KOENIG
step forward

KUBERNETES ADMINISTRATOR

Gaurav

RHCSA | RHCE | CKA | SCA | SCE | SCA+ HA | ANSIBLE | DOCKER | OPENSHIFT | OPEN SOURCE TECHNOLOGIES

KOENIG
step forward

TABLE OF CONTENT

KOENIG
step forward

WHAT IS CONTAINER?

Containers are a way for developers to easily package and deliver applications, and for operations to easily run them anywhere in seconds, with no installation or setup necessary

KOENIG
step forward

WHY CONTAINER?

Containers require less system resources than traditional or hardware virtual machine environments because they don't include operating system images. Increased portability. Applications running in containers can be deployed easily to multiple different operating systems and hardware platforms.

KOENIG
step forward

CONTAINER ORCHESTRATION

KOENIG
step forward

SCENARIO

KOENIG
step forward

SCENARIO OF GUARD

- You are owner of a building and you have 5 spots where people can enter your building and you want 5 security guards guarding the spots. All good till now.

- Now consider one of the guard was out of service for 2 hours due to some personal reasons. Now as a building owner its your responsibility to guard or employ another guard replacing the existing. Do you like to be manually interrupted from your task to look after who is out and whom to replace.

- No, no one likes to be. Now the solution could be, go to a third party vendor who provides 24*7 availability of the guards. Its the responsibility of the vendor to make 24*7 availability based on the requirements.

KOENIG
step forward

WHAT PROBLEM DOES KUBERNETES SOLVE?

Now lets replace the characters.

- Building -> Your application
- Owner -> The application owner
- Guards -> Containers
- Vendor -> Kubernetes
- Configuration (repliica)-> That we make a deal with vendor to maintain appropriate conf 24*7.

KOENIG
step forward

CONTAINER ORCHESTRATION

- Container orchestration automates the deployment, management, scaling, and networking of containers. Enterprises that need to deploy and manage hundreds or thousands of Linux® containers so in this case we don’t want downtime so container orchestration will give this facility.

KOENIG
step forward

What is container orchestration used for?

- Provisioning and deployment
- Configuration and scheduling
- Resource allocation
- Container availability
- Scaling or removing containers based on balancing workloads across your infrastructure
- Load balancing and traffic routing
- Monitoring container health
- Configuring applications based on the container in which they will run
- Keeping interactions between containers secure

KOENIG
step forward

TOOLS

- DOCKER SWARM
- KUBERNETES

KOENIG
step forward

DIFFERENCE BETWEEN DOCKER SWARM AND KUBERNETES

KOENIG
step forward

INSTALLATION CLUSTER CONFIGURATION

DOCKER SWARM

- SETTING UP THE CLUSTER IS SIMPLE
- REQUIRES ONLY 2 COMMANDS

KUBERNETES

- SETTING UP THE CLUSTER IS CHALLENGING AND COMPLICATED

KOENIG
step forward

USER INTERFACE

DOCKER SWARM

- THERE IS NOT GUI AVAILABLE

KUBERNETES

- PROVIDES A GUI (KUBERNETES DASHBOARD)

KOENIG
step forward

SCALABILITY

DOCKER SWARM

- Scalling up is 5X faster than kubernetes
- Better when 10-20 containers

KUBERNETES

- Scalling up is easy
- Better when 100-1000 containers in pod

KOENIG
step forward

AUTO SCALING

DOCKER SWARM

- Scaling up or scaling down has to be done manually

KUBERNETES

- Based on server traffic, containers will be scalled automatically

KOENIG
step forward

LOAD BALANCING

DOCKER SWARM

- SWARM DOES AUTO LOAD BALANCING

KUBERNETES

- Manual configuration needed for load balancing traffic

KOENIG
step forward

ROLLING UPDATE AND ROLLBACKS

DOCKER SWARM

- Rolling updates progressively updates the containers one after the other while ensuring HA
- No automatic Rollbacks

KUBERNETES

- Rolling updates progressively updates the pods one after the other while ensuring HA
- Automatic Rollbacks in case of failure

KOENIG
step forward

DATA VOLUMES

DOCKER SWARM

- Storage volumes can be shared with any other container in the node

KUBERNETES

- Storage volumes can be shared only between containers within the same pod

KOENIG
step forward

LOGGING AND MONITORING

DOCKER SWARM

- 3rd party logging and monitoring tools required

KUBERNETES

- In-built logging and monitoring tools in place

KOENIG
step forward

WHAT IS KUBERNETES

Kubernetes is a system for managing containerized applications across a cluster of nodes.

KOENIG
step forward

Why you need Kubernetes and what can it do?

Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Wouldn’t it be easier if this behavior was handled by a system?

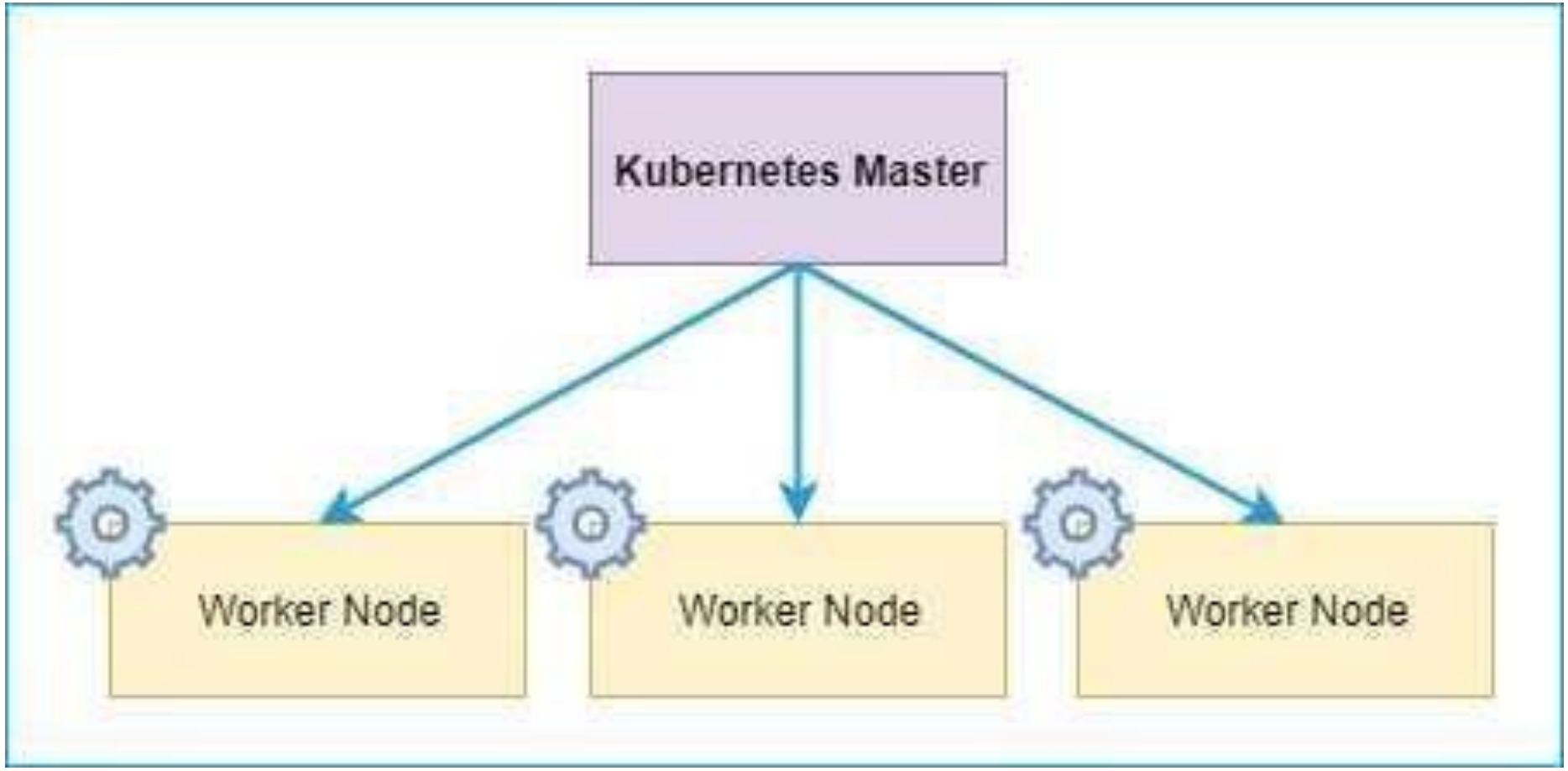
That’s how Kubernetes comes to the rescue!

kubernetes

- Self-Healing
- Automated Rollbacks
- Auto Scaling
- Load Balancing

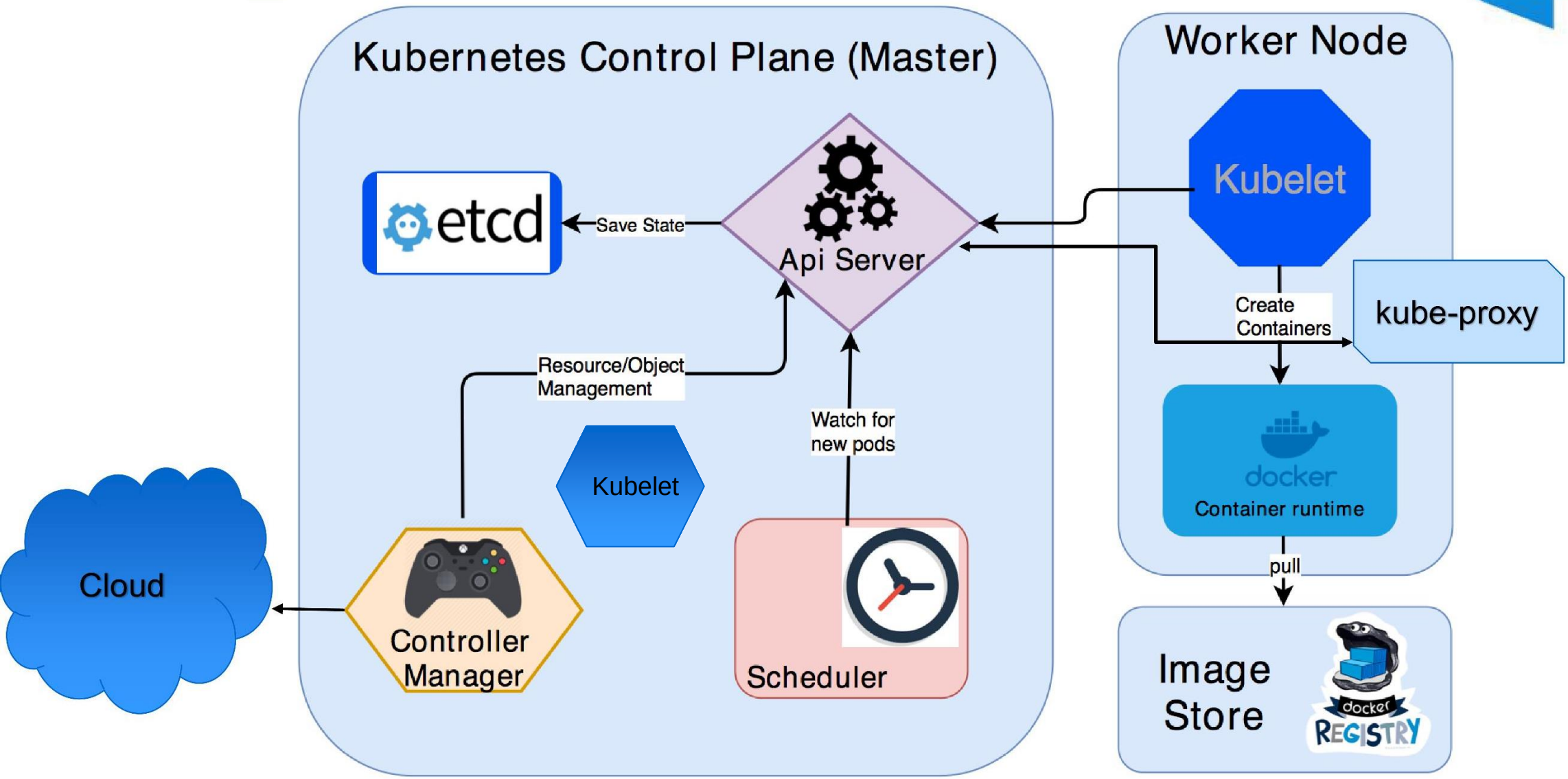
KOENIG
step forward

KUBERNETES COMPONENTS



KOENIG
step forward

KUBERNETES COMPONENTS / ARCHITECTURE



KOENIG
step forward

KUBE-APISERVER

Kubernetes is an API server which provides all the operation on cluster using the API. API server implements an interface, which means different tools and libraries can readily communicate with it. Kubeconfig is a package along with the server side tools that can be used for communication. It exposes Kubernetes API.

KOENIG
step forward

Kube-api Server

1. Authenticate User
2. Validate Request
3. Retrieve data
4. Update ETCD
5. Scheduler
6. Kubelet

KOENIG
step forward

ETCD

- ETCD is a distributed reliable key-vaue store that is simple,secure and fast.
- It stores the configuration information which can be used by each of the nodes in the cluster. It is a high availability key value store that can be distributed among multiple nodes.
- It is storing all the information of master and worker nodes
- It is accessible only by Kubernetes API server as it may have some sensitive information.
- It is a distributed key value Store which is accessible to all.

KOENIG
step forward

ETCD STORE information of

1) Nodes
2) POD's
3) Configs
4) Secrets
5) Accounts
6) Roles
7) Binding

KOENIG
step forward

KUBE-CONTROLLER-MANAGER

Controllers are the brain behind the orchestration. They are responsible for noticing and responding when nodes, containers or end point goes down. The controllers make decision to make up new containers in such cases. This component is responsible for most of the collectors that regulates the state of cluster and performs a task.

KOENIG
step forward

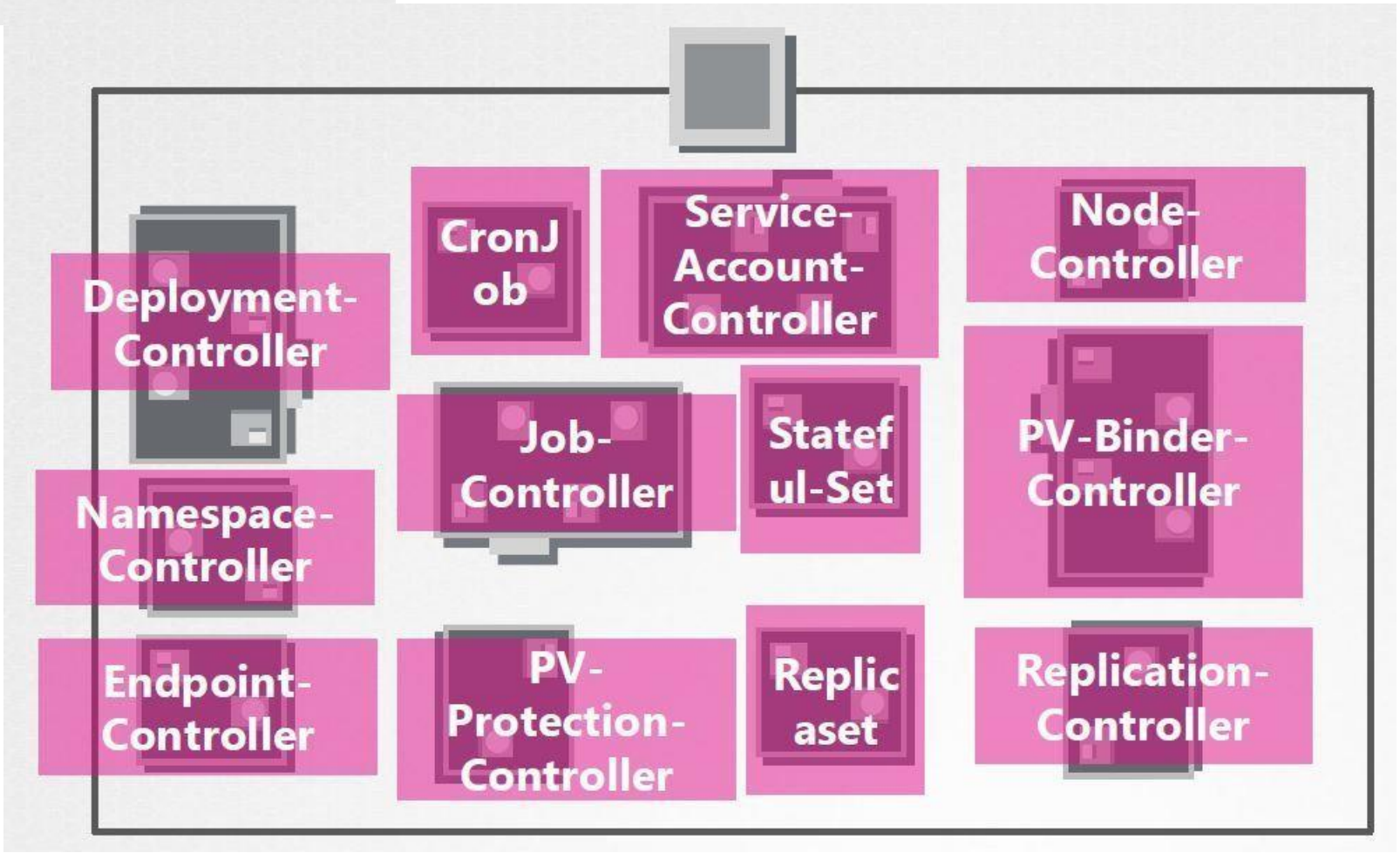
KUBE -CONTROLLER-MANAGER

Some of the controllers that controller-manager include:

- **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.

KOENIG
step forward

Controller

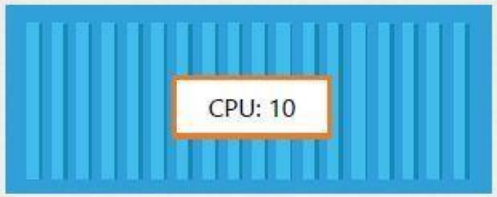


KOENIG
step forward

KUBE-SCHEDULER

This is one of the key components of Kubernetes master. It is a service in master responsible for distributing the workload. It is responsible for tracking utilization of working load on cluster nodes and then placing the workload on which resources are available and accept the workload. In other words, this is the mechanism responsible for allocating pods to available nodes. The scheduler is responsible for workload utilization and allocating pod to new node.

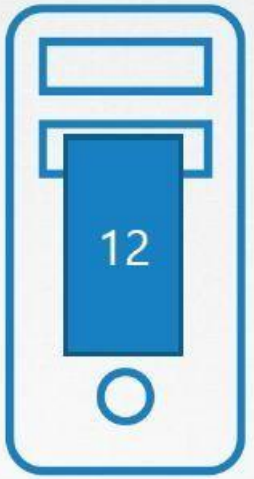
Kube-Scheduler



3rd - 12 - 10 = 2
4th - 16 - 10 = 6

1. Filter Nodes

2. Rank Nodes



KOENIG
step forward

CONTAINER RUNTIME

- The container runtime is the software that is responsible for running containers.

- Kubernetes supports several container runtimes: Docker, containerd, CRI-O, and any implementation of the Kubernetes CRI (Container Runtime Interface).

KOENIG
step forward

CONTAINER RUNTIME

containerd

rkt

frakti

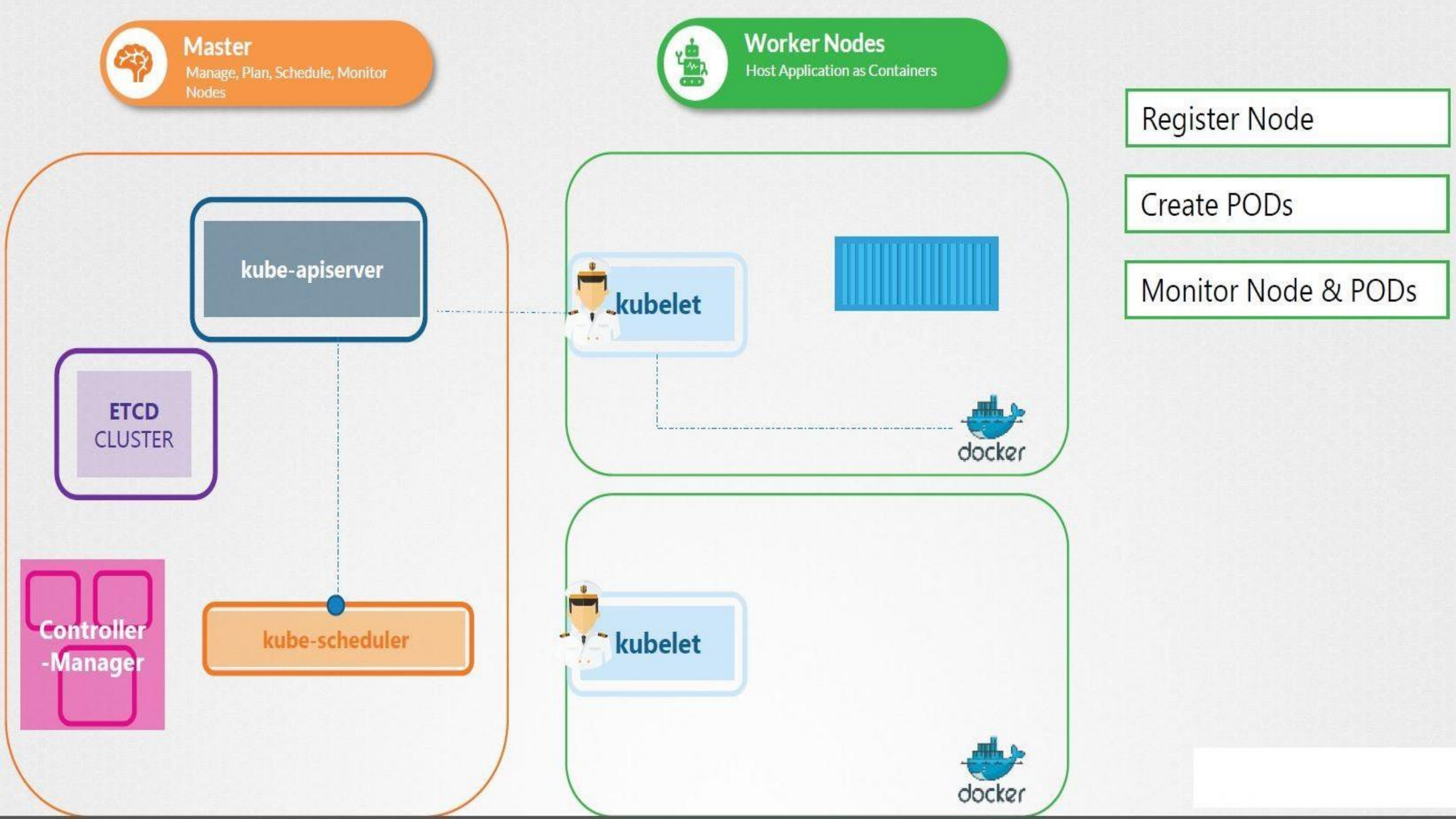
cri-o

KOENIG
step forward

KUBELET

It is the agent that runs on each node in the cluster. It is responsible for various tasks:

- making sure that the containers are running on the node as expected.
- for relaying information to and from control plane service through API server.
- It interacts with etcd store to read configuration details and wright values. This communicates with the master component to receive commands and work.
- The kubelet process then assumes responsibility for maintaining the state of work and the node server.
- It manages network rules, port forwarding, etc.

