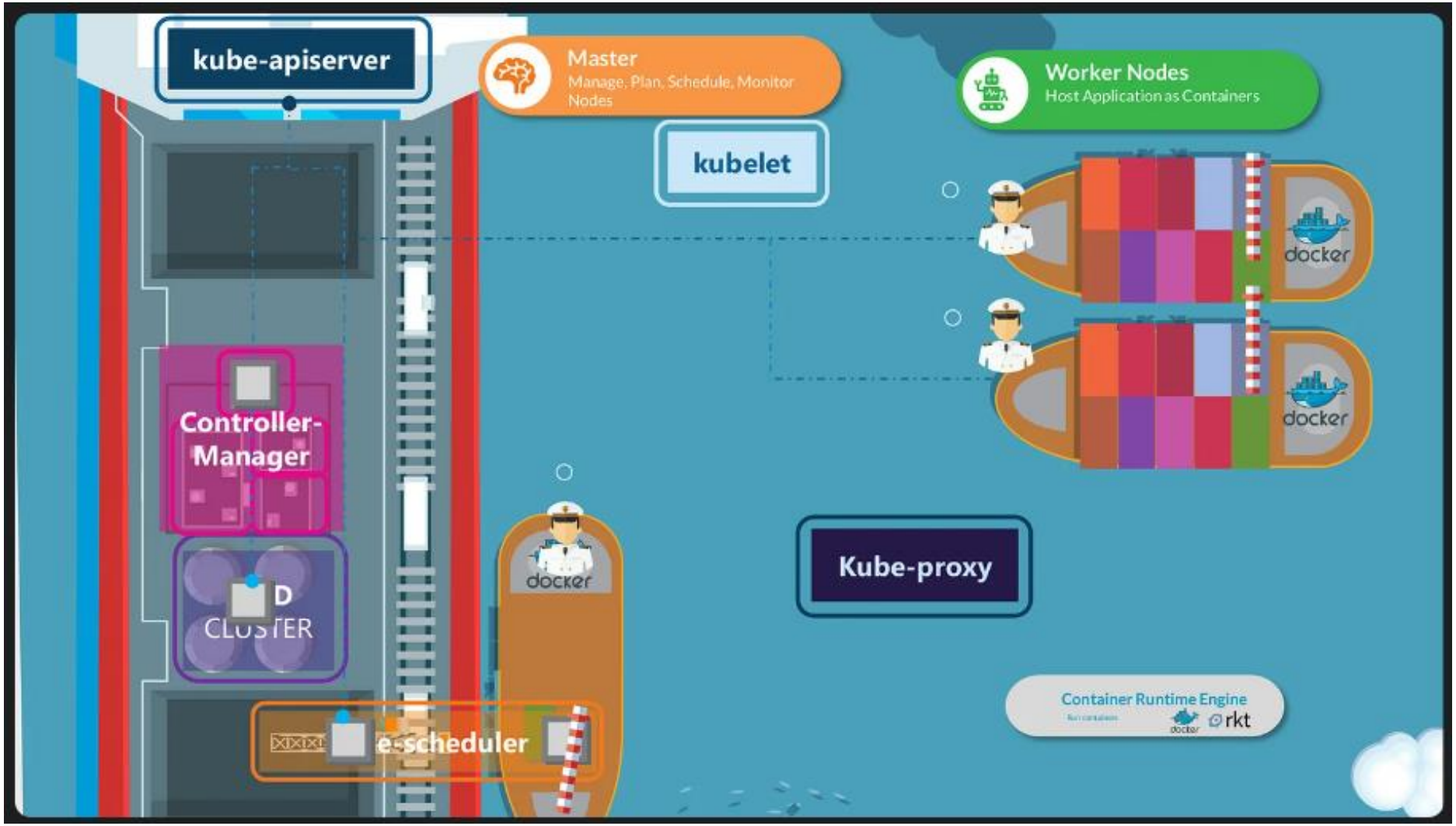
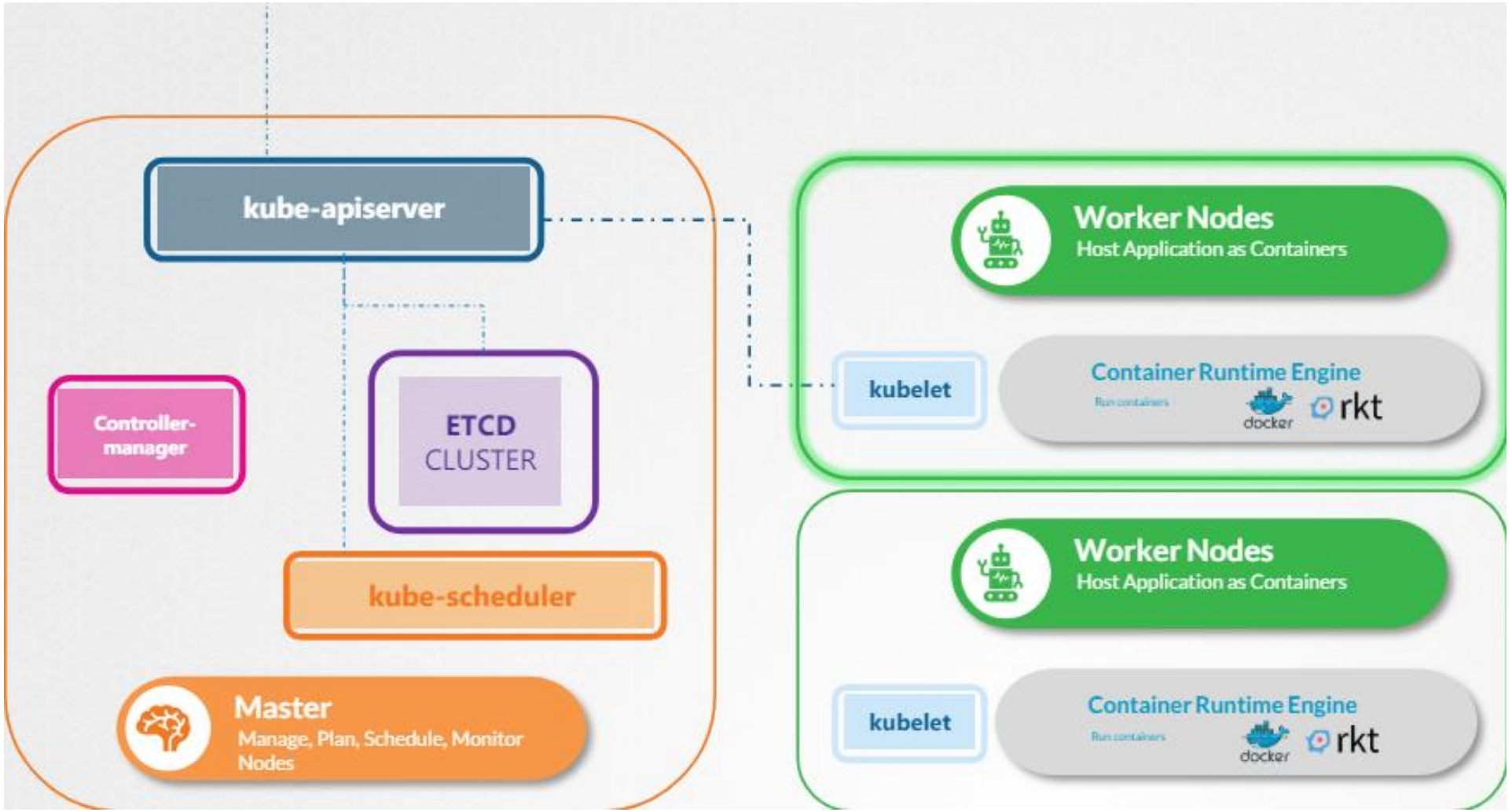
Certified Kubernetes Application Developer

Nobelprog UK

Kubernetes Cluster



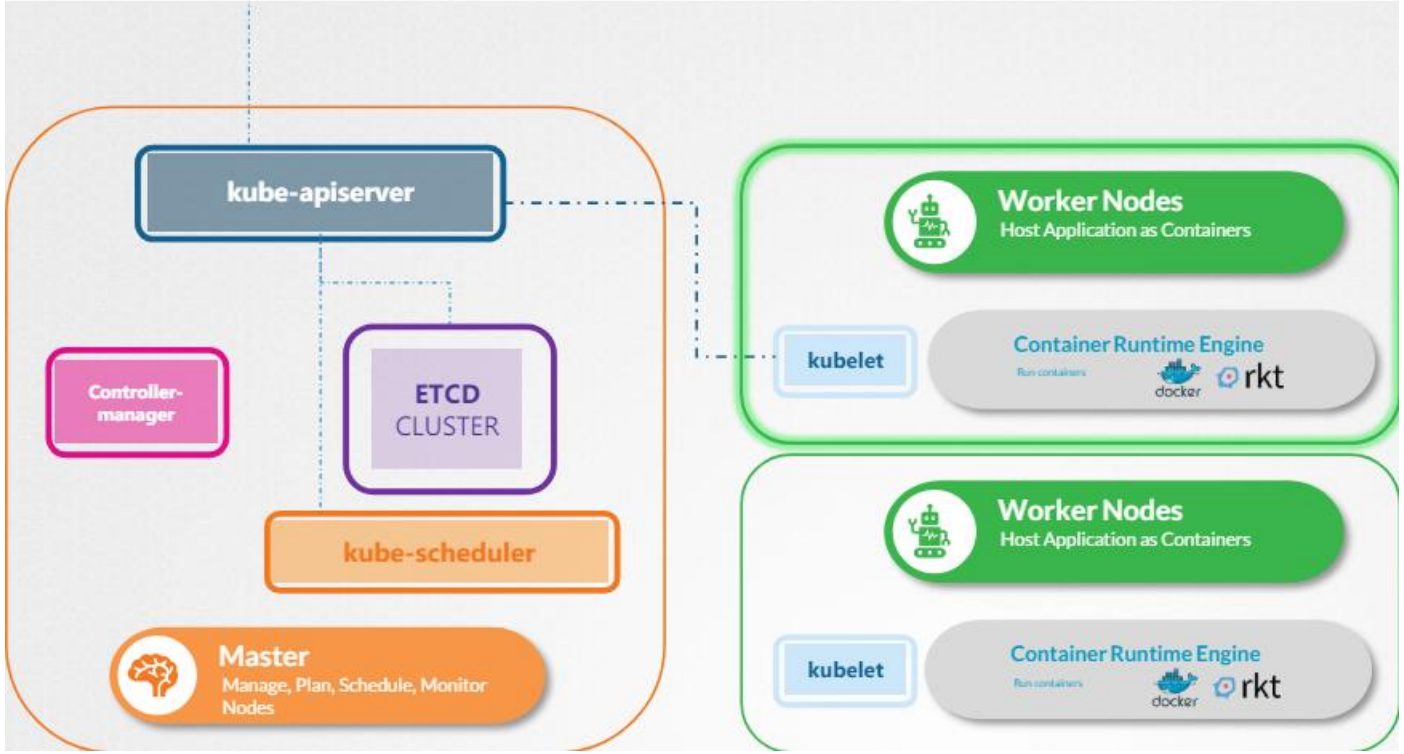
Kubernetes cluster



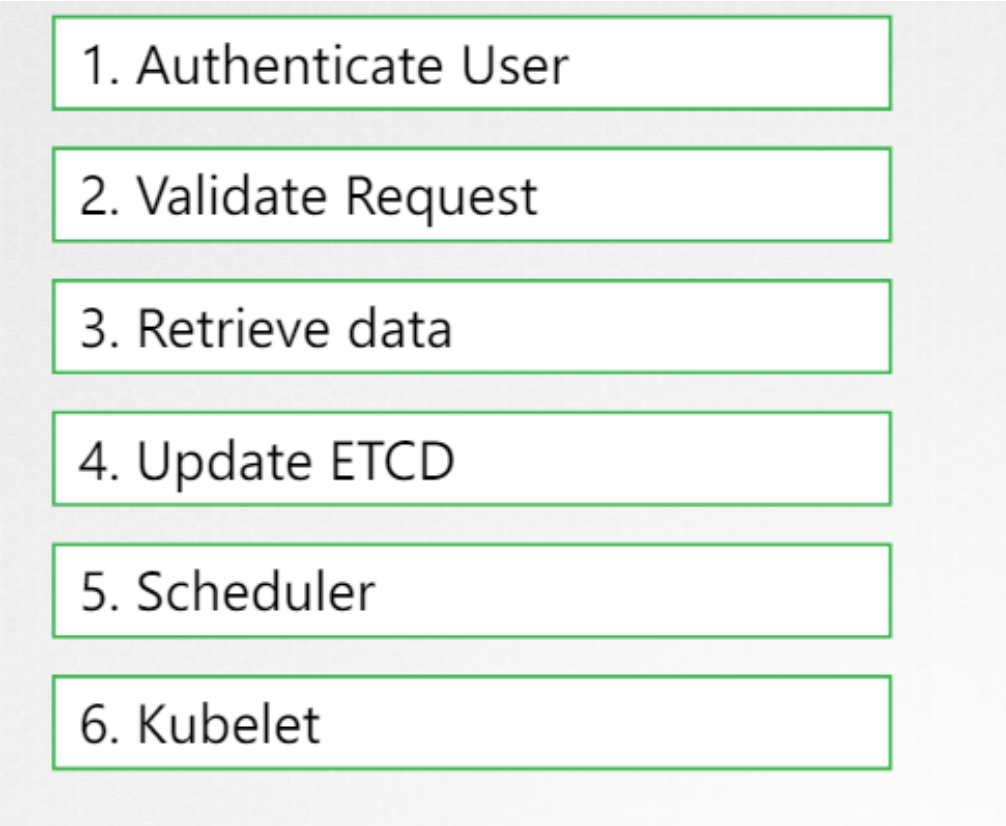
ETCD in Kubernetes

- ETCD is a distributed, reliable, key-value store that is simple, secure, and fast
- The etcd data store stores information regarding the cluster such as the nodes, pods, configs, secrets, accounts, roles, role bindings, and others
- - Version 2 commands: etcdctl backup, etcdctl cluster-health, etcdctl set, etcdctl set, etcdctl get
- - version 3 - etcdctl snapshot save, etcdctl endpoint health, etcdctl get, etcdctl put

Kube-API server



Kube-API server



Kube-API Server

- The API server creates a pod object without assigning it to a node. Updates the information in the etcd server, updates the user that the pod has been created
- The scheduler continuously monitors the API server and realizes that there is a new pod with no node assigned. The scheduler identifies the right node to place the new pod on and communicates that back to the kube-apiserver
- The API server then updates the information in the etcd cluster
- The API server then passes that information to the kubelet in the appropriate worker node

Kube-API Server

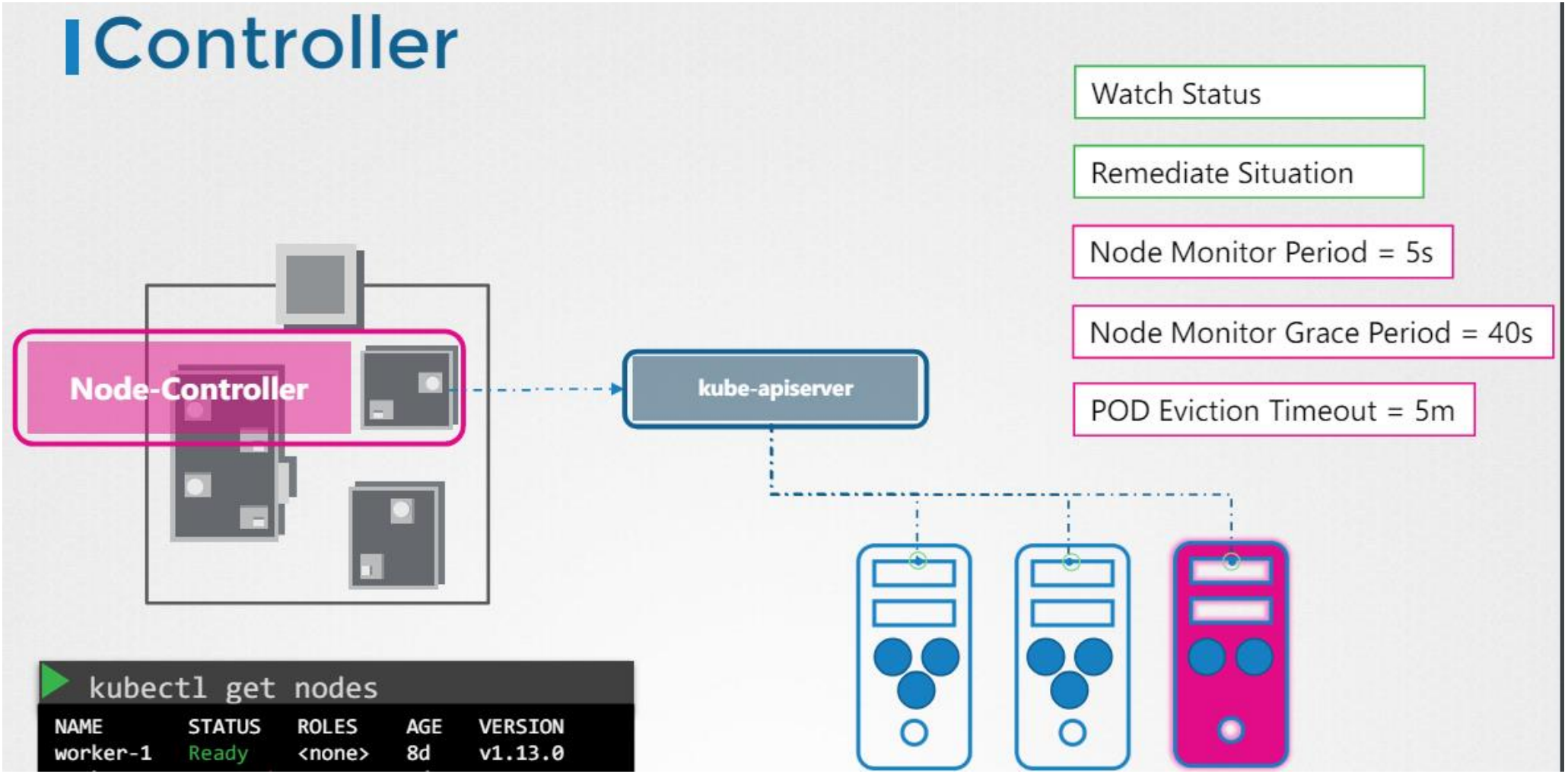
```
cat /etc/kubernetes/manifests/kube-apiserver.yaml
```

```
pec:
 containers:
 - command:
   - kube-apiserver
   - --authorization-mode=Node,RBAC
   - --advertise-address=172.17.0.32
   - --allow-privileged=true
   - --client-ca-file=/etc/kubernetes/pki/ca.crt
   - --disable-admission-plugins=PersistentVolumeLabel
   - --enable-admission-plugins=NodeRestriction
   - --enable-bootstrap-token-auth=true
   - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
   - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
   - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
   - --etcd-servers=https://127.0.0.1:2379
   - --insecure-port=0
   - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
   - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
   - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
   - --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt
   - --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key
   - --requestheader-allowed-names=front-proxy-client
   - --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt
   - --requestheader-extra-headers-prefix=X-Remote-Extra-
   - --requestheader-group-headers=X-Remote-Group
   - --requestheader-username-headers=X-Remote-User
```

Kube Controller-Manager

- Kube controller manager, manages various controllers in Kubernetes
- Controller is a process that continuously monitors the state of various components within the system and works towards bringing the whole system to the desired functioning state
- For example, the node controller is responsible for monitoring the status of the nodes and taking necessary actions to keep the applications running

Node Controller

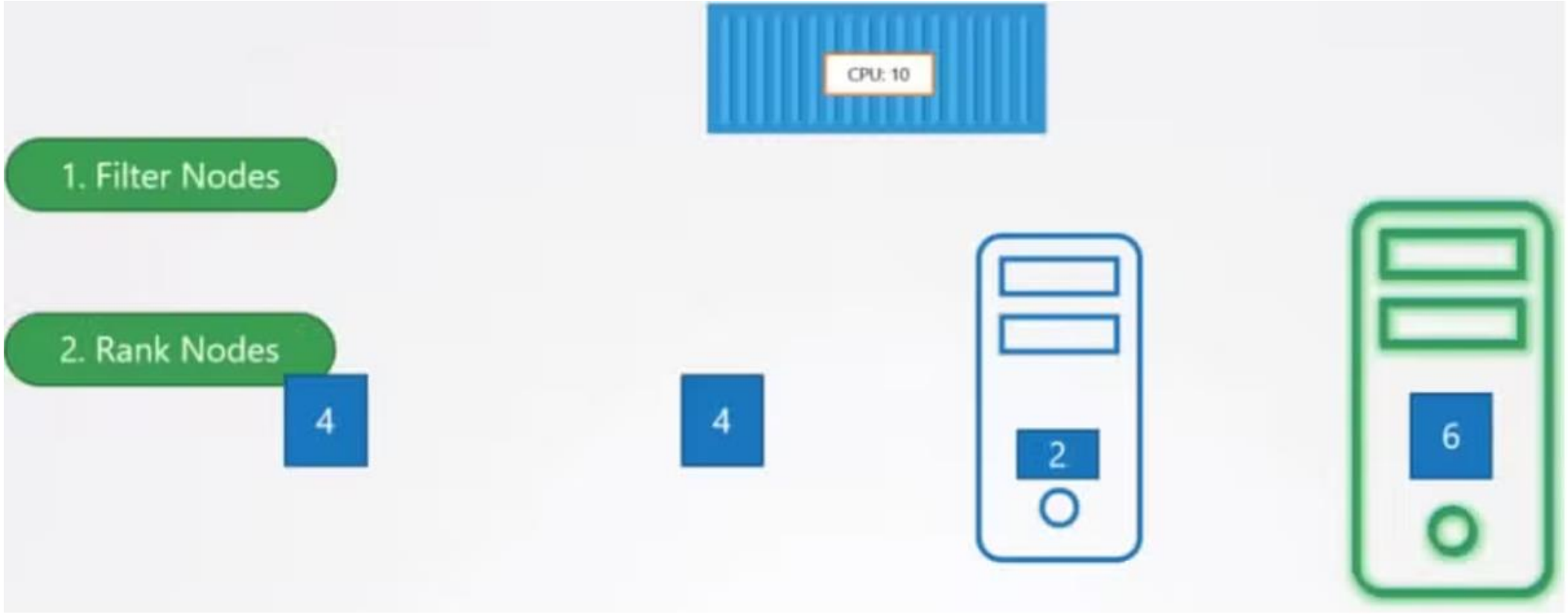


Kube Controller-Manager

```
cat /etc/kubernetes/manifests/kube-controller-manager.yaml
```

```
spec:
  containers:
  - command:
    - kube-controller-manager
    - --address=127.0.0.1
    - --cluster-signing-cert-file=/etc/kubernetes/pki/ca.crt
    - --cluster-signing-key-file=/etc/kubernetes/pki/ca.key
    - --controllers=*,bootstrapsigner,tokencleaner
    - --kubeconfig=/etc/kubernetes/controller-manager.conf
    - --leader-elect=true
    - --root-ca-file=/etc/kubernetes/pki/ca.crt
    - --service-account-private-key-file=/etc/kubernetes/pki/sa.key
    - --use-service-account-credentials=true
```

Kube Scheduler



Kube Scheduler

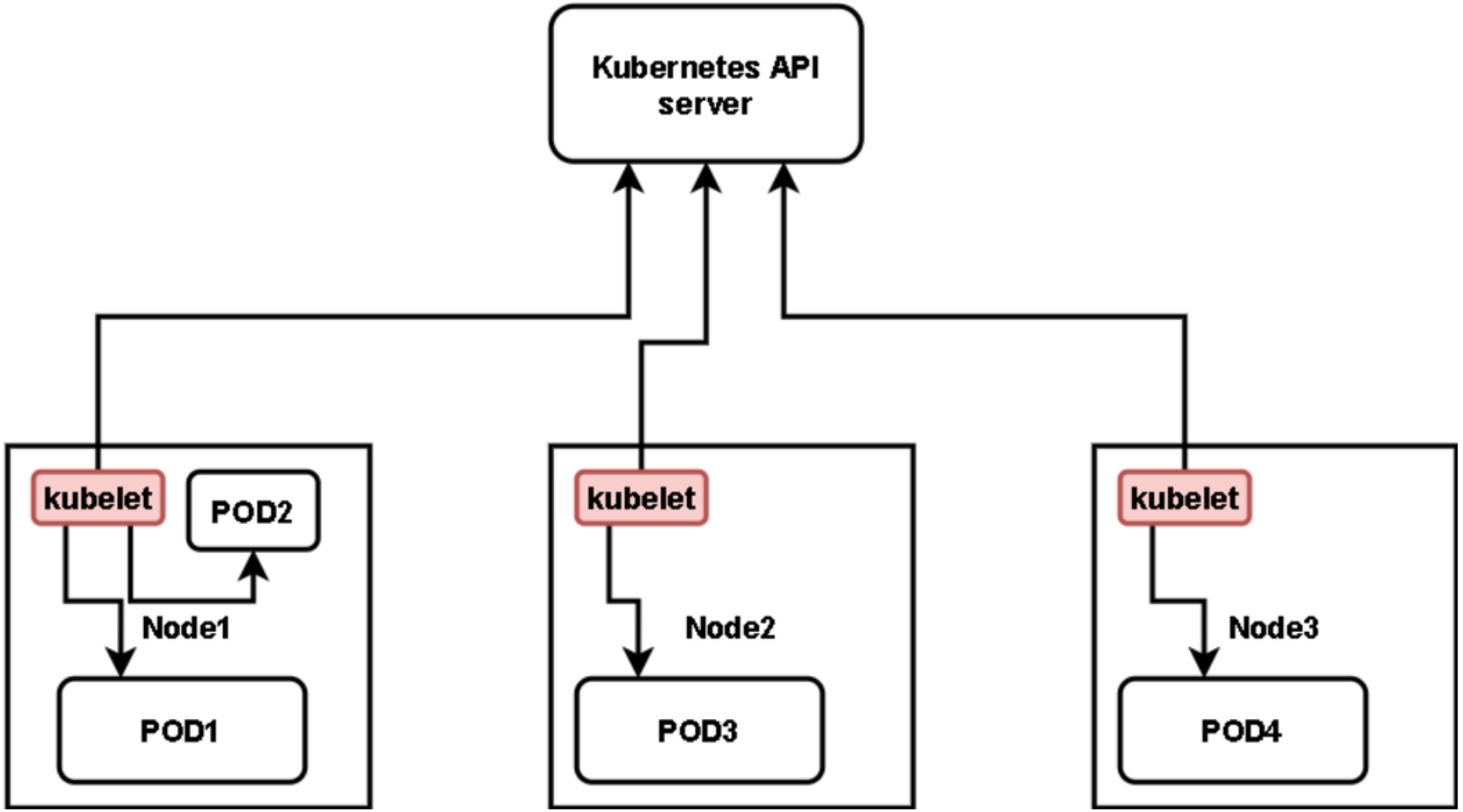
- The Kubernetes scheduler is responsible for scheduling pods on nodes
- scheduler is only responsible for deciding which pod goes on which node. It doesn't actually place the pod on the nodes. That's the job of the kubelet
- You may have pods with different resource requirements. It may have a set of CPU and memory requirements
- The scheduler ranks the nodes to identify the best fit for the pod. It uses a priority function to assign a score to the nodes on a scale of zero to 10

Kube Scheduler

```
cat /etc/kubernetes/manifests/kube-scheduler.yaml
```

```
spec:
  containers:
  - command:
    - kube-scheduler
    - --address=127.0.0.1
    - --kubeconfig=/etc/kubernetes/scheduler.conf
    - --leader-elect=true
```

Kubelet


Kubernetes API server
kubelet
POD2
Node1
POD1
kubelet
Node2
POD3
kubelet
Node3
POD4

Kubelet

- The kubelet in the Kubernetes worker node registers the node with a Kubernetes cluster
- The kubelet continues to monitor the state of the pod and containers in it and reports to the kube API server on a timely basis
- If you use the kubeadm tool to deploy your cluster,it does not automatically deploy the kubelet

Kubelet

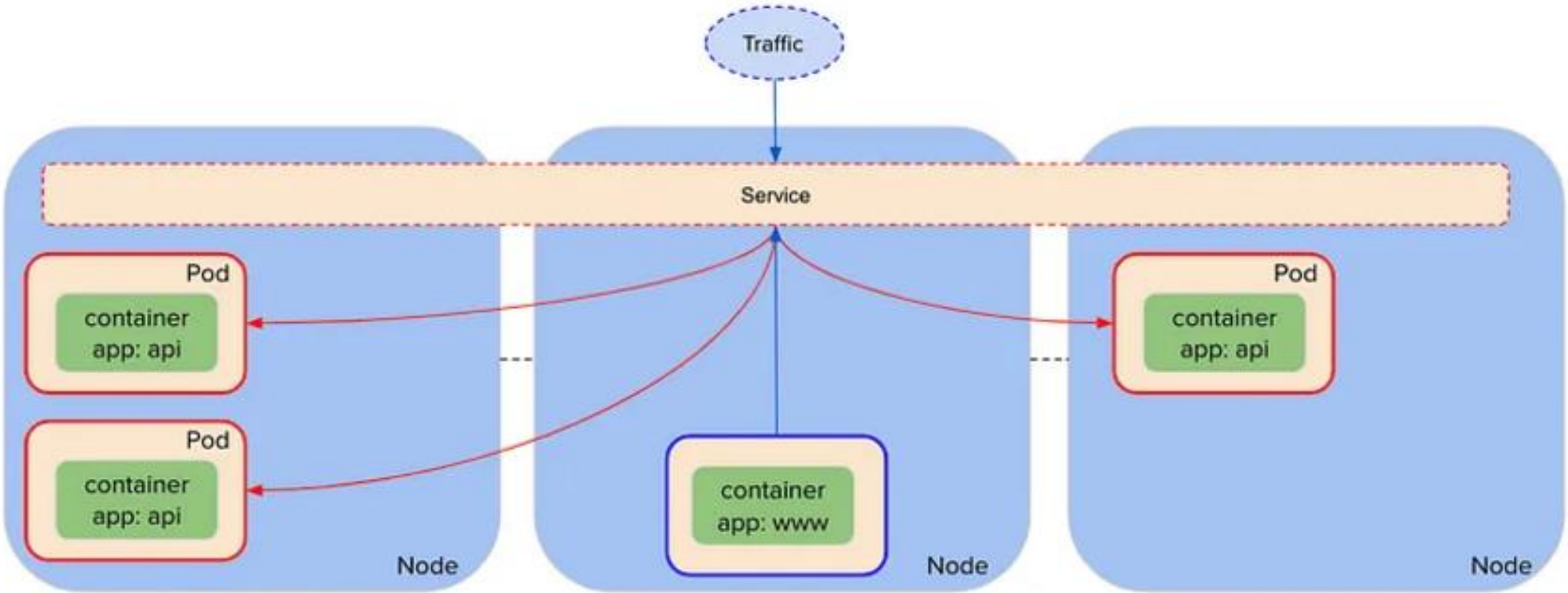
kubelet.service

```
ExecStart=/usr/local/bin/kubelet \\
  --config=/var/lib/kubelet/kubelet-config.yaml \\
  --container-runtime=remote \\
  --container-runtime-endpoint=unix:///var/run/containerd/containerd.sock \\
  --image-pull-progress-deadline=2m \\
  --kubeconfig=/var/lib/kubelet/kubeconfig \\
  --network-plugin=cni \\
  --register-node=true \\
  --v=2
```

Kube proxy

- Kube-proxy is a process that runs on each node in the Kubernetes cluster
- Its job is to look for new services, and every time a new service is created, it creates the appropriate rules on each node to forward traffic on those services to the pods
- Kube-proxy is deployed as a DaemonSet, so a single pod is always deployed on each node in the cluster

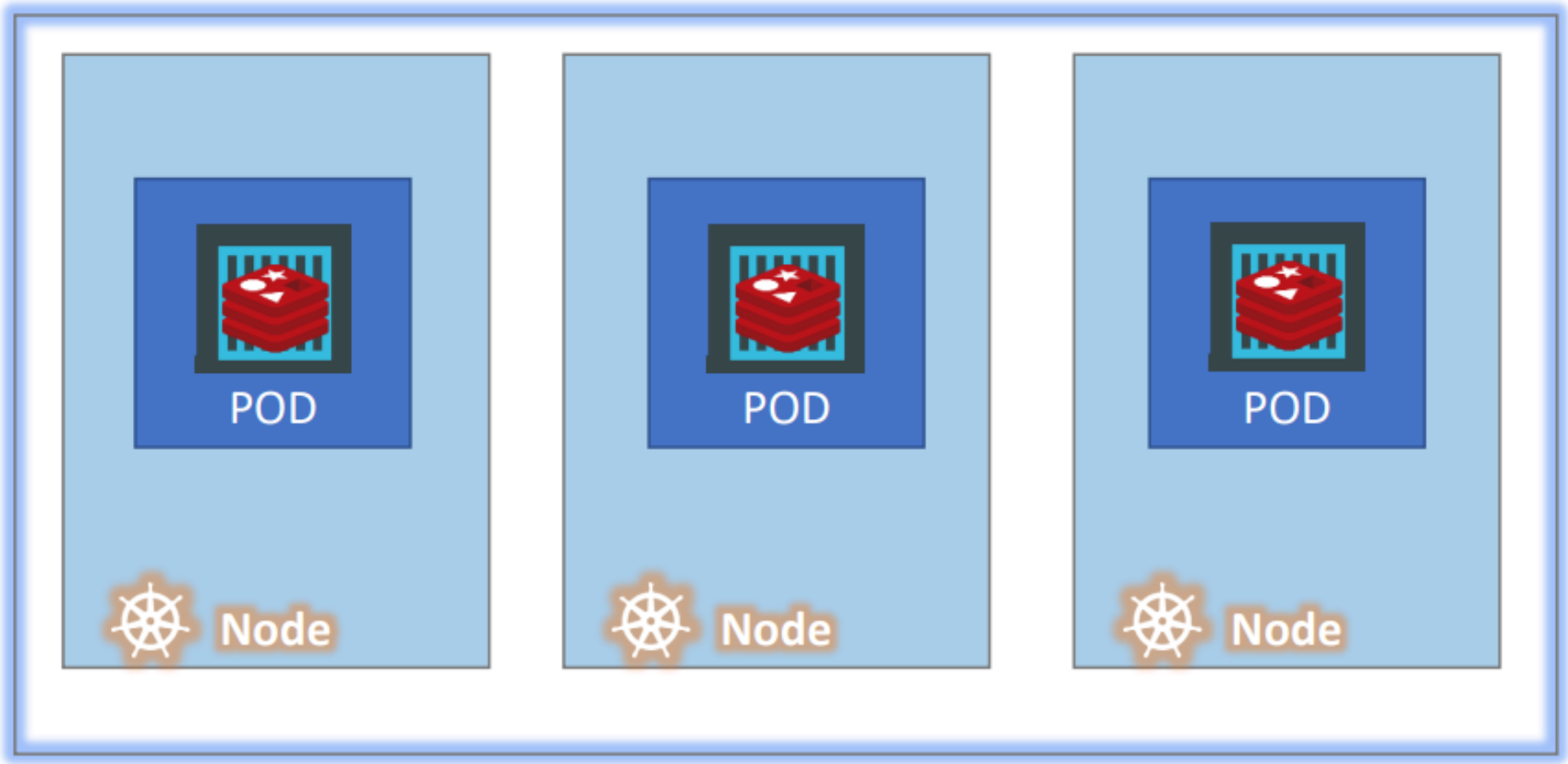
Kube proxy


Traffic
Service
Pod
container
app: api
Pod
container
app: api
container
app: www
Pod
container
app: api
Node
Node
Node

PODs

- Kubernetes does not deploy application containers directly on the worker nodes.
- The containers are encapsulated into a Kubernetes object known as pods
- pods usually have a one-to-one relationship with the containers but a single pod can have multiple containers
- Deploy a pod: kubectl run nginx --image nginx

