

Certified Kubernetes Administrator (CKA) Master Cheat Sheet

Course Introduction

This course focuses on Administration part of the kubernetes.

Course Structure

- Lectures
- Demos
- Quizzes
- Practice Questions
- Q & A Section

Pre-Requisites

- Docker
- Basics of kubernetes
 - PODS, Deployments, Services ...
- YAML
- Setting up a basic lab environment with VirtualBox

Course Objectives

The objectives of this course are aligned to match the **Certified Kubernetes Administration Exam (CKA)**

- Core Concepts
 - Cluster Architecture
 - API Primitives
 - Services & Other Network Primitives
- Scheduling
 - Labels & Selectors
 - Daemon Sets
 - Resource Limits
 - Multiple Schedulers
 - Manual Scheduling
 - Scheduler Events

- Configure Kubernetes Scheduler
- Logging & Monitoring
 - Monitor Cluster Components
 - Monitor Cluster Components Logs
 - Monitor Applications
 - Application Logs
- Application Lifecycle Management
 - Rolling Updates and Rollbacks in Deployments
 - Configuring Applications
 - Scale Applications
 - Self-Healing Applications
- Cluster Maintenance
 - Cluster Upgrade Process
 - Operating System Upgrades
 - Backup and Restore Methodologies
- Security
 - Authentication & Authorization
 - Kubernetes Security
 - Network Policies
 - TLS Certificates for Cluster Components
 - Image Security
 - Network Policies
 - Security Contexts
 - Secure Persistent Key Value Store
- Storage
 - Persistent Volumes
 - Access Modes for Volumes
 - Persistent Volume Claims
 - Kubernetes Storage Object
 - Configure Applications with Persistent Storage
- Networking
 - Pre-Requisites - Network, Switching, Routing, Tools
 - Pre-Requisites - Network Namespaces
 - Pre-Requisites - Networking in Docker
 - Networking Configuration on Cluster Nodes
 - Service Networking
 - POD Networking Concepts
 - Network Loadbalancer
 - Ingress
 - Cluster DNS

- CNI
- Installation, Configuration & Validation
 - Design a Kubernetes Cluster
 - Install Kubnernetes Master and Nodes
 - Secure Cluster Communication
 - HA Kubernetes Cluster
 - Kubenetes Release Binaries
 - Provision Infrastructure
 - Choose a Network Solution
 - Kubernetes Infrastructure Config
 - Run & Analyze end-to-end test
 - Node end-to-end tests
- Troubleshooting
 - Application Failure
 - Control Plane Failure
 - Worker Node Failure
 - Networking

Practice Tests

CKA exam is a practical hands-on exam it is very important to practice what you learn. Which is why we build a custom solution that will give you access to a **Real Kubernetes Environment** right in your browser along with **Quiz Portal**

Core Concepts Section Introduction

In this section, we will take a look at the below

- Cluster Architecture
- API Primitives
- Services & Other Network Primitives

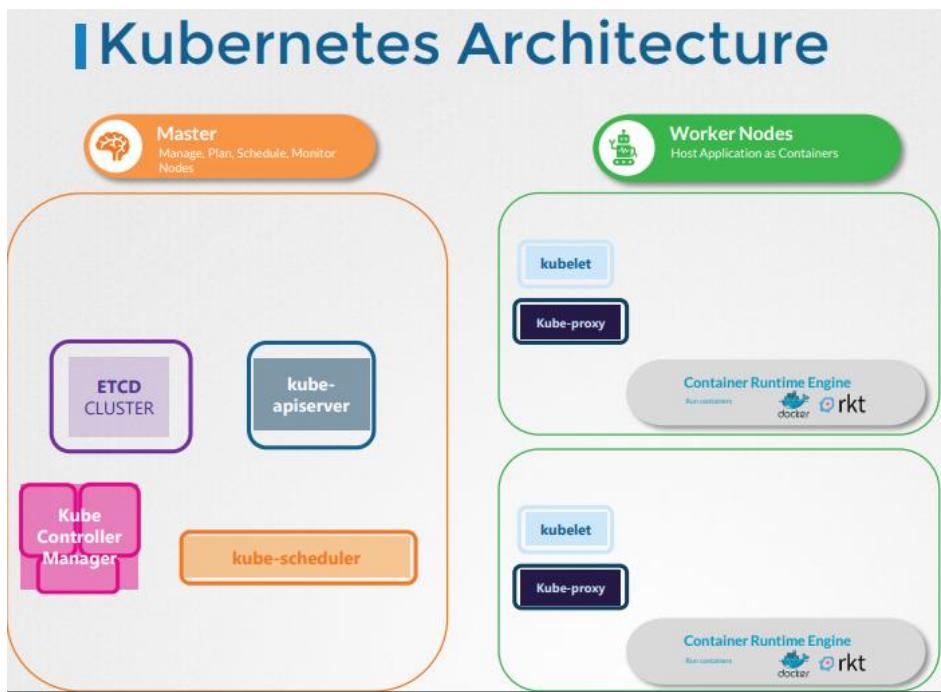
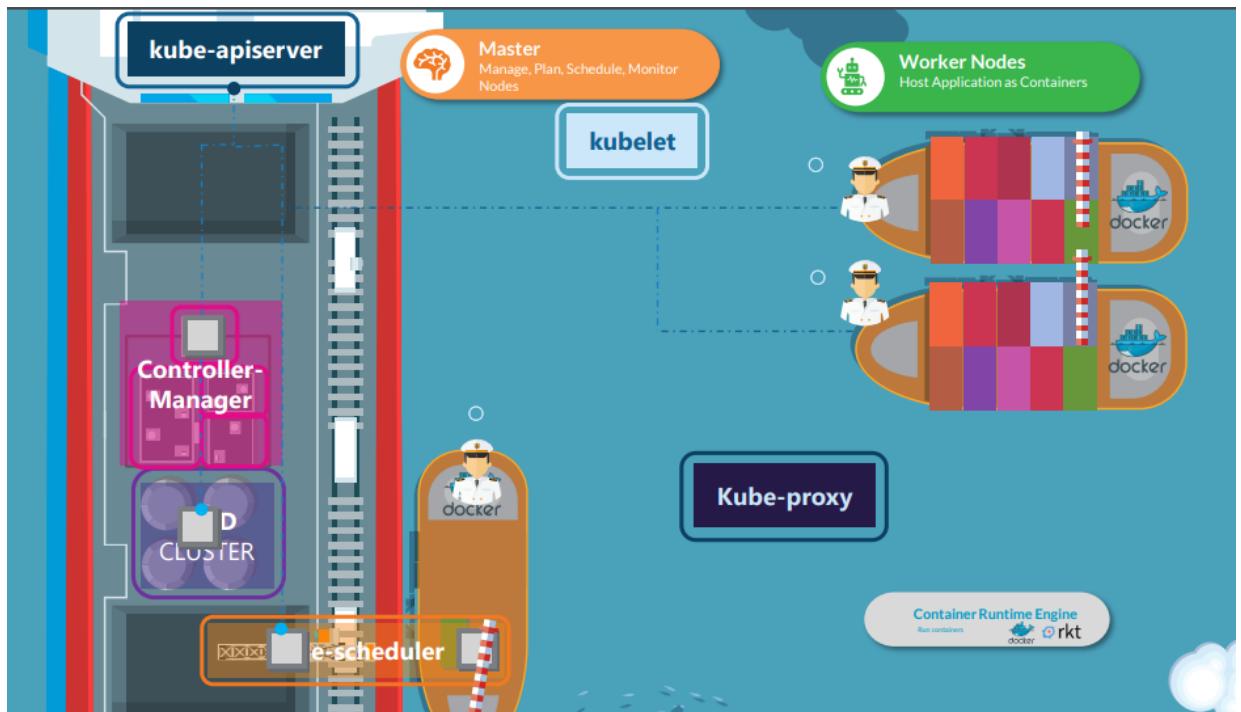
k8s reference docs:

- <https://kubernetes.io/docs/concepts/overview/kubernetes-api/>
- <https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/>
- <https://kubernetes.io/docs/concepts/architecture/>
- <https://kubernetes.io/docs/concepts/overview/components/>
- <https://kubernetes.io/docs/concepts/services-networking/>

Cluster Architecture

In this section , we will take a look at the kubernetes Architecture at high level.

- 10,000 Feet Look at the Kubernetes Architecture



K8s Reference Docs:

- <https://kubernetes.io/docs/concepts/architecture/>

ETCD for Beginners

In this section, we will take a quick look at introduction to ETCD for beginners.

- What is ETCD?
- What is a Key-Value Store?
- How to get started quickly with ETCD?
- How to operate ETCD?

What is a ETCD?

- ETCD is a distributed reliable key-value store that is simple, secure & Fast.

What is a Key-Value Store

- Traditionally, databases have been in tabular format, you must have heard about SQL or Relational databases. They store data in rows and columns

Tabular/Relational Databases		
Name	Age	Location
John Doe	45	New York
Dave Smith	34	New York
Aryan Kumar	10	New York
Lauren Rob	13	Bangalore
Lily Oliver	15	Bangalore

- A Key-Value Store stores information in a Key and Value format.

The diagram illustrates a key-value store with the following components:

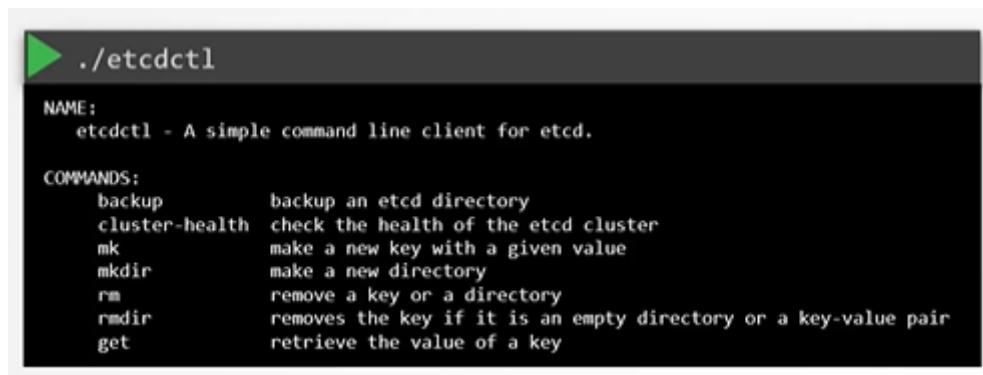
- Table:** A grid showing data pairs. The columns are labeled "Key" and "Value". The rows contain:

Key	Value
Name	John Doe
Age	45
Location	New York
Salary	5000
- Operations:** A sequence of actions:
 - Put Name "John Doe"
 - Get Name
 - "John Doe"

Install ETCD

- Its easy to install and get started with **ETCD**.
 - Download the relevant binary for your operating system from github releases page (<https://github.com/etcd-io/etcd/releases>)
 - For Example: To download ETCD V3.3.11, run the below curl command
 - `$ https://github.com/etcd-io/etcd/releases/download/v3.3.11/etcd-v3.3.11-linux-amd64.tar.gz`
 - Extract it.
 - `$ tar xvzf etcd-v3.3.11-linux-amd64.tar.gz`
 - Run the ETCD Service
 - `$./etcd`
 - When you start **ETCD** it will by default listens on port **2379**
 - The default client that comes with **ETCD** is the **etcdctl** client. You can use it to store and retrieve key-value pairs.
 - Syntax: To Store a Key-Value pair
`$./etcdctl set key1 value1`
 - Syntax: To retrieve the stored data

- \$./etcdctl get key1
- Syntax: To view more commands. Run etcdctl without any arguments
- \$./etcdctl



```

./etcdctl

NAME:
  etcdctl - A simple command line client for etcd.

COMMANDS:
  backup      backup an etcd directory
  cluster-health check the health of the etcd cluster
  mk          make a new key with a given value
  mkdir       make a new directory
  rm          remove a key or a directory
  rmdir      removes the key if it is an empty directory or a key-value pair
  get         retrieve the value of a key

```

- K8s Reference Docs:
 - <https://kubernetes.io/docs/concepts/overview/components/>
 - <https://etcd.io/docs/>
 - <https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/>

ETCD in Kubernetes

In this section, we will take a look at ETCD role in kubernetes

ETCD Datastore

- The ETCD Datastore stores information regarding the cluster such as **Nodes**, **PODS**, **Configs**, **Secrets**, **Accounts**, **Roles**, **Bindings** and **Others**.
- Every information you see when you run the **kubectl get** command is from the **ETCD Server**.

Setup - Manual

- If you setup your cluster from scratch then you deploy **ETCD** by downloading ETCD Binaries yourself
- Installing Binaries and Configuring ETCD as a service in your master node yourself.
- `$ wget -q --https-only "https://github.com/etcd-io/etcd/releases/download/v3.3.11/etcd-v3.3.11-linux-amd64.tar.gz"`

Setup - Manual

```
▶ wget -q --https-only \
  "https://github.com/coreos/etcd/releases/download/v3.3.9/etcd-v3.3.9-linux-amd64.tar.gz"

[etcd.service]
ExecStart=/usr/local/bin/etcd \\
--name ${ETCD_NAME} \\
--cert-file=/etc/etcd/kubernetes.pem \\
--key-file=/etc/etcd/kubernetes-key.pem \\
--peer-cert-file=/etc/etcd/kubernetes.pem \\
--peer-key-file=/etc/etcd/kubernetes-key.pem \\
--trusted-ca-file=/etc/etcd/ca.pem \\
--peer-trusted-ca-file=/etc/etcd/ca.pem \\
--peer-client-cert-auth \\
--client-cert-auth \\
--initial-advertise-peer-urls https://${INTERNAL_IP}:2380 \\
--listen-peer-urls https://${INTERNAL_IP}:2380 \\
--listen-client-urls https://${INTERNAL_IP}:2379,https://127.0.0.1:2379 \\
--advertise-client-urls https://${INTERNAL_IP}:2379 \\
--initial-cluster-token etcd-cluster-0 \\
--initial-cluster controller-0=https://${CONTROLLER0_IP}:2380,controller-1=https://${CONTROLLER1_IP}:2380 \\
--initial-cluster-state new \\
--data-dir=/var/lib/etcd
```

Setup - Kubeadm

- If you setup your cluster using `kubeadm` then kubeadm will deploy etcd server for you as a pod in `kube-system` namespace.
- `$ kubectl get pods -n kube-system`

I Setup - kubeadm

kubectl get pods -n kube-system						
NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	
kube-system	coredns-78fcdf6894-prwvl	1/1	Running	0	1h	
kube-system	coredns-78fcdf6894-vqd9w	1/1	Running	0	1h	
kube-system	etcd-master	1/1	Running	0	1h	
kube-system	kube-apiserver-master	1/1	Running	0	1h	
kube-system	kube-controller-manager-master	1/1	Running	0	1h	
kube-system	kube-proxy-f6k26	1/1	Running	0	1h	
kube-system	kube-proxy-hnzsw	1/1	Running	0	1h	
kube-system	kube-scheduler-master	1/1	Running	0	1h	
kube-system	weave-net-924k8	2/2	Running	1	1h	
kube-system	weave-net-hzfcz	2/2	Running	1	1h	

Explore ETCD

- To list all keys stored by kubernetes, run the below command
- `$ kubectl exec etcd-master -n kube-system etcdctl get / --prefix -key`
- Kubernetes Stores data in a specific directory structure, the root directory is the `registry` and under that you have varies kubernetes constructs such as `minions`, `nodes`, `pods`, `replicasets`, `deployments`, `roles`, `secrets` and `Others`.

Explore ETCD

```
▶ kubectl exec etcd-master -n kube-system etcdctl get / --prefix -keys-only
/registry/apiregistration.k8s.io/apiservices/v1.
/registry/apiregistration.k8s.io/apiservices/v1.apps
/registry/apiregistration.k8s.io/apiservices/v1.authentication.k8s.io
/registry/apiregistration.k8s.io/apiservices/v1.authorization.k8s.io
/registry/apiregistration.k8s.io/apiservices/v1.autoscaling
/registry/apiregistration.k8s.io/apiservices/v1.batch
/registry/apiregistration.k8s.io/apiservices/v1.networking.k8s.io
/registry/apiregistration.k8s.io/apiservices/v1.rbac.authorization.k8s.io
/registry/apiregistration.k8s.io/apiservices/v1.storage.k8s.io
/registry/apiregistration.k8s.io/apiservices/v1beta1.admissionregistration.k8s.io
```

Run inside the etcd-master POD

- Registry
- minions
- pods
- replicasets
- deployments
- roles
- secrets

ETCD in HA Environment

- In a high availability environment, you will have multiple master nodes in your cluster that will have multiple ETCD Instances spread across the master nodes.
- Make sure etcd instances know each other by setting the right parameter in the `etcd.service` configuration. The `--initial-cluster` option where you need to specify the different instances of the etcd service.

IETCD in HA Environment



```
etcd.service
ExecStart=/usr/local/bin/etcd \
--name ${ETCD_NAME} \
--cert-file=/etc/etcd/kubernetes.pem \
--key-file=/etc/etcd/kubernetes-key.pem \
--peer-cert-file=/etc/etcd/kubernetes.pem \
--peer-key-file=/etc/etcd/kubernetes-key.pem \
--trusted-ca-file=/etc/etcd/ca.pem \
--peer-trusted-ca-file=/etc/etcd/ca.pem \
--peer-client-cert-auth \
--client-cert-auth \
--initial-advertise-peer-urls https://${INTERNAL_IP}:2380 \
--listen-peer-urls https://${INTERNAL_IP}:2380 \
--listen-client-urls https://${INTERNAL_IP}:2379,https://127.0.0.1:2379 \
--initial-cluster-token etcd-cluster-0 \
--initial-cluster controller-0=https://${CONTROLLER0_IP}:2380,controller-1=https://${CONTROLLER1_IP}:2380 \
--initial-cluster-state new \
--data-dir=/var/lib/etcd
```

K8s Reference Docs:

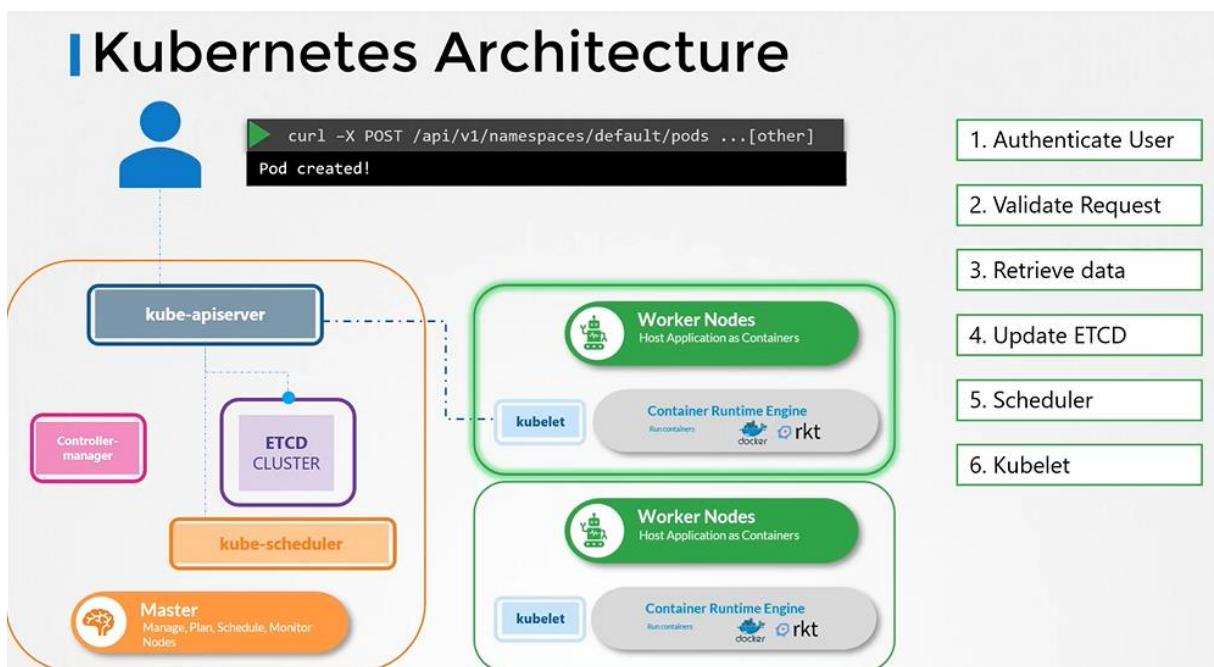
- <https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/>
- <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/setup-ha-etcd-with-kubeadm/>
- <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/#stacked-control-plane-and-etcd-nodes>
- <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/#external-etcd-nodes>

Kube API Server

In this section, we will talk about kube-apiserver in kubernetes

Kube-apiserver is the primary component in kubernetes.

- Kube-apiserver is responsible for **authenticating, validating requests, retrieving and updating** data in ETCD key-value store. In fact kube-apiserver is the only component that interacts directly to the etcd datastore. The other components such as kube-scheduler, kube-controller-manager and kubelet uses the API-Server to update in the cluster in their respective areas.



Installing kube-apiserver

- If you are bootstrapping kube-apiserver using **kubeadm** tool, then you don't need to know this, but if you are setting up using the hardway then kube-apiserver is available as a binary in the kubernetes release page.

- For example: You can download the kube-apiserver v1.13.0 binary here [kube-apiserver](#)
- \$ wget https://storage.googleapis.com/kubernetes-release/release/v1.13.0/bin/linux/amd64/kube-apiserver

Installing kube-api server

```
wget https://storage.googleapis.com/kubernetes-release/release/v1.13.0/bin/linux/amd64/kube-apiserver
[...]
kube-apiserver.service
[...]
ExecStart=/usr/local/bin/kube-apiserver \
--advertise-address=${INTERNAL_IP} \
--allow-privileged=true \
--apiserver-count=3 \
--authorization-mode=Node,RBAC \
--bind-address=0.0.0.0 \
--client-ca-file=/var/lib/kubernetes/ca.pem \
--enable-admission-
plugins=Initializers,NamespaceLifecycle,NodeRestriction,LimitRanger,ServiceAccount,DefaultStorageClass,ResourceQuota \
--enable-swagger-ui=true \
--etcd-cafile=/var/lib/kubernetes/ca.pem \
--etcd-certfile=/var/lib/kubernetes/kubernetes.pem \
--etcd-keyfile=/var/lib/kubernetes/kubernetes-key.pem \
--etcd-servers=https://127.0.0.1:2379 \
--event-ttl=1h \
--experimental-encryption-provider-config=/var/lib/kubernetes/encryption-config.yaml \
--kubelet-certificate-authority=/var/lib/kubernetes/ca.pem \
--kubelet-client-certificate=/var/lib/kubernetes/kubernetes.pem \
--kubelet-client-key=/var/lib/kubernetes/kubernetes-key.pem \
--kubelet-https=true \
--runtime-config=api/all \
--service-account-key-file=/var/lib/kubernetes/service-account.pem \
--service-cluster-ip-range=10.32.0.0/24 \\\n
```

View kube-apiserver - Kubeadm

- kubeadm deploys the kube-apiserver as a pod in kube-system namespace on the master node.
- \$ kubectl get pods -n kube-system

View api-server - kubeadm

kubectl get pods -n kube-system						
NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	
kube-system	coredns-78fcdf6894-hwrq9	1/1	Running	0	16m	
kube-system	coredns-78fcdf6894-rzhjr	1/1	Running	0	16m	
kube-system	etcd-master	1/1	Running	0	15m	
kube-system	kube-apiserver-master	1/1	Running	0	15m	
kube-system	kube-controller-manager-master	1/1	Running	0	15m	
kube-system	kube-proxy-lzt6f	1/1	Running	0	16m	
kube-system	kube-proxy-zm5qd	1/1	Running	0	16m	
kube-system	kube-scheduler-master	1/1	Running	0	15m	
kube-system	weave-net-29z42	2/2	Running	1	16m	
kube-system	weave-net-snndl	2/2	Running	1	16m	-

View kube-apiserver options - Kubeadm

- You can see the options with in the pod definition file located at `/etc/kubernetes/manifests/kube-apiserver.yaml`
- `$ cat /etc/kubernetes/manifests/kube-apiserver.yaml`

IView api-server options - kubeadm

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

spec:
  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
```

View kube-apiserver options - Manual

- In a Non-kubeadm setup, you can inspect the options by viewing the `kube-apiserver.service`
- `$ cat /etc/systemd/system/kube-apiserver.service`

IView api-server options

```
▶ cat /etc/systemd/system/kube-apiserver.service
[Service]
ExecStart=/usr/local/bin/kube-apiserver \
--advertise-address=${INTERNAL_IP} \
--allow-privileged=true \
--apiserver-count=3 \
--audit-log-maxage=30 \
--audit-log-maxbackup=3 \
--audit-log-maxsize=100 \
--audit-log-path=/var/log/audit.log \
--authorization-mode=Node,RBAC \
--bind-address=0.0.0.0 \
--client-ca-file=/var/lib/kubernetes/ca.pem \
--enable-admission-
plugins=Initializers,NamespaceLifecycle,NodeRestriction,LimitRanger,ServiceAccount,DefaultStorageClass,ResourceQuota \
--enable-swagger-ui=true \
--etcd-cafile=/var/lib/kubernetes/ca.pem \
--etcd-certfile=/var/lib/kubernetes/kubernetes.pem \
--etcd-keyfile=/var/lib/kubernetes/kubernetes-key.pem \
--etcd-
servers=https://10.240.0.10:2379,https://10.240.0.11:2379,https://10.240.0.12:2379 \
--event-ttl=1h \
--experimental-encryption-provider-config=/var/lib/kubernetes/encryption-config.yaml \
 \
--kubelet-certificate-authority=/var/lib/kubernetes/ca.pem \
--kubelet-client-certificate=/var/lib/kubernetes/kubernetes.pem \\\
```

- You can also see the running process and affective options by listing the process on master node and searching for kube-apiserver.
- `$ ps -aux | grep kube-apiserver`

```
▶ ps -aux | grep kube-apiserver
root      2348  3.3 15.4 399040 315604 ?      Ssl 15:46  1:22 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 --secure-
port=6443 --service-account-key-file=/etc/kubernetes/pki/sa.pub --service-cluster-ip-range=10.96.0.0/12 --tls-
cert-file=/etc/kubernetes/pki/apiserver.crt --tls-private-key-file=/etc/kubernetes/pki/apiserver.key
```

K8s Reference Docs:

- <https://kubernetes.io/docs/reference/command-line-tools-reference/kube-apiserver/>
- <https://kubernetes.io/docs/concepts/overview/components/>
- <https://kubernetes.io/docs/concepts/overview/kubernetes-api/>
- <https://kubernetes.io/docs/tasks/access-application-cluster/access-cluster/>
- <https://kubernetes.io/docs/tasks/administer-cluster/access-cluster-api/>

Kube Controller Manager

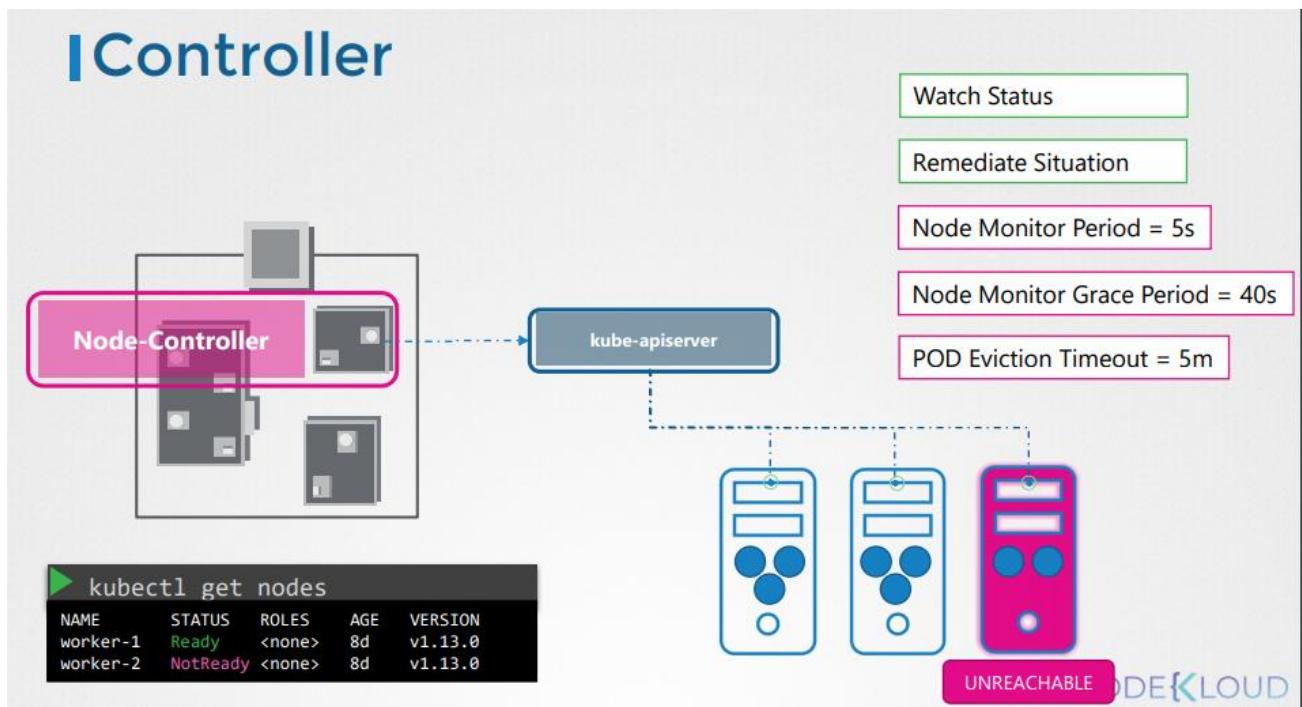
In this section, we will take a look at kube-controller-manager.

Kube Controller Manager manages various controllers in kubernetes.

- In kubernetes terms, a controller is a process that continuously monitors the state of the components within the system and works towards bringing the whole system to the desired functioning state.

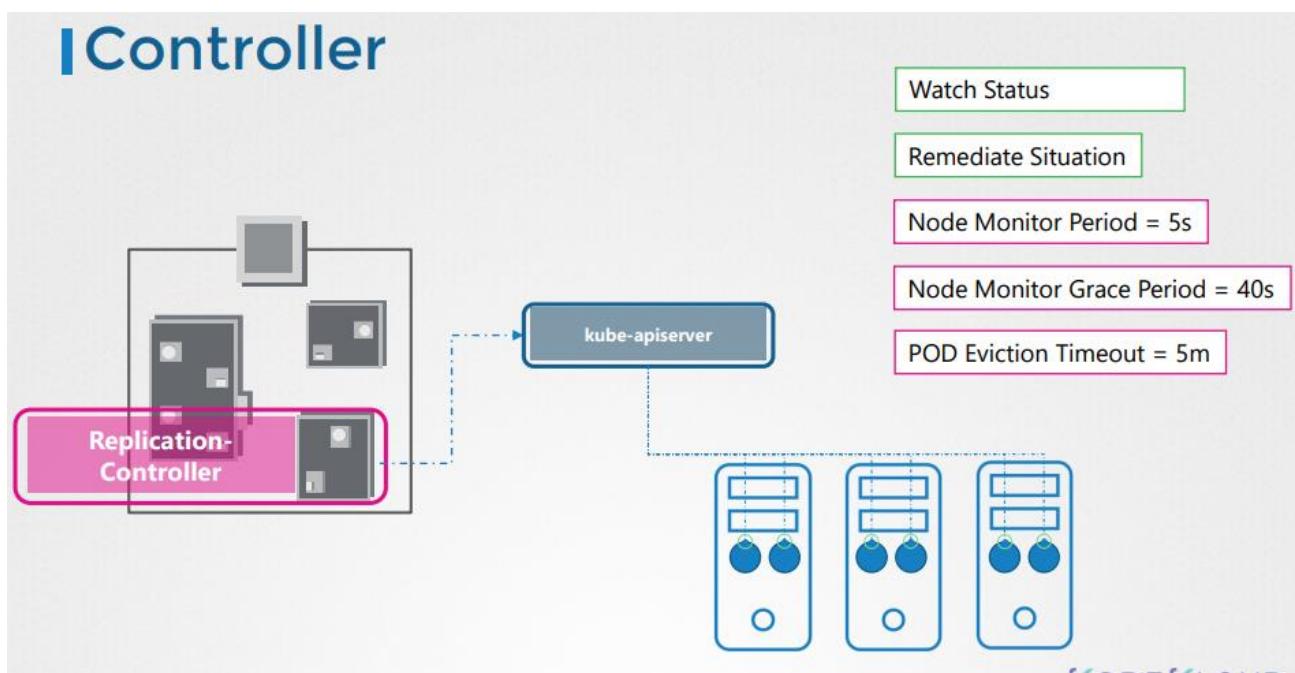
Node Controller

- Responsible for monitoring the state of the Nodes and taking necessary actions to keep the application running.



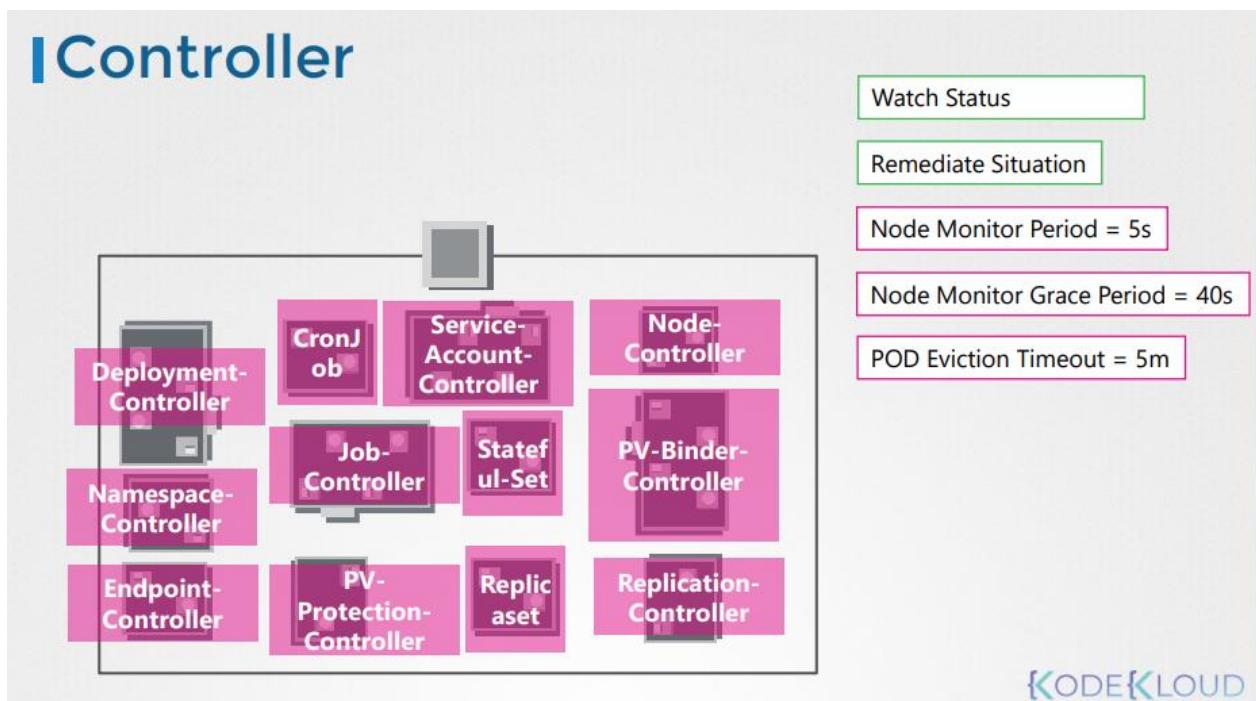
Replication Controller

- It is responsible for monitoring the status of replicaset and ensuring that the desired number of pods are available at all time within the set.



Other Controllers

- There are many more such controllers available within kubernetes



Installing Kube-Controller-Manager

- When you install `kube-controller-manager` the different controllers will get installed as well.

- Download the kube-controller-manager binary from the kubernetes release page. For example: You can download kube-controller-manager v1.13.0 here [kube-controller-manager](#)
- \$ wget https://storage.googleapis.com/kubernetes-release/release/v1.13.0/bin/linux/amd64/kube-controller-manager
- By default all controllers are enabled, but you can choose to enable specific one from `kube-controller-manager.service`
- \$ cat /etc/systemd/system/kube-controller-manager.service

I Installing kube-controller-manager

```
▶ wget https://storage.googleapis.com/kubernetes-release/release/v1.13.0/bin/linux/amd64/kube-controller-manager
kube-controller-manager.service
ExecStart=/usr/local/bin/kube-controller-manager \
--address=0.0.0.0 \
--cluster-cidr=10.200.0.0/16 \
--cluster-name=kubernetes \
--cluster-signing-cert-file=/var/lib/kubernetes/ca.pem \
--cluster-signing-key-file=/var/lib/kubernetes/ca-key.pem \
--kubeconfig=/var/lib/kubernetes/kube-controller-manager.kubeconfig \
--leader-elect=true \
--root-ca-file=/var/lib/kubernetes/ca.pem \
--service-account-private-key-file=/var/lib/kubernetes/service-account-key.pem \
--service-cluster-ip-range=10.32.0.0/24 \
--use-service-account-credentials=true \
--v=2
--node-monitor-period=5s
--node-monitor-grace-period=40s
--pod-eviction-timeout=5m0s
--controllers stringSlice      Default: [*]
A list of controllers to enable. '*' enables all on-by-default controllers, 'foo' enables the controller named 'foo', '-foo' disables the controller named 'foo'.
All controllers: attachdetach, bootstrapsigner, clusterrole-aggregation, cronjob, csapproving,
csrcleaner, csrsigning, daemonset, deployment, disruption, endpoint, garbagecollector,
horizontalpodautoscaling, job, namespace, nodeipam, nodelifecycle, persistentvolume-binder,
persistentvolume-expander, podgc, pv-protection, pvc-protection, replicaset, replicationcontroller,
```

View kube-controller-manager - kubeadm

- kubeadm deploys the kube-controller-manager as a pod in kube-system namespace
- \$ kubectl get pods -n kube-system

I View kube-controller-manager - kubeadm

kubectl get pods -n kube-system						
NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	
kube-system	coredns-78fcdf6894-hwrq9	1/1	Running	0	16m	
kube-system	coredns-78fcdf6894-rzhjr	1/1	Running	0	16m	
kube-system	etcd-master	1/1	Running	0	15m	
kube-system	kube-apiserver-master	1/1	Running	0	15m	
kube-system	kube-controller-manager-master	1/1	Running	0	15m	
kube-system	kube-proxy-lzt6f	1/1	Running	0	16m	
kube-system	kube-proxy-zm5qd	1/1	Running	0	16m	
kube-system	kube-scheduler-master	1/1	Running	0	15m	
kube-system	weave-net-29z42	2/2	Running	1	16m	
kube-system	weave-net-snmdl	2/2	Running	1	16m	-

View kube-controller-manager options - kubeadm

- You can see the options within the pod located at `/etc/kubernetes/manifests/kube-controller-manager.yaml`
- `$ cat /etc/kubernetes/manifests/kube-controller-manager.yaml`

I View kube-controller-manager options - kubeadm

```
▶ 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
```

View kube-controller-manager options - Manual

- In a non-kubeadm setup, you can inspect the options by viewing the `kube-controller-manager.service`
- `$ cat /etc/systemd/system/kube-controller-manager.service`

I View controller-manager options

```
▶ cat /etc/systemd/system/kube-controller-manager.service
[Service]
ExecStart=/usr/local/bin/kube-controller-manager \\
--address=0.0.0.0 \\
--cluster-cidr=10.200.0.0/16 \\
--cluster-name=kubernetes \\
--cluster-signing-cert-file=/var/lib/kubernetes/ca.pem \\
--cluster-signing-key-file=/var/lib/kubernetes/ca-key.pem \\
--kubeconfig=/var/lib/kubernetes/kube-controller-manager.kubeconfig \\
--leader-elect=true \\
--root-ca-file=/var/lib/kubernetes/ca.pem \\
--service-account-private-key-file=/var/lib/kubernetes/service-account-key.pem \\
--service-cluster-ip-range=10.32.0.0/24 \\
--use-service-account-credentials=true \\
--v=2
Restart=on-failure
RestartSec=5
```

- You can also see the running process and affective options by listing the process on master node and searching for kube-controller-manager.
- `$ ps -aux | grep kube-controller-manager`

```
ps -aux | grep kube-controller-manager
root      1994  2.7  5.1 154360 105024 ?      Ssl  06:45   1:25 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
```

K8s Reference Docs:

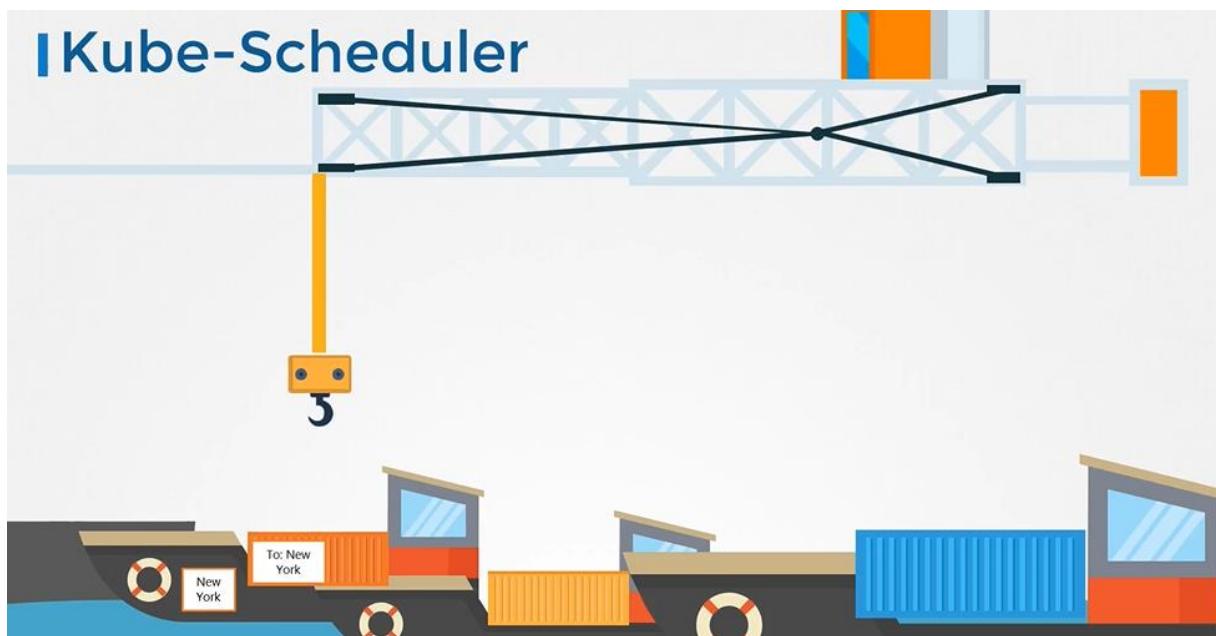
- <https://kubernetes.io/docs/reference/command-line-tools-reference/kube-controller-manager/>
- <https://kubernetes.io/docs/concepts/overview/components/>

Kube Scheduler

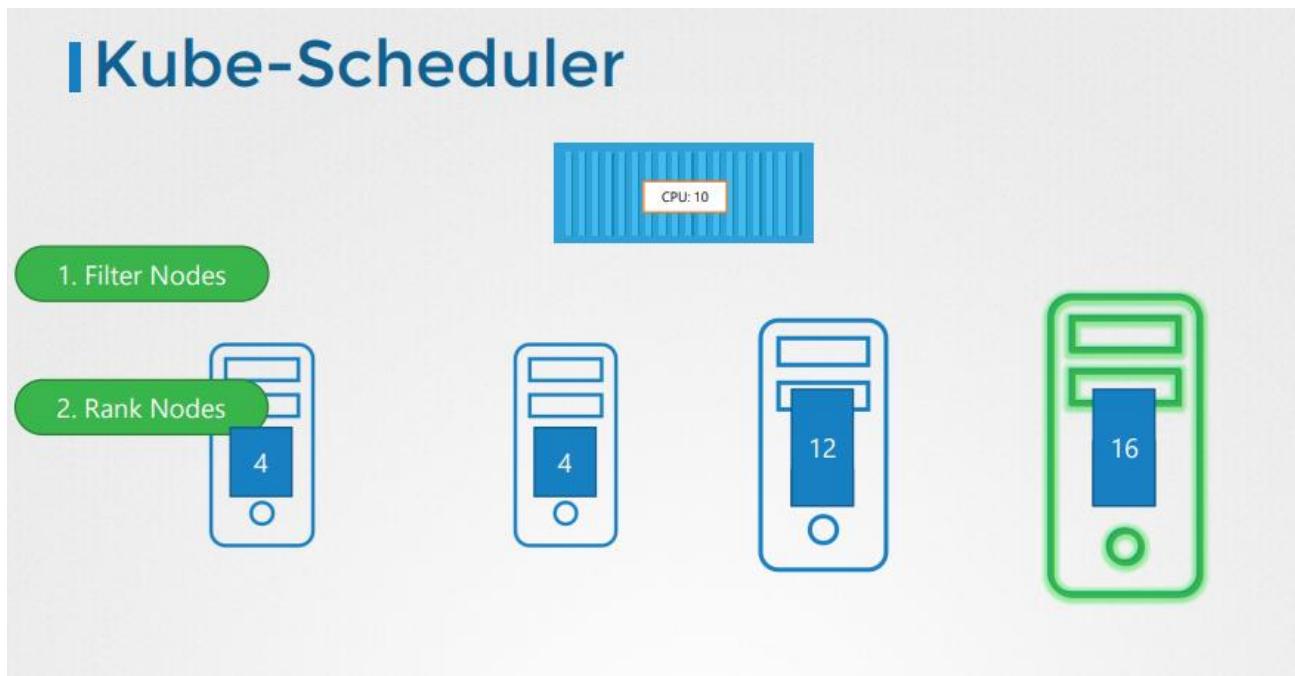
In this section, we will take a look at kube-scheduler.

kube-scheduler is responsible for scheduling pods on nodes.

- The kube-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**.



Why do you need a Scheduler?



Install kube-scheduler - Manual

- Download the kubescheduler binary from the kubernetes release pages [kube-scheduler](#). For example: To download kube-scheduler v1.13.0, Run the below command.
- ```
$ wget https://storage.googleapis.com/kubernetes-release/release/v1.13.0/bin/linux/amd64/kube-scheduler
```
- Extract it
- Run it as a service

### Installing kube-scheduler

```
wget https://storage.googleapis.com/kubernetes-release/release/v1.13.0/bin/linux/amd64/kube-scheduler
kube-scheduler.service
ExecStart=/usr/local/bin/kube-scheduler \
--config=/etc/kubernetes/config/kube-scheduler.yaml \
--v=2
```

## View kube-scheduler options - kubeadm

- If you set it up with kubeadm tool, kubeadm tool will deploy the kube-scheduler as pod in kube-system namespace on master node.
- \$ kubectl get pods -n kube-system
- You can see the options for kube-scheduler in pod definition file that is located at /etc/kubernetes/manifests/kube-scheduler.yaml
- \$ cat /etc/kubernetes/manifests/kube-scheduler.yaml

## I View kube-scheduler options - kubeadm

```
▶ 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
```

- You can also see the running process and affective options by listing the process on master node and searching for kube-apiserver.
- \$ ps -aux | grep kube-scheduler

```
▶ ps -aux | grep kube-scheduler
root 2477 0.8 1.6 48524 34044 ? Ssl 17:31 0:08 kube-scheduler --
address=127.0.0.1 --kubeconfig=/etc/kubernetes/scheduler.conf --leader-elect=true
```

K8s Reference Docs:

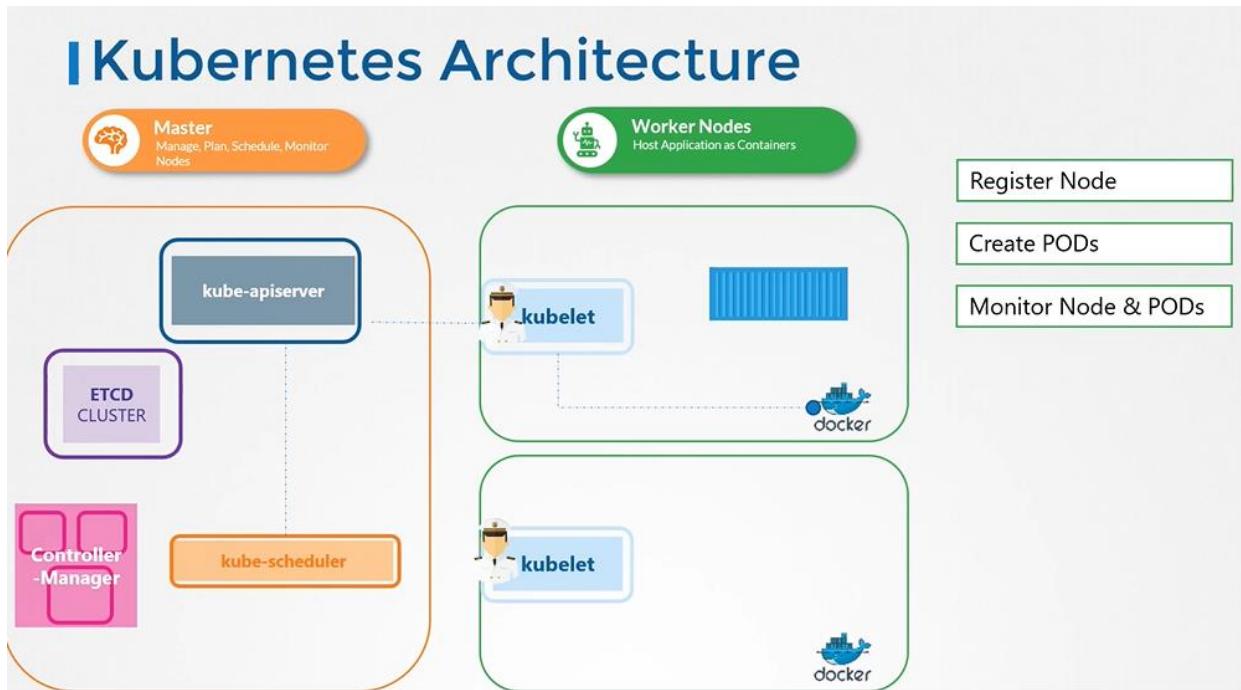
- <https://kubernetes.io/docs/reference/command-line-tools-reference/kube-scheduler/>
- <https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/>
- <https://kubernetes.io/docs/concepts/overview/components/>
- <https://kubernetes.io/docs/tasks/extend-kubernetes/configure-multiple-schedulers/>

## Kubelet

In this section we will take a look at kubelet.

*Kubelet is the sole point of contact for the kubernetes cluster*

- The **kubelet** will create the pods on the nodes, the scheduler only decides which pods goes where.



## Install kubelet

- Kubeadm does not deploy kubelet by default. You must manually download and install it.
- Download the kubelet binary from the Kubernetes release pages [kubelet](#). For example: To download kubelet v1.13.0, Run the below command.
- `$ wget https://storage.googleapis.com/kubernetes-release/release/v1.13.0/bin/linux/amd64/kubelet`
- Extract it
- Run it as a service

## Installing kubelet

```
▶ wget https://storage.googleapis.com/kubernetes-release/release/v1.13.0/bin/linux/amd64/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
```



Kubeadm does not deploy Kubelets

### View kubelet options

- You can also see the running process and affective options by listing the process on worker node and searching for kubelet.
- \$ ps -aux | grep kubelet

## View kubelet options

```
▶ ps -aux | grep kubelet
root 2095 1.8 2.4 960676 98788 ? Ssl 02:32 0:36 /usr/bin/kubelet --bootstrap-
kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf --
config=/var/lib/kubelet/config.yaml --cgroup-driver=cgroupfs --cni-bin-dir=/opt/cni/bin --cni-
conf-dir=/etc/cni/net.d --network-plugin=cni
```

### K8s Reference Docs:

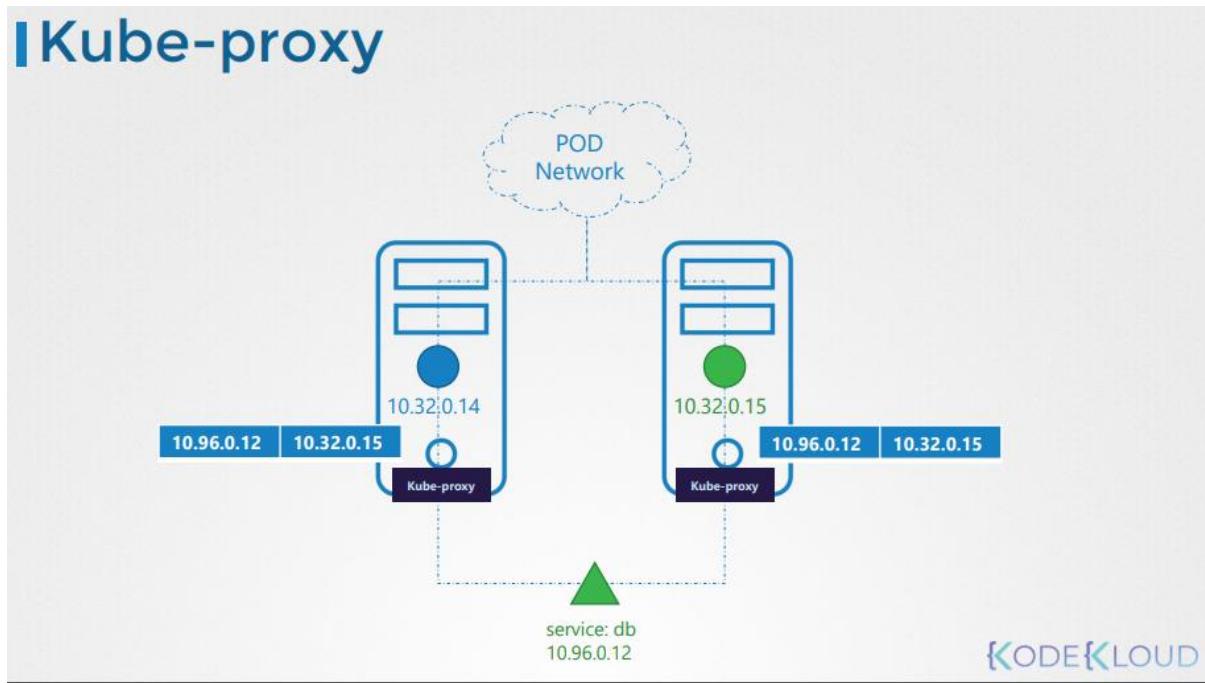
- <https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/>
- <https://kubernetes.io/docs/concepts/overview/components/>
- <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/kubelet-integration/>

## Kube Proxy

In this section, we will take a look at kube-proxy.

Within Kubernetes Cluster, every pod can reach every other pod, this is accomplished by deploying a pod networking cluster to the cluster.

- Kube-Proxy is a process that runs on each node in the kubernetes cluster.



### Install kube-proxy - Manual

- Download the kube-proxy binary from the kubernetes release pages [kube-proxy](#). For example: To download kube-proxy v1.13.0, Run the below command.
- ```
$ wget https://storage.googleapis.com/kubernetes-release/release/v1.13.0/bin/linux/amd64/kube-proxy
```
- Extract it
- Run it as a service

II Installing kube-proxy

```
> wget https://storage.googleapis.com/kubernetes-release/release/v1.13.0/bin/linux/amd64/kube-proxy
[ kube-proxy.service ]
ExecStart=/usr/local/bin/kube-proxy \\
--config=/var/lib/kube-proxy/kube-proxy-config.yaml
Restart=on-failure
RestartSec=5
```

View kube-proxy options - kubeadm

- If you set it up with kubeadm tool, kubeadm tool will deploy the kube-proxy as pod in kube-system namespace. In fact it is deployed as a daemonset on master node.
- \$ kubectl get pods -n kube-system

IView kube-proxy - kubeadm

```
kubectl get pods -n kube-system
NAMESPACE      NAME          READY   STATUS    RESTARTS   AGE
kube-system    coredns-78fcdf6894-hwrq9   1/1     Running   0          16m
kube-system    coredns-78fcdf6894-rzhjr   1/1     Running   0          16m
kube-system    etcd-master             1/1     Running   0          15m
kube-system    kube-apiserver-master   1/1     Running   0          15m
kube-system    kube-controller-manager-master  1/1     Running   0          15m
kube-system    kube-proxy-lzt6f        1/1     Running   0          16m
kube-system    kube-proxy-zm5qd        1/1     Running   0          16m
kube-system    kube-scheduler-master   1/1     Running   0          15m
kube-system    weave-net-29z42        2/2     Running   1          16m
kube-system    weave-net-snmdl       2/2     Running   1          16m

kubectl get daemonset -n kube-system
NAME           DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
kube-proxy      2         2         2       2           2           2           beta.kubernetes.io/arch=amd64   1h
```

K8s Reference Docs:

- <https://kubernetes.io/docs/reference/command-line-tools-reference/kube-proxy/>
- <https://kubernetes.io/docs/concepts/overview/components/>

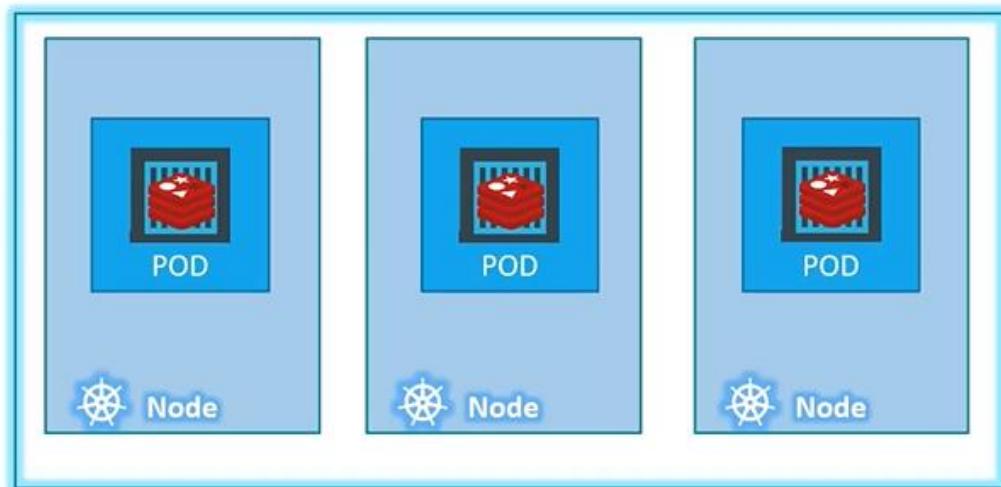
Pods

In this section, we will take a look at PODS.

- POD introduction
- How to deploy pod?

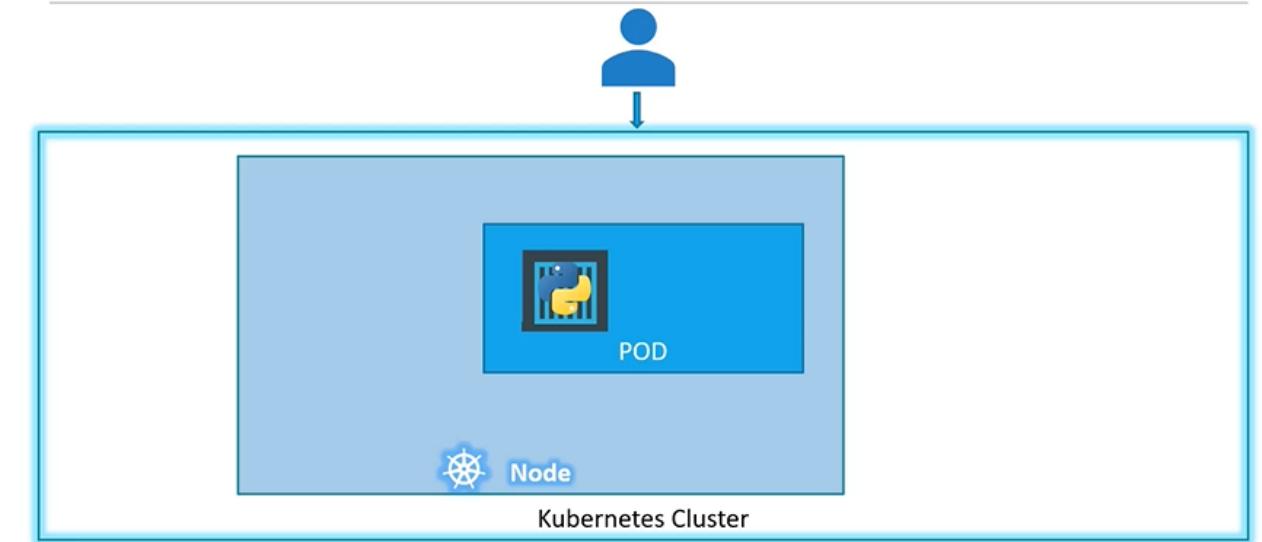
Kubernetes doesn't deploy containers directly on the worker node.

POD



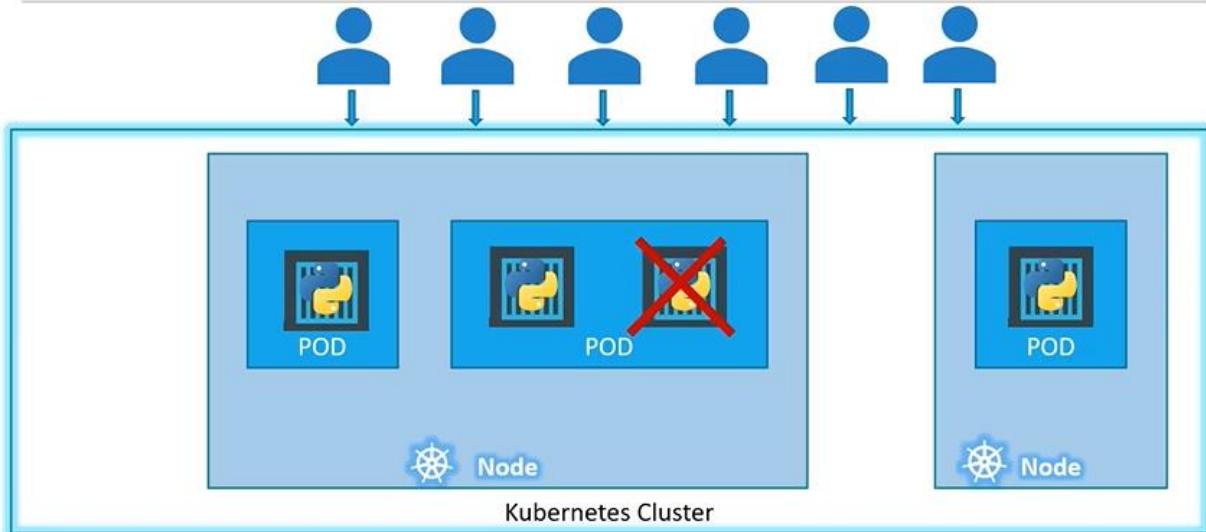
Here is a single node kubernetes cluster with single instance of your application running in a single docker container encapsulated in the pod.

POD



Pod will have a one-to-one relationship with containers running your application.

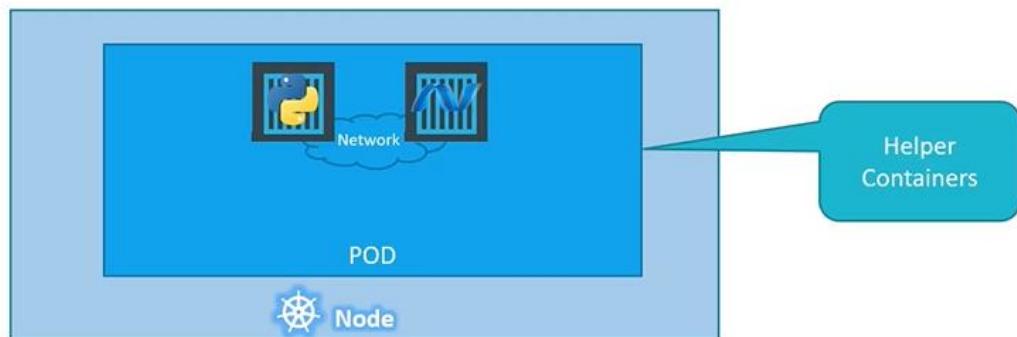
POD



Multi-Container PODs

- A single pod can have multiple containers except for the fact that they are usually not multiple containers of the **same kind**.

Multi-Container PODs



Docker Example (Docker Link)

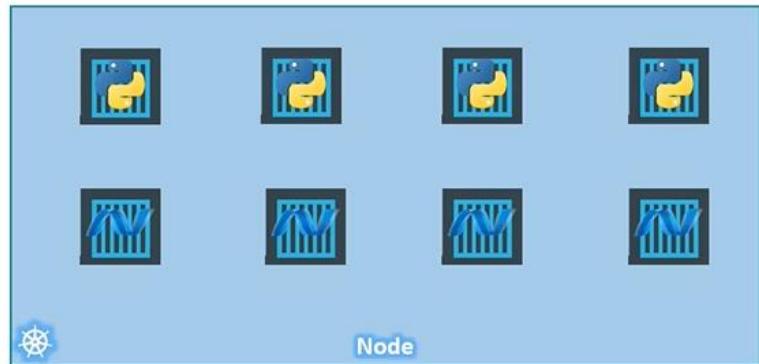
PODs Again!

```
docker run python-app
docker run python-app
docker run python-app
docker run python-app

docker run helper -link app1
docker run helper -link app2
docker run helper -link app3

docker run helper -link app4
```

App	Helper	Volume
Python1	App1	Vol1
Python2	App2	Vol2



Note: I am avoiding networking and load balancing details to keep explanation simple.

How to deploy pods?

Lets now take a look to create a nginx pod using **kubectl**.

- To deploy a docker container by creating a POD.
- \$ kubectl run nginx --image nginx
- To get the list of pods
- \$ kubectl get pods

kubectl

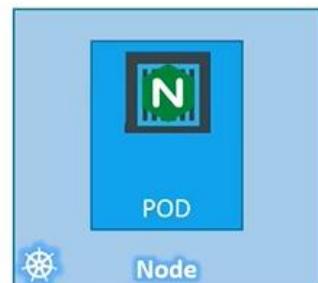


```
kubectl run nginx --image nginx
```

```
kubectl get pods
```

```
C:\Kubernetes>kubectl get pods
NAME          READY   STATUS        RESTARTS   AGE
nginx-8586cf59-whssr  0/1   ContainerCreating   0          3s
```

```
C:\Kubernetes>kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
nginx-8586cf59-whssr  1/1   Running   0          8s
```



K8s Reference Docs:

- <https://kubernetes.io/docs/concepts/workloads/pods/pod/>
- <https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/>
- <https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/>

ReplicaSets

In this section, we will take a look at the below

- Replication Controller
- ReplicaSet

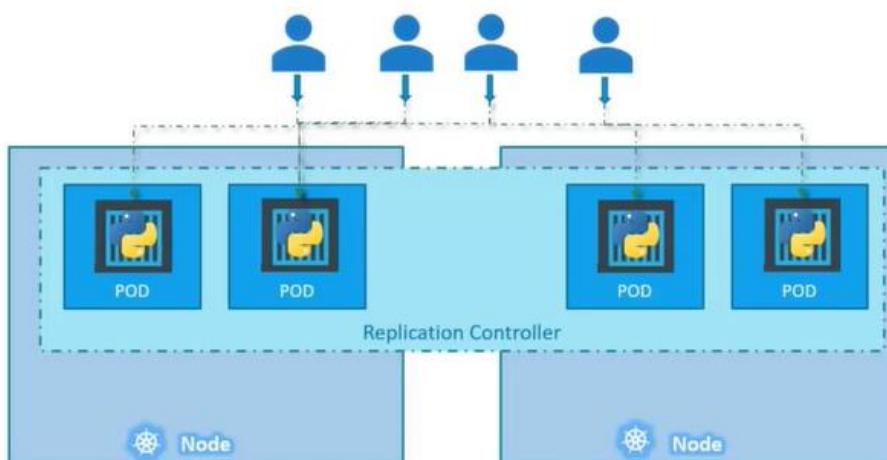
Controllers are brain behind kubernetes

What is a Replica and Why do we need a replication controller?

High Availability



Load Balancing & Scaling



Difference between ReplicaSet and Replication Controller

- **Replication Controller** is the older technology that is being replaced by a **ReplicaSet**.
- **ReplicaSet** is the new way to setup replication.

Creating a Replication Controller

Replication Controller Definition File

```
rc-definition.yml
apiVersion: v1
kind: ReplicationController
metadata:
  name: myapp-rc
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
        - name: nginx-container
          image: nginx
  replicas: 3
```

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: myapp-rc
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
```

```
spec:  
  containers:  
    - name: nginx-container  
      image: nginx  
  replicas: 3
```

- To Create the replication controller
- \$ kubectl create -f rc-definition.yaml
- To list all the replication controllers
- \$ kubectl get replicationcontroller
- To list pods that are launch by the replication controller
- \$ kubectl get pods

```
> kubectl create -f rc-definition.yaml  
replicationcontroller "myapp-rc" created
```

```
> kubectl get replicationcontroller  
NAME      DESIRED   CURRENT   READY     AGE  
myapp-rc   3         3         3         19s
```

```
> kubectl get pods  
NAME        READY   STATUS    RESTARTS   AGE  
myapp-rc-41vk9  1/1    Running   0          20s  
myapp-rc-mc2mf  1/1    Running   0          20s  
myapp-rc-px9pz  1/1    Running   0          20s
```

Creating a ReplicaSet

ReplicaSet Definition File

```
replicaset-definition.yml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
        - name: nginx-container
          image: nginx
    replicas: 3
  selector:
    matchLabels:
      type: front-end
```

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
        - name: nginx-container
          image: nginx
  replicas: 3
```

```
selector:
  matchLabels:
    type: front-end
```

ReplicaSet requires a selector definition when compare to Replication Controller.

- To Create the replicaset
- \$ kubectl create -f replicaset-definition.yaml
- To list all the replicaset
- \$ kubectl get replicaset
- To list pods that are launch by the replicaset
- \$ kubectl get pods

```
> kubectl create -f replicaset-definition.yaml
replicaset "myapp-replicaset" created

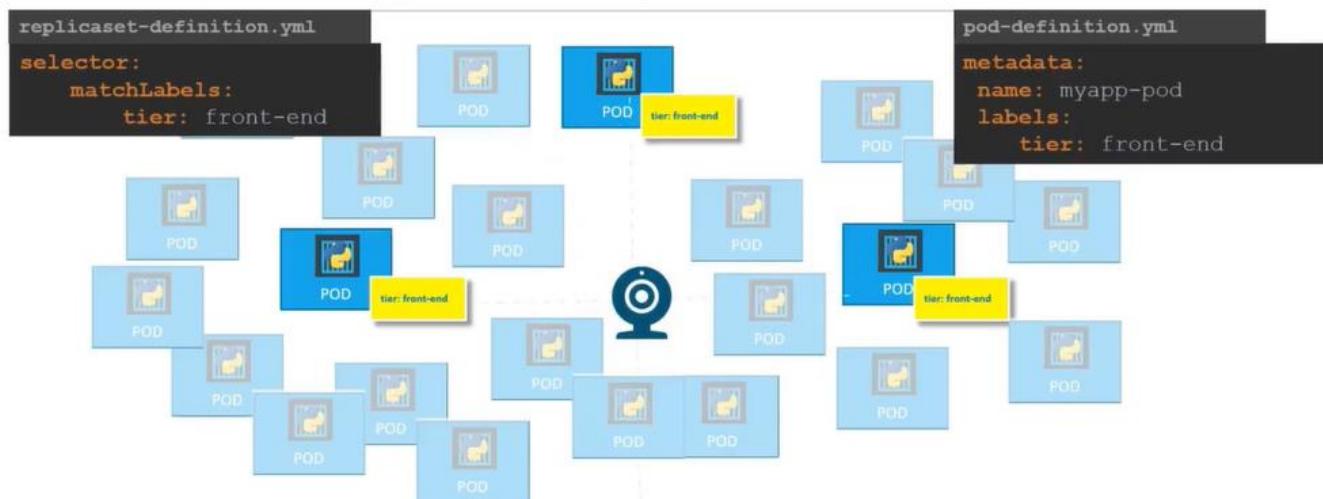
> kubectl get replicaset
NAME      DESIRED   CURRENT   READY   AGE
myapp-replicaset   3        3        3      19s

> kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
myapp-replicaset-9ddl19  1/1     Running   0          45s
myapp-replicaset-9jtpx   1/1     Running   0          45s
myapp-replicaset-hq84m   1/1     Running   0          45s
```

Labels and Selectors

What is the deal with Labels and Selectors? Why do we label pods and objects in kubernetes?

Labels and Selectors



How to scale replicaset

- There are multiple ways to scale replicaset
 - First way is to update the number of replicas in the replicaset-definition.yaml definition file. E.g replicas: 6 and then run

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
        - name: nginx-container
          image: nginx
  replicas: 6
  selector:
    matchLabels:
      type: front-end
$ kubectl apply -f replicaset-definition.yaml
```

- Second way is to use **kubectl scale** command.

```
$ kubectl scale --replicas=6 -f replicaset-definition.yaml
```

- Third way is to use **kubectl scale** command with type and name

```
$ kubectl scale --replicas=6 replicaset myapp-replicaset
```

Scale

```
> kubectl replace -f replicaset-definition.yml
```

```
> kubectl scale --replicas=6 -f replicaset-definition.yml
```

```
> kubectl scale --replicas=6 replicaset myapp-replicaset
```



```
replicaset-definition.yml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
        - name: nginx-container
          image: nginx
  replicas: 6
  selector:
    matchLabels:
      type: front-end
```

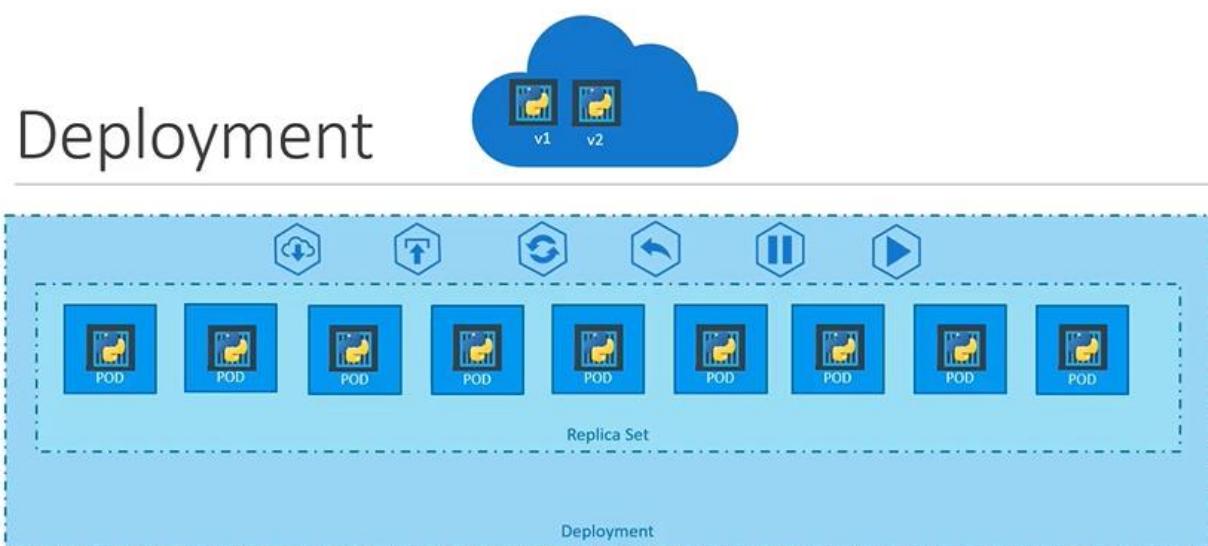
K8s Reference Docs:

- <https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/>
- <https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller/>

Deployments

In this section, we will take a look at kubernetes deployments

Deployment is a kubernetes object.



How do we create deployment?

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
        - name: nginx-container
          image: nginx
  replicas: 3
  selector:
    matchLabels:
      type: front-end
```

- Once the file is ready, create the deployment using deployment definition file
- `$ kubectl create -f deployment-definition.yaml`
- To see the created deployment
- `$ kubectl get deployment`
- The deployment automatically creates a **ReplicaSet**. To see the replicaset
- `$ kubectl get replicaset`
- The replicaset ultimately creates **PODs**. To see the PODs
- `$ kubectl get pods`

Definition

```
> kubectl create -f deployment-definition.yml
deployment "myapp-deployment" created

> kubectl get deployments
NAME      DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
myapp-deployment  3        3        3           3          21s

> kubectl get replicaset
NAME      DESIRED  CURRENT  READY  AGE
myapp-deployment-6795844b58  3        3        3          2m

> kubectl get pods
NAME          READY  STATUS  RESTARTS  AGE
myapp-deployment-6795844b58-5rbjl  1/1    Running   0          2m
myapp-deployment-6795844b58-h4w55  1/1    Running   0          2m
myapp-deployment-6795844b58-1fjhv  1/1    Running   0          2m
```

```
deployment-definition.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
        - name: nginx-container
          image: nginx
  replicas: 3
  selector:
    matchLabels:
      type: front-end
```

- To see the all objects at once
- \$ kubectl get all

```
> kubectl get all
NAME      DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
deploy/myapp-deployment  3        3        3           3          9h

NAME      DESIRED  CURRENT  READY  AGE
rs/myapp-deployment-6795844b58  3        3        3          9h

NAME          READY  STATUS  RESTARTS  AGE
po/myapp-deployment-6795844b58-5rbjl  1/1    Running   0          9h
po/myapp-deployment-6795844b58-h4w55  1/1    Running   0          9h
po/myapp-deployment-6795844b58-1fjhv  1/1    Running   0          9h
```

K8s Reference Docs:

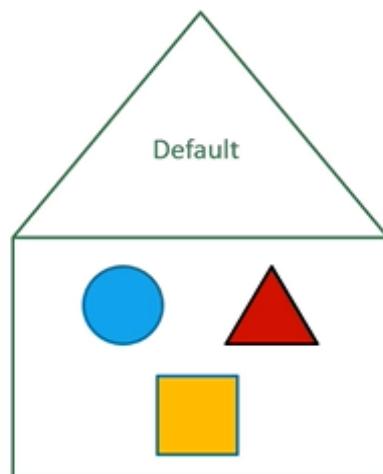
- <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>
- <https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/>
- <https://kubernetes.io/docs/concepts/cluster-administration/manage-deployment/>
- <https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/>

Namespaces

In this section, we will take a look at **Namespaces**

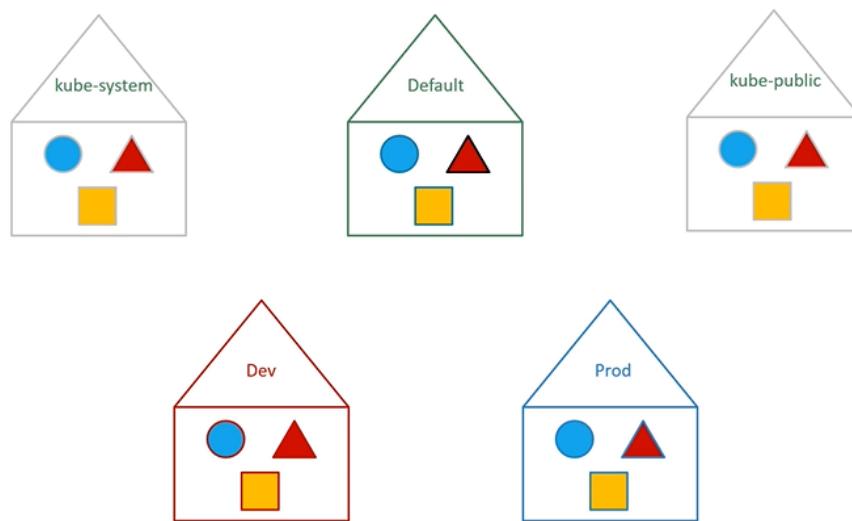
So far in this course we have created **Objects** such as **PODs**, **Deployments** and **Services** in our cluster. Whatever we have been doing we have been doing in a **NAMESPACE**.

- This namespace is the **default** namespace in kubernetes. It is automatically created when kubernetes is setup initially.



- You can create your own namespaces as well.

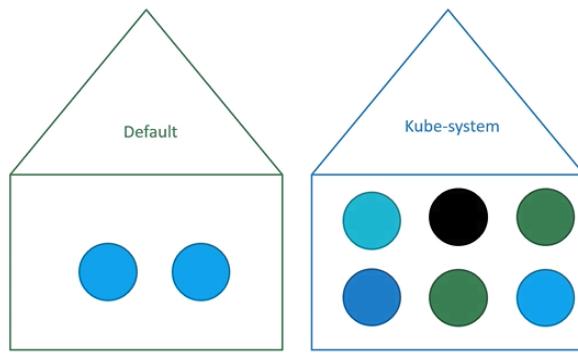
Namespace - Isolation



- To list the pods in default namespace
- `$ kubectl get pods`
- To list the pods in another namespace. Use `kubectl get pods` command along with the `--namespace` flag or argument.
- `$ kubectl get pods --namespace=kube-system`

```
> kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
Pod-1    1/1     Running   0          3d
Pod-2    1/1     Running   0          3d
```

```
> kubectl get pods --namespace=kube-system
NAME                  READY   STATUS    RESTARTS   AGE
coredns-78fcdf6894-92d52 1/1     Running   7          14d
coredns-78fcdf6894-jx25g 1/1     Running   7          14d
etcd-master           1/1     Running   7          14d
kube-apiserver-master 1/1     Running   7          14d
kube-controller-manager-master 1/1     Running   7          14d
kube-flannel-ds-amd64-hz4cf 1/1     Running   14         14d
kube-proxy-4b8tn        1/1     Running   7          14d
kube-proxy-98db4        1/1     Running   7          14d
kube-proxy-jjrb8        1/1     Running   7          14d
kube-scheduler-master   1/1     Running   7          14d
```



- Here we have a pod definition file, when we create a pod with pod-definition file, the pod is created in the default namespace.

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
  containers:
    - name: nginx-container
      image: nginx
```

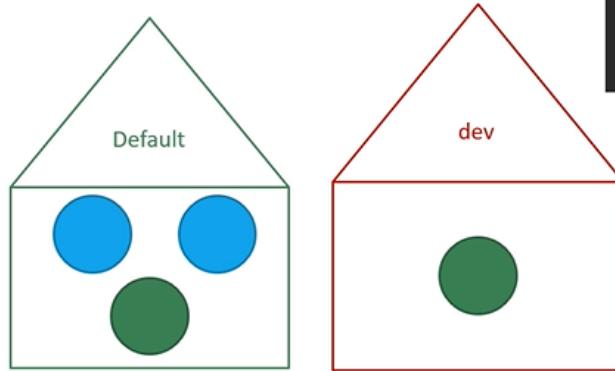
```
$ kubectl create -f pod-definition.yaml
```

- To create the pod with the pod-definition file in another namespace, use the **-namespace** option.
- `$ kubectl create -f pod-definition.yaml --namespace=dev`

```
> kubectl create -f pod-definition.yml
pod/myapp-pod created
```

```
> kubectl create -f pod-definition.yml --namespace=dev
pod/myapp-pod created
```

```
pod-definition.yml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
  containers:
    - name: nginx-container
      image: nginx
```



- If you want to make sure that this pod gets you created in the `dev` env all the time, even if you don't specify in the command line, you can move the `--namespace` definition into the pod-definition file.

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  namespace: dev
  labels:
    app: myapp
    type: front-end
spec:
  containers:
    - name: nginx-container
      image: nginx
```

```
pod-definition.yaml
apiVersion: v1
kind: Pod

metadata:
  name: myapp-pod
  namespace: dev

labels:
  app: myapp
  type: front-end

spec:
  containers:
    - name: nginx-container
      image: nginx
```

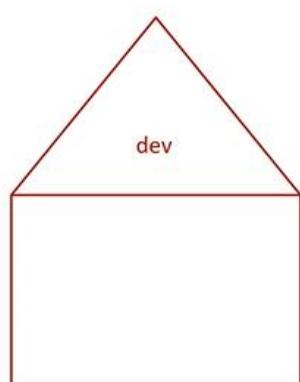
- To create a new namespace, create a namespace definition as shown below and then run `kubectl create`

```
apiVersion: v1
kind: Namespace
metadata:
  name: dev
$ kubectl create -f namespace-dev.yaml
```

Another way to create a namespace

```
$ kubectl create namespace dev
```

Create Namespace



```
namespace-dev.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: dev
```

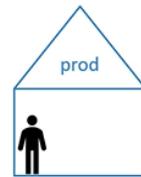
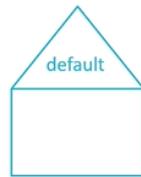
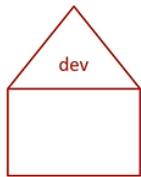
```
> kubectl create -f namespace-dev.yaml
namespace/dev created
```

```
> kubectl create namespace dev
namespace/dev created
```

- By default, we will be in a `default` namespace. To switch to a particular namespace permanently run the below command.
- `$ kubectl config set-context $(kubectl config current-context) --namespace=dev`

- To view pods in all namespaces
- \$ kubectl get pods --all-namespaces

Switch

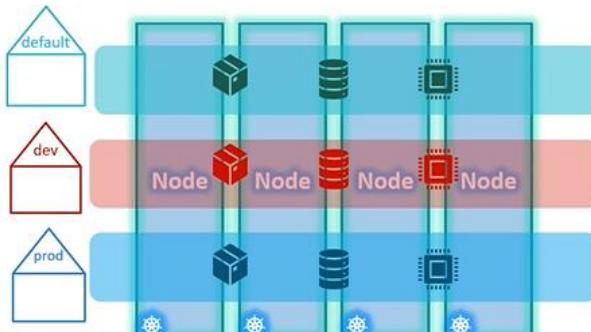


```
> kubectl get pods --namespace=dev      > kubectl get pods      > kubectl get pods --namespace=prod
> kubectl config set-context $(kubectl config current-context) --namespace=dev
> kubectl get pods      > kubectl get pods --namespace=default      > kubectl get pods --namespace=prod
> kubectl config set-context $(kubectl config current-context) --namespace=prod
> kubectl get pods --namespace=dev      > kubectl get pods --namespace=default      > kubectl get pods
> kubectl get pods --all-namespaces
```

- To limit resources in a namespace, create a resource quota. To create one start with **ResourceQuota** definition file.

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

Resource Quota



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

```
> kubectl create -f compute-quota.yaml
```

K8s Reference Docs:

- <https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/>
- <https://kubernetes.io/docs/tasks/administer-cluster/namespaces-walkthrough/>
- <https://kubernetes.io/docs/tasks/administer-cluster/namespaces/>
- <https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/quota-memory-cpu-namespace/>
- <https://kubernetes.io/docs/tasks/access-application-cluster/list-all-running-container-images/>

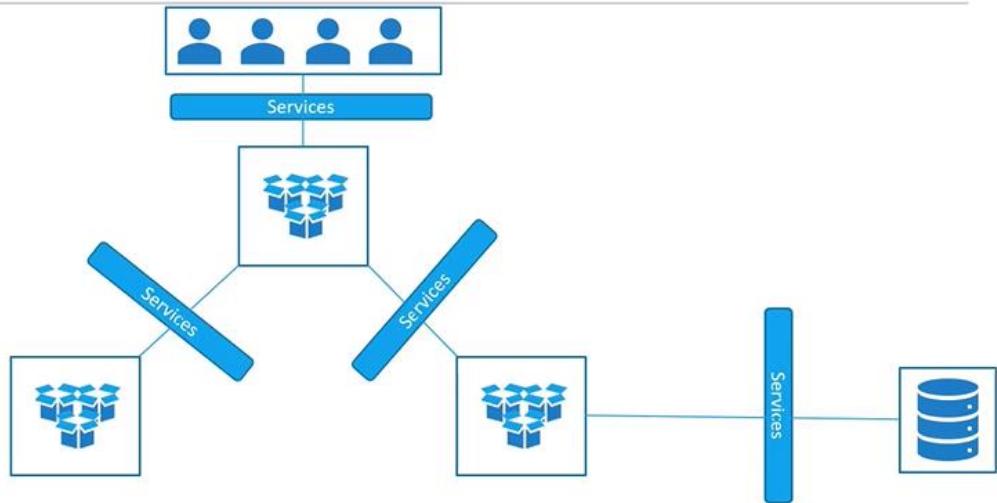
Kubernetes Services

In this section we will take a look at **services** in kubernetes

Services

- Kubernetes Services enables communication between various components within and outside of the application.

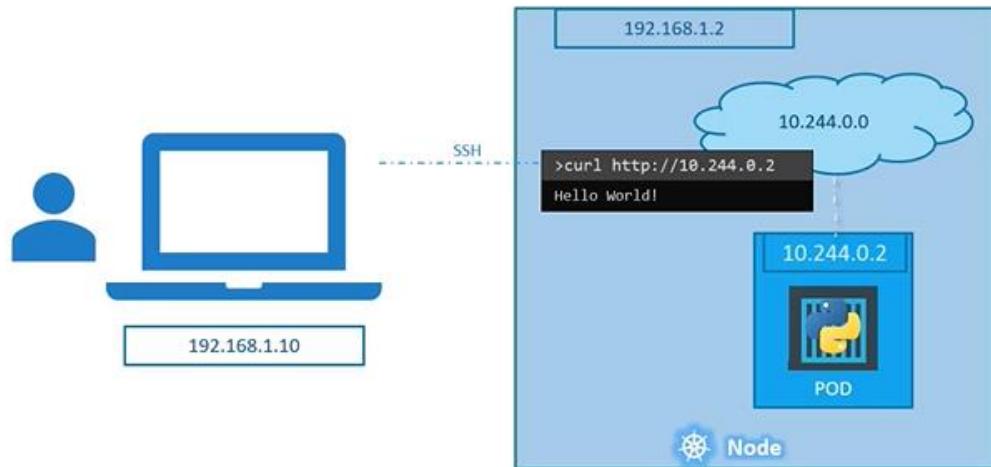
Services



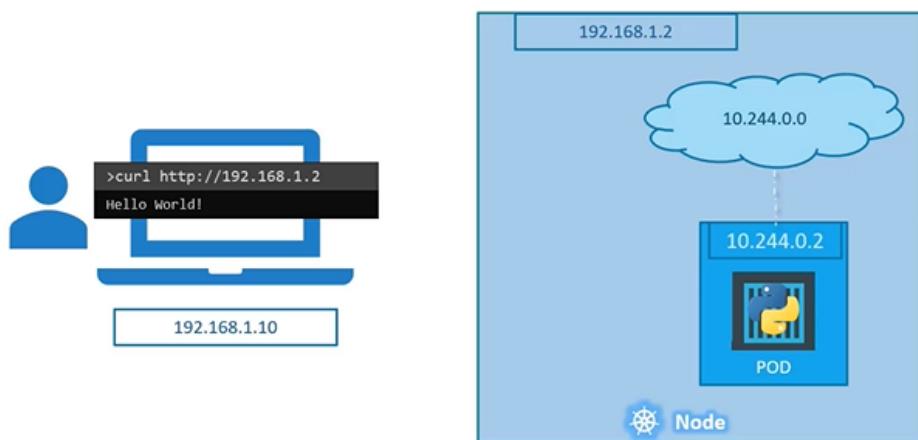
Let's look at some other aspects of networking

External Communication

- How do we as an **external user** access the **web page**?
 - From the node (Able to reach the application as expected)



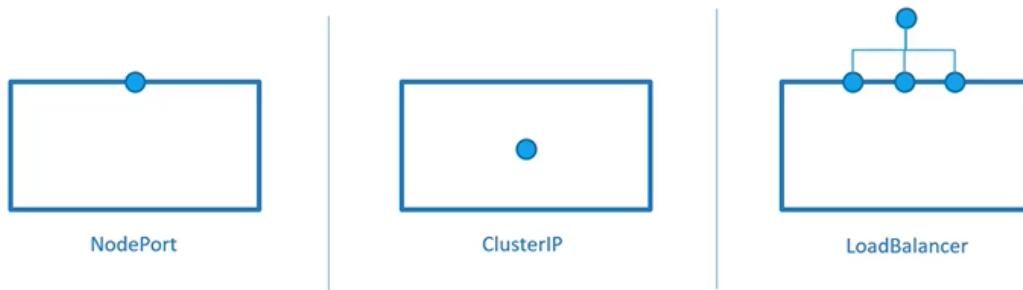
- From outside world (This should be our expectation, without something in the middle it will not reach the application)



Service Types

There are 3 types of service types in kubernetes

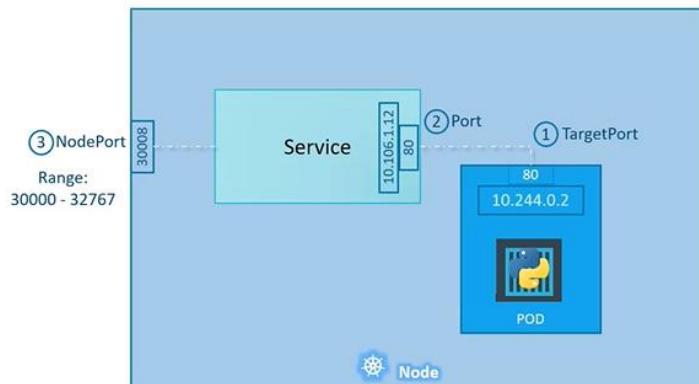
Services Types



1. NodePort

- Where the service makes an internal POD accessible on a POD on the NODE.
- apiVersion: v1
- kind: Service
- metadata:
- name: myapp-service
- spec:
- types: NodePort
- ports:
- - targetPort: 80
- port: 80
- nodePort: 30008

Service - NodePort



```
service-definition.yml
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: NodePort
  ports:
    - targetPort: 80
      *port: 80
      nodePort: 30008
```

To connect the service to the pod

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: NodePort
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30008
  selector:
    app: myapp
    type: front-end
```

Service - NodePort

```
service-definition.yml
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: NodePort
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30008
  selector:
    app: myapp
    type: front-end
```

```
pod-definition.yml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
spec:
  containers:
    - name: nginx-container
      image: nginx
```

To create the service

```
$ kubectl create -f service-definition.yaml
```

To list the services

```
$ kubectl get services
```

To access the application from CLI instead of web browser

```
$ curl http://192.168.1.2:30008
```

Service - NodePort

```
service-definition.yaml
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: NodePort
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30008
  selector:
    app: myapp
    type: front-end
```

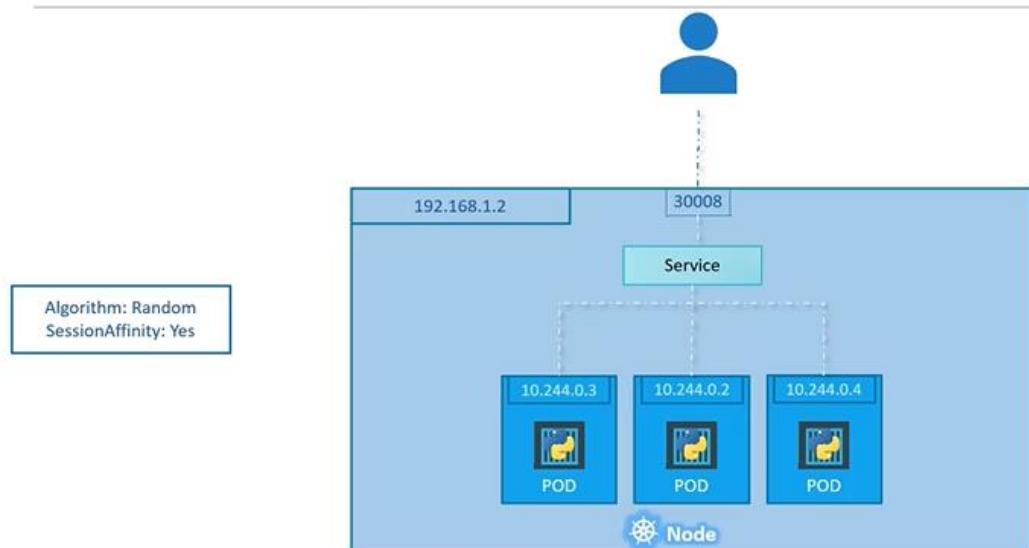
```
> kubectl create -f service-definition.yaml
service "myapp-service" created

> kubectl get services
NAME         TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)     AGE
kubernetes   ClusterIP 10.96.0.1    <none>        443/TCP    16d
myapp-service NodePort   10.106.127.123 <none>        80:30008/TCP 5m

> curl http://192.168.1.2:30008
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
```

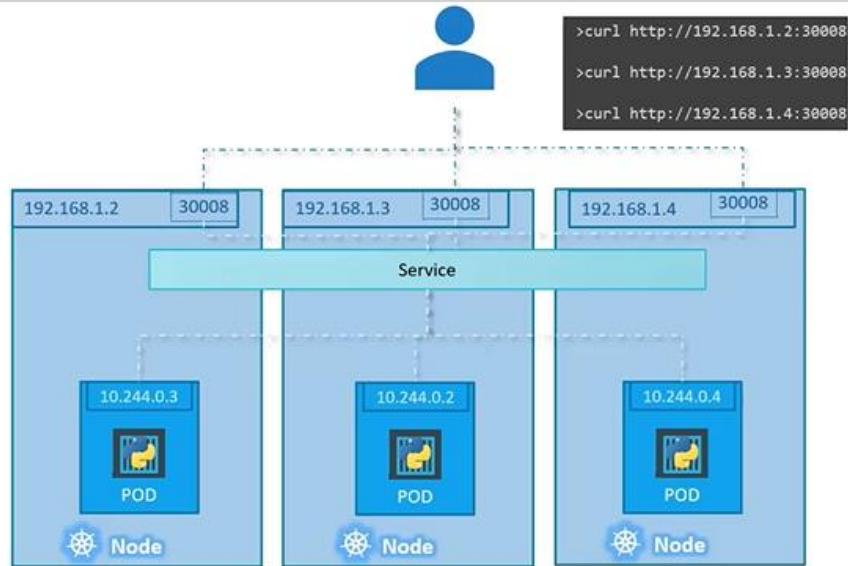
A service with multiple pods

Service - NodePort



When Pods are distributed across multiple nodes

Service - NodePort



2. ClusterIP

- In this case the service creates a **Virtual IP** inside the cluster to enable communication between different services such as a set of frontend servers to a set of backend servers.

3. LoadBalancer

- Where the service provisions a **loadbalancer** for our application in supported cloud providers.

K8s Reference Docs:

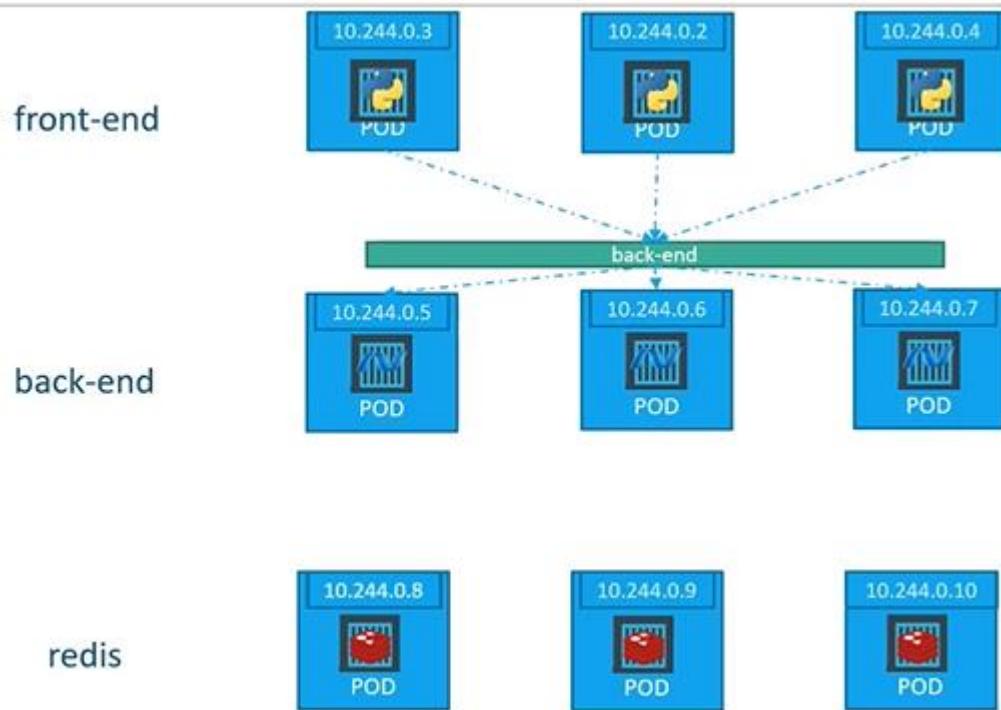
- <https://kubernetes.io/docs/concepts/services-networking/service/>
- <https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/>

Kubernetes Services - ClusterIP

In this section we will take a look at **services - ClusterIP** in kubernetes

ClusterIP

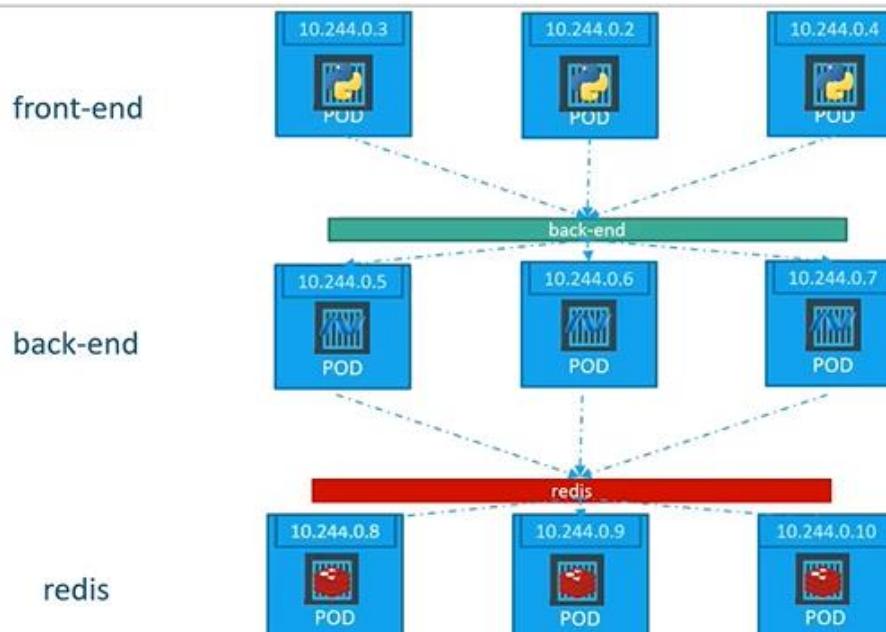
- In this case the service creates a **Virtual IP** inside the cluster to enable communication between different services such as a set of frontend servers to a set of backend servers.



What is a right way to establish connectivity between these services or tiers

- A kubernetes service can help us group the pods together and provide a single interface to access the pod in a group.

ClusterIP



To create a service of type ClusterIP

```
apiVersion: v1
kind: Service
metadata:
  name: back-end
spec:
  type: ClusterIP
  ports:
  - targetPort: 80
    port: 80
  selector:
    app: myapp
    type: back-end
$ kubectl create -f service-definition.yaml
```

To list the services

```
$ kubectl get services
```

```
service-definition.yaml
apiVersion: v1
kind: Service
metadata:
  name: back-end
spec:
  type: ClusterIP
  ports:
  - targetPort: 80
    port: 80
  selector:
    app: myapp
    type: back-end
```

```
> kubectl create -f service-definition.yaml
service "back-end" created
```

```
> kubectl get services
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	16d
back-end	ClusterIP	10.106.127.123	<none>	80/TCP	2m

K8s Reference Docs:

- <https://kubernetes.io/docs/concepts/services-networking/service/>
- <https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/>

Manual Scheduling

In this section, we will take a look at **Manually Scheduling a POD** on a node.

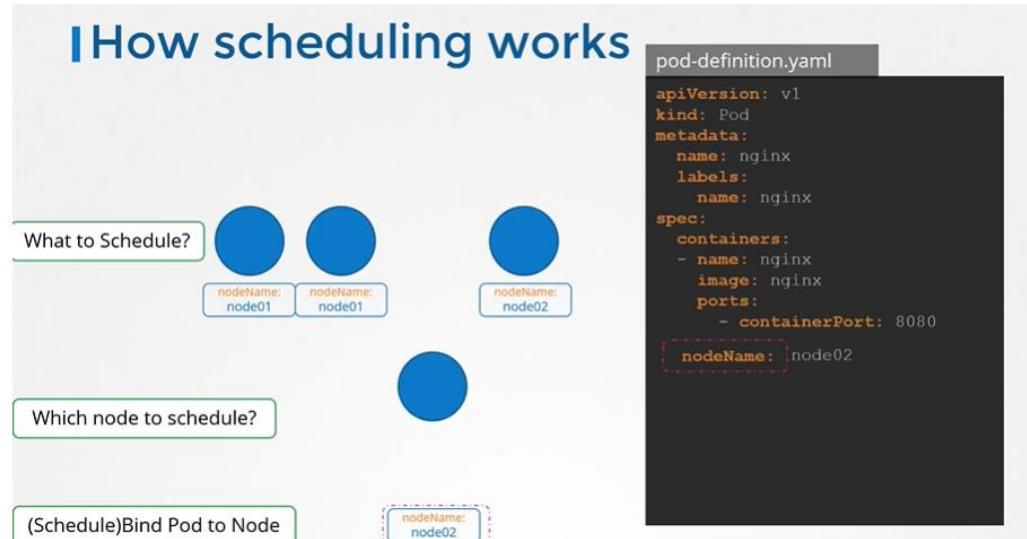
How Scheduling Works

- What do you do when you do not have a scheduler in your cluster?

- Every POD has a field called `nodeName` that by default is not set. You don't typically specify this field when you create the manifest file, kubernetes adds it automatically.
 - Once identified it schedules the POD on the node by setting the `nodeName` property to the name of the node by creating a binding object.
- ```

○ apiVersion: v1
○ kind: Pod
○ metadata:
○ name: nginx
○ labels:
○ name: nginx
○ spec:
○ containers:
○ - name: nginx
○ image: nginx
○ ports:
○ - containerPort: 8080
○ nodeName: node02

```



## No Scheduler

- You can manually assign pods to node itself. Well without a scheduler, to schedule pod is to set `nodeName` property in your pod definition file while creating a pod.

# I No Scheduler!

```
▶ kubectl get pods
```

| NAME  | READY | STATUS  | RESTARTS | AGE |
|-------|-------|---------|----------|-----|
| nginx | 0/1   | Pending | 0        | 3s  |

```
▶ kubectl get pods
```

| NAME  | READY | STATUS  | RESTARTS | AGE | IP        | NODE   |
|-------|-------|---------|----------|-----|-----------|--------|
| nginx | 1/1   | Running | 0        | 9s  | 10.40.0.4 | node02 |

pod-definition.yaml

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx
 labels:
 name: nginx
spec:
 containers:
 - name: nginx
 image: nginx
 ports:
 - containerPort: 8080
 nodeName: node02
```

- Another way
- apiVersion: v1
- kind: Binding
- metadata:
- name: nginx
- target:
- apiVersion: v1
- kind: Node
- name: node02
- apiVersion: v1
- kind: Pod
- metadata:
- name: nginx
- labels:
- name: nginx
- spec:
- containers:
- - name: nginx
- image: nginx
- ports:
- - containerPort: 8080

# I No Scheduler!

Pod-bind-definition.yaml

```
apiVersion: v1
kind: Binding
metadata:
 name: nginx
target:
 apiVersion: v1
 kind: Node
 name: node02
```

pod-definition.yaml

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx
 labels:
 name: nginx
spec:
 containers:
 - name: nginx
 image: nginx
 ports:
 - containerPort: 8080
```

```
▶ curl --header "Content-Type:application/json" --request POST --data '{"apiVersion":"v1", "kind": "Binding" ... }' http://$SERVER/api/v1/namespaces/default/pods/$PODNAME/binding/
```

K8s Reference Docs:

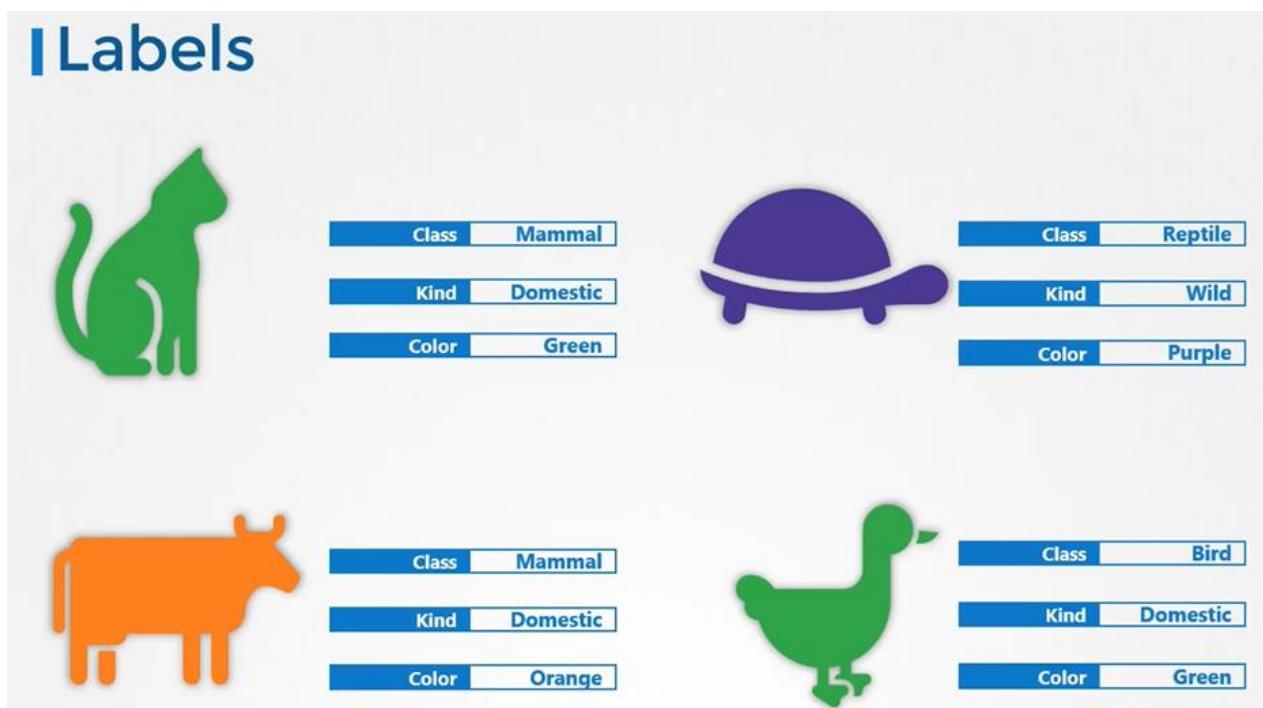
- <https://kubernetes.io/docs/reference/using-api/api-concepts/>
- <https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#nodename>

## Labels and Selectors

In this section, we will take a look at **Labels and Selectors**

*Labels and Selectors are standard methods to group things together.*

*Labels are properties attached to each item.*



Selectors help you to filter these items

## I Selectors



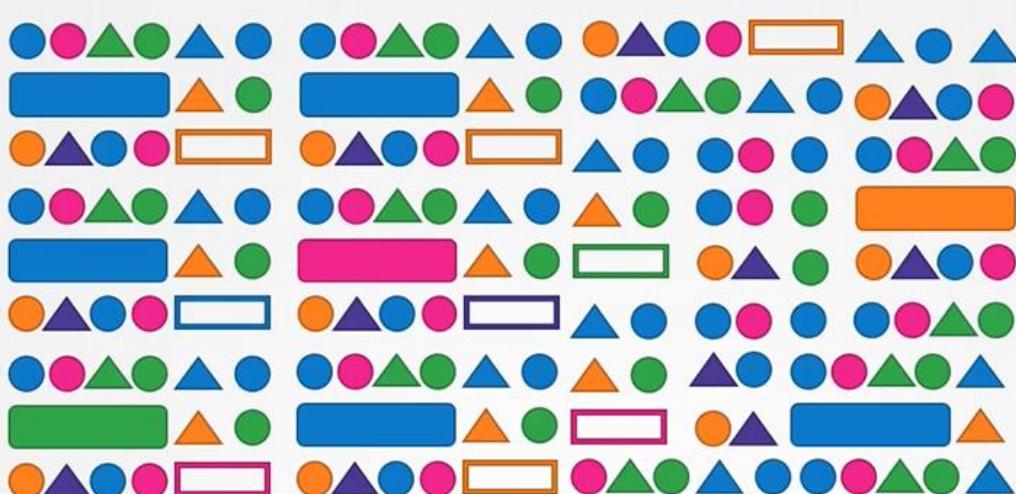
|       |          |
|-------|----------|
| Class | Mammal   |
| Kind  | Domestic |
| Color | Green    |

Class = Mammal  
&  
Color = Green

How are labels and selectors are used in kubernetes?

- We have created different types of objects in kubernetes such as **PODs**, **ReplicaSets**, **Deployments** etc.

## I Labels & Selectors in Kubernetes



How do you specify labels?

```
apiVersion: v1
kind: Pod
metadata:
 name: simple-webapp
 labels:
 app: App1
```

```

function: Front-end
spec:
 containers:
 - name: simple-webapp
 image: simple-webapp
 ports:
 - containerPort: 8080

```

## Labels



### pod-definition.yaml

```

apiVersion: v1
kind: Pod
metadata:
 name: simple-webapp
 labels:
 app: App1
 function: Front-end

spec:
 containers:
 - name: simple-webapp
 image: simple-webapp
 ports:
 - containerPort: 8080

```

Once the pod is created, to select the pod with labels run the below command

```
$ kubectl get pods --selector app=App1
```

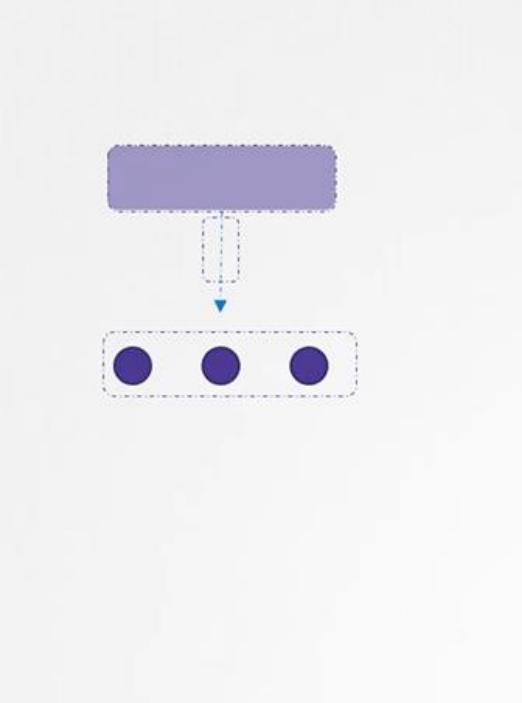
Kubernetes uses labels to connect different objects together

```

apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: simple-webapp
 labels:
 app: App1
 function: Front-end
spec:
 replicas: 3
 selector:
 matchLabels:
 app: App1
 template:
 metadata:
 labels:
 app: App1
 function: Front-end
 spec:
 containers:
 - name: simple-webapp
 image: simple-webapp

```

# ReplicaSet



```
replicaset-definition.yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: simple-webapp
 labels:
 app: App1
 function: Front-end
spec:
 replicas: 3
 selector:
 matchLabels:
 app: App1
 template:
 metadata:
 labels:
 app: App1
 function: Front-end
 spec:
 containers:
 - name: simple-webapp
 image: simple-webapp
```

For services

```
...
apiVersion: v1
kind: Service
metadata:
 name: my-service
spec:
 selector:
 app: App1
 ports:
 - protocol: TCP
 port: 80
 targetPort: 9376
...
```

# IService



## service-definition.yaml

```
apiVersion: v1
kind: Service
metadata:
 name: my-service
spec:
 selector:
 app: App1
 ports:
 - protocol: TCP
 port: 80
 targetPort: 9376
```

## replicaset-definition.yaml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: simple-webapp
 labels:
 app: App1
 function: Front-end
spec:
 replicas: 3
 selector:
 matchLabels:
 app: App1
 template:
 metadata:
 labels:
 app: App1
 function: Front-end
 spec:
 containers:
 - name: simple-webapp
 image: simple-webapp
```

## Annotations

- While labels and selectors are used to group objects, annotations are used to record other details for informative purpose.
- apiVersion: apps/v1
- kind: ReplicaSet
- metadata:
- name: simple-webapp
- labels:
- app: App1
- function: Front-end
- annotations:
- buildversion: 1.34
- spec:
- replicas: 3
- selector:
- matchLabels:
- app: App1
- template:
- metadata:
- labels:
- app: App1
- function: Front-end
- spec:
- containers:
- - name: simple-webapp

- image: simple-webapp

# Annotations

replicaset-definition.yaml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: simple-webapp
 labels:
 app: Appl
 function: Front-end
 annotations:
 buildversion: 1.34
spec:
 replicas: 3
 selector:
 matchLabels:
 app: Appl
 template:
 metadata:
 labels:
 app: Appl
 function: Front-end
 spec:
 containers:
 - name: simple-webapp
 image: simple-webapp
```

K8s Reference Docs:

- <https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/>

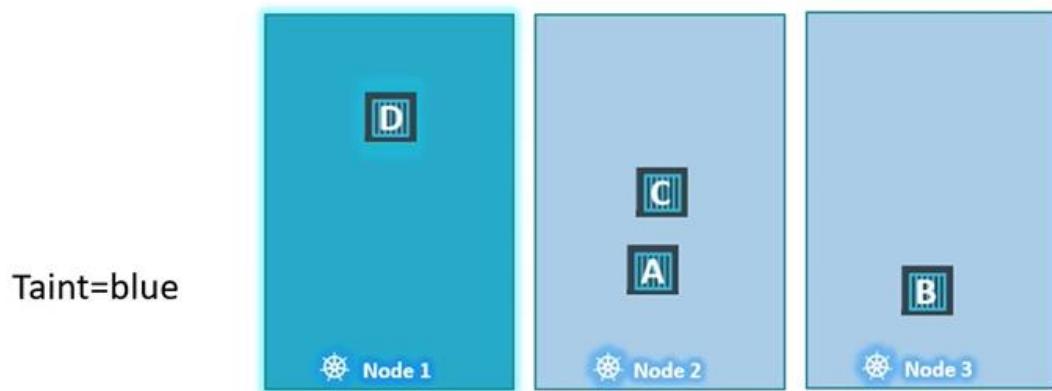
## Taints and Tolerations

In this section, we will take a look at taints and tolerations.

- Pod to node relationship and how you can restrict what pods are placed on what nodes.

*Taints and Tolerations are used to set restrictions on what pods can be scheduled on a node.*

- Only pods which are tolerant to the particular taint on a node will get scheduled on that node.



## Taints

- Use `kubectl taint nodes` command to taint a node.

### Syntax

```
$ kubectl taint nodes <node-name> key=value:taint-effect
```

### Example

```
$ kubectl taint nodes node1 app=blue:NoSchedule
```

- The taint effect defines what would happen to the pods if they do not tolerate the taint.
- There are 3 taint effects
  - `NoSchedule`
  - `PreferNoSchedule`
  - `NoExecute`

## ✍ Taints - Node

```
kubectl taint nodes node-name key=value:taint-effect
```

What happens to PODS  
that do not tolerate this taint?

NoSchedule | PreferNoSchedule | NoExecute

```
kubectl taint nodes node1 app=blue:NoSchedule
```

## Tolerations

- Tolerations are added to pods by adding a `tolerations` section in pod definition.
- `apiVersion: v1`

- kind: Pod
- metadata:
- name: myapp-pod
- spec:
- containers:
- - name: nginx-container
- image: nginx
- tolerations:
- - key: "app"
- operator: "Equal"
- value: "blue"
- effect: "NoSchedule"

## ★ Tolerations - PODs

```
kubectl taint nodes node1
```

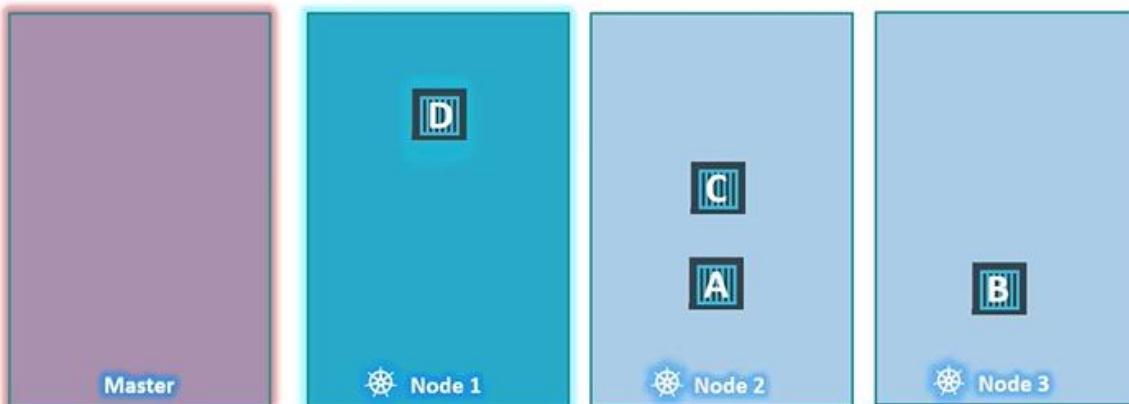
```
pod-definition.yml
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
spec:
 containers:
 - name: nginx-container
 image: nginx

 tolerations:
 - key: "app"
 operator: "Equal"
 value: "blue"
 effect: "NoSchedule"
```

*Taints and Tolerations do not tell the pod to go to a particular node. Instead, they tell the node to only accept pods with certain tolerations.*

- To see this taint, run the below command
- \$ kubectl describe node kubemaster |grep Taint

```
kubectl describe node kubemaster | grep Taint
Taints: node-role.kubernetes.io/master:NoSchedule
```



#### K8s Reference Docs

- <https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/>

## Node Selectors

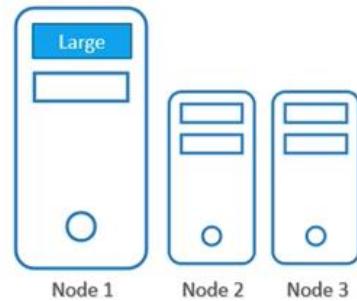
In this section, we will take a look at Node Selectors in Kubernetes

*We add new property called Node Selector to the spec section and specify the label.*

- The scheduler uses these labels to match and identify the right node to place the pods on.
- apiVersion: v1
- kind: Pod
- metadata:
- name: myapp-pod
- spec:
- containers:
- - name: data-processor
- image: data-processor
- nodeSelector:
- size: Large

# Node Selectors

```
pod-definition.yml
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
spec:
 containers:
 - name: data-processor
 image: data-processor
 nodeSelector:
 size: Large
```



- To label nodes

## Syntax

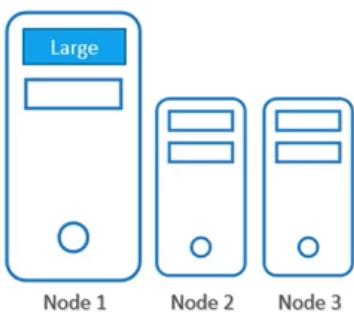
```
$ kubectl label nodes <node-name> <label-key>=<label-value>
```

## Example

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

# Label Nodes

```
▶ kubectl label nodes <node-name> <label-key>=<label-value>
▶ kubectl label nodes node-1 size=Large
```

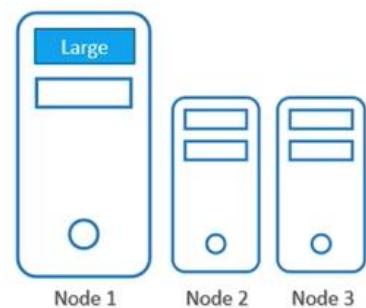


- To create a pod definition
- apiVersion: v1
- kind: Pod
- metadata:
- name: myapp-pod

- spec:
- containers:
- - name: data-processor
- image: data-processor
- nodeSelector:
- size: Large
- \$ kubectl create -f pod-definition.yml

## Node Selectors

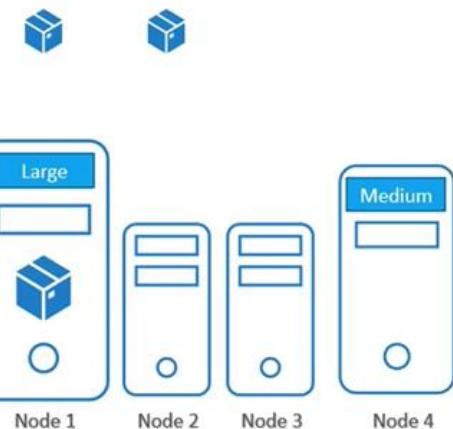
```
pod-definition.yml
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
spec:
 containers:
 - name: data-processor
 image: data-processor
 nodeSelector:
 size: Large
```



### Node Selector - Limitations

- We used a single label and selector to achieve our goal here. But what if our requirement is much more complex.

## Node Selector - Limitations



- Large OR Medium?
- NOT Small

- For this we have **Node Affinity and Anti Affinity**

## K8s Reference Docs

- <https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#nodeselector>

# Node Affinity

In this section, we will talk about "Node Affinity" feature in kubernetes.

*The primary feature of Node Affinity is to ensure that the pods are hosted on particular nodes.*

- With **Node Selectors** we cannot provide the advance expressions.
- apiVersion: v1
- kind: Pod
- metadata:
- name: myapp-pod
- spec:
- containers:
- - name: data-processor
- image: data-processor
- nodeSelector:
- size: Large

## Node Selectors

---

```
pod-definition.yml
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
spec:
 containers:
 - name: data-processor
 image: data-processor
 nodeSelector:
 size: Large
```

- Large OR Medium?
- NOT Small

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
spec:
 containers:
 - name: data-processor
 image: data-processor
```

```
affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: size
 operator: In
 values:
 - Large
 - Medium
```

# Node Affinity

pod-definition.yml

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
spec:
 containers:
 - name: data-processor
 image: data-processor
 affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: size
 operator: In
 values:
 - Large
 - Medium
```

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
spec:
 containers:
 - name: data-processor
 image: data-processor
 affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: size
 operator: NotIn
 values:
 - Small
```

# Node Affinity

```
pod-definition.yml
```

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
spec:
 containers:
 - name: data-processor
 image: data-processor
 affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: size
 operator: NotIn
 values:
 - Small
```

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
spec:
 containers:
 - name: data-processor
 image: data-processor
 affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: size
 operator: Exists
```

# Node Affinity

```
pod-definition.yml
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
spec:
 containers:
 - name: data-processor
 image: data-processor
 affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: size
 operator: Exists
```

## Node Affinity Types

- Available
  - requiredDuringSchedulingIgnoredDuringExecution
  - preferredDuringSchedulingIgnoredDuringExecution
- Planned
  - requiredDuringSchedulingRequiredDuringExecution
  - preferredDuringSchedulingRequiredDuringExecution

# Node Affinity Types

Available:

**requiredDuringSchedulingIgnoredDuringExecution**

**preferredDuringSchedulingIgnoredDuringExecution**

Planned:

**requiredDuringSchedulingRequiredDuringExecution**

**preferredDuringSchedulingRequiredDuringExecution**

## Node Affinity Types States

# Node Affinity Types

Available:

**requiredDuringSchedulingIgnoredDuringExecution**

**preferredDuringSchedulingIgnoredDuringExecution**

|        | DuringScheduling | DuringExecution |
|--------|------------------|-----------------|
| Type 1 | Required         | Ignored         |
| Type 2 | Preferred        | Ignored         |

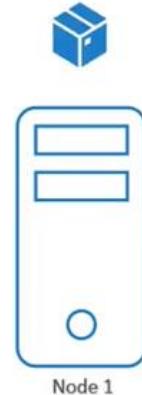
# Node Affinity Types

Planned:

**requiredDuringSchedulingRequiredDuringExecution**

**preferredDuringSchedulingRequiredDuringExecution**

|        | DuringScheduling | DuringExecution |
|--------|------------------|-----------------|
| Type 1 | Required         | Ignored         |
| Type 2 | Preferred        | Ignored         |
| Type 3 | Required         | Required        |
| Type 4 | Preferred        | Required        |



Node 1

## K8s Reference Docs

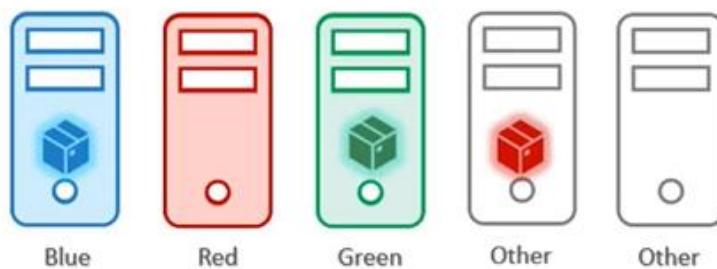
- <https://kubernetes.io/docs/tasks/configure-pod-container/assign-pods-nodes-using-node-affinity/>
- <https://kubernetes.io/blog/2017/03/advanced-scheduling-in-kubernetes/>

## Taints and Tolerations vs Node Affinity

In this section, we will take a look at Taints and Tolerations vs Node Affinity

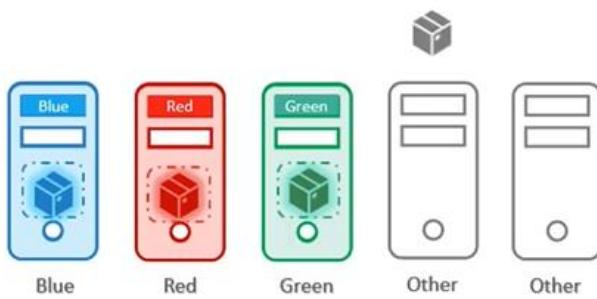
- Taints and Tolerations do not guarantee that the pods will only prefer these nodes; in this case, the red pods may end up on one of the other nodes that do not have a taint or toleration set.

# Taints and Tolerations



- As such, a combination of taints and tolerations and node affinity rules can be used together to completely dedicate nodes for specific parts.

## Taints/Tolerations and Node Affinity



Taints/Tolerations 4

K8s Reference Docs:

- <https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/>
- <https://kubernetes.io/docs/tasks/configure-pod-container/assign-pods-nodes-using-node-affinity/>

# Resource Limits

In this section we will take a look at Resource Limits

*Let us take a look at 3 node kubernetes cluster.*

- Each node has a set of CPU, Memory and Disk resources available.
- If there is no sufficient resources available on any of the nodes, kubernetes holds the scheduling the pod. You will see the pod in pending state. If you look at the events, you will see the reason as insufficient CPU.



## Resource Requirements

- By default, K8s assume that a pod or container within a pod requires **0.5 CPU** and **256Mi** of memory. This is known as the **Resource Request for a container**.

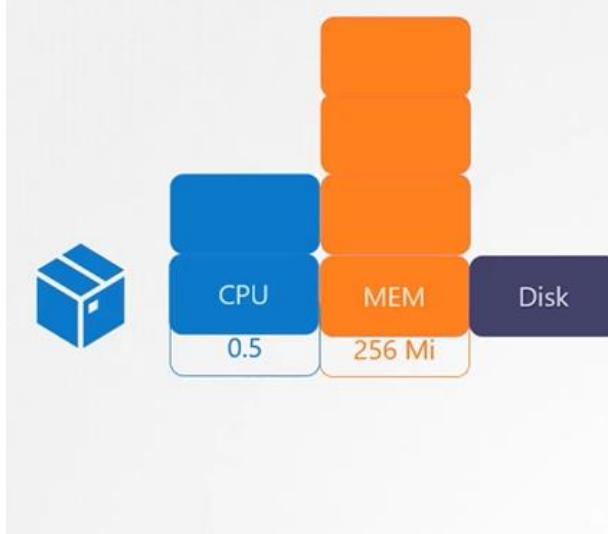
# Resource Requests



- If your application within the pod requires more than the default resources, you need to set them in the pod definition file.

```
• apiVersion: v1
• kind: Pod
• metadata:
• name: simple-webapp-color
• labels:
• name: simple-webapp-color
• spec:
• containers:
• - name: simple-webapp-color
• image: simple-webapp-color
• ports:
• - containerPort: 8080
• resources:
• requests:
• memory: "1Gi"
• cpu: "1"
```

# I Resource Requests



pod-definition.yaml

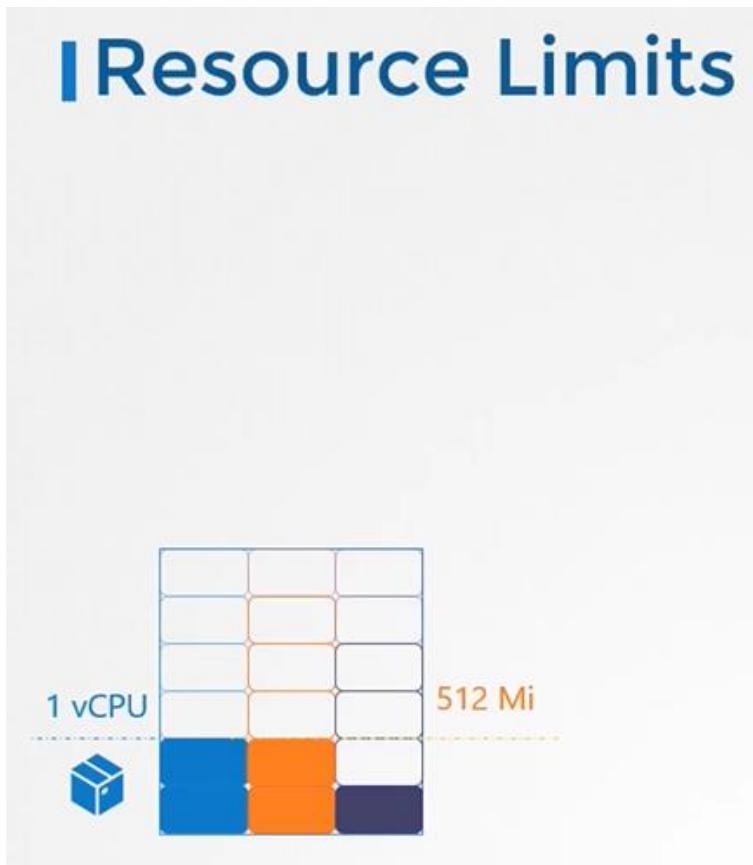
```

apiVersion: v1
kind: Pod
metadata:
 name: simple-webapp-color
 labels:
 name: simple-webapp-color
spec:
 containers:
 - name: simple-webapp-color
 image: simple-webapp-color
 ports:
 - containerPort: 8080
 resources:
 requests:
 memory: "1Gi"
 cpu: 1

```

## Resources - Limits

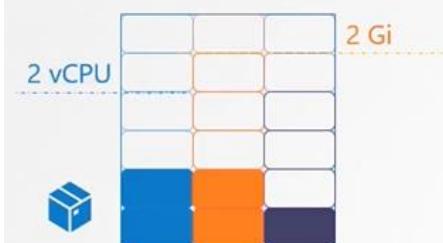
- By default, k8s sets resource limits to 1 CPU and 512Mi of memory



- You can set the resource limits in the pod definition file.

- apiVersion: v1
- kind: Pod
- metadata:
- name: simple-webapp-color
- labels:
  - name: simple-webapp-color
- spec:
  - containers:
    - name: simple-webapp-color
  - image: simple-webapp-color
  - ports:
    - containerPort: 8080
  - resources:
    - requests:
      - memory: "1Gi"
      - cpu: "1"
    - limits:
      - memory: "2Gi"
      - cpu: "2"

## I Resource Limits



pod-definition.yaml

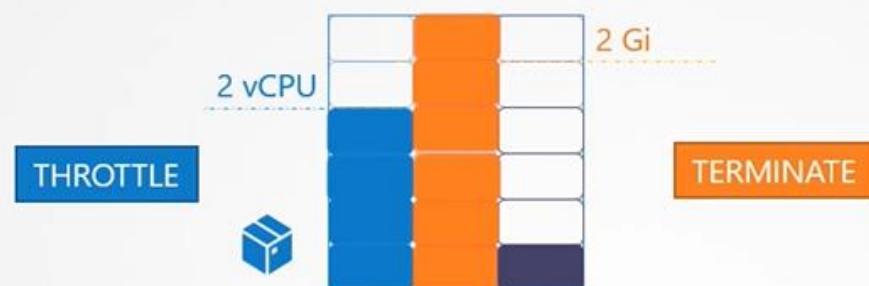
```
apiVersion: v1
kind: Pod
metadata:
 name: simple-webapp-color
 labels:
 name: simple-webapp-color
spec:
 containers:
 - name: simple-webapp-color
 image: simple-webapp-color
 ports:
 - containerPort: 8080
 resources:
 requests:
 memory: "1Gi"
 cpu: 1
 limits:
 memory: "2Gi"
 cpu: 2
```

*Note: Remember Requests and Limits for resources are set per container in the pod.*

### Exceed Limits

- what happens when a pod tries to exceed resources beyond its limits?

# I Exceed Limits



*K8s Reference Docs:*

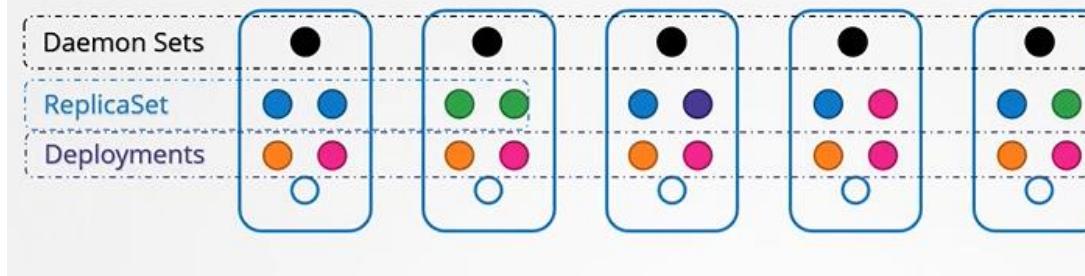
- <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

## DaemonSets

In this section, we will take a look at DaemonSets.

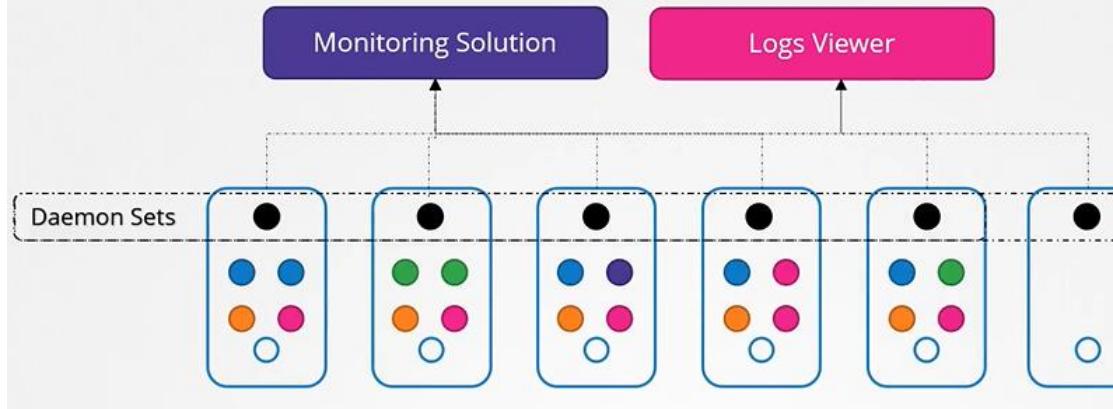
*DaemonSets are like replicsets, as it helps in to deploy multiple instances of pod. But it runs one copy of your pod on each node in your cluster.*

## I Daemon Sets

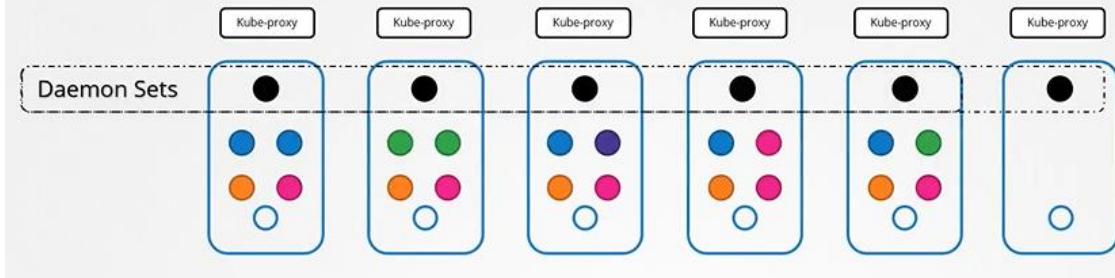


DaemonSets - UseCases

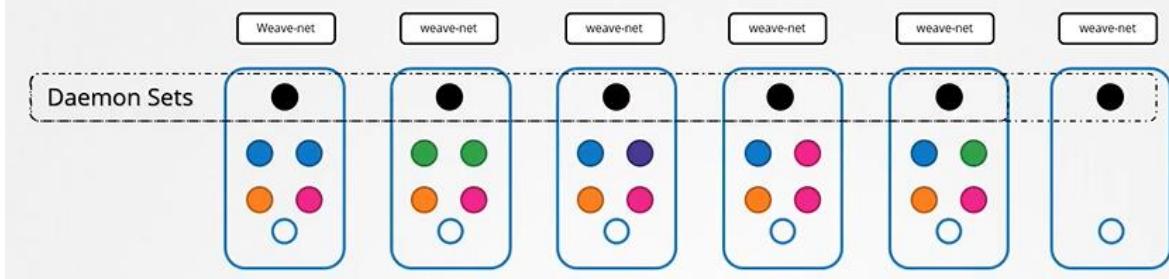
## I Daemon Sets – UseCase



## I Daemon Sets - UseCase - kube-proxy



## I Daemon Sets - UseCase - Networking



### DaemonSets - Definition

- Creating a DaemonSet is similar to the ReplicaSet creation process.
- For DaemonSets, we start with apiVersion, kind as **DaemonSets** instead of **ReplicaSet**, metadata and spec.
- apiVersion: apps/v1
- kind: Replicaset
- metadata:
- name: monitoring-daemon
- labels:
- app: nginx
- spec:
- selector:
- matchLabels:
- app: monitoring-agent
- template:
- metadata:
- labels:

```

• app: monitoring-agent
• spec:
• containers:
• - name: monitoring-agent
• image: monitoring-agent
• apiVersion: apps/v1
• kind: DaemonSet
• metadata:
• name: monitoring-daemon
• labels:
• app: nginx
• spec:
• selector:
• matchLabels:
• app: monitoring-agent
• template:
• metadata:
• labels:
• app: monitoring-agent
• spec:
• containers:
• - name: monitoring-agent
• image: monitoring-agent

```

## DaemonSet Definition

daemon-set-definition.yaml

```

apiVersion: apps/v1
kind: DaemonSet
metadata:
 name: monitoring-daemon
spec:
 selector:
 matchLabels:
 app: monitoring-agent
 template:
 metadata:
 labels:
 app: monitoring-agent
 spec:
 containers:
 - name: monitoring-agent
 image: monitoring-agent

```

replicaset-definition.yaml

```

apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: monitoring-daemon
spec:
 selector:
 matchLabels:
 app: monitoring-agent
 template:
 metadata:
 labels:
 app: monitoring-agent
 spec:
 containers:
 - name: monitoring-agent
 image: monitoring-agent

```

kubectl create -f daemon-set-definition.yaml  
daemon-set Created

- To create a daemonset from a definition file
- \$ kubectl create -f daemon-set-definition.yaml

### View DaemonSets

- To list daemonsets
- \$ kubectl get daemonsets

- For more details of the daemonsets
- \$ kubectl describe daemonsets monitoring-daemon

## IView DaemonSets

```
▶ kubectl get daemonsets
```

| NAME              | DESIRED | CURRENT | READY | UP-TO-DATE | AVAILABLE | AGE |
|-------------------|---------|---------|-------|------------|-----------|-----|
| monitoring-daemon | 1       | 1       | 1     | 1          | 1         | 41  |

```
▶ kubectl describe daemonsets monitoring-daemon
```

```
Name: monitoring-daemon
Selector: name=monitoring-daemon
Node-Selector: <none>
Labels: name=monitoring-daemon
Desired Number of Nodes Scheduled: 2
Current Number of Nodes Scheduled: 2
Number of Nodes Scheduled with Up-to-date Pods: 2
Number of Nodes Scheduled with Available Pods: 1
Number of Nodes Misscheduled: 0
Pods Status: 2 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
 Labels: app=monitoring-agent
 Containers:
```

## How DaemonSets Works

### I How does it work?



Default Behavior till v1.12

From v1.12 - uses NodeAffinity and default scheduler



## K8s Reference Docs

- <https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/#writing-a-daemonset-spec>

# Static Pods

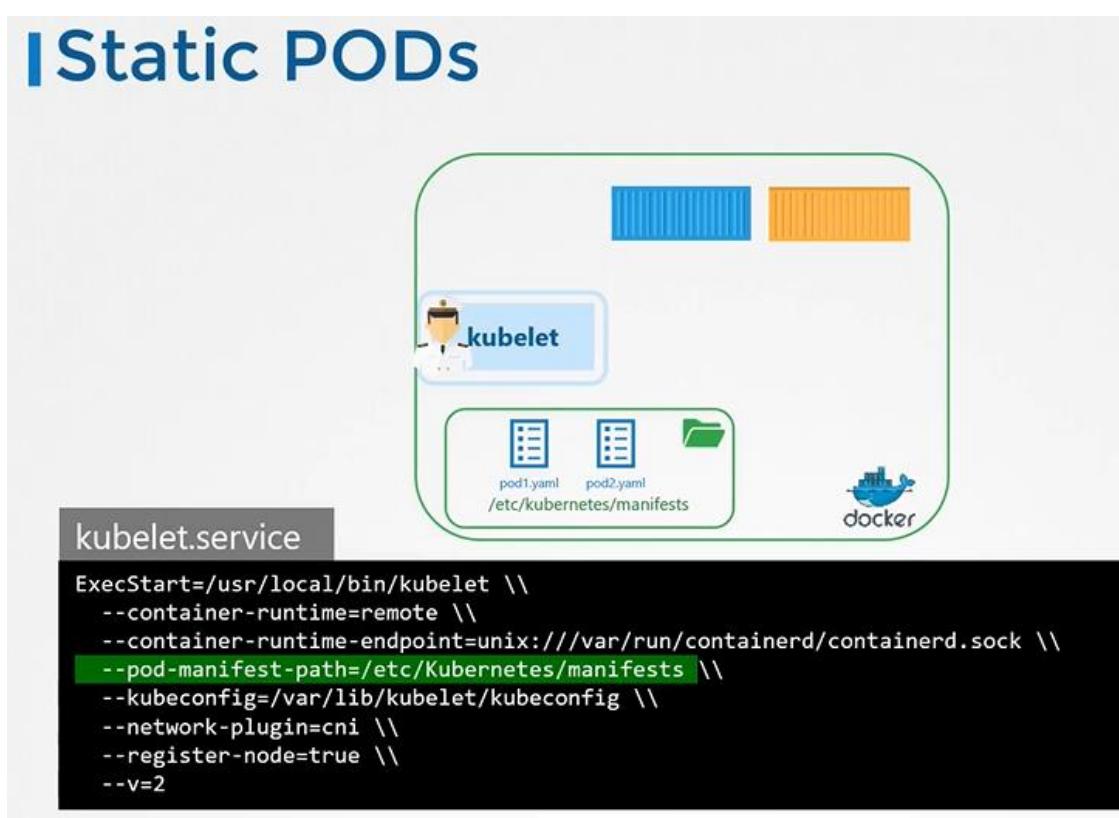
In this section, we will take a look at Static Pods

*How do you provide a pod definition file to the kubelet without a kube-apiserver?*

- You can configure the kubelet to read the pod definition files from a directory on the server designated to store information about pods.

## Configure Static Pod

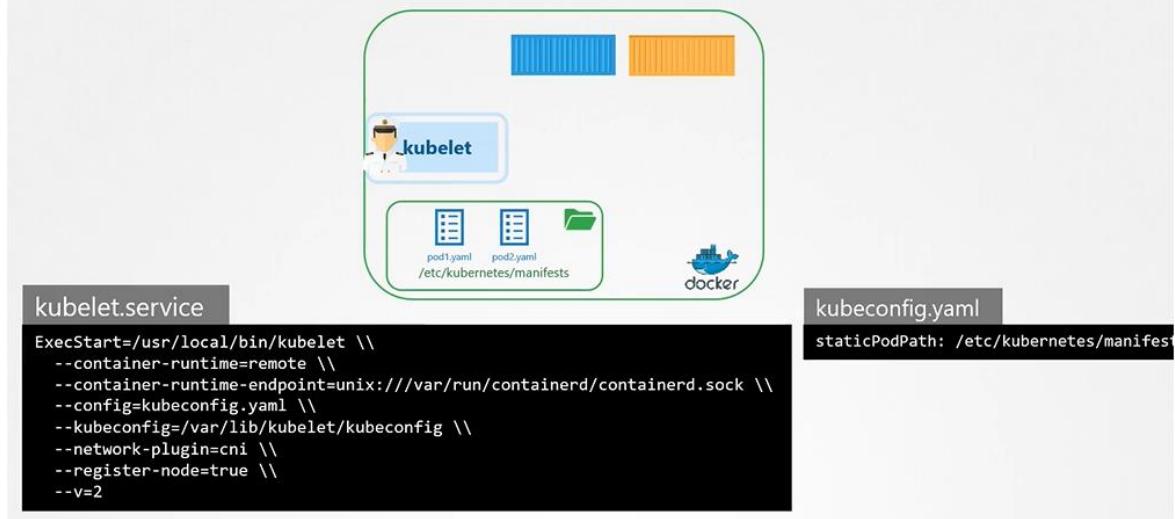
- The designated directory can be any directory on the host and the location of that directory is passed in to the kubelet as an option while running the service.
    - The option is named as `--pod-manifest-path`



Another way to configure static pod

- Instead of specifying the option directly in the `kubelet.service` file, you could provide a path to another config file using the `config` option, and define the directory path as `staticPodPath` in the file.

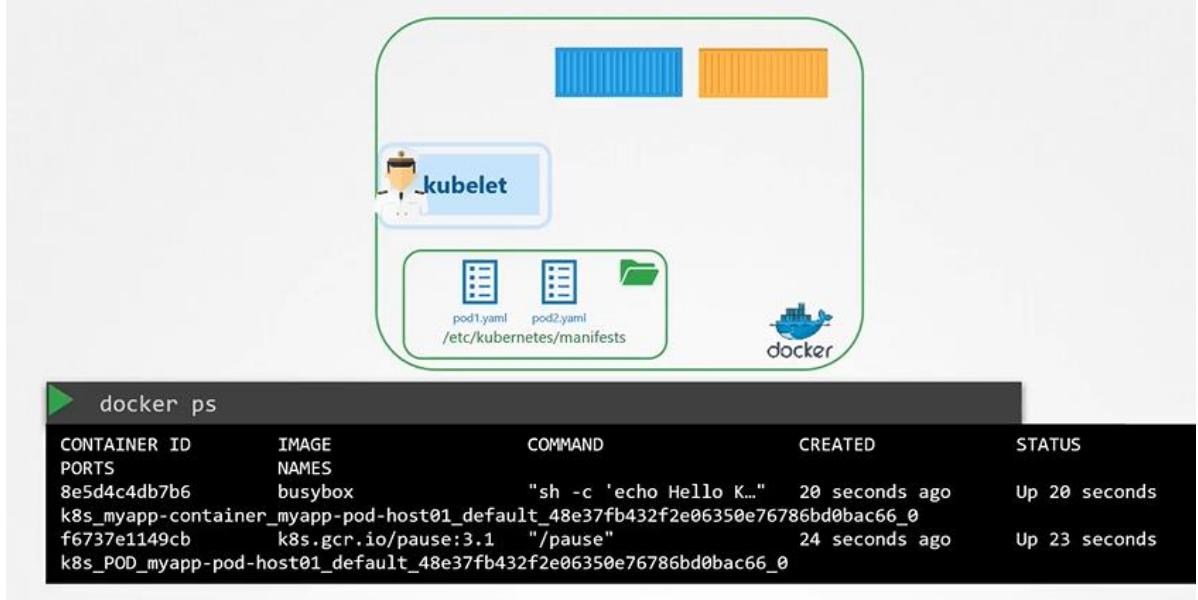
## I Static PODs



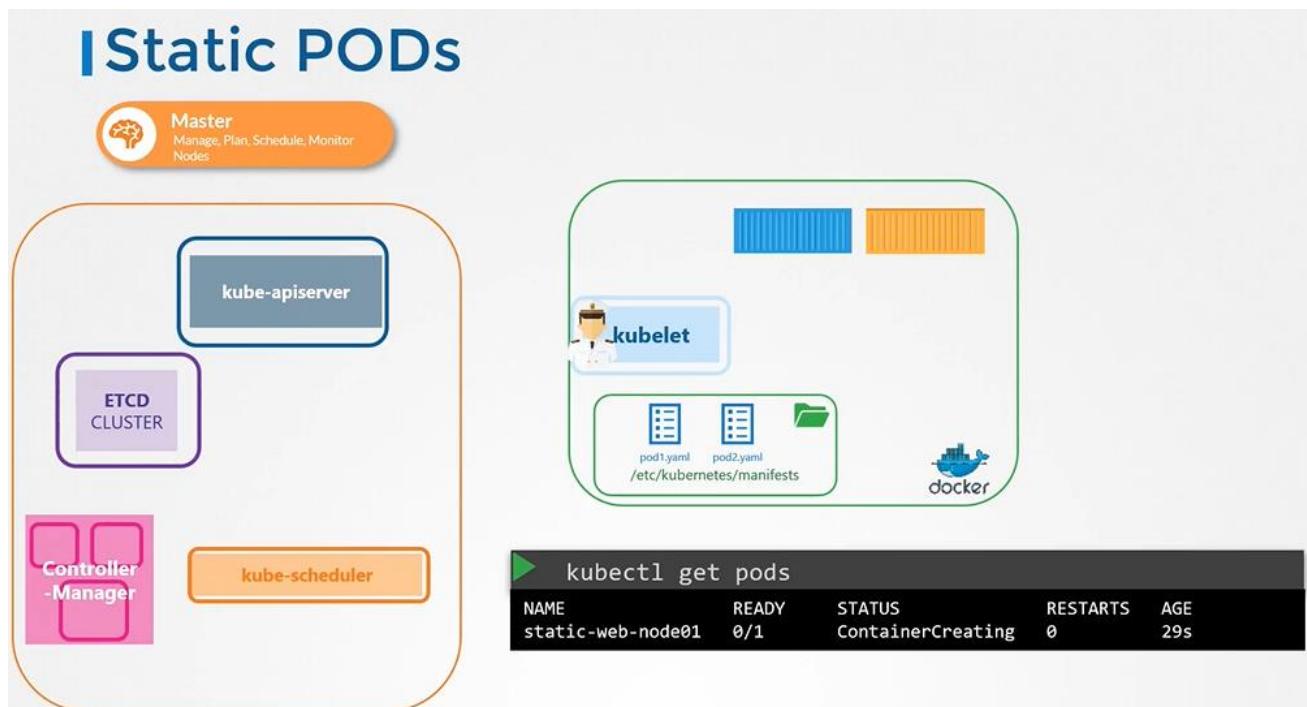
View the static pods

- To view the static pods
- \$ docker ps

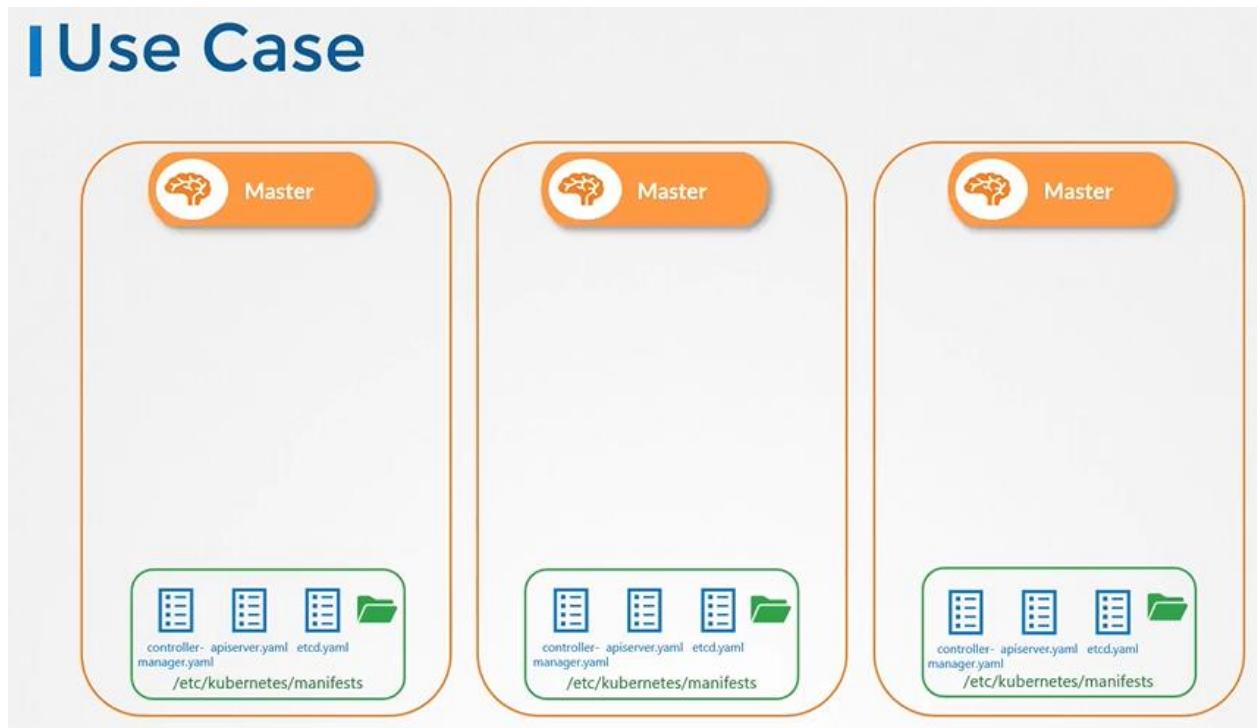
## I Static PODs



The kubelet can create both kinds of pods - the static pods and the ones from the api server at the same time.



### Static Pods - Use Case



## I Use Case

| kubectl get pods -n kube-system |                                |       |         |          |     |
|---------------------------------|--------------------------------|-------|---------|----------|-----|
| NAMESPACE                       | NAME                           | READY | STATUS  | RESTARTS | AGE |
| kube-system                     | coredns-78fcdf6894-hwrq9       | 1/1   | Running | 0        | 16m |
| kube-system                     | coredns-78fcdf6894-rzhjr       | 1/1   | Running | 0        | 16m |
| kube-system                     | etcd-master                    | 1/1   | Running | 0        | 15m |
| kube-system                     | kube-apiserver-master          | 1/1   | Running | 0        | 15m |
| kube-system                     | kube-controller-manager-master | 1/1   | Running | 0        | 15m |
| kube-system                     | kube-proxy-lzt6f               | 1/1   | Running | 0        | 16m |
| kube-system                     | kube-proxy-zm5qd               | 1/1   | Running | 0        | 16m |
| kube-system                     | kube-scheduler-master          | 1/1   | Running | 0        | 15m |
| kube-system                     | weave-net-29z42                | 2/2   | Running | 1        | 16m |
| kube-system                     | weave-net-snmdl                | 2/2   | Running | 1        | 16m |

Static Pods vs DaemonSets

## I Static PODs vs DaemonSets

| Static PODs                                    | DaemonSets                                        |
|------------------------------------------------|---------------------------------------------------|
| Created by the Kubelet                         | Created by Kube-API server (DaemonSet Controller) |
| Deploy Control Plane components as Static Pods | Deploy Monitoring Agents, Logging Agents on nodes |
| Ignored by the Kube-Scheduler                  |                                                   |

K8s Reference Docs

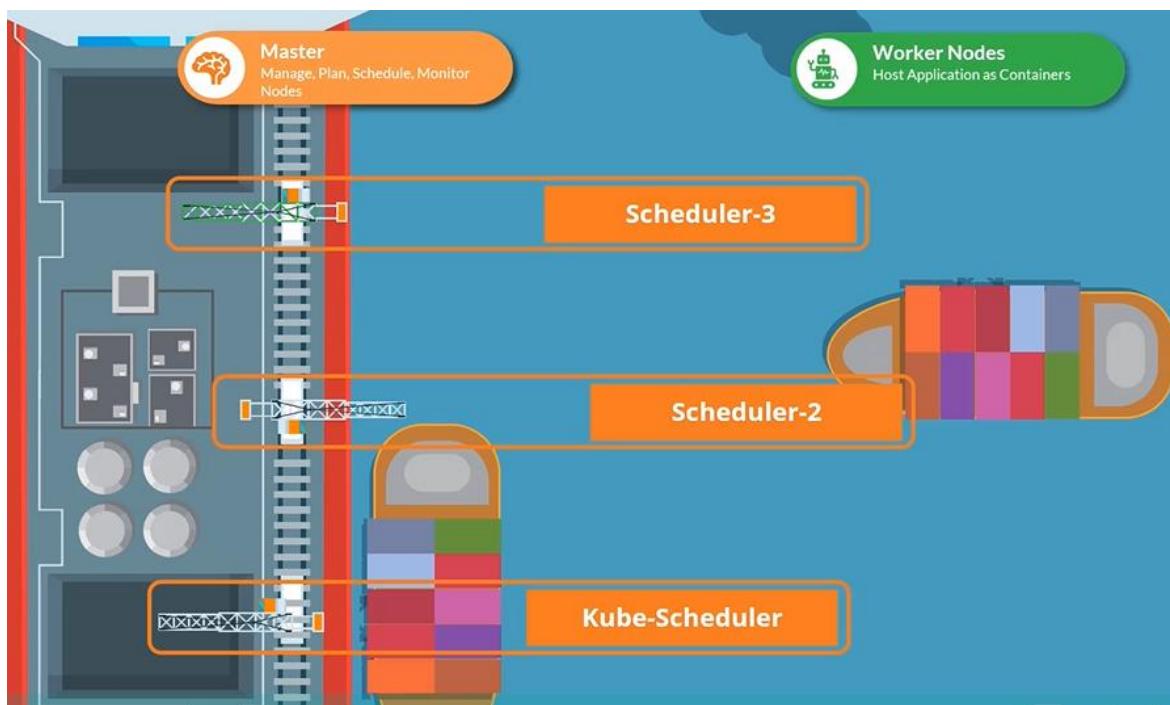
- <https://kubernetes.io/docs/tasks/configure-pod-container/static-pod/>

## Multiple Schedulers

In this section, we will take a look at multiple schedulers

Custom Schedulers

- Your kubernetes cluster can schedule multiple schedulers at the same time.



## Deploy additional scheduler

- Download the binary
- `$ wget https://storage.googleapis.com/kubernetes-release/release/v1.12.0/bin/linux/amd64/kube-scheduler`

## | Deploy Additional Scheduler

```
▶ wget https://storage.googleapis.com/kubernetes-release/release/v1.12.0/bin/linux/amd64/kube-scheduler
```

```
kube-scheduler.service
```

```
ExecStart=/usr/local/bin/kube-scheduler \
--config=/etc/kubernetes/config/kube-scheduler.yaml \
--scheduler-name= default-scheduler
```

```
my-custom-scheduler.service
```

```
ExecStart=/usr/local/bin/kube-scheduler \
--config=/etc/kubernetes/config/kube-scheduler.yaml \
--scheduler-name= my-custom-scheduler
```

Deploy additional scheduler - kubeadm

## | Deploy Additional Scheduler - kubeadm

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

apiVersion: v1
kind: Pod
metadata:
 name: kube-scheduler
 namespace: kube-system
spec:
 containers:
 - command:
 - kube-scheduler
 - --address=127.0.0.1
 - --kubeconfig=/etc/kubernetes/scheduler.conf
 - --leader-elect=true
 image: k8s.gcr.io/kube-scheduler-amd64:v1.11.3
 name: kube-scheduler
```

```
my-custom-scheduler.yaml

apiVersion: v1
kind: Pod
metadata:
 name: my-custom-scheduler
 namespace: kube-system
spec:
 containers:
 - command:
 - kube-scheduler
 - --address=127.0.0.1
 - --kubeconfig=/etc/kubernetes/scheduler.conf
 - --leader-elect=true
 - --scheduler-name=my-custom-scheduler
 - --lock-object-name=my-custom-scheduler
 image: k8s.gcr.io/kube-scheduler-amd64:v1.11.3
 name: kube-scheduler
```

- To create a scheduler pod
- \$ kubectl create -f my-custom-scheduler.yaml

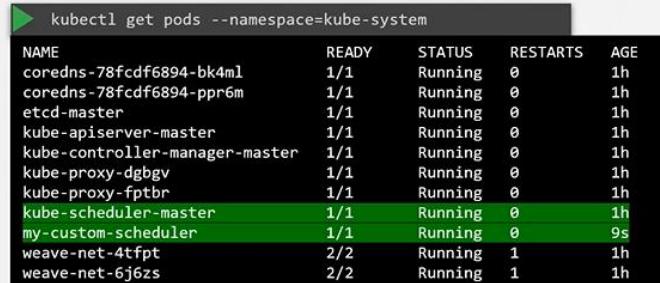
### View Schedulers

- To list the scheduler pods
- \$ kubectl get pods -n kube-system

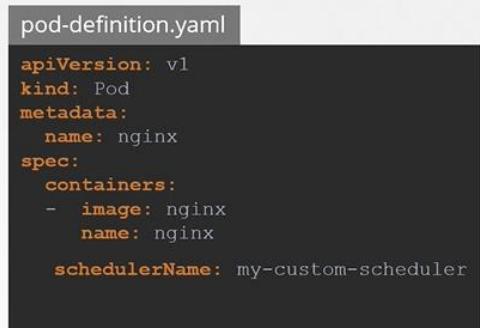
### Use the Custom Scheduler

- Create a pod definition file and add new section called **schedulerName** and specify the name of the new scheduler
- apiVersion: v1
- kind: Pod
- metadata:
- name: nginx
- spec:
- containers:
- - image: nginx
- name: nginx
- schedulerName: my-custom-scheduler

## I Use Custom Scheduler



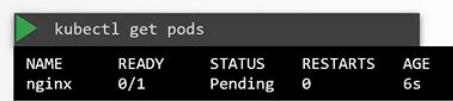
```
kubectl get pods --namespace=kube-system
NAME READY STATUS RESTARTS AGE
coredns-78fcdf6894-bk4ml 1/1 Running 0 1h
coredns-78fcdf6894-ppr6m 1/1 Running 0 1h
etcd-master 1/1 Running 0 1h
kube-apiserver-master 1/1 Running 0 1h
kube-controller-manager-master 1/1 Running 0 1h
kube-proxy-dbgv 1/1 Running 0 1h
kube-proxy-fptbr 1/1 Running 0 1h
kube-scheduler-master 1/1 Running 0 1h
my-custom-scheduler 1/1 Running 0 9s
weave-net-4tfpt 2/2 Running 1 1h
weave-net-6j6zs 2/2 Running 1 1h
```



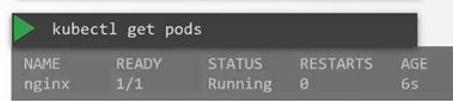
```
apiVersion: v1
kind: Pod
metadata:
 name: nginx
spec:
 containers:
 - image: nginx
 name: nginx
 schedulerName: my-custom-scheduler
```



```
kubectl create -f pod-definition.yaml
```




```
kubectl get pods
NAME READY STATUS RESTARTS AGE
nginx 0/1 Pending 0 6s
```

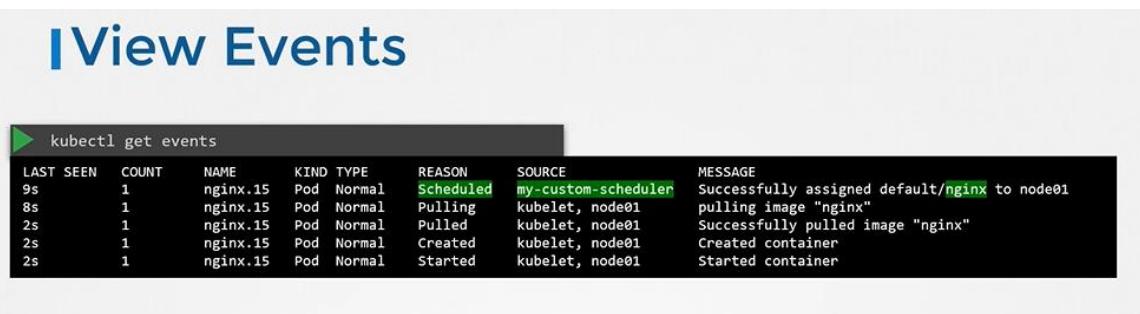



```
kubectl get pods
NAME READY STATUS RESTARTS AGE
nginx 1/1 Running 0 6s
```

- To create a pod definition
- \$ kubectl create -f pod-definition.yaml
- To list pods
- \$ kubectl get pods

## View Events

- To view events
- \$ kubectl get events



```
kubectl get events
LAST SEEN COUNT NAME KIND TYPE REASON SOURCE MESSAGE
9s 1 nginx.15 Pod Normal Scheduled my-custom-scheduler Successfully assigned default/nginx to node01
8s 1 nginx.15 Pod Normal Pulling kubelet, node01 pulling image "nginx"
2s 1 nginx.15 Pod Normal Pulled kubelet, node01 Successfully pulled image "nginx"
2s 1 nginx.15 Pod Normal Created kubelet, node01 Created container
2s 1 nginx.15 Pod Normal Started kubelet, node01 Started container
```

## View Scheduler Logs

- To view scheduler logs
- \$ kubectl logs my-custom-scheduler -n kube-system

## I View Scheduler Logs

```
kubectl logs my-custom-scheduler --name-space=kube-system
I0204 09:42:25.819338 1 server.go:126] Version: v1.11.3
W0204 09:42:25.822720 1 authorization.go:47] Authorization is disabled
W0204 09:42:25.822745 1 authentication.go:55] Authentication is disabled
I0204 09:42:25.822801 1 insecure_serving.go:47] Serving healthz insecurely on 127.0.0.1:10251
I0204 09:45:14.725407 1 controller_utils.go:1025] Waiting for caches to sync for scheduler controller
I0204 09:45:14.825634 1 controller_utils.go:1032] Caches are synced for scheduler controller
I0204 09:45:14.825814 1 leaderelection.go:185] attempting to acquire leader lease kube-system/my-custom-scheduler...
I0204 09:45:14.834953 1 leaderelection.go:194] successfully acquired lease kube-system/my-custom-scheduler...
```

### K8s Reference Docs

- <https://kubernetes.io/docs/tasks/extend-kubernetes/configure-multiple-schedulers/>

## Configuring Kubernetes Schedulers

In this section, we will take a look at configuring kubernetes schedulers.

## I Deploy Additional Scheduler - kubeadm

```
/etc/kubernetes/manifests/kube-scheduler.yaml
apiVersion: v1
kind: Pod
metadata:
 name: kube-scheduler
 namespace: kube-system
spec:
 containers:
 - command:
 - kube-scheduler
 - --address=127.0.0.1
 - --kubeconfig=/etc/kubernetes/scheduler.conf
 - --leader-elect=true
 image: k8s.gcr.io/kube-scheduler-amd64:v1.11.3
 name: kube-scheduler
```

```
my-custom-scheduler.yaml
apiVersion: v1
kind: Pod
metadata:
 name: my-custom-scheduler
 namespace: kube-system
spec:
 containers:
 - command:
 - kube-scheduler
 - --address=127.0.0.1
 - --kubeconfig=/etc/kubernetes/scheduler.conf
 - --leader-elect=true
 - --scheduler-name=my-custom-scheduler
 - --lock-object-name=my-custom-scheduler
 image: k8s.gcr.io/kube-scheduler-amd64:v1.11.3
 name: kube-scheduler
```

## References

- <https://kubernetes.io/blog/2017/03/advanced-scheduling-in-kubernetes/>
- <https://jvns.ca/blog/2017/07/27/how-does-the-kubernetes-scheduler-work/>
- <https://stackoverflow.com/questions/28857993/how-does-kubernetes-scheduler-work>

# Logging and Monitoring Section

## Introduction

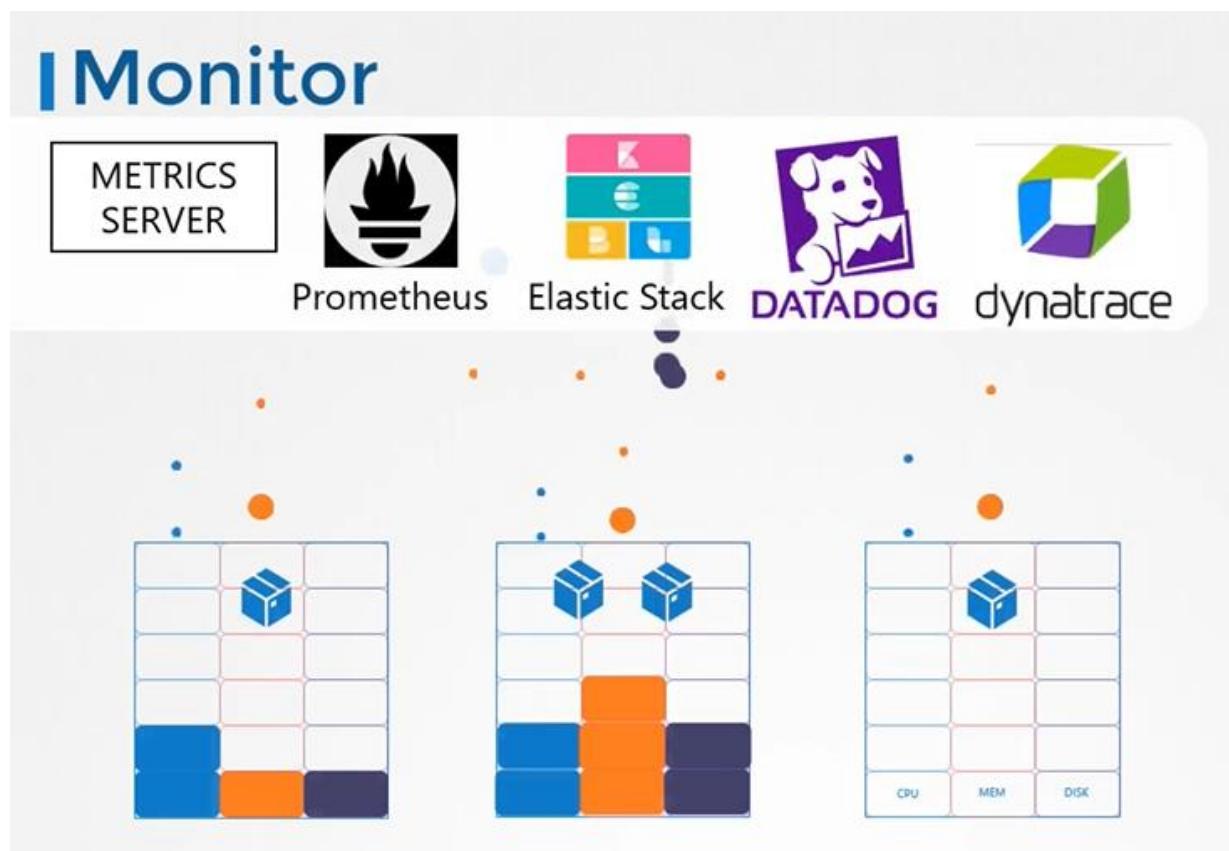
In this section, we will take a look at the below

- Monitor Cluster Components
- Monitor Applications
- Monitor Cluster Components Logs
- Application Logs

## Monitor Cluster Components

In this section, we will take a look at monitoring kubernetes cluster

*How do you monitor resource consumption in kubernetes? or more importantly, what would you like to monitor?*



## Heapster vs Metrics Server

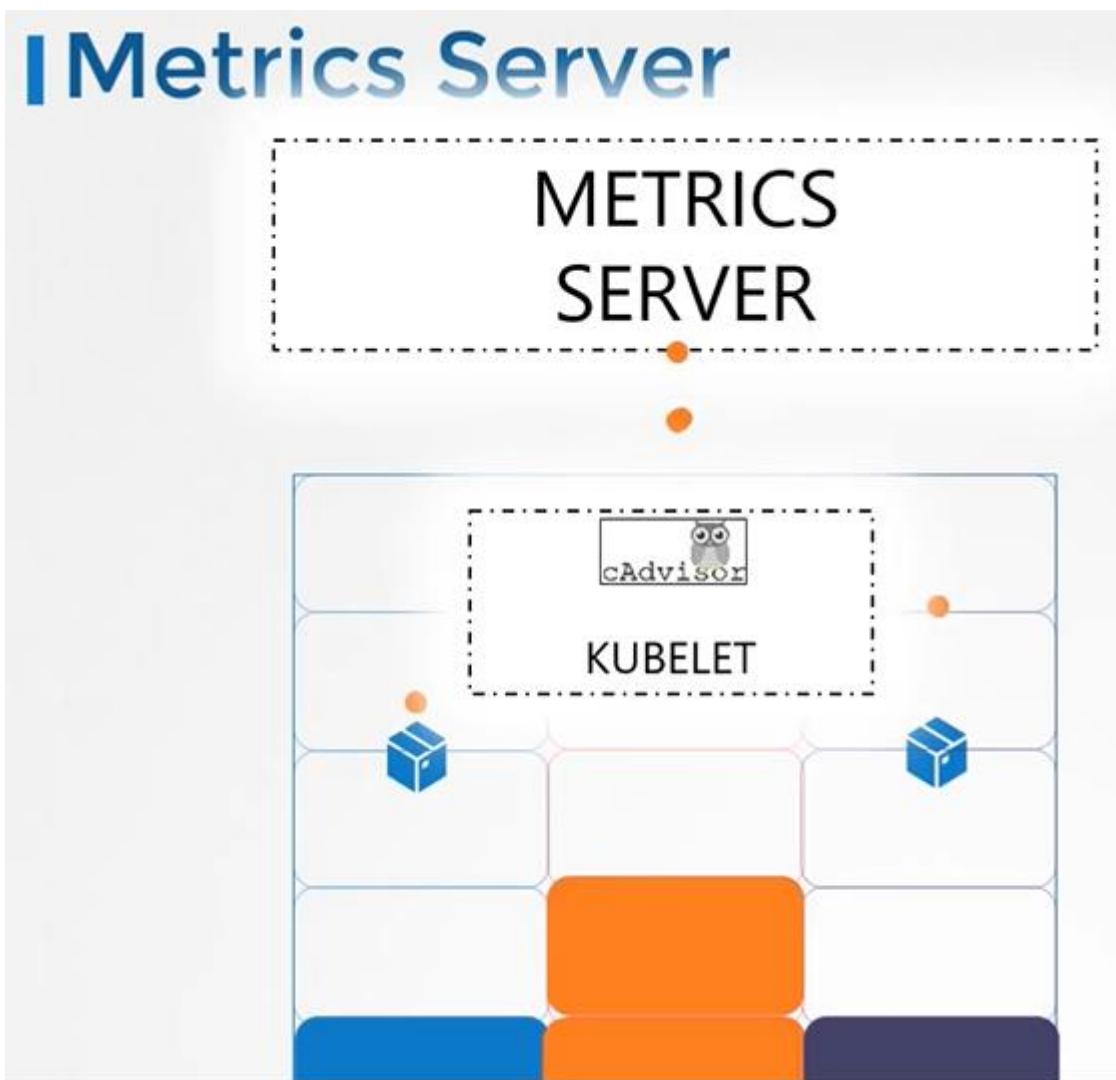
- Heapster is now deprecated and a slimmed down version was formed known as the `metrics server`.



## Metrics Server



How are the metrics generated for the PODs on these nodes?



## Metrics Server - Getting Started

# Metrics Server - Getting Started

The screenshot shows a terminal window with several command-line inputs:

- A blue hexagonal icon with a white Kubernetes logo next to the text "minikube".
- A command: `minikube addons enable metrics-server`
- A section labeled "others" containing:
  - A command: `git clone https://github.com/kubernetes-incubator/metrics-server.git`
  - A command: `kubectl create -f deploy/1.8+/-` followed by its output:

```
clusterrolebinding "metrics-server:system:auth-delegator" created
rolebinding "metrics-server-auth-reader" created
apiservice "v1beta1.metrics.k8s.io" created
serviceaccount "metrics-server" created
deployment "metrics-server" created
service "metrics-server" created
clusterrole "system:metrics-server" created
clusterrolebinding "system:metrics-server" created
```

- Clone the metric server from github repo
- `$ git clone https://github.com/kubernetes-incubator/metrics-server.git`
- Deploy the metric server
- `$ kubectl create -f metric-server/deploy/1.8+/-`
- View the cluster performance
- `$ kubectl top node`
- View performance metrics of pod
- `$ kubectl top pod`

# IView

```
▶ kubectl top node
```

| NAME       | CPU(cores) | CPU% | MEMORY(bytes) | MEMORY% |
|------------|------------|------|---------------|---------|
| kubemaster | 166m       | 8%   | 1337Mi        | 70%     |
| kubenode1  | 36m        | 1%   | 1046Mi        | 55%     |
| kubenode2  | 39m        | 1%   | 1048Mi        | 55%     |

```
▶ kubectl top pod
```

| NAME  | CPU(cores) | CPU% | MEMORY(bytes) | MEMORY% |
|-------|------------|------|---------------|---------|
| nginx | 166m       | 8%   | 1337Mi        | 70%     |
| redis | 36m        | 1%   | 1046Mi        | 55%     |

## Managing Application Logs

In this section, we will take a look at managing application logs

Let us start with logging in docker

# I Logs - Docker

```
▶ docker run kodekloud/event-simulator

2018-10-06 15:57:15,937 - root - INFO - USER1 logged in
2018-10-06 15:57:16,943 - root - INFO - USER2 logged out
2018-10-06 15:57:17,944 - root - INFO - USER2 is viewing page2
2018-10-06 15:57:18,951 - root - INFO - USER3 is viewing page3
2018-10-06 15:57:19,954 - root - INFO - USER4 is viewing page1
2018-10-06 15:57:20,955 - root - INFO - USER2 logged out
2018-10-06 15:57:21,956 - root - INFO - USER1 logged in
2018-10-06 15:57:22,957 - root - INFO - USER3 is viewing page2
2018-10-06 15:57:23,959 - root - INFO - USER1 logged out
2018-10-06 15:57:24,959 - root - INFO - USER2 is viewing page2
2018-10-06 15:57:25,961 - root - INFO - USER1 logged in
2018-10-06 15:57:26,965 - root - INFO - USER4 is viewing page3
2018-10-06 15:57:27,965 - root - INFO - USER4 is viewing page3
2018-10-06 15:57:28,967 - root - INFO - USER2 is viewing page1
2018-10-06 15:57:29,967 - root - INFO - USER3 logged out
2018-10-06 15:57:30,972 - root - INFO - USER1 is viewing page2
2018-10-06 15:57:31,972 - root - INFO - USER4 logged out
2018-10-06 15:57:32,973 - root - INFO - USER1 logged in
2018-10-06 15:57:33,974 - root - INFO - USER1 is viewing page3
```

# I Logs - Docker

```
▶ docker run -d kodekloud/event-simulator
```

```
▶ docker logs -f ecf
```

```
2018-10-06 15:57:15,937 - root - INFO - USER1 logged in
2018-10-06 15:57:16,943 - root - INFO - USER2 logged out
2018-10-06 15:57:17,944 - root - INFO - USER2 is viewing page2
2018-10-06 15:57:18,951 - root - INFO - USER3 is viewing page3
2018-10-06 15:57:19,954 - root - INFO - USER4 is viewing page1
2018-10-06 15:57:20,955 - root - INFO - USER2 logged out
2018-10-06 15:57:21,956 - root - INFO - USER1 logged in
2018-10-06 15:57:22,957 - root - INFO - USER3 is viewing page2
2018-10-06 15:57:23,959 - root - INFO - USER1 logged out
2018-10-06 15:57:24,959 - root - INFO - USER2 is viewing page2
2018-10-06 15:57:25,961 - root - INFO - USER1 logged in
2018-10-06 15:57:26,965 - root - INFO - USER4 is viewing page3
2018-10-06 15:57:27,965 - root - INFO - USER4 is viewing page3
2018-10-06 15:57:28,967 - root - INFO - USER2 is viewing page1
```

## Logs - Kubernetes

```
apiVersion: v1
kind: Pod
metadata:
 name: event-simulator-pod
spec:
 containers:
 - name: event-simulator
 image: kodekloud/event-simulator
```

# I Logs - Kubernetes

```
▶ kubectl create -f event-simulator.yaml
```

```
▶ kubectl logs -f event-simulator-pod
```

```
2018-10-06 15:57:15,937 - root - INFO - USER1 logged in
2018-10-06 15:57:16,943 - root - INFO - USER2 logged out
2018-10-06 15:57:17,944 - root - INFO - USER2 is viewing page2
2018-10-06 15:57:18,951 - root - INFO - USER3 is viewing page3
2018-10-06 15:57:19,954 - root - INFO - USER4 is viewing page1
```

event-simulator.yaml

```
apiVersion: v1
kind: Pod
metadata:
 name: event-simulator-pod
spec:
 containers:
 - name: event-simulator
 image: kodekloud/event-simulator
```

- To view the logs
- \$ kubectl logs -f event-simulator-pod

- If there are multiple containers in a pod then you must specify the name of the container explicitly in the command.
- `$ kubectl logs -f <pod-name> <container-name>`
- `$ kubectl logs -f even-simulator-pod event-simulator`

## Logs - Kubernetes

```
▶ kubectl logs -f event-simulator-pod event-simulator
2018-10-06 15:57:15,937 - root - INFO - USER1 logged in
2018-10-06 15:57:16,943 - root - INFO - USER2 logged out
2018-10-06 15:57:17,944 - root - INFO - USER2 is viewing page2
2018-10-06 15:57:18,951 - root - INFO - USER3 is viewing page3
2018-10-06 15:57:19,954 - root - INFO - USER4 is viewing page1
2018-10-06 15:57:20,955 - root - INFO - USER2 logged out
2018-10-06 15:57:21,956 - root - INFO - USER1 logged in
2018-10-06 15:57:22,957 - root - INFO - USER3 is viewing page2
```

### event-simulator.yaml

```
apiVersion: v1
kind: Pod
metadata:
 name: event-simulator-pod
spec:
 containers:
 - name: event-simulator
 image: kodekloud/event-simulator
 - name: image-processor
 image: some-image-processor
```

### K8s Reference Docs

- <https://kubernetes.io/blog/2015/06/cluster-level-logging-with-kubernetes/>

## Application Lifecycle Management Section Introduction

In this section, we will take a look at application lifecycle management

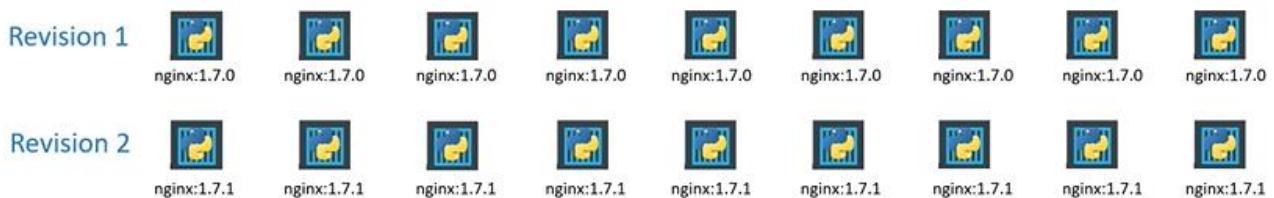
- Rolling Updates and Rollbacks in Deployments
- Configure Applications
- Scale Applications
- Self-Healing Application

## Rolling Updates and Rollback

In this section, we will take a look at rolling updates and rollback in a deployment

## Rollout and Versioning in a Deployment

# Rollout and Versioning



## Rollout commands

- You can see the status of the rollout by the below command
- `$ kubectl rollout status deployment/myapp-deployment`
- To see the history and revisions
- `$ kubectl rollout history deployment/myapp-deployment`

# Rollout Command

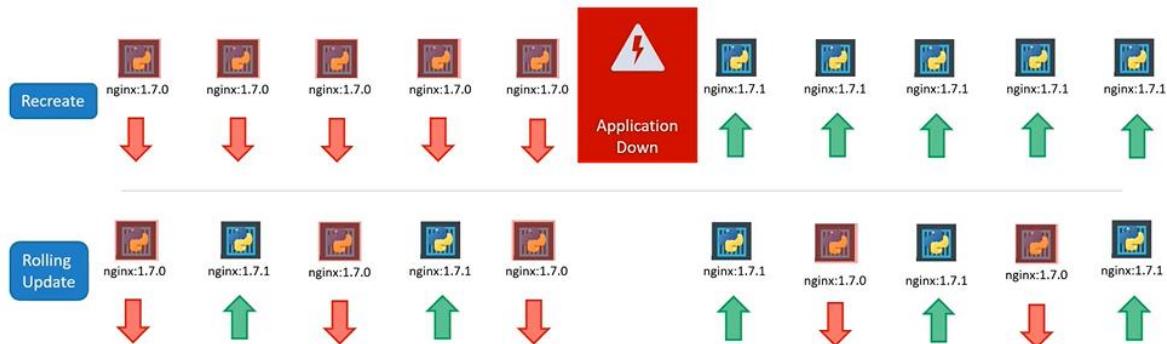
```
> kubectl rollout status deployment/myapp-deployment
Waiting for rollout to finish: 0 of 10 updated replicas are available...
Waiting for rollout to finish: 1 of 10 updated replicas are available...
Waiting for rollout to finish: 2 of 10 updated replicas are available...
Waiting for rollout to finish: 3 of 10 updated replicas are available...
Waiting for rollout to finish: 4 of 10 updated replicas are available...
Waiting for rollout to finish: 5 of 10 updated replicas are available...
Waiting for rollout to finish: 6 of 10 updated replicas are available...
Waiting for rollout to finish: 7 of 10 updated replicas are available...
Waiting for rollout to finish: 8 of 10 updated replicas are available...
Waiting for rollout to finish: 9 of 10 updated replicas are available...
deployment "myapp-deployment" successfully rolled out
```

```
> kubectl rollout history deployment/myapp-deployment
deployments "myapp-deployment"
REVISION CHANGE-CAUSE
1 <none>
2 kubectl apply --filename=deployment-definition.yml --record=true
```

## Deployment Strategies

- There are 2 types of deployment strategies
  - i. Recreate
  - ii. RollingUpdate (Default Strategy)

## Deployment Strategy



## kubectl apply

- To update a deployment, edit the deployment and make necessary changes and save it. Then run the below command.
- `apiVersion: apps/v1`
- `kind: Deployment`
- `metadata:`
- `name: myapp-deployment`
- `labels:`
- `app: nginx`
- `spec:`
- `template:`
- `metadata:`
- `name: myap-pod`
- `labels:`
- `app: myapp`
- `type: front-end`
- `spec:`
- `containers:`
- `- name: nginx-container`
- `image: nginx:1.7.1`
- `replicas: 3`
- `selector:`
- `matchLabels:`
- `type: front-end`
- `$ kubectl apply -f deployment-definition.yaml`
- Alternate way to update a deployment say for example for updating an image.
- `$ kubectl set image deployment/myapp-deployment nginx=nginx:1.9.1`

## Kubectl apply

```
> kubectl apply -f deployment-definition.yaml
deployment "myapp-deployment" configured
```

```
> kubectl set image deployment/myapp-deployment \
 nginx=nginx:1.9.1
deployment "myapp-deployment" image is updated
```

```
deployment-definition.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: myapp-deployment
 labels:
 app: myapp
 type: front-end
spec:
 template:
 metadata:
 name: myapp-pod
 labels:
 app: myapp
 type: front-end
 spec:
 containers:
 - name: nginx-container
 image: nginx:1.7.1
replicas: 3
selector:
 matchLabels:
 type: front-end
```

## Recreate vs RollingUpdate

```

:::Kubernetes>kubectl describe deployment myapp-deployment
Name: myapp-deployment
Namespace: default
CreationTimestamp: Sat, 03 Mar 2018 17:01:55 +0800
Labels: app=app
Annotations: deployment.kubernetes.io/revision=2
 kubectl.kubernetes.io/last-updated-configuration={"apiVersion":"apps/v1","kind":"Deployment","metadata":{"name":"myapp-deployment","namespace":"default","selfLink":"/apis/apps/v1/namespaces/default/deployments/myapp-deployment","uid":"54c7d6ccc","resourceVersion":1,"creationTimestamp":"2018-03-03T09:01:55Z","labels":{"app":"app"}, "annotations":{}}, "spec":{"replicas":1, "selector":{"matchLabels":{"app":"app"}}, "template":{"metadata":{"labels":{"app":"app"}}, "spec":{"containers":[{"name": "nginx", "image": "nginx:1.7.1", "ports": [{"port": 80}], "resources": {}, "volumeMounts": [{"name": "nginx-etc", "mountPath": "/etc/nginx"}, {"name": "nginx-var", "mountPath": "/var/www/html"}], "env": [{"name": "NGINX_ENVIRONMENT", "value": "prod"}, {"name": "NGINX_MNT_PATH", "value": "/var/www/html"}], "livenessProbe": {"httpGet": {"path": "/"}, "initialDelaySeconds": 30, "periodSeconds": 10}, "readinessProbe": {"httpGet": {"path": "/"}, "initialDelaySeconds": 30, "periodSeconds": 10}, "terminationProbe": {"httpGet": {"path": "/"}, "initialDelaySeconds": 30, "periodSeconds": 10}, "imagePullPolicy": "IfNotPresent"}]}, "replicas": 1, "selector": {"matchLabels": {"app": "app"}}, "strategyType": "Recreate", "minReadySeconds": 0, "podTemplate": {"labels": {"app": "app"}, "spec": {"containers": [{"name": "nginx", "image": "nginx:1.7.1", "ports": [{"port": 80}], "resources": {}, "volumeMounts": [{"name": "nginx-etc", "mountPath": "/etc/nginx"}, {"name": "nginx-var", "mountPath": "/var/www/html"}], "env": [{"name": "NGINX_ENVIRONMENT", "value": "prod"}, {"name": "NGINX_MNT_PATH", "value": "/var/www/html"}], "livenessProbe": {"httpGet": {"path": "/"}, "initialDelaySeconds": 30, "periodSeconds": 10}, "readinessProbe": {"httpGet": {"path": "/"}, "initialDelaySeconds": 30, "periodSeconds": 10}, "terminationProbe": {"httpGet": {"path": "/"}, "initialDelaySeconds": 30, "periodSeconds": 10}, "imagePullPolicy": "IfNotPresent"}]}}, "conditions": [{"type": "Available", "status": "True", "reason": "MinimumReplicasAvailable"}, {"type": "Progressing", "status": "True", "reason": "NewReplicaSetAvailable"}], "oldReplicaSets": null, "newReplicaSet": "myapp-deployment-54c7d6ccc (5/5 replicas created)"}
Events:
 Type Reason Age From Message
 ---- ---- -- -- --
 Normal ScalingReplicaSet 1m deployment-controller Scaled up replica set myapp-deployment-6795844b58 to 5
 Normal ScalingReplicaSet 1s deployment-controller Scaled down replica set myapp-deployment-6795844b58 to 0
 Normal ScalingReplicaSet 56s deployment-controller Scaled up replica set myapp-deployment-54c7d6ccc to 5

```

```

$ (Kubernetes) kubectl describe deployment myapp-deployment
Name: myapp-deployment
Namespace: default
CreationTimestamp: Sat, 03 Mar 2018 17:16:53 +0800
Labels: app=myapp
 type-front-end
Annotations: deployment.kubernetes.io/revision=2
 kubelet.kubernetes.io/last-applied-configuration={"apiVersion":"apps/v1","kind":"Deployment","metadata":{},"spec":{"selector":{"matchLabels":{"app":"myapp","type":"front-end"}}, "replicas":1, "template":{"metadata":{"labels":{"app":"myapp","type":"front-end"}}, "spec":{"containers":[{"name":"nginx-container", "image": "nginx:latest", "ports": [{"containerPort": 80}], "resources": {"limits": {"cpu": "1", "memory": "128Mi"}, "requests": {"cpu": "1", "memory": "128Mi"}}, "livenessProbe": {"httpGet": {"path": "/"}, "initialDelaySeconds": 3, "periodSeconds": 10, "timeoutSeconds": 5, "failureThreshold": 3}, "readinessProbe": {"httpGet": {"path": "/"}, "initialDelaySeconds": 3, "periodSeconds": 10, "timeoutSeconds": 5, "failureThreshold": 3}, "restartPolicy": "Always"}]}, "strategy": {"type": "RollingUpdate", "rollingUpdate": {"maxSurge": "25%", "maxUnavailable": "25%"}}, "minReadySeconds": 0}, "status": {"availableReplicas": 1, "desiredReplicas": 1, "fullyLabeledReplicas": 1, "observedGeneration": 2, "readyReplicas": 1, "replicas": 1, "unavailableReplicas": 0}, "events": [{"type": "Normal", "reason": "ScalingReplicaSet", "age": "1m", "from": "deployment-controller", "message": "Scaled up replica set myapp-deployment-67c749c58c by 1"}, {"type": "Normal", "reason": "ScalingReplicaSet", "age": "1m", "from": "deployment-controller", "message": "Scaled up replica set myapp-deployment-7d57dbd8d0 by 2"}, {"type": "Normal", "reason": "ScalingReplicaSet", "age": "1m", "from": "deployment-controller", "message": "Scaled down replica set myapp-deployment-67c749c58c by 1 to 0"}, {"type": "Normal", "reason": "ScalingReplicaSet", "age": "1m", "from": "deployment-controller", "message": "Scaled up replica set myapp-deployment-7d57dbd8d0 by 3"}, {"type": "Normal", "reason": "ScalingReplicaSet", "age": "1m", "from": "deployment-controller", "message": "Scaled down replica set myapp-deployment-67c749c58c by 2 to 3"}, {"type": "Normal", "reason": "ScalingReplicaSet", "age": "1m", "from": "deployment-controller", "message": "Scaled up replica set myapp-deployment-7d57dbd8d0 by 4"}, {"type": "Normal", "reason": "ScalingReplicaSet", "age": "1m", "from": "deployment-controller", "message": "Scaled down replica set myapp-deployment-67c749c58c by 3 to 4"}, {"type": "Normal", "reason": "ScalingReplicaSet", "age": "1m", "from": "deployment-controller", "message": "Scaled up replica set myapp-deployment-7d57dbd8d0 by 5"}, {"type": "Normal", "reason": "ScalingReplicaSet", "age": "1m", "from": "deployment-controller", "message": "Scaled down replica set myapp-deployment-67c749c58c by 1"}]}

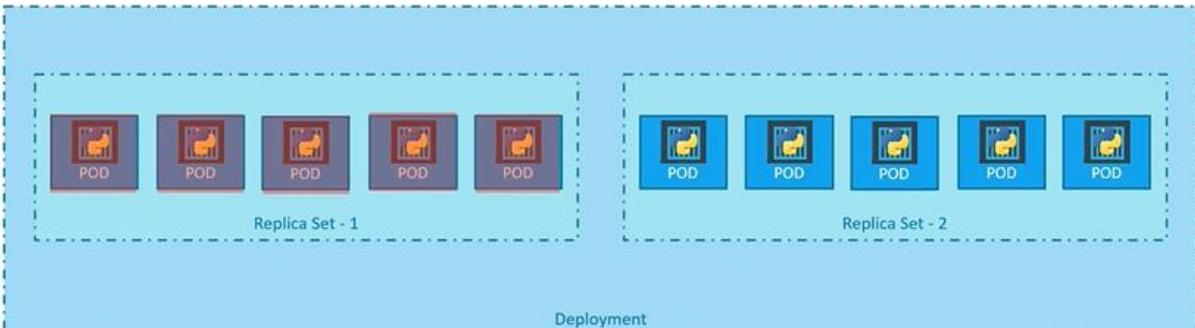
```

## Recreate

## RollingUpdate

## Upgrades

# Upgrades



```
> kubectl get replicaset
```

| NAME                        | DESIRED | CURRENT | READY | AGE |
|-----------------------------|---------|---------|-------|-----|
| myapp-deployment-67c749c58c | 0       | 0       | 0     | 22m |
| myapp-deployment-7d57dbdb8d | 5       | 5       | 5     | 20m |

## Rollback

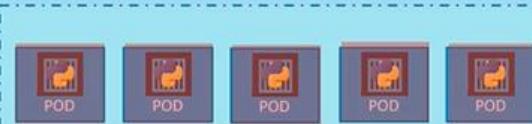
# Rollback

```
> kubectl get replicaset
```

| NAME                        | DESIRED | CURRENT | READY | AGE |
|-----------------------------|---------|---------|-------|-----|
| myapp-deployment-67c749c58c | 0       | 0       | 0     | 22m |
| myapp-deployment-7d57dbdb8d | 5       | 5       | 5     | 20m |

```
> kubectl get replicaset
```

| NAME                        | DESIRED | CURRENT | READY | AGE |
|-----------------------------|---------|---------|-------|-----|
| myapp-deployment-67c749c58c | 5       | 5       | 5     | 22m |
| myapp-deployment-7d57dbdb8d | 0       | 0       | 0     | 20m |



```
> kubectl rollout undo deployment/myapp-deployment
deployment "myapp-deployment" rolled back
```

- To undo a change
- \$ kubectl rollout undo deployment/myapp-deployment

## kubectl create

- To create a deployment

- \$ kubectl create deployment nginx --image=nginx

Summarize kubectl commands

```
$ kubectl create -f deployment-definition.yaml
$ kubectl get deployments
$ kubectl apply -f deployment-definition.yaml
$ kubectl set image deployment/myapp-deployment nginx=nginx:1.9.1
$ kubectl rollout status deployment/myapp-deployment
$ kubectl rollout history deployment/myapp-deployment
$ kubectl rollout undo deployment/myapp-deployment
```

## Summarize Commands

|          |                                                                   |
|----------|-------------------------------------------------------------------|
| Create   | > kubectl create -f deployment-definition.yml                     |
| Get      | > kubectl get deployments                                         |
| Update   | > kubectl apply -f deployment-definition.yaml                     |
|          | > kubectl set image deployment/myapp-deployment nginx=nginx:1.9.1 |
| Status   | > kubectl rollout status deployment/myapp-deployment              |
|          | > kubectl rollout history deployment/myapp-deployment             |
| Rollback | > kubectl rollout undo deployment/myapp-deployment                |

K8s Reference Docs

- <https://kubernetes.io/docs/concepts/workloads/controllers/deployment>
- <https://kubernetes.io/docs/tasks/run-application/run-stateless-application-deployment>

## Commands and Arguments in Docker

In this section, we will take a look at commands and arguments in docker

- To run a docker container
- \$ docker run ubuntu
- To list running containers
- \$ docker ps
- To list all containers including that are stopped

- \$ docker ps -a

```
▶ docker run ubuntu
▶ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
▶ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS EXited (0) 41 seconds ago
45aacca36850 ubuntu "/bin/bash" 43 seconds ago

```

*Unlike virtual machines, containers are not meant to host operating system.*

- Containers are meant to run a specific task or process such as to host an instance of a webserver or application server or a database server etc.



*How do you specify a different command to start the container?*

- One Option is to append a command to the docker run command and that way it overrides the default command specified within the image.
- \$ docker run ubuntu sleep 5
- This way when the container starts it runs the sleep program, waits for 5 seconds and then exists. How do you make that change permanent?

```
FROM Ubuntu
CMD sleep 5
```

- There are different ways of specifying the command either the command simply as is in a shell form or in a JSON array format.

|                           |                    |                 |
|---------------------------|--------------------|-----------------|
| CMD command param1        | CMD sleep 5        |                 |
| CMD ["command", "param1"] | CMD ["sleep", "5"] | CMD ["sleep 5"] |

- Now, build the docker image
- \$ docker build -t ubuntu-sleeper .
- Run docker container
- \$ docker run ubuntu-sleeper



### Entrypoint Instruction

- The entrypoint instruction is like the command instruction as in you can specify the program that will be run when the container starts and whatever you specify on the command line.

*K8s Reference Docs*

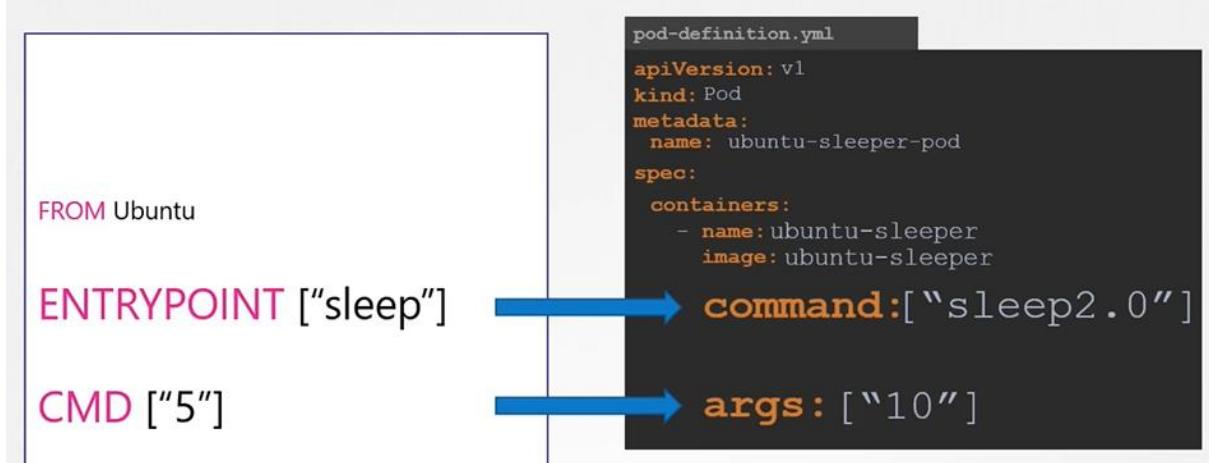
- <https://docs.docker.com/engine/reference/builder/#cmd>

## Commands and Arguments in Kubernetes

In this section, we will take a look at commands and arguments in kubernetes

- Anything that is appended to the docker run command will go into the **args** property of the pod definition file in the form of an array.
- The command field corresponds to the entrypoint instruction in the Dockerfile so to summarize there are 2 fields that correspond to 2 instructions in the Dockerfile.
- apiVersion: v1

- kind: Pod
- metadata:
- name: ubuntu-sleeper-pod
- spec:
- containers:
- - name: ubuntu-sleeper
- image: ubuntu-sleeper
- command: ["sleep2.0"]
- args: ["10"]



*K8s Reference Docs*

- <https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/>

## Configure Environment Variables In Applications

*ENV variables in Docker*

```
$ docker run -e APP_COLOR=pink simple-webapp-color
```

*ENV variables in kubernetes*

- To set an environment variable set an `env` property in pod definition file.
- `apiVersion: v1`
- `kind: Pod`
- `metadata:`
- `name: simple-webapp-color`
- `spec:`
- `containers:`
- - `name: simple-webapp-color`
- `image: simple-webapp-color`
- `ports:`
- - `containerPort: 8080`

- env:
- - name: APP\_COLOR
- value: pink

```
▶ docker run -e APP_COLOR=pink simple-webapp-color
```

pod-definition.yaml

```
apiVersion: v1
kind: Pod
metadata:
 name: simple-webapp-color
spec:
 containers:
 - name: simple-webapp-color
 image: simple-webapp-color
 ports:
 - containerPort: 8080
 env:
 - name: APP COLOR
 value: pink
```

- There are other ways of setting the environment variables such as **ConfigMaps** and **Secrets**

## IENV Value Types

```
env:
 - name: APP_COLOR
 value: pink
```



```
env:
 - name: APP_COLOR
 valueFrom:
 configMapKeyRef:
```



```
env:
 - name: APP_COLOR
 valueFrom:
 secretKeyRef:
```



## K8s Reference Docs

- <https://kubernetes.io/docs/tasks/inject-data-application/define-environment-variable-container/>

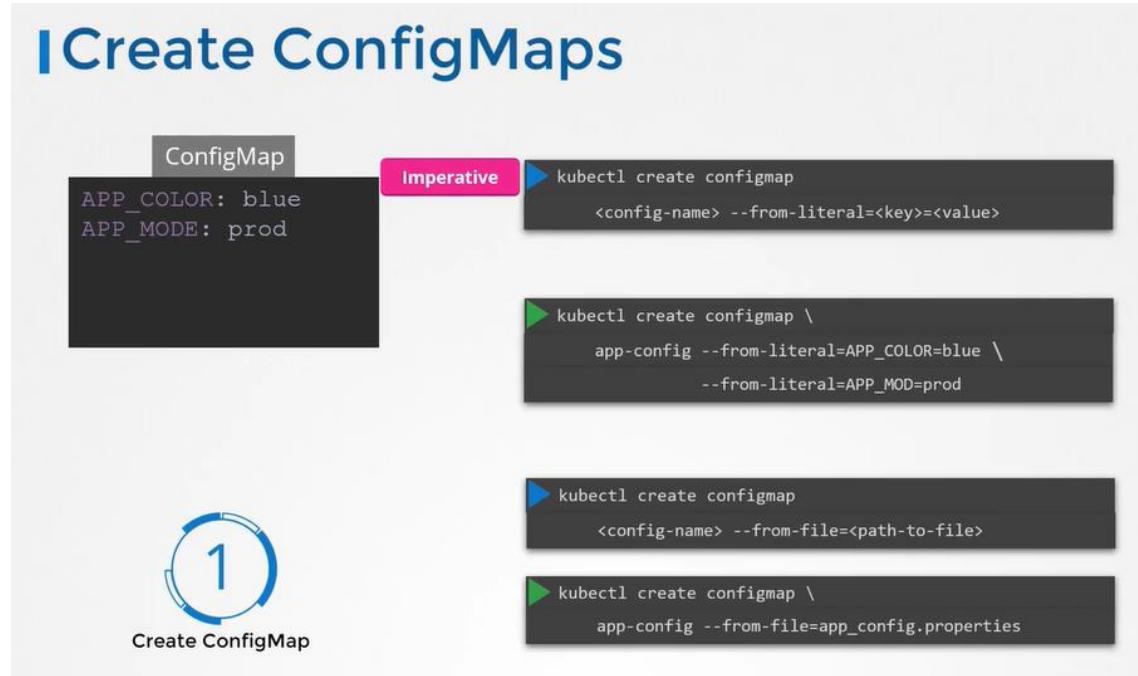
# Configure ConfigMaps in Applications

In this section, we will take a look at configuring configmaps in applications

## ConfigMaps

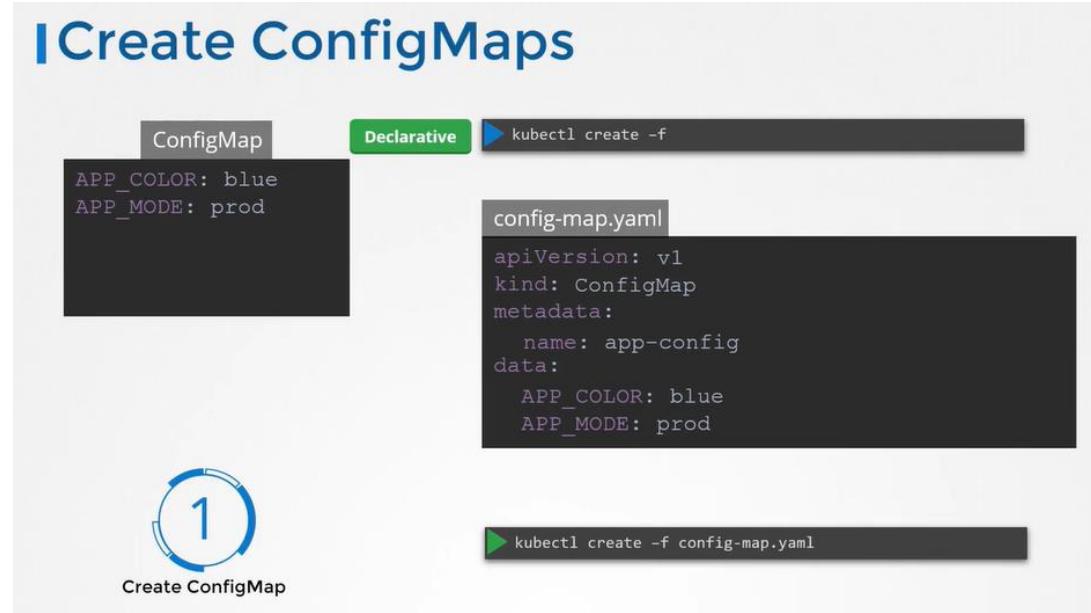
- There are 2 phases involved in configuring ConfigMaps.
  - First, create the configMaps
  - Second, Inject them into the pod.
- There are 2 ways of creating a configmap.
  - The Imperative way
  - ```
$ kubectl create configmap app-config --from-literal=APP_COLOR=blue
--from-literal=APP_MODE=prod
```
 - ```
$ kubectl create configmap app-config --from-file=app_config.properties
```

(Another way)



- The Declarative way
  - ```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
```

- data:
- APP_COLOR: blue
- APP_MODE: prod
- Create a config map definition file and run the 'kubectl create' command to deploy it.
- \$ kubectl create -f config-map.yaml



View ConfigMaps

- To view configMaps
- \$ kubectl get configmaps (or)
- \$ kubectl get cm
- To describe configmaps
- \$ kubectl describe configmaps

IView ConfigMaps

```
▶ kubectl get configmaps
```

NAME	DATA	AGE
app-config	2	3s

```
▶ kubectl describe configmaps
```

```
Name:          app-config
Namespace:    default
Labels:        <none>
Annotations:   <none>

Data
=====
APP_COLOR:
-----
blue
APP_MODE:
-----
prod
Events:  <none>
```

ConfigMap in Pods

- Inject configmap in pod
- apiVersion: v1
- kind: Pod
- metadata:
- name: simple-webapp-color
- spec:
- containers:
- - name: simple-webapp-color
- image: simple-webapp-color
- ports:
- - containerPort: 8080
- envFrom:
- - configMapRef:
- name: app-config
- apiVersion: v1
- kind: ConfigMap
- metadata:
- name: app-config
- data:
- APP_COLOR: blue

- APP_MODE: prod
- \$ kubectl create -f pod-definition.yaml

IConfigMap in Pods

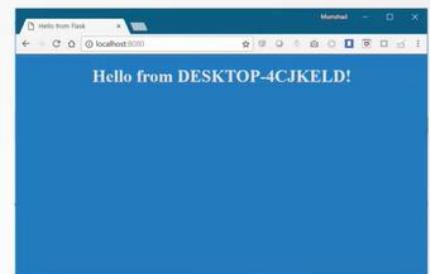
pod-definition.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: simple-webapp-color
  labels:
    name: simple-webapp-color
spec:
  containers:
  - name: simple-webapp-color
    image: simple-webapp-color
    ports:
      - containerPort: 8080
    envFrom:
      - configMapRef:
          name: app-config
```

▶ kubectl create -f pod-definition.yaml

config-map.yaml

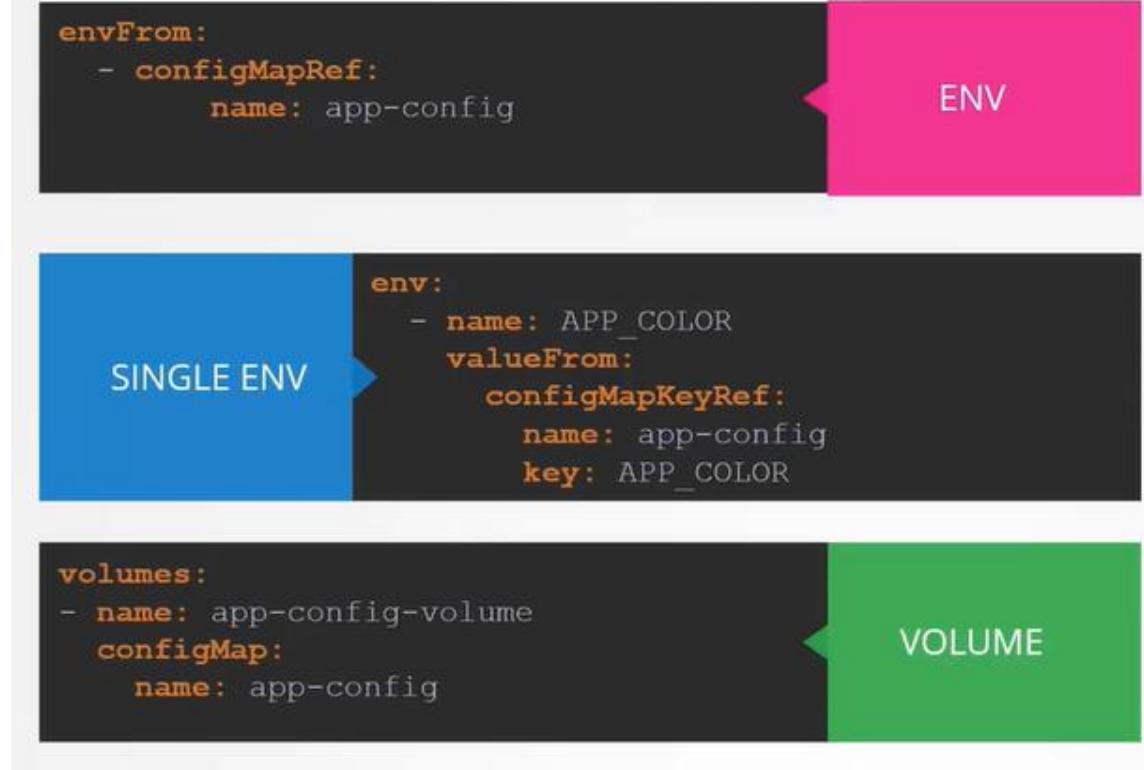
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  APP_COLOR: blue
  APP_MODE: prod
```



There are other ways to inject configuration variables into pod

- You can inject it as a **Single Environment Variable**
- You can inject it as a file in a **Volume**

I ConfigMap in Pods



K8s Reference Docs

- <https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/>
- <https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/#define-container-environment-variables-using-configmap-data>
- <https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/#create-configmaps-from-files>

Secrets

In this section, we will take a look at secrets in kubernetes

Web-Mysql Application

Web-MySQL Application

app.py

```
import os
from flask import Flask

app = Flask(__name__)

@app.route("/")
def main():

    mysql.connector.connect(host='mysql', database='mysql',
                           user='root', password='paswrd')

    return render_template('hello.html', color=fetchcolor())

if __name__ == "__main__":
    app.run(host="0.0.0.0", port="8080")
```

 SUCCESS

From simple-welapp-mysq

- One way is to move the app properties/envs into a configmap. But the configmap stores data into a plain text format. It is definitely not a right place to store a password.
- apiVersion: v1
- kind: ConfigMap
- metadata:
- name: app-config
- data:
 - DB_Host: mysql
 - DB_User: root
 - DB_Password: paswrd

app.py

```
import os
from flask import Flask

app = Flask(__name__)

@app.route("/")
def main():

    mysql.connector.connect(host='mysql', database='mysql',
                           user='root', password='paswrd')

    return render_template('hello.html', color=fetchcolor())

if __name__ == "__main__":
    app.run(host="0.0.0.0", port="8080")
```

config-map.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  DB_Host: mysql
  DB_User: root
  DB_Password: paswrd
```

- Secrets are used to store sensitive information. They are similar to configmaps but they are stored in an encrypted format or a hashed format.

There are 2 steps involved with secrets

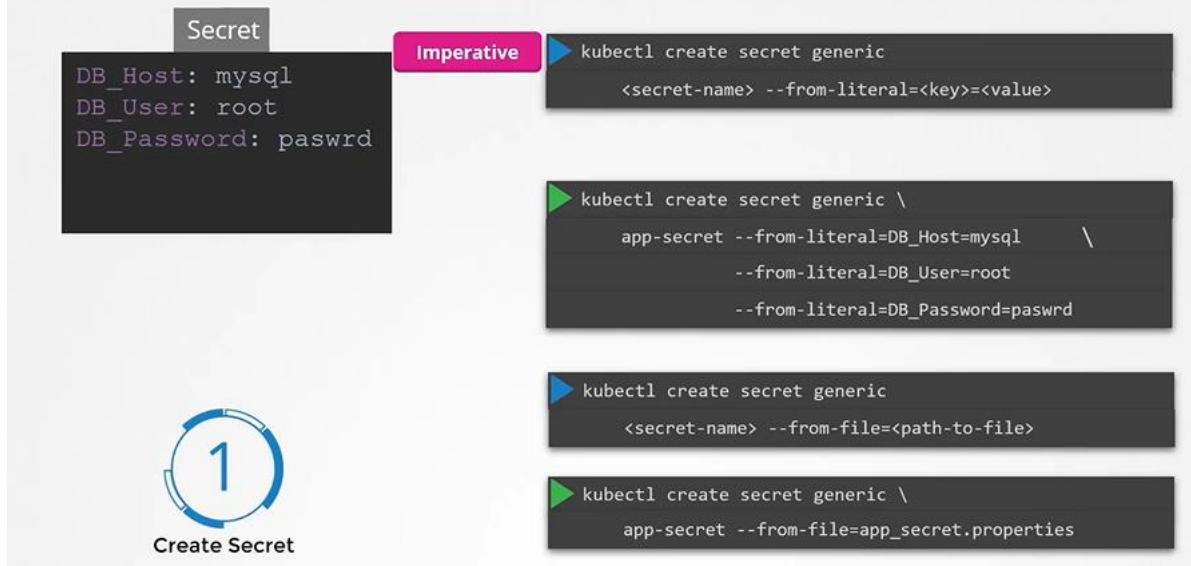
- First, Create a secret
- Second, Inject the secret into a pod.



There are 2 ways of creating a secret

- The Imperative way
- ```
$ kubectl create secret generic app-secret --from-literal=DB_Host=mysql --from-literal=DB_User=root --from-literal=DB_Password=paswrd
```
- ```
$ kubectl create secret generic app-secret --from-file=app_secret.properties
```

Create Secrets

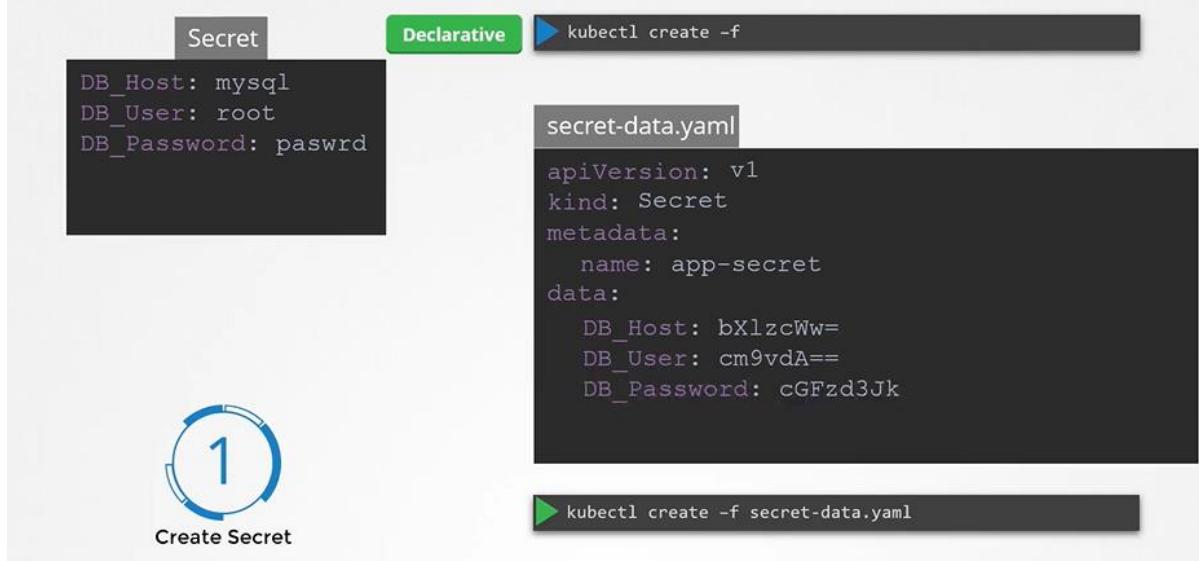


- The Declarative way
- Generate a hash value of the password and pass it to secret-data.yaml definition value as a value to DB_Password variable.
- \$ echo -n "mysql" | base64
- \$ echo -n "root" | base64
- \$ echo -n "passwrd" | base64

Create a secret definition file and run kubectl create to deploy it

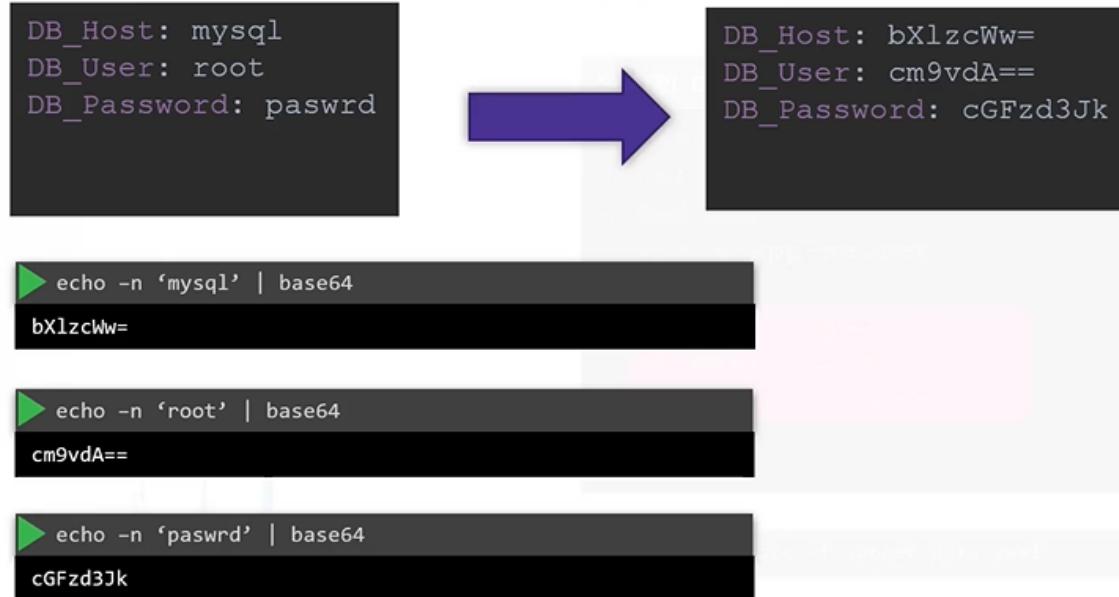
```
apiVersion: v1
kind: Secret
metadata:
  name: app-secret
data:
  DB_Host: bX1zcWw=
  DB_User: cm9vdA==
  DB_Password: cGFzd3Jk
$ kubectl create -f secret-data.yaml
```

I Create Secrets



Encode Secrets

I Encode Secrets



View Secrets

- To view secrets

- \$ kubectl get secrets
- To describe secret
- \$ kubectl describe secret
- To view the values of the secret
- \$ kubectl get secret app-secret -o yaml

IView Secrets

```
kubectl get secrets
NAME          TYPE        DATA   AGE
app-secret    Opaque      3      10m
default-token-mvtkv  kubernetes.io/service-account-token 3      2h

kubectl describe secrets
Name:         app-secret
Namespace:    default
Labels:       <none>
Annotations: <none>

Type:        Opaque

Data
====
DB_Host:     10 bytes
DB_Password: 6 bytes
DB_User:     4 bytes

kubectl get secret app-secret -o yaml
apiVersion: v1
data:
  DB_Host: bX1zcWw=
  DB_Password: cGFzd3Jk
  DB_User: cm9vdA==
kind: Secret
metadata:
  creationTimestamp: 2018-10-18T10:01:12Z
  labels:
    name: app-secret
    name: app-secret
    namespace: default
  uid: be96e989-d2bc-11e8-a545-080027931072
type: Opaque
```

Decode Secrets

- To decode secrets
- \$ echo -n "bX1zcWw=" | base64 --decode
- \$ echo -n "cm9vdA==" | base64 --decode
- \$ echo -n "cGFzd3Jk" | base64 --decode

IDecode Secrets



Configuring secret with a pod

- To inject a secret to a pod add a new property `envFrom` followed by `secretRef` name and then create the pod-definition
- `apiVersion: v1`
- `kind: Secret`
- `metadata:`
- `name: app-secret`
- `data:`
 - `DB_Host: bXlzcWw=`
 - `DB_User: cm9vdA==`
 - `DB_Password: cGFzd3Jk`
- `apiVersion: v1`
- `kind: Pod`
- `metadata:`
 - `name: simple-webapp-color`
- `spec:`
 - `containers:`
 - `name: simple-webapp-color`
 - `image: simple-webapp-color`
 - `ports:`
 - `containerPort: 8080`
 - `envFrom:`
 - `secretRef:`
 - `name: app-secret`
- `$ kubectl create -f pod-definition.yaml`

I Secrets in Pods

pod-definition.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: simple-webapp-color
  labels:
    name: simple-webapp-color
spec:
  containers:
  - name: simple-webapp-color
    image: simple-webapp-color
    ports:
      - containerPort: 8080
    envFrom:
      - secretRef:
          name: app-secret
```

secret-data.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: app-secret
data:
  DB_Host: bXlzcWw=
  DB_User: cm9vdA==
  DB_Password: cGFzd3Jk
```



Inject into Pod

There are other ways to inject secrets into pods.

- You can inject as **Single ENV variable**
- You can inject as whole secret as files in a **Volume**

Secrets in Pods

```
envFrom:  
  - secretRef:  
      name: app-config
```

ENV

SINGLE ENV

```
env:  
  - name: DB_Password  
    valueFrom:  
      secretKeyRef:  
        name: app-secret  
        key: DB_Password
```

VOLUME

```
volumes:  
  - name: app-secret-volume  
    secret:  
      secretName: app-secret
```

Secrets in pods as volume

- Each attribute in the secret is created as a file with the value of the secret as its content.

Secrets in Pods as Volumes



K8s Reference Docs

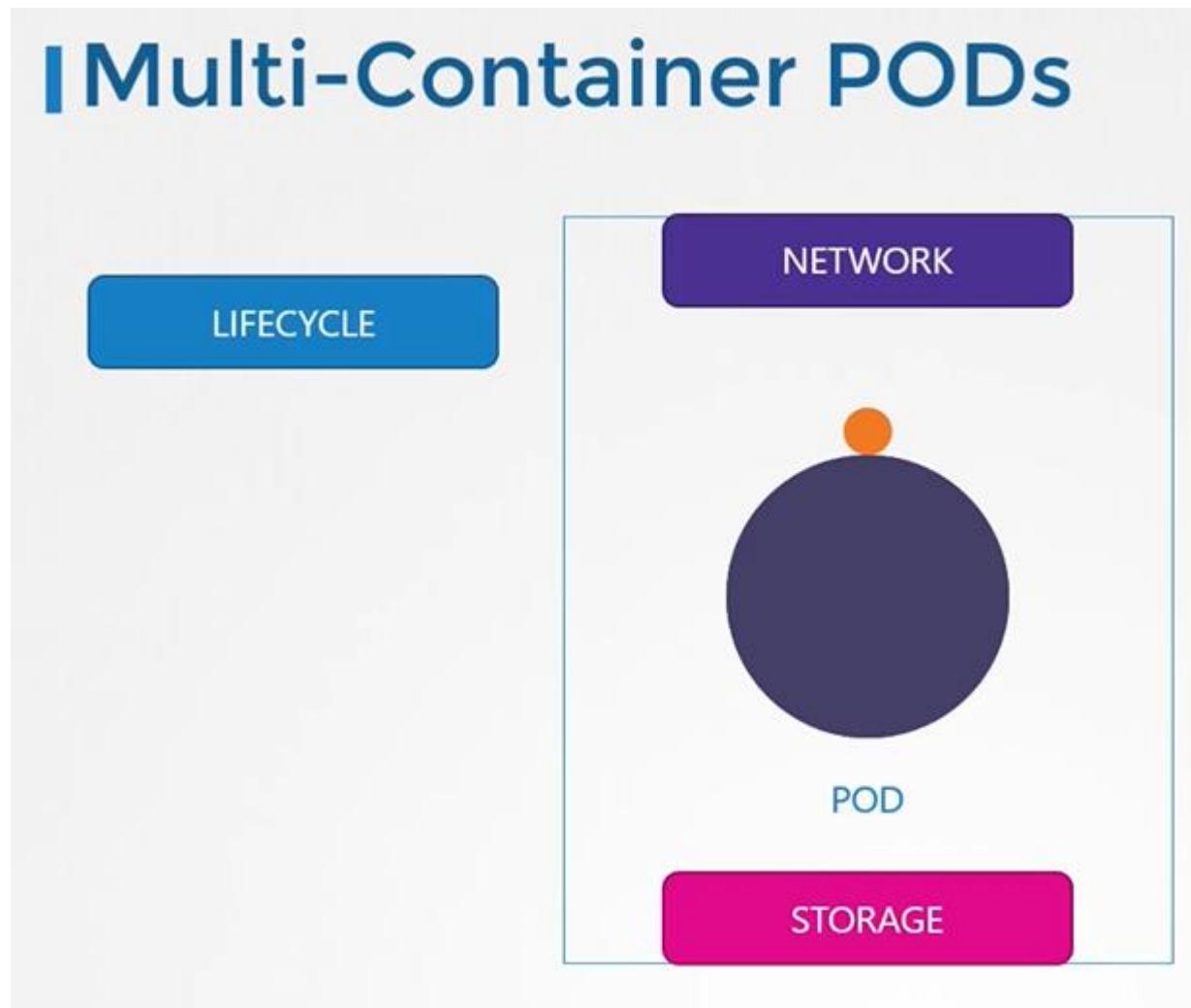
- <https://kubernetes.io/docs/concepts/configuration/secret/>
- <https://kubernetes.io/docs/concepts/configuration/secret/#use-cases>
- <https://kubernetes.io/docs/tasks/inject-data-application/distribute-credentials-secure/>

Multi-Container Pods

In this section, we will take a look at multi-container pods

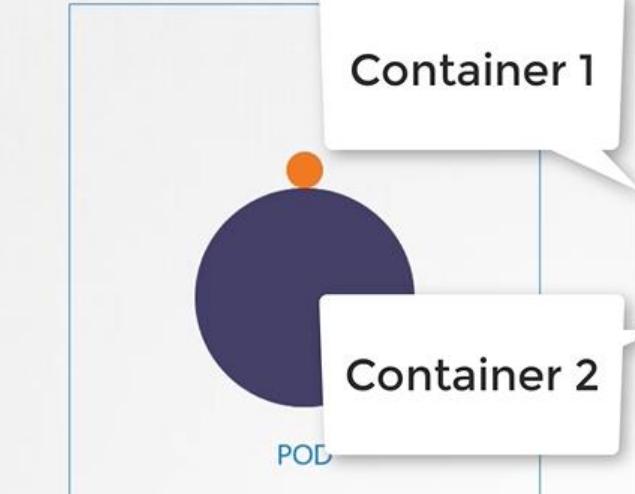
Monolith and Microservices



Multi-Container Pods

- To create a new multi-container pod, add the new container information to the pod definition file.
- apiVersion: v1
- kind: Pod
- metadata:
- name: simple-webapp
- labels:
- name: simple-webapp
- spec:
- containers:
- - name: simple-webapp
- image: simple-webapp
- ports:
- - ContainerPort: 8080
- - name: log-agent
- image: log-agent

I Create



Container 1

Container 2

POD

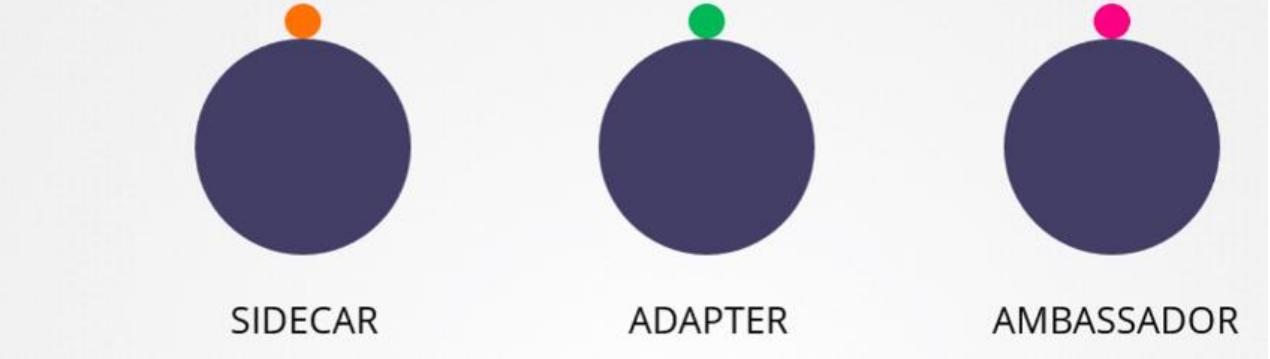
```
pod-definition.yaml
apiVersion: v1
kind: Pod
metadata:
  name: simple-webapp
  labels:
    name: simple-webapp
spec:
  containers:
    - name: simple-webapp
      image: simple-webapp
      ports:
        - containerPort: 8080
    - name: log-agent
      image: log-agent
```

K8s Reference Docs

- <https://kubernetes.io/docs/tasks/access-application-cluster/communicate-containers-same-pod-shared-volume/>

Multi-Container Pods Design Patterns

IDesign Patterns



SIDECAR ADAPTER AMBASSADOR

K8s Reference Docs

- <https://kubernetes.io/blog/2015/06/the-distributed-system-toolkit-patterns/>

Init-Containers

K8s Reference Docs

- <https://kubernetes.io/docs/concepts/workloads/pods/init-containers/>
- <https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-initialization/>

Cluster Maintenance Section Introduction

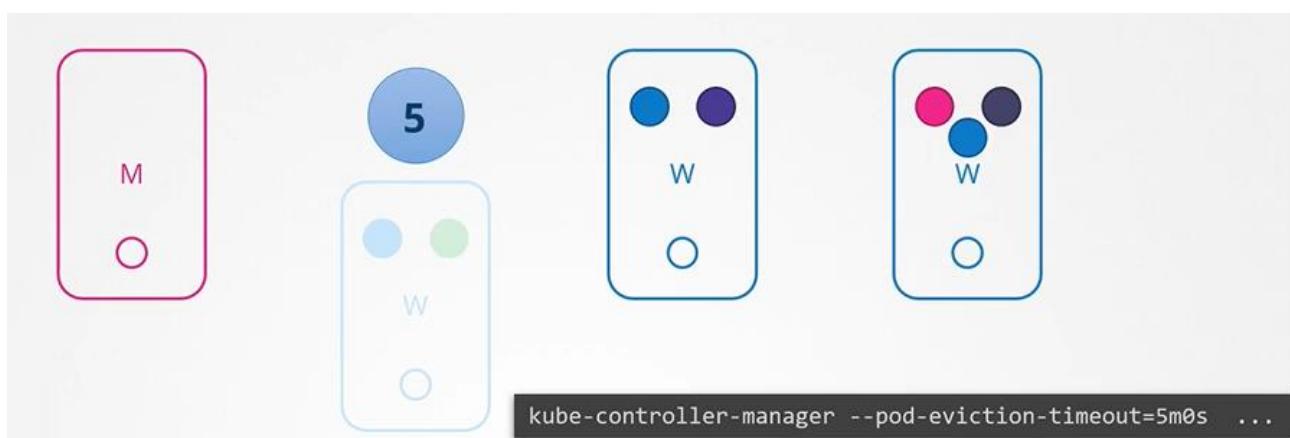
In this section, we will take a look at cluster maintenance

- Cluster Upgrade Process
- Operating System Upgrades
- Backup and Restore Methodologies

OS Upgrades

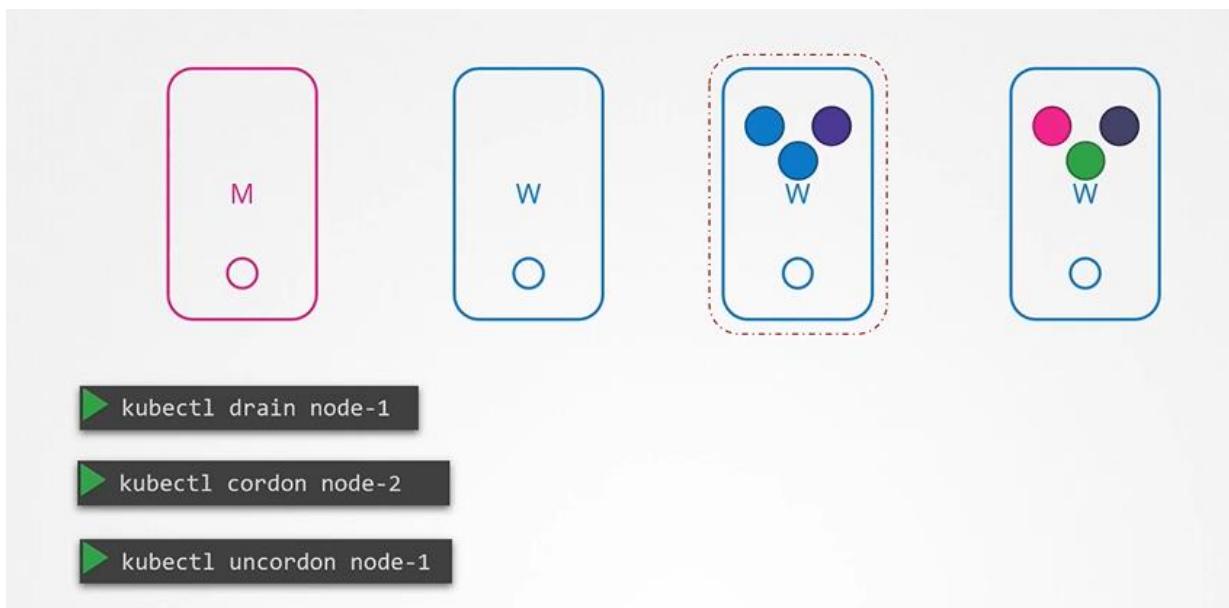
In this section, we will take a look at os upgrades.

If the node was down for more than 5 minutes, then the pods are terminated from that node



- You can purposefully **drain** the node of all the workloads so that the workloads are moved to other nodes.
- \$ kubectl drain node-1

- The node is also cordoned or marked as unschedulable.
- When the node is back online after a maintenance, it is still unschedulable. You then need to uncordon it.
- `$ kubectl uncordon node-1`
- There is also another command called cordon. Cordon simply marks a node unschedulable. Unlike drain it does not terminate or move the pods on an existing node.



K8s Reference Docs

- <https://kubernetes.io/docs/tasks/administer-cluster/safely-drain-node/>

Kubernetes Software Versions

In this section, we will take a look at various kubernetes releases and versions

We can see the kubernetes version that we installed

```
$ kubectl get nodes
```

kubectl get nodes				
NAME	STATUS	ROLES	AGE	VERSION
master	Ready	master	1d	v1.11.3
node-1	Ready	<none>	1d	v1.11.3
node-2	Ready	<none>	1d	v1.11.3

Let's take a closer look at the version number

- It consists of 3 parts
 - First is the major version
 - Second is the minor version
 - Finally, the patch version

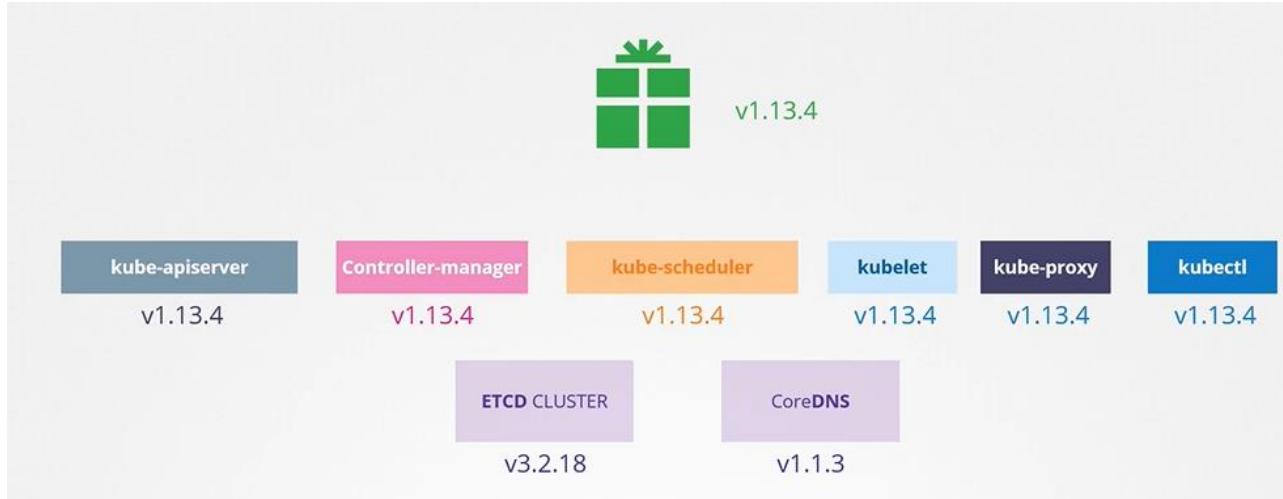


Kubernetes follows a standard software release versioning procedure

- You can find all kubernetes releases at <https://github.com/kubernetes/kubernetes/releases>



Downloaded package has all the kubernetes components in it except ETCD Cluster and CoreDNS as they are separate projects.



References

- <https://github.com/kubernetes/community/blob/master/contributors/design-proposals/release/versioning.md>
- <https://github.com/kubernetes/community/blob/master/contributors/design-proposals/api-machinery/api-group.md>
- <https://blog.risingstack.com/the-history-of-kubernetes/>
- <https://kubernetes.io/docs/setup/release/version-skew-policy/>

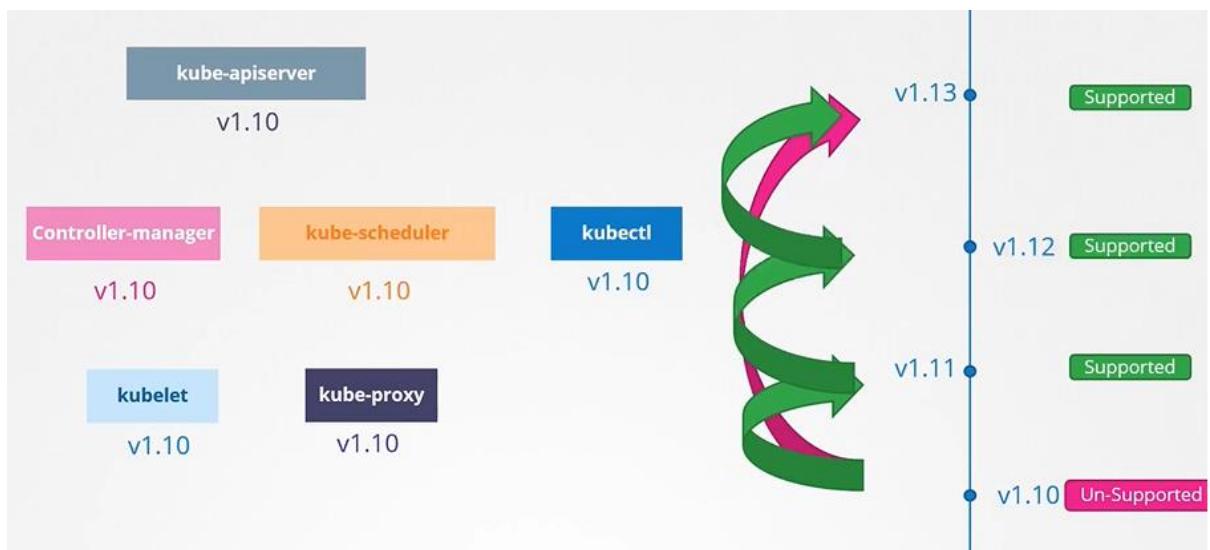
Cluster Upgrade Introduction

Is it mandatory for all of the kubernetes components to have the same versions?

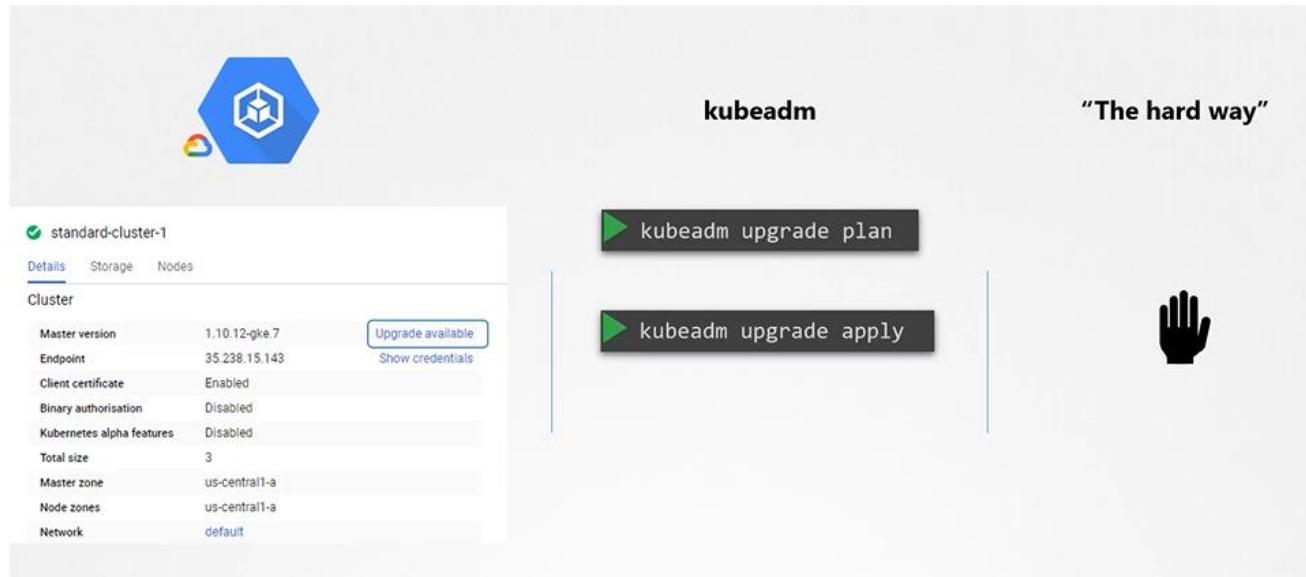
- No, The components can be at different release versions.

At any time, kubernetes supports only up to the recent 3 minor versions

- The recommended approach is to upgrade one minor version at a time.



Options to upgrade k8s cluster



Upgrading a Cluster

- Upgrading a cluster involves 2 major steps

There are different strategies that are available to upgrade the worker nodes

- One is to upgrade all at once. But then your pods will be down and users will not be able to access the

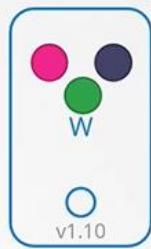
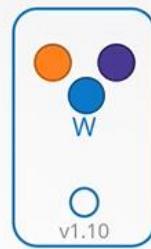
applications.

Strategy - 1

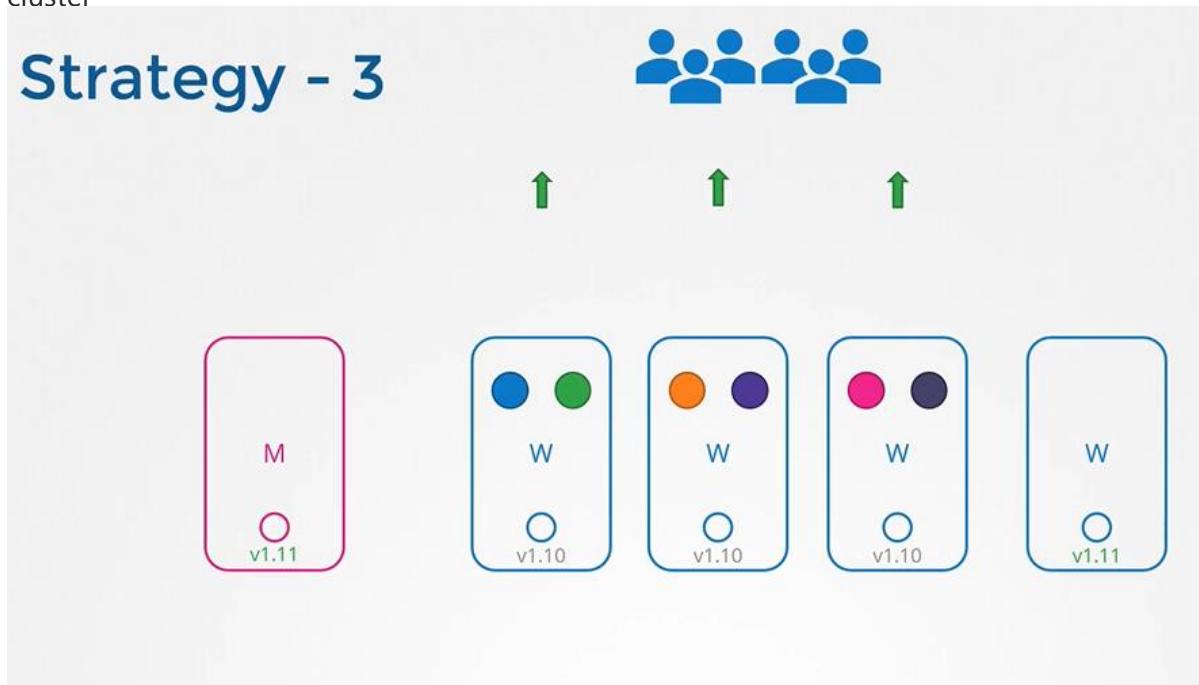


- Second one is to upgrade one node at a time.

Strategy - 2



- Third one would be to add new nodes to the cluster



kubeadm - Upgrade master node

- kubeadm has an upgrade command that helps in upgrading clusters.
- \$ kubeadm upgrade plan

kubeadm - up

```
kubeadm upgrade plan
[preflight] Running pre-flight checks.
[upgrade] Making sure the cluster is healthy:
[upgrade/config] Making sure the configuration is correct:
[upgrade] Fetching available versions to upgrade to
[upgrade/versions] Cluster version: v1.11.8
[upgrade/versions] kubeadm version: v1.11.3
[upgrade/versions] Latest stable version: v1.13.4
[upgrade/versions] Latest version in the v1.11 series: v1.11.8

Components that must be upgraded manually after you have
upgraded the control plane with 'kubeadm upgrade apply':
COMPONENT      CURRENT      AVAILABLE
Kubelet        3 x v1.11.3  v1.13.4

Upgrade to the latest stable version:

COMPONENT      CURRENT      AVAILABLE
API Server    v1.11.8      v1.13.4
Controller Manager  v1.11.8  v1.13.4
Scheduler     v1.11.8      v1.13.4
Kube Proxy    v1.11.8      v1.13.4
CoreDNS       1.1.3        1.1.3
Etcd          3.2.18       N/A

You can now apply the upgrade by executing the following command:
  kubeadm upgrade apply v1.13.4

Note: Before you can perform this upgrade, you have to update kubeadm to v1.13.4.
```

- Upgrade kubeadm from v1.11 to v1.12

- \$ apt-get upgrade -y kubeadm=1.12.0-00
- Upgrade the cluster
- \$ kubeadm upgrade apply v1.12.0
- If you run the 'kubectl get nodes' command, you will see the older version. This is because in the output of the command it is showing the versions of kubelets on each of these nodes registered with the API Server and not the version of API Server itself
- \$ kubectl get nodes

kubeadm - upgrade



```

apt-get upgrade -y kubeadm=1.12.0-00
kubeadm upgrade apply v1.12.0
...
[upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.12.0". Enjoy!
[upgrade/kubelet] Now that your control plane is upgraded, please proceed with
upgrading your kubelets if you haven't already done so.

kubectl get nodes
NAME      STATUS   ROLES      AGE      VERSION
master    Ready    master    1d       v1.11.3
node-1   Ready    <none>   1d       v1.11.3
node-2   Ready    <none>   1d       v1.11.3

```

- Upgrade 'kubelet' on the master node
- \$ apt-get upgrade kubelet=1.12.0-00
- Restart the kubelet
- \$ systemctl restart kubelet
- Run 'kubectl get nodes' to verify
- \$ kubectl get nodes

kubeadm - upgrade



```
▶ kubectl get nodes
NAME      STATUS   ROLES     AGE      VERSION
master    Ready    master    1d      v1.11.3
node-1   Ready    <none>   1d      v1.11.3
node-2   Ready    <none>   1d      v1.11.3
```



```
▶ apt-get upgrade -y kubelet=1.12.0-00
▶ systemctl restart kubelet
```



```
▶ kubectl get nodes
NAME      STATUS   ROLES     AGE      VERSION
master    Ready    master    1d      v1.12.0
node-1   Ready    <none>   1d      v1.11.3
node-2   Ready    <none>   1d      v1.11.3
```

kubeadm - Upgrade worker nodes

- From master node, run 'kubectl drain' command to move the workloads to other nodes
- \$ kubectl drain node-1
- Upgrade kubeadm and kubelet packages
- \$ apt-get upgrade -y kubeadm=1.12.0-00
- \$ apt-get upgrade -y kubelet=1.12.0-00
- Update the node configuration for the new kubelet version
- \$ kubeadm upgrade node config --kubelet-version v1.12.0
- Restart the kubelet service
- \$ systemctl restart kubelet
- Mark the node back to schedulable
- \$ kubectl uncordon node-1

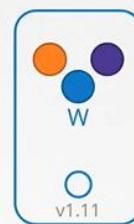
kubeadm - upgrade



```
▶ kubectl drain node-1
▶ kubectl uncordon node-1
```



```
▶ apt-get upgrade -y kubeadm=1.12.0-00
▶ apt-get upgrade -y kubelet=1.12.0-00
▶ kubeadm upgrade node config --kubelet-version v1.12.0
▶ systemctl restart kubelet
```



- Upgrade all worker nodes in the same way

kubeadm - upgrade



```
▶ kubectl drain node-1
▶ kubectl uncordon node-1
▶ kubectl drain node-2
▶ kubectl uncordon node-2
▶ kubectl drain node-3
▶ kubectl uncordon node-3
```



K8s Reference Docs

- <https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/>
- <https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm-upgrade/>

Backup and Restore Methods

In this section, we will take a look at backup and restore methods

Backup Candidates

Backup Candidates



Resource Configuration



ETCD Cluster



Persistent Volumes

Resource Configuration

- Imperative way

Imperative



Resource Configuration

```
▶ kubectl create namespace new-namespace
```

```
▶ kubectl create secret
```

```
▶ kubectl create configmap
```

- Declarative Way (Preferred approach)

```
• apiVersion: v1
• kind: Pod
• metadata:
  •   name: myapp-pod
  •   labels:
    •     app: myapp
    •     type: front-end
  • spec:
    • containers:
      • - name: nginx-container
        • image: nginx
```

I Declarative



Resource Configuration

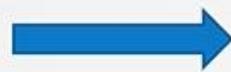
```
pod-definition.yml
apiVersion: v1
kind: Pod

metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
  containers:
    - name: nginx-container
      image: nginx
```

▶ kubectl apply -f pod-definition.yml

- A good practice is to store resource configurations on source code repositories like github.

I Declarative



GitHub

Resource Configuration

Backup - Resource Configs

```
$ kubectl get all --all-namespaces -o yaml > all-deploy-services.yaml (only for few resource groups)
```

- There are many other resource groups that must be considered. There are tools like **ARK** or now called **Velero** by Heptio that can do this for you.

I Backup - Resource Configs

kube-apiserver



Resource Configuration

```
▶ kubectl get all --all-namespaces -o yaml > all-deploy-services.yaml
```



VELERO

Formerly called ARK by HeptIO

Backup - ETCD

- So, instead of backing up resources as before, you may choose to backup the ETCD cluster itself.

I Backup - ETCD

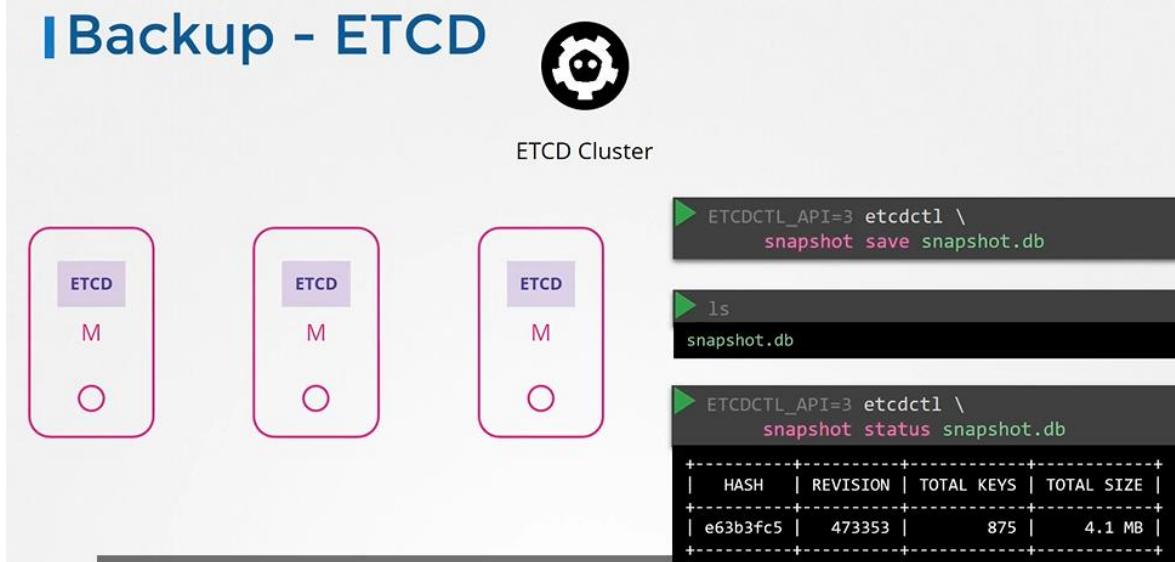


ETCD Cluster



```
etcd.service
ExecStart=/usr/local/bin/etcd \
--name ${ETCD_NAME} \
--cert-file=/etc/etcd/kubernetes.pem \
--key-file=/etc/etcd/kubernetes-key.pem \
--peer-cert-file=/etc/etcd/kubernetes.pem \
--peer-key-file=/etc/etcd/kubernetes-key.pem \
--trusted-ca-file=/etc/etcd/ca.pem \
--peer-trusted-ca-file=/etc/etcd/ca.pem \
--peer-client-cert-auth \
--client-cert-auth \
--initial-advertise-peer-urls https://$(INTERNAL_IP):2380 \
--listen-peer-urls https://$(INTERNAL_IP):2380 \
--listen-client-urls https://$(INTERNAL_IP):2379,https \
--advertise-client-urls https://$(INTERNAL_IP):2379 \
--initial-cluster-token etcd-cluster-0 \
--initial-cluster controller-0=https://$(CONTROLLER0) \
--initial-cluster-state new \
--data-dir=/var/lib/etcd
```

- You can take a snapshot of the etcd database by using `etcdctl` utility `snapshot save` command.
- `$ ETCDCTL_API=3 etcdctl snapshot save snapshot.db`
- `$ ETCDCTL_API=3 etcdctl snapshot status snapshot.db`



Restore - ETCD

- To restore etcd from the backup at later in time. First stop kube-apiserver service
- `$ service kube-apiserver stop`
- Run the `etcdctl snapshot restore` command
- Update the etcd service
- Reload system configs
- `$ systemctl daemon-reload`
- Restart etcd
- `$ service etcd restart`

Restore - ETCD



ETCD Cluster

```
▶ ETCDCCTL_API=3 etcdctl \
snapshot restore snapshot.db \
--data-dir /var/lib/etcd-from-backup \
--initial-cluster master-
1=https://192.168.5.11:2380,master-
2=https://192.168.5.12:2380 \
--initial-cluster-token etcd-cluster-1 \
--initial-advertise-peer-urls
https://${INTERNAL_IP}:2380

I | mvcc: restore compact to 475629
I | etcdserver/membership: added member 5e89ccdfc3
[https://192.168.5.12:2380] to cluster 894c7131f5165a78
I | etcdserver/membership: added member c8246cee7c
[https://192.168.5.11:2380] to cluster 894c7131f5165a78

▶ systemctl daemon-reload

▶ service etcd restart
Service etcd restarted
```

```
▶ ETCDCCTL_API=3 etcdctl \
snapshot save snapshot.db

▶ ls
snapshot.db

▶ service kube-apiserver stop
Service kube-apiserver stopped

[etcd.service]
ExecStart=/usr/local/bin/etcd \\
--name ${ETCD_NAME} \\
--cert-file=/etc/etcd/kubernetes.pem \\
--key-file=/etc/etcd/kubernetes-key.pem \\
--peer-cert-file=/etc/etcd/kubernetes.pem \\
--peer-key-file=/etc/etcd/kubernetes-key.pem \\
--trusted-ca-file=/etc/etcd/ca.pem \\
--peer-trusted-ca-file=/etc/etcd/ca.pem \\
--peer-client-cert-auth \\
--client-cert-auth \\
--initial-advertise-peer-urls https://${INTERNAL_IP}:2380 \\
--listen-peer-urls https://${INTERNAL_IP}:2380 \\
--listen-client-urls https://${INTERNAL_IP}:2379,https://${INTERNAL_IP}:2380 \\
--advertise-client-urls https://${INTERNAL_IP}:2379 \\
--initial-cluster-token etcd-cluster-1 \\
--initial-cluster controller-0=https://${CONTROLLER_IP}:2380 \\
--initial-cluster-state new \\
--data-dir=/var/lib/etcd-from-backup
```

- Start the kube-apiserver
- `$ service kube-apiserver start`

With all etcdctl commands specify the cert,key,cacert and endpoint for authentication.

```
$ ETCDCCTL_API=3 etcdctl --endpoints=https://[127.0.0.1]:2379 --
cacert=/etc/kubernetes/pki/etcd/ca.crt \
--cert=/etc/kubernetes/pki/etcd/etcd-server.crt \
--key=/etc/kubernetes/pki/etcd/etcd-server.key snapshot save /tmp/snapshot.db
```

```
▶ ETCDCCTL_API=3 etcdctl \
snapshot save snapshot.db \
--endpoints=https://127.0.0.1:2379 \
--cacert=/etc/etcd/ca.crt \
--cert=/etc/etcd/etcd-server.crt \
--key=/etc/etcd/etcd-server.key
```

K8s Reference Docs

- <https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/>

Security Section Introduction

In this section, we will take a look at security section introduction

- Kubernetes Security Primitives
- Authentication
- TLS certificates for cluster components
- Secure Persistent key-value store
- Authorization
- Images Security
- Security Contexts
- Network Policies

Kubernetes Security Primitives

In this section, we will take a look at kubernetes security primitives

Secure Hosts

| Secure Hosts



- Password based authentication disabled
- SSH Key based authentication

Secure Kubernetes

- We need to make two types of decisions.
 - Who can access?
 - What can they do?



Authentication

- Who can access the API Server is defined by the Authentication mechanisms.

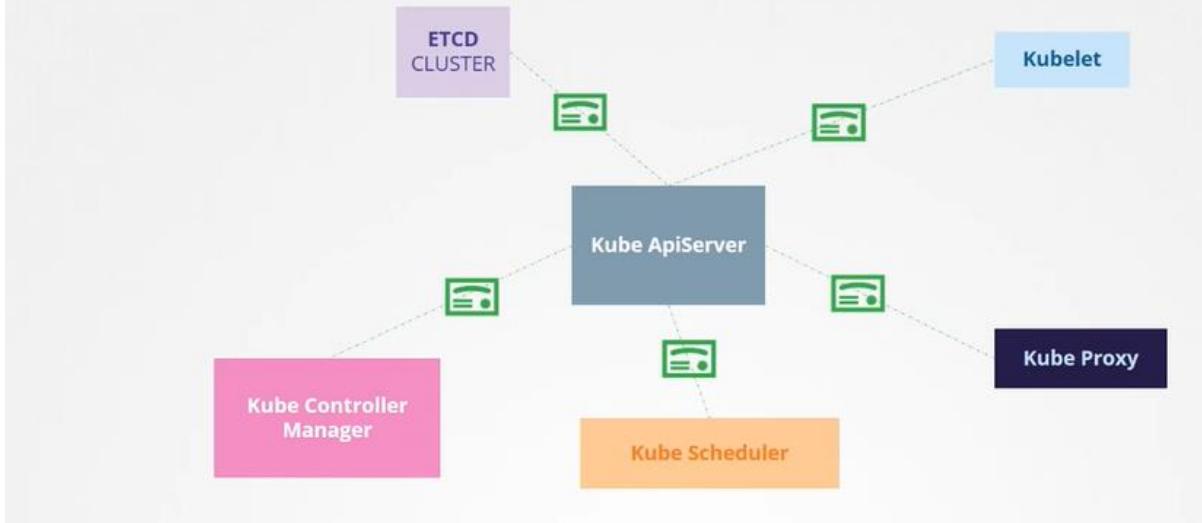
Authorization

- Once they gain access to the cluster, what they can do is defined by authorization mechanisms.

TLS Certificates

- All communication with the cluster, between the various components such as the ETCD Cluster, kube-controller-manager, scheduler, api server, as well as those running on the working nodes such as the kubelet and kubeproxy is secured using TLS encryption.

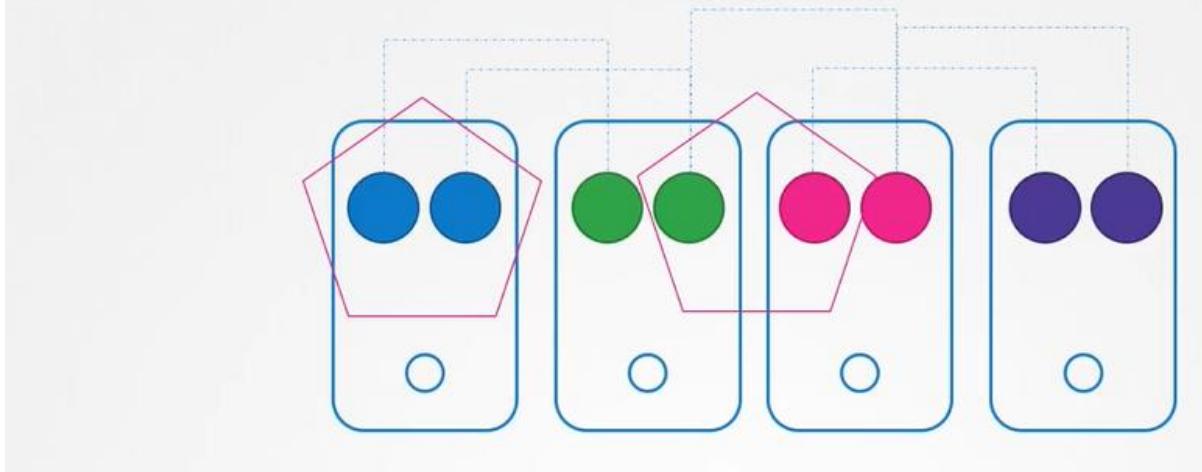
ITLS Certificates



Network Policies

What about communication between applications within the cluster?

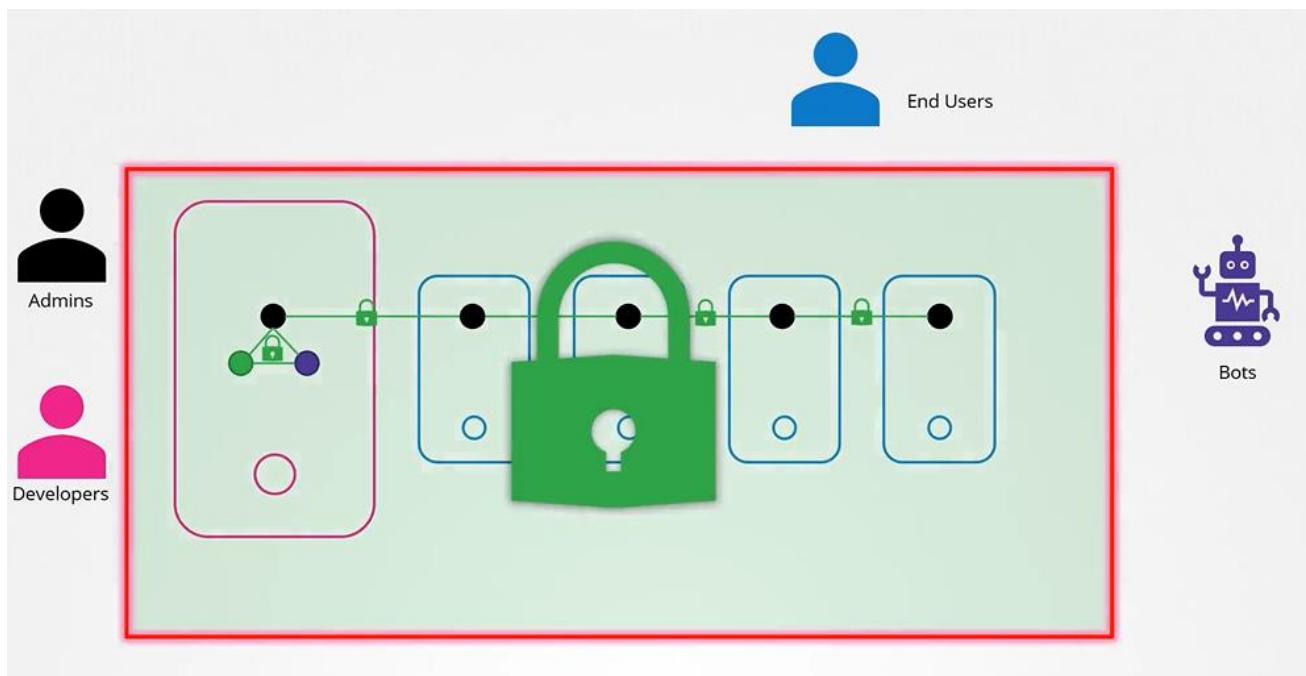
Network Policies



Authentication

In this section, we will take a look at authentication in a kubernetes cluster

Accounts



Different users that may be accessing the cluster security of end users who access the applications deployed on the cluster is managed by the applications themselves internally.



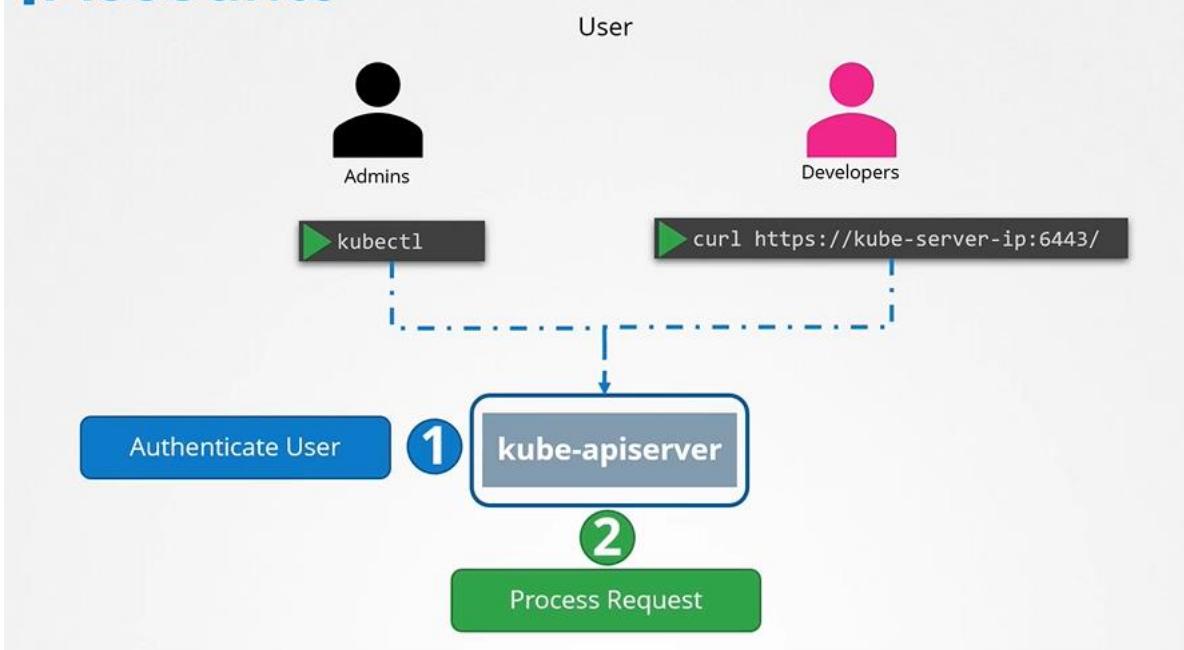
- So, we left with 2 types of users
 - Humans, such as the Administrators and Developers
 - Robots such as other processes/services or applications that require access to the cluster.

IAccounts



- All user access is managed by apiserver and all of the requests goes through apiserver.

IAccounts



Authentication Mechanisms

- There are different authentication mechanisms that can be configured.

IAuth Mechanisms

kube-apiserver

Static Password File



Static Token File



Certificates



Identity Services



Authentication Mechanisms - Basic

kube-apiserver

--basic-auth-file=user-details.csv

user-details.csv

```
password123,user1,u0001
password123,user2,u0002
password123,user3,u0003
password123,user4,u0004
password123,user5,u0005
```

kube-apiserver.service

```
ExecStart=/usr/local/bin/kube-apiserver \\
--advertise-address=${INTERNAL_IP} \\
--allow-privileged=true \\
--apiserver-count=3 \\
--authorization-mode=Node,RBAC \\
--bind-address=0.0.0.0 \\
--enable-swagger-ui=true \\
--etcd-servers=https://127.0.0.1:2379 \\
--event-ttl=1h \\
--runtime-config=api/all \\
--service-cluster-ip-range=10.32.0.0/24 \\
--service-node-port-range=30000-32767 \\
--v=2
--basic-auth-file=user-details.csv
```

kube-apiserver configuration

- If you set up via kubeadm then update kube-apiserver.yaml manifest file with the option.

Kube-api Server Configuration

kube-apiserver.service

```
ExecStart=/usr/local/bin/kube-apiserver \
--advertise-address=${INTERNAL_IP} \
--allow-privileged=true \
--apiserver-count=3 \
--authorization-mode=Node,RBAC \
--bind-address=0.0.0.0 \
--enable-swagger-ui=true \
--etcd-servers=https://127.0.0.1:2379 \
--event-ttl=1h \
--runtime-config=api/all \
--service-cluster-ip-range=10.32.0.0/24 \
--service-node-port-range=30000-32767 \
--v=2 \
--basic-auth-file=user-details.csv
```

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

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
    - command:
        - kube-apiserver
        - --authorization-mode=Node,RBAC
        - --advertise-address=172.17.0.107
        - --allow-privileged=true
        - --enable-admission-plugins=NodeRestriction
        - --enable-bootstrap-token-auth=true
        - --basic-auth-file=user-details.csv
      image: k8s.gcr.io/kube-apiserver-amd64:v1.11.3
      name: kube-apiserver
```

Note: Showing fewer options for simplicity

Authenticate User

- To authenticate using the basic credentials while accessing the API server specify the username and password in a curl command.
- \$ curl -v -k http://master-node-ip:6443/api/v1/pods -u "user1:password123"

I Authenticate User

```
curl -v -k https://master-node-ip:6443/api/v1/pods -u "user1:password123"
{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "selfLink": "/api/v1/pods",
    "resourceVersion": "3594"
  },
  "items": [
    {
      "metadata": {
        "name": "nginx-64f497f8fd-krkg6",
        "generateName": "nginx-64f497f8fd-",
        "namespace": "default",
        "selfLink": "/api/v1/namespaces/default/pods/nginx-64f497f8fd-krkg6",
        "uid": "77dd7dfb-2914-11e9-b468-0242ac11006b",
        "resourceVersion": "3569",
        "creationTimestamp": "2019-02-05T07:05:49Z",
        "labels": {
          "pod-template-hash": "2090539498",
          "run": "nginx"
        }
      }
    }
  ]
}
```

- We can have additional column in the user-details.csv file to assign users to specific groups.

I Auth Mechanisms - Basic

Static Password File

user-details.csv

```
password123,user1,u0001,group1
password123,user2,u0002,group1
password123,user3,u0003,group2
password123,user4,u0004,group2
password123,user5,u0005,group2
```

Static Token File

user-token-details.csv

```
KpjCVbI7rCFAHYPkByTlzRb7gu1cUc4B,user10,u0010,group1
rJjncHmvtxHc6M1WQddhtvNyyhgTdxSC,user11,u0011,group1
mjpOFIEiFOkL9toikaRNtt59ePtczzSq,user12,u0012,group2
PG41IXhs7QjqwWkmBkvqGT9g1OyUqZij,user13,u0013,group2
```

--token-auth-file=user-details.csv

```
curl -v -k https://master-node-ip:6443/api/v1/pods --header "Authorization: Bearer KpjCVbI7rCFAHYPkBzRb7gu1cUc4B"
```

Note

I Note

- This is not a recommended authentication mechanism
- Consider volume mount while providing the auth file in a kubeadm setup
- Setup Role Based Authorization for the new users

K8s Reference Docs

- <https://kubernetes.io/docs/reference/access-authn-authz/authentication/>

TLS Certificates

In this section, we will take a look at TLS certificates

- What are TLS certificates?
- How does kubernetes use certificates?
- How to generate them?
- How to configure them?
- How to view them?
- How to troubleshoot issues related to certificates

TLS Basics

In this section, we will take a look at TLS Basics

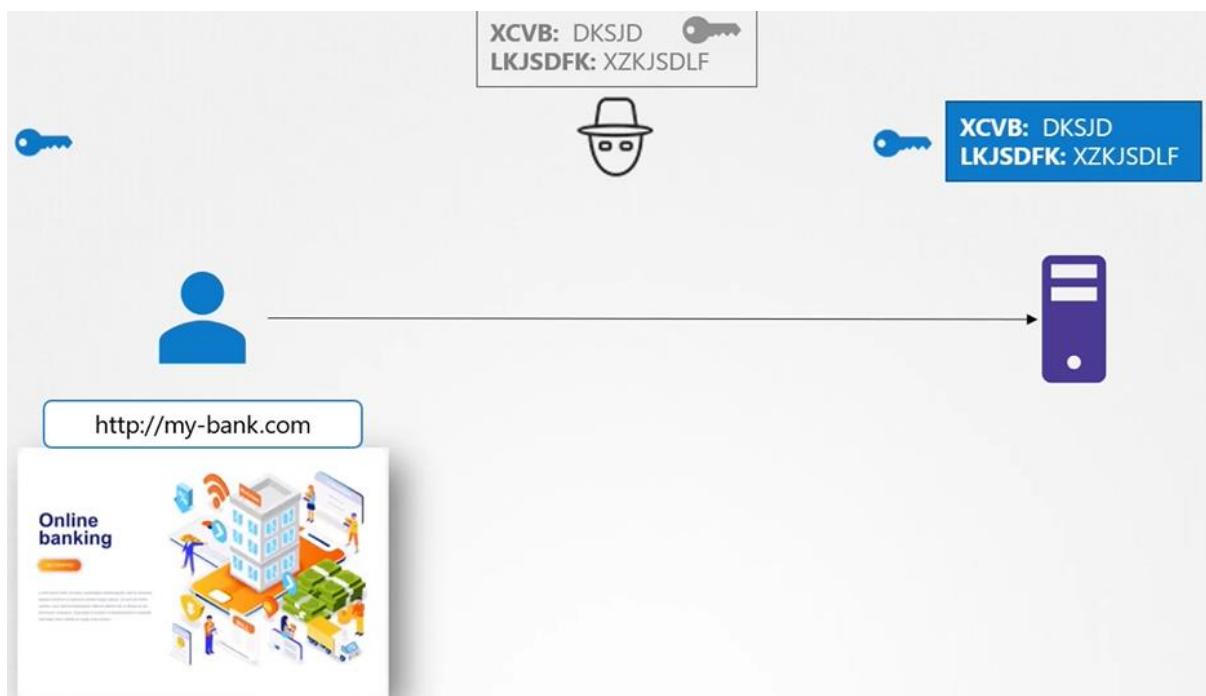
Certificate

- A certificate is used to guarantee trust between 2 parties during a transaction.
- Example: when a user tries to access web server, tls certificates ensure that the communication between them is encrypted.



Symmetric Encryption

- It is a secure way of encryption, but it uses the same key to encrypt and decrypt the data and the key has to be exchanged between the sender and the receiver, there is a risk of a hacker gaining access to the key and decrypting the data.



Asymmetric Encryption

- Instead of using single key to encrypt and decrypt data, asymmetric encryption uses a pair of keys, a private key and a public key.

ASYMMETRIC ENCRYPTION - SSH

```
▶ ssh-keygen  
id_rsa  id_rsa.pub
```



Private Key

```
▶ cat ~/.ssh/authorized_keys  
ssh-rsa AAAAB3NzaC1yc...KhtUBfoTz1BqR  
V1NThvOo4opzEwRQo1mWx user1
```



```
▶ ssh -i id_rsa user1@server1  
Successfully Logged In!
```

ASYMMETRIC ENCRYPTION - SSH

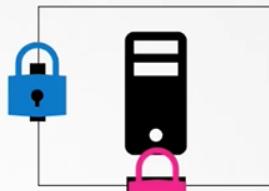
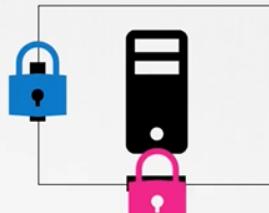


Private Key

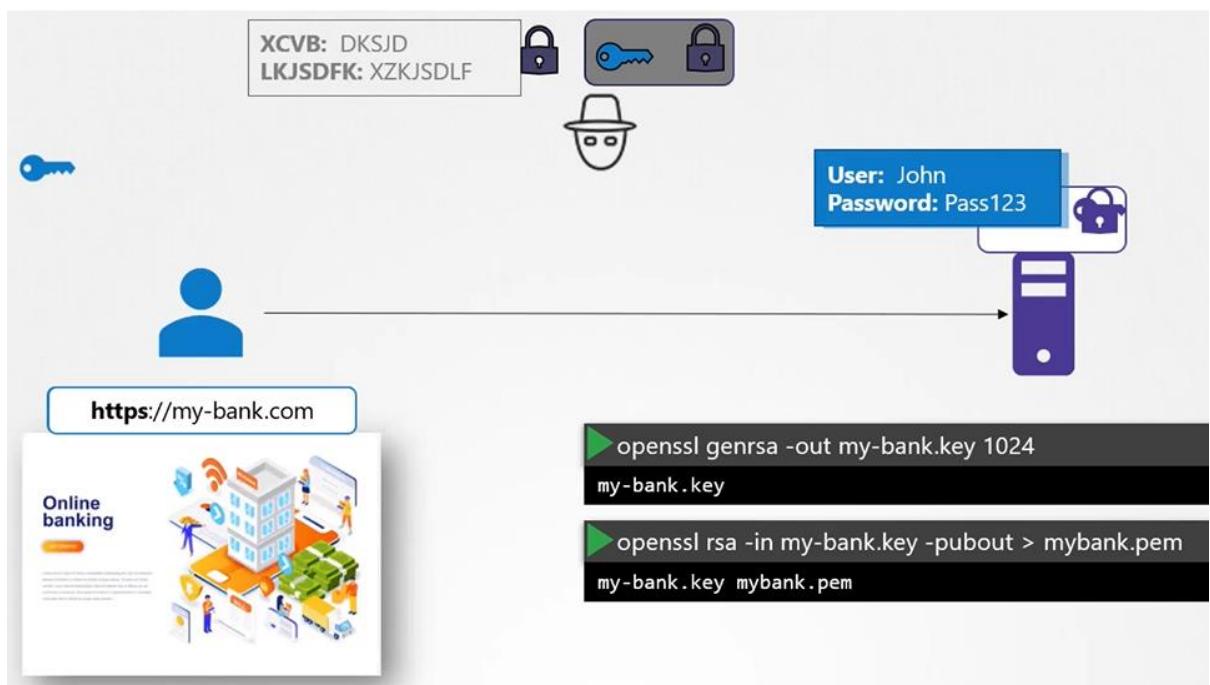


Private Key

Public Lock

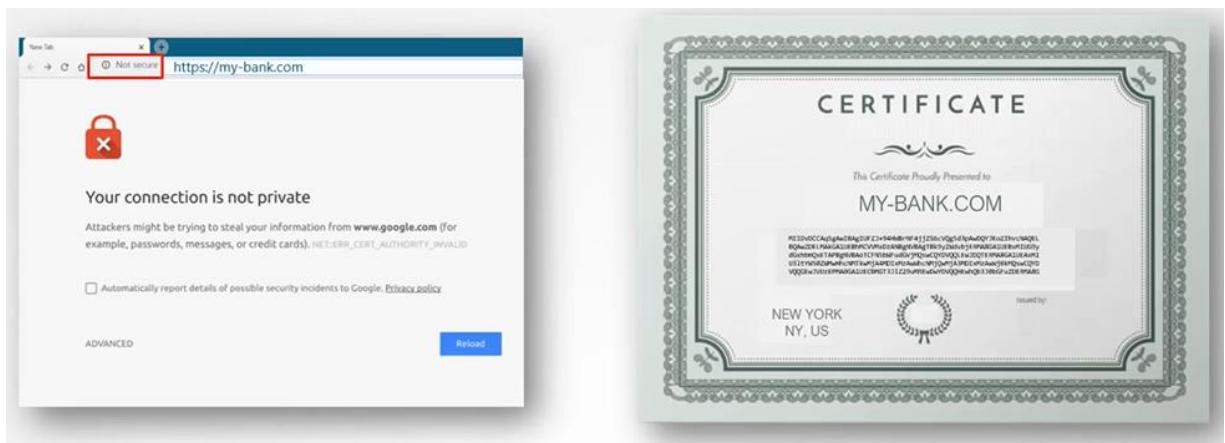


```
▶ cat ~/.ssh/authorized_keys  
ssh-rsa AAAAB3NzaC1yc...KhtUBfoTz1BqR  
V1NThvOo4opzEwRQo1mWx user1  
ssh-rsa AAAXCVJSDFDF...SLKJSDLKFw23423xckjSDFDFLKJLSDFKJLx user2
```



How do you look at a certificate and verify if it is legit?

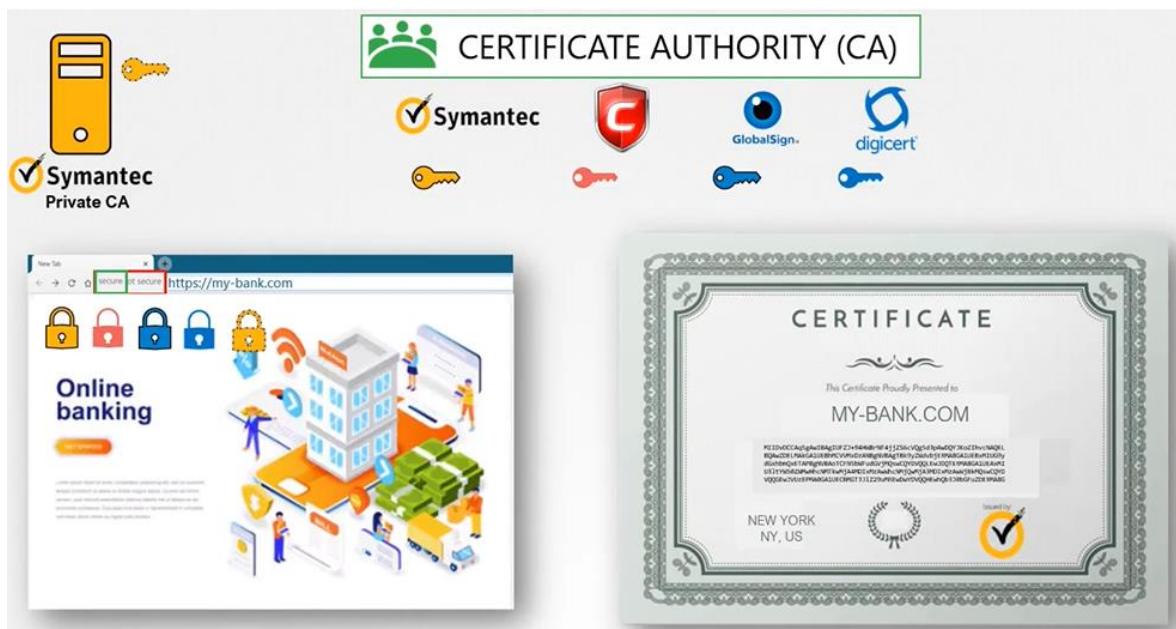
- who signed and issued the certificate.
- If you generate the certificate then you will have it sign it by yourself; that is known as self-signed certificate.



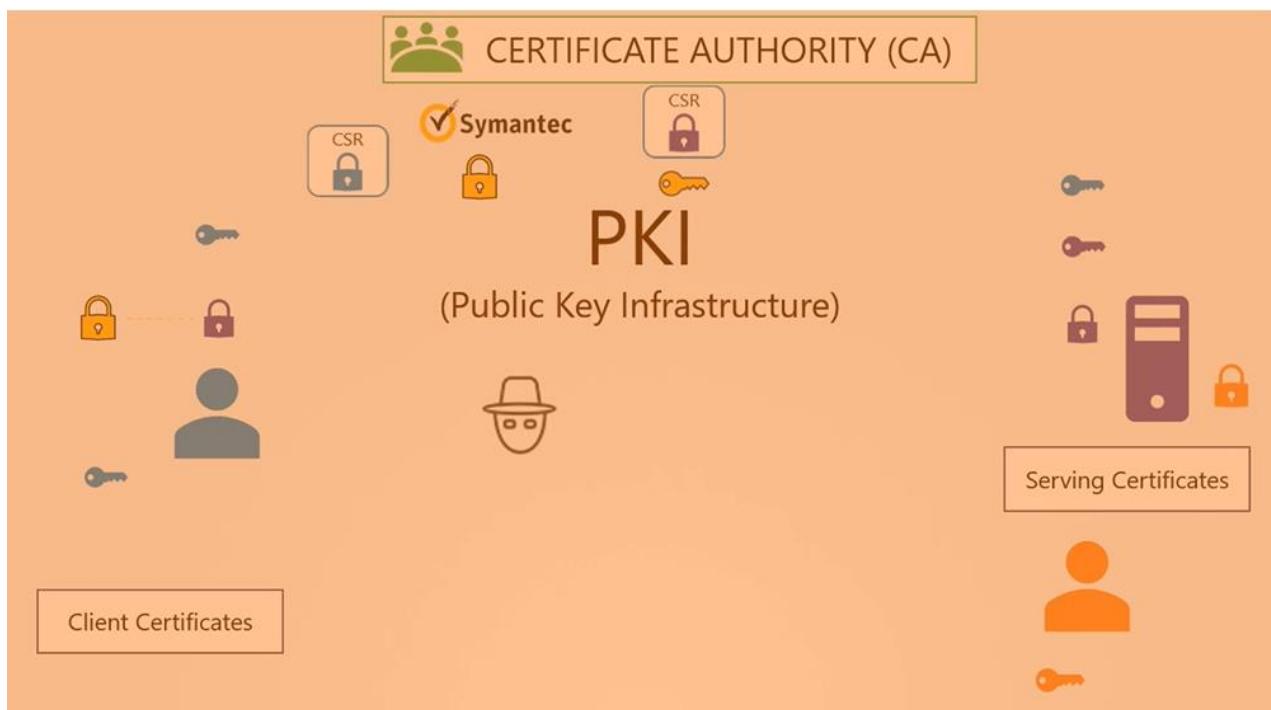
How do you generate legitimate certificate? How do you get your certificates singed by someone with authority?

- That's where **Certificate Authority (CA)** comes in for you. Some of the popular ones are Symantec, DigiCert, Comodo, GlobalSign etc.

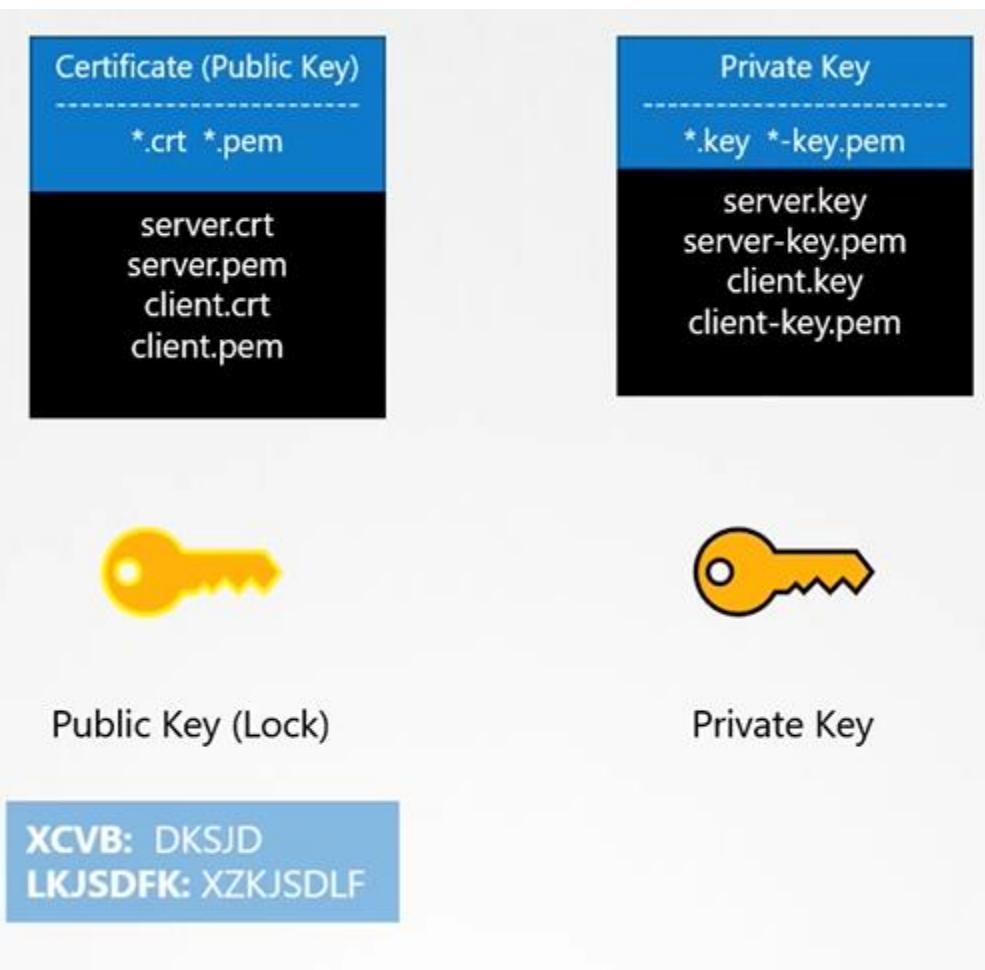
```
openssl req -new -key my-bank.key -out my-bank.csr
-subj "/C=US/ST=CA/O=MyOrg, Inc./CN=my-bank.com"
my-bank.key my-bank.csr
```



Public Key Infrastructure



Certificates naming convention



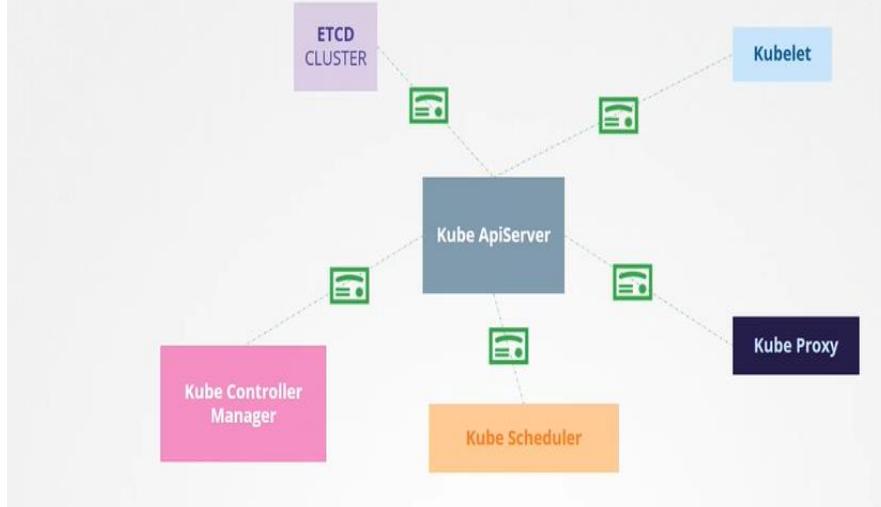
TLS in Kubernetes

In this section, we will take a look at TLS in kubernetes

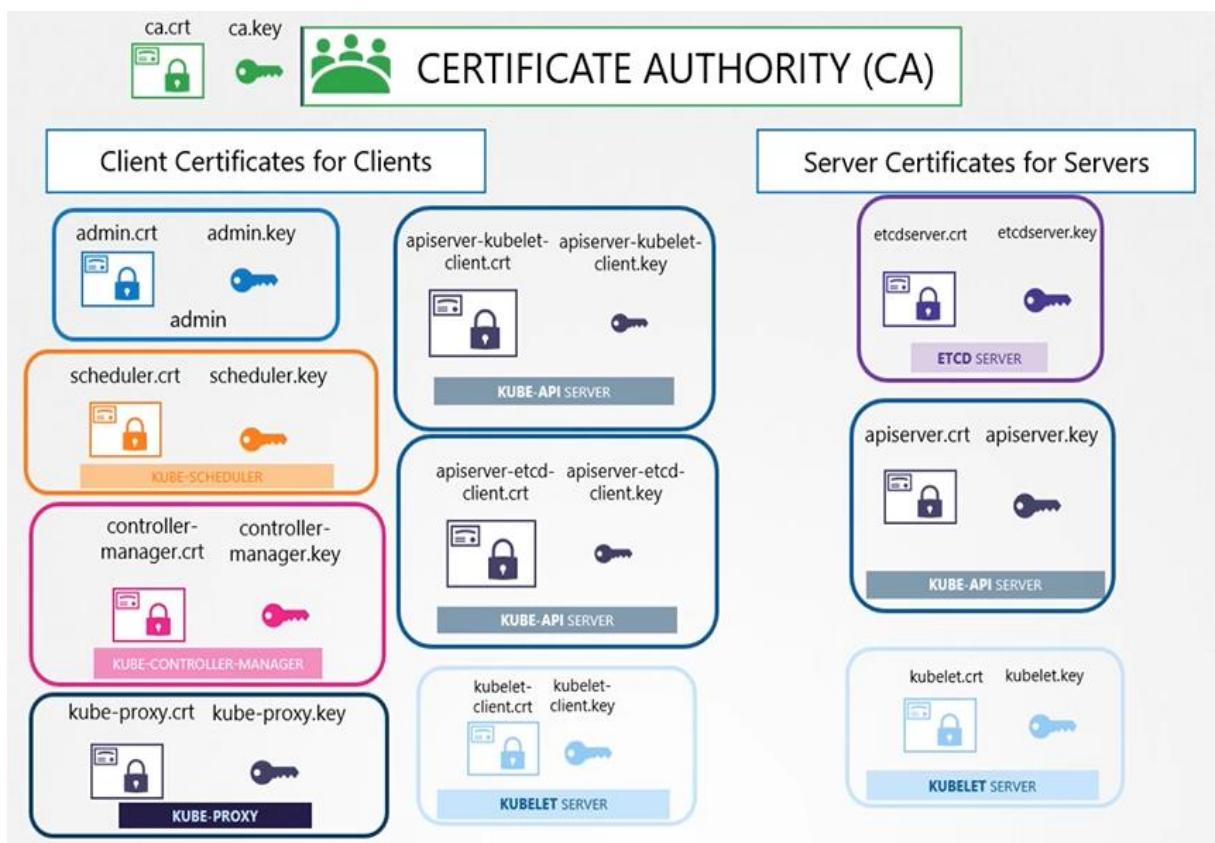
The two primary requirements are to have all the various services within the cluster to use server certificates and all clients to use client certificates to verify they are who they say they are.

- Server Certificates for Servers
- Client Certificates for Clients

ITLS Certificates



Let's look at the different components within the k8s cluster and identify the various servers and clients and who talks to whom.



TLS in kubernetes - Certificate Creation

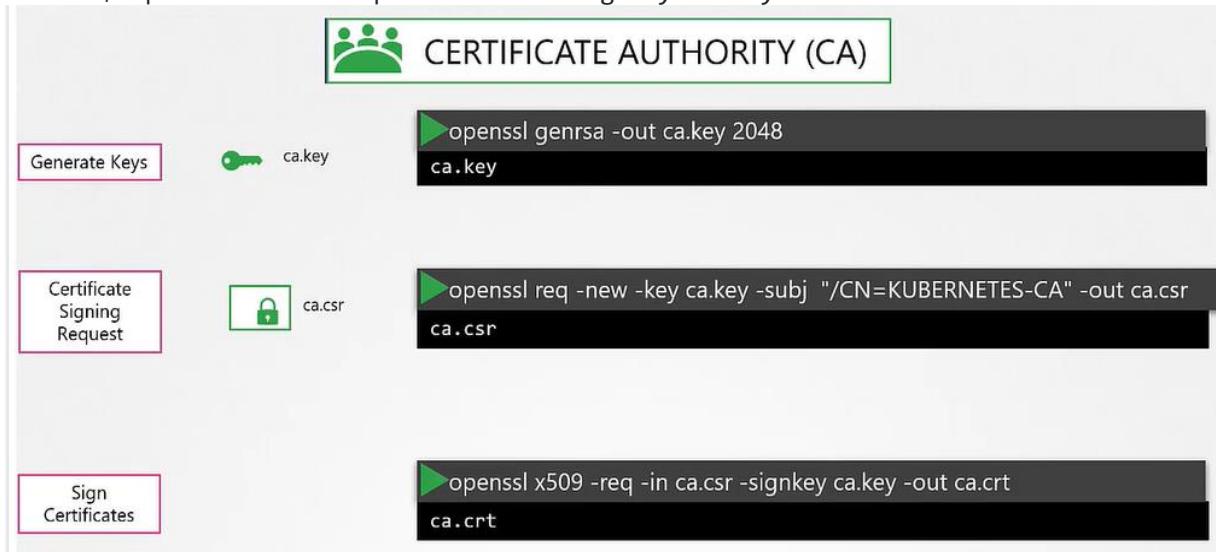
In this section, we will take a look at TLS certificate creation in kubernetes

Generate Certificates

- There are different tools available such as easyrsa, openssl or cfssl etc. or many others for generating certificates.

Certificate Authority (CA)

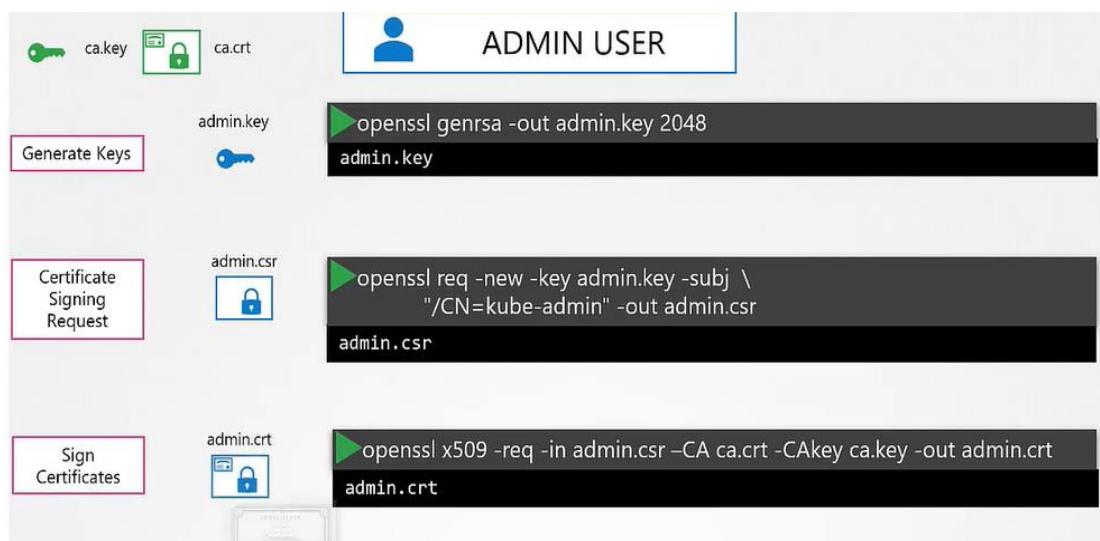
- Generate Keys
- \$ openssl genrsa -out ca.key 2048
- Generate CSR
- \$ openssl req -new -key ca.key -subj "/CN=KUBERNETES-CA" -out ca.csr
- Sign certificates
- \$ openssl x509 -req -in ca.csr -signkey ca.key -out ca.crt



Generating Client Certificates

Admin User Certificates

- Generate Keys
- \$ openssl genrsa -out admin.key 2048
- Generate CSR
- \$ openssl req -new -key admin.key -subj "/CN=kube-admin" -out admin.csr
- Sign certificates
- \$ openssl x509 -req -in admin.csr -CA ca.crt -CAkey ca.key -out admin.crt



- Certificate with admin privileges
- `$ openssl req -new -key admin.key -subj "/CN=kube-admin/O=system:masters" -out admin.csr`

We follow the same procedure to generate client certificate for all other components that access the kube-apiserver.



KUBE CONTROLLER MANGER

ca.key ca.crt

Generate Keys

controller-manager.key

Certificate Signing Request

controller-manager.csr

Sign Certificates

controller-manager.crt

CERTIFICATE

This Certificate Proudly Presented to
SYSTEMKUBE-CONTROLLER-MANGER

NEW YORK
NY, US

Issued by:

KUBE PROXY

ca.key ca.crt

Generate Keys

kube-proxy.key

Certificate Signing Request

kube-proxy.csr

Sign Certificates

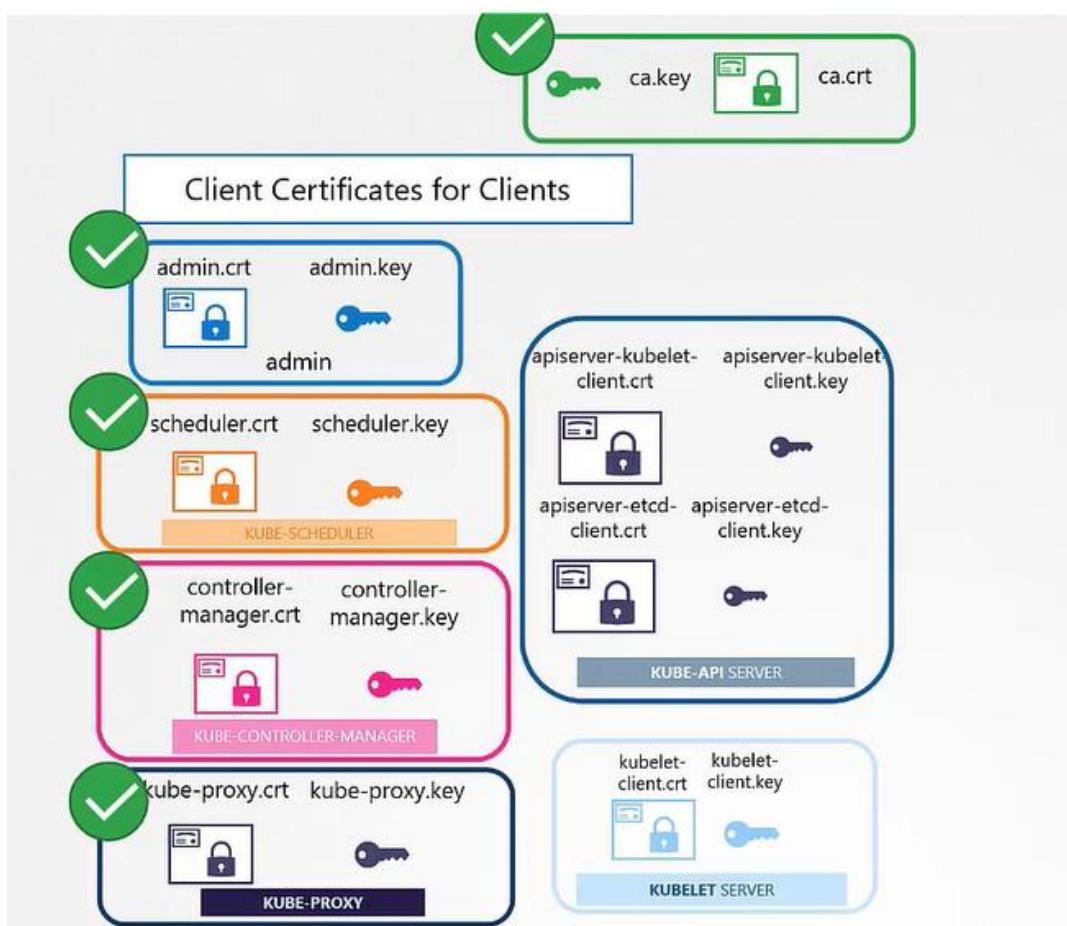
kube-proxy.crt

CERTIFICATE

This Certificate Proudly Presented to
KUBE-PROXY

NEW YORK
NY, US

Issued by:



Generating Server Certificates

ETCD Server certificate



Kube-apiserver certificate

KUBE API SERVER

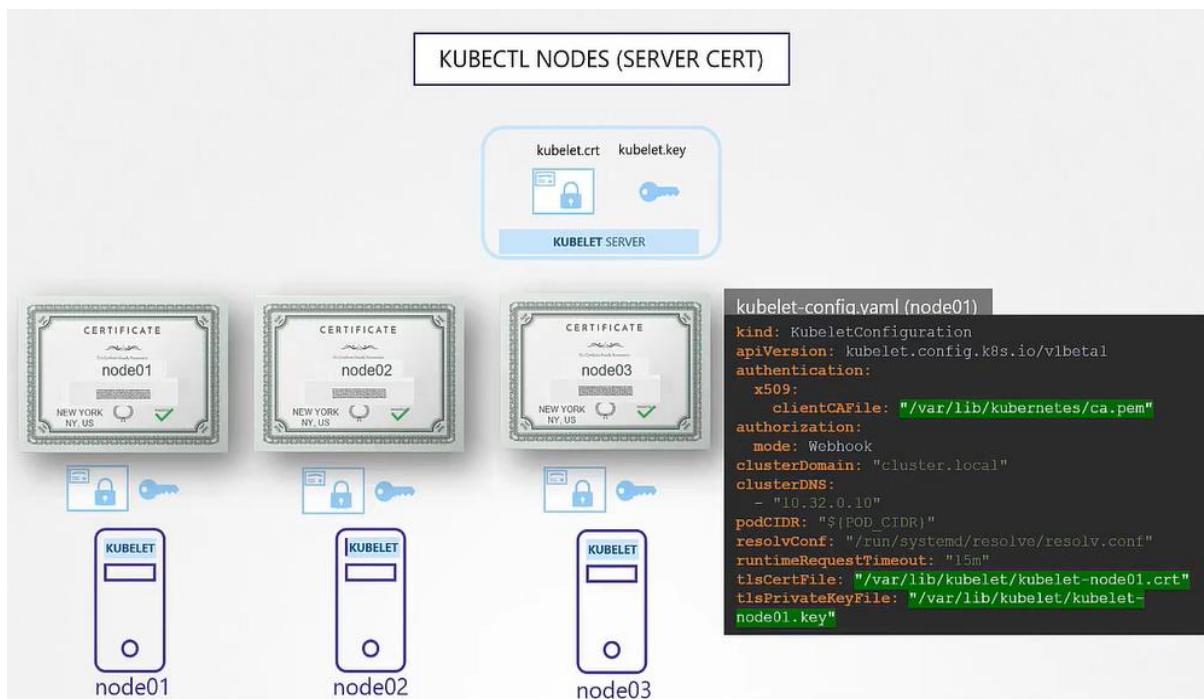
```
openssl req -new -key apiserver.key -subj "/CN=kube-apiserver" -out apiserver.csr -config openssl.cnf
openssl.cnf
[req]
req_extensions = v3_req
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, subjectAltName = @alt_names
subjectAltName = @alt_names
DNS.1 = kubernetes
DNS.2 = kubernetes.default
DNS.3 = kubernetes.default.svc
DNS.4 = kubernetes.default.svc.cluster.local
IP.1 = 10.96.0.1
IP.2 = 172.17.0.87

openssl x509 -req -in apiserver.csr \
-CA ca.crt -CAkey ca.key -out apiserver.crt
apiserver.crt
```

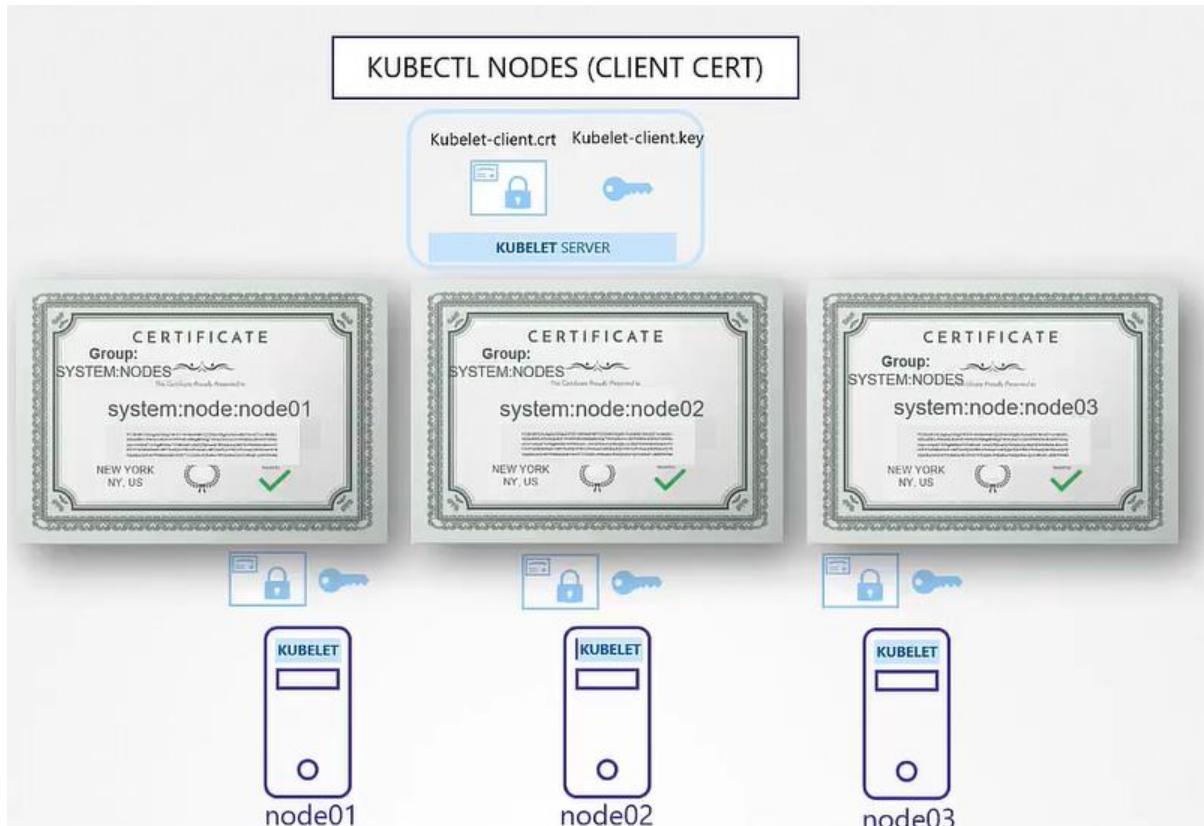
KUBE API SERVER

```
ExecStart=/usr/local/bin/kube-apiserver \
--advertise-address=${INTERNAL_IP} \
--allow-privileged=true \
--apiserver-count=3 \
--authorization-mode=Node,RBAC \
--bind-address=0.0.0.0 \
--enable-swagger-ui=true \
--etcd-cafile=/var/lib/kubernetes/ca.pem \
--etcd-certfile=/var/lib/kubernetes/apiserver-etcd-client.crt \
--etcd-keyfile=/var/lib/kubernetes/apiserver-etcd-client.key \
--etcd-servers=https://127.0.0.1:2379 \
--event-ttl=1h \
--kubelet-certificate-authority=/var/lib/kubernetes/ca.pem \
--kubelet-client-certificate=/var/lib/kubernetes/apiserver-etcd-client.crt \
--kubelet-client-key=/var/lib/kubernetes/apiserver-etcd-client.key \
--kubelet-https=true \
--runtime-config=api/all \
--service-account-key-file=/var/lib/kubernetes/service-account.pem \
--service-cluster-ip-range=10.32.0.0/24 \
--service-node-port-range=30000-32767 \
--client-ca-file=/var/lib/kubernetes/ca.pem \
--tls-cert-file=/var/lib/kubernetes/apiserver.crt \
--tls-private-key-file=/var/lib/kubernetes/apiserver.key \
--v=2
```

Kubectl Nodes (Server Cert)



Kubectl Nodes (Client Cert)



View Certificate Details

In this section, we will take a look how to view certificates in a kubernetes cluster.

View Certs

"The Hard Way"	kubeadm
<pre><code>▶ cat /etc/systemd/system/kube-apiserver.service</code></pre> <pre>[Service] ExecStart=/usr/local/bin/kube-apiserver \ --advertise-address=172.17.0.32 \ --allow-privileged=true \ --apiserver-count=3 \ --authorization-mode=Node,RBAC \ --bind-address=0.0.0.0 \ --client-ca-file=/var/lib/kubernetes/ca.pem \ --enable-swagger-ui=true \ --etcd-cafile=/var/lib/kubernetes/ca.pem \ --etcd-certfile=/var/lib/kubernetes/kubernetes.pem \ --etcd-keyfile=/var/lib/kubernetes/kubernetes-key.pem \ --event-ttl=1h \ --kubelet-certificate-authority=/var/lib/kubernetes/ca.pem \ --kubelet-client-key=/var/lib/kubernetes/kubernetes-key.pem \ --kubelet-https=true \ --service-node-port-range=30000-32767 \ --tls-cert-file=/var/lib/kubernetes/kubernetes.pem \ --tls-private-key-file=/var/lib/kubernetes/kubernetes-key.pem --v=2</pre>	<pre><code>▶ cat /etc/kubernetes/manifests/kube-apiserver.yaml</code></pre> <pre>spec: 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,HostId - --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</pre>

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

spec:
  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
        - --secure-port=6443
        - --service-account-key-file=/etc/kubernetes/pki/sa.pub
        - --service-cluster-ip-range=10.96.0.0/12
        - --tls-cert-file=/etc/kubernetes/pki/apiserver.crt
        - --tls-private-key-file=/etc/kubernetes/pki/apiserver.key
```

- To view the details of the certificate
- `$ openssl x509 -in /etc/kubernetes/pki/apiserver.crt -text -noout`

```
/etc/kubernetes/pki/apiserver.crt

▶ openssl x509 -in /etc/kubernetes/pki/apiserver.crt -text -noout
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number: 3147495682089747350 (0x2bae26a58f090396)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=kubernetes
    Validity
        Not Before: Feb 11 05:39:19 2019 GMT
        Not After : Feb 11 05:39:20 2020 GMT
    Subject: CN=kube-apiserver
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
                Modulus:
                    00:d9:69:38:80:68:3b:b7:2e:9e:25:00:e8:fd:01:
                    Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Key Usage: critical
            Digital Signature, Key Encipherment
        X509v3 Extended Key Usage:
            TLS Web Server Authentication
        X509v3 Subject Alternative Name:
            DNS:master, DNS:kubernetes, DNS:kubernetes.default,
            DNS:kubernetes.default.svc, DNS:kubernetes.default.svc.cluster.local, IP
            Address:10.96.0.1, IP Address:172.17.0.27
```

Follow the same procedure to identify information about of all the other certificates

Certificate Path	CN Name	ALT Names	Organization	Issuer	Expiration
/etc/kubernetes/pki/apiserver.crt	kube-apiserver	DNS:master DNS:kubernetes DNS:kubernetes.default DNS:kubernetes.default.svc IP Address:10.96.0.1 IP Address:172.17.0.27		kubernetes	Feb 11 05:39:20 2020
/etc/kubernetes/pki/apiserver.key					
/etc/kubernetes/pki/ca.crt	kubernetes			kubernetes	Feb 8 05:39:19 2029
/etc/kubernetes/pki/apiserver-kubelet-client.crt	kube-apiserver-kubelet-client		system:masters	kubernetes	Feb 11 05:39:20 2020
/etc/kubernetes/pki/apiserver-kubelet-client.key					
/etc/kubernetes/pki/apiserver-etcd-client.crt	kube-apiserver-etcd-client		system:masters	self	Feb 11 05:39:22 2020
/etc/kubernetes/pki/apiserver-etcd-client.key					
/etc/kubernetes/pki/etcd/ca.crt	kubernetes			kubernetes	Feb 8 05:39:21 2017

Inspect Server Logs - Hardware setup

- Inspect server logs using journalctl
- \$ journalctl -u etcd.service -l

Inspect Service Logs

```
▶ journalctl -u etcd.service -l
2019-02-13 02:53:28.144631 I | etcdmain: etcd Version: 3.2.18
2019-02-13 02:53:28.144680 I | etcdmain: Git SHA: eddf599c6
2019-02-13 02:53:28.144684 I | etcdmain: Go Version: go1.8.7
2019-02-13 02:53:28.144688 I | etcdmain: Go OS/Arch: linux/amd64
2019-02-13 02:53:28.144692 I | etcdmain: setting maximum number of CPUs to 4, total number of available CPUs is 4
2019-02-13 02:53:28.144734 N | etcdmain: the server is already initialized as member before, starting as etcd
member...
2019-02-13 02:53:28.146625 I | etcdserver: name = master
2019-02-13 02:53:28.146637 I | etcdserver: data dir = /var/lib/etcd
2019-02-13 02:53:28.146642 I | etcdserver: member dir = /var/lib/etcd/member
2019-02-13 02:53:28.146645 I | etcdserver: heartbeat = 100ms
2019-02-13 02:53:28.146648 I | etcdserver: election = 1000ms
2019-02-13 02:53:28.146651 I | etcdserver: snapshot count = 10000
2019-02-13 02:53:28.146677 I | etcdserver: advertise client URLs = 2019-02-13 02:53:28.185353 I | etcdserver/api:
enabled capabilities for version 3.2
2019-02-13 02:53:28.185588 I | embed: ClientTLS: cert = /etc/kubernetes/pki/etcd/server.crt, key =
/etc/kubernetes/pki/etcd/server.key, ca = , trusted-ca = /etc/kubernetes/pki/etcd/old-ca.crt, client-cert-auth =
true
2019-02-13 02:53:30.080017 I | embed: ready to serve client requests
2019-02-13 02:53:30.080130 I | etcdserver: published {Name:master ClientURLs:[https://127.0.0.1:2379]} to cluster
c9be114fc2da2776
2019-02-13 02:53:30.080281 I | embed: serving client requests on 127.0.0.1:2379
WARNING: 2019/02/13 02:53:30 Failed to dial 127.0.0.1:2379: connection error: desc = "transport: authentication
handshake failed: remote error: tls: bad certificate"; please retry.
```

Inspect Server Logs - kubeadm setup

- View logs using kubectl
- \$ kubectl logs etcd-master

View Logs

```
▶ kubectl logs etcd-master
2019-02-13 02:53:28.144631 I | etcdmain: etcd Version: 3.2.18
2019-02-13 02:53:28.144680 I | etcdmain: Git SHA: eddf599c6
2019-02-13 02:53:28.144684 I | etcdmain: Go Version: go1.8.7
2019-02-13 02:53:28.144688 I | etcdmain: Go OS/Arch: linux/amd64
2019-02-13 02:53:28.144692 I | etcdmain: setting maximum number of CPUs to 4, total number of available CPUs is 4
2019-02-13 02:53:28.144734 N | etcdmain: the server is already initialized as member before, starting as etcd
member...
2019-02-13 02:53:28.146625 I | etcdserver: name = master
2019-02-13 02:53:28.146637 I | etcdserver: data dir = /var/lib/etcd
2019-02-13 02:53:28.146642 I | etcdserver: member dir = /var/lib/etcd/member
2019-02-13 02:53:28.146645 I | etcdserver: heartbeat = 100ms
2019-02-13 02:53:28.146648 I | etcdserver: election = 1000ms
2019-02-13 02:53:28.146651 I | etcdserver: snapshot count = 10000
2019-02-13 02:53:28.146677 I | etcdserver: advertise client URLs = 2019-02-13 02:53:28.185353 I | etcdserver/api:
enabled capabilities for version 3.2
2019-02-13 02:53:28.185588 I | embed: ClientTLS: cert = /etc/kubernetes/pki/etcd/server.crt, key =
/etc/kubernetes/pki/etcd/server.key, ca = , trusted-ca = /etc/kubernetes/pki/etcd/old-ca.crt, client-cert-auth =
true
2019-02-13 02:53:30.080017 I | embed: ready to serve client requests
2019-02-13 02:53:30.080130 I | etcdserver: published {Name:master ClientURLs:[https://127.0.0.1:2379]} to cluster
c9be114fc2da2776
2019-02-13 02:53:30.080281 I | embed: serving client requests on 127.0.0.1:2379
WARNING: 2019/02/13 02:53:30 Failed to dial 127.0.0.1:2379: connection error: desc = "transport: authentication
handshake failed: remote error: tls: bad certificate"; please retry.
```

- View logs using docker ps and docker logs
- \$ docker ps -a
- \$ docker logs <container-id>

IView Logs

```
▶ docker logs 87fc
2019-02-13 02:53:28.144631 I | etcdmain: etcd Version: 3.2.18
2019-02-13 02:53:28.144680 I | etcdmain: Git SHA: eddf599c6
2019-02-13 02:53:28.144684 I | etcdmain: Go Version: go1.8.7
2019-02-13 02:53:28.144688 I | etcdmain: Go OS/Arch: linux/amd64
2019-02-13 02:53:28.144692 I | etcdmain: setting maximum number of CPUs to 4, total number of available CPUs is 4
2019-02-13 02:53:28.144734 N | etcdmain: the server is already initialized as member before, starting as etcd
member...
2019-02-13 02:53:28.146625 I | etcdserver: name = master
2019-02-13 02:53:28.146637 I | etcdserver: data dir = /var/lib/etcd
2019-02-13 02:53:28.146642 I | etcdserver: member dir = /var/lib/etcd/member
2019-02-13 02:53:28.146645 I | etcdserver: heartbeat = 100ms
2019-02-13 02:53:28.146648 I | etcdserver: election = 1000ms
2019-02-13 02:53:28.146651 I | etcdserver: snapshot count = 10000
2019-02-13 02:53:28.146677 I | etcdserver: advertise client URLs = 2019-02-13 02:53:28.185353 I | etcdserver/api: enabled capabilities for version 3.2
2019-02-13 02:53:28.185588 I | embed: ClientTLS: cert = /etc/kubernetes/pki/etcd/server.crt, key = /etc/kubernetes/pki/etcd/server.key, ca = , trusted-ca = /etc/kubernetes/pki/etcd/old-ca.crt, client-cert-auth = true
2019-02-13 02:53:30.080017 I | embed: ready to serve client requests
2019-02-13 02:53:30.080130 I | etcdserver: published {Name:master ClientURLs:[https://127.0.0.1:2379]} to cluster c9be114fc2da2776
2019-02-13 02:53:30.080281 I | embed: serving client requests on 127.0.0.1:2379
WARNING: 2019/02/13 02:53:30 Failed to dial 127.0.0.1:2379: connection error: desc = "transport: authentication handshake failed: remote error: tls: bad certificate"; please retry.
```

K8s Reference Docs

- <https://kubernetes.io/docs/setup/best-practices/certificates/#certificate-paths>

Certificate API

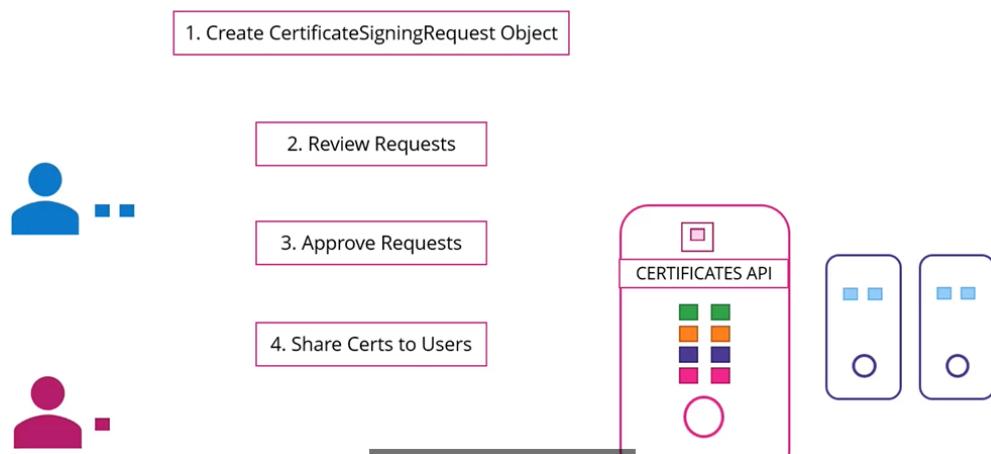
In this section, we will take a look at how to manage certificates and certificate API's in kubernetes

CA (Certificate Authority)

- The CA is really just the pair of key and certificate files that we have generated, whoever gains access to these pair of files can sign any certificate for the kubernetes environment.

Kubernetes has a built-in certificates API that can do this for you.

- With the certificate API, we now send a certificate signing request (CSR) directly to kubernetes through an API call.

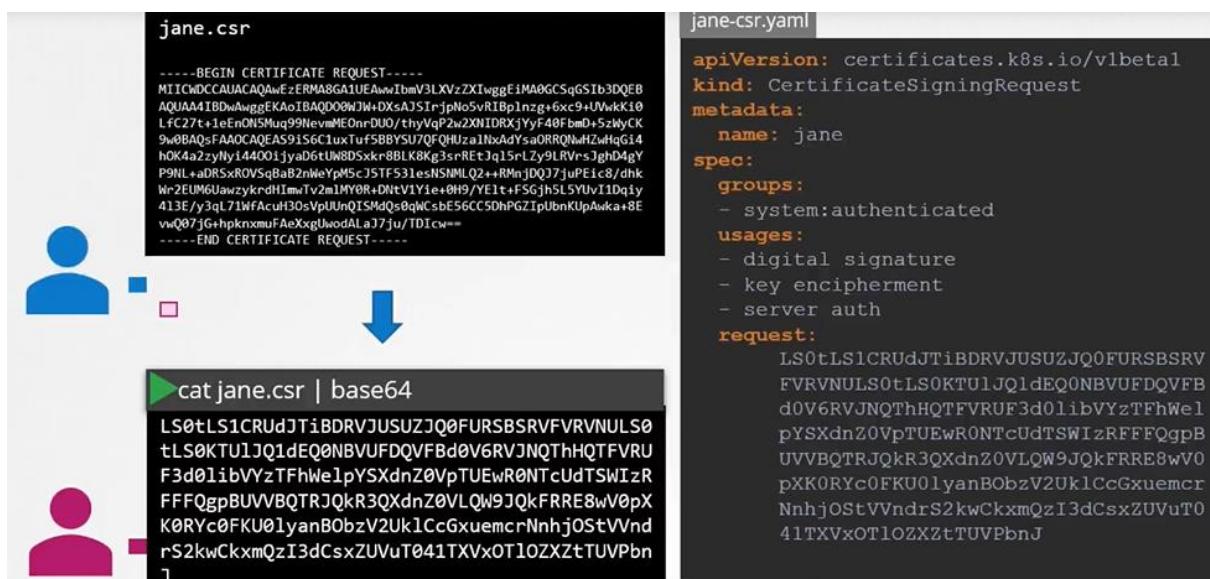


This certificate can then be extracted and shared with the user.

- A user first creates a key
 - `$ openssl genrsa -out jane.key 2048`
 - Generates a CSR
 - `$ openssl req -new -key jane.key -subj "/CN=jane" -out jane.csr`
 - Sends the request to the administrator and the administrator takes the key and creates a CSR object, with kind as "CertificateSigningRequest" and a encoded "jane.csr"
 - `apiVersion: certificates.k8s.io/v1beta1`
 - `kind: CertificateSigningRequest`
 - `metadata:`
 - `name: jane`
 - `spec:`
 - `groups:`
 - `- system:authenticated`
 - `usages:`
 - `- digital signature`
 - `- key encipherment`
 - `- server auth`
 - `request:`
 - `<certificate-goes-here>`
- ```

$ cat jane.csr |base64 $ kubectl create -f jane.yaml

```



- To list the csr's
- `$ kubectl get csr`
- Approve the request
- `$ kubectl certificate approve jane`
- To view the certificate
- `$ kubectl get csr jane -o yaml`
- To decode it
- `$ echo "<certificate>" |base64 --decode`

The diagram shows two terminal windows. The left window displays the YAML output of `kubectl get csr jane -o yaml`, and the right window shows the decoded content of the certificate using `echo "LS0...Qo=" | base64 --decode`.

```
kubectl get csr jane -o yaml
apiVersion: certificates.k8s.io/v1beta1
kind: CertificateSigningRequest
metadata:
 creationTimestamp: 2019-02-13T16:36:43Z
 name: new-user
spec:
 groups:
 - system:masters
 - system:authenticated
 usages:
 - digital signature
 - key encipherment
 - server auth
 username: kubernetes-admin
status:
 certificate:
 LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURDaKNDQWZLZ0f3SUJBZ01VRmwY
 Q2wxYXoxaNI5M3JNVisreFRYQJouw3nd0RRWUpLb1pJaHzjTkFRRUwKQ1FBd0ZURVRN
 QkVhQTFRUF4TUthM1ZpWhkdvpYumxjekF1rncweE9UQ0X1NVE14TmpNeU1QmFgd1n
 Y0ZFd12ajnusXY3efdDS1NIRm5sU01c8t5Z0wXlkzwTFM5V29Ge1hHZDdwCmIEZ2F0
 MVVRMF8XTVjhN09FVnVjsWc1YK4weEVHTkVwRU5tdU1BN1ZwelHVjs1hGag1dY0MEd1
 MGU0YXFkWV1KwMVMbjBvRTFCY3dod2xic0I1ND0KLS0tL51FTk0gQ0VSE1GSUNBVEmt
 LS0tLQo=
 conditions:
 - lastUpdateTime: 2019-02-13T16:37:21Z
 message: This CSR was approved by kubectl certificate approve.
 reason: KubectlApprove
 type: Approved
```

```
echo "LS0...Qo=" | base64 --decode
-----BEGIN CERTIFICATE -----
MIICWDCCAUACAQAwEzERMA8GA1UEAwIBmV3LXvzZXIwggEiMA0GCSqGSIb3DQE8
AQUAA4IBDwAwggEKAoIBAQD0WJW+DXsAJSIrjpNo5vRIBpInzg+6x9+Uwkk10
Lfc27t+1eNnONSMuq99NevmMEOnrDUO/thyVqP2w2XNIDRjYyF40fbmD5zWyCK
9w0BAqsFAAOCAQEAS9iS6C1uxTuf5BBSYU7QFQHuzalNxAdYsaORRQnH2wHqG14
hOK4a2zyNy144001jyaD6tUW8DSxr8BLK8Kg3srREtJq15rLzy9LRVsJghDgY
P9NL+aDRSxROVSqBaB2nWeYpM5cJ5TF31esNSNMLQ2++RMnJDQ7juPe1c8/dhk
Wr2EUM6UawzykrdfHimwTv2m1MY0R+DtV1Yie+0H9/YEl+FSGjhLS5UViIDqiy
413E/y3ql71kfAcuh3OsVpUUnQISMdQs0qWCsbE56CC5DhPGZTpUbnKUpAwka+E
vwQ07jG+hpknxmuFAeXxgUwodALaJ7ju/TDIcw==
-----END CERTIFICATE -----
```

All the certificate related operations are carried out by the controller manager.

- If anyone has to sign the certificates they need the CA Servers, route certificate and private key. The controller manager configuration has two options where you can specify this.



```
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
```

K8s Reference Docs

- <https://kubernetes.io/docs/reference/access-authn-authz/certificate-signing-requests/>
- <https://kubernetes.io/docs/tasks/tls/managing-tls-in-a-cluster/>

## KubeConfig

In this section, we will take a look at kubeconfig in kubernetes

*Client uses the certificate file and key to query the kubernetes Rest API for a list of pods using curl.*

- You can specify the same using kubectl



```
curl https://my-kube-playground:6443/api/v1/pods \
--key admin.key
--cert admin.crt
--cacert ca.crt

{
 "kind": "PodList",
 "apiVersion": "v1",
 "metadata": {
 "selfLink": "/api/v1/pods",
 },
 "items": []
}

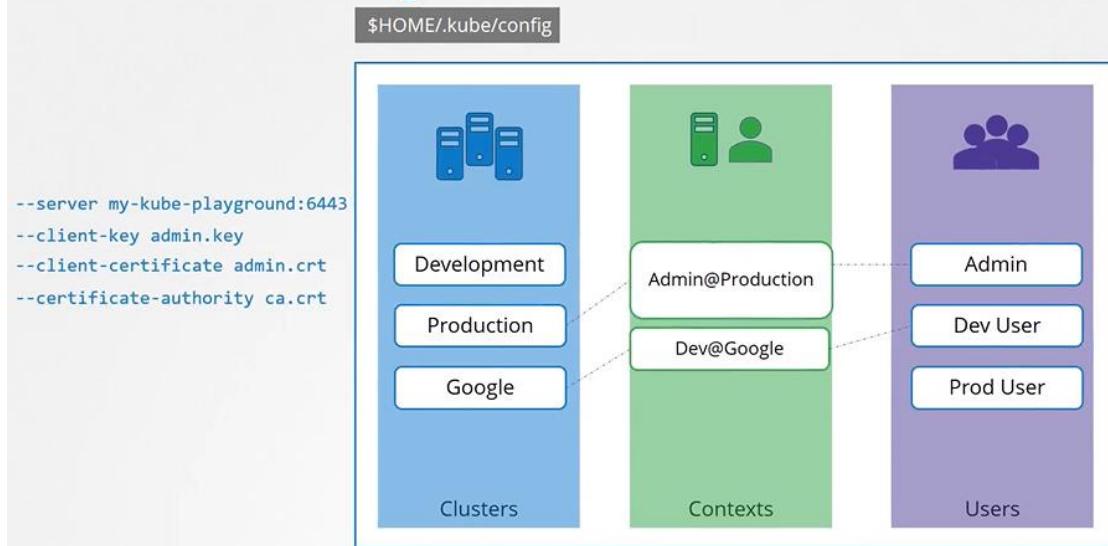
▶ kubectl get pods
--server my-kube-playground:6443
--client-key admin.key
--client-certificate admin.crt
--certificate-authority ca.crt
No resources found.
```

- We can move these information to a configuration file called kubeconfig. And the specify this file as the kubeconfig option in the command.
- \$ kubectl get pods --kubeconfig config

## Kubeconfig File

- The kubeconfig file has 3 sections
  - Clusters
  - Contexts
  - USers

# KubeConfig File



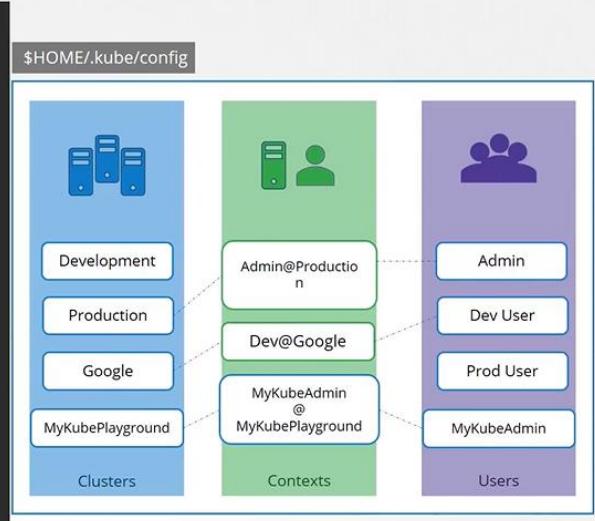
# KubeConfig File

```
apiVersion: v1
kind: Config
current-context: dev-user@google

clusters:
- name: my-kube-playground (values hidden...)
- name: development
- name: production
- name: google

contexts:
- name: my-kube-admin@my-kube-playground
- name: dev-user@google
- name: prod-user@production

users:
- name: my-kube-admin
- name: admin
- name: dev-user
- name: prod-user
```



- To view the current file being used
- \$ kubectl config view
- You can specify the kubeconfig file with kubectl config view with "--kubeconfig" flag
- \$ kubectl config view --kubeconfig=my-custom-config

## Kubectl config

```
▶ kubectl config view
apiVersion: v1

kind: Config
current-context: kubernetes-admin@kubernetes

clusters:
- cluster:
 certificate-authority-data: REDACTED
 server: https://172.17.0.5:6443
 name: kubernetes

contexts:
- context:
 cluster: kubernetes
 user: kubernetes-admin
 name: kubernetes-admin@kubernetes

users:
- name: kubernetes-admin
 user:
 client-certificate-data: REDACTED
 client-key-data: REDACTED
```

```
▶ kubectl config view --kubeconfig=my-custom-config
apiVersion: v1

kind: Config
current-context: my-kube-admin@my-kube-playground

clusters:
- name: my-kube-playground
- name: development
- name: production

contexts:
- name: my-kube-admin@my-kube-playground
- Name: prod-user@production

users:
- name: my-kube-admin
- name: prod-user
```

- How do you update your current context? Or change the current context
- \$ kubectl config view --kubeconfig=my-custom-config

## Kubectl config

```
▶ kubectl config view
apiVersion: v1

kind: Config
current-context: my-kube-admin@my-kube-playground

clusters:
- name: my-kube-playground
- name: development
- name: production

contexts:
- name: my-kube-admin@my-kube-playground
- Name: prod-user@production

users:
- name: my-kube-admin
- name: prod-user
```

```
▶ kubectl config use-context prod-user@production
apiVersion: v1

kind: Config
current-context: prod-user@production

clusters:
- name: my-kube-playground
- name: development
- name: production

contexts:
- name: my-kube-admin@my-kube-playground
- Name: prod-user@production

users:
- name: my-kube-admin
- name: prod-user
```

- kubectl config help
- \$ kubectl config -h

# I Kubectl config

```
kubectl config -h
Available Commands:
 current-context Displays the current-context
 delete-cluster Delete the specified cluster from the kubeconfig
 delete-context Delete the specified context from the kubeconfig
 get-clusters Display clusters defined in the kubeconfig
 get-contexts Describe one or many contexts
 rename-context Renames a context from the kubeconfig file.
 set Sets an individual value in a kubeconfig file
 set-cluster Sets a cluster entry in kubeconfig
 set-context Sets a context entry in kubeconfig
 set-credentials Sets a user entry in kubeconfig
 unset Unsets an individual value in a kubeconfig file
 use-context Sets the current-context in a kubeconfig file
 view Display merged kubeconfig settings or a specified kubeconfig file
```

What about namespaces?

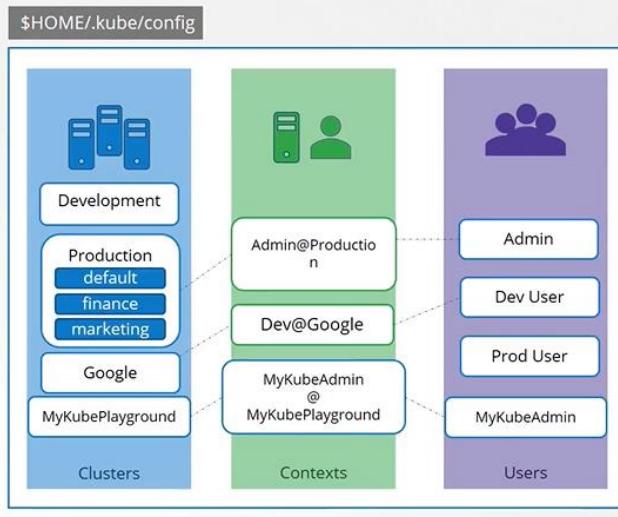
## Namespaces

```
apiVersion: v1
kind: Config

clusters:
- name: production
 cluster:
 certificate-authority: ca.crt
 server: https://172.17.0.51:6443

contexts:
- name: admin@production
 context:
 cluster: production
 user: admin
 namespace: finance

users:
- name: admin
 user:
 client-certificate: admin.crt
 client-key: admin.key
```



## Certificates in kubeconfig

# Certificates in KubeConfig

```

apiVersion: v1
kind: Config

clusters:
- name: production
 cluster:
 certificate-authority: ca.crt
 server: https://172.17.0.51:6443

contexts:
- name: admin@production
 context:
 cluster: production
 user: admin
 namespace: finance

users:
- name: admin
 user:
 client-certificate: admin.crt
 client-key: admin.key

```

# Certificates in KubeConfig

```

apiVersion: v1
kind: Config

clusters:
- name: production
 cluster:
 certificate-authority: /etc/kubernetes/pki/ca.crt
 certificate-authority-data:

```

```

-----BEGIN CERTIFICATE-----
MIICWDCCAUACAQAwEzERMA8GA1UEAwIbmV3LXVzZXIxwggEiMA0G
AQUAA4IBDwAwggEKAoIBAQD00WJW+DxsAJSInjnN05vRIBpInzg+e
Lfc27t+iEeN0NSMu99NevmMEOnrDUO/thyqP2w2XNIDRxjyF4d
y3Bihh893Mj70q13UTvZ8TELqyaDknR1/jv/Sxgxkok0ABUTpWmx
IF5nxAttMVKDPQ7NbeZR643b+Qw1VGR/z6DWOfJnbfezoTaAydgL
EcCXawqChjBLkzzBHP4J89D6x8k39pu6jpyngv6uP0tzb0zpgNv
j2qEl+hZEwkKfz0lNtyT5LxMgENDnCnIgwC4GzirGbrAgmAAGg
9w0BAQsfAAOCQAfES9iS6C1uxTuF5BYSU7QFQHuzalNxAdYsaORF
hOK4a2zyNyia400ijya6tUW8DSxkr8BLK8Kg3srRetjq15rLzy9t
P9NL+ADRSxROVSGBaB2nWeYpM5cJ5TF53lesNSMNLQ2+RMnjDQJ7
Wr2EUM6awzykrdHxmWtV2mIMYOR+DNTV1Yie+OH9/Yelt+FSGjh
413E/y3qL71wfAcuH3OsVpUUnQTSMDQs0qQCsbe56CC5DhpGZIpUt
vwQ07jG+hpknxmuFAeXxguwodAlaJ7ju/TD1cw==
-----END CERTIFICATE-----

```

```

cat ca.crt | base64
LS0tLS1CRUdjTiBDRVJUSUZJQ0FURSBDRVFRVN
tLS0KTU1JQ1dEQ0NBVUFQVF8d0V6RVJNQThHQ
F3d0libVYzTFhWelpYSXdnZ0VpTUEWR0NTcUdTS
FFFQgpBUUVBQTRJQkR3QXdnZ0VLQW9JQkFRRE8w
K0RYc0FKU01yanB0bzV2UklCcGxuemcrNnhjOST
rS2kwCkxmQzI3dCsxZUVuT041TXvxOTl0ZXZtTU

```

# Certificates in KubeConfig

```
apiVersion: v1
kind: Config

clusters:
- name: production
 cluster:
 certificate-authority: /etc/kubernetes/pki/ca.crt

 certificate-authority-data: LS0tLS1CRUDJTiBDRVJU
SUZJQ0FURSBDRVVRVNULS0tLS0KTU1J
Q1dEQ0NBVUDQVFbd0V6RVJNQThHQTfV
RUF3d0l1bVYzTFhWeIpYSXdnZ0VpTUEw
R0NTcUdTSW1zRFFFQgpBUVVBQTRJQkR3
QXdznZ0VLQW9JQkFRRE8wV0pXKORYc0FK
U0lyanBoBzV2Uk1CcGxuemcrNnhjOSTv
VndrS2kwCkxmQzI3dCsxZUVuT041TXVx
OT1oZXztTUVPbnJ
```

```
echo "LS0...bnJ" | base64 --decode
-----BEGIN CERTIFICATE-----
MIICDCCAUACAQwEzERMA8GA1UEAwIBmV3LXVzzXiWggEiMA0GCS
AQAA4IBDwAwggEKAoIBAQD0WJW+DXsAJSIrjpNo5vRIBp1nzg+6x
Lfc2t+1eFnON5luq99NevmEOnrDUO/thyVgP2w2XNIDRXjVyF40Fb
y3BiuhB93M70q13UTVz8TELyqaDknR1/jv/SgxXkok0ABUTpMx48p
IFNxAttMVkDQ7NbeZRG43b+QhIVGR/Z6DWOfJnbfezotaAyGLTzF
EcXAwgChjBLkz2BHP4J89D6xbK39pu6pyngv6uPo7ibD0zpqnV0Y
j2qEL+hZEWkkFz80LNntyT5LxMgENDCnIgwC4GZiRGbrAgIBAAgGAD
9wBAQsfAAOCQAQEAS9i56C1uxTuf5BVSU7QFQHUzalNxAdysaORRN
hOKa2zyNy14400ijyaD6tUm8DSxxr88LK8Kg3srRETJql5rLz9LRV
P9NL+aDRSXROV5Ba2nWeYpm5C5J5F53lesfSNMLO2++RMnjDQ7ju
Wr2EUMGUawazykrdrHmwTv2m1My8R+DNTV1ie+0H9/yEl+;+FSGjh5L5
413E/y3qL71wfAcuH3OsVpUUnQISMdQs0qWCsbE56CC5DhPGZIpUbnK
vwQ07jG+hpknxmuFaExgUwodLaJ7ju/TDiCw==
-----END CERTIFICATE-----
```

## K8s Reference Docs

- <https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/>
- <https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#config>

# API Groups

In this section, we will take a look at API Groups in kubernetes

To return version and list pods via API's

```
curl https://kube-master:6443/version
{
 "major": "1",
 "minor": "13",
 "gitVersion": "v1.13.0",
 "gitCommit": "ddfd7ac13c1a9483ea035a79cd7c10005ff21a6d",
 "gitTreeState": "clean",
 "buildDate": "2018-12-03T20:56:12Z",
 "goVersion": "go1.11.2",
 "compiler": "gc",
 "platform": "linux/amd64"
}
```

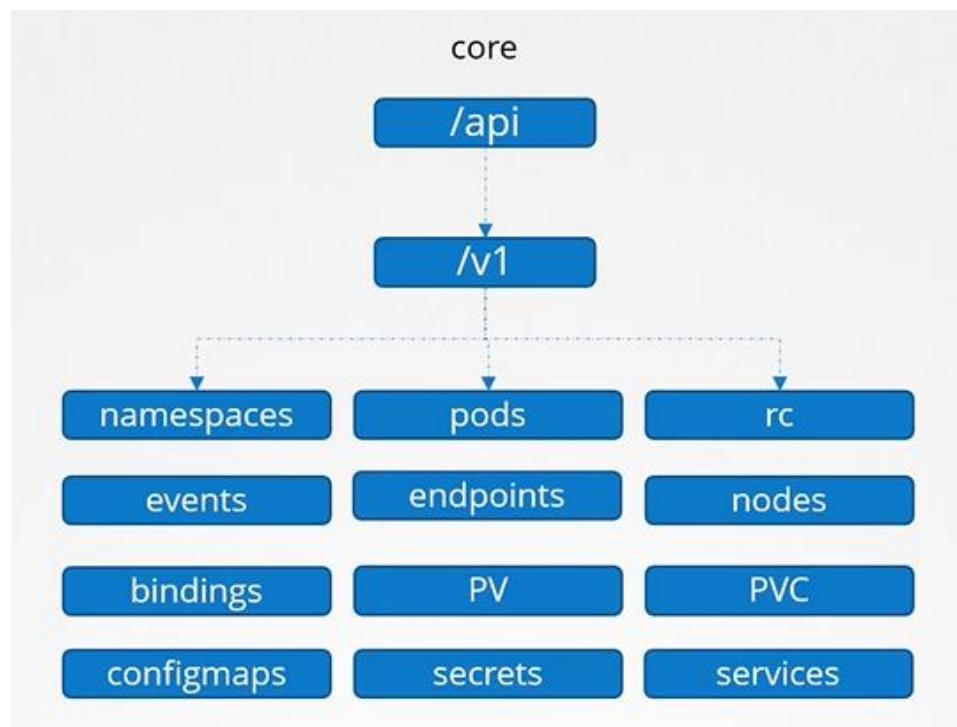
```
curl https://kube-master:6443/api/v1/pods
{
 "kind": "PodList",
 "apiVersion": "v1",
 "metadata": {
 "selfLink": "/api/v1/pods",
 "resourceVersion": "153068"
 },
 "items": [
 {
 "metadata": {
 "name": "nginx-5c7588df-ghsbd",
 "generateName": "nginx-5c7588df-",
 "namespace": "default",
 "creationTimestamp": "2019-03-20T10:57:48Z",
 "labels": {
 "app": "nginx",
 "pod-template-hash": "5c7588df"
 },
 "ownerReferences": [
 {
 "apiVersion": "apps/v1",
 "kind": "ReplicaSet",
 "name": "nginx-5c7588df",
 "uid": "398ce179-4af9-11e9-beb6-020d3114c7a7",
 "controller": true,
 "blockOwnerDeletion": true
 }
],
 "status": {
 "phase": "Running"
 }
 }
 }
]
}
```

- The kubernetes API is grouped into multiple such groups based on their purpose. Such as one for **APIs**, one for **healthz**, **metrics** and **logs** etc.

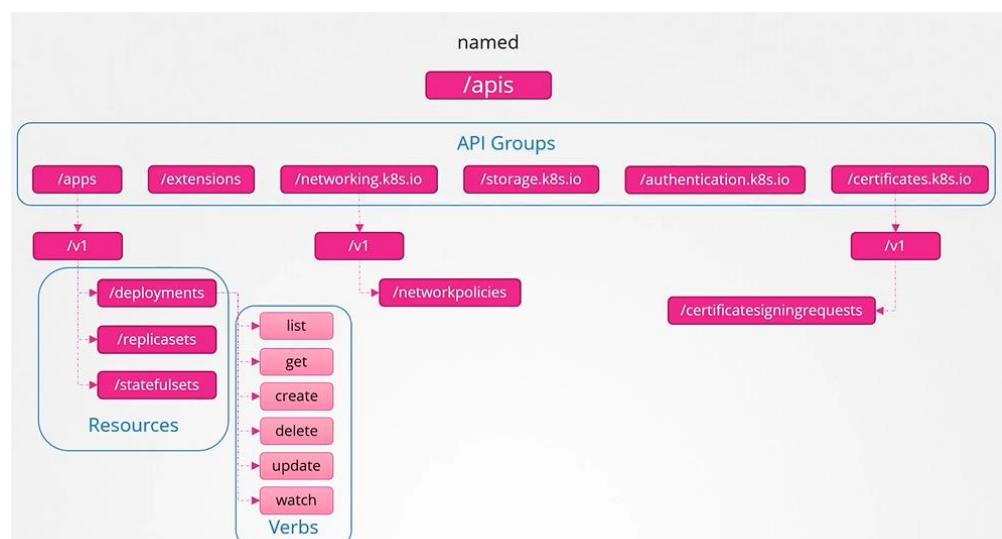


## API and APIs

- These APIs are categorized into two.
  - The core group - Where all the functionality exists



- The Named group - More organized and going forward all the newer features are going to be made available to these named groups.



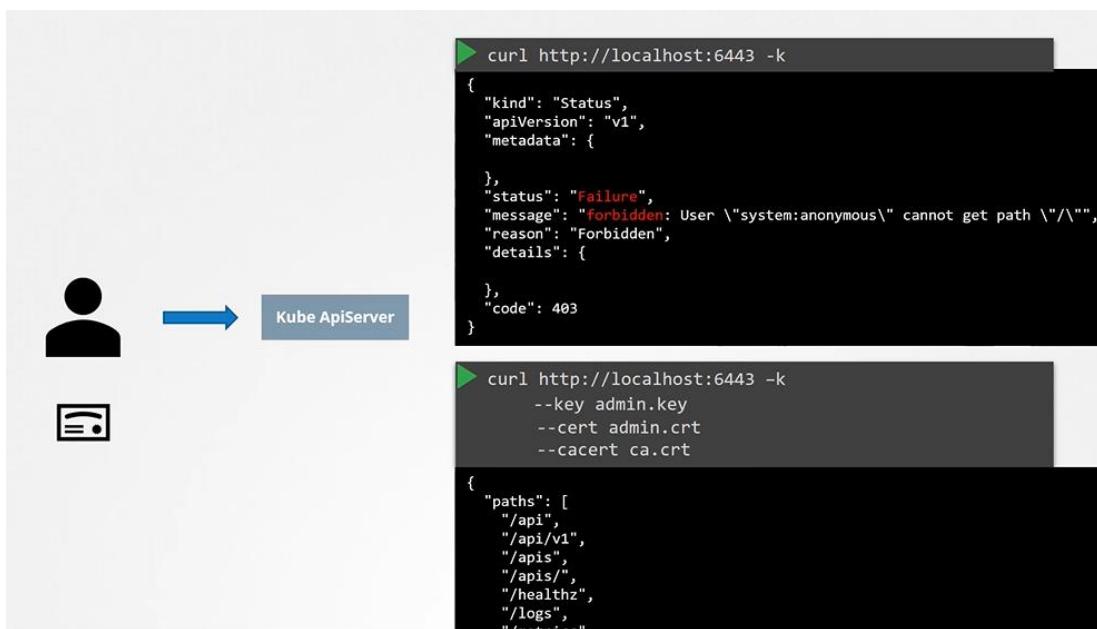
- To list all the api groups

```
▶ curl http://localhost:6443 -k
{
 "paths": [
 "/api",
 "/api/v1",
 "/apis",
 "/apis/",
 "/healthz",
 "/logs",
 "/metrics",
 "/openapi/v2",
 "/swagger-2.0.0.json",
]
}

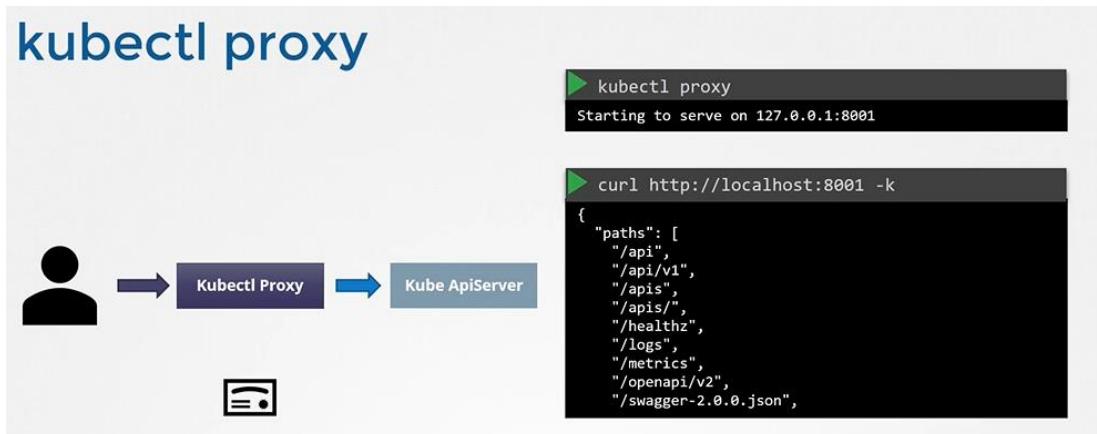
▶ curl http://localhost:6443/apis -k | grep "name"
 "name": "extensions",
 "name": "apps",
 "name": "events.k8s.io",
 "name": "authentication.k8s.io",
 "name": "authorization.k8s.io",
 "name": "autoscaling",
 "name": "batch",
 "name": "certificates.k8s.io",
 "name": "networking.k8s.io",
 "name": "policy",
 "name": "rbac.authorization.k8s.io",
 "name": "storage.k8s.io",
 "name": "admissionregistration.k8s.io",
 "name": "apiextensions.k8s.io",
 "name": "scheduling.k8s.io",
```

Note on accessing the kube-apiserver

- You have to authenticate by passing the certificate files.



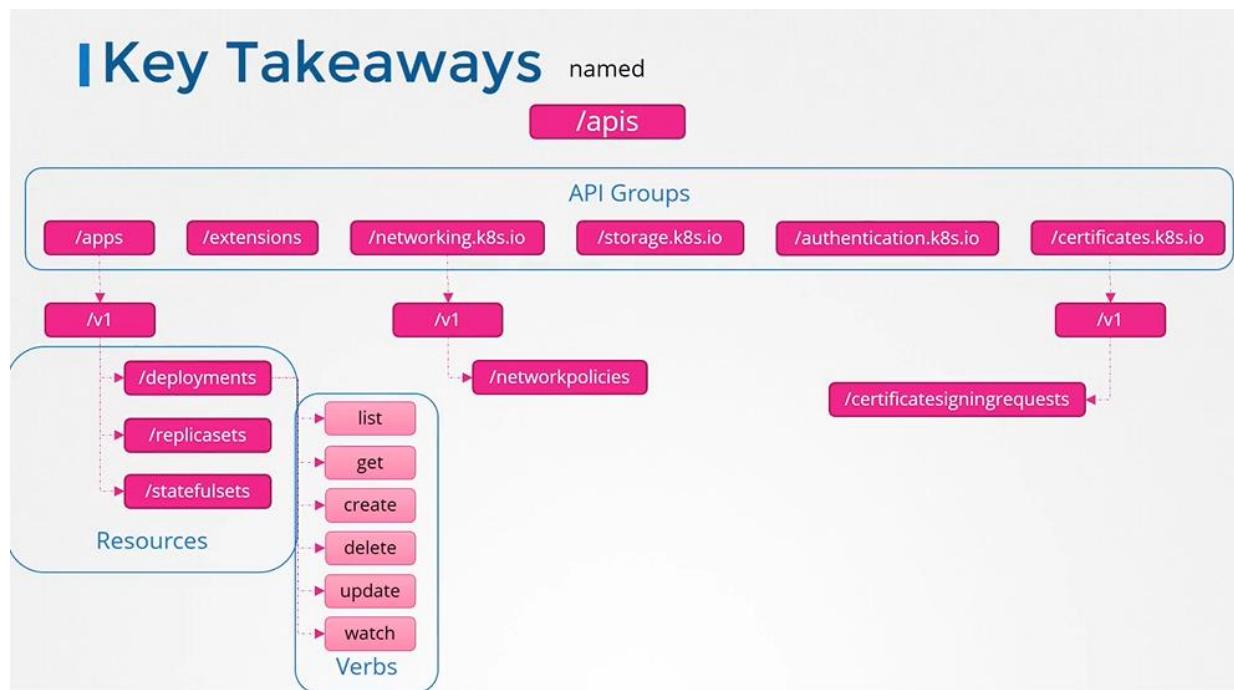
- An alternate is to start a **kubeproxy** client



kube proxy vs kubectl proxy

**Kube proxy**  **Kubectl proxy**

## Key Takeaways



### K8s Reference Docs

- <https://kubernetes.io/docs/concepts/overview/kubernetes-api/>
- <https://kubernetes.io/docs/reference/using-api/api-concepts/>
- <https://kubernetes.io/docs/tasks/extend-kubernetes/http-proxy-access-api/>

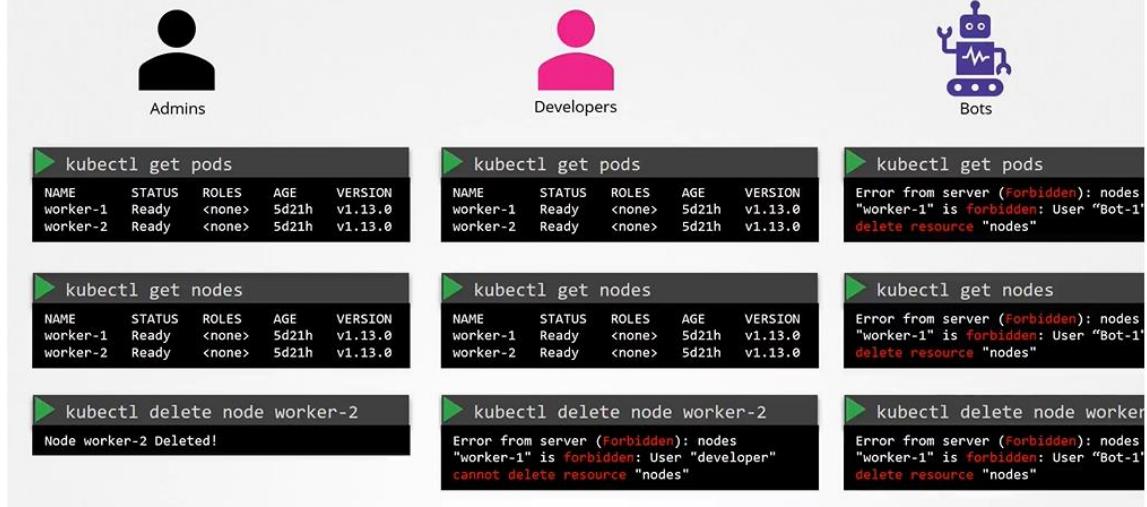
# Authorization

In this section, we will take a look at authorization in kubernetes

Why do you need Authorization in your cluster?

- As an admin, you can do all operations
- `$ kubectl get nodes`
- `$ kubectl get pods`
- `$ kubectl delete node worker-2`

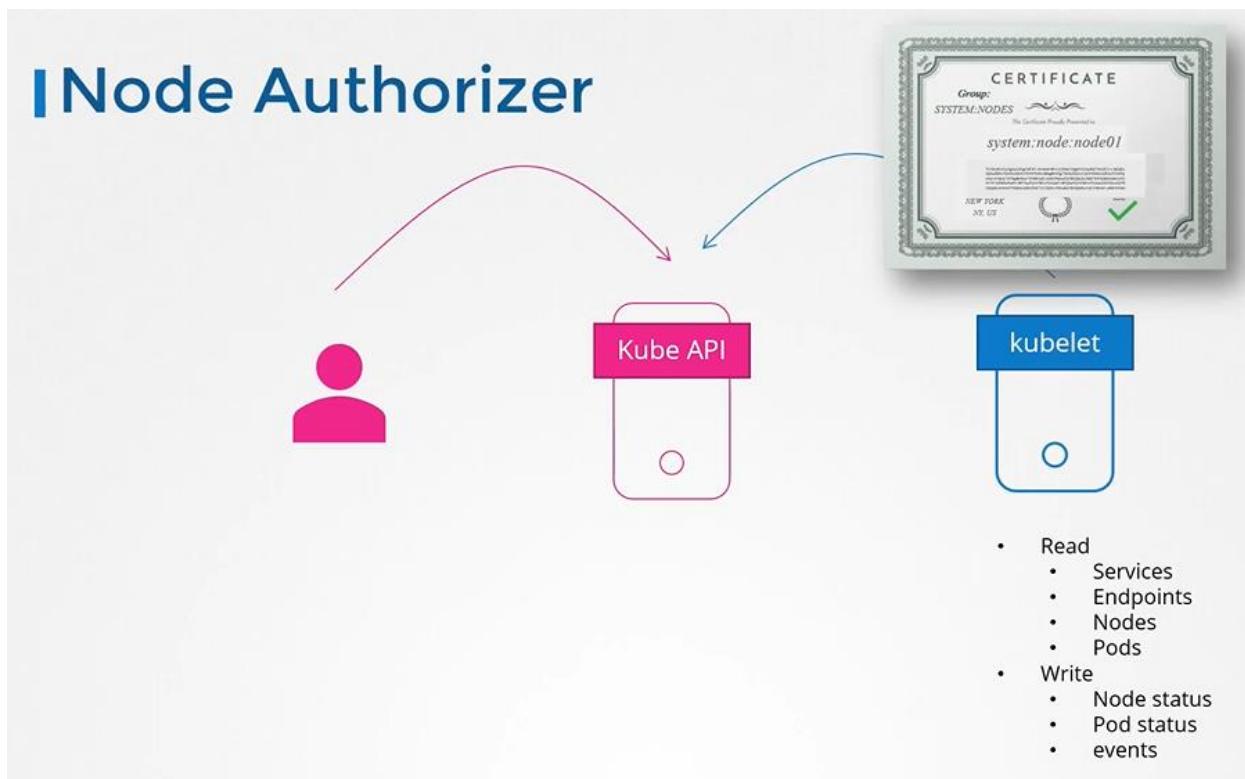
# Why Authorization?



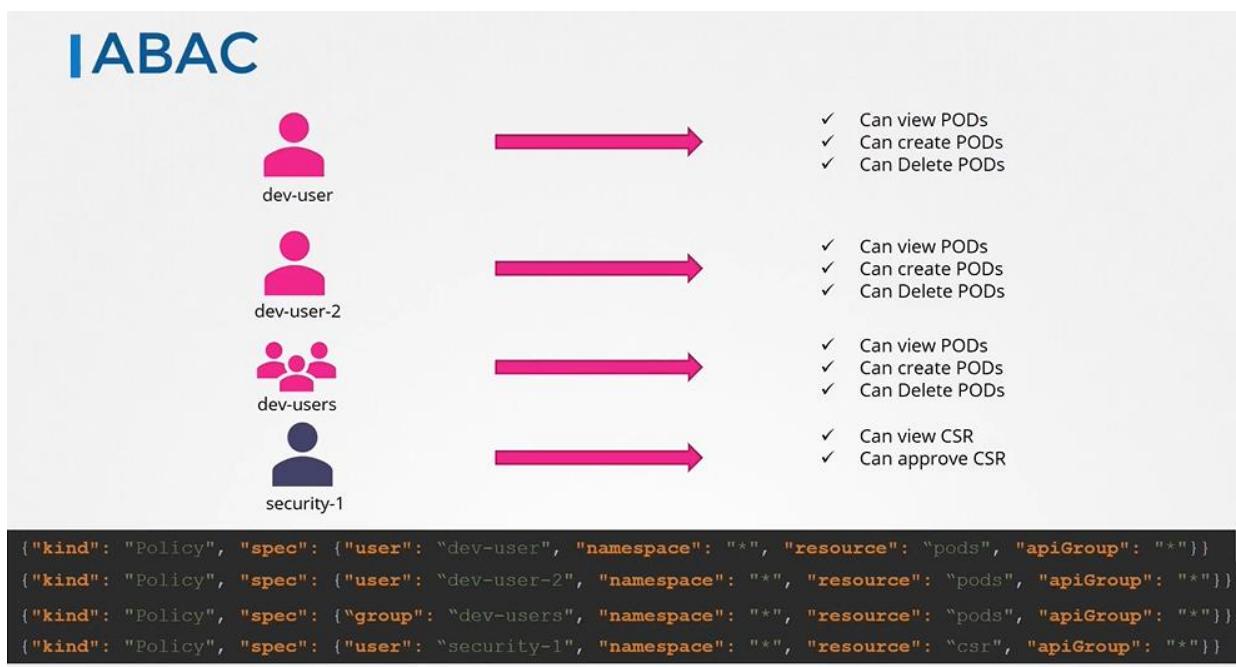
## Authorization Mechanisms

- There are different authorization mechanisms supported by kubernetes
  - Node Authorization
  - Attribute-based Authorization (ABAC)
  - Role-Based Authorization (RBAC)
  - Webhook

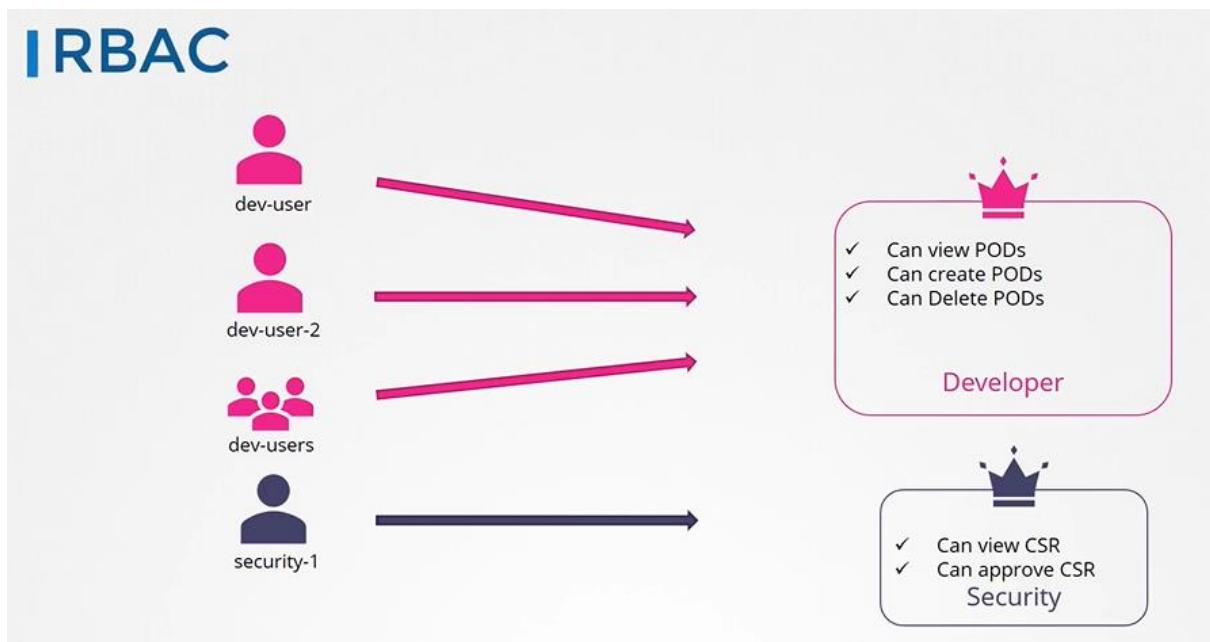
## Node Authorization



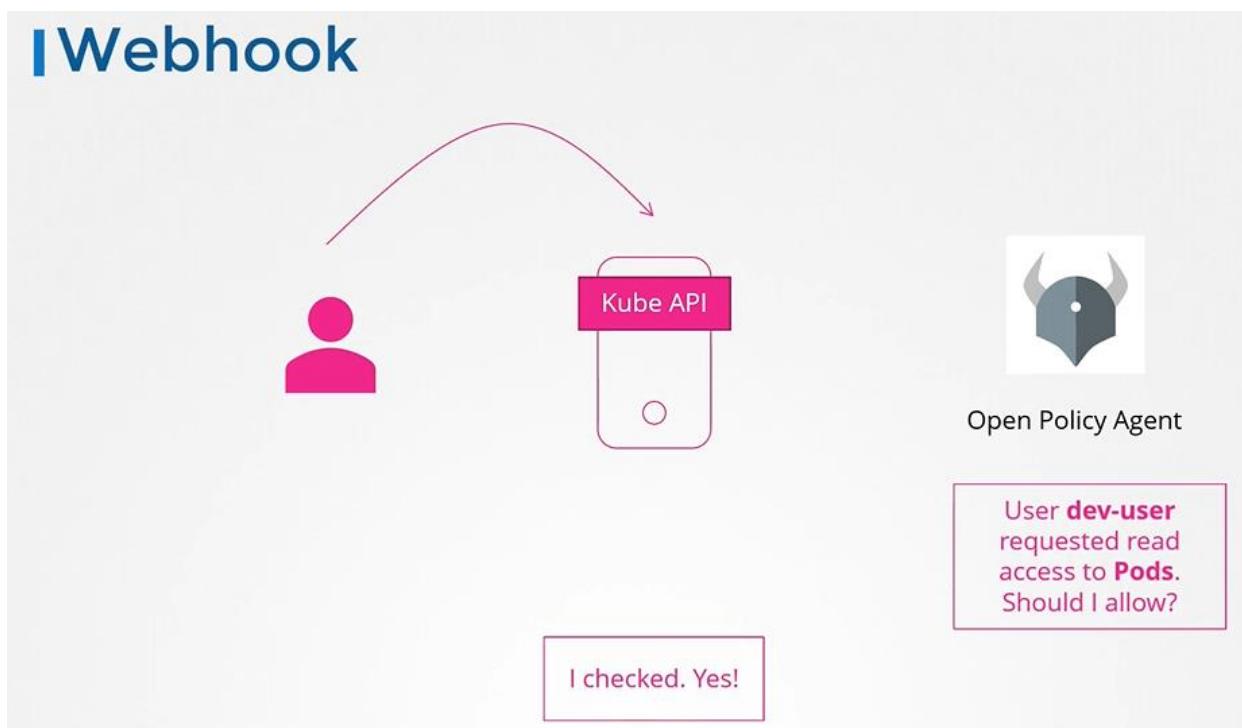
## ABAC



## RBAC



## Webhook



## Authorization Modes

- The mode options can be defined on the kube-apiserver

# I Authorization Mode

AlwaysAllow

NODE

ABAC

RBAC

WEBHOOK

AlwaysDeny

```
ExecStart=/usr/local/bin/kube-apiserver \
--advertise-address=${INTERNAL_IP} \
--allow-privileged=true \
--apiserver-count=3 \
--authorization-mode=Node,RBAC,Webhook \
--bind-address=0.0.0.0 \
--enable-swagger-ui=true \
--etcd-cafile=/var/lib/kubernetes/ca.pem \
--etcd-certfile=/var/lib/kubernetes/apiserver-etcd-client.crt \
--etcd-keyfile=/var/lib/kubernetes/apiserver-etcd-client.key \
--etcd-servers=https://127.0.0.1:2379 \
--event-ttl=1h \
--kubelet-certificate-authority=/var/lib/kubernetes/ca.pem \
--kubebundle-client-certificate=/var/lib/kubernetes/apiserver-etcd-client.crt \
--kubebundle-client-key=/var/lib/kubernetes/apiserver-etcd-client.key \
--service-node-port-range=30000-32767 \
--client-ca-file=/var/lib/kubernetes/ca.pem \
--tls-cert-file=/var/lib/kubernetes/apiserver.crt \
--tls-private-key-file=/var/lib/kubernetes/apiserver.key \
--v=2
```

- When you specify multiple modes, it will authorize in the order in which it is specified



NODE

RBAC

WEBHOOK

```
ExecStart=/usr/local/bin/kube-apiserver \
--advertise-address=${INTERNAL_IP} \
--allow-privileged=true \
--apiserver-count=3 \
--authorization-mode=Node,RBAC,Webhook \
--bind-address=0.0.0.0 \\\
```

## K8s Reference Docs

- <https://kubernetes.io/docs/reference/access-authn-authz/authorization/>

## RBAC

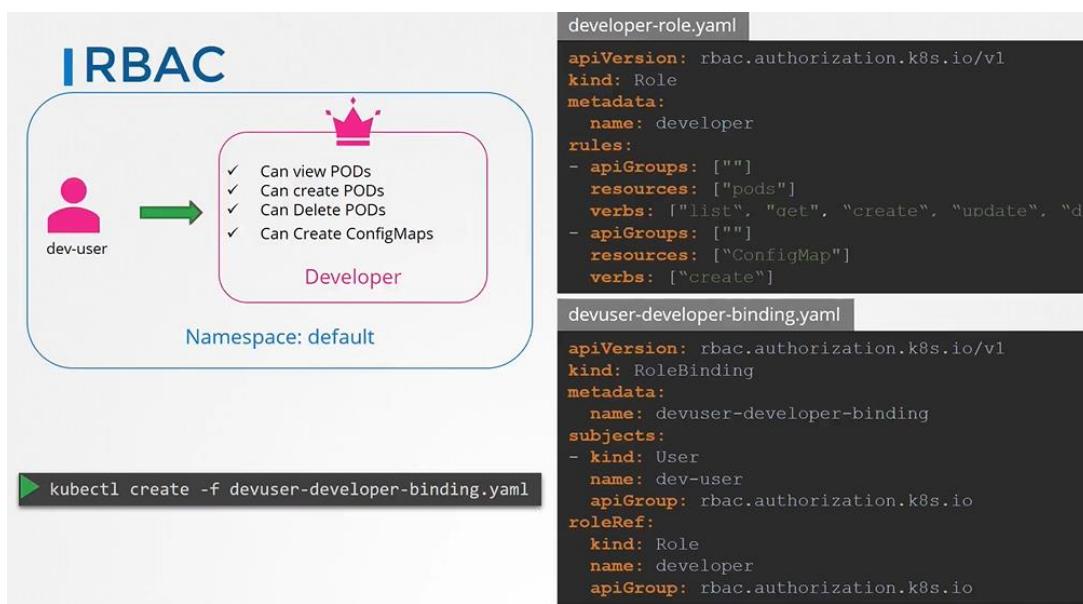
In this section, we will take a look at RBAC

## How do we create a role?

- Each role has 3 sections
  - apiGroups
  - resources
  - verbs
- create the role with kubectl command
- \$ kubectl create -f developer-role.yaml

The next step is to link the user to that role.

- For this we create another object called **RoleBinding**. This role binding object links a user object to a role.
- create the role binding using kubectl command
- \$ kubectl create -f devuser-developer-binding.yaml
- Also note that the roles and role bindings fall under the scope of namespace.
- apiVersion: rbac.authorization.k8s.io/v1
- kind: Role
- metadata:
  - name: developer
- rules:
  - apiGroups: [""]
    - resources: ["pods"]
    - verbs: ["get", "list", "update", "delete", "create"]
  - apiGroups: [""]
    - resources: ["ConfigMap"]
    - verbs: ["create"]
- apiVersion: rbac.authorization.k8s.io/v1
- kind: RoleBinding
- metadata:
  - name: devuser-developer-binding
- subjects:
  - kind: User
    - name: dev-user # "name" is case sensitive
    - apiGroup: rbac.authorization.k8s.io
- roleRef:
  - kind: Role
  - name: developer
  - apiGroup: rbac.authorization.k8s.io



## View RBAC

- To list roles
- \$ kubectl get roles
- To list rolebindings
- \$ kubectl get rolebindings
- To describe role
- \$ kubectl describe role developer

# IView RBAC

```
▶ kubectl get roles
```

| NAME      | AGE |
|-----------|-----|
| developer | 4s  |

```
▶ kubectl get rolebindings
```

| NAME                      | AGE |
|---------------------------|-----|
| devuser-developer-binding | 24s |

```
▶ kubectl describe role developer
```

| Name:              | developer         |                |                                |
|--------------------|-------------------|----------------|--------------------------------|
| Labels:            | <none>            |                |                                |
| Annotations:       | <none>            |                |                                |
| <b>PolicyRule:</b> |                   |                |                                |
| Resources          | Non-Resource URLs | Resource Names | Verbs                          |
| -----              | -----             | -----          | -----                          |
| ConfigMap          | [ ]               | [ ]            | [create]                       |
| pods               | [ ]               | [ ]            | [get watch list create delete] |

- To describe rolebinding
- \$ kubectl describe rolebinding devuser-developer-binding

```
▶ kubectl describe rolebinding devuser-developer-binding
```

| Name:            | devuser-developer-binding |           |
|------------------|---------------------------|-----------|
| Labels:          | <none>                    |           |
| Annotations:     | <none>                    |           |
| <b>Role:</b>     |                           |           |
| Kind:            | Role                      |           |
| Name:            | developer                 |           |
| <b>Subjects:</b> |                           |           |
| Kind             | Name                      | Namespace |
| ----             | ----                      | -----     |
| User             | dev-user                  |           |

*What if you being a user would like to see if you have access to a particular resource in the cluster.*

## Check Access

- You can use the kubectl auth command
- \$ kubectl auth can-i create deployments
- \$ kubectl auth can-i delete nodes
- \$ kubectl auth can-i create deployments --as dev-user
- \$ kubectl auth can-i create pods --as dev-user
- \$ kubectl auth can-i create pods --as dev-user --namespace test

# Check Access

```
▶ kubectl auth can-i create deployments
yes
```

```
▶ kubectl auth can-i delete nodes
no
```

```
▶ kubectl auth can-i create deployments --as dev-user
no
```

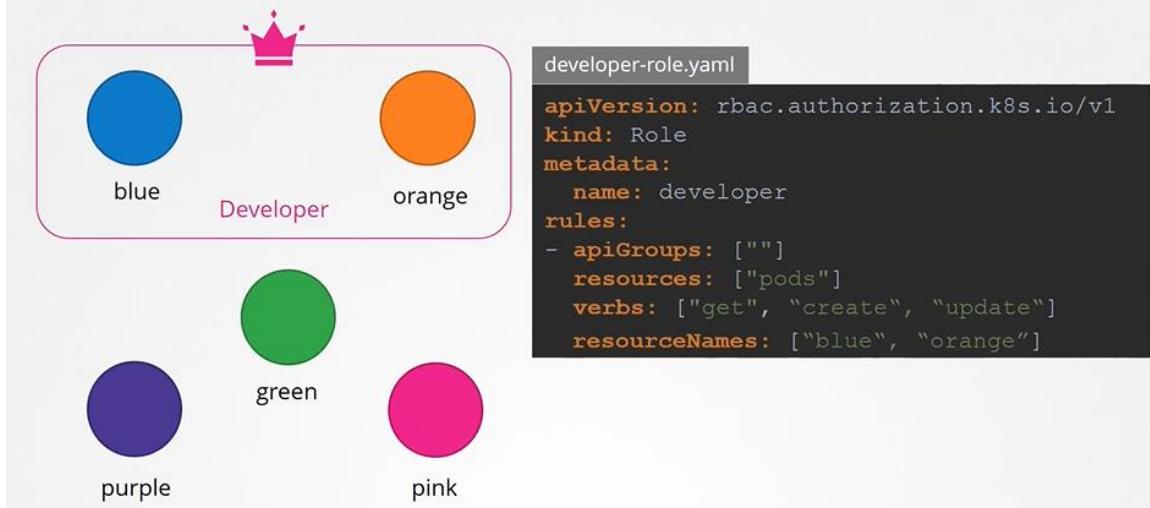
```
▶ kubectl auth can-i create pods --as dev-user
yes
```

```
▶ kubectl auth can-i create pods --as dev-user --namespace test
no
```

## Resource Names

- Note on resource names we just saw how you can provide access to users for resources like pods within the namespace.
- apiVersion: rbac.authorization.k8s.io/v1
- kind: Role
- metadata:
- name: developer
- rules:
- - apiGroups: [""]
 # "" indicates the core API group
- resources: ["pods"]
- verbs: ["get", "update", "create"]
- resourceNames: ["blue", "orange"]

## I Resource Names



*K8s Reference Docs*

- <https://kubernetes.io/docs/reference/access-authn-authz/rbac/>
- <https://kubernetes.io/docs/reference/access-authn-authz/rbac/#command-line-utilities>

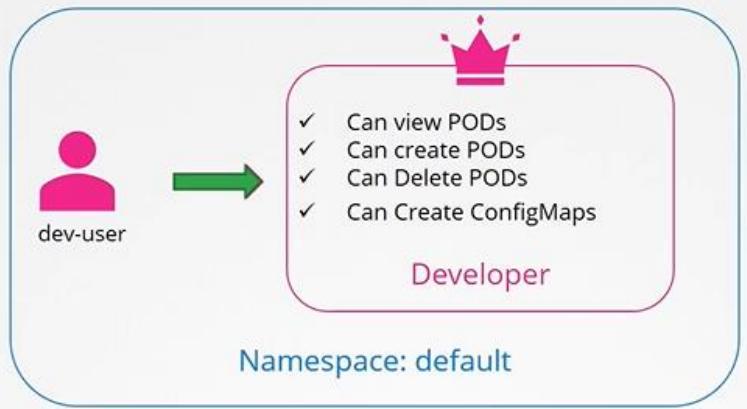
## Cluster Roles

In this section, we will take a look at cluster roles

### Roles

- Roles and Rolebindings are namespaced meaning they are created within namespaces.

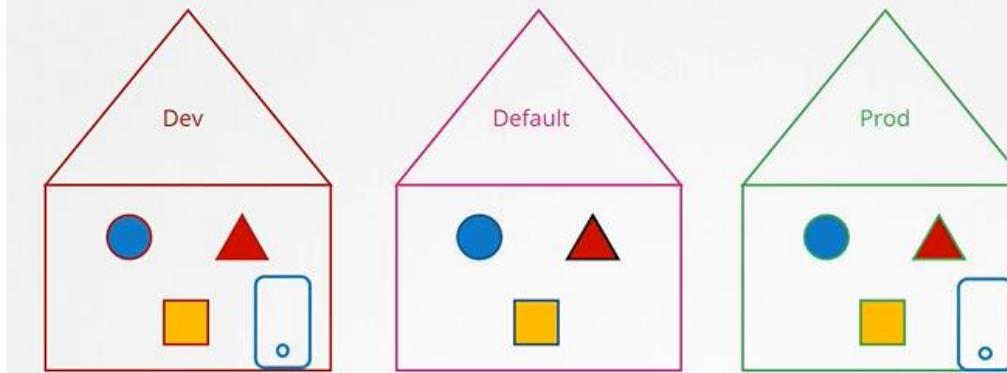
## I Roles



## Namespaces

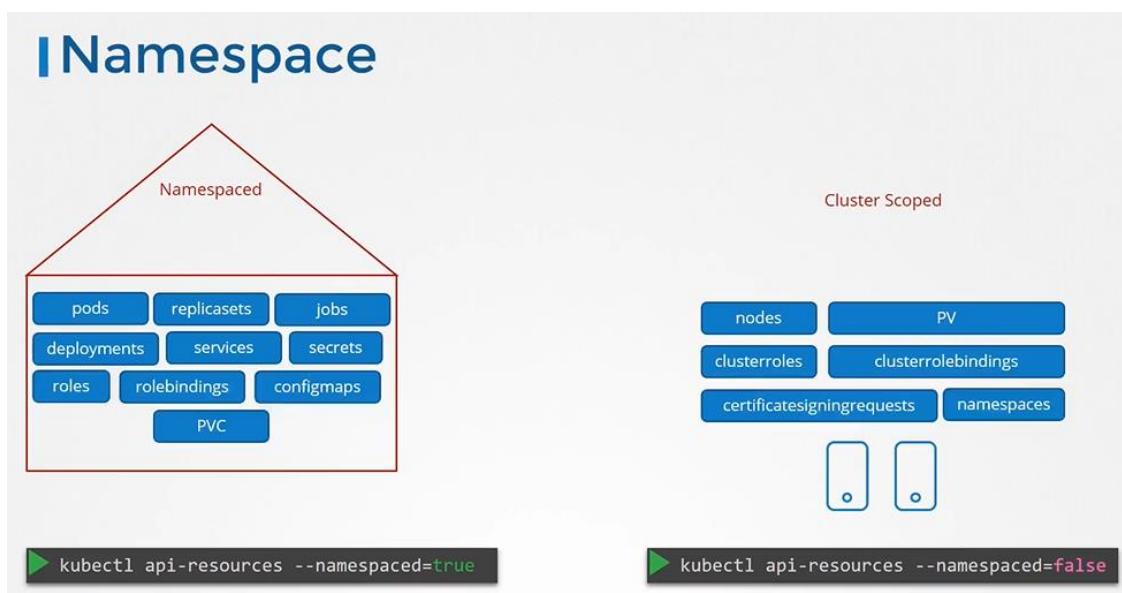
- Can you group or isolate nodes within a namespace?
  - No, those are cluster wide or cluster scoped resources. They cannot be associated to any particular namespace.

## I Namespace



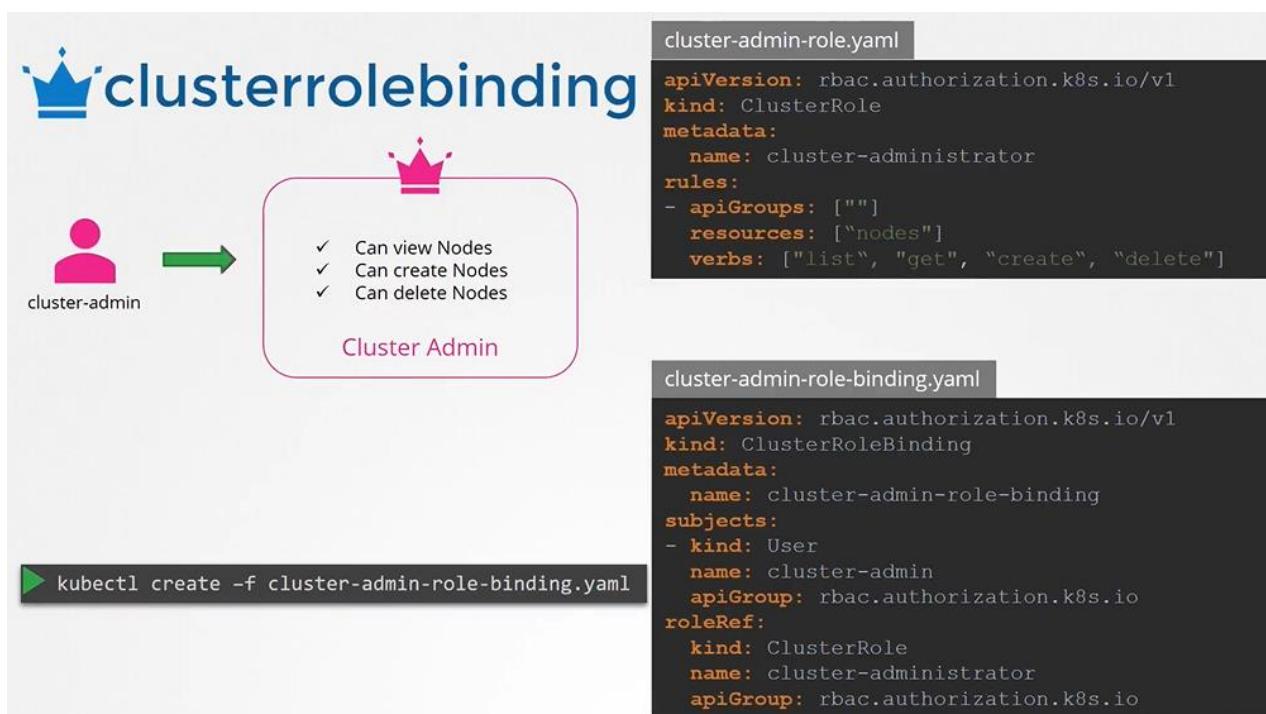
- So the resources are categorized as either namespaced or cluster scoped.
- To see namespaced resources
- `$ kubectl api-resources --namespaced=true`
- To see non-namespaced resources

- `$ kubectl api-resources --namespaced=false`



## Cluster Roles and Cluster Role Bindings

- Cluster Roles are roles except they are for a cluster scoped resources. Kind as **ClusterRole**
- `apiVersion: rbac.authorization.k8s.io/v1`
- `kind: ClusterRole`
- `metadata:`
- `name: cluster-administrator`
- `rules:`
- `- apiGroups: [""]` # "" indicates the core API group
- `resources: ["nodes"]`
- `verbs: ["get", "list", "delete", "create"]`
- `apiVersion: rbac.authorization.k8s.io/v1`
- `kind: ClusterRoleBinding`
- `metadata:`
- `name: cluster-admin-role-binding`
- `subjects:`
- `- kind: User`
- `name: cluster-admin`
- `apiGroup: rbac.authorization.k8s.io`
- `roleRef:`
- `kind: ClusterRole`
- `name: cluster-administrator`
- `apiGroup: rbac.authorization.k8s.io`
- `$ kubectl create -f cluster-admin-role.yaml`
- `$ kubectl create -f cluster-admin-role-binding.yaml`



- You can create a cluster role for namespace resources as well. When you do that user will have access to these resources across all namespaces.

#### K8s Reference Docs

- <https://kubernetes.io/docs/reference/access-authn-authz/rbac/#role-and-clusterrole>
- <https://kubernetes.io/docs/reference/access-authn-authz/rbac/#command-line-utilities>

## Image Security

In this section we will take a look at image security

## Image

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx-pod
spec:
 containers:
 - name: nginx
 image: nginx
```

## Image

```
nginx-pod.yaml
apiVersion: v1
kind: Pod
metadata:
 name: nginx-pod
spec:
 containers:
 - name: nginx
 image: nginx
```

## Image

image: docker.io/nginx/nginx

Registry      User/ Account      Image/ Repository

gcr.io/ kubernetes-e2e-test-images/dnsutils

## Private Registry

- To login to the registry
- \$ docker login private-registry.io
- Run the application using the image available at the private registry
- \$ docker run private-registry.io/apps/internal-app

# IPrivate Repository

```
> docker login private-registry.io
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to
https://hub.docker.com to create one.
Username: registry-user
Password:
WARNING! Your password will be stored unencrypted in /home/vagrant/.docker/config.json.

Login Succeeded
```

```
> docker run private-registry.io/apps/internal-app
```

- To pass the credentials to the docker untaged on the worker node for that we first create a secret object with credentials in it.
- ```
$ kubectl create secret docker-registry regcred \
--docker-server=private-registry.io \
--docker-username=registry-user \
--docker-password=registry-password \
--docker-email=registry-user@org.com
```
- We then specify the secret inside our pod definition file under the imagePullSecret section
- ```
apiVersion: v1
kind: Pod
metadata:
 name: nginx-pod
spec:
 containers:
 - name: nginx
 image: private-registry.io/apps/internal-app
 imagePullSecrets:
 - name: regcred
```

# IPrivate Repository

```
> docker login private-registry.io
```

```
> docker run private-registry.io/apps/internal-app
```

**nginx-pod.yaml**

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx-pod
spec:
 containers:
 - name: nginx
 image: private-registry.io/apps/internal-app
 imagePullSecrets:
 - name: regcred
```

```
> kubectl create secret docker-registry regcred \
--docker-server= private-registry.io \
--docker-username= registry-user \
--docker-password= registry-password \
--docker-email= registry-user@org.com
```

### K8s Reference Docs

- <https://kubernetes.io/docs/concepts/containers/images/>

## Security Context

In this section, we will take a look at security context

### Container Security

```
$ docker run --user=1001 ubuntu sleep 3600
$ docker run -cap-add MAC_ADMIN ubuntu
```

# Container Security

The diagram illustrates container security settings. It features two terminal command examples:

- `docker run --user=1001 ubuntu sleep 3600`
- `docker run --cap-add MAC_ADMIN ubuntu`

Below the commands is a user icon represented by a blue silhouette of a person. To the right of the icon is the text "MAC\_ADMIN".

### Kubernetes Security

- You may choose to configure the security settings at a container level or at a pod level.

# I Kubernetes Security



## Security Context

- To add security context on the container and a field called **securityContext** under the spec section.
- `apiVersion: v1`
- `kind: Pod`
- `metadata:`
- `name: web-pod`
- `spec:`
- `securityContext:`
- `runAsUser: 1000`
- `containers:`
- `- name: ubuntu`
- `image: ubuntu`
- `command: ["sleep", "3600"]`

## I Security Context

```
apiVersion: v1
kind: Pod
metadata:
 name: web-pod
spec:
 securityContext:
 runAsUser: 1000

 containers:
 - name: ubuntu
 image: ubuntu
 command: ["sleep", "3600"]
```



- To set the same context at the container level, then move the whole section under container section.
- apiVersion: v1
- kind: Pod
- metadata:
- name: web-pod
- spec:
- containers:
- - name: ubuntu
- image: ubuntu
- command: ["sleep", "3600"]
- securityContext:
- runAsUser: 1000

## I Security Context

```
apiVersion: v1
kind: Pod
metadata:
 name: web-pod
spec:
 containers:
 - name: ubuntu
 image: ubuntu
 command: ["sleep", "3600"]
 securityContext:
 runAsUser: 1000
```



- To add capabilities use the **capabilities** option
- apiVersion: v1
- kind: Pod

- metadata:
- name: web-pod
- spec:
- containers:
- - name: ubuntu
- image: ubuntu
- command: ["sleep", "3600"]
- securityContext:
- runAsUser: 1000
- capabilities:
- add: ["MAC\_ADMIN"]

## Security Context

```
apiVersion: v1
kind: Pod
metadata:
 name: web-pod
spec:
 containers:
 - name: ubuntu
 image: ubuntu
 command: ["sleep", "3600"]
 securityContext:
 runAsUser: 1000
 capabilities:
 add: ["MAC_ADMIN"]
```



**Note:** Capabilities are only supported at the container level and not at the POD level

## K8s Reference Docs

- <https://kubernetes.io/docs/tasks/configure-pod-container/security-context/>

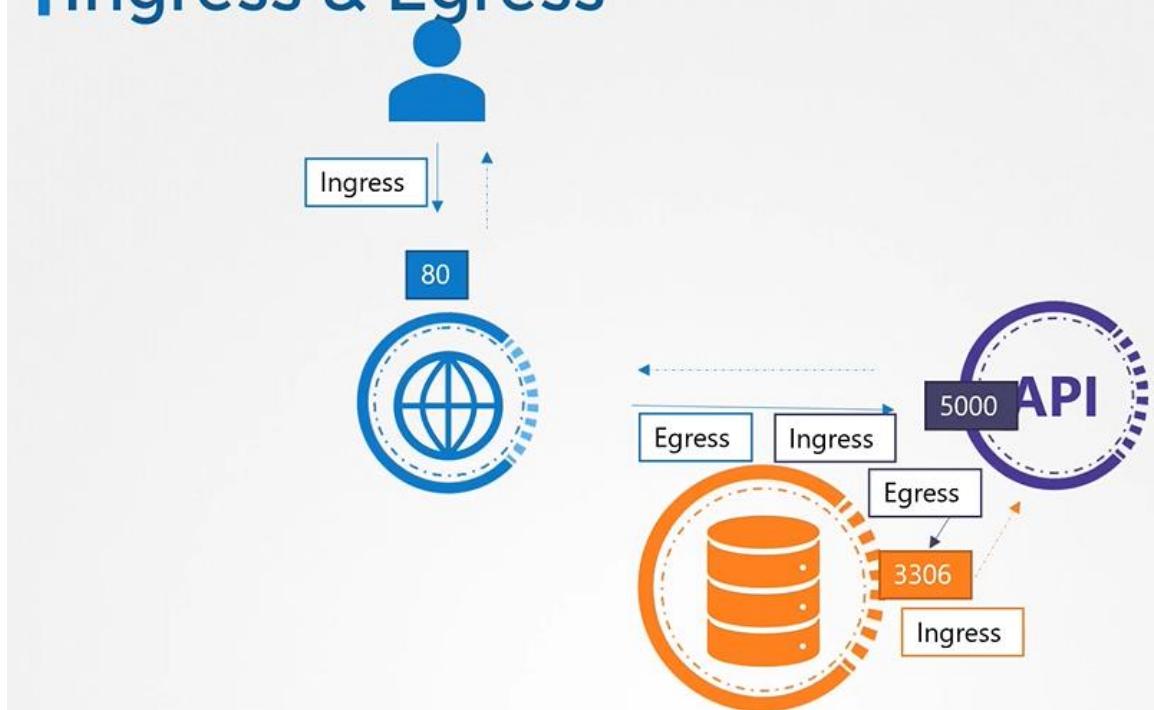
## Network Policies

Traffic flowing through a webserver serving frontend to users an app server serving backend API and a database server



- There are two types of traffic
  - Ingress
  - Egress

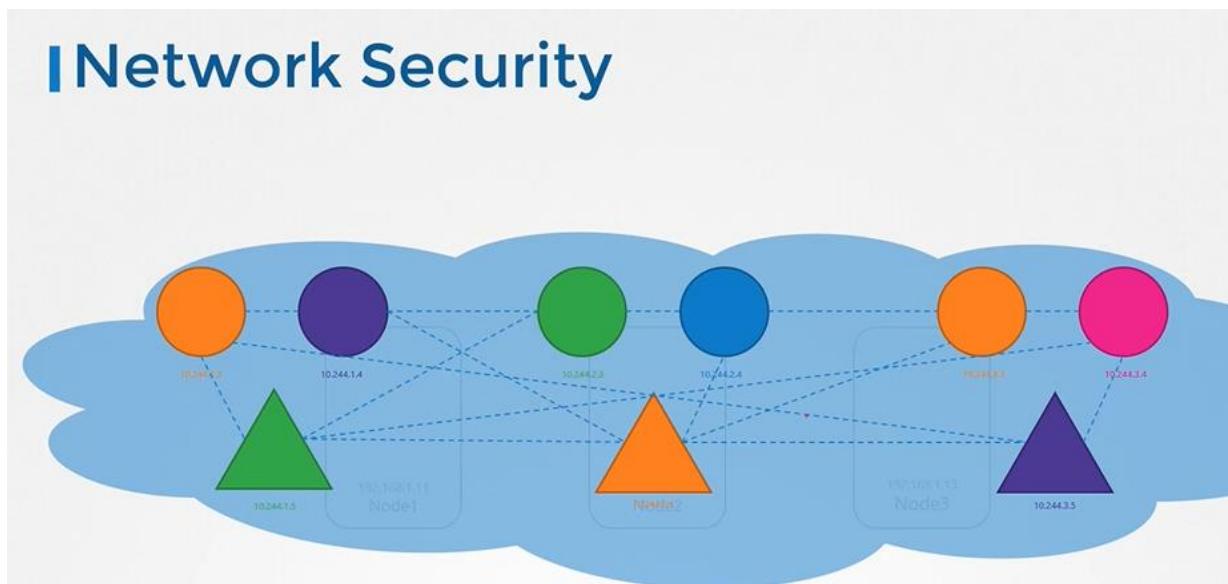
## Ingress & Egress



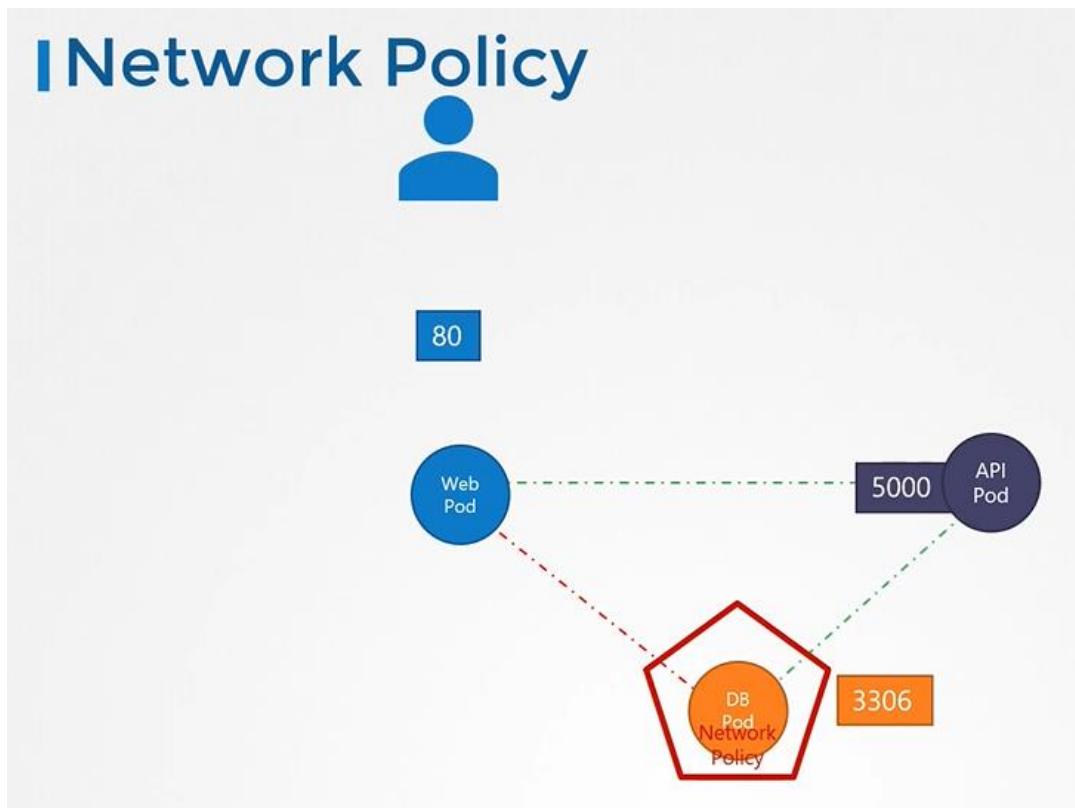
## Traffic



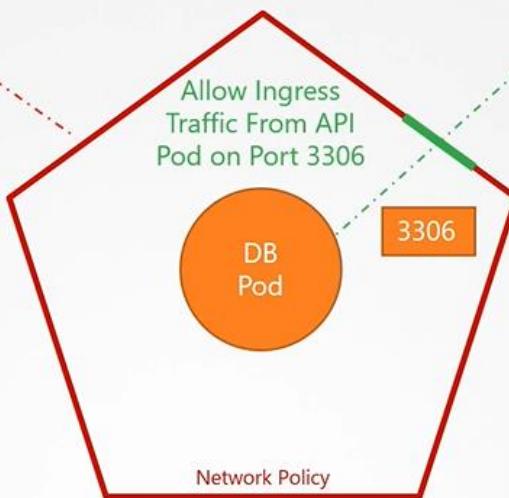
## Network Security



## Network Policy



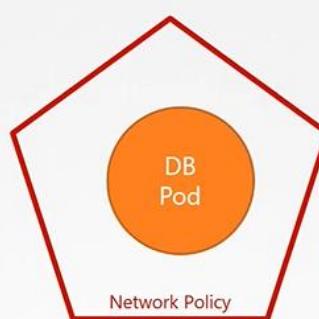
# Network Policy



## Network Policy Selectors

### Network Policy - Selectors

Allow Ingress  
Traffic From API  
Pod on Port 3306



```
podSelector:
 matchLabels:
 role: db
```

```
labels:
 role: db
```

## Network Policy Rules

# Network Policy - Rules

```

policyTypes:
- Ingress
ingress:
- from:
 - podSelector:
 matchLabels:
 name: api-pod
 ports:
 - protocol: TCP
 port: 3306

```

Allow  
Ingress  
Traffic  
From  
API Pod  
on  
Port 3306

## Create network policy

- To create a network policy
- apiVersion: networking.k8s.io/v1
- kind: NetworkPolicy
- metadata:
- name: db-policy
- spec:
- podSelector:
- matchLabels:
- role: db
- policyTypes:
- - Ingress
- ingress:
- - from:
- - podSelector:
- matchLabels:
- role: api-pod
- ports:
- - protocol: TCP
- port: 3306

\$ kubectl create -f policy-definition.yaml

![npol3](../../images/npol3.PNG)

![npol4](../../images/npol4.PNG)

Note

## | Note

### Solutions that Support Network Policies:

- Kube-router
- Calico
- Romana
- Weave-net

### Solutions that DO NOT Support Network Policies:

- Flannel

*K8s Reference Docs*

- <https://kubernetes.io/docs/concepts/services-networking/network-policies/>
- <https://kubernetes.io/docs/tasks/administer-cluster/declare-network-policy/>

# Storage Section Introduction

In this section, we will take a look at **Storage**

- Introduction of Docker Storage
- Persistent Volume Claim
- Persistent Volume
- Storage Class

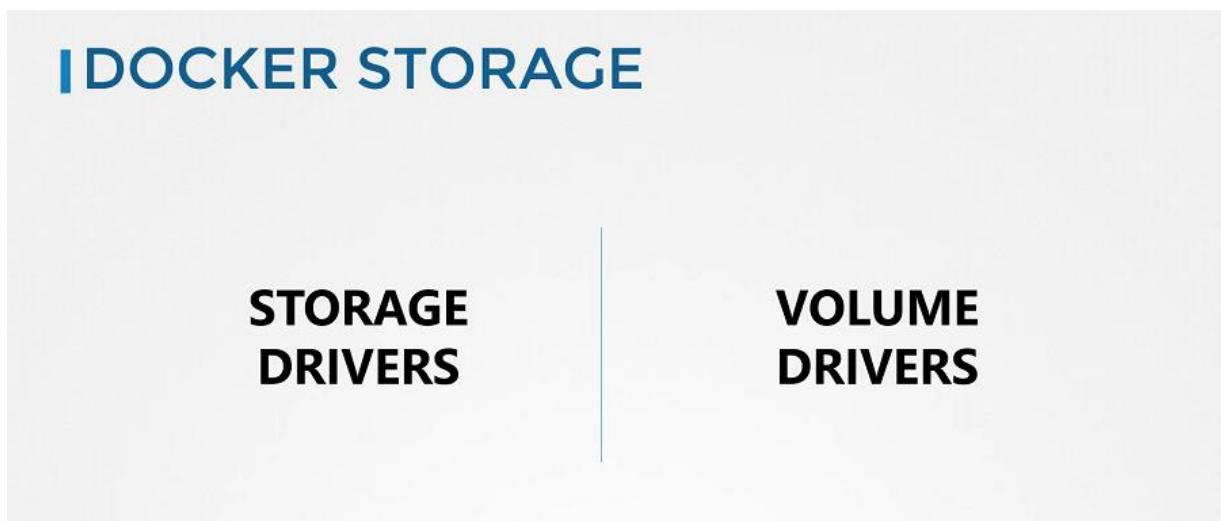
# Introduction to Docker Storage

In this section, we will take a look at **Docker storage**.

- To understand storage in the container orchestration tools like **Kubernetes**, It is important to first understand how storage works with containers. Understanding how storage works with **Docker** first and getting all the basics right will later make it so much easier to understand how it works in **Kubernetes**.
- If you are new to **Docker** then you can learn some basics of docker from the course [Docker for the absolute beginner course](#), that is free.

## Docker Storage

- There are two concepts comes into the docker, Storage drivers and Volume drivers plugins.



We will first discuss about Storage drivers.

### *Docker Reference Docs*

- <https://docs.docker.com/storage/storagedriver/>
- <https://docs.docker.com/storage/volumes/>

## Storage in Docker

In this section, we will take a look at docker **Storage driver and Filesystem**.

- We are going to see where and how Docker stores data and how it manages filesystem of the containers.

### Filesystem

- Let's see how Docker stores data on the local file system. At the first time, When you install Docker on a system, it creates this directory structures at /var/lib/docker.
- \$ cd /var/lib/docker/
-

## File system

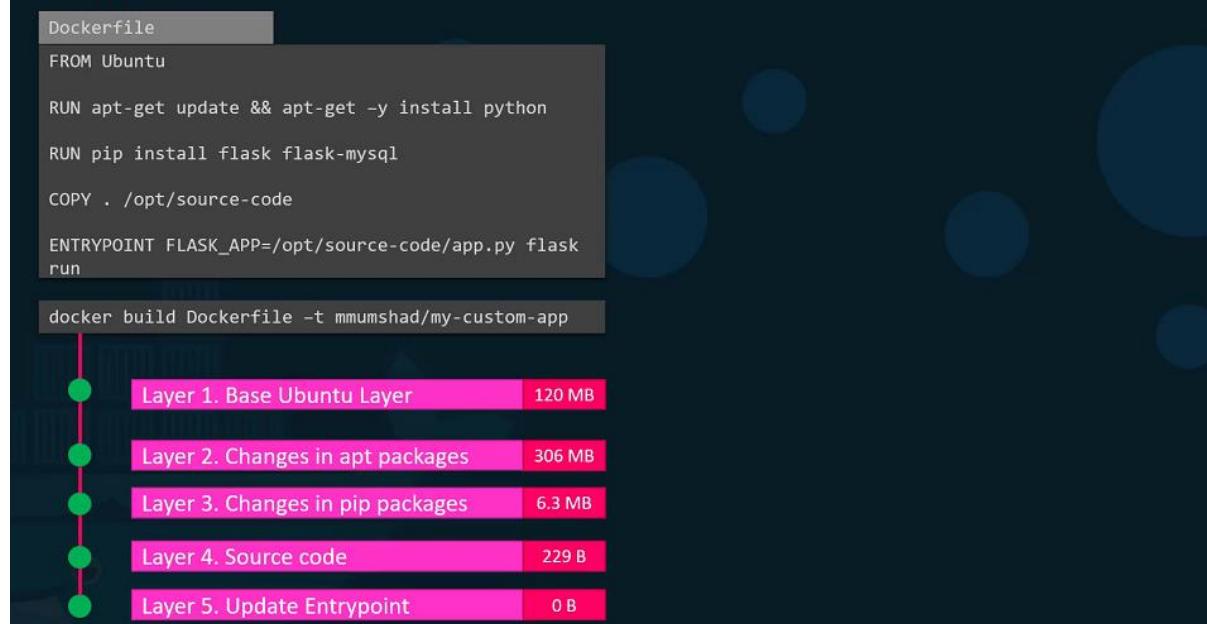


- You have multiple directories under it called aufs, containers, image, volumes etc.
- This is where Docker stores all its data by default.
- All files related to containers are stored under the containers directory and the files related to images are stored under the image directory. Any volumes created by the Docker containers are created under the volumes directory.
- For now, let's just understand where Docker stores its files of an image and a container and in what format.
- To understand that we need to understand Dockers layered architecture.

## Layered Architecture

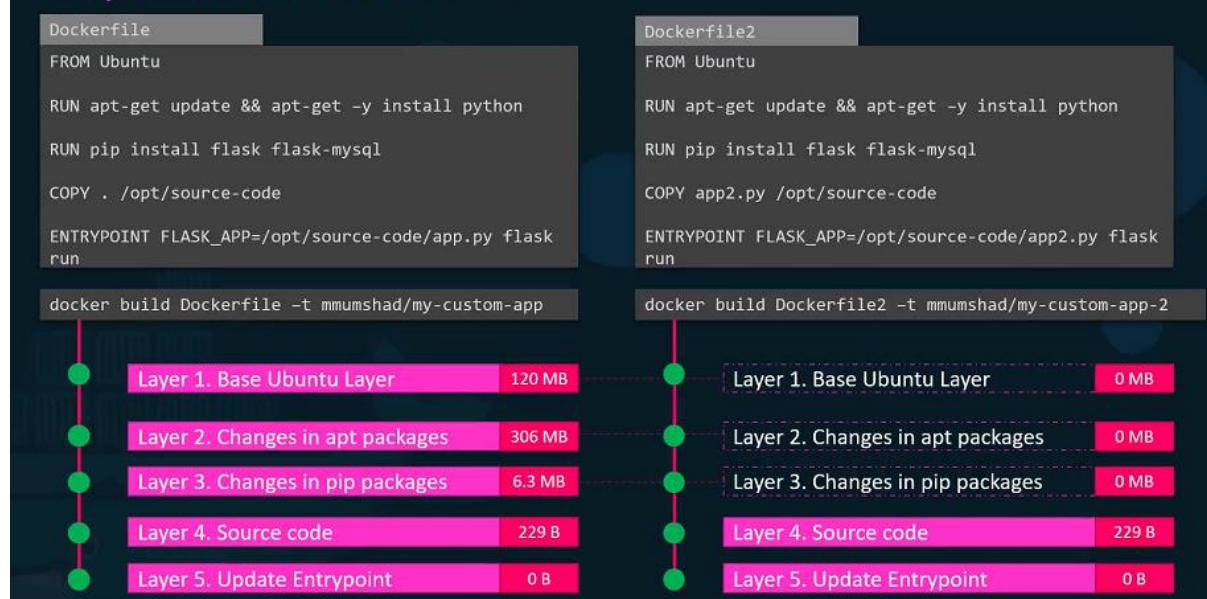
- When docker builds images, it builds these in a layered architecture. Each line of instruction in the Docker file creates a new layer in the Docker image with just the changes from the previous layer.
- As you can see in the image, layer-1 is the Ubuntu base layer, layer-2 install the packages, layer-3 install the python packages, layer-4 update the source code, layer-5 updates the entry point of the image. Since each layer only stores the changes from the previous layer. It is reflected in the size as well.

## Layered architecture

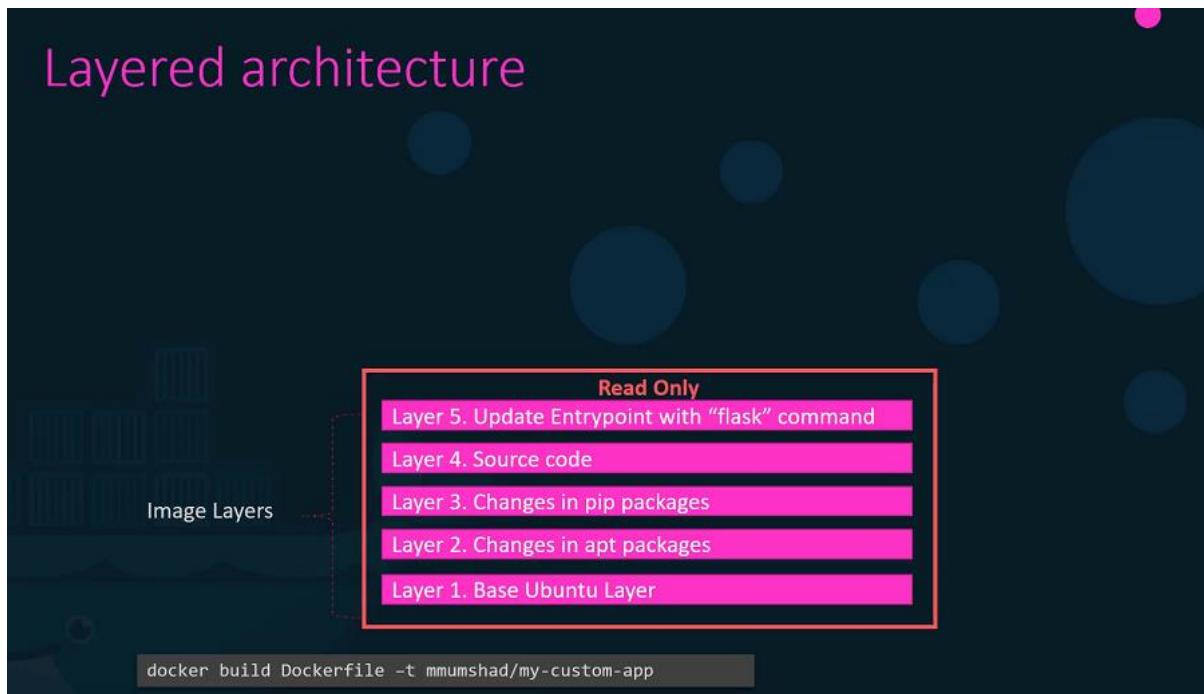


- For better understanding the advantages of this layered architecture, Let's take a different Dockerfile, this is very similar to our first application. only source code and entry point is different to create this application.
- When image builds, docker is not going to build first three layers instead of it reuses the same three layers it built for the first application from the cache. Only creates the last two layers with the new sources and the new entry point.
- This way, Docker builds images faster and efficiently saves disk spaces. This is also applicable, if you were to update your application code. Docker simply reuses all the previous layers from the cache and quickly rebuild the application image by updating the latest source code.

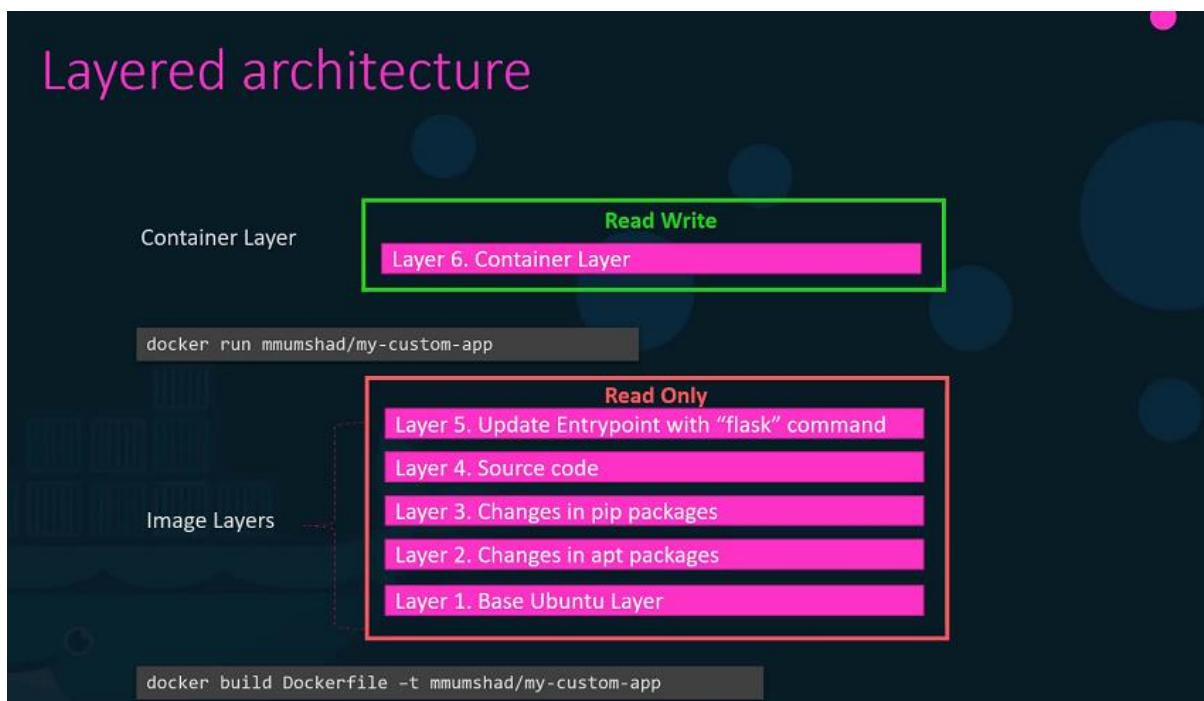
## Layered architecture



- Let's rearrange the layers bottom up so we can understand it better. All of these layers are created when we run the Docker build command to form the final Docker image. Once the build is complete, you cannot modify the contents of these layers and so they are read-only and you can only modify them by initiating a new build.



- When you run a container based off of this image, using the **Docker run** command Docker creates a container based off of these layers and creates a new writeable layer on top of the image layer. The writeable layer is used to store data created by the container such as log files written by the applications, any temporary files generated by the container.



- When the container is destroyed, this layer and all of the changes stored in it are also destroyed. Remember that the same image layer is shared by all containers created using this image.
- If I will create a new file called temp.txt in the newly created container, which is read and write.
- The files in the image layer are read-only meaning you cannot edit anything in those layers.



- Let's take an example of our application code. Since we bake our code into the image, the code is part of the image and as such, its read-only. After running a container, what if I wish to modify the source code.
- Yes, I can still modify this file, but before I saved the modified file, Docker automatically creates a copy of the file in the read-write layer and I will then be modifying a different version of the file in the read-write layer. All future modifications will be done on this copy of the file in the read-write layer. This is called copy-on-right mechanism.
- The Image layer being a read-only just means that the files in these layers will not be modified in the image itself. So, the image will remain the same all the time until you rebuild the image using the Docker build command. If container destroyed then all of the data that was stored in the container layer also gets deleted.

## Volumes

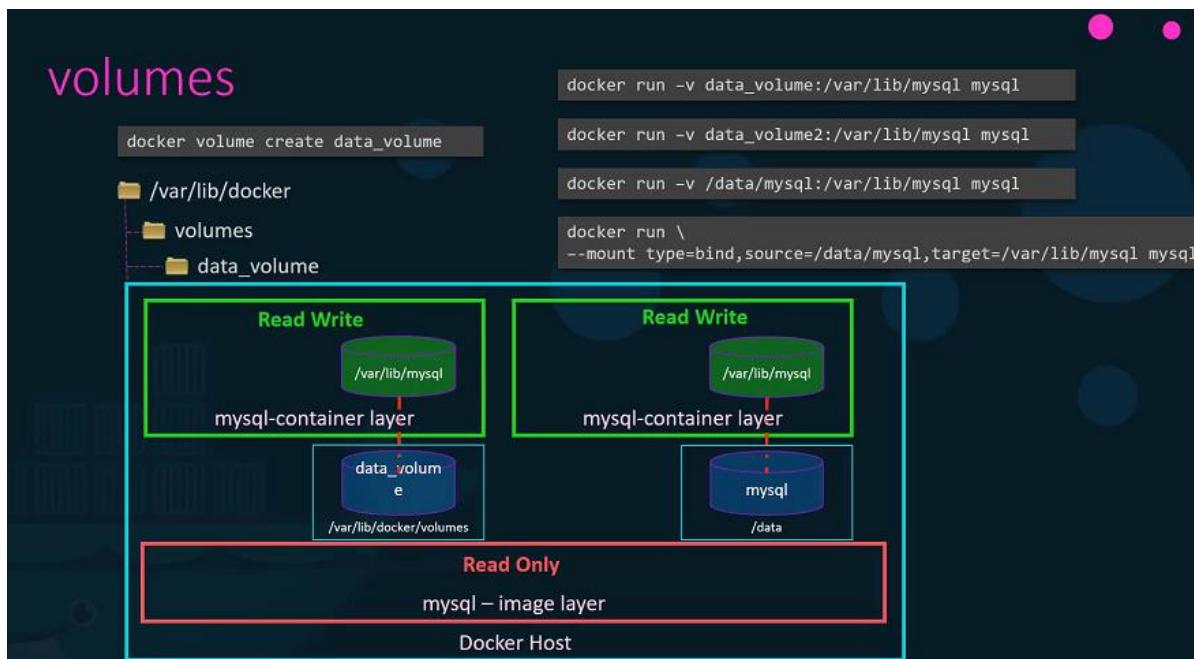
- To get a persistent data in the containers, we need to create a volume using the `docker volume create` command. So, when I run the `docker volume create data_volume` command, it creates a directory called `data_volume` under the `/var/lib/docker/volumes` directory.
- `$ docker volume create data_volume`
-

- \$ ls -l /var/lib/docker/volumes/
  - drwxr-xr-x 3 root root 4096 Aug 01 17:53 data\_volume
  - 
  - \$ docker volume ls
  - 
  - DRIVER VOLUME NAME
 

|         |             |
|---------|-------------|
| • local | data_volume |
|---------|-------------|
- When I run the Docker container using the docker run command, I could mount this volume inside the Docker containers, with -v options.
- So when I do a docker run -v then specify my newly created volume name followed by a colon(:) and the location inside my container, which is the default location where MySQL stores data and that is /var/lib/mysql and at the last image name of MySQL. This will create a new container and mount the data volume.
- \$ docker run -v data\_volume:/var/lib/mysql mysql
- Even if the container is destroyed, the data is still available.
- What if you didn't create the Docker volume before the Docker run command. In this image, Docker will automatically create a volume named data\_volume2 and mount it to the container.
- \$ docker run -v data\_volume2:/var/lib/mysql mysql
- 
- \$ docker volume ls
- 
- DRIVER VOLUME NAME
 

|         |              |
|---------|--------------|
| • local | data_volume  |
| • local | data_volume2 |
- You should be able to see all these volumes if you list the contents of the /var/lib/docker volumes directory. This is called **Volume mounting**.
- What if we had our data already at another location?
- Let's say we have some external storage on the Docker host at /data path and we would like to store database data on that volume and not in the default /var/lib/docker volumes directory. In that case, we would run a container using the command docker run -v. But in this case, we will provide the complete path to the directory we would like to mount. That is /data/mysql and so it will create a container and mount the directory to the container. This is called **Bind mounting**.
- So there are two types of mounts, volume mount and bind mount.
  - Volume mount, mounts of volume from the volumes directory and bind mount, mounts indirectly from any location on the Docker host.
- Instead of -v option, we can preferred --mount option.
- \$ mkdir -p /data/mysql

- \$ docker run --mount type=bind,source=/data/mysql,target=/var/lib/mysql mysql
- So, who is responsible for doing all of these Operations?
- Maintaining of layered architecture, creating a writeable layer, moving files across layers to enable copy and write etc. It's the **Storage Drivers**.
- Docker uses storage drivers to enable layered architecture.



### Common Storage Drivers

- AUFS
- ZFS
- BTRFS
- Device Mapper
- Overlay
- Overlay2
- To Selection of the storage drivers depends on the underlying OS. Docker will choose the best storage driver available automatically based on the operating system.

*Docker References*

- <https://docs.docker.com/storage/>
- [https://docs.docker.com/engine/reference/commandline/volume\\_create/](https://docs.docker.com/engine/reference/commandline/volume_create/)
- [https://docs.docker.com/engine/reference/commandline/volume\\_ls/](https://docs.docker.com/engine/reference/commandline/volume_ls/)

## Volume Driver Plugins in Docker

In this section, we will take a look at **Volume Driver Plugins in Docker**

- We discussed about Storage drivers. Storage drivers help to manage storage on images and containers.
- We have already seen that if you want to persist storage, you must create volumes. Volumes are not handled by the storage drivers. Volumes are handled by volume driver plugins. The default volume driver plugin is local.
- The local volume plugin helps to create a volume on Docker host and store its data under the /var/lib/docker/volumes/ directory.
- There are many other volume driver plugins that allow you to create a volume on third-party solutions like Azure file storage, DigitalOcean Block Storage, Portworx, Google Compute Persistent Disks etc.



- When you run a Docker container, you can choose to use a specific volume driver, such as the RexRay EBS to provision a volume from the Amazon EBS. This will create a container and attach a volume from the AWS cloud. When the container exits, your data is safe in the cloud.

```
$ docker run -it --name mysql --volume-driver rexray/ebs --mount src=ebs-
vol,target=/var/lib/mysql mysql
```

# VOLUME DRIVERS

```
▶ docker run -it \
 --name mysql
 --volume-driver rexray/ebs
 --mount src=ebs-vol,target=/var/lib/mysql
mysql
```



*Docker Reference Docs*

- [https://docs.docker.com/engine/extend/legacy\\_plugins/](https://docs.docker.com/engine/extend/legacy_plugins/)
- <https://github.com/rexray/rexray>

## Container Storage Interface

In this section, we will take a look at **Container Storage Interface**

### Container Runtime Interface

- Kubernetes used Docker alone as the container runtime engine, and all the code to work with Docker was embedded within the Kubernetes source code. Other container runtimes, such as rkt and CRI-O.
- The Container Runtime Interface is a standard that defines how an orchestration solution like Kubernetes would communicate with container runtimes like Docker. If any new container runtime interface is developed, they can simply follow the CRI standards.

# I Container Runtime Interface



CRI



## Container Networking Interface

- To support different networking solutions, the container networking interface was introduced. Any new networking vendors could simply develop their plugin based on the CNI standards and make their solution work with Kubernetes.

# I Container Network Interface



CRI



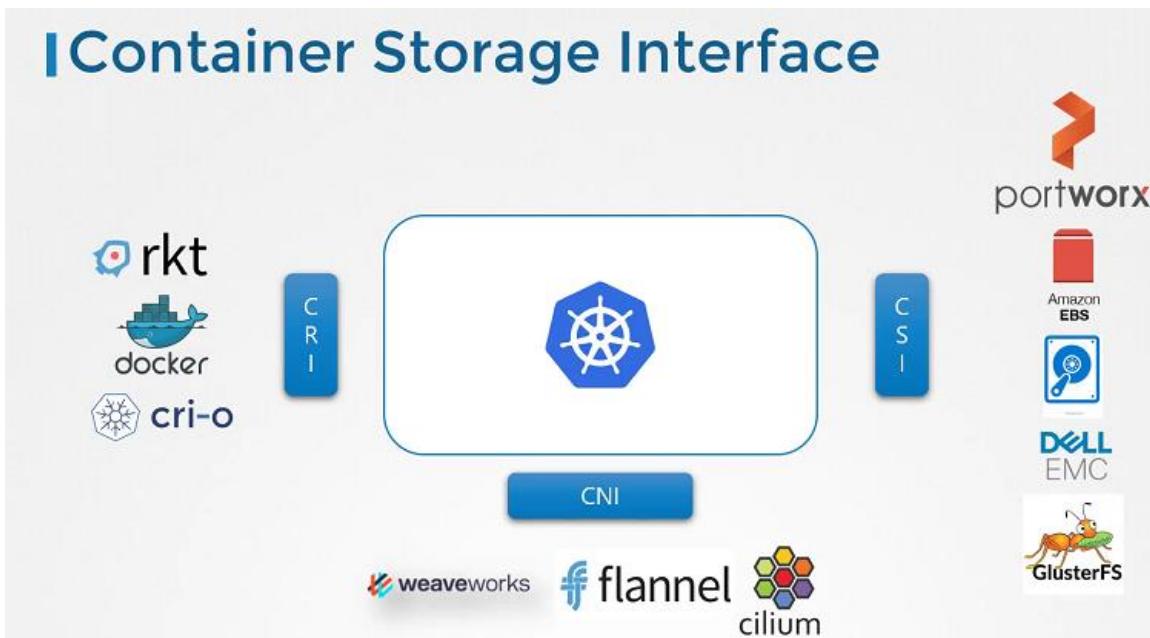
CNI



## Container Storage Interface

- The container storage interface was developed to support multiple storage solutions. With CSI, you can now write your own drivers for your own storage to work with Kubernetes. Portworx, Amazon EBS, Azure Disk, GlusterFS etc.

- CSI is not a Kubernetes specific standard. It is meant to be a universal standard and if implemented, allows any container orchestration tool to work with any storage vendor with a supported plugin. Kubernetes, Cloud Foundry and Mesos are onboard with CSI.
- It defines a set of RPCs or remote procedure calls that will be called by the container orchestrator. These must be implemented by the storage drivers.



### *Container Storage Interface*

- <https://github.com/container-storage-interface/spec>
- <https://kubernetes-csi.github.io/docs/>
- <http://mesos.apache.org/documentation/latest/csi/>
- <https://www.nomadproject.io/docs/internals/plugins/csi#volume-lifecycle>

## Volumes

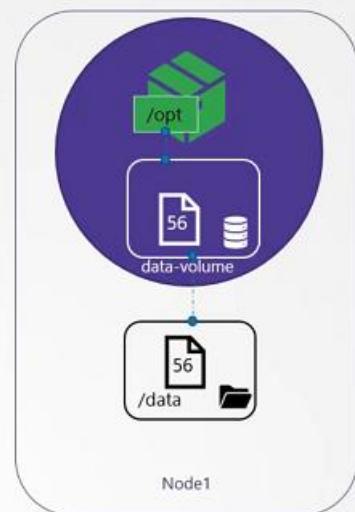
In this section, we will take a look at **Volumes**

- We discussed about Docker storage, If we don't attach the volume in the container runtime, when container destroyed and then all data will be lost. So, We need to persist data into the Docker container so we attach a volume to the containers when they are created.
- The data are processed by the container is now placed in this volume thereby retaining it permanently. Even if the container is deleted the data remains in the volume.

- In the Kubernetes world, the PODs created in Kubernetes are transient in nature. When a POD is created to process data and then deleted, the data processed by it gets deleted as well.
- For example, We create a simple POD that generates a random between 1 and 100 and writes that to a file at /opt/number.out. To persist into the volume.
- We create a volume for that. In this case I specify a path /data on the host. Files are stored in the directory data on my node. We use the volumeMounts field in each container to mount the data-volume to the directory /opt within the container. The random number will now be written to /opt mount inside the container, which happens to be on the data-volume which is in fact /data directory on the host. When the pod gets deleted, the file with the random number still lives on the host.

## Volumes & Mounts

```
apiVersion: v1
kind: Pod
metadata:
 name: random-number-generator
spec:
 containers:
 - image: alpine
 name: alpine
 command: ["/bin/sh", "-c"]
 args: ["shuf -i 0-100 -n 1 > /opt/number.out;"]
 volumeMounts:
 - mountPath: /opt
 name: data-volume
 volumes:
 - name: data-volume
 hostPath:
 path: /data
 type: Directory
```



### Volume Storage Options

- In the volumes, hostPath volume type is fine with the single node. It's not recommended for use with the multi node cluster.
- In the Kubernetes, it supports several types of standard storage solutions such as NFS, GlusterFS, CephFS or public cloud solutions like AWS EBS, Azure Disk or Google's Persistent Disk.

# Volume Types

```
volumes:
- name: data-volume
 hostPath:
 path: /data
 type: Directory
```



NFS



SCALEIO



```
volumes:
- name: data-volume
awsElasticBlockStore:
 volumeID: <volume-id>
 fsType: ext4
```

*Kubernetes Volumes Reference Docs*

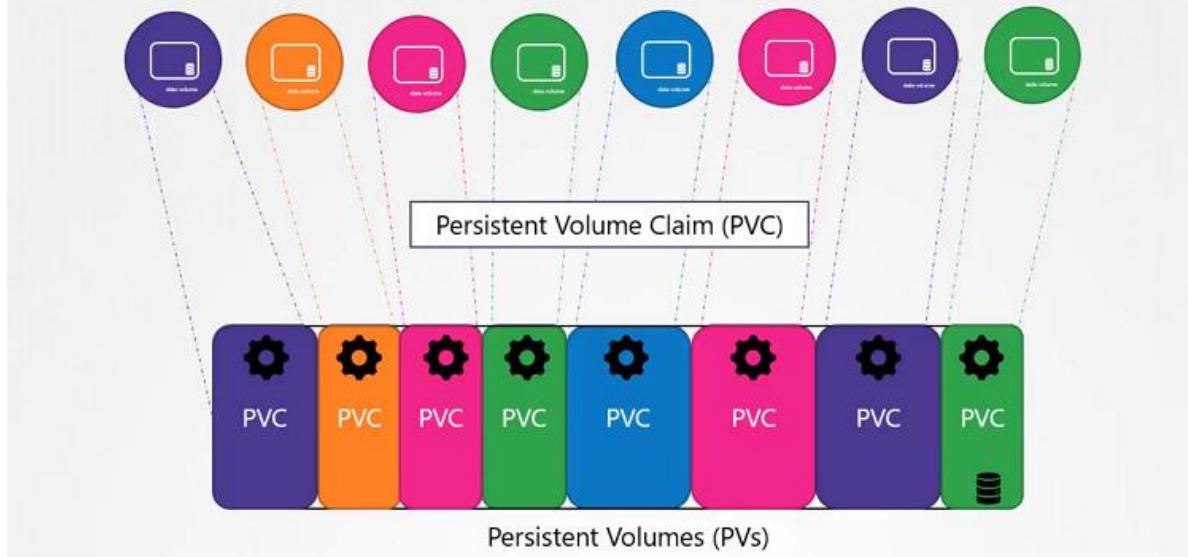
- <https://kubernetes.io/docs/concepts/storage/volumes/>
- <https://kubernetes.io/docs/tasks/configure-pod-container/configure-volume-storage/>
- <https://unofficial-kubernetes.readthedocs.io/en/latest/concepts/storage/volumes/>
- <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#volume-v1-core>

## Persistent Volumes

In this section, we will take a look at **Persistent Volumes**

- In the large environment, with a lot of users deploying a lot of pods, the users would have to configure storage every time for each Pod.
- Whatever storage solution is used, the users who deploys the pods would have to configure that on all pod definition files in his environment. Every time a change is to be made, the user would have to make them on all of his pods.

# Persistent Volume



- A Persistent Volume is a cluster-wide pool of storage volumes configured by an administrator to be used by users deploying application on the cluster. The users can now select storage from this pool using Persistent Volume Claims.
- `pv-definition.yaml`
- 
- `kind: PersistentVolume`
- `apiVersion: v1`
- `metadata:`
- `name: pv-vol1`
- `spec:`
- `accessModes: [ "ReadWriteOnce" ]`
- `capacity:`
- `storage: 1Gi`
- `hostPath:`
- `path: /tmp/data`
- `$ kubectl create -f pv-definition.yaml`
- `persistentvolume/pv-vol1 created`
- 
- `$ kubectl get pv`
- | NAME    | CAPACITY | ACCESS MODES | RECLAIM POLICY | STATUS    | CLAIM |
|---------|----------|--------------|----------------|-----------|-------|
| pv-vol1 | 1Gi      | RWO          | Retain         | Available |       |
|         | 3min     |              |                |           |       |
- 
- `$ kubectl delete pv pv-vol1`
- `persistentvolume "pv-vol1" deleted`

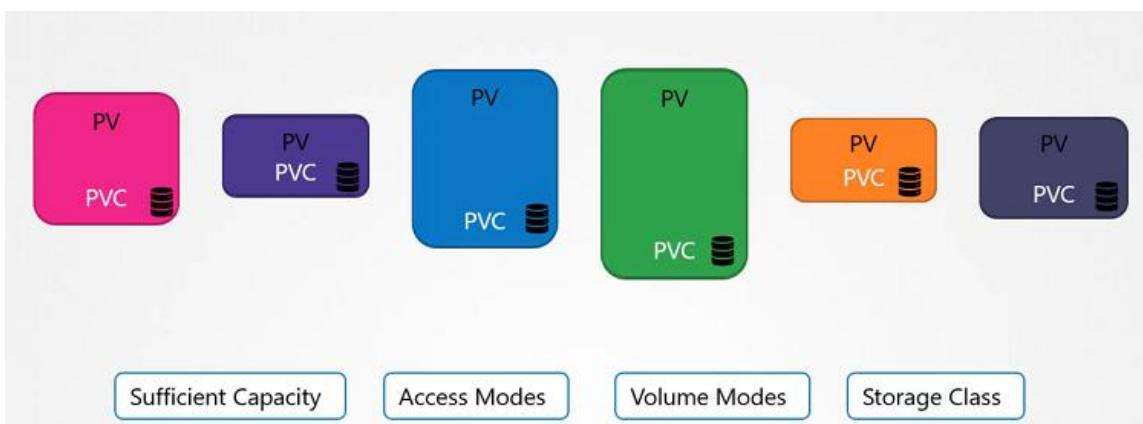
## Kubernetes Persistent Volumes

- <https://kubernetes.io/docs/concepts/storage/persistent-volumes/>
- <https://portworx.com/tutorial-kubernetes-persistent-volumes/>

# Persistent Volume Claims

In this section, we will take a look at **Persistent Volume Claim**

- Now we will create a Persistent Volume Claim to make the storage available to the node.
- Volumes and Persistent Volume Claim are two separate objects in the Kubernetes namespace.
- Once the Persistent Volume Claim created, Kubernetes binds the Persistent Volumes to claim based on the request and properties set on the volume.



- If properties not matches or Persistent Volume is not available for the Persistent Volume Claim then it will display the pending state.

```
pvc-definition.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: myclaim
spec:
 accessModes: ["ReadWriteOnce"]
 resources:
 requests:
 storage: 1Gi
```

```
pv-definition.yaml
```

```
kind: PersistentVolume
apiVersion: v1
metadata:
 name: pv-vol1
spec:
 accessModes: ["ReadWriteOnce"]
 capacity:
 storage: 1Gi
 hostPath:
 path: /tmp/data
```

*Create the Persistent Volume*

```
$ kubectl create -f pv-definition.yaml
persistentvolume/pv-vol1 created

$ kubectl get pv
NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM
STORAGECLASS REASON AGE
pv-vol1 1Gi RWO Retain Available
10s
```

*Create the Persistent Volume Claim*

```
$ kubectl create -f pvc-definition.yaml
persistentvolumeclaim/myclaim created

$ kubectl get pvc
NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE
myclaim Pending
35s

$ kubectl get pvc
NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE
myclaim Bound pv-vol1 1Gi RWO default 1min
```

*Delete the Persistent Volume Claim*

```
$ kubectl delete pvc myclaim
```

*Delete the Persistent Volume*

```
$ kubectl delete pv pv-vol1
```

*Kubernetes Persistent Volume Claims Reference Docs*

- <https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims>
- <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#persistentvolumeclaim-v1-core>
- <https://docs.cloud.oracle.com/en-us/iaas/Content/ContEng/Tasks/contengcreatingpersistentvolumeclaim.htm>

# Using PVC in PODs

In this section, we will take a look at **Using PVC in PODs**

- In this case, Pods access storage by using the claim as a volume. Persistent Volume Claim must exist in the same namespace as the Pod using the claim.

- The cluster finds the claim in the Pod's namespace and uses it to get the Persistent Volume backing the claim. The volume is then mounted to the host and into the Pod.
- Persistent Volume is a cluster-scoped and Persistent Volume Claim is a namespace-scoped.

*Create the Persistent Volume*

```
pv-definition.yaml

kind: PersistentVolume
apiVersion: v1
metadata:
 name: pv-vol1
spec:
 accessModes: ["ReadWriteOnce"]
 capacity:
 storage: 1Gi
 hostPath:
 path: /tmp/data
$ kubectl create -f pv-definition.yaml
```

*Create the Persistent Volume Claim*

```
pvc-definition.yaml

kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: myclaim
spec:
 accessModes: ["ReadWriteOnce"]
 resources:
 requests:
 storage: 1Gi
$ kubectl create -f pvc-definition.yaml
```

*Create a Pod*

```
pod-definition.yaml

apiVersion: v1
kind: Pod
metadata:
 name: mypod
spec:
 containers:
 - name: myfrontend
 image: nginx
 volumeMounts:
 - mountPath: "/var/www/html"
 name: web
 volumes:
 - name: web
 persistentVolumeClaim:
```

```
claimName: myclaim
$ kubectl create -f pod-definition.yaml
```

*List the Pod,Persistent Volume and Persistent Volume Claim*

```
$ kubectl get pod,pvc,pv
```

*References Docs*

- <https://kubernetes.io/docs/concepts/storage/persistent-volumes/#claims-as-volumes>

## Storage Class

In this section, we will take a look at **Storage Class**

- We discussed about how to create Persistent Volume and Persistent Volume Claim and We also saw that how to use into the Pod's volume to claim that volume space.
- We created Persistent Volume but before this if we are taking a volume from Cloud providers like GCP, AWS, Azure. We need to first create disk in the Google Cloud as an example.
- We need to create manually each time when we define in the Pod definition file. that's called **Static Provisioning**.

*Static Provisioning*

# | Static Provisioning

```
▶ gcloud beta compute disks create \
 --size 1GB
 --region us-east1
 pd-disk
```

pv-definition.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-voll
spec:
 accessModes:
 - ReadWriteOnce
 capacity:
 storage: 500Mi
 gcePersistentDisk:
 pdName: pd-disk
 fsType: ext4
```

PV

*Dynamic Provisioning*

# | Dynamic Provisioning



sc-definition.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: google-storage
provisioner: kubernetes.io/gce-pd
```

SC

pvc-definition.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: myclaim
spec:
 accessModes:
 - ReadWriteOnce
 storageClassName: google-storage
 resources:
 requests:
 storage: 500Mi
```

PV

PVC

pod-definition.yaml

```
apiVersion: v1
kind: Pod
metadata:
 name: random-number-generator
spec:
 containers:
 - image: alpine
 name: alpine
 command: ["/bin/sh","-c"]
 args: ["shuf -i 0-100 -n 1 >> /opt/volume"]
 volumeMounts:
 - mountPath: /opt
 name: data-volume
 volumes:
 - name: data-volume
 persistentVolumeClaim:
 claimName: myclaim
```



- No we have a Storage Class, So we no longer to define Persistent Volume. It will create automatically when a Storage Class is created. It's called **Dynamic Provisioning**.

```
sc-definition.yaml

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: google-storage
provisioner: kubernetes.io/gce-pd
```

*Create a Storage Class*

```
$ kubectl create -f sc-definition.yaml
storageclass.storage.k8s.io/google-storage created
```

*List the Storage Class*

```
$ kubectl get sc
NAME PROVISIONER RECLAIMPOLICY VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION AGE
google-storage kubernetes.io/gce-pd Delete Immediate false
20s
```

*Create a Persistent Volume Claim*

```
pvc-definition.yaml

kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: myclaim
spec:
 accessModes: ["ReadWriteOnce"]
 storageClassName: google-storage
 resources:
 requests:
 storage: 500Mi
$ kubectl create -f pvc-definition.yaml
```

*Create a Pod*

```
pod-definition.yaml

apiVersion: v1
kind: Pod
metadata:
 name: mypod
spec:
 containers:
 - name: frontend
 image: nginx
 volumeMounts:
 - mountPath: "/var/www/html"
 name: web
 volumes:
 - name: web
```

```

persistentVolumeClaim:
 claimName: myclaim
$ kubectl create -f pod-definition.yaml

```

*Provisioner*

The screenshot shows a presentation slide with the title 'Storage Class' in large blue font. Below the title is a code block titled 'sc-definition.yaml' containing YAML configuration for a StorageClass. To the right of the slide is a sidebar titled 'Volume Plugin' listing various storage providers.

```

sc-definition.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: google-storage
provisioner: kubernetes.io/gce-pd
parameters:
 type: pd-standard [pd-standard | pd-ssd]
 replication-type: none [none | regional-pd]

```

| Volume Plugin     |
|-------------------|
| AWSVolume         |
| AzureFile         |
| AzureDisk         |
| CephFS            |
| Cinder            |
| FC                |
| FlexVolume        |
| Flocker           |
| GCEPersistentDisk |
| Glusterfs         |
| iSCSI             |
| Quobyte           |
| NFS               |
| RBD               |
| VsphereVolume     |
| PortworxVolume    |
| ScaleIO           |
| StorageOS         |
| Local             |

*Kubernetes Storage Class Reference Docs*

- <https://kubernetes.io/docs/concepts/storage/storage-classes/>
- <https://cloud.google.com/kubernetes-engine/docs/concepts/persistent-volumes#storageclasses>
- <https://docs.aws.amazon.com/eks/latest/userguide/storage-classes.html>

## Section Introduction

In this section, we will take a look at **Networking Section**

- Prerequisite
  - Switching, Routing and Gateways
    - Switching
    - Routing
    - Default Gateway
  - DNS
    - DNS Configuration on Linux
  - CoreDNS
  - Network Namespace

- Docker Networking
- CNI
- Cluster Networking
- Pod Networking
- CNI in Kubernetes
- CoreDNS
- Ingress

## Pre-requisite Switching Routing Gateways

In this section, we will take a look at **Switching, Routing and Gateways**

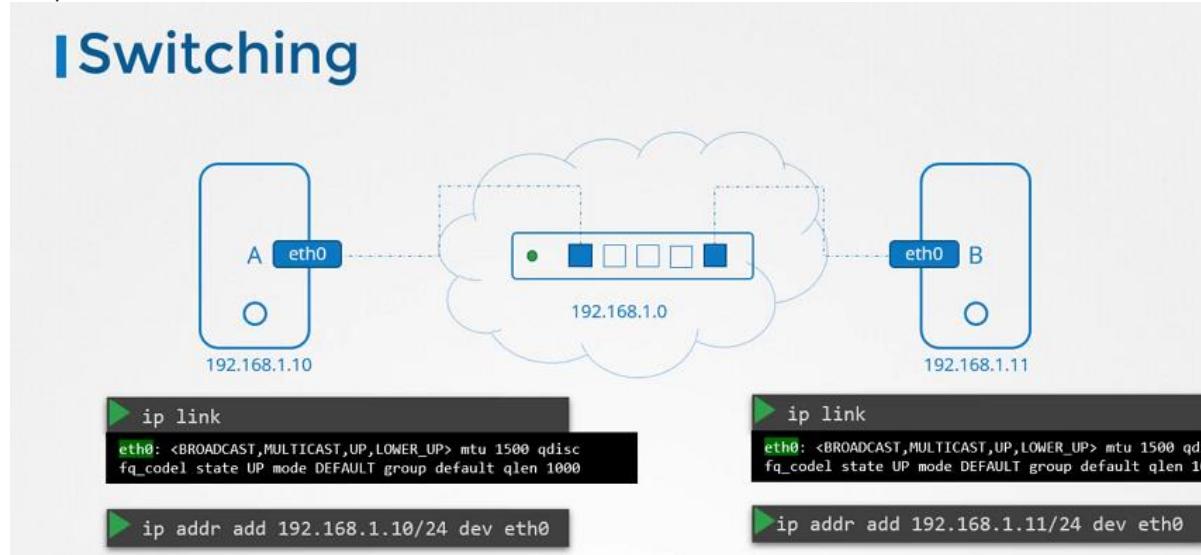
### Switching

- To see the interface on the host system

```
$ ip link
```

- To see the IP Address interfaces.

```
$ ip addr
```



### Routing

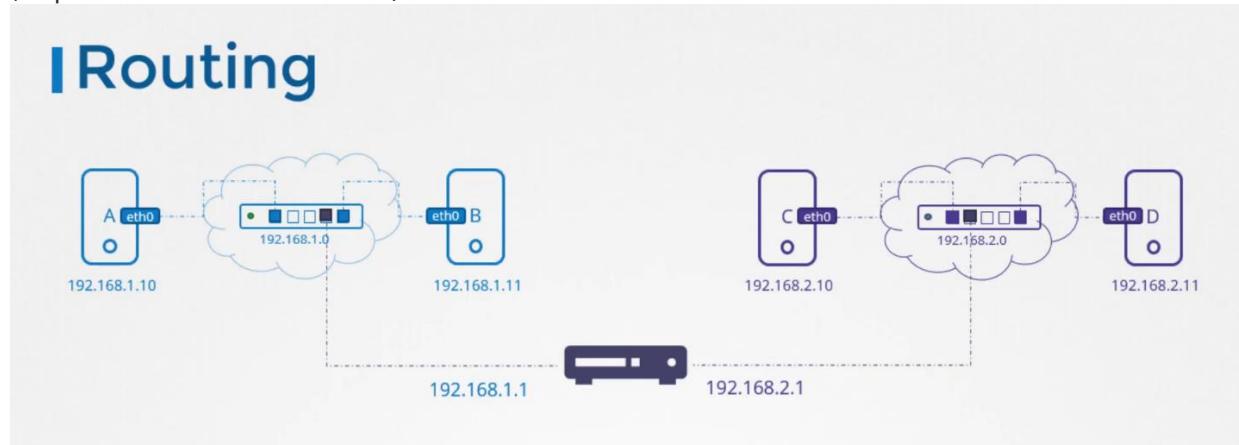
- To see the existing routing table on the host system.

```
$ route
$ ip route show
or
```

```
$ ip route list
```

- To add entries into the routing table.

```
$ ip route add 192.168.1.0/24 via 192.168.2.1
```



## Gateways

- To add a default route.

```
$ ip route add default via 192.168.2.1
```

- To check the IP forwarding is enabled on the host.

```
$ cat /proc/sys/net/ipv4/ip_forward
0
```

```
$ echo 1 > /proc/sys/net/ipv4/ip_forward
```

- Enable packet forwarding for IPv4.

```
$ cat /etc/sysctl.conf
```

```
Uncomment the line
net.ipv4.ip_forward=1
```

- To view the sysctl variables.

```
$ sysctl -a
```

- To reload the sysctl configuration.

```
$ sysctl --system
```

## Pre-requisite DNS

In this section, we will take a look at **DNS in the Linux**

## Name Resolution

- With help of the ping command. Checking the reachability of the IP Addr on the Network.

```
$ ping 172.17.0.64
PING 172.17.0.64 (172.17.0.64) 56(84) bytes of data.
64 bytes from 172.17.0.64: icmp_seq=1 ttl=64 time=0.384 ms
64 bytes from 172.17.0.64: icmp_seq=2 ttl=64 time=0.415 ms
```

- Checking with their hostname

```
$ ping web
ping: unknown host web
```

- Adding entry in the /etc/hosts file to resolve by their hostname.

```
$ cat >> /etc/hosts
172.17.0.64 web
```

```
Ctrl + c to exit
```

- It will look into the /etc/hosts file.

```
$ ping web
PING web (172.17.0.64) 56(84) bytes of data.
64 bytes from web (172.17.0.64): icmp_seq=1 ttl=64 time=0.491 ms
64 bytes from web (172.17.0.64): icmp_seq=2 ttl=64 time=0.636 ms
```

```
$ ssh web
```

```
$ curl http://web
```

## DNS

- Every host has a DNS resolution configuration file at /etc/resolv.conf.

```
$ cat /etc/resolv.conf
nameserver 127.0.0.53
options edns0
```

- To change the order of dns resolution, we need to do changes into the /etc/nsswitch.conf file.

```
$ cat /etc/nsswitch.conf
```

```
hosts: files dns
networks: files
```

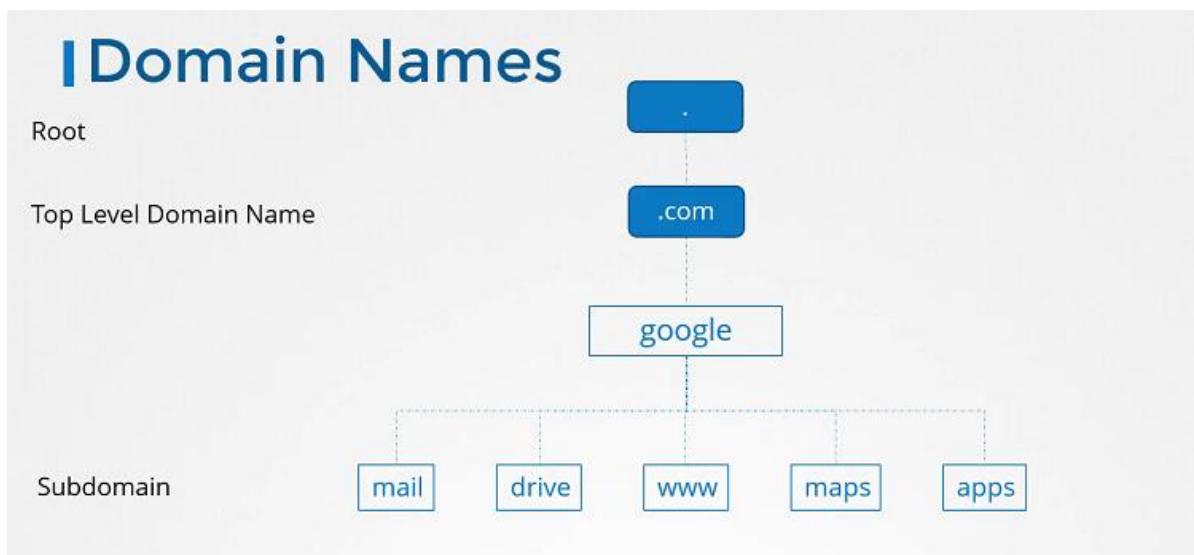
- If it fails in some conditions.

```
$ ping www.github.com
ping: www.github.com: Temporary failure in name resolution
```

- Adding well known public nameserver in the /etc/resolv.conf file.

```
$ cat /etc/resolv.conf
nameserver 127.0.0.53
nameserver 8.8.8.8
options edns0
$ ping www.github.com
PING github.com (140.82.121.3) 56(84) bytes of data.
64 bytes from 140.82.121.3 (140.82.121.3): icmp_seq=1 ttl=57 time=7.07 ms
64 bytes from 140.82.121.3 (140.82.121.3): icmp_seq=2 ttl=57 time=5.42 ms
```

## Domain Names



## Search Domain

The screenshot shows a search interface for a domain named "mycompany.com". At the top, there's a box labeled "Org DNS" with a small icon. Below it, a blue button says "mycompany.com". Underneath are several smaller buttons: "nfs", "web", "mail", "drive", "www", "pay", "hr", and "sql". To the right, a table lists IP addresses and their corresponding hostnames:

|              |                     |
|--------------|---------------------|
| 192.168.1.10 | web.mycompany.com   |
| 192.168.1.11 | db.mycompany.com    |
| 192.168.1.12 | nfs.mycompany.com   |
| 192.168.1.13 | web-1.mycompany.com |
| 192.168.1.14 | sql.mycompany.com   |

Below this, four terminal-like windows show command-line interactions:

- cat >> /etc/resolv.conf**: Shows the configuration of a nameserver to 192.168.1.100.
- ping web**: Shows a successful ping to the IP 192.168.1.10.
- ping web**: Shows a failed ping with the error "Temporary failure in name resolution".
- ping web.mycompany.com**: Shows a successful ping to the IP 192.168.1.10.

## Record Types

The screenshot shows a table of DNS record types:

|       |                 |                                         |
|-------|-----------------|-----------------------------------------|
| A     | web-server      | 192.168.1.1                             |
| AAAA  | web-server      | 2001:0db8:85a3:0000:0000:8a2e:0370:7334 |
| CNAME | food.web-server | eat.web-server, hungry.web-server       |

## Networking Tools

- Useful networking tools to test dns name resolution.

*nslookup*

```
$ nslookup www.google.com
Server: 127.0.0.53
Address: 127.0.0.53#53
```

Non-authoritative answer:

```
Name: www.google.com
Address: 172.217.18.4
```

Name: www.google.com

*dig*

```
$ dig www.google.com

; <>> DiG 9.11.3-1 ...
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 8738
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;www.google.com. IN A

;; ANSWER SECTION:
www.google.com. 63 IN A 216.58.206.4

;; Query time: 6 msec
;; SERVER: 127.0.0.53#53(127.0.0.53)
```

## Pre-requisite CoreDNS

In this section, we will take a look at **CoreDNS**

### Installation of CoreDNS

```
$ wget
https://github.com/coredns/coredns/releases/download/v1.7.0/coredns_1.7.0_linux_amd64.tgz
coredns_1.7.0_linux_amd64.tgz
```

### Extract tar file

```
$ tar -xzvf coredns_1.7.0_linux_amd64.tgz
coredns
```

### Run the executable file

- Run the executable file to start a DNS server. By default, it's listen on port 53, which is the default port for a DNS server.

```
$./coredns
```

### Configuring the hosts file

- Adding entries into the /etc/hosts file.

- CoreDNS will pick the ips and names from the /etc/hosts file on the server.

```
$ cat > /etc/hosts
192.168.1.10 web
192.168.1.11 db
192.168.1.15 web-1
192.168.1.16 db-1
192.168.1.21 web-2
192.168.1.22 db-2
```

### Adding into the Corefile

```
$ cat > Corefile
. {
 hosts /etc/hosts
}
```

### Run the executable file

```
$./coredns
```

### *References Docs*

- <https://github.com/kubernetes/dns/blob/master/docs/specification.md>
- <https://coredns.io/plugins/kubernetes/>
- <https://github.com/coredns/coredns/releases>

## Pre-requisite Network Namespaces

In this section, we will take a look at **Network Namespaces**

### Process Namespace

On the container

```
$ ps aux
```

On the host

```
$ ps aux
```

### Network Namespace

```
$ route
$ arp
```

## Create Network Namespace

```
$ ip netns add red
$ ip netns add blue
• List the network namespace
$ ip netns
```

## Exec in Network Namespace

- List the interfaces on the host

```
$ ip link
• Exec inside the network namespace
```

```
$ ip netns exec red ip link
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```
$ ip netns exec blue ip link
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

- You can try with other options as well. Both works the same.

```
$ ip -n red link
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

## ARP and Routing Table

### On the host

| Address     | Hwtype | Hwaddress         | Flags | Mask | Iface |
|-------------|--------|-------------------|-------|------|-------|
| 172.17.0.21 | ether  | 02:42:ac:11:00:15 | C     |      | ens3  |
| 172.17.0.55 | ether  | 02:42:ac:11:00:37 | C     |      | ens3  |

### On the Network Namespace

| Address | Hwtype | Hwaddress | Flags | Mask | Iface |
|---------|--------|-----------|-------|------|-------|
|---------|--------|-----------|-------|------|-------|

| Address | Hwtype | Hwaddress | Flags | Mask | Iface |
|---------|--------|-----------|-------|------|-------|
|---------|--------|-----------|-------|------|-------|

### On the host

```
$ route
```

### On the Network Namespace

```
$ ip netns exec red route
```

```
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
$ ip netns exec blue route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
```

## Virtual Cable

- To create a virtual cable

```
$ ip link add veth-red type veth peer name veth-blue
```

- To attach with the network namespaces

```
$ ip link set veth-red netns red
```

```
$ ip link set veth-blue netns blue
```

- To add an IP address

```
$ ip -n red addr add 192.168.15.1/24 dev veth-red
```

```
$ ip -n blue addr add 192.168.15.2/24 dev veth-blue
```

- To turn it up ns interfaces

```
$ ip -n red link set veth-red up
```

```
$ ip -n blue link set veth-blue up
```

- Check the reachability

```
$ ip netns exec red ping 192.168.15.2
PING 192.168.15.2 (192.168.15.2) 56(84) bytes of data.
64 bytes from 192.168.15.2: icmp_seq=1 ttl=64 time=0.035 ms
64 bytes from 192.168.15.2: icmp_seq=2 ttl=64 time=0.046 ms
```

```
$ ip netns exec red arp
Address HWtype HWaddress Flags Mask Iface
192.168.15.2 ether da:a7:29:c4:5a:45 C veth-red
```

```
$ ip netns exec blue arp
Address HWtype HWaddress Flags Mask Iface
192.168.15.1 ether 92:d1:52:38:c8:bc C veth-blue
```

- Delete the link.

```
$ ip -n red link del veth-red
```

On the host

```
Not available
```

```
$ arp
Address HWtype HWaddress Flags Mask Iface
172.16.0.72 ether 06:fe:61:1a:75:47 C ens3
172.17.0.68 ether 02:42:ac:11:00:44 C ens3
172.17.0.74 ether 02:42:ac:11:00:4a C ens3
172.17.0.75 ether 02:42:ac:11:00:4b C ens3
```

## Linux Bridge

- Create a network namespace

```
$ ip netns add red
```

```
$ ip netns add blue
```

- To create a internal virtual bridge network, we add a new interface to the host

```
$ ip link add v-net-0 type bridge
```

- Display in the host

```
$ ip link
8: v-net-0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT
group default qlen 1000
 link/ether fa:fd:d4:9b:33:66 brd ff:ff:ff:ff:ff:ff
```

- Currently it's down, so turn it up

```
$ ip link set dev v-net-0 up
```

- To connect network namespace to the bridge. Creating a virtual cable

```
$ ip link add veth-red type veth peer name veth-red-br
```

```
$ ip link add veth-blue type veth peer name veth-blue-br
```

- Set with the network namespaces

```
$ ip link set veth-red netns red
```

```
$ ip link set veth-blue netns blue
```

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

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

- To add an IP address

```
$ ip -n red addr add 192.168.15.1/24 dev veth-red
```

```
$ ip -n blue addr add 192.168.15.2/24 dev veth-blue
```

- To turn it up ns interfaces

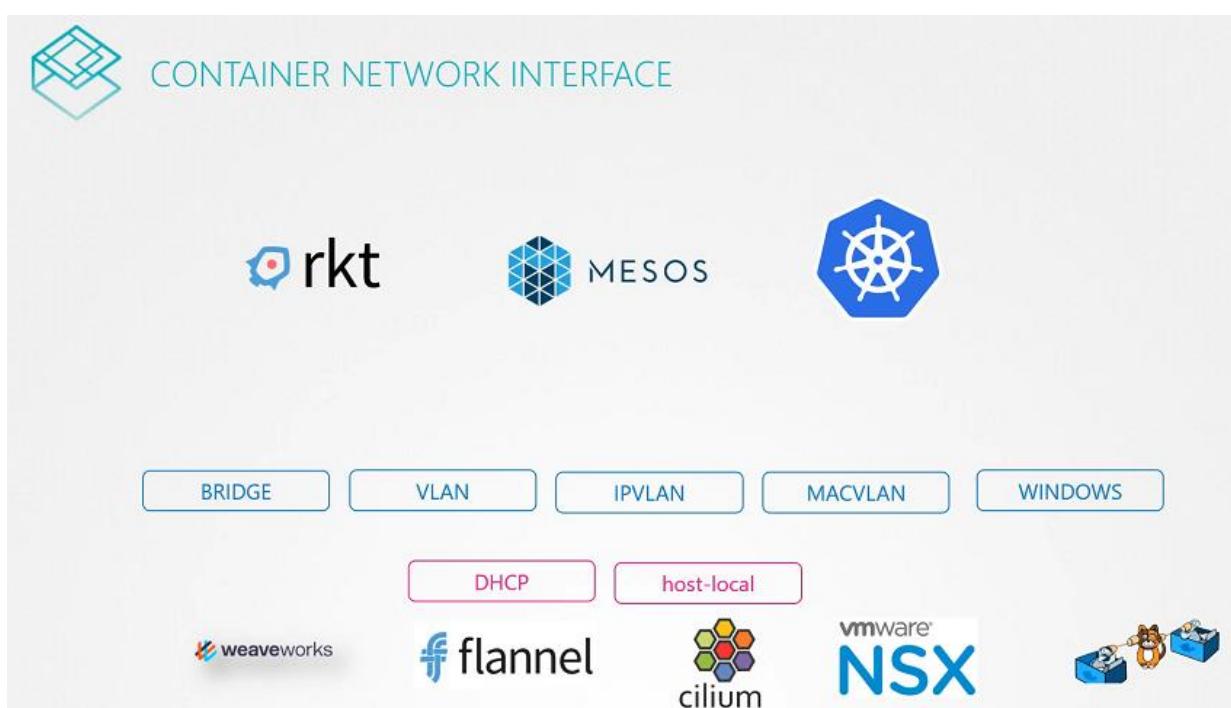
```
$ ip -n red link set veth-red up
$ ip -n blue link set veth-blue up
 • To add an IP address
$ ip addr add 192.168.15.5/24 dev v-net-0
 • Turn it up added interfaces on the host
$ ip link set dev veth-red-br up
$ ip link set dev veth-blue-br up
On the host
$ ping 192.168.15.1
On the ns
$ ip netns exec blue ping 192.168.1.1
Connect: Network is unreachable
$ ip netns exec blue route
$ ip netns exec blue ip route add 192.168.1.0/24 via 192.168.15.5
Check the IP Address of the host
$ ip a
$ ip netns exec blue ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.

$ iptables -t nat -A POSTROUTING -s 192.168.15.0/24 -j MASQUERADE
$ ip netns exec blue ping 192.168.1.1
$ ip netns exec blue ping 8.8.8.8
$ ip netns exec blue route
$ ip netns exec blue ip route add default via 192.168.15.5
$ ip netns exec blue ping 8.8.8.8
 • Adding port forwarding rule to the iptables
```

```
$ iptables -t nat -A PREROUTING --dport 80 --to-destination 192.168.15.2:80 -j DNAT
$ iptables -nvL -t nat
```

## Pre-requisite CNI

In this section, we will take a look at **Pre-requisite Container Network Interface(CNI)**



### Third Party Network Plugin Providers

- [Weave](#)
- [Calico](#)
- [Flannel](#)
- [Cilium](#)

To view the CNI Network Plugins

- CNI comes with the set of supported network plugins.

```
$ ls /opt/cni/bin/
bridge dhcp flannel host-device host-local ipvlan loopback macvlan portmap
ptp sample tuning vlan
```

*References Docs*

- <https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins/>

## Pre-requisite Cluster Networking

In this section, we will take a look at **Pre-requisite of the Cluster Networking**

- Set the unique hostname.
- Get the IP addr of the system (master and worker node).

- Check the Ports.

## IP and Hostname

- To view the hostname

```
$ hostname
```

- To view the IP addr of the system

```
$ ip a
```

## Set the hostname

```
$ hostnamectl set-hostname <host-name>
```

```
$ exec bash
```

## View the Listening Ports of the system

```
$ netstat -nltp
```

### *References Docs*

- <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/#check-required-ports>
- <https://kubernetes.io/docs/concepts/cluster-administration/networking/>

# Pod Networking

In this section, we will take a look at **Pod Networking**

- To add bridge network on each node

```
node01
$ ip link add v-net-0 type bridge
node02
$ ip link add v-net-0 type bridge
node03
$ ip link add v-net-0 type bridge
```

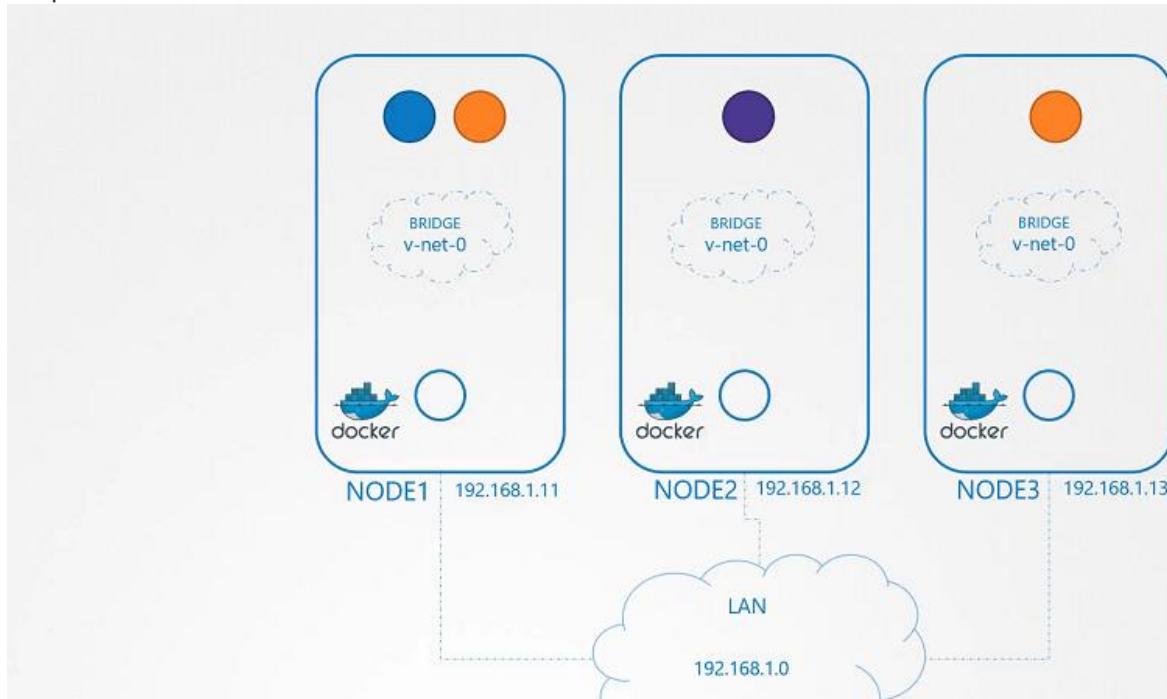
- Currently it's down, turn it up.

```
node01
$ ip link set dev v-net-0 up
node02
$ ip link set dev v-net-0 up
```

```
node03
$ ip link set dev v-net-0 up
```

- Set the IP Addr for the bridge interface

```
node01
$ ip addr add 10.244.1.1/24 dev v-net-0
node02
$ ip addr add 10.244.2.1/24 dev v-net-0
node03
$ ip addr add 10.244.3.1/24 dev v-net-0
```



- Check the reachability

```
$ ping 10.244.2.2
Connect: Network is unreachable
```

- Add route in the routing table

```
$ ip route add 10.244.2.2 via 192.168.1.12
node01
$ ip route add 10.244.2.2 via 192.168.1.12

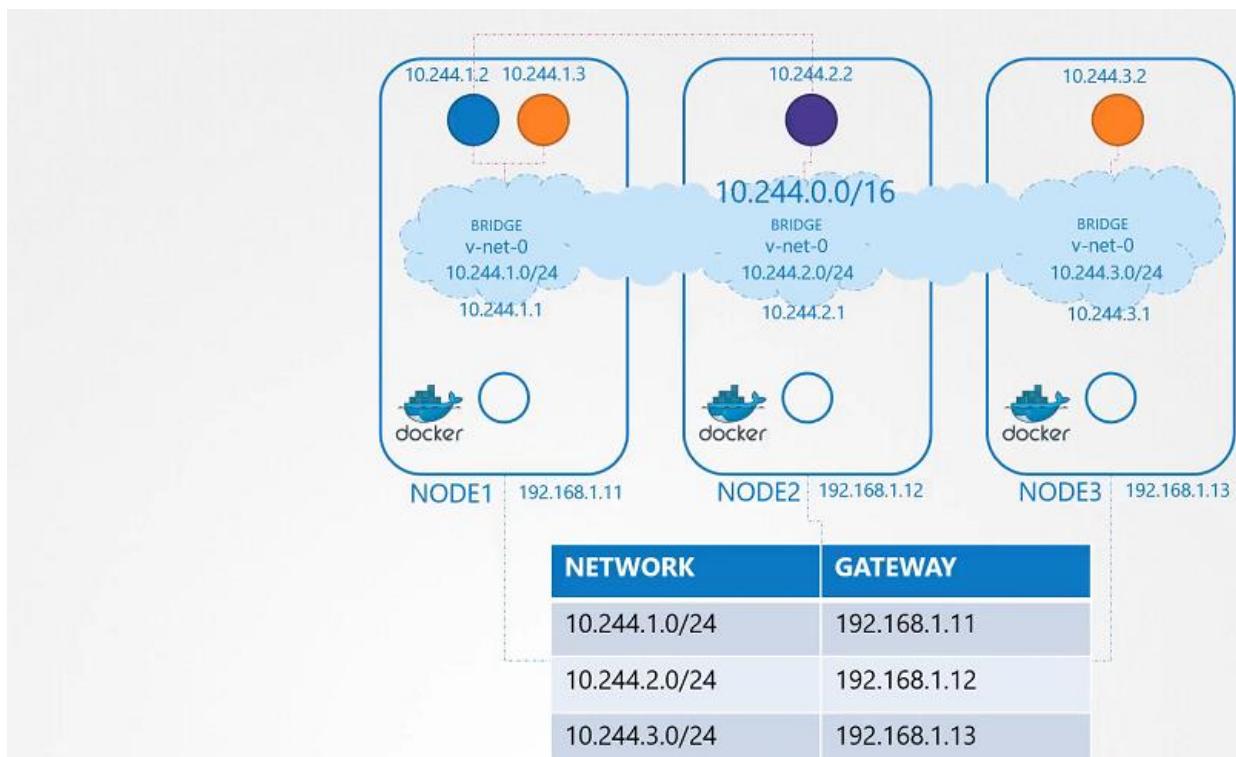
$ ip route add 10.244.3.2 via 192.168.1.13
node02
$ ip route add 10.244.1.2 via 192.168.1.11

$ ip route add 10.244.3.2 via 192.168.1.13

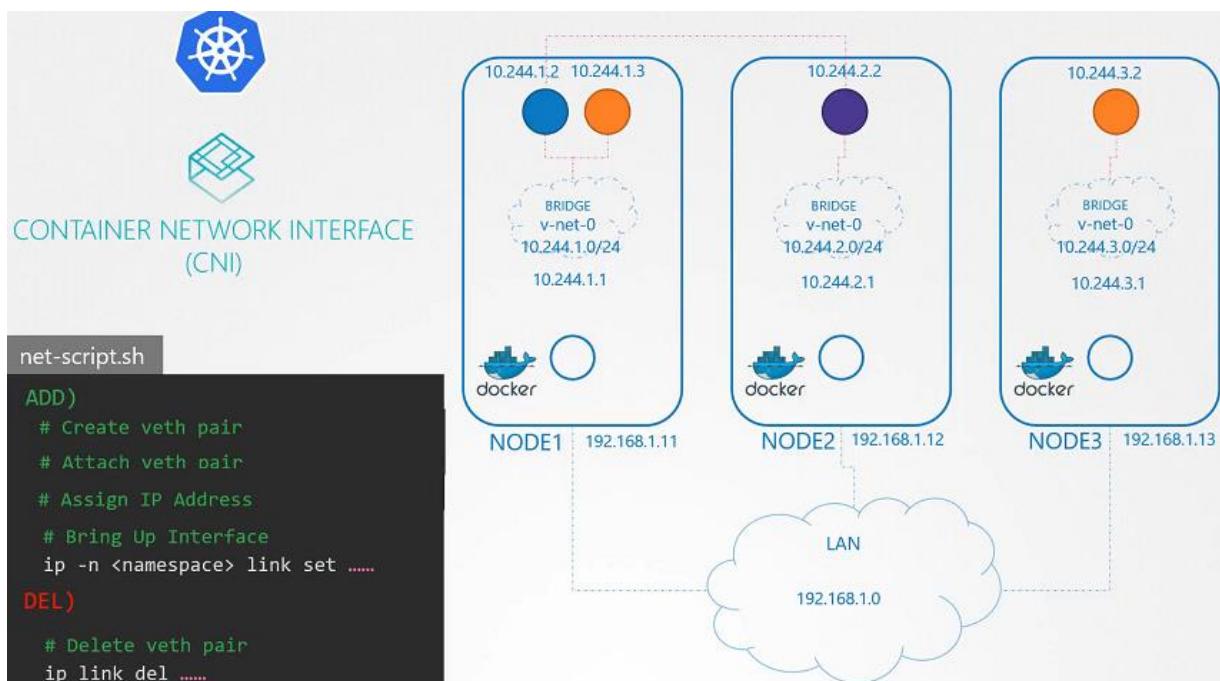
node03
$ ip route add 10.244.1.2 via 192.168.1.11
```

```
$ ip route add 10.244.2.2 via 192.168.1.12
```

- Add a single large network



## Container Network Interface (CNI)



*References Docs*

- <https://kubernetes.io/docs/concepts/workloads/pods/>

# CNI in Kubernetes

In this section, we will take a look at **Container Networking Interface (CNI) in Kubernetes**

## Configuring CNI

### |Configuring CNI

```
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 \\
--cni-bin-dir=/opt/cni/bin \\
--cni-conf-dir=/etc/cni/net.d \\
--register-node=true \\
--v=2
```

- Check the status of the Kubelet Service

```
$ systemctl status kubelet.service
```

## View Kubelet Options

```
$ ps -aux | grep kubelet
```

## Check the Supportable Plugins

- To check the all supportable plugins available in the /opt/cni/bin directory.

```
$ ls /opt/cni/bin
```

## Check the CNI Plugins

- To check the cnf plugins which kubelet needs to be used.

```
ls /etc/cni/net.d
```

Format of Configuration File

## View kubelet options

```
ls /etc/cni/net.d
10-bridge.conf

cat /etc/cni/net.d/10-bridge.conf
{
 "cniVersion": "0.2.0",
 "name": "mynet",
 "type": "bridge",
 "bridge": "cni0",
 "isGateway": true,
 "ipMasq": true,
 "ipam": {
 "type": "host-local",
 "subnet": "10.22.0.0/16",
 "routes": [
 { "dst": "0.0.0.0/0" }
]
 }
}
```

*References Docs*

- <https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/>
- <https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins/>

## CNI weave

In this section, we will take a look at "CNI Weave in the Kubernetes Cluster"

Deploy Weave

- Installing [weave net](#) onto the Kubernetes cluster with a single command.

```
$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"
serviceaccount/weave-net created
clusterrole.rbac.authorization.k8s.io/weave-net created
clusterrolebinding.rbac.authorization.k8s.io/weave-net created
role.rbac.authorization.k8s.io/weave-net created
rolebinding.rbac.authorization.k8s.io/weave-net created
daemonset.apps/weave-net created
```

## Weave Peers

```
$ kubectl get pods -n kube-system
NAME READY STATUS RESTARTