



Containerized In Action (From Docker to Kubernetes)

Who We Are



Joe

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



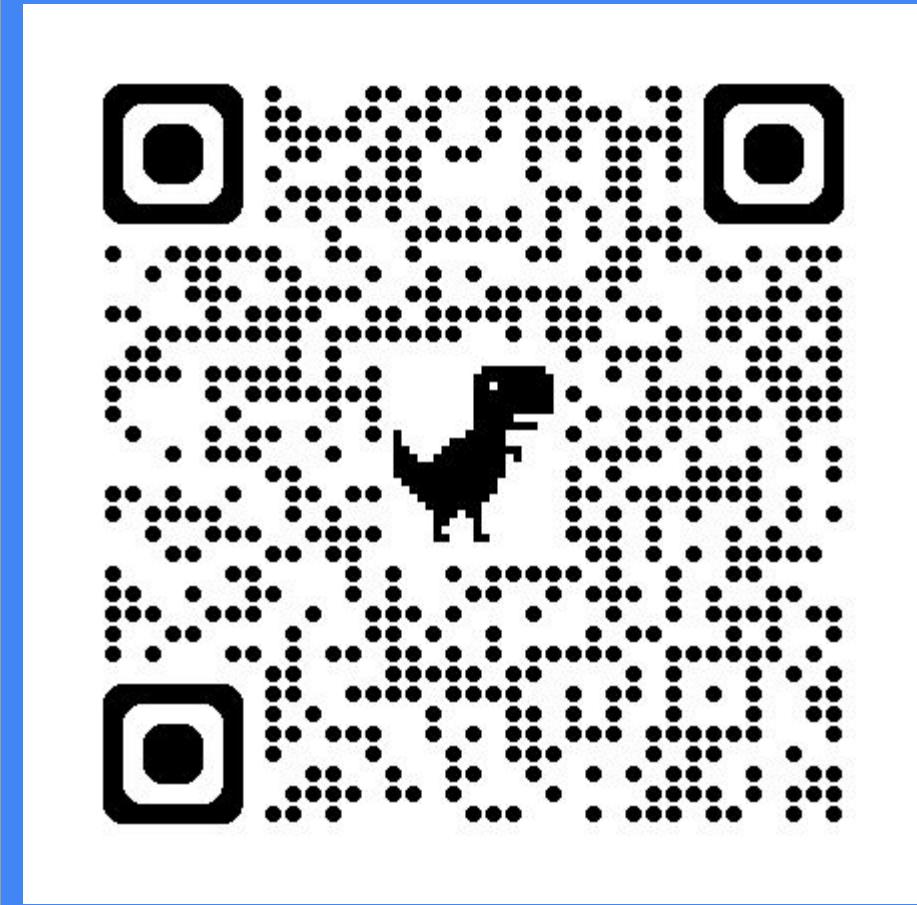
Meng

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

Our Playground

<https://miro.com/app/board/uXjVKHU6Ggl=/>

<https://tinyurl.com/sckdojo>



Day 1

Day 1 Agenda

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

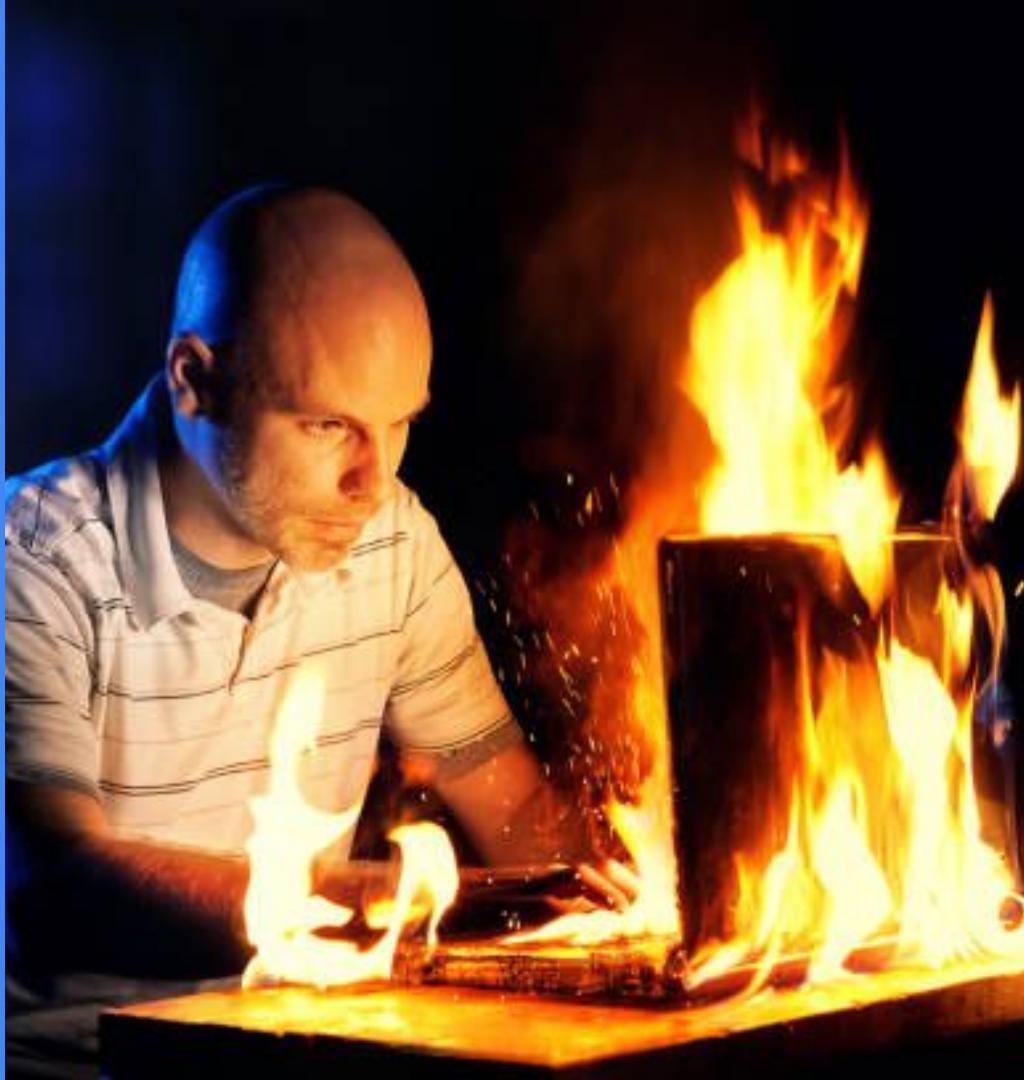
A large cargo ship, the MSC ZEPHYRUS, is shown from a low angle, sailing on the ocean under a clear blue sky. The ship's hull is dark blue with white horizontal stripes and features the MSC logo. It is heavily loaded with thousands of shipping containers stacked high along its sides. A small white pilot boat is visible in the water near the ship.

What problem is Docker trying to Solve

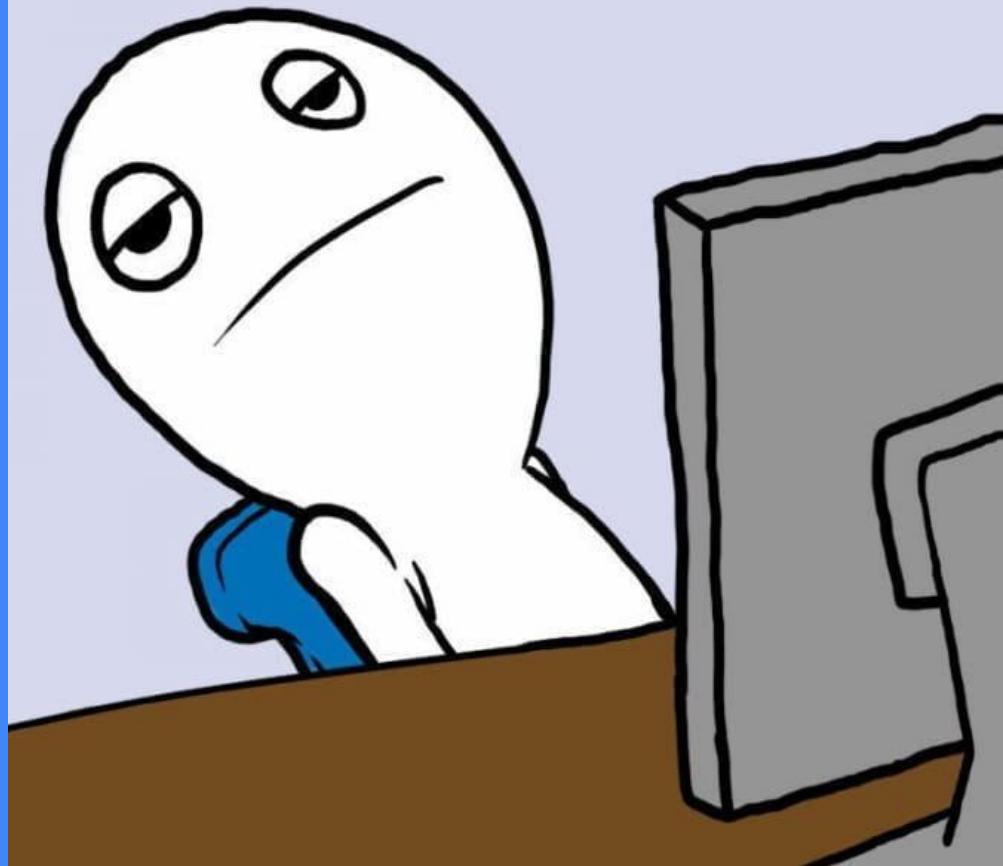
#1 Inconsistency Dependency And Complexity



#2
It works on
my machine



#3
Install,
Upgrade
Remove
and repeat.



#4 Security?

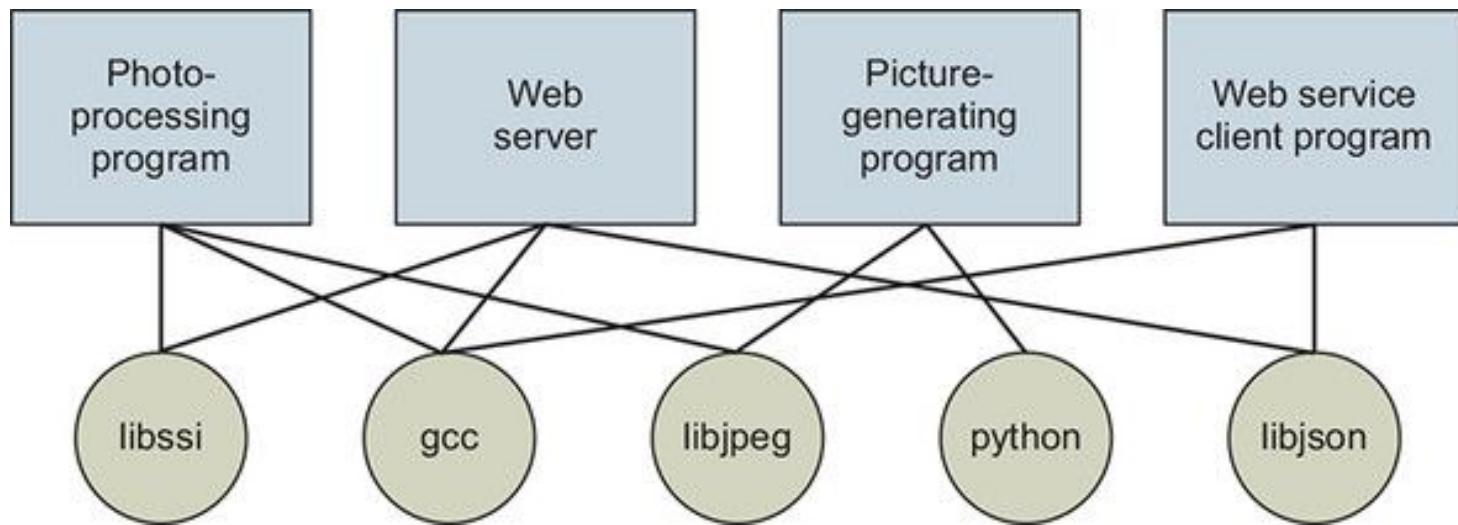


#5 Testing?



**TEST
ON YOUR
MACHINE**

**TEST IN
PRODUCTION**



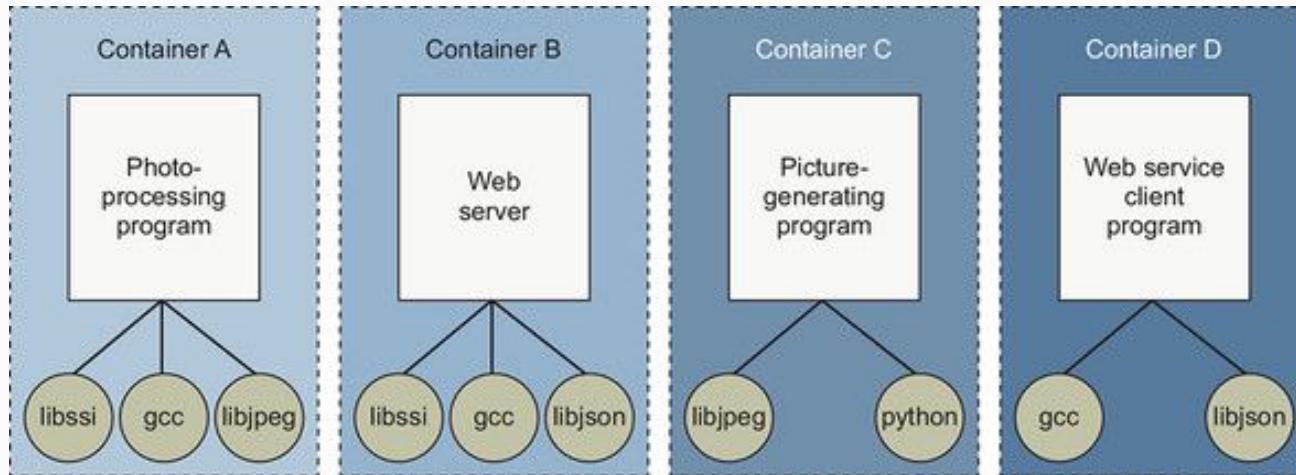
Complexity

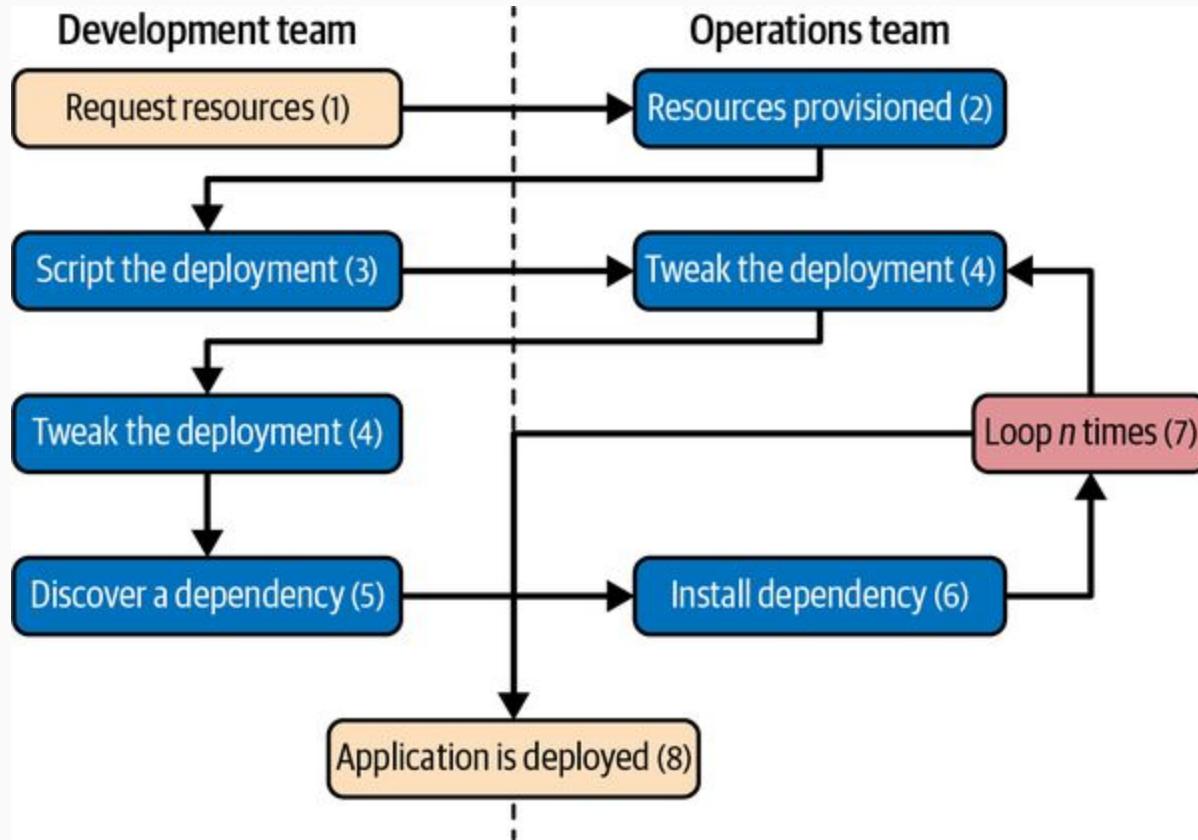
LEAD TO

Complicated

- | | |
|------------------------------------|-----------------------------|
| 1. Tightly coupling to host | 1. Difficult to understand. |
| 2. Variety of version | 2. Difficult to test |
| 3. Unstable environment | 3. Difficult to manage |
| 4. Inversion of system environment | 4. Difficult to develop |
| 5. Specific of physical hardware | 5. Difficult to maintain |

Container Principle is Isolation





Development team

Build image

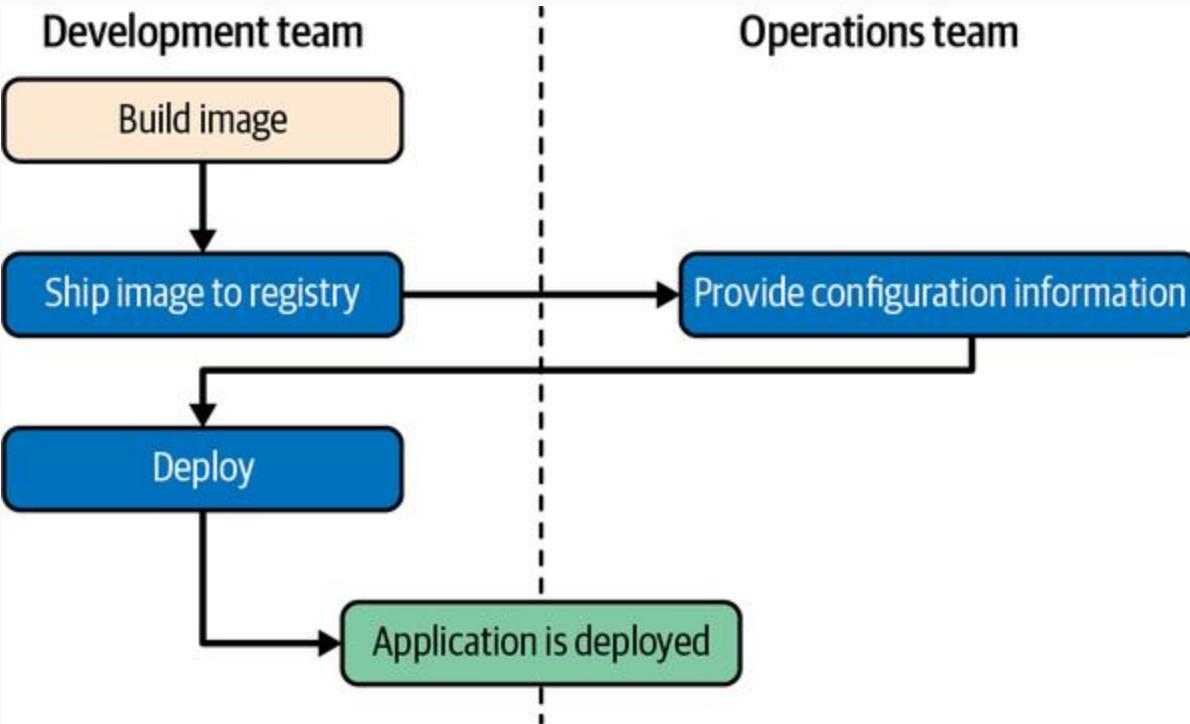
Ship image to registry

Deploy

Operations team

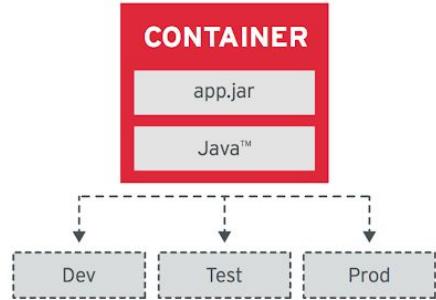
Provide configuration information

Application is deployed

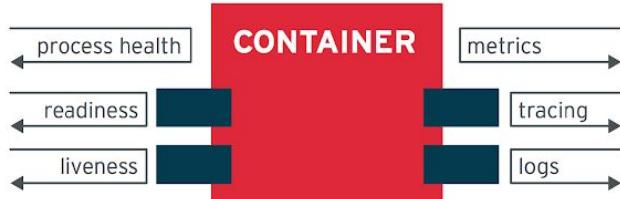


Principles of Container-based

Image Immutability Principle



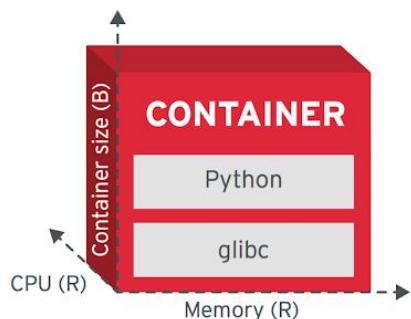
High Observability Principle



Process Disposability Principle



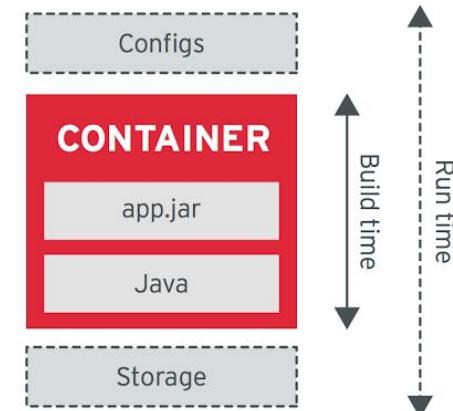
Runtime Confinement Principle



Lifecycle Conformance Principle



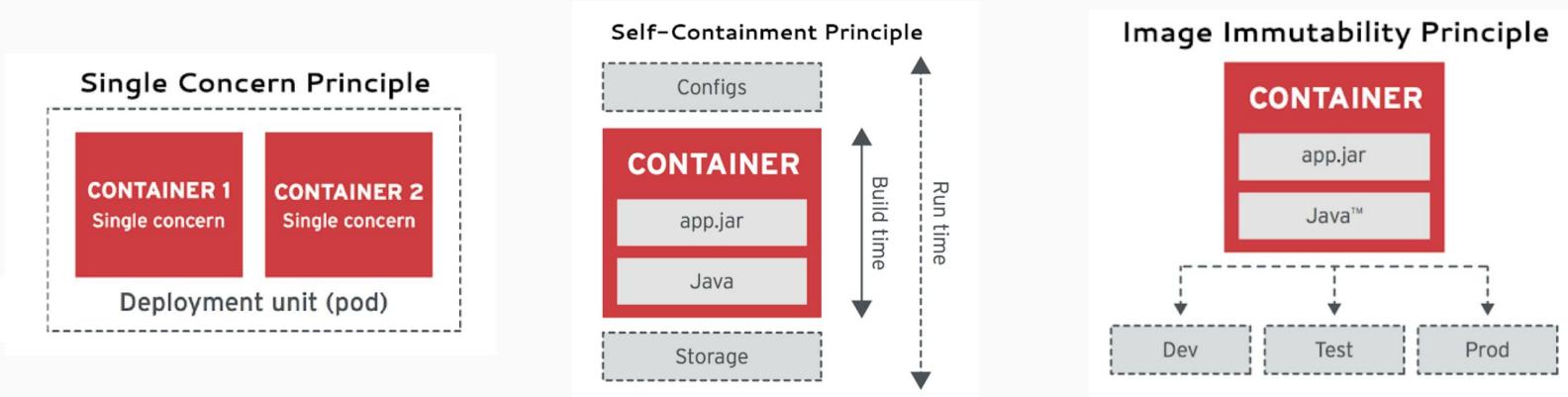
Self-Containment Principle



Single Concern Principle

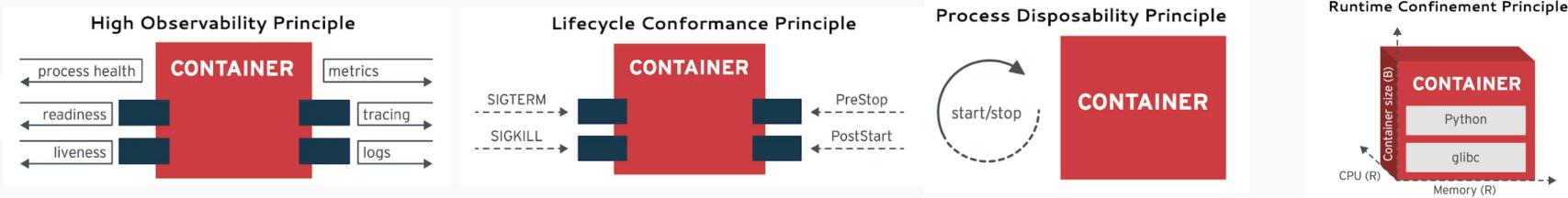
Build Time

- **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.



Runtime

- **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.

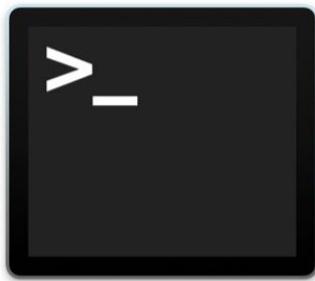


Docker Container



Check-up Your Machine

Mac OS



Window OS



Workshop

Hello Docker



Hello Docker!

\$ docker run --rm hello-world

Docker Architecture

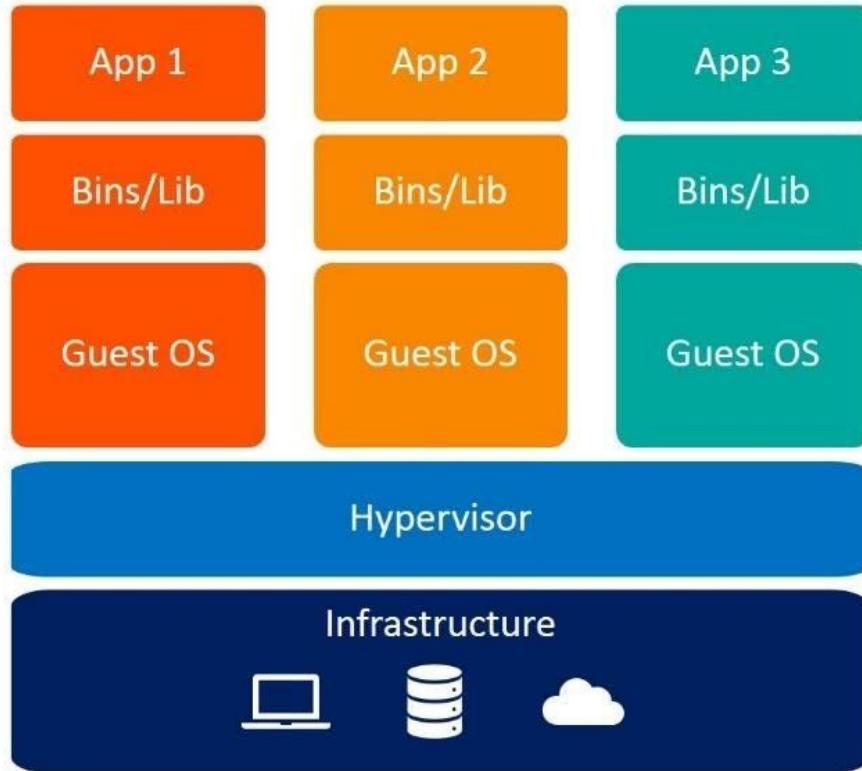


Exploring Docker Machine

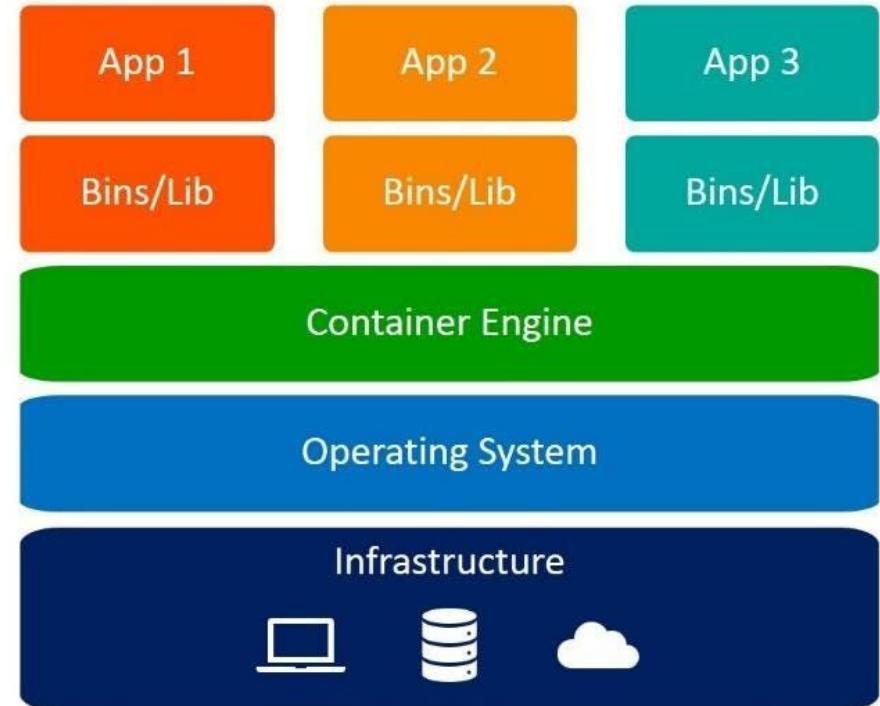
\$ docker system info

\$ docker context ls

\$ docker context desktop-linux

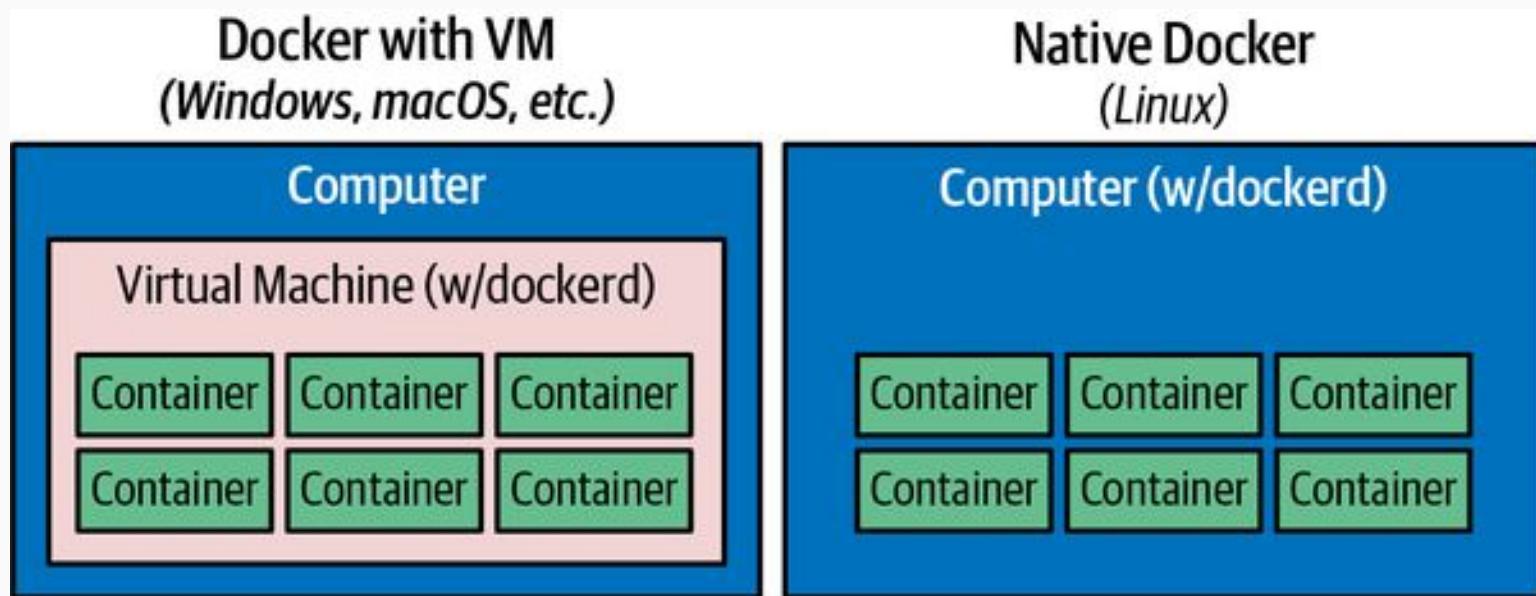


Virtual Machines

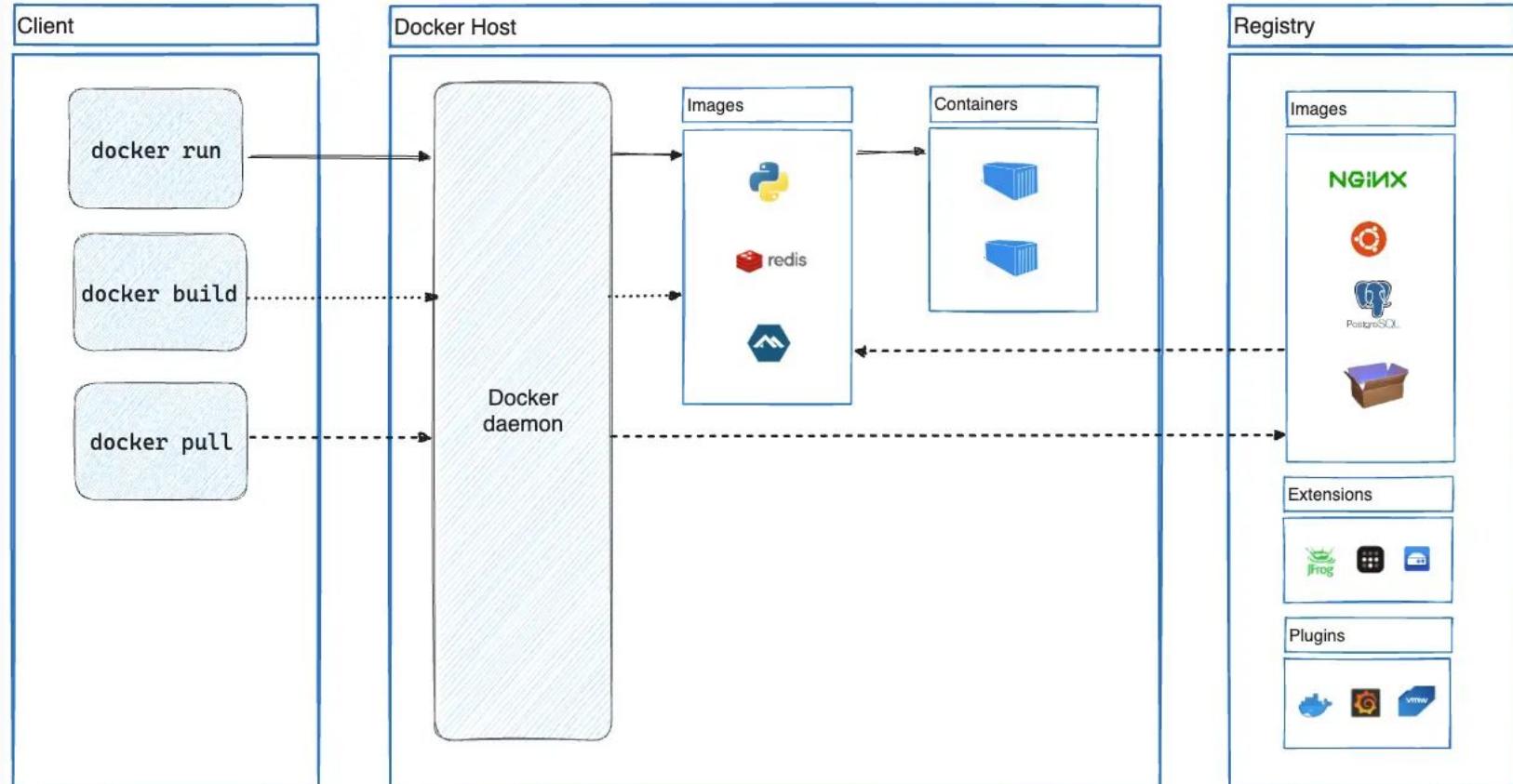


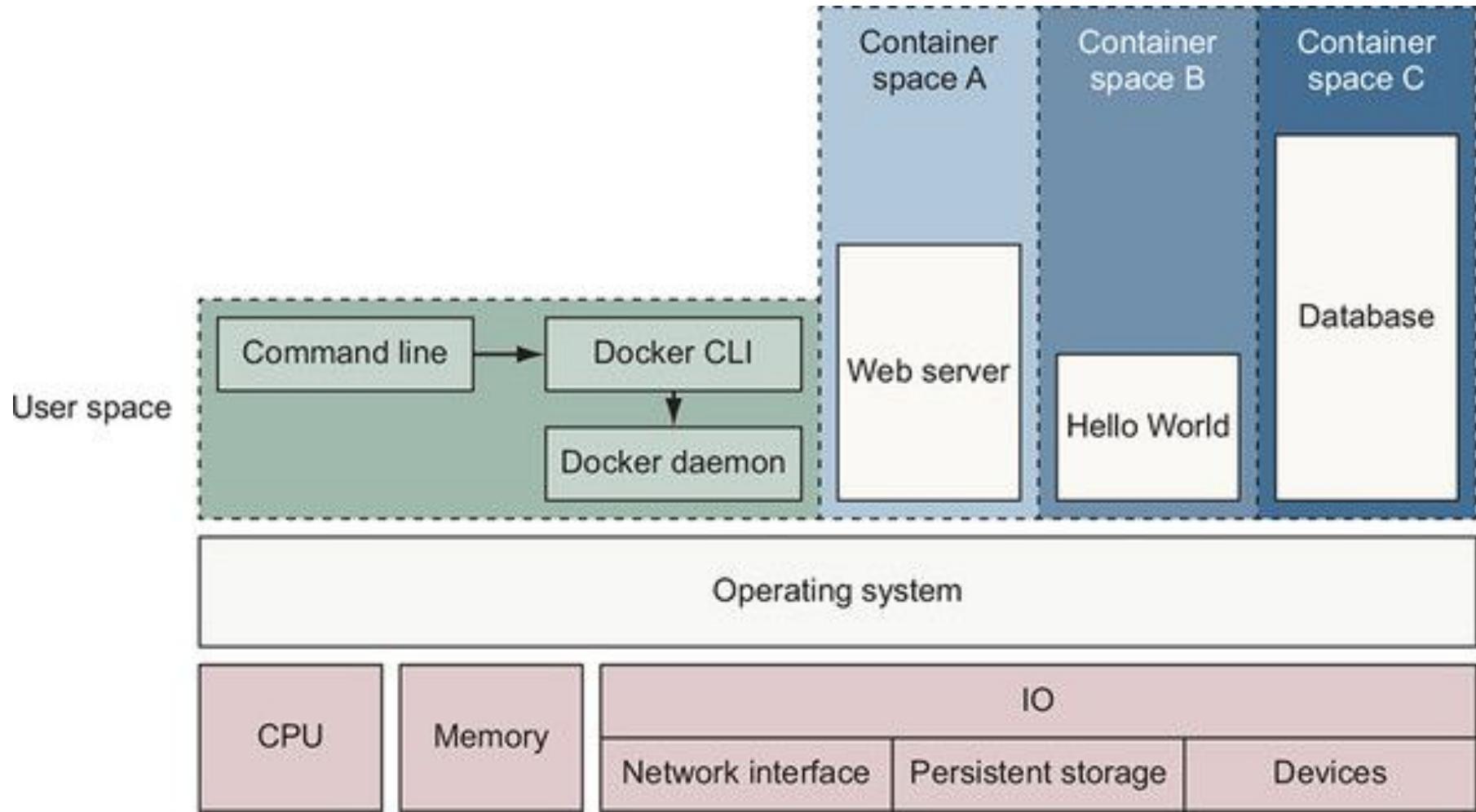
Containers

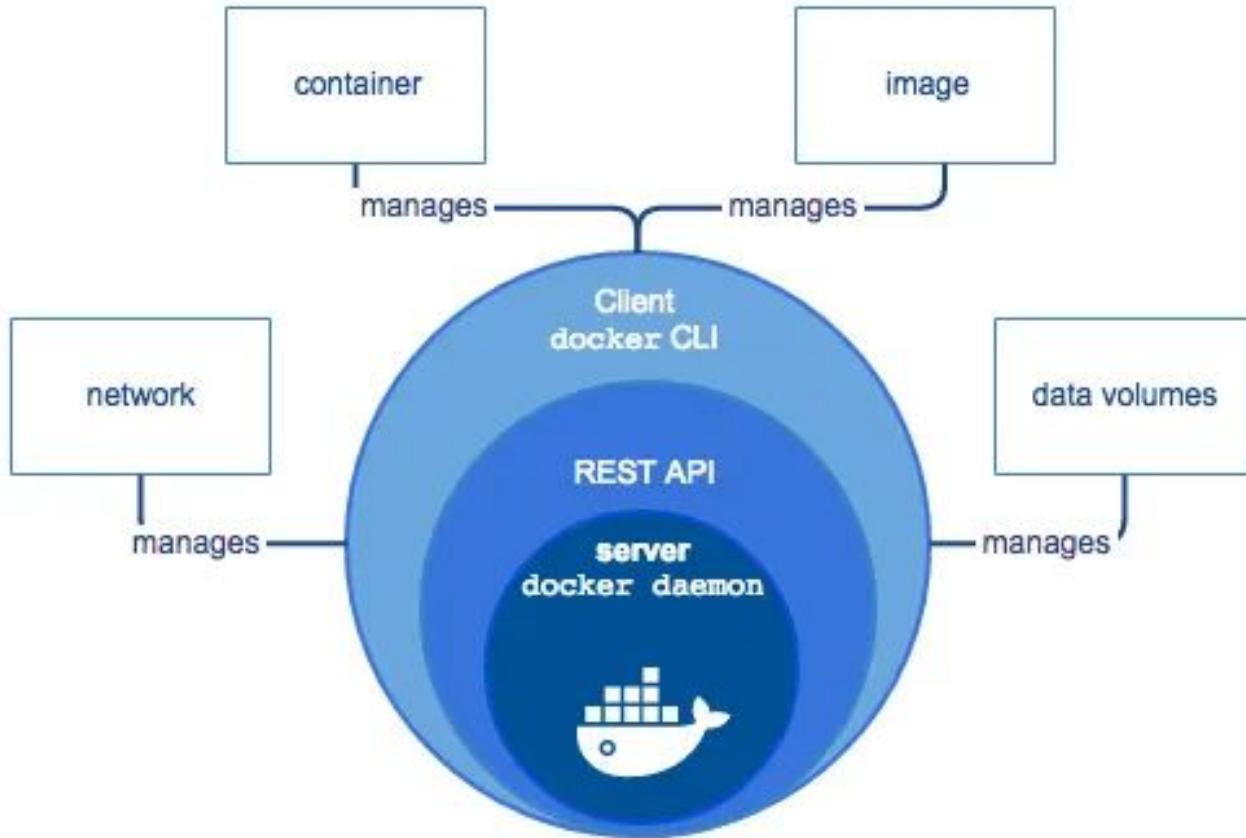
Docker Under the Hood



Docker Architecture







Docker Image



```
# 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
```

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  
  
# 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 {YourName} -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
```

```
$ docker container exec -it my-nginx-container-v1 bash
```

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



Dockerfile

build



Docker Image

run



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" ]
```