PODs



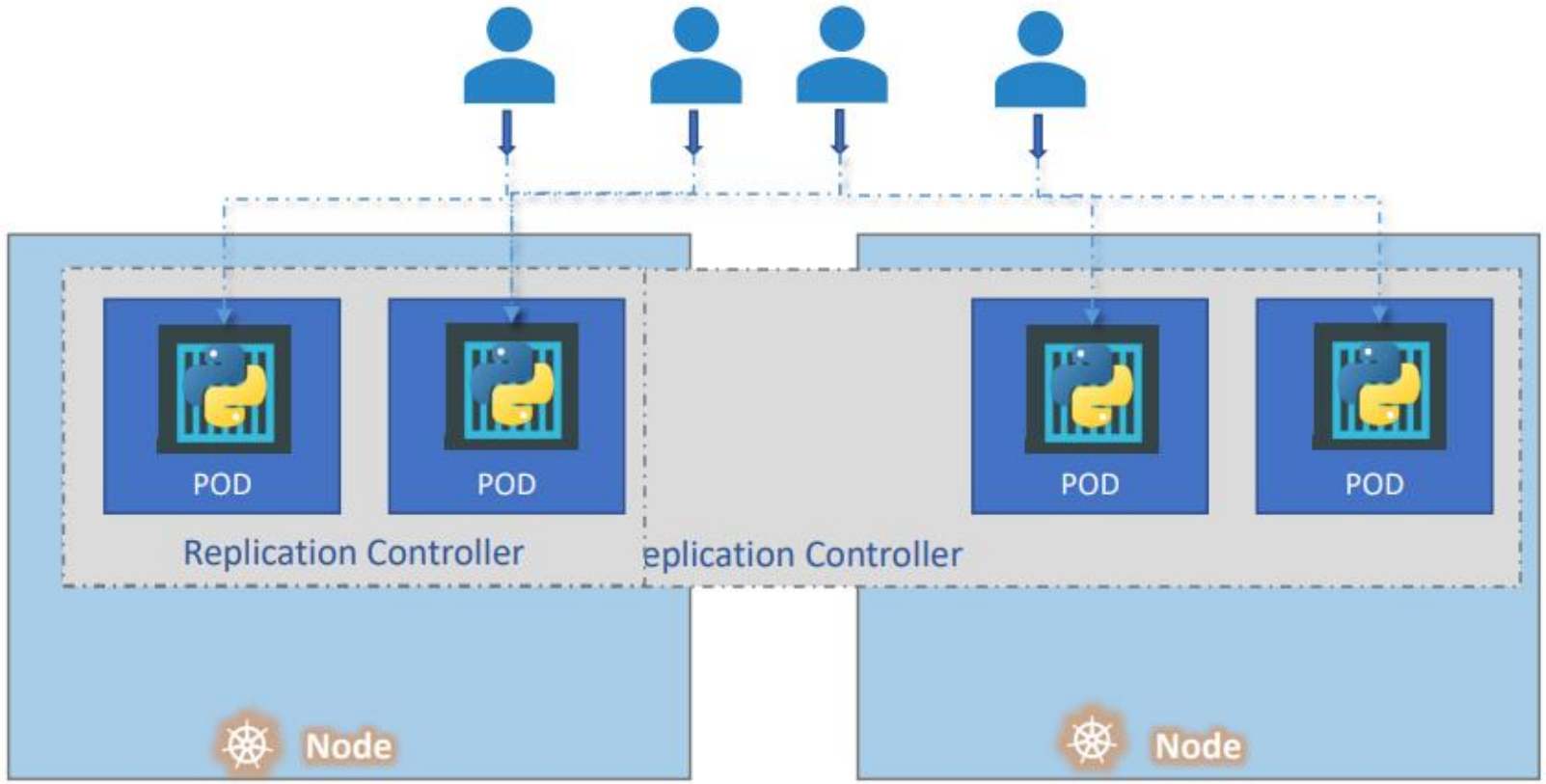
Creating PODs declaratively

- Kubernetes uses YAML files as inputs for the creation of objects such as pods, replicas, deployments, services etc
- 4 top level fields in every yaml are the API version, kind, metadata, and spec
- apiVersion: This is the version of the Kubernetes API
- kind: type of the object (pod, replica-set, deployment, service etc)
- metadata: data about the object like its name, labels etc. Its a dictionary
- spec: The specification section which is written as spec. Depending on the object we are going to create, this is where we would provide additional information to Kubernetes pertaining to that object

Replica sets

- Controllers are the brain behind Kubernetes. They are the processes that monitor Kubernetes objects and respond accordingly
- The Replication Controller helps us run multiple instances of a single pod in the Kubernetes cluster, thus providing high availability
- Replication Controller is the older technology that is being replaced by Replica Set
- version in replication controller is v1 vs apps/v1 in replica set

Replica sets

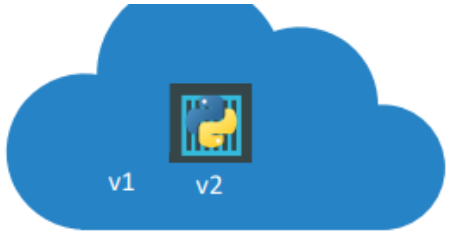


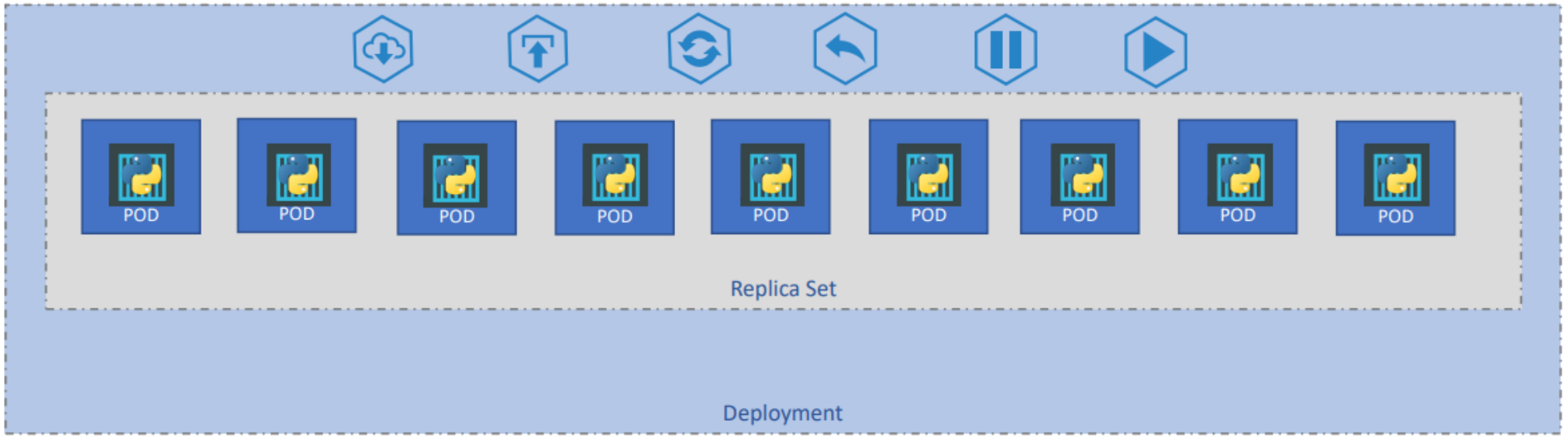
Replica sets

- kind in replication controller is ReplicationController vs ReplicaSet in replica set
- template is same for both but selector is optional in replication controller but mandatory in replica set
- The selector section helps the Replica Set identify what pods fall under it
- Upgrade replica set kubectl scale --replicas=6 replicaset myapp-replicaset

Deployment

Deployment

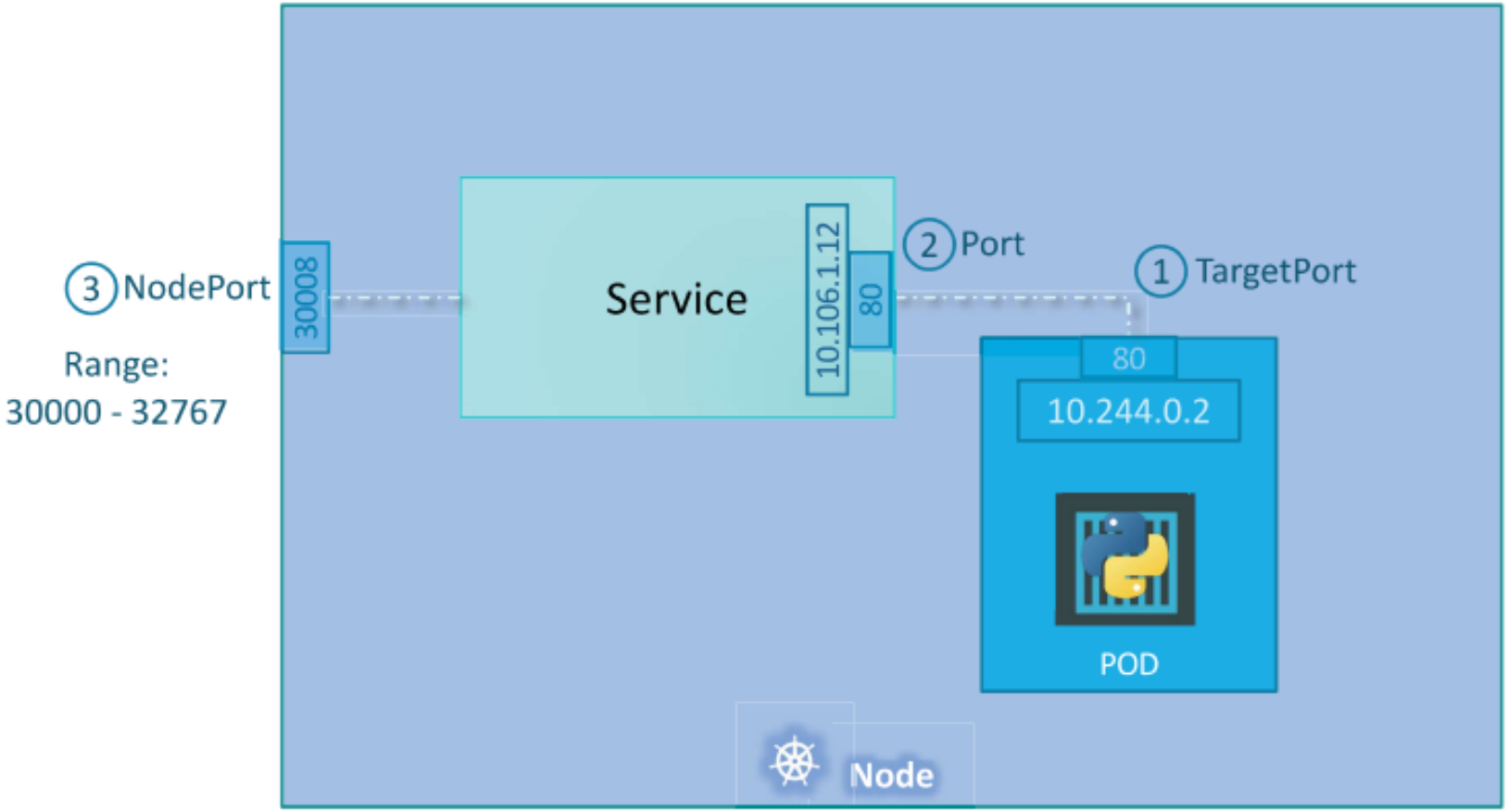




Deployment

- Deployment is a Kubernetes object that comes higher in the hierarchy of replica set and pods
- The deployment provides us with the capability to upgrade the underlying instances seamlessly using rolling updates, undo changes, and pause, and resume changes as required
- Imperative creation: kubectl create deployment --image=httpd:1.4-alpine httpd-frontend --replicas=3
- To edit the deployment: kubectl edit deployment web-app

Services: Nodeport



Services: Nodeport

- Kubernetes Services enable communication between various components within and outside of the application
- NodePort service is the service that listens to a port on the node and forward request to the Pods
- The service is like a virtual server inside the node. Inside the cluster it has its own IP address called ClusterIP of service
- target port is the port on the Pod where the actual container is running
- The second port is the port on the service itself
- Finally, node port is the port on the node itself
- When pods are distributed across multiple nodes k8s creates a service which spans across all the nodes

