Kubernetes

Container orchestration Engine

What is Kubernetes ?

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

→ Google

declarative config → using YAML syntax
→ simple text file
→ can be tracked by scm tool

automation → using commands

Ecosystem → no of tools supporting / extending K8S

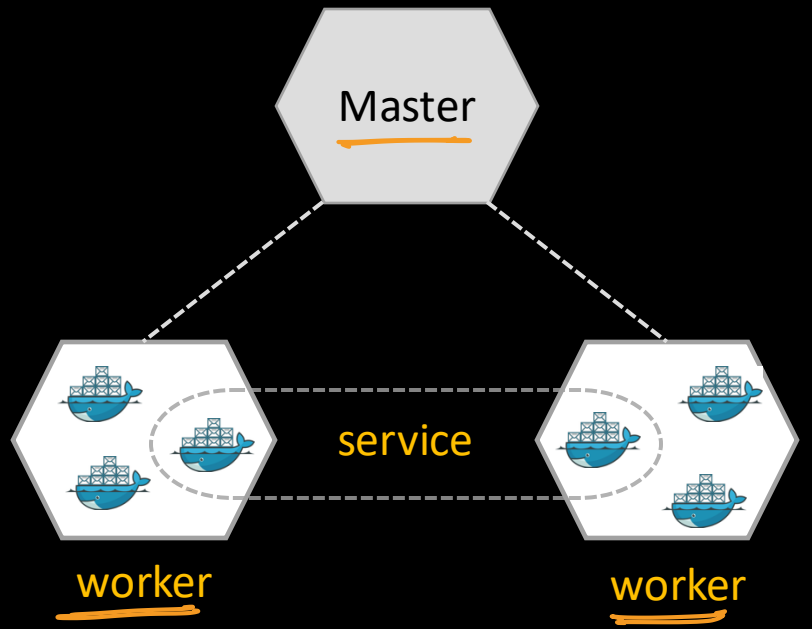
- → observability → prometheus
- → monitoring → Grafana, Kibana
- → Logging → Elastic Search, Logstash
- → Service mesh → Istio

K8S cluster configuration

- → on-prem
  - → private infrastructure
- → cloud
  - → Self managed
  - → cloud managed service
    - → AWS → EKS → Elastic K8S Service
    - → Azure → AKS → Azure K8S Service
    - → GCP → GKE → GCP K8S Engine

Kubernetes Cluster

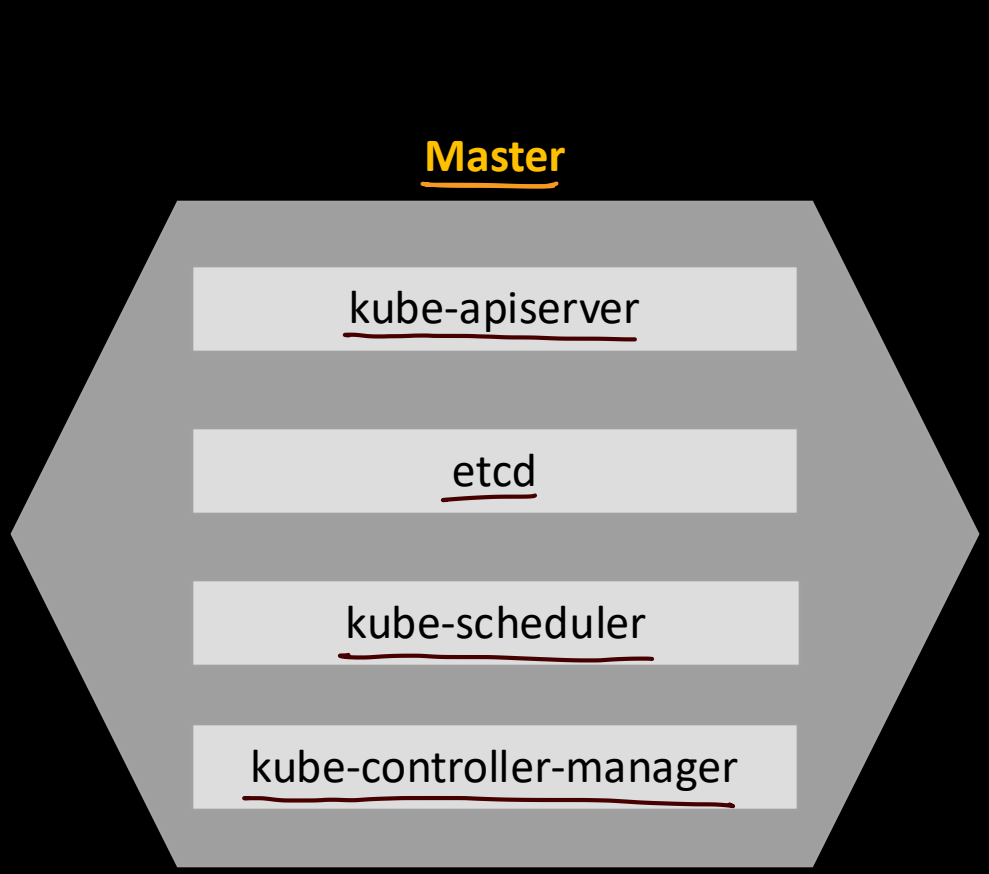
- When you deploy Kubernetes, you get a cluster.
- A cluster is a set of machines (nodes), that run containerized applications managed by Kubernetes
- A cluster has at least one worker node and at least one master node
- The worker node(s) host the pods that are the components of the application
- The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

