Kubernetes: Crafting High Availability of Services

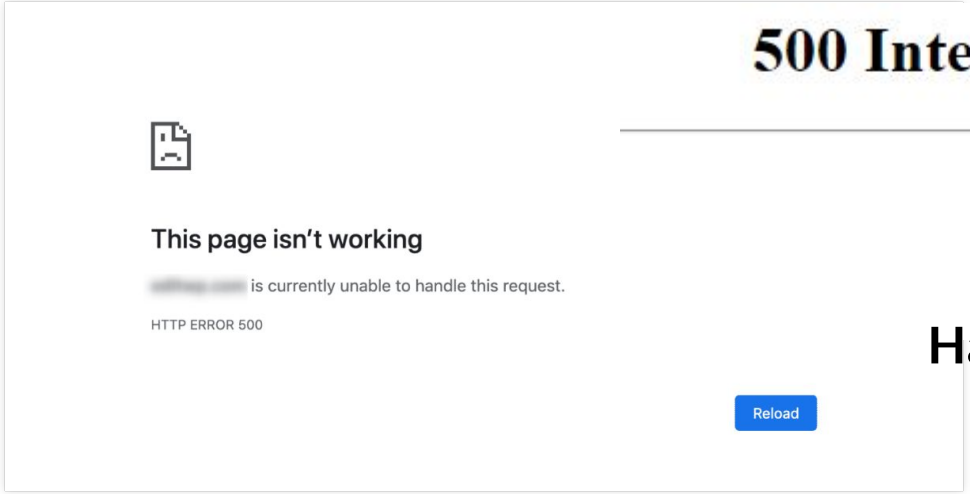
Sharing Session
Speaker : Gilang Virga Perdana

Adaptive Network
Laboratory

Profile

- Nick -> Gilang
- Education -> Bachelor of Telecommunication Engineering at Telkom University (2019 - 2023)
- Specialty Field -> Cloud Computing (DevOps, Cloud Infra)
- Professional Experience :
    - Laboratory -> Adaptive Network Laboratory (2020- 2023) as Cloud Research Assistant
    - Industry -> PT. Boer Technology (2022) as Cloud Engineer
- Certification :
    - Amazon Cloud Practitioner
    - Alibaba Cloud Associate
    - Adinusa (Kubernetes, Openstack, Ceph, Ansible, etc)
- Let’s connect on Linkedin -> linkedin.gbesar.com

Before, Have you ever ?



Have you ever see that page?

Have you ever thought about how Google, Microsoft, etc have minimum downtime?

Kubernetes: Crafting High Availability of Services

Disclaimer

- If you want to Handson together, **please read slide 27**
- This presentation contains **basic** understanding, **equivalent to beginner-intermediate level**
- This presentation is just an **outline**, please ask during the rest of the **QnA session**
- *Nobody is smarter here. Let's study together.*

Agenda

- Background
- Introduction
- Demo & Use Case
- QnA

Objective

- Understanding of Virtualization
- Understanding of Containerization
- Understanding of Kubernetes cluster Implementation
- Understand the deployment of applications running on Kubernetes
- Understand today's application architecture

Background

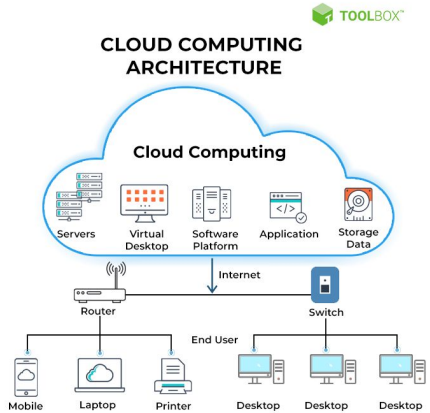
- All digital public service now are **Software Based**.
- Service that is present **requires** having **good availability**.
- Services that can be accessed by everyone are hosted in an environment called the **Cloud.**
- Cloud architecture **must be well designed** to **achieve good availability**.
- One approach to improving good availability on **Cloud** is to implement container orchestration (**Kubernetes**).

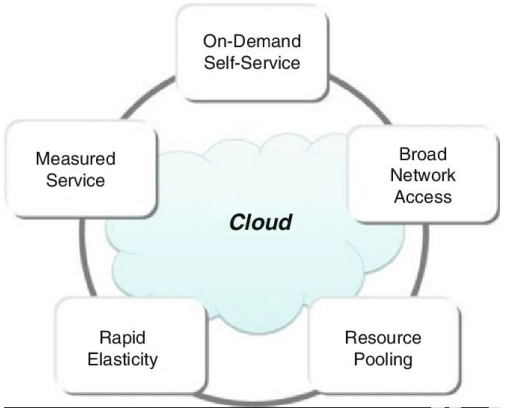
Introduction

Cloud Technology & Characteristic

What is **Cloud** ?

- A computational paradigm in which resources are **unlimited** and it is not necessary to **know where they are (Rapid Elasticity)**.
- The technology supporting the running of cloud computing is usually called **Cloud Native.**
- Type of Cloud : Hybrid, Private, Public
- Cloud technology adopts a lot of concept of **Virtualization**.





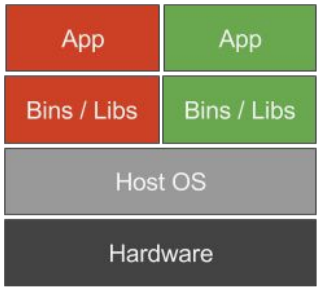
More : https://landscape.cncf.io/

Virtualization

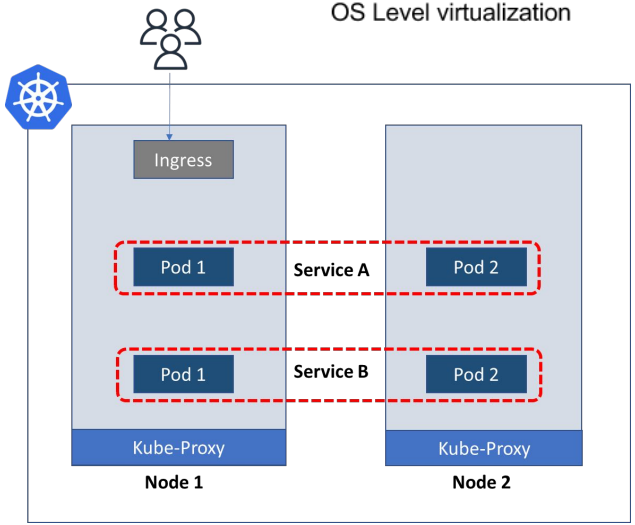
Elasticity in cloud computing gave birth to virtualization technology.

Types of virtualization :

- Full Virtualization -> VMWare Esxi, Virtualbox, Openstack.
- Paravirtualization -> Microsoft Hyper-V, Citrix, Xen.
- **OS-Level Virtualization** -> Containerization (Docker,Containerd -> **Kubernetes**).
- **Network Function Virtualization** -> **Kubernetes** Networking (Implementation of Overlay Network)

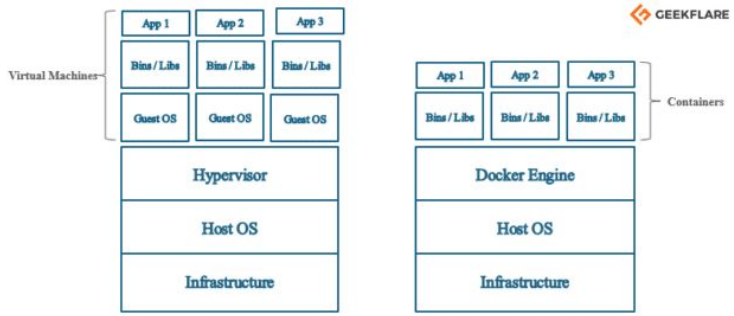


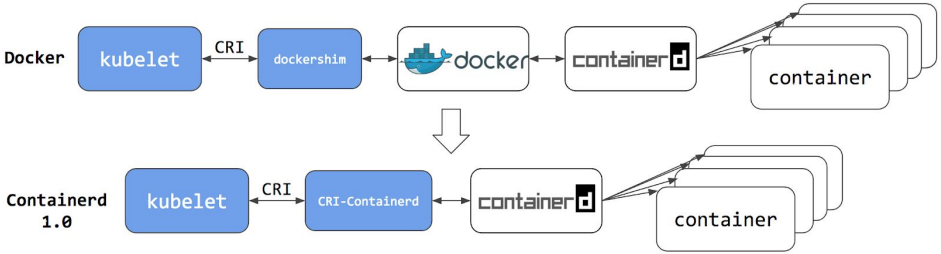
OS Level virtualization

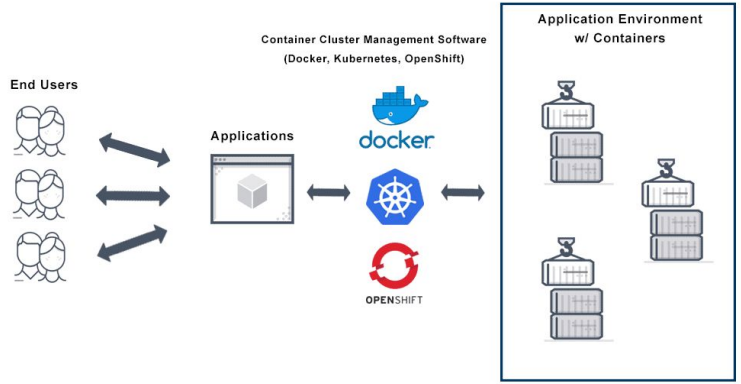


Container

- Now, all software-based services will always be wrapped into a paradigm called **Containers**.
- **Why**? This was originally intended to make it easier to run an application wherever the host is.
- But in this session the **container** is **intended** for **Kubernetes, Kubernetes only accepts applications that have been packaged into containers.**
- Example Container Tech : *Docker*, *Containerd*, *CRI-O*.





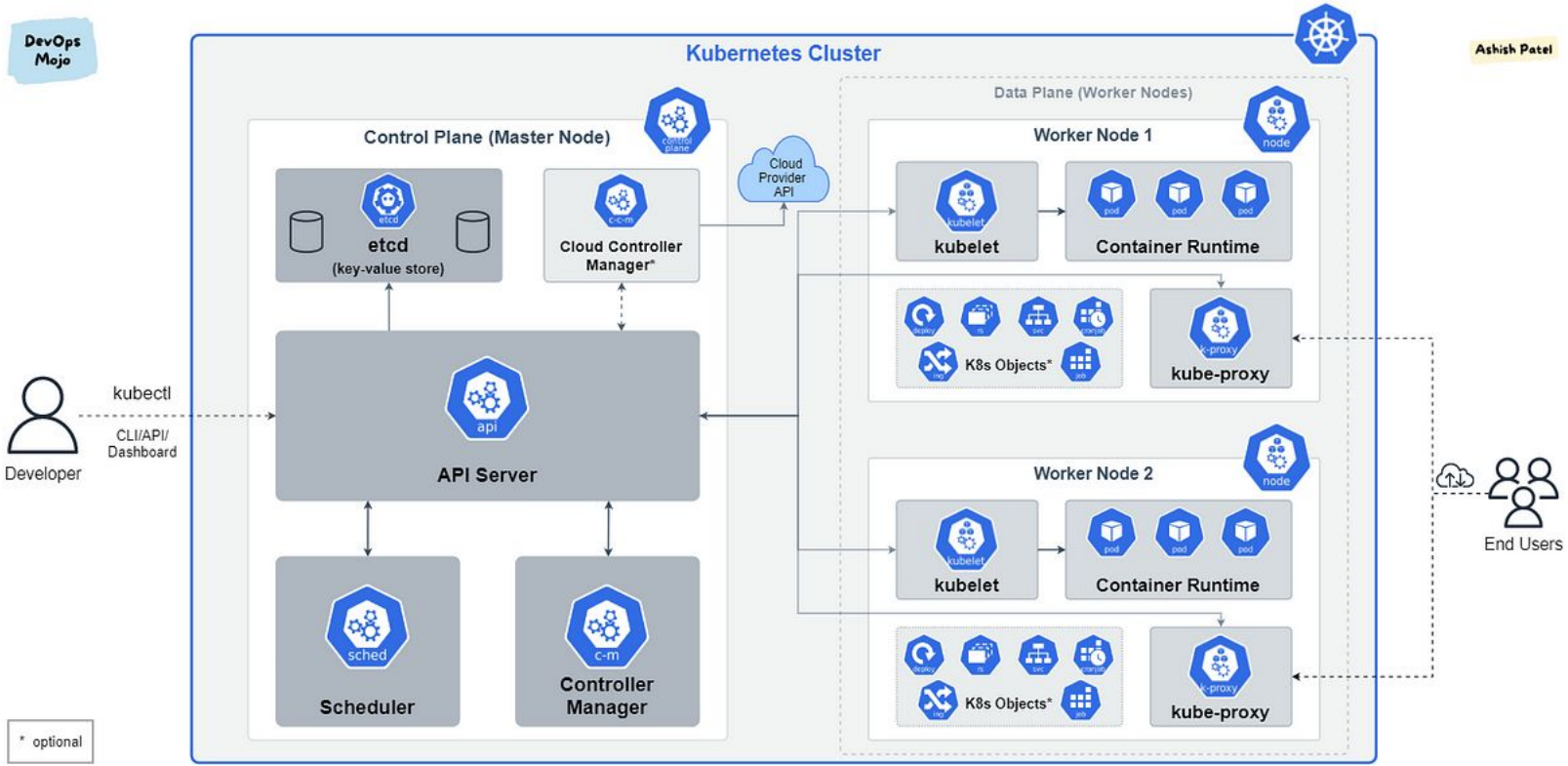


Paradigm on Kubernetes on this Session

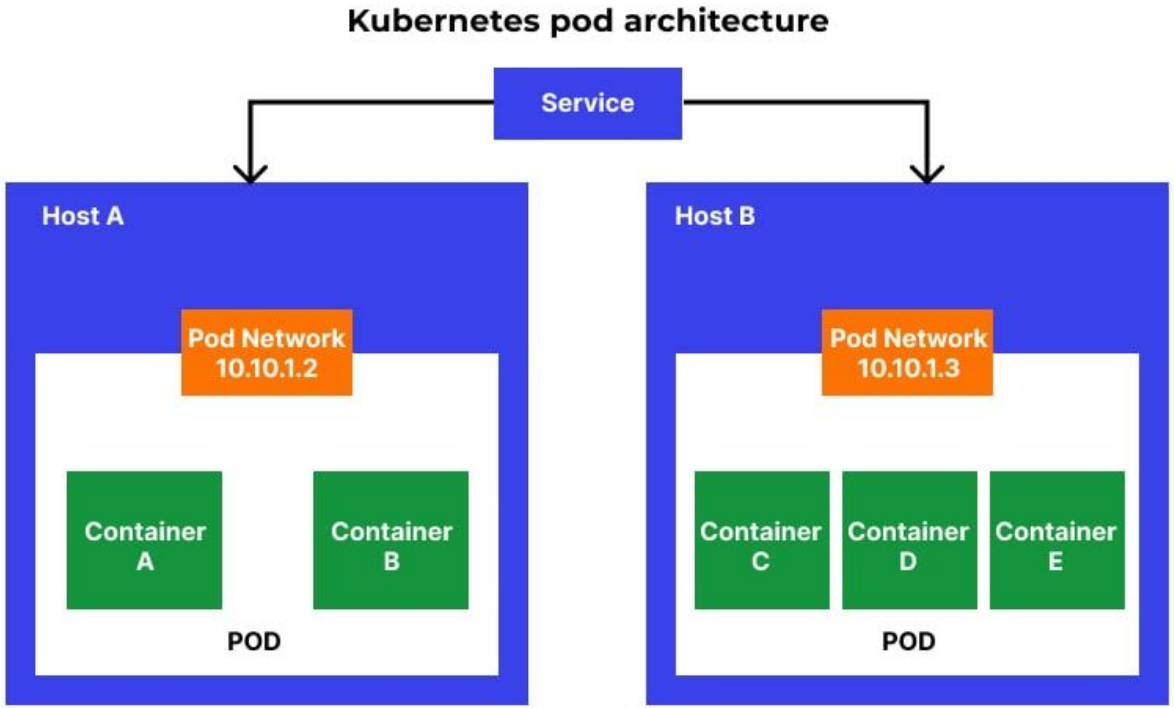
Paradigm	Description
Node	A worker machine in Kubernetes and may be either a virtual or a physical machine.
Pod	Contains containers, can be more than 1 container in 1 pod.
Deployment	Responsible for keeping a set of pods running.
Service	Responsible for enabling network access to a set of pods.
Ingress	Responsible for assigning network rules to each pod accessed from the client.
Namespace	A groups of resources on a Kubernetes single cluster.
Volume	Represents a piece of storage that you can attach to your Pod(s).

More *: https://kodekloud.com/blog/kubernetes-objects/*

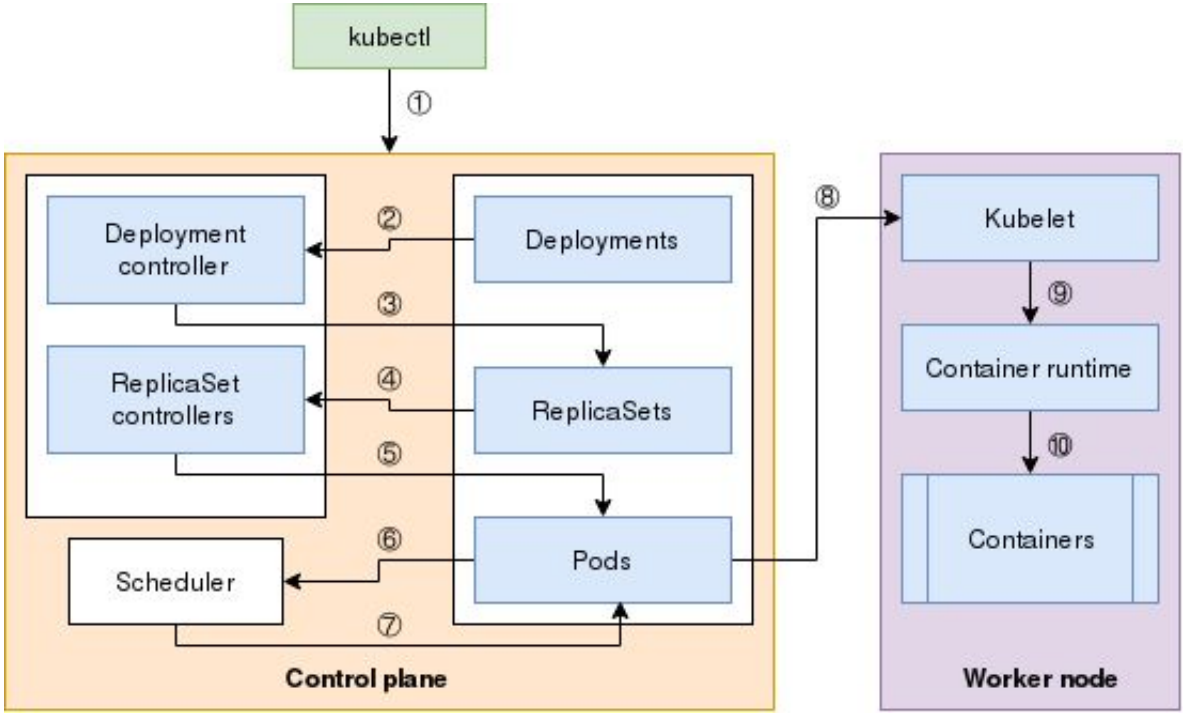
Node


DevOps Mojo
Ashish Patel
Kubernetes Cluster
Data Plane (Worker Nodes)
Control Plane (Master Node)
Worker Node 1
Worker Node 2
etcd
(key-value store)
Cloud Controller Manager*
Cloud Provider API
kubelet
Container Runtime
K8s Objects*
kube-proxy
API Server
Scheduler
Controller Manager
kubectl
CLI/API/ Dashboard
Developer
End Users
* optional

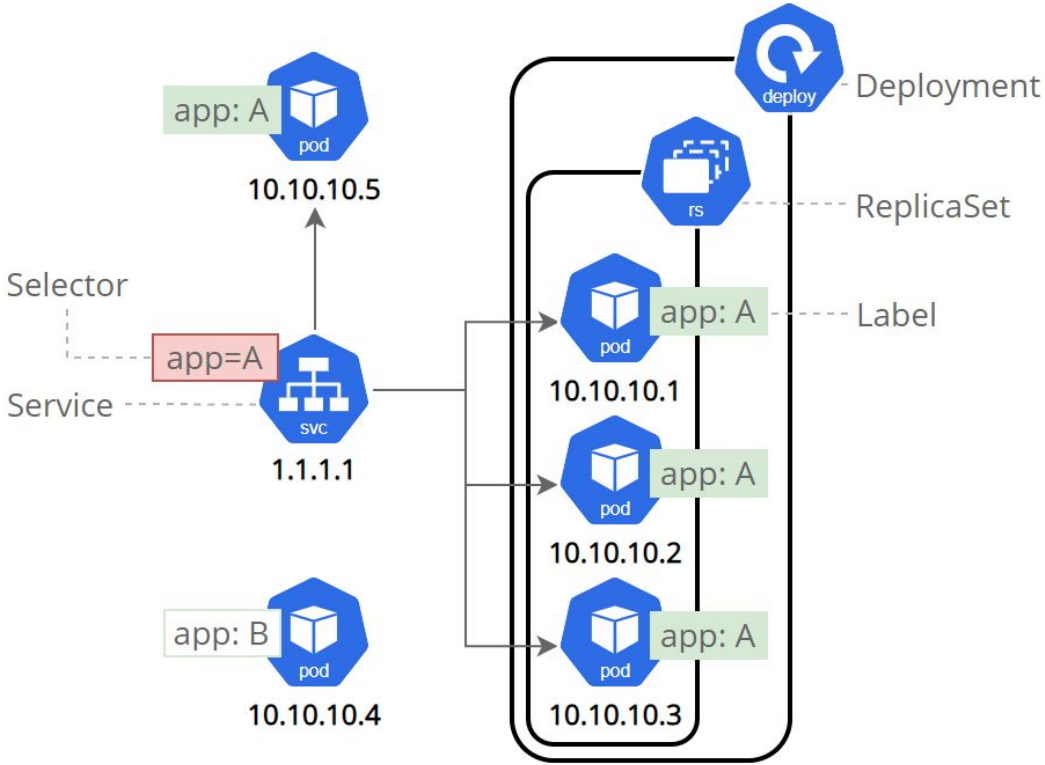
Pod


Kubernetes pod architecture
Service
Host A
Pod Network
10.10.1.2
Container
A
Container
B
POD
Host B
Pod Network
10.10.1.3
Container
C
Container
D
Container
E
POD

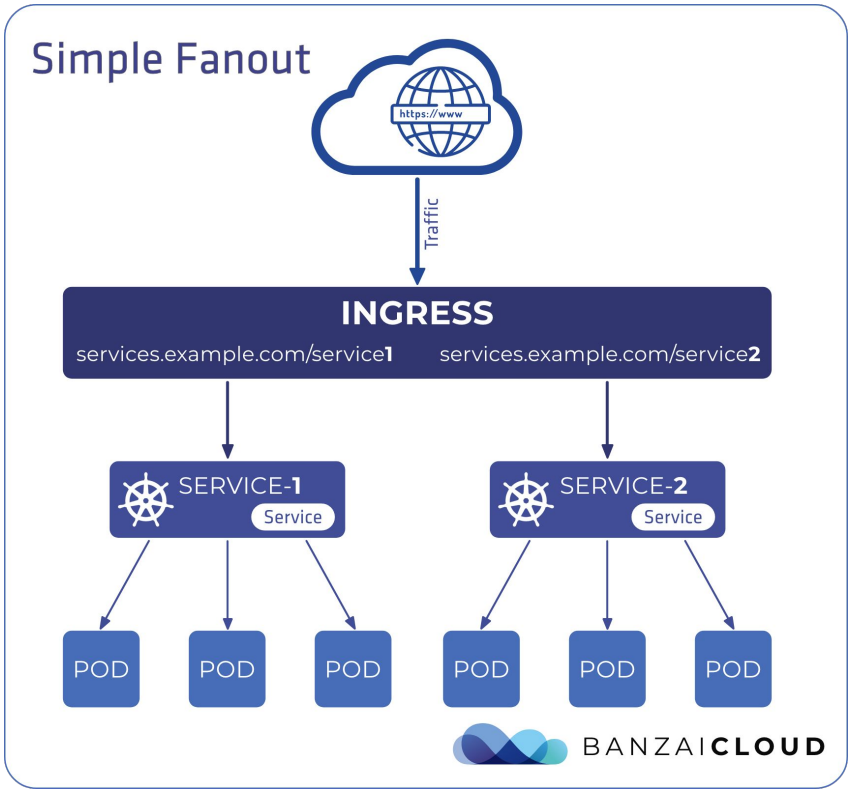
Deployment


kubectl
①
Deployment controller
②
Deployments
③
ReplicaSet controllers
④
ReplicaSets
⑤
Scheduler
⑥
Pods
⑦
Control plane
⑧
Kubelet
⑨
Container runtime
⑩
Containers
Worker node

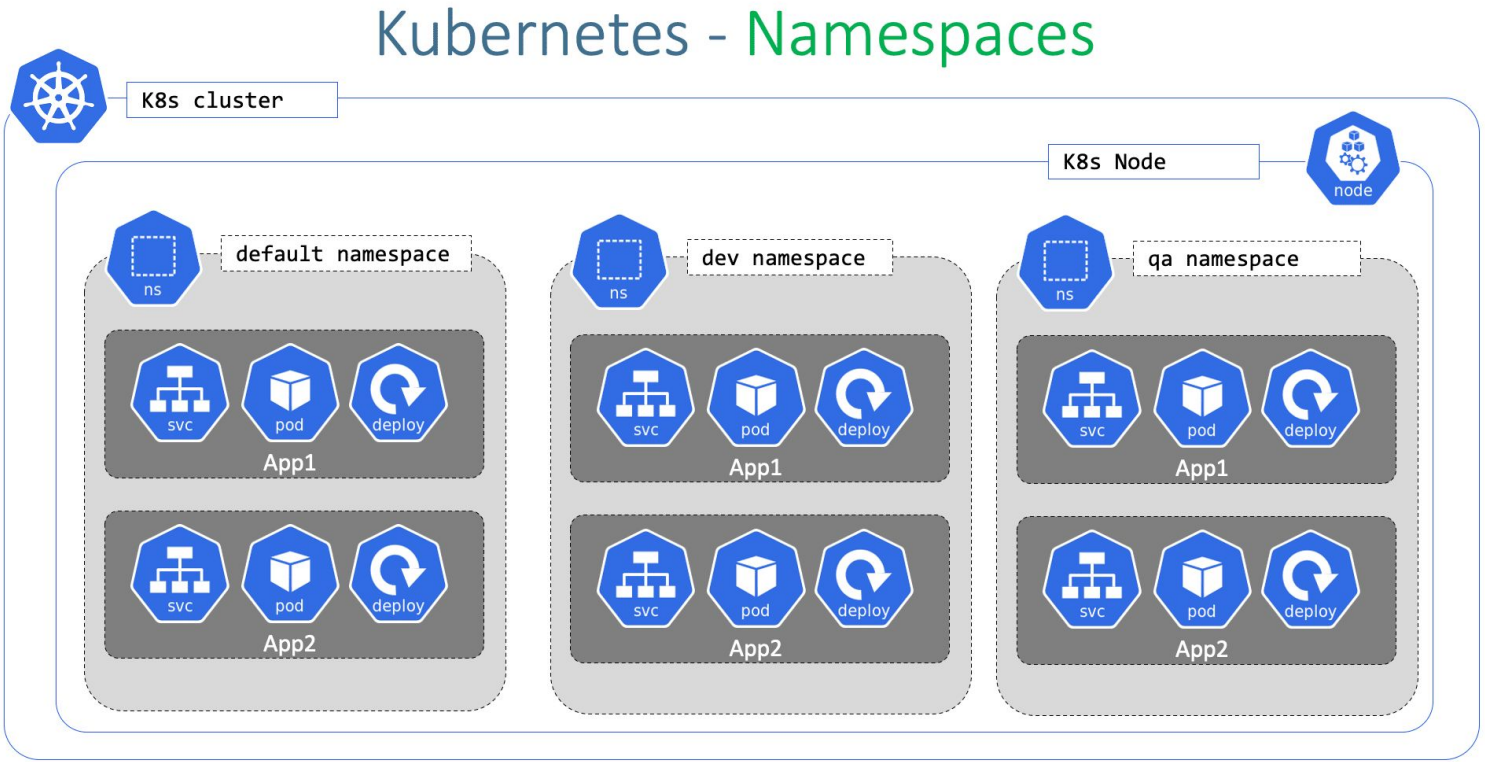
Services


app: A
pod
10.10.10.5
Selector
app=A
Service
svc
1.1.1.1
app: B
pod
10.10.10.4
deploy
Deployment
rs
ReplicaSet
pod
app: A
Label
10.10.10.1
pod
app: A
10.10.10.2
pod
app: A
10.10.10.3

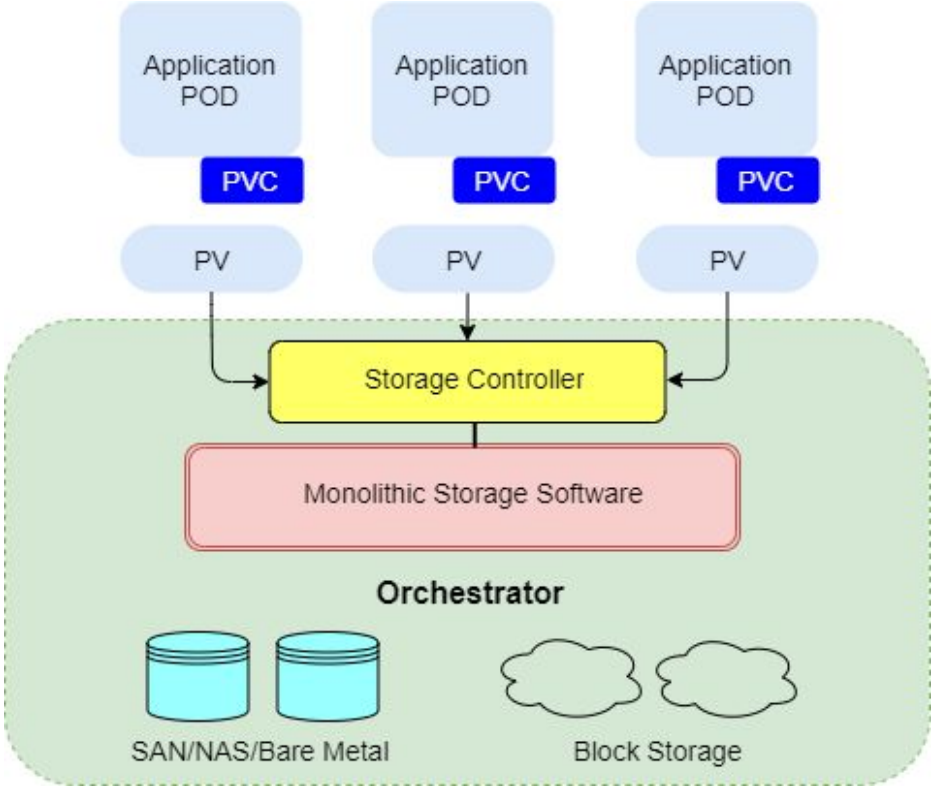
Ingresses


Simple Fanout
https://www
Traffic
INGRESS
services.example.com/service1
services.example.com/service2
SERVICE-1
Service
SERVICE-2
Service
POD
POD
POD
POD
POD
POD
BANZAICLOUD

Namespace


Kubernetes - Namespaces
K8s cluster
K8s Node
node
ns
default namespace
svc
pod
deploy
App1
App2
ns
dev namespace
svc
pod
deploy
App1
App2
ns
qa namespace
svc
pod
deploy
App1
App2

Volume

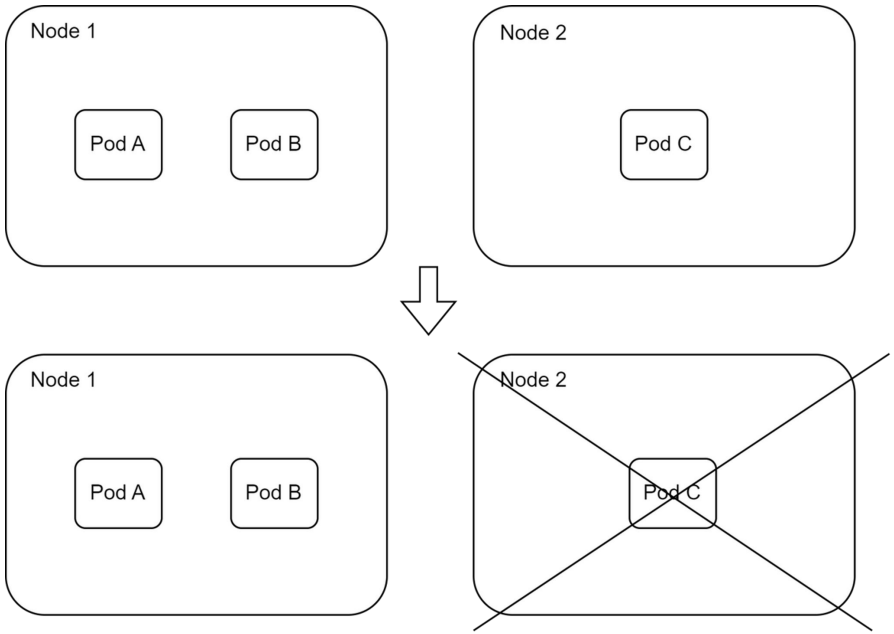

Application POD
Application POD
Application POD
PVC
PVC
PVC
PV
PV
PV
Storage Controller
Monolithic Storage Software
Orchestrator
SAN/NAS/Bare Metal
Block Storage

But,
What is Kubernetes?
Why Kubernetes?

“Container Orchestrator”

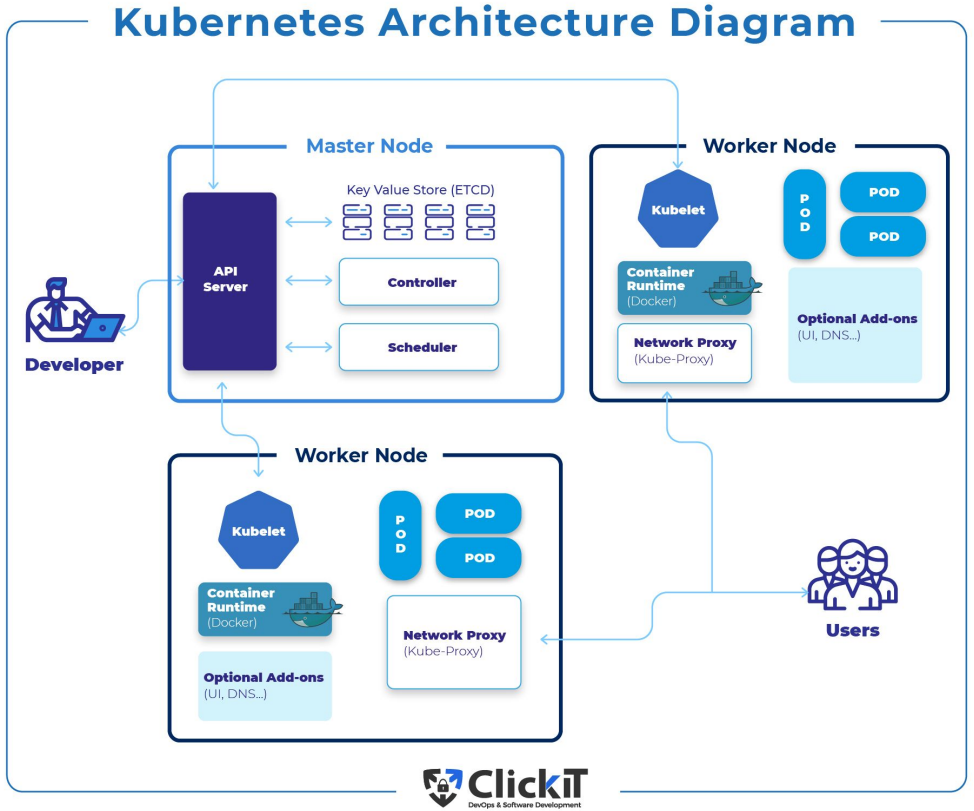
Kubernetes ensure all containers/pods are running -> **Guaranteed application availability.**

Top Company Use : Google, Udemy, Gojek, etc.



