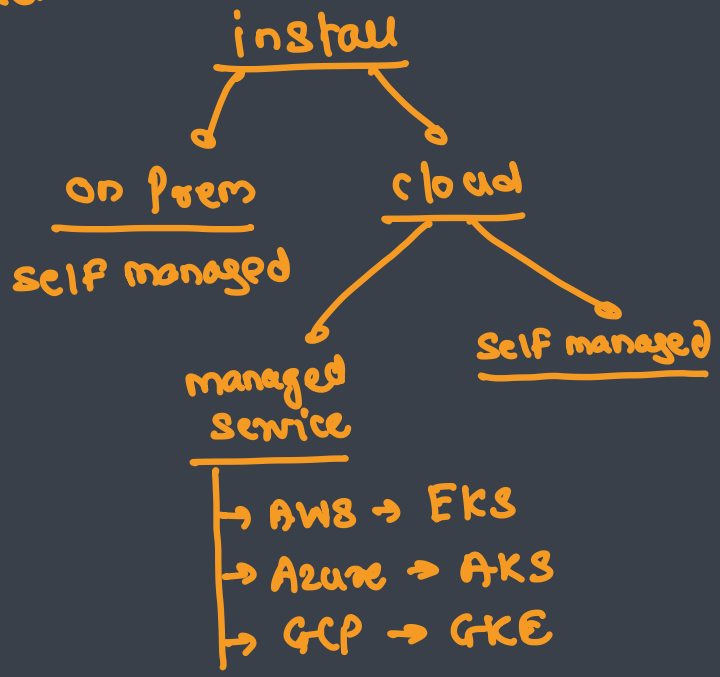
developed by Google

Kubernetes

orchestration tool

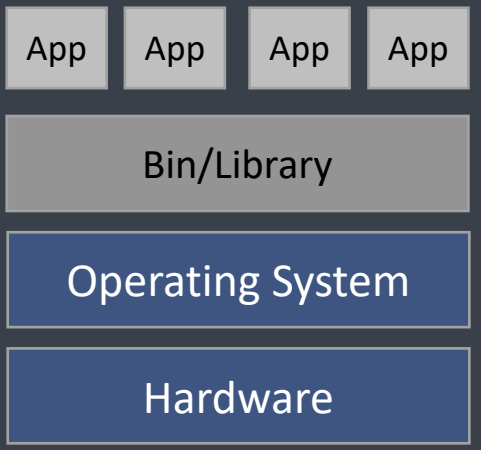
What is Kubernetes ?

- Portable, extensible, open-source platform for managing containerized workloads and services → orchestration
- Facilitates both declarative configuration and automation → YAML
- It has a large, rapidly growing ecosystem → Prometheus, ArgoCD, Istio, helm → cloud native apps
- Kubernetes services, support, and tools are widely available → on prem or cloud
- The name Kubernetes originates from Greek, meaning helmsman or pilot
- Google open-sourced the Kubernetes project in 2014