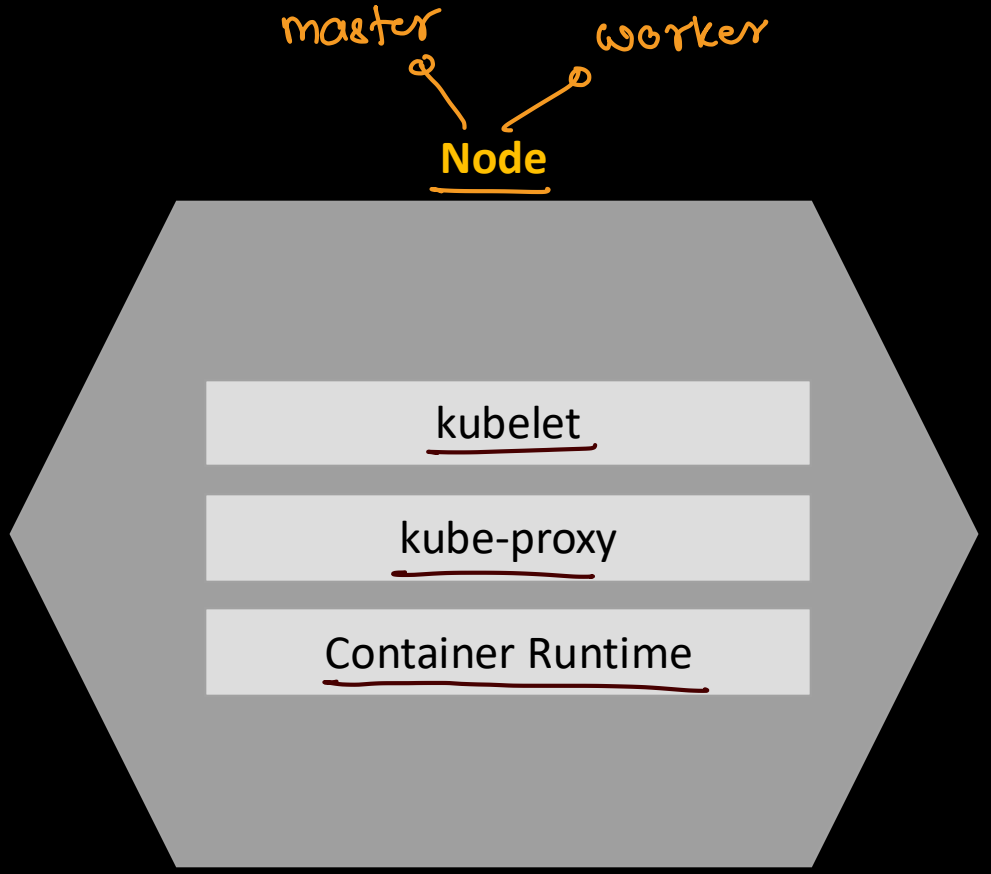


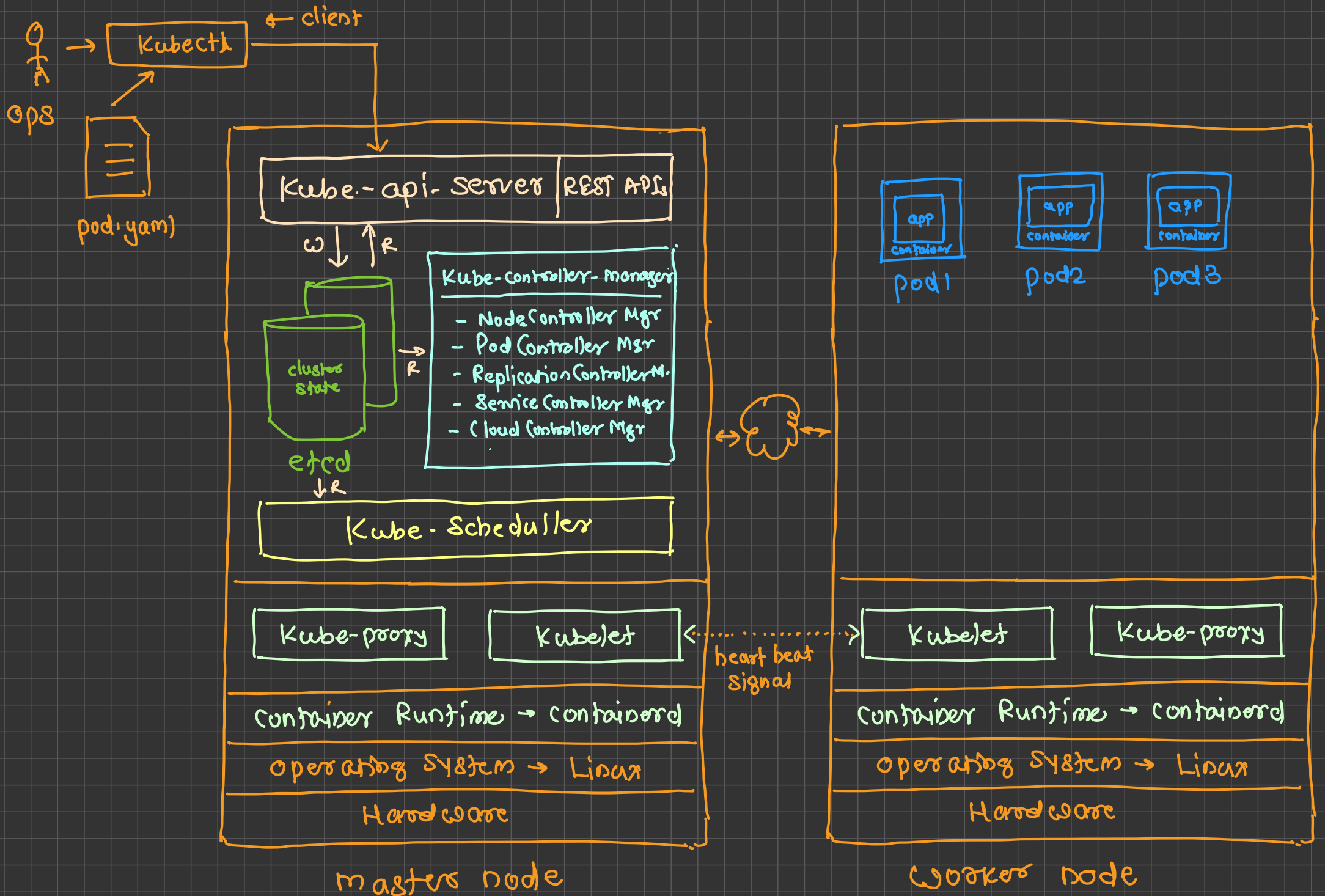
K8S cluster

- single node cluster
  - minikube → fake cluster
  - used only for learning k8s
- multi-node cluster
  - single master cluster
    - only one master and one or more workers
    - used in dev and staging/testing env.
  - multi-master cluster
    - more than one masters and more than one workers
    - also known HA cluster
    - used in pre-prod/UAT/production env.

Kubernetes Components






Kubectl
client
ops
pod.yaml
Kube-api-server
REST APIs
W
R
Kube-controller-manager
- NodeController Mgr
- Pod Controller Mgr
- ReplicationController M.
- Service Controller Mgr
- Cloud Controller Mgr
cluster state
R
etcd
R
Kube-Schedulller
Kube-proxy
Kubelet
heart beat signal
Kubelet
Kube-proxy
Container Runtime → containerd
Operating System → Linux
Hardware
master node
app container
pod1
app container
pod2
app container
pod3
Container Runtime → containerd
Operating System → Linux
Hardware
worker node