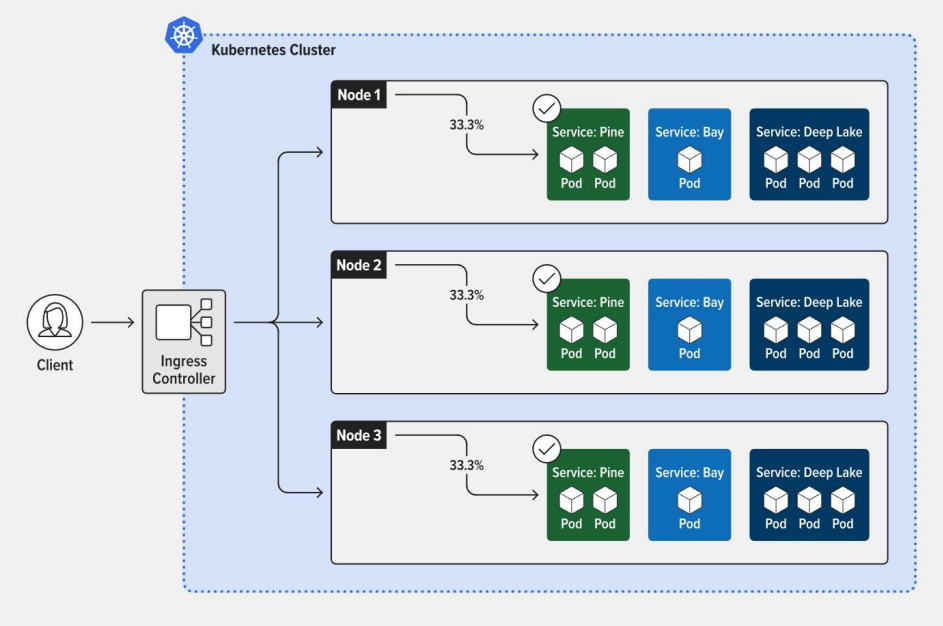
How Kubernetes Work?

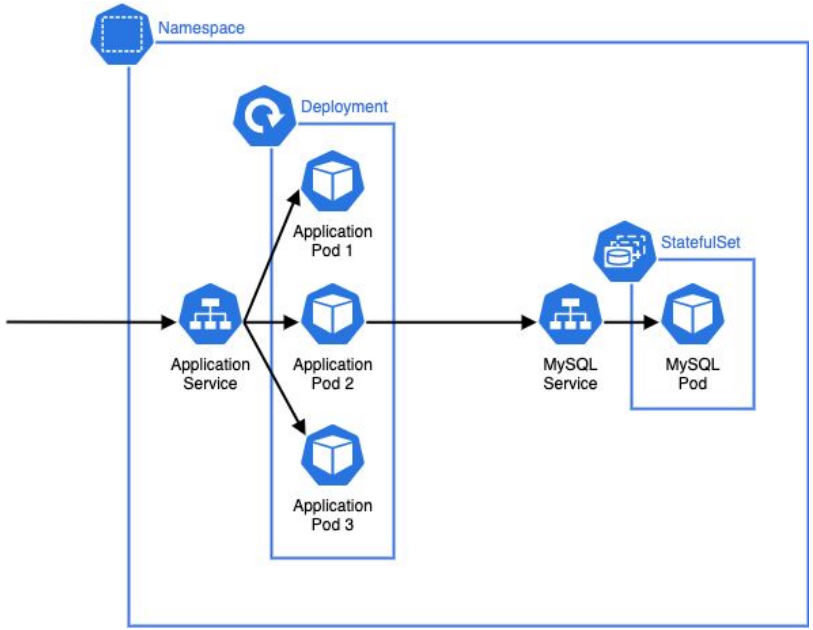
The components that exist in Kubernetes make it possible to be responsible for the availability of existing services.



Zoom In Architecture

***Usecase** : An application that runs on Kubernetes and is accessed by the outside world (public).*



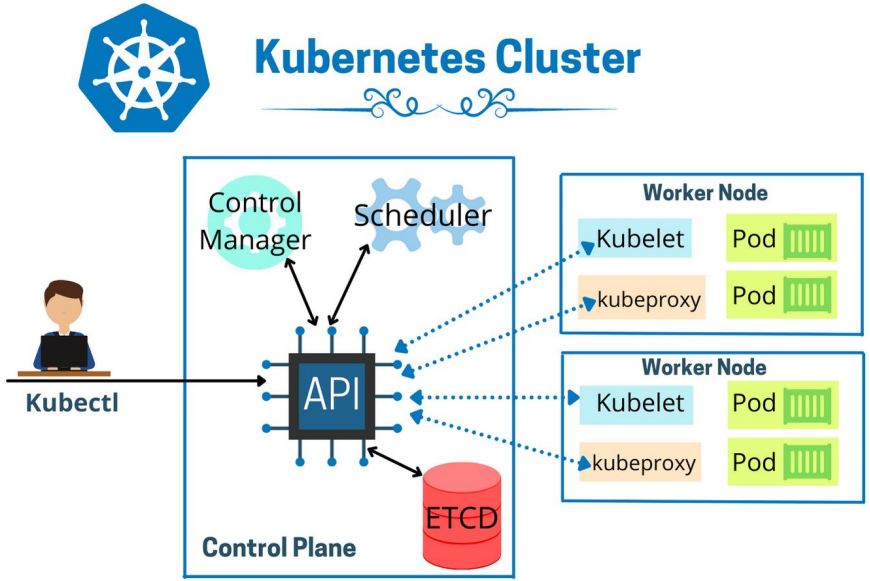


But how to Controlling & Provision Apps on K8s ? (1)