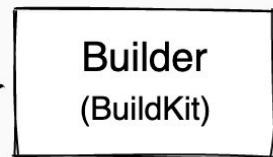


Docker Build high-level architecture

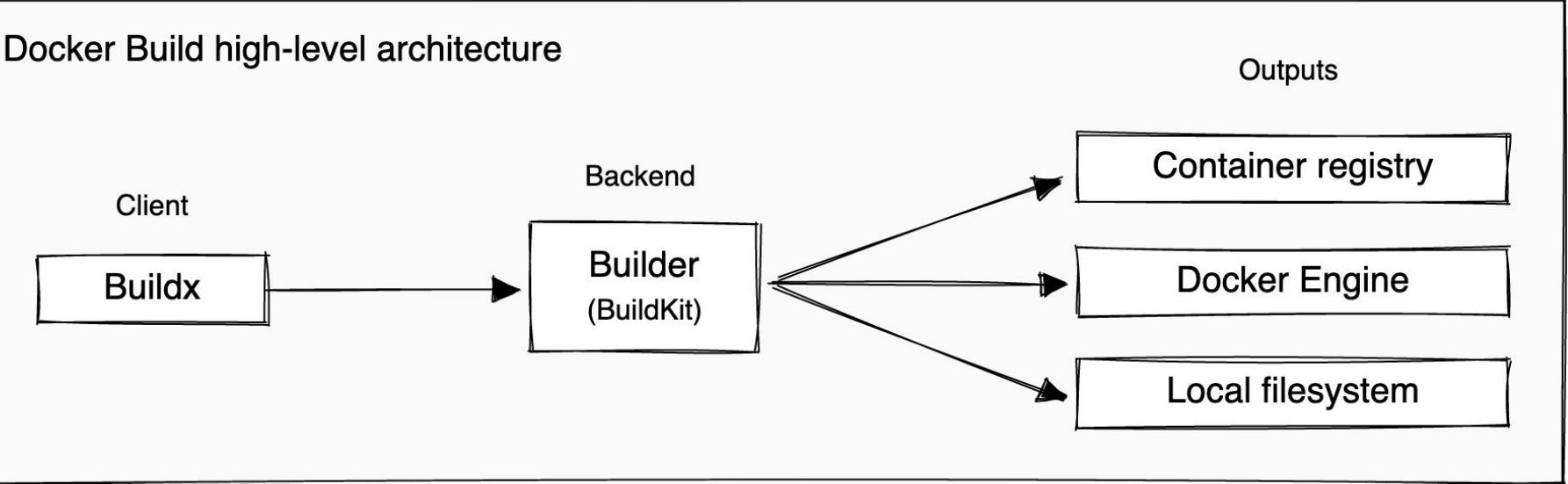
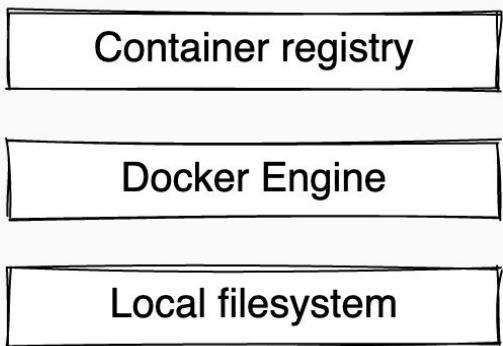
Client



Backend



Outputs



Workshop

Publish Your Image to Registry



Structure of Docker Image Name

{Registry-host}/{User or Org}/{ImageName}:{Tag}

Ex. docker.io/kkosittaruk/mynginx:1.0.0

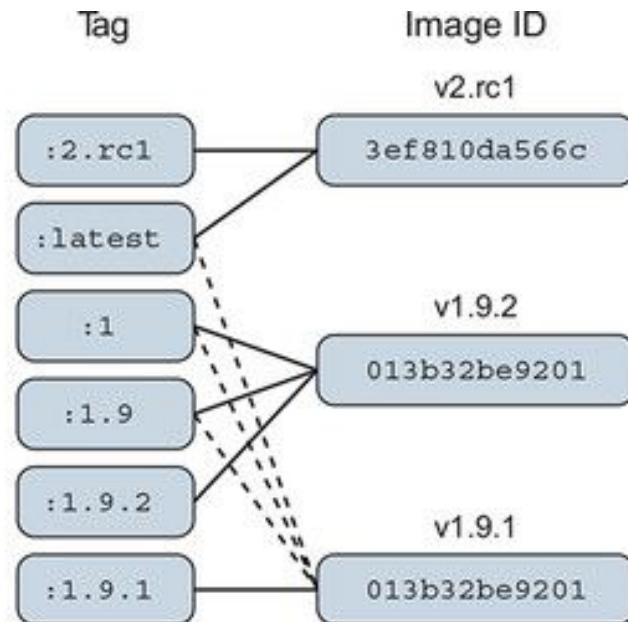
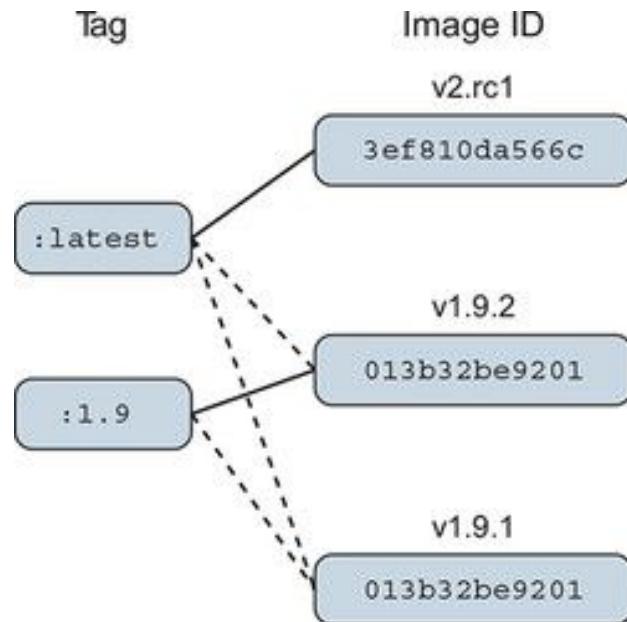
Docker Image Tag Pattern

Docker Tag 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 be multiple tag
- **Avoid** to use tag **latest** in production and test environment





Explore

Repositories

Organizations



Search Docker Hub



Account Settings / Security



keattiw



User



Joined November 24, 2023

General

Security

Default Privacy

Notifications

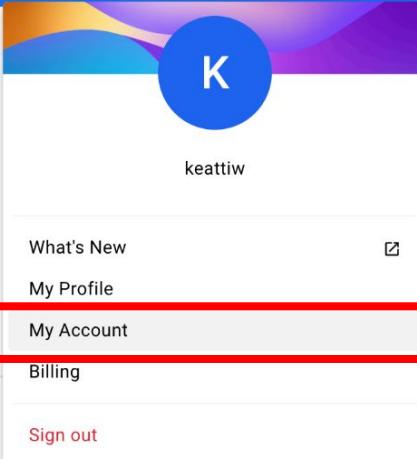
Convert Account

Deactivate Account

Access Tokens

Tokens marked **AUTO-GENERATED** are created on your behalf by Docker Desktop for the CLI to use for authentication. You can have a maximum of 5 auto-generated tokens associated with your account. [Learn more](#).

| <input type="checkbox"/> | Description | Source | Scope | Last Used | Created | ⋮ |
|--------------------------|-------------------|------------------------|--------------|-----------------------|-----------------------|----------|
| <input type="checkbox"/> | ● docker-workshop | MANUAL | Read & Write | Jun 02, 2024 23:03:52 | Jun 02, 2024 23:03:39 | ⋮ |



[Explore](#)[Repositories](#)[Organizations](#)

Search Docker Hub

[Repositories](#) / [Create](#)Using 0 of 1 private repositories. [Get more](#)

Create repository

Namespace

keattiw



Repository Name *

Short description

A short description to identify your repository. If the repository is public, this description is used to index your content on Docker Hub and in search engines, and is visible to users in search results.

Visibility

Using 0 of 1 private repositories. [Get more](#)

Public

Appears in Docker Hub search results



Private

Only visible to you

[Cancel](#)[Create](#)

Pushing images

You can push a new image to this repository using the CLI:

```
docker tag local-image:tagname new-repo:tagname  
docker push new-repo:tagname
```

Make sure to replace `tagname` with your desired image repository tag.

Push Your Image to Docker.io Registry

```
$ docker login -u {docker account}
```

```
$ docker image tag my-nginx-image:v1 {docker account}/my-nginx-image:v1
```

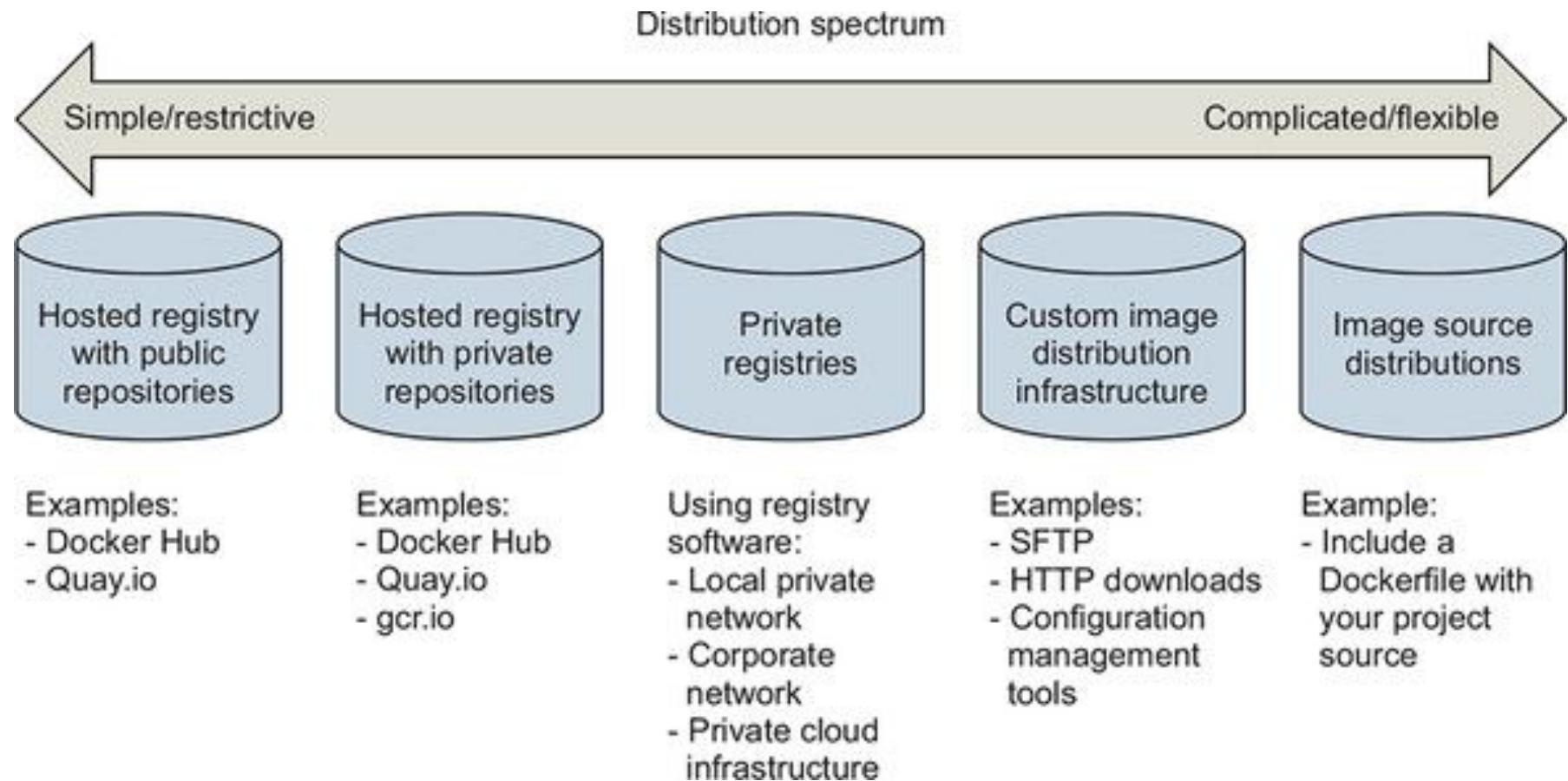
```
$ docker image push {docker account}/my-nginx-image:v1
```

Push Your Image to Docker.io Registry

```
$ docker login -u {docker account}
```

```
$ docker image tag my-nginx-image:v1 {docker account}/my-nginx-image:v1
```

```
$ docker image push {docker account}/my-nginx-image:v1
```



Workshop

Publish to

Private Registry



Push Your Image to local Registry

```
$ docker image pull registry:2
```

```
$ docker container run -rm -d -p 8050:5000 --name registry registry:2
```

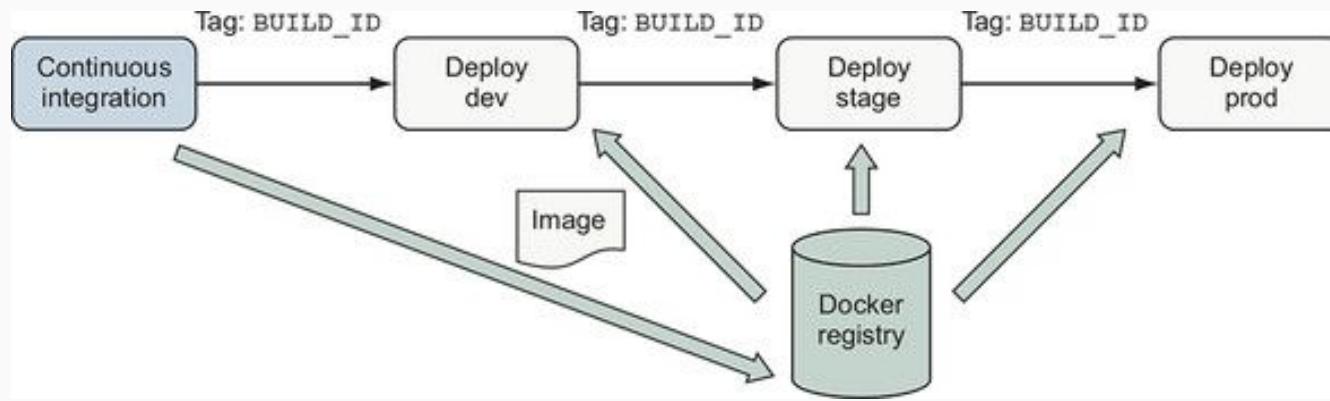
```
$ docker image tag my-nginx-image:v1 \
  localhost:8050/my-nginx-image:v1
```

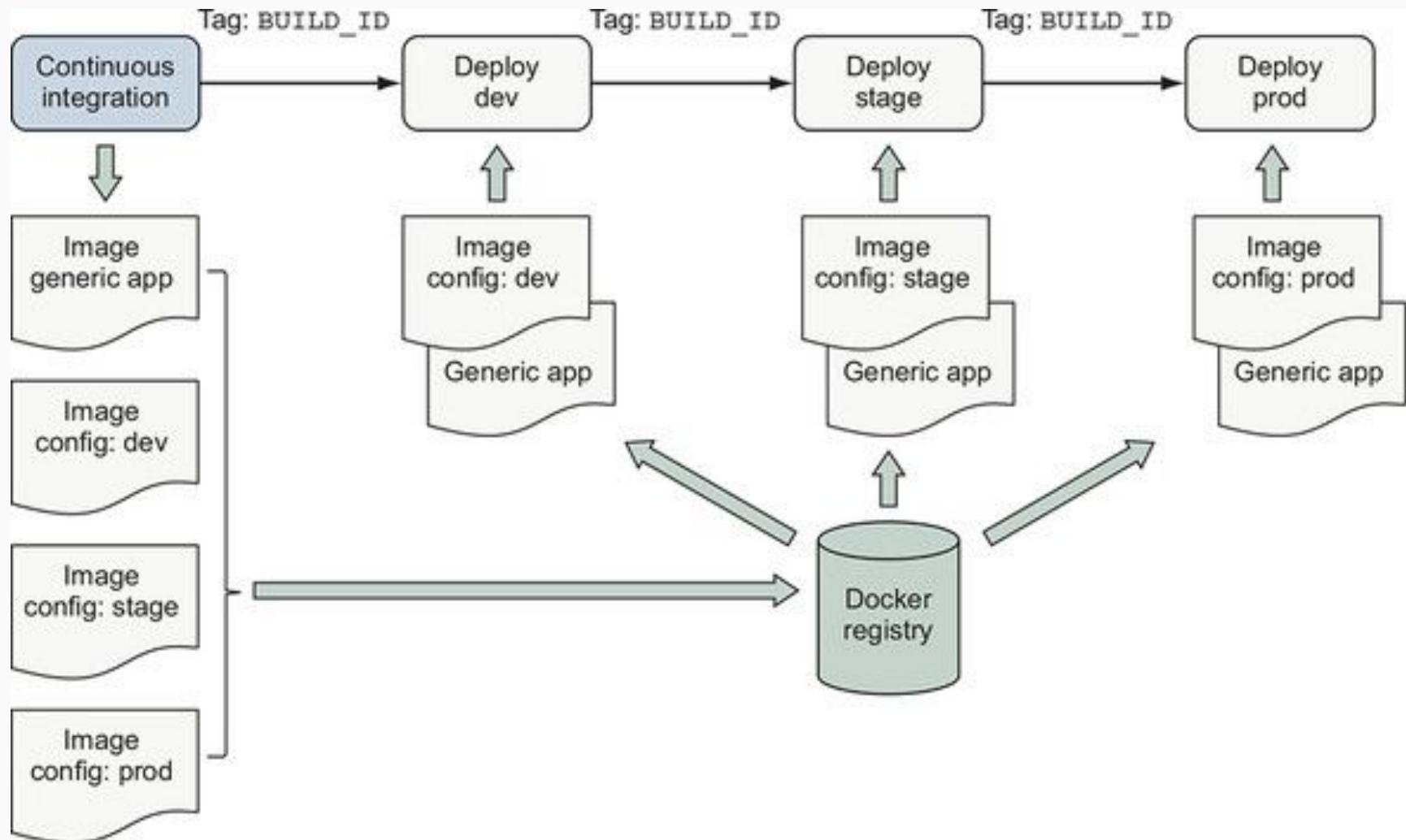
```
$ docker images
```

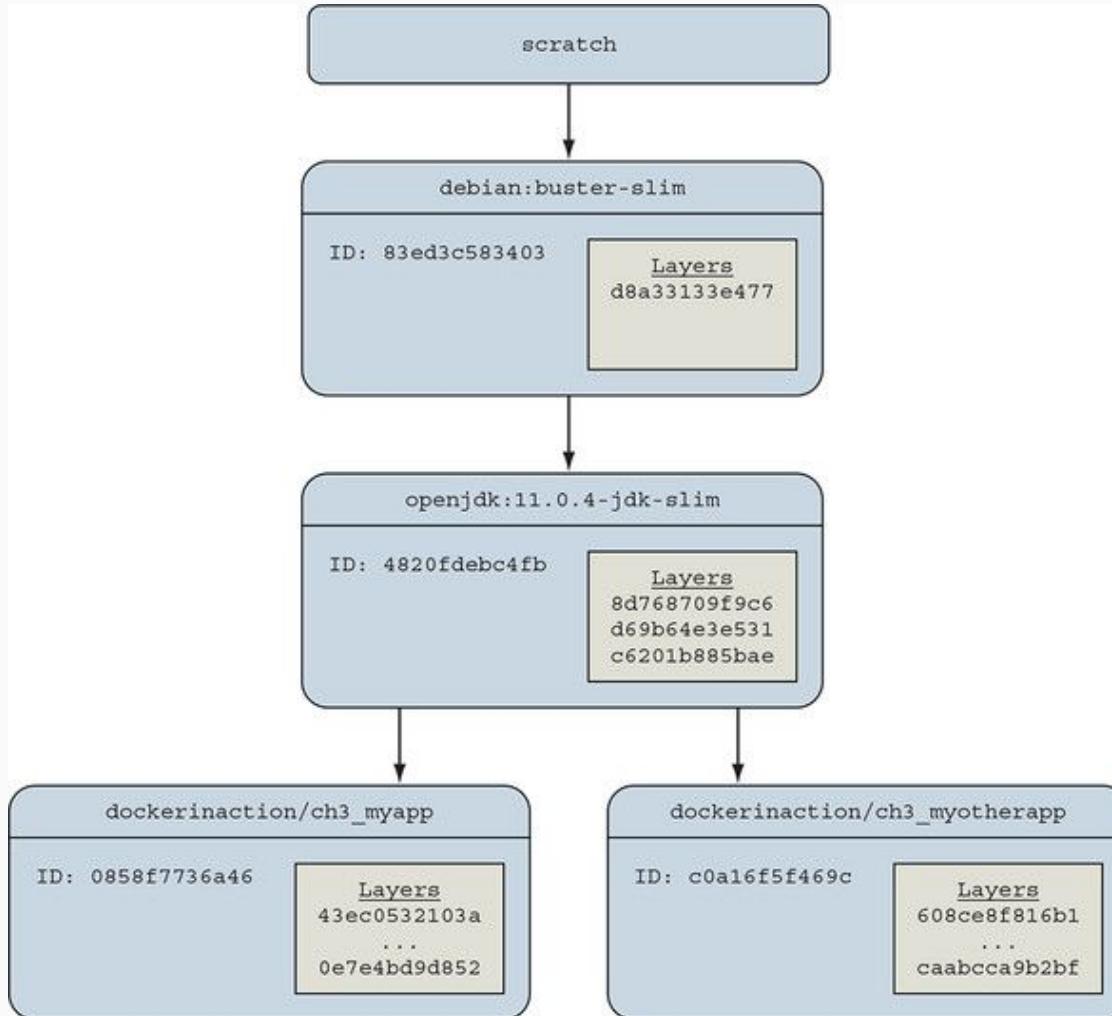
```
$ docker image push localhost:8050/my-nginx-image:v1
```

```
$ docker image rm localhost:8050/my-nginx-image
```

```
$ docker image pull localhost:8050/my-nginx-image
```







Docker Container



Workshop

Deep Dive Docker Container



| | |
|-----------------------------|--|
| # Exploring in container | # Remove Container |
| \$ docker container inspect | \$ docker container rm |
| \$ docker container ps | # Container Logs |
| # List Container | \$ docker container logs |
| \$ docker container ls | # Exec Command to Container |
| # Start Stop Container | \$ docker container exec |
| \$ docker container start | # Remove all container |
| \$ docker container stop | \$ 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
```

```
# 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
```

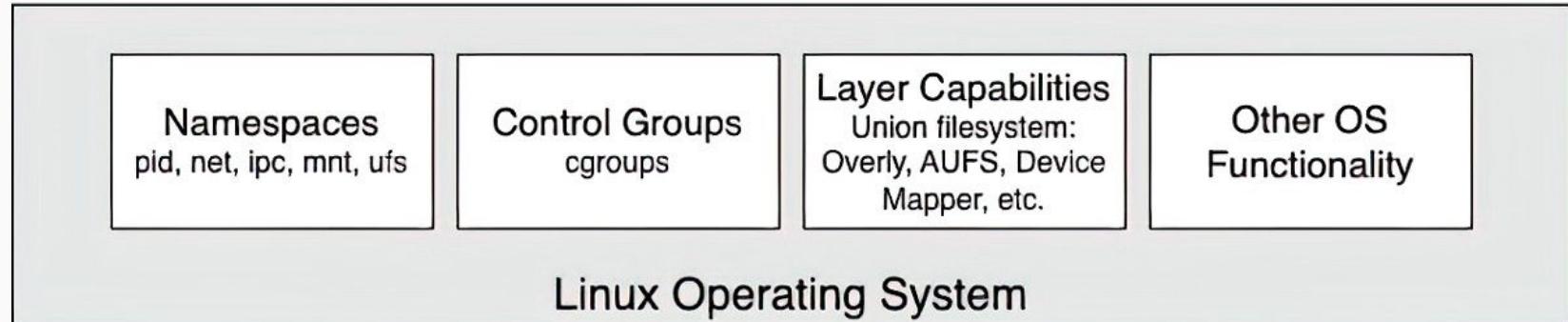
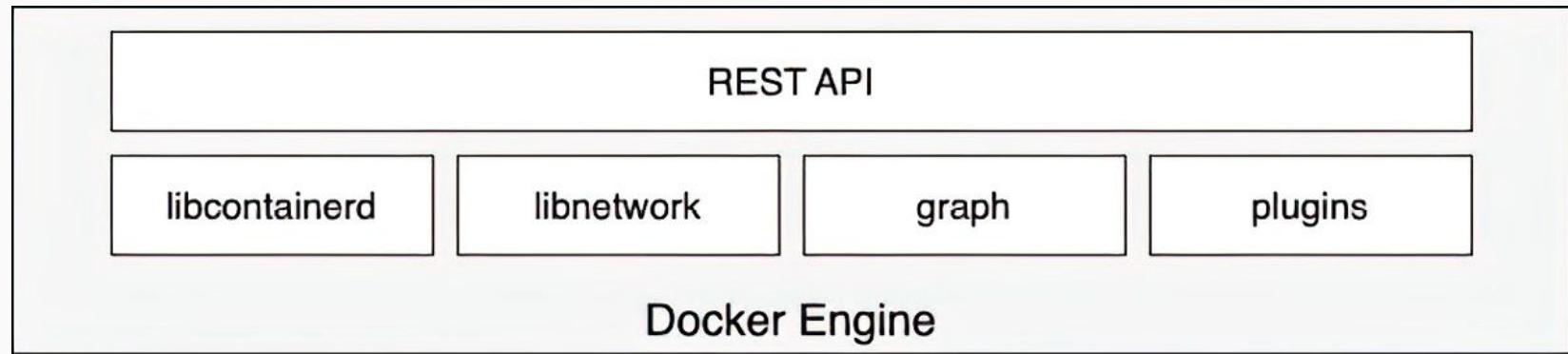
```
$ cat <<EOF >>.env
```

```
HELLO=again?
```

```
SECRET=Don't tell my wife
```

```
EOF
```

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



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



CNCF Graduated



CNCF Incubating



CNCF Incubating



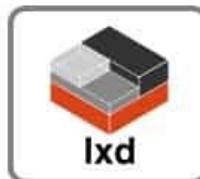
Firecracker



gVisor



kata



lxd



Nabla Containers



Pouch



OPEN
CONTAINER
INITIATIVE
runc



Singularity



SmartOS



unik

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!!!
```

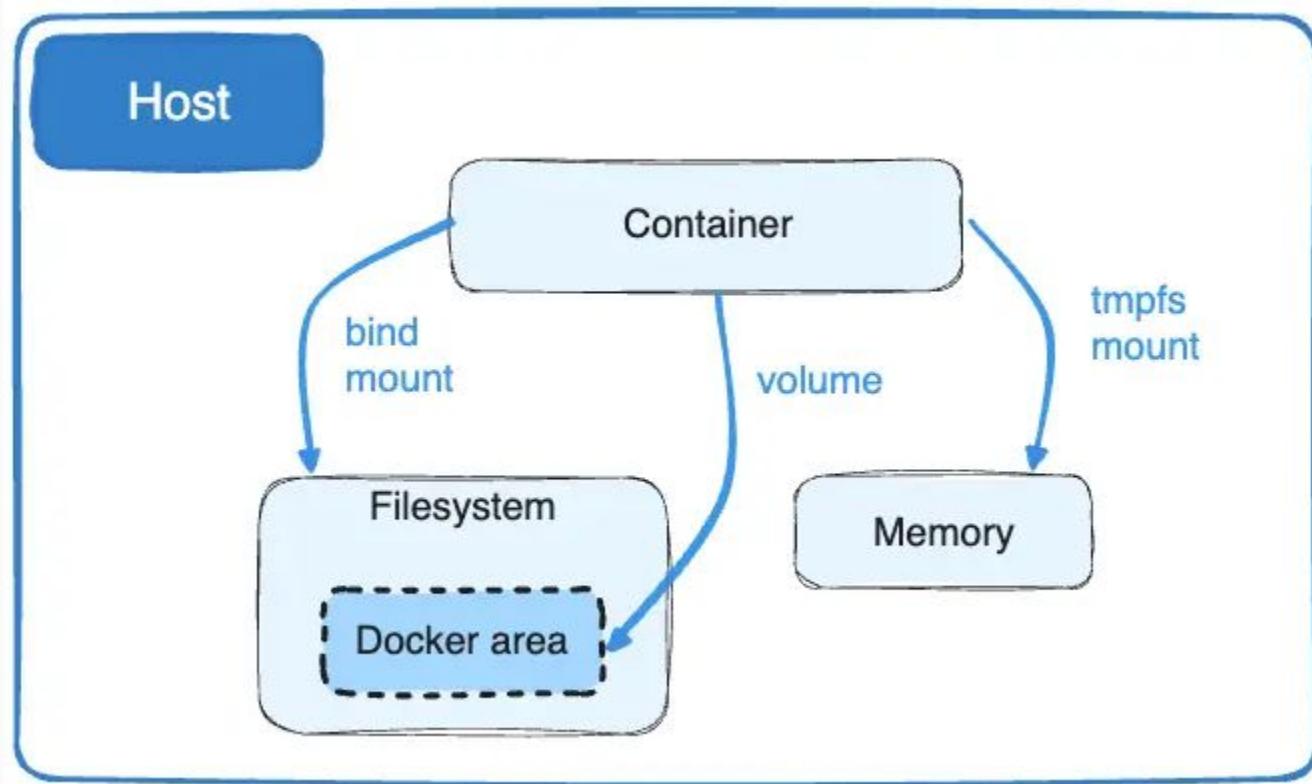
```
# no such file or directory: /var/lib/docker/volumes/sample/_data
```

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



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
```

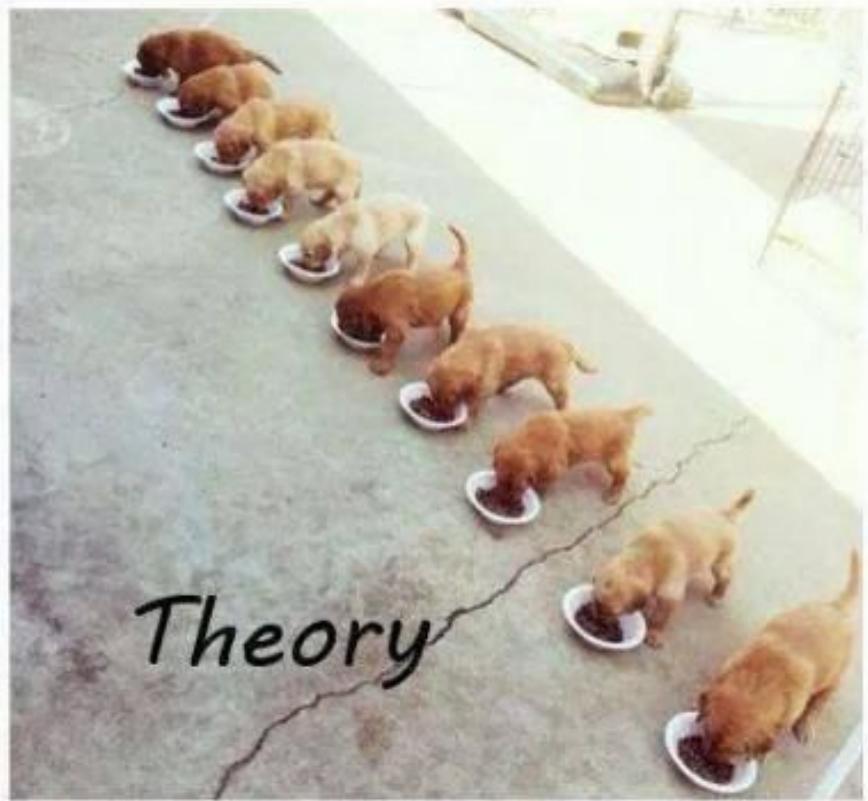
Let's add more data from another container

```
$ 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
```

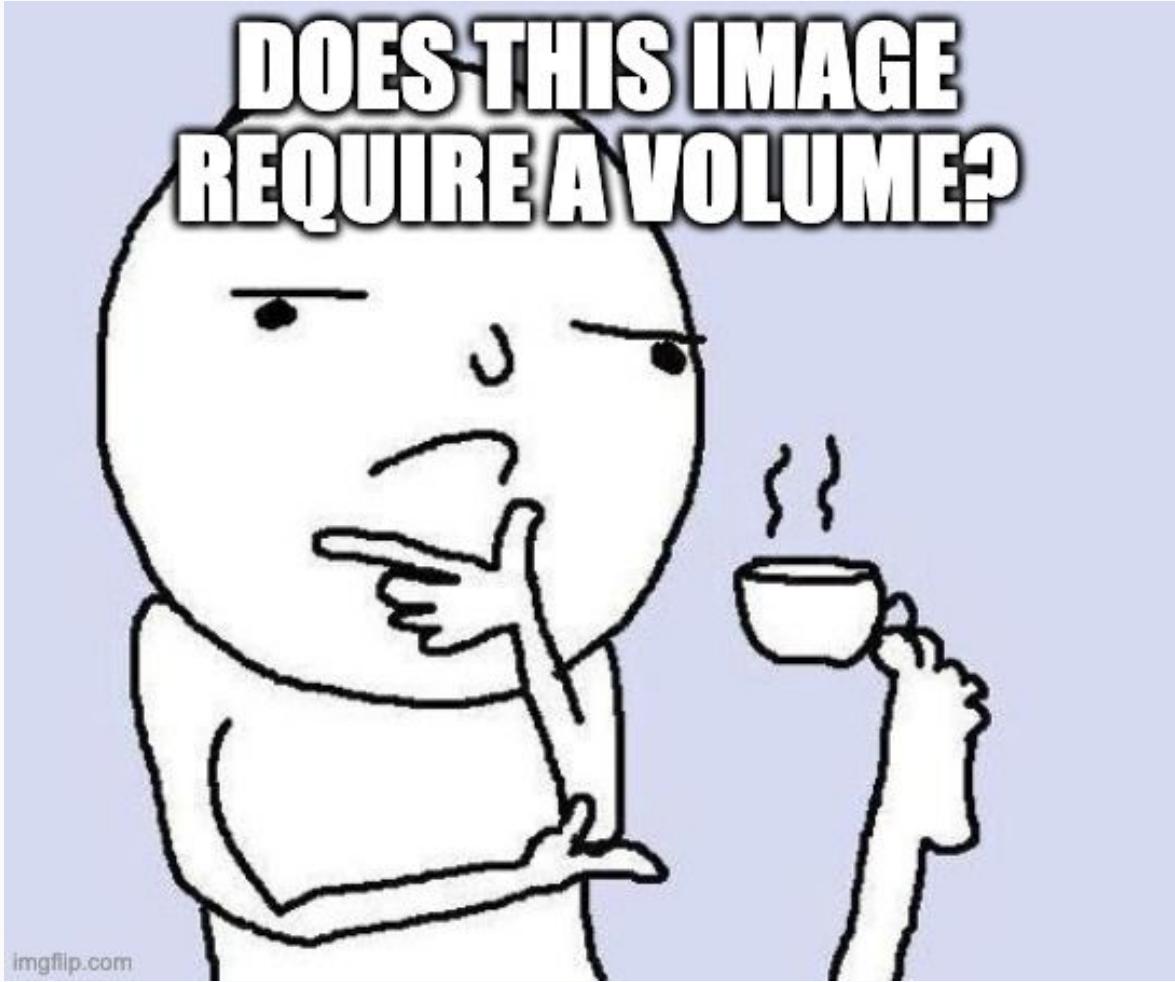
```
$ docker run -d --rm -p 8080:80 \  
-v ./usr/share/nginx/html/ \  
nginx-volume:v1
```

```
$ 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?**



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

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

Backup Docker Volume to Host

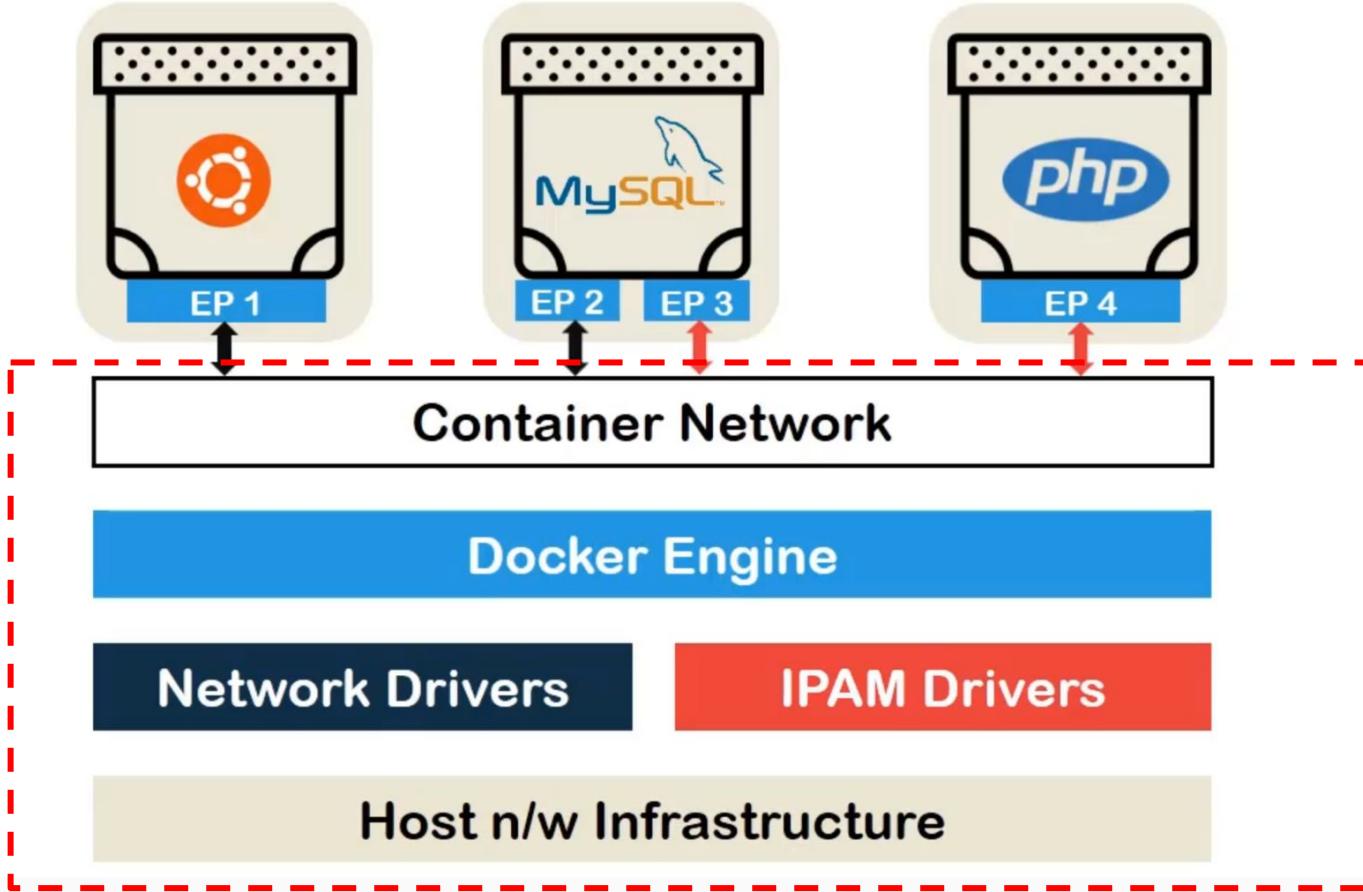
```
$ docker run --rm \  
-v sample:/volume \  
-v $(pwd):/backup \  
busybox \  
tar czvf /backup/sample-volume-backup.tar.gz -C /volume .
```


Docker Network

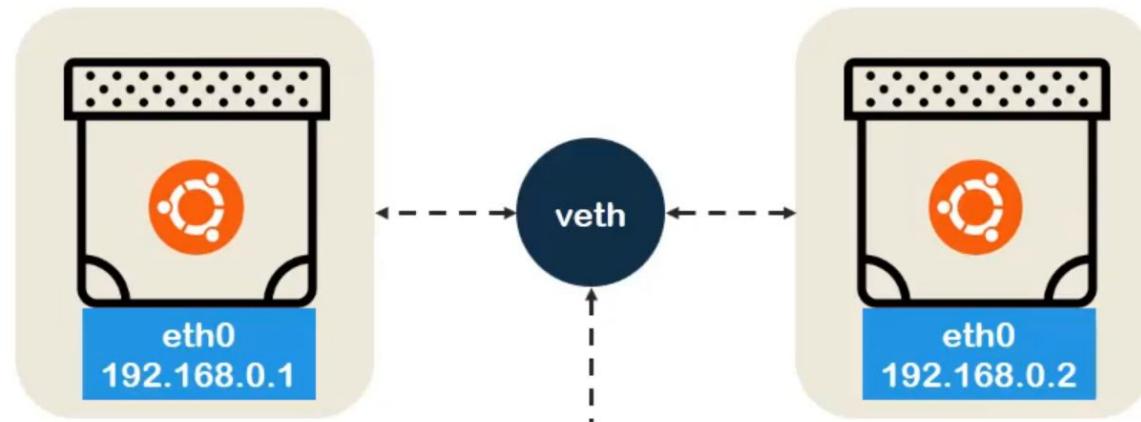


Container Network Model (CNM)





Bridge Network



Docker Network Type

| 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 |

Workshop

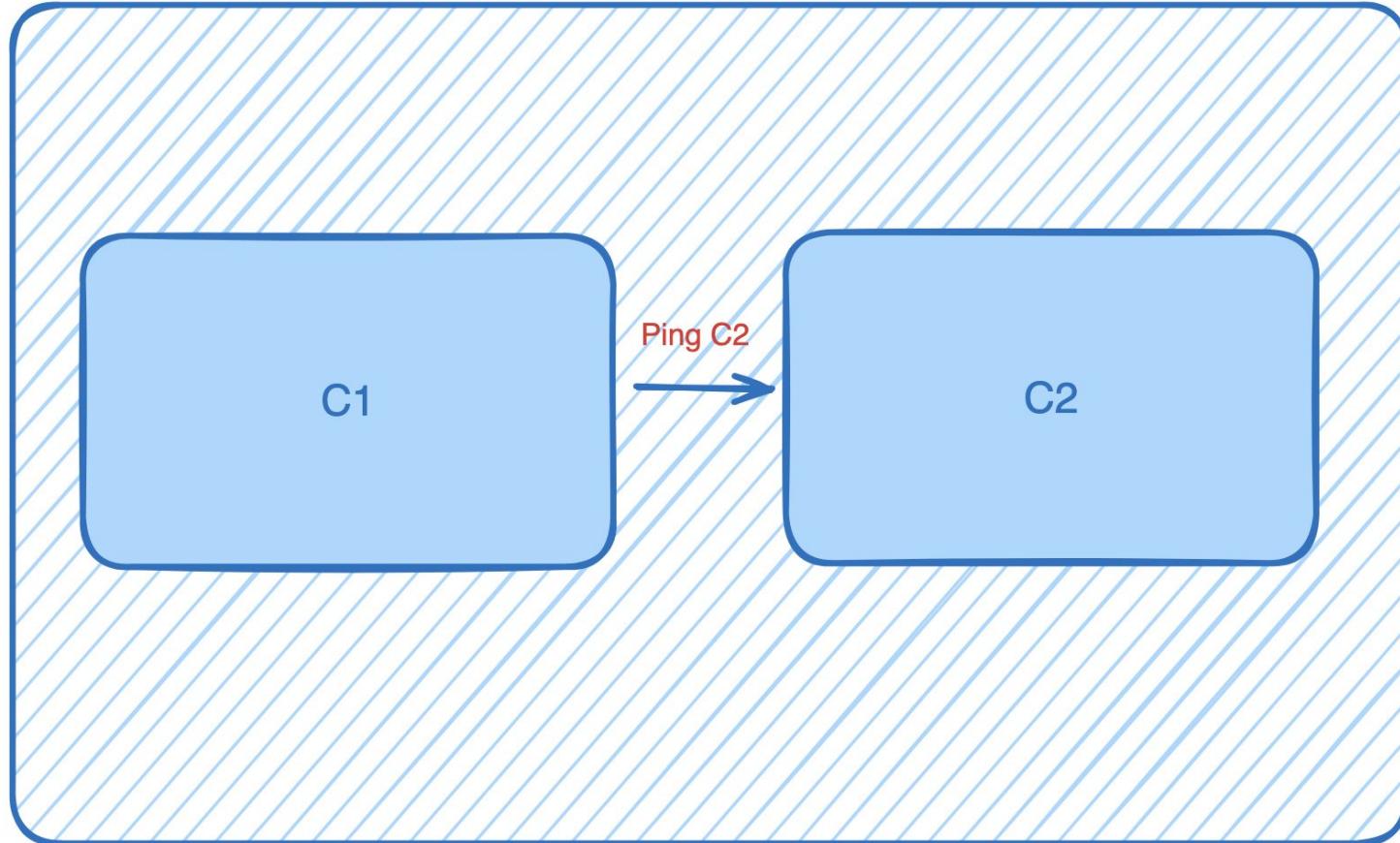
Connect Container



Explore Network in Docker

```
$ docker network ls
```

```
$ docker network inspect {network id}
```