Service: ClusterIP

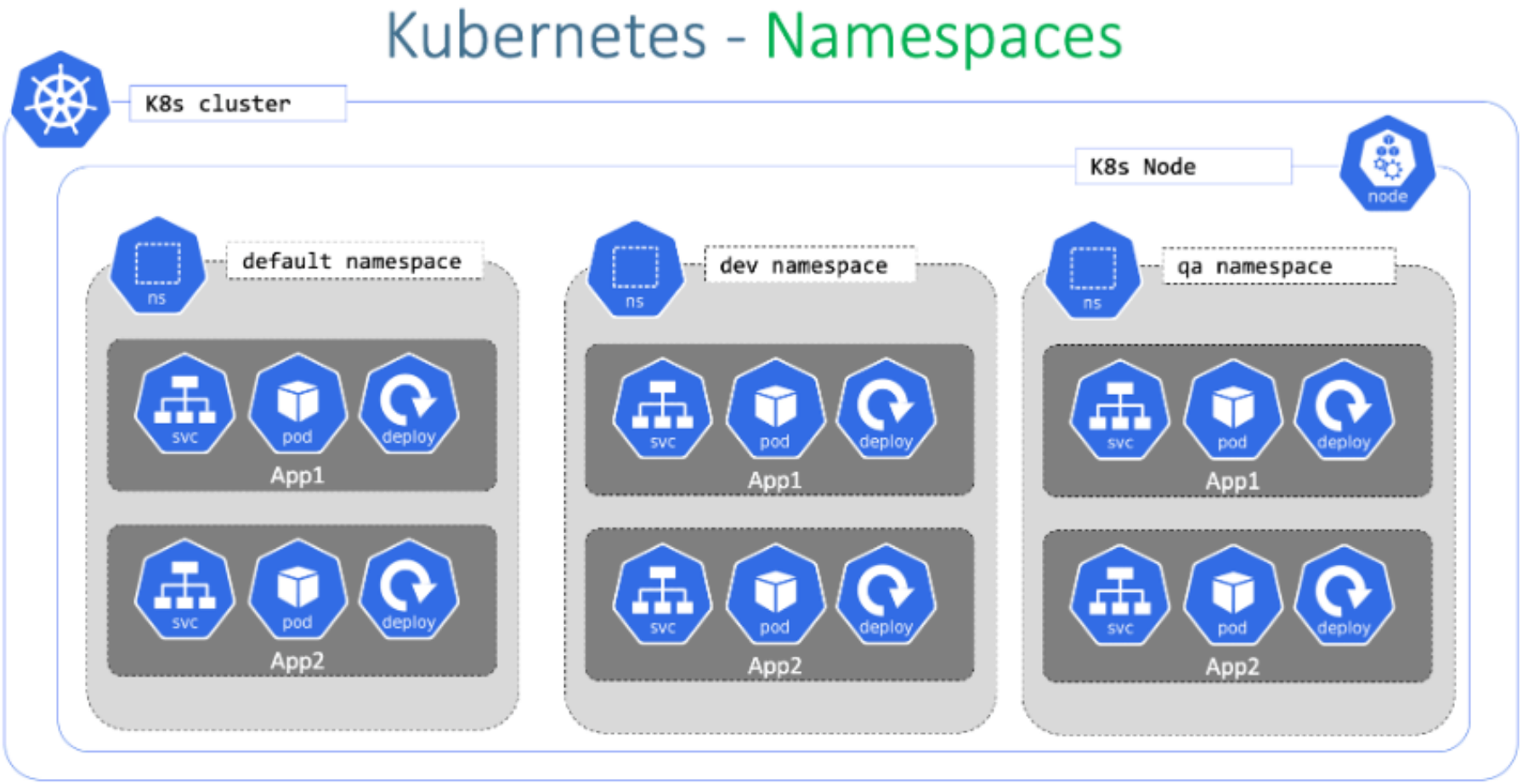
Service: ClusterIP

- We cannot rely on these IP addresses for internal communication between the applications because IP address assigned to PODs can change
- A Kubernetes ClusterIP service helps us group the pods together and provide a single interface to access the pods in a group
- This enables us to easily and effectively deploy a microservices based application on Kubernetes cluster

Service: LoadBalancer

- To load balance requests on the all nodes in the cluster where pods are running
- Only usable on cloud platforms with support of native load balancer (eg AWS, GCP, Azure etc)
- Service definition is similar to NodePort except type: LoadBalancer

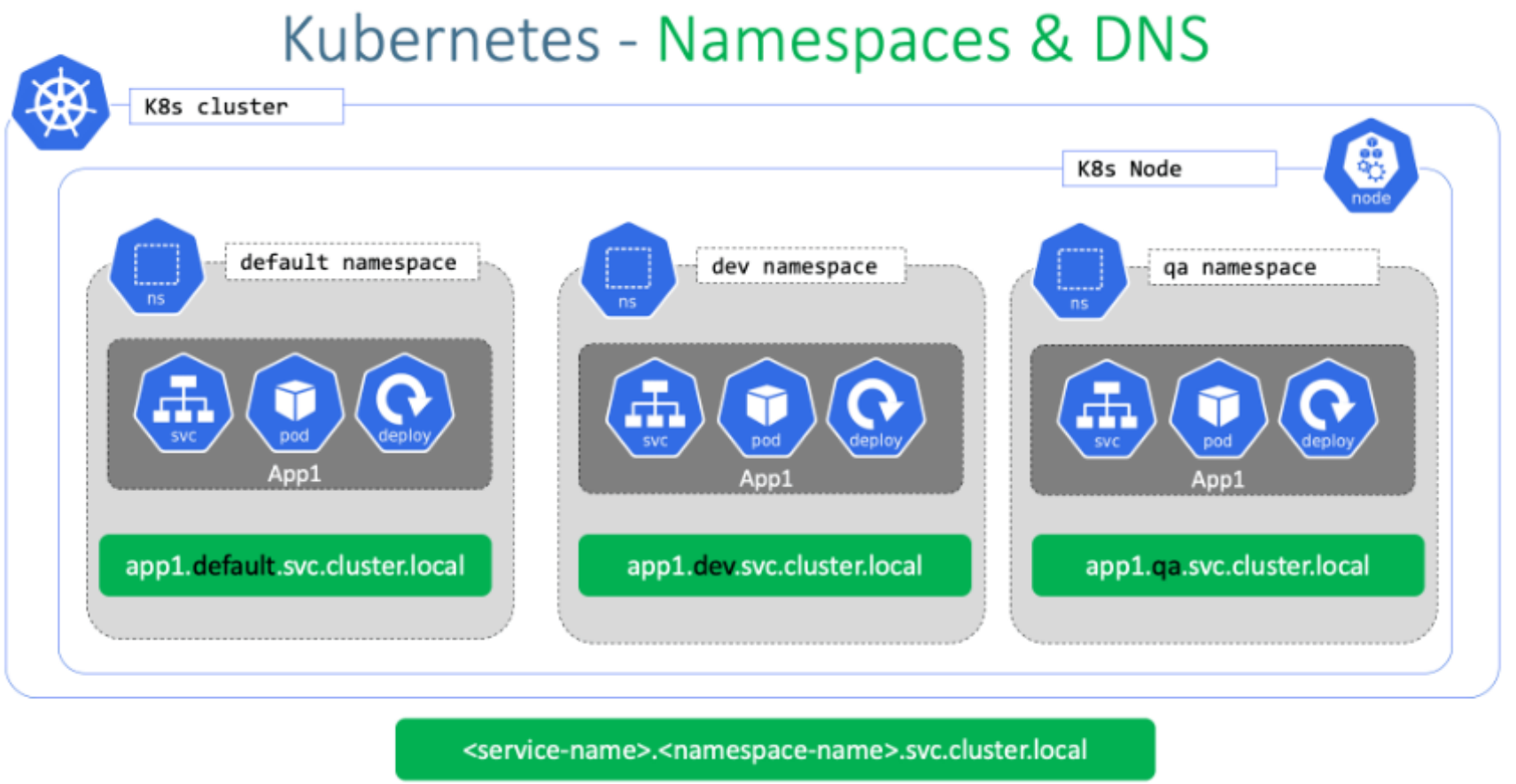
Namespace


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

Namespace

- Whatever Kubernetes objects we created such as pods, deployments, and services in our cluster. we have been doing all this within a name space. This name space is known as the default name space
- Kubernetes creates a set of pods and services for its internal purpose
- To isolate these from the user and to prevent you from accidentally deleting or modifying these services, Kubernetes creates them under another name space created at cluster startup named kube-system
- One example of using a namespace is if you wanted to use the same cluster for both dev and production environment, but at the same time, isolate the resources between them

Namespace


Kubernetes - Namespaces & DNS
K8s cluster
K8s Node
node
default namespace
dev namespace
qa namespace
ns
svc
pod
deploy
App1
app1.default.svc.cluster.local
app1.dev.svc.cluster.local
app1.qa.svc.cluster.local
<service-name>.<namespace-name>.svc.cluster.local

Namespace

- You can also assign quota of resources to each of these name spaces. That way, each name space is guaranteed a certain amount and does not use more than its allowed limit
- the resources within a namespace can refer to each other simply by their names
- kubectl get pods --namespace=kube-system
- To switch to a different namespace: kubectl config set-context $(kubectl config current-context) --namespace=dev
- To get pods from all namespace: kubectl get pods --all-namespaces

Scheduling: Manual scheduling

- To manually schedule a pod is to simply set the node name field to the name of the node in your pod specification file while creating the pod
- You can only specify the node name at creation time
- Kubernetes won't allow you to modify the node name property of a pod

Labels and Selectors

- Labels and selectors are a standard method to group things together
- Labels are properties attached to each item, Selectors help you filter these items
- We can create a lot of different types of objects in Kubernetes pods, services, replica sets, deployments etc. Then you will need a way to filter and view different objects by different categories
- kubectl get pod --selector tier=frontend,env=prod,bu=finance
- kubectl get all --selector env=prod

Taints and Tolerations

- Taints and Tolerations tells the node to only accept pods with certain tolerations
- We may have dedicated resources on a node for a particular use case or application
- By placing a taint on the node, it prevent any pod from being placed on the node
- To enable certain pods to be placed on this node, We must specify which pods are tolerant to this particular taint.
- kubectl taint nodes node-name app=blue:taint-effect(NoSchedule|PreferNoSchedule|NoExecute)
- Remove Taint: kubectl taint nodes controlplan app=blue:NoSchedule-

Node selector

- we can set a limitation on the pods so that they only run on particular nodes (for example larger nodes with more compute power or resources)

- Node selector is simple and easy way of doing this

- To label a node: kubectl label nodes node01 size=Large

Node Affinity

- We cannot provide advanced expressions like "or" or "not" with node selectors.
- The node affinity feature provides us with advanced capabilities to limit pod placement on specific nodes
- There are currently two types of node affinity available
  - requiredDuringSchedulingIgnoredDuringExecution
  - preferredDuringSchedulingIgnoredDuringExecution
  - For required type if a matching node is not found, pod is not scheduled
  - For preferred type if a matching node is not found, pod is scheduled on any node availabel

Resource Requirements and Limits

- you can specify the amount of CPU and memory required for a pod when creating one, and this is known as the resource request for a container
- By default, a container has no limit to the resources it can consume on a node. However, you can set a limit for the resource usage on these pods
- A container cannot use more CPU resources than its limit. However, this is not the case with memory. A container can use more memory resources than its limit

Resource Requirements and Limits

- Kubernetes does not have default resource requests or limits configured for every pod in cluster
- we can ensure that every pod created has some default set
- This is possible with limit ranges. LimitRange is applicable at the namespace level
- If you create or change a limit range, it does not affect existing pods. It'll only affect newer pods that are created
- If we had to say that all the pods together shouldn't consume more than this much of CPU or memory then we could create quotas at a namespace level

DaemonSets

- DaemonSets are like ReplicaSets, as in it helps you deploy multiple instances of pods. But it runs one copy of your pod on each node in your cluster
- Say you would like to deploy a monitoring agent or log collector on each of your nodes in the cluster,so you can monitor your cluster better
- The kube-proxy component can be deployed as a DaemonSet in the cluster

Static PODs

- Static Pods are created by the kubelet on its own without the intervention from the API server or rest of the Kubernetes cluster components are known as static Pods
- You can only create Pods this way. You cannot create replica sets or deployments or services
- You can configure the kubelet to read the Pod definition files from a directory on the server designated to store information about Pod

Static PODs

- When the kubelet creates a static Pod, if it is a part of a cluster, it also creates a mirror object in the kube-apiserver
- Kube-apiserver can view details about the Pod,but you cannot edit or delete it like the usual Pods
- Since static Pods are not dependent on the Kubernetes control plane, you can use static Pods to deploy the control plane components itself as Pods on a Node
- That's how the kube admin tool sets up a Kubernetes cluster

