Kubernetes

cluster

single node

- development
- trial
- testing

$\Downarrow$

Minikube

Learning

multi-node

single master
manager

testing

multi-master
#### manager

production

What is Kubernetes ?

- Portable, extensible, open-source platform for managing containerized workloads and services
- Facilitates both declarative configuration and automation   *declarative — YAML*
- It has a large, rapidly growing ecosystem   *Extra tools → prometheus, istio*
- Kubernetes services, support, and tools are widely available
- The name Kubernetes originates from Greek, meaning helmsman or pilot   *→ K8S*
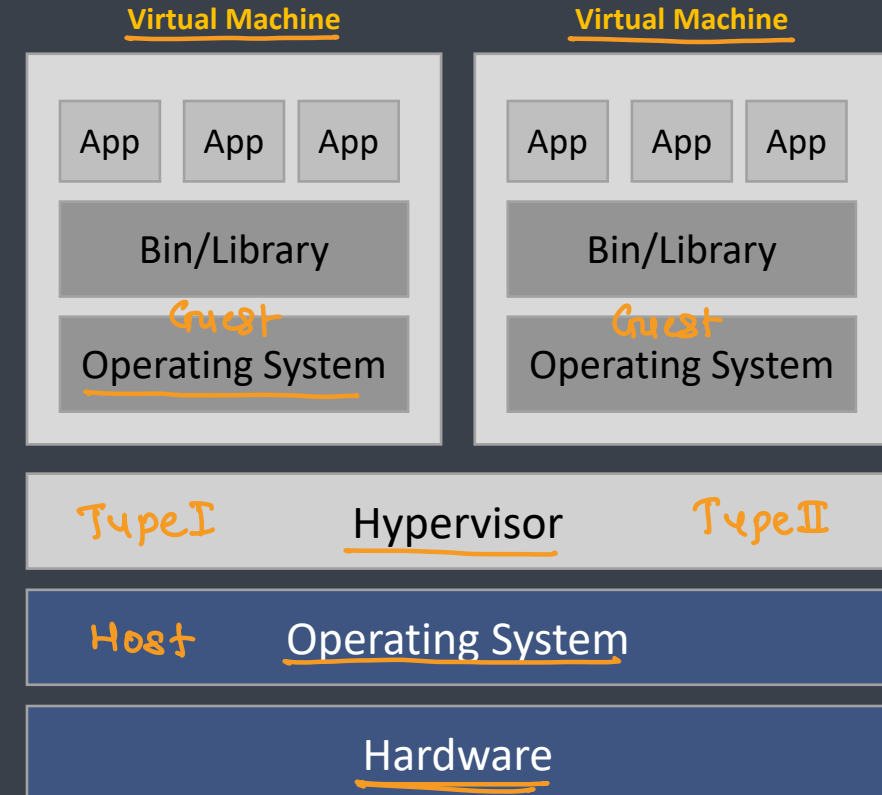- Google open-sourced the Kubernetes project in 2014

Traditional Deployment

→ using physical machines

- Early on, organizations ran applications on physical servers

- There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues

- For example, if multiple applications run on a physical server, there can be instances where one application would take up most of the resources, and as a result, the other applications would underperform

- A solution for this would be to run each application on a different physical server

- But this did not scale as resources were underutilized, and it was expensive for organizations to maintain many physical servers

App	App	App	App

Bin/Library

| Operating System |
|------------------|

Hardware
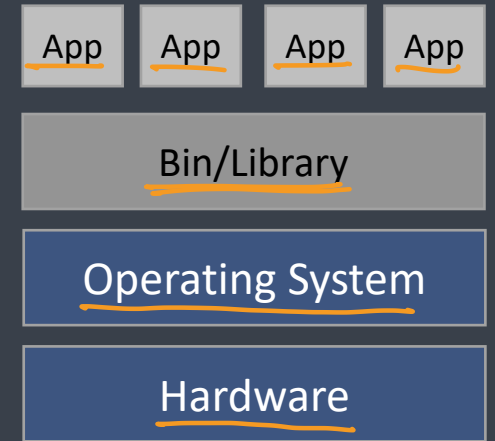
Virtualized Deployment

- It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU

- Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application

- Virtualization allows better utilization of resources in a physical server and allows better scalability because
  - an application can be added or updated easily
  - reduces hardware costs