Kubectl & Manifest

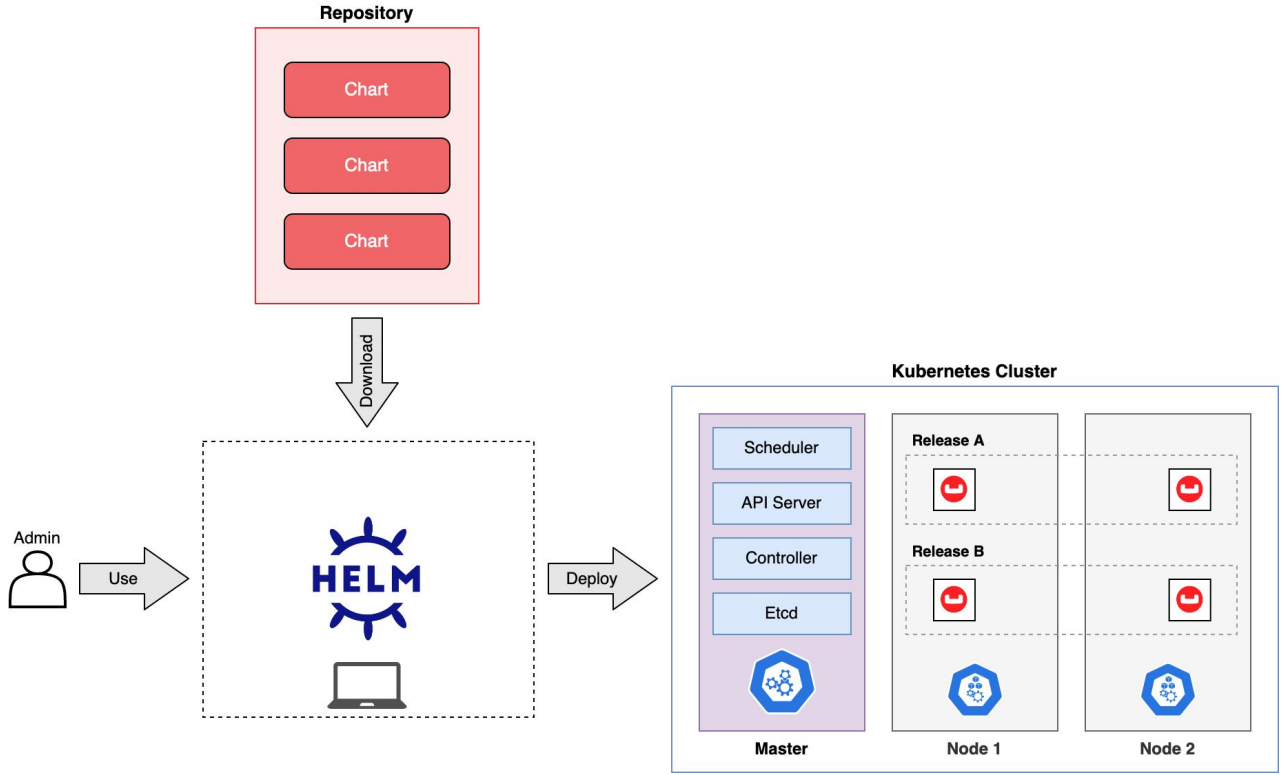
object1.yaml

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: hello-world
        image: hello-world:latest
        ports:
        - containerPort: 80
```



But how to Controlling & Provision Apps on K8s ? (2)

Helm for Package Manager



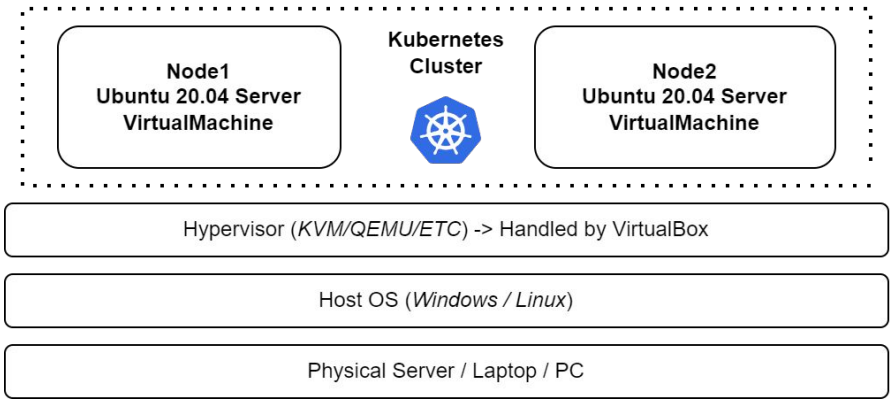
More :
https://artifacthub.io/
https://bitnami.com/stacks/helm

Demo

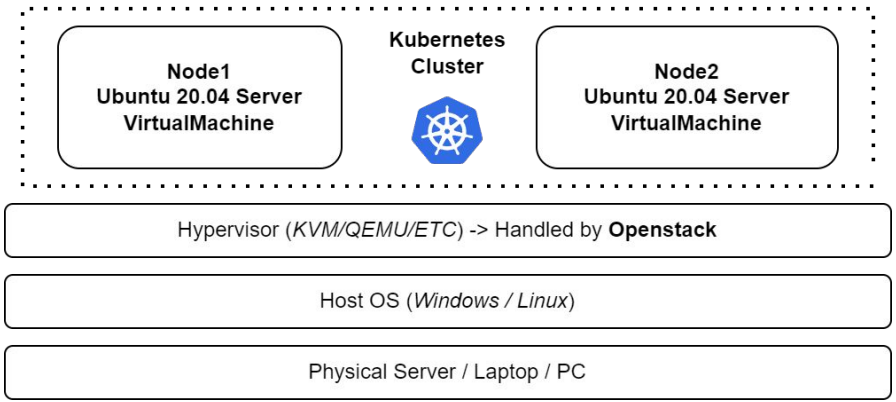
Agenda Demo Flow

- Install Prerequisites Tools (VirtualBox, Terminal, Lens)
- *Install Linux VM* (**Optional**)
- Install Kubernetes Cluster
- Deploy Some Apps to Kubernetes Cluster
- Test Failure
- Use Case Overview of K8s
- Done

Actually, We Have 3 Env



Local (VirtualBox)



Cloud (Openstack & Proxmox for Backup)

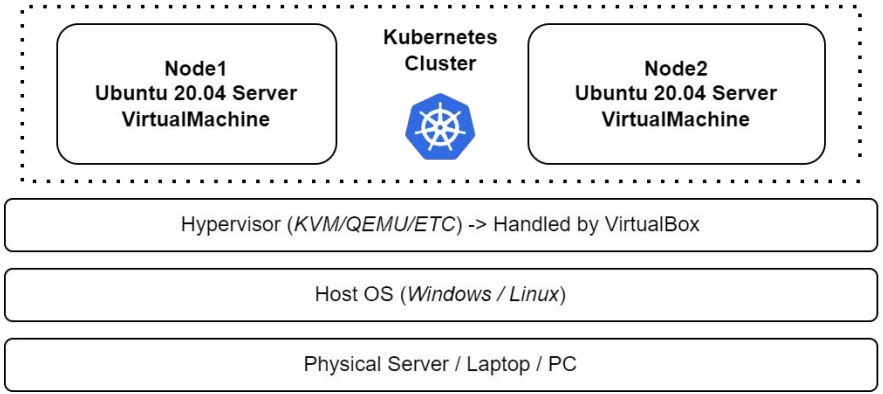
Local installation is not mandatory, please adjust. If you just want to watch, go ahead.

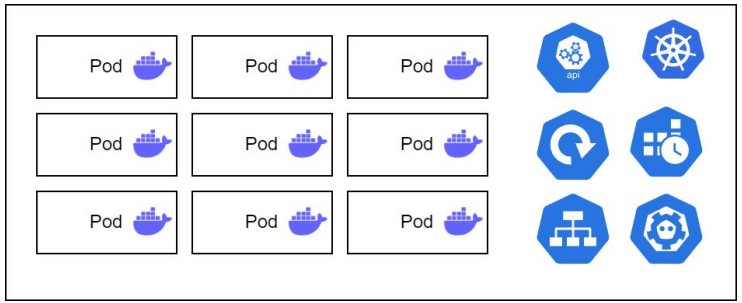
Prerequisites

- Internet Connection
- If you want to try **Locally** you **must** have :
  - VirtualBox -> https://www.virtualbox.org/wiki/Downloads **(Self)**
  - Linux Ubuntu Server 20.04 LTS -> https://cloud-images.ubuntu.com/focal/current/ **(Self)**
  - **OR** OVA Template -> https://drive.google.com/drive/folders/1jVBkjdH3-IZia-CXvdgsPXEd7ADaYz6d?usp=sharing **(Self)**

- Terminal (Termius // Command Prompt // Putty) -> https://www.putty.org/ **(Self)**
- Lens (Optional) -> https://k8slens.dev/ **(Self)**
- NFS Server **(Optionally Self, Handson Together)**
- Docker & Containerd **(Optionally Self, Handson Together)**
- Kubernetes Cluster **(Optionally Self, Handson Together)**
  - Kubectl
    - Kubectl on Windows -> https://gist.github.com/gilangvperdana/875a267286cb3bef503383add223e67e
    - Kubectl on Linux -> https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/
  - Helm

LAB Local Goals Specification & Topology

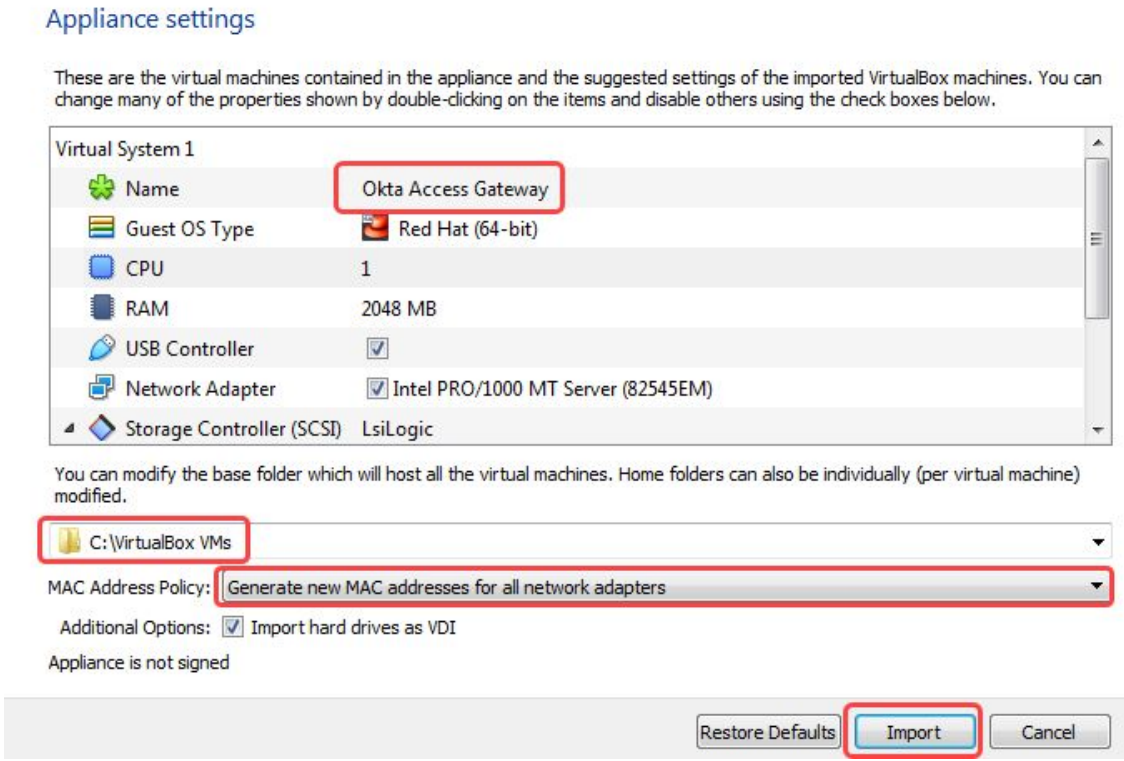




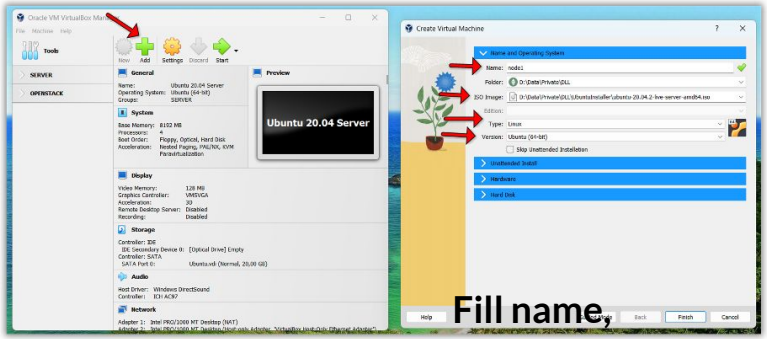
Node Name	Spesification	OS	Role
node1	2VCPU & 2GB RAM	Ubuntu 20.04 LTS	Master, Worker & NFS Server
node2	2VCPU & 2GB RAM	Ubuntu 20.04 LTS	Worker
Container Runtime		**Containerd**	
Virtualization Platform		**VirtualBox**	

Import OVA to VirtualBox

https://help.okta.com/oag/en-us/Content/Topics/Access-Gateway/deploy-ovb.htm



Create Virtual Machine on VirtualBox (Skip if you use Cloud env)

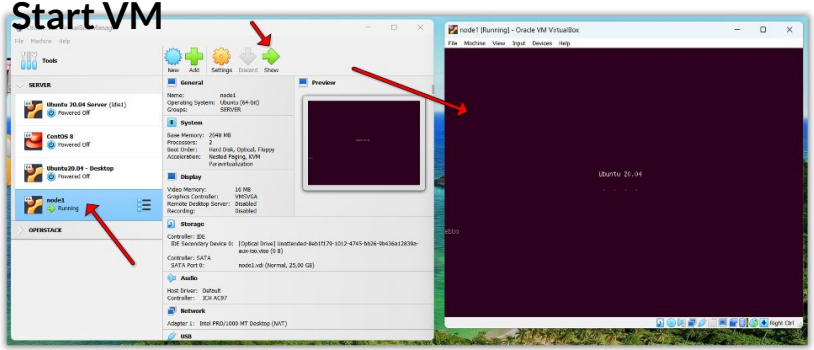


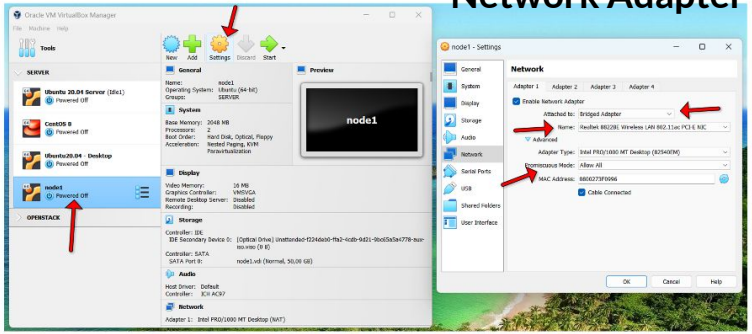
Fill name,
Image Ubuntu
Path

2GB RAM
2 CPU

Change to Your
Network Adapter

Start VM

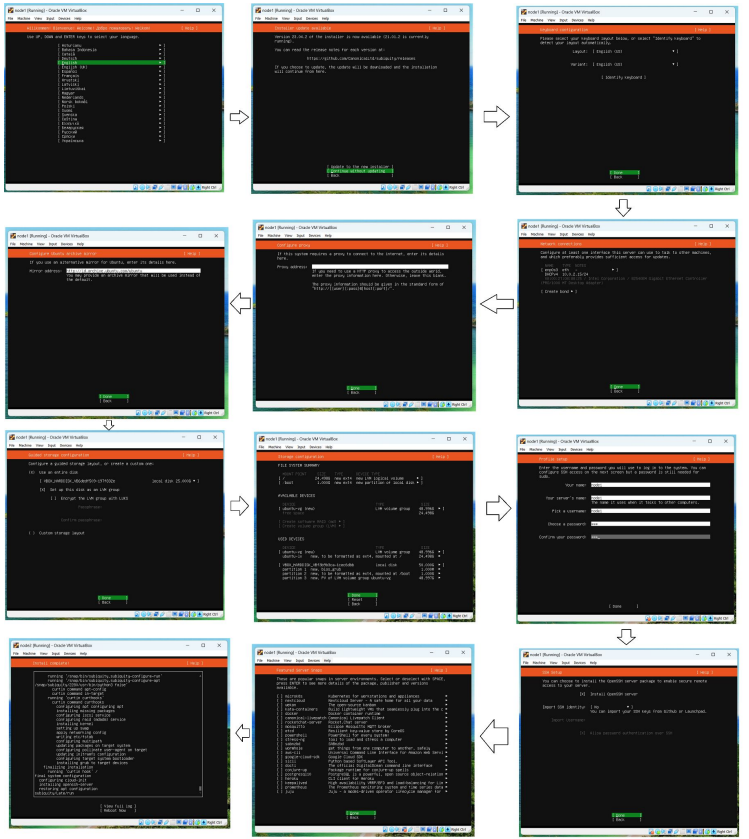




Source : https://gist.github.com/gilangvperdana/bb5ba6b9f14a01fce3e85ef337afb457/raw/7c1cd16b35bcfb92cf793a124aeeca973829df38/VBOXInstall.png

Linux Installation

(Skip if you use Cloud env)



Source : https://gist.githubusercontent.com/gilangvperdana/bb5ba6b9f14a01fce3e85ef337afb457/raw/7c1cd16b35bcfb92cf793a124aeeca973829df38/LinuxInstall.png

Deploy Kubernetes Cluster (Kubeadm)

(Skip if you use Cloud env)

After Import OVA or from ISO,
We will deploy Kubernetes Cluster with Kubeadm as Deployer

More on :
https://gist.github.com/gilangvperdana/7251bcc3c923dc55856366da68875952

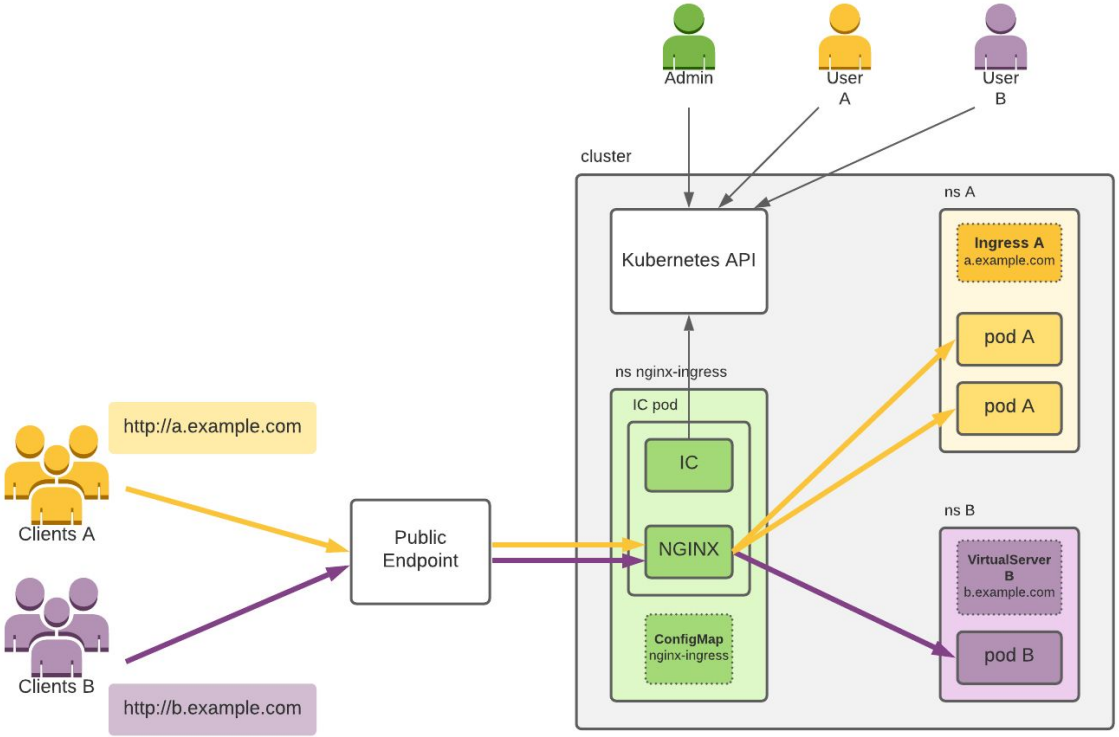
Actually there are many other Kubernetes deployers, such as Minikube, Kubespray, etc.

Cloud Native Stack on K8s Needed for this Session

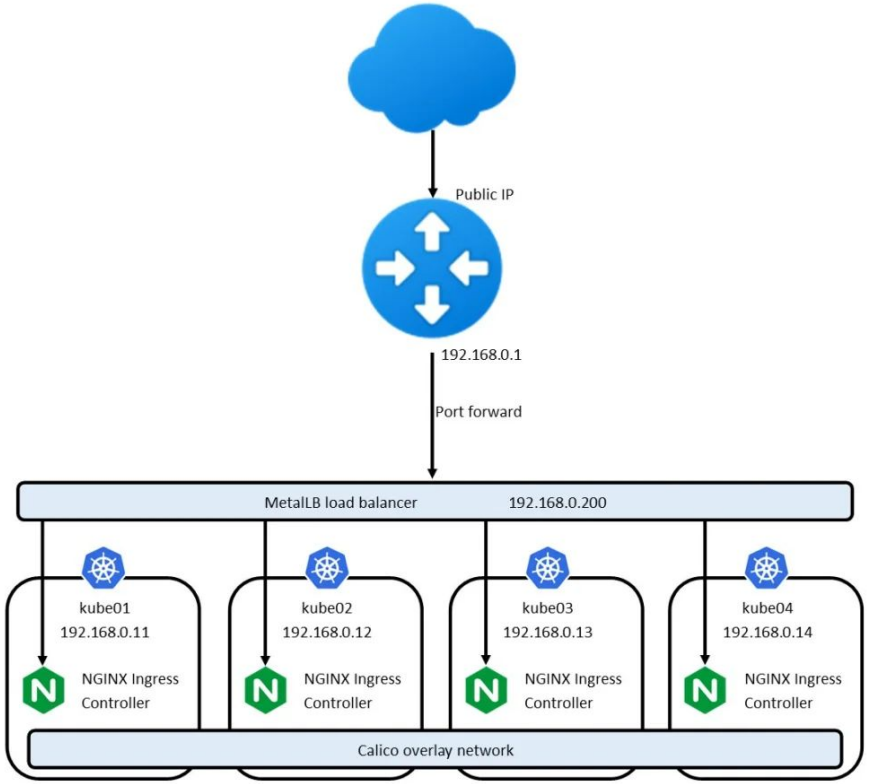
Describe	Stack
Ingress Gateway (API Gateway)	Nginx Ingress
Baremetal Load Balancer	Metallb
Storage Volume Provisioner	NFS Subdir External Provisioner

More: https://landscape.cncf.io/

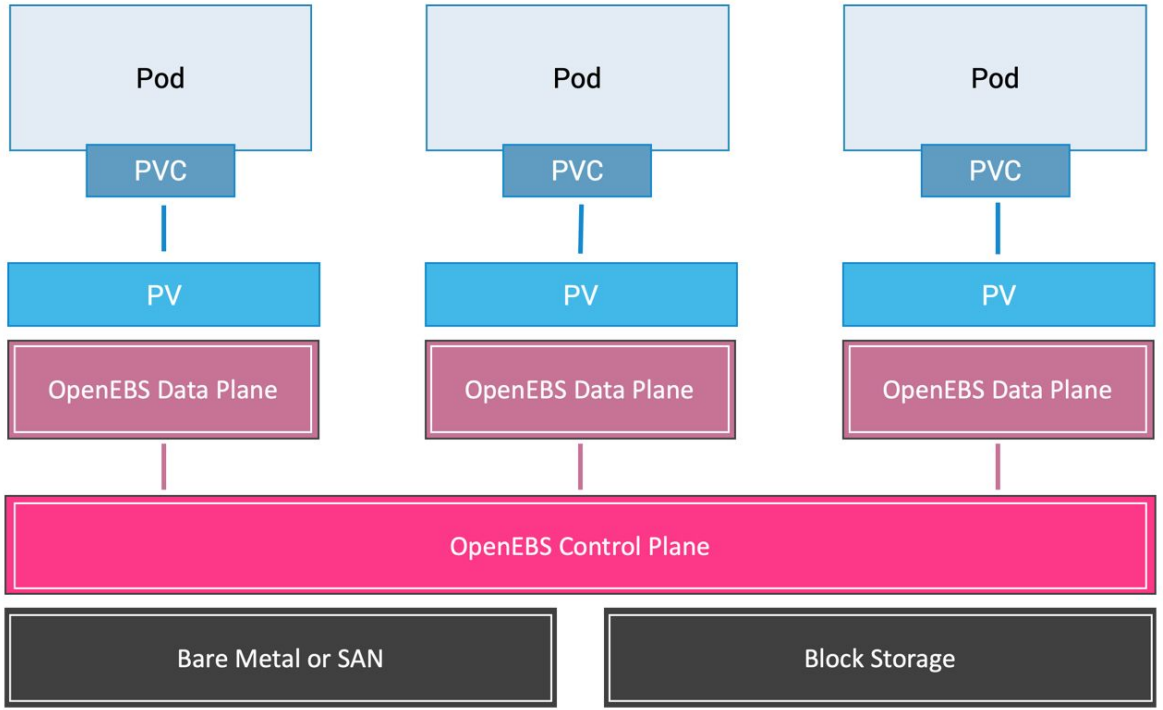
API Gateway (NGINX Ingress)



Baremetal Load Balancer (Metallb)



Storage Provisioner (OpenEBS)



Deploy Support Platform on K8s

(Skip if you use Cloud env)

Before Continue make sure we have a Helm Package as Package Manager

$ curl https://gist.githubusercontent.com/gilangvperdana/4413897e4a2a92ada6a330bb8b111ea1/raw/15c0e0db7e9a64a539c0cb024d94fa39965ce674/helm.sh | bash

- Metallb