Logging and Monitoring

- kubectl top node command provides the CPU and memory consumption of each of the node.
- kubectl top pod command to view performance metrics of pods in Kubernetes.
- Kubernetes does not come with a full-featured built-in monitoring solution. However, there are a number of open-source solutions available today uch as Metrics Server, Prometheus, the Elastic Stack, and proprietary solutions like Datadog and Dynatrace
- kubectl logs -f pod-name [container-name]

Rolling updates and Rollbacks

- When you first create a deployment, it triggers a rollout. A new rollout creates a new deployment revision. In the future when the application is upgraded,a new rollout is triggered, and a new deployment revision is created.
- There are two types of deployment strategies, first is the recreate strategy, destroy all of these and then create newer versions of application instances
- Other one is Rolling update, the default deployment strategy. In this strategy k8s take down the older version and bring up a newer version one by one.This way, the application never goes down

Rolling updates and Rollbacks

- Rolling update strategies can also be seen when you view the deployments in detail. Run the kubectl describe deployment command to see the detailed information regarding the deployments
- Under the hood. When a new deployment is created, say, to deploy five replicas,it first creates a replica set automatically, which in turn creates the number of pods required to meet the number of replicas

Commands, Arguments, Env Variables

- In POD Definition command field overrides the ENTRYPOINT and args field overrides the CMD
- To set an environment variable, use the ENV property. ENV is array, so every item under the ENV property is name and value
- When a pod is created, inject the config map into the pod so the key value pairs are available as environment variables
- Imperative way: kubectl create configmap appp-config --from-literal=APP_COLOR=blue \ --from-literal=APP_MOD=prod

Configuring ConfigMaps

- We can take variable information out of the pod definition file and manage it centrally using configuration maps
- Config maps are used to pass configuration data in the form of key value pairs in Kubernetes
- When a pod is created, inject the config map into the pod so the key value pairs are available as environment variables

Configuring secrets in application

- Secrets are used to store sensitive information like passwords or keys. They're similar to ConfigMaps except that they're stored in an encoded format
- To encode in linux: echo -n "mysql" | base64
- Imperative ways to create secrets: kubectl create secret generic app-secret --from-literal=DB_HOS=mysql
- You can inject secrets in POD same as ConfigMaps

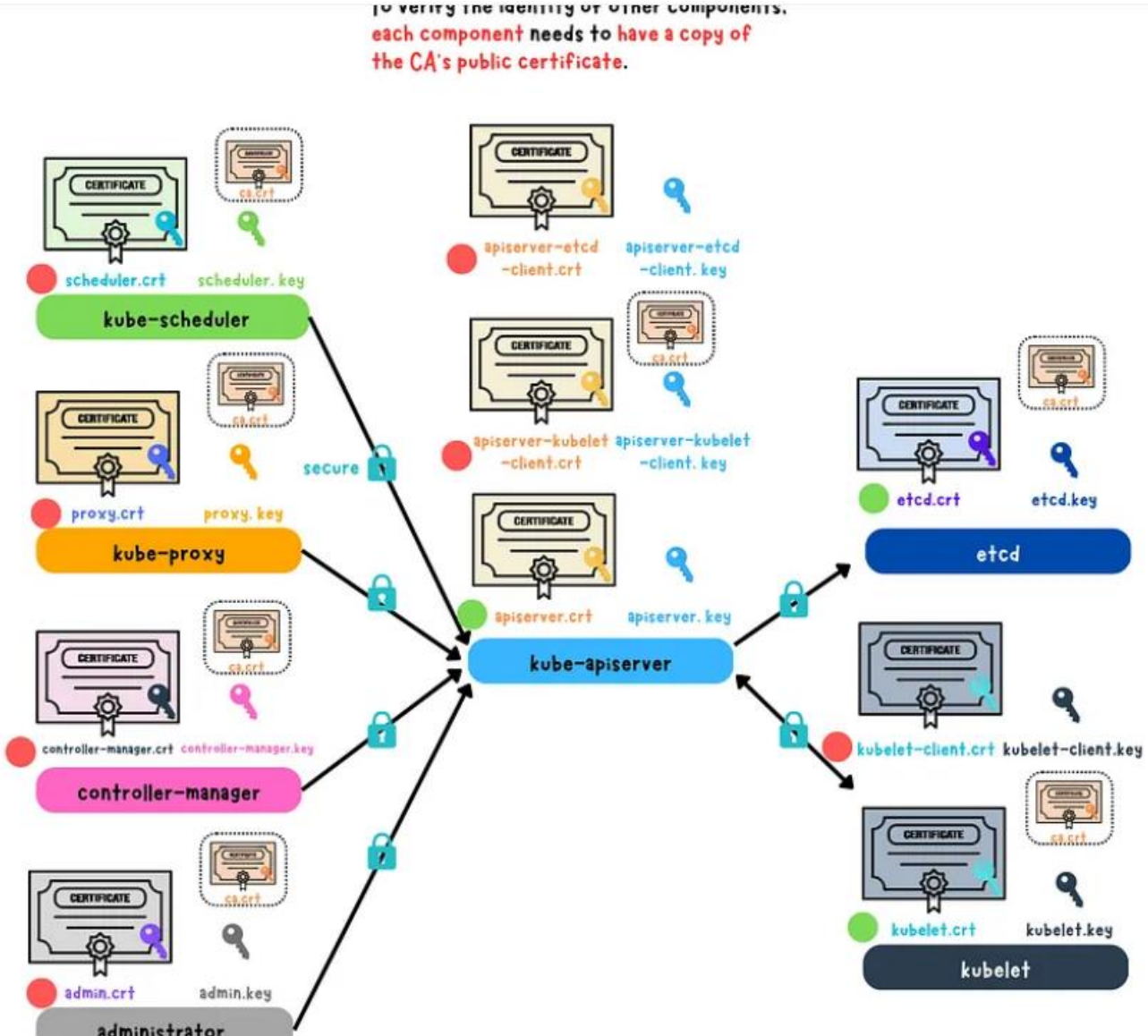
Multicontainer PODs

- You have multi container pods that share the same life cycle
- All pods in a container are created together and destroyed together. They share the same network space, which means they can refer to each other as local host, and they have access to the same storage volumes
- To create a multi-container pod, add the new container information to the pod definition file. As container section under spec is an array
- In a multi-container pod, each container is expected to run a process that stays alive as long as the POD's lifecycle

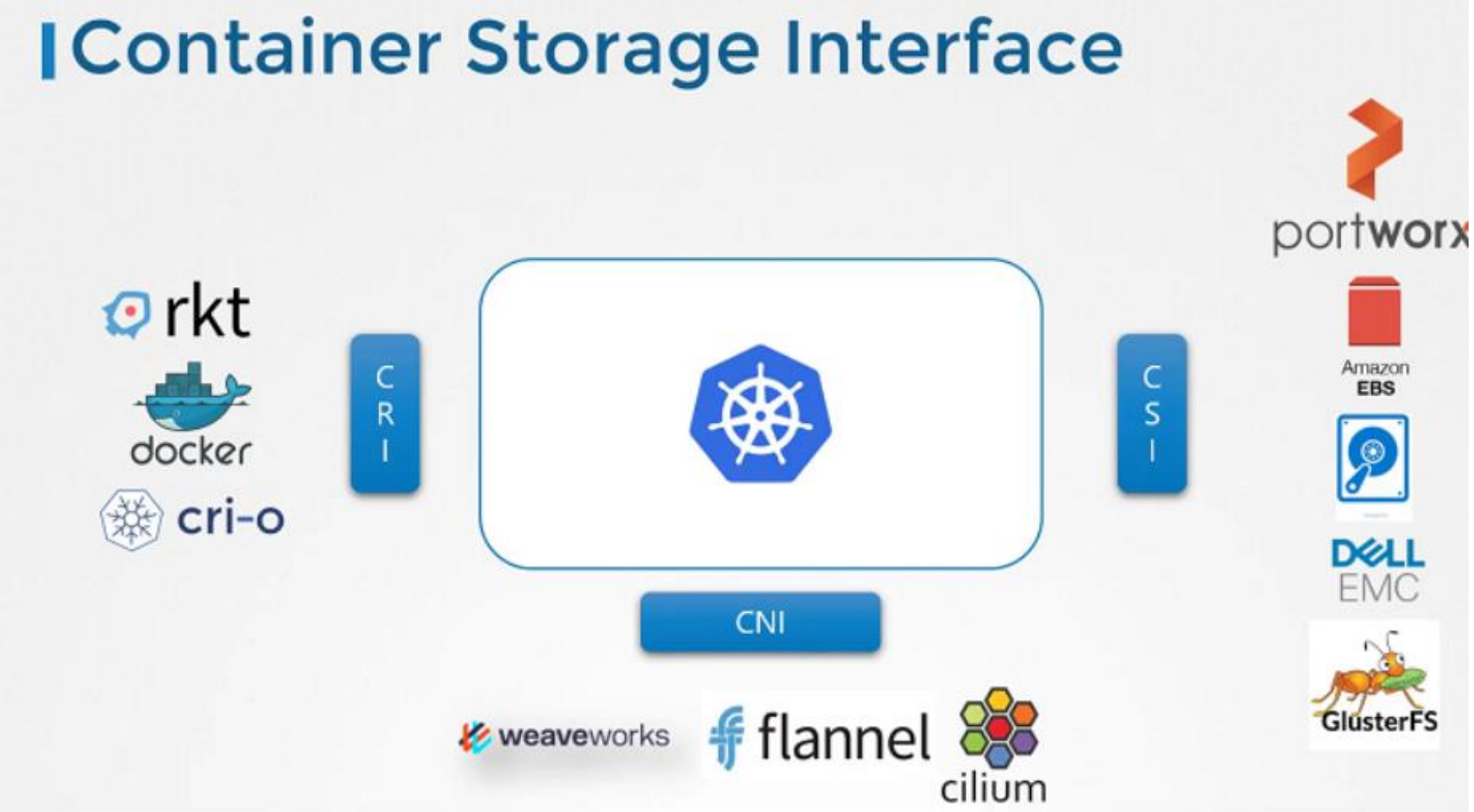
Init Containers

- At times you may want to run a process that runs to completion in a container. For example, a process that pulls a code or binary from a repository that will be used by the main web application
- An initContainer is configured in a pod like all other containers, except that it is specified inside an initContainers section
- When a POD is first created the InitContainer is run, and the process in the initContainer must run to a completion before the real container hosting the application starts

Kubernets TLS



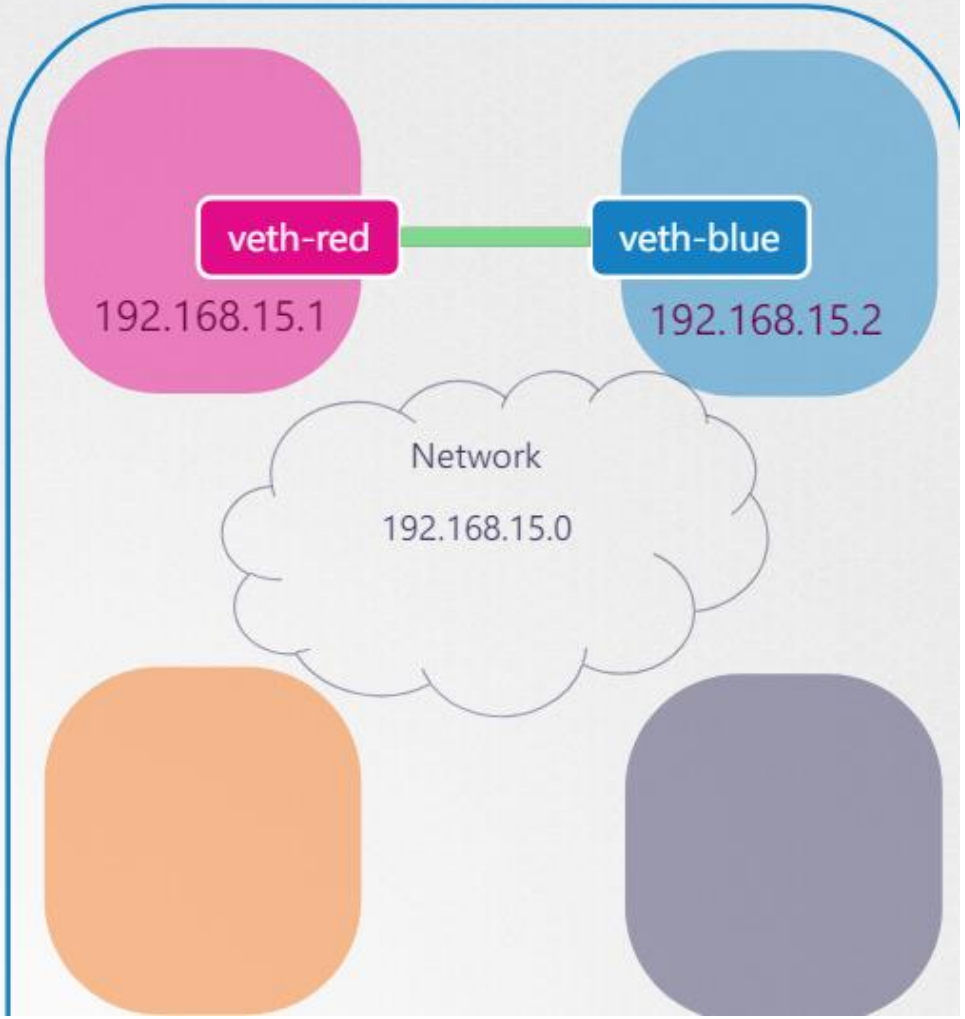
CSI


Container Storage Interface
rkt
docker
cri-o
CRI
CSI
CNI
weaveworks
flannel
cilium
portworx
Amazon EBS
DELL EMC
GlusterFS

Network Namespace

LINUX BRIDGE

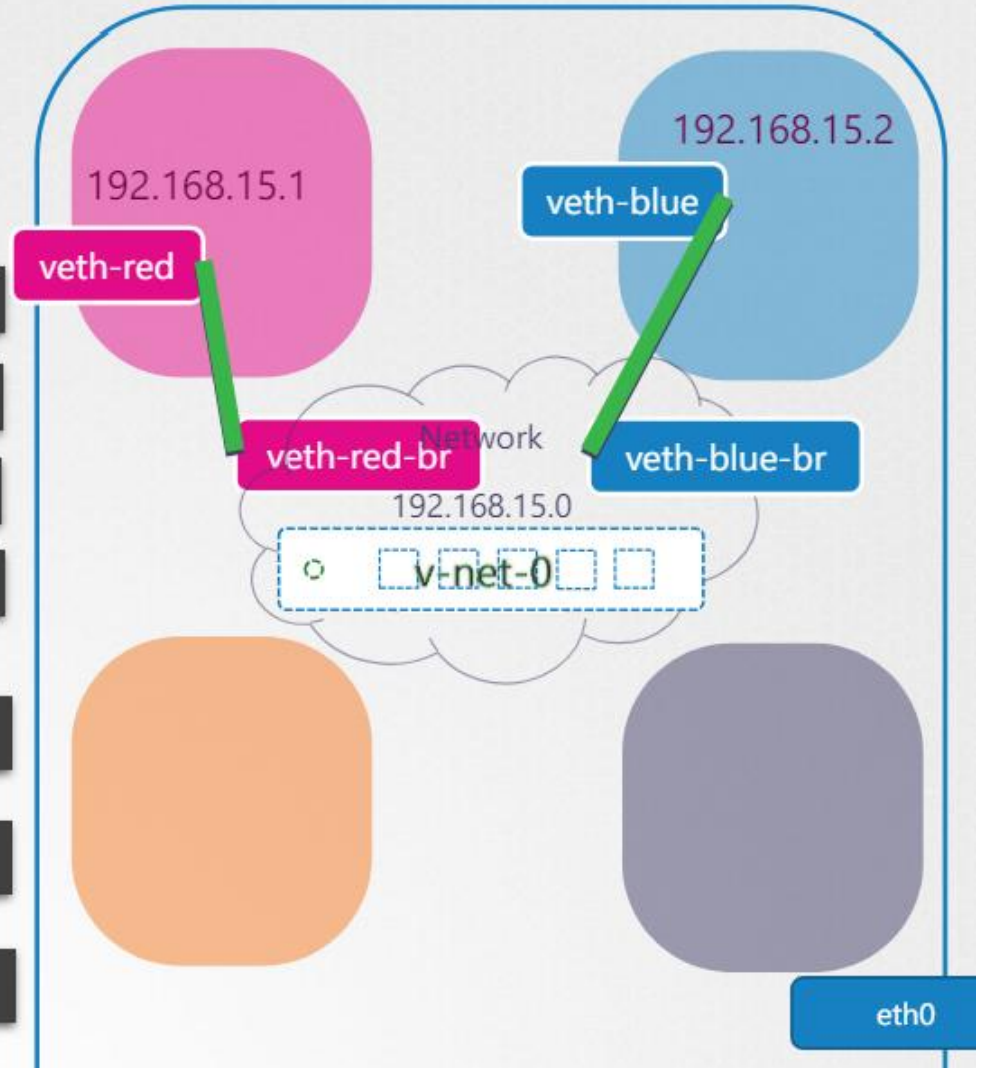
OvS Open vSwitch



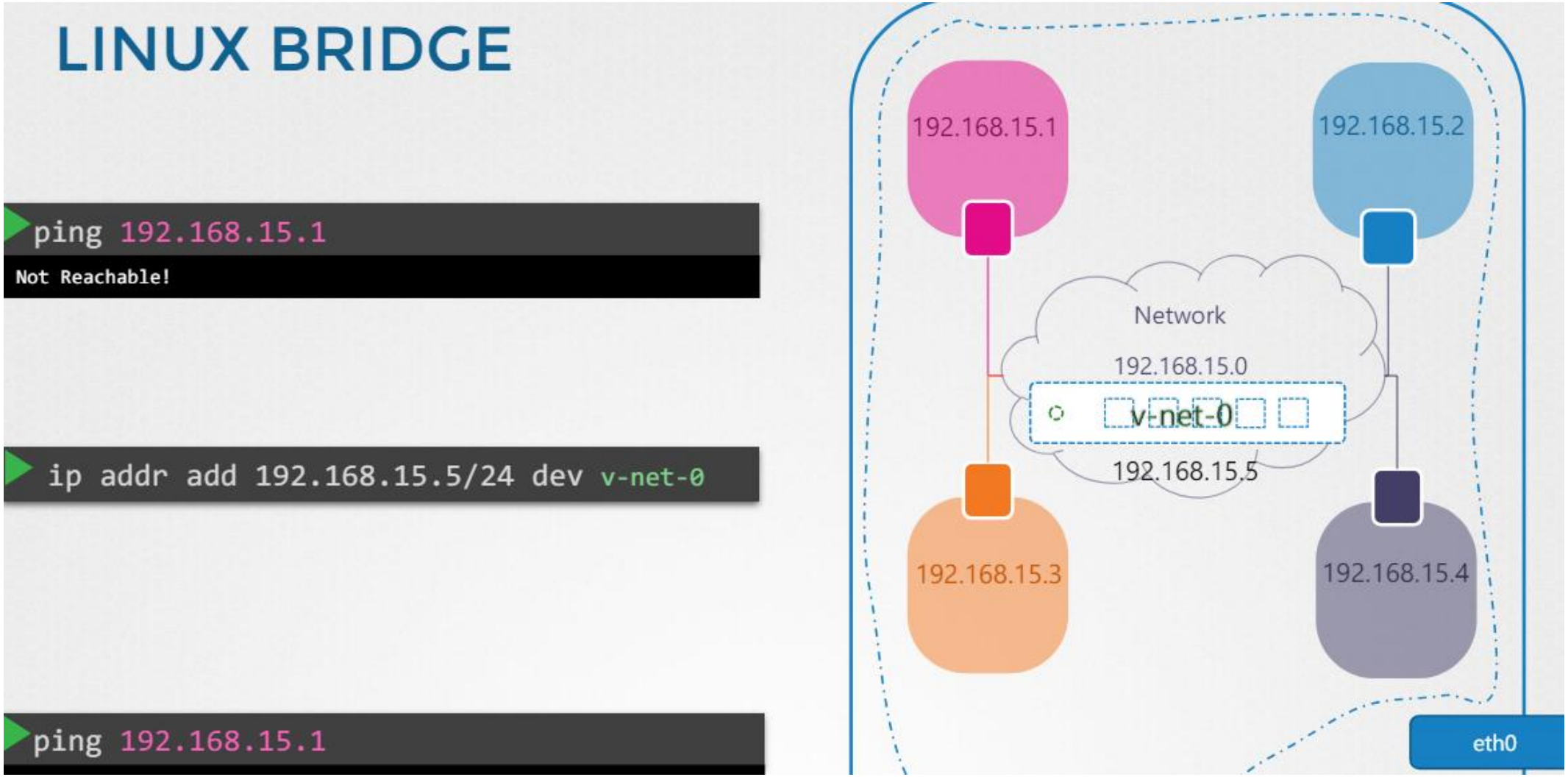
Network Namespace

LINUX BRIDGE

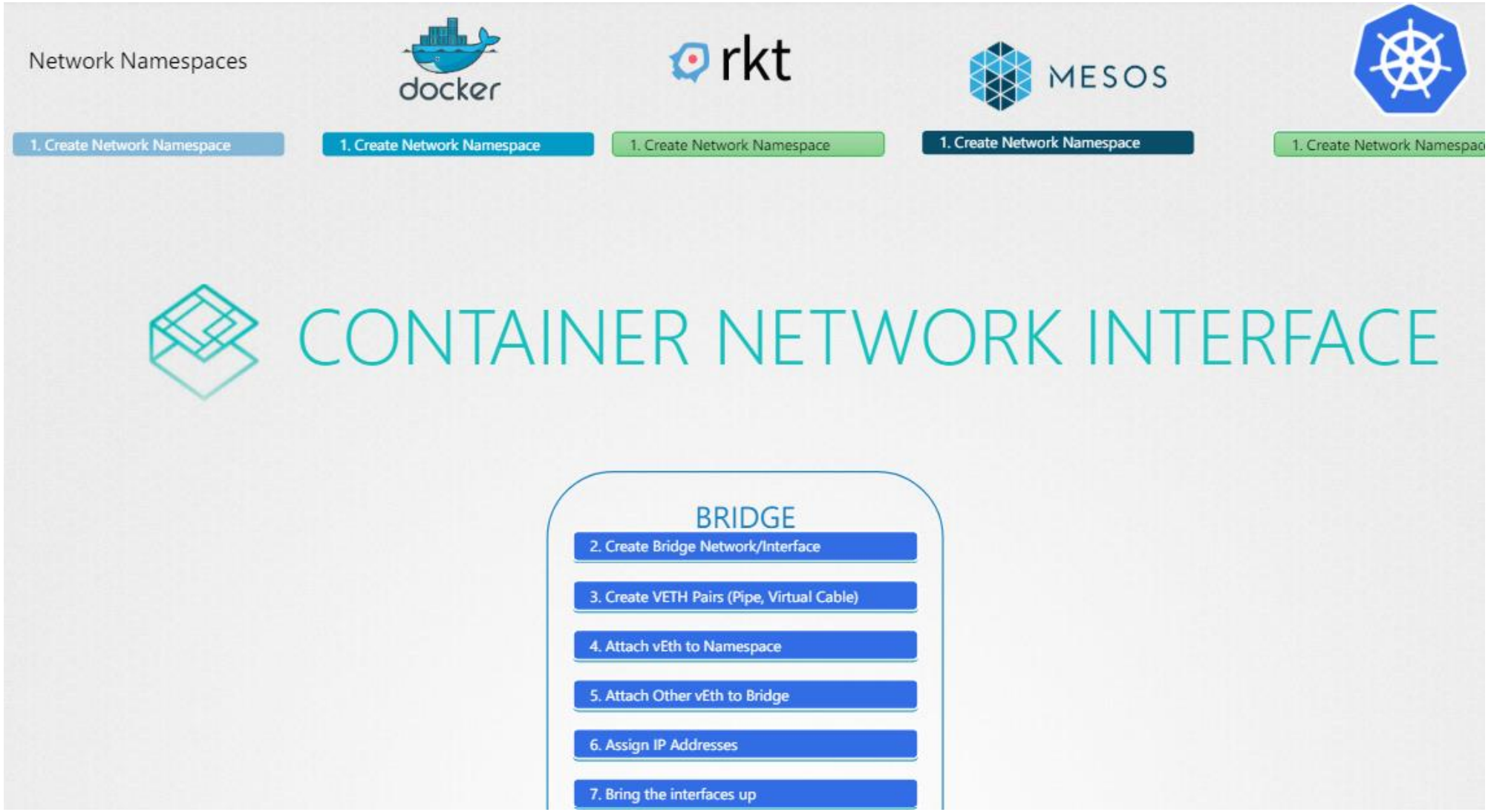
```
ip link set veth-red netns red
ip link set veth-red-br master v-net-0
ip link set veth-blue netns blue
ip link set veth-blue-br master v-net-0

ip -n red addr add 192.168.15.1 dev veth-red
ip -n blue addr add 192.168.15.2 dev veth-blue
ip -n red link set veth-red up
```