- With virtualization you can present a set of physical resources as a cluster of disposable virtual machines

- Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware

Virtual Machine

App	App	App

Bin/Library

Guest Operating System

Virtual Machine

App	App	App

Bin/Library

Guest Operating System

Type I   Hypervisor   Type II

Host   Operating System

Hardware

Container deployment

- Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications

- Therefore, containers are considered lightweight

- Similar to a VM, a container has its own filesystem, CPU, memory, process space, and more

- As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions
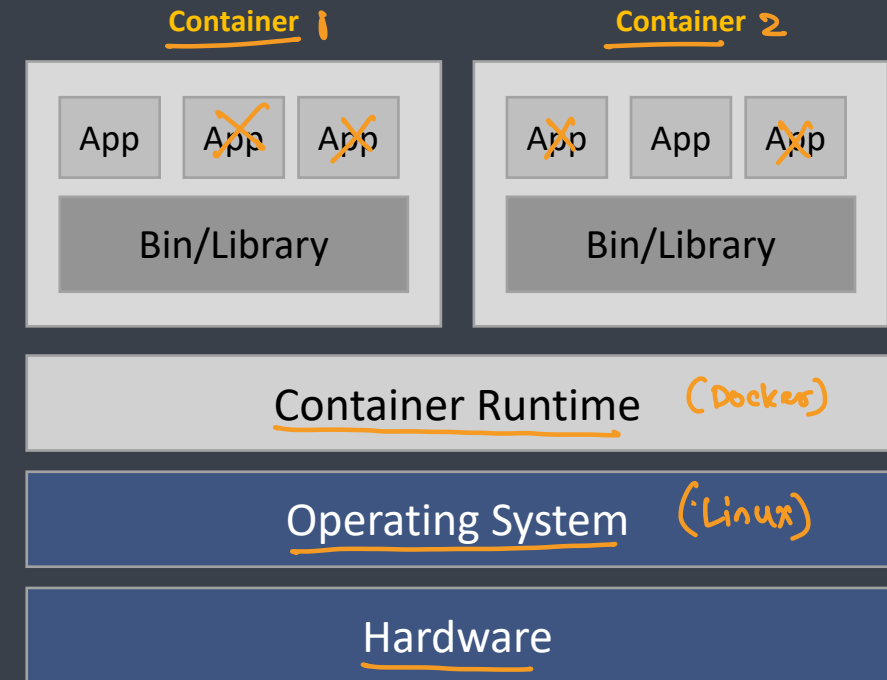
Container benefits

- Increased ease and efficiency of container image creation compared to VM image use
- Continuous development, integration, and deployment
- Dev and Ops separation of concerns
- Observability not only surfaces OS-level information and metrics, but also application health and other signals
- Cloud and OS distribution portability
- Application-centric management:
- Loosely coupled, distributed, elastic, liberated micro-services
- Resource isolation: predictable application performance

What Kubernetes provide?

- **Service discovery and load balancing**
  - Kubernetes can expose a container using the DNS name or using their own IP address
  - If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable

- **Storage orchestration**
  - Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more

- **Automated rollouts and rollbacks**
  - You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate

- **Automatic bin packing**
  - You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks
  - You tell Kubernetes how much CPU and memory (RAM) each container needs
  - Kubernetes can fit containers onto your nodes to make the best use of your resources

What Kubernetes provide?

- **Self-healing**
    - Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve

- **Secret and configuration management**
    - Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys
    - You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration

DevOps

1. Requirements → Jira, google drive

2. Code → IDEs, → vs, pycharm, webstorm
   SDKs, → Java SDK, Android SDK
   SCM → git, svn, cvs

3. Build → ant, maven, gradle

4. testing → selenium, jasmine, jest

5. release → Jenkins, Argo CD, GitlabCI (CI-CD)

6. deployment → VM (vmware, vbox), container (docker, podman)
   container orchestration (swarm, K8s, marathon)

7. configuration tools → ansible, puppet, chef

8. monitoring tool → nagios, data Dog

What Kubernetes is not

- Does not limit the types of applications supported → service, job, frontend etc
- Does not deploy source code and does not build your application
- Does not provide application-level services as built-in services
- Does not dictate logging, monitoring, or alerting solutions → prometheus,
- Does not provide nor mandate a configuration language/system
- Does not provide nor adopt any comprehensive machine configuration, maintenance, management, or self-healing systems

