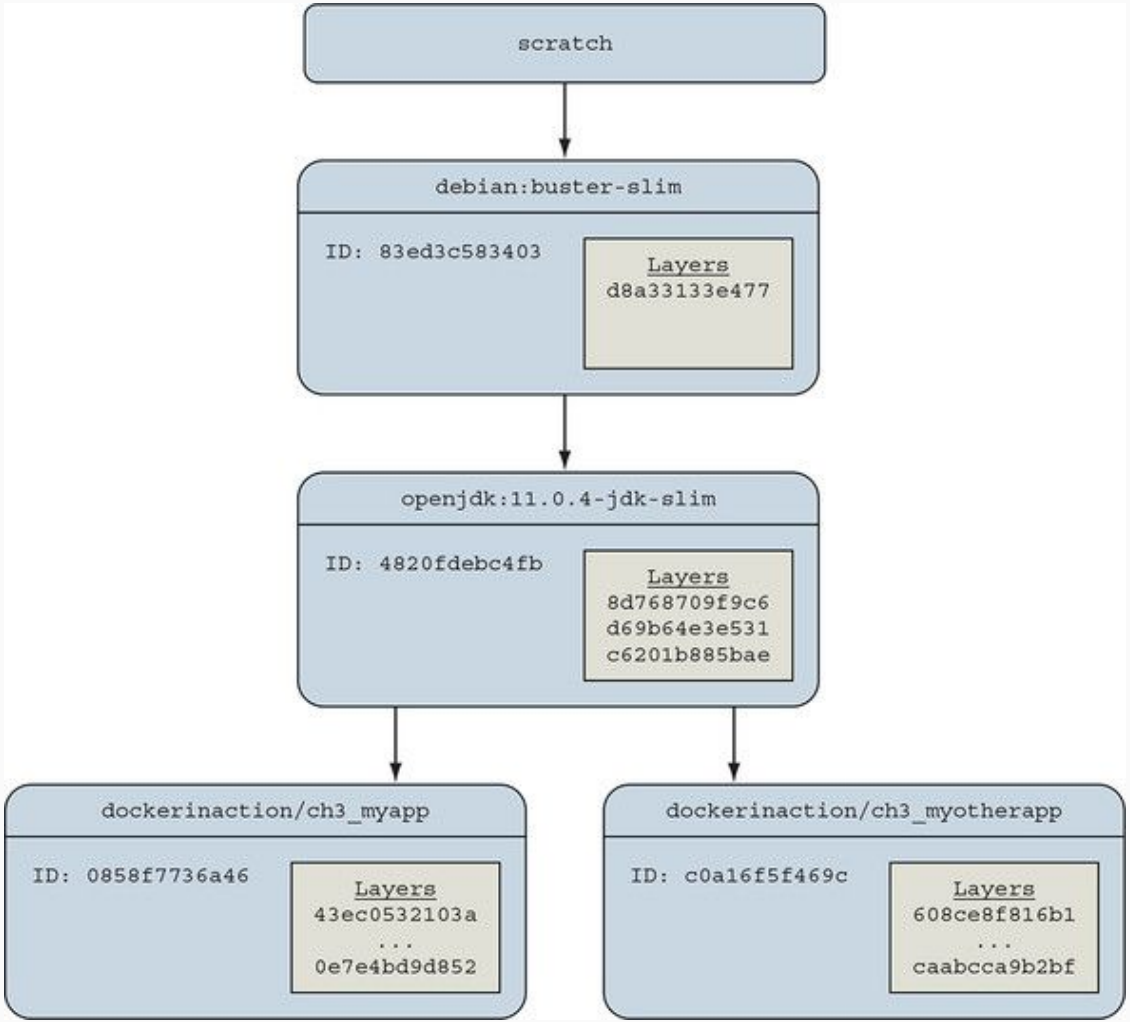
Need to know about tag

- Tags are **mutable**
- Single image can multiple tag
- **Avoid** to use tag **latest** in production and test environment


Tag
Image ID
v2.rc1
3ef810da566c
:latest
v1.9.2
013b32be9201
:1.9
v1.9.1
013b32be9201
Tag
Image ID
v2.rc1
:2.rc1
3ef810da566c
:latest
:1
v1.9.2
013b32be9201
:1.9
:1.9.2
v1.9.1
:1.9.1
013b32be9201


Tag: BUILD_ID
Tag: BUILD_ID
Tag: BUILD_ID
Continuous integration
Deploy dev
Deploy stage
Deploy prod
Image
Docker registry


Tag: BUILD_ID
Tag: BUILD_ID
Tag: BUILD_ID
Continuous integration
Deploy dev
Deploy stage
Deploy prod
Image generic app
Image config: dev
Image config: stage
Image config: prod
Image config: dev
Generic app
Image config: stage
Generic app
Image config: prod
Generic app
Docker registry


scratch
debian:buster-slim
ID: 83ed3c583403
Layers
d8a33133e477
openjdk:11.0.4-jdk-slim
ID: 4820fdebc4fb
Layers
8d768709f9c6
d69b64e3e531
c6201b885bae
dockerinaction/ch3_myapp
ID: 0858f7736a46
Layers
43ec0532103a
...
0e7e4bd9d852
dockerinaction/ch3_myotherapp
ID: c0a16f5f469c
Layers
608ce8f816b1
...
caabcca9b2bf


Docker
Container

Workshop

Deep Dive Docker Container

Exploring in container

$ docker container inspect

$ docker container ps

List Container

$ docker container ls

Start Stop Container

$ docker container start

$ docker container stop

Remove Container

$ docker container rm

Container Logs

$ docker container logs

Exec Command to Container

$ docker container exec

Remove all container

$ docker container rm $(docker container ls -aq)

Workshop

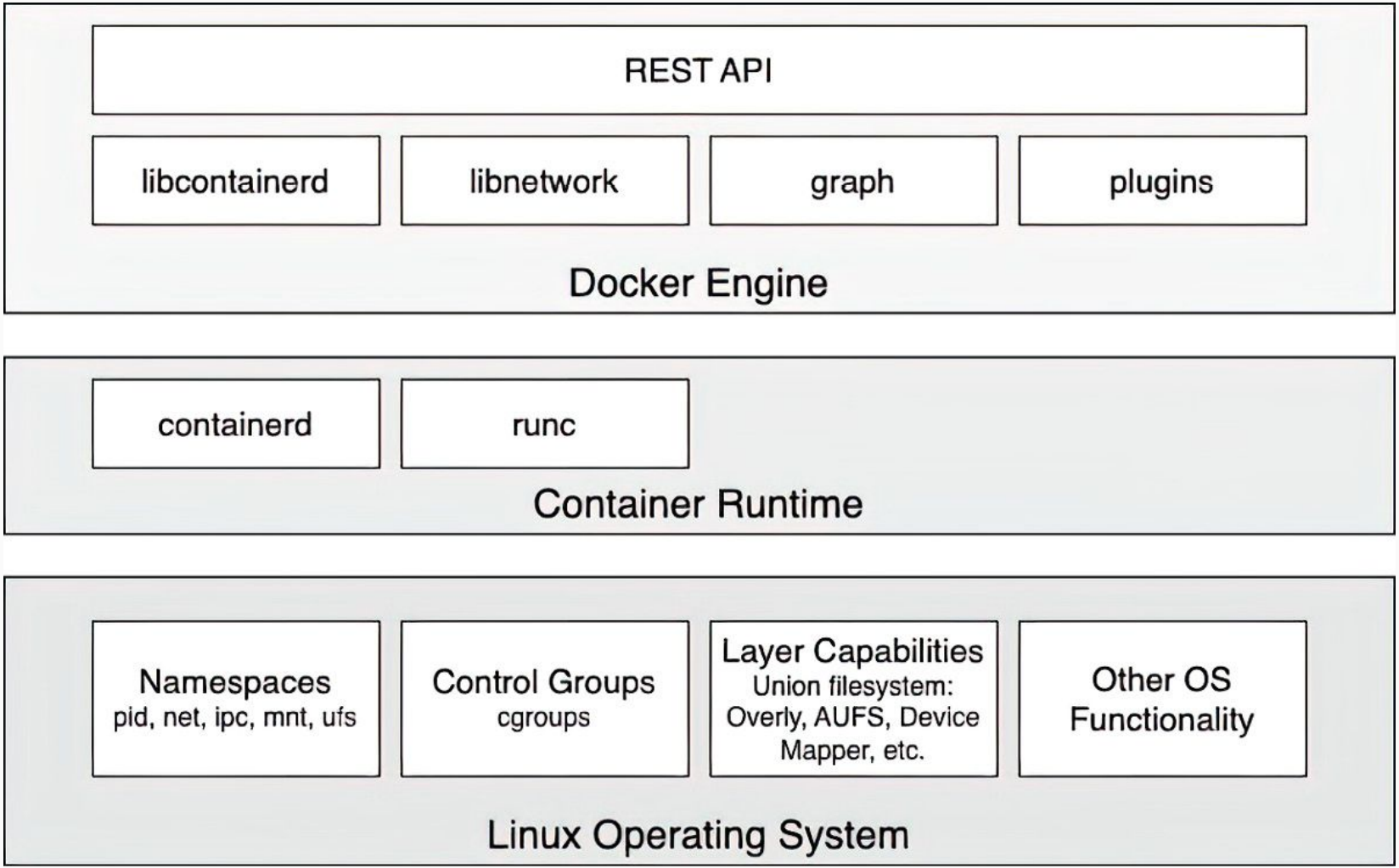
Configure Container with Environment

```
# Set Environment via Command line

$ docker image pull alpine

$ docker image inspect alpine

$ docker run --rm alpine /bin/sh

#root: printenv

$ docker run -it --rm -e HELLO=world alpine /bin/sh
```

Set Environment via Dockerfile

```
$ cat <<EOF >>Dockerfile.env
FROM alpine
ENV HELLO=world
EOF

$ docker build -t alpine-env:v1 -f Dockerfile.env .

$ docker run --rm -it alpine-env:v1 /bin/sh

$ docker image inspect -f "{{json .Config.Env}}" alpine-env:v1 | jq .
```

```
# Set Environment via .env file

$ cat <<EOF >>.env
HELLO=again?
SECRET=Don’t tell my wife
EOF

$ docker run --rm -it -env-file .env alpine-env:v1 printenv
```


REST API
libcontainerd
libnetwork
graph
plugins
Docker Engine
containerd
runc
Container Runtime
Namespaces
pid, net, ipc, mnt, ufs
Control Groups
cgroups
Layer Capabilities
Union filesystem:
Overly, AUFS, Device
Mapper, etc.
Other OS
Functionality
Linux Operating System

runc

*runc is a lightweight, portable **container runtime**. It provides full support for Linux namespaces as well as native support for all security features available on Linux, such as SELinux, AppArmor, seccomp, and cgroups.*

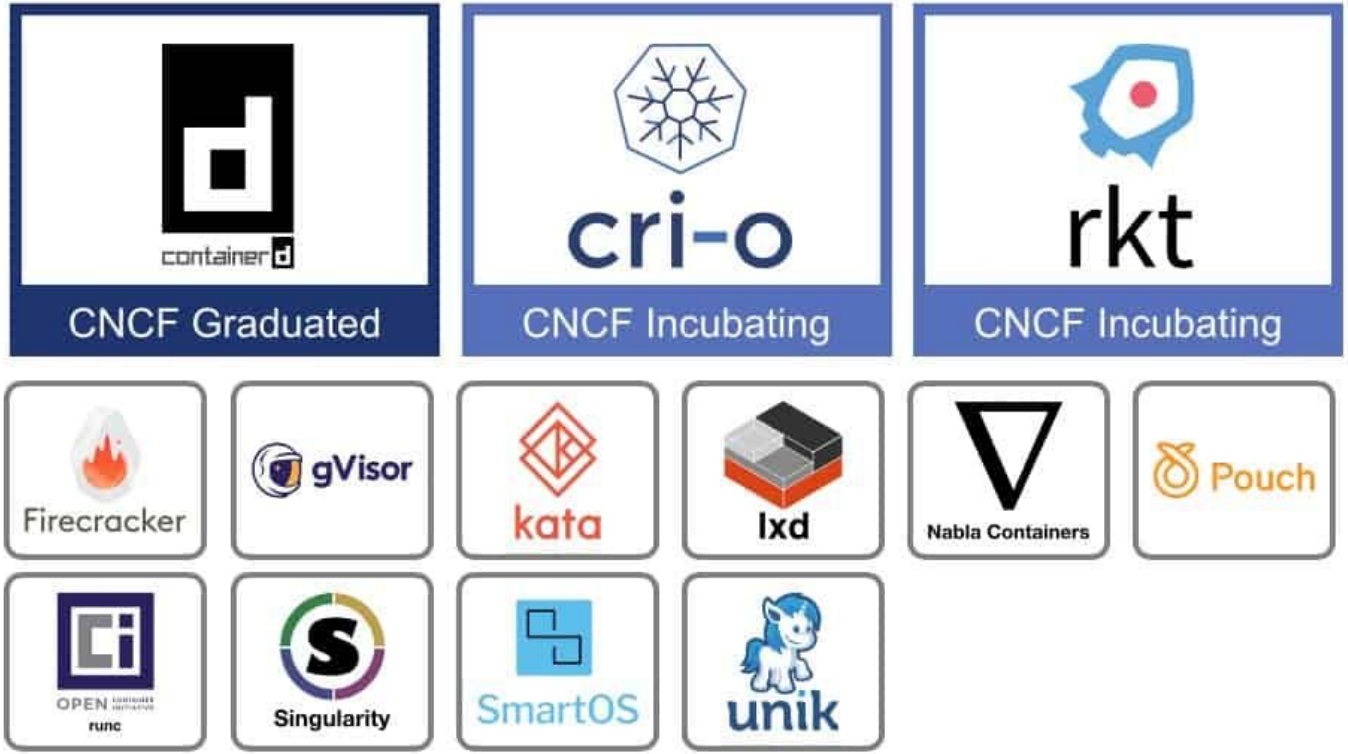
containerd

*containerd builds on top of runc and adds higher-level features, such as image transfer and storage, container execution, and supervision as well as network and storage attachments. With this, **it manages the complete life cycle of containers.** containerd is the reference implementation of the OCI specifications and is by far the most popular and widely used container runtime.*

Check your container runtime

$ docker system info

Alternative Container Runtime




Docker
Volume


Workshop
with Volume

```
# Data in container

$ docker container run --name demo \
      alpine /bin/sh -c 'echo "This is a test" > sample.txt'

# Check container change with `diff`

$ docker container diff demo
```

```
# Volume for persistent your data

$ docker volume create sample

$ docker volume inspect sample

# Mount volume to container

$ docker container run --name test -it \
    -v sample:/data \
    alpine /bin/sh

# cd data && touch data1.txt && touch data2.txt
# exit

$ docker container rm test
```

```
# Create new container with different OS

$ docker container run --name test2 -it --rm \
    -v sample:/app/data \
    centos:7 /bin/bash

# ls -l /app/data/
```

Where is the data stored? Let’s Investigate

$ docker volume ls

$ docker volume inspect sample

$ docker volume inspect -f "{{json .Mountpoint}}" sample

$ cd /var/lib/docker/volumes/sample/_data

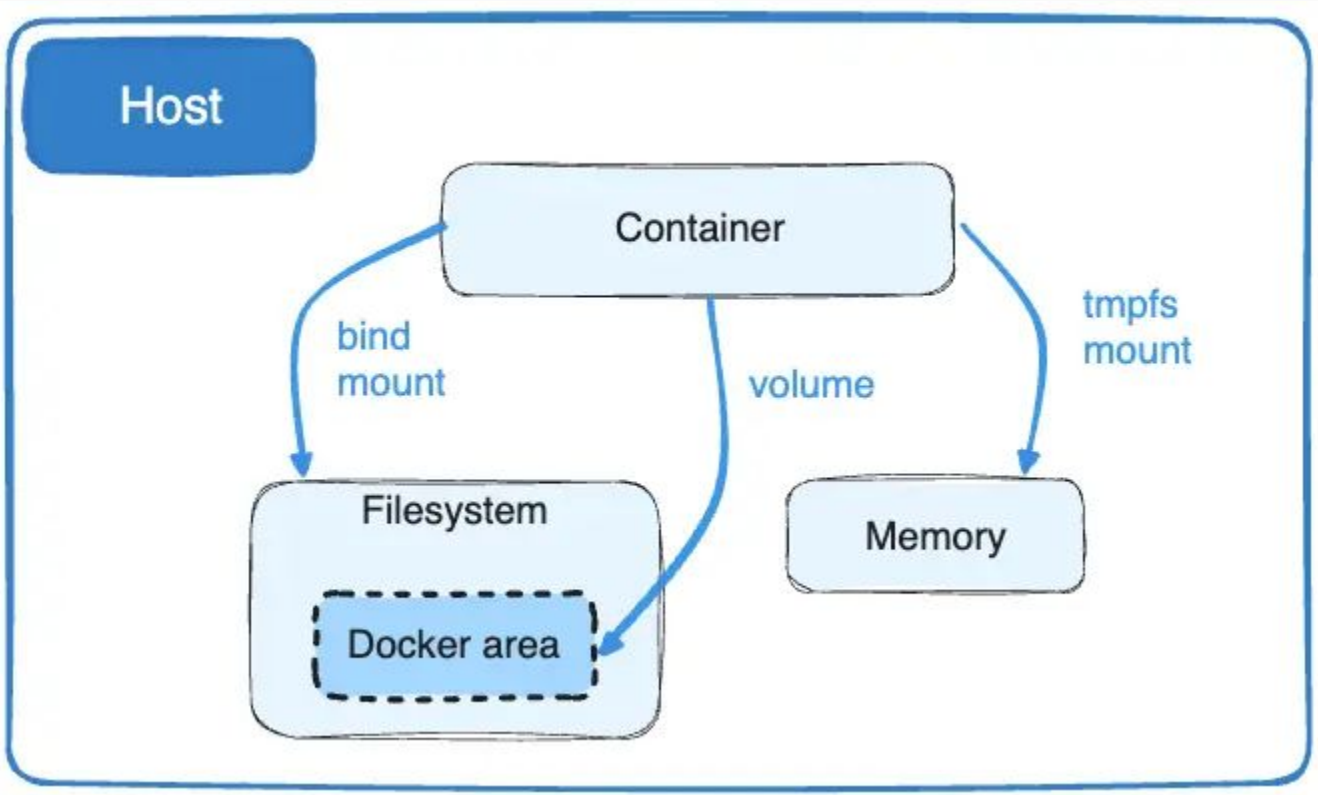
no such file or directory: /var/lib/docker/volumes/sample/_data

 # WHY!!!

Answer is :

Docker does not run natively on Mac directly; *instead, it runs inside a slim VM. The data for the VM that Docker creates can be found in this directory.*

~/Library/Containers/com.docker.docker/Data/vms/0 folder.


Host
Container
bind
mount
volume
tmpfs
mount
Filesystem
Docker area
Memory

Fantastic volume and where to find them”

```
$ docker container run -it --privileged --pid=host \
    debian nsenter -t 1 -m -u -n -i sh

#  ls -l /var/lib/docker/volumes
#  ls -l /var/lib/docker/volumes/sample/_data
```

Let’s add more data from another container

```
$ docker container run --rm -it -v sample:/data alpine /bin/sh

# echo "Hello world" > /data/sample.txt
# echo "Hello Other" > /data/other.txt
```

!!! Warning !!! Share Data Volume in Multiple Container

Workshop

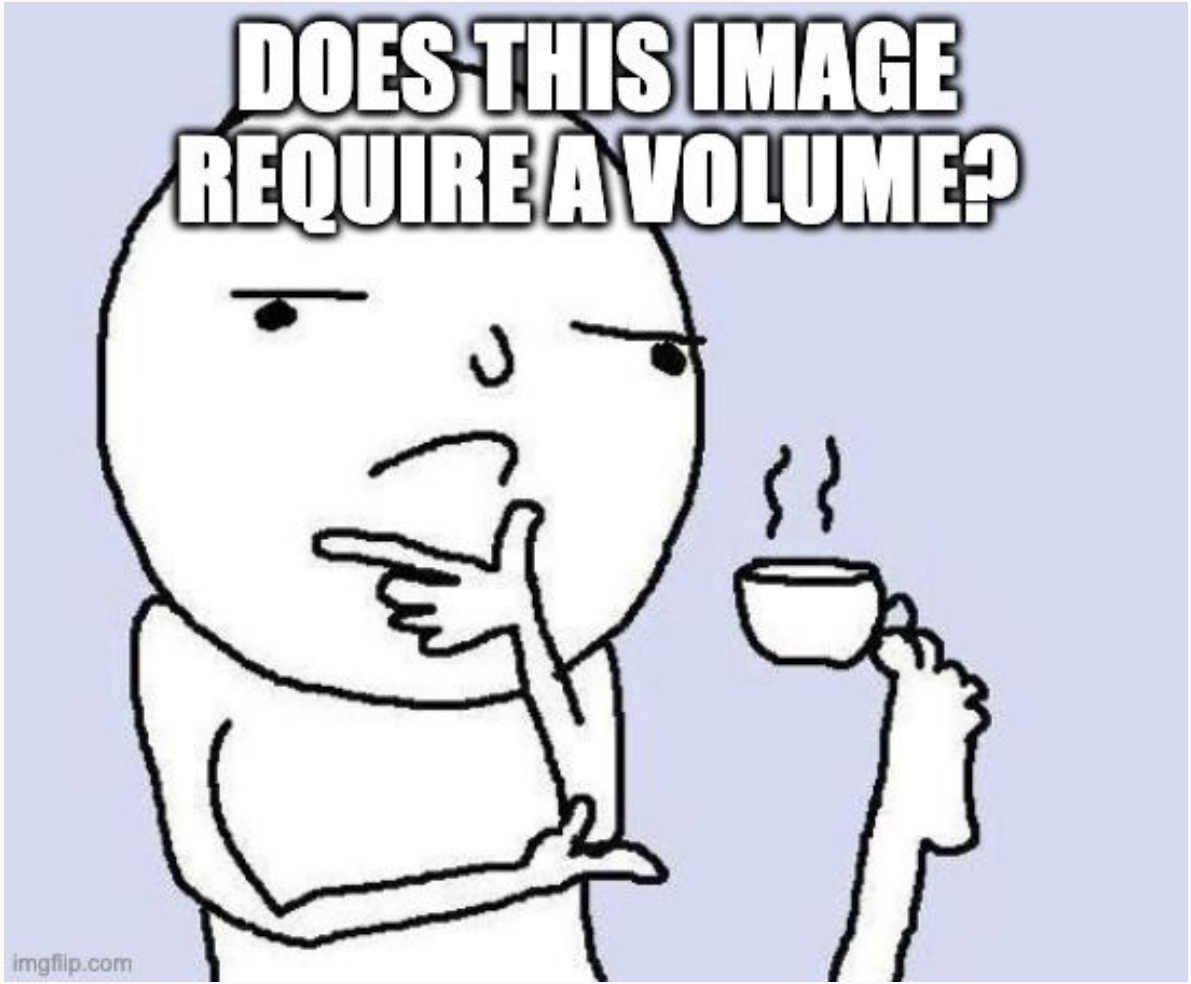
Using Host Volume

```
# I want to host dynamic content

$ echo "<h1>HELLO, Docker</h1>" > index.html

$ cat <<EOF >>Dockerfile-volume
FROM nginx:latest
COPY index.html /usr/share/nginx/html/index.html
EOF

$ docker build -f Dockerfile-volume -t nginx-volume:v1 .

$ docker run -d --rm -p 8080:80 nginx-volume:v1

$ curl http://localhost:8080

$ echo -e "<h2> new awesome title </h2>" >> index.html
```

```
# Add new content to website

$ echo -e "<h2> new awesome title </h2>" >> index.html

# Let’s get our content

$ curl http://localhost:8080
```

Mount Host volume mount to container

```
$ docker run -d --rm -p 8080:80 \
      -v $(pwd)/index.html:/usr/share/nginx/html/index.html \
      nginx-volume:v1

$ curl http://localhost:8080

$ echo -e "<h3>I was made for lovin’ you baby</h3>" >> index.html

$ curl http://localhost:8080
```


DOES THIS IMAGE
REQUIRE A VOLUME?
imgflip.com

```
# Let's define a volume for our image.

$ cat <<EOF >>Dockerfile-define-volume
FROM nginx
VOLUME /usr/share/nginx/html/index.html
EOF

$ docker build -f Dockerfile-define-volume \
        -t nginx-volume:v2 .

$ docker image inspect \
        -f "{{json .Config.Volumes}}"  \
        nginx-volume:v2 | jq .
```

Real Life Example

```
$ docker image pull mongo

$ docker image inspect -f "{{json .Config.Volumes}}"  mongo | jq .

$ docker run --rm --name my-mongo -d mongo

$ docker container inspect -f "{{json .Mounts}}"  my-mongo | jq .
```


Docker
Network


Workshop
Connect Container

Explore Network in Docker

$ docker network ls

$ docker network inspect {network id}

Network	Company	Scope	Description
Bridge	Docker	Local	Simple network based on Linux bridges to allow networking on a single host
Macvlan	Docker	Local	Configures multiple layer-2 (that is, MAC) addresses on a single physical host interface
Overlay	Docker	Global	Multi-node capable container network based on **Virtual Extensible LAN (VXLan)**
Weave Net	Weaveworks	Global	Simple, resilient, multi-host Docker networking
Contiv Network Plugin	Cisco	Global	Open source container networking

Explore Network in Docker

```
$ docker container run --name c1 -d --rm alpine:latest ping 127.0.0.1

$ docker container run --name c2 -d --rm alpine:latest ping 127.0.0.1

$ docker container exec -it c1 /bin/sh

# ping c2
```

Connect Container with Bride Network

$ docker network create --driver bridge sample-net

$ docker network inspect sample-net

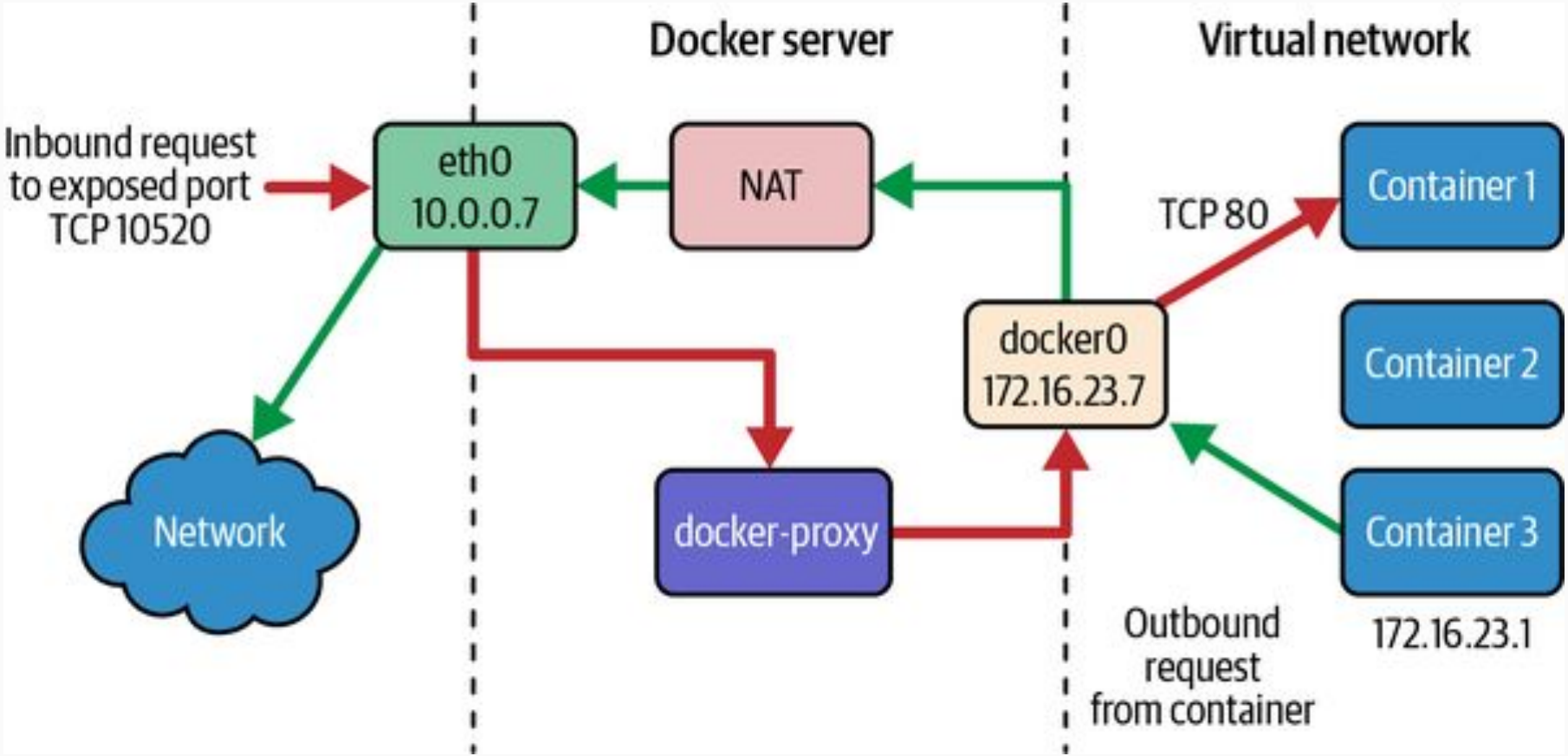
$ docker network connect sample-net c1

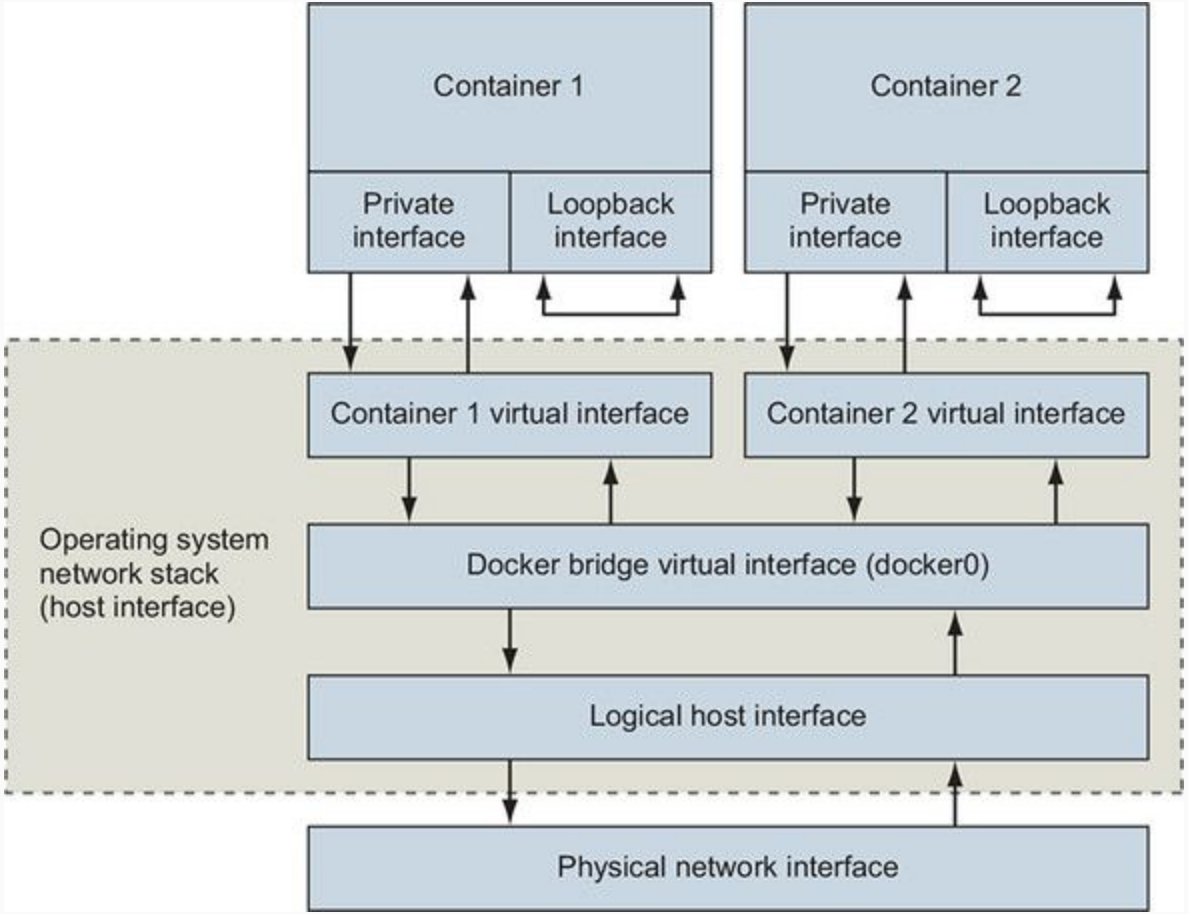
$ docker network connect sample-net c2

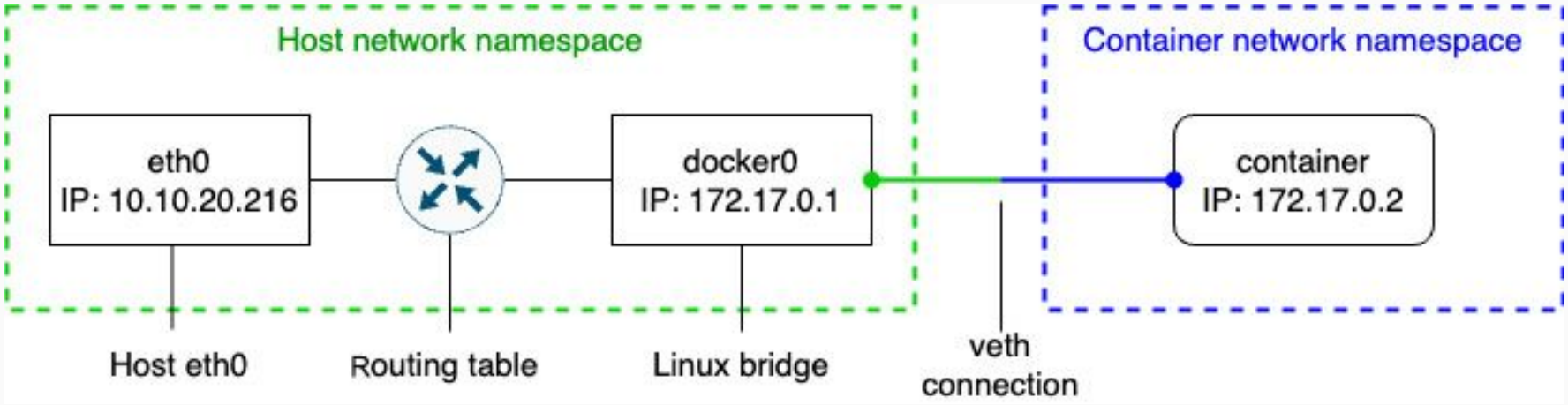
$ docker container exec -it c1 sh

ping c2

$ docker container inspect c1


Docker server
Virtual network
Inbound request
to exposed port
TCP 10520
eth0
10.0.0.7
NAT
TCP 80
Container 1
docker0
172.16.23.7
Container 2
Network
docker-proxy
Container 3
Outbound
request
from container
172.16.23.1