```
$ kubectl apply -f
https://raw.githubusercontent.com/metallb/metallb/v0.11.0/manifests/namespace.yaml &&
kubectl apply -f
https://raw.githubusercontent.com/metallb/metallb/v0.11.0/manifests/metallb.yaml
$ kubectl apply -f - <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.80.2-192.168.80.254 #Define to your network pool.
EOF
```

- Nginx Ingress

```
$ helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
$ helm repo update
$ helm install myingress ingress-nginx/ingress-nginx -n ingress-nginx --create-namespace
```

- NFS Server (Node1)

```
$ apt update -y
$ sudo apt install nfs-kernel-server
$ sudo mkdir -p /mnt/nfs_share
$ sudo chown -R nobody:nogroup /mnt/nfs_share/
$ sudo chmod 777 /mnt/nfs_share/
$ sudo nano /etc/exports
/mnt/nfs_share 0.0.0.0/0 (rw,sync,no_subtree_check)
$ sudo exportfs -a
$ sudo systemctl restart nfs-kernel-server
```

- Mount NFS on K8s

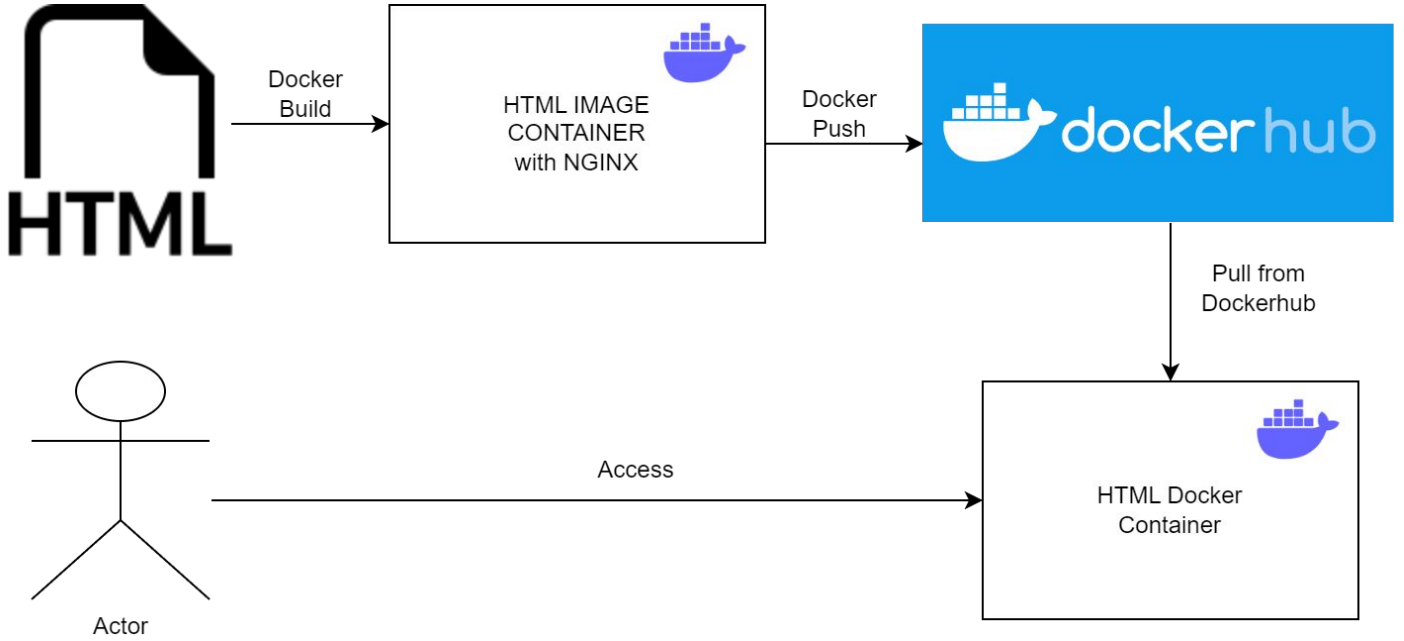
```
$ helm repo add nfs-subdir-external-provisioner https://kubernetes-sigs.github.io/nfs-subdir-external-provisioner/
$ helm install nfs-cluster-provisioner nfs-subdir-external-provisioner/nfs-subdir-external-provisioner \
    --set nfs.server=172.16.1.10 \
    --set nfs.path=/mnt/nfs_share \
    --set storageClass.name=nfs-client \
    --set storageClass.provisionerName=cluster.local/nfs-cluster-provisioner
$ kubectl patch storageclass nfs-client -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

Demo Case

- Build Docker Image (**Pra Scenario**)
- Deployment Docker Image to K8s **without** Volume (**Scenario 1**)
- Deployment Docker Image to K8s **with** Volume (**Scenario 2**)