Requirement → Linux

Raft

Cluster

Leader

Master 1

Master 2

Worker1

Worker 2

Worker3
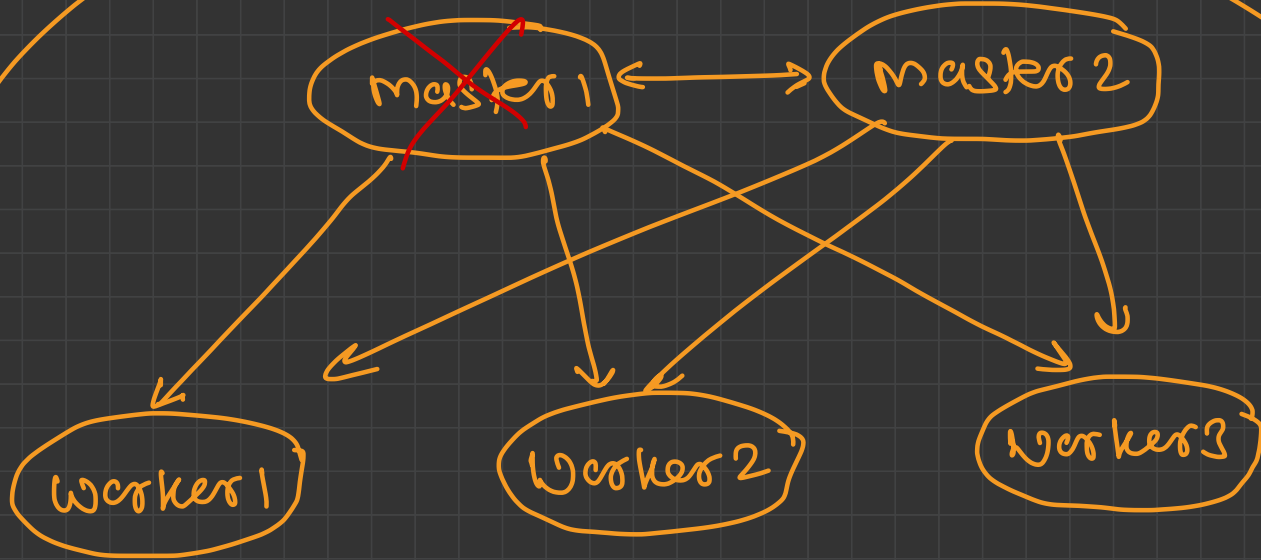
Kubernetes Cluster

node = machine

- When you deploy Kubernetes, you get a cluster.

- A cluster is a set of machines (nodes), that run containerized applications managed by Kubernetes

- A cluster has at least one worker node and at least one master node

- The worker node(s) host the pods that are the components of the application

- The master node(s) manages the worker nodes and the pods in the cluster

- Multiple master nodes are used to provide a cluster with failover and high availability

Master

service

worker

worker
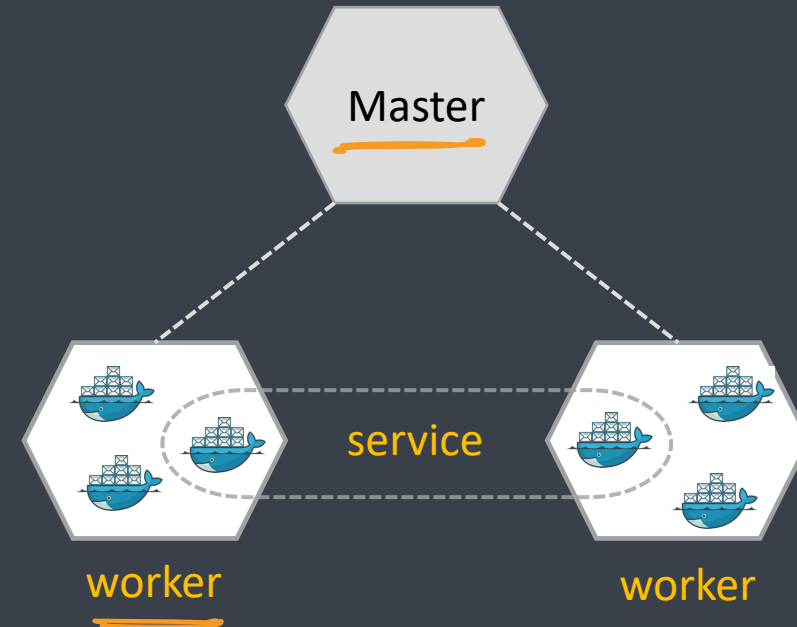
Kubernetes Components

Master

- kube-apiserver
- etcd
- kube-scheduler
- kube-controller-manager
- cloud-controller-manager