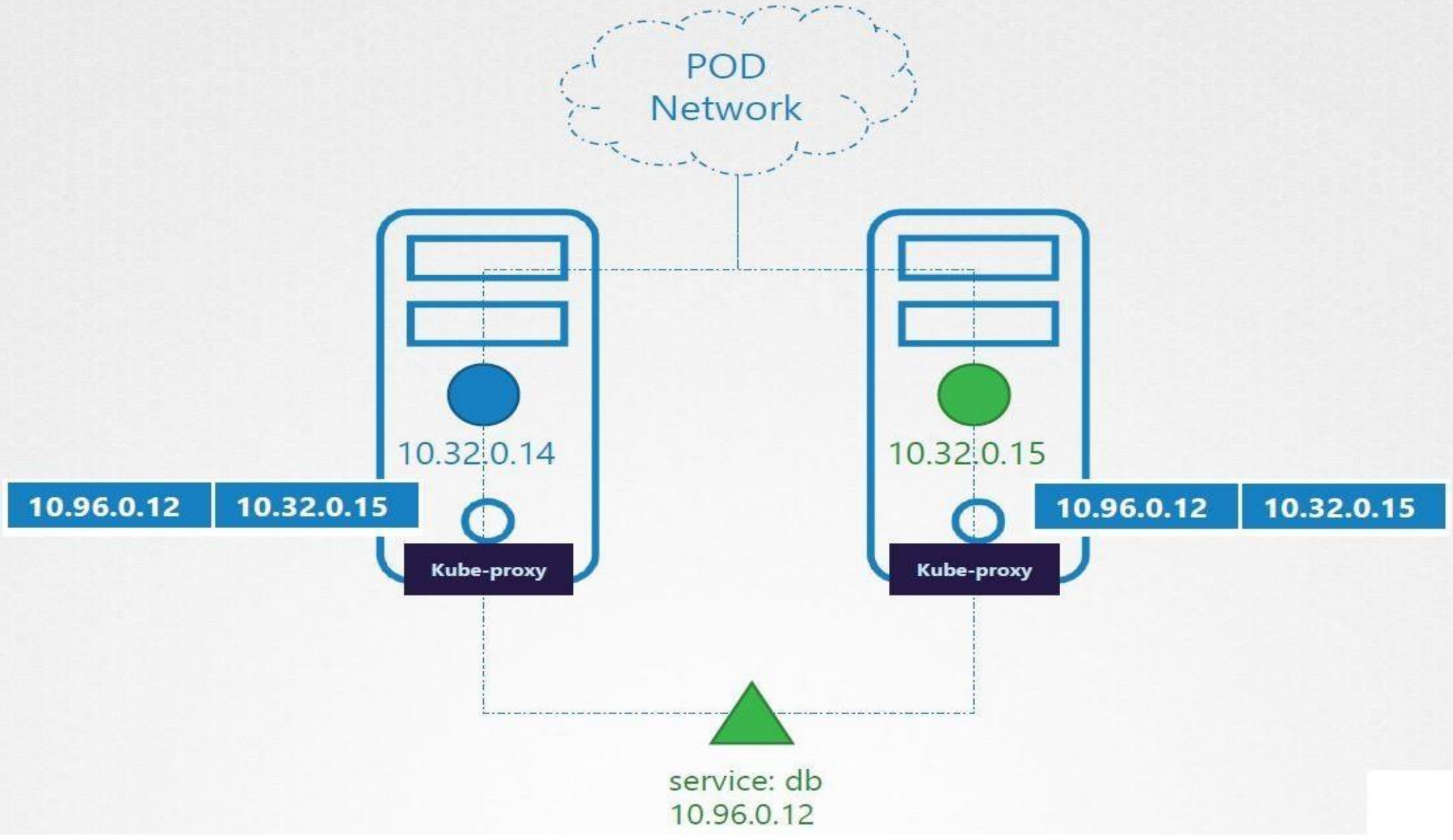

Master
Manage, Plan, Schedule, Monitor Nodes
Worker Nodes
Host Application as Containers
Register Node
Create PODs
Monitor Node & PODs
kube-apiserver
ETCD CLUSTER
Controller -Manager
kube-scheduler
kubelet
kubelet
docker
docker

KOENIG
step forward

KUBE-PROXY

- This is a proxy service which runs on each node and helps in making services available to the external host. It helps in forwarding the request to correct containers and is capable of performing primitive load balancing. It makes sure that the networking environment is predictable and accessible and at the same time it is isolated as well. It manages pods on node, volumes, secrets, creating new containers' health checkup, etc.

Kube-proxy


POD
Network
10.32.0.14
10.32.0.15
10.96.0.12
10.32.0.15
10.96.0.12
10.32.0.15
Kube-proxy
Kube-proxy
service: db
10.96.0.12

KOENIG
step forward

KUBERNETES INSTALLATION

KOENIG
step forward

REQUIREMENTES

One or more machines running one of:

- Ubuntu 16.04+
- Debian 9+
- CentOS 7
- Red Hat Enterprise Linux (RHEL) 7
- Fedora 25+
- HypriotOS v1.0.1+
- Container Linux (tested with 1800.6.0)

KOENIG
step forward

CHECK REQUIRED PORTS

Control-plane node(s)

Protocol	Direction	Port Range	Purpose	Used By
TCP	Inbound	6443*	Kubernetes API server	All
TCP	Inbound	2379-2380	etcd server client API	kube-apiserver, etcd
TCP	Inbound	10250	Kubelet API	Self, Control plane
TCP	Inbound	10251	kube-scheduler	Self
TCP	Inbound	10252	kube-controller-manager	Self

Worker node(s)

Protocol	Direction	Port Range	Purpose	Used By
TCP	Inbound	10250	Kubelet API	Self, Control plane
TCP	Inbound	30000-32767	NodePort Services†	All

KOENIG
step forward

CONFIGURATION

ON ALL NODES

- Disable SWAP

#vim /etc/fstab

then remove the entry of swap or comment

#swapoff -a -> to disable swap

•Set selinux to permissive mode

#vim /etc/sysconfig/selinux

SELINUX=permissive

#setenforce 0

•Stop firewall service

#systemctl stop firewalld

#systemctl disable firewalld

KOENIG
step forward

INSTALL CONTAINER RUNTIME

ON ALL NODES

```
wget -O /etc/yum.repos.d/docker.repo https://download.docker.com/linux/centos/docker-ce.repo
yum install docker -y
systemctl start docker
systemctl enable docker
```

KOENIG
step forward

Installing kubeadm, kubelet and kubectl

- **kubeadm:** the command to bootstrap the cluster.
- **kubelet:** the component that runs on all of the machines in your cluster and does things like starting pods and containers.
- **kubectl:** the command line util to talk to your cluster.

DEBIAN BASED

- #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
- #apt-get update
- #apt-get install -y kubelet kubeadm kubectl
- #apt-mark hold kubelet kubeadm kubectl

KOENIG
step forward

RPM BASED

ON ALL NODES

#vim /etc/yum.repos.d/kubernetes.repo

```
[kube]
name=kube.repo
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/r
pm-package-key.gpg
```

#yum install kubelet kubectl kubeadm –y
#systemctl start kubelet
#systemctl enable kubelet

KOENIG
step forward

INITIALIZE THE CLUSTER

- Now initialize the master node->

#kubeadm init --pod-network-cidr=10.244.0.0/16

- Then after execution of above command you will have some commands so do all showing commands to configure kube configuration

- Run the join command in every worker node

KOENIG
step forward

VERIFY INSTALLATION

```
[root@master ~]# kubectl get nodes
NAME       STATUS     ROLES    AGE   VERSION
master     NotReady   master   39s   v1.19.2
worker1    NotReady   <none>   10s   v1.19.2
worker2    NotReady   <none>   6s    v1.19.2
[root@master ~]#
```

KOENIG
step forward

NETWORK SOLUTION

- we are using calico network for the installation

#kubectl apply -f https://docs.projectcalico.org/v3.14/manifests/calico.yaml

```
[root@master ~]# kubectl get nodes
NAME      STATUS   ROLES    AGE     VERSION
master    Ready    master   2m27s   v1.19.2
worker1   Ready    <none>   118s    v1.19.2
worker2   Ready    <none>   114s    v1.19.2
[root@master ~]#
```

KOENIG
step forward

BASH COMPLETION

#yum install bash-completion -y

#cd ~/.kube

#kubectl completion bash > kube.sh

#source kube.sh

To make it permanent:-

#vim $HOME/.bashrc

Add a entry

source $HOME/.kube/kube.sh

KOENIG
step forward

NODES

A node is a worker machine in Kubernetes, previously known as a minion. A node may be a VM or physical machine, depending on the cluster. Each node contains the services necessary to run pods and is managed by the master components.

KOENIG
step forward

NODE STATUS

```
kubectl describe node <insert-node-name-here>
```

A node's status contains the following information:

- Addresses
- Conditions
- Capacity and Allocatable
- Info

KOENIG
step forward

NODES STATE

Node Condition	Description
`Ready`	`True` if the node is healthy and ready to accept pods, `False` if the node is not healthy and is not accepting pods, and `Unknown` if the node controller has not heard from the node in the last `node-monitor-grace-period` (default is 40 seconds)
`MemoryPressure`	`True` if pressure exists on the node memory – that is, if the node memory is low; otherwise `False`
`PIDPressure`	`True` if pressure exists on the processes – that is, if there are too many processes on the node; otherwise `False`
`DiskPressure`	`True` if pressure exists on the disk size – that is, if the disk capacity is low; otherwise `False`
`NetworkUnavailable`	`True` if the network for the node is not correctly configured, otherwise `False`

KOENIG
step forward

KUBECTL

#kubectl run <name> --image=<name>

#kubectl cluster-info

#kubectl get nodes

#kubectl describe node <insert-node-name-here>

KOENIG
step forward

PODS

- A pod is a collection of containers and its storage inside a node of a Kubernetes cluster. It is possible to create a pod with multiple containers inside it. For example, keeping a database container and data container in the same pod.

- There are two types of Pods –
  - Single container pod
  - Multi container pod

KOENIG
step forward

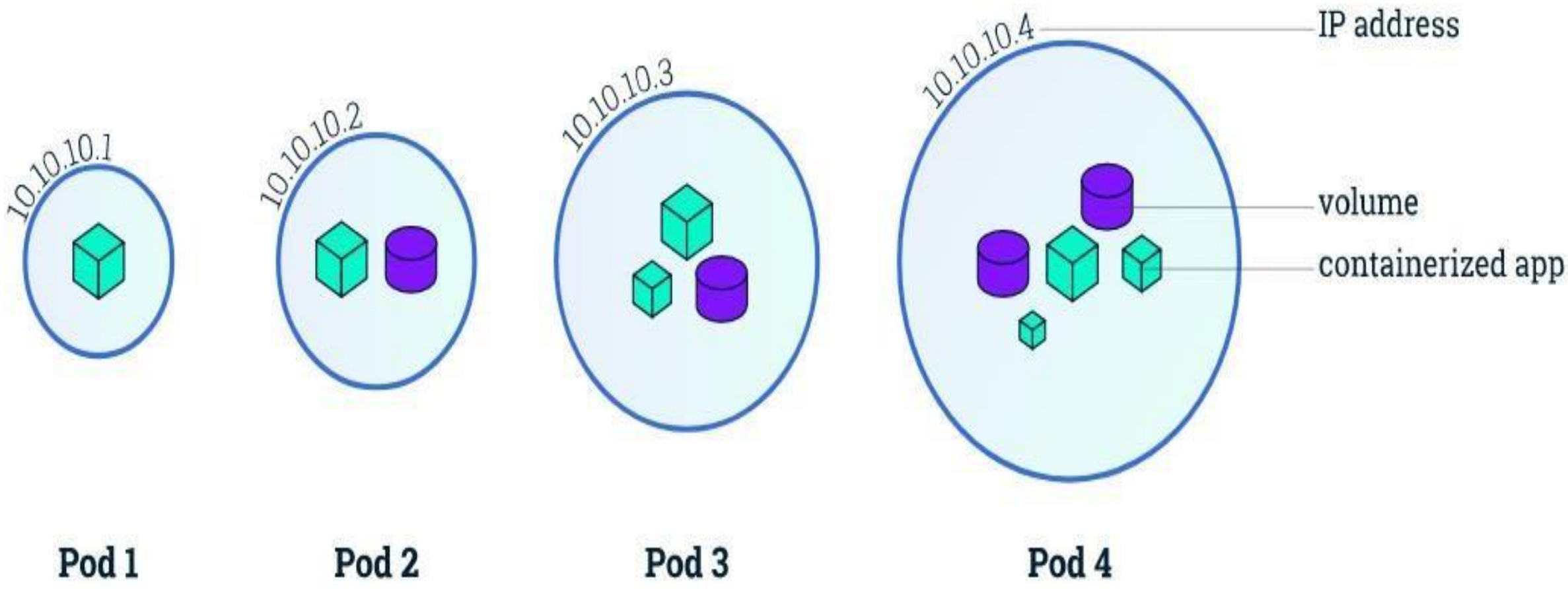
PODS

- Pods provide two kinds of shared resources for their constituent containers: networking and storage.

1)Networking-> Each Pod is assigned a unique IP address. Every container in a Pod shares the network namespace, including the IP address and network ports.

2)Storage -> A Pod can specify a set of shared storage Volumes. All containers in the Pod can access the shared volumes, allowing those containers to share data.

KOENIG
step forward



KOENIG
step forward

HOW TO CREATE PODS

#kubectl run <podname> --image=nginx

#kubectl get pods

#kubectl get pods -A

#kubectl get pods -o wide

#kubectl describe pods <podname>

#kubectl explain <resource> like pod,deployment

#kubectl exec -it <podname> -- bash

KOENIG
step forward

YAML INTRODUCTION

- Four fields are required to create yaml file->

1)apiVersion - Which version of the Kubernetes API you're using to create this object

2) kind - What kind of object you want to create

3)metadata - Data that helps uniquely identify the object, including a name string, UID, and optional namespace

4) spec - What state you desire for the object

apiVersion

apiVersion: version of api depend on the Resource.

kind	Version
Pod	v1
Service	v1
ReplicaSet	apps/v1
Deployment	apps/v1

To check API Version:-

#kubectl explain <resourcename>

KOENIG
step forward

HOW TO CREATE POD

#vim pod.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: my-first-pod
spec:
 containers:
 - name: mycontainer
   image: quay.io/app-sre/nginx
   ports:
   - containerPort: 80
```

#kubectl create -f pod.yml
#kubectl create -f pod.yml --dry-run=client
#kubectl delete -f pod.yml

KOENIG
step forward

LABELS AND SELECTORS

Labels :

- Labels enable end users to map their own, custom, organizational structures onto system resources in a loosely coupled fashion.

Selectors :

- Labels do not provide uniqueness. In general, we can say many objects can carry the same labels. Labels selector are core grouping primitive in Kubernetes. They are used by the users to select a set of objects or to filter the tags.

KOENIG
step forward

EXAMLE OF LABEL

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: mypod
 labels:
  env: development
spec:
 containers:
 - name: mycontainer
   image: quay.io/app-sre/nginx
   ports:
   - containerPort: 80
```

KOENIG
step forward

LABELS COMMANDS

#kubectl apply -f <filename>

#kubectl get pods

#kubectl get pods <podname>--show-labels

#kubectl label pod <podname> tier=backend

#kubectl get pod <podname> --show-labels

#kubectl label pod <podname> tier-

#kubectl label --overwrite pods <podname> env=prod

KOENIG
step forward

What is Selector?

Selector is used to filter the labels.

Two types of Selectors:

- Equity-Based Selector
- Set-Based Selector

```
[root@master kube]# kubectl get pods --show-labels
NAME     READY   STATUS    RESTARTS   AGE     LABELS
mypod    1/1     Running   0          7m56s   env=development
mypod2   1/1     Running   0          7m35s   env=production
mypod3   1/1     Running   0          7m2s    env=marketting
[root@master kube]#
```

KOENIG
step forward

Equity-Based Selector

```
[root@master kube]# kubectl get pods --selector env=marketting
NAME     READY   STATUS    RESTARTS   AGE
mypod3   1/1     Running   0          83s
[root@master kube]# kubectl get pods --selector env!=marketting
NAME     READY   STATUS    RESTARTS   AGE
mypod    1/1     Running   0          2m29s
mypod2   1/1     Running   0          2m8s
[root@master kube]#
```

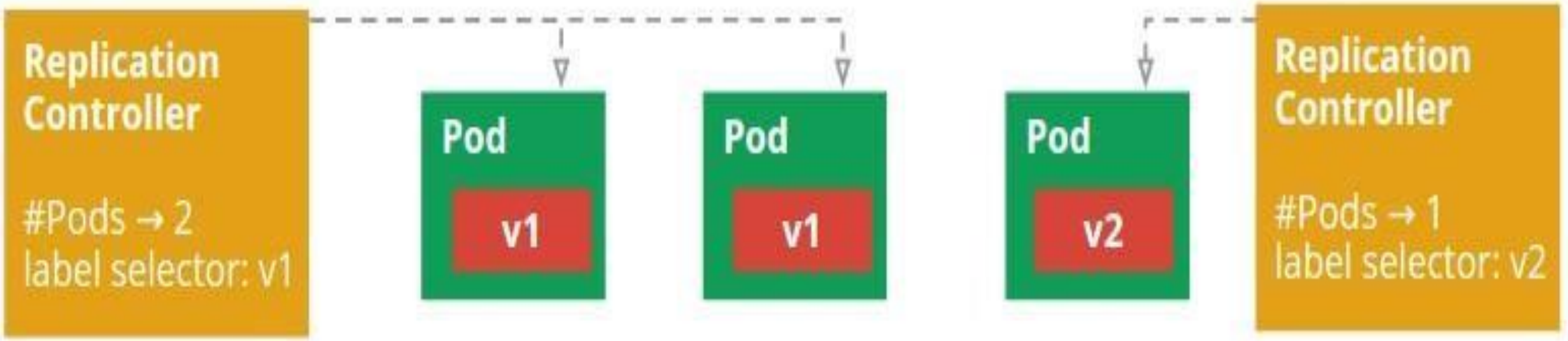
KOENIG
step forward

Set-Based Selector

```
[root@master kube]# kubectl get pods --selector 'env in (development,marketting)'
NAME     READY   STATUS    RESTARTS   AGE
mypod    1/1     Running   0          3m42s
mypod3   1/1     Running   0          2m48s
[root@master kube]#
```

KOENIG step forward

netes Replication Controllers



Behavior		Benefits
● Keeps Pods running	→	Restarts Pods, desired state
● Gives direct control of Pod #s	→	Fine-grained control for scaling

replication_controller.yml

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
 name: rc-web
spec:
 replicas: 3
 template:
  metadata:
   name: web
   labels:
    app: nginx
  spec:
   containers:
   - name: web
     image: quay.io/app-sre/nginx
```

Pod definition

```
#kubectl create -f replica_controller.yml

#kubectl get replicationcontroller
                    or
#kubectl get rc
#kubectl get pods
#kubectl get pods -o wide
#kubectl get pods -A --show-labels
```

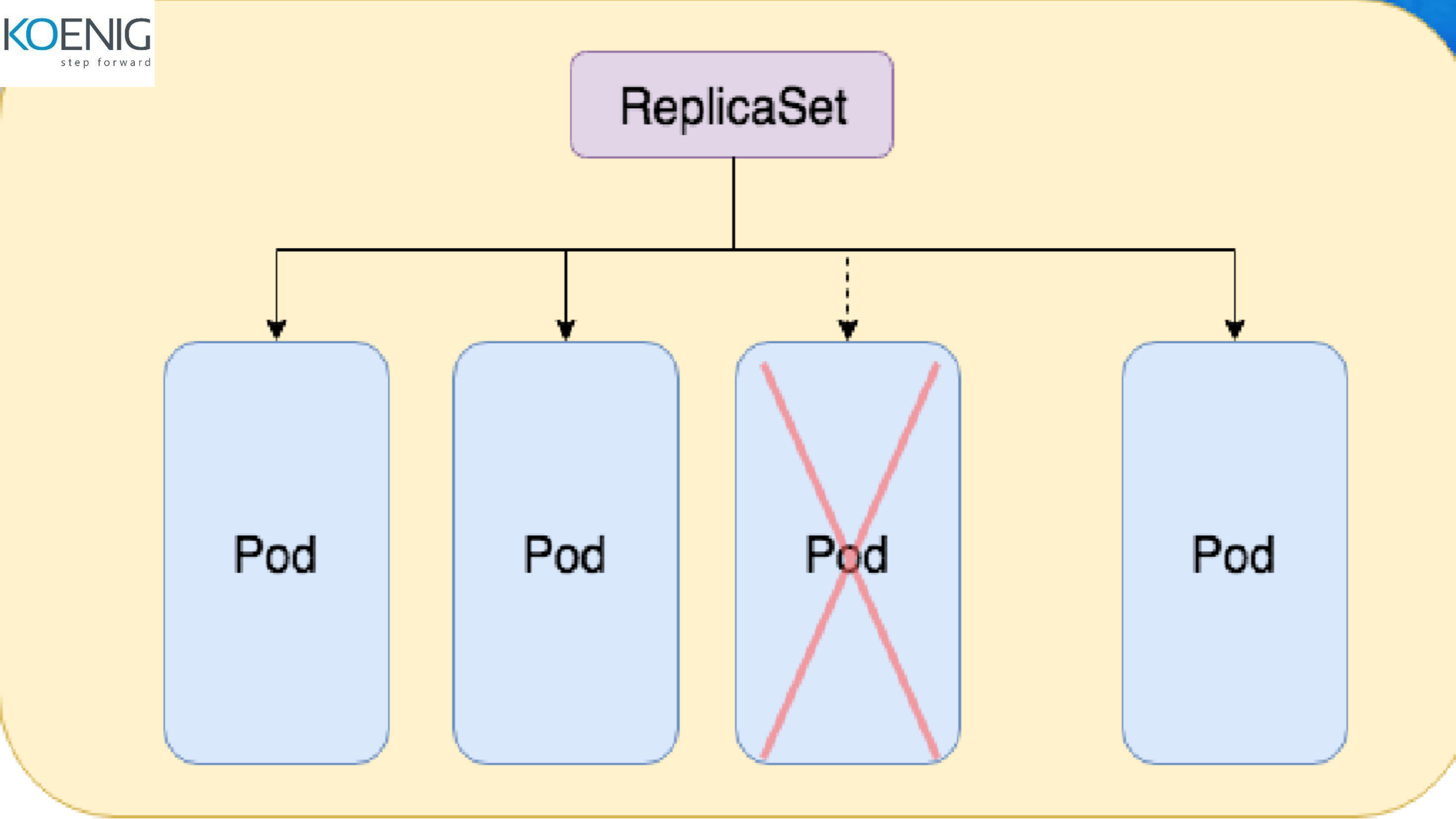
KOENIG
step forward

ReplicaSet

- A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time.As such, it is often used to guarantee the availability of a specified number of identical Pods.

- How ReplicaSet Works?

A ReplicaSet is defined with fields, including a selector that specifies how to identify Pods it can acquire, a number of replicas indicating how many Pods it should be maintaining, and a pod template specifying the data of new Pods it should create to meet the number of replicas criteria. A ReplicaSet then fulfills its purpose by creating and deleting Pods as needed to reach the desired number. When a ReplicaSet needs to create new Pods, it uses its Pod template.


KOENIG
step forward
ReplicaSet
Pod
Pod
Pod
Pod

KOENIG
step forward

replica_set.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: rs-web
spec:
 replicas: 3
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   name: webserver
   labels:
    app: nginx
  spec:
   containers:
   - name: webcontainer
     image: quay.io/app-sre/nginx
     ports:
     - containerPort: 80
```