Container 1
Container 2
Private interface
Loopback interface
Private interface
Loopback interface
Container 1 virtual interface
Container 2 virtual interface
Operating system network stack (host interface)
Docker bridge virtual interface (docker0)
Logical host interface
Physical network interface


Host network namespace
Container network namespace
eth0
IP: 10.10.20.216
docker0
IP: 172.17.0.1
container
IP: 172.17.0.2
Host eth0
Routing table
Linux bridge
veth
connection

Orchestration Your Container


Workshop
Deploy Postgres Cluster

Create PostgresQL Cluster in Docker

```
$ docker run -d --rm --name postgres \
    -e POSTGRES_USER=dockeruser \
    -e POSTGRES_PASSWORD=dockerpass \
    -e POSTGRES_DB=pets \
    -v pg-data:/var/lib/postgresql/data \
    -v $(pwd)/db:/docker-entrypoint-initdb.d \
    -p 5432:5432 \
    postgres:alpine
```

Create PostgresQL Cluster in Docker

```
$ $ docker volume create admin-data

$ docker run -d --rm --name pgadmin \
    -e PGADMIN_DEFAULT_EMAIL=admin@acme.com \
    -e PGADMIN_DEFAULT_PASSWORD=admin \
    -v admin-data:/var/lib/pgadmin \
    -p 5050:80 \
    dpage/pgadmin4
```

Create PostgresQL Cluster in Docker

Trying to Login PGAdmin

Localhost:5050

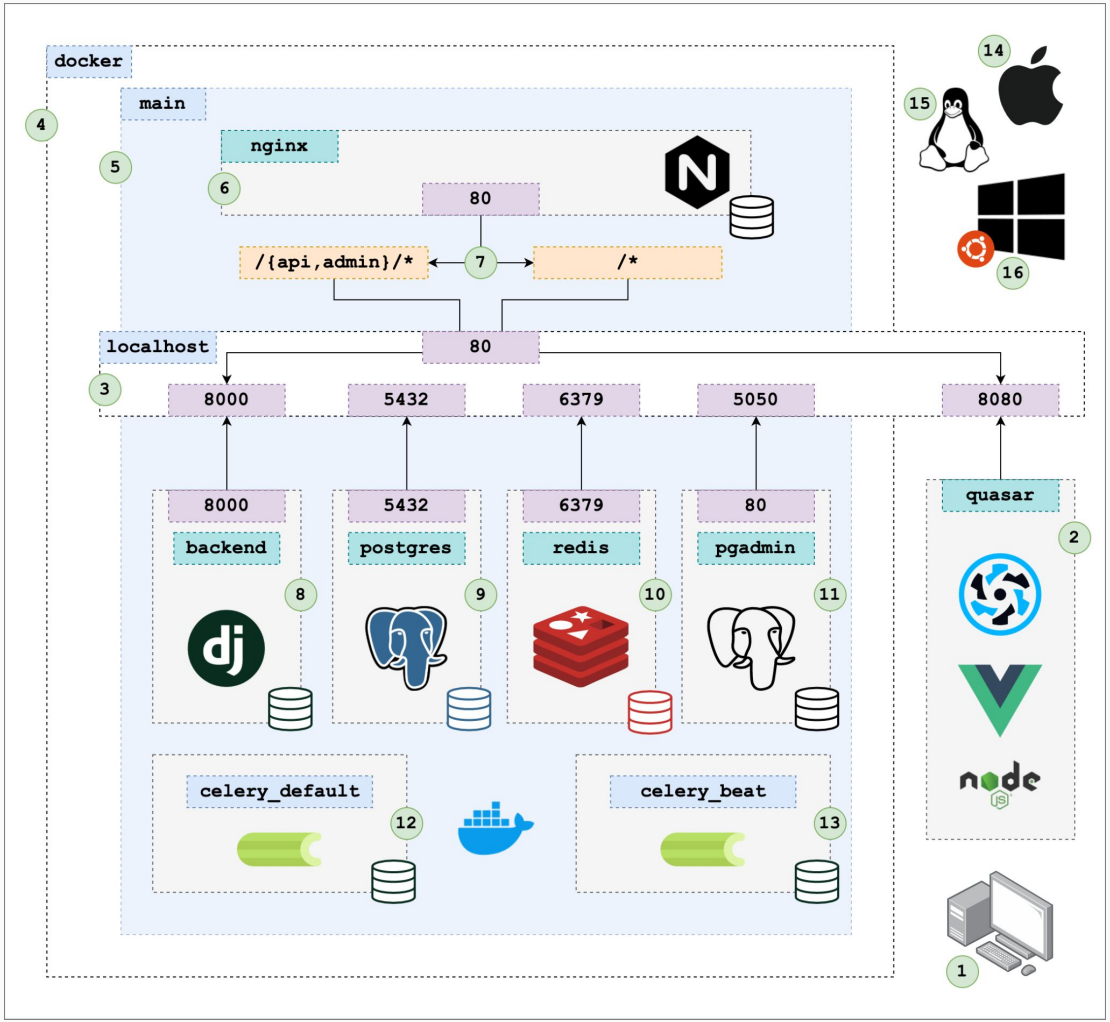
$ docker network db-net

$ docker network connect db-net postgres

$ docker network connect db-net pgadmin

Create PostgresQL Cluster in Docker

```
$ touch docker-compose.yaml

$ docker compose up

$ docker compose up -d

$ docker compose ls

$ docker container ps
```


docker
main
4
nginx
5
6
80
/{api,admin}/*
7
/*
localhost
80
3
8000
5432
6379
5050
8080
8000
backend
8
5432
postgres
9
6379
redis
10
80
pgadmin
11
celery_default
12
celery_beat
13
quasar
2
14
15
16
1


Workshop
Using Docker Compose
for Development

Setup Development Cluster

$ touch docker-compose.yaml

$ docker compose up

$ docker compose up -d

$ docker compose ls

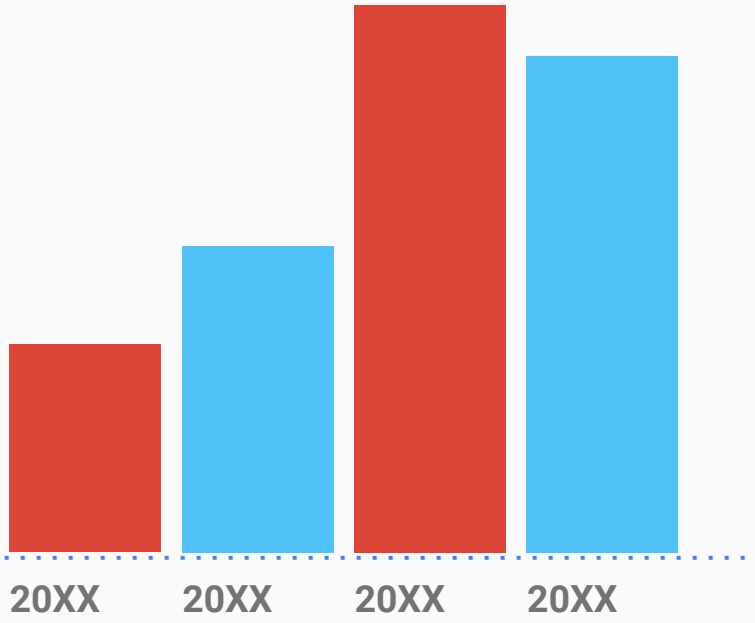
$ docker container ps

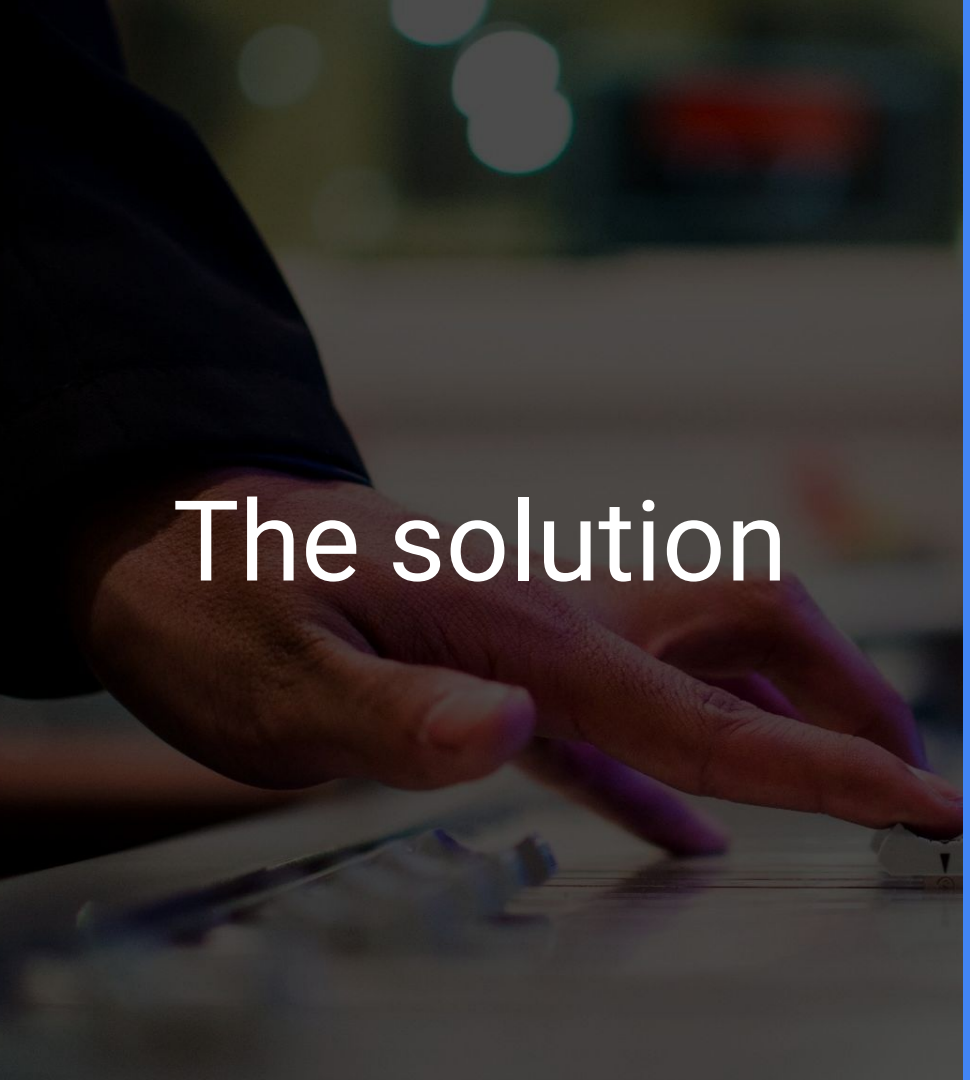

Develop Cloud Native
with Twelve Factors

The problem

Frame the problem for the audience.

Quantify the scope of the problem and connect it to your audience.





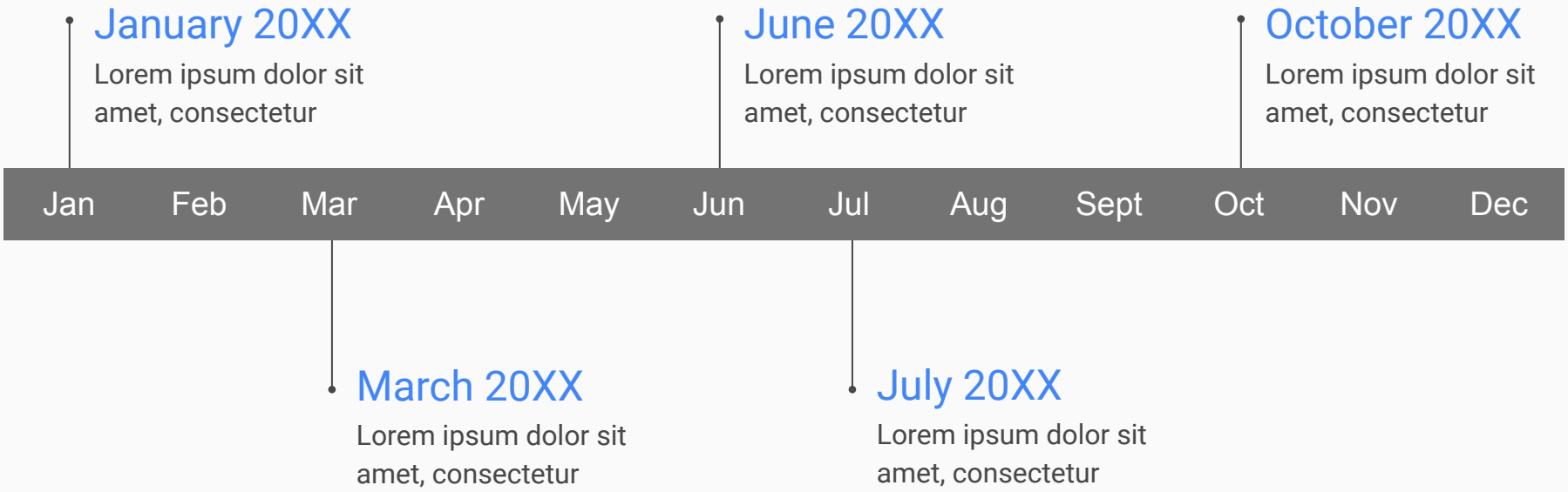
The solution

Show how you solve the problem you identified.

What will be different when the problem is solved (by you)?

Milestones

Show where you are in the process and what's left to tackle



Appendix

Show the audience you anticipated their questions.

Leave room for Q&A, but use the Appendix as a way to show that you both thought about those questions and have solid answers with supporting information. Let the audience test their understanding of the problem and the solution you've outlined - questions give them a chance to talk themselves into your approach, and give you a chance to show mastery of the subject.

How it works

Step 1

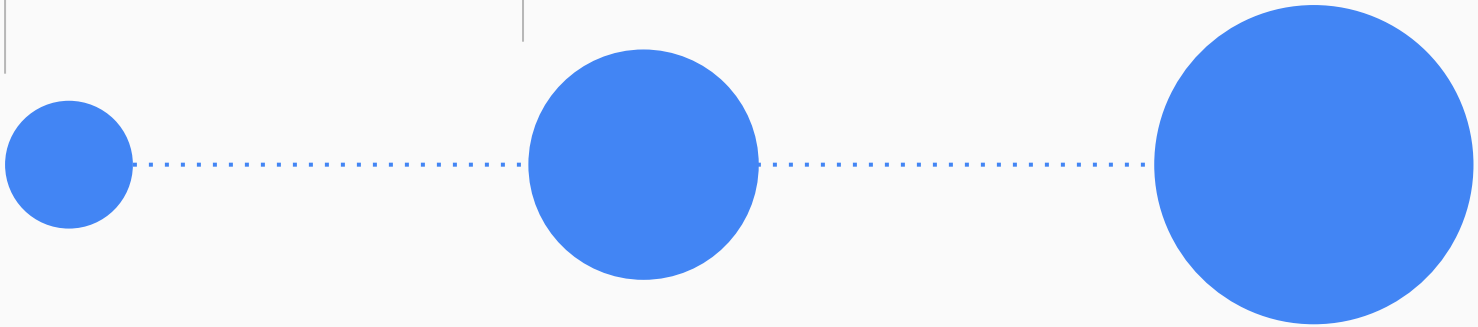
Lorem ipsum dolor sit amet, consectetur

Step 2

Lorem ipsum dolor sit amet, consectetur

Step 3

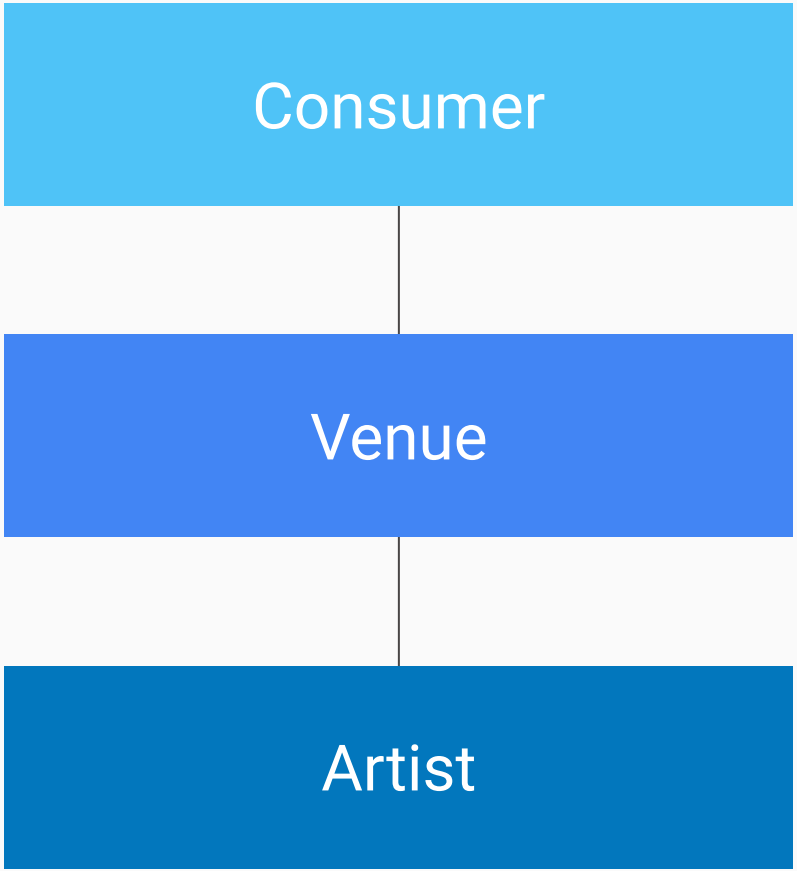
Lorem ipsum dolor sit amet, consectetur




The technology:
GPS + RFID

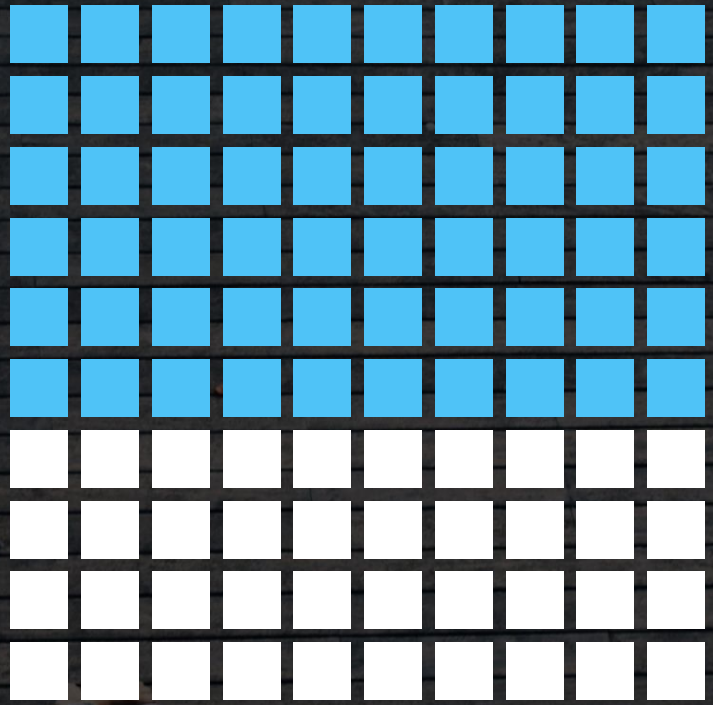
Revenue model

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt



Why now?

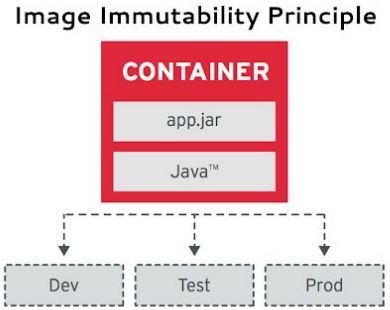
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt

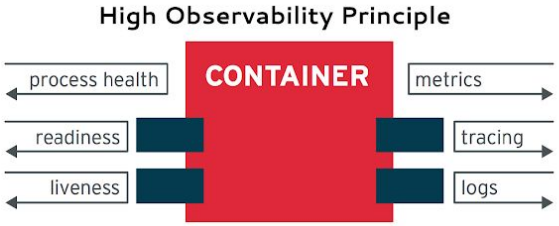


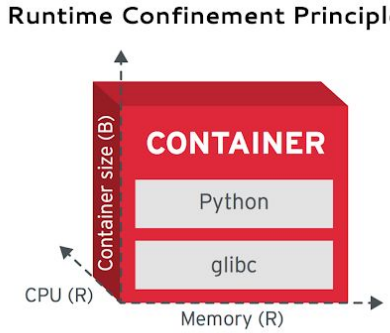
Day 2

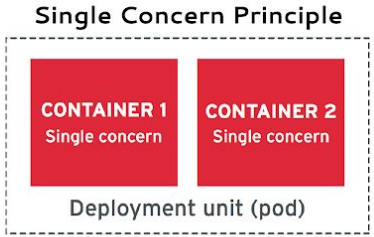

Manage Your
Enterprise
Application with
Kubernetes (K8S)

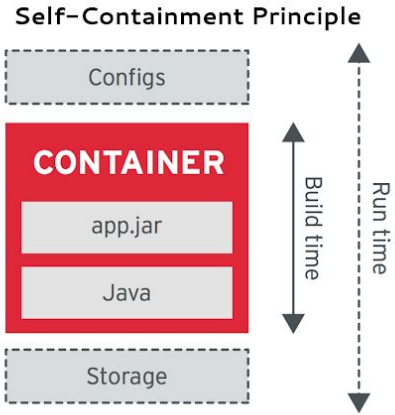
Principles of Container-based







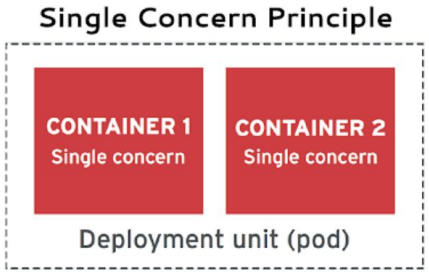


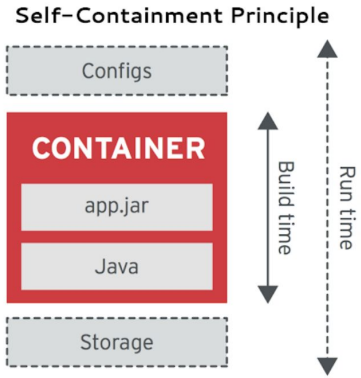


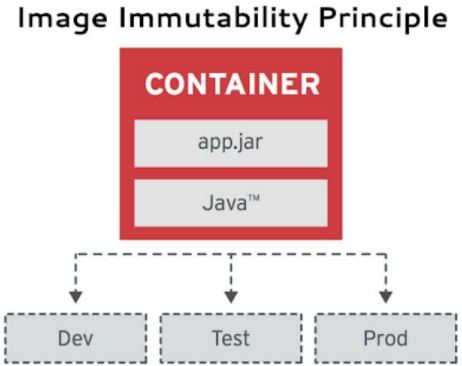
Principles of Container-based

Build Time

- **Single Concern:** Each container addresses a single concern and does it well.
- **Self-Containment:** A container relies only on the presence of the Linux kernel. Additional libraries are added when the container is built.
- **Image Immutability:** Containerized applications are meant to be immutable, and once built are not expected to change between different environments.



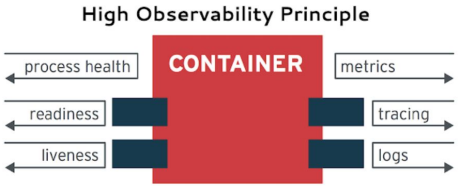


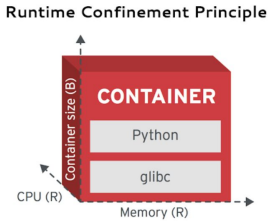


Principles of Container-based

Run Time

- **High Observability:** Every container must implement all necessary APIs to help the platform observe and manage the application in the best way possible.
- **Lifecycle Conformance:** A container must have a way to read events coming from the platform and conform by reacting to those events.
- **Process Disposability:** Containerized applications must be as ephemeral as possible and ready to be replaced by another container instance at any point in time.
- **Runtime Confinement:** Every container must declare its resource requirements and restrict resource use to the requirements indicated.





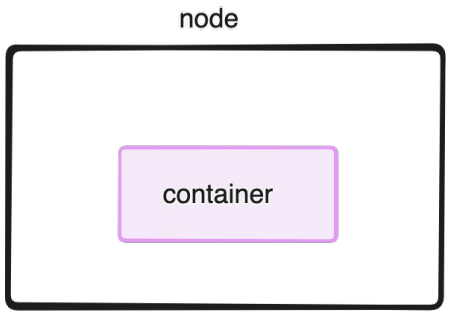
kubernetes

Agenda

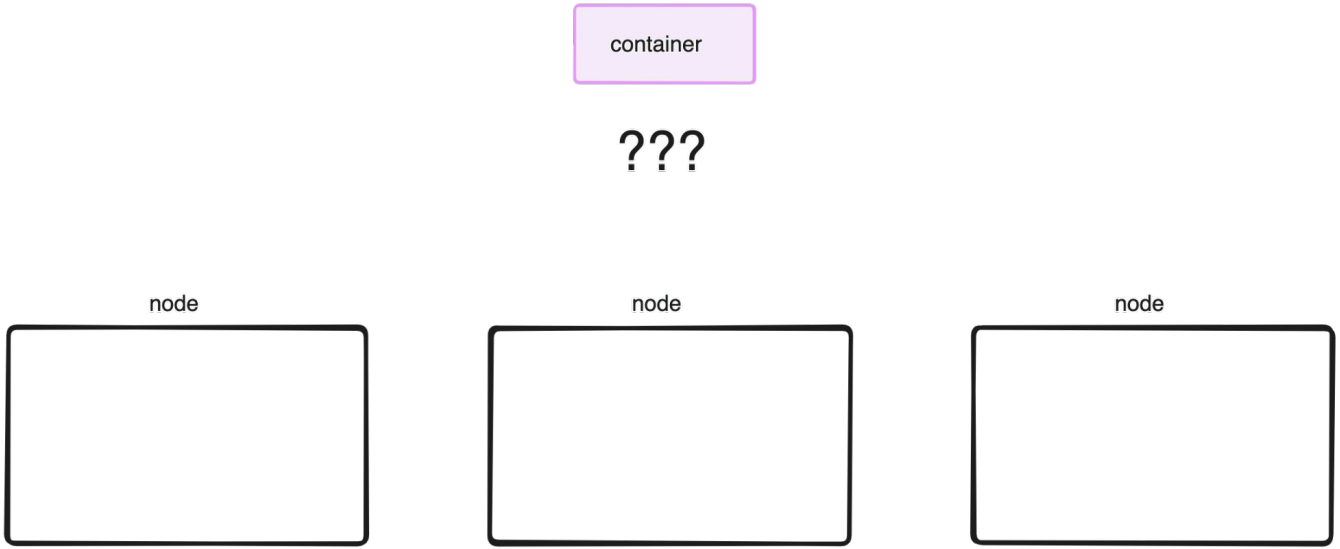
- Introduction
- Kubernetes Architecture
- Resources / Objects
- Namespace
- RBAC (Role Based Access Control)
- HELM
- Monitoring
- HPA (Horizontal Pod Autoscaler)
- Deployment & Pipelines

Introduction

Docker on one server

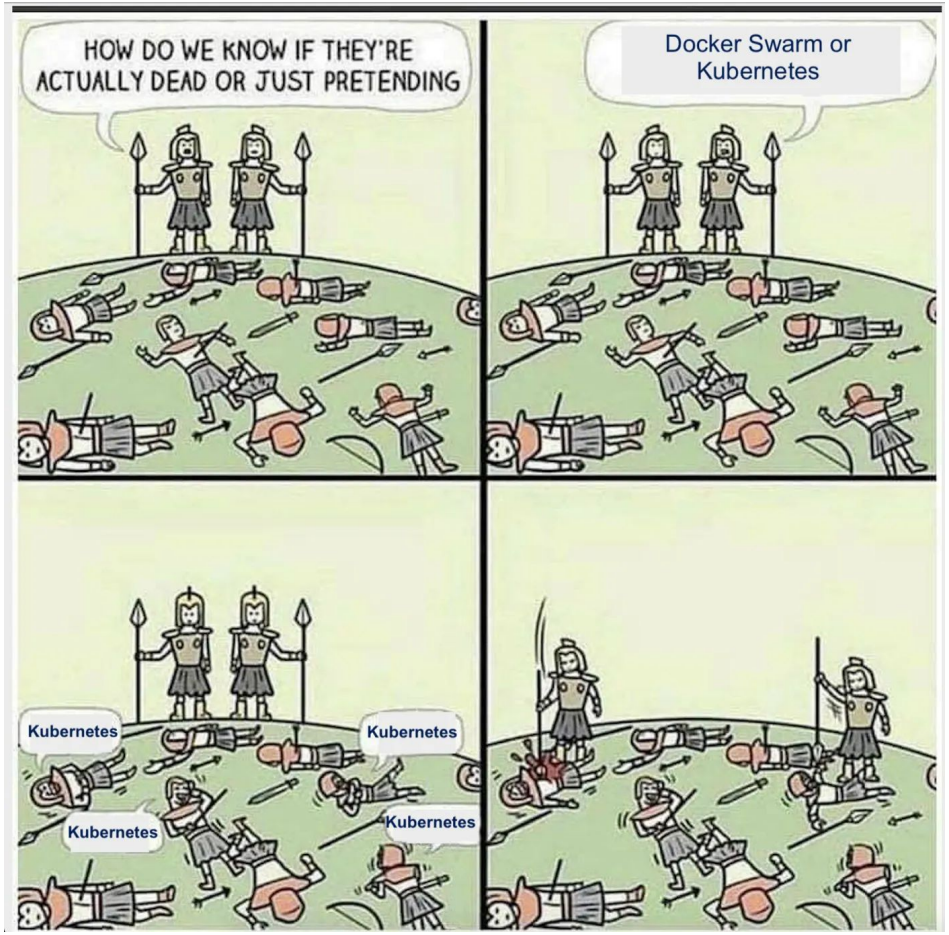


Docker on multi servers

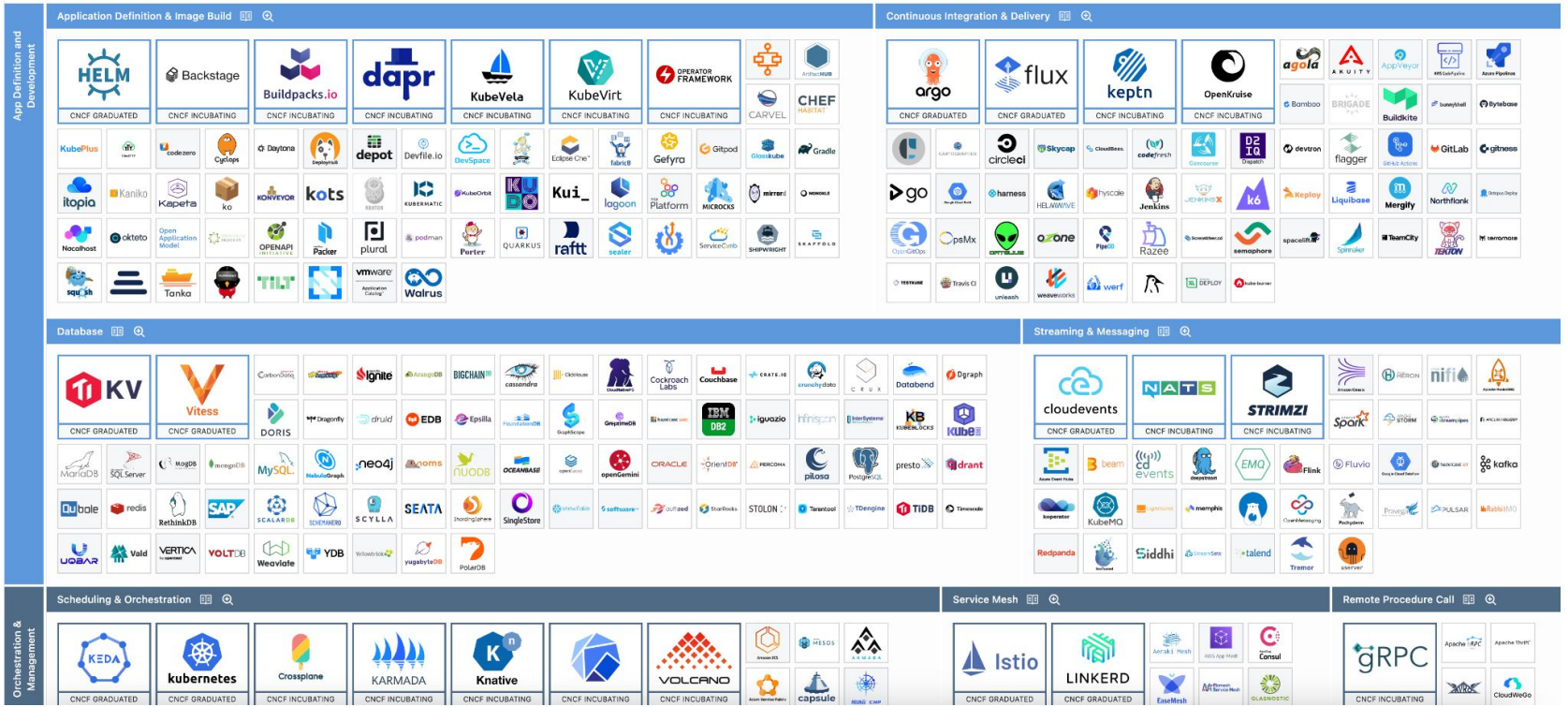


What is Kubernetes?