Pra Scenario

- Pra Scenario -> Create self docker image container then push to Dockerhub



Pra Scenario

Clone Repo
git clone https://github.com/gilangvperdana/k8s-sharingsession.git
cd k8s-sharingsession && cd skenario1

Build Image
cd v1
docker build -t yourDockerHubusername/yourRepository:tag .
docker push yourDockerHubusername/yourRepository:tag
docker pull yourDockerHubusername/yourRepository:tag
docker run -d -p 80:80 yourDockerHubusername/yourRepository:tag

Access
YourIP:80

Repeat to V2

Conclusion Pra Scenario

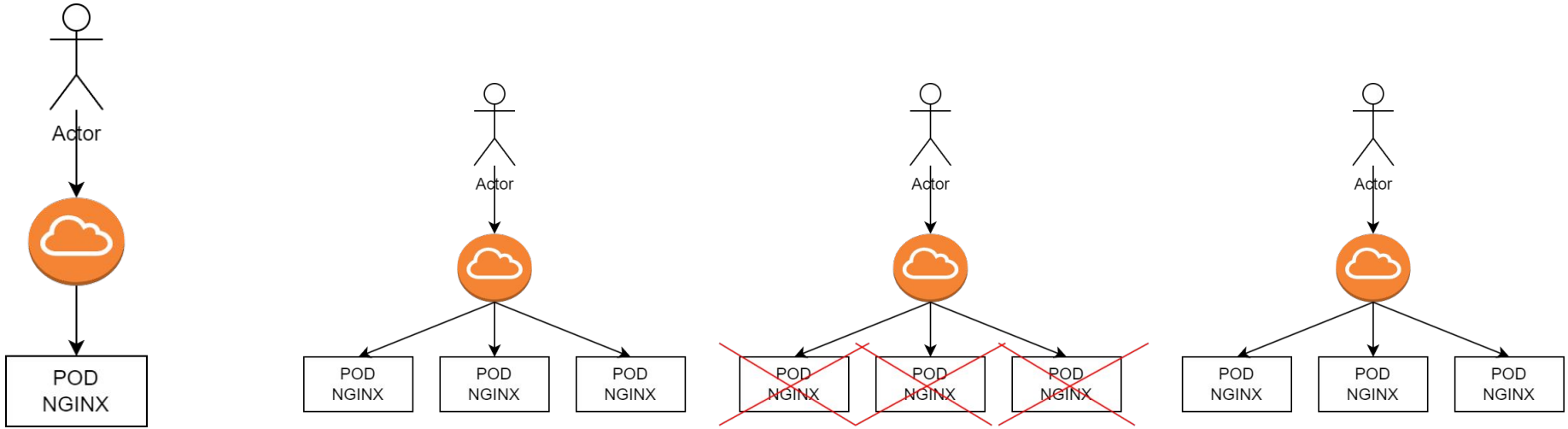
From pra-scenario we can conclude :

- Containerized applications can be stored on the hub (Dockerhub) -> **Flexibility to run around the world.**
- Containerized applications can run on any machine -> **Guaranteed Support.**

*from there, this is the approach why the digital world is now full of containerization, namely the **convenience** and **flexibility** that makes it easy for the process of **replicating**, **scaling**, **etc**.*

Use Case (Handson 1)

- Scenario **1**-> Create apps **without** volume then test failure scenario



Scenario 1

Clone Repo
git clone
https://github.com/gilangvperdana/k8s-sharingsession.git
cd k8s-sharingsession && cd skenarioA

Change Image to Yours
nano deployment.yaml
nano deployment2.yaml

Deploy
kubectl apply -f deployment.yaml
kubectl apply -f deployment2.yaml

Verify
kubectl get pod -o wide

Access

Test Failure
kubectl delete pod nginx-deployment-***

Pod will be running again automatically

Change some index.html
kubectl exec -it nginx-deployment-*** bash
apt update -y && apt install -y nano
nano /usr/share/nginx/html/index.html

Delete Pod
kubectl delete pod nginx-deployment-***

Correct behaviour -> Data will rollback.

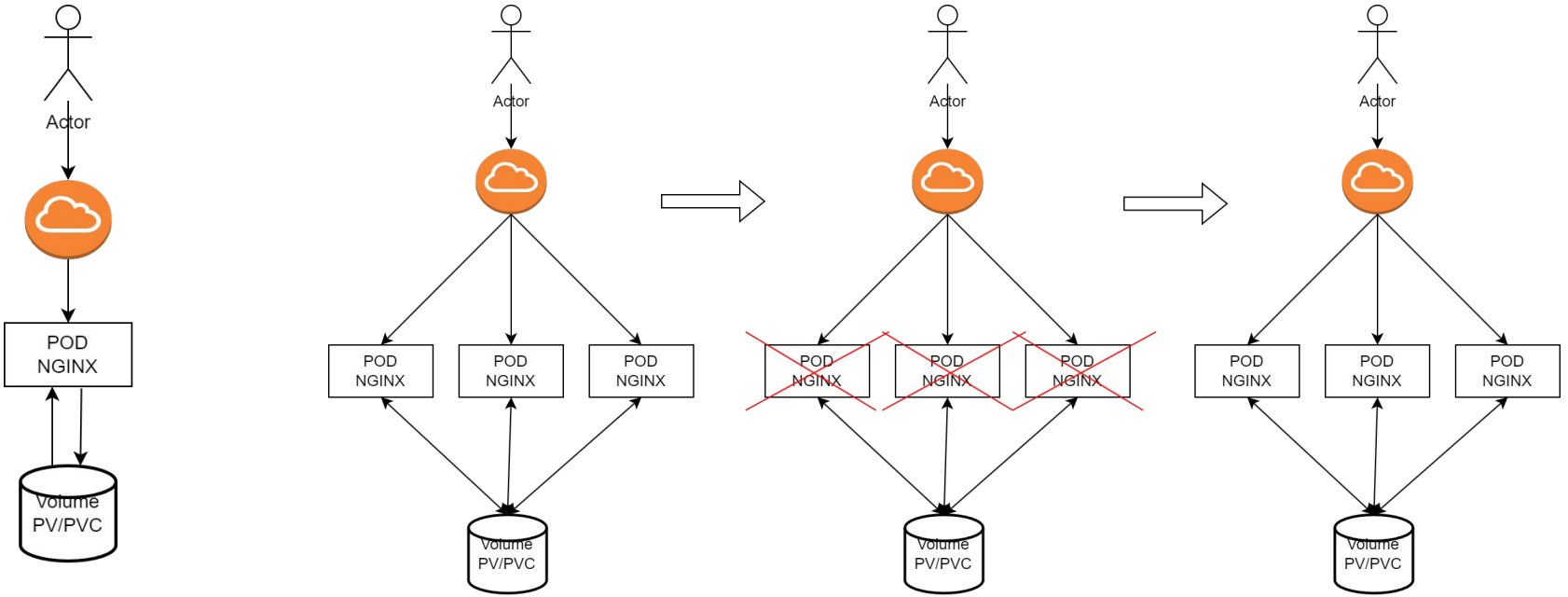
Conclusion Scenario 1

From scenario 1 we can conclude :

- Pod automatically start -> **Reliability**
- Kubernetes Ingresses automatically load balance to all pod -> **Reliability**
- Pod without Volume will not persistence data

Use Case (Handson 2)

- Scenario **2** -> Create apps with volume then test failure scenario



Scenario 2

Clone Repo
git clone
https://github.com/gilangvperdana/k8s-sharingsession.git
cd k8s-sharingsession && cd skenarioB

Deploy
kubectl apply -f deployment.yaml

Verify
kubectl get pod -o wide

Access

Verify on NFS
cd /mnt/nfs_share/

Test Failure
kubectl delete pod nginx-deployment-***

Pod will be running again automatically

Change some index.html
kubectl exec -it nginx-deployment-*** bash
apt update -y && apt install -y nano
nano /usr/share/nginx/html/index.html

Delete Pod
kubectl delete pod nginx-deployment-***

Correct behaviour -> New data is not lost.

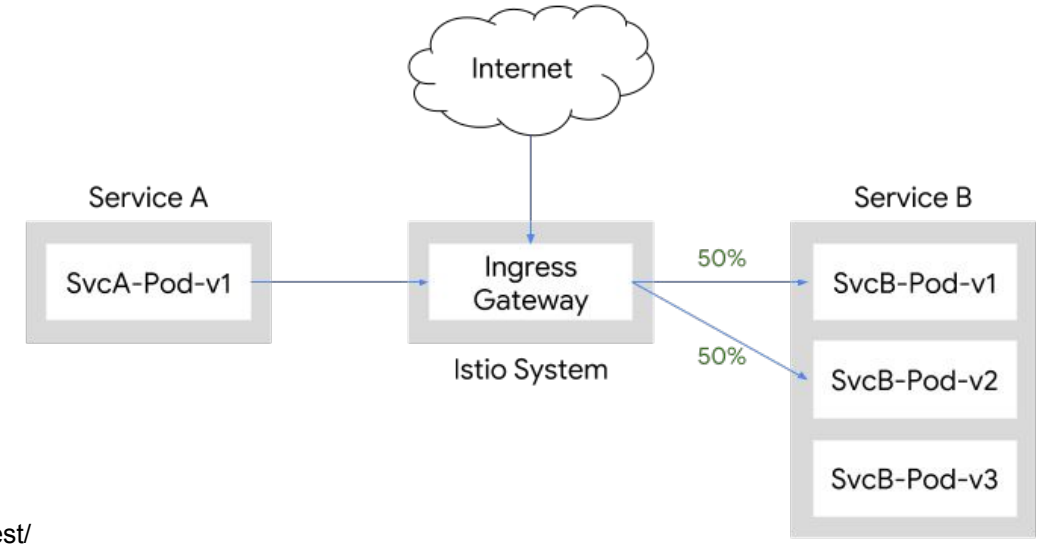
Conclusion Scenario 2

From scenario 2 we can conclude :

- Pod automatically start -> **Reliability**
- Kubernetes Ingresses automatically load balance to all pod -> **Reliability**
- Pod **with** Volume will save data
- If 1 pod from 3 pod data change, all pod will be sync data -> **keep all data in sync**
- One volume on **RWX mode** can use with multiple pods -> **keep all data in sync**

Another Use Case (Speaker Explanation)

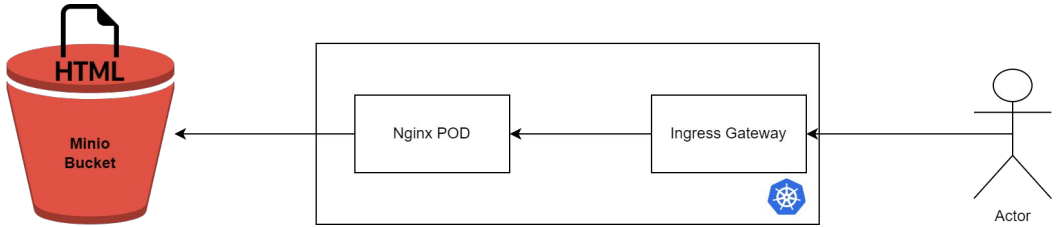
- Overview traffic splitting
  - https://github.com/gilangvperdana/k8s-sharingsession/blob/master/skenarioA/deployment3-istio-splitting.yaml



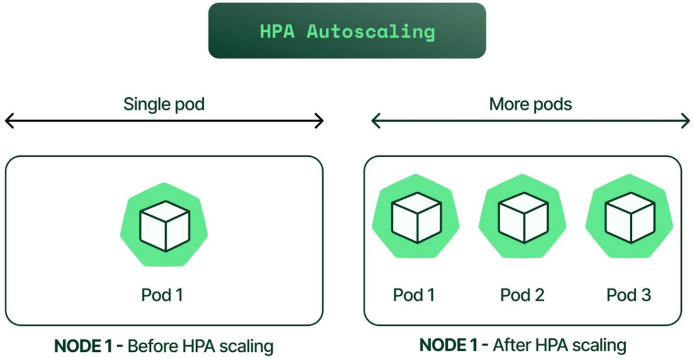
Source : https://istio.io/latest/

Another Use Case (Speaker Explanation)

- Overview S3 hosted html
  - https://github.com/gilangvperdana/k8s-sharingsession/blob/master/skenarioB/deployment2-minio.yaml