KOENIG
step forward

replica_set.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: rs-web
spec:
 replicas: 3
 selector:
  matchExpressions:
  - key: app
    operator: In
    values:
    - app1
    - app2
 template:
  metadata:
   name: webserver
   labels:
    app: app1
  spec:
   containers:
   - name: webcontainer
     image: quay.io/app-sre/nginx
```

KOENIG
step forward

Replication Controller VS Replica Set

- Selector is optional in Replication Controller, but it’s mandatory in Replica Set.
- Set-Based selector is not supported by Replication Controller, but it’s supported by Replica Set.

KOENIG
step forward

SERVICE

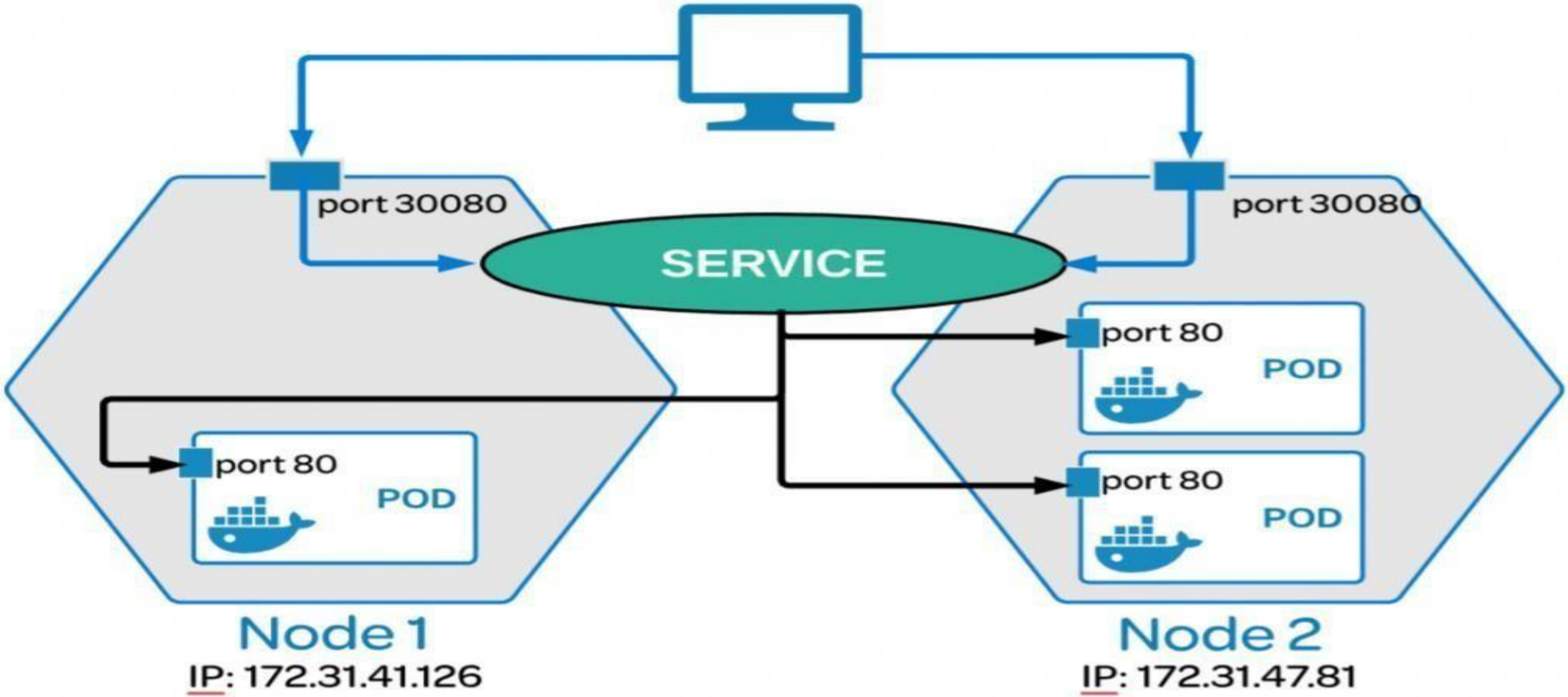
- Services are resources in Kubernetes (like pods, deployments, etc.) that provide a single point of access from the outside world, into your pod(s) which run your application.

- Each service has an IP address and a port that never change, and the client always knows where to find the pods due to the flat nat-less network.

KOENIG
step forward

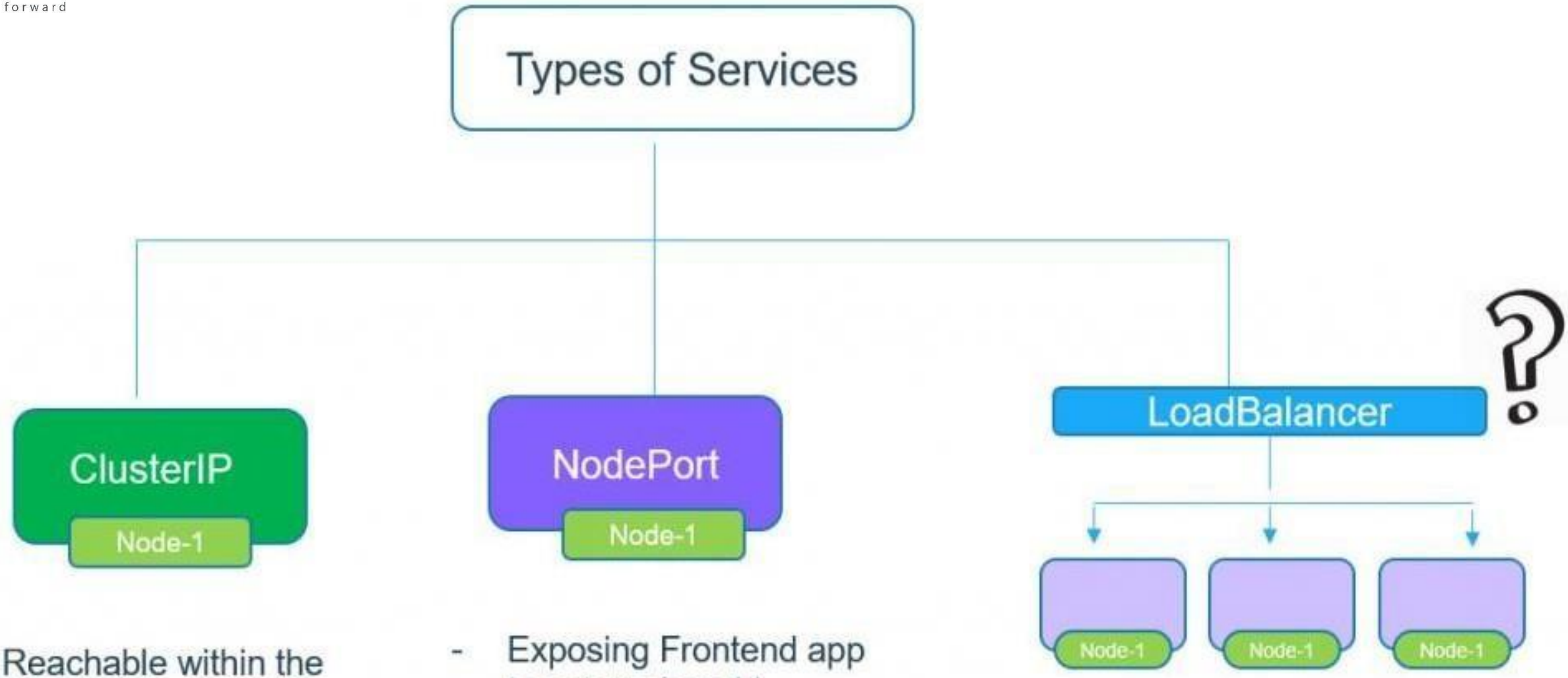
Kubernetes Service

A service allows you to dynamically access a group of replica pods.



KOENIG
step forward

Types of Services



- Reachable within the cluster.
- Connects Frontend Pods to Backend Pods

- Exposing Frontend app to external world

- Equally distribute the loads

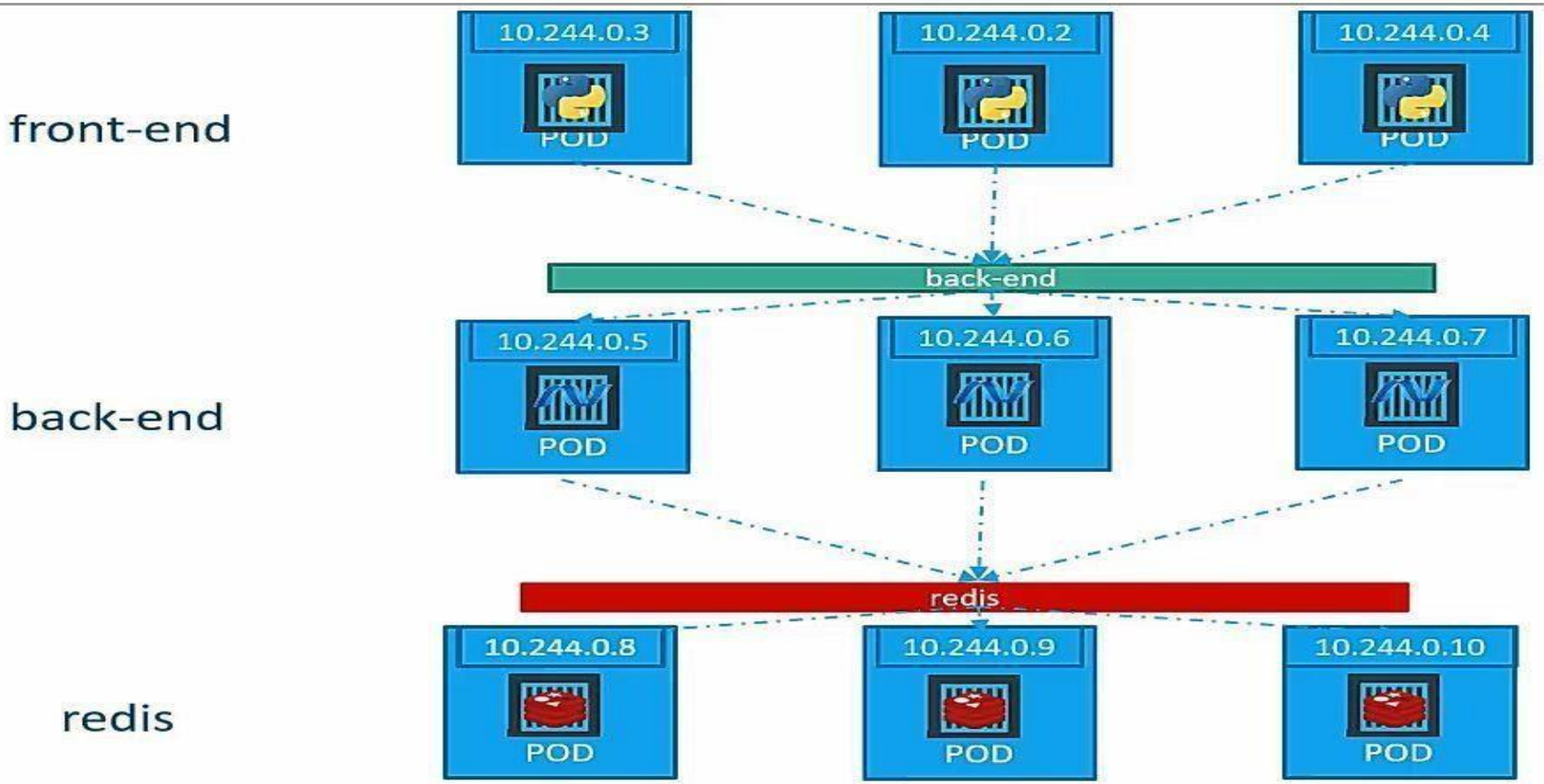
KOENIG
step forward

CLUSTER IP

- ClusterIP: Exposes the Service on a cluster-internal IP. Choosing this value makes the Service only reachable from within the cluster. This is the default ServiceType.

KOENIG
step forward

ClusterIP



KOENIG
step forward

APP DEPLOYMENT

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: rs-web
spec:
 replicas: 3
 selector:
  matchLabels:
   app: web
 template:
  metadata:
   name: webserver
   labels:
    app: web
  spec:
   containers:
   - name: webcontainer
     image: quay.io/gauravkumar9130/nginxdemo
     ports:
     - containerPort: 80
```

KOENIG
step forward

CLUSTER IP SERVICE

```yaml
apiVersion: v1
kind: Service
metadata:
 name: my-web-service
spec:
 type: ClusterIP
 ports:
 - targetPort: 80  #container port number
   port: 8081      #bind port number with cluster ip
 selector:
  app: web
```

KOENIG
step forward

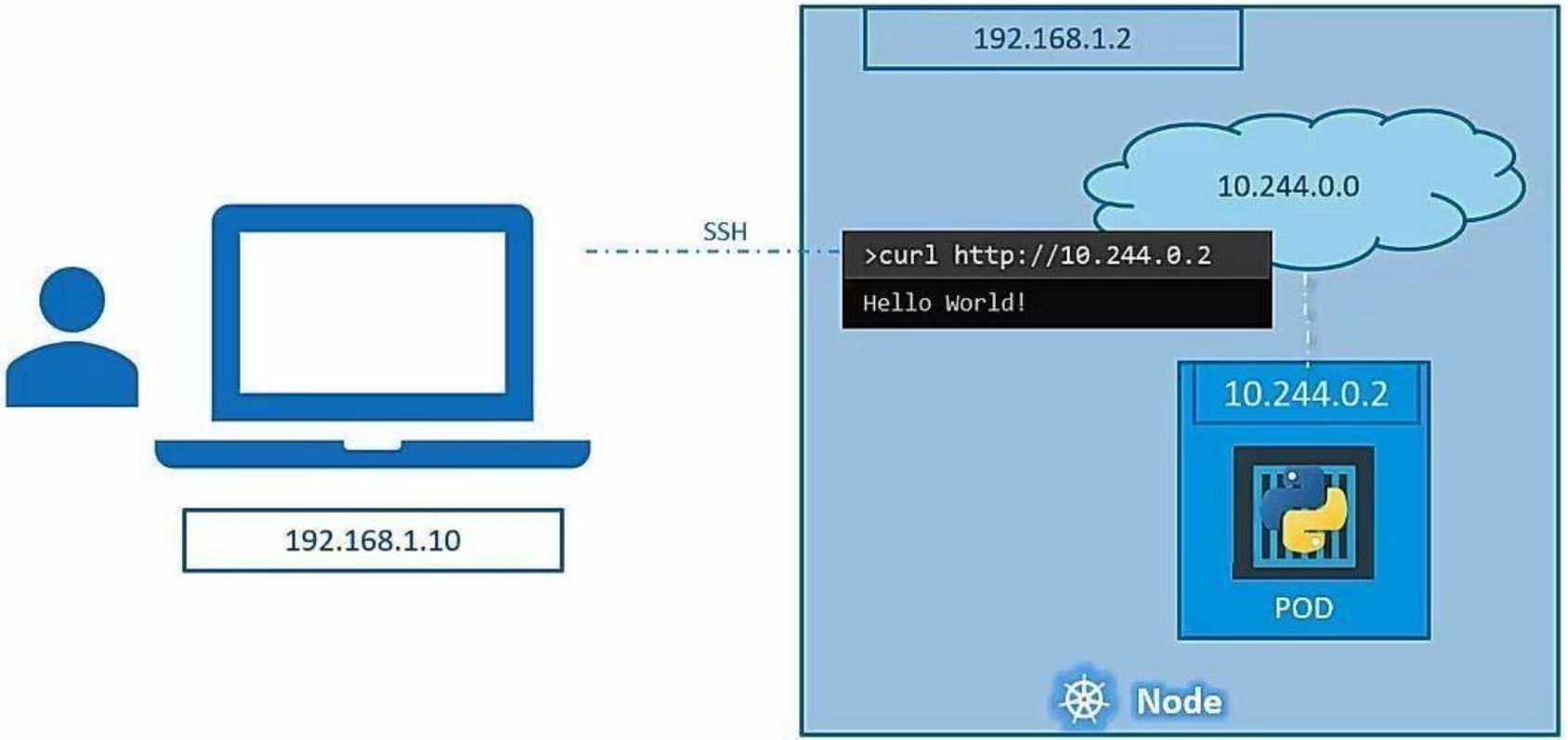
Access Application

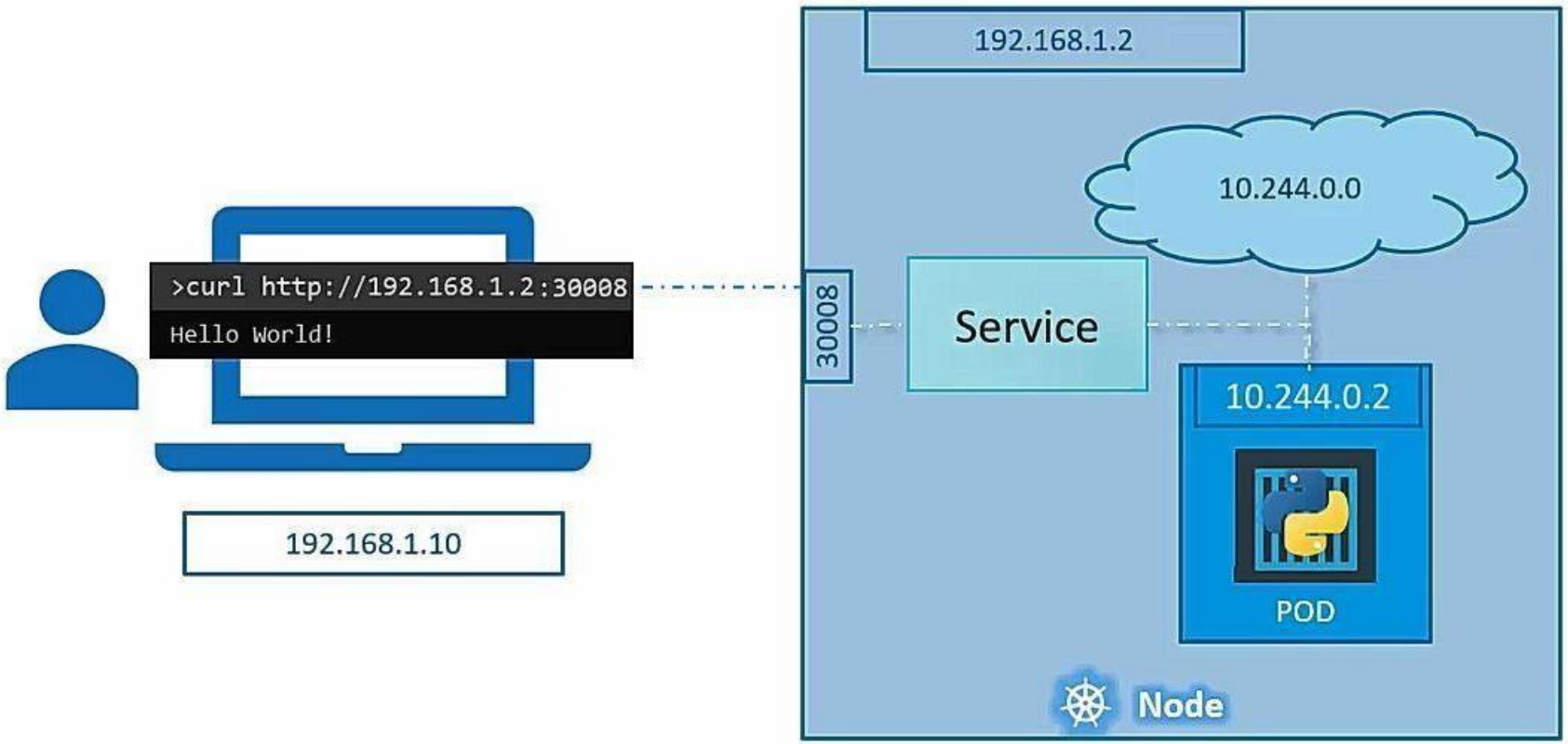
```
[root@master ~]# kubectl get svc
NAME             TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
kubernetes       ClusterIP   10.96.0.1        <none>        443/TCP    56m
my-web-service   ClusterIP   10.111.137.181   <none>        8081/TCP   70s
[root@master ~]# curl 10.111.137.181:8081
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

KOENIG
step forward

NODE PORT

- NodePort: Exposes the Service on each Node’s IP at a static port (the NodePort). A ClusterIP Service, to which the NodePort Service routes, is automatically created. You’ll be able to contact the NodePort Service, from outside the cluster, by requesting <NodeIP>:<NodePort>.


192.168.1.2
10.244.0.0
SSH
>curl http://10.244.0.2
Hello World!
10.244.0.2
POD
Node
192.168.1.10


>curl http://192.168.1.2:30008
Hello World!
192.168.1.10
192.168.1.2
10.244.0.0
30008
Service
10.244.0.2
POD
Node

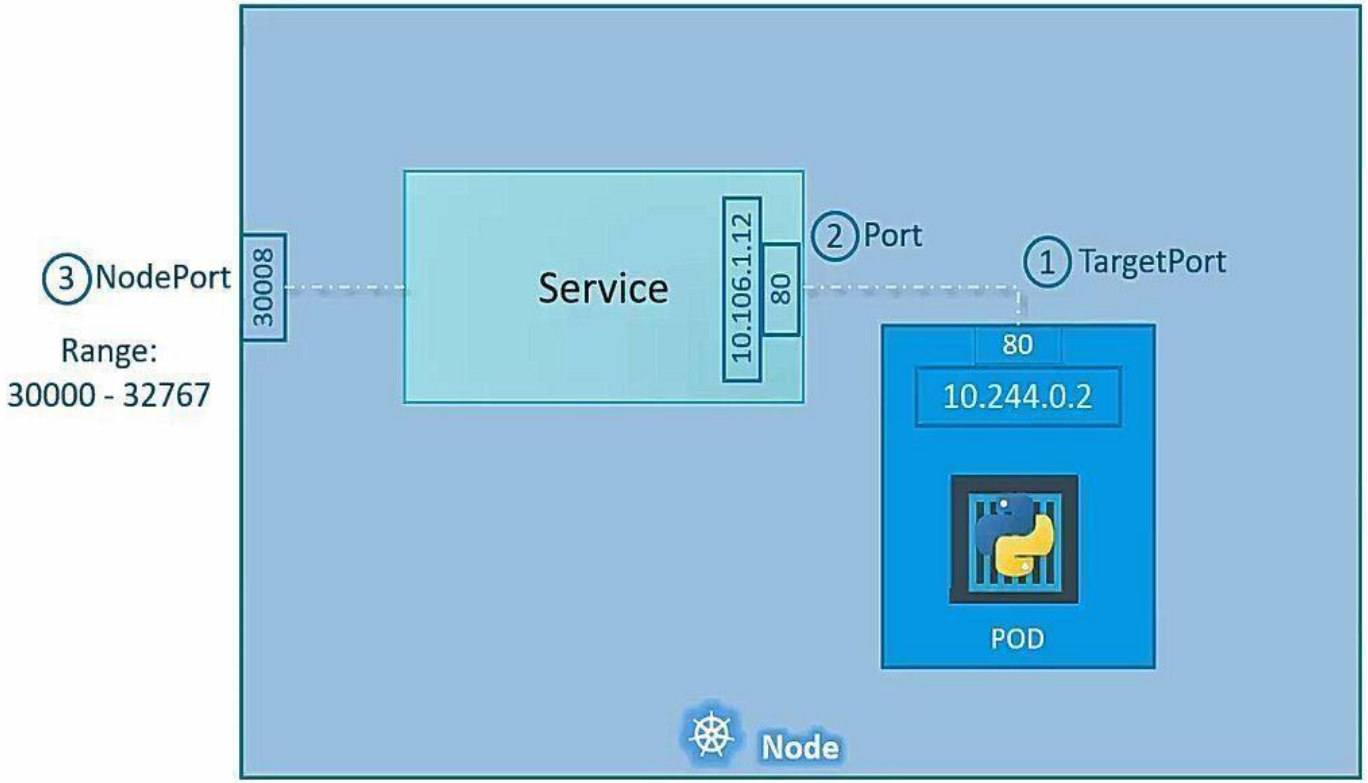
KOENIG
step forward

APP DEPLOYMENT

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: rs-web
spec:
 replicas: 3
 selector:
  matchLabels:
   app: web
 template:
  metadata:
   name: webserver
   labels:
    app: web
  spec:
   containers:
   - name: webcontainer
     image: quay.io/gauravkumar9130/nginxdemo
     ports:
     - containerPort: 80
```

KOENIG
step forward

Service - NodePort




```yaml
apiVersion: v1
kind: Service
metadata:
 name: web-outside-service
spec:
 type: NodePort
 ports:
 - targetPort: 80
   port: 80
   nodePort: 30008
 selector:
  app: web
```

KOENIG
step forward

LOAD BALANCER

- LoadBalancer: Exposes the Service externally using a cloud provider’s load balancer. NodePort and ClusterIP Services, to which the external load balancer routes, are automatically created.

KOENIG
step forward

MANUAL SCHEDULING

KOENIG
step forward

SCHEDULER

A scheduler watches for newly created Pods that have no Node assigned. For every Pod that the scheduler discovers, the scheduler becomes responsible for finding the best Node for that Pod to run on. The scheduler reaches this placement decision taking into account the scheduling principles described below.

KOENIG
step forward

SCHEDULING WITH KUBE SCHEDULER

kube-scheduler selects a node for the pod in a 2-step operation:

- Filtering
- Scoring

The filtering step finds the set of Nodes where it's feasible to schedule the Pod. For example, the PodFitsResources filter checks whether a candidate Node has enough available resource to meet a Pod's specific resource requests. After this step, the node list contains any suitable Nodes; often, there will be more than one. If the list is empty, that Pod isn't (yet) schedulable.

In the scoring step, the scheduler ranks the remaining nodes to choose the most suitable Pod placement. The scheduler assigns a score to each Node that survived filtering, basing this score on the active scoring rules.

Finally, kube-scheduler assigns the Pod to the Node with the highest ranking. If there is more than one node with equal scores, kube-scheduler selects one of these at random.

KOENIG
step forward

MANUAL SCHEDULING

```
apiVersion: v1
kind: Pod
metadata:
 name: worker1-pod
 labels:
  app: nginx
spec:
 containers:
 - name: nginx
   image: nginx
   ports:
   - containerPort: 80
 nodeName: worker1
~
```

KOENIG
step forward

```
[root@master ~]# kubectl get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
worker1-pod   1/1     Running   0          26s   10.244.235.142   worker1   <none>           <none>
```

KOENIG
step forward

TAINTS AND TOLERATIONS

- Taints and tolerations work together to ensure that pods are not scheduled onto inappropriate nodes. One or more taints are applied to a node; this marks that the node should not accept any pods that do not tolerate the taints. Tolerations are applied to pods, and allow (but do not require) the pods to schedule onto nodes with matching taints.