Master Components

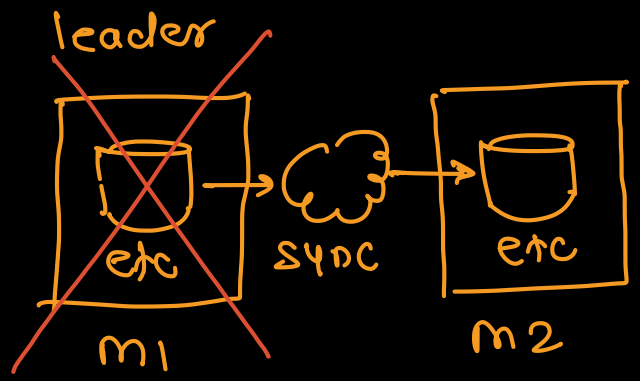
- These are the **brain** of the Kubernetes cluster — they make all the decisions about what runs where, how scaling happens, and how the system stays healthy.
- Master components can be run on any machine in the cluster

Component	Function
kube-apiserver	The **front door** of the Kubernetes control plane. It exposes the **Kubernetes API**, which is used by all components, kubectl, and external clients.
etcd	The **database** of Kubernetes — stores all cluster data, configurations, and states in a key-value format.
kube-scheduler	Decides **which node** each Pod should run on based on resource requirements and constraints.
kube-controller-manager	Ensures that the **actual state** of the cluster matches the **desired state** defined in manifests.
cloud-controller-manager (optional)	Manages integration between Kubernetes and your **cloud provider** (e.g., AWS, GCP, Azure).

Master Components

- **kube-apiserver** — REST APIs
  - Acts as the **communication hub** between users, components, and the cluster.
  - Every kubectl command goes through it.
  - Validates requests and updates etcd accordingly.
  - It's a **stateless** service — you can run multiple instances for high availability

- **etcd** → database
  - A **distributed, consistent key-value store**.
  - Stores all cluster data, including:
    - Pod states
    - Configurations
    - Secrets
    - Node information
  - It's the **source of truth** for your cluster.
  - It's critical — if etcd is lost, your cluster loses its state.



Master Components

- **kube-scheduler**
  - Watches for **new Pods** that don't have a Node assigned.
  - Chooses the **best Node** to run the Pod based on:
    - Resource requests (CPU, memory)
    - Node affinity/anti-affinity
    - Taints and tolerations
    - Pod priorities
    - Custom policies
- **kube-controller-manager**
  - Runs a set of **controller loops**, each responsible for maintaining part of the system's desired state.
  - Examples of controllers:
    - **Node Controller** – manages node status.
    - **Replication Controller** – ensures the desired number of pod replicas are running.
    - **Endpoint Controller** – manages endpoint objects.
    - **Service Account & Token Controllers** – create default accounts and tokens.

Node Components

- Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment
- **kubelet**
  - **Primary agent** that runs on every node.
  - Communicates with the **API Server**.
  - Ensures the containers defined in **PodSpecs** are running and healthy.
  - Reports node and pod status back to the control plane.
  - Watches for Pod definitions assigned to the node and runs them using the **container runtime**.
- **kube-proxy**
  - Maintains **network rules** on nodes.
  - Ensures that networking is consistent across the cluster.
  - Implements **Kubernetes Service abstraction** — enabling communication between different Pods and Services.
  - Uses **iptables** or **IPVS** to forward traffic to the correct Pod endpoints.
  - Supports load balancing between Pods behind a Service.
- **Container Runtime**
  - The actual **software responsible for running containers**.
  - The kubelet interacts with the runtime through the **Container Runtime Interface (CRI)**.
  - Common runtimes:
    - **containerd** (default on most modern clusters)
    - **CRI-O**
    - **Docker Engine** (deprecated as of K8s v1.24+)
    - **Mirantis Container Runtime**