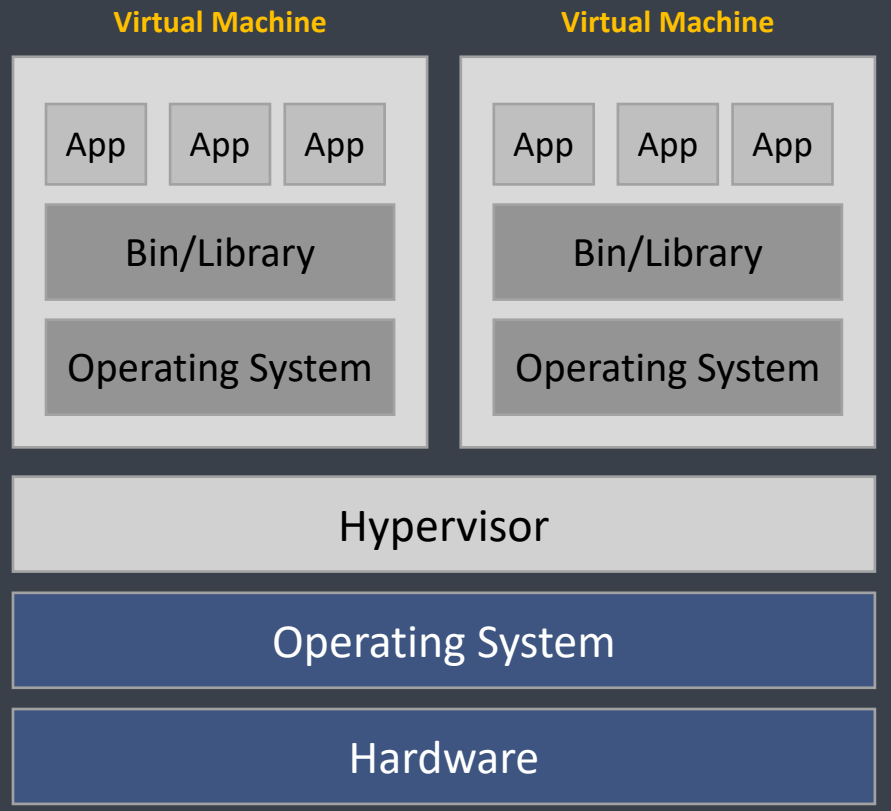
Traditional Deployment → using physical

- 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



Virtualized Deployment → using VM

- 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



Container deployment → using containers

- 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

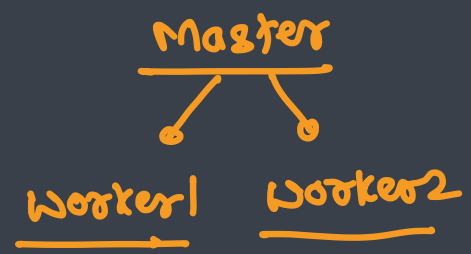
Container benefits

- 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
docker swarm — container mgmt, LB
Kubernetes — LB, Exposure of app

- **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** → management of storage
  - 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** → upgrading app
  - 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** → selection of worker to run the pod
  - 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** → ConfigMap
  - 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

Docker Swarm

service → hostport : container port

K8S

Service → ClusterIP
Service → NodePort → External Exposure
Service → LoadBalancer

What Kubernetes is not

- websites
- backend
- database
- job / cron jobs

- Does not limit the types of applications supported
- 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
- Does not provide nor mandate a configuration language/system
- Does not provide nor adopt any comprehensive machine configuration, maintenance, management, or self-healing systems