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
```

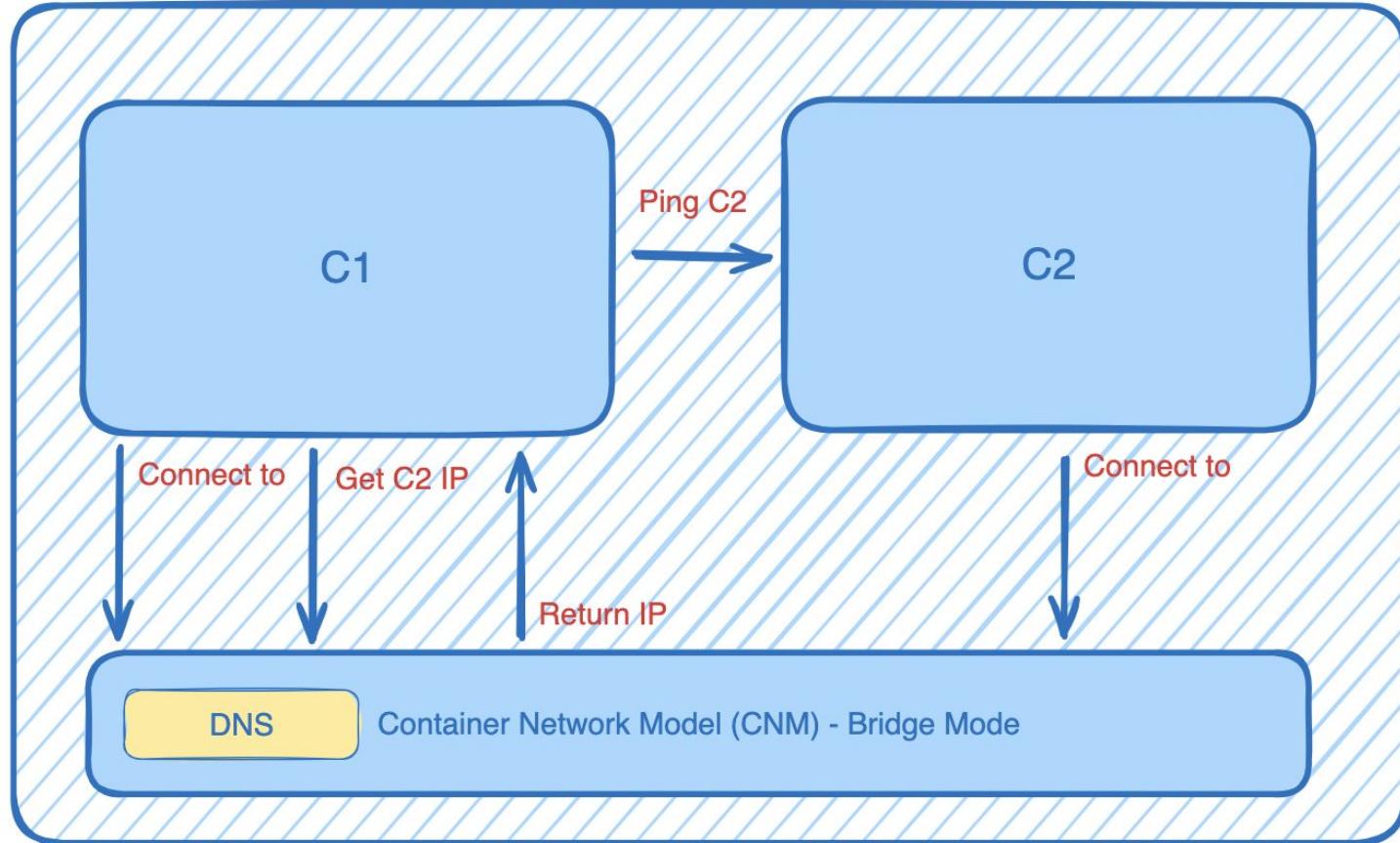
```
$ docker network connect sample-net c1
```

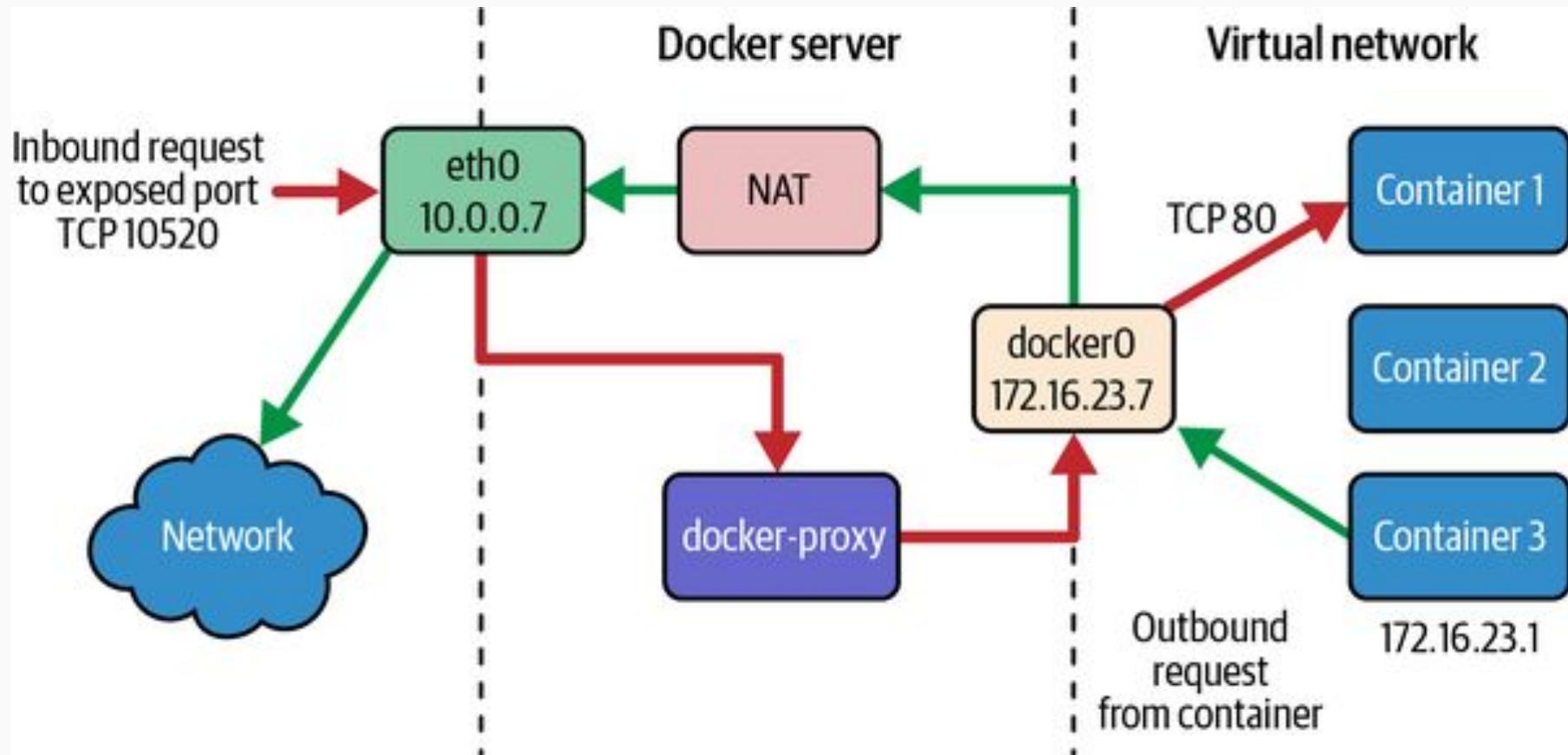
```
$ docker network connect sample-net c2
```

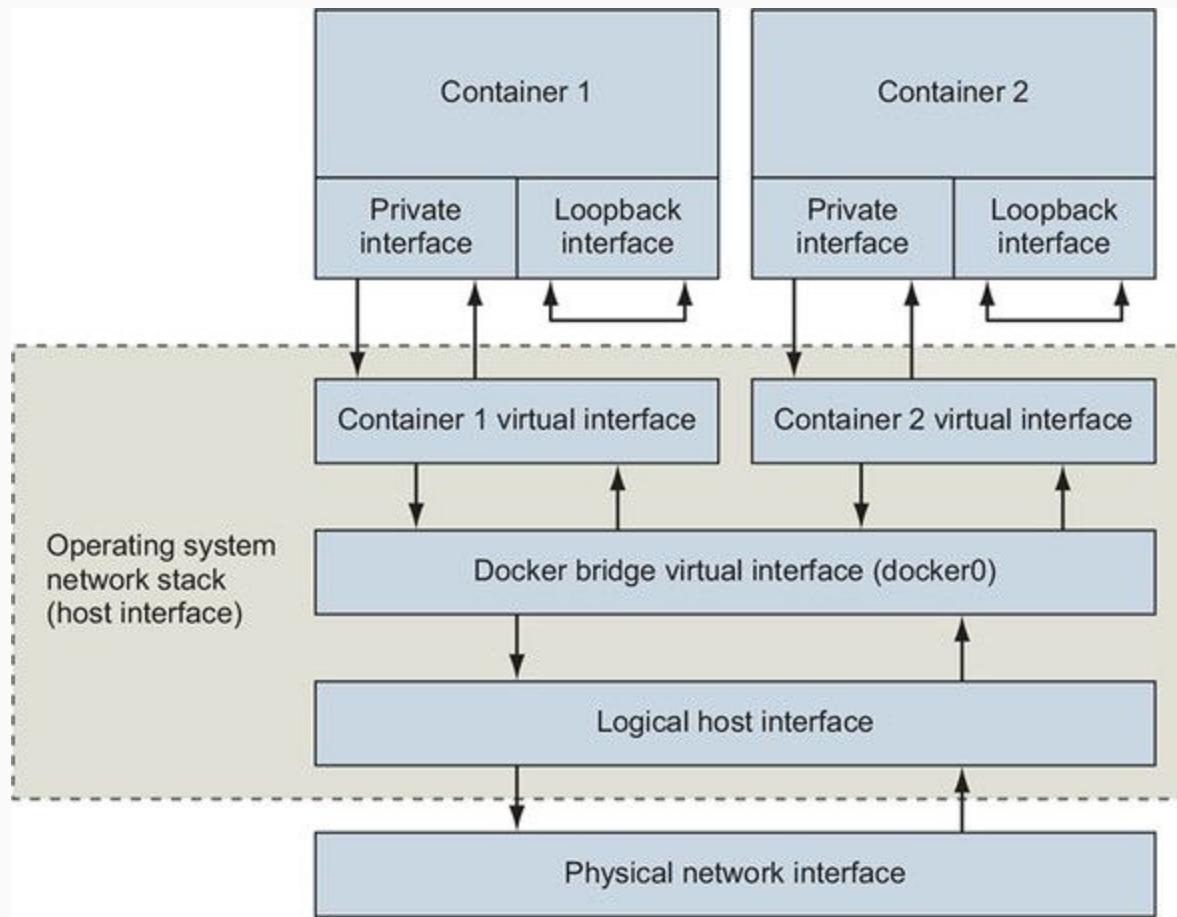
```
$ docker container exec -it c1 sh
```

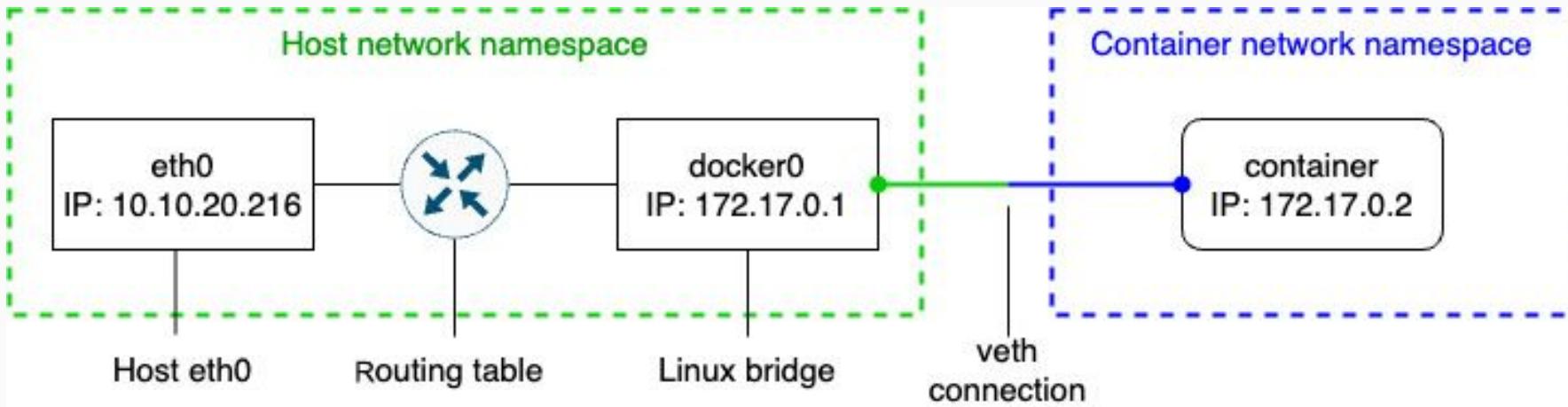
```
# ping c2
```

```
$ docker container inspect c1
```









A large cargo ship is docked at a port terminal. In the background, several tall industrial cranes with complex lattice booms are visible against a clear blue sky. The ship's hull is dark, and its deck is covered with numerous shipping containers stacked in neat rows. The water in the foreground is calm.

Orchestrate with Docker Compose

Workshop

Deploy Postgres Cluster



```
# Create PostgreSQL Cluster in Docker via CLI
```

```
$ 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

$ docker container logs -f postgres
```

```
# 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
```

```
# Try to Login PGAdmin
```

```
http://localhost:5050
```

```
# Access to Our PostgreSQL
```

```
# Create PostgreSQL Cluster in Docker
```

```
# Connect our container together with Docker Network
```

```
$ docker network db-net
```

```
$ docker network connect db-net postgres
```

```
$ docker network connect db-net pgadmin
```

```
# Create PostgreSQL Cluster in Docker
```

```
$ touch docker-compose.yml
```

```
$ code ./docker-compose.yml
```

```
# Create PostgreSQL Cluster in Docker  
# docker-compose.yml
```

```
services:  
  db:  
    image: postgres:alpine  
    environment:  
      - POSTGRES_USER=dockeruser  
      - POSTGRES_PASSWORD=dockerpass  
      - POSTGRES_DB=pets  
    volumes:  
      - pg-data:/var/lib/postgresql/data  
      - ./db:/docker-entrypoint-initdb.d
```

```
# Create PostgreSQL Cluster in Docker  
# docker-compose.yml
```

```
pgadmin:  
    image: dpage/pgadmin4  
    ports:  
        - "5050:80"  
    environment:  
        PGADMIN_DEFAULT_EMAIL: admin@acme.com  
        PGADMIN_DEFAULT_PASSWORD: admin  
    volumes:  
        - pgadmin-data:/var/lib/pgadmin
```

```
# Create PostgreSQL Cluster in Docker  
# docker-compose.yml
```

```
    volumes:  
      pg-data:  
      pgadmin-data:
```

```
# Create PostgreSQL Cluster in Docker
```

```
$ docker compose build {services...}
```

```
$ docker compose up
```

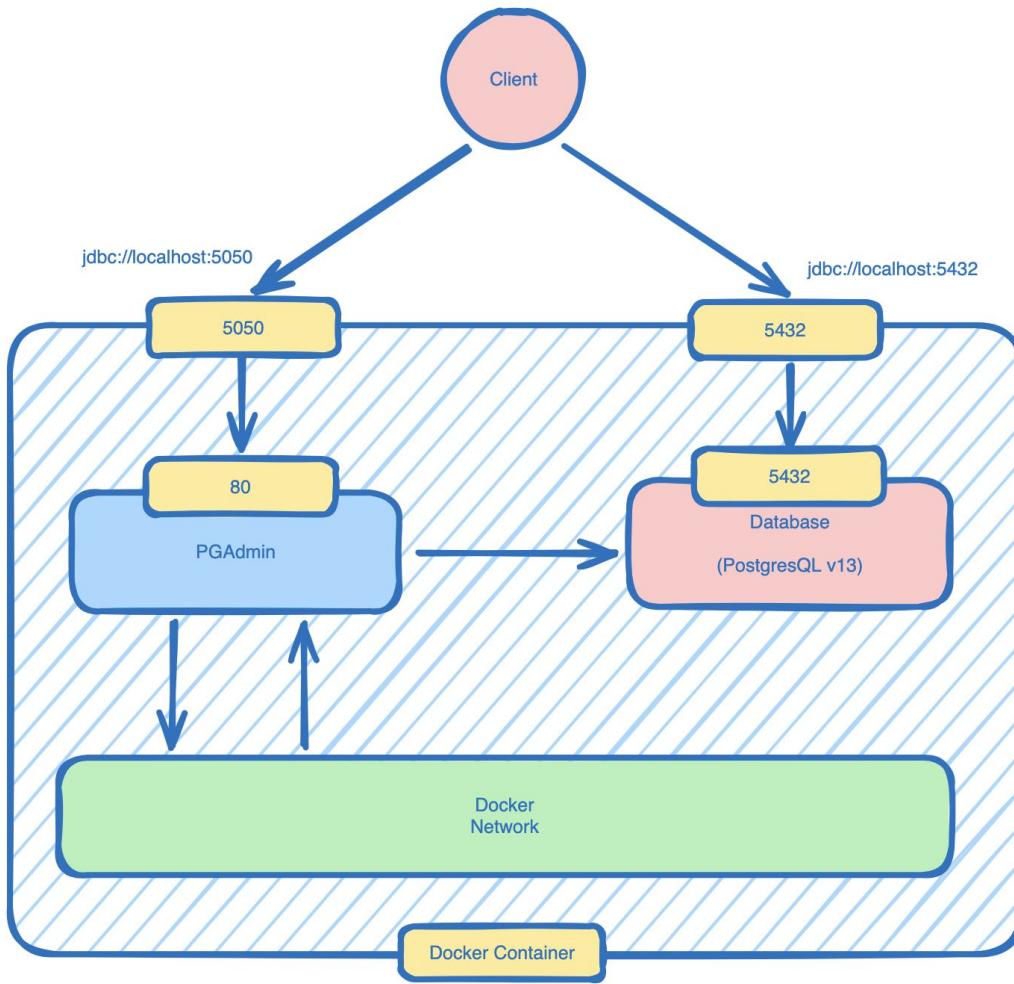
```
$ docker compose up --detach
```

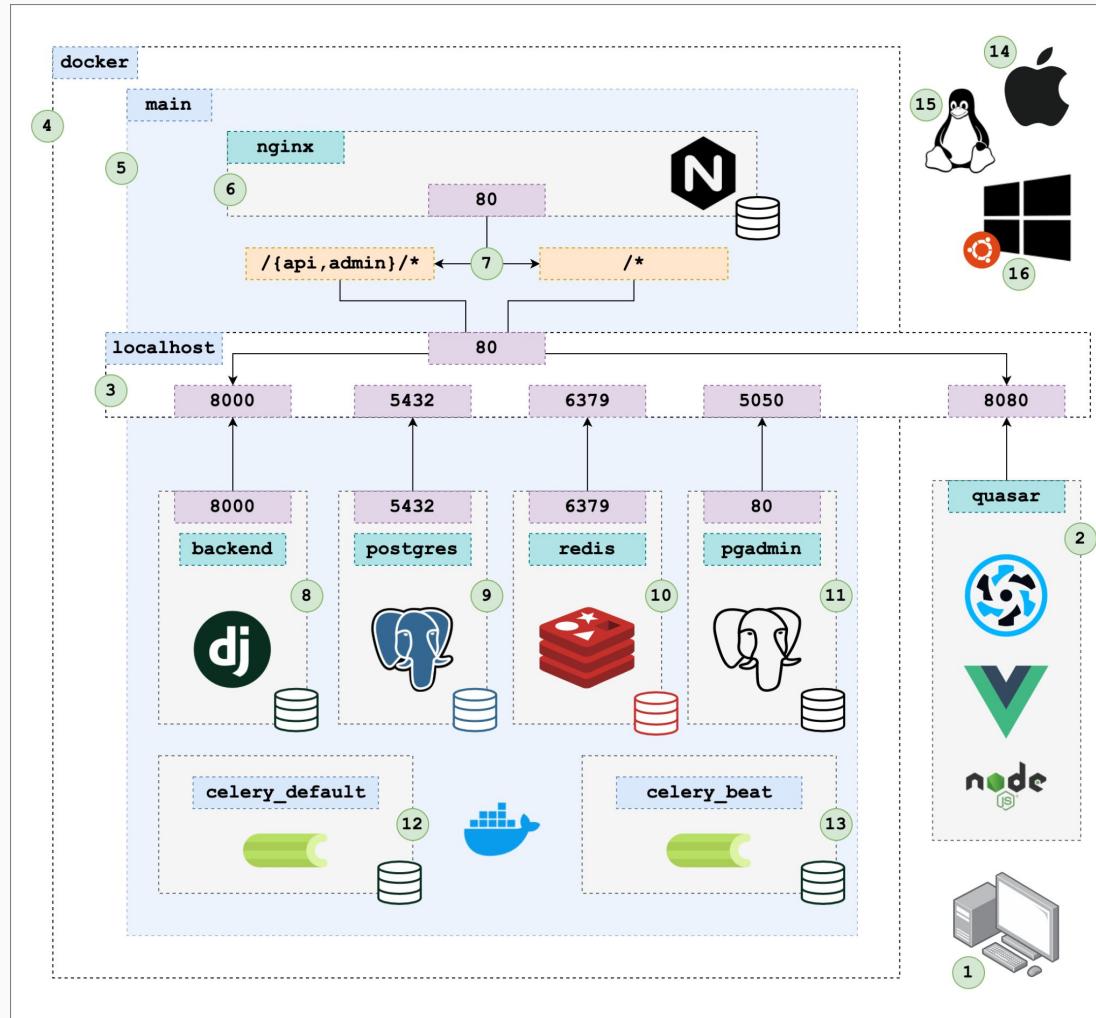
```
$ docker compose ls
```

```
$ docker container ps
```

```
$ docker compose logs
```

```
$ docker compose down
```

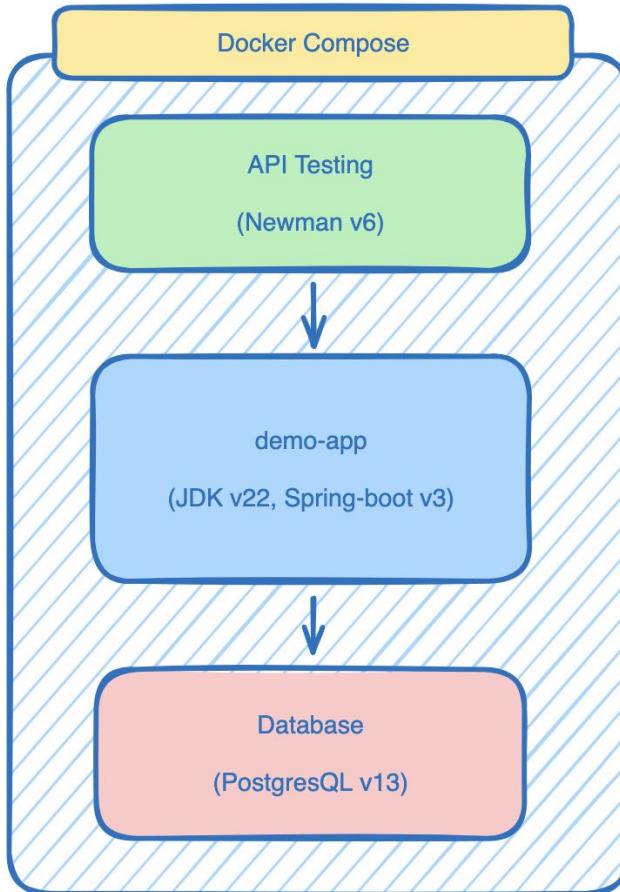




Workshop

Using Docker Compose for Development





Docker Compose.yml

?

```
# Build Spring-boot-app
```

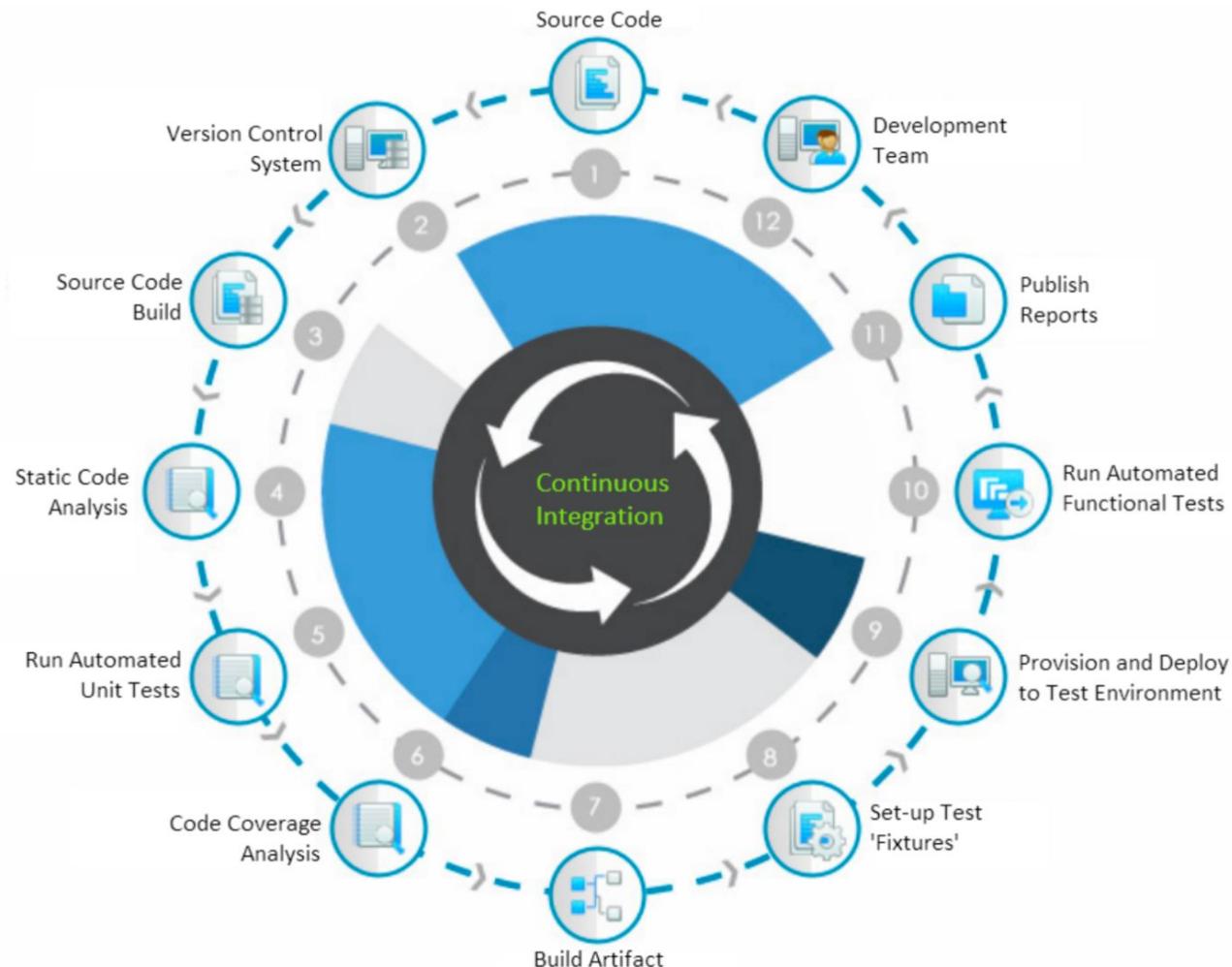
```
$ ./mvnw clean install
```