Node

- kubelet
- kube-proxy
- Container Runtime

REST

Kubectl

cluster

master

Kube. apiserver

Kube-scheduler

etcd | Kube-controller

Kube - proxy

Kubelet

Container Runtime

O.S.

worker

Kube - proxy

Kubelet

container runtime

O.S.

Kube - proxy

Kubelet

container runtime

O.S.
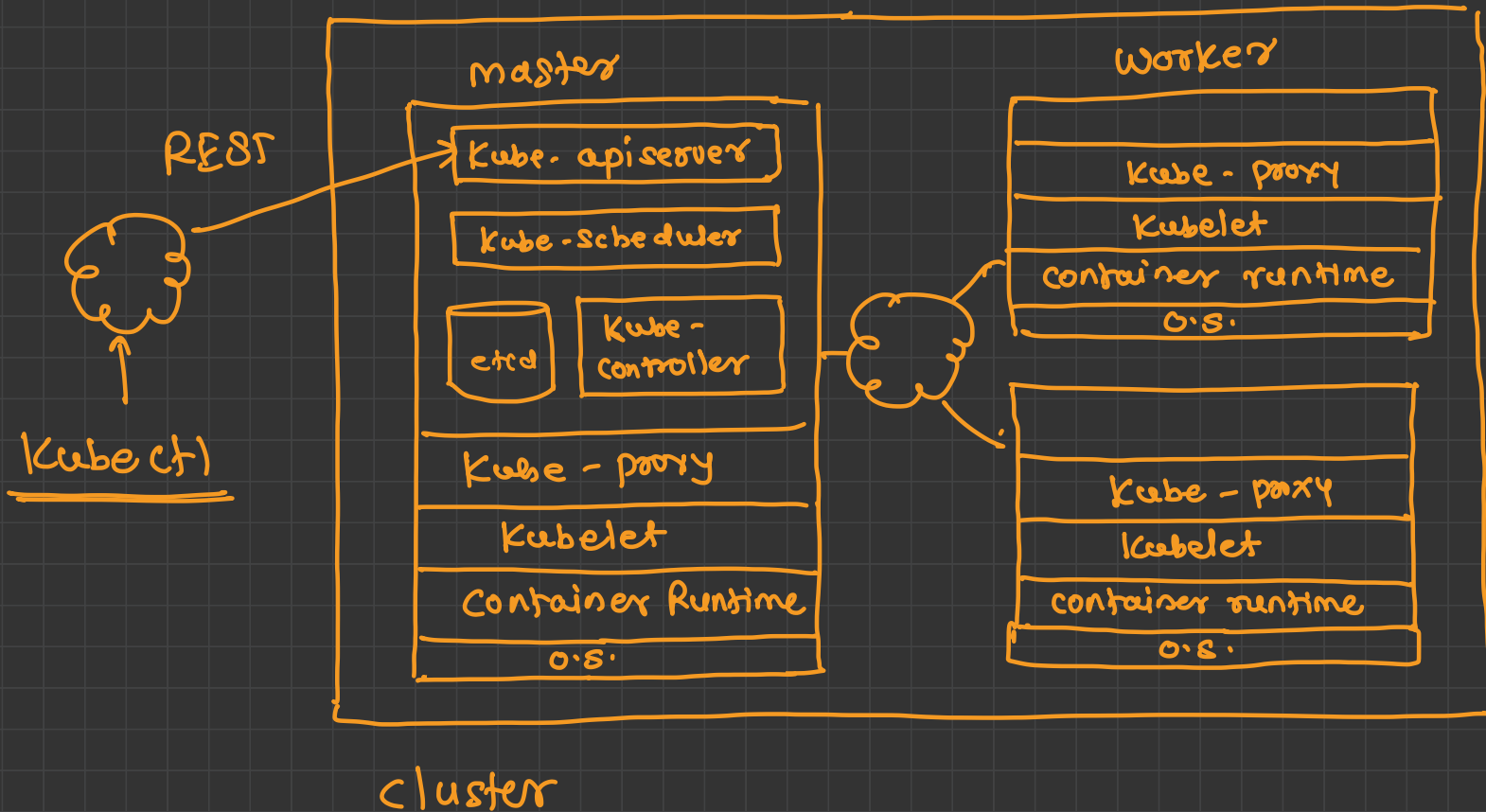
Master Components

- Master components make global decisions about the and they detect and respond to cluster events