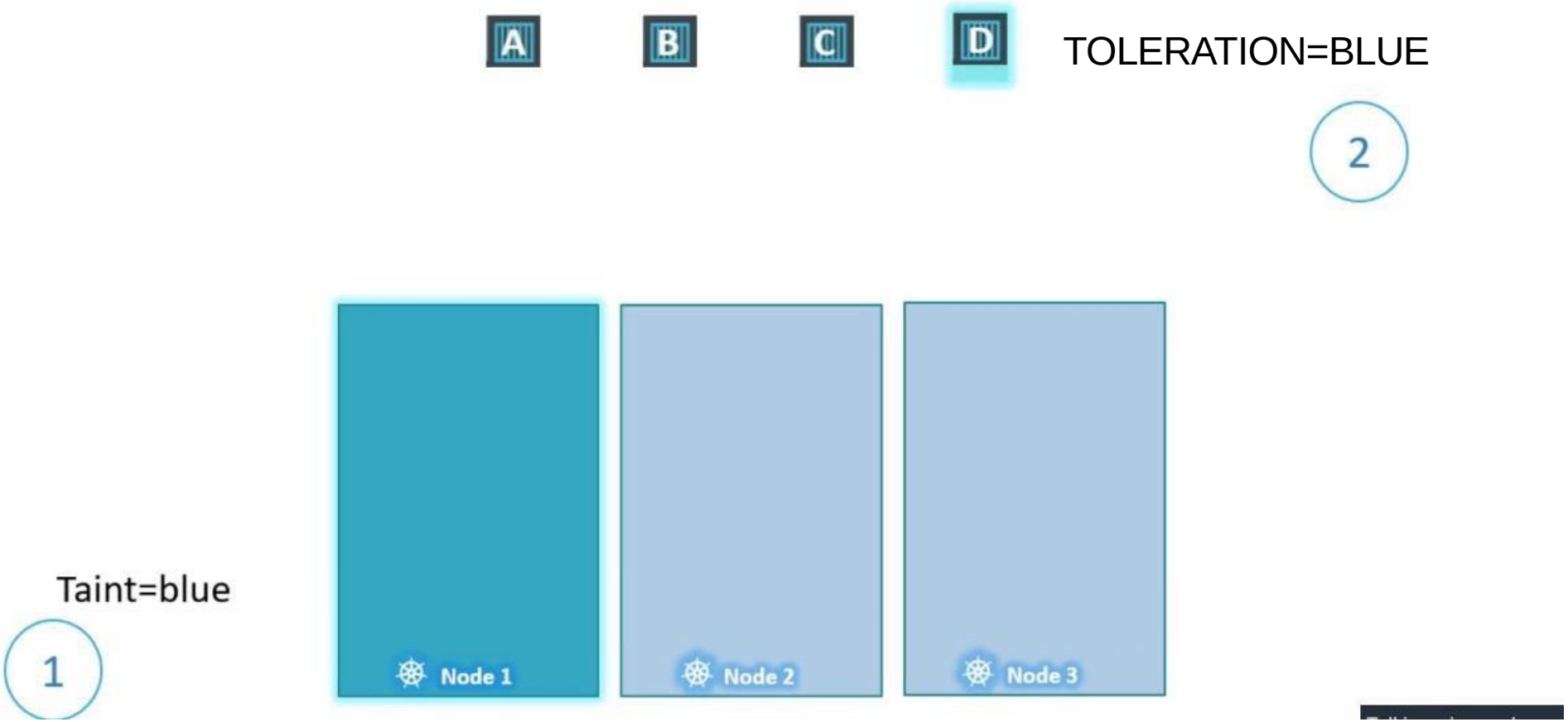
Use Case:-

1) Dedicated Nodes

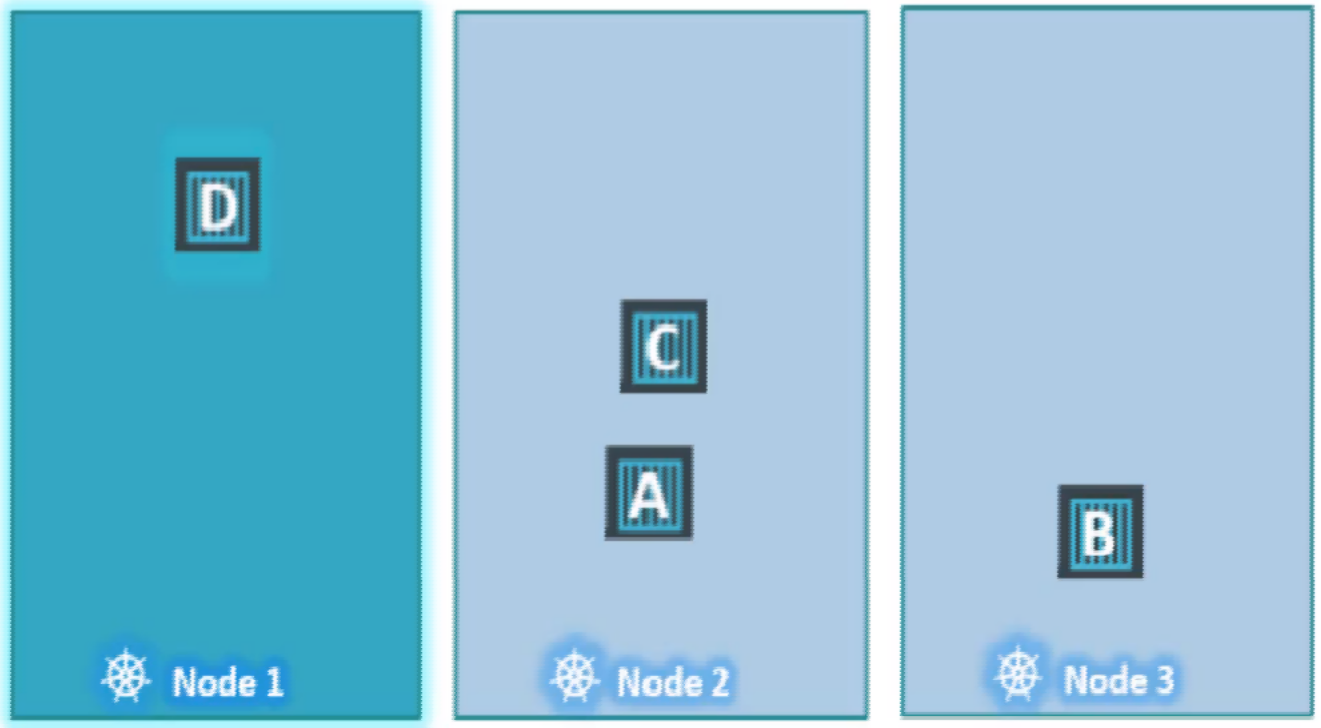
2) Nodes with special hardware

taint will be on node level.

toleration will be on pod level.

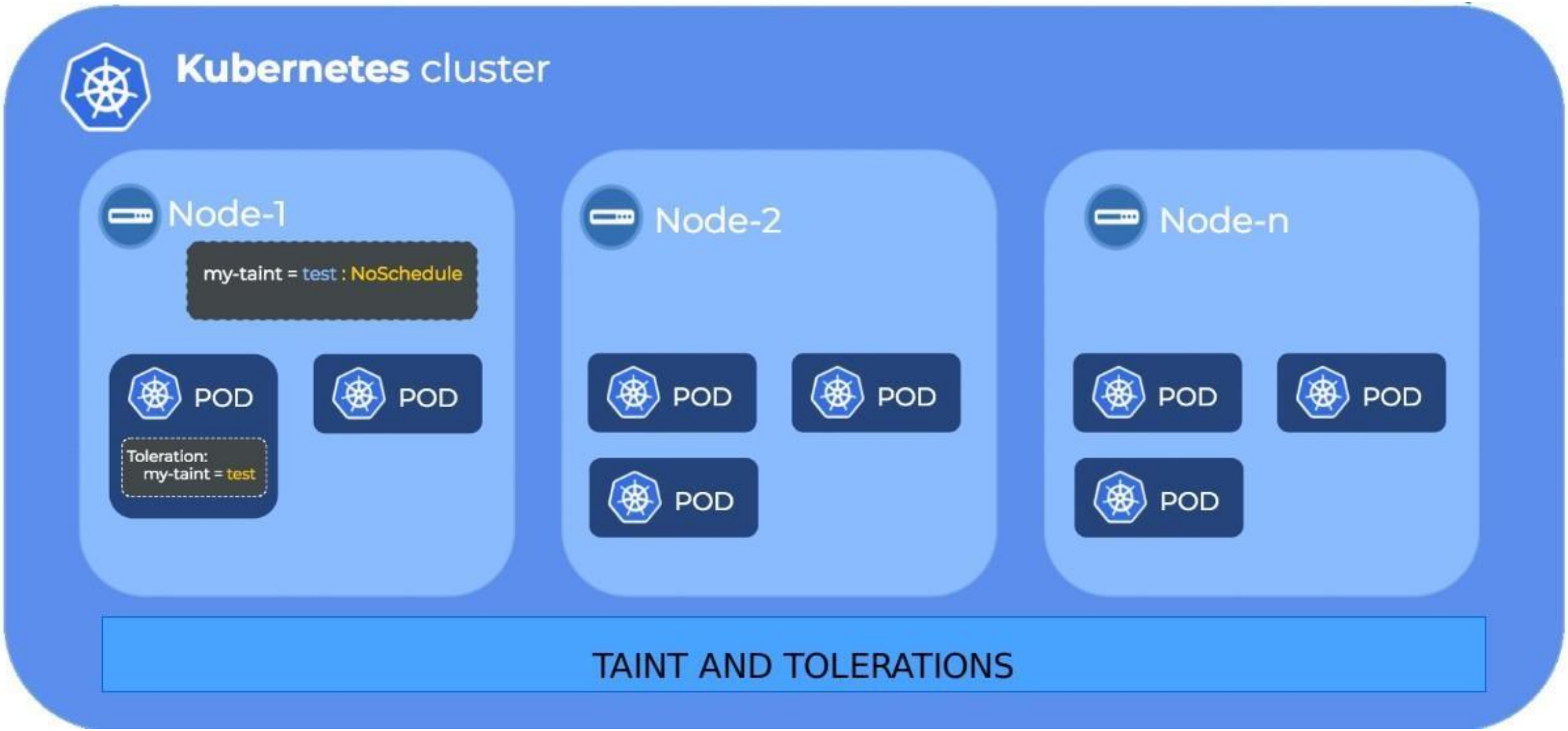

A
B
C
D
TOLERATION=BLUE
2
Taint=blue
1
Node 1
Node 2
Node 3

Taint=blue



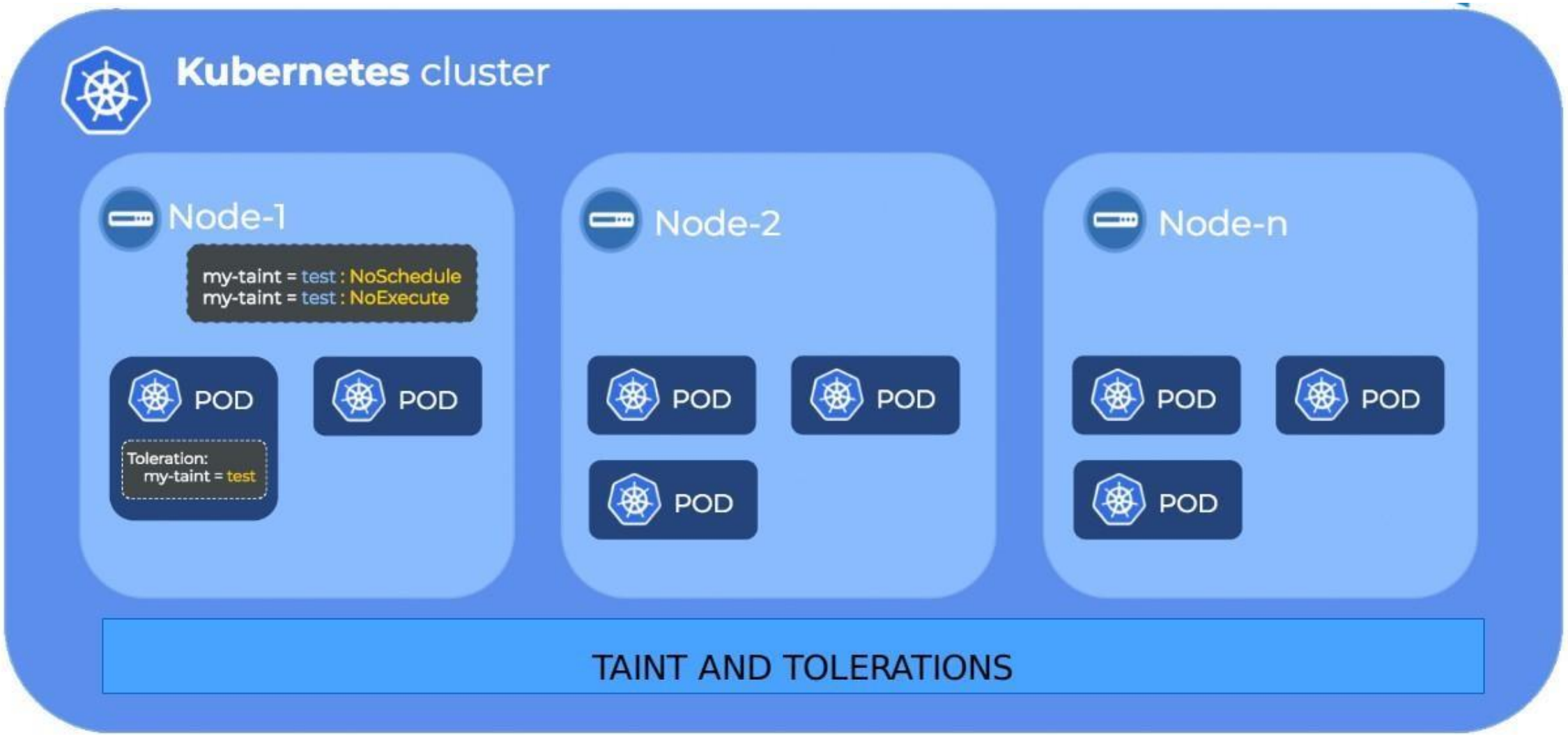
KOENIG
step forward

TAINT EFFECT:NOSchedule



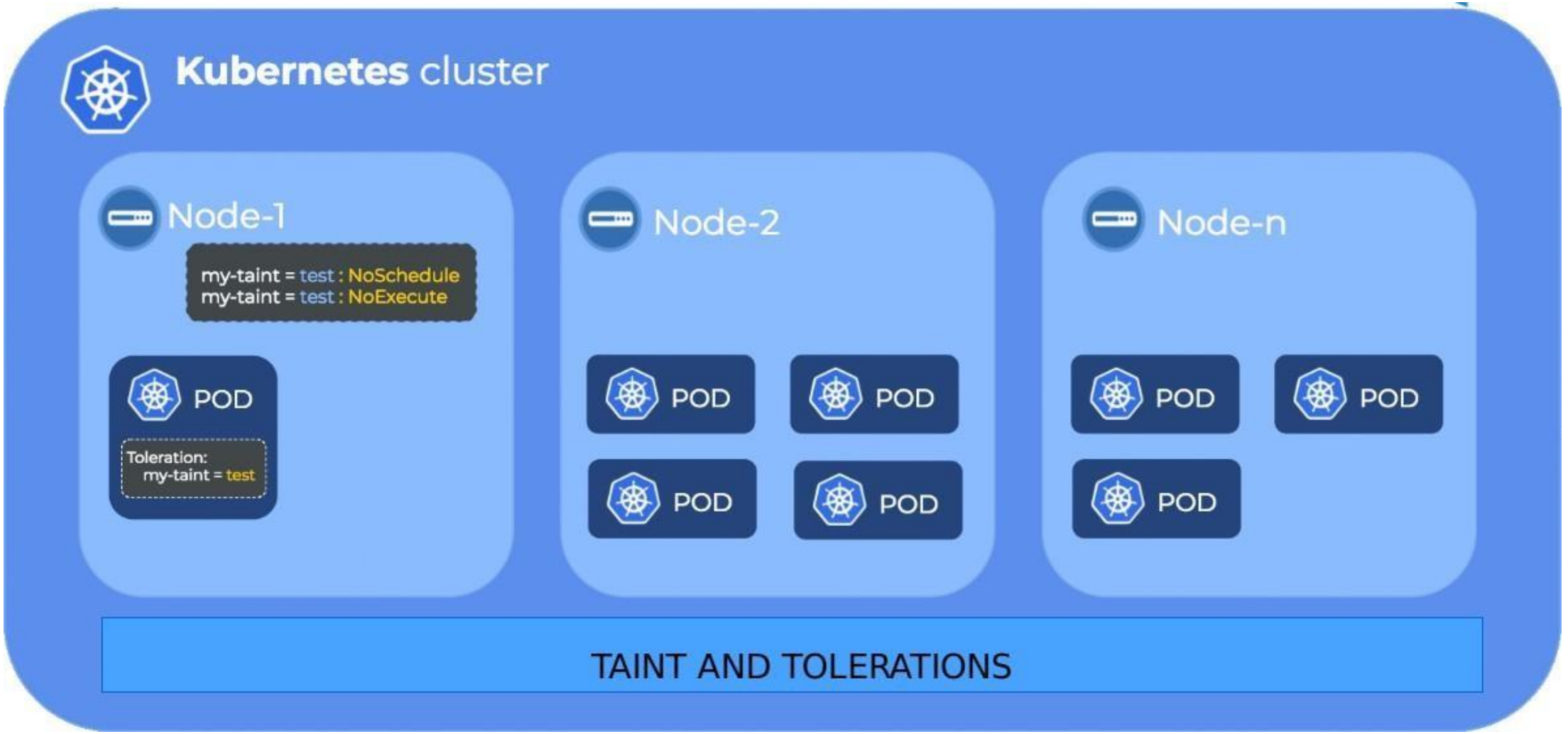
KOENIG
step forward

TAINT EFFECT:NoExecute



KOENIG
step forward

TAINT EFFECT:NoExecute



Taint on Node

#kubectl taint nodes node1 app=blue:NoSchedule
(effects-noschedule, noexecute)

KOENIG
step forward

TOLERATIONS ON POD

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: mypod
5  spec:
6    containers:
7    - name: nginx-container
8      image: nginx
9
10   tolerations:
11   - key: "app"
12     operator: "Equal"
13     value: "blue"
14     effect: "No Schedule"
15
16
```

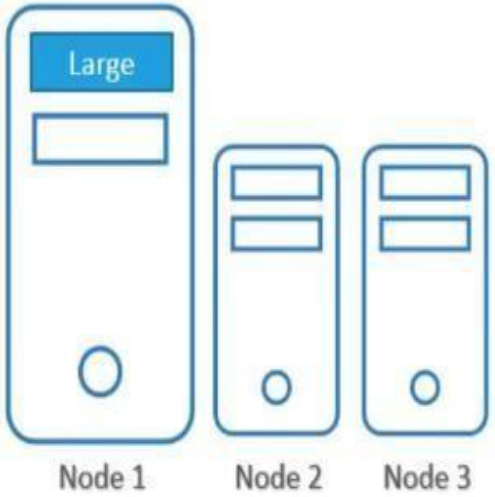
KOENIG
step forward

NODE SELECTOR (LABEL ON NODES)

Label Nodes

```
kubectl label nodes <node-name> <label-key>=<label-value>
```

```
kubectl label nodes node-1 size=Large
```



KOENIG
step forward

NODE SELECTOR(LABEL ON PODS)

```
apiVersion: v1
kind: Pod
metadata:
 name: mypod
spec:
 containers:
 - name: large-pod
   image: nginx
 nodeSelector:
  size: large
```

KOENIG
step forward

NODE AFFINITY

Affinity types->

1) requiredDuringSchedulingIgnoredDuringExecution
2) preferredDuringSchedulingIgnoredDuringExecution
3) requiredDuringExecutionRequiredDuringExecution
4) preferredDuringSchedulingRequiredDuringExecution

KOENIG
step forward

NODE AFFINITY

- **requiredDuringscheduling** -> means at the time of creation it is must to have label on node otherwise it will not create pod

- **preferredDuringScheduling**-> means at the time of create it is not mandatory to have label on node if label is exist on node then it will create pod otherwise it will create pod on another available nodes.

- **ignoredDuringExecution->** it means if the pod is already exist on node and administrator changes the node label so it will ignore and it will be running on same node.

KOENIG
step forward

NODE AFFINITY

In the Node-Selector we can only use one condition at a time but in affinity we can define multiple condition like size=large or size=medium.

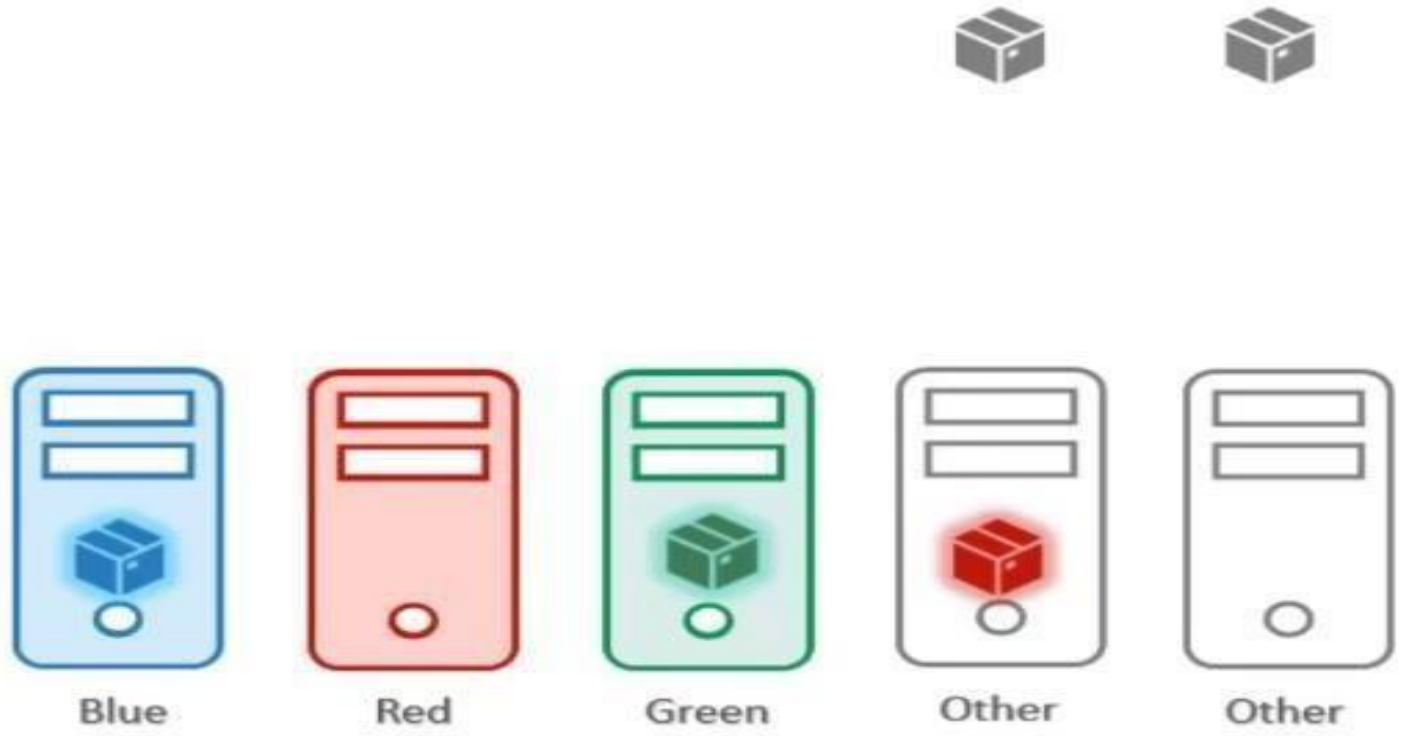
```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4      name: myapp-pod
5  spec:
6      containers:
7      - name: data-processor
8        image: data-processor
9
10     affinity:
11         nodeAffinity:
12             requiredDuringSchedulingIgnoredDuringExecution:
13                 nodeSelectorTerms:
14                 - matchExpressions:
15                     - key: size
16                       operator: In
17                       values:
18                       - Large
19                       - Medium
20
21
```

KOENIG
step forward

TAINT AND TOLERATIONS VS NODEAFFINITY

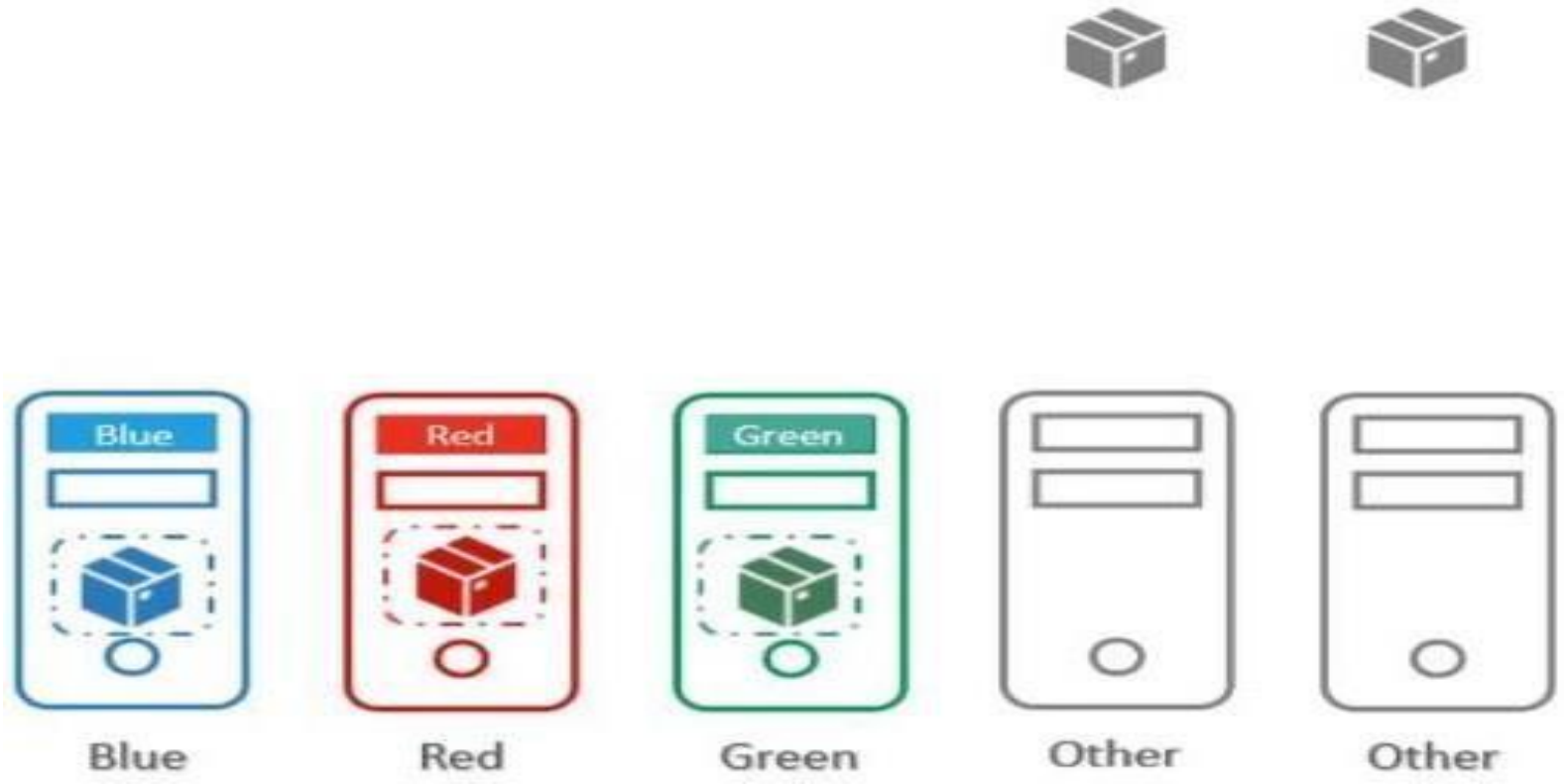
KOENIG
step forward

TAINT AND TOLERATIONS



KOENIG
step forward

NODE AFFINITY



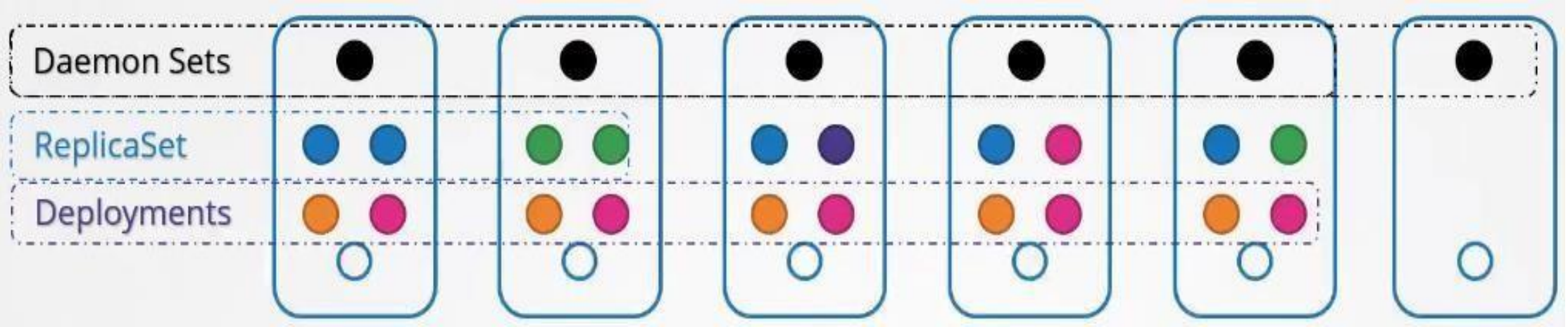
KOENIG
step forward

Daemon Set

- A DaemonSet ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created.

Some typical uses of a DaemonSet are:

- running a cluster storage daemon, such as glusterd, ceph, on each node.
- running a logs collection daemon on every node, such as fluentd or filebeat.


Daemon Sets
ReplicaSet
Deployments

KOENIG
step forward

DAEMON SET

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
 name: monitoring-tool
spec:
 selector:
  matchLabels:
   app: monitoring
 template:
  metadata:
   labels:
    app: monitoring
  spec:
   containers:
   - name: monitoring-container
     image: nginx
```