- Kubernetes in Greek means the Helmsman
- Also known as k8s
- Inspired and informed by Google’s experiences and internal systems
- 100% opensource, written in GO
- Container orchestrator, runs and manages containers
- Kubernetes v1.0 released on July 2015, join CNCF on March 2016
- Container choose Docker, Container Orchestrator choose Kubernetes


HOW DO WE KNOW IF THEY'RE ACTUALLY DEAD OR JUST PRETENDING
Docker Swarm or Kubernetes
Kubernetes
Kubernetes
Kubernetes
Kubernetes

CNCF Landscape



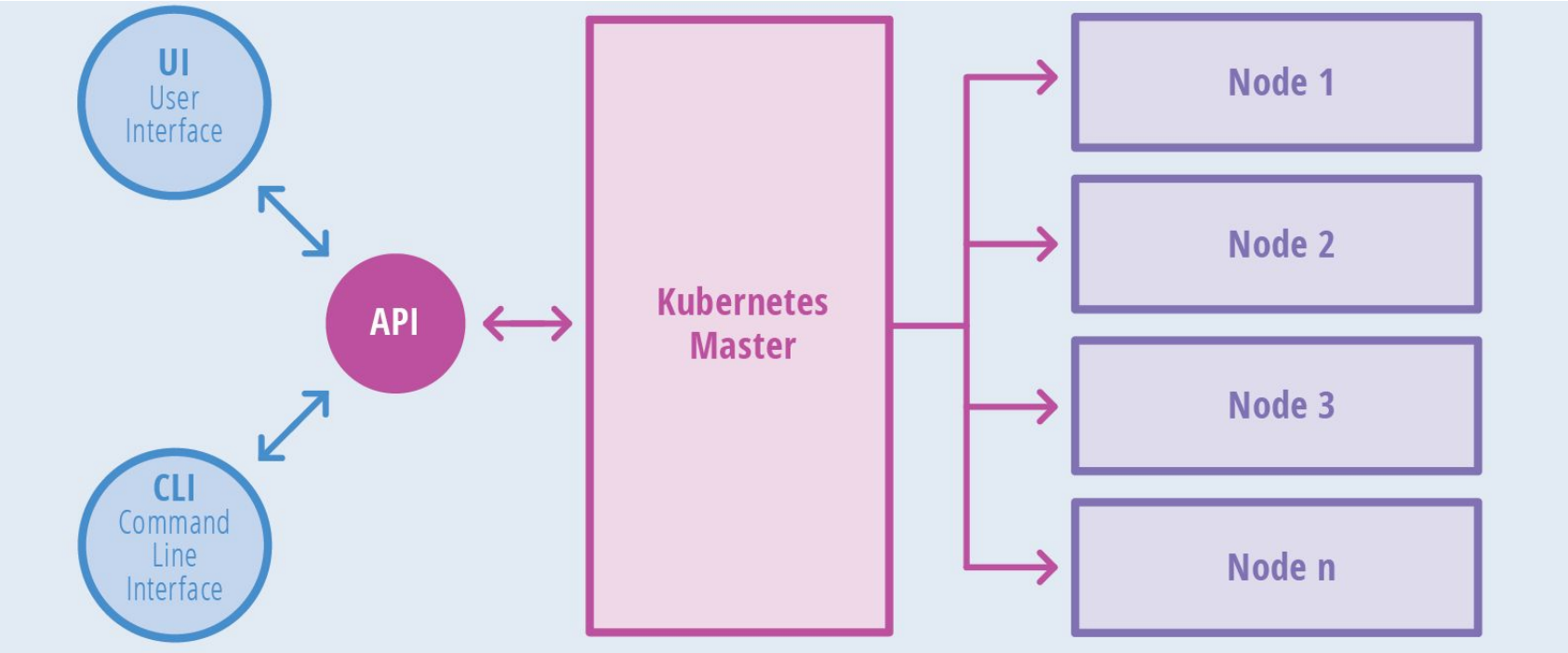
https://landscape.cncf.io/

The features of Kubernetes

- Automatic bin packing
- Self healing
- Automated rollout & rollback
- Service Discovery & load Balancing
- Horizontal scaling
- Declarative Configuration

Kubernetes Architecture

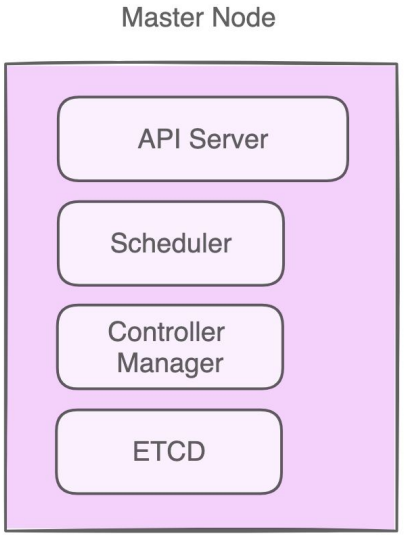
Kubernetes Architecture



Kubernetes Architecture

Master node

- Responsible for managing cluster
- Access master node via CLI, UI or API
- For fault tolerance, can be more than 1 master in cluster
- Include 4 components: **API Server, Scheduler, Controller manager** and **ETCD**

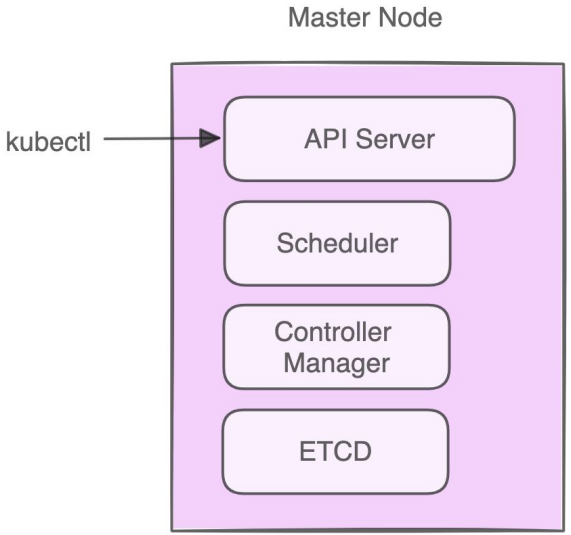


Kubernetes Architecture

Master node

API Server

- Centralized component where all the cluster components are communicated
- Execute and validate from user command

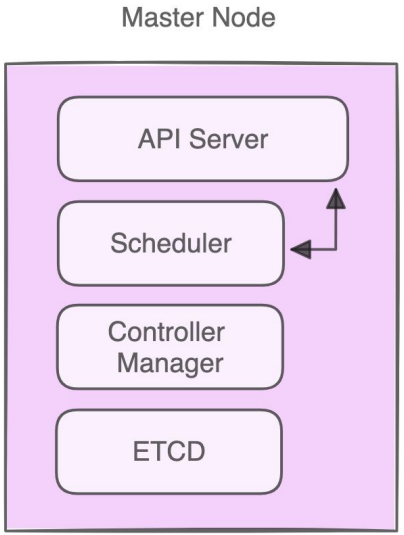


Kubernetes Architecture

Master node

Scheduler

- Assigning the application to worker node
- Automatically detect which pod to place on which node based on all factors

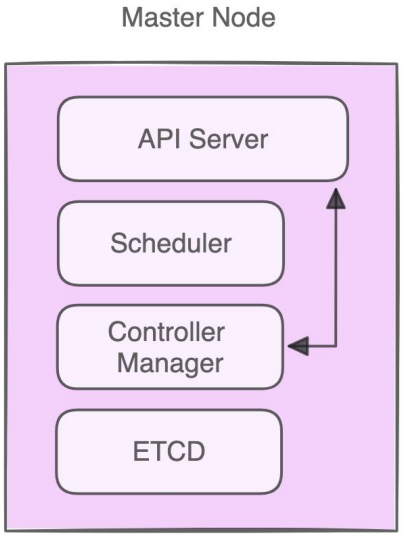


Kubernetes Architecture

Master node

Controller Manager

- The Controller Manager maintains the cluster
- Handles node failures, replicating components, maintaining the correct number of pods, etc.

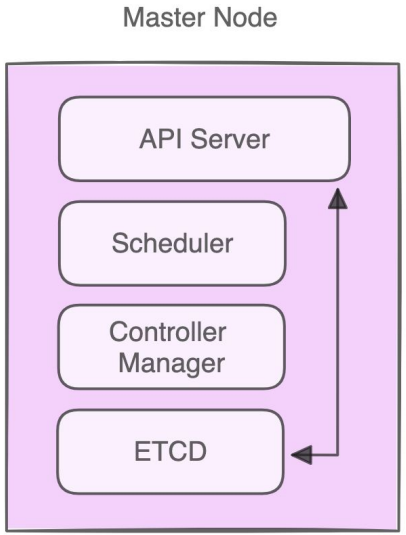


Kubernetes Architecture

Master node

ETCD

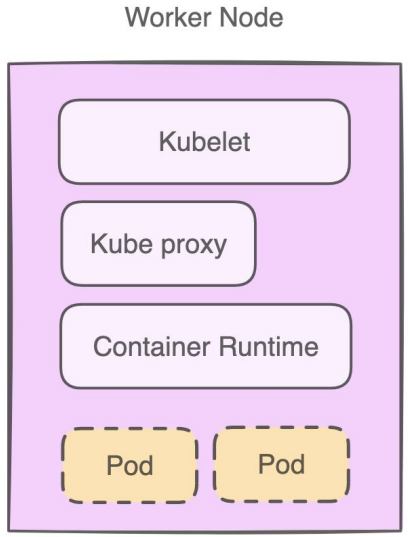
- ETCD is a distributed reliable key-value store use by Kubernetes to store all data used to manage the cluster
- When you have multi master node in cluster, ETCD stores all information in a distributed manner
- Application data doesn’t store in ETCD



Kubernetes Architecture

Worker node

- Responsible about applications
- Include 3 components: **Container Runtime, Kubelet** and **Kube Proxy**

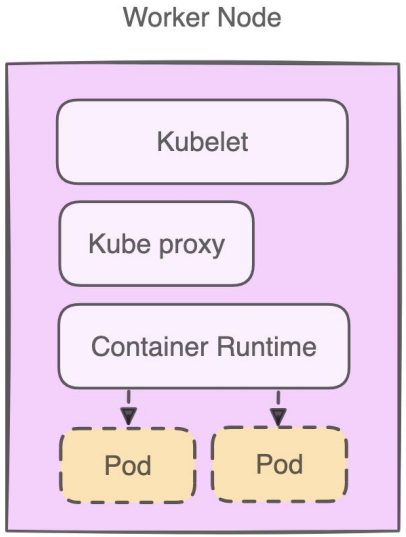


Kubernetes Architecture

Worker node

Container Runtime

- Container runtime which runs the containers like Docker, crio or containerd
- Pulling the images and and run the containers

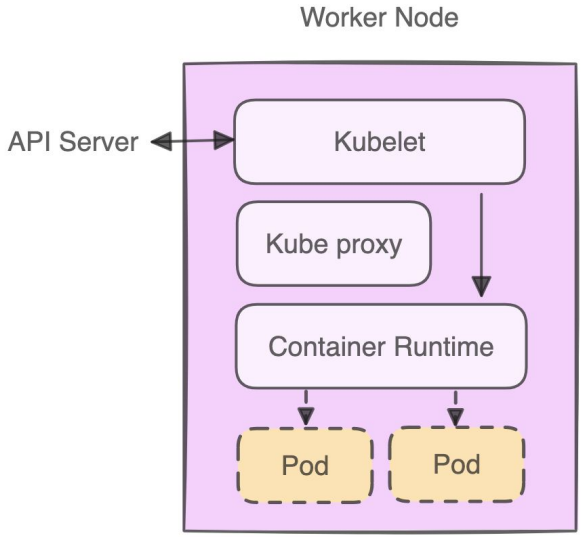


Kubernetes Architecture

Worker node

Kubelet

- Interacting with the master node
- Contact the container runtime for create pod depend on Pod Spec.
- Provide health information of the worker node

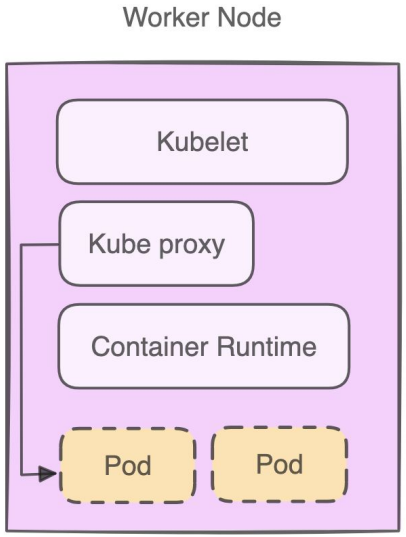


Kubernetes Architecture

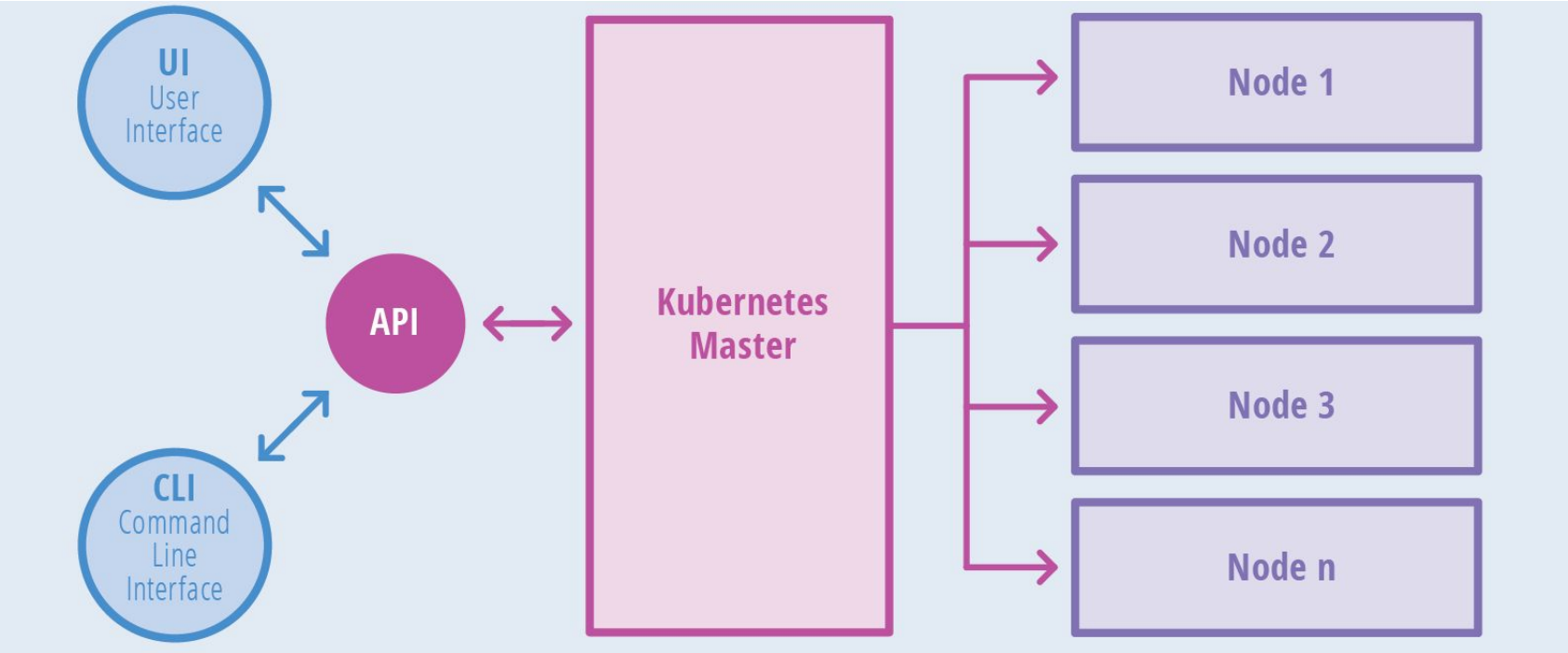
Worker node

Kube Proxy

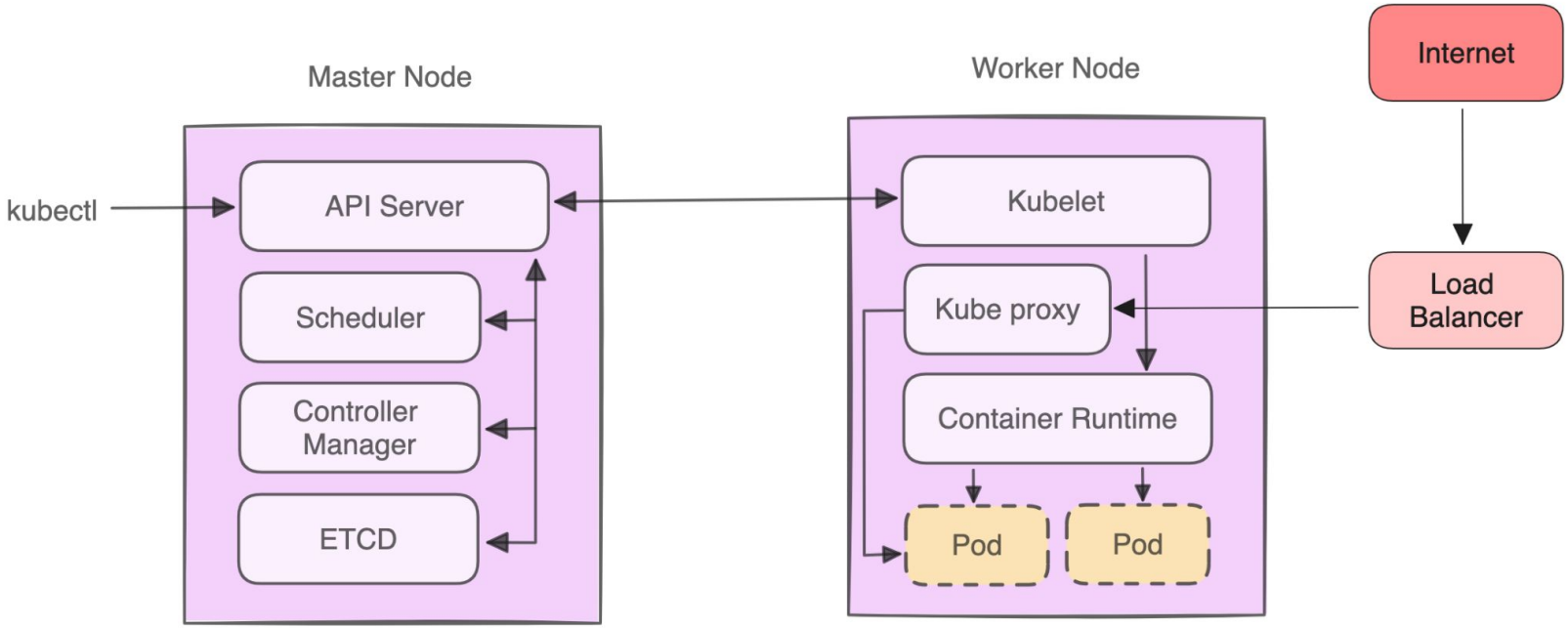
- Responsible for ensuring network traffic is routed properly to internal and external services
- Manage route for reduce latency time for example pod will communicate with another pod on same node first



Kubernetes Architecture



Kubernetes Architecture



Kubectl

Kubectl

- Command line
- Control the Kubernetes cluster manager
- CRUD Kubernetes resources

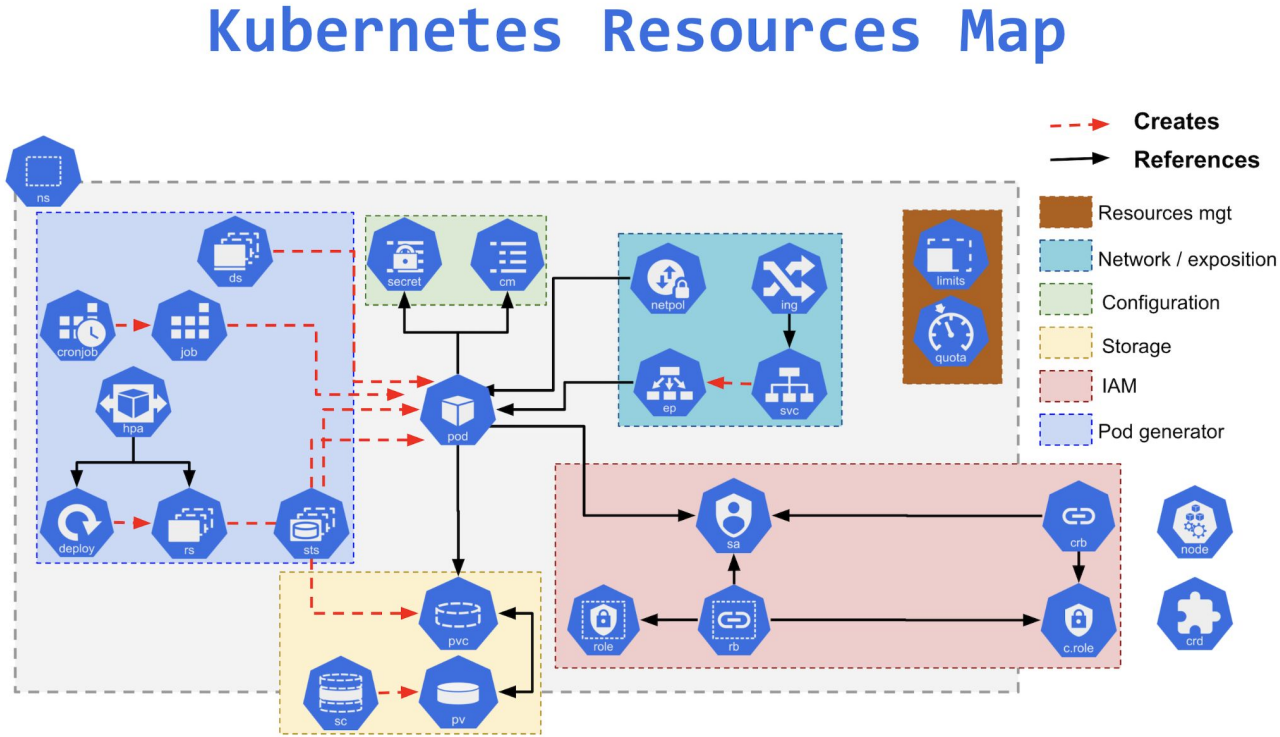
Kubectl

- $ kubectl apply
- $ kubectl create
- $ kubectl get
- $ kubectl describe
- $ kubectl delete


Resources / Objects

Resources / Objects

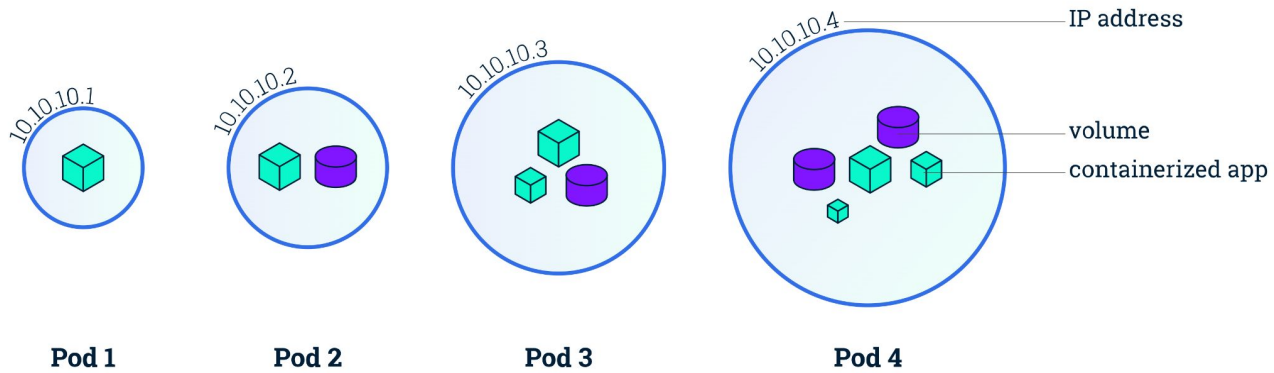
- Pod
- Deployment
- Service
- Ingress
- Config map
- Secret
- PV & PVC



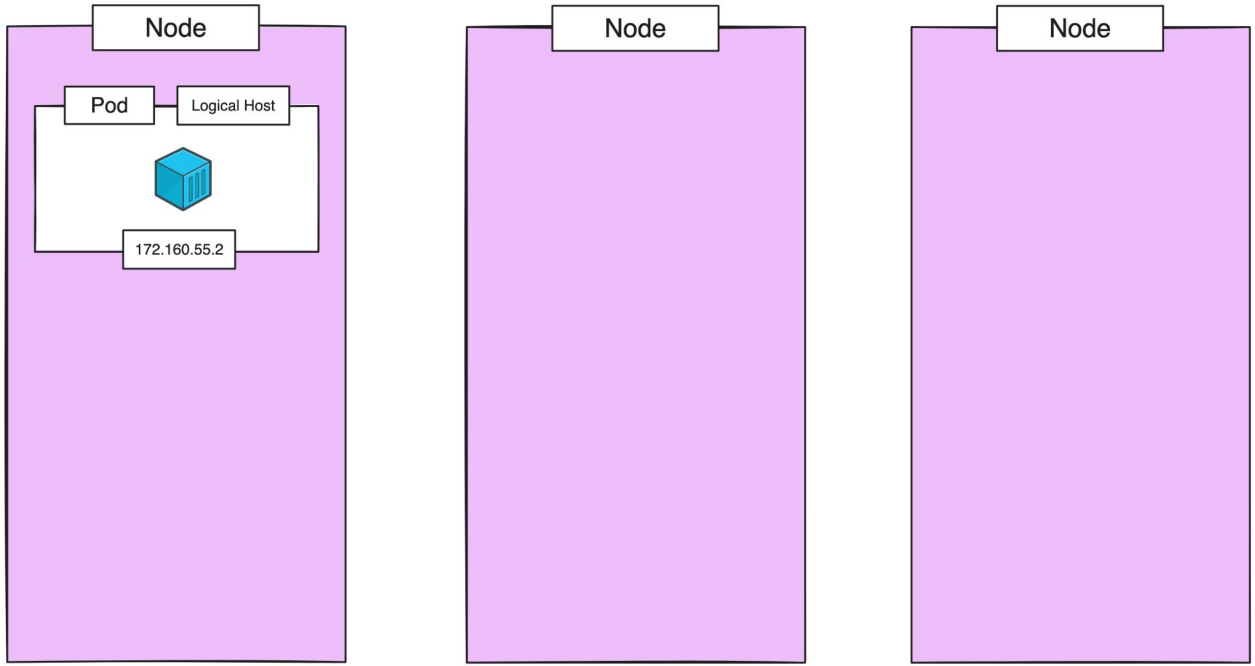
Pod

Pod

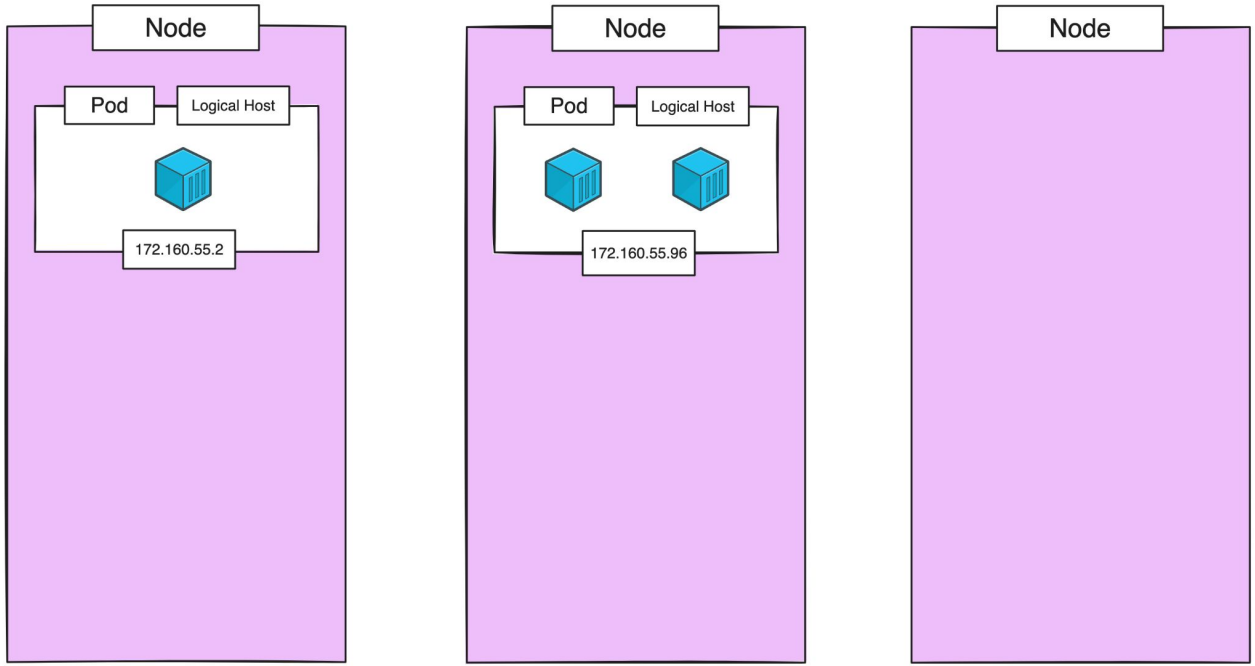
- Smallest deployable units of computing that you can create and manage in Kubernetes
- Logical Application
- One or more container in Pod (recommend 1 container per pod)
- Any containers in same pod will share storage volumes and network resources
- Each pod is assigned a unique ip



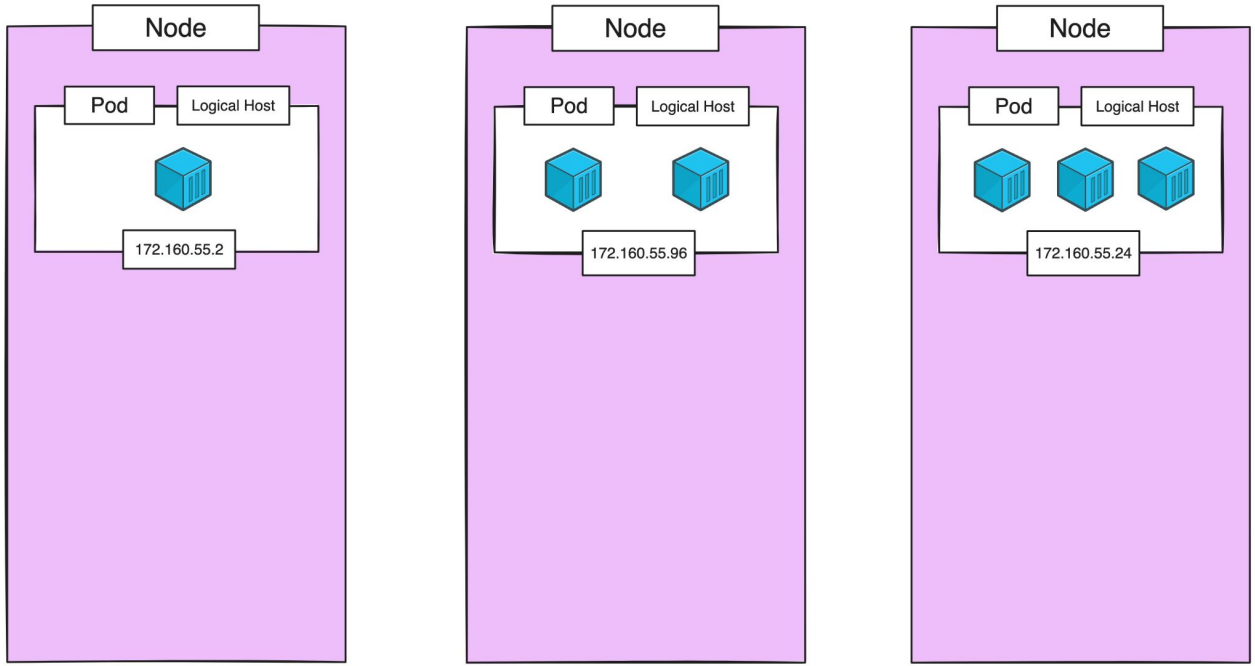
Node & Pod



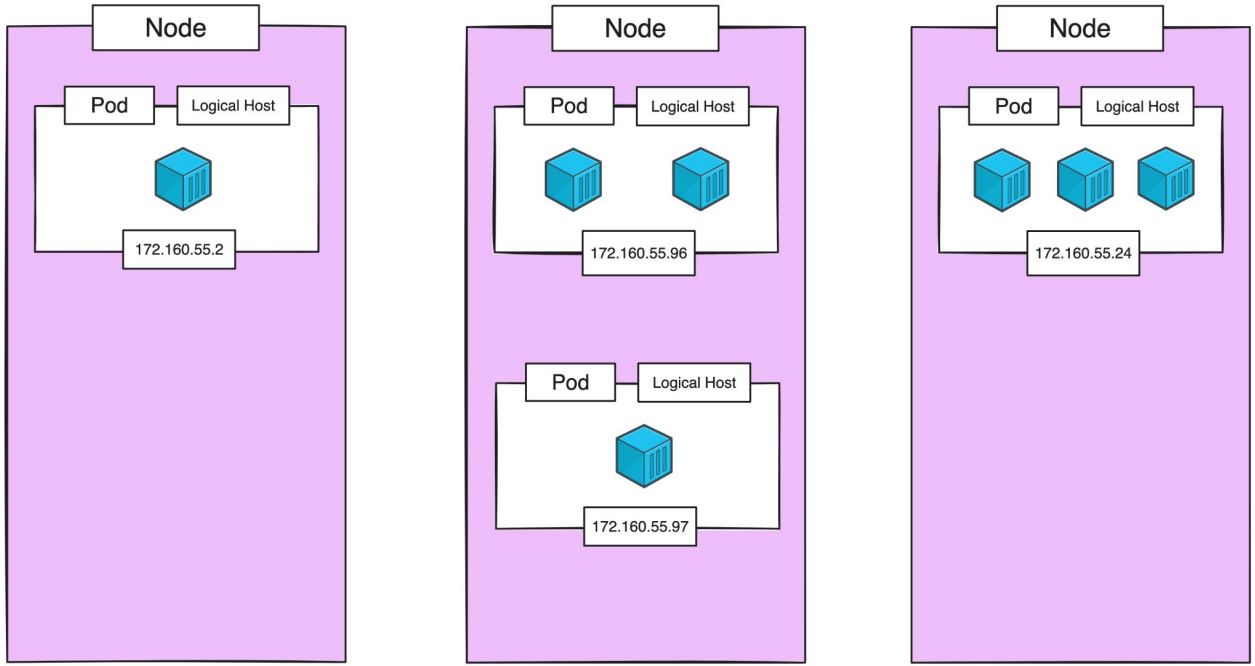
Node & Pod


Node
Pod
Logical Host
172.160.55.2
Node
Pod
Logical Host
172.160.55.96
Node

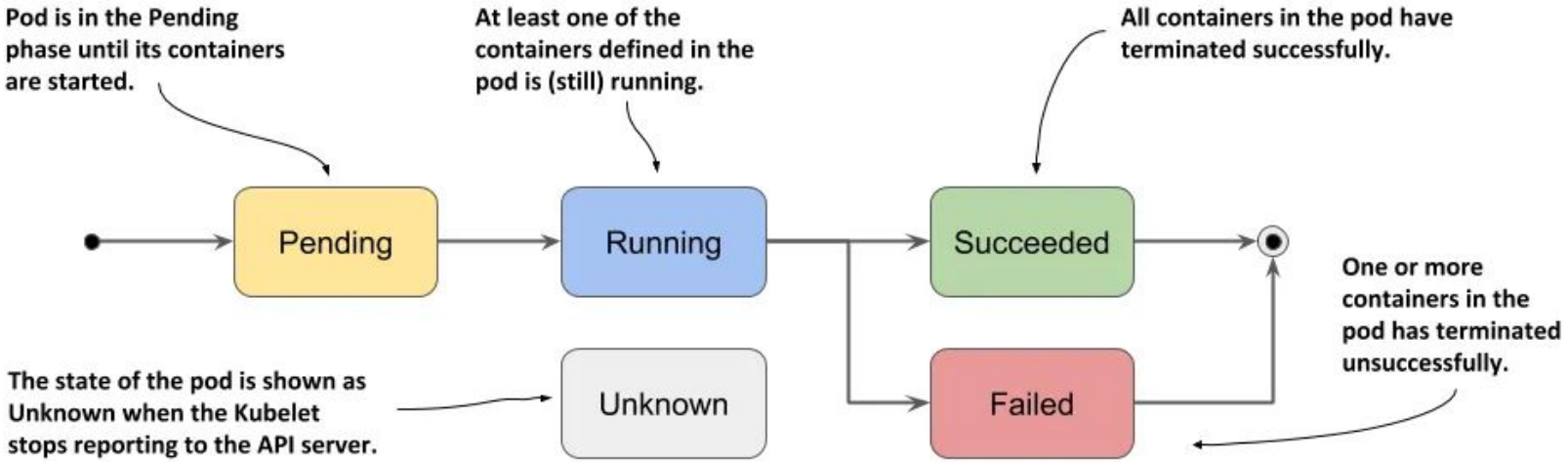
Node & Pod


Node
Pod
Logical Host
172.160.55.2
Node
Pod
Logical Host
172.160.55.96
Node
Pod
Logical Host
172.160.55.24