```
$ docker compose up
```

```
$ docker compose ls
```

```
$ docker container ps
```

```
$ docker compose down
```



Docker Best Practices

1. Only launch one application per container
2. Only install what you need
3. Keep image smallest as possible
4. In production think about

Develop Cloud Native with Twelve Factors Principle



What is 12 Factors Principle for Cloud Native?

- The factors represent a **set of guidelines or best practices** for **portable, resilient** applications that will thrive in cloud environments (specifically software as a service applications).
- The Principle was published in 2011 by **Adam Wiggins** (Co-Founder of Heroku)

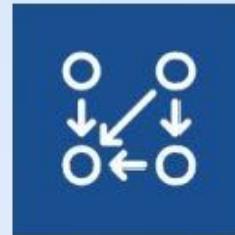


12 FACTOR APP

12 FACTOR APP



Codebase



Dependencies



Configuration



Backing Services



Build, Release, Run



Processes



Port Binding



Concurrency



Disposability



Dev/Prod Parity



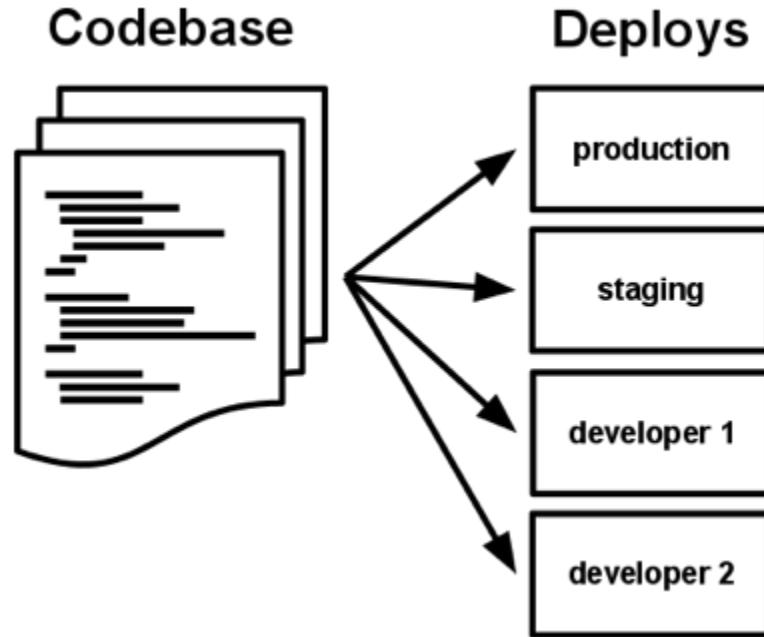
Logs



Admin Processes

Codebase

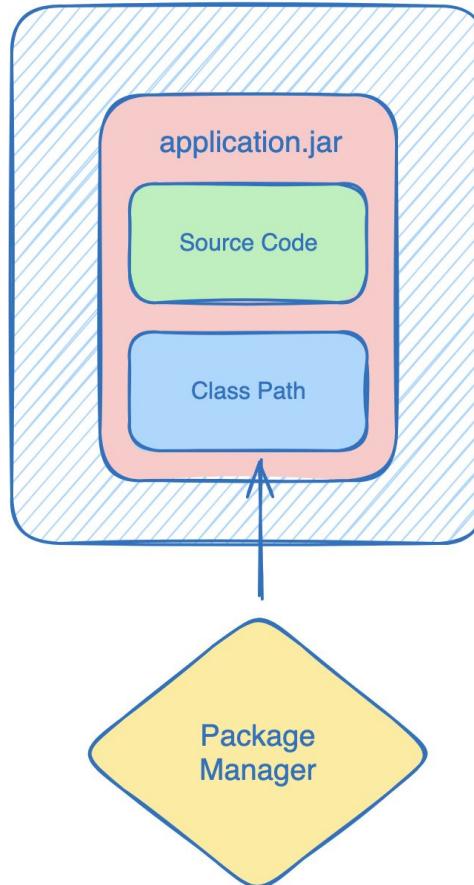
One codebase tracked in revision control, many deploys



II Dependencies

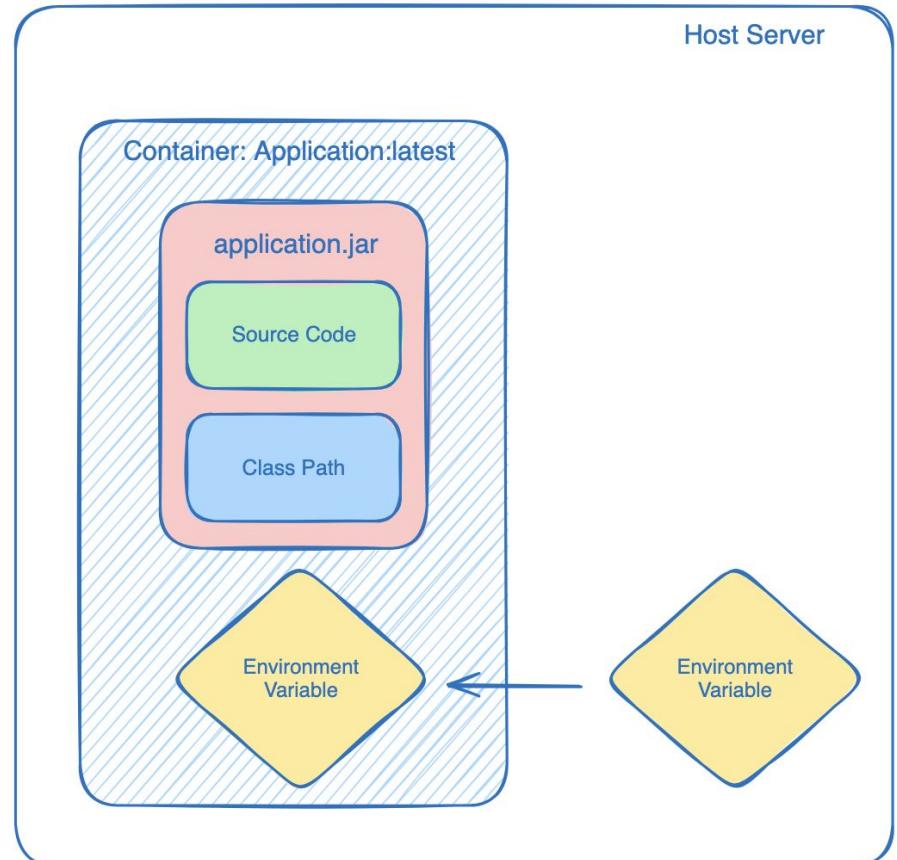
Explicitly declare and isolate dependencies

Container: Application:latest



III Config

Store config in the environment

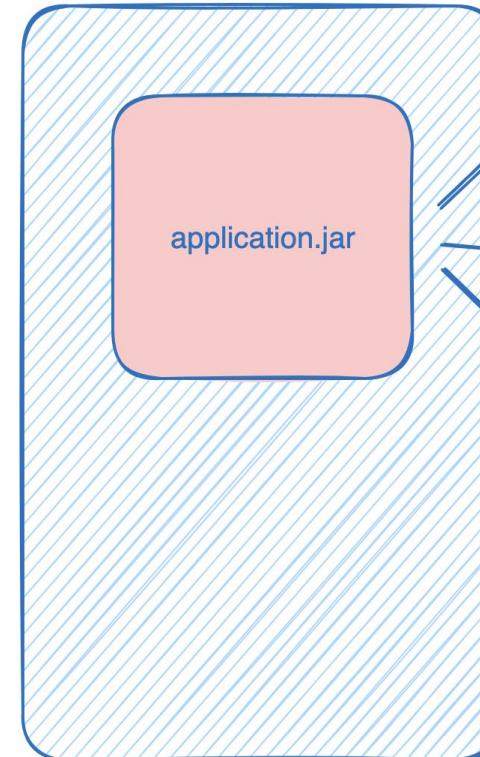


IV

Backing services

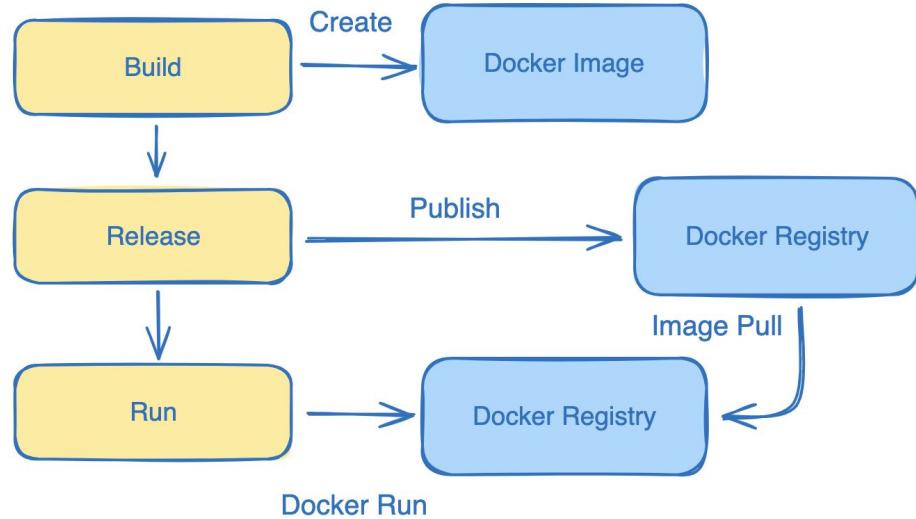
Treat backing services as attached resources

Container: Application:latest



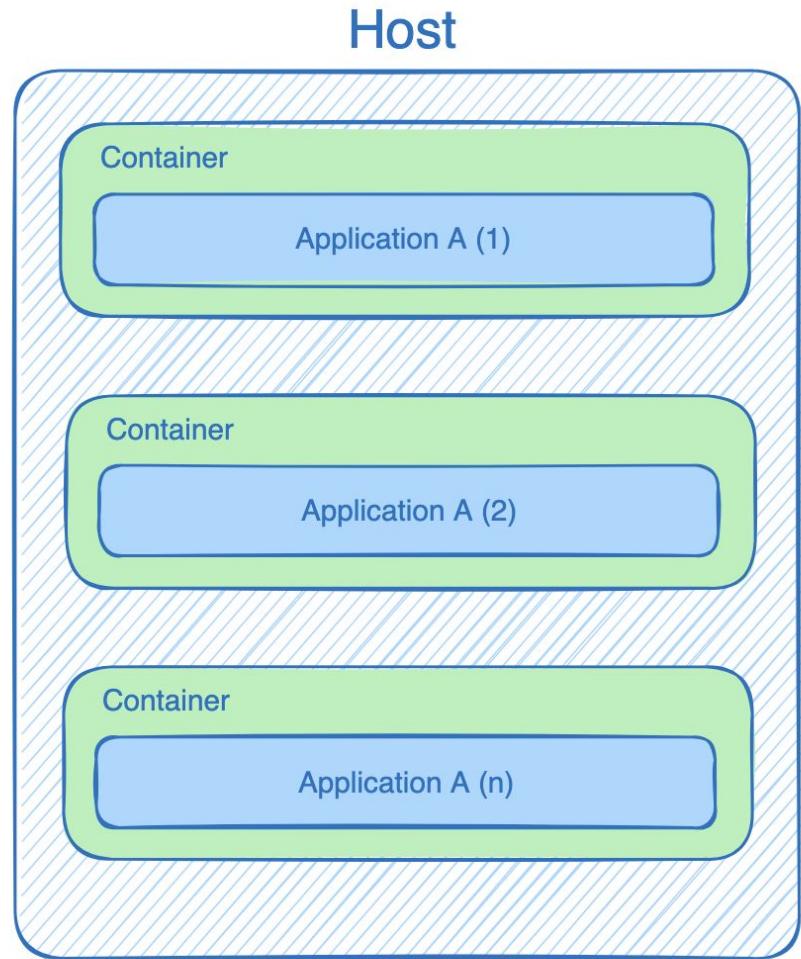
V Build, release, run

Strictly separate build and run stages



VI Processes

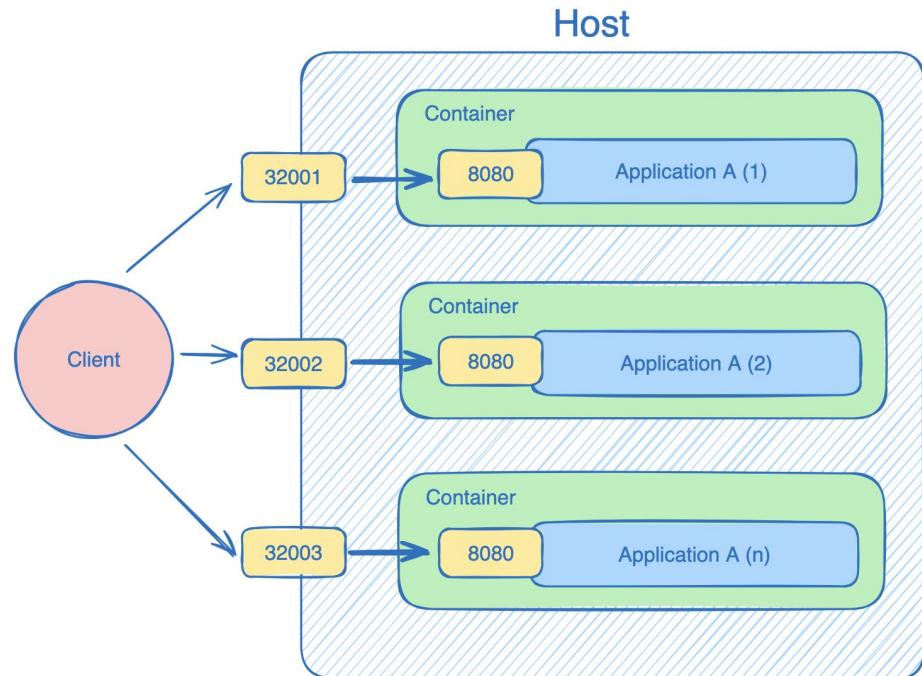
Execute the app as one or more stateless processes



VII

Port binding

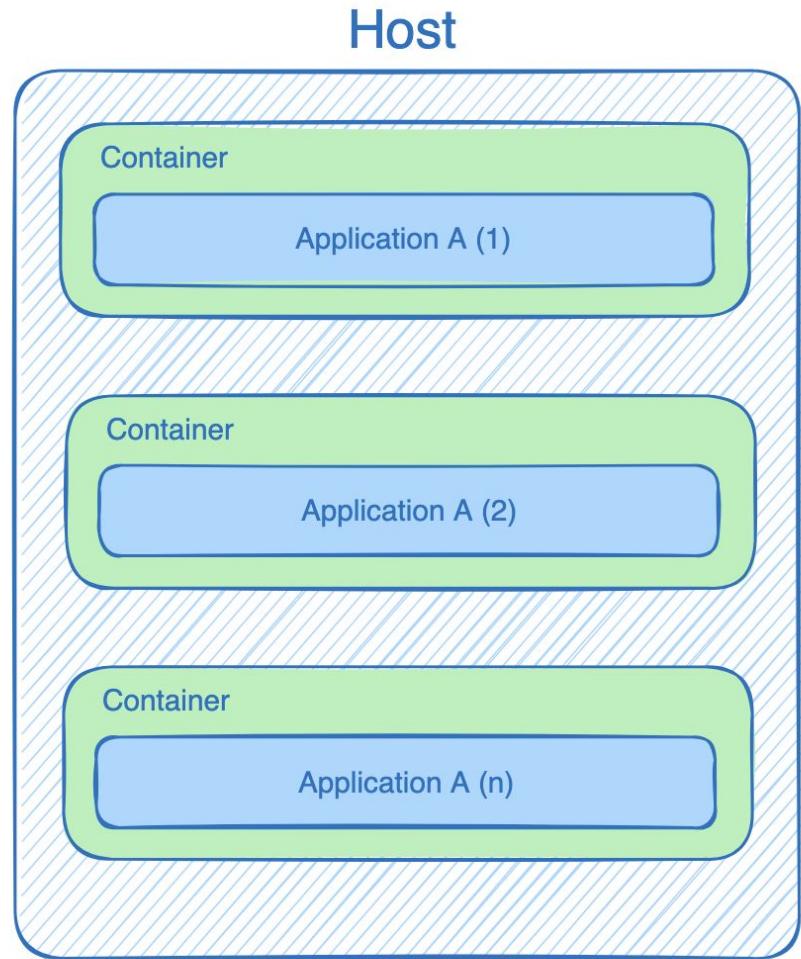
Export services via port binding



VIII

Concurrency

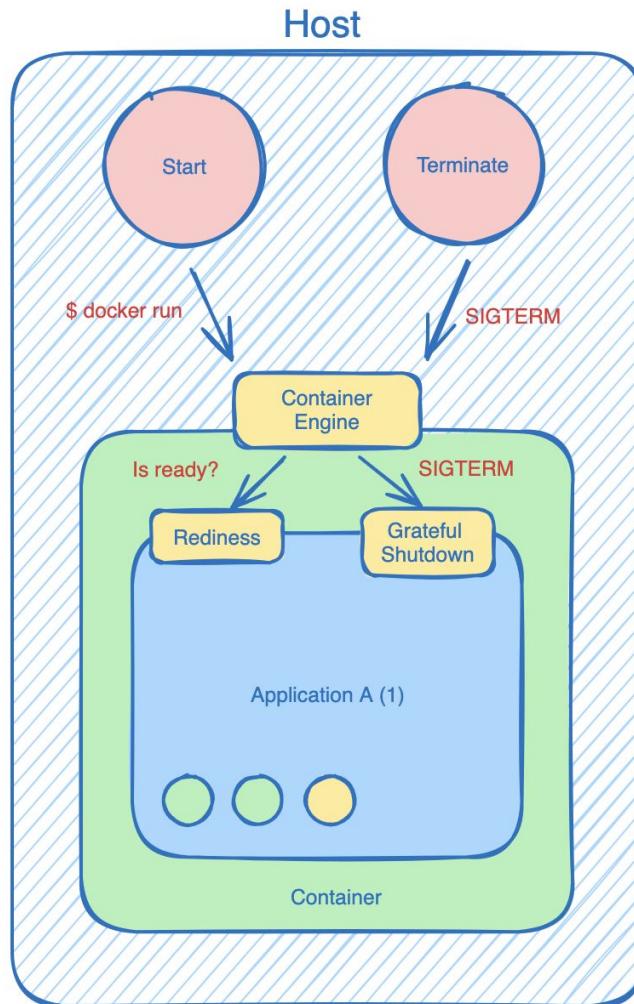
Scale out via the process model



IX

Disposability

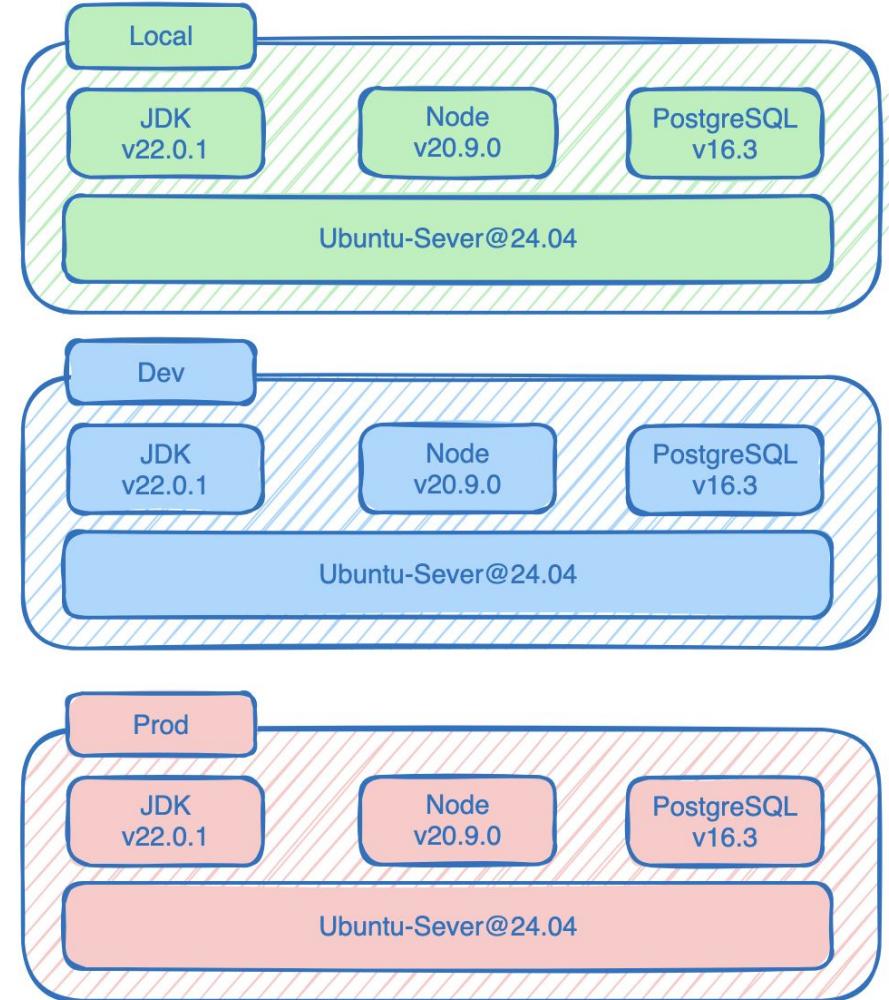
Maximize robustness with fast startup and graceful shutdown



X

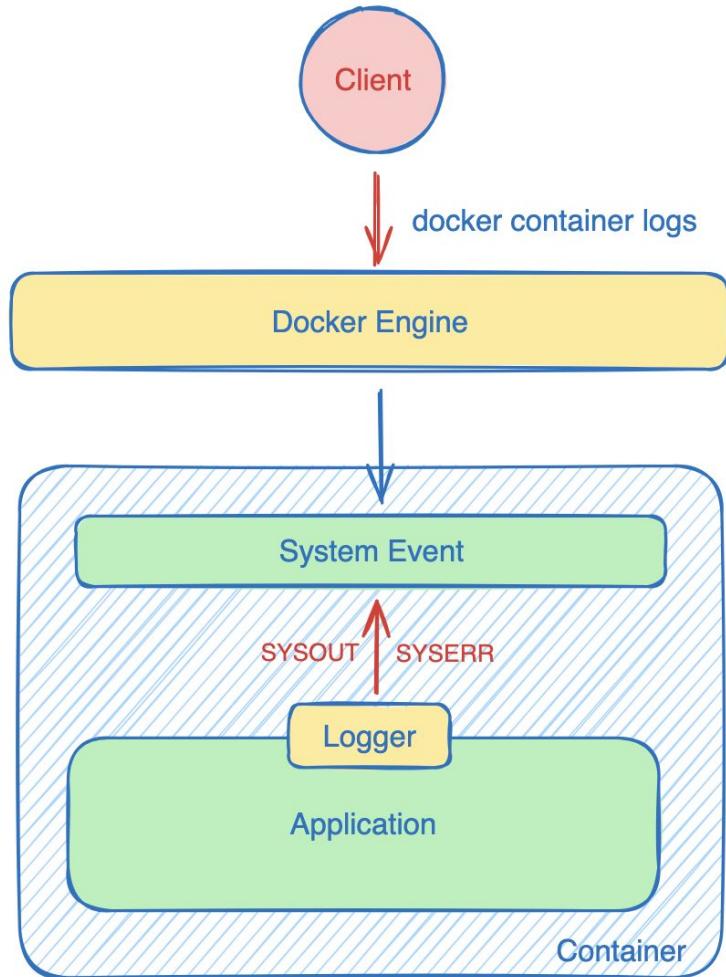
Dev/prod parity

Keep development, staging, and production as similar as possible



XI Logs

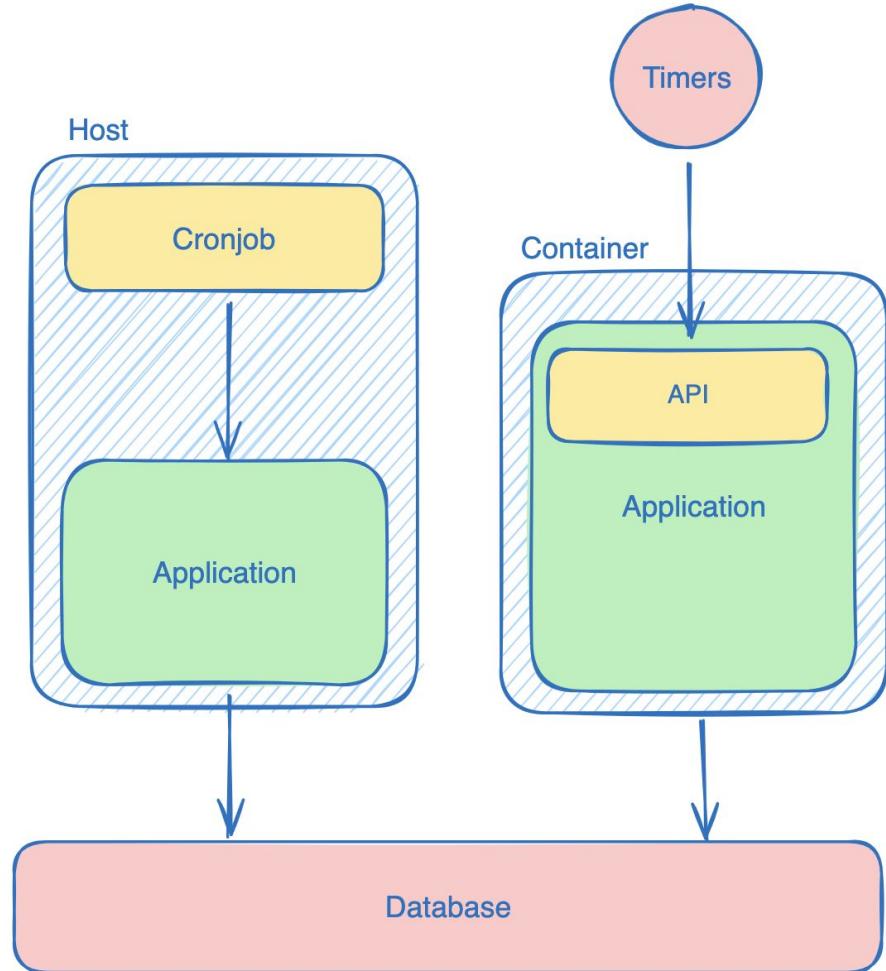
Treat logs as event streams



XII

Admin processes

Run admin/management tasks
as one-off processes



Scaling Application With Docker Swarm



Additional 3 Factors



12 Factors

Codebase

Dependency

Config

Backing Service

Build, Release,
Run

Processes

Port Binding

Concurrency

Disposability

Dev/Prod Parity

Logs

Admin
Processes



- API First
- Telemetry
- Authentication & Authorization

XIII

API First.

XIV

Telemetry

XV

Authentication & Authorization

Day 2

Manage Your Enterprise Application with Kubernetes (K8S)





kubernetes

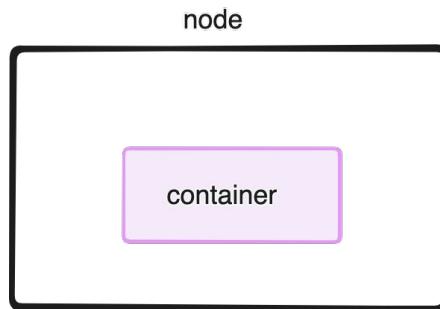
Agenda

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

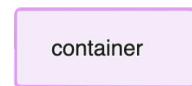
Introduction



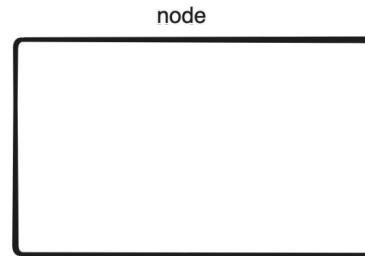
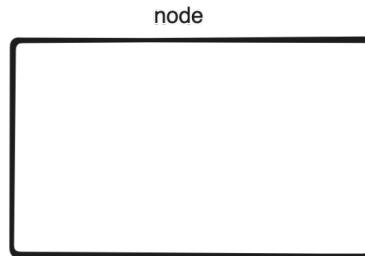
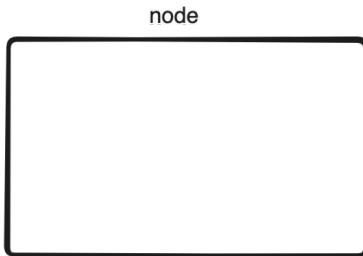
Docker on one server



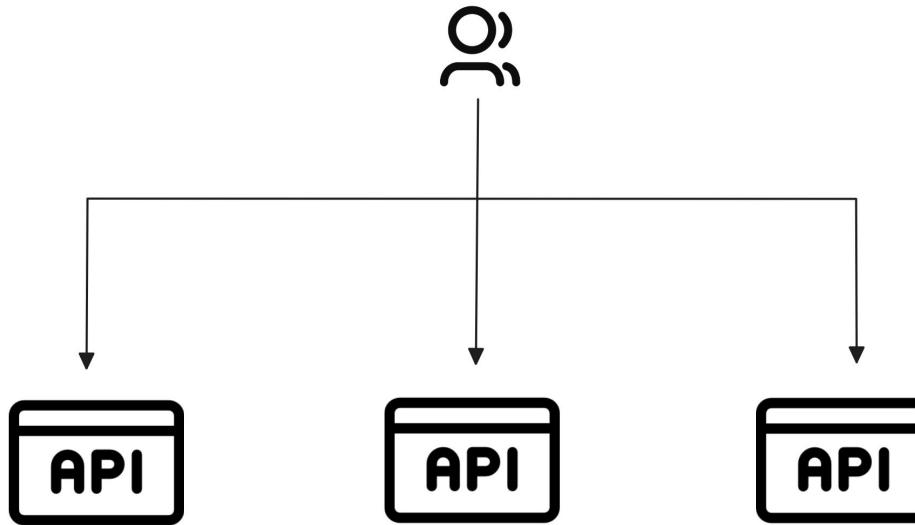
Docker on multi servers



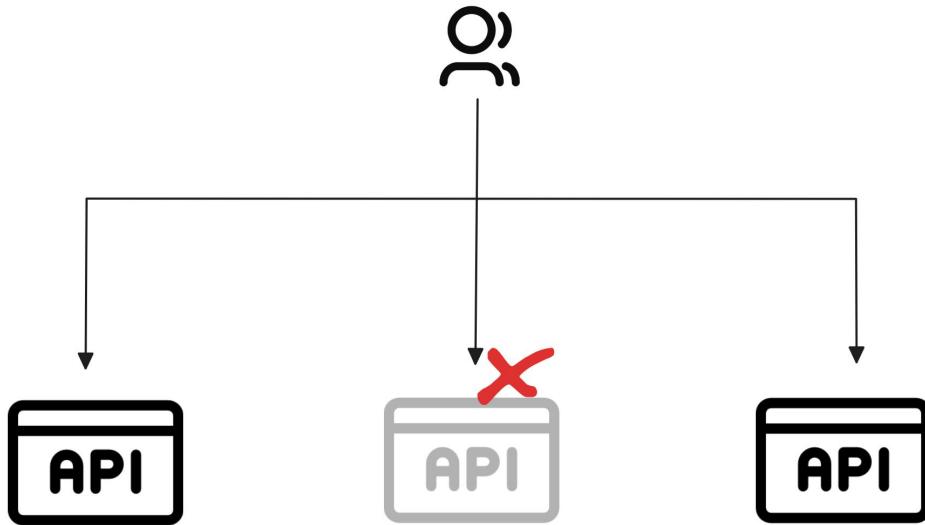
???



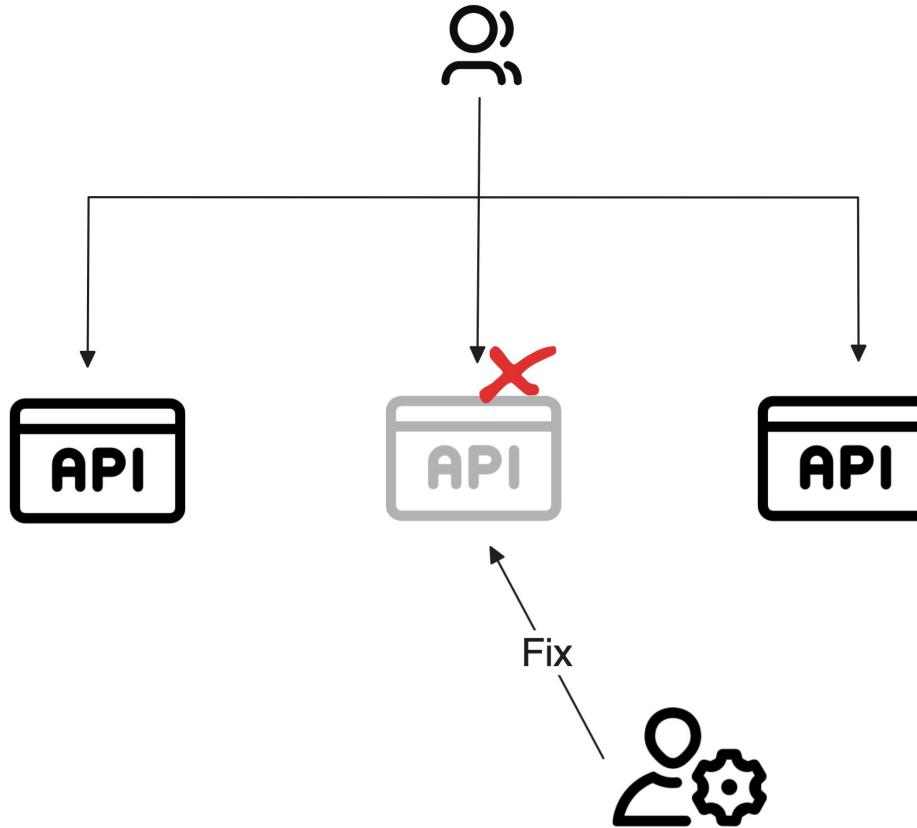
Applications problem



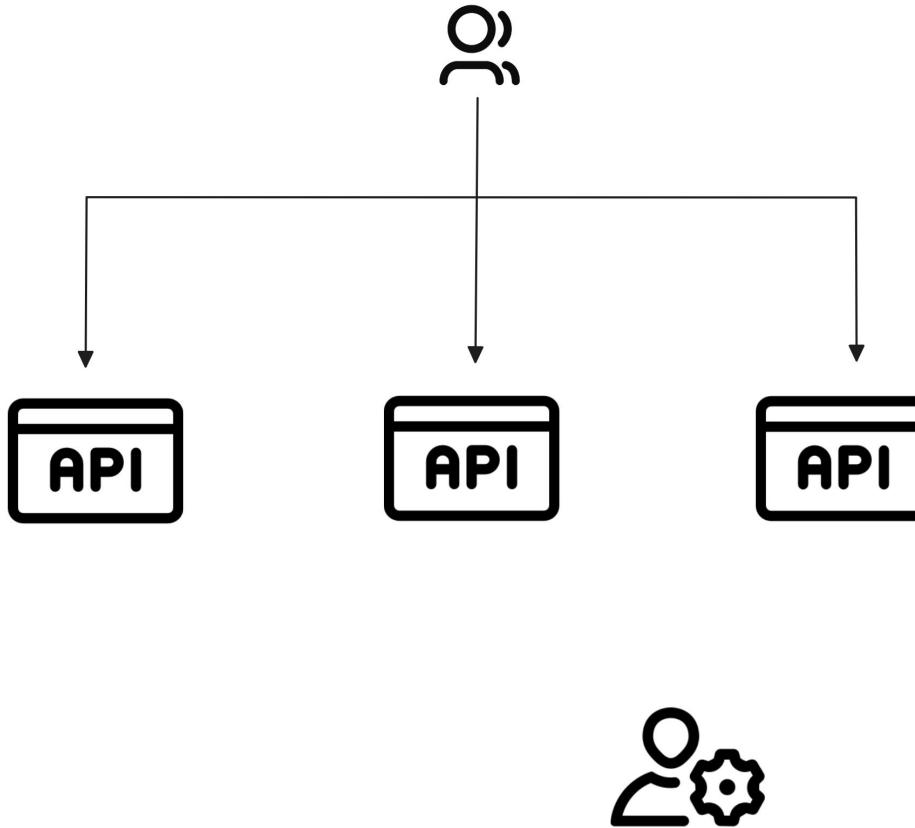
Applications problem



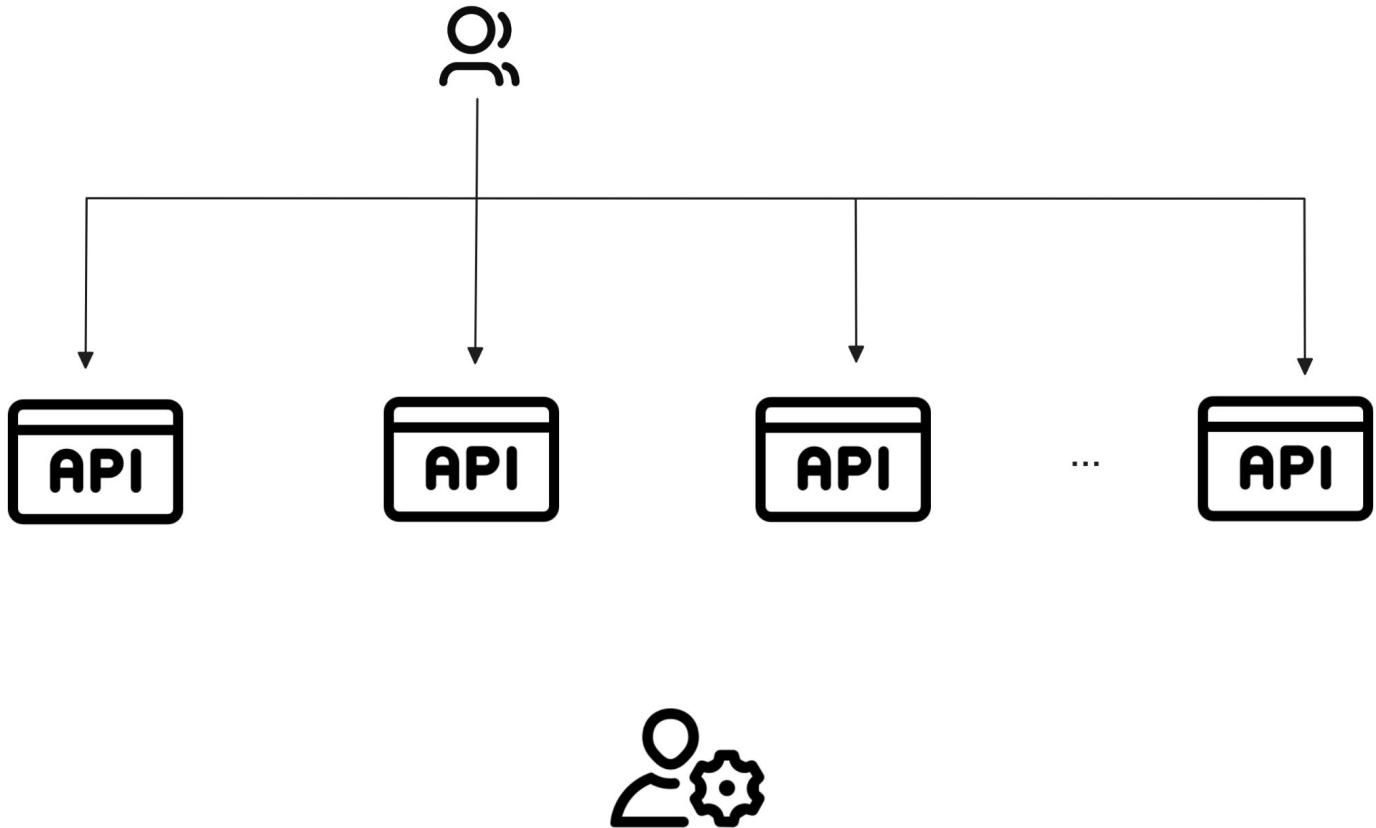
Applications problem



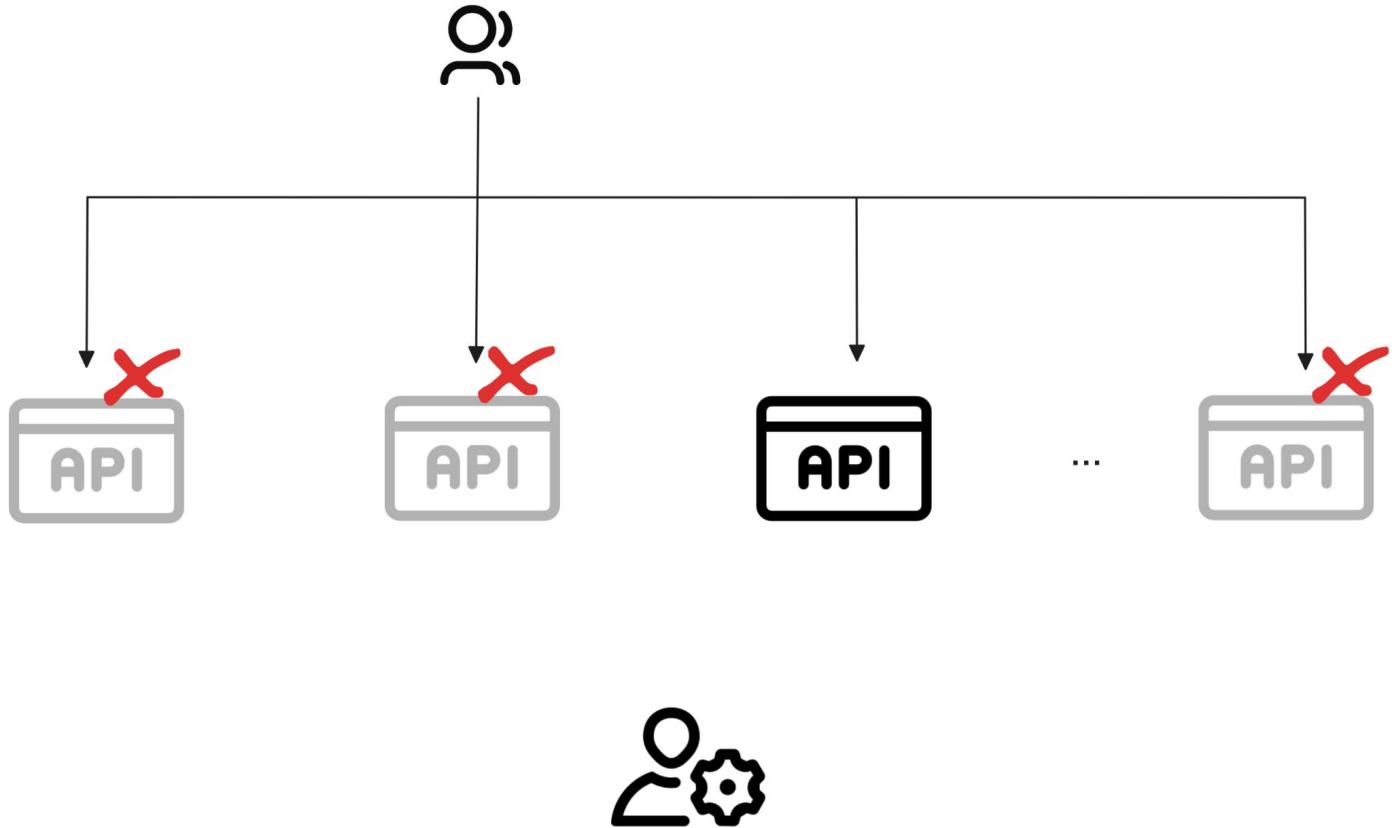
Applications problem



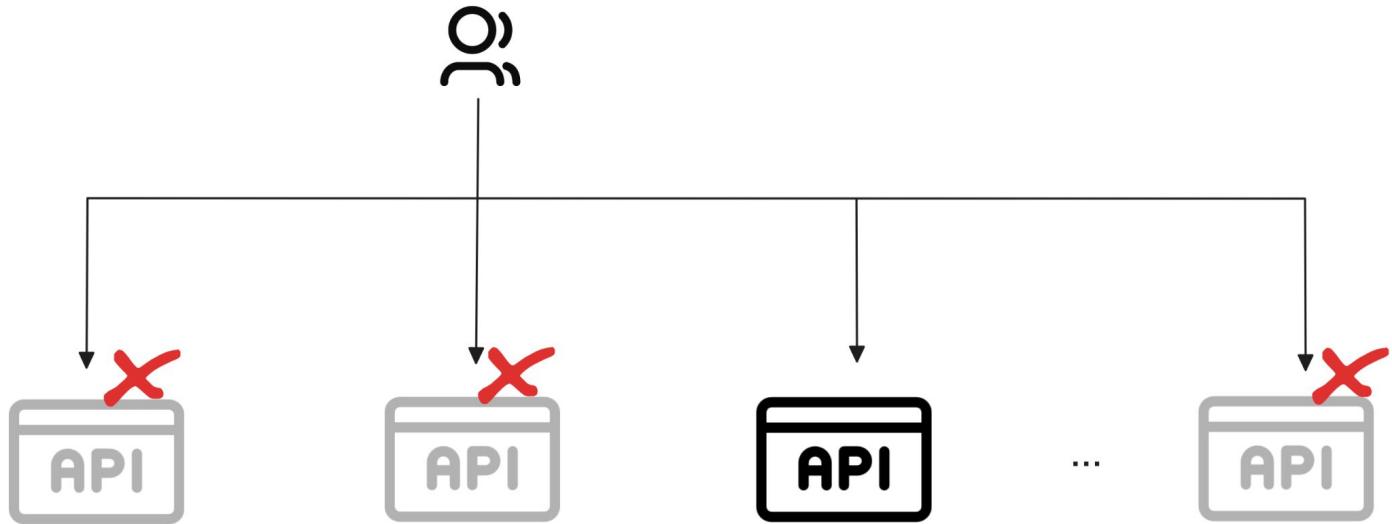
Applications problem



Applications problem



Applications problem

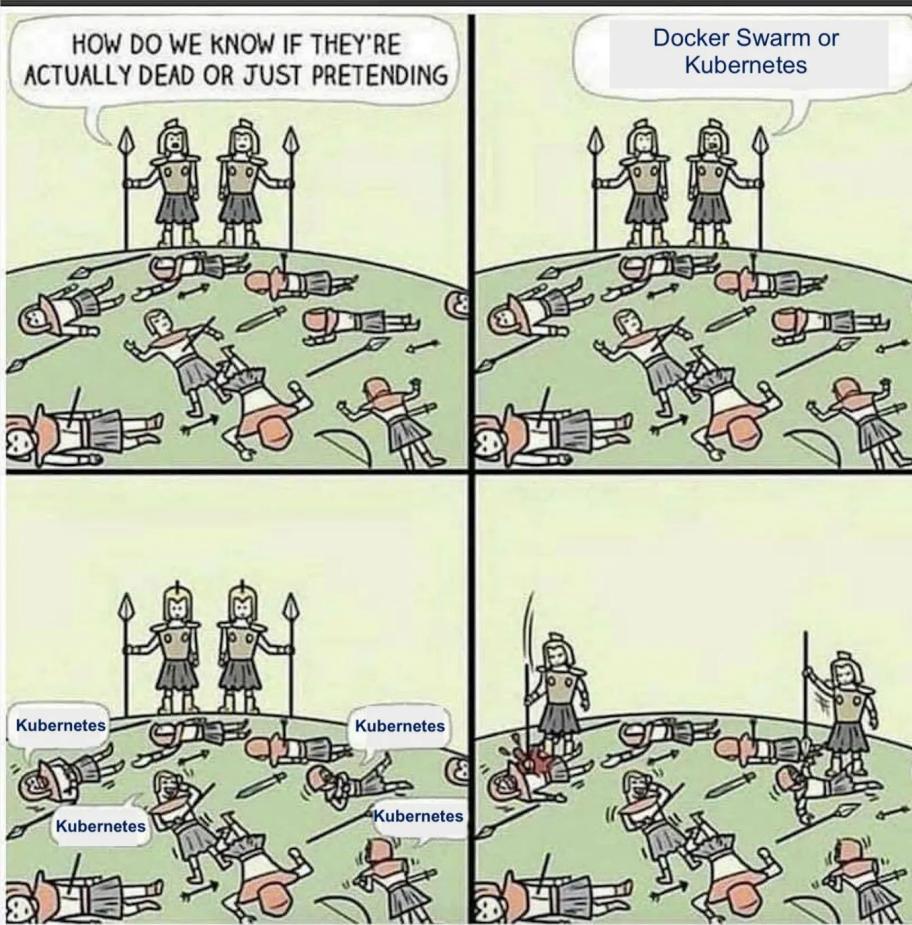