KOENIG
step forward

Verify

```
[root@master kube]# kubectl get pods -o wide
NAME                  READY   STATUS    RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
monitoring-tool-66zlx   1/1     Running   0          22s   10.244.235.143   worker1   <none>           <none>
monitoring-tool-jmfch   1/1     Running   0          22s   10.244.189.72    worker2   <none>           <none>
[root@master kube]#
```

KOENIG
step forward

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 define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

KOENIG
step forward

Uses of Deployment

- Create a Deployment to rollout a ReplicaSet
- Declare the new state of the Pods
- Rollback to an earlier Deployment revision
- Scale up the Deployment to facilitate more load
- Pause the Deployment
- Use the status of the Deployment
- Clean up older ReplicaSets

KOENIG
step forward

DEPLOYMENT STRATEGY

Two types of strategy->

1)Recreate -> in case of recreate lets say we have a pod and in pod there are 5 containers so it will first down all the containers then it will create new containers.

so in a production environment it's not a good strategy because of there is downtime.

2)Rolling Update -> in case of rolling update it will down one container then update one then down the second container then create second.

by default Kubernetes is using rolling update.


Recreate
nginx:1.7.0
nginx:1.7.0
nginx:1.7.0
nginx:1.7.0
nginx:1.7.0
Application Down
nginx:1.7.1
nginx:1.7.1
nginx:1.7.1
nginx:1.7.1
nginx:1.7.1
Rolling Update
nginx:1.7.0
nginx:1.7.1
nginx:1.7.0
nginx:1.7.1
nginx:1.7.0
nginx:1.7.1
nginx:1.7.0
nginx:1.7.1
nginx:1.7.0
nginx:1.7.1

KOENIG
step forward

DEPLOYMENT

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment
 labels:
  app: nginx
spec:
 replicas: 3
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   name: nginx-deployment
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
     image: nginx:1.14.2
```

KOENIG
step forward

DEPLOYMENT

To update manually->

#kubectl set image deployment/myapp-deployment nginx=nginx:1.7.1

To check deployment status and we can see strategy type-> #kubectl describe deployment myapp-deployment

When you are doing upgrade so automatically it create a new replicasets.

#kubectl get rs

Now let's say we have some issue with update so we can rollout to previous version.->

#kubectl rollout undo deployment/myapp-dep

#kubectl get rs

KOENIG
step forward

ROLLOUT AND VERSIONING

- When we update any application so automatically it is called as revision so when we deploy any container so it takes as a revision 1 then when we update so revision2 and so on.

```
#kubectl rollout status deployment/myapp-deployment -> to check rollout status
#kubectl rollout history deployment/myapp-deployment -> to check history of rollout
```

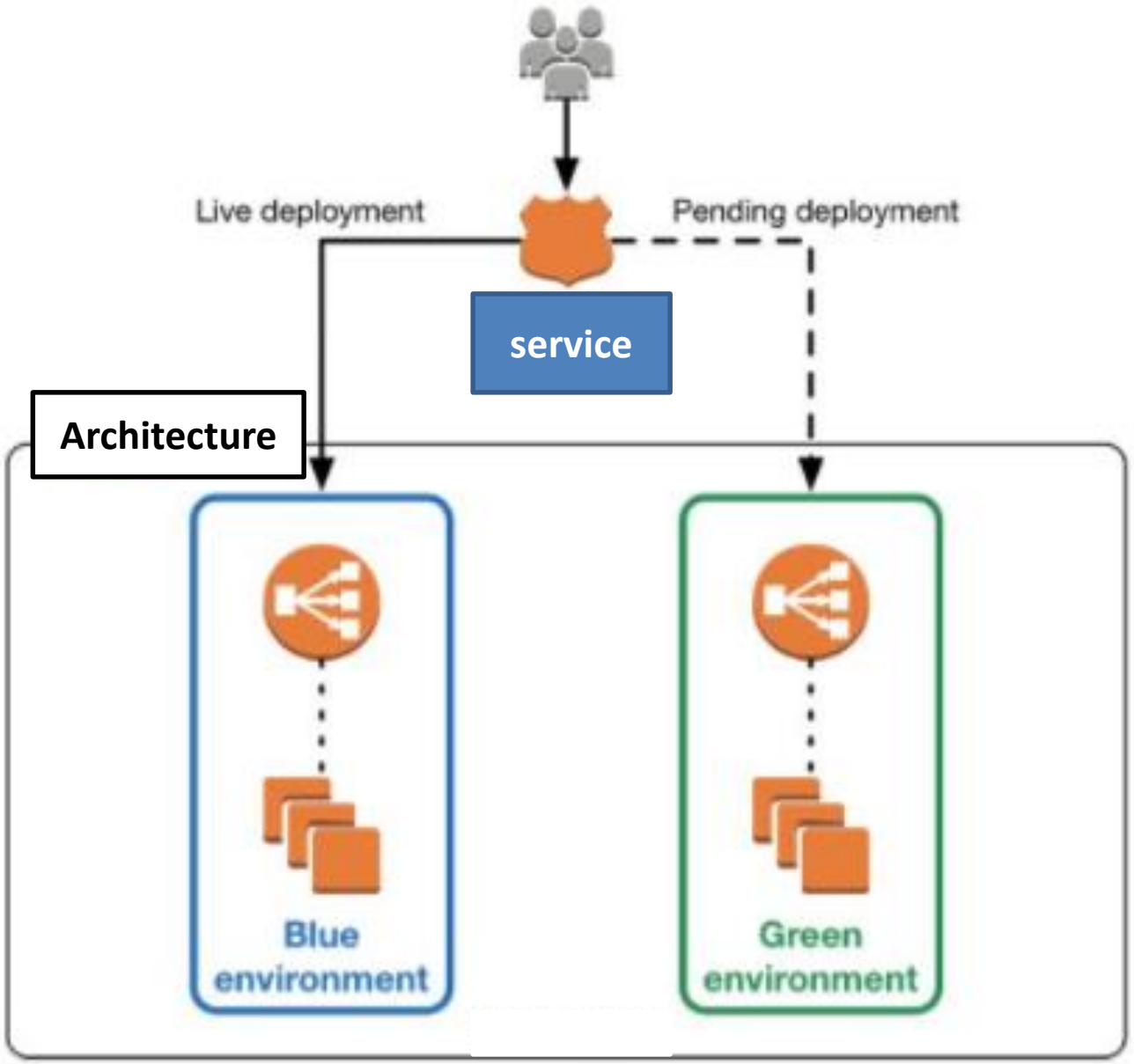
KOENIG
step forward

Blue-Green Deployment

Blue green deployment is an application release model that gradually transfers user traffic from a previous version of an app or microservice to a nearly identical new release—both of which are running in production.

The old version can be called the blue environment while the new version can be known as the green environment.

Example


Live deployment
Pending deployment
service
Architecture
Blue environment
Green environment

KOENIG
step forward

Blue Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: blue-deployment
spec:
 replicas: 5
 selector:
  matchLabels:
   app: nginx
   version: blue
 template:
  metadata:
   labels:
    app: nginx
    version: blue
  spec:
   containers:
   - name: abc
     image: nginx
     imagePullPolicy: IfNotPresent
```

```yaml
apiVersion: v1
kind: Service
metadata:
 name: mysvc
spec:
 ports:
 - port: 80
   targetPort: 80
 selector:
  app: nginx
  version: blue
```

KOENIG
step forward

Scenario

Now we have a requirement to upgrade our application, so as we are using blue-green deployment, we will create a new deployment with new updated application and update the service selector.

KOENIG
step forward

Application

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: green-deployment
spec:
 replicas: 5
 selector:
  matchLabels:
   app: nginx
   version: green
 template:
  metadata:
   labels:
    app: nginx
    version: green
  spec:
   containers:
   - name: abc
     image: quay.io/gauravkumar9130/nginxdemo
     imagePullPolicy: IfNotPresent
```

Now update the service selector:

#kubectl edit svc mysvc

```
selector:
  app: nginx
  version: green
```

KOENIG
step forward

Namespace

KOENIG
step forward

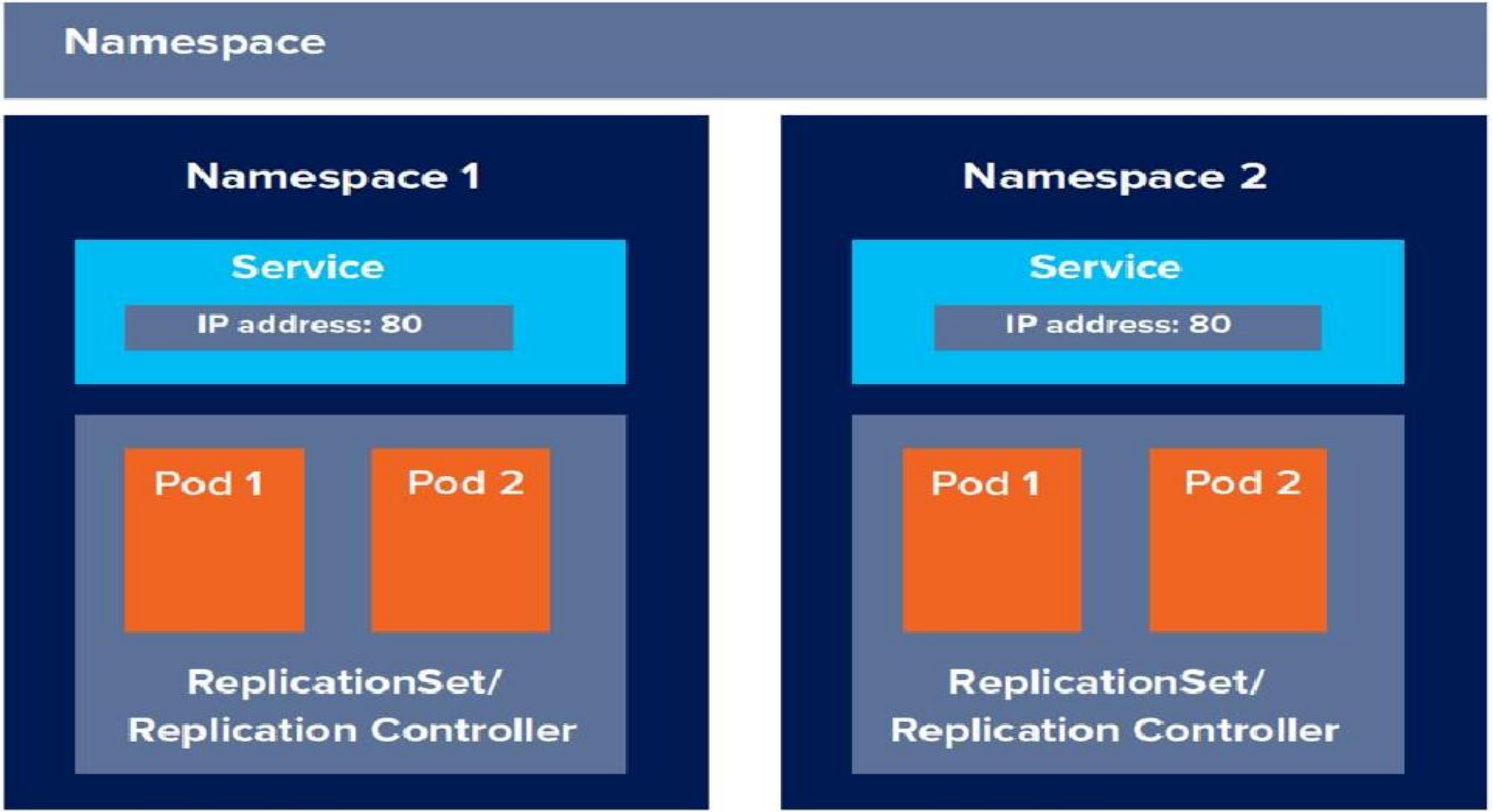
NAMESPACE

- Namespace provides an additional qualification to a resource name. This is helpful when multiple teams are using the same cluster and there is a potential of name collision. It can be as a virtual wall between multiple clusters.
- It offers an isolation for process interaction within OS.
- It limits the visibility that a process has on other process, network filesystem and userID component.
- Process from the containers or the host processes are not directly accessible from within this container process.

KOENIG
step forward

FUNCTIONALITY OF NAMESPACE

- Namespaces help pod-to-pod communication using the same namespace.
- Namespaces are virtual clusters that can sit on top of the same physical cluster.
- They provide logical separation between the teams and their environments.


Namespace
Namespace 1
Service
IP address: 80
Pod 1
Pod 2
ReplicationSet/
Replication Controller
Namespace 2
Service
IP address: 80
Pod 1
Pod 2
ReplicationSet/
Replication Controller

KOENIG
step forward

CREATE NAMESPACE

```
apiVersion: v1
kind: Namespace
metadata:
 name: dev

          OR

#kubectl create namespace dev
```

KOENIG
step forward

LIST NAMESPACE

```
[root@localhost ~]# kubectl get namespace
NAME                 STATUS   AGE
default              Active   6h45m
dev                  Active   7s
kube-node-lease      Active   6h45m
kube-public          Active   6h45m
kube-system          Active   6h45m
[root@localhost ~]#
```

KOENIG
step forward

TO ADD PODS IN NAMESPACE

```
apiVersion: v1
kind: Pod
metadata:
 name: dev-pod
 namespace: dev
spec:
 containers:
 - name: nginx-container
   image: nginx
~
```

```
[root@localhost ~]# kubectl get pods -n dev
NAME      READY   STATUS    RESTARTS   AGE
dev-pod   1/1     Running   0          13s
[root@localhost ~]#
```

KOENIG
step forward

CONTROL THE NAMESPACES

#kubectl get namespace

 # kubectl describe namespace <Namespace name>

#kubectl delete namespace <Namespace name>

To switch from default namespace to another namespace:

#kubectl config set-context $(kubectl config current-context) --namespace=dev

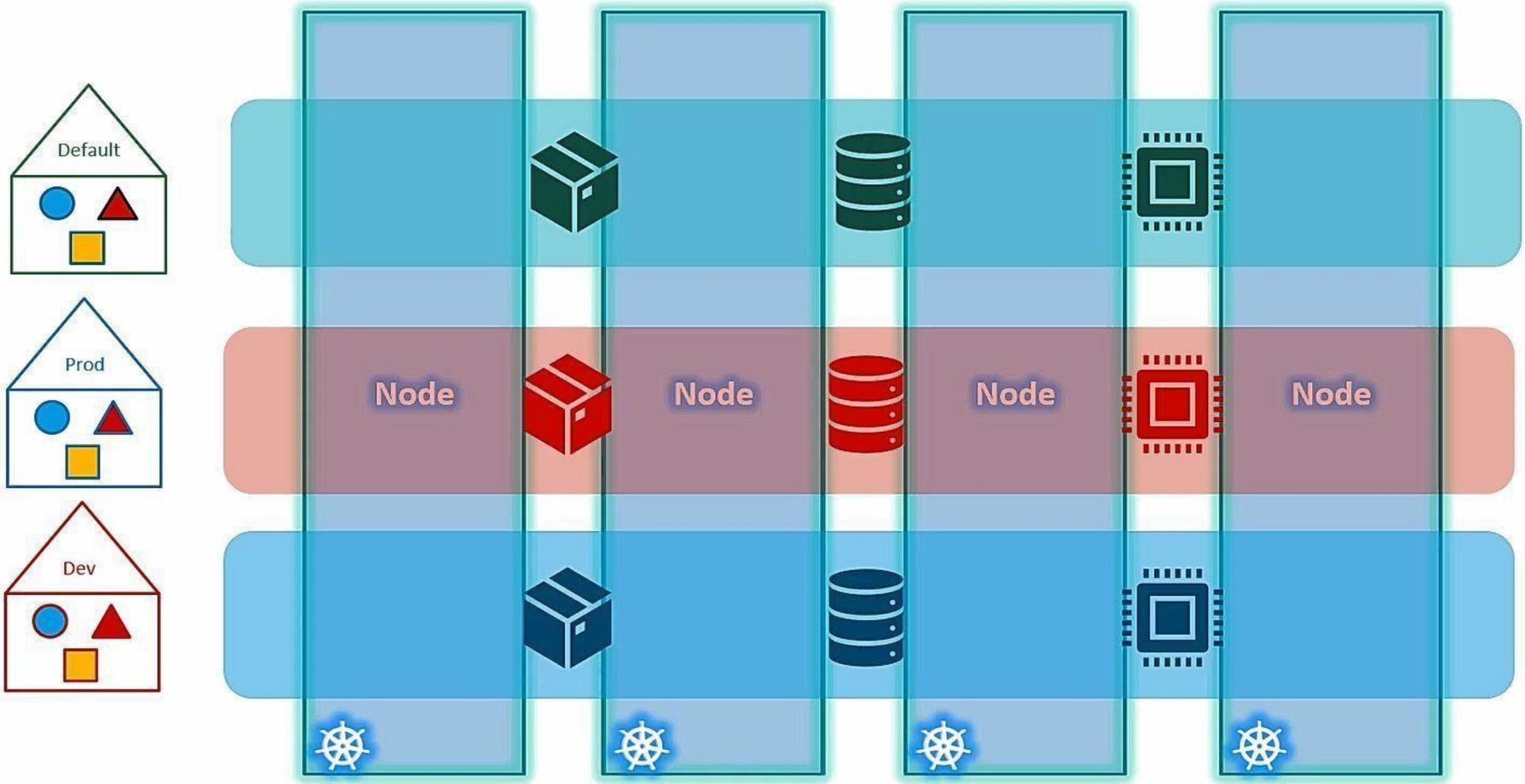
To view pods from all namespaces

#kubectl get pods --all-namespaces

OR

#kubectl get pods -A

Namespace – Resource Limits



KOENIG
step forward

RESOURCE QUOTA IN NAMESPACE

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
 name: dev-quota
 namespace: dev
spec:
 hard:
  pods: "10"
  requests.cpu: "4"
  requests.memory: "5Gi"
  limits.cpu: "10"
  limits.memory: "10Gi"
~
```

```
[root@localhost ~]# kubectl get quota -n dev
NAME        AGE   REQUEST                                                 LIMIT
dev-quota   7s    pods: 1/10, requests.cpu: 0/4, requests.memory: 0/5Gi   limits.cpu: 0/10, limits.memory: 0/10Gi
[root@localhost ~]#
```

KOENIG
step forward

Create Pod In given Namespace

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: dev-app
 namespace: dev
spec:
 containers:
 - name: dev-app
   image: nginx
   ports:
   - containerPort: 80
   resources:
    requests:
      memory: "1Gi"
      cpu: "1"
    limits:
      memory: "2Gi"
      cpu: 2
```

Pod will scheduled on a node with at least 1Gi memory and 1 CPU

Pod will be killed if it tries to use more than 2Gi memory.

Pod will be throttled if it uses more than 2 core.

KOENIG
step forward

ENVIRONMENT VARIABLE IN KUBERNETES

TYPES OF ENVIRONMENT VARIABLES

- Plain key value
- ConfigMap
- Secrets

KOENIG
step forward

PLAIN KEY VALUE

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: db-pod
spec:
 containers:
 - name: mydb
   image: quay.io/gauravkumar9130/mysql
   env:
    - name: MYSQL_ROOT_PASSWORD
      value: root123
    - name: MYSQL_USER
      value: gaurav
    - name: MYSQL_PASSWORD
      value: gaurav123
```

CONFIG MAP

- A ConfigMap is a dictionary of configuration settings. This dictionary consists of key-value pairs of strings. Kubernetes provides these values to your containers. Like with other dictionaries (maps, hashes, ...) the key lets you get and set the configuration value.

KOENIG
step forward

CONFIG MAP

- STEP 1: CREATE CONFIG-MAP

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: db-app
data:
 MYSQL_ROOT_PASSWORD: root123
 MYSQL_USER: gaurav
 MYSQL_USER_PASSWORD: gaurav123
```

```
[root@master ~]# kubectl get configmaps
NAME      DATA   AGE
db-app    3      14s
```

CONFIG MAP

- STEP 2: inject configmap in pod file

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: db-pod
spec:
 containers:
 - name: mydb
   image: quay.io/gauravkumar9130/mysql
   envFrom:
    - configMapRef:
       name: db-app
```

KOENIG
step forward

SECRETS

- Secrets can be defined as Kubernetes objects used to store sensitive data such as user name and passwords with encryption.