Node & Pod

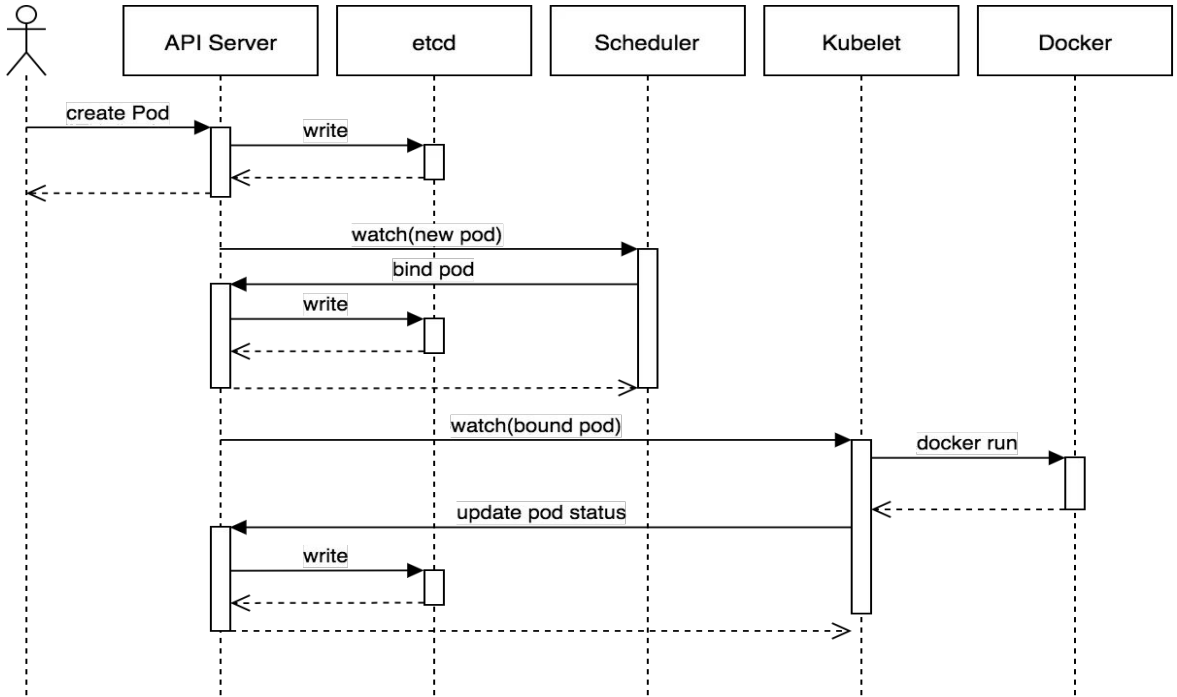

Node
Pod
Logical Host
172.160.55.2
Node
Pod
Logical Host
172.160.55.96
Pod
Logical Host
172.160.55.97
Node
Pod
Logical Host
172.160.55.24

States of a Pod



https://medium.com/@seifeddinerajhi/navigating-the-journey-of-a-pod-a-guide-to-the-exciting-world-of-pod-lifecycle-a1fbc2c98c55

Creation of a Pod



https://medium.com/@seifeddinerajhi/navigating-the-journey-of-a-pod-a-guide-to-the-exciting-world-of-pod-lifecycle-a1fbc2c98c55

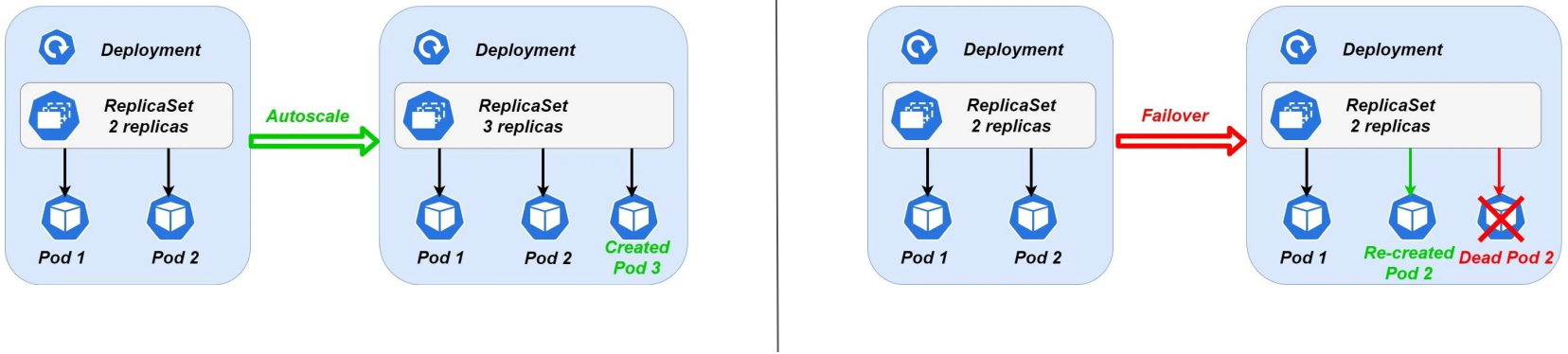
Workshop

Pod

Deployment

Deployment

- A Deployment provides declarative updates for Pods and ReplicaSets
- You describe a **desired state** in a Deployment, and the Deployment Controller **changes the actual state to the desired state** at a controlled rate
- Manage for rollout and rollback
- Deployment => ReplicaSet => pod

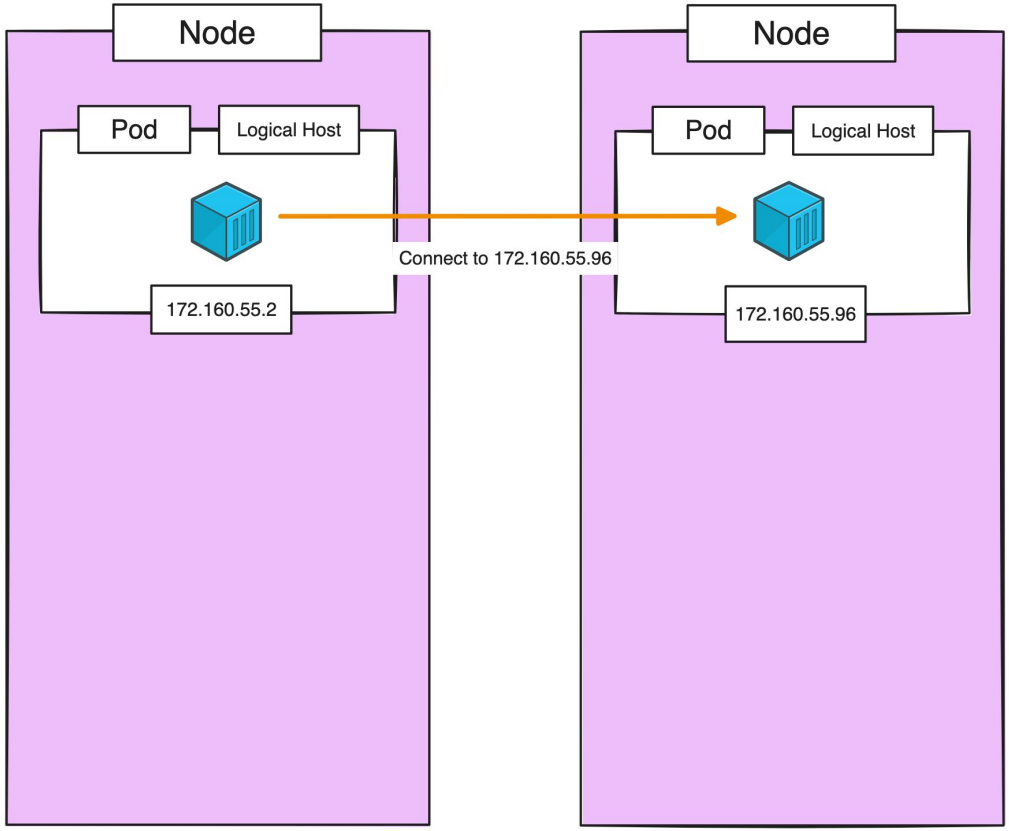



Me trying to
delete a pod
kubectl delete pod my-app-6b97-2j
pod
deploy
replicas: 12

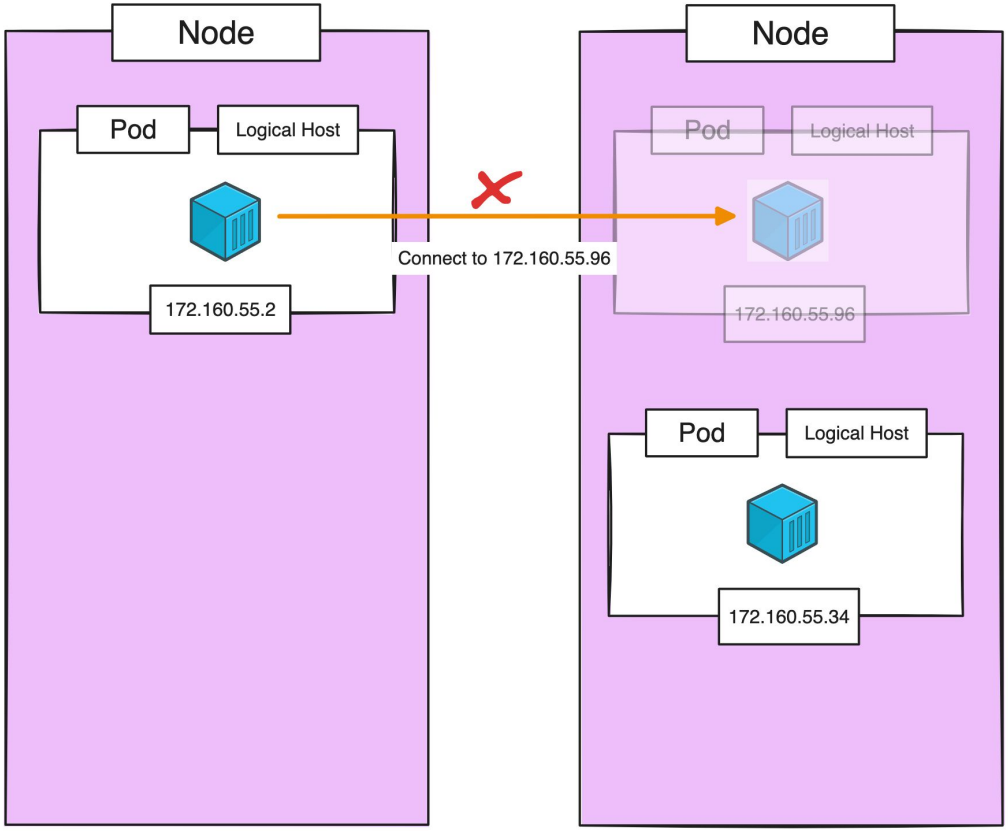
Workshop

Deployment

Problem


Node
Pod
Logical Host
172.160.55.2
Connect to 172.160.55.96
Node
Pod
Logical Host
172.160.55.96

Problem

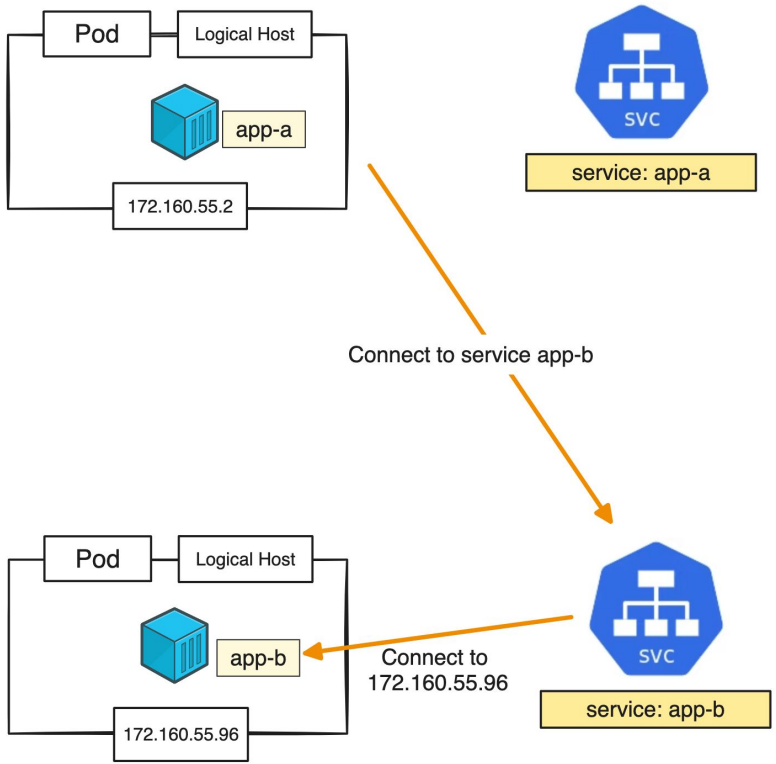

Node
Pod
Logical Host
172.160.55.2
Connect to 172.160.55.96
Node
Pod
Logical Host
172.160.55.96
Pod
Logical Host
172.160.55.34

Service

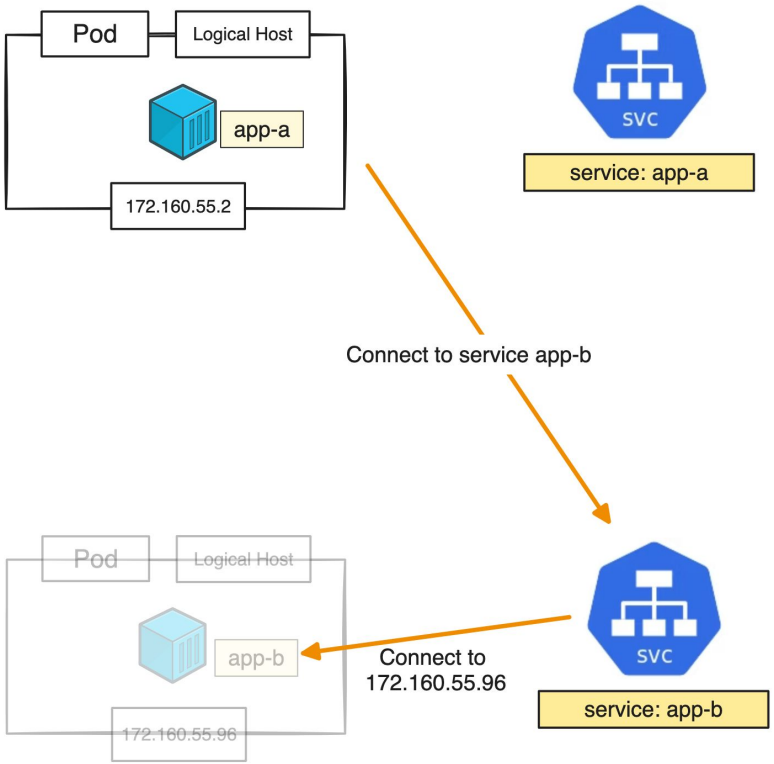
Service

- Acts as a **virtual endpoint** for your application
- Providing a stable and discoverable way to access your pods without worrying about their individual IP addresses or network details
- Logical load balancer
- Communicating for outside and inside of cluster
- 4 types:
    - ClusterIP
    - NodePort
    - LoadBalancer
    - ExternalName

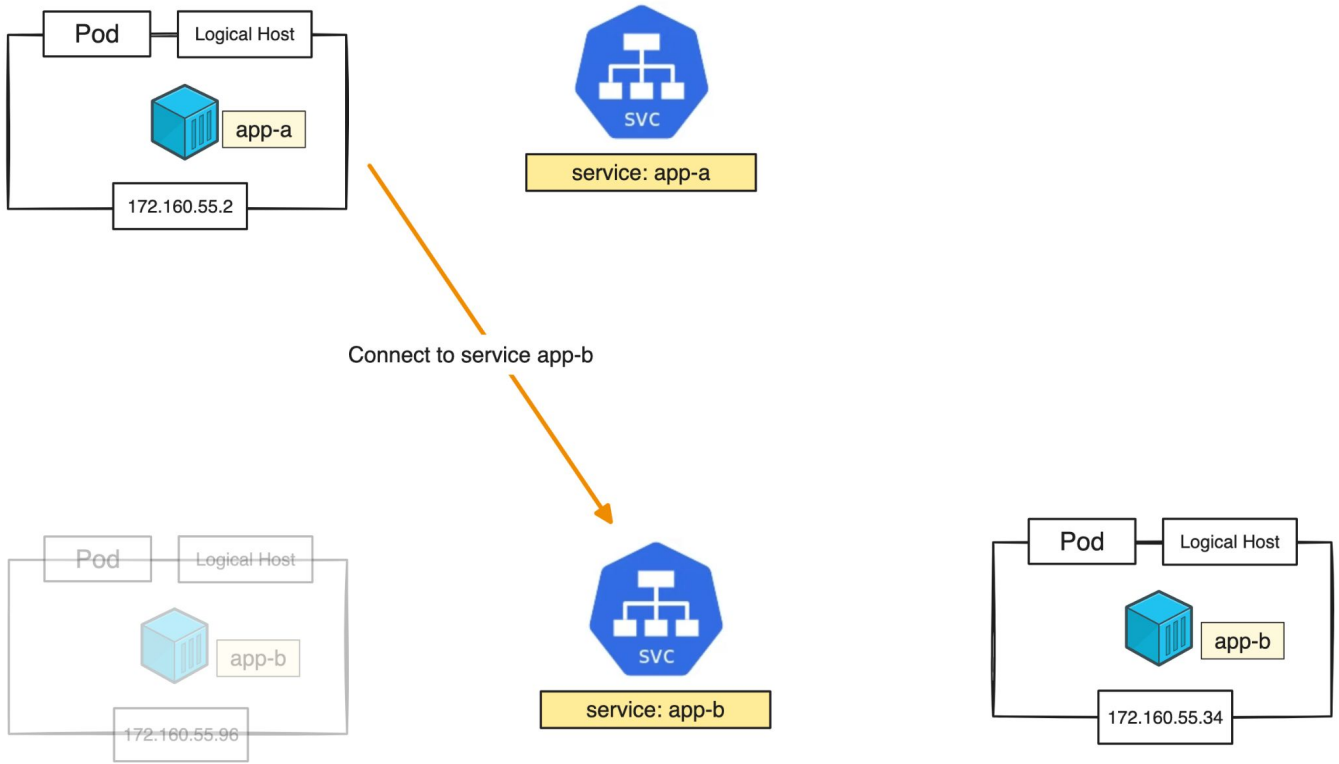
Service: Stable endpoint



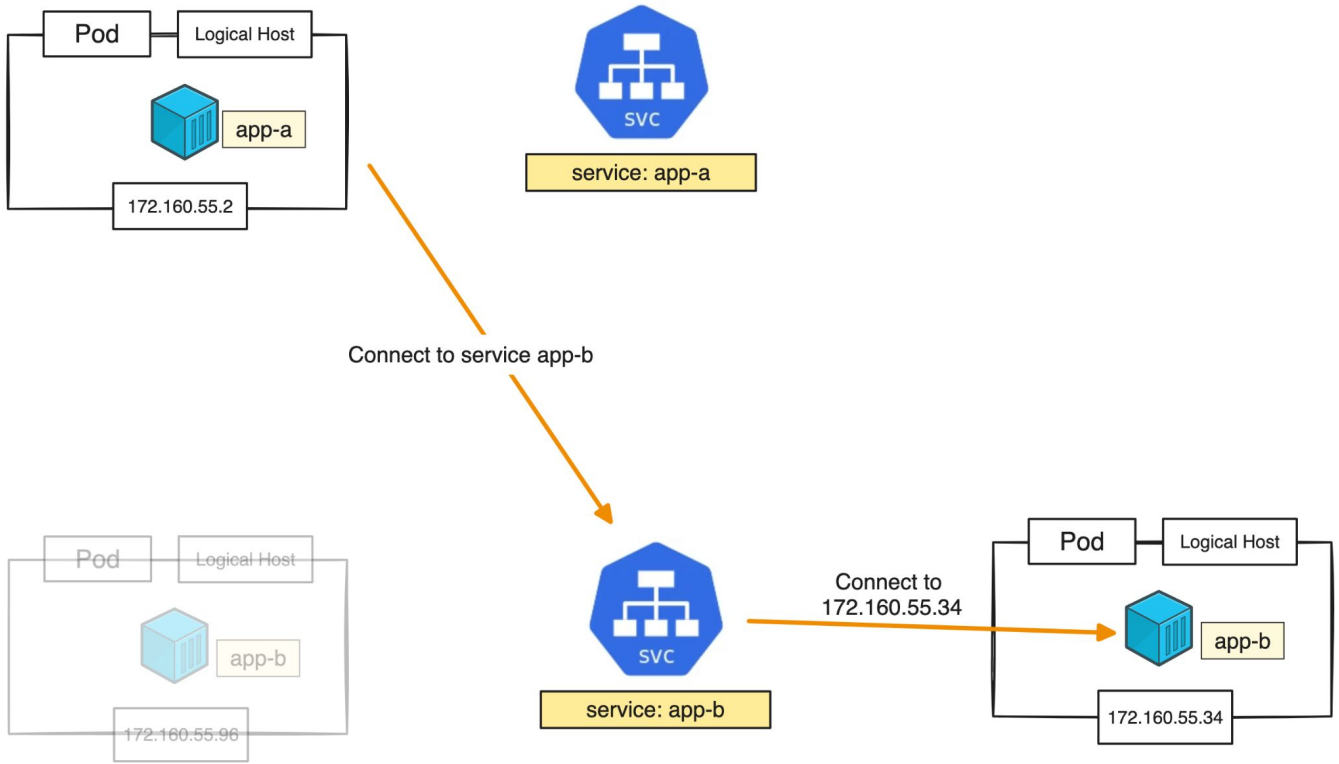
Service: Stable endpoint



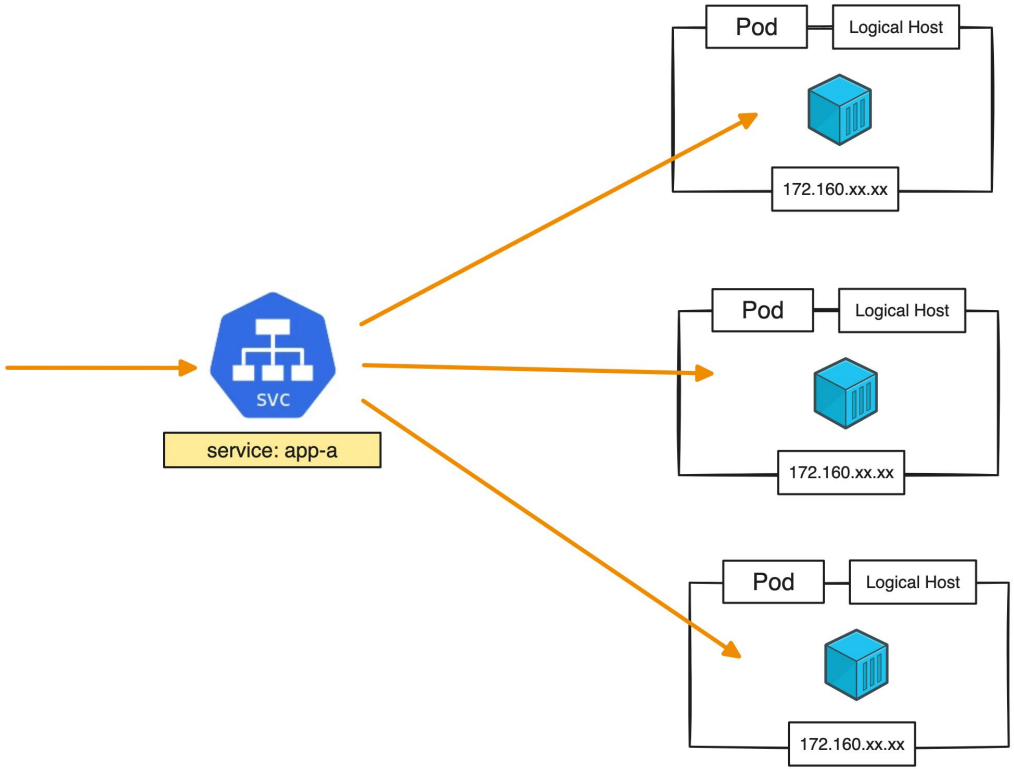
Service: Stable endpoint



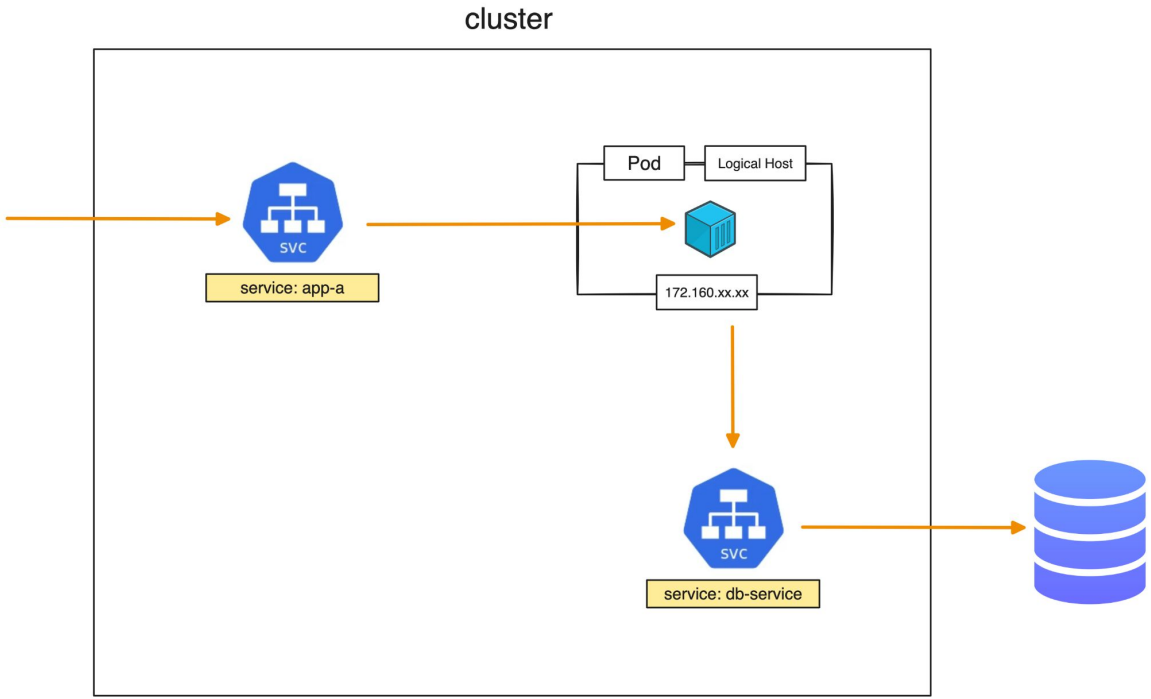
Service: Stable endpoint



Service: Load balancer

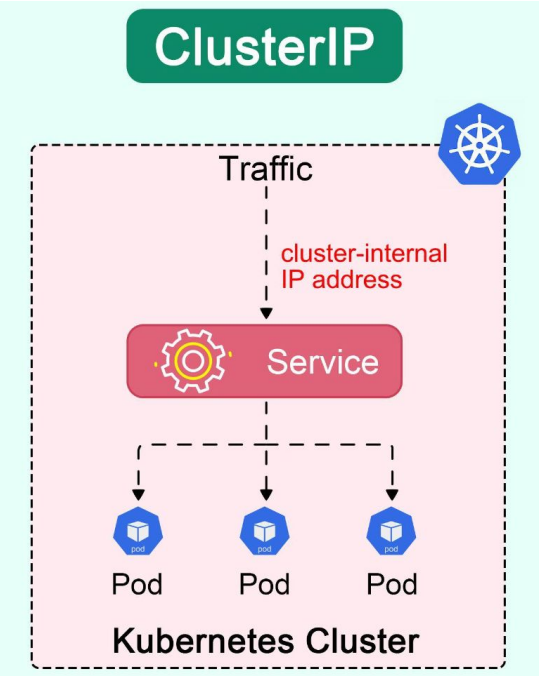


Service: Communicating inside & outside of cluster



Service type: ClusterIP

- ClusterIP is the default service type in Kubernetes
- It exposes the service on an internal IP address reachable **only within** the **cluster**



Advantages

- Suitable for inter-pod communication and internal services

Disdvantages

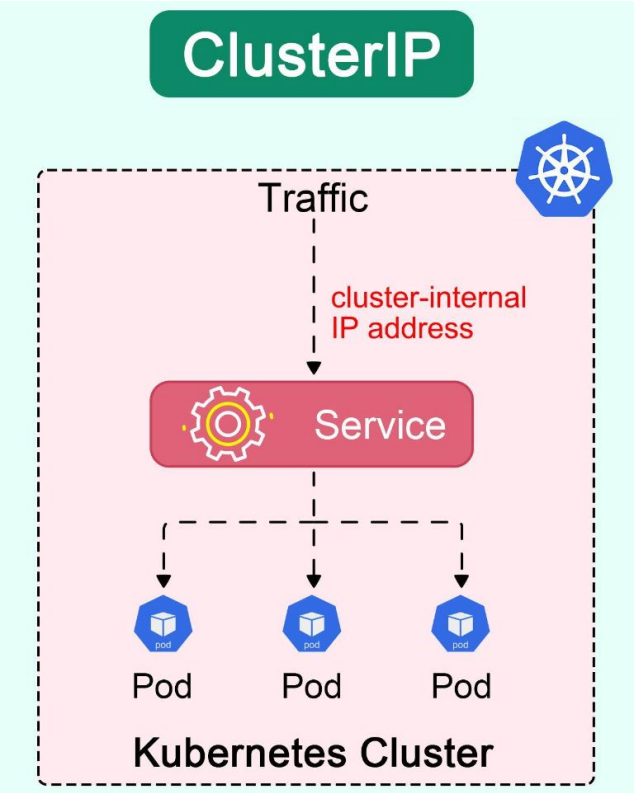
- Limited to internal cluster communication
- Not accessible externally

Service type: ClusterIP

Use Cases

- Inter-service communication within the cluster.