Fix !!!!

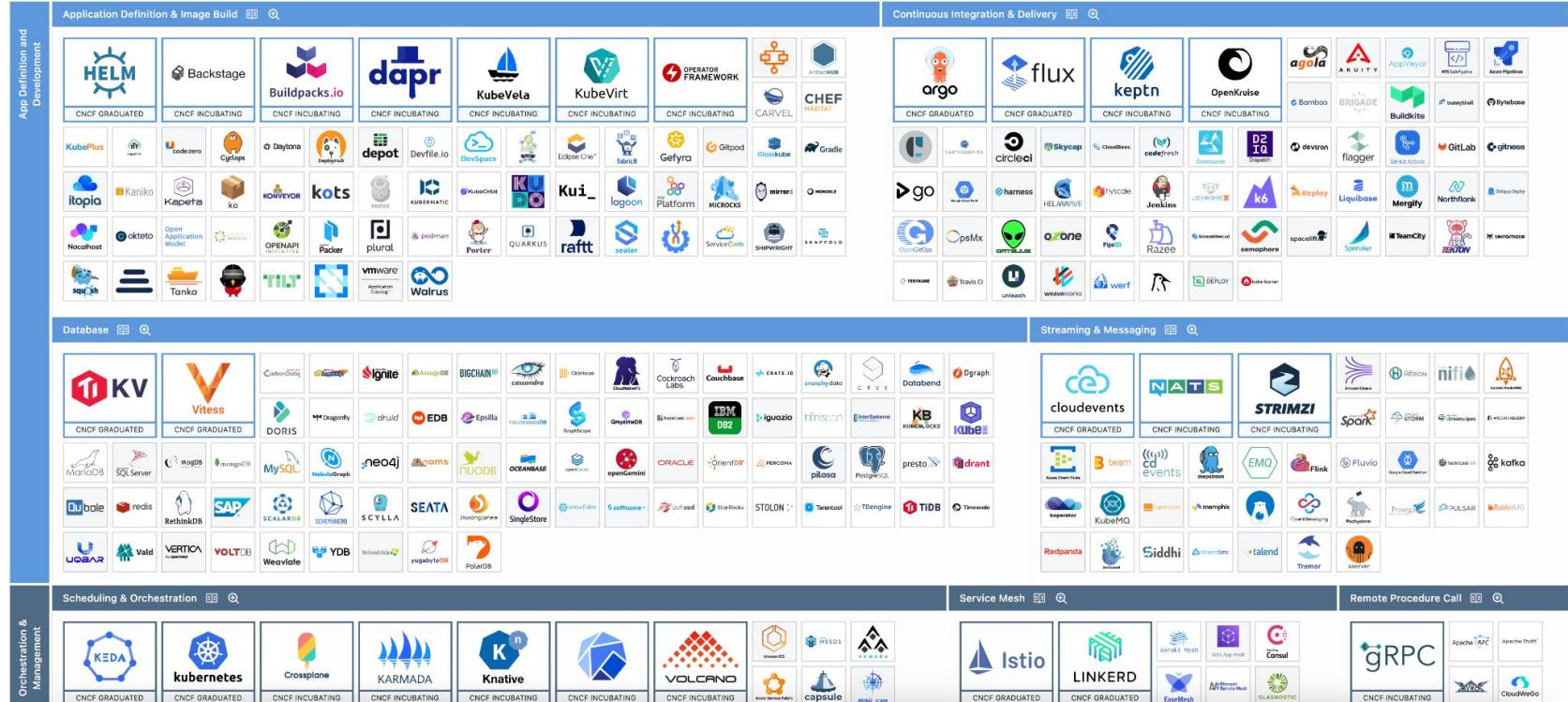


What is Kubernetes?

- Kubernetes in Greek means the Helmsman
- Also known as k8s
- Inspired and informed by Google's experiences and internal systems
- 100% open source, 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



CNCF Landscape



<https://landscape.cncf.io/>

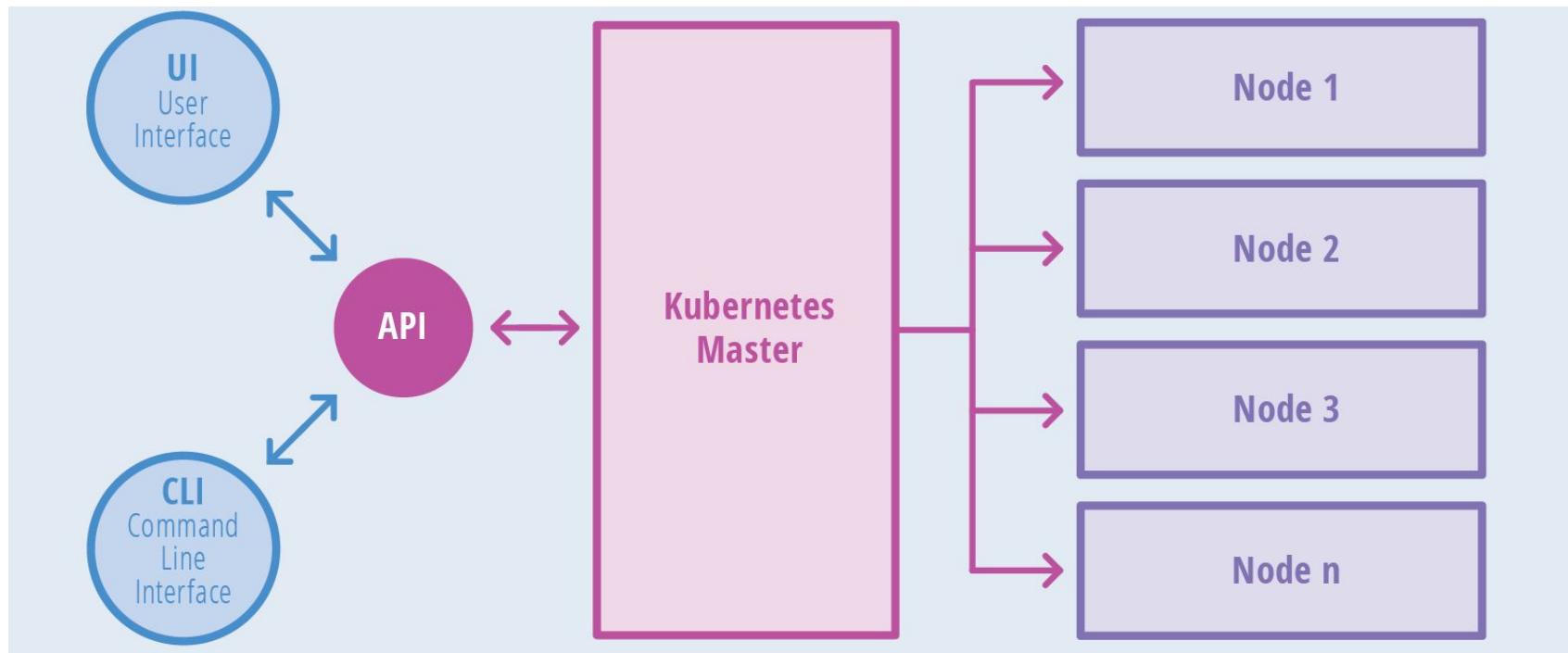
The features of Kubernetes

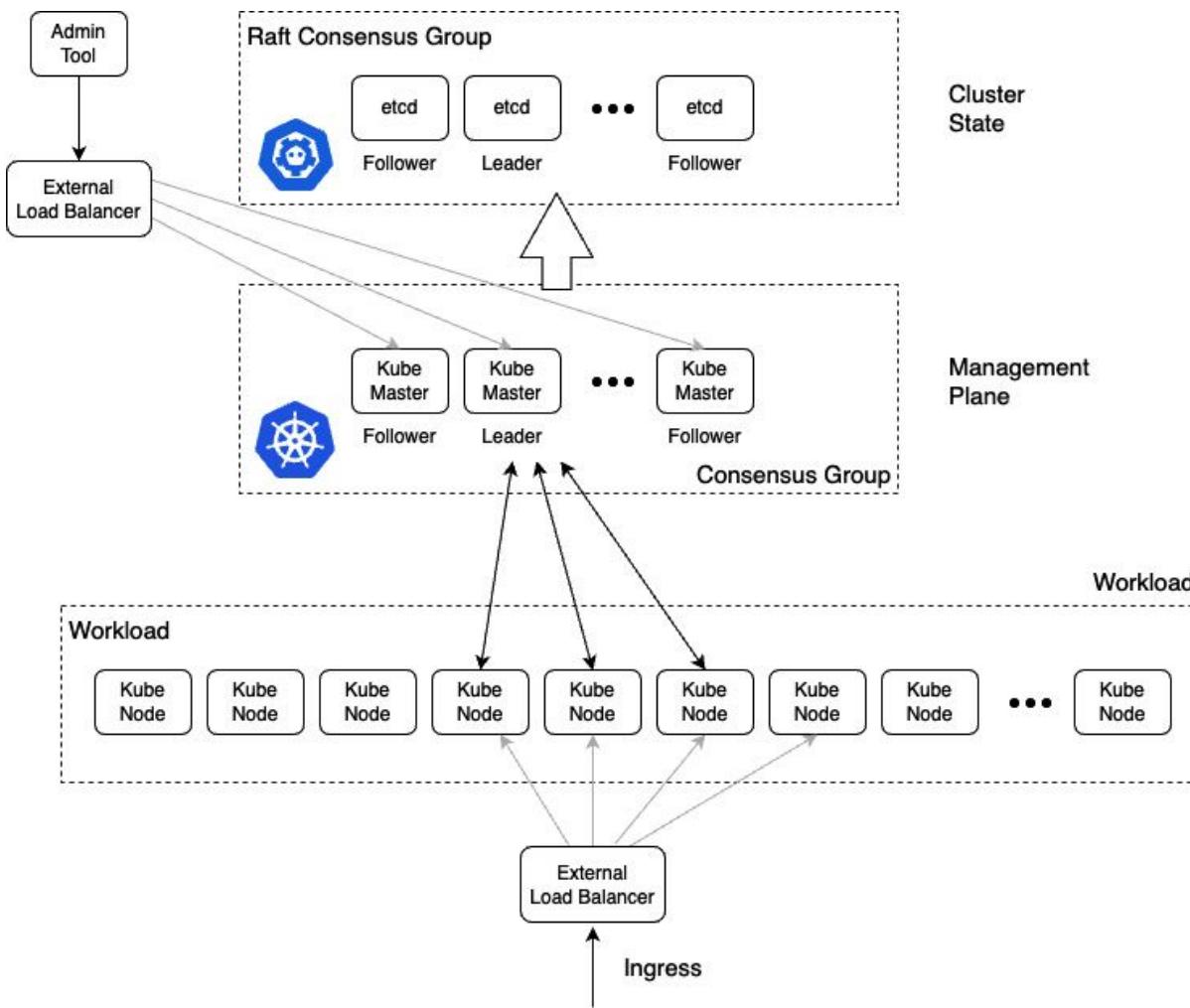
- Automatic bin packing
- Self healing
- Automated rollout & rollback
- Service Discovery & load Balancing
- Horizontal scaling
- Declarative Configuration

Kubernetes Architecture



Kubernetes Architecture

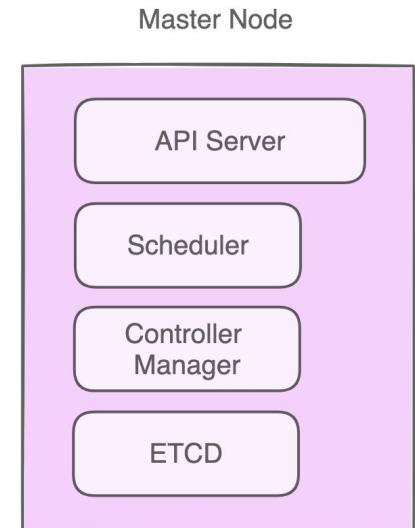




Kubernetes Architecture

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**

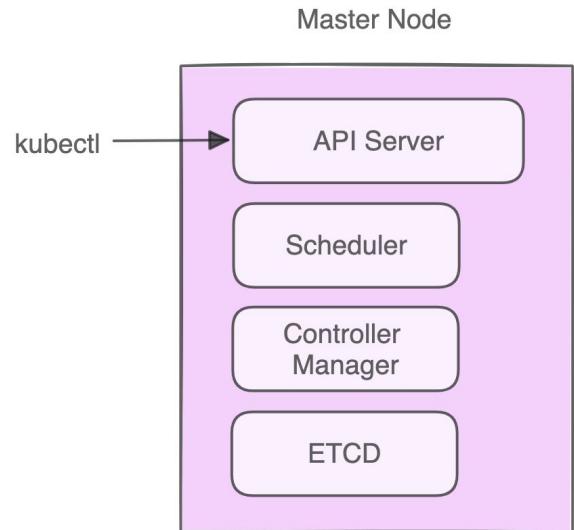


Kubernetes Architecture

Master node

API Server

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

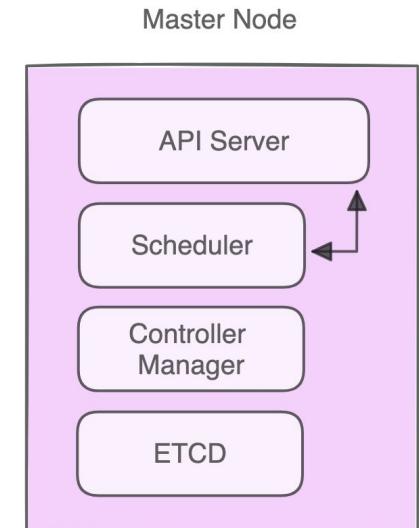


Kubernetes Architecture

Master node

Scheduler

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

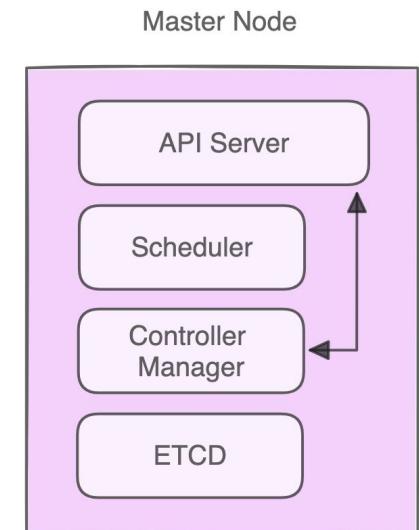


Kubernetes Architecture

Master node

Controller Manager

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

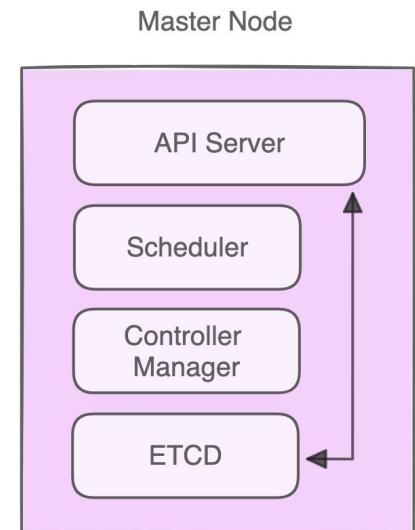


Kubernetes Architecture

Master node

ETCD

- ETCD is a distributed reliable key-value store used 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

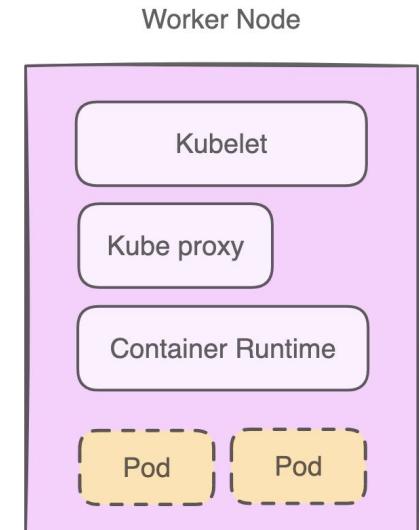


Kubernetes Architecture

Worker node

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

Kube Proxy

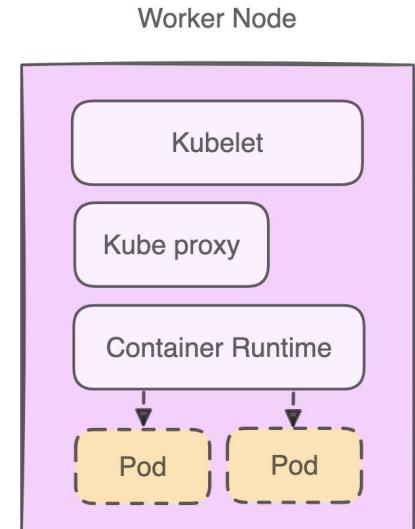


Kubernetes Architecture

Worker node

Container Runtime

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

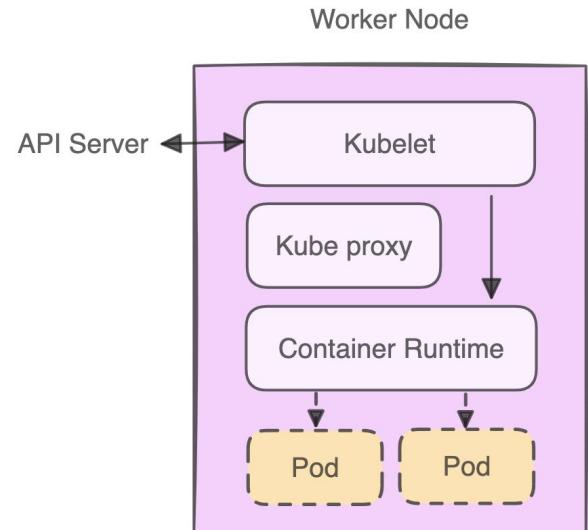


Kubernetes Architecture

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

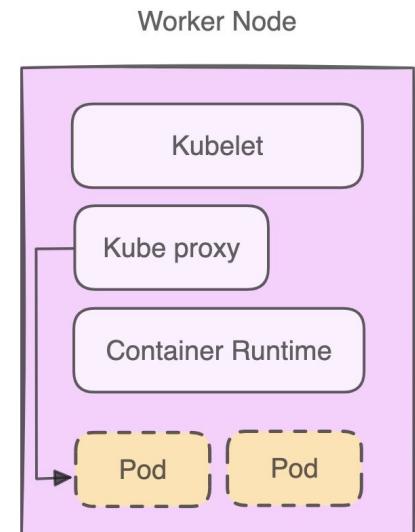


Kubernetes Architecture

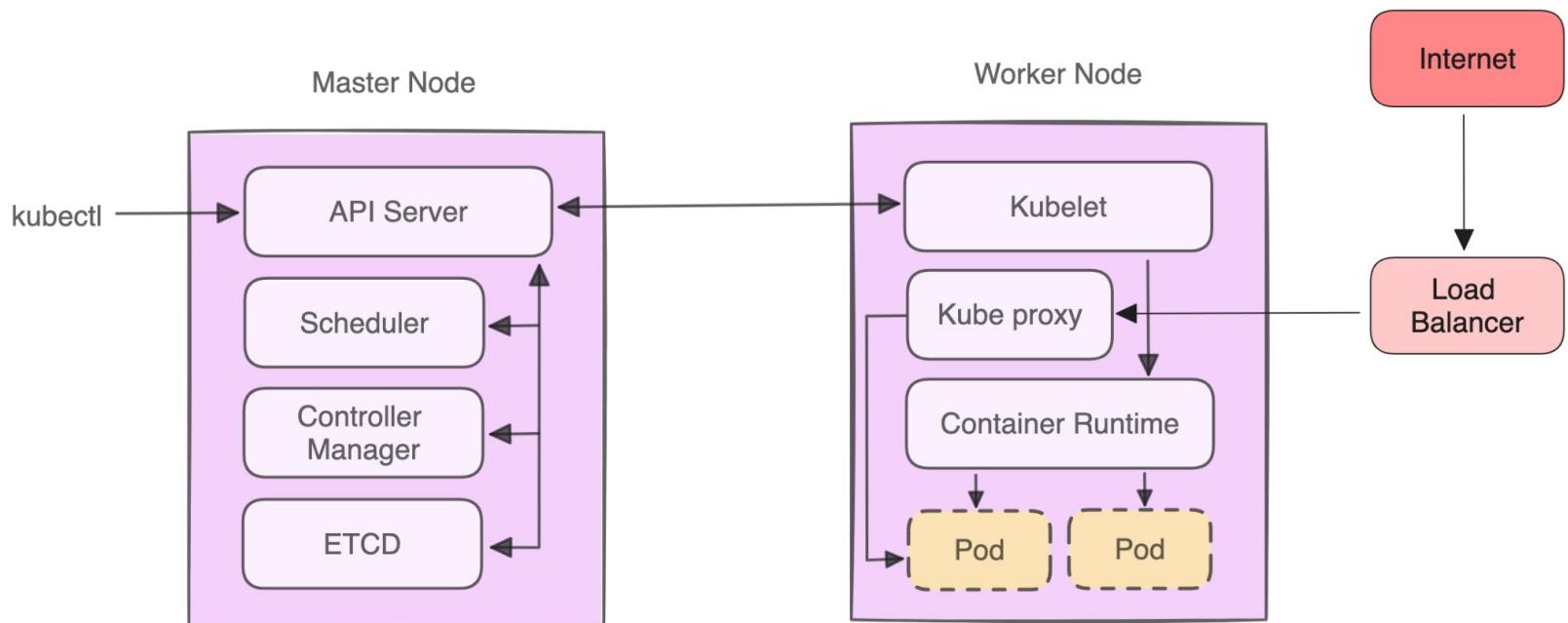
Worker node

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



Kubectl

Kubectl

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

Kubectl

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

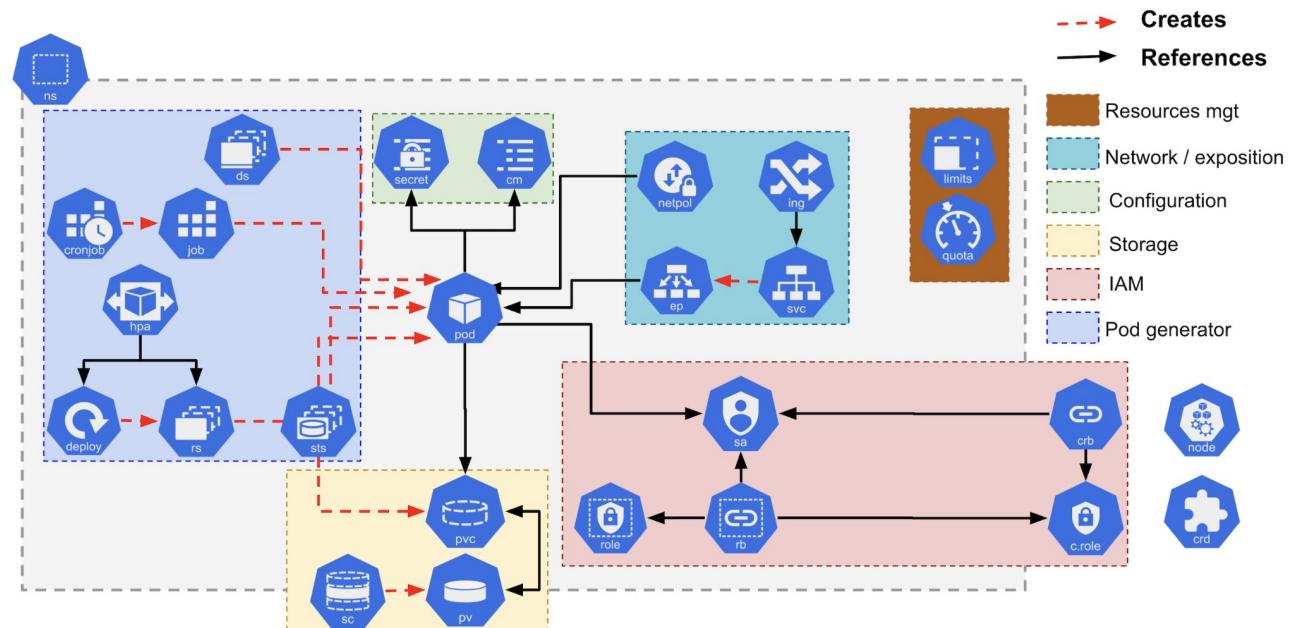
Resources / Objects



Resources / Objects

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

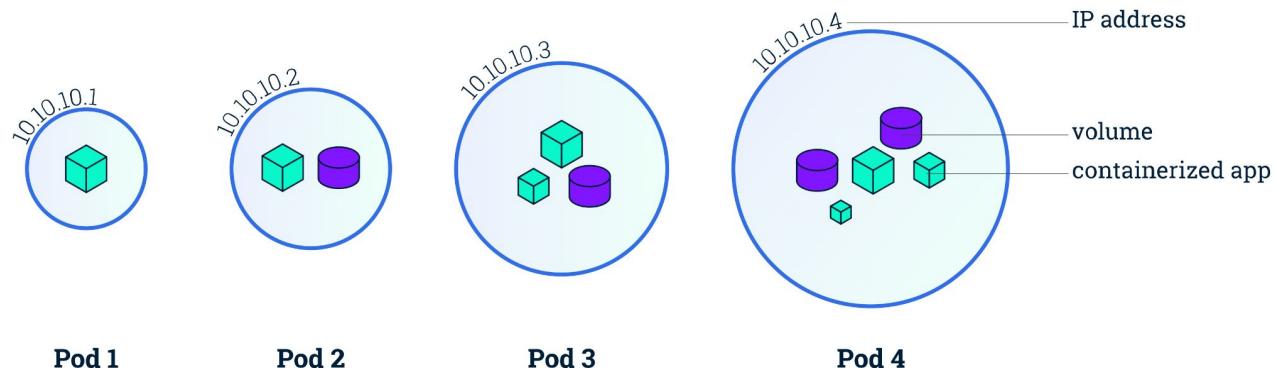
Kubernetes Resources Map



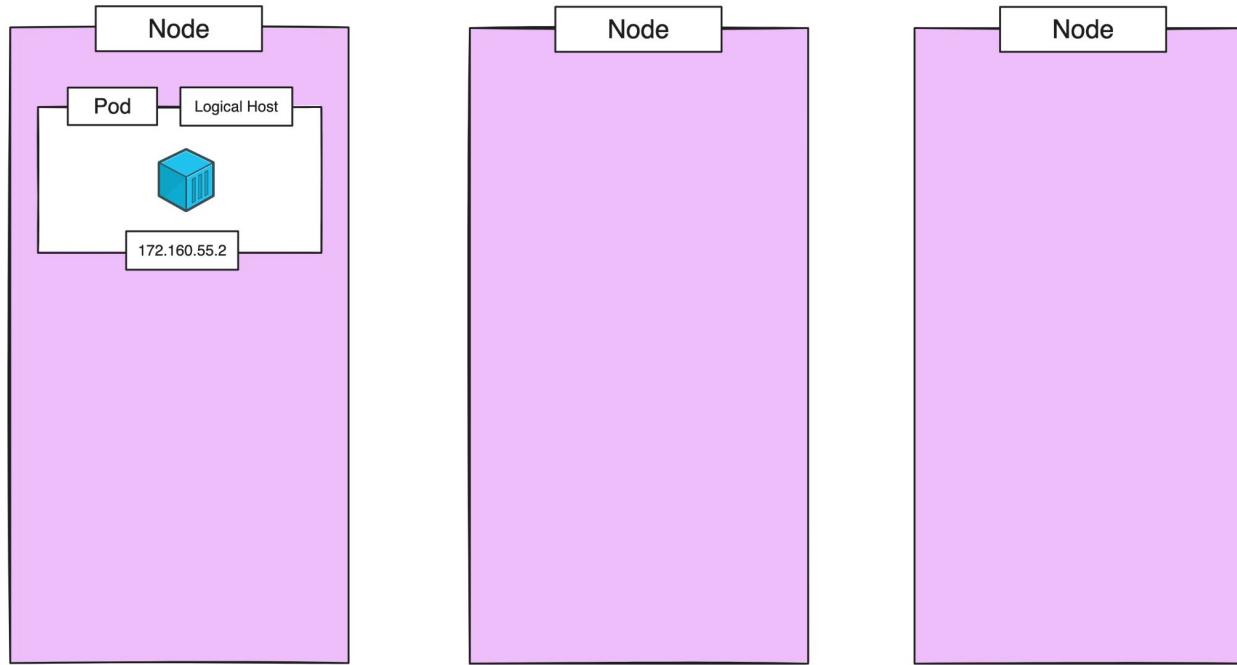
Pod

Pod

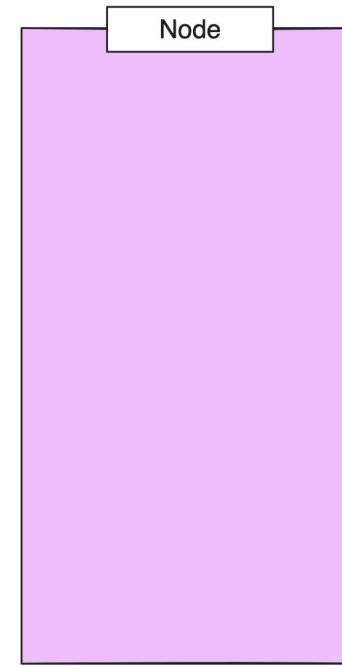
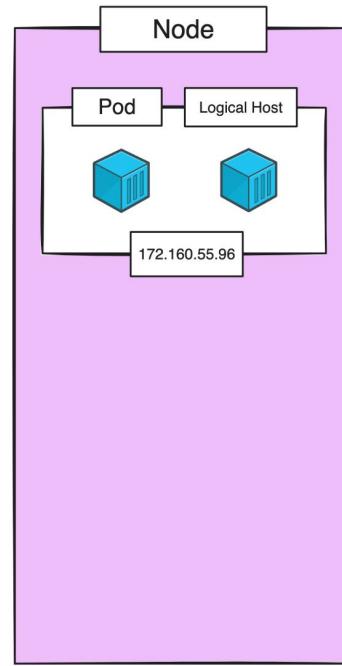
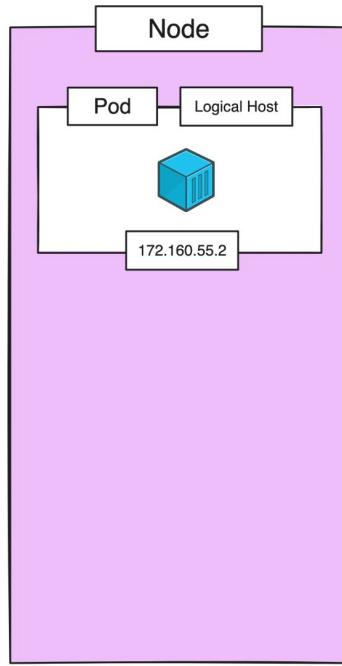
- Smallest deployable units of computing that you can create and manage in Kubernetes
- Logical Application
- One or more container in Pod (recommend 1 container per pod)
- Any containers in same pod will share storage volumes and network resources
- Each pod is assigned a unique ip



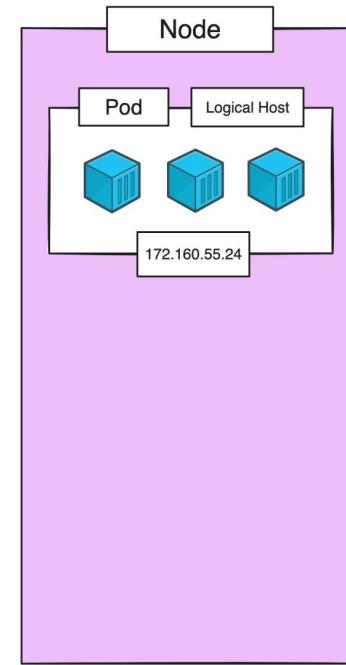
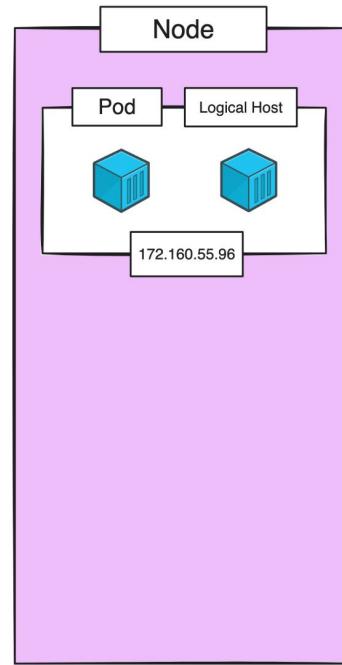
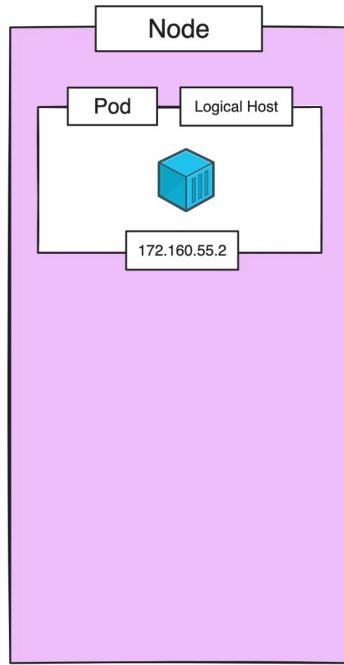
Node & Pod



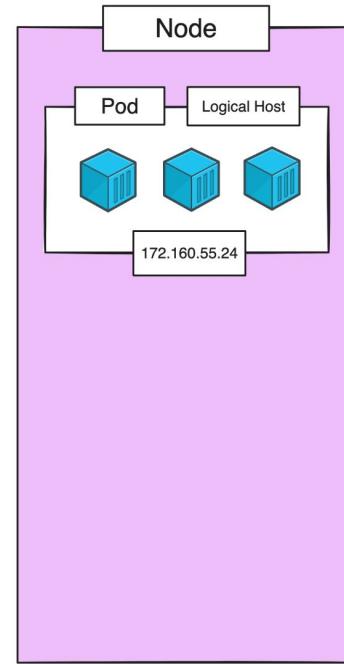
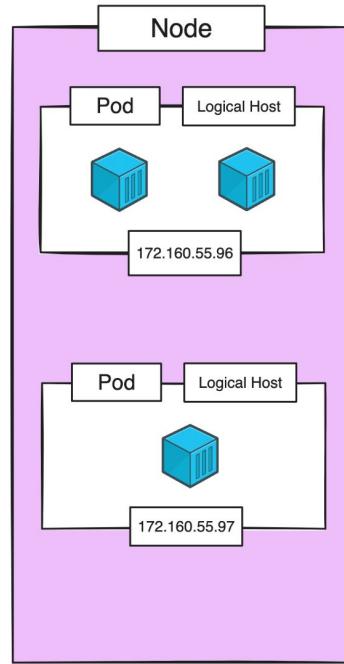
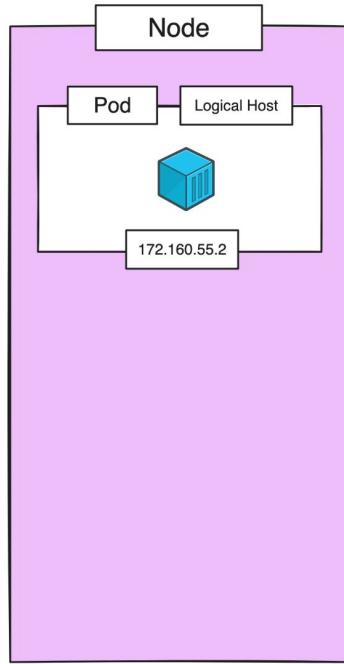
Node & Pod



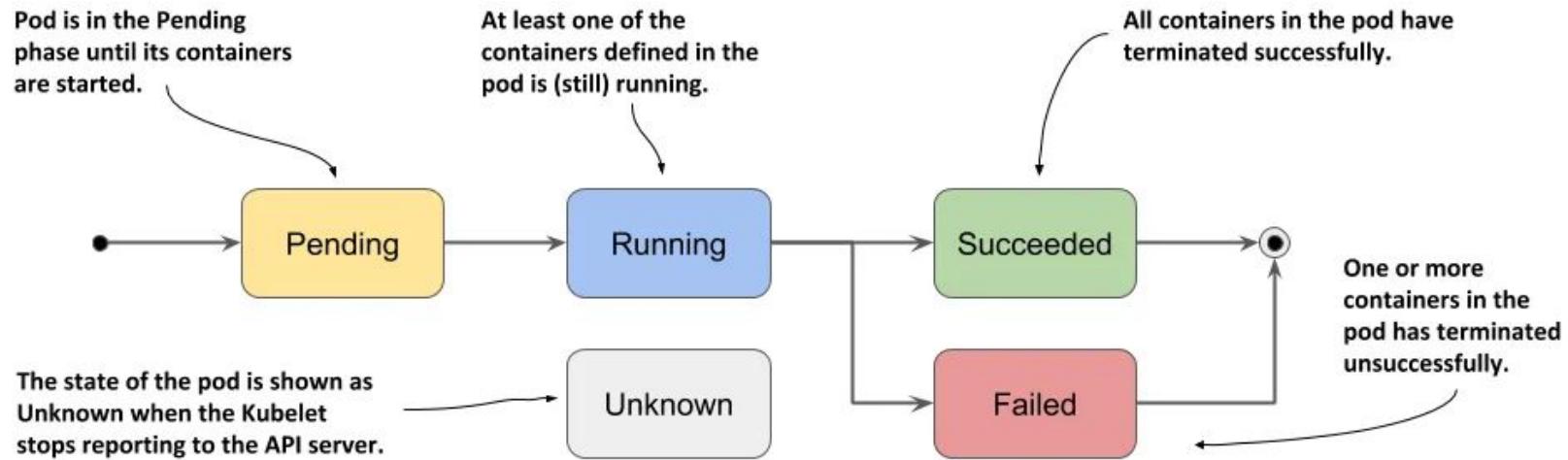
Node & Pod



Node & Pod

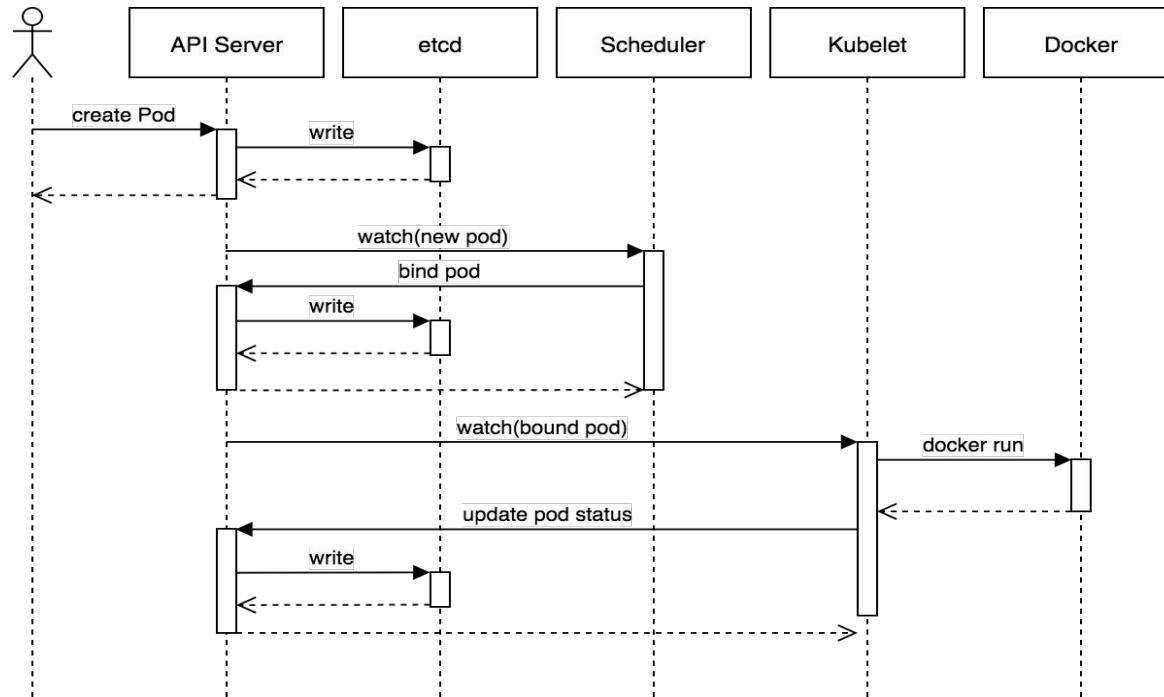


States of a Pod



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

Creation of a Pod



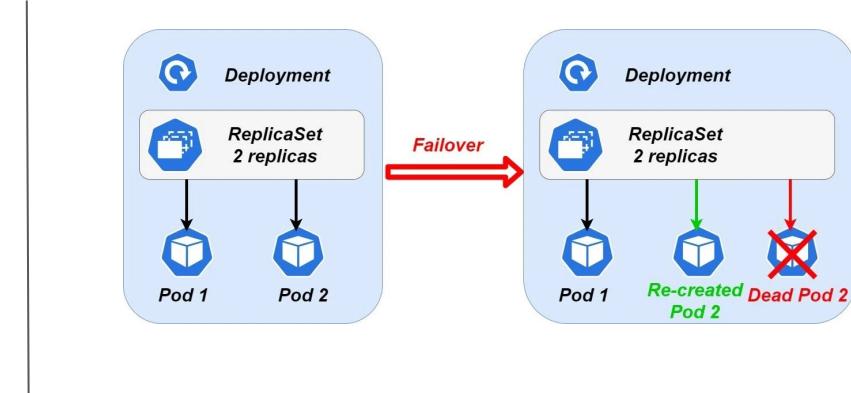
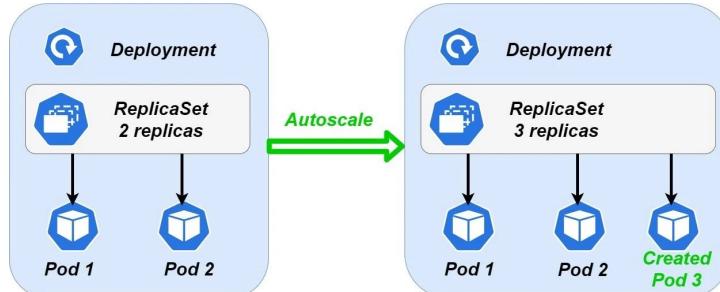
<https://medium.com/@seifeddinerajhi/navigating-the-journey-of-a-pod-a-guide-to-the-explaining-world-of-pod-lifecycle-a1fbc2c98c55>