- Master components can be run on any machine in the cluster

- **kube-apiserver**
  - The API server is a component that exposes the Kubernetes API
  - The API server is the front end for the Kubernetes

- **etcd**
  - Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data

- **kube-scheduler**
  - Component on the master that watches newly created pods that have no node assigned, and selects a node for them to run on

Master Components

- **kube-controller-manager**
    - Component on the master that runs controllers
    - Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process
    - Types
        - Node Controller: Responsible for noticing and responding when nodes go down.
        - Replication Controller: Responsible for maintaining the correct number of pods for every replication controller object in the system
        - Endpoints Controller: Populates the Endpoints object (that is, joins Services & Pods)
        - Service Account & Token Controllers: Create default accounts and API access tokens for new namespaces
- **cloud-controller-manager**
    - Runs controllers that interact with the underlying cloud providers
    - The cloud-controller-manager binary is an alpha feature introduced in Kubernetes release 1.6

Node Components

- Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment

- **kubelet**
    - An agent that runs on each node in the cluster
    - It makes sure that containers are running in a pod

- **kube-proxy**
    - Network proxy that runs on each node in your cluster, implementing part of the Kubernetes service concept
    - kube-proxy maintains network rules on nodes
    - These network rules allow network communication to your Pods from network sessions inside or outside of your cluster

- **Container Runtime**
    - The container runtime is the software that is responsible for running containers
    - Kubernetes supports several container runtimes: Docker, containerd, rktlet, cri-o etc.

Create Cluster

- Use following commands on both master and worker nodes

> sudo apt-get update && sudo apt-get install -y apt-transport-https curl

> curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

> cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list deb https://apt.kubernetes.io/kubernetes-xenial main EOF

> sudo apt-get update
> sudo apt-get install -y kubelet kubeadm kubectl
> sudo apt-mark hold kubelet kubeadm kubectl

Initialize Cluster Master Node

- Execute following commands on master node

> kubeadm init --apiserver-advertise-address=<ip-address> --pod-network-cidr=10.244.0.0/16

> mkdir -p $HOME/.kube

> sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

> sudo chown $(id -u):$(id -g) $HOME/.kube/config
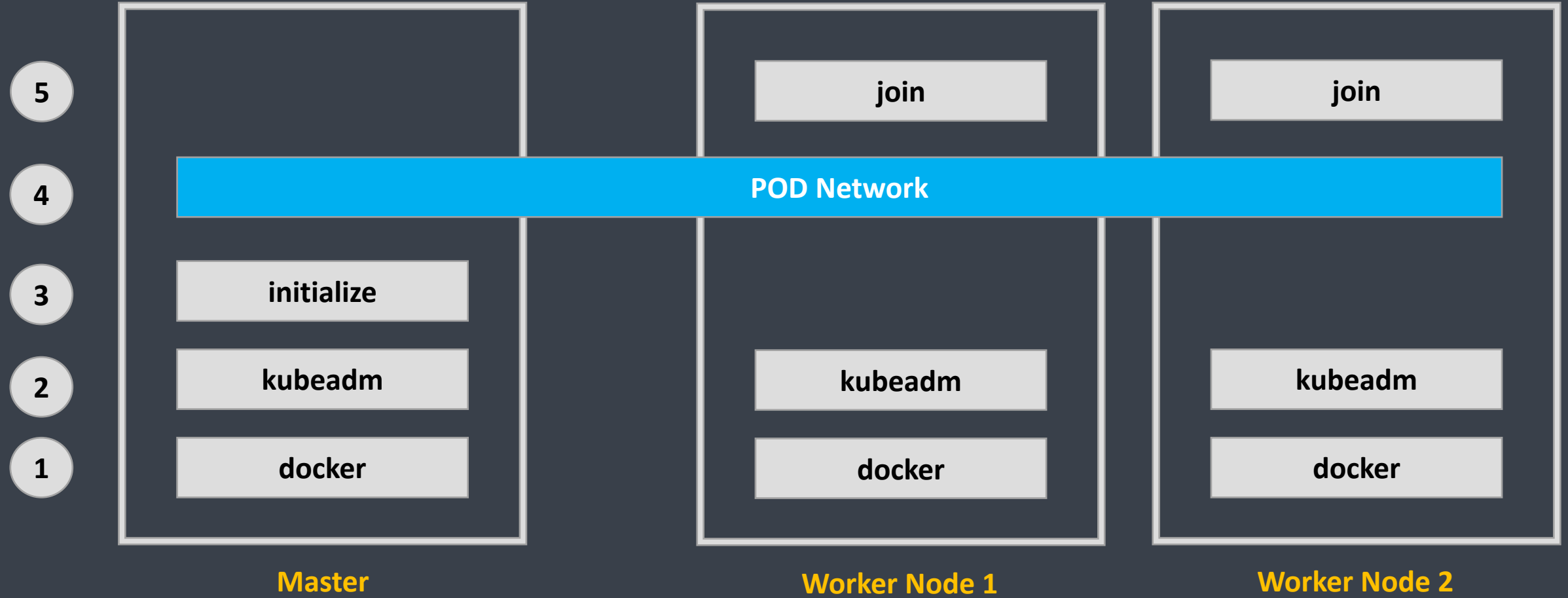