  For example, communication between the front-end and back-end components of your app



Service type: NodePort

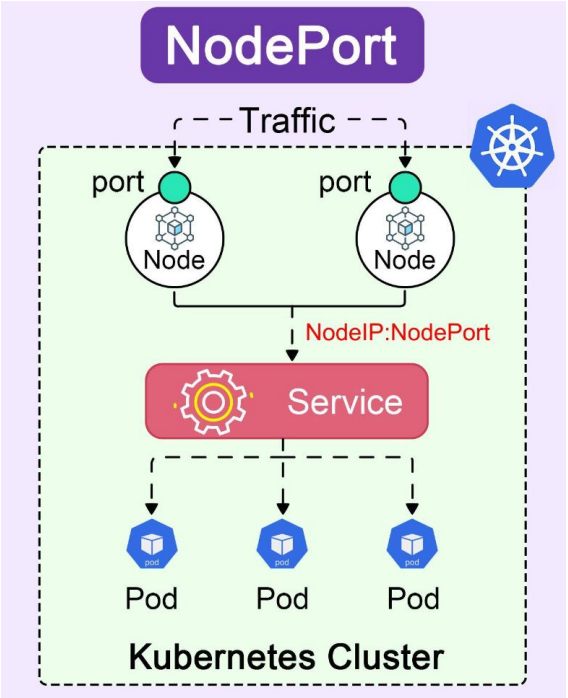
- It enables **external access** to the service using the cluster **nodes' IP address** and the assigned **NodePort**
- When a NodePort service is created, Kubernetes allocates a port from a predefined range (default: **30000-32767**)

Advantages

- Provides external access to the service
- Suitable for development and testing environments

Disdvantages

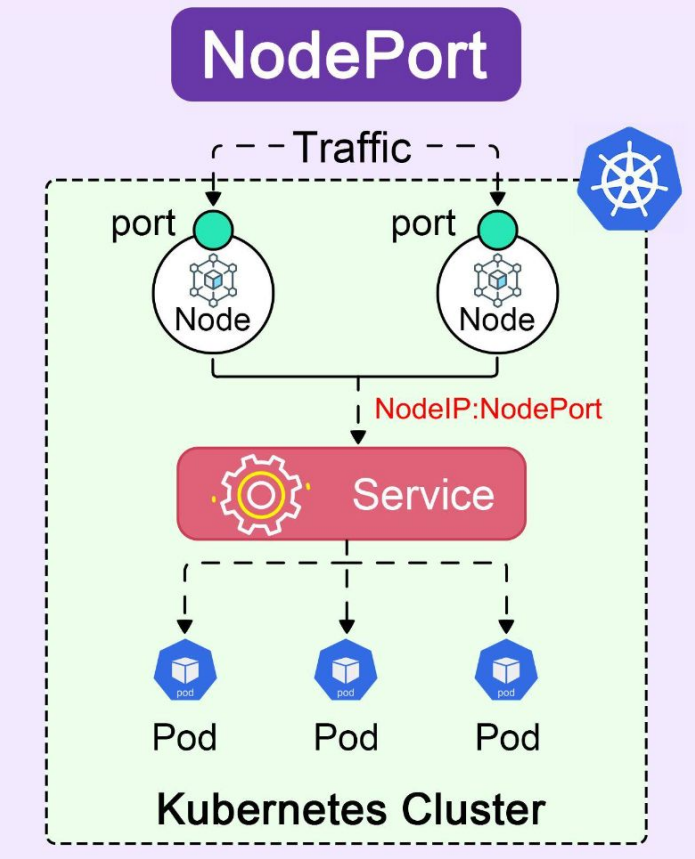
- Requires manual management of port allocations
- IP Node maybe have changed
- Should not use in production



Service type: NodePort

Use Cases

- When you want to enable external connectivity to your service
- Best for testing access for a moment time



Service type: LoadBalancer

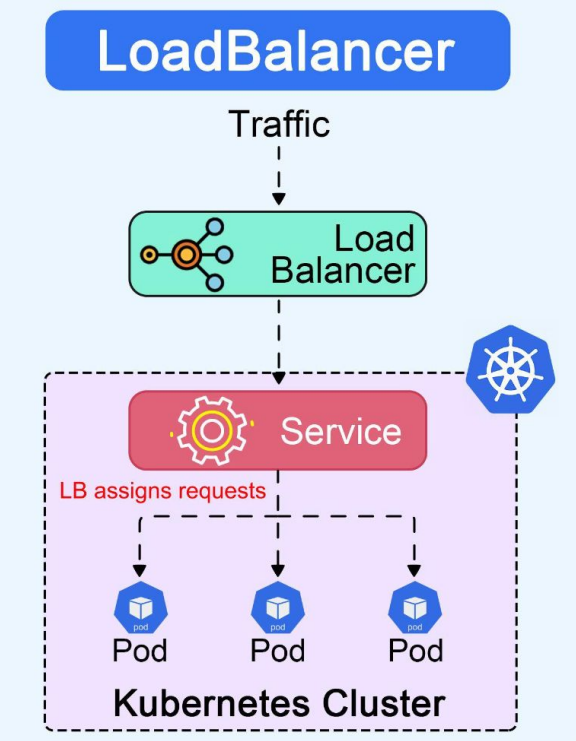
- LoadBalancer is a service type that **provisions** an **external load balancer** to distribute traffic across the pods
- Typically provided by a cloud provider's infrastructure
- When a LoadBalancer service is created, Kubernetes interacts with the cloud provider's API to an external load balancer

Advantages

- Provides external access to the service
- Automatically provisions an external load balancer for the service
- Supports scaling and high availability

Disdvantages

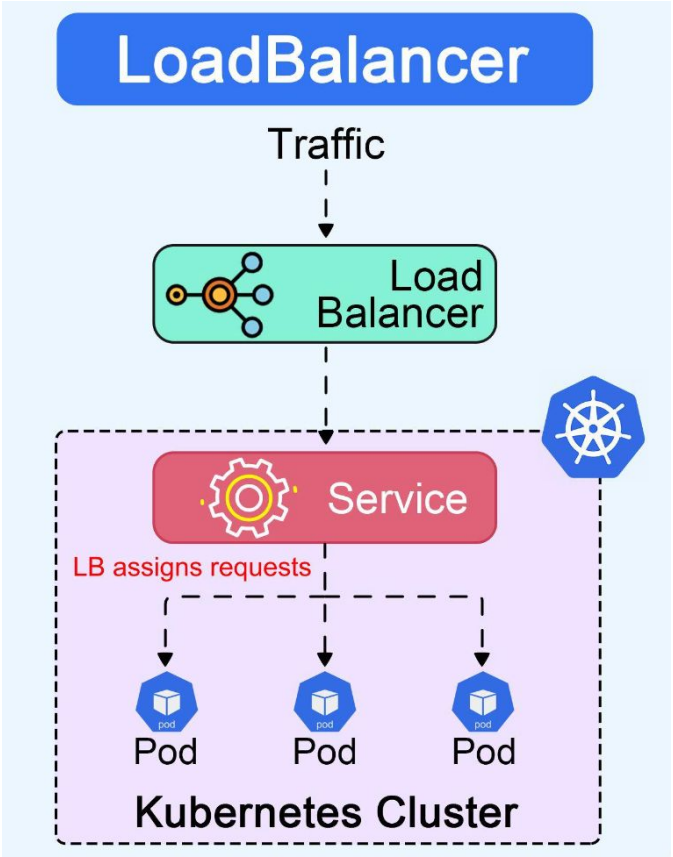
- Requires a cloud provider
- Additional costs for using the load balancer service on cloud provider



Service type: LoadBalancer

Use Cases

- When you want to enable external connectivity to your service
- When you are using a cloud provider to host your Kubernetes cluster

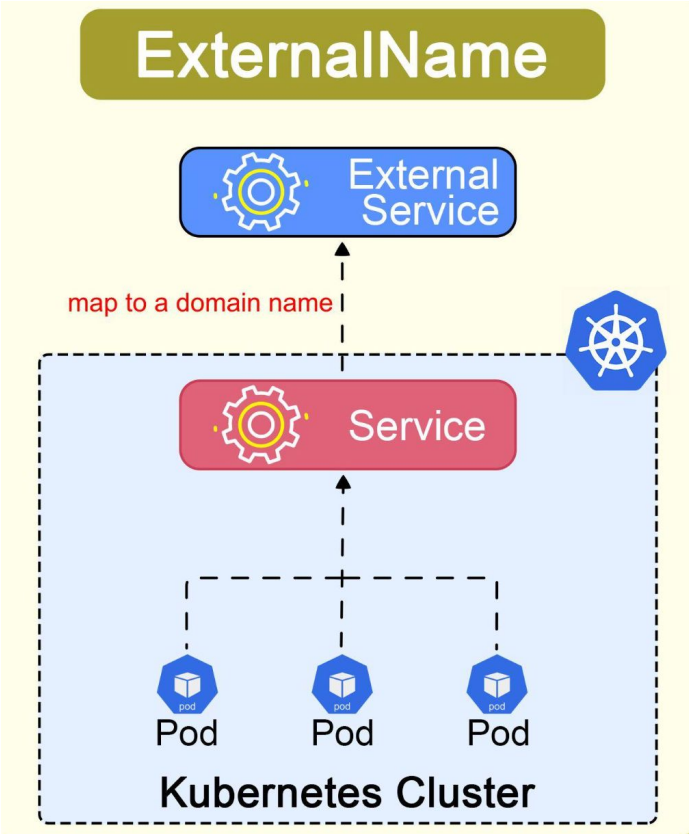


Service type: ExternalName

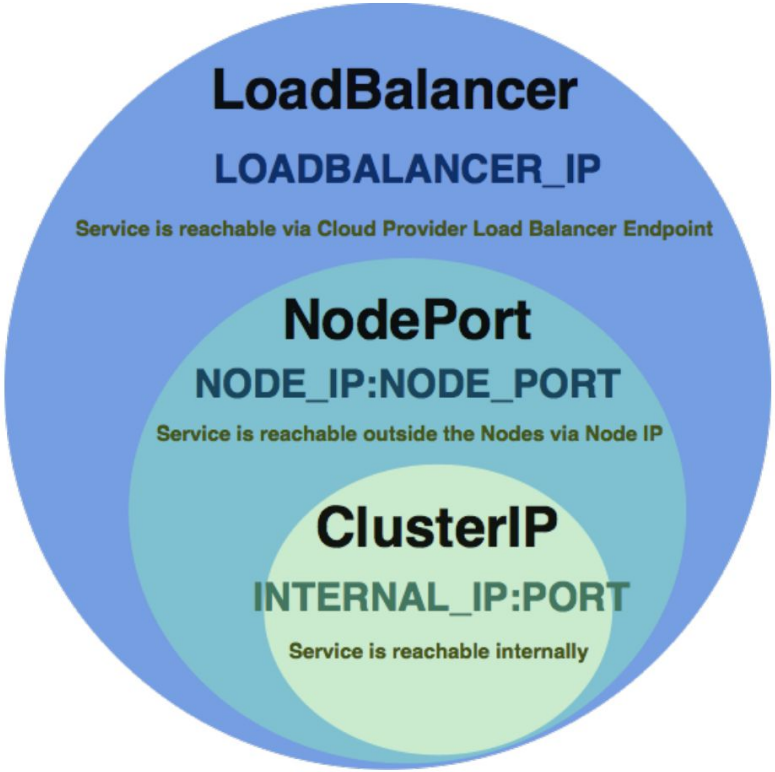
- An ExternalName Service is a special type of Kubernetes service that allows you to expose an existing external service (like a database or a third-party API) to your cluster without deploying any Pods

Use Cases

- Accessing an external database to your Kubernetes cluster
- Connecting a third-party API from within your cluster



Service type



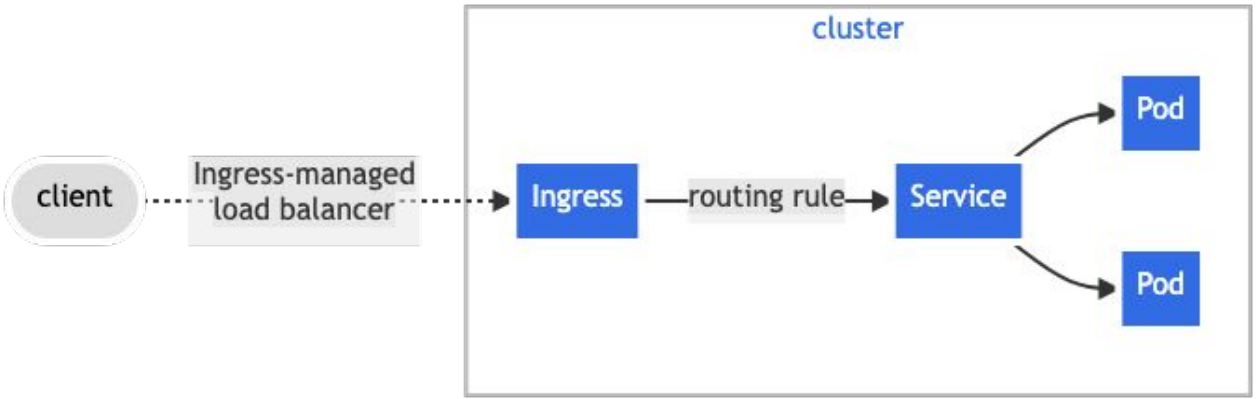
Workshop

Service

Ingress

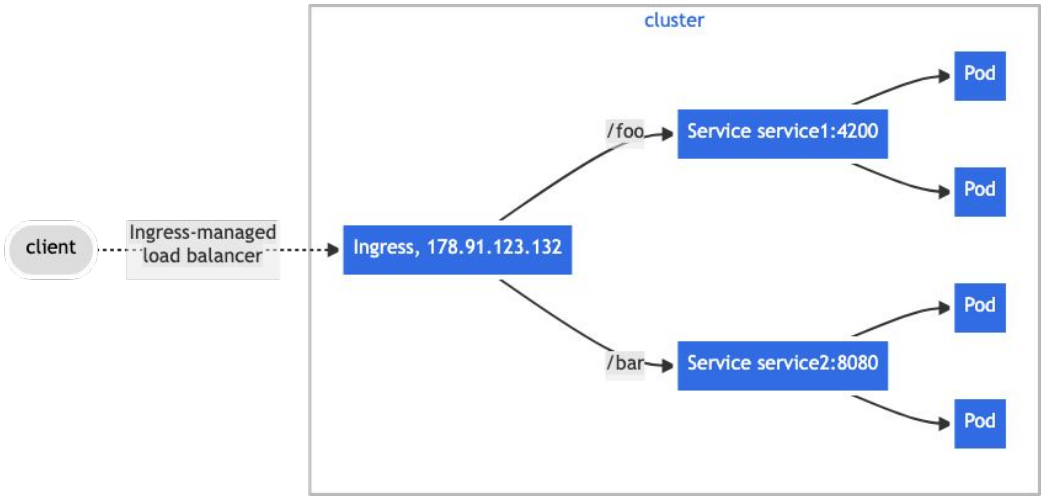
Ingress

- Routing from outside the cluster to services within the cluster
- Traffic routing is controlled by rules defined on the ingress resource
- An **Ingress controller** is responsible for fulfilling the Ingress



Ingress Controllers

- The Ingress Controller is an application hosted inside a Kubernetes cluster that actively **manages traffic** following Ingress Resources and pre-defined Ingress Rules
- Ingress controllers doesn't come with standard, have to be deployed separately
- Popular controllers include ingress-nginx, HAProxy, Traefik



Workshop

Ingress

ConfigMap & Secret

ConfigMap

- Store non-confidential data in key-value pairs
- Pods can consume ConfigMaps as environment variables
- The data stored in a ConfigMap cannot exceed 1 MiB
- If need to store larger than this limit, may want to consider mounting a volume or use a separate database or file service

Secret

- Secrets are similar to ConfigMaps but are specifically intended to hold confidential data
- Kubernetes Secrets are, by default, stored **unencrypted** in the API server's underlying data store (etcd)
- Anyone with API access can retrieve or modify a Secret
- In order to safely use Secrets, take at least the following steps
    1. Enable Encryption at Rest for Secrets
    2. Enable or configure RBAC rules with least-privilege access to Secrets
    3. Restrict Secret access to specific containers
    4. Consider using external Secret store providers

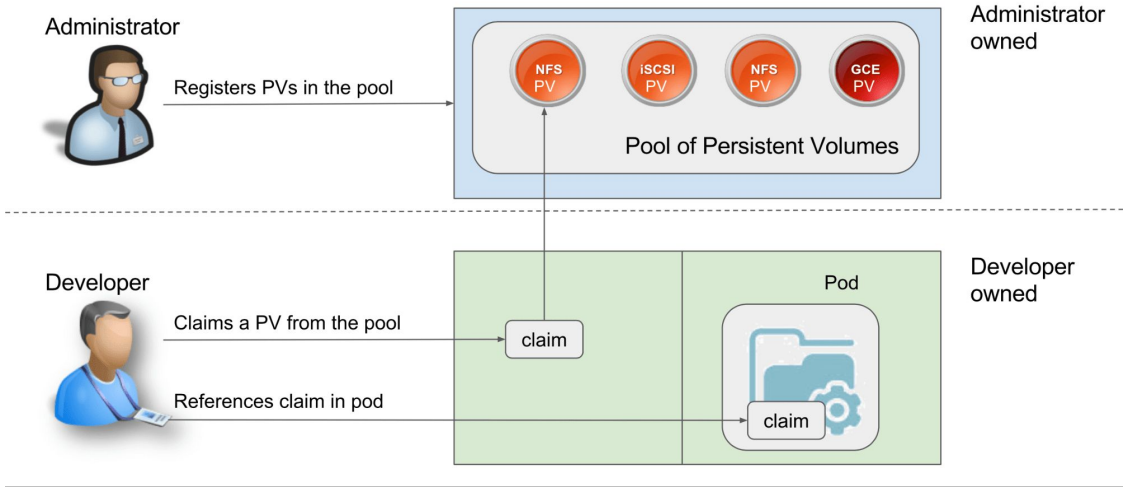
Workshop

ConfigMap & Secret

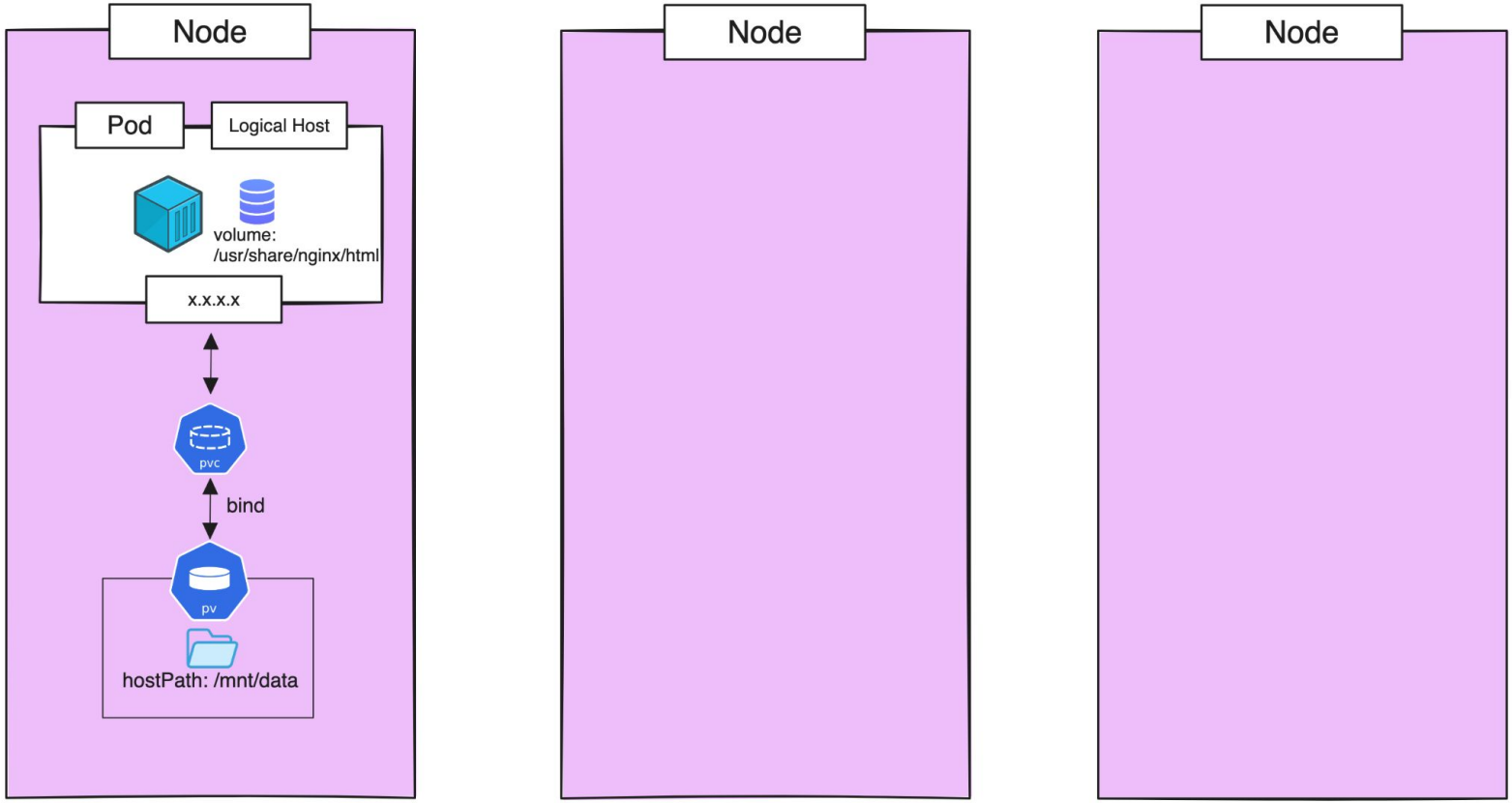
Persistent Volume

Persistent Volume

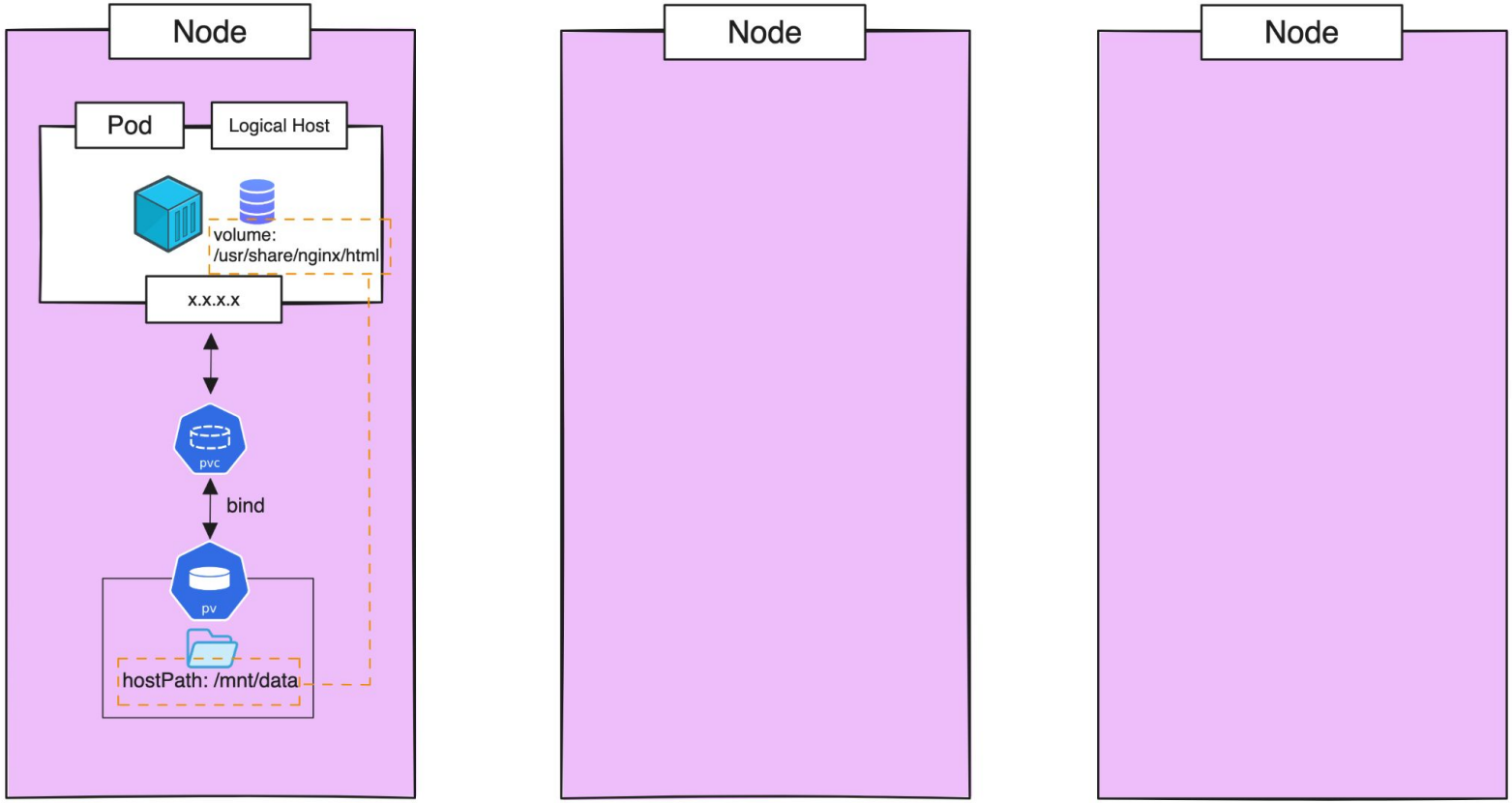
- Persistent Volume (PV) is a piece of storage in the cluster
- Can be provisioned from various sources like host local storage, cloud providers storage, or NFS.
- Persistent Volume Claims (PVC) allow a user to consume abstract storage resources
- A PVC to PV binding is a one-to-one mapping



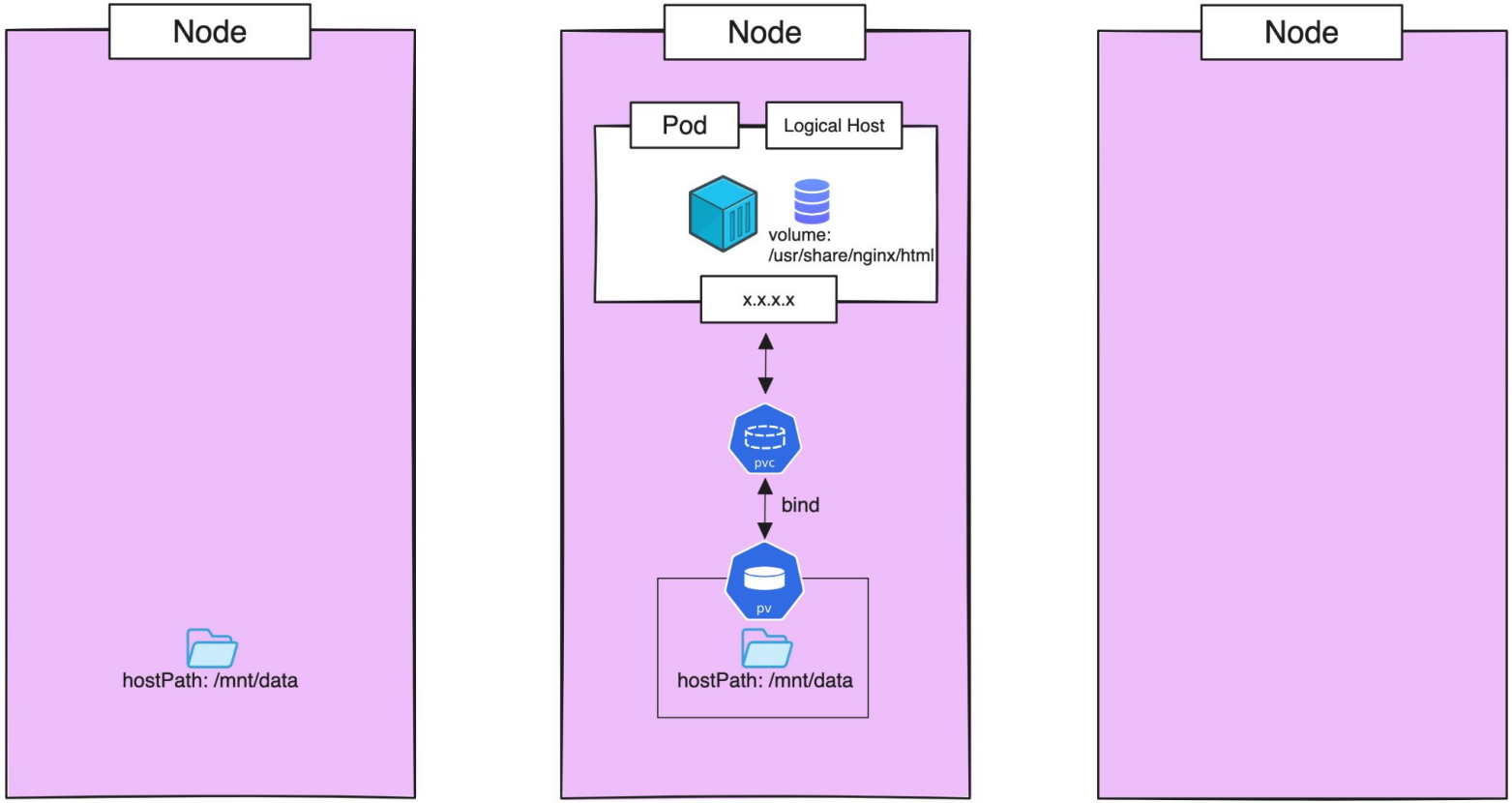
PV & PVC


Node
Pod
Logical Host
volume:
/usr/share/nginx/html
x.x.x.x
pvc
bind
pv
hostPath: /mnt/data
Node
Node

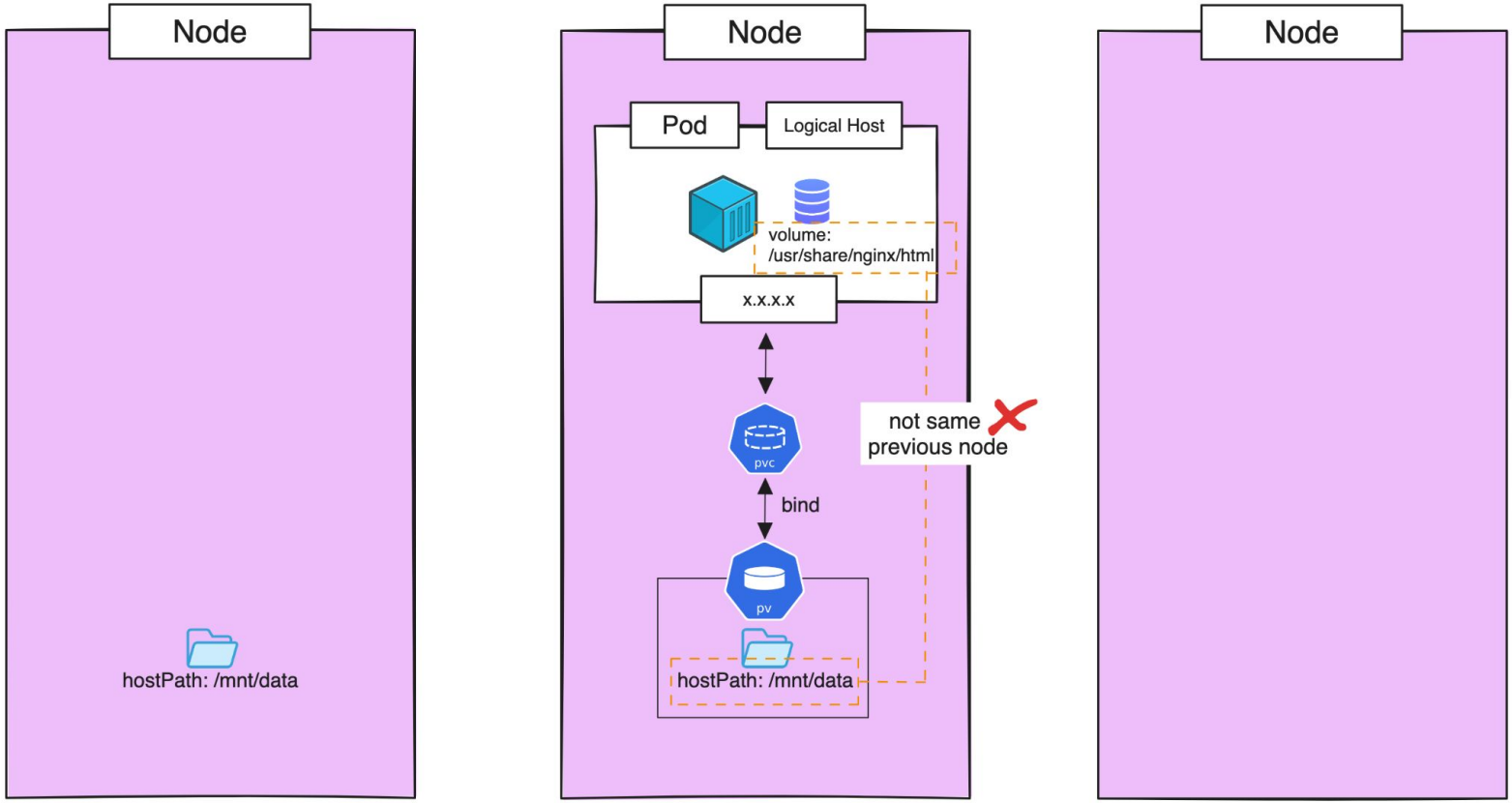
PV & PVC


Node
Pod
Logical Host
volume:
/usr/share/nginx/html
x.x.x.x
pvc
bind
pv
hostPath: /mnt/data
Node
Node

PV & PVC


Node
hostPath: /mnt/data
Node
Pod
Logical Host
volume:
/usr/share/nginx/html
x.x.x.x
pvc
bind
pv
hostPath: /mnt/data
Node

PV & PVC


Node
hostPath: /mnt/data
Node
Pod
Logical Host
volume:
/usr/share/nginx/html
x.x.x.x
pvc
bind
pv
hostPath: /mnt/data
not same
previous node
Node

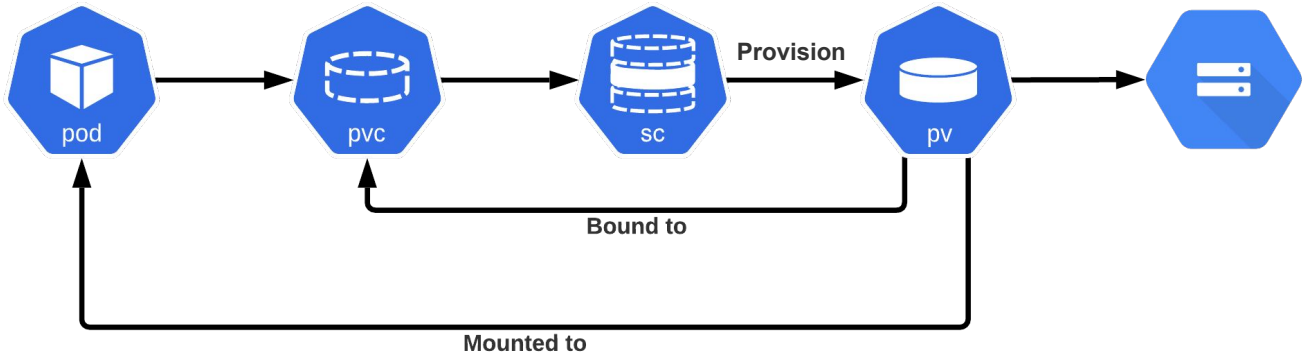
PV & PVC

Solutions

- Using selector node or nodeAffinity for ensure the target node is same
- Using shared storages like NFS or cloud storages

Storage Classes

- Template for Provisioning PersistentVolumes (PV)
- A StorageClass provides a way for administrators to describe the classes of storage
- Automatically provisions PV when defined PVC (Dynamic Volume Provisioning)



Workshop

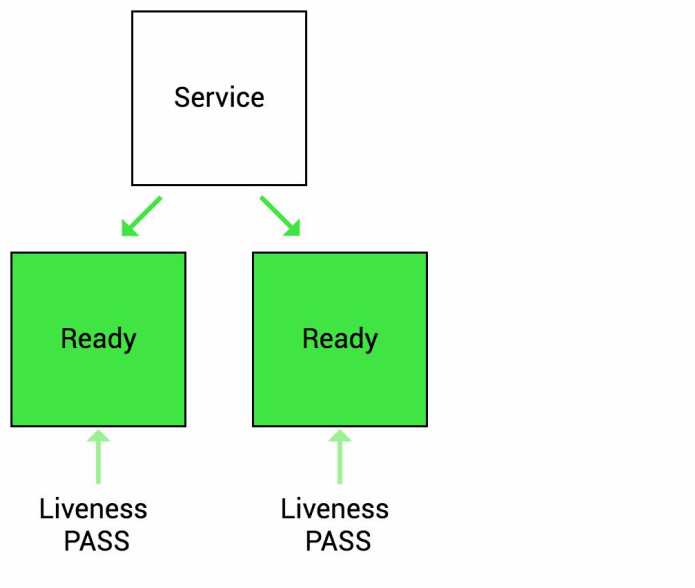
Persistent Volume

Day 3


Good to know

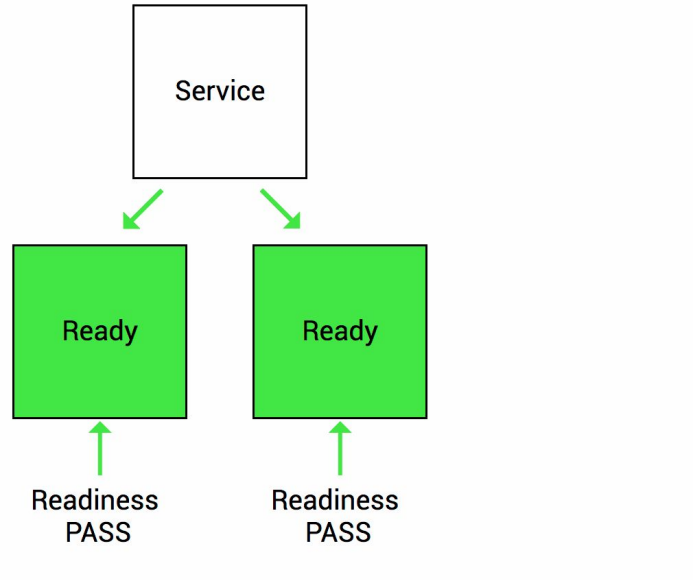
Pod liveness & readiness

- **Liveness**: Indicates whether the Container is running. and if for some reason the liveness probe fails, it restarts the container



Pod liveness & readiness

- **Readiness**: Indicates whether the Container is ready to service requests. If the condition inside readiness probe passes, only then our application can serve traffic.