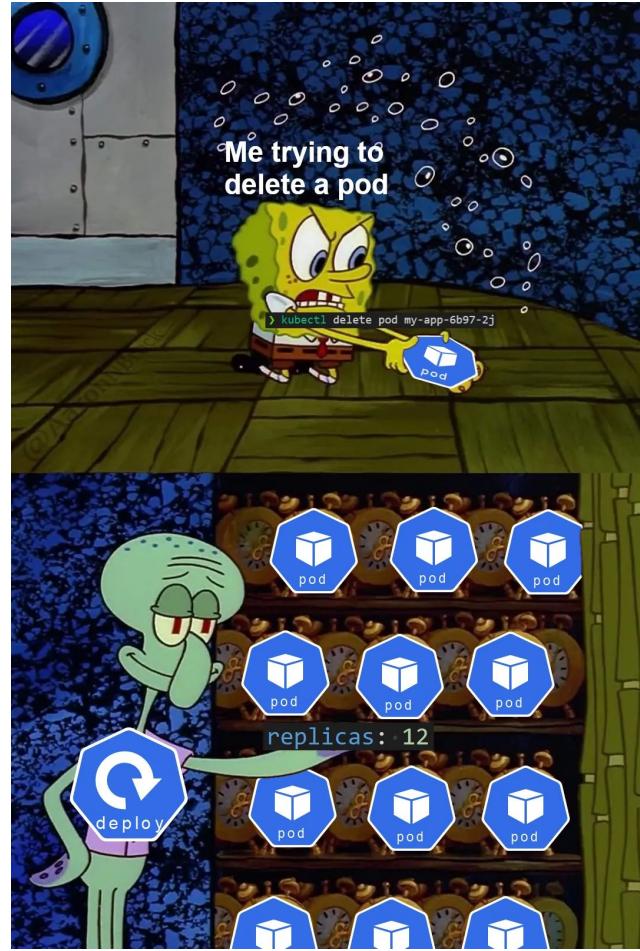
Workshop Pod

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

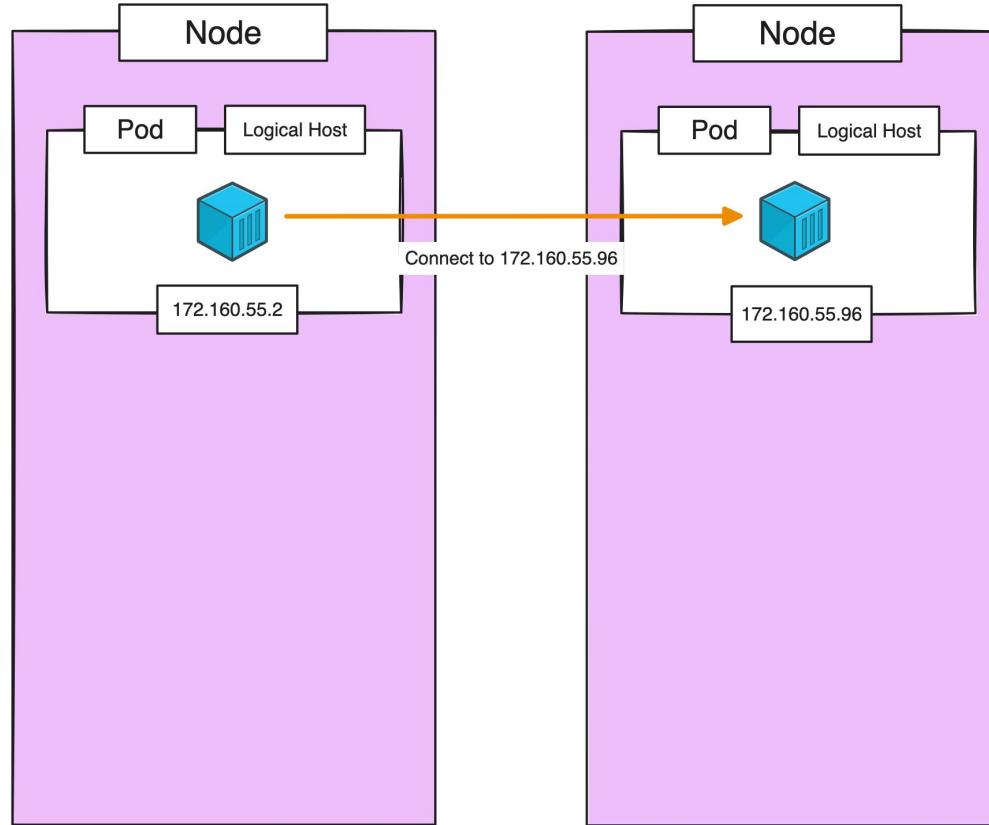




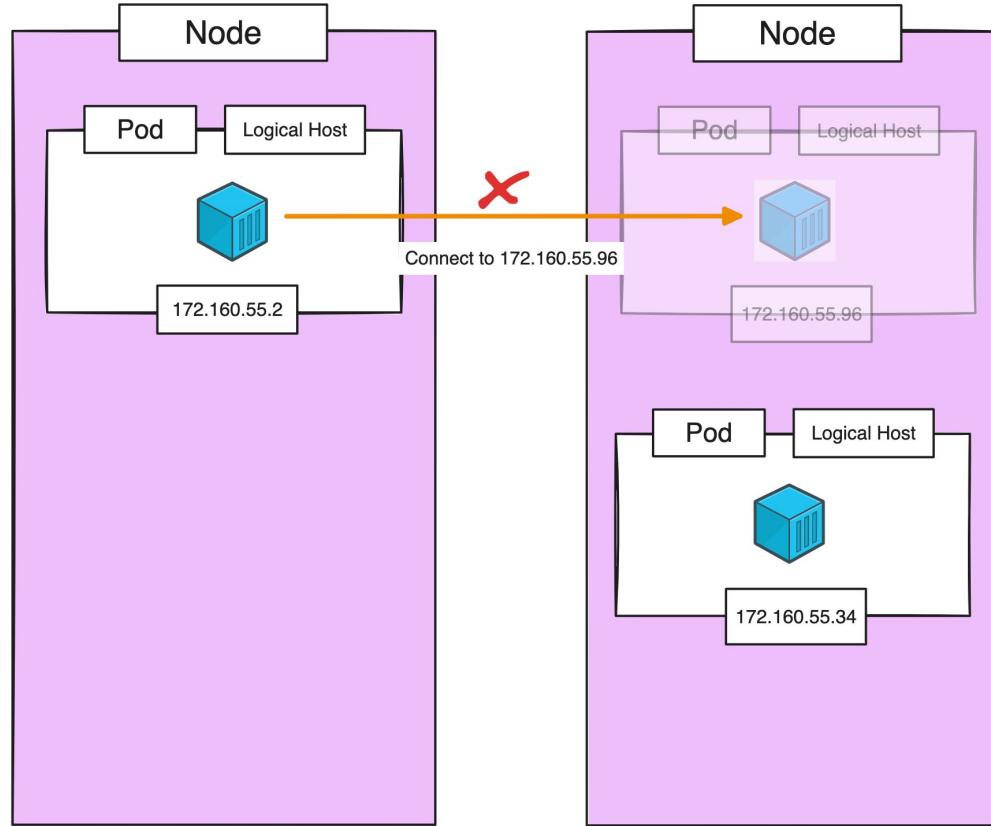
Workshop

Deployment

Problem



Problem

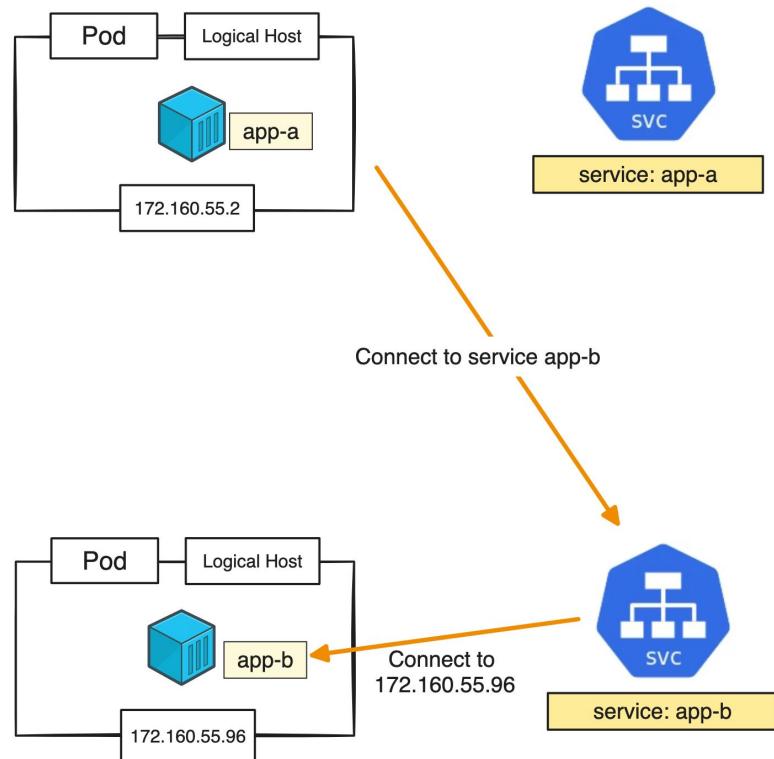


Service

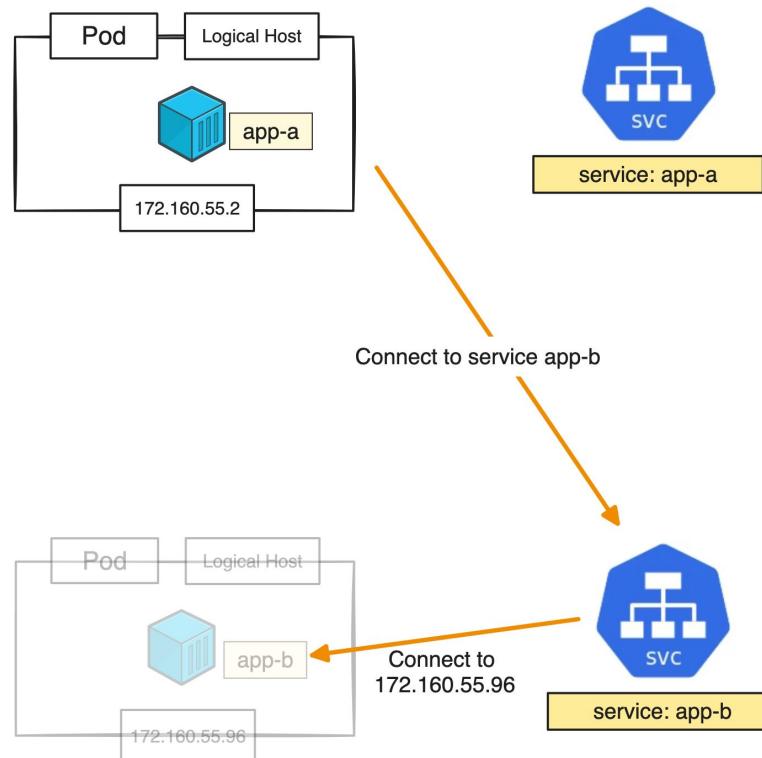
Service

- 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

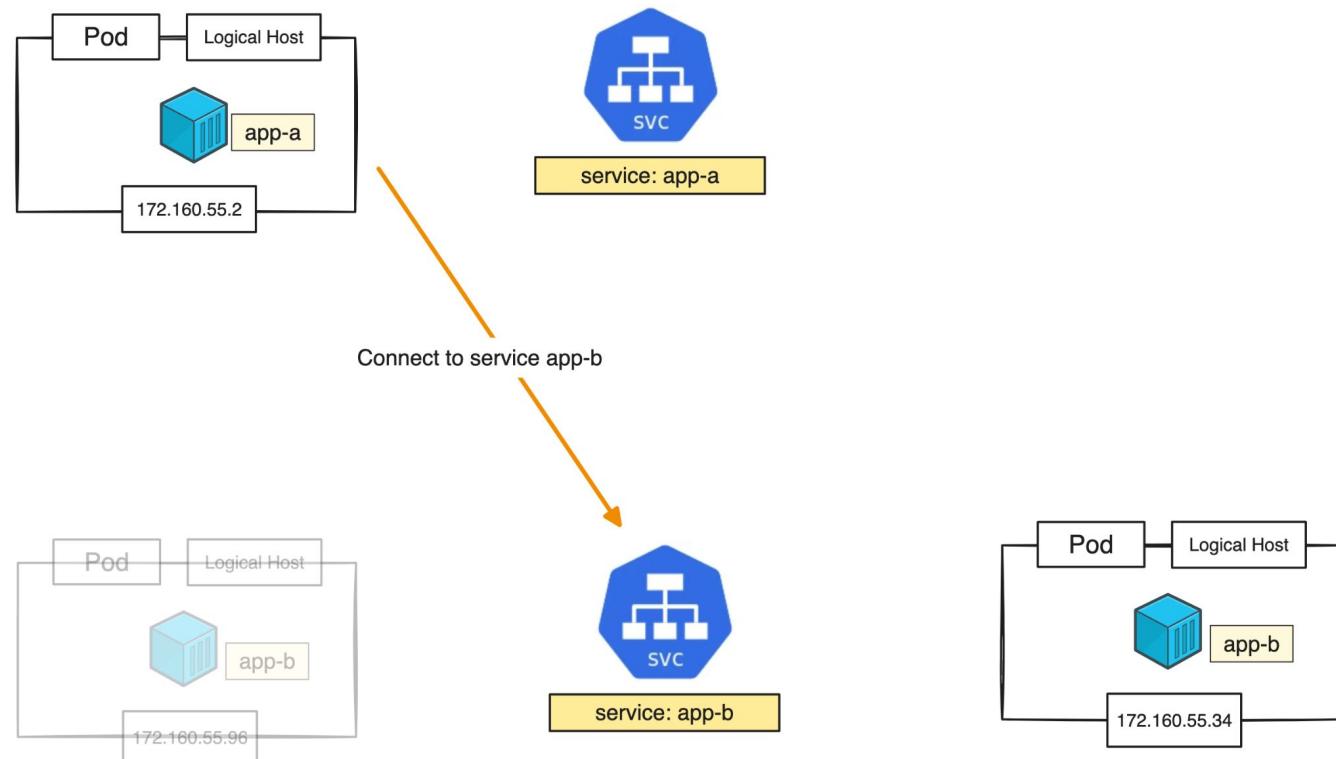
Service: Stable endpoint



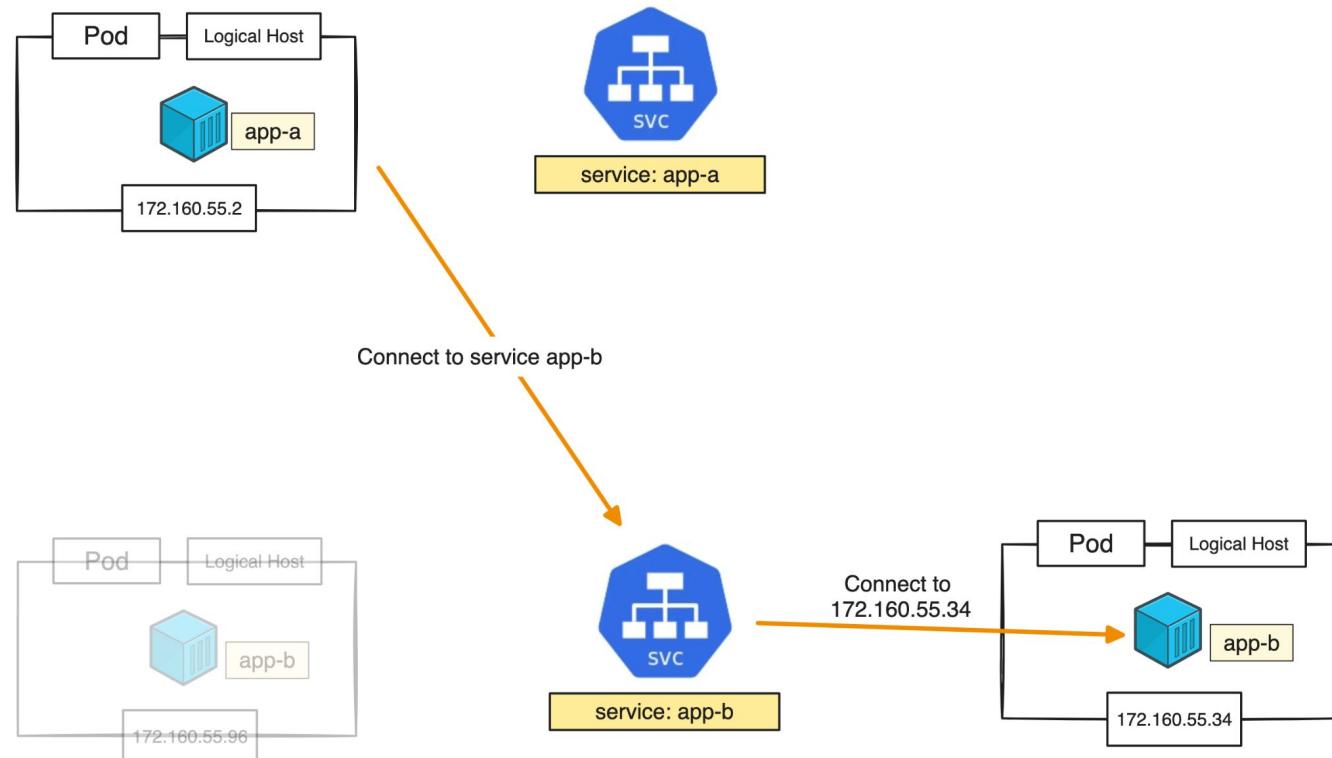
Service: Stable endpoint



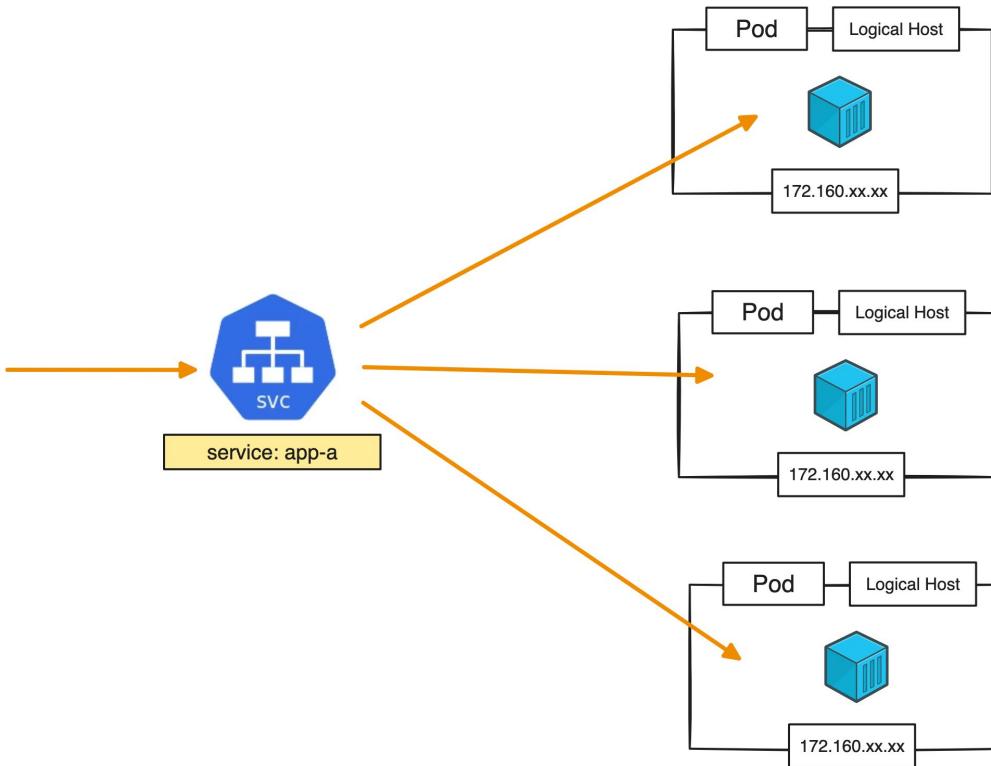
Service: Stable endpoint



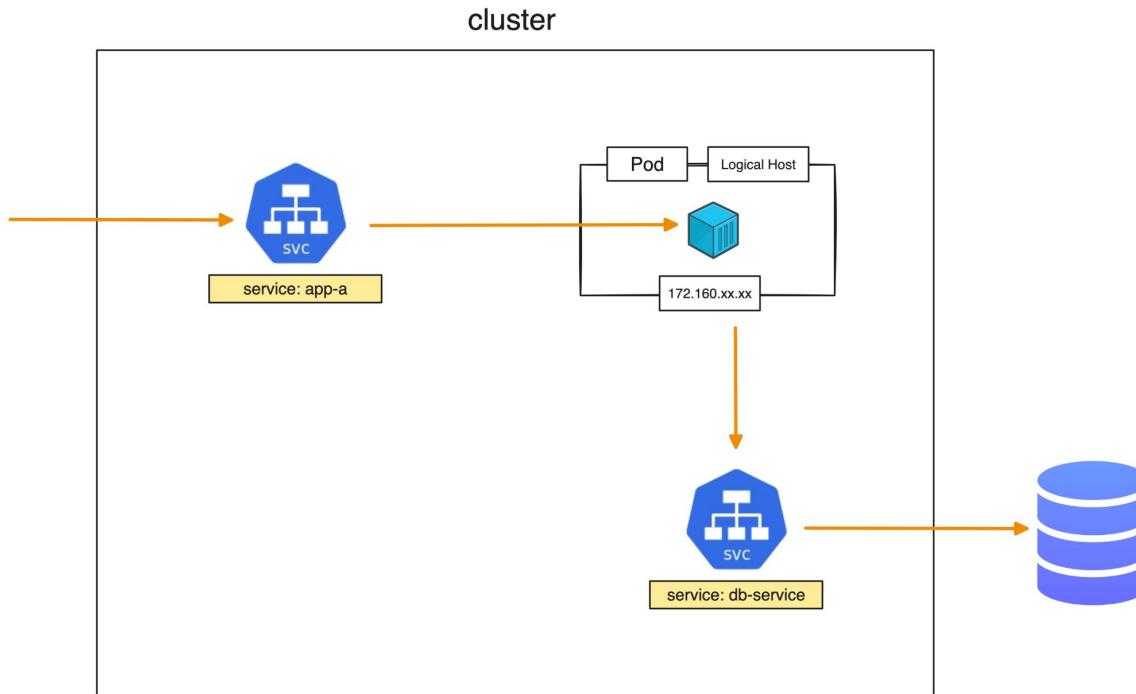
Service: Stable endpoint



Service: Load balancer



Service: Communicating inside & outside of cluster



Service type: ClusterIP

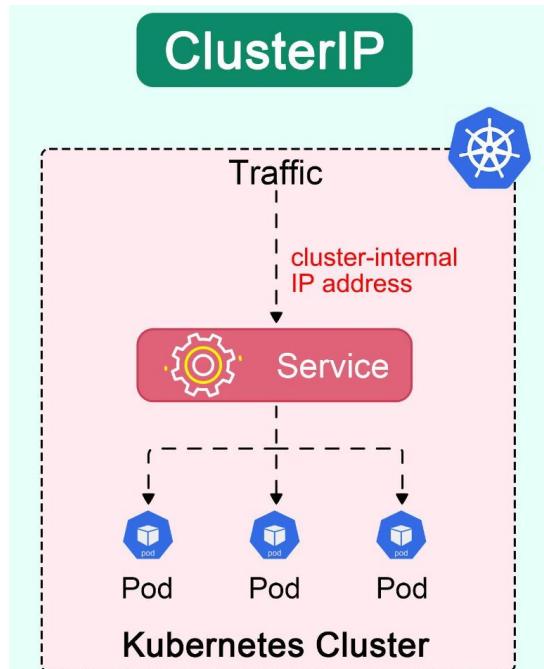
- ClusterIP is the default service type in Kubernetes
- It exposes the service on an internal IP address reachable **only within the cluster**

Advantages

- Suitable for inter-pod communication and internal services

Disadvantages

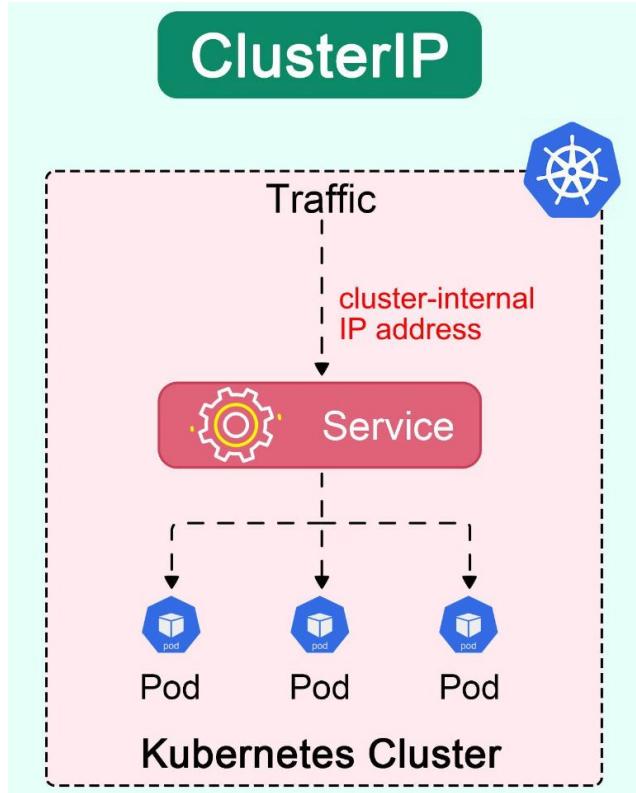
- Limited to internal cluster communication
- Not accessible externally



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



Workshop

ClusterIP Service

Service type: NodePort

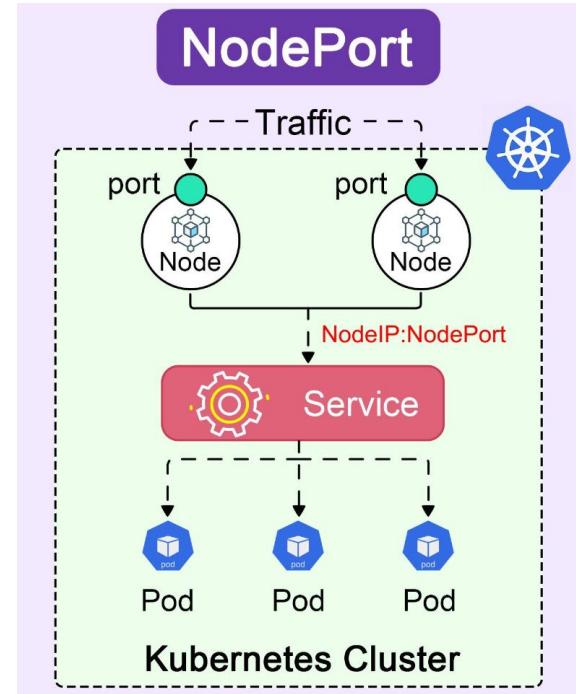
- It enables **external access** to the service using the cluster **nodes' IP address** and the assigned **NodePort**
- When a NodePort service is created, Kubernetes allocates a port from a predefined range (default: **30000-32767**)

Advantages

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

Disadvantages

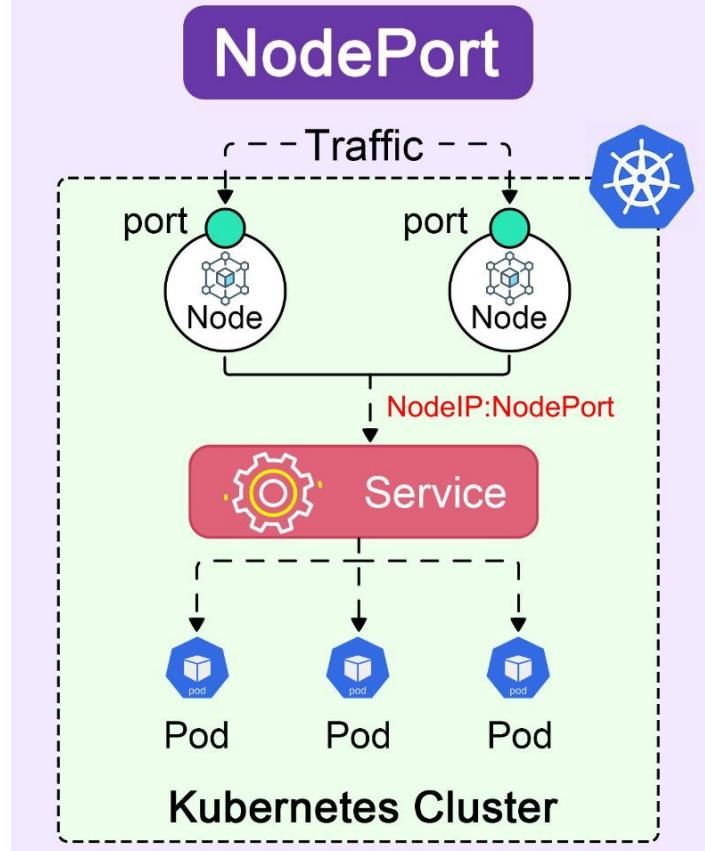
- Requires manual management of port allocations
- IP Node maybe have changed
- Should not use in production



Service type: NodePort

Use Cases

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



Workshop

NodePort Service

Service type: LoadBalancer

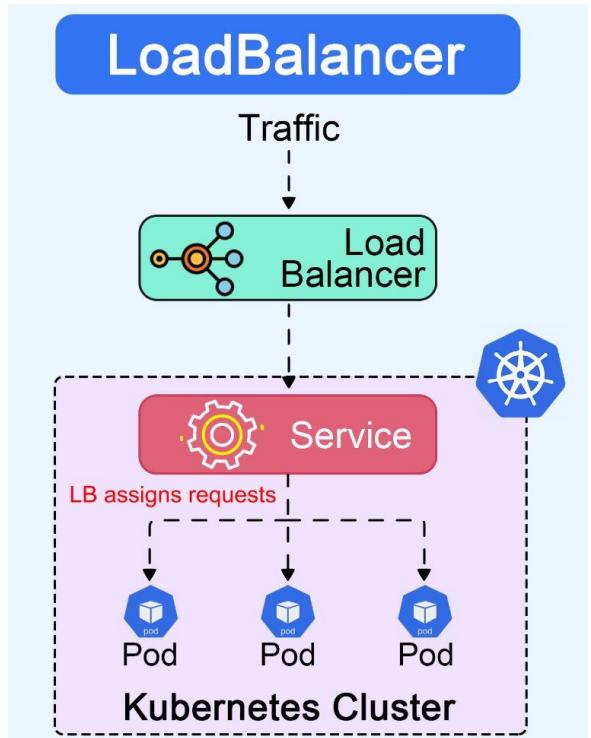
- LoadBalancer is a service type that **provisions an external load balancer** to distribute traffic across the pods
- Typically provided by a cloud provider's infrastructure
- When a LoadBalancer service is created, Kubernetes interacts with the cloud provider's API to an external load balancer

Advantages

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

Disadvantages

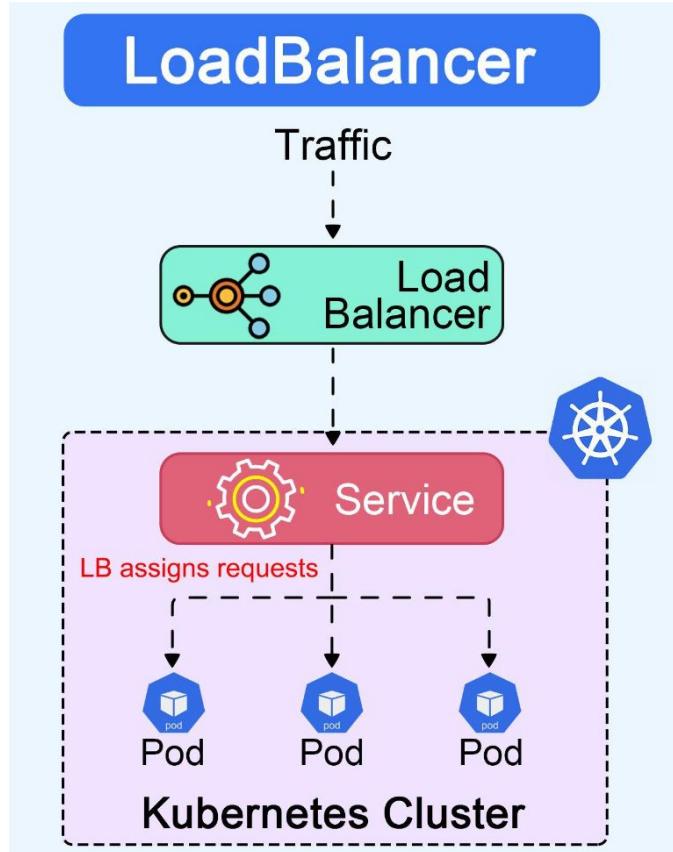
- Requires a cloud provider
- Additional costs for using the load balancer service on cloud provider



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



Workshop

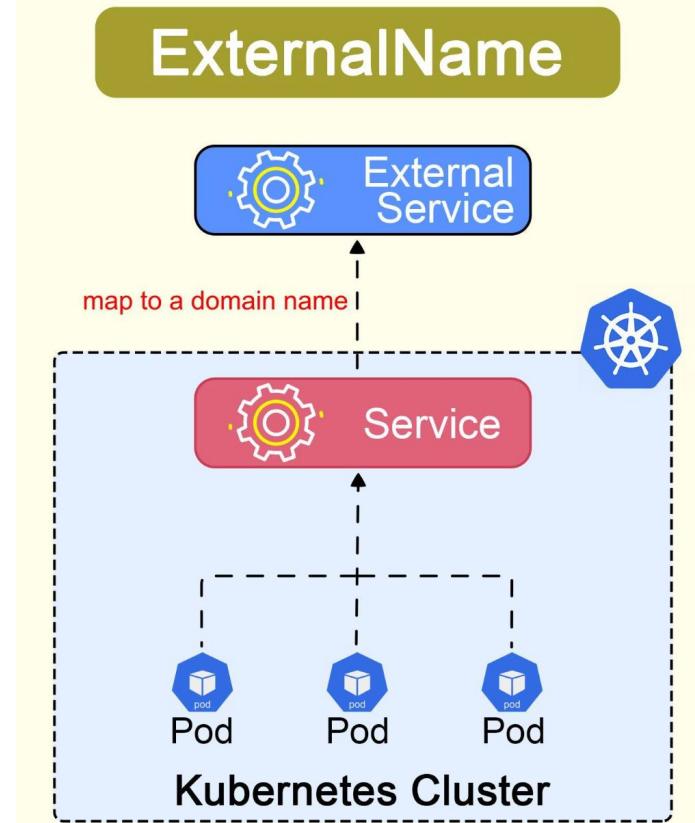
LoadBalancer Service

Service type: ExternalName

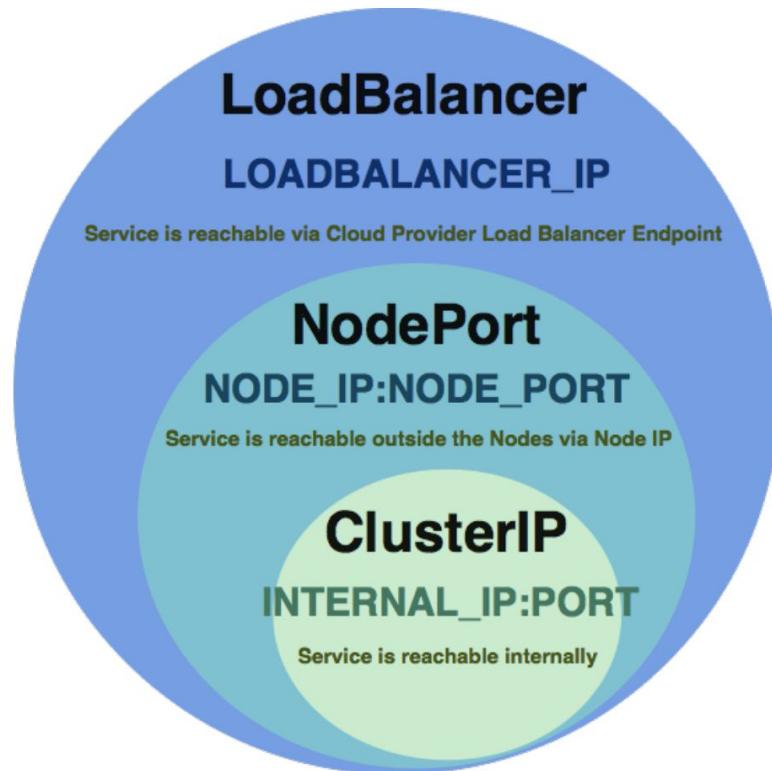
- An ExternalName Service is a special type of Kubernetes service that allows you to expose an existing external service (like a database or a third-party API) to your cluster without deploying any Pods

Use Cases

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



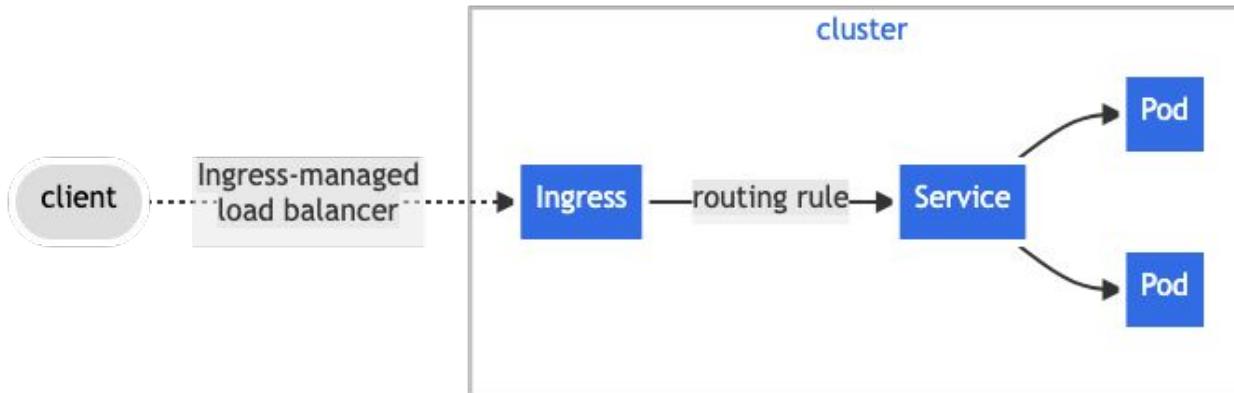
Service type



Ingress

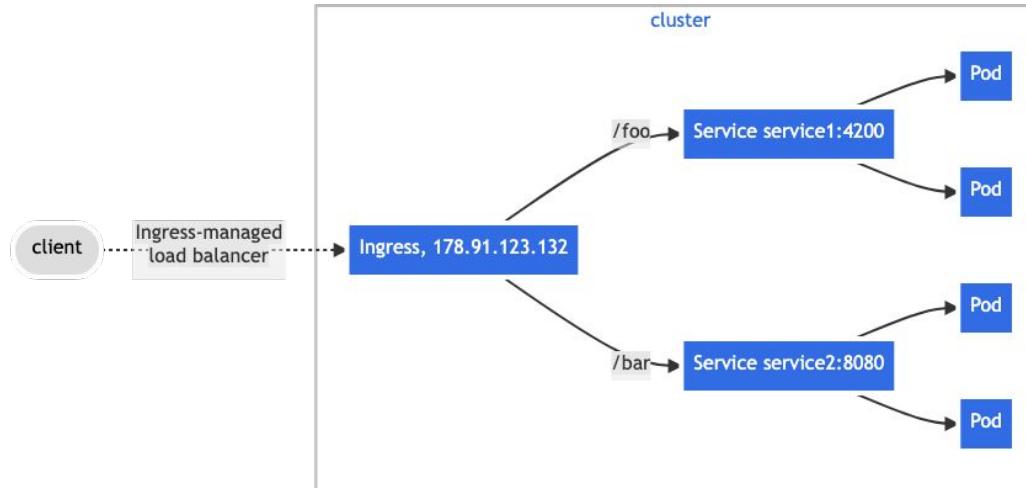
Ingress

- 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

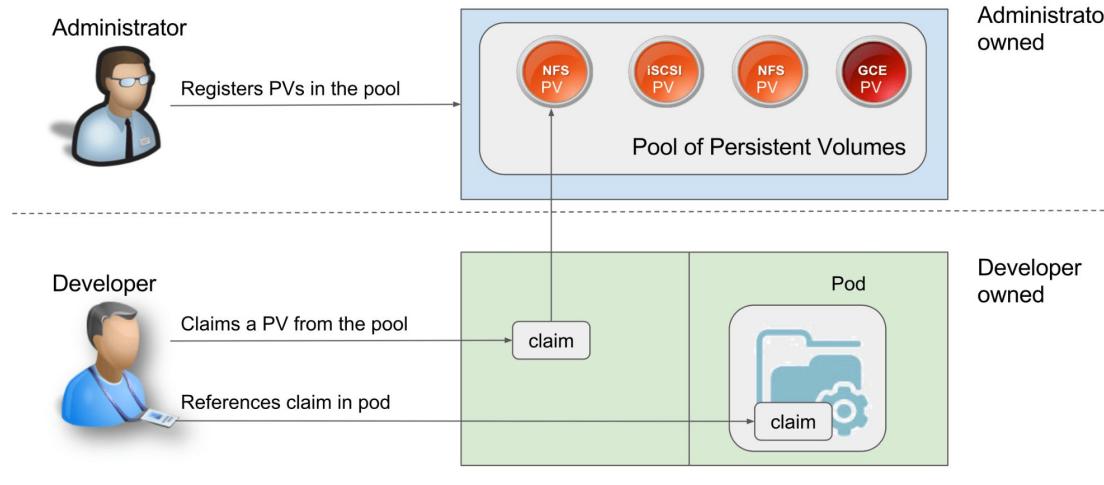
Workshop

ConfigMap & Secret

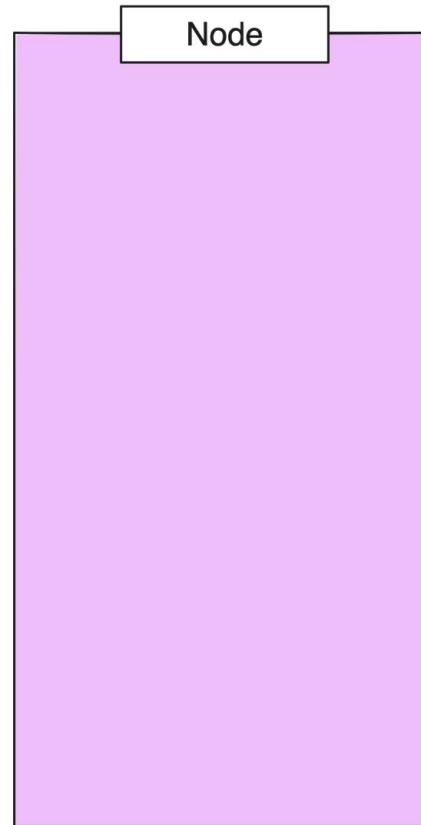
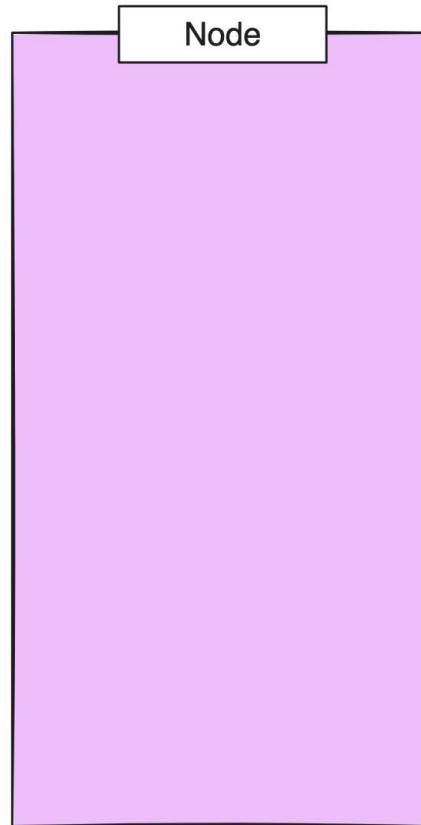
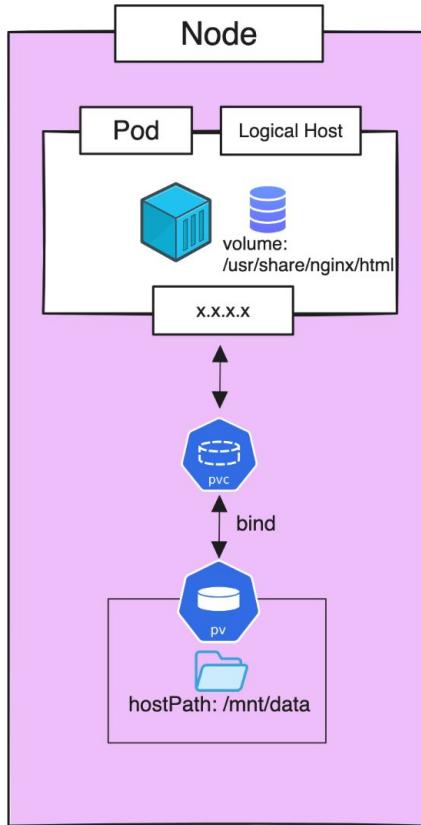
Persistent Volume

Persistent Volume

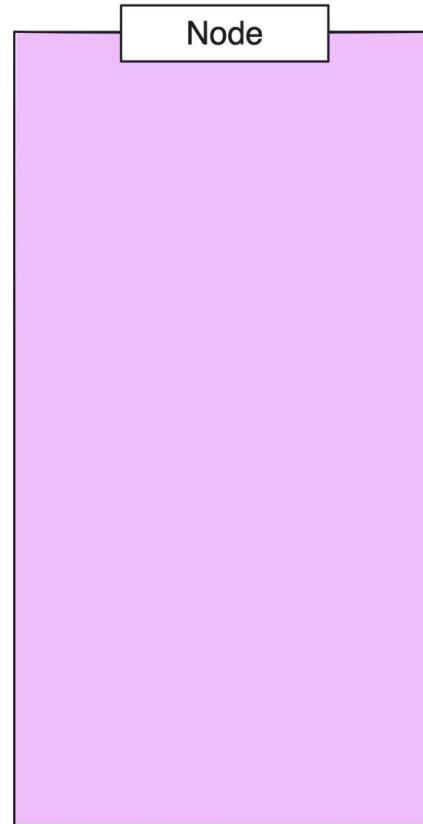
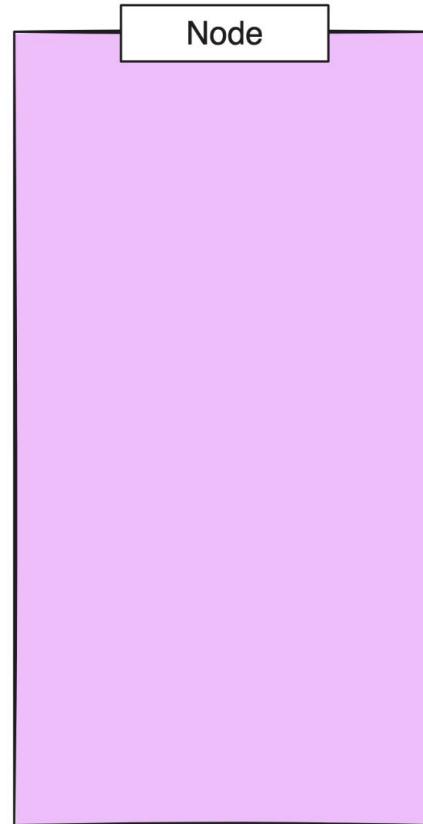
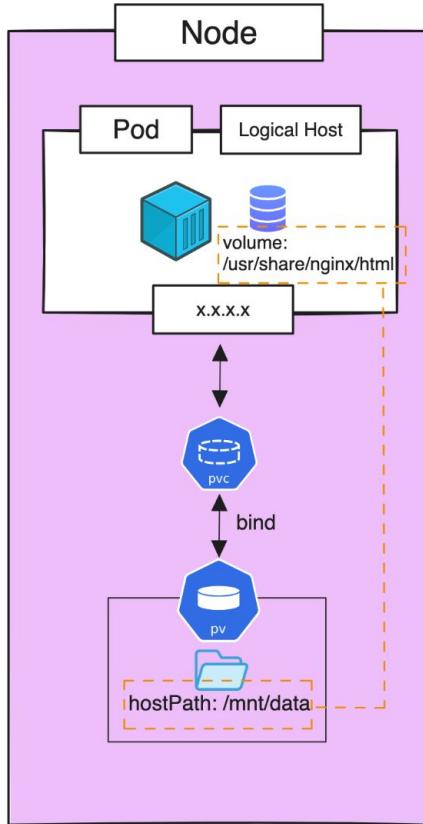
- Persistent Volume (PV) is a piece of storage in the cluster