Network Namespace



CNI


Network Namespaces
docker
rkt
MESOS
1. Create Network Namespace
1. Create Network Namespace
1. Create Network Namespace
1. Create Network Namespace
1. Create Network Namespac
CONTAINER NETWORK INTERFACE
BRIDGE
2. Create Bridge Network/Interface
3. Create VETH Pairs (Pipe, Virtual Cable)
4. Attach vEth to Namespace
5. Attach Other vEth to Bridge
6. Assign IP Addresses
7. Bring the interfaces up

CNI

CONTAINER NETWORK INTERFACE

- ☑ Container Runtime must create network namespace
- ☑ Identify network the container must attach to
- ☑ Container Runtime to invoke Network Plugin (bridge) when container is ADDed.
- ☑ Container Runtime to invoke Network Plugin (bridge) when container is DELeted.
- ☑ JSON format of the Network Configuration

rkt

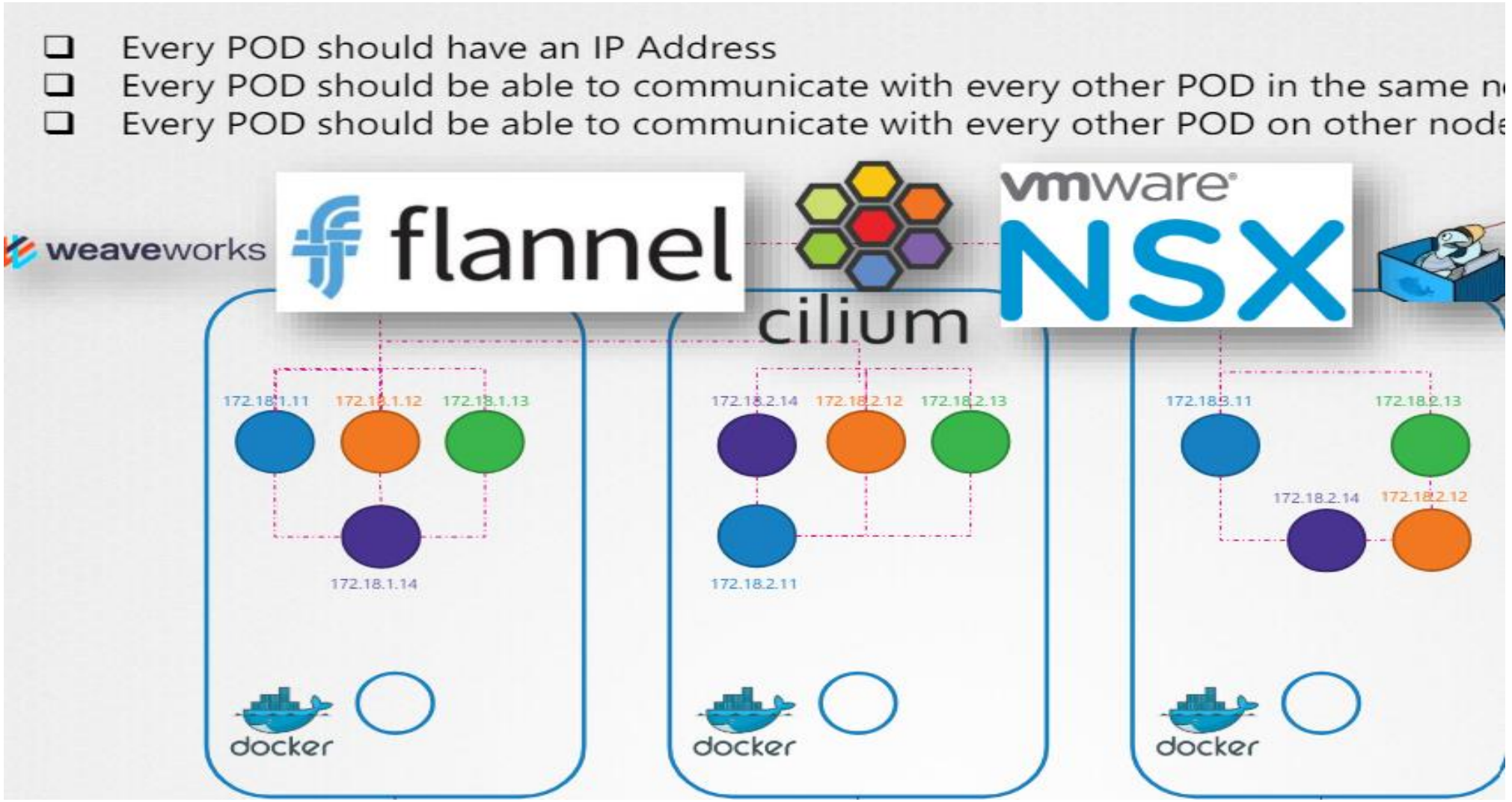
MESOS

- ☑ Must support command line arguments ADD/DEL/CHECK
- ☑ Must support parameters container id, network ns etc..
- ☑ Must manage IP Address assignment to PODs
- ☑ Must Return results in a specific format

BRIDGE

2. Create Bridge Network/Interface
3. Create VETH Pairs (Pipe, Virtual Cable)
4. Attach vEth to Namespace
5. Attach Other vEth to Bridge
6. Assign IP Addresses
7. Bring the interfaces up
8. Enable NAT – IP Masquerade

Pod Networking



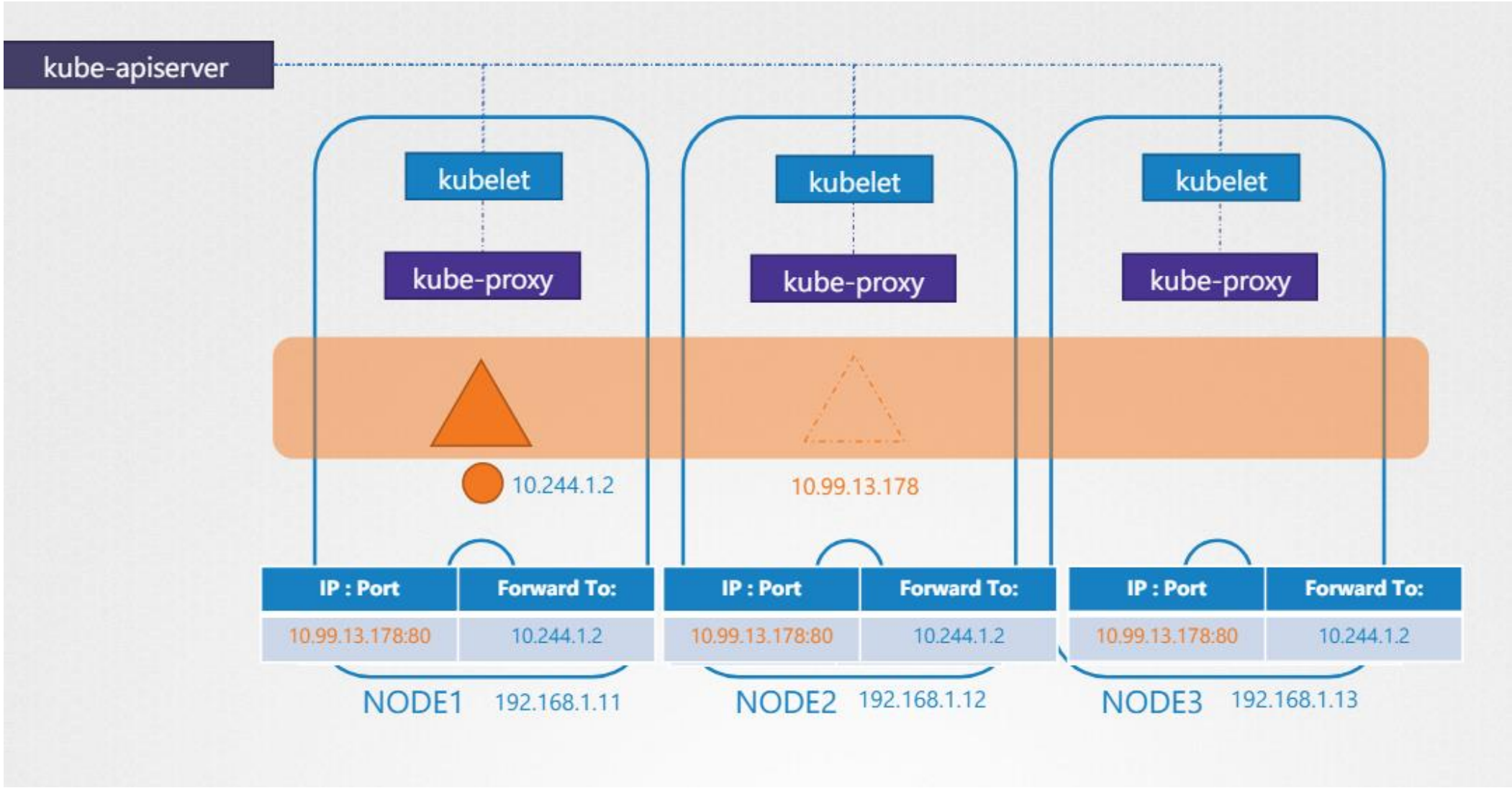
CNI Weave

Weave Peers

```
kubectl get pods -n kube-system
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE
coredns-78fcdf6894-99khw	1/1	Running	0	19m	10.44.0.2	master	<none>
coredns-78fcdf6894-p7dpj	1/1	Running	0	19m	10.44.0.1	master	<none>
etcd-master	1/1	Running	0	18m	172.17.0.11	master	<none>
kube-apiserver-master	1/1	Running	0	18m	172.17.0.11	master	<none>
kube-scheduler-master	1/1	Running	0	17m	172.17.0.11	master	<none>
weave-net-5gcmb	2/2	Running	1	19m	172.17.0.30	node02	<none>
weave-net-fr9n9	2/2	Running	1	19m	172.17.0.11	master	<none>
weave-net-mc6s2	2/2	Running	1	19m	172.17.0.23	node01	<none>

Service Networking



Cluster Maintenance: OS Upgrade

- you might have to take down nodes as part of your cluster, say for maintenance purposes, like upgrading a software
- If the node was down for more than five minutes, then the pods are terminated from that node
- The time it waits for a pod to come back online is known as the pod-eviction-timeout and is set on the controller manager with a default value of five minutes
- command for drain is kubectl drain node-1
- kubectl uncordon node-1
- you can manually mark a node cordoned using kubectl cordon node-1

Cluster Upgrade process

- Since the kube-apiserver is the primary component in the control plane, and that is the component that all other components talk to,none of the other components should ever be at a version higher than the kube-apiserver
- The controller-manager and scheduler can be at one version lower
- kubelet and kube-proxy components can be at two versions lower
- kubectl utility could 1 version higher, same or 1 version lower than api-server
- Kubernetes supports only up to the recent three minor versions, if current is 1.29 then 1.28,1.27 are supported

Cluster Upgrade process

- tools like kubeadm, then the tool can help you plan and upgrade the cluster
- First, you upgrade your master nodes and then upgrade the worker nodes
- kubeadm has an upgrade command, it will give you a lot of good information, you must upgrade the kubeadm tool itself before you can upgrade the cluster

CKA Exam References

- Certified Kubernetes Administrator:
  - https://www.cncf.io/certification/ckad/
- Exam Curriculum (Topics):
  - https://github.com/cncf/curriculum
- Exam Tips:
  - http://training.linuxfoundation.org/go//Important-Tips-CKA-CKAD
- Candidate Handbook:
  - https://www.cncf.io/certification/candidate-handbook