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

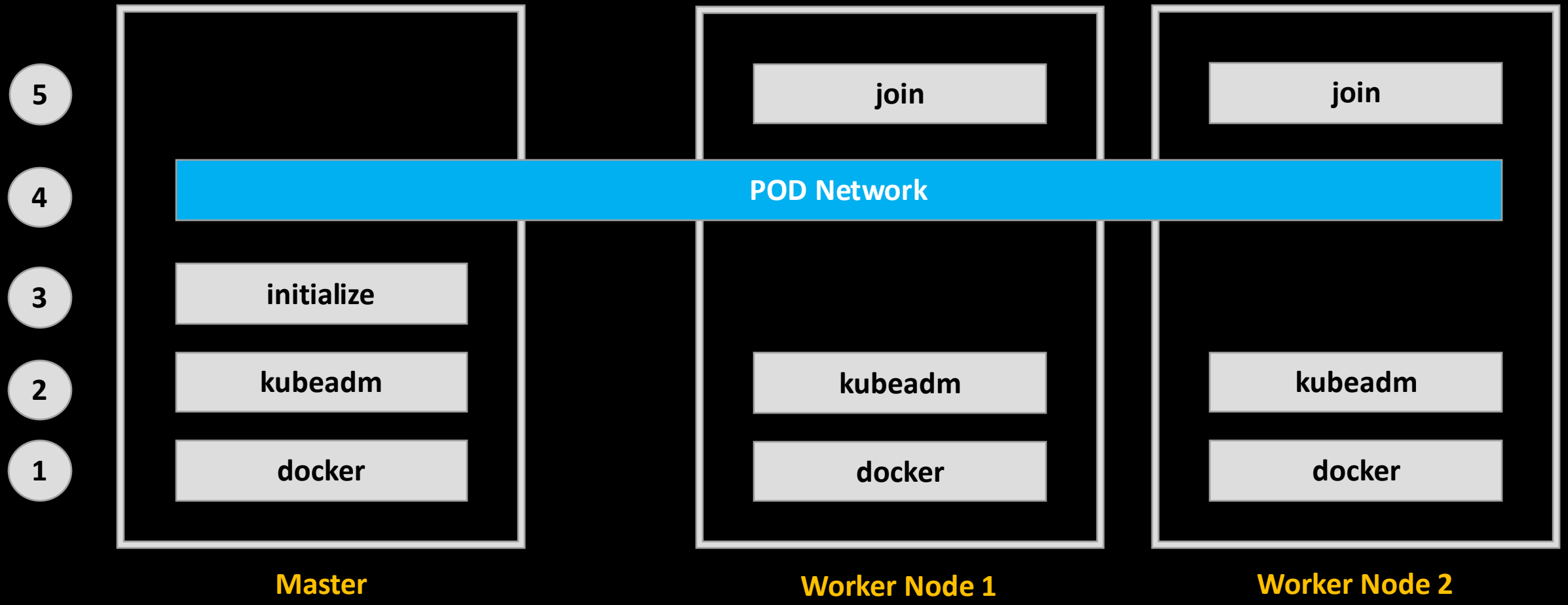
> kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/2140ac876ef134e0ed5af15c65e414cf26827915/Documentation/kube-flannel.yml