There are multiple ways of creating secrets in Kubernetes.

- Creating from txt files.
- Creating from yaml file.

#kubectl create secret generic mysql --from-literal=password=mypassword

KOENIG
step forward

SECRETS

```yaml
apiVersion: v1
kind: Secret
metadata:
 name: db-secret
data:
 MYSQL_ROOT_PASSWORD: root123
 MYSQL_USER: gaurav
 MYSQL_USER_PASSWORD: gaurav123
```

```
[root@master files]# kubectl apply -f secrets.yml
Error from server (BadRequest): error when creating "secrets.yml": Secret in version "v1" cannot be handled as a Se
cret: v1.Secret.Data: decode base64: illegal base64 data at input byte 4, error found in #10 byte of ...|:"root123"
,"MYSQL_US|..., bigger context ...|sion":"v1","data":{"MYSQL_ROOT_PASSWORD":"root123","MYSQL_USER":"gaurav","MYSQL_
USER_PASSWORD":"gaur|...
[root@master files]#
```

```
#echo -n "mysql" | base64
bxiZcWw=     --> output
```

SECRETS

```
[root@master ~]# echo -n "root123" | base64
cm9vdDEyMw==
[root@master ~]# echo -n "gaurav" | base64
Z2F1cmF2
[root@master ~]# echo -n "gaurav123" | base64
Z2F1cmF2MTIz
[root@master ~]#
```

```yaml
apiVersion: v1
kind: Secret
metadata:
 name: db-secret
data:
 MYSQL_ROOT_PASSWORD: cm9vdDEyMw==
 MYSQL_USER: Z2F1cmF2
 MYSQL_USER_PASSWORD: Z2F1cmF2MTIz
```

#kubectl get secrets #kubectl describe secrets

#kubectl get secret app-secret -o yaml -> to show all information and encrypted password

#echo -n ‘bxiZcWw= ‘ | base64 --decode -> to decode

KOENIG
step forward

SECRETS

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: db-pod
spec:
 containers:
 - name: mydb
   image: quay.io/gauravkumar9130/mysql
   envFrom:
    - secretRef:
       name: db-secret
```

KOENIG
step forward

Mount Environment Variables as Volume

KOENIG
step forward

Create Config Map

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: app-config
data:
 name: gaurav
 password: redhat
~
```

KOENIG
step forward

Mount Config Map As Volume

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: config-pod
spec:
 containers:
 - name: config
   image: nginx
   volumeMounts:
   - name: config-vol
     mountPath: /data
 volumes:
 - name: config-vol
   configMap:
    name: app-config
```

KOENIG
step forward

Creating Secret

```
apiVersion: v1
kind: Secret
metadata:
 name: app-secret
data:
 name: Z2F1cmF2
 password: cmVkaGF0
~
```

KOENIG
step forward

Mount Secret as Volumes

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: secret-pod
spec:
 containers:
 - name: secret
   image: nginx
   volumeMounts:
   - name: secret-vol
     mountPath: /data
 volumes:
 - name: secret-vol
   secret:
    secretName: app-secret
```

KOENIG
step forward

Secrets Environement Variable Mapping

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: secret-pod
spec:
 containers:
 - name: secret
   image: nginx
   env:
   - name: CREDENTIALS
     valueFrom:
      secretKeyRef:
       name: app-secret
       key: password
```

KOENIG
step forward

STORAGE

KOENIG
step forward

Storage

Volumes

- Exposed at Pod level and backend different ways
- emptyDir: ephermal scratch directory that lives for the life of pod

Persistent Volumes

- Abstraction away from the storage provider(AWS, GCE)
- Have a lifecycle independent of any individual pod that uses it

Persistent Volume Claims

- Request for storage with specific details(size,access modes, etc.)

KOENIG
step forward

TYPES OF KUBERNETES VOLUME

KOENIG
step forward

Persistent Volumes

- By default, containers write to ephermal storage
- As a result, when a pod is terminated, all data written by it’s container is lost
- We can attach Persistent Volumes to pods which persist any data written to them

KOENIG
step forward

emptyDir

An emptyDir volume is first created when a Pod is assigned to a Node, and exists as long as that Pod is running on that node

KOENIG
step forward

EmptyDir

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: cache-pod
spec:
 containers:
 - name: cache-container
   image: quay.io/gauravkumar9130/nginxdemo
   volumeMounts:
   - name: cache-vol
     mountPath: /mytest
 volumes:
 - name: cache-vol
   emptyDir: {}
```

KOENIG
step forward

Persistent Volumes

- Persistent Volumes can be provisioned either statically or dynamically.
  - Static: Apre-provisioned pool of volumes such as ISCSI or Fiber Channel
  - Dynamic: Volumes are created on demand by calling a storage provider's API such as Amazon EBS

- Persistent Volumes have a lifecycle independent of the pods that use them

KOENIG
step forward

Persistent Volume Claims

- Created by users to request a persistent volume
- Users can request various properties such as capacity and access modes (e.g. can be mounted once read/write or many times read-only)

KOENIG
step forward

Access Modes

The access modes are:

- ReadWriteOnce -- the volume can be mounted as read-write by a single node
- ReadOnlyMany -- the volume can be mounted read-only by many nodes
- ReadWriteMany -- the volume can be mounted as read-write by many nodes

KOENIG
step forward

CREATING PERSISTENT VOLUME

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-vol1
spec:
 accessModes:
 - ReadWriteOnce
 capacity:
  storage: "1Gi"
 hostPath:
  path: "/webvolumedata"                    #save data in the scheduled node
```

```
[root@localhost ~]# kubectl get pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
pv-vol1   1Gi        RWO            Retain           Available                                   17s
[root@localhost ~]#
```

KOENIG
step forward

CREATING PERSISTENT VOLUME CLAIM

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pv-vol-claim
spec:
 accessModes:
 - ReadWriteOnce
 resources:
  requests:
   storage: "500Mi"
~
```

```
[root@localhost ~]# kubectl get pvc
NAME           STATUS   VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
pv-vol-claim   Bound    pv-vol1   1Gi        RWO                           30s
[root@localhost ~]#
```

KOENIG
step forward

USE PV IN POD

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: my-pv-pod
spec:
 containers:
 - name: my-web-persistent
   image: quay.io/app-sre/nginx
   ports:
   - containerPort: 80
   volumeMounts:
   - name: mydatavol
     mountPath: "/var/www/html"
 volumes:
 - name: mydatavol
   persistentVolumeClaim:
    claimName: pv-vol-claim
```

KOENIG
step forward

SECURITY

KOENIG
step forward

Kubernetes Authentication

- Authentication machanism handled by kubeapi server when we use kubectl command.

KOENIG
step forward

Transport Layer Security (TLS)

KOENIG
step forward

TLS

- Kubernetes provides a certificates.k8s.io API, which lets you provision TLS certificates signed by a Certificate Authority (CA) that you control. These CA and certificates can be used by your workloads to establish trust.

KOENIG
step forward

CERTIFICATE AUTHORITY (CA)

Client Certificates for Clients

admin.crt admin.key

admin

scheduler.crt scheduler.key

KUBE-SCHEDULER

controller-manager.crt controller-manager.key

KUBE-CONTROLLER-MANAGER

kube-proxy.crt kube-proxy.key

KUBE-PROXY

apiserver-kubelet-client.crt apiserver-kubelet-client.key

KUBE-API SERVER

apiserver-etcd-client.crt apiserver-etcd-client.key

KUBE-API SERVER

kubelet-client.crt kubelet-client.key

KUBELET SERVER

Server Certificates for Servers

etcdserver.crt etcdserver.key

ETCD SERVER

apiserver.crt apiserver.key

KUBE-API SERVER

kubelet.crt kubelet.key

KUBELET SERVER

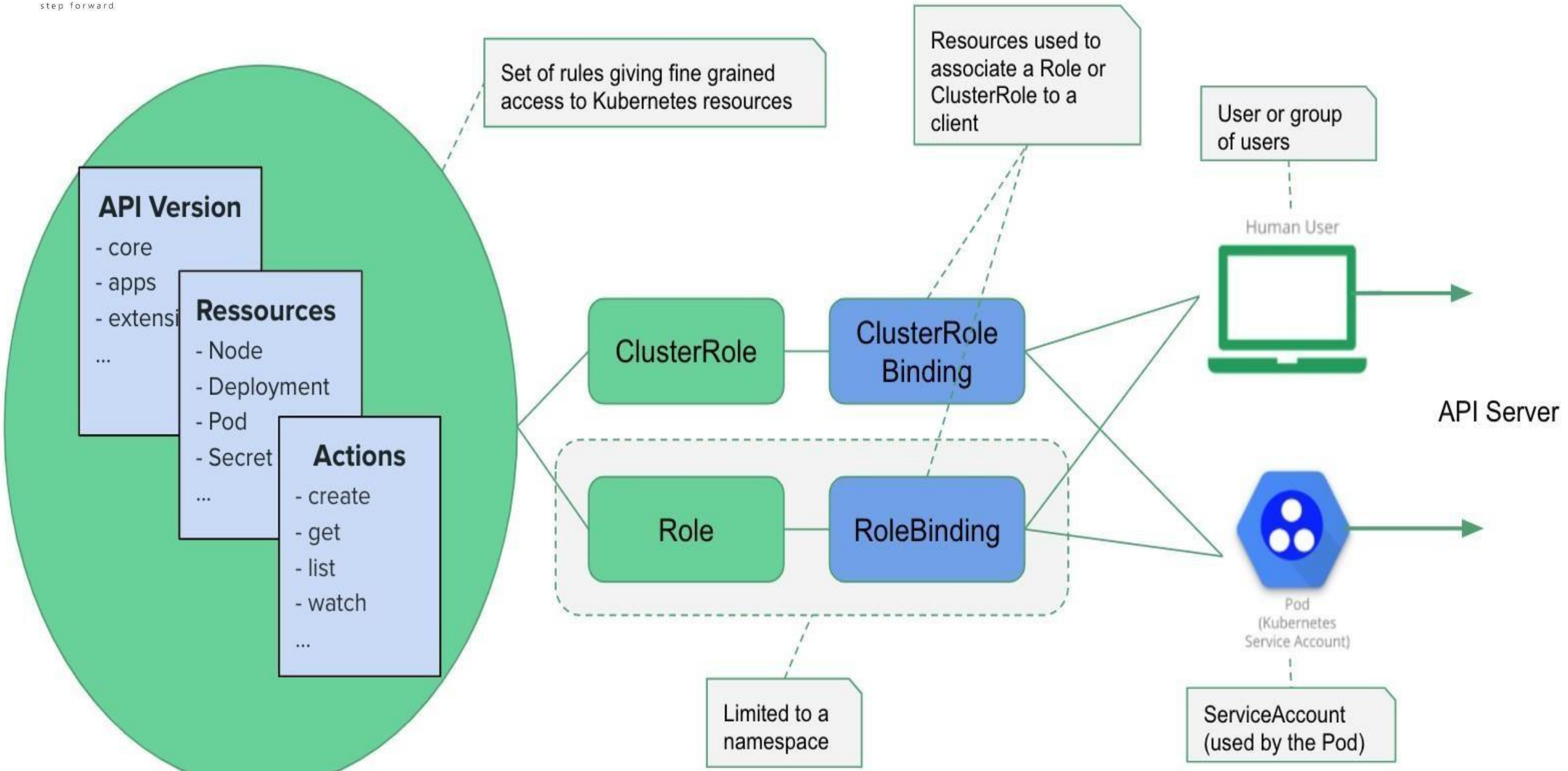
KOENIG
step forward

Managing Users in Kubernetes

KOENIG
step forward

What is Service Account?

A ServiceAccount is used by containers running in a Pod, to communicate with the API server of the Kubernetes cluster.


Set of rules giving fine grained access to Kubernetes resources
Resources used to associate a Role or ClusterRole to a client
User or group of users
API Version
- core
- apps
- extensi
...
Ressources
- Node
- Deployment
- Pod
- Secret
...
Actions
- create
- get
- list
- watch
...
ClusterRole
ClusterRole Binding
Human User
API Server
Role
RoleBinding
Pod
(Kubernetes Service Account)
Limited to a namespace
ServiceAccount (used by the Pod)

KOENIG
step forward

User accounts versus service accounts

User Accounts	Service Accounts
User accounts for human	Service Accounts for processes, which run in pods
User accounts are intended to be global. Names must be unique across all namespaces of a cluster, future user resource will not be namespaced.	Service accounts are namespaced.
cluster's User accounts might be synced from a corporate database, where new user account creation requires special privileges and is tied to complex business processes.	Service account creation is intended to be more lightweight, allowing cluster users to create service accounts for specific tasks (i.e. principle of least privilege).

Important

When you create a pod, if you do not specify a service account, it is automatically assigned the default service account in the same namespace for authentication.

#kubectl get sa

KOENIG
step forward

Creating Service Account

```
apiVersion: v1
kind: ServiceAccount
metadata:
 name: sales-service-account
 namespace: sales
```

#kubectl get sa -n sales
#kubectl describe sales-service-account –n sales

```
[root@master test]# kubectl describe sa sales-service-account -n sales
Name:                sales-service-account
Namespace:           sales
Labels:              <none>
Annotations:         <none>
Image pull secrets:  <none>
Mountable secrets:   sales-service-account-token-zkb6s
Tokens:              sales-service-account-token-zkb6s
Events:              <none>
[root@master test]#
```

Each service account creates a secrets token which is used to authenticate with API server.

KOENIG
step forward

Create Pod with Service Account

```
apiVersion: v1
kind: Pod
metadata:
 name: pod1
 namespace: sales
spec:
 containers:
 - name: pod1
   image: nginx
 serviceAccountName: sales-service-account
```

KOENIG
step forward

Authorization

KOENIG
step forward

Authorization

In Kubernetes, you must be authenticated (logged in) before your request can be authorized (granted permission to access).

Kubernetes authorizes API requests using the API server. It evaluates all of the request attributes against all policies and allows or denies the request. All parts of an API request must be allowed by some policy in order to proceed. This means that permissions are denied by default.

KOENIG
step forward

Authorization Modes

- **Node Based**
  - grants permissions to kubelets based on the pods they are scheduled to run

- **Attribute Based Access Control (ABAC)**
  - Manual Manage Permissions in API

- **Role Based Access Control (RBAC)**
  - Roles and Role Bindings

- **Webhook**
  - manage third party authentication

KOENIG
step forward

Kube API Server Yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint: 172.25.230.143:6443
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=172.25.230.143
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
```

KOENIG
step forward

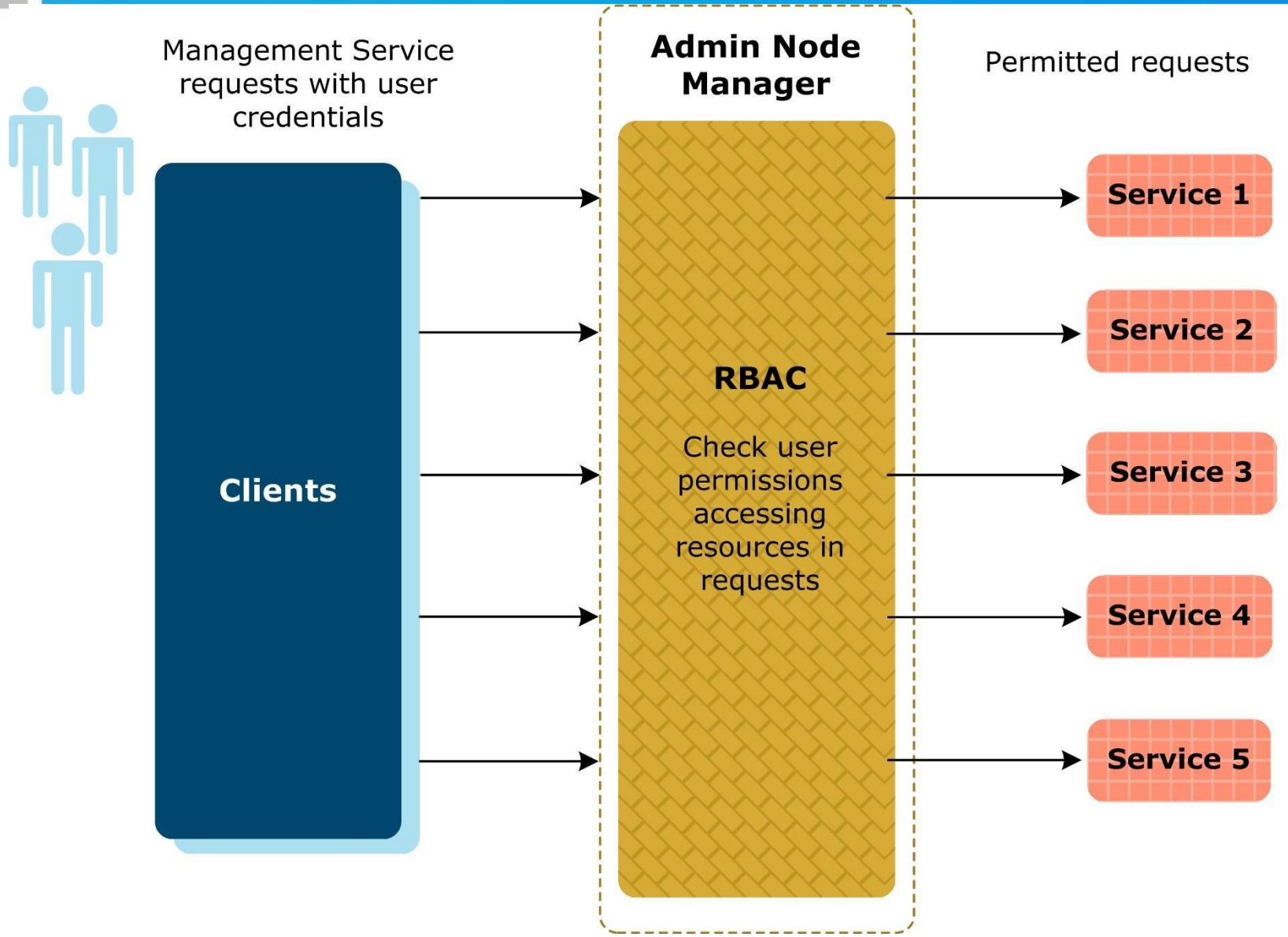
Role-Based Access Control (RBAC) Authorization

What is RBAC?

Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within your organization.

KOENIG
step forward

RBAC



KOENIG
step forward

Managing Roles with Imperative Commands

To create role to list pods and services:-

#kubectl create role pod-reader --verb=get --resource=pods,services

To create role to list pods (only for pod1 and pod2):-

#kubectl create role pod-reader --verb=get --resource=pods
--resource-name=pod1 --resource-name=pod2

To List Role:-

#kubectl get roles

To Describe Role:-

#kubectl describe role <rolename>

KOENIG
step forward

Managing RoleBindings with Imperative

To Assign Role to bob user in namespace acme:-

#kubectl create rolebinding bob-admin-binding --role=admin --user=bob --namespace=acme

To List Role Binding:-

#kubectl get rolebindings

To Describe Role Binding:-

#kubectl describe rolebinding <rolebindingname>

KOENIG
step forward

Check Access

#kubectl auth can-i <verb> <resource>

For example:

#kubectl auth can-i create pods

#kubectl auth can-i create pods --as gaurav

KOENIG
step forward

IMPORTANT NOTE

Roles and Rolebindings are restricted to namespace.

If we want to give access cluster wide then we will use clusterrole and clusterrolebinding.

To check role access->

#kubectl api-resources --namespaced=true

To check clusterrole access->

#kubectl api-resources --namespaced=false

KOENIG
step forward

Managing ClusterRoles with Imperative Commands

To create cluster role to list pods and services:-

#kubectl create clusterrole pod-reader --verb=get --resource=nodes

To create cluster role to list pods (only for pod1 and pod2):-

#kubectl create clusterrole pod-reader --verb=get --resource=pods
--resource-name=pod1 --resource-name=pod2

To List Cluster Role:-

#kubectl get clusterroles

To Describe Cluster Role:-

#kubectl describe clusterrole <clusterrolename>

KOENIG
step forward

Managing ClusterRoleBindings with Imperative

To Assign Cluster Role to bob user in namespace acme:-

#kubectl create clusterrolebinding bob-admin-binding --clusterrole=admin --user=bob

To List Cluster Role Binding:-

#kubectl get clusterrolebindings

To Describe Cluster Role Binding:-

#kubectl describe clusterrolebinding <clusterrolebindingname>

KOENIG
step forward

SECURITY CONTEXT

Security Context facilitates management of access rights, privileges, and permissions for processes and filesystems in Kubernetes.

KOENIG
step forward

SECURITY CONTEXT

```
apiVersion: v1
kind: Pod
metadata:
 name: security-context
spec:
 containers:
 - name: security-pod
   image: nginx
   securityContext:
    capabilities:
     add: ["SYS_TIME"]
```

KOENIG
step forward

NETWORKING

KOENIG
step forward

DNS

- Kubernetes DNS schedules a DNS Pod and Service on the cluster, and configures the kubelets to tell individual containers to use the DNS Service's IP to resolve DNS names.

- Every Service defined in the cluster (including the DNS server itself) is assigned a DNS name. By default, a client Pod's DNS search list will include the Pod's own namespace and the cluster's default domain.

KOENIG
step forward

CUSTOM DNS FOR CONTAINER

```
apiVersion: v1
kind: Pod
metadata:
 name: custom-dns-pod
spec:
 hostAliases:
 - ip: "127.0.0.1"
   hostnames:
   - "localhost"
   - "gaurav.example.com"
 - ip: "172.25.230.111"
   hostnames:
   - "remote.example.com"
 containers:
 - name: dns-hosts
   image: nginx
```

KOENIG
step forward

Verify

```
[root@master ~]# kubectl exec -it custom-dns-pod -- bash
root@custom-dns-pod:/# cat /etc/hosts
# Kubernetes-managed hosts file.
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
fe00::0 ip6-mcastprefix
fe00::1 ip6-allnodes
fe00::2 ip6-allrouters
10.244.235.132  custom-dns-pod

# Entries added by HostAliases.
127.0.0.1       localhost       gaurav.example.com
172.25.230.111  remote.example.com
root@custom-dns-pod:/#
```

KOENIG
step forward

CONTAINER NETWORK INTERFACE

- Kubernetes does not provide any default network implementation, rather it only defines the model and leaves to other tools to implement it. We can use **Calico** Solution.

NETWORK SOLUTION

We have popular Network Solution->

- Calico (https://www.tigera.io/blog/kubernetes-networking-with-calico/)

https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/

KOENIG
step forward

CALICO

- Calico is a networking and network policy provider. Calico supports a flexible set of networking options so you can choose the most efficient option for your situation.