- Install pod network add-on

> kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/2140ac876ef134e0ed5af15c65e414cf26827915/Documentation/kube-flannel.yml

Add worker nodes

- Execute following command on every worker node

> kubeadm join --token <token> <control-plane-host>:<control-plane-port> --discovery-token-ca-cert-hash sha256:<hash>

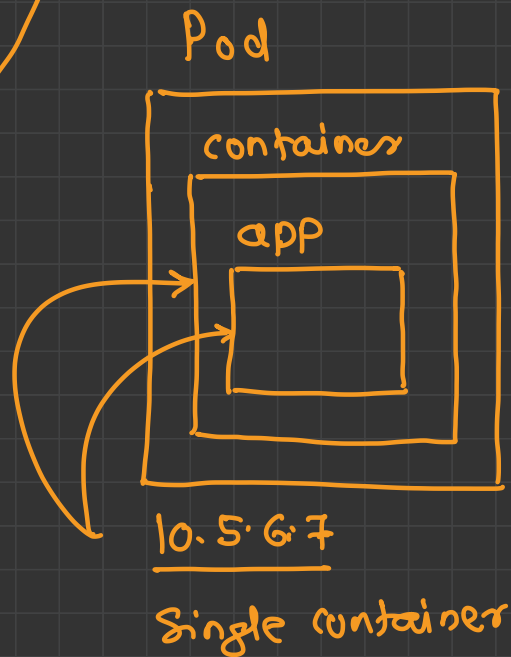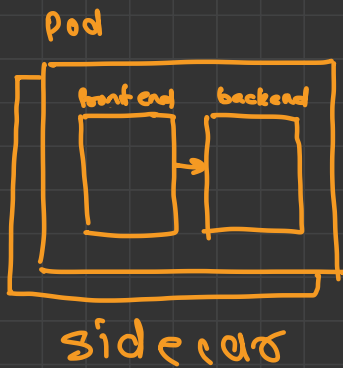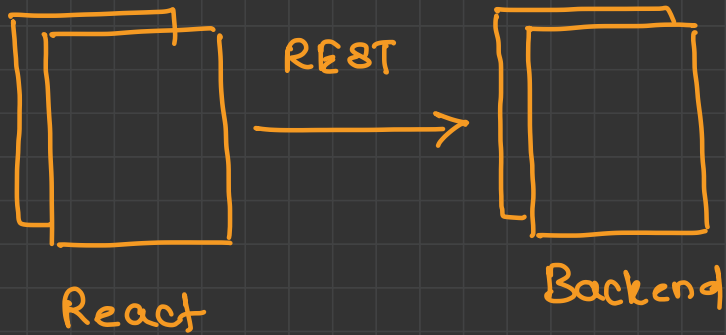# Steps to install Kubernetes

Kubernetes Objects

- The basic Kubernetes objects include
    - Pod
    - Service
    - Volume
    - Namespace