deployment → CI/CD tool
building → ant / maven / gradle

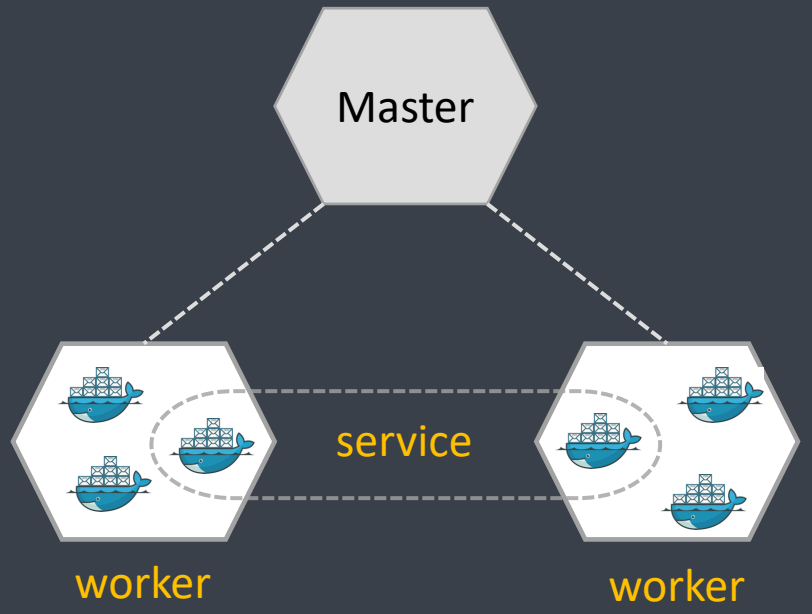
Minimum requirements

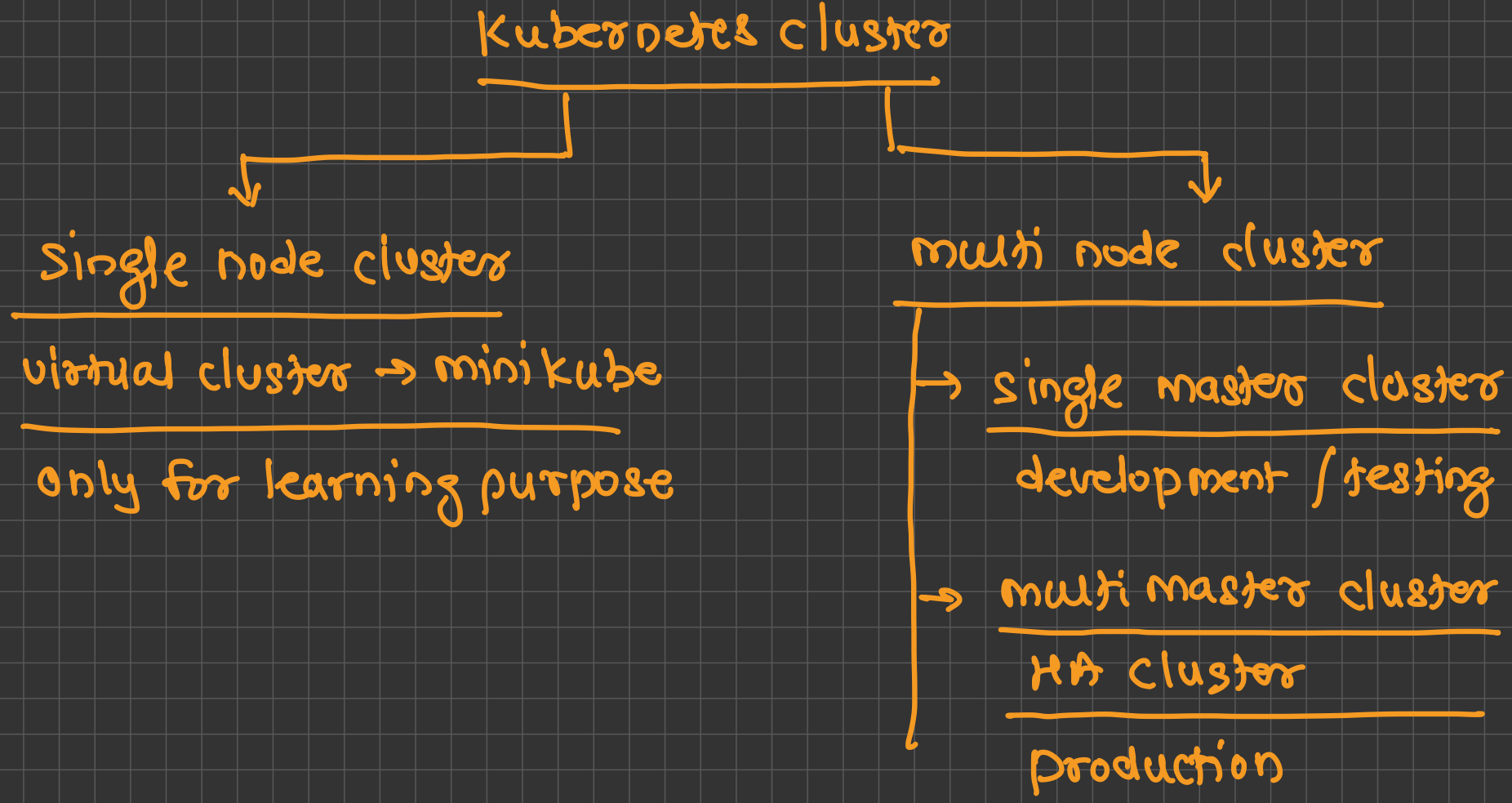
CPU - 2
RAM - 4 GB
OS → Linux → RHEL / SUSE / Ubuntu

Kubernetes Cluster

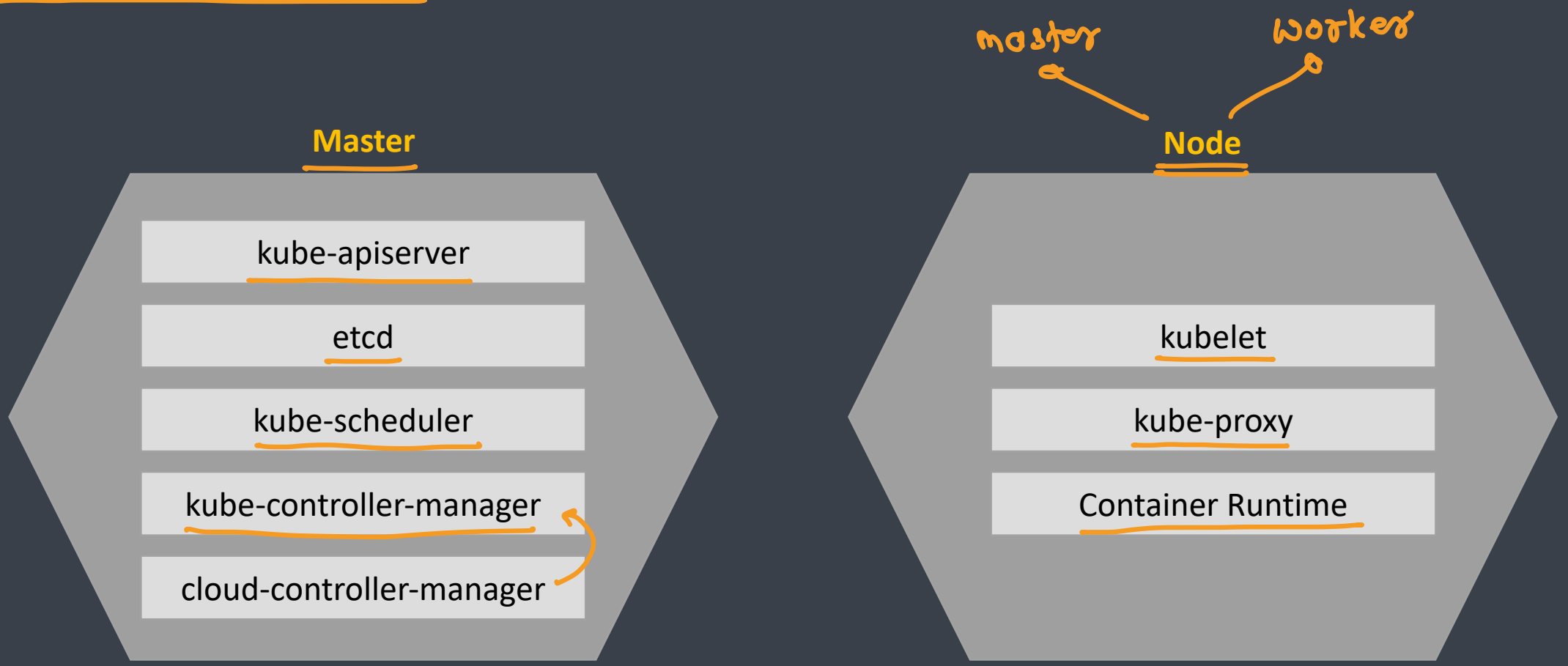
- When you deploy Kubernetes, you get a cluster.
- A cluster is a set of machines (nodes), that run containerized applications managed by Kubernetes
- A cluster has at least one worker node and at least one master node
- The worker node(s) host the pods that are the components of the application
- The 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

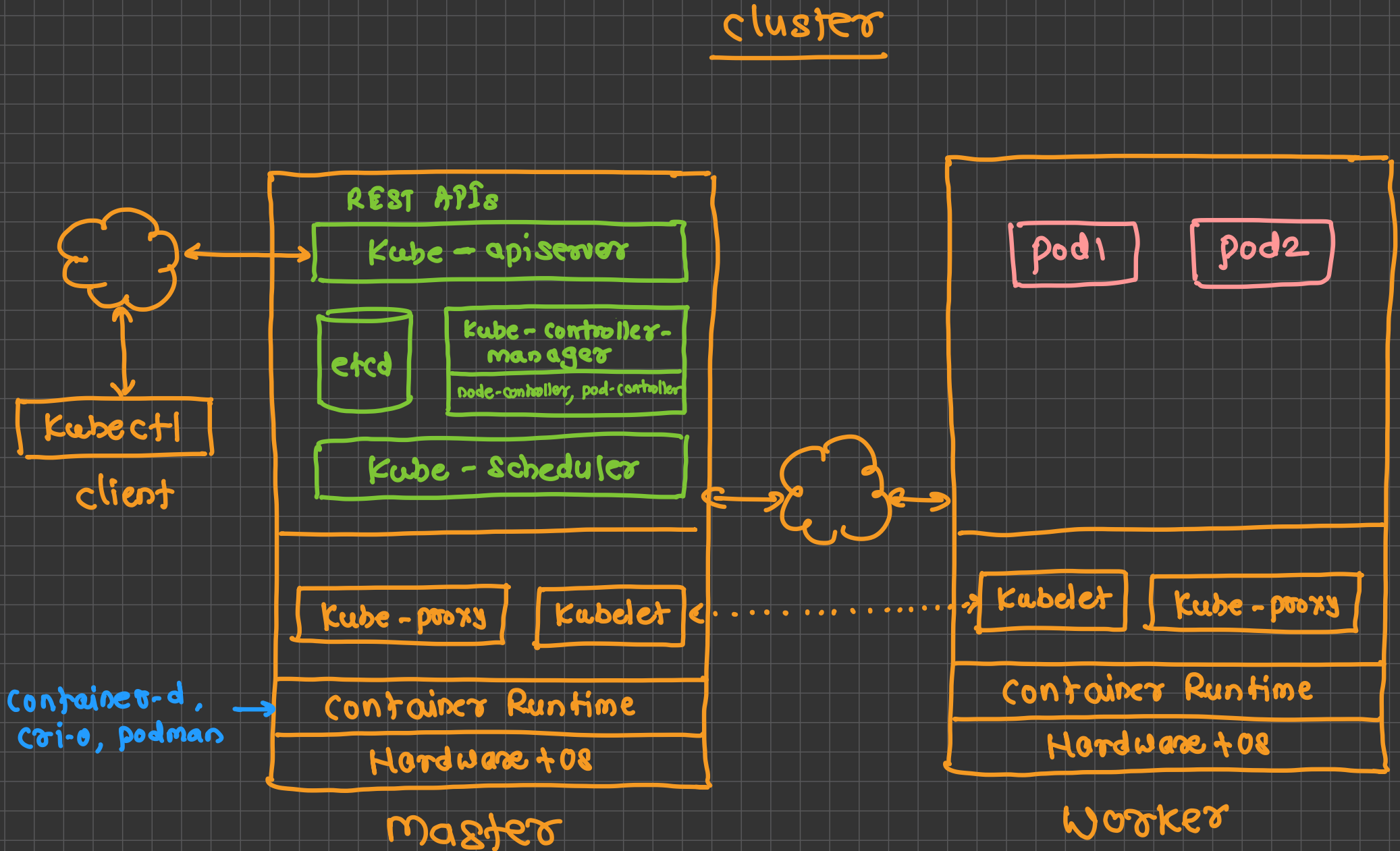
↳ multi master cluster → prod




Kubernetes cluster
Single node cluster
virtual cluster → minikube
only for learning purpose
multi node cluster
single master cluster
development / testing
multi master cluster
HA cluster
Production

Kubernetes Components




cluster
REST APIs
Kube - apiserver
etcd
Kube - controller-manager
node-controller, pod-controller
Kube - scheduler
Kubectl
client
Kube - proxy
Kubelet
Container Runtime
Hardware + OS
Master
containerd, cri-o, podman
pod1
pod2
Kubelet
Kube - proxy
Container Runtime
Hardware + OS
Worker

Master Components

- Master components make global decisions about the and they detect and respond to cluster events
  - pod stopping
  - node updating / stopping
  - ....
- 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 → monolithic app
  - 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 → desired state
    - 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 → RBAC
- **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

master
worker

- 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

} provides information to master

- **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

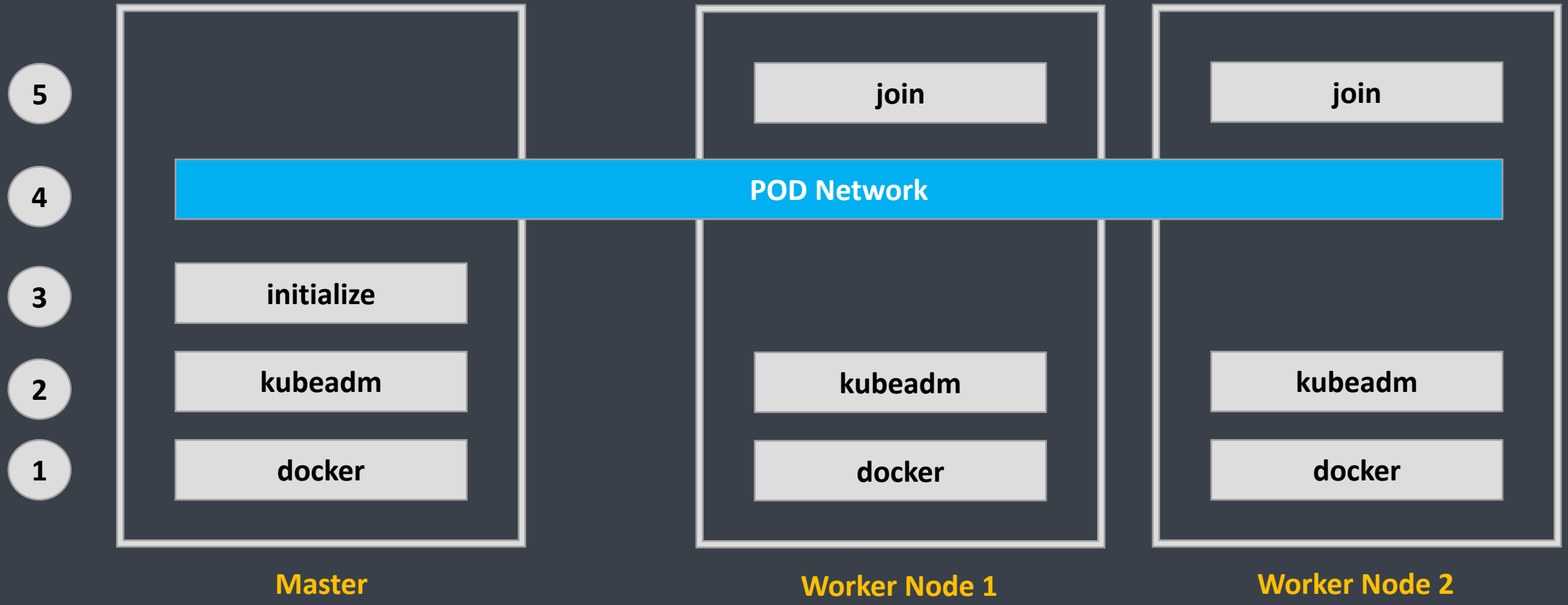
```
> kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/2140ac876ef134e0ed5af15c65e414cf26827915/Documentation/kube-flannel.yml
```

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

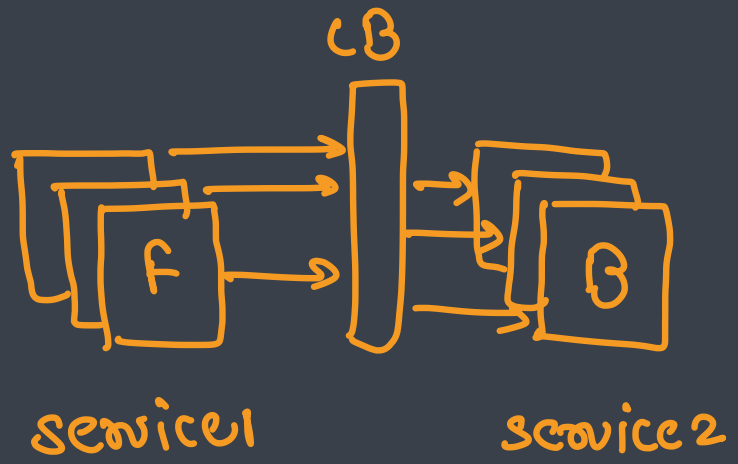
- The basic Kubernetes objects include
  - Pod
  - Service
  - Volume
  - Namespace
- Kubernetes also contains higher-level abstractions build upon the basic objects
  - Deployment
  - DaemonSet
  - StatefulSet
  - ReplicaSet
  - Job

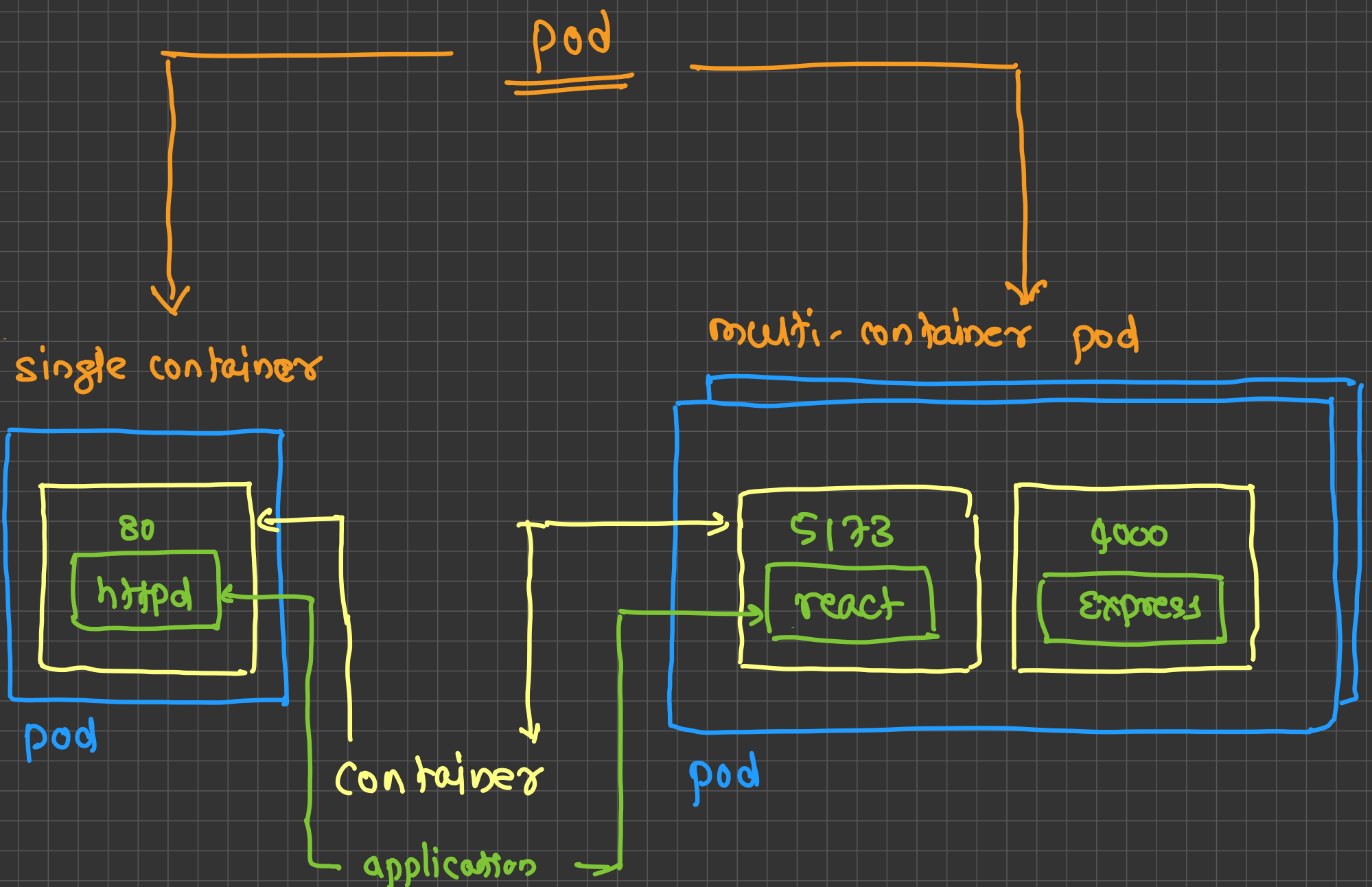
Namespace → collection of objects

- 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 / apps / environments / projects

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 pod
80
httpd
pod
Container
application
5173
react
4000
Express
pod

YAML to create Pod

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

apiVersion : k8s api version in which object schema is defined
- basic object → v1 (pod / service / volume / namespace)
- higher-order object → apps/v1 (Replicaset / Deployment / statefulstate ..)

kind : the type of object to be created (Pod / Service / Replicaset ..)

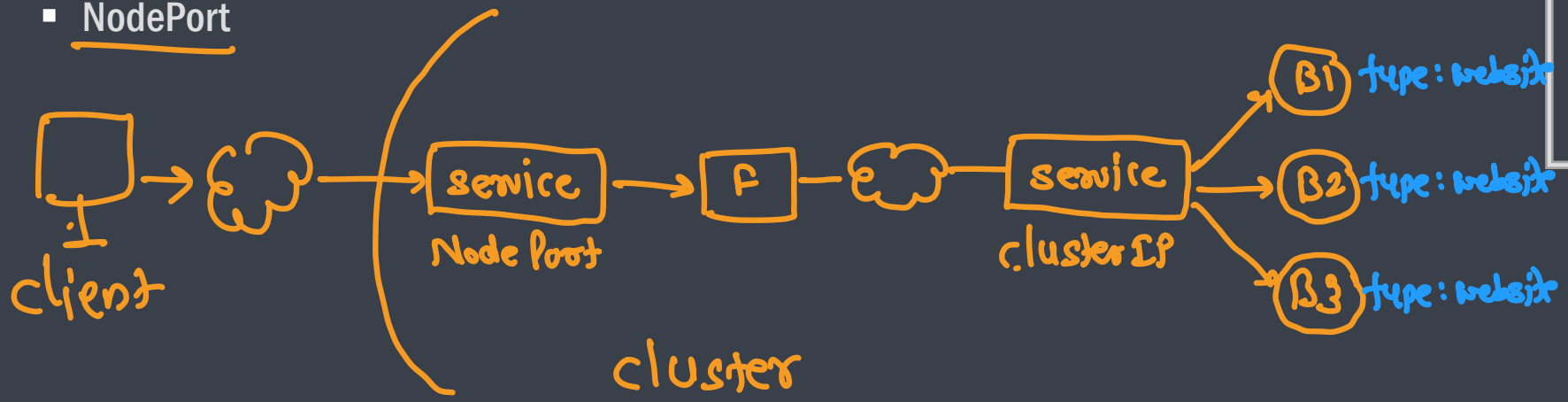
metadata : data about data (object)
- name : name of the object
- labels : labels to be attached on object

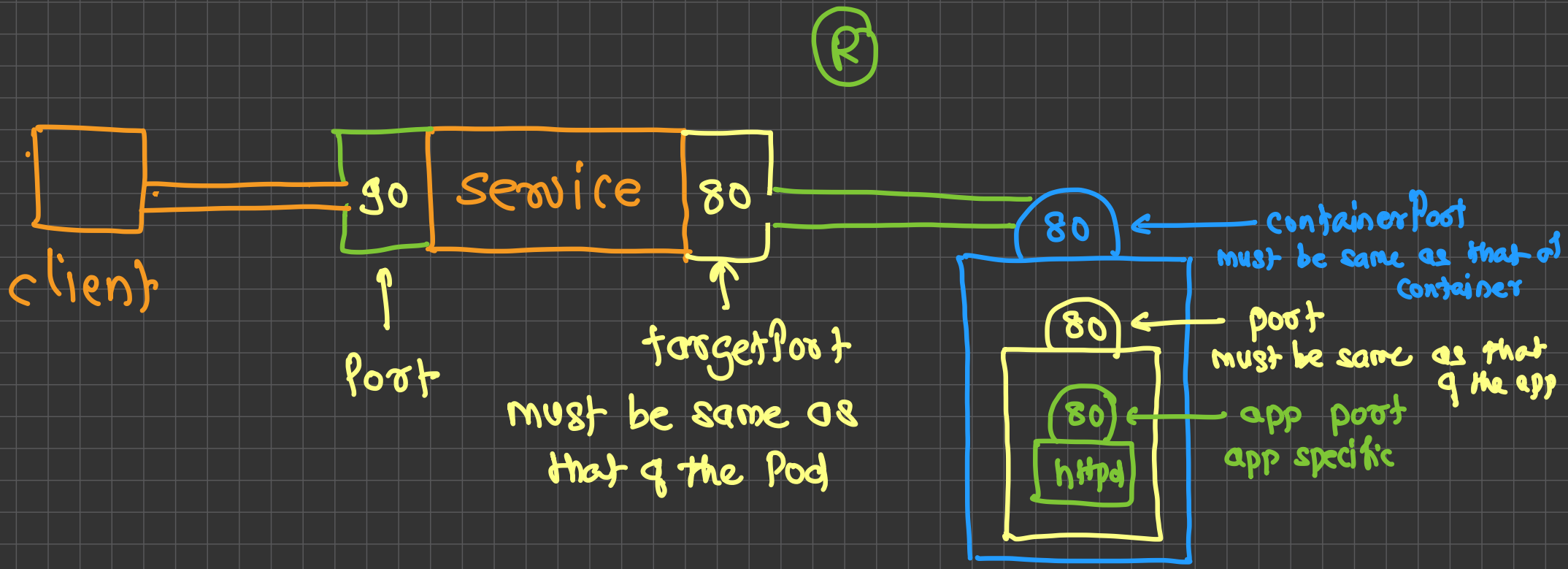
spec : specification / configuration of the object to be created

Service → LB + External app exposure

- 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 → used to load balance internal pods
    - 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

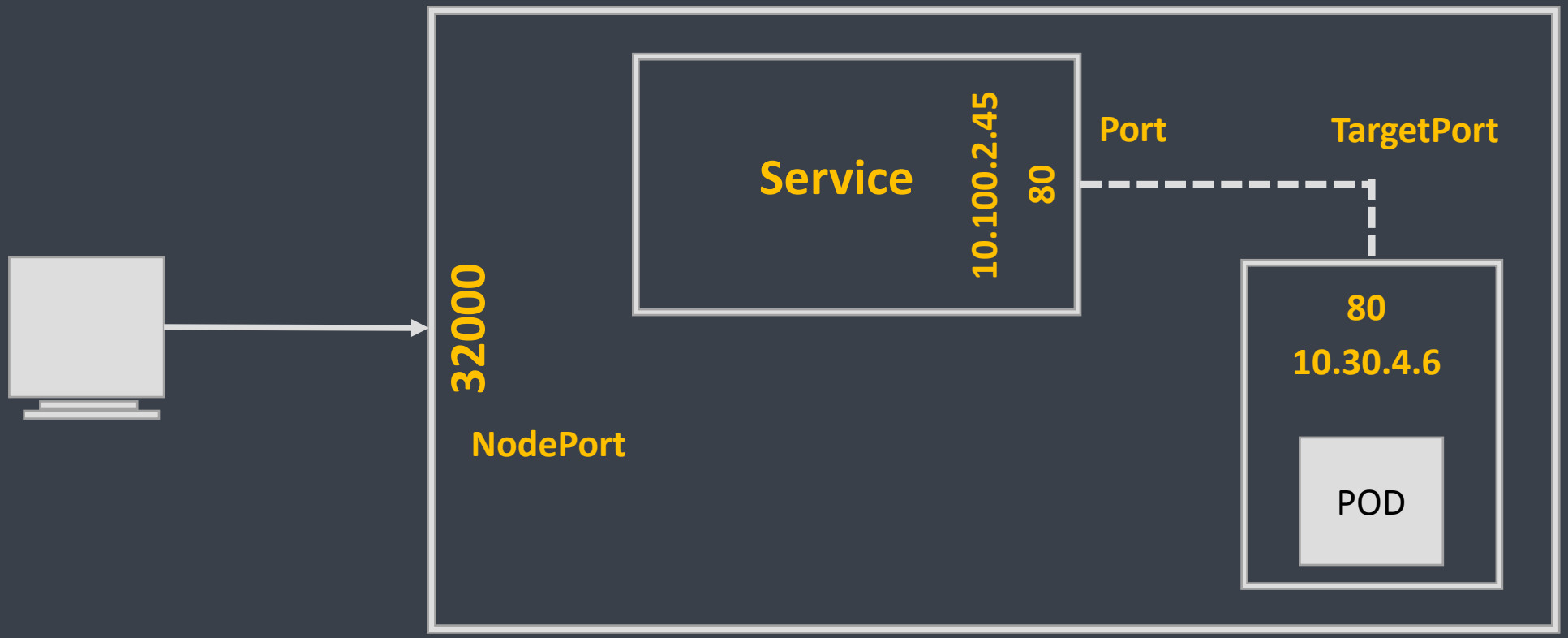
```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:          ← label
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376 80
```




R
client
90
service
80
Port
targetPort
must be same as that of the Pod
80
containerPort
must be same as that of Container
80
port
must be same as that of the app
80
app port
app specific
httpd

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>



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

```
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

```
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