- Can be provisioned from various sources like host local storage, cloud providers storage, or NFS.
- Persistent Volume Claims (PVC) allow a user to consume abstract storage resources
- A PVC to PV binding is a one-to-one mapping



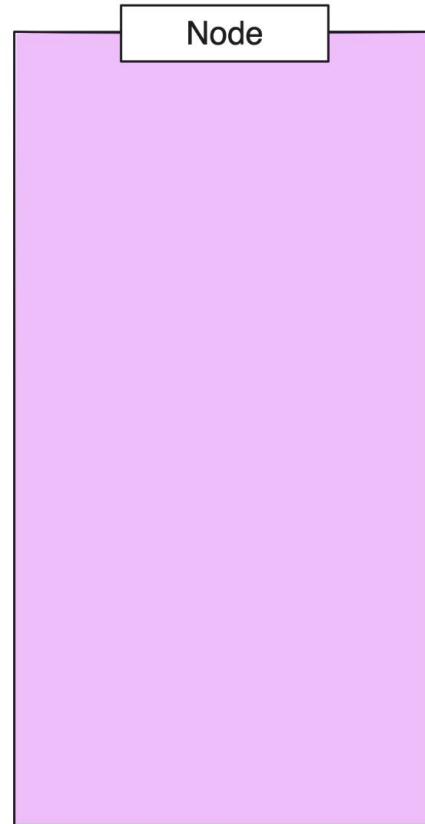
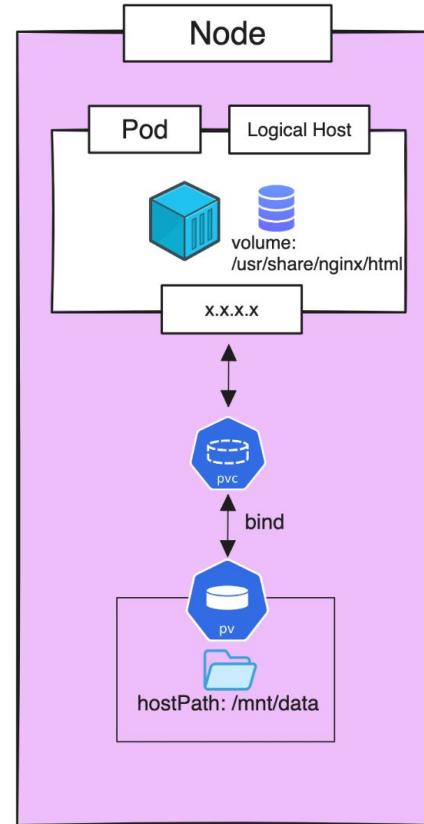
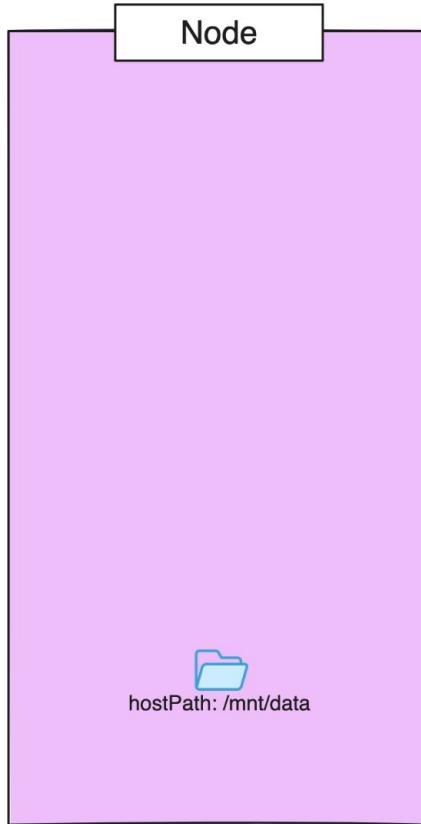
PV & PVC



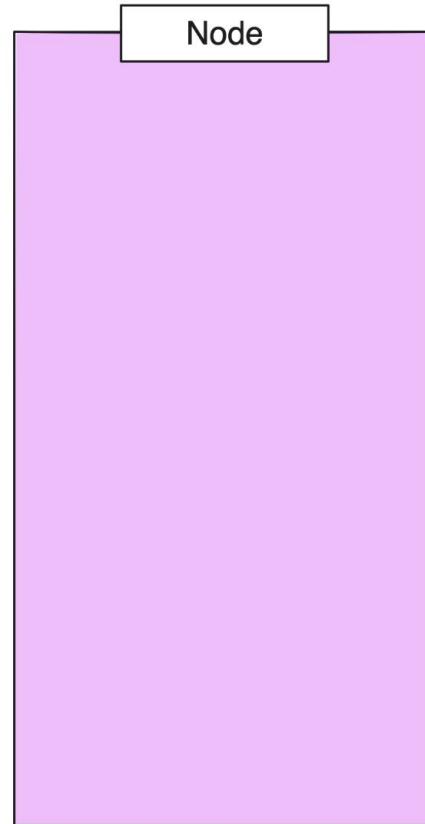
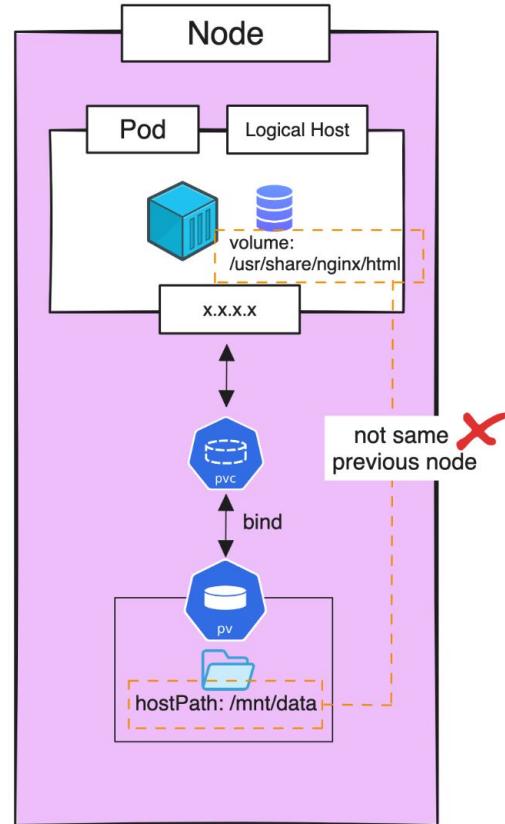
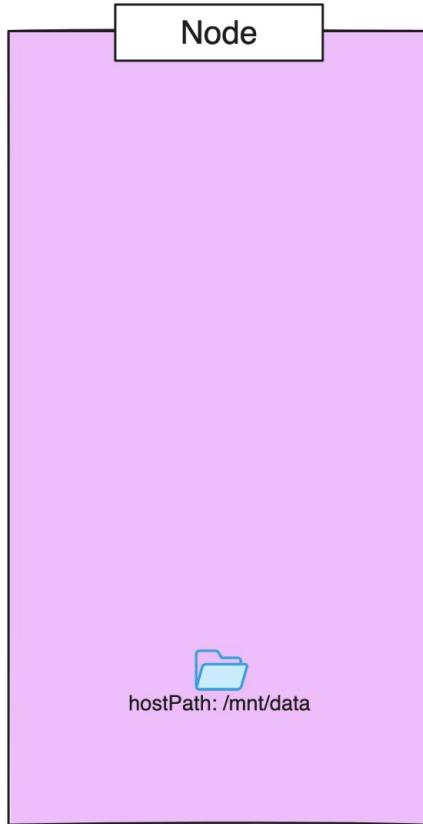
PV & PVC



PV & PVC



PV & PVC



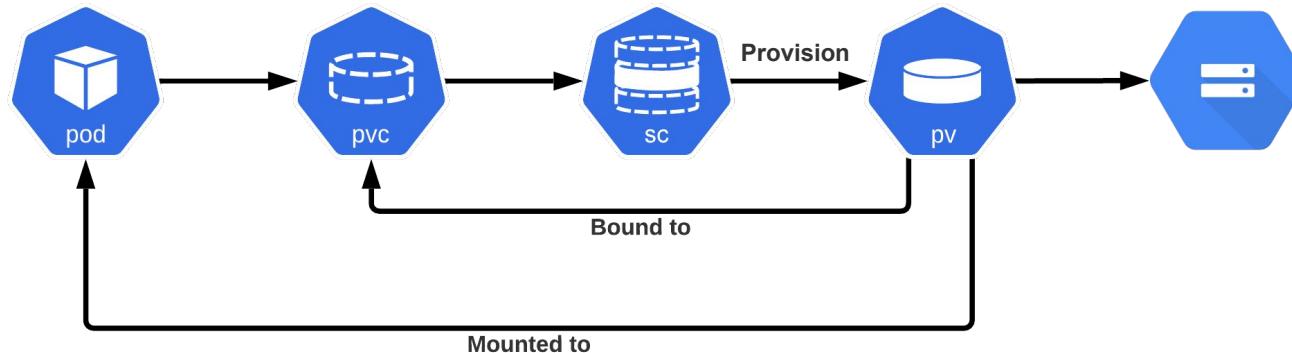
PV & PVC

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

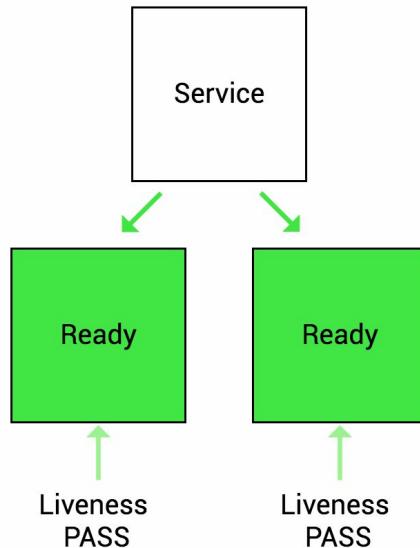
Day 3

Good to know I



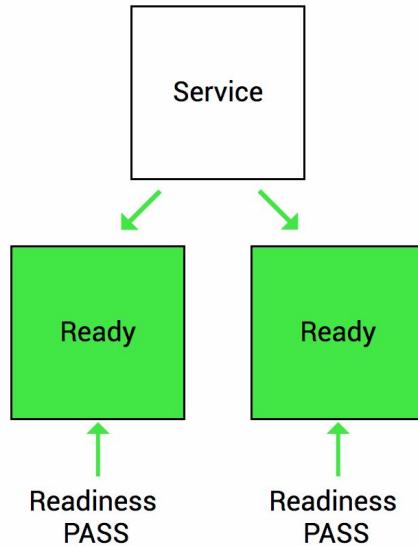
Pod liveness & readiness

- **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

The motivation for using namespaces

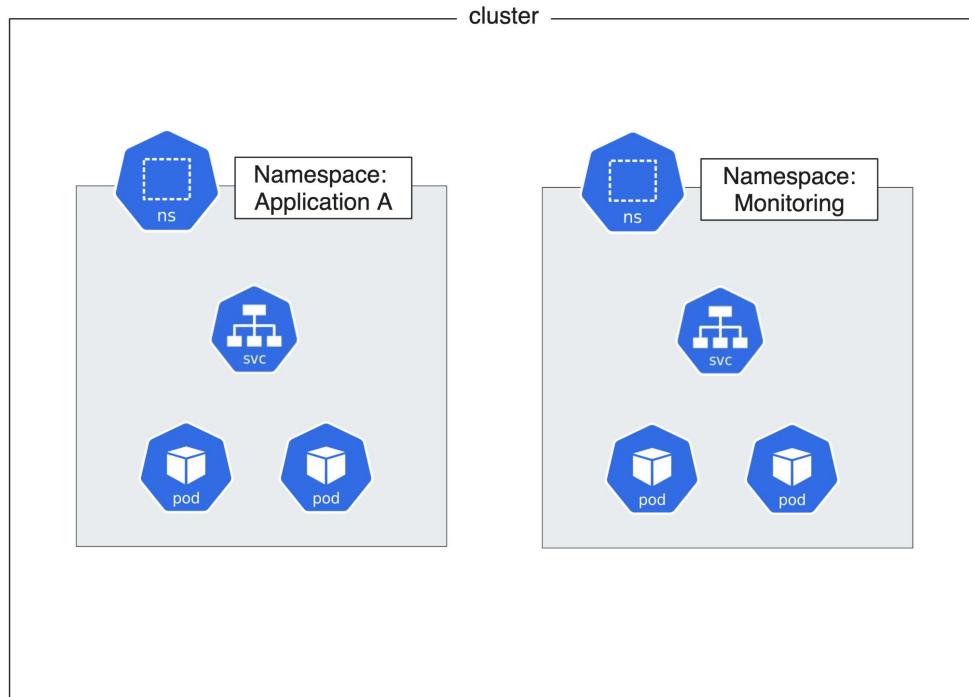
Each user community wants to be able to work in isolation from other communities.

Each user community has its own:

1. Resources (pods, services, replication controllers, etc.)
2. Policies (who can or cannot perform actions in their community)
3. Constraints (this community is allowed this much quota, etc.)

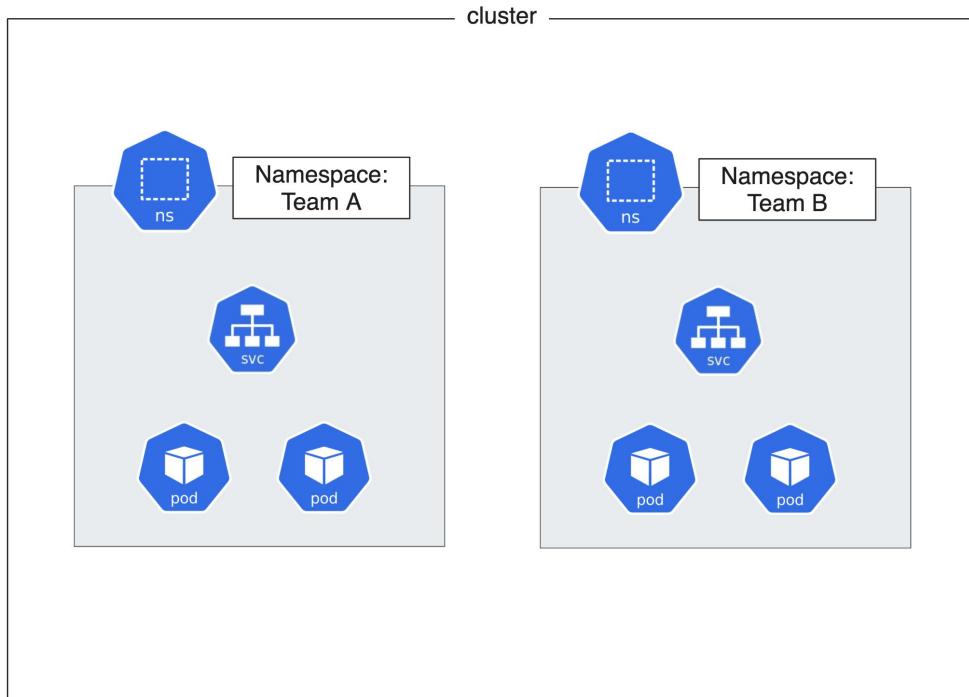
Namespace - use cases

- Application domain



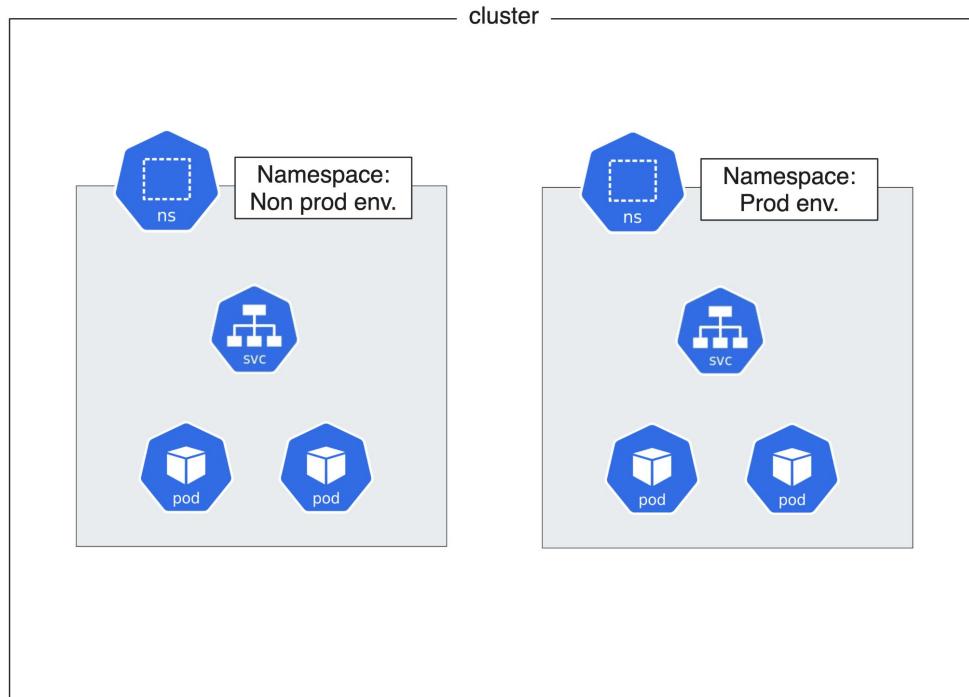
Namespace - use cases

- Teams driven



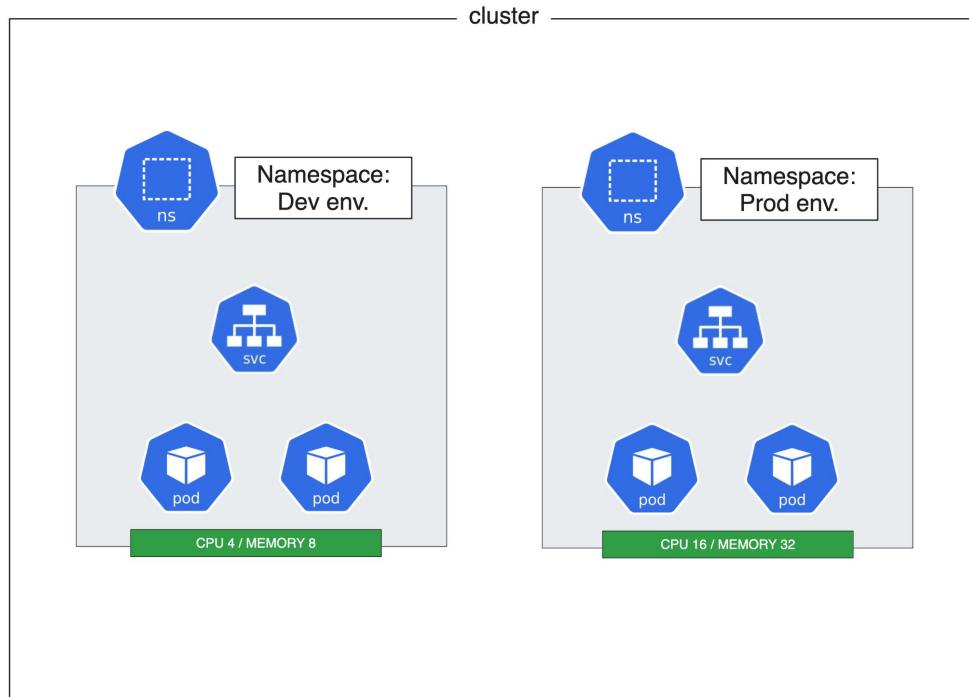
Namespace - use cases

- Environments



Namespace - use cases

- Resources quota limit



Workshop

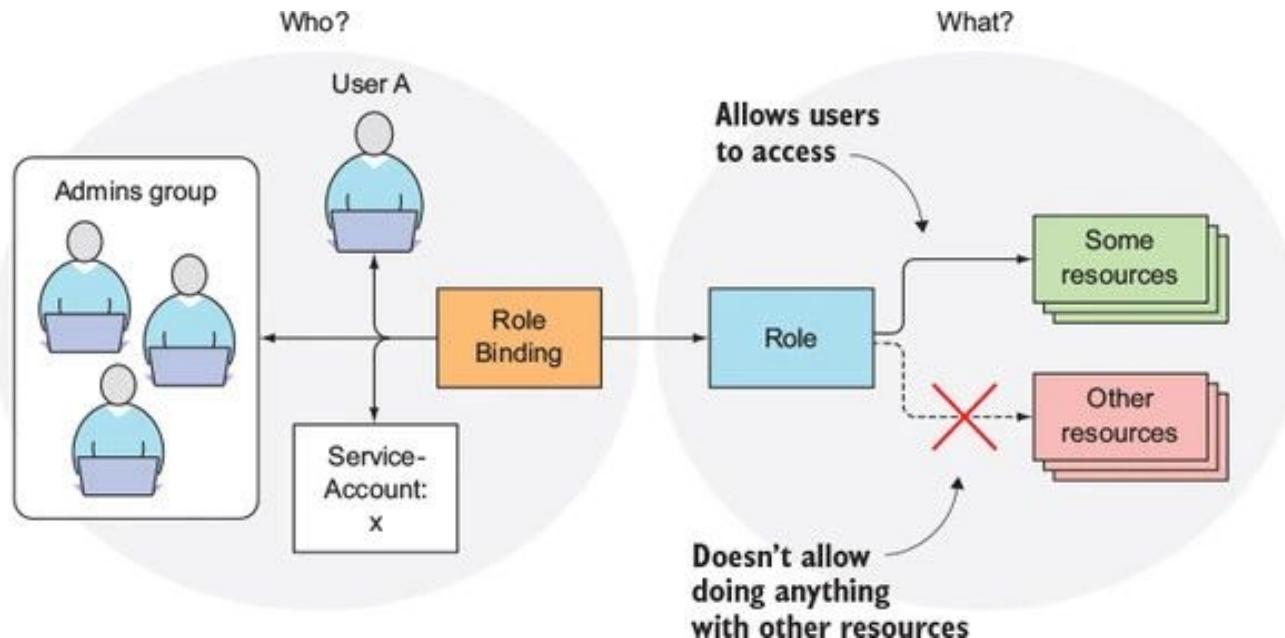
Namespace

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>

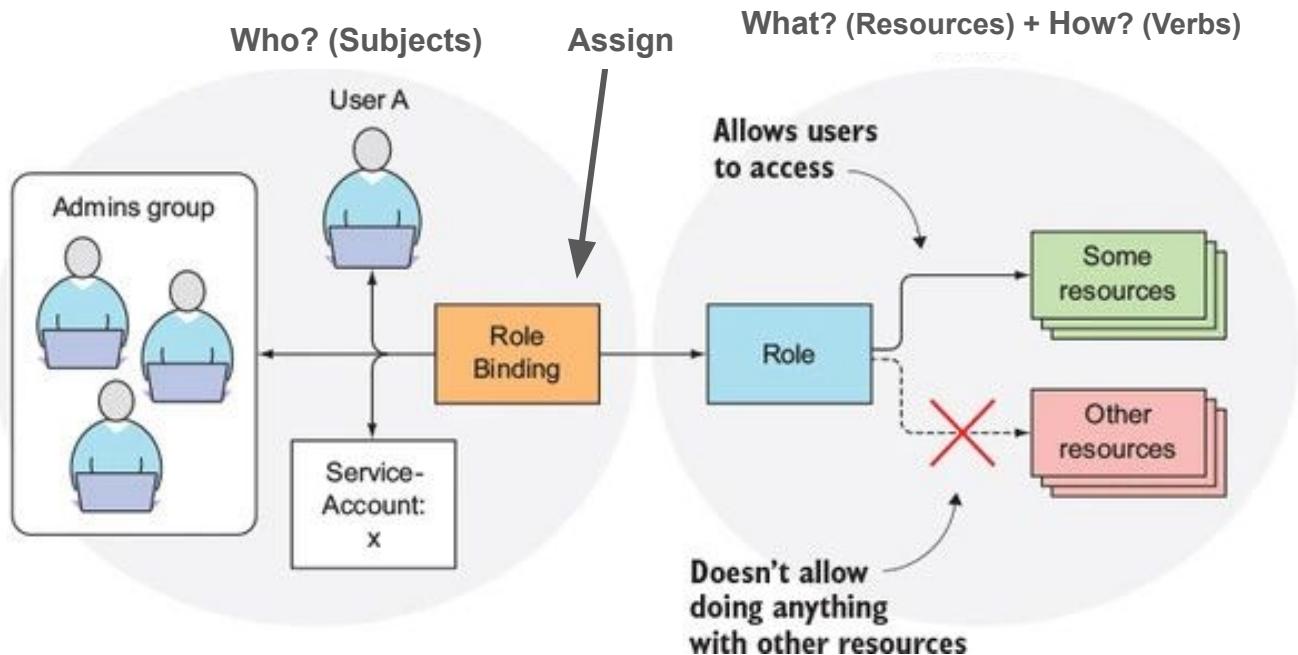
RBAC

What ? (Kubernetes Resource)

- pod
- deployment
- service
- etc.

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

RBAC



RBAC

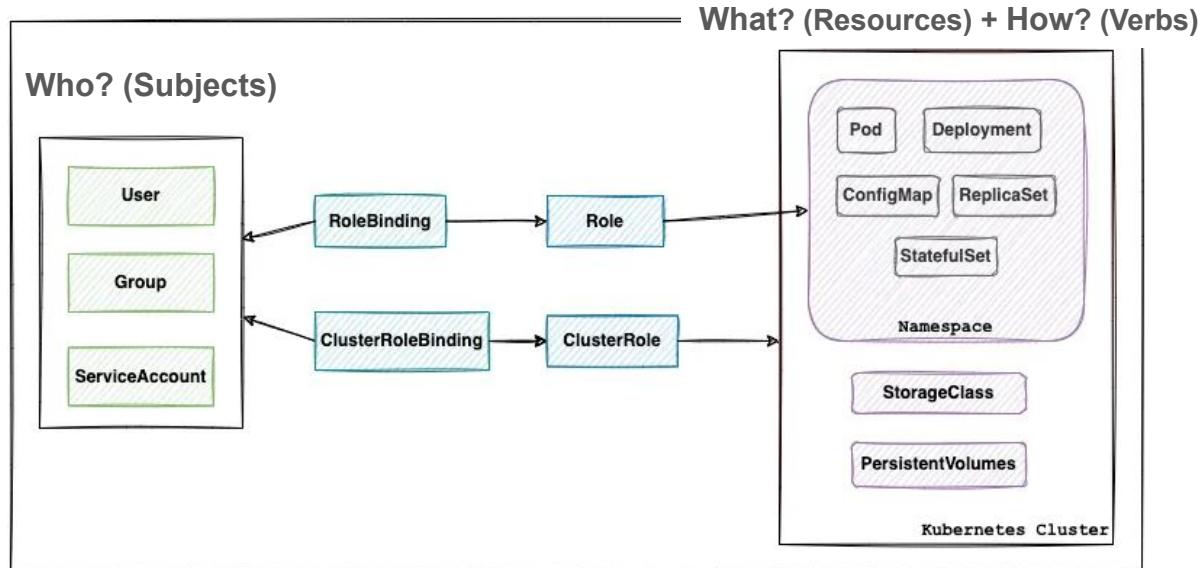
Permission Scopes

- 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

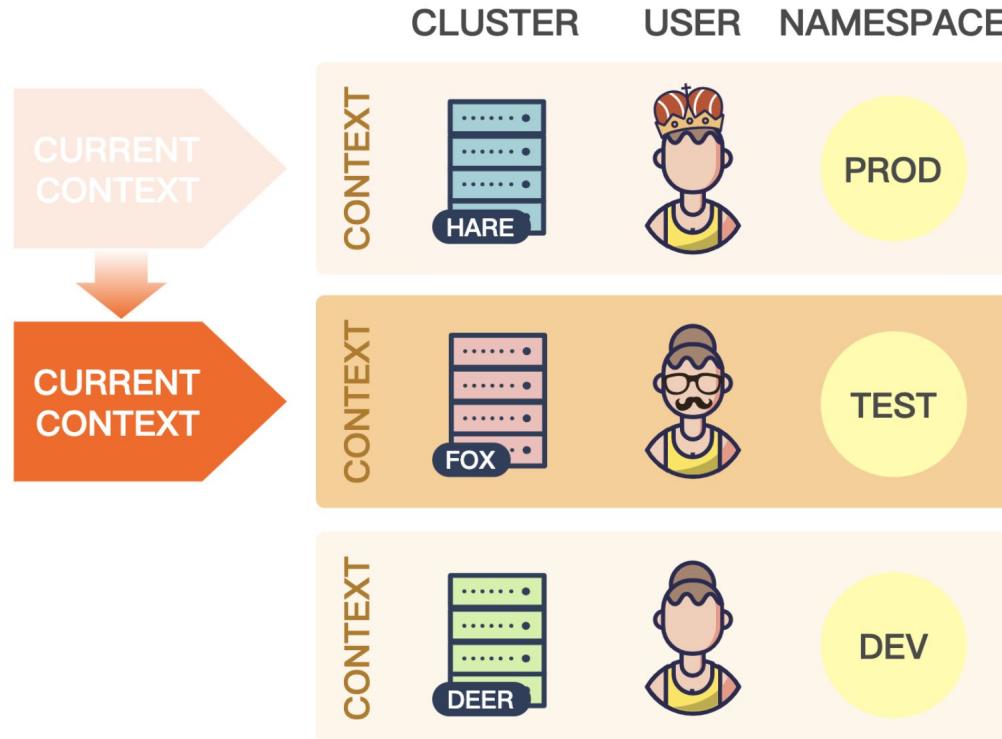
Binding

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



Context

- Access to Cluster(s)



Workshop

RBAC

Good to know II



Labels

- Labels are key/value pairs that are attached to objects such as Pods. Labels are intended to be used to specify identifying attributes of objects that are meaningful and relevant to users.

Example labels

- release: stable, canary
- environment: dev, qa, production
- tier: frontend, backend
- partition: customerA, customerB
- track: daily, weekly

Recommend Labels

In order to take full advantage of using these labels, they should be applied on every resource object.

| Key | Description | Example | Type |
|---|---|--------------------------------|--------|
| <code>app.kubernetes.io/name</code> | The name of the application | <code>mysql</code> | string |
| <code>app.kubernetes.io/instance</code> | A unique name identifying the instance of an application | <code>mysql-
abcxyz</code> | string |
| <code>app.kubernetes.io/version</code> | The current version of the application (e.g., a SemVer 1.0 , revision hash, etc.) | <code>5.7.21</code> | string |
| <code>app.kubernetes.io/component</code> | The component within the architecture | <code>database</code> | string |
| <code>app.kubernetes.io/part-of</code> | The name of a higher level application this one is part of | <code>wordpress</code> | string |
| <code>app.kubernetes.io/managed-by</code> | The tool being used to manage the operation of an application | <code>Helm</code> | string |

Labels with command

Get resources with show labels