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

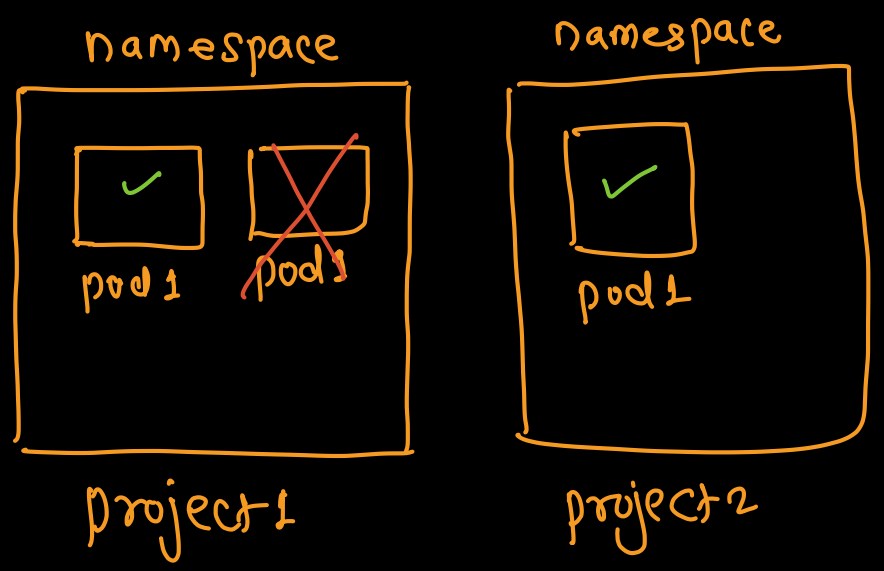


Kubernetes Objects

- 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

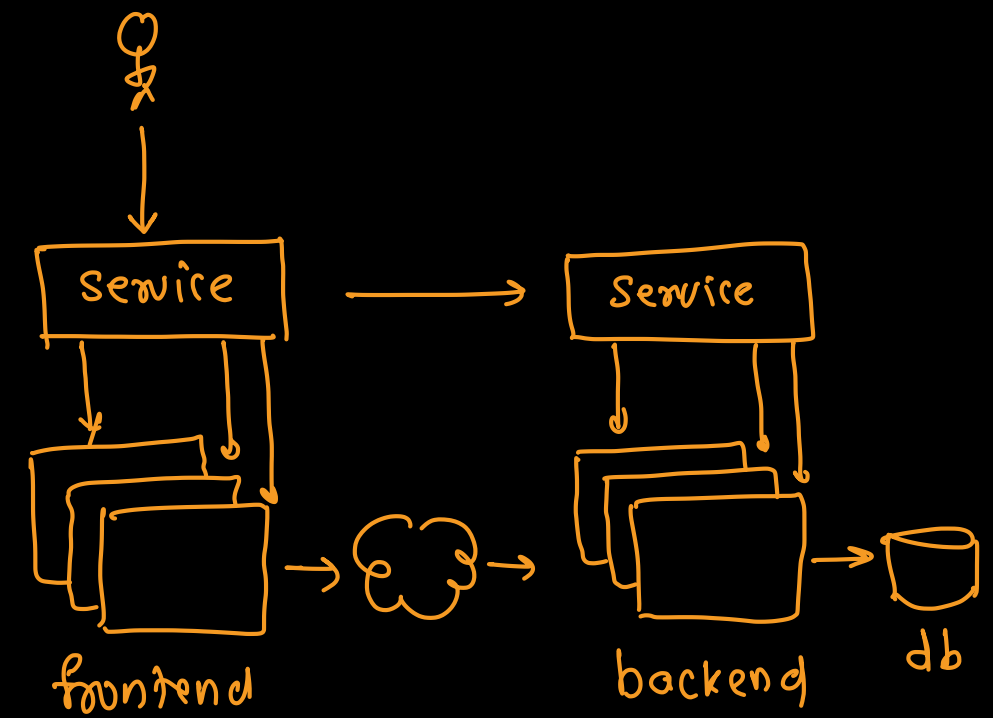
- Namespaces are intended for use in environments with many users spread across multiple teams, or projects
- Namespaces provide a scope for names
- Names of resources need to be unique within a namespace, but not across namespaces
- Namespaces can not be nested inside one another and each Kubernetes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