- Kubernetes also contains higher-level abstractions build upon the basic objects
    - Deployment
    - DaemonSet
   - StatefulSet
    - ReplicaSet
    - Job

Namespace

- Namespaces are intended for use in environments with many users spread across multiple teams, or projects
- Namespaces provide a scope for names
- Names of resources need to be unique within a namespace, but not across namespaces
- Namespaces can not be nested inside one another and each Kubernetes resource can only be in one namespace
- Namespaces are a way to divide cluster resources between multiple users

React

REST

Backend

Pod

container

app

10.5.6.7

Single container

Pod

frontend   backend

sidecar

Pod

- A Pod is the basic execution unit of a Kubernetes application

- The smallest and simplest unit in the Kubernetes object model that you create or deploy

- A Pod represents processes running on your Cluster

- Pod represents a unit of deployment

- A Pod encapsulates
  - application's container (or, in some cases, multiple containers)
  - storage resources
  - a unique network IP
  - options that govern how the container(s) should run

Pod

Single container

multi-container

Sidecar

init container
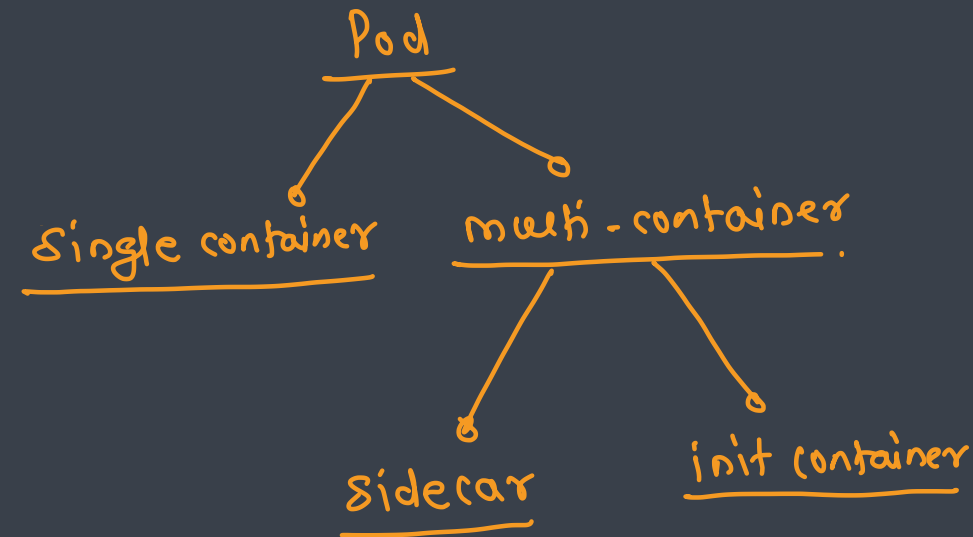
YAML

1] apiVersion: K8S api version < V1
                                   app/V1

2] kind: type of object [Pod, Replicaset, service...]

3] metadata: configuration of object [name, label, ns.~]

4] spec: specification of object

YAML to create Pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: httpd
```

Service

- An abstract way to expose an application running on a set of Pods as a network service

- Service is an abstraction which defines a logical set of Pods and a policy by which to access them (sometimes this pattern is called a micro-service)

- Service Types
    - ClusterIP
        - Exposes the Service on a cluster-internal IP
        - Choosing this value makes the Service only reachable from within the cluster
    - LoadBalancer
        - Used for load balancing the containers
    - NodePort

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
   - protocol: TCP
     port: 80
     targetPort: 9376
```

Service Type: NodePort

- Exposes the Service on each Node's IP at a static port (the NodePort)
- You'll be able to contact the NodePort Service, from outside the cluster, by requesting <NodeIP>:<NodePort>