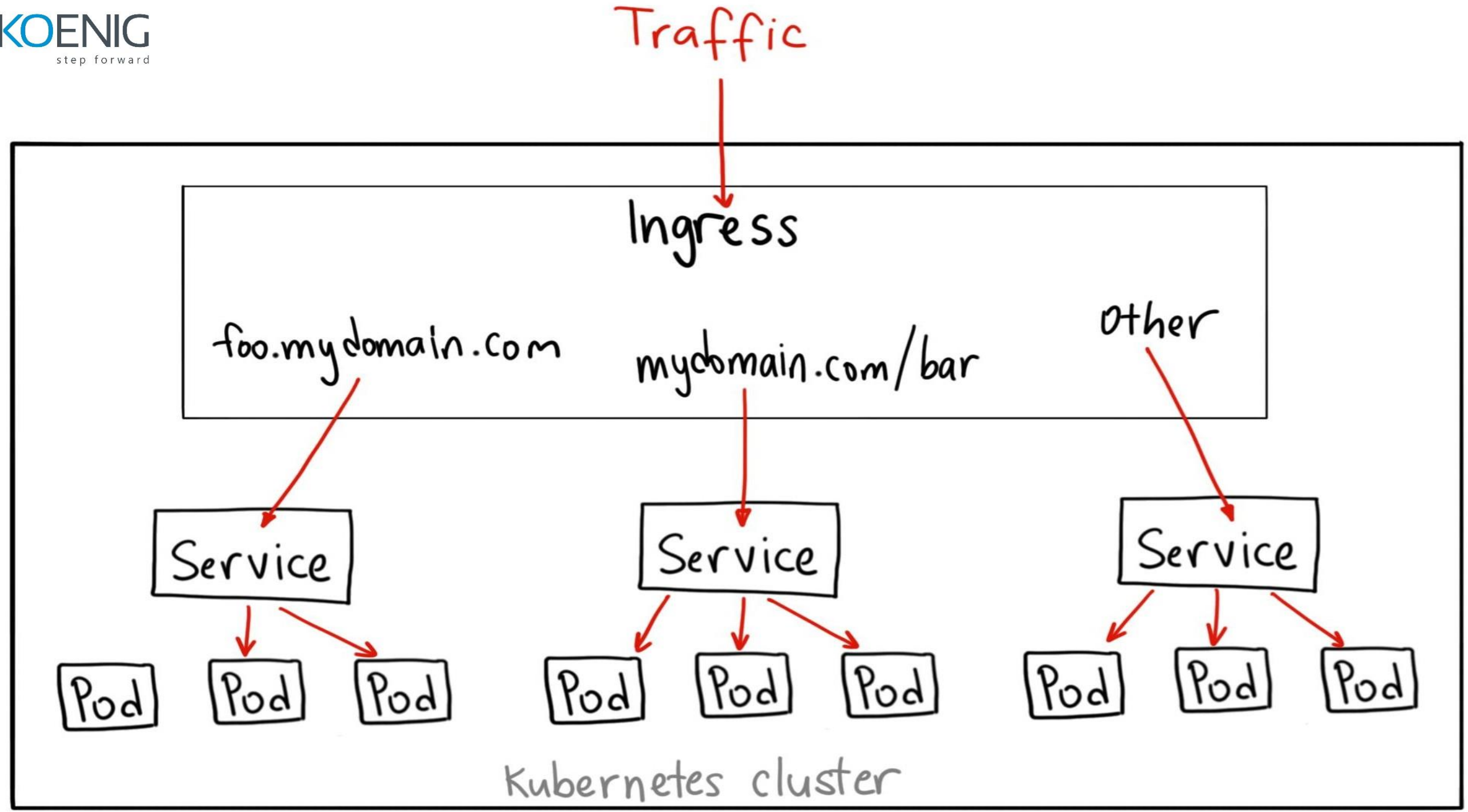
- By default, Calico uses any ipaddress other than you cluster ip, as the Pod network CIDR, though this can be configured in the calico.yaml file. For Calico to work correctly, you need to pass this same CIDR to the kubeadm init command using the **--pod-network-cidr** flag or via kubeadm's configuration.

- kubectl apply -f https://docs.projectcalico.org/v3.14/manifests/calico.yaml

KOENIG
step forward

INGRESS

- In Kubernetes, an Ingress is an object that allows access to your Kubernetes services from outside the Kubernetes cluster. You configure access by creating a collection of rules that define which inbound connections reach which services.


Traffic
Ingress
foo.mydomain.com
mydomain.com/bar
Other
Service
Service
Service
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Kubernetes cluster

KOENIG
step forward

HOW TO DEPLOY INGRESS

```
yum install -y git
git clone https://github.com/nginxinc/kubernetes-ingress
cd kubernetes-ingress/
git checkout v1.8.1
cd deployments/
kubectl apply -f common/ns-and-sa.yaml
kubectl apply -f common/default-server-secret.yaml
kubectl apply -f common/nginx-config.yaml
kubectl apply -f rbac/rbac.yaml
kubectl apply -f daemon-set/nginx-ingress.yaml
```

KOENIG
step forward

VERIFY INSTALLATION

```
[root@master ~]# kubectl get pods -n nginx-ingress
NAME                    READY   STATUS    RESTARTS   AGE
nginx-ingress-bsr8s     1/1     Running   0          51s
nginx-ingress-rk24l     1/1     Running   0          51s
[root@master ~]#
```

KOENIG
step forward

Host Based Ingress
(Hotel Application)

Application

- we are creating a website for :

  hotel.example.com

  With using hotel application and service

Application(HostBased)

#mkdir ingress-hostbased

Creating Namespace:

#kubectl create ns data

KOENIG
step forward

Deploying Hotel Application

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: hotel
 namespace: data
spec:
 replicas: 5
 selector:
  matchLabels:
   app: hotel
 template:
  metadata:
   labels:
    app: hotel
  spec:
   containers:
   - name: hotel
     image: nginxdemos/hello:plain-text
```

KOENIG
step forward

Creating Service for Application

```yaml
apiVersion: v1
kind: Service
metadata:
 name: hotel-svc
 namespace: data
spec:
 ports:
 - port: 80
   targetPort: 80
   protocol: TCP
   name: http
 selector:
  app: hotel
```

KOENIG
step forward

Create Ingress Rule(Host Based)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: hotel-ingress
 namespace: data
spec:
 rules:
 - host: hotel.example.com
   http:
    paths:
    - path: /
      pathType: Prefix
      backend:
       service:
        name: hotel-svc
        port:
         number: 80
~
```

KOENIG
step forward

Verify

- #kubectl get all --namespace hotel
- #kubectl describe ingress hotel-ingress --namespace hotel

Add entry in host file because of we are not using any loadbalancer:

#vim /etc/hosts

<ipofworker> hotel.example.com

```
[root@master ~]# curl hotel.example.com
Server address: 10.244.189.100:80
Server name: hotel-64f7d9c9fc-w8ngb
Date: 15/Oct/2020:19:52:20 +0000
URI: /
Request ID: cc95feeb212a93e54582bff551c4188f
[root@master ~]#
```

KOENIG
step forward

Load Balancer

KOENIG
step forward

WHY?

- Kubernetes does not offer an implementation of network load-balancers (Services of type LoadBalancer) for bare metal clusters.

- LoadBalancers will remain in the “pending” state indefinitely when created.

KOENIG
step forward

METAL LOAD BALANCER

- MetalLB hooks into your Kubernetes cluster, and provides a network load-balancer implementation. In short, it allows you to create Kubernetes services of type “LoadBalancer” in clusters.

- It has two features that work together to provide this service: address allocation, and external announcement.

KOENIG
step forward

Prerequisites

#kubectl edit configmap -n kube-system kube-proxy

```
apiVersion: kubeproxy.config.k8s.io/v1alpha1
kind: KubeProxyConfiguration
mode: "ipvs"
ipvs:
  strictARP: true
```

KOENIG
step forward

Installation By Manifests

```
kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.9.3/manifests/namespace.yaml
kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.9.3/manifests/metallb.yaml
# On first install only
kubectl create secret generic -n metallb-system memberlist --from-literal=secretkey="$(openssl rand -base64
128)"
```

https://metallb.universe.tf/installation/

KOENIG
step forward

Create ConfigMap

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
```

https://metallb.universe.tf/configuration/

KOENIG
step forward

LET'S DEPLOY A APPLICATAION

```
[root@master abc]# cat deploy.yml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment
 labels:
   app: nginx
spec:
 replicas: 4
 selector:
   matchLabels:
    app: nginx
 template:
   metadata:
    name: nginx-deployment
    labels:
      app: nginx
   spec:
    containers:
    - name: nginx-deployment
      image: nginx
      ports:
       - containerPort: 80
```

KOENIG
step forward

SERVICE FOR APPLICATION

```
[root@master abc]# cat service.yml
apiVersion: v1
kind: Service
metadata:
 name: nginx
spec:
 ports:
 - name: http
   port: 80
   protocol: TCP
   targetPort: 80
 selector:
  app: nginx
 type: LoadBalancer
[root@master abc]#
```

KOENIG
step forward

Verify

```
[root@master abc]# kubectl get svc
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)        AGE
kubernetes   ClusterIP      10.96.0.1       <none>           443/TCP        10h
nginx        LoadBalancer   10.98.253.120   172.25.230.240   80:31739/TCP   13m
[root@master abc]#
```

KOENIG
step forward

Network Policy

Network policies are Kubernetes resources that control the traffic between pods and/or network endpoints. They uses labels to select pods and specify the traffic that is directed toward those pods using rules.

The most popular CNI plugins with network policy support are:

- Weave
- Calico
- Cilium
- Kube-router
- Romana

KOENIG
step forward

Create Scenario

```
[root@master ~]# kubectl run web --image=quay.io/gauravkumar9130/nginxdemo
pod/web created
[root@master ~]# kubectl run storage --image=quay.io/gauravkumar9130/nginxdemo
pod/storage created
[root@master ~]# kubectl run test-pod --image=quay.io/gauravkumar9130/nginxdemo
pod/test-pod created
[root@master ~]#
```

KOENIG
step forward

Example 1

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-web-only-for-storage
spec:
  podSelector:
    matchLabels:
      run: storage
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          run: web
```

Storage pod will only accept traffic from web pod.

KOENIG
step forward

Logging and Monitoring

What are the kinds of logs we want to capture?

- Container Logs
- Host OS Logs
- Control Plane Logs
- Event Messages

KOENIG
step forward

Logging in Kubernetes and Containers

Helpful Command:

#kubectl logs <podname> -c <containername>

KOENIG
step forward

Monitoring

- Free
  - Kube-state-metrics- collects metrics from KubernetesAPI
  - Prometheus
  - metricbeat – standard metric monitoring that is Kubernetes aware
  - Ganglia
  - And more……
- Third Party Services – Paid
  - Datadog
  - Honeycomb
  - Stackdriver with google monitoring
  - And more…

KOENIG
step forward

Setup Prometheus Monitoring Tool

Prometheus

Monitoring

KOENIG
step forward

Configuration

Download and Install Prometheus:-

#git clone https://github.com/gauravkumar9130/prometheus.git

#cd prometheus

#kubectl create -f prometheus.yaml

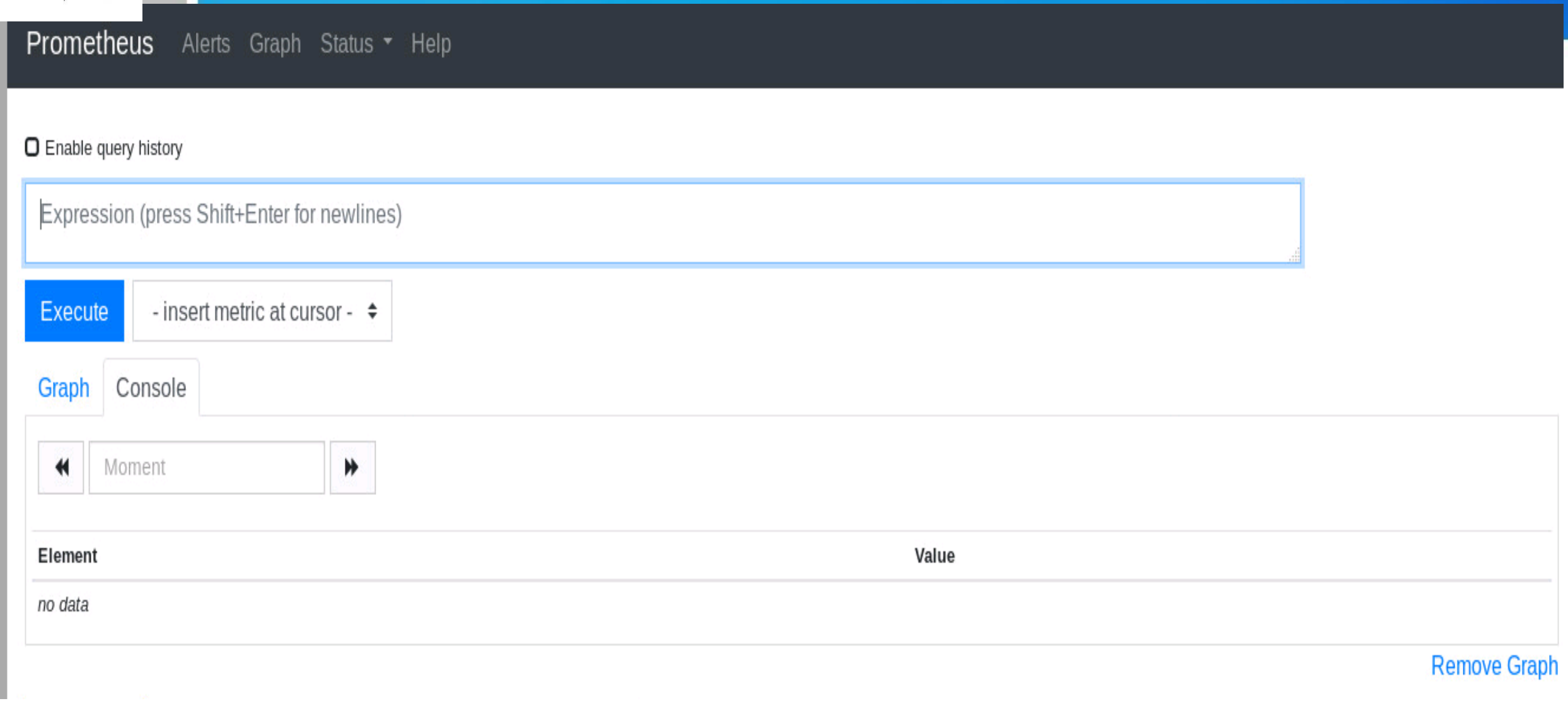
#kubectl create -f kube-state-metrics-configs/.

To Verify:-

#kubectl get pods -n monitoring

#kubectl get svc –n monitoring

KOENIG
step forward



KOENIG
step forward

CLUSTER MAINTAINANCE

KOENIG
step forward

OS UPGRADE

Before doing os upgradation first drain->

#kubectl drain worker1 --force --ignore-daemonsets

Technically containers are not removed(in case of replica) they are gracefully terminated and recreated on another node and our node1 become unschedulable i.e no container can be created on it then we reboot node1 still it is in unscheduable state to make it schedulable

#kubectl uncordon worker1

KOENIG
step forward

KUBECTL VERSION

#kubectl get nodes -> to check current version

v1.11.3

v1 -> major

11 -> minor (added features and functionalities)

3 -> patch (bug fixes)

KOENIG
step forward

UPGRADE STRATEGIES

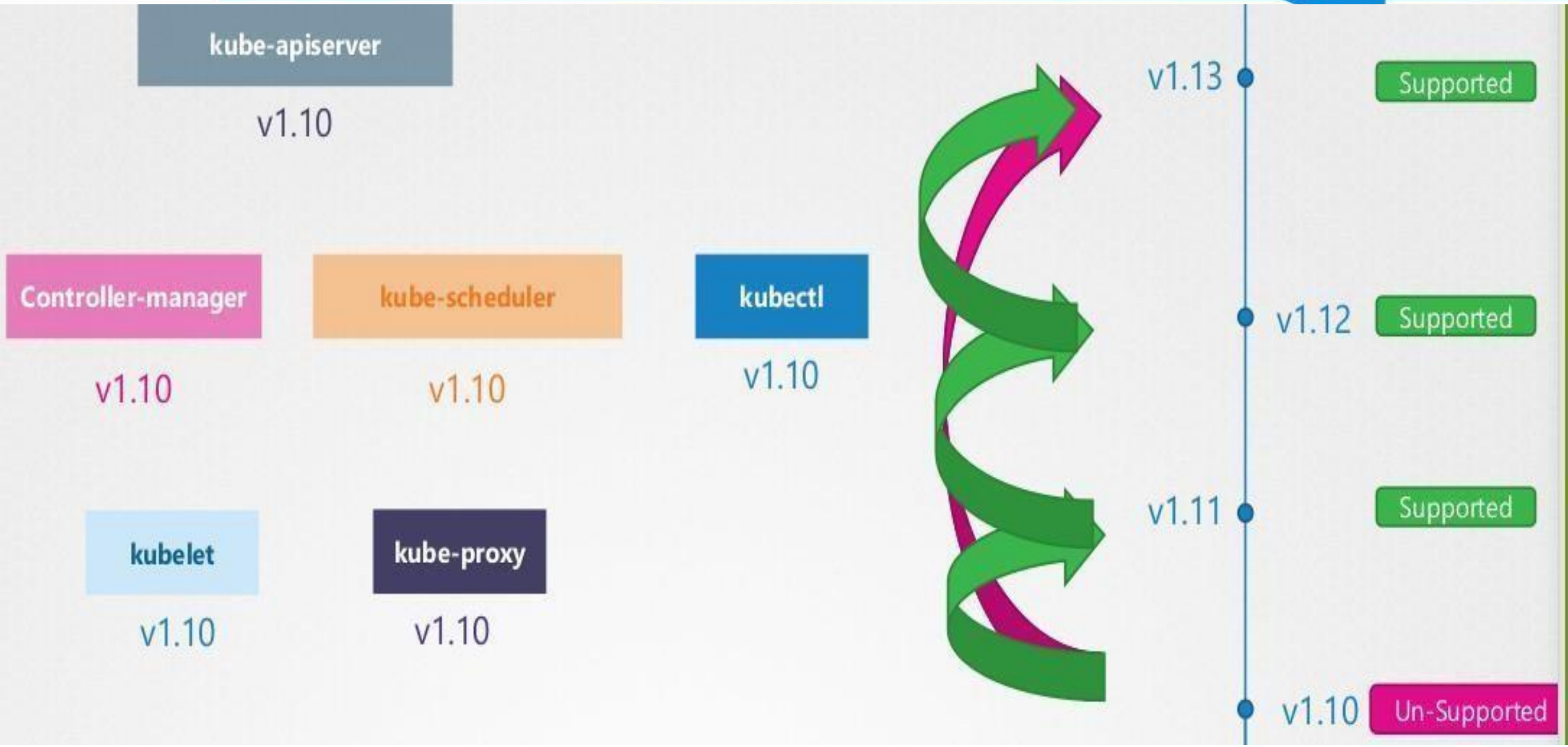
1) upgrade all nodes at once -> but then your pods will be down.

2) upgrade one at a time-> first drain then upgrade. in this case there will be no downtime.

3) add a new upgraded node -> then transfer one nodes data to upgraded node then delete oldest then do same with all

KOENIG
step forward

CLUSTER UPGRADE PROCESS



KOENIG
step forward

UPGRADE CLUSTER VERSION

- kubernetes supports last 3 release that means let's say we have v.12 that means it will support v1.12,v1.11.,v1.10

- we can not directly upgrade from v1.10 to v1.13 we need to upgrade first to v1.11 then v1.12 then v1.13

with kubeadm->

#kubeadm upgrade plan -> to check current versions and what versions are available(first do in the master)

#kubeadm upgrade apply v1.13.4 -> to upgrade with version 1.13

KOENIG
step forward

UPGRADE MASTER(UBUNTU)

1) #apt-get upgrade -y kubeadm=1.12.0-00
2) #kubeadm upgrade apply v1.12.0
3) #kubectl get nodes -> to check version(but you will see it's not upgrades so we need to upgrade kubelet also)
4) #apt-get upgrade -y kubelet=1.12.0-00
5) #systemctl restart kubelet
6) #kubectl get nodes -> now it's uptaded do on all nodes

now for node upgradation->

do on master->

#kubectl drain node-1

UPGRADE NODE

8) #apt-get upgrade -y kubeadm=1.12.0-00
9) #apt-get upgrade -y kubelet=1.12.0-00
10) #kubeadm upgrade node config --kubelet-version v1.12.0
11) #systemctl restart kubelet

Do on master->

11) #kubectl uncordon node-1

do same for another nodes as above.

KOENIG
step forward

STATIC POD

What is Static Pod?

Static Pods are managed directly by the kubelet daemon on a specific node, without the API server observingthem.

Config file: /var/lib/kubelet/config.yaml

KOENIG
step forward

Creating Static Pod

```
# Run this command on the node where kubelet is running
mkdir /etc/kubelet.d/
cat <<EOF >/etc/kubelet.d/static-web.yaml
apiVersion: v1
kind: Pod
metadata:
  name: static-web
  labels:
    role: myrole
spec:
  containers:
    - name: web
      image: nginx
      ports:
        - name: web
          containerPort: 80
          protocol: TCP
EOF
```

KOENIG
step forward

ETCD Backup

KOENIG
step forward

ETCD

etcd is a consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.

KOENIG
step forward

Take Backup

```
[root@localhost ~]# #yum install epel-release -y
[root@localhost ~]# #yum install etcd -y
[root@localhost ~]# ETCDCTL_API=3 etcdctl --endpoints https://127.0.0.1:2379 --cacert /etc/kubernetes/pk
i/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/healthcheck-client.crt --key /etc/kubernetes/pki/etcd/heal
thcheck-client.key snapshot save cluster_etcd.db
Snapshot saved at cluster_etcd.db
[root@localhost ~]#
```

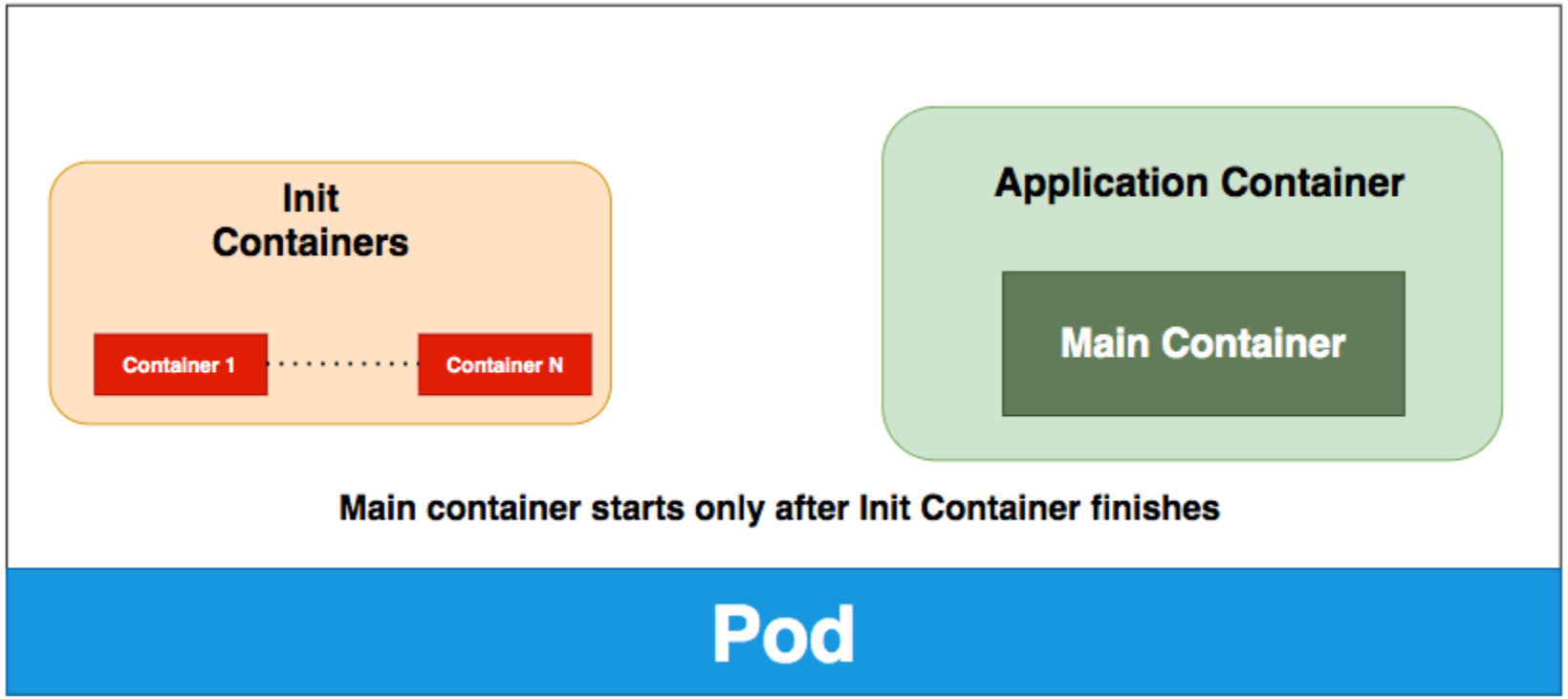
KOENIG
step forward

Check Backup

```
[root@master ~]# ETCDCTL_API=3 etcdctl --endpoints https://127.0.0.1:2379 --cacert /etc/kuberne
tes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/healthcheck-client.crt --key /etc/kubernete
s/pki/etcd/healthcheck-client.key snapshot status cluster_etcd.db
85bd211, 101185, 1876, 5.7 MB
[root@master ~]#
```

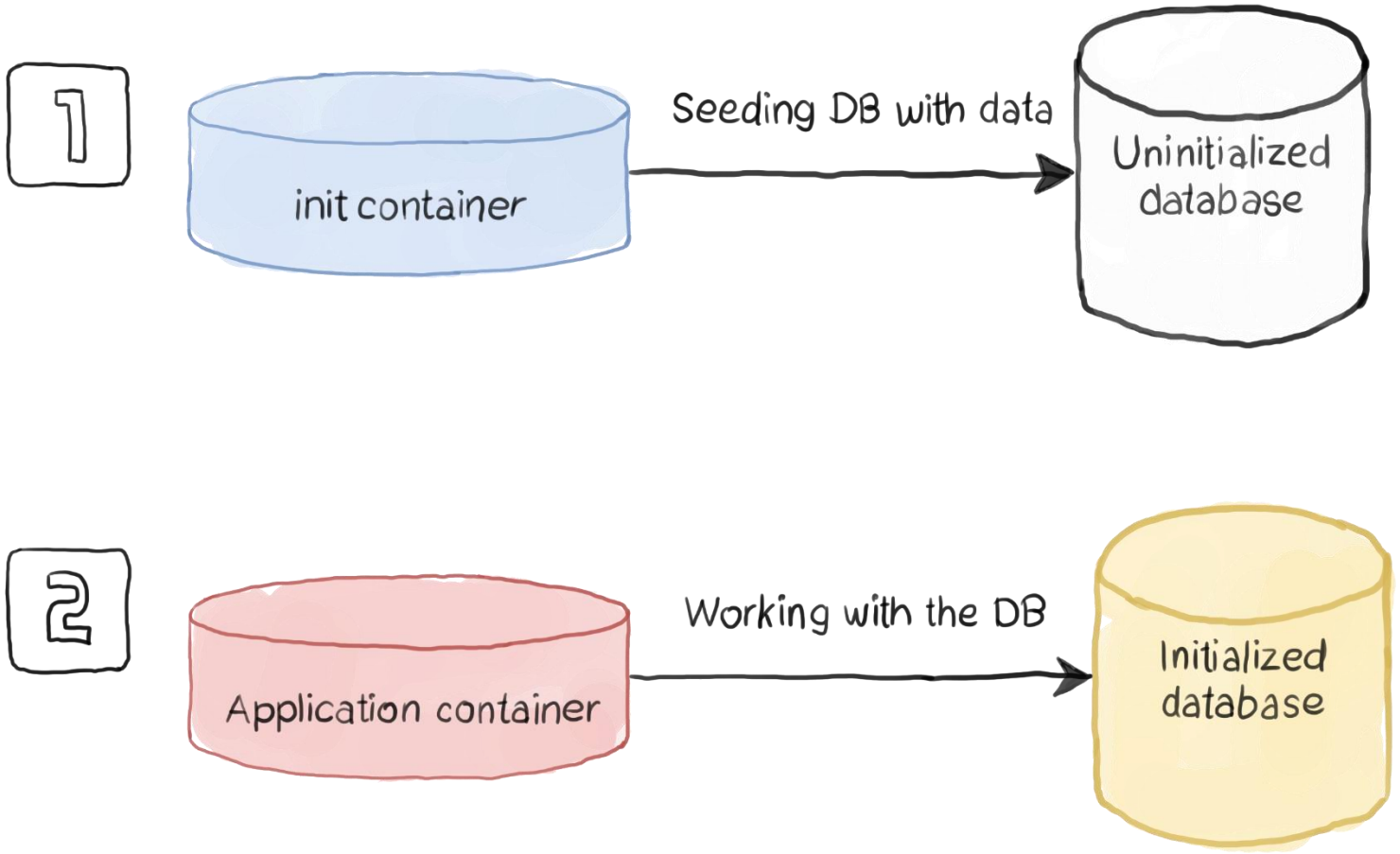
Init Containers

A Pod can have multiple containers running apps within it, but it can also have one or more init containers, which are run before the app containers are started.



KOENIG
step forward

Example



KOENIG
step forward

Example of Init Container

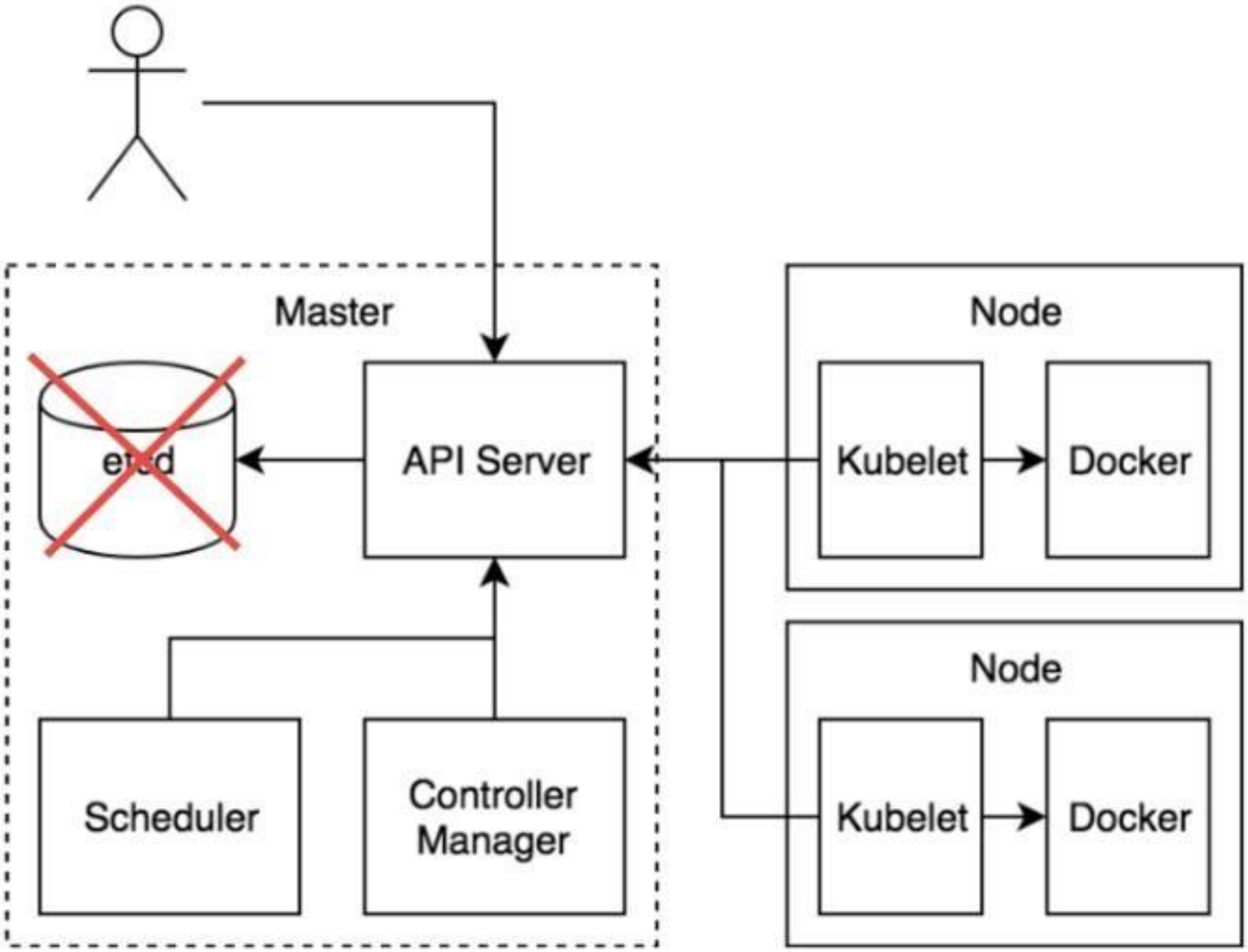
```yaml
apiVersion: v1
kind: Pod
metadata:
 name: test-sql-pod
spec:
 containers:
 - name: container1
   image: mysql
   env:
    - name: MYSQL_ROOT_PASSWORD
      value: mypass
   volumeMounts:
   - name: mysqldb
     mountPath: /mydb
 initContainers:
 - name: initcontainer
   image: busybox:1.28
   command: ["wget","-O","/mydb/docker.repo","https://download.docker.com/linux/centos/docker-ce.repo"]
   volumeMounts:
   - name: mysqldb
     mountPath: /mydb
 volumes:
 - name: mysqldb
   emptyDir: {}
```

KOENIG
step forward

Troubleshooting

KOENIG
step forward

Cluster Backing Storage (etcd) lost



Note

In a real cluster, this would be caused by etcd crashing or data corruption. In the lab we'll simulate with the command below.

#docker pause k8s_etcd_etcd-master<tab to autocomplete>

KOENIG
step forward

Symptoms

- apiserver should fail to come up
- kubelet will not be able to reach it but will continue to run existing pods
- manual recovery necessary before apiserver is restarted

KOENIG
step forward

Let’s try to list Nodes

```
[root@master ~]# docker pause k8s_etcd_etcd-master_kube-system_1192c804e7b913a442898a0ba829cd80_0
k8s_etcd_etcd-master_kube-system_1192c804e7b913a442898a0ba829cd80_0
[root@master ~]# kubectl get nodes
Error from server: rpc error: code = Unavailable desc = transport is closing
[root@master ~]#
```

Troubleshooting

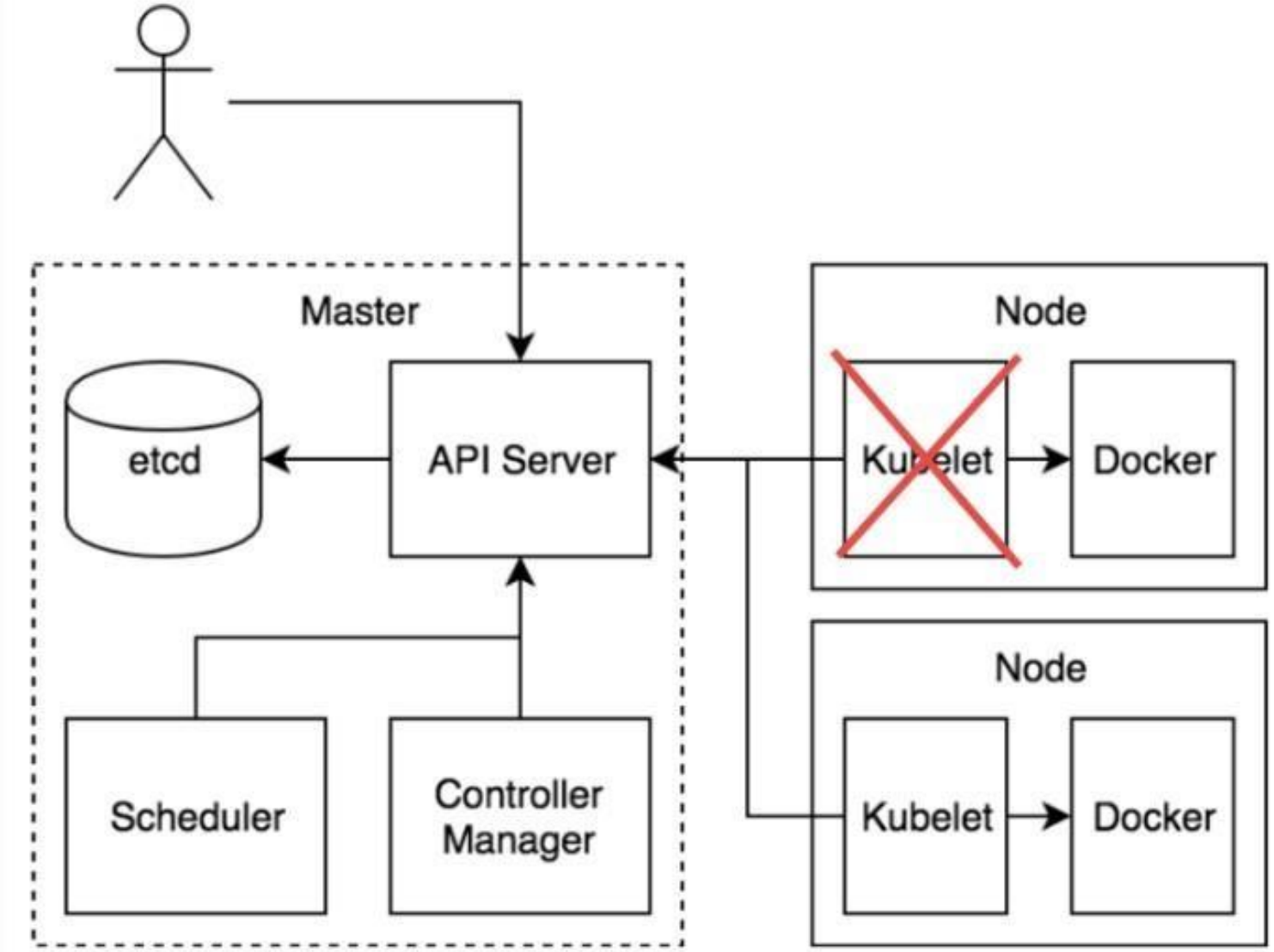
#journalctl -u kubelet –f

#systemctl status kubelet

#docker ps

KOENIG
step forward

Kubelet is not running on a worker



KOENIG
step forward

Note

In a real cluster, this would be caused be worker node going offline or kubelet crashing

#systemctl stop kubelet (on worker1)

Symptoms

- unable to stop, update, or start new pods, services, replication controller

KOENIG
step forward

Check Nodes

```
[root@master ~]# kubectl get nodes
NAME       STATUS     ROLES    AGE   VERSION
master     Ready      master   11h   v1.18.5
worker1    NotReady   <none>   11h   v1.18.5
worker2    Ready      <none>   11h   v1.18.5
[root@master ~]#
```

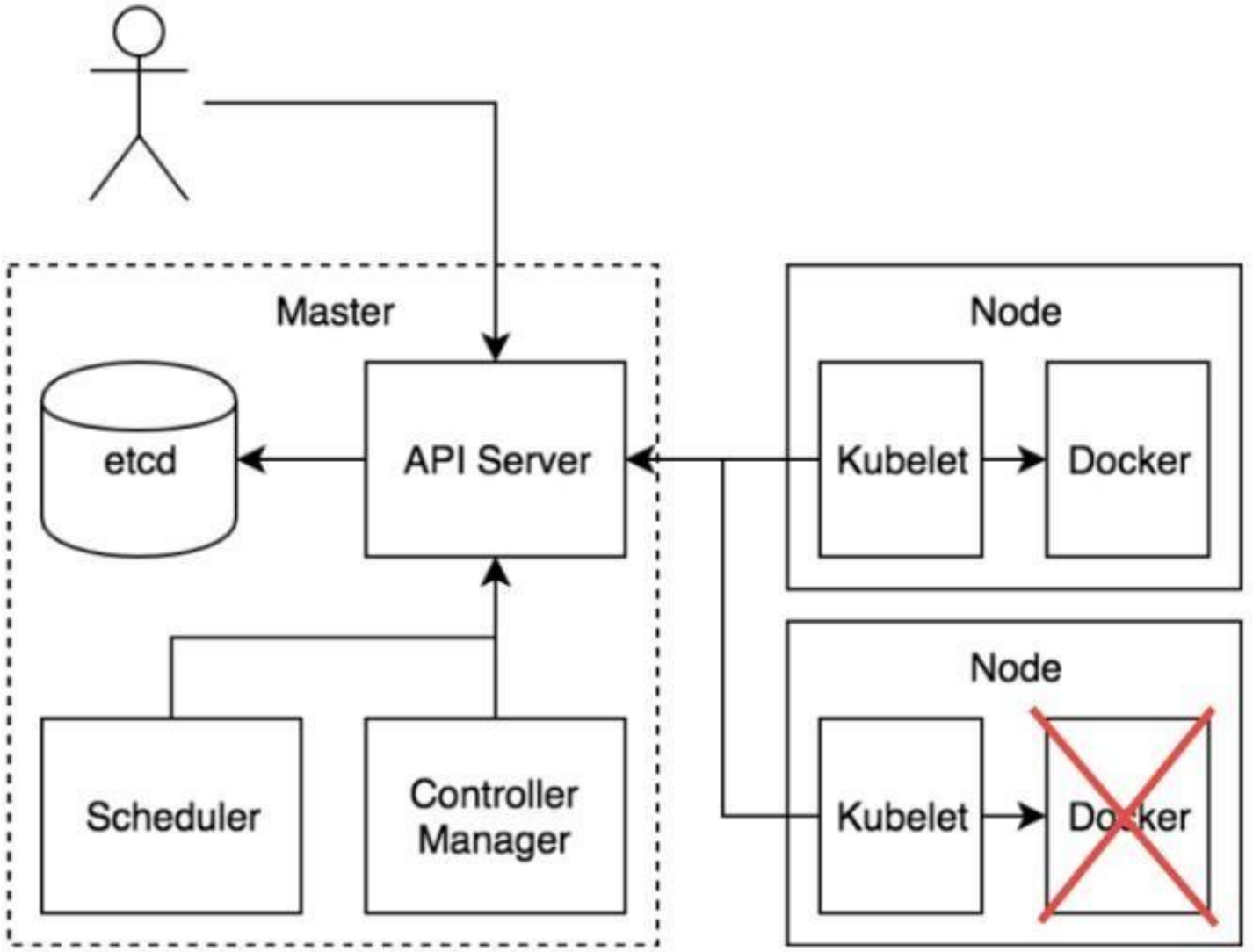
Let’s try to create a application

#kubectl run hello --image=nginx

This will not create on worker1?

KOENIG
step forward

Container Engine crashing on worker2



Note

In a real cluster, this would be caused be worker node going offline or docker crashing.

#systemctl stop docker (worker2)

Symptoms

- unable to create new workloads. Existing workloads are stopped.

KOENIG
step forward

Troubleshooting

Worker Node:

#journalctl -u kubelet –f

#docker ps

#systemctl status docker

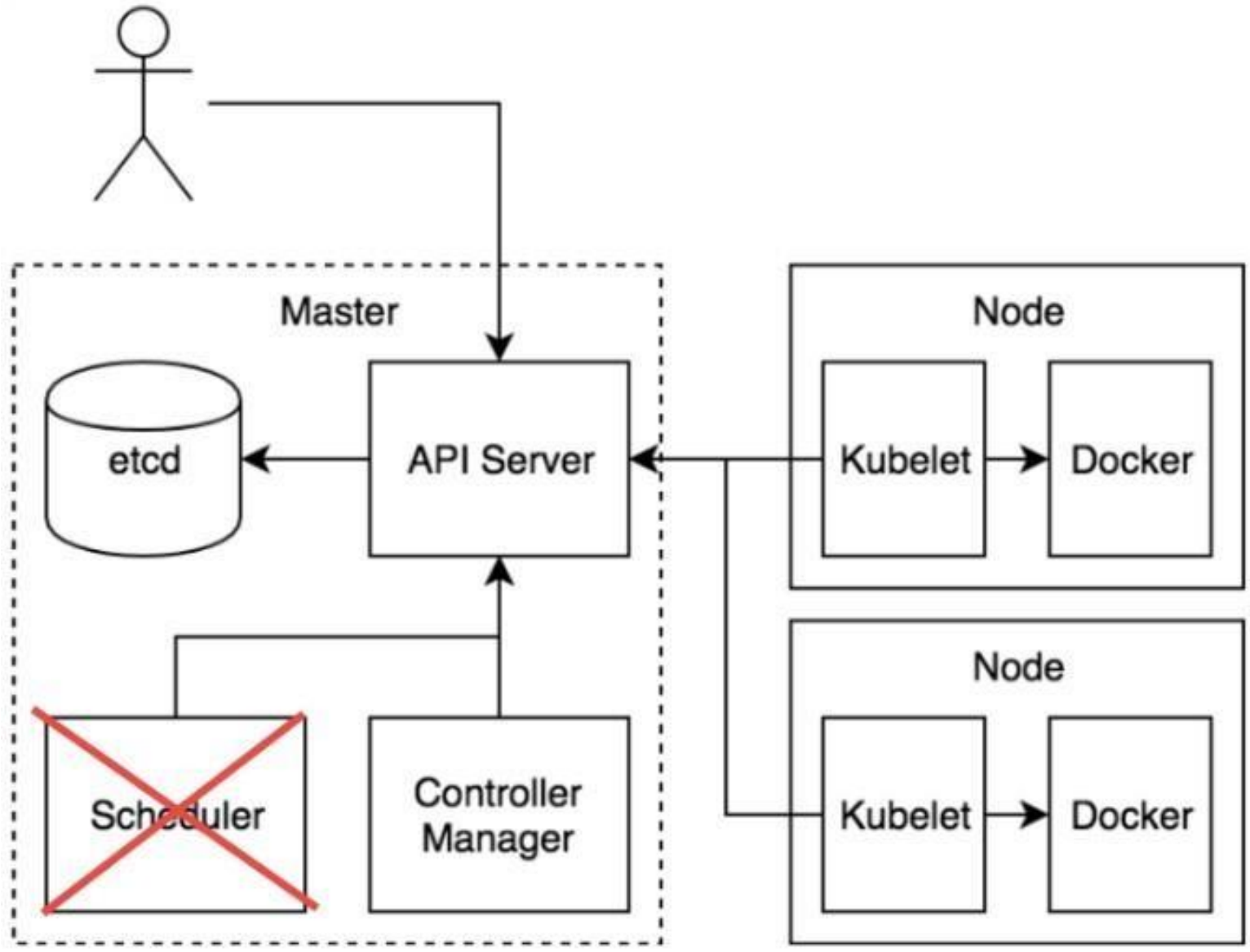
Master Node:

#kubectl get nodes

#kubectl describe node worker2

KOENIG
step forward

Scheduler service outage



Note

In a real cluster, this would be caused by kube-scheduler vm shutdown or kube-scheduler crashing.

#mv /etc/kubernetes/manifests/kube-scheduler.yaml /home/

KOENIG
step forward

Symptoms

- pods get created but will not be scheduled to a node

Let’s create a pod

#kubectl run pod1 –image=nginx

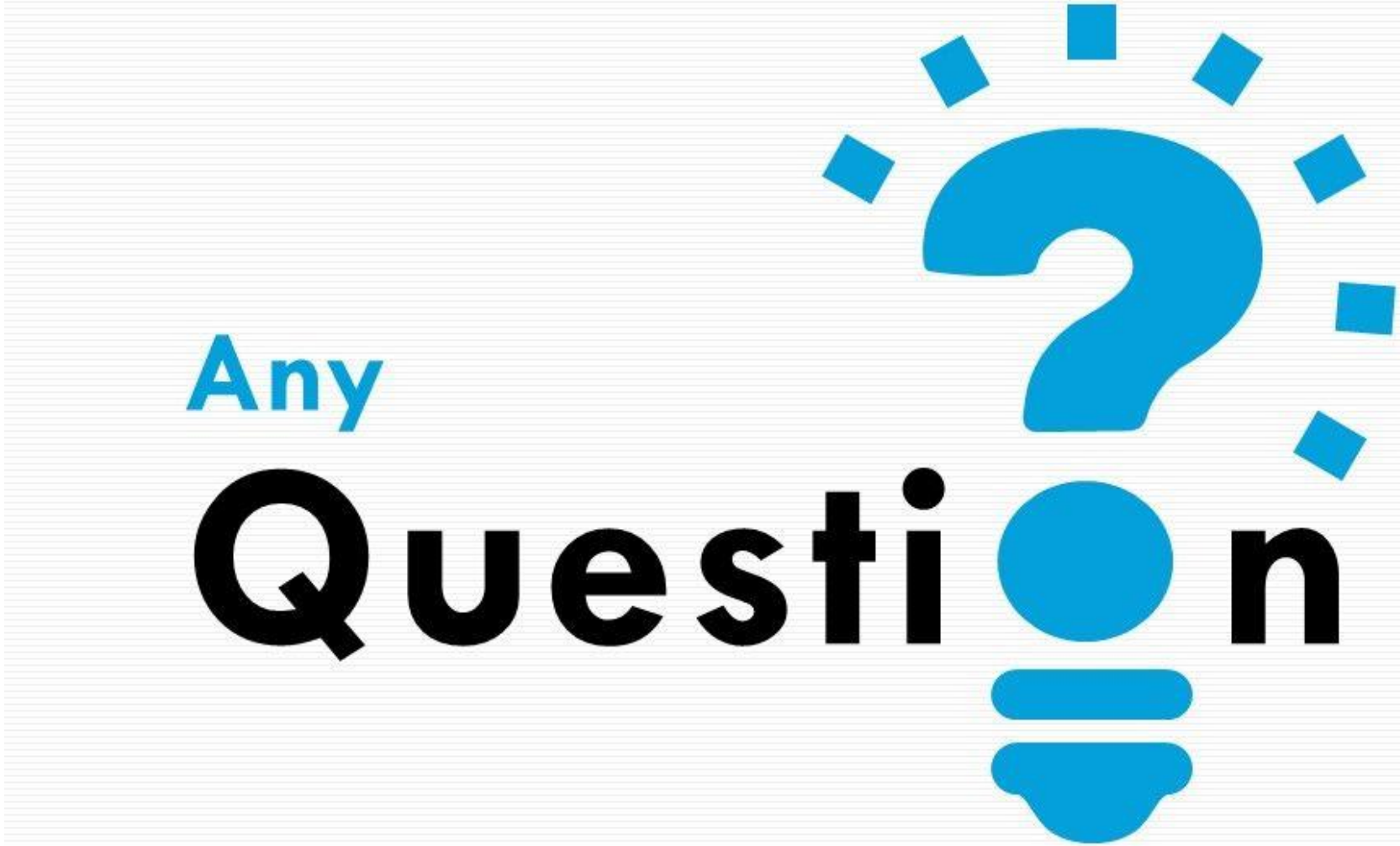
#kubectl get pod

Status will be pending

KOENIG
step forward

Restore

#mv /home/kube-scheduler.yaml /etc/kubernetes/manifests/


Any
Question

“Thank You”