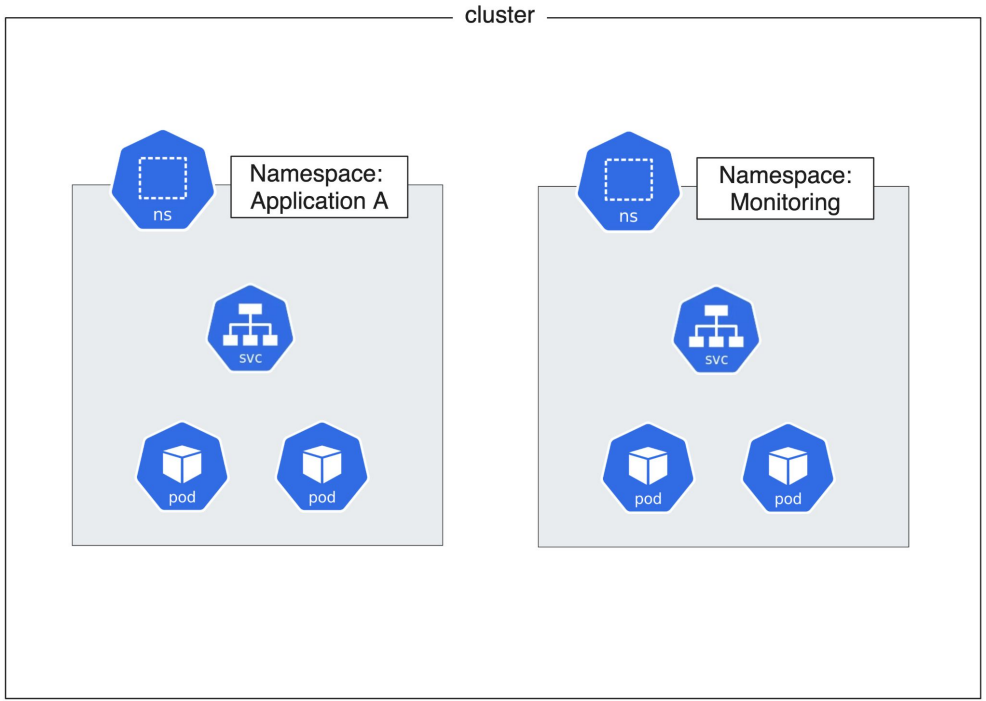
Namespace

Namespace

- Partition a single kubernetes cluster into multiples clusters
- 4 defaults namespaces
  - **kube-system**: core system processes
  - **kube-public**: public accessible data, ex: ConfigMaps & Secrets
  - **kube-node-lease**: heartbeat of nodes, so that the control plane can detect node failure
  - **default**: your resources/objects

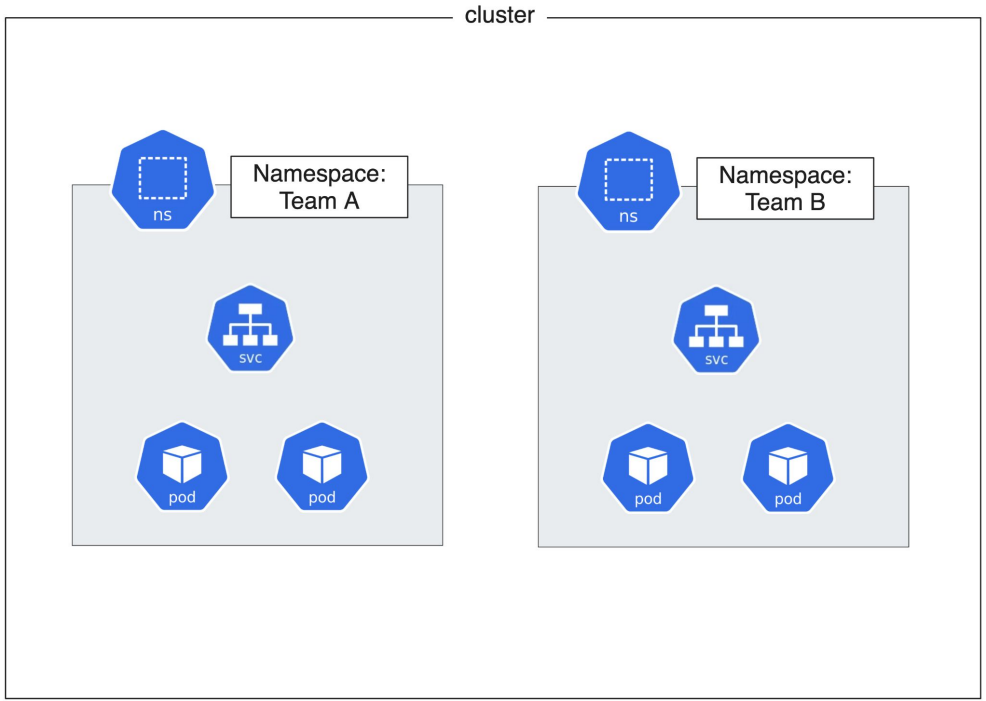
Namespace - use cases

- Application domain



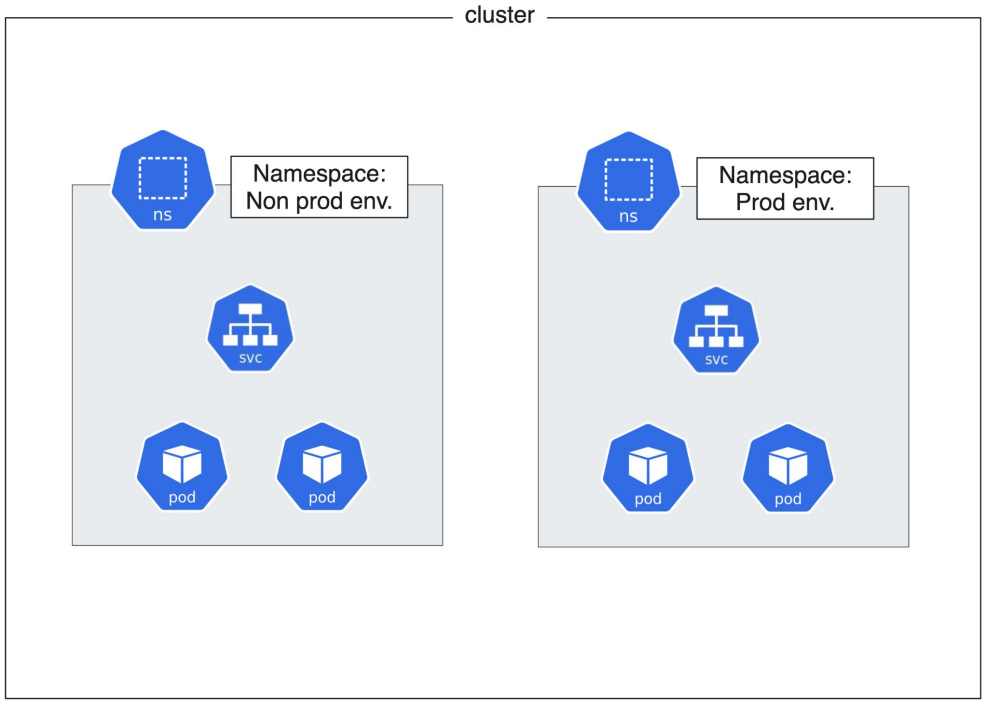
Namespace - use cases

- Teams driven



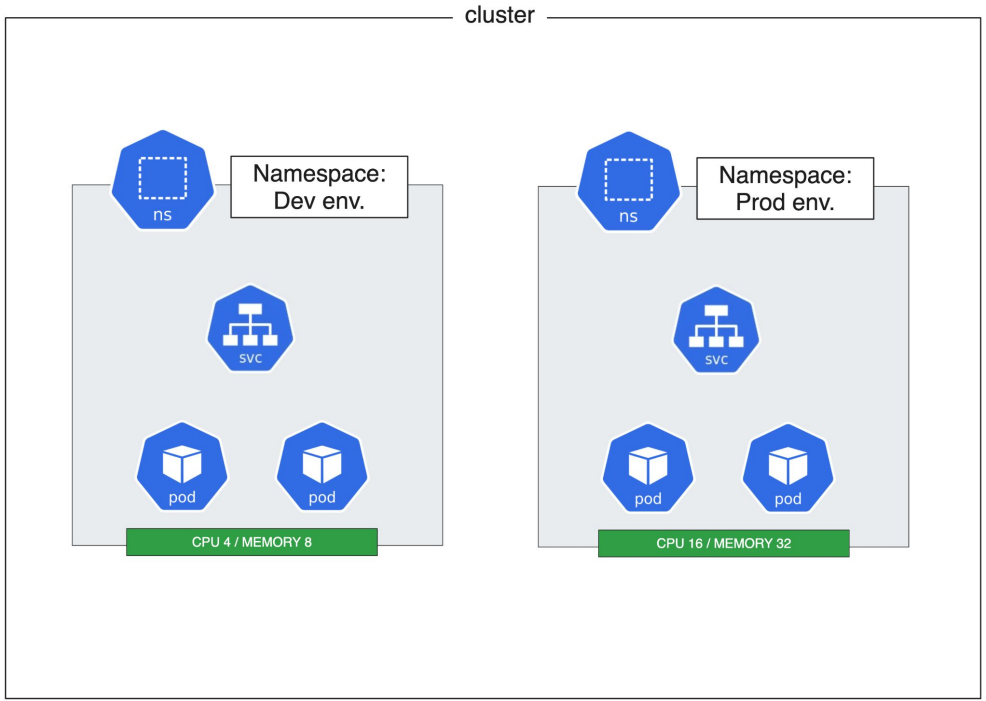
Namespace - use cases

- Environments



Namespace - use cases

- Resources quota limit



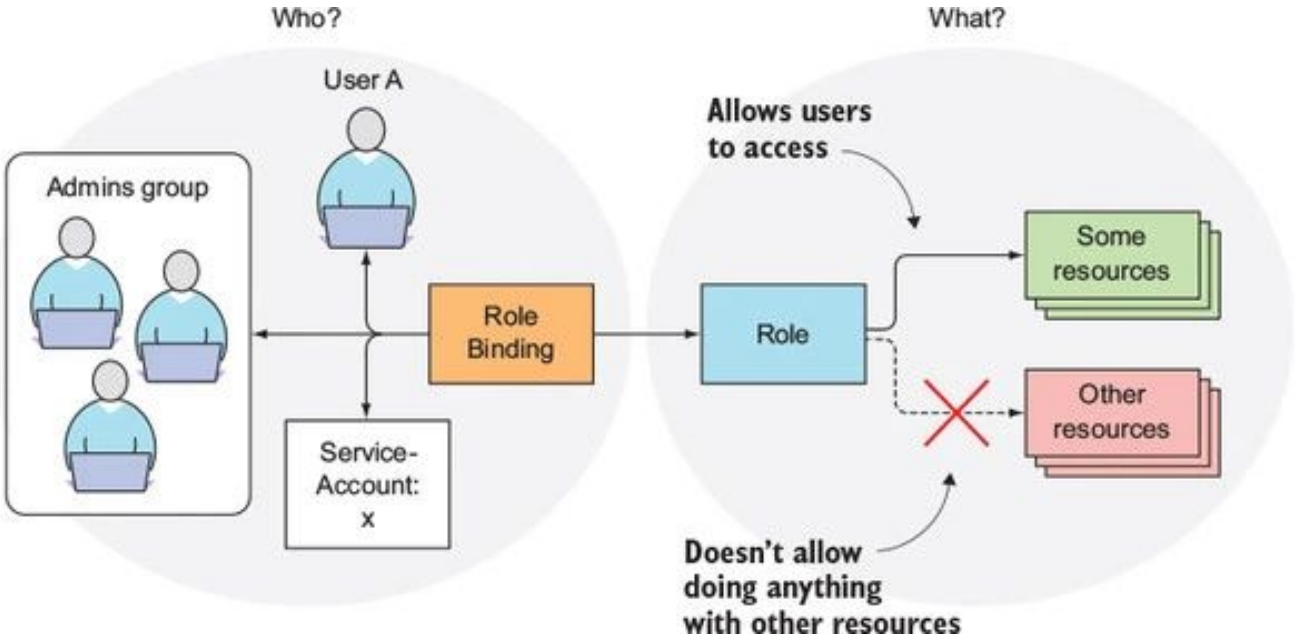
Workshop

Namespace


RBAC
(Role Based Access
Control)

RBAC

- Who ?
- How ?
- What ?



RBAC

Who ? (Kubernetes Subject)

1. User
2. Group
3. Service account

RBAC

How ? (Kubernetes Verb)

- get
- list
- watch
- create
- update
- patch
- delete

https://kubernetes.io/docs/reference/access-authn-authz/authorization/#determine-the-request-verb

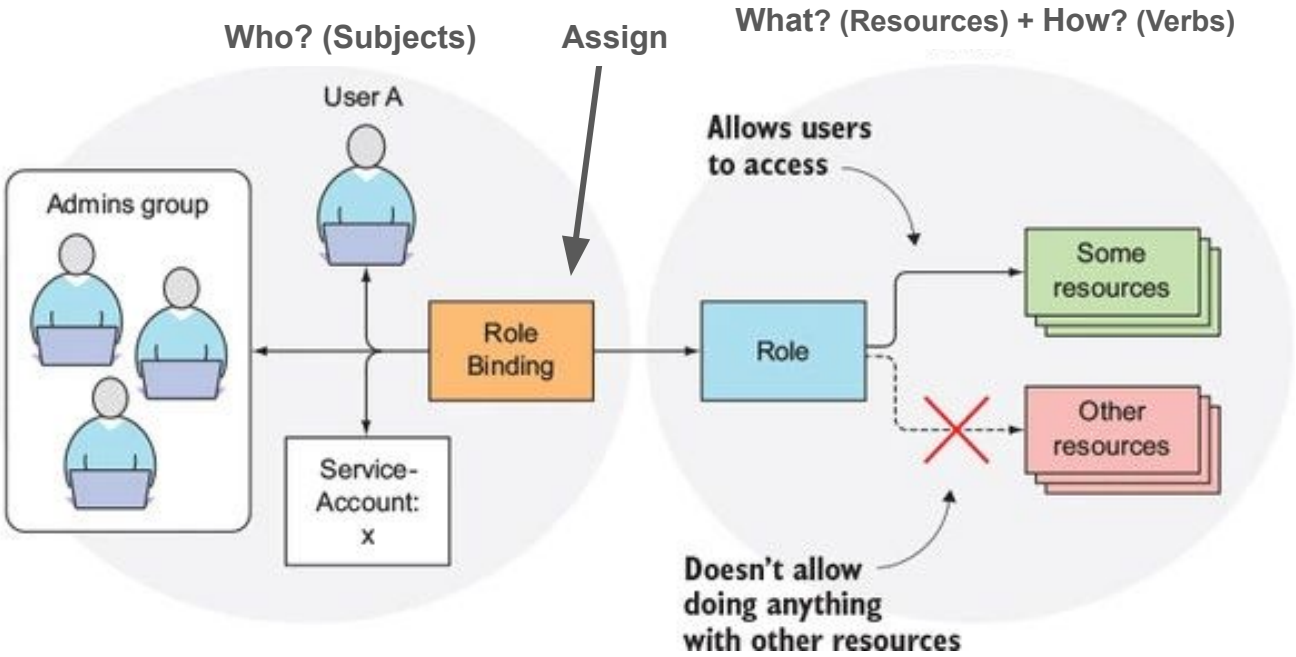
RBAC

What ? (Kubernetes Resource)

- pod
- deployment
- service
- etc.

https://kubernetes.io/docs/reference/kubectl/#resource-types

RBAC


Who? (Subjects)
Assign
What? (Resources) + How? (Verbs)
User A
Admins group
Role Binding
Service-Account: x
Allows users to access
Role
Some resources
Other resources
Doesn't allow doing anything with other resources

RBAC

Permission Scopes

- Role

  A Role can only be used to grant access to resources **within** a single namespace

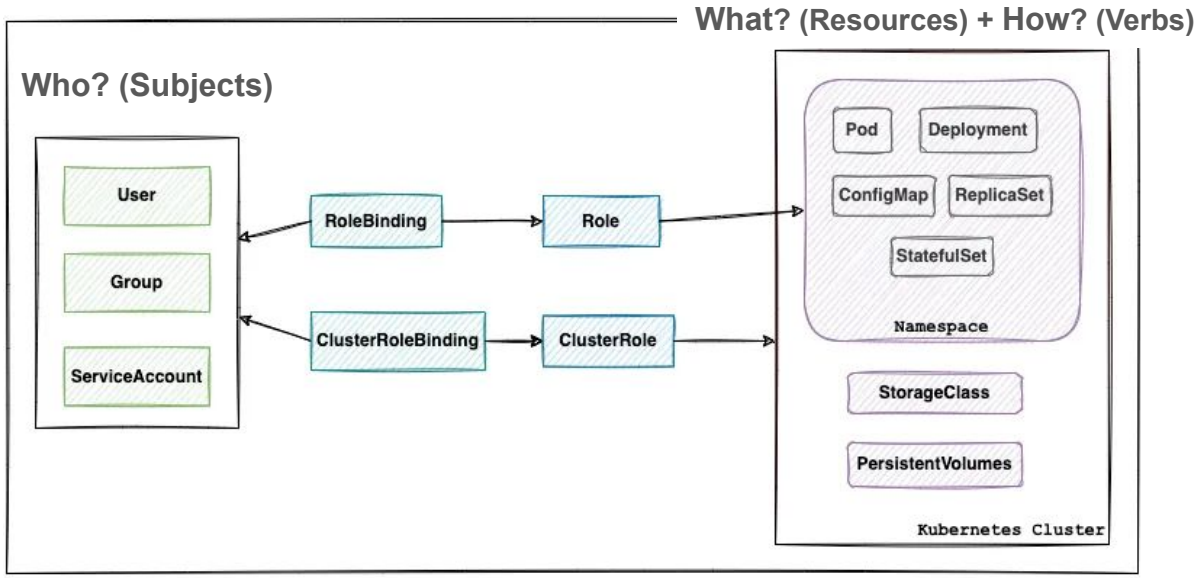
- ClusterRole

  A ClusterRole can be used to grant access to resources **across** all namespaces in the cluster

RBAC

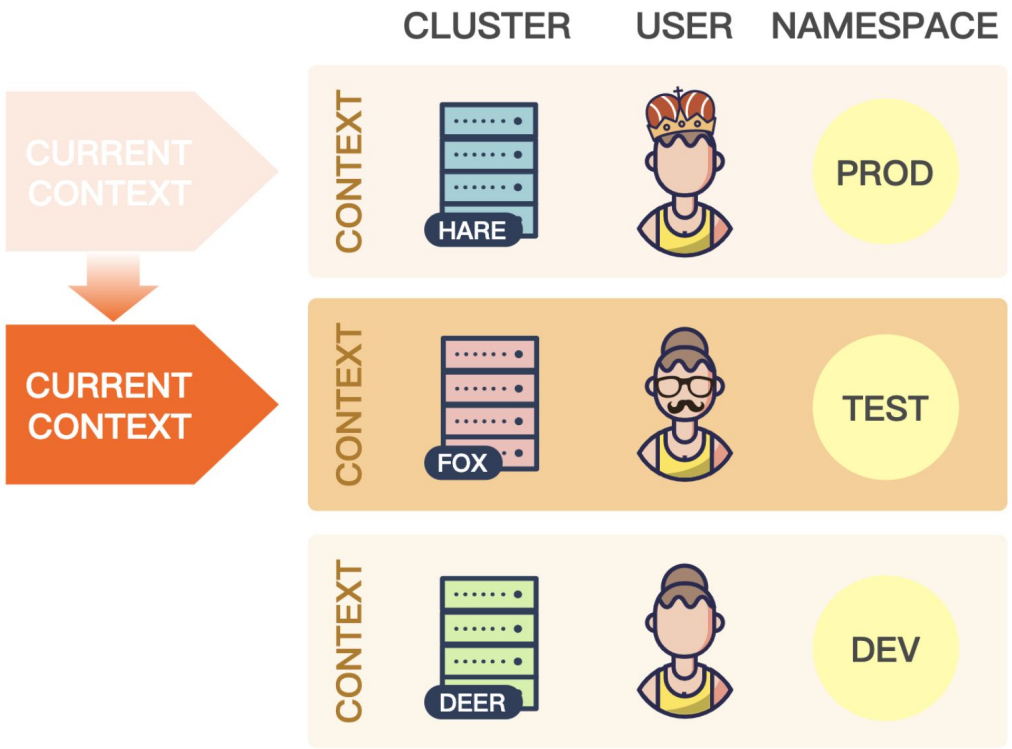
Binding

- RoleBinding: use with Role
- ClusterRoleBinding: use with ClusterRole



Context

- Access to Cluster(s)



Workshop

RBAC

HELM

HELM

HELM is the package manager for Kubernetes

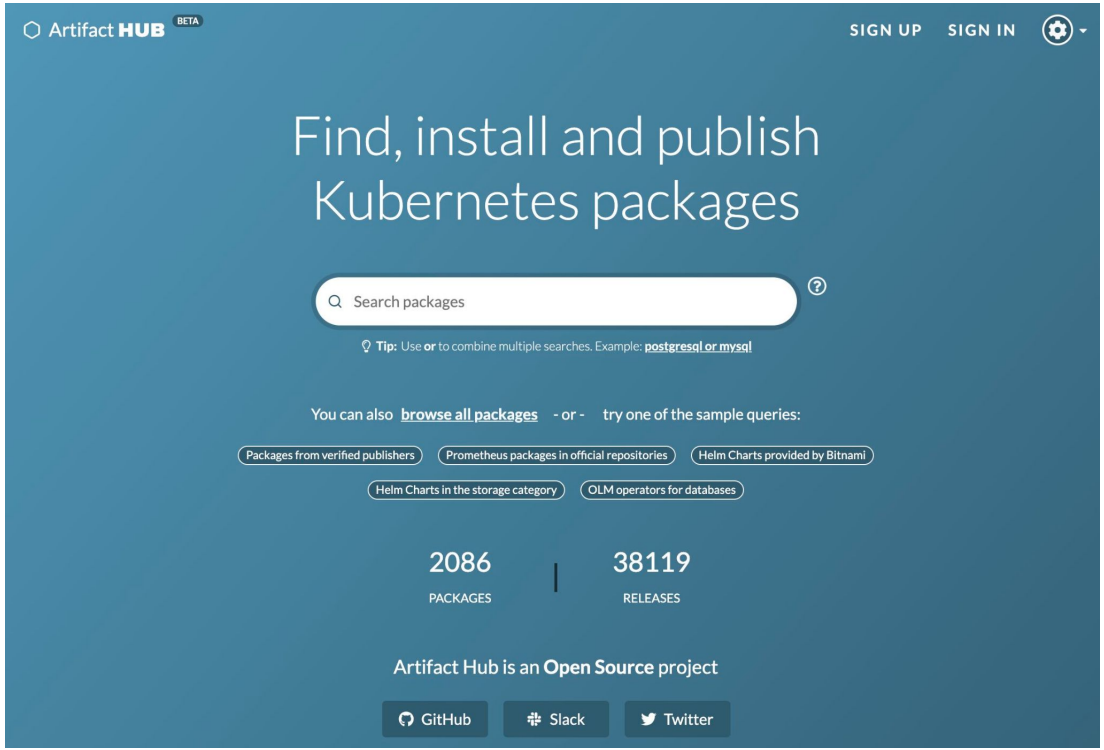
HELM

HELM is the package manager for Kubernetes

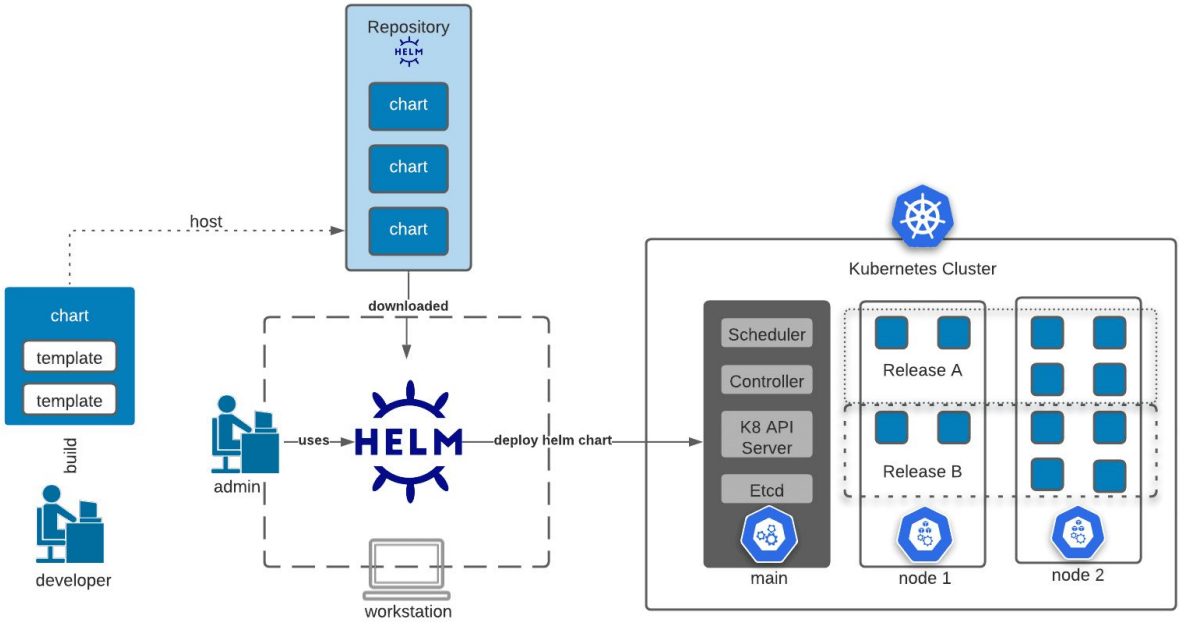
- Boosts productivity
- Reduces complexity
- Simple sharing
- Versioning and Upgrades