Service

10.100.2.45
80

Port

TargetPort

32000

NodePort

80
10.30.4.6

POD
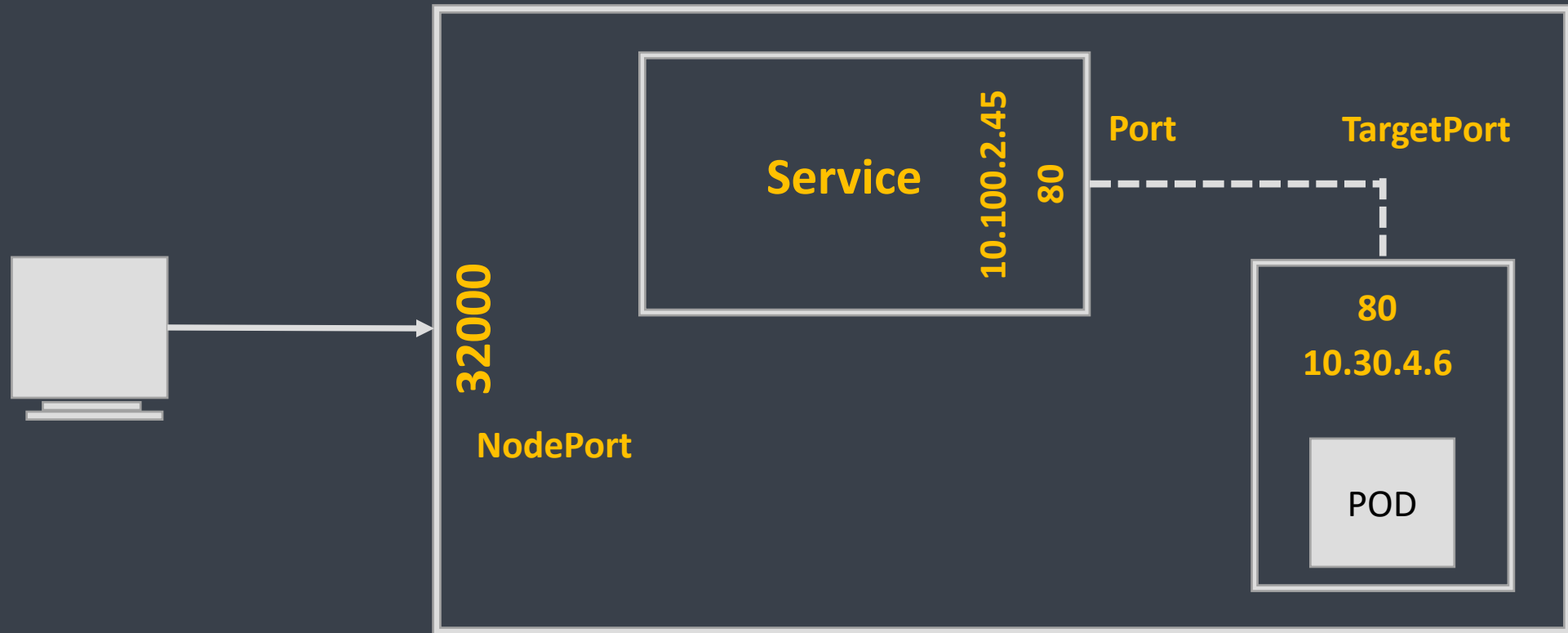
Replica Set

- A Replica Set ensures that a specified number of pod replicas are running at any one time

- In other words, a Replica Set makes sure that a pod or a homogeneous set of pods is always up and available

- If there are too many pods, the Replica Set terminates the extra pods

- If there are too few, the Replica Set starts more pods

- Unlike manually created pods, the pods maintained by a Replica Set are automatically replaced if they fail, are deleted, or are terminated

```yaml
apiVersion: v1
kind: ReplicaSet
metadata:
 name: nginx
spec:
 replicas: 3
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   name: nginx
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
     image: nginx
     ports:
     - containerPort: 80
```

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

- You can use deployment for
    - Rolling out ReplicaSet
    - Declaring new state of Pods
    - Rolling back to earlier deployment version
    - Scaling up deployment policies
    - Cleaning up existing ReplicaSet

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: website-deployment
spec:
 selector:
  matchLabels:
   app: website
 replicas: 10
 template:
  metadata:
   name: website-pod
   labels:
    app: website
  spec:
   containers:
   - name: website-container
     image: pythoncpp/test_website
     ports:
     - containerPort: 80
```

Volume

- On-disk files in a Container are ephemeral, which presents some problems for non-trivial applications when running in Containers

- Problems
    - When a Container crashes, kubelet will restart it, but the files will be lost
    - When running Containers together in a Pod it is often necessary to share files between those Containers

- The Kubernetes Volume abstraction solves both of these problems

- A volume outlives any Containers that run within the Pod, and data is preserved across Container restarts