```
$ kubectl get pod --show-labels
```

Get resources with filter labels

```
$ kubectl get pod -l "app=xxx"  
$ kubectl get pod --selector "app=xxx"  
$ kubectl get pod -l "app!=xxx"  
$ kubectl get pod --selector "app!=xxx"
```

HELM



HELM

HELM is the package manager for Kubernetes



HELM

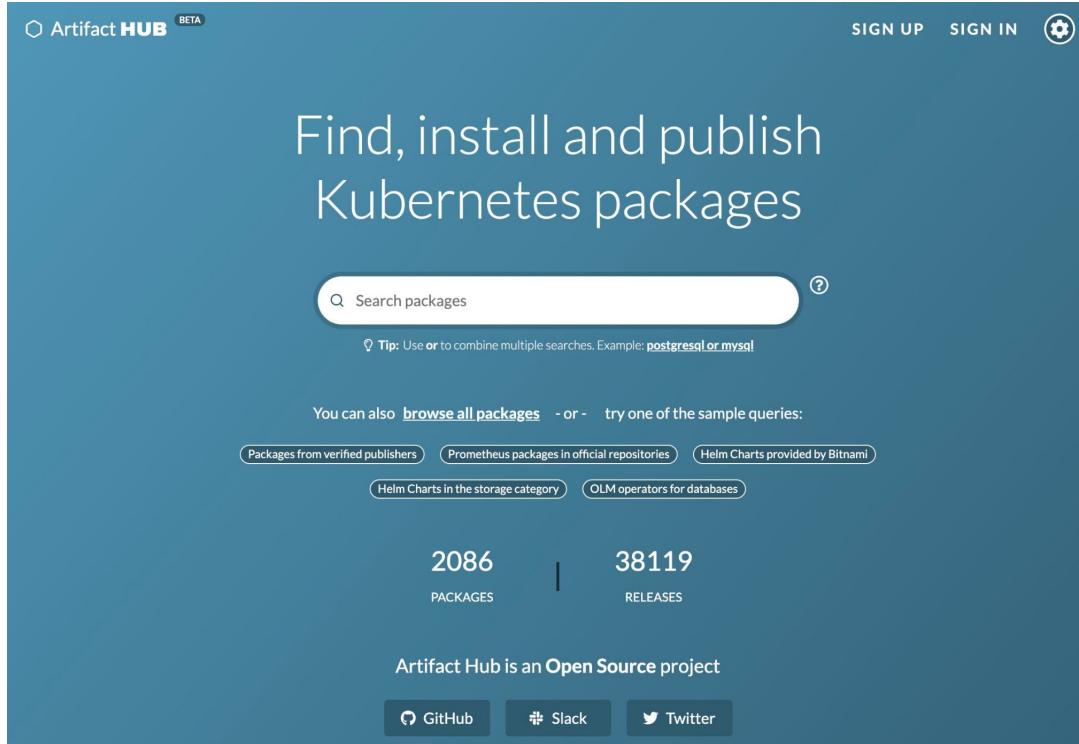
HELM is the package manager for Kubernetes

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

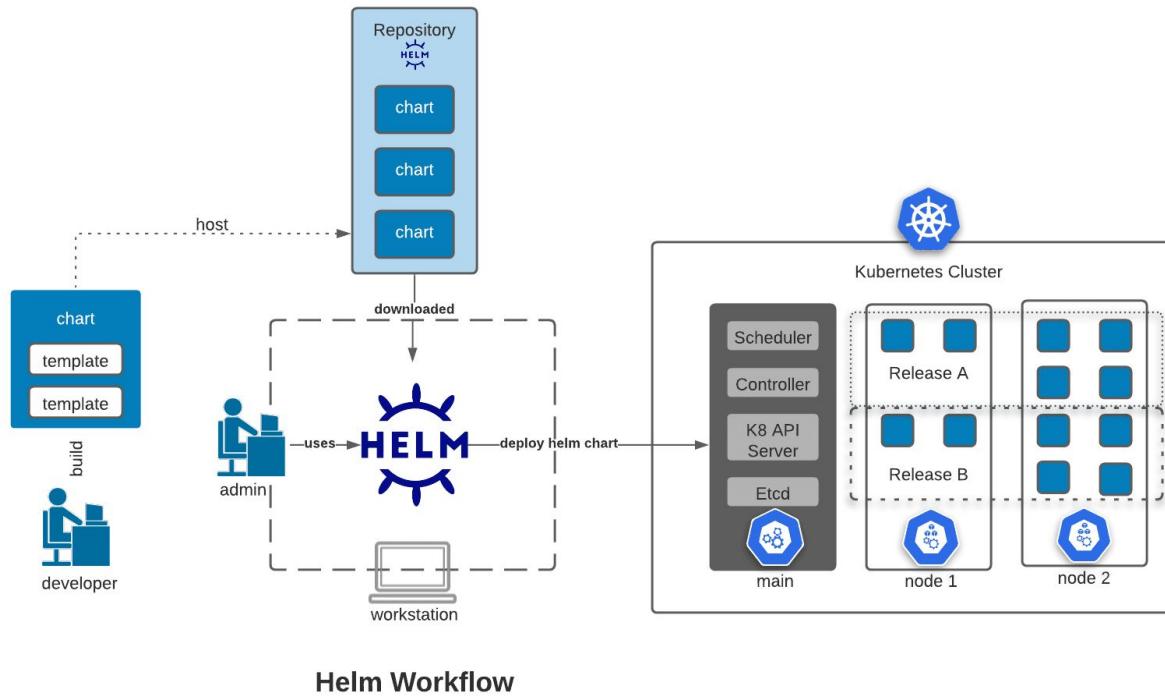


HELM Charts

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



HELM Charts



Workshop

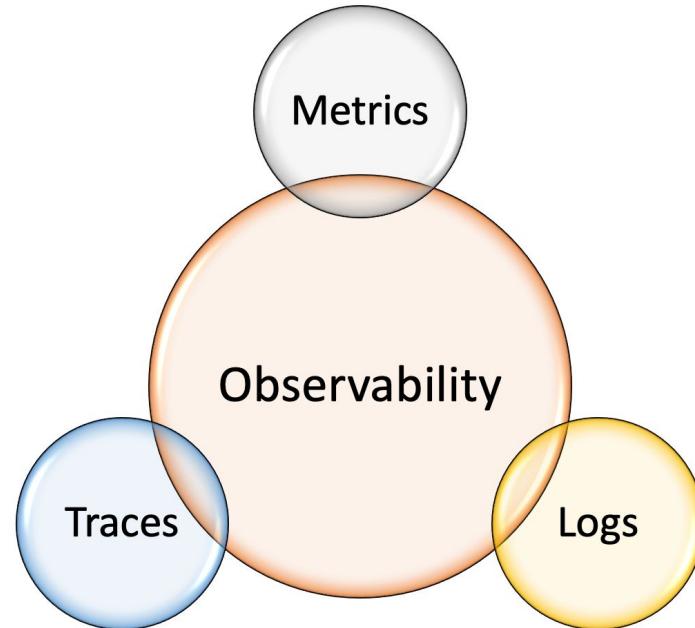
HELM

MONITORING



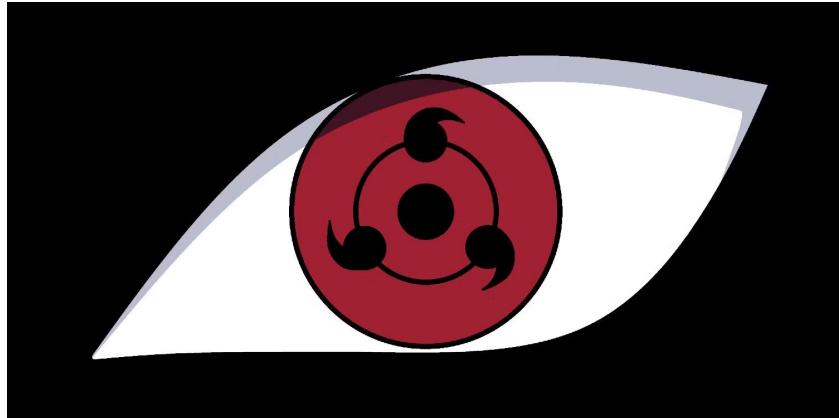
Three pillars of Observability

- Metrics
- Logs
- Traces

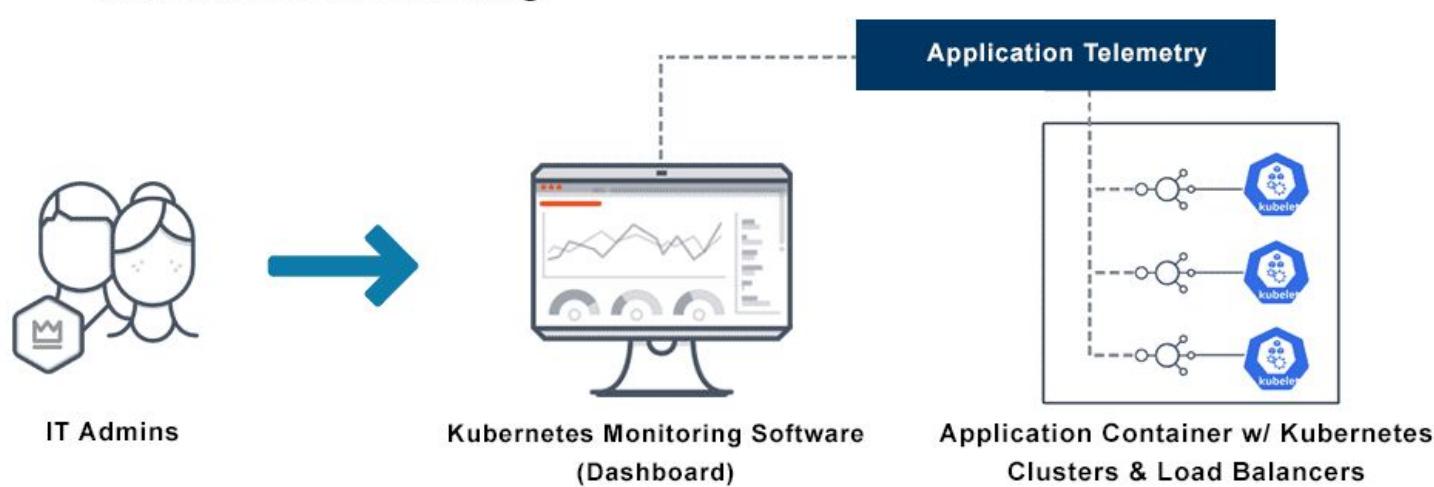


Three pillars of Observability

- 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

The screenshot shows the Kubernetes Dashboard interface. At the top, there is a navigation bar with a logo, a dropdown menu set to "default", a search bar, and several icons for creating, deleting, and filtering.

Workloads

Three large green circles represent the status of Deployments, Pods, and Replica Sets, each showing the count of running pods:

- Deployments: Running: 2
- Pods: Running: 3
- Replica Sets: Running: 2

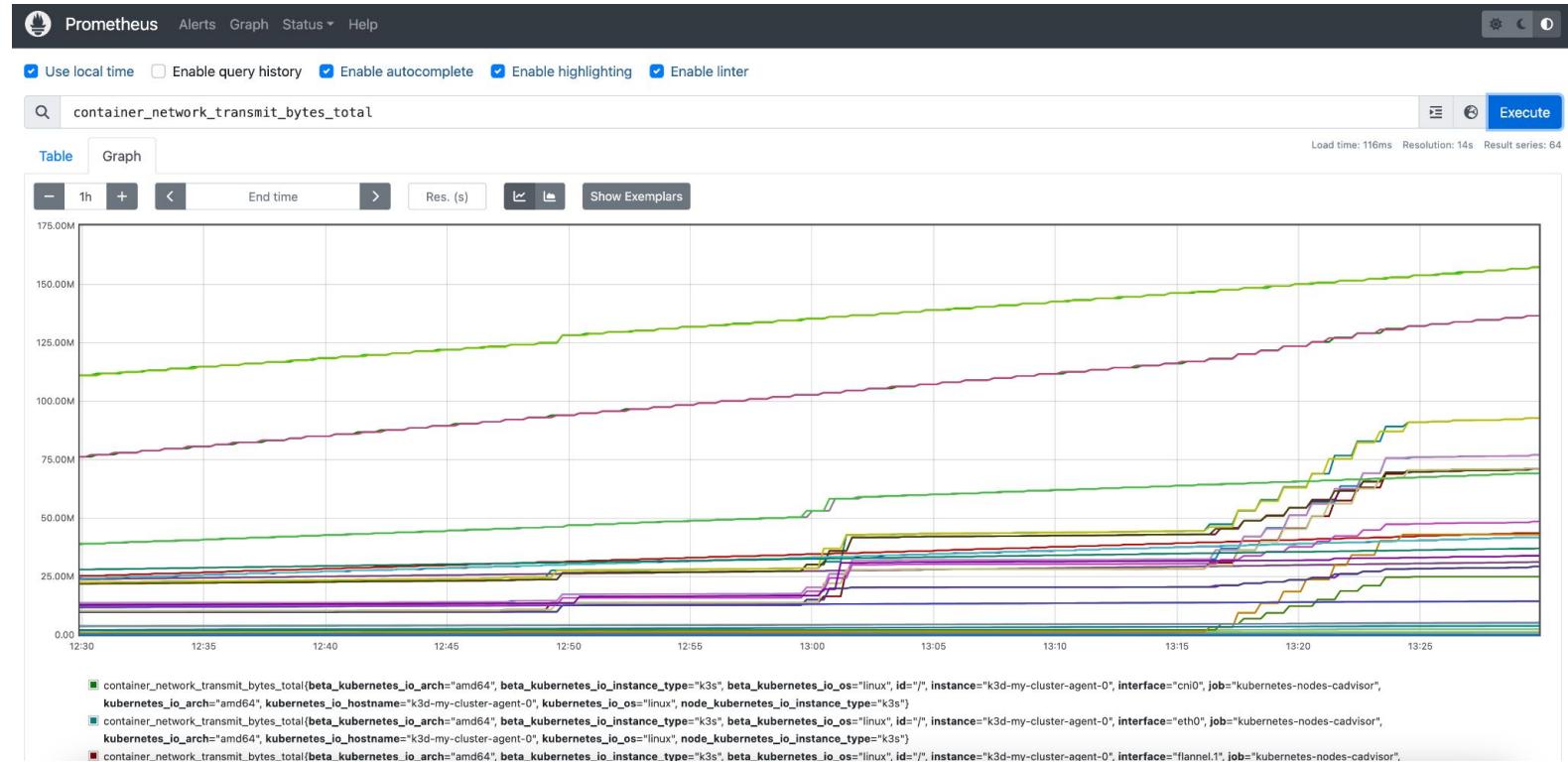
Deployments

| Name | Images | Labels | Pods | Created |
|----------------------|------------------------|--------|-------|-------------|
| mysql-deployment | mysql:8.0 | - | 1 / 1 | 3 hours ago |
| wordpress-deployment | wordpress:6.5.0-php8.1 | - | 2 / 2 | 3 hours ago |

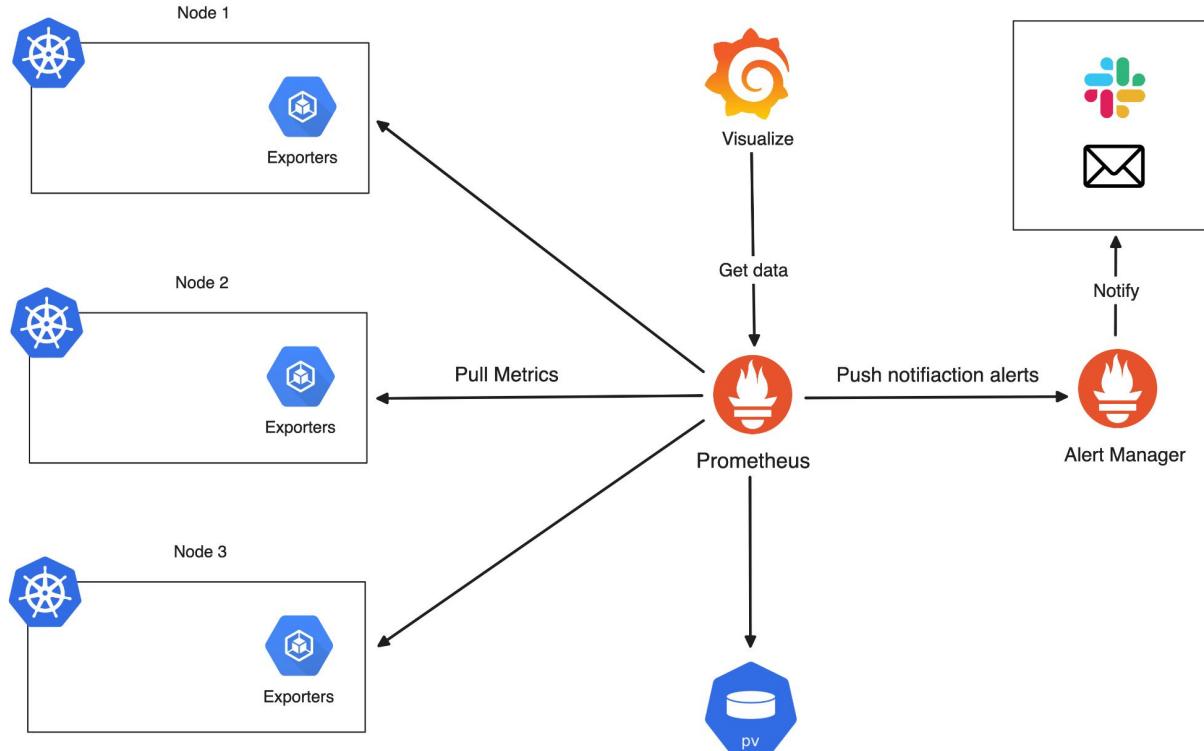
Pods

| Name | Images | Labels | Node | Status | Restarts | CPU Usage (cores) | Memory Usage (bytes) | Created |
|---------------------------------------|------------------------|--|------------------------|---------|----------|---|---|--------------|
| wordpress-deployment-5cd7bf5cff-7xmnc | wordpress:6.5.0-php8.1 | app: my-wordpress
pod-template-hash: 5cd7bf5cff | k3d-my-cluster-agent-0 | Running | 0 | <div style="width: 1.00m">1.00m</div> | <div style="width: 10.89Mi">10.89Mi</div> | a minute ago |
| wordpress-deployment-5cd7bf5cff-Szpc5 | wordpress:6.5.0-php8.1 | app: my-wordpress
pod-template-hash: 5cd7bf5cff | k3d-my-cluster-agent-1 | Running | 0 | <div style="width: 1.00m">1.00m</div> | <div style="width: 107.11Mi">107.11Mi</div> | 3 hours ago |
| mysql-deployment-57d5f9946c-jq4rb | mysql:8.0 | app: my-mysql
pod-template-hash: 57d5f9946c | k3d-my-cluster-agent-1 | Running | 0 | <div style="width: 18.00m">18.00m</div> | <div style="width: 370.61Mi">370.61Mi</div> | 3 hours ago |

Prometheus



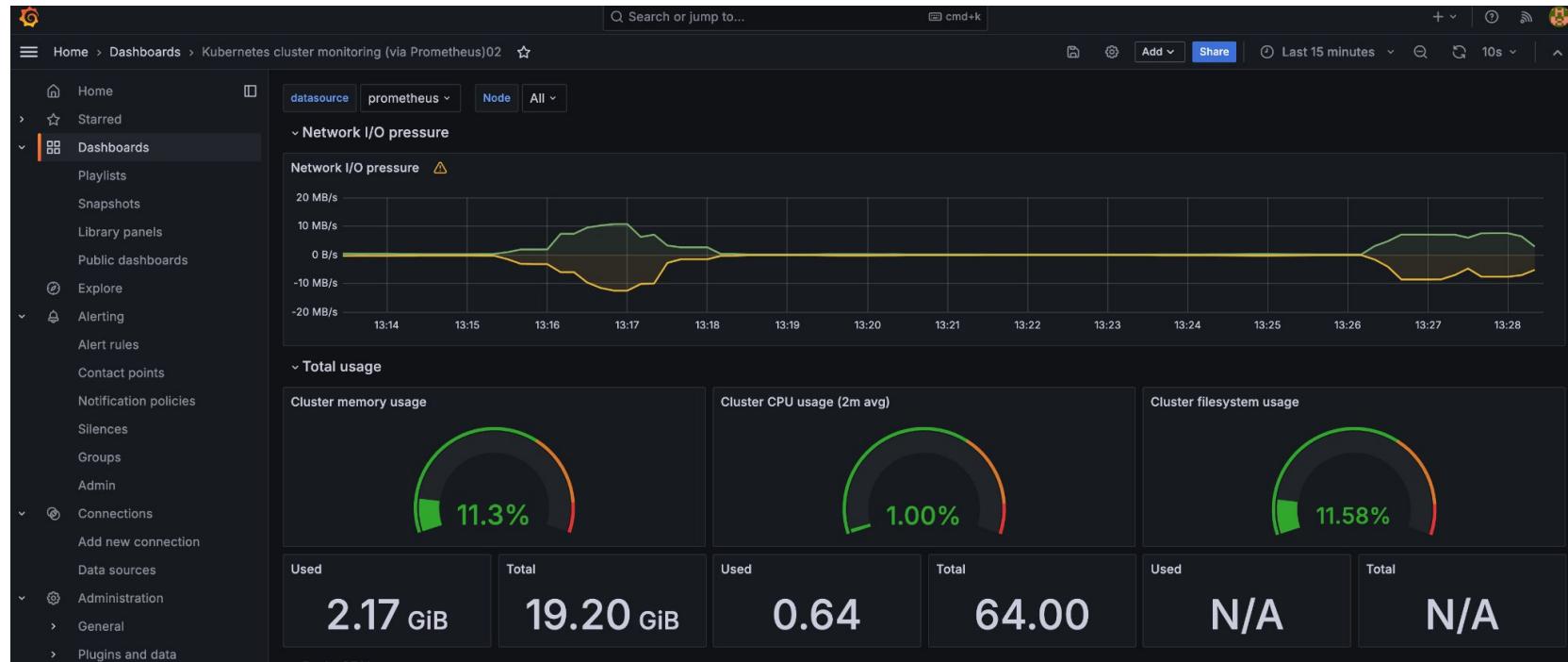
Architecture



Prometheus Alert Rule Examples

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

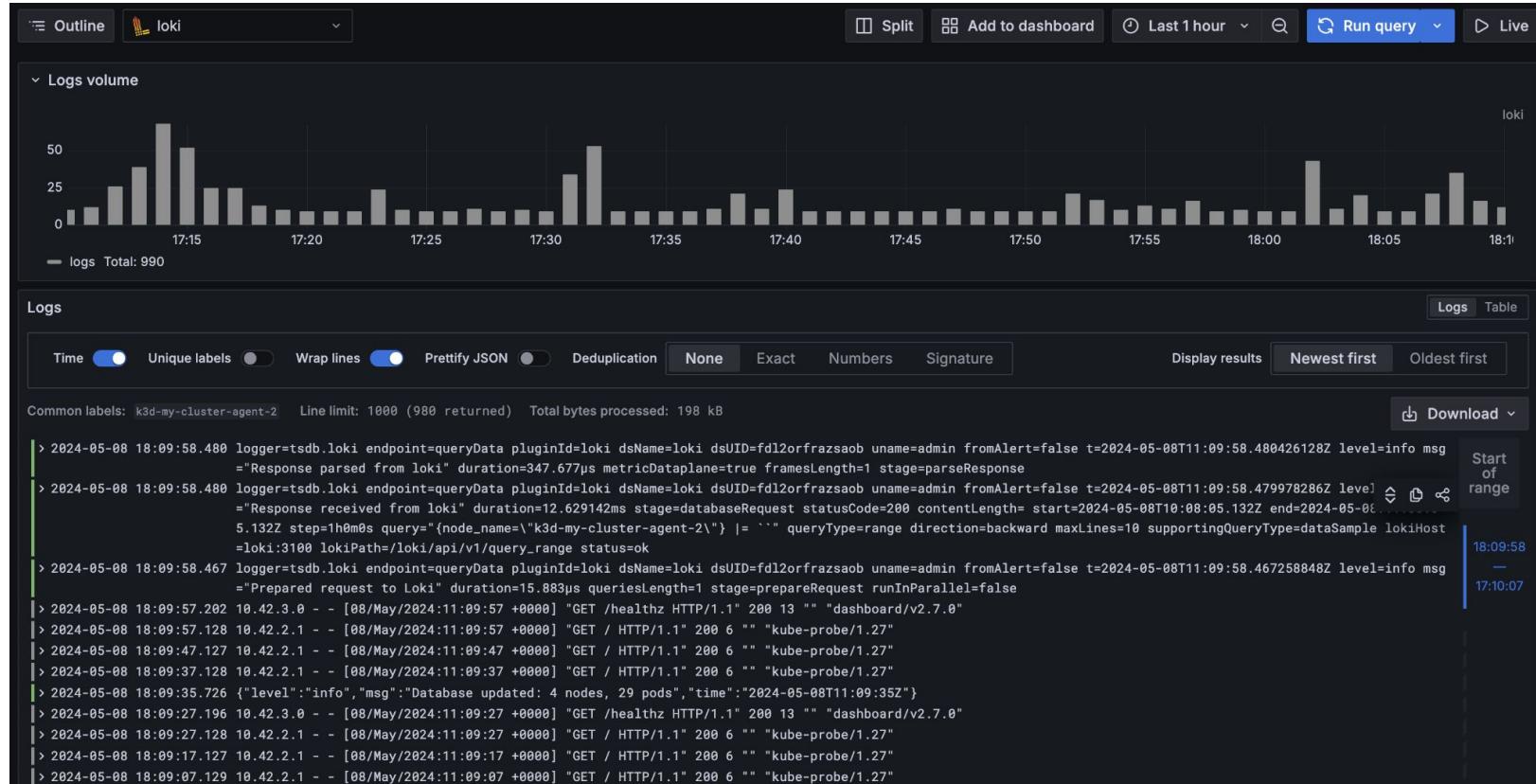
Grafana



Logs Monitoring

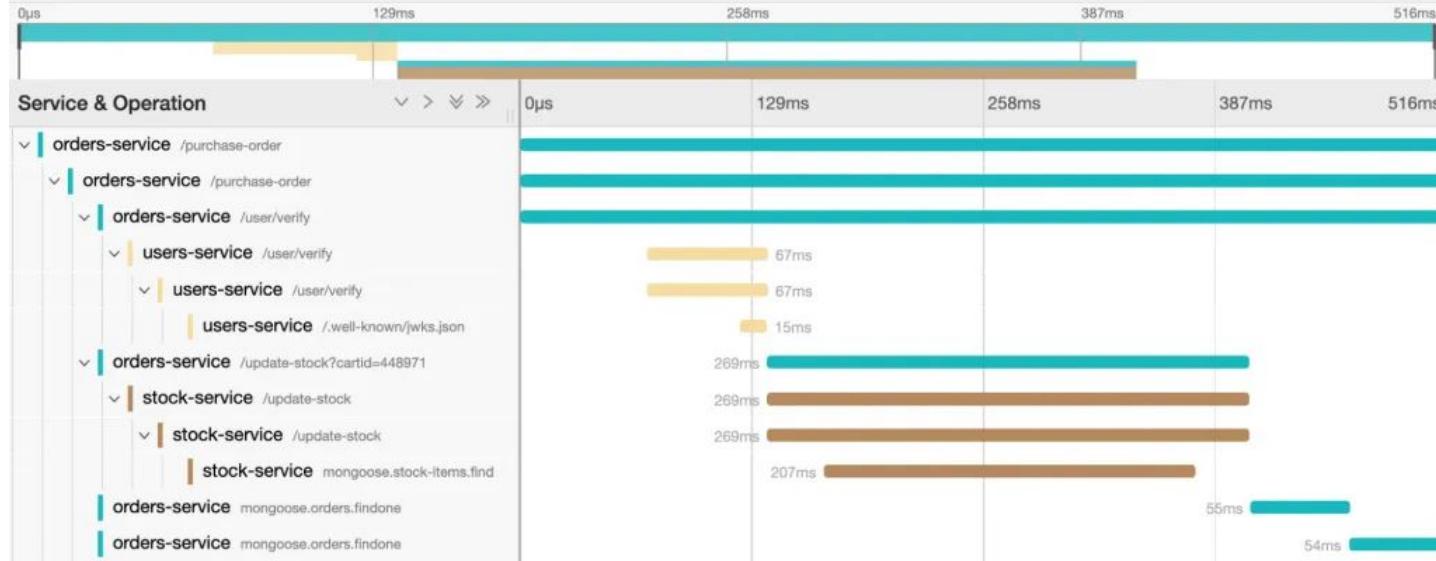
- Cluster logs
- Node logs
- Container logs
- Kubernetes Events

Loki

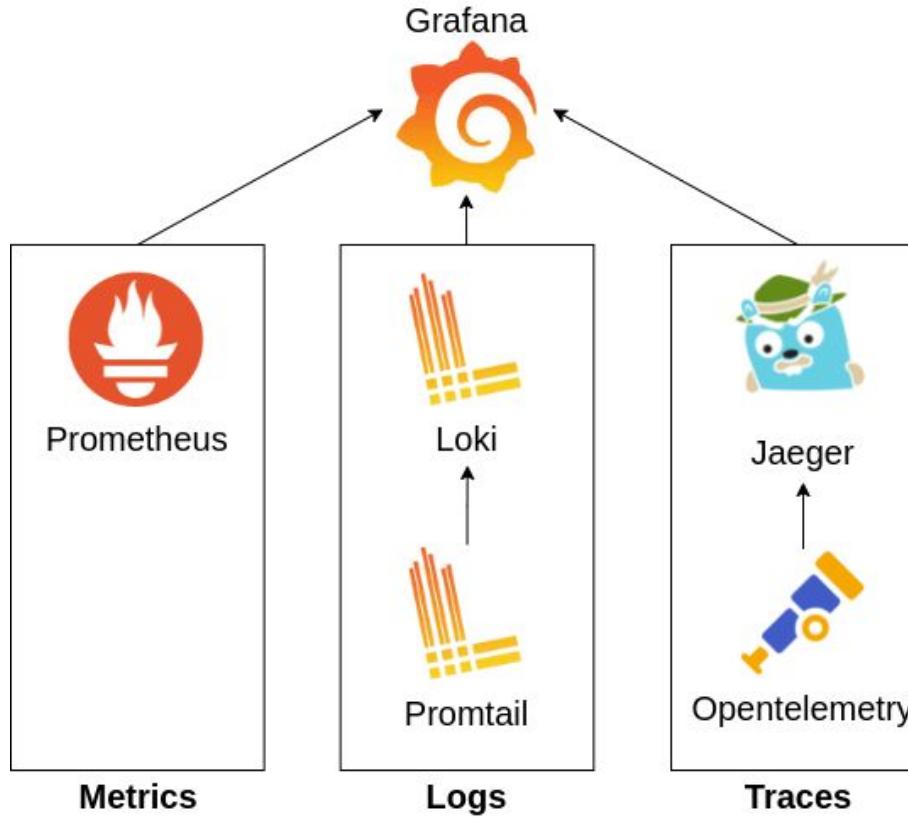


Tracing Monitoring

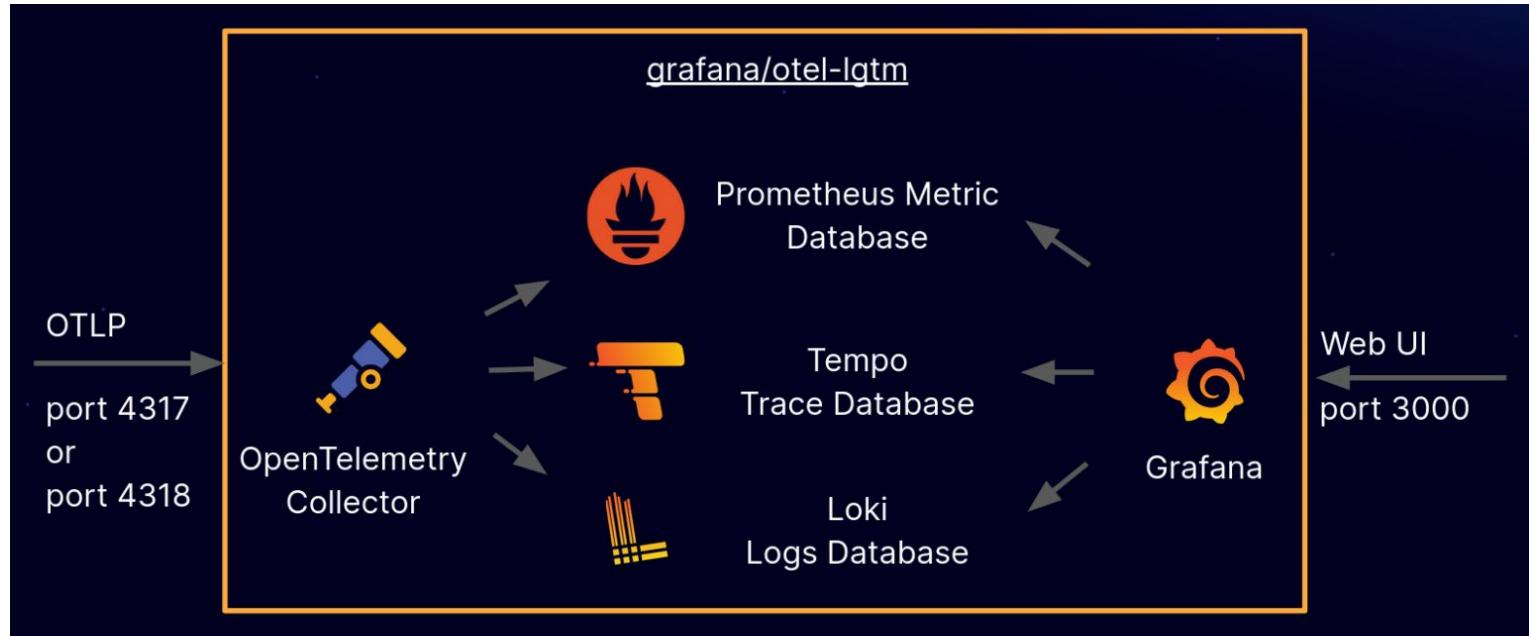
- A trace represents the entire journey of a request or action as it moves through all the nodes of a distributed system



Monitoring



Monitoring



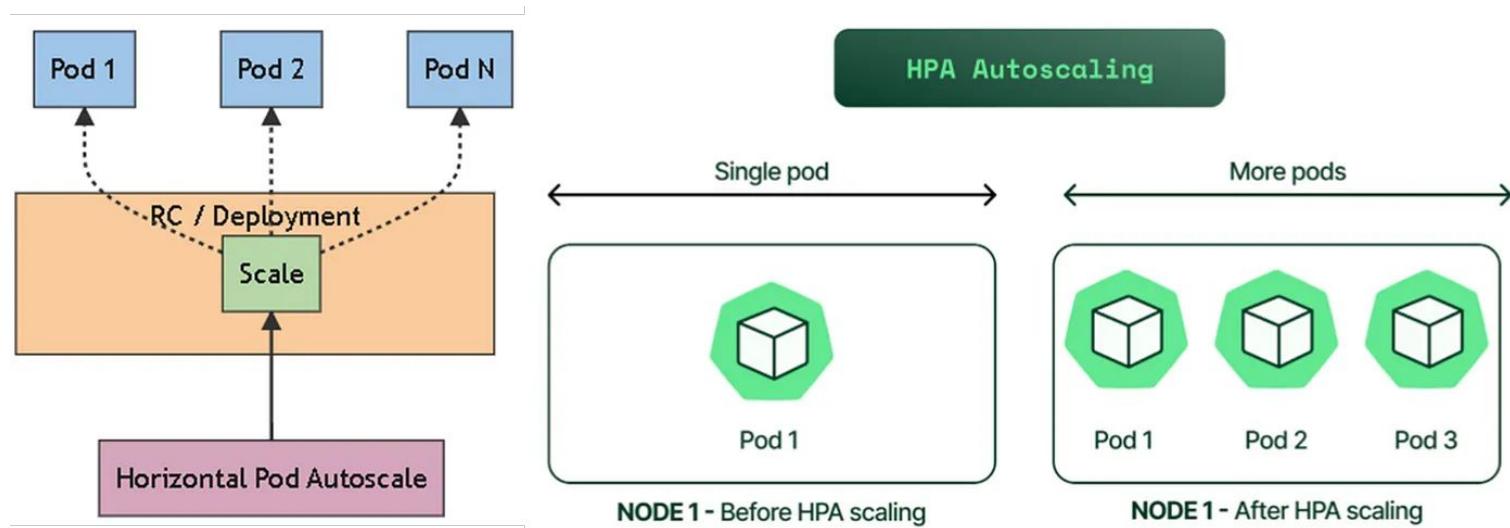
Workshop Monitoring

HPA (Horizontal Pod Autoscaler)

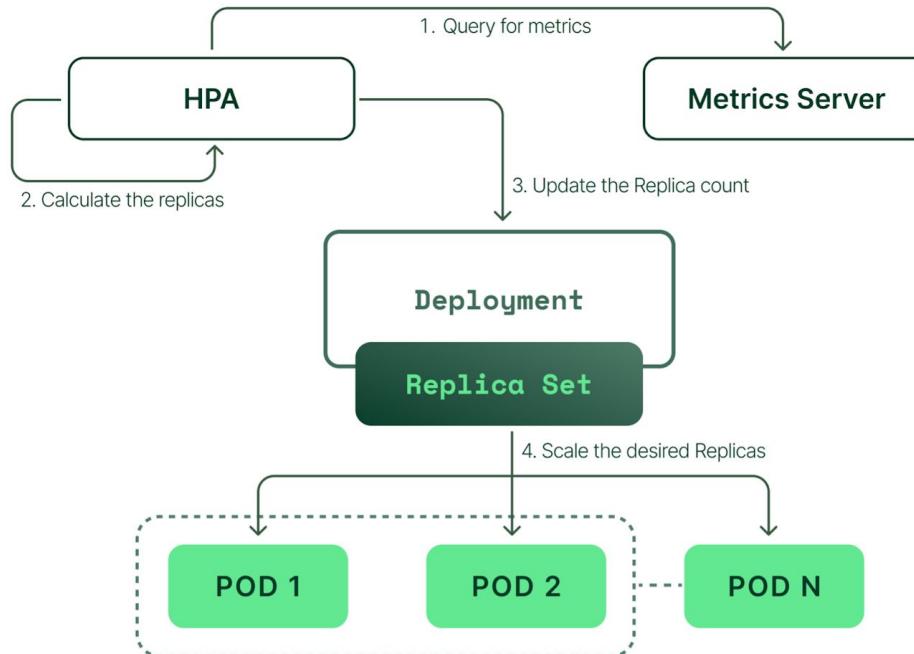


HPA

- 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

Recreate deployment

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

Advantage

- Application state entirely renewed

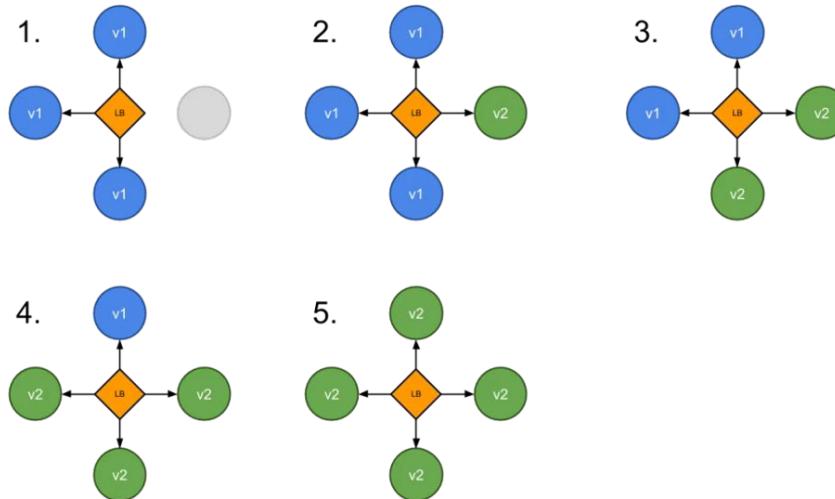
Disadvantage

- 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

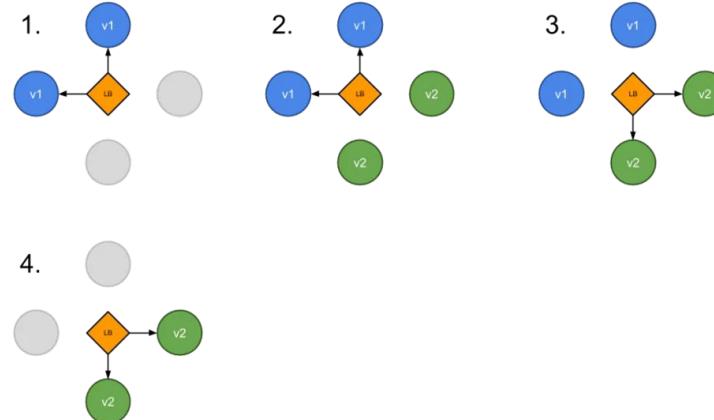
Disadvantages

- 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

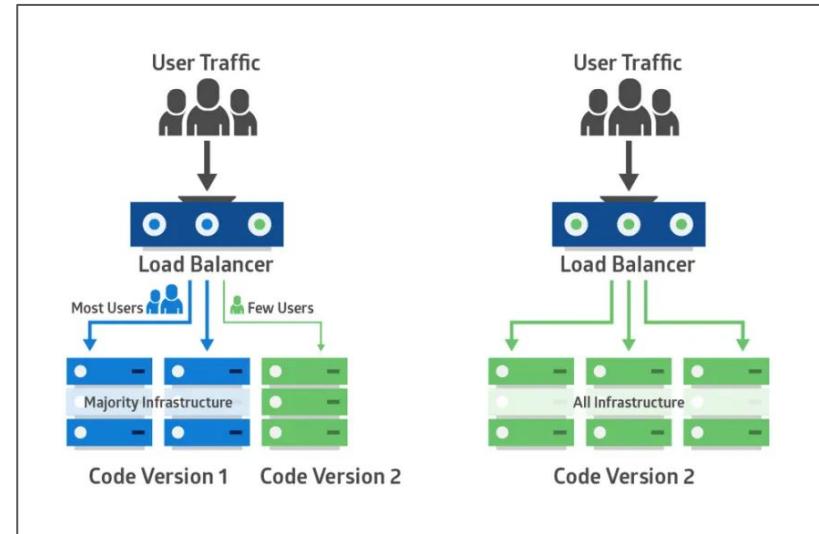
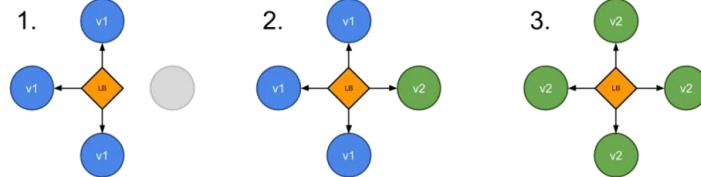
Disadvantages

- 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

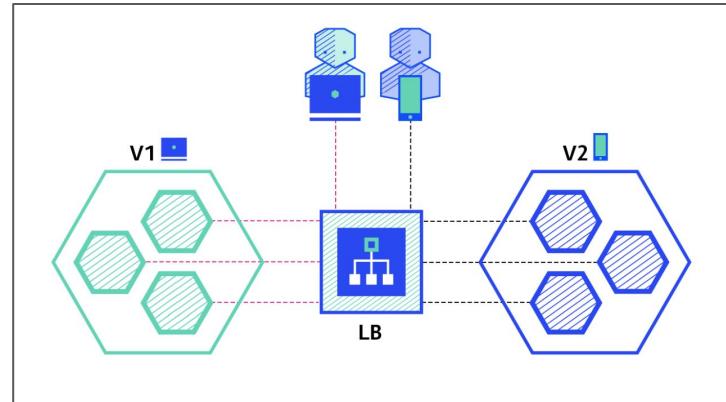
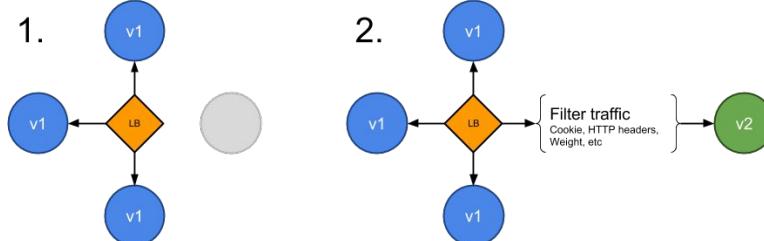
Disadvantages

- 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

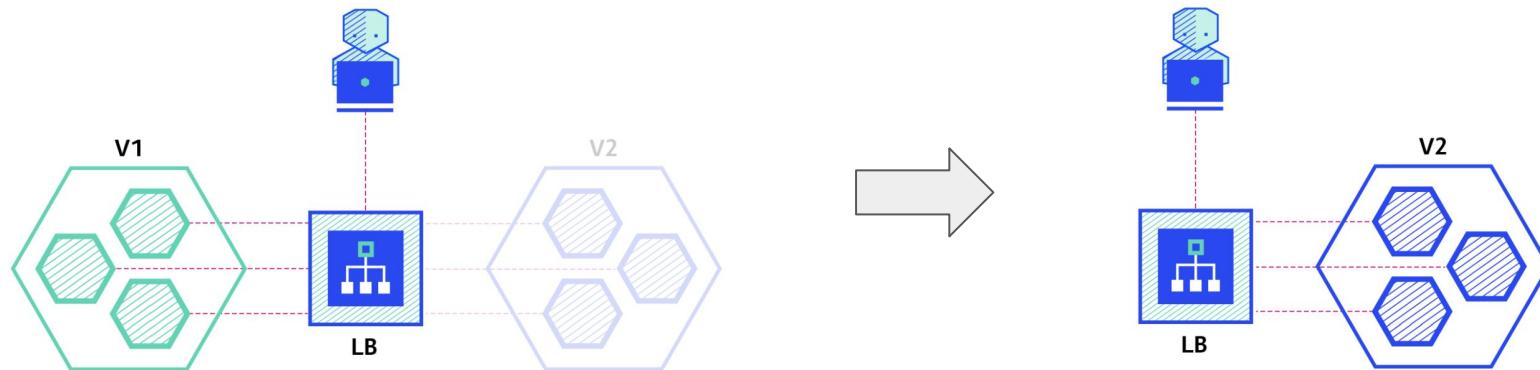
Disadvantages

- 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.



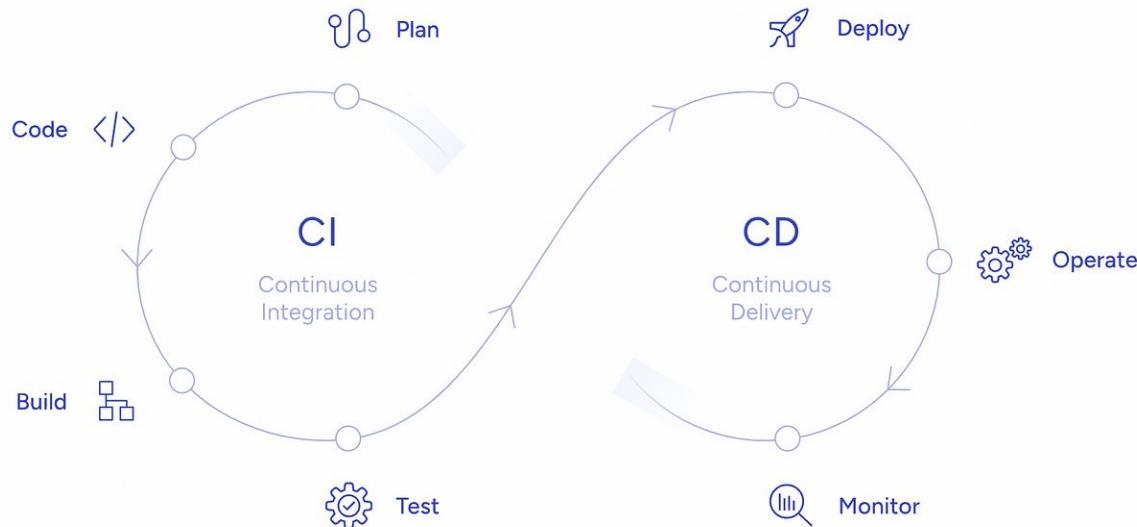
Container
Solutions

| 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 ?

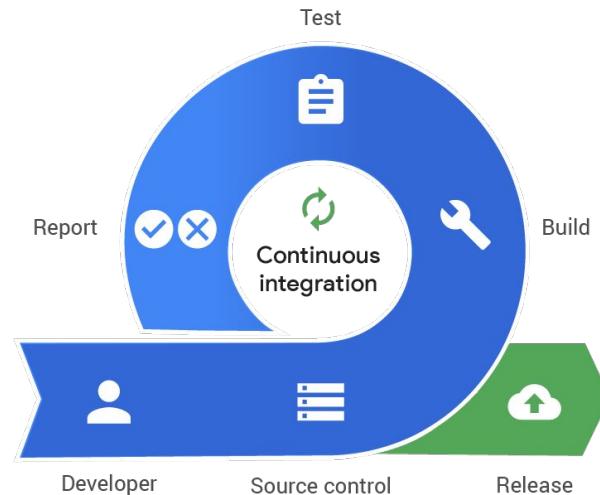
- Continuous Integration and Continuous Deployment (CI/CD) pipeline represents an **automatic workflow** that continuously integrates code developed by software developers and deploys them into a target environment with less human intervention.



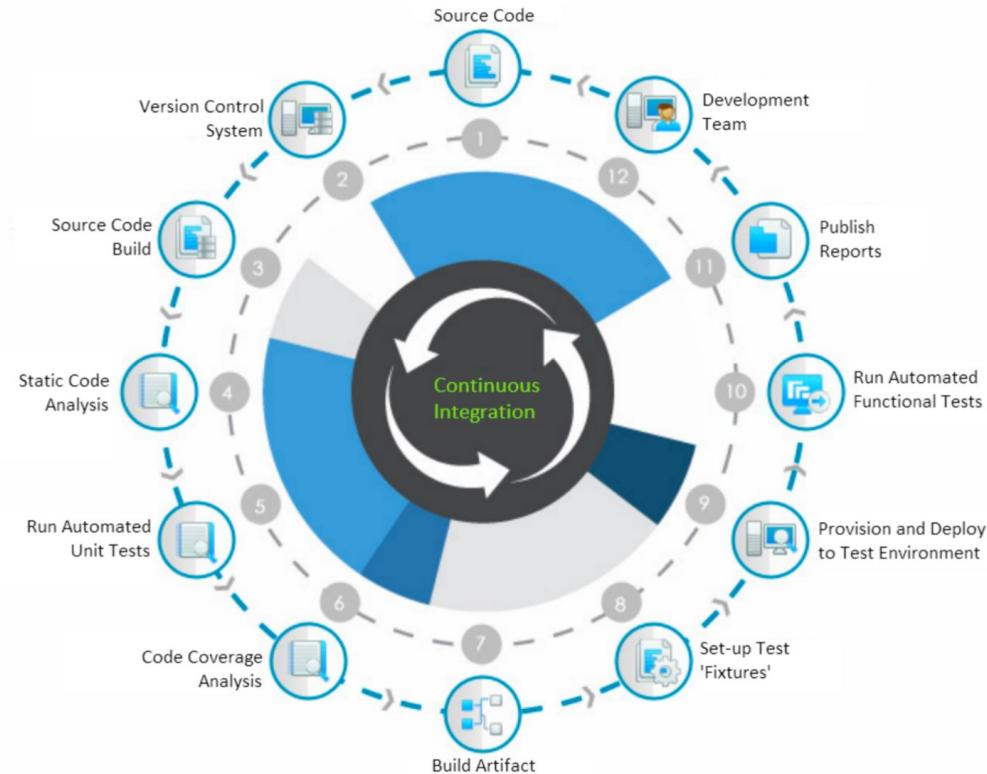
Continuous Integration or CI pipeline

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



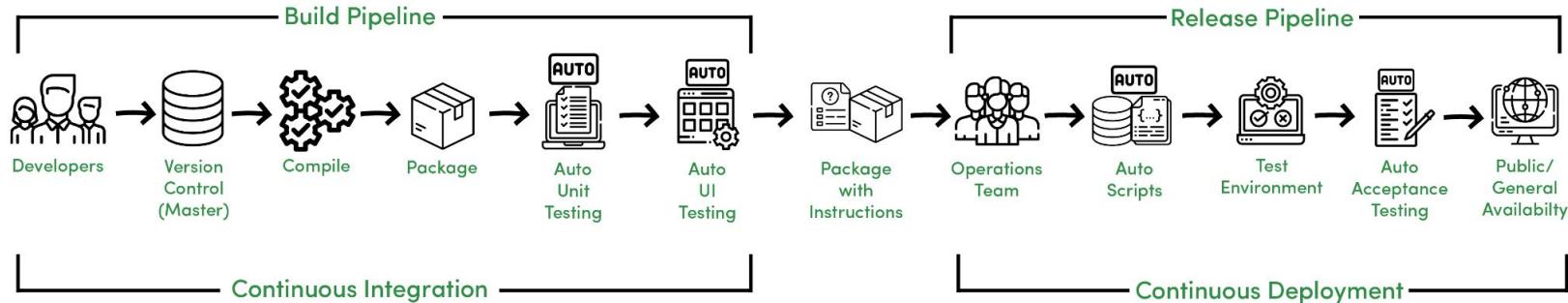
Continuous Integration or CI pipeline



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

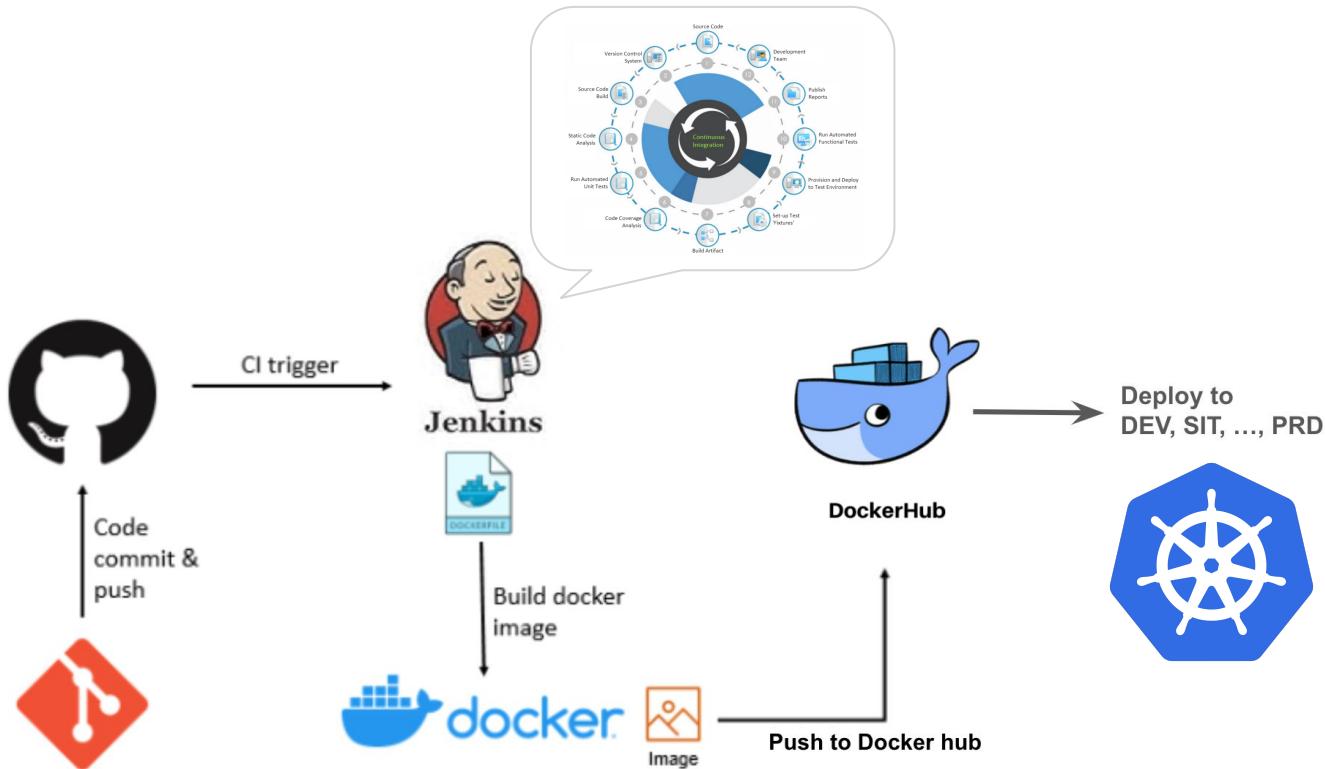
- Once the code passes CI checks, CD takes over
- Some approaches call **Continuous Delivery**: Makes the built code readily available for deployment, allowing for manual review before pushing it live



Benefits of CI/CD Pipelines

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

Apply Docker & k8s with pipelines



Workshop

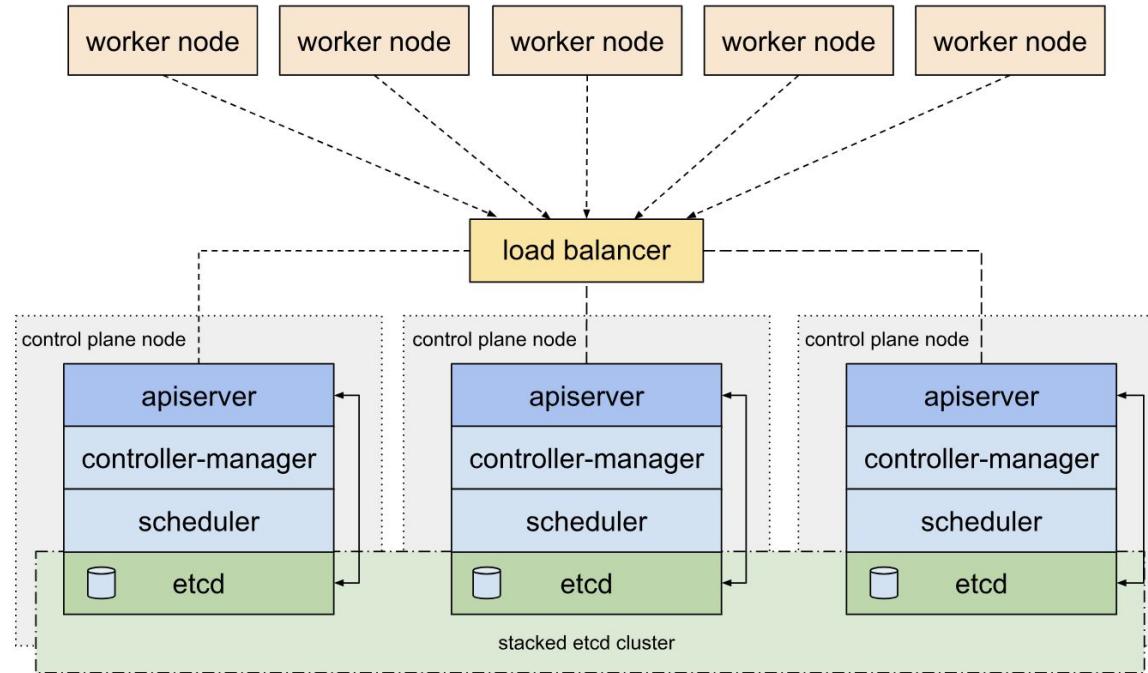
CI / CD Pipelines

Cluster Setup

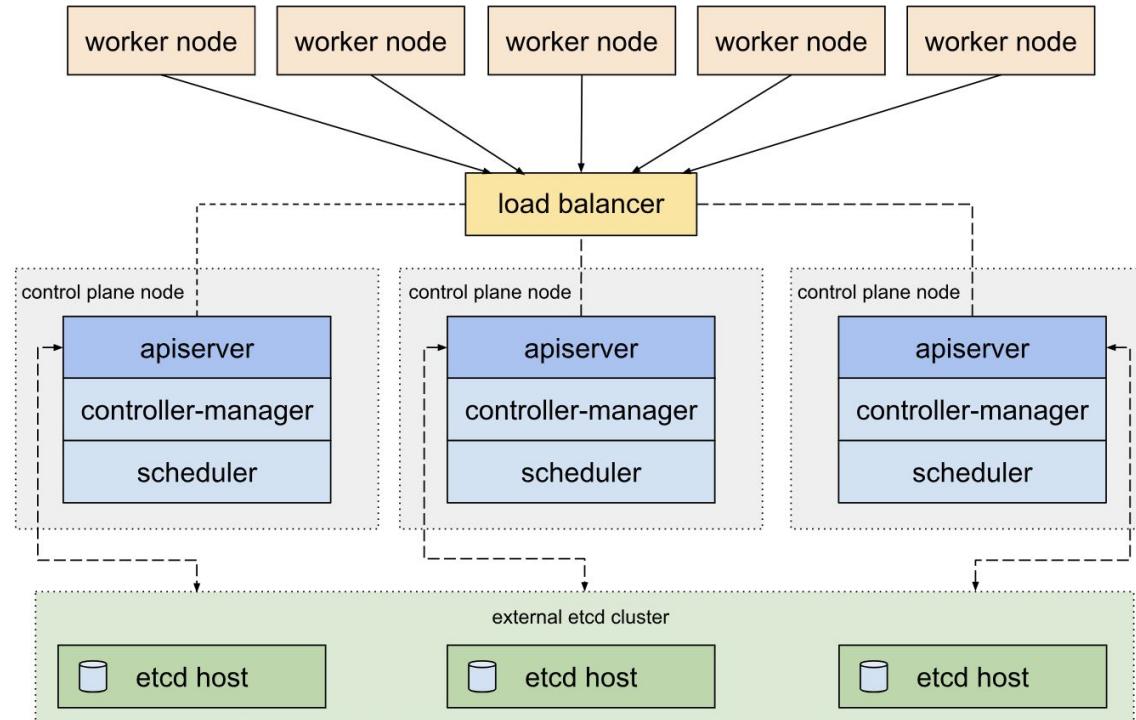
Options for Highly Available Topology

- Stacked etcd topology
- External etcd topology

Stacked etcd topology



External etcd 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 |

Kubernetes Engine - On Cloud



EKS

Amazon
Elastic Kubernetes Service



AKS

Azure
Kubernetes Service

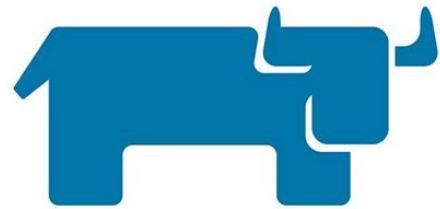


GKE

Google
Kubernetes Engine

Kubernetes - On Premise

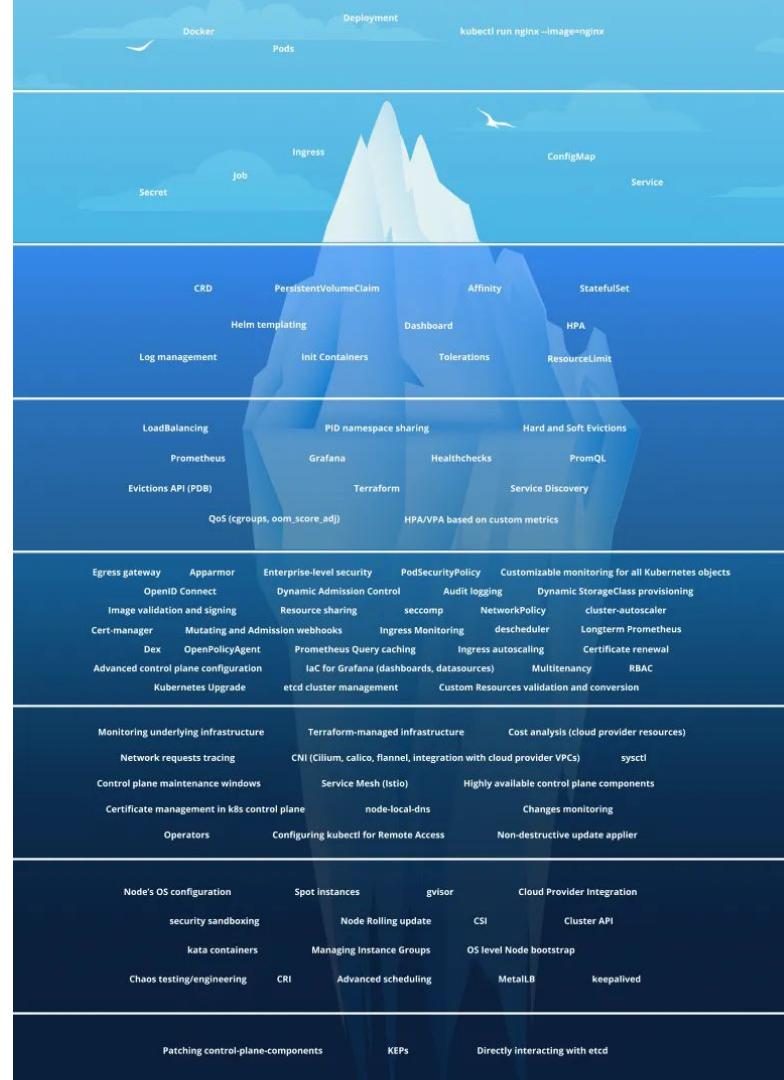


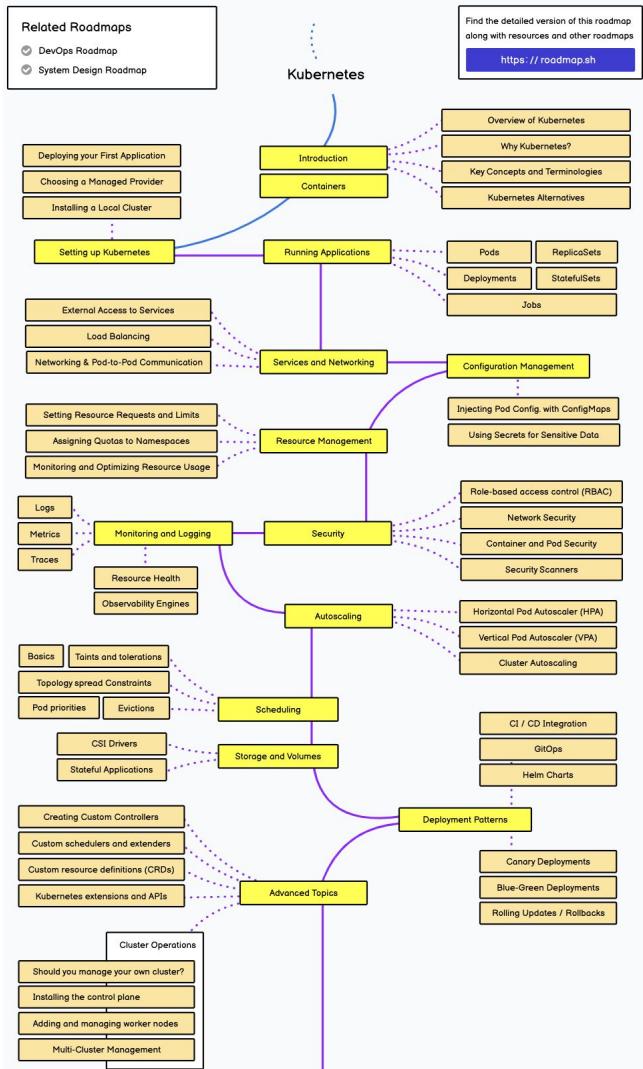


RANCHER

Rancher

| Managed Infrastructure Size | Maximum Number of Clusters | Maximum Number of Nodes | vCPUs | RAM |
|-----------------------------|----------------------------|-------------------------|-------|-------|
| Small | 150 | 1500 | 4 | 16 GB |
| Medium | 300 | 3000 | 8 | 32 GB |
| Large (*) | 500 | 5000 | 16 | 64 GB |
| Larger (†) | (†) | (†) | (†) | (†) |





**First day
working with 3 months later...
Kubernetes**



Q & A