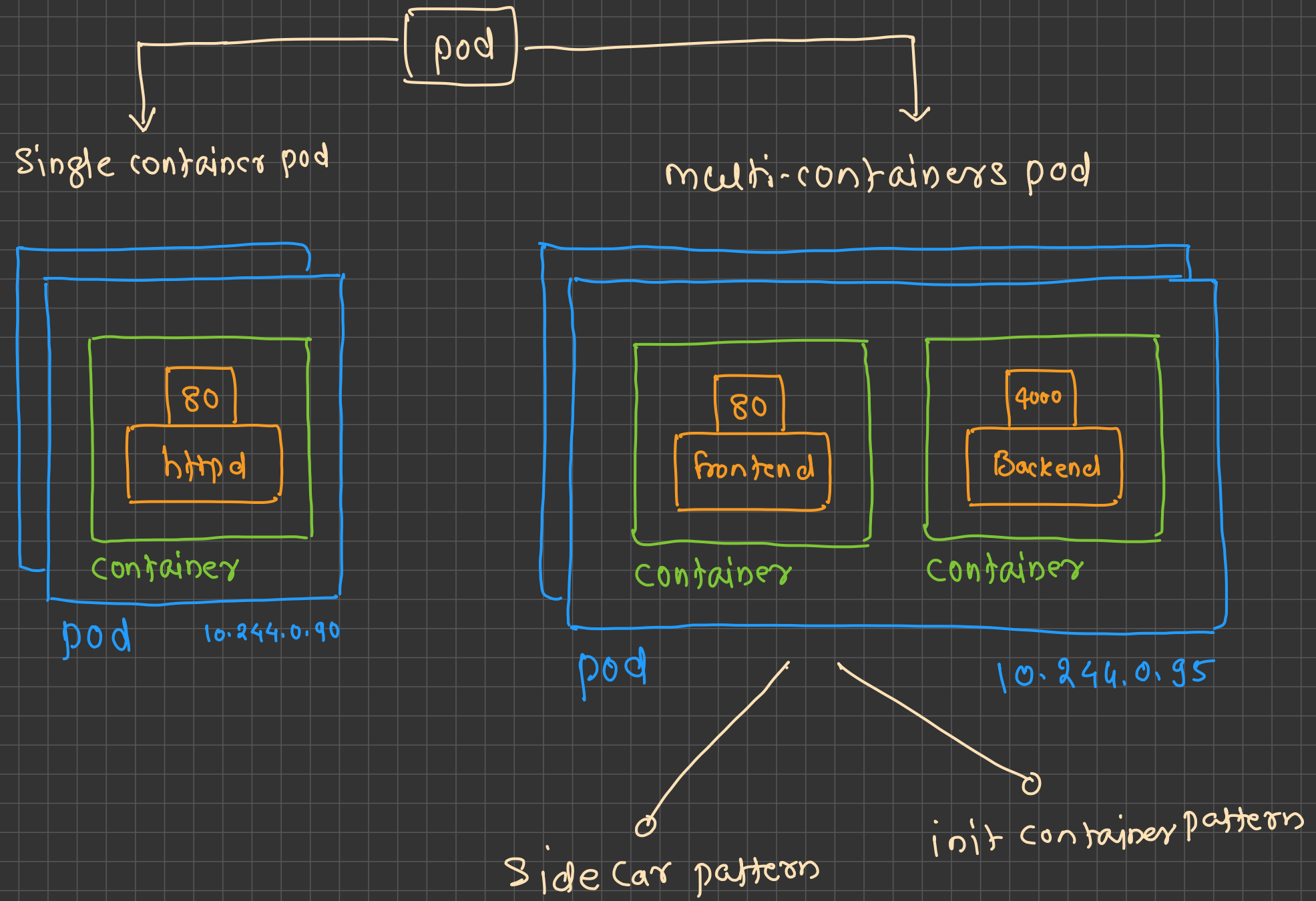


Pod

- 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

↳ configuration → resources → CPU/memory




pod
Single container pod
multi-containers pod
80
httpd
container
pod
10.244.0.90
80
frontend
container
4000
Backend
container
pod
10.244.0.95
Side car pattern
init container pattern

YAML → YAML Ain't Markup Language → recursive acronymes

20, person1, karad, p@test.com ← raw data → meaningless data

⇓

serialization format

⇓

meaningful information

XML

```
<person>
  <age> 20 </age>
  <name> person1 </name>
  <city> karad </city>
    :
</person>
```

JSON

```
{
  "age": 20,
  "name": "person1",
  "city": "karad",
    :
}
```

YAML

```
age: 20
name: "person1"
city: 'karad'
email: p1@test.com
```

YAML

- case sensitive → city & City are different
- tabs are NOT allowed
- spaces are used instead of tabs
- use .yaml or .yml as file extension
- space is required after : & -
- data types
  - scalar
    - single values → 60, person1, kanad
    - " or ' are NOT required for string values
  - map
    - pair of key-value
    - e.g. name: person1 , age: 20
  - list
    - collection of values (scalar or map)
    - eg.

```
# languages
- C
- C++
- python
```

```
- name: person1
  age: 20
- name: person2
  age: 30
```

YAML to create Pod

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

apiVersion
→ version of k8s api
→ for basic objects like pod, service etc = v1
→ for higher level objects like ReplicaSet, Deployment = apps/v1

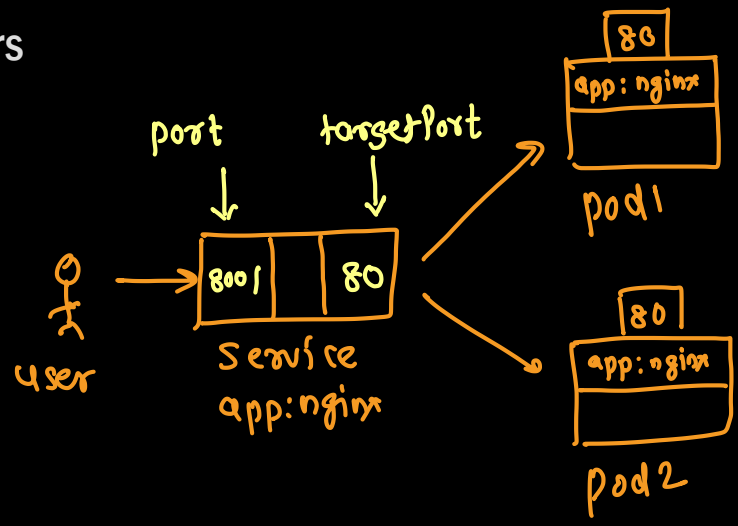
kind
→ type of object/resource to be created
→ eg. Pod, Service, Deployment etc.

metadata
→ more information of object to be created
→ eg. name, namespace, labels

Spec
→ specification of object to be created
→ depends on the object

Service → load balancer

- 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)
  - ↳ load balancing
- 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: 80
```

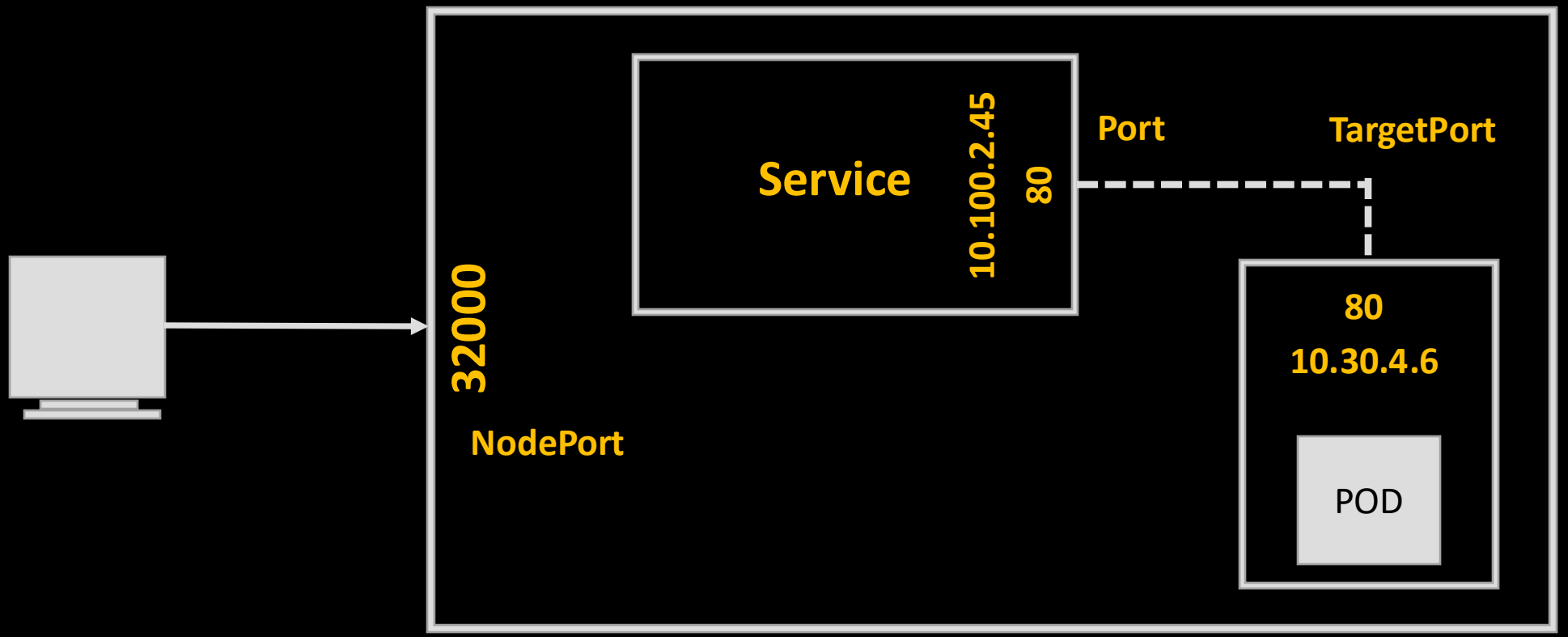
label configured on Pod(s)

user sends request

↳ pod's port

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

label → map (key-value pair)

desired count

- 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

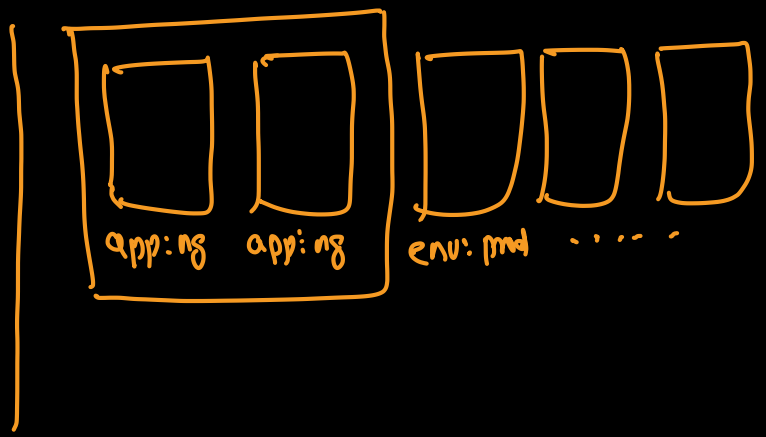
```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx
spec:
  replicas: 3 ← desired count
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
   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

criteria

same

Docker swarm →
- Container Scaling → Service
- load balancing → service

Kubernetes →
- Container Scaling → ReplicaSet
- load balancing → service



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
```