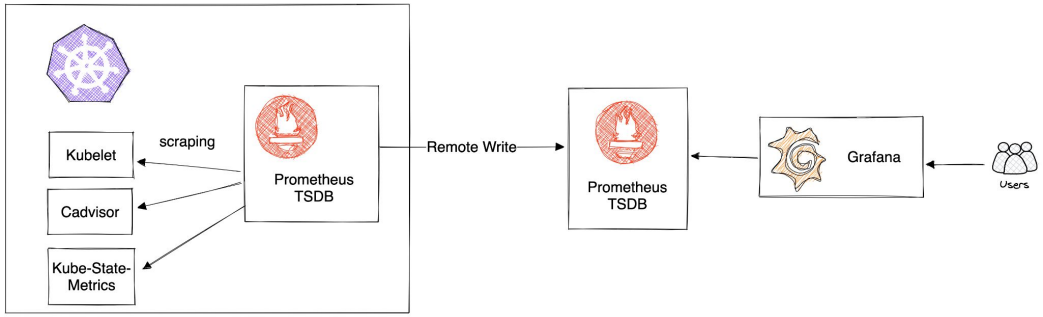


- Overview auto scaling when high traffic
  - https://github.com/gilangvperdana/K8s-PlayGround/blob/master/dll/testloadHPA.md
  - https://github.com/gilangvperdana/k8s-sharingsession/blob/master/skenarioB/deployment3-hpa.yaml

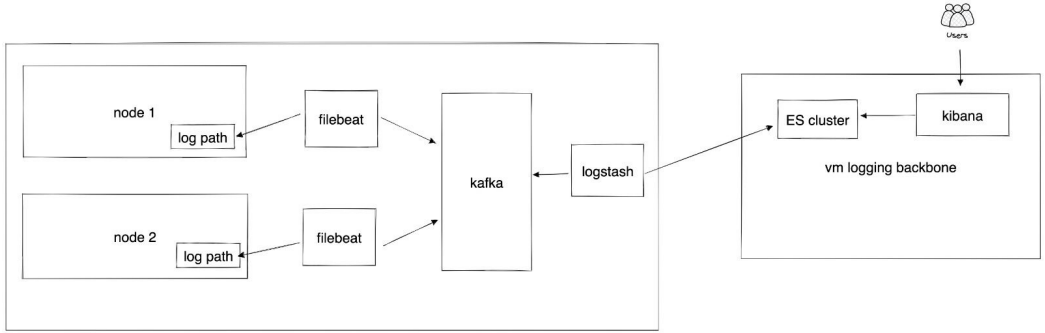


Another Use Case (Speaker Explanation)

- Overview Monitoring Cluster, Pod

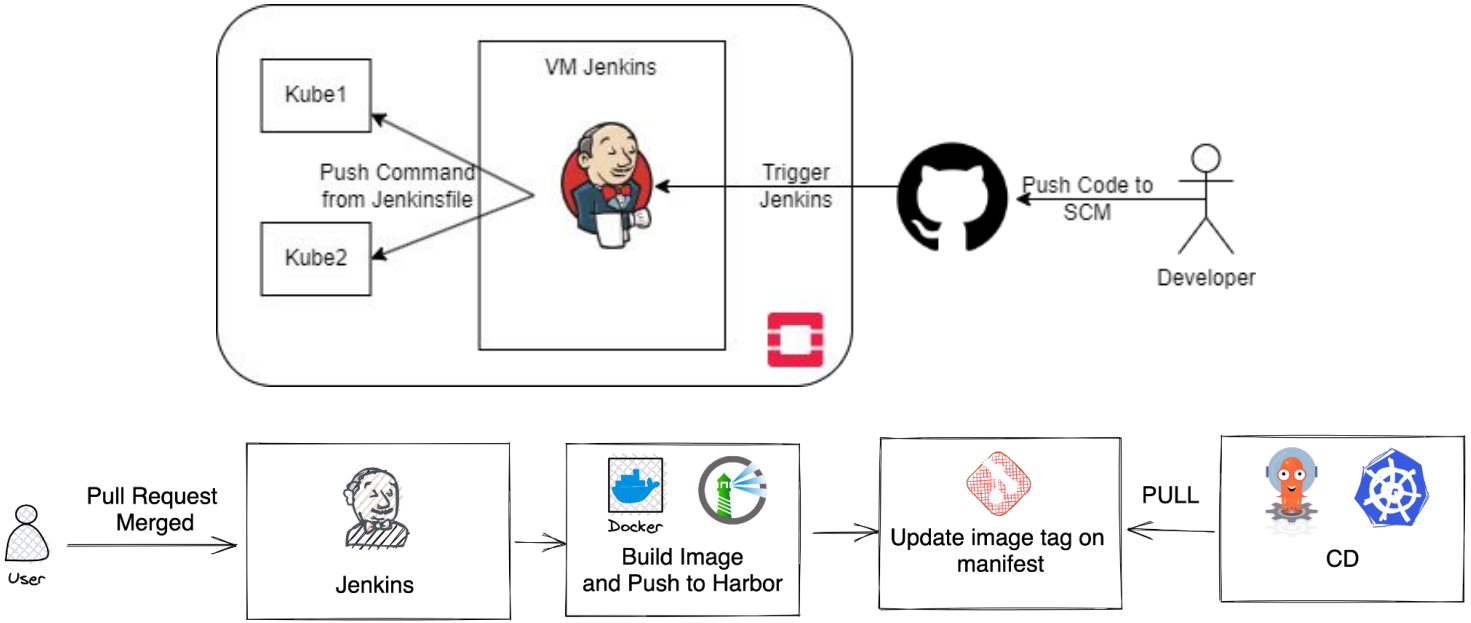


- Overview Logging



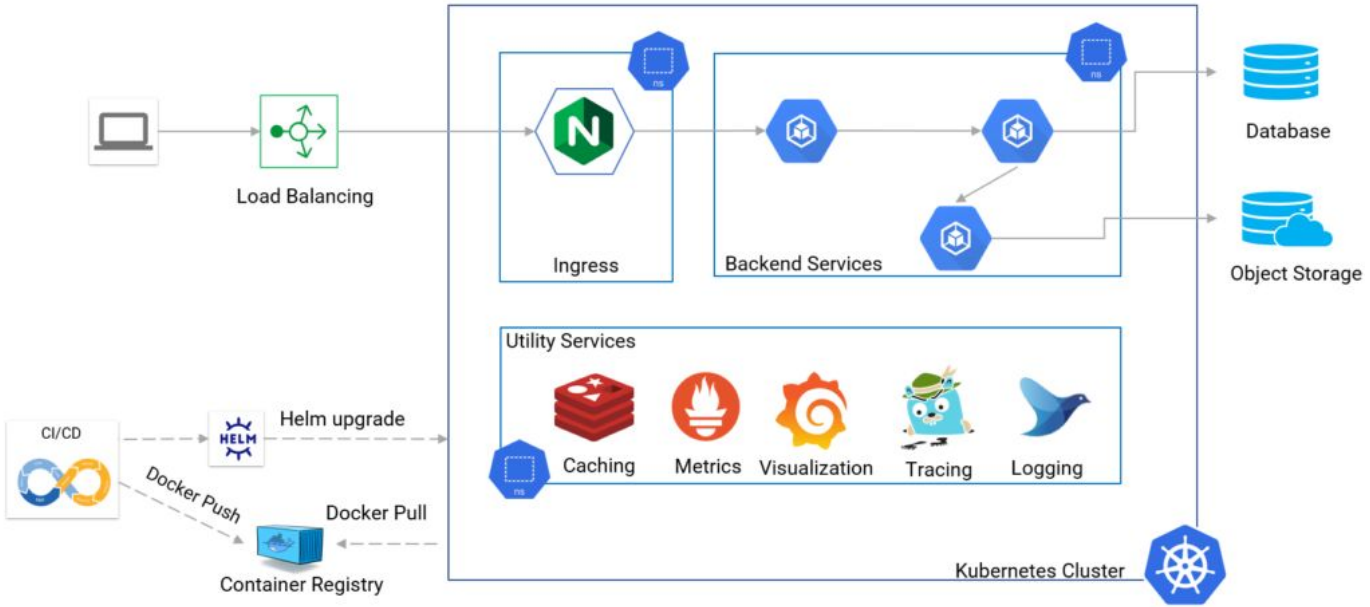
Another Use Case (Speaker Explanation)

CICD



Another Use Case (Speaker Explanation)

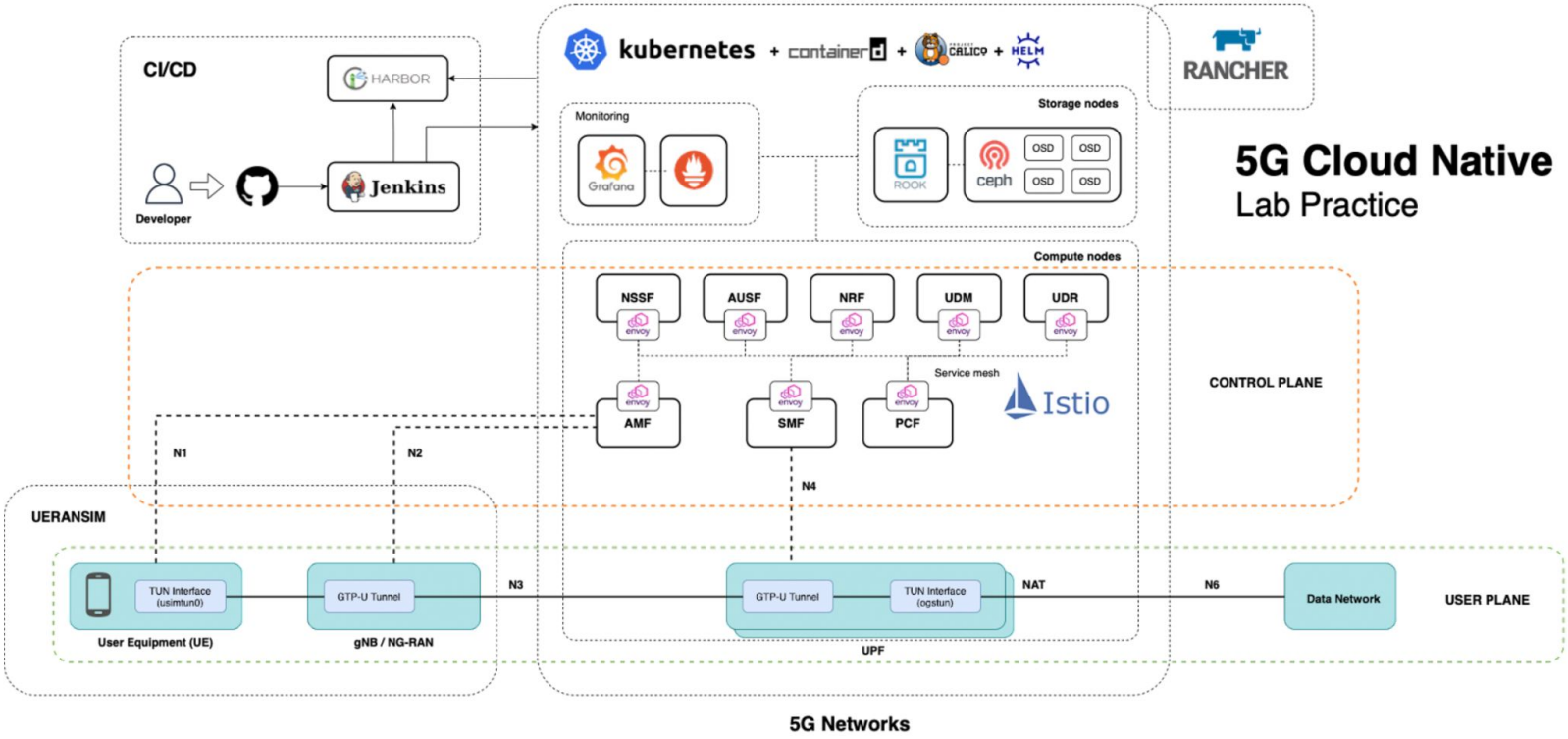
- Microservices (Today Web Apps Architecture)



QnA

***Mini Conclusion** : With all the components owned by Kubernetes, Kubernetes offers a better availability solution for today's digital services.*

Open5GS



***Source** : https://luthfi.dev/posts/5g-cloud-native-simulation-with-open5gs/*

Thanks

Sorry for the mistake during the presentation earlier, greetings to all of you

Adaptive Network
Laboratory