HELM Charts

- Public HELM Chart: https://artifacthub.io/



HELM Charts



Helm Workflow

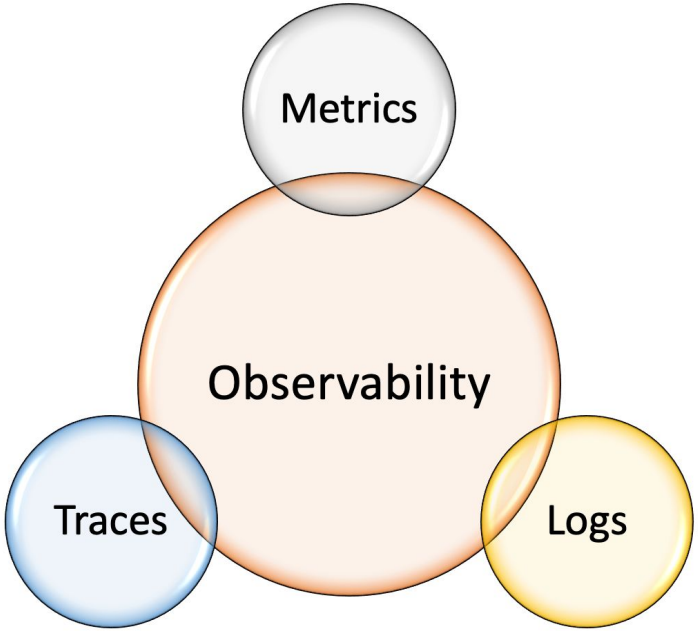
Workshop

HELM


MONITORING

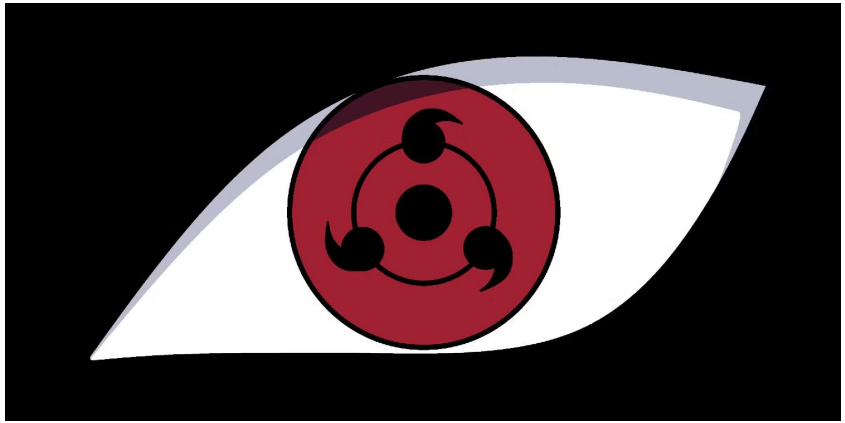
Three pillars of Observability

- Metrics
- Logs
- Traces

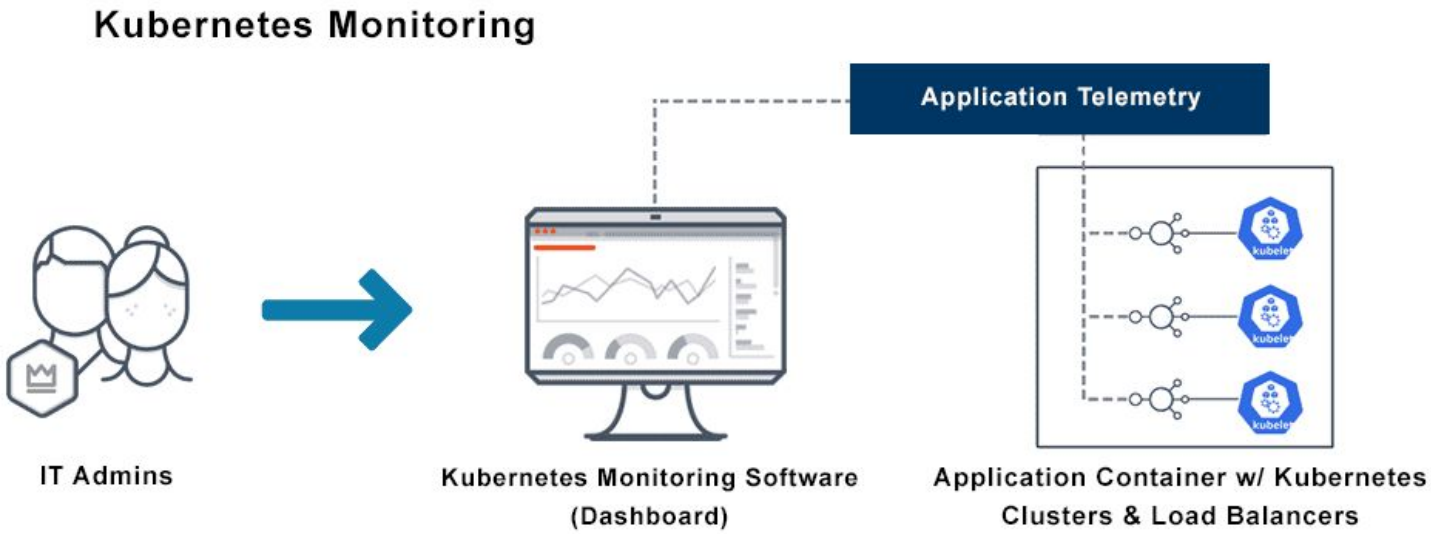


Three pillars of Observability

- Metrics
- Logs
- Traces



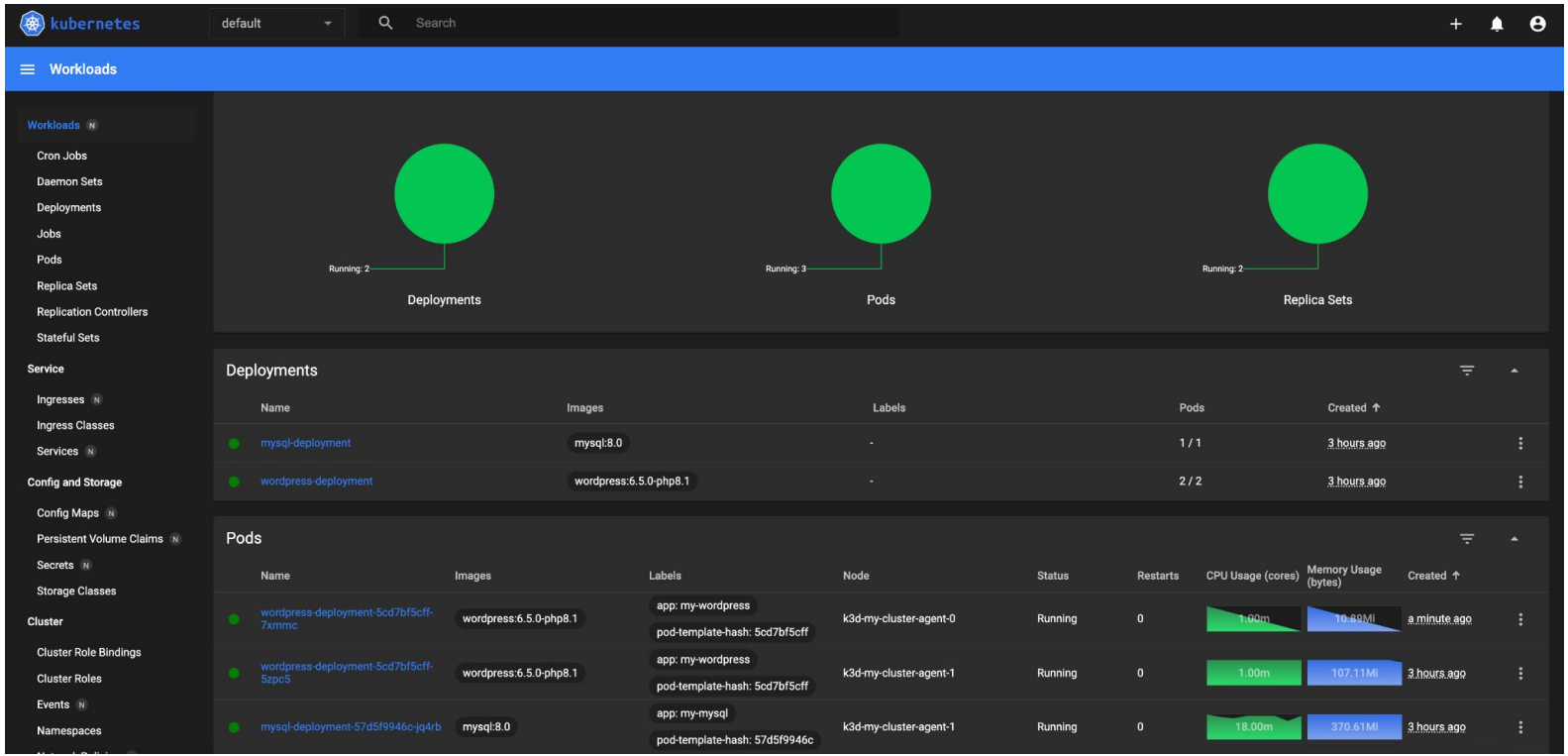
Metric Monitoring



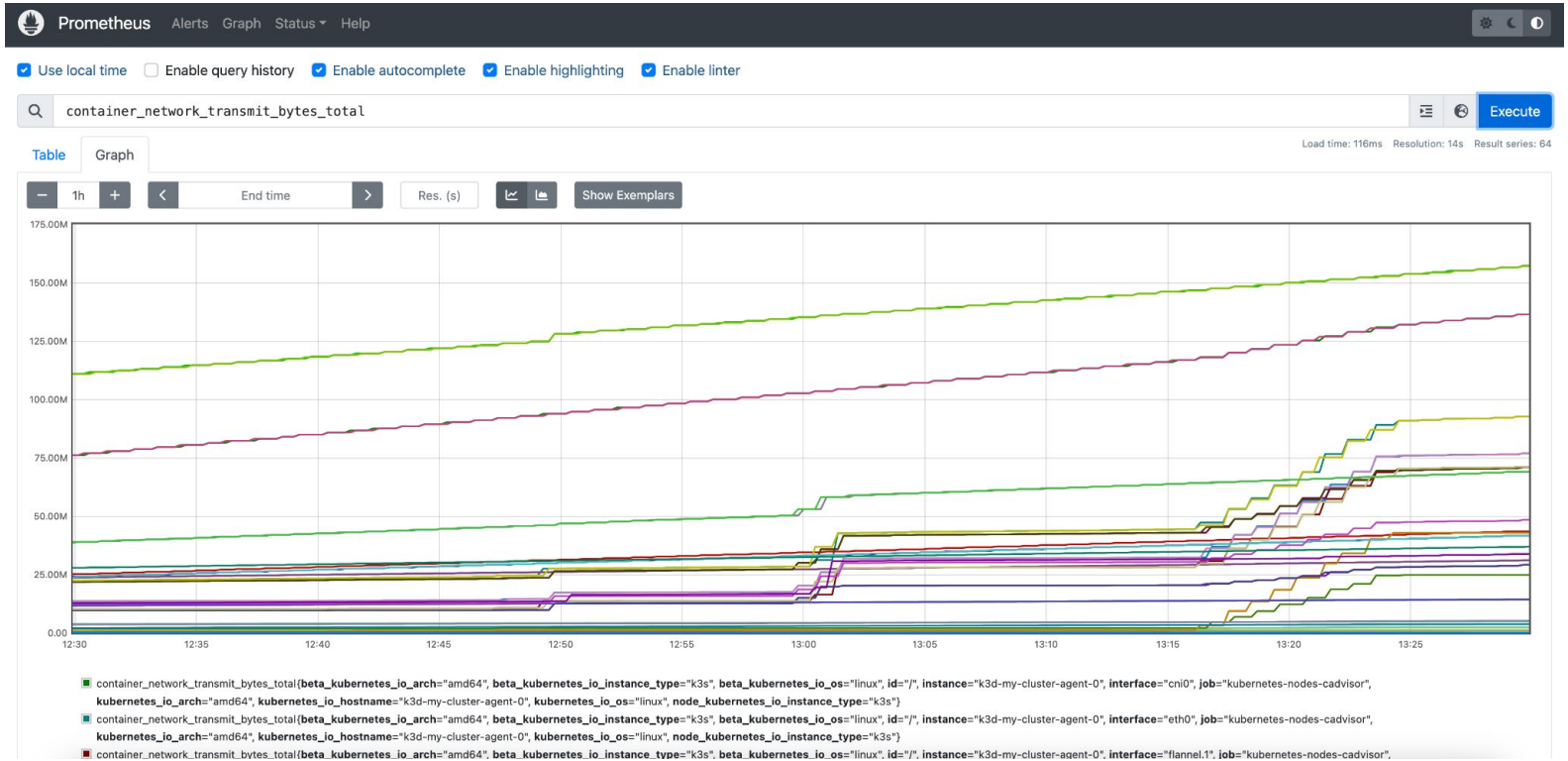
Metric Monitoring

- API request latency, the lower the better
- Cluster state metrics, including the availability and health of pods
- CPU utilization
- Disk utilization
- Node status, including disk or processor overload, memory, network availability, and readiness
- Pod availability (unavailable pods may indicate poorly designed readiness probes or configuration issues)

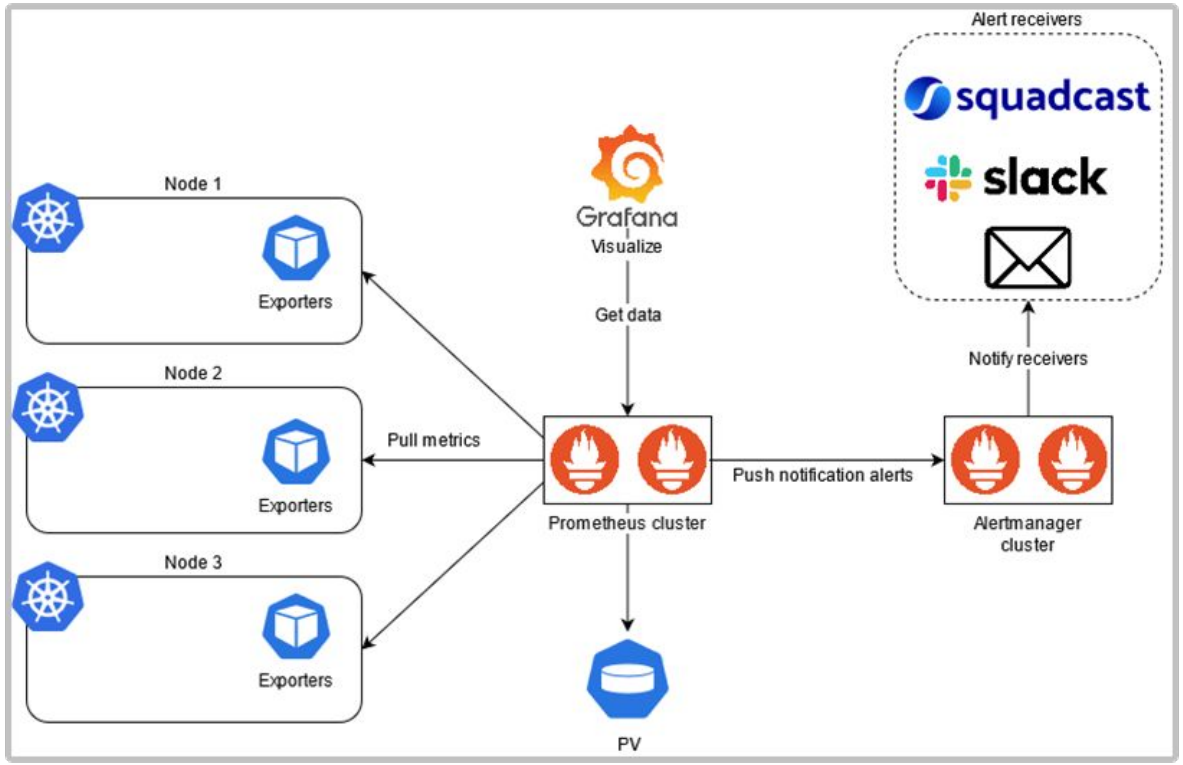
Kubernetes Dashboard



Prometheus


Prometheus
Alerts
Graph
Status
Help
Use local time
Enable query history
Enable autocomplete
Enable highlighting
Enable linter
container_network_transmit_bytes_total
Execute
Table
Graph
Load time: 116ms
Resolution: 14s
Result series: 64
1h
End time
Res. (s)
Show Exemplars
175.00M
150.00M
125.00M
100.00M
75.00M
50.00M
25.00M
0.00
12:30
12:35
12:40
12:45
12:50
12:55
13:00
13:05
13:10
13:15
13:20
13:25
container_network_transmit_bytes_total{beta_kubernetes_io_arch="amd64", beta_kubernetes_io_instance_type="k3s", beta_kubernetes_io_os="linux", id="/", instance="k3d-my-cluster-agent-0", interface="cni0", job="kubernetes-nodes-cadvisor", kubernetes_io_arch="amd64", kubernetes_io_hostname="k3d-my-cluster-agent-0", kubernetes_io_os="linux", node_kubernetes_io_instance_type="k3s"}
container_network_transmit_bytes_total{beta_kubernetes_io_arch="amd64", beta_kubernetes_io_instance_type="k3s", beta_kubernetes_io_os="linux", id="/", instance="k3d-my-cluster-agent-0", interface="eth0", job="kubernetes-nodes-cadvisor", kubernetes_io_arch="amd64", kubernetes_io_hostname="k3d-my-cluster-agent-0", kubernetes_io_os="linux", node_kubernetes_io_instance_type="k3s"}
container_network_transmit_bytes_total{beta_kubernetes_io_arch="amd64", beta_kubernetes_io_instance_type="k3s", beta_kubernetes_io_os="linux", id="/", instance="k3d-my-cluster-agent-0", interface="flannel.1", job="kubernetes-nodes-cadvisor",

Prometheus Alert



Prometheus Alert Rule Examples

- https://samber.github.io/awesome-prometheus-alerts/rules

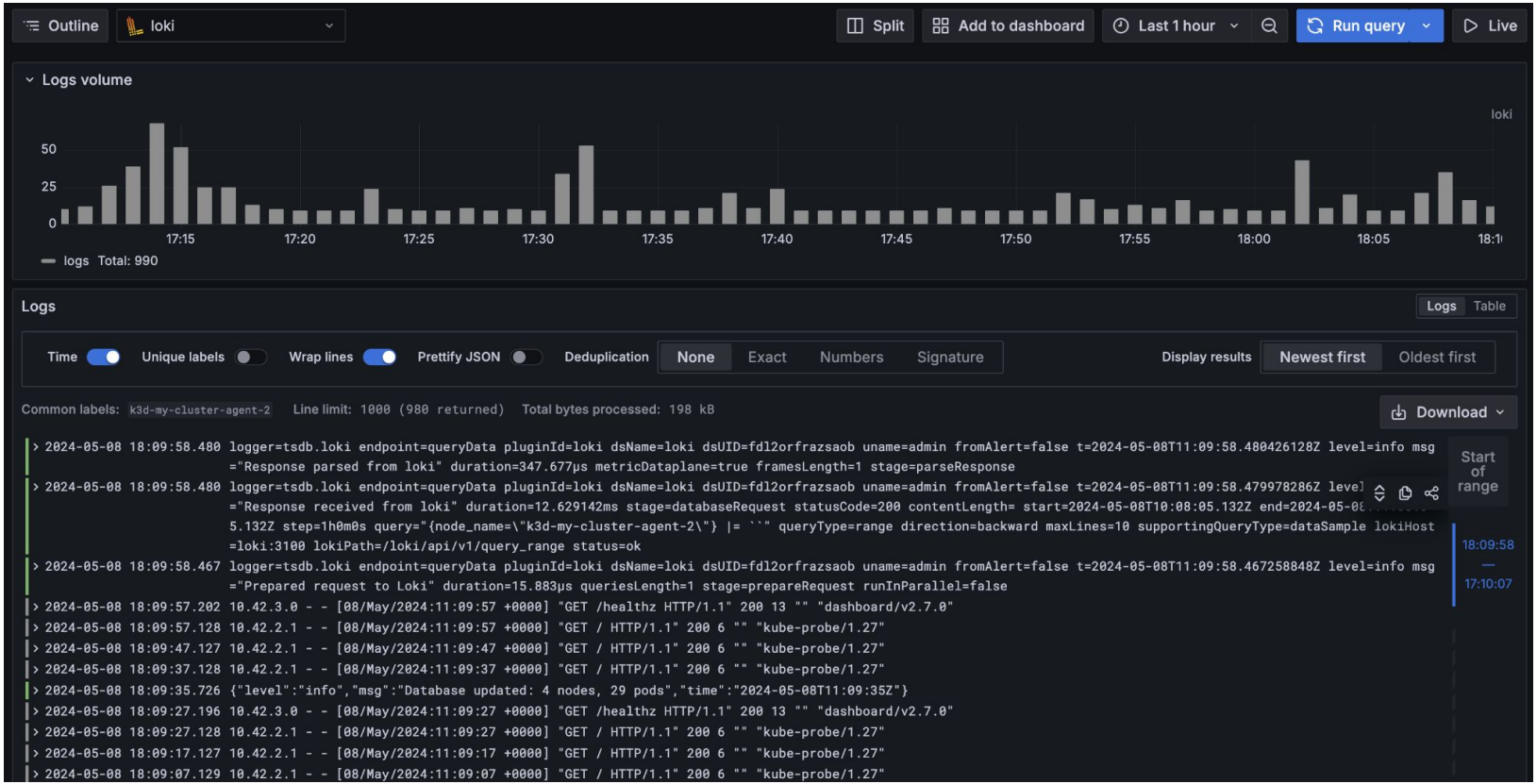
Grafana


Search or jump to...
cmd+k
Home
Dashboards
Kubernetes cluster monitoring (via Prometheus)02
Add
Share
Last 15 minutes
10s
Home
Starred
Dashboards
Playlists
Snapshots
Library panels
Public dashboards
Explore
Alerting
Alert rules
Contact points
Notification policies
Silences
Groups
Admin
Connections
Add new connection
Data sources
Administration
General
Plugins and data
datasource
prometheus
Node
All
Network I/O pressure
Network I/O pressure
20 MB/s
10 MB/s
0 B/s
-10 MB/s
-20 MB/s
13:14
13:15
13:16
13:17
13:18
13:19
13:20
13:21
13:22
13:23
13:24
13:25
13:26
13:27
13:28
Total usage
Cluster memory usage
11.3%
Cluster CPU usage (2m avg)
1.00%
Cluster filesystem usage
11.58%
Used
2.17 GiB
Total
19.20 GiB
Used
0.64
Total
64.00
Used
N/A
Total
N/A

Logs Monitoring

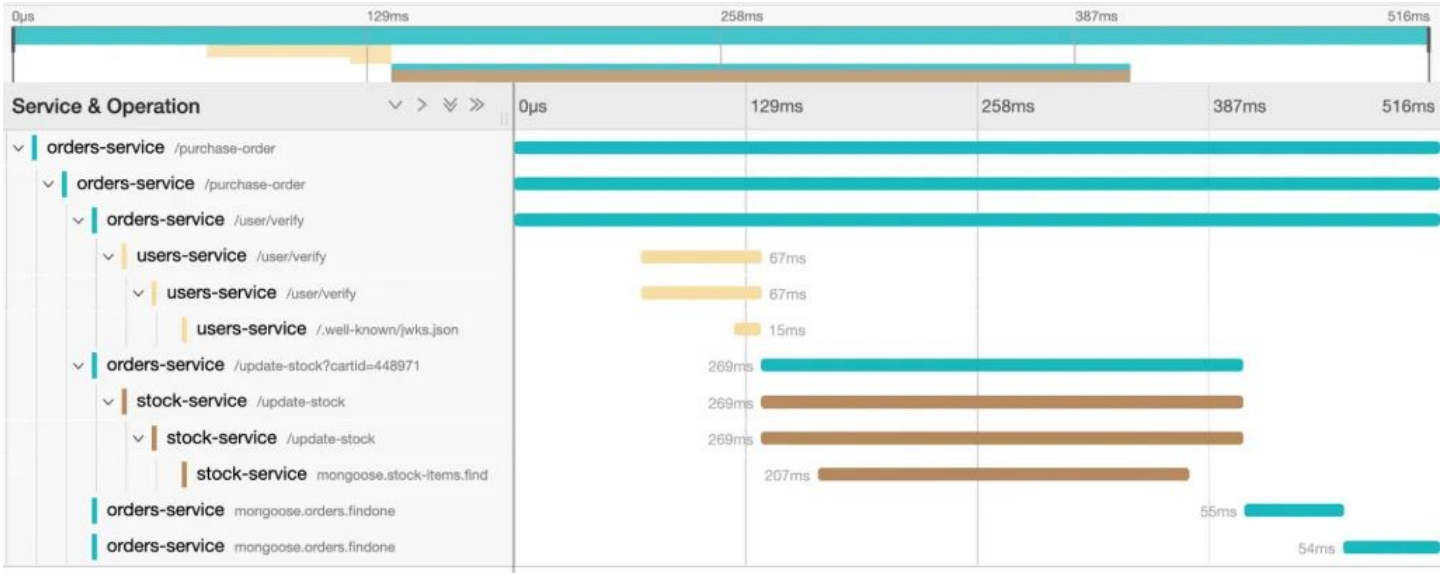
- Cluster logs
- Node logs
- Container logs
- Kubernetes Events

Loki

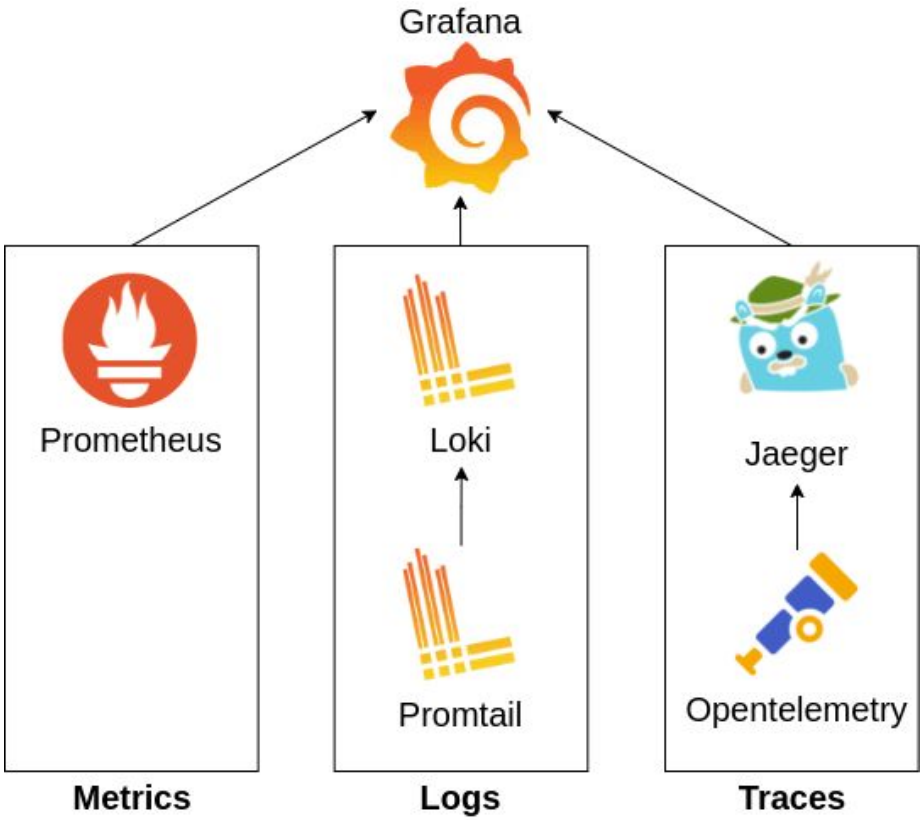

Outline
loki
Split
Add to dashboard
Last 1 hour
Run query
Live
Logs volume
loki
50
25
0
17:15
17:20
17:25
17:30
17:35
17:40
17:45
17:50
17:55
18:00
18:05
logs Total: 990
Logs
Logs
Table
Time
Unique labels
Wrap lines
Prettify JSON
Deduplication
None
Exact
Numbers
Signature
Display results
Newest first
Oldest first
Common labels: k3d-my-cluster-agent-2
Line limit: 1000 (980 returned)
Total bytes processed: 198 kB
Download
› 2024-05-08 18:09:58.480 logger=tsdb.loki endpoint=queryData pluginId=loki dsName=loki dsUID=fdl2orfrazsaob uname=admin fromAlert=false t=2024-05-08T11:09:58.480426128Z level=info msg="Response parsed from loki" duration=347.677µs metricDataplane=true framesLength=1 stage=parseResponse
› 2024-05-08 18:09:58.480 logger=tsdb.loki endpoint=queryData pluginId=loki dsName=loki dsUID=fdl2orfrazsaob uname=admin fromAlert=false t=2024-05-08T11:09:58.479978286Z level="Response received from loki" duration=12.629142ms stage=databaseRequest statusCode=200 contentLength= start=2024-05-08T10:08:05.132Z end=2024-05-08T11:09:05.132Z step=1h0m0s query="{node_name=\"k3d-my-cluster-agent-2\"} |= `` queryType=range direction=backward maxLines=10 supportingQueryType=dataSample lokiHost=loki:3100 lokiPath=/loki/api/v1/query_range status=ok
› 2024-05-08 18:09:58.467 logger=tsdb.loki endpoint=queryData pluginId=loki dsName=loki dsUID=fdl2orfrazsaob uname=admin fromAlert=false t=2024-05-08T11:09:58.467258848Z level=info msg="Prepared request to Loki" duration=15.883µs queriesLength=1 stage=prepareRequest runInParallel=false
› 2024-05-08 18:09:57.202 10.42.3.0 - - [08/May/2024:11:09:57 +0000] "GET /healthz HTTP/1.1" 200 13 "" "dashboard/v2.7.0"
› 2024-05-08 18:09:57.128 10.42.2.1 - - [08/May/2024:11:09:57 +0000] "GET / HTTP/1.1" 200 6 "" "kube-probe/1.27"
› 2024-05-08 18:09:47.127 10.42.2.1 - - [08/May/2024:11:09:47 +0000] "GET / HTTP/1.1" 200 6 "" "kube-probe/1.27"
› 2024-05-08 18:09:37.128 10.42.2.1 - - [08/May/2024:11:09:37 +0000] "GET / HTTP/1.1" 200 6 "" "kube-probe/1.27"
› 2024-05-08 18:09:35.726 {"level":"info","msg":"Database updated: 4 nodes, 29 pods","time":"2024-05-08T11:09:35Z"}
› 2024-05-08 18:09:27.196 10.42.3.0 - - [08/May/2024:11:09:27 +0000] "GET /healthz HTTP/1.1" 200 13 "" "dashboard/v2.7.0"
› 2024-05-08 18:09:27.128 10.42.2.1 - - [08/May/2024:11:09:27 +0000] "GET / HTTP/1.1" 200 6 "" "kube-probe/1.27"
› 2024-05-08 18:09:17.127 10.42.2.1 - - [08/May/2024:11:09:17 +0000] "GET / HTTP/1.1" 200 6 "" "kube-probe/1.27"
› 2024-05-08 18:09:07.129 10.42.2.1 - - [08/May/2024:11:09:07 +0000] "GET / HTTP/1.1" 200 6 "" "kube-probe/1.27"
Start of range
18:09:58
—
17:10:07

Tracing Monitoring

- A trace represents the entire journey of a request or action as it moves through all the nodes of a distributed system



Monitoring


Grafana
Prometheus
Loki
Promtail
Jaeger
Opentelemetry
Metrics
Logs
Traces

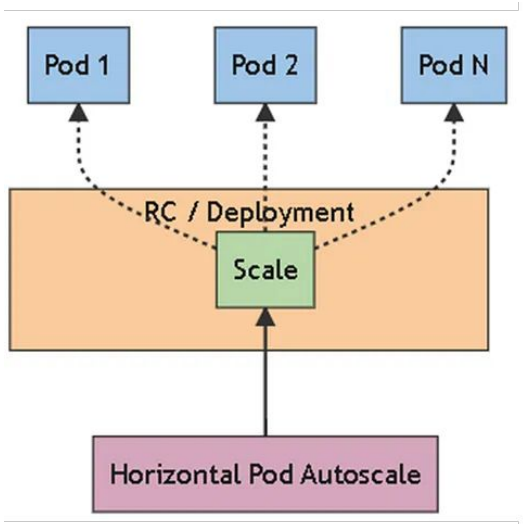
Workshop

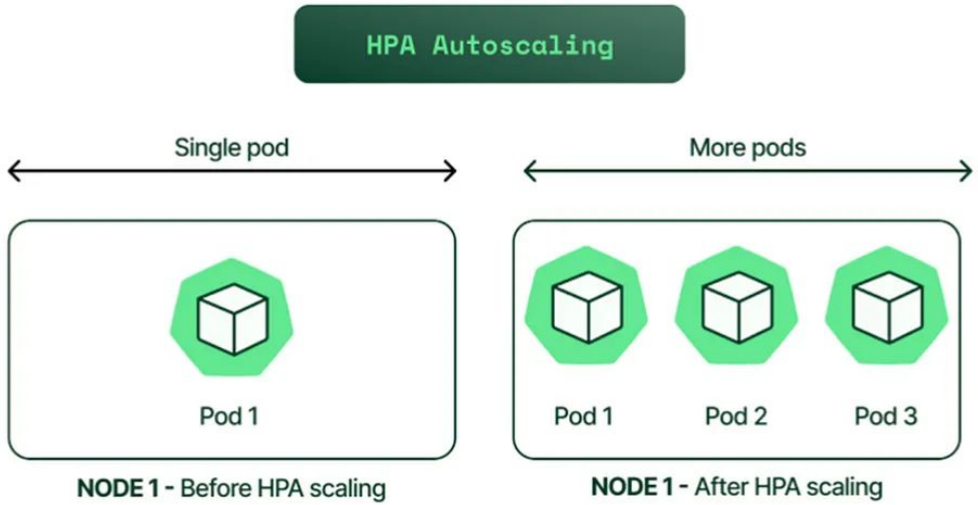
Monitoring

HPA (Horizontal Pod Autoscaler)

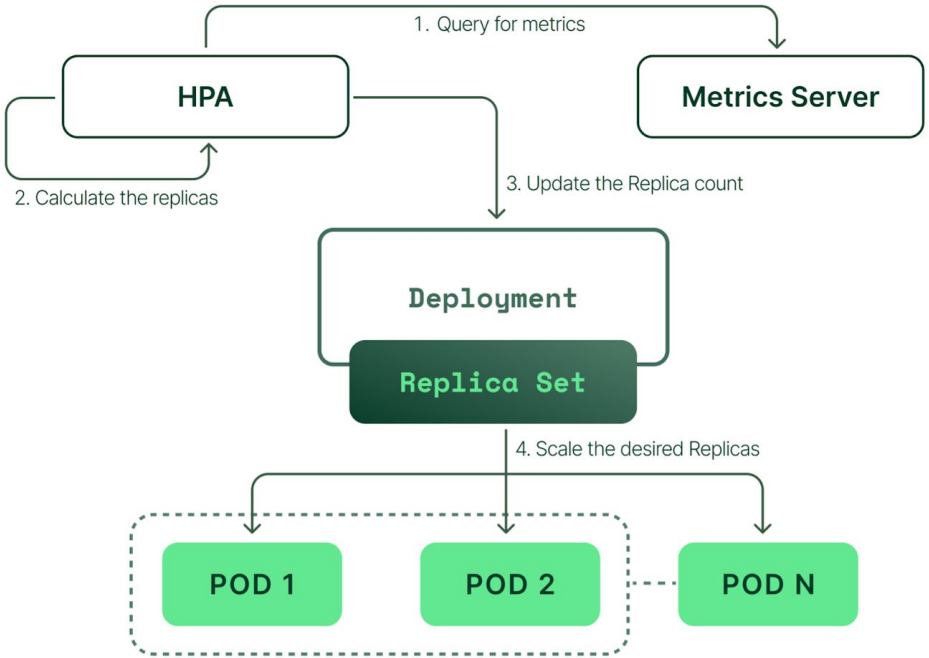
HPA

- A HorizontalPodAutoscaler (HPA for short) automatically updates a workload resource, with the aim of automatically scaling the workload to match demand





How does HPA work?



Workshop

HPA


Deployment &
Pipelines

Deployment strategies

Deployment strategies

- Recreate deployment
- Ramped (Rolling deployment)
- Blue/green deployment
- Canary deployment
- A/B testing
- Shadow deployment

https://thenewstack.io/deployment-strategies/

Recreate deployment

- Terminating all the running instances then recreate them with the newer version

Advantage

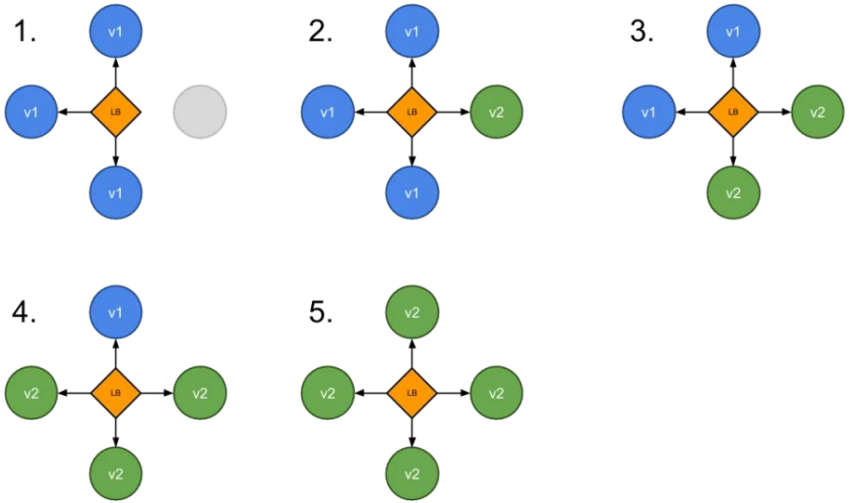
- Application state entirely renewed

Disadvantage

- Downtime that depends on both shutdown and boot duration of the application

Ramped (Rolling deployment)

- The rolling update is the **default** deployment strategy in Kubernetes that update the pods one by one without causing cluster downtime. the deployment objects create new pods using the new pod specifications and shut down the older ones when the application starts running successfully



Ramped (Rolling deployment)

Advantages

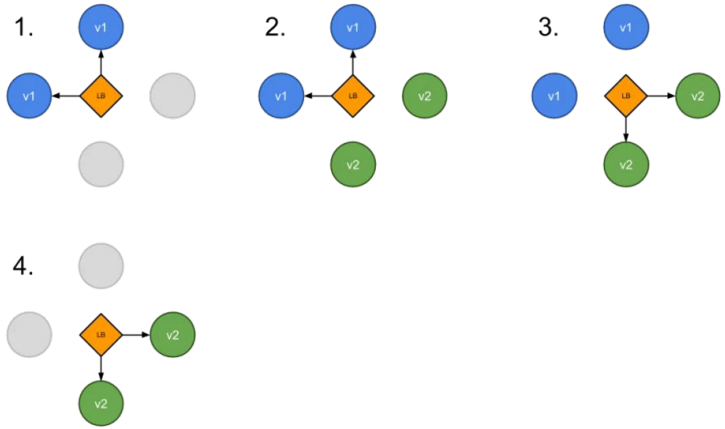
- Enables zero-downtime deployments
- Well-suited for large deployments and stateful applications

Disadvantages

- Rollout/rollback can take time
- Supporting multiple APIs is hard
- No control over traffic

Blue/green deployment

- While the “Blue” version is running, engineers deploy the “green” version in another deployment and run basic tests to ensure that the pods within the deployments are stable, then using services, you switch to the newer version of the application
- Best to avoid API versioning issues



Blue/green deployment

Advantages

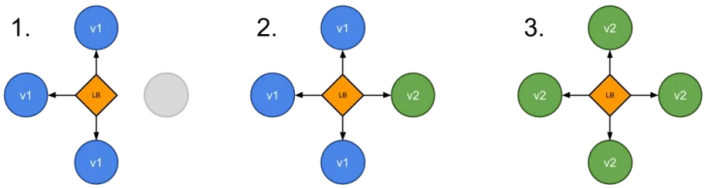
- Instant rollout/rollback
- Clear separation between old and new versions

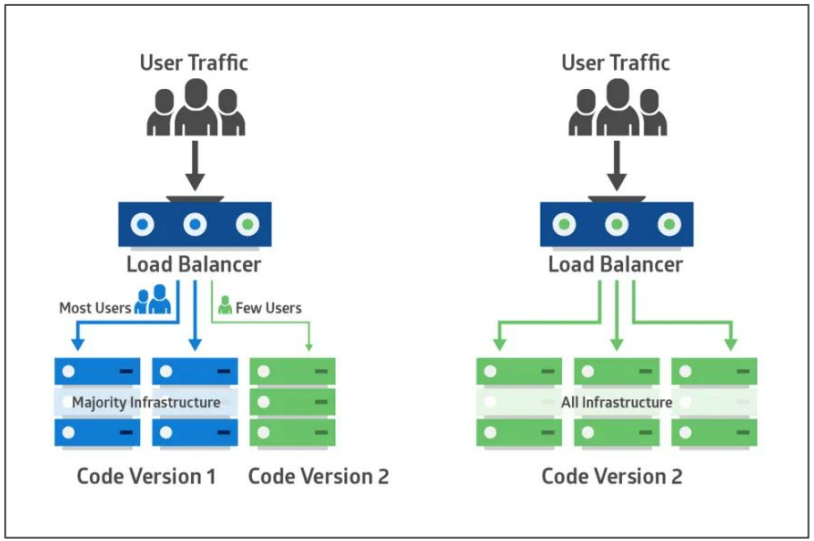
Disadvantages

- Requires double the resources
- More complex configuration

Canary deployment

- Similar to blue/green deployments, but only a **small subset** of traffic is directed to the new version (canary) initially. This allows for early detection of issues before a full rollout





Canary deployment

Advantages

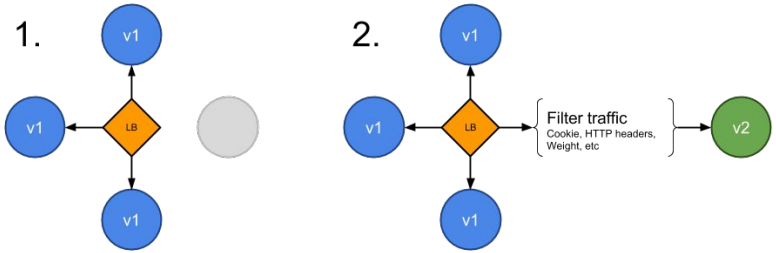
- Lowers risk by gradually rolling out new version to a small subset of users
- Enables early detection of issues

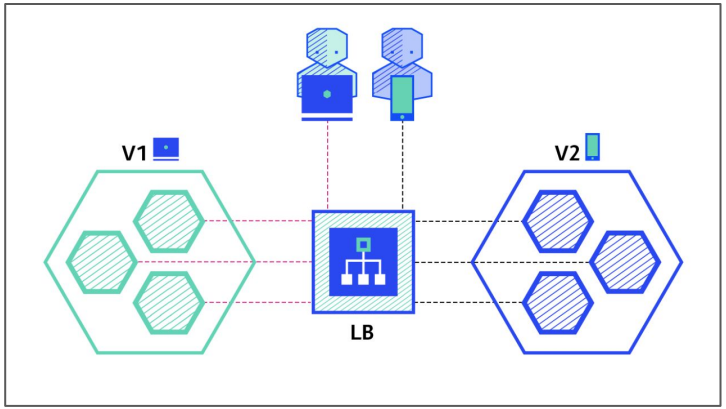
Disadvantages

- Requires additional configuration for traffic routing
- Slow rollout, most of users cannot use new versions
- Application must support both of old & new versions

A/B testing

- A/B testing deployments consists of routing a subset of users to a new functionality under **specific conditions**. It is usually a technique for making business decisions based on statistics, rather than a deployment strategy.
- Best for feature testing on a subset of users





A/B testing

Advantages

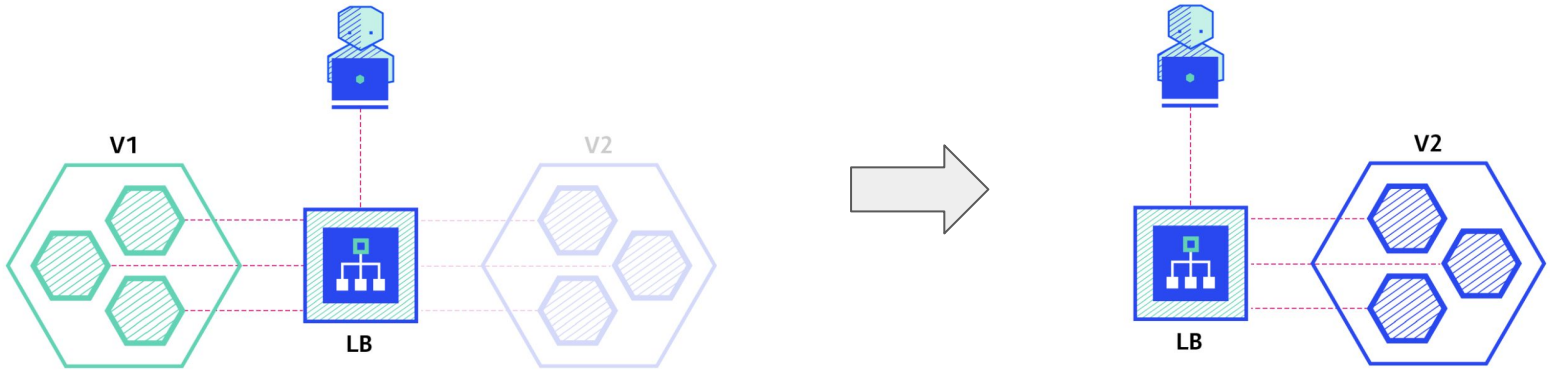
- Several versions run in parallel
- Full control over the traffic distribution

Disadvantages

- Requires intelligent load balancer
- Not straightforward, you need to setup additional tools
- Hard to troubleshoot errors for a given session

Shadow deployment

- Consists of releasing version B alongside version A, fork version A's incoming requests and send them to version B as well without impacting production traffic. This is particularly useful to test production load on a new feature



Shadow deployment

Advantages

- Performance testing, auto scale of the application with production traffic,
- No impact on the user

Disadvantages

- Expensive as it requires double the resources.
- Complex to setup

DEPLOYMENT STRATEGIES

When it comes to production, a ramped or blue/green deployment is usually a good fit, but proper testing of the new platform is necessary.

Blue/green and shadow strategies have more impact on the budget as it requires double resource capacity. If the application lacks in tests or if there is little confidence about the impact/stability of the software, then a canary, a/b testing or shadow release can be used.

If your business requires testing of a new feature amongst a specific pool of users that can be filtered depending on some parameters like geolocation, language, operating system or browser features, then you may want to use the a/b testing technique.

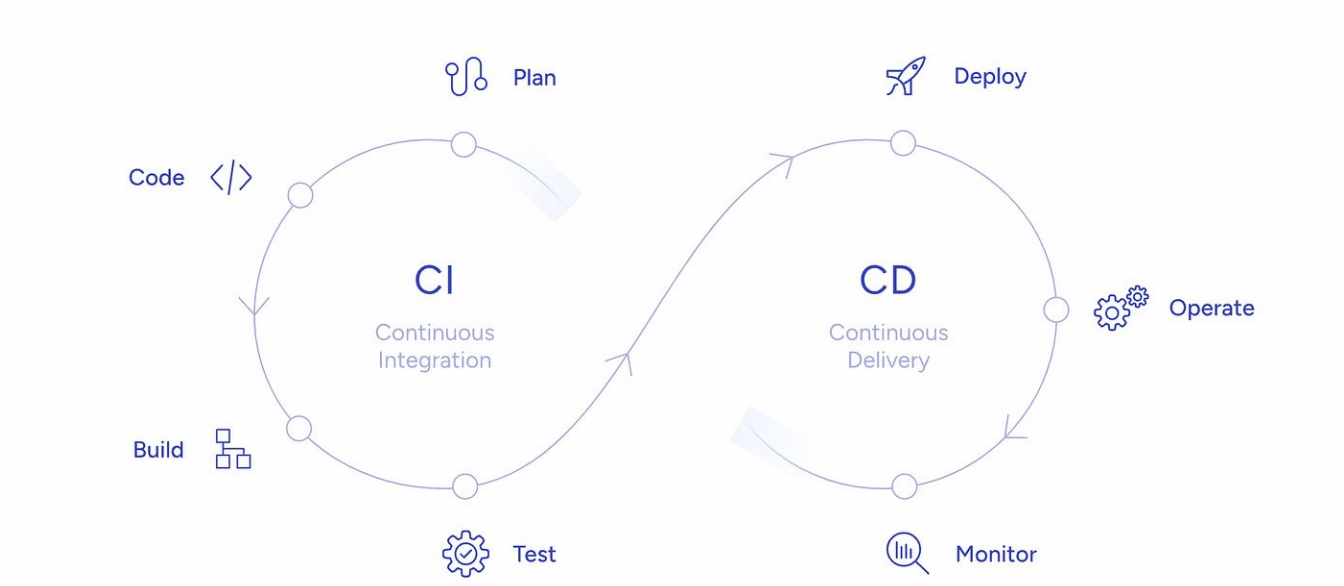
Container Solutions

Strategy	ZERO DOWNTIME	REAL TRAFFIC TESTING	TARGETED USERS	CLOUD COST	ROLLBACK DURATION	NEGATIVE IMPACT ON USER	COMPLEXITY OF SETUP
RECREATE version A is terminated then version B is rolled out	✗	✗	✗	■□□	■■■	■■■	□□□
RAMPED version B is slowly rolled out and replacing version A	✓	✗	✗	■□□	■■■	■□□	■□□
BLUE/GREEN version B is released alongside version A, then the traffic is switched to version B	✓	✗	✗	■■■	□□□	■■□	■■□
CANARY version B is released to a subset of users, then proceed to a full rollout	✓	✓	✗	■□□	■□□	■□□	■■□
A/B TESTING version B is released to a subset of users under specific condition	✓	✓	✓	■□□	■□□	■□□	■■■
SHADOW version B receives real world traffic alongside version A and doesn't impact the response	✓	✓	✗	■■■	□□□	□□□	■■■

CI / CD Pipelines

What is a CI/CD pipelines ?

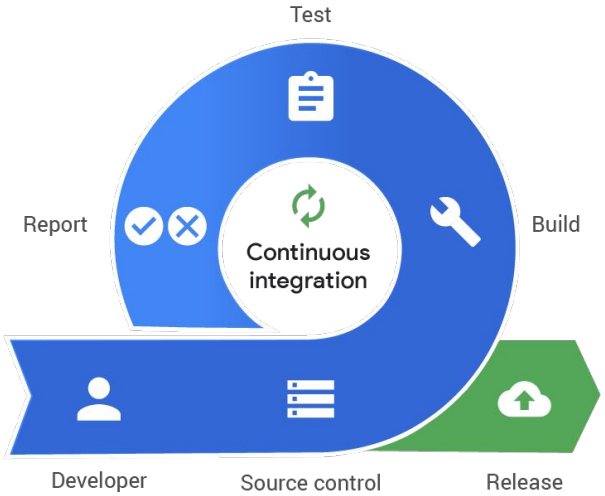
- Continuous Integration and Continuous Deployment (CI/CD) pipeline represents an **automatic workflow** that continuously integrates code developed by software developers and deploys them into a target environment with less human intervention.



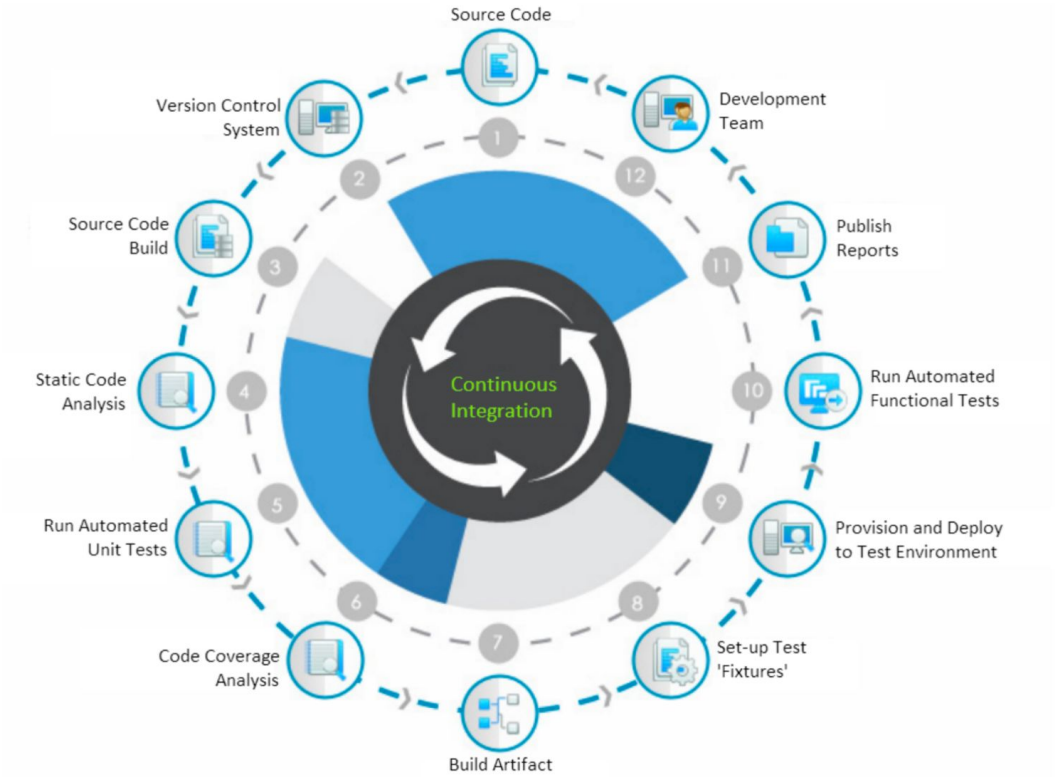
Continuous Integration or CI pipeline

CI is part of the development process. The central idea in this stage is to automatically build the software whenever new software is **committed** by developers

- Focuses on **frequently** merging code changes into a central repository
- Each code change, trigger automated builds and tests
- Early detection
- This ensures that the code compiles correctly and passes all the essential tests



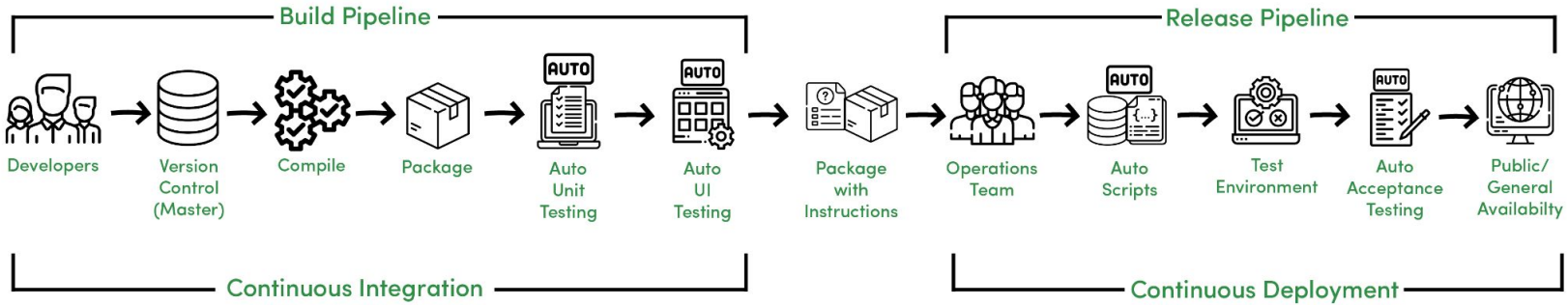
Continuous Integration or CI pipeline



Continuous Deployment or CD pipeline

Continuous deployment stage refers to pulling the artifact from the repository and deploying it frequently and safely. Automate the deployment into the each environments with less human intervention

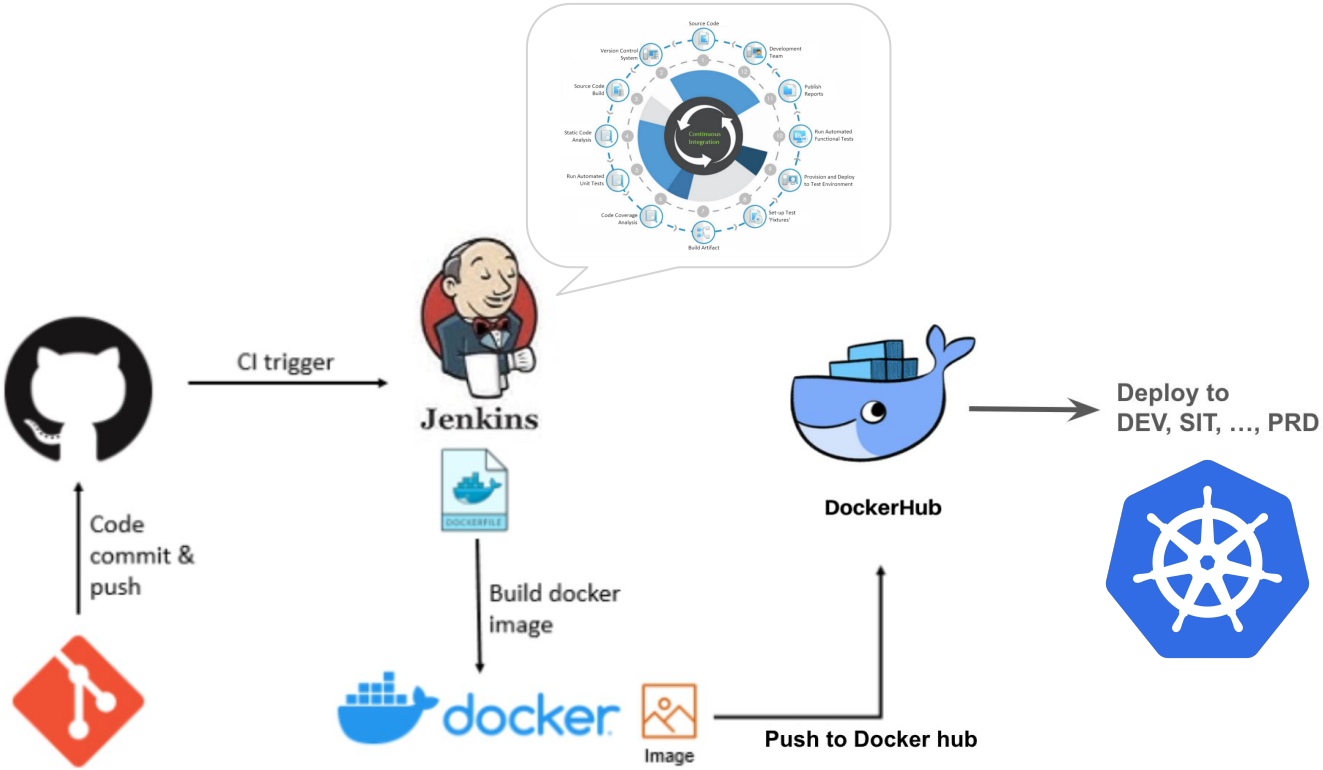
- Once the code passes CI checks, CD takes over
- Some approaches call **Continuous Delivery:** Makes the built code readily available for deployment, allowing for manual review before pushing it live



Benefits of CI/CD Pipelines

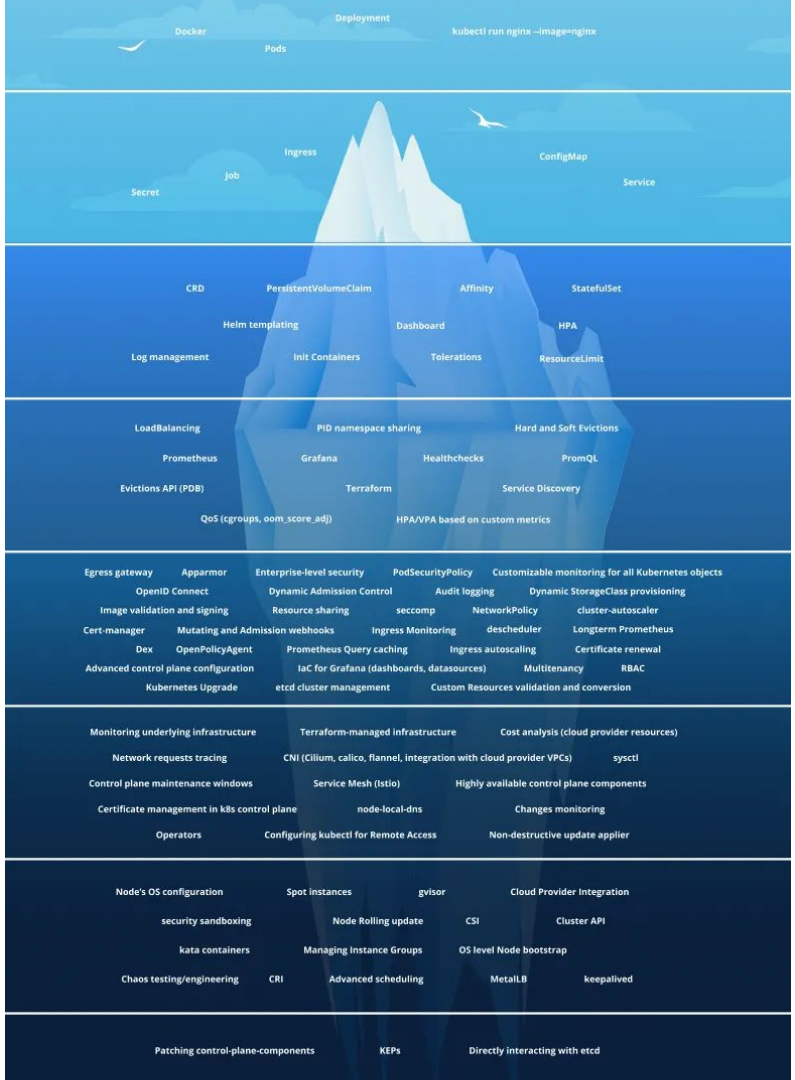
- Faster Software Delivery
- Improved Software Quality
- Reduced Risk
- Increased Collaboration
- Reliable & cheaper deployments

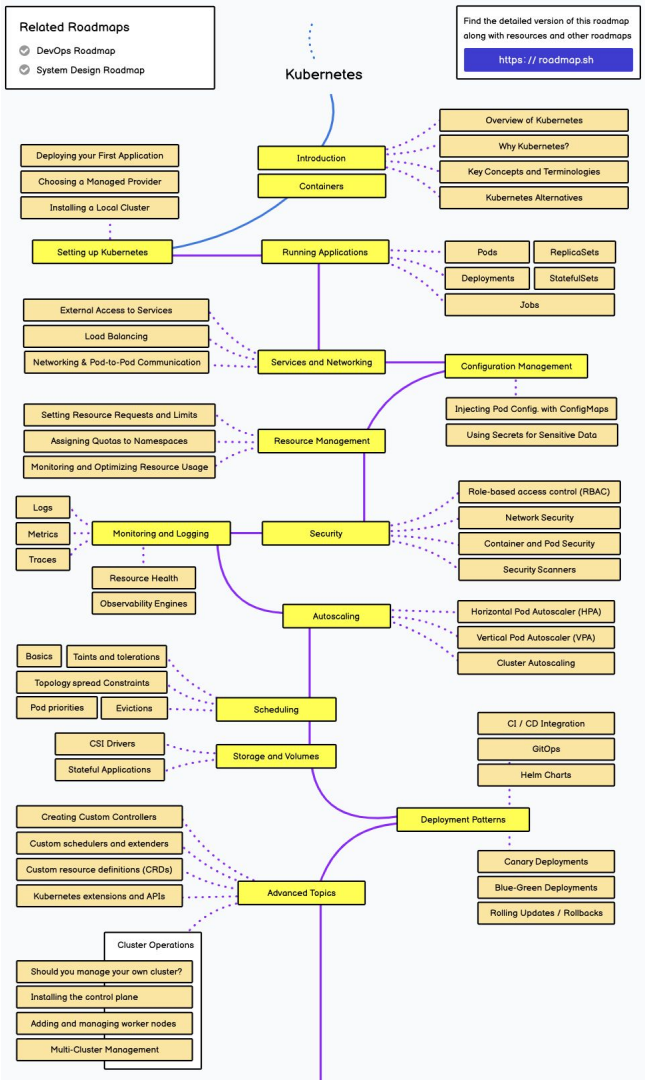
Apply Docker & k8s with pipelines



Workshop

CI / CD Pipelines


Deployment
Docker
kubectl run nginx --image=nginx
Pods
Ingress
ConfigMap
Job
Service
Secret
CRD
PersistentVolumeClaim
Affinity
StatefulSet
Helm templating
Dashboard
HPA
Log management
Init Containers
Tolerations
ResourceLimit
LoadBalancing
PID namespace sharing
Hard and Soft Evictions
Prometheus
Grafana
Healthchecks
PromQL
Evictions API (PDB)
Terraform
Service Discovery
QoS (cgroups, oom_score_adj)
HPA/VPA based on custom metrics
Egress gateway
Apparmor
Enterprise-level security
PodSecurityPolicy
Customizable monitoring for all Kubernetes objects
OpenID Connect
Dynamic Admission Control
Audit logging
Dynamic StorageClass provisioning
Image validation and signing
Resource sharing
seccomp
NetworkPolicy
cluster-autoscaler
Cert-manager
Mutating and Admission webhooks
Ingress Monitoring
descheduler
Longterm Prometheus
Dex
OpenPolicyAgent
Prometheus Query caching
Ingress autoscaling
Certificate renewal
Advanced control plane configuration
IaC for Grafana (dashboards, datasources)
Multitenancy
RBAC
Kubernetes Upgrade
etcd cluster management
Custom Resources validation and conversion
Monitoring underlying infrastructure
Terraform-managed infrastructure
Cost analysis (cloud provider resources)
Network requests tracing
CNI (Cilium, calico, flannel, integration with cloud provider VPCs)
sysctl
Control plane maintenance windows
Service Mesh (Istio)
Highly available control plane components
Certificate management in k8s control plane
node-local-dns
Changes monitoring
Operators
Configuring kubectl for Remote Access
Non-destructive update applier
Node's OS configuration
Spot instances
gvisor
Cloud Provider Integration
security sandboxing
Node Rolling update
CSI
Cluster API
kata containers
Managing Instance Groups
OS level Node bootstrap
Chaos testing/engineering
CRI
Advanced scheduling
MetalLB
keepalived
Patching control-plane-components
KEPs
Directly interacting with etcd



https://roadmap.sh/kubernetes

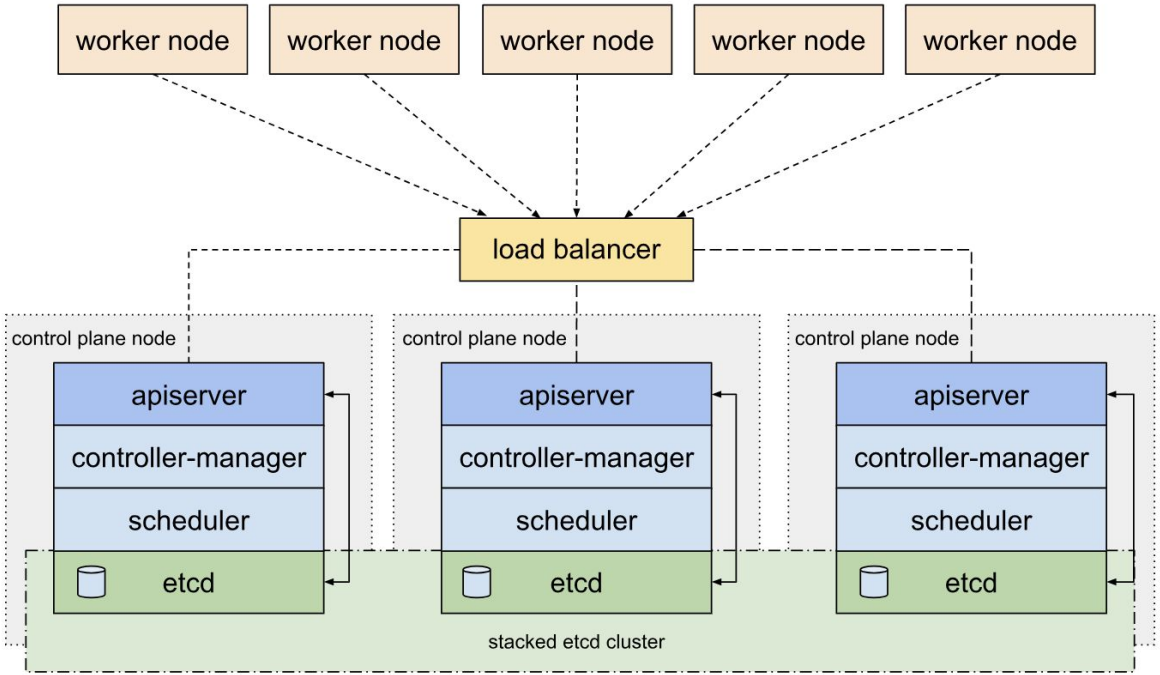
Cluster Setup

Options for Highly Available Topology

- Stacked etcd topology
- External etcd topology

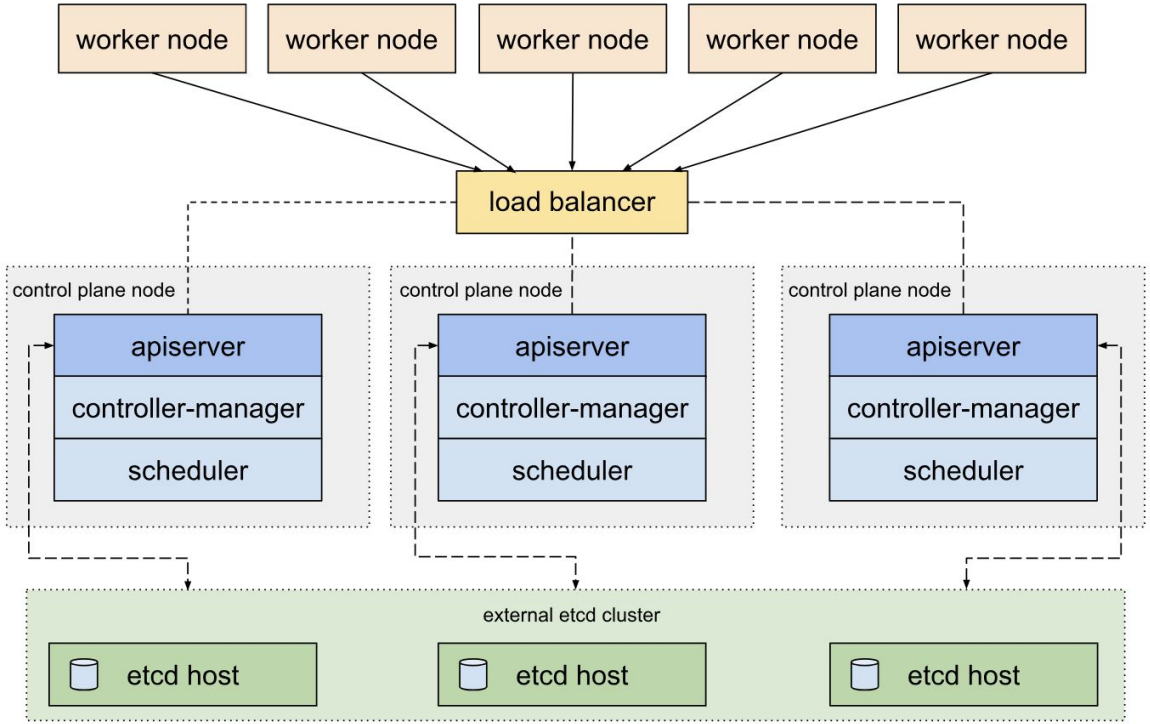
https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/

Stacked etcd topology



https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/

External etcd topology



https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/

Fault Tolerance Table

Cluster Size	Majority	Failure Tolerance
1	1	0
2	2	0
3	2	**1**
4	3	1
5	3	**2**
6	4	2
7	4	**3**
8	5	3
9	5	**4**

https://etcd.io/docs/v2.3/admin_guide/#fault-tolerance-table

Q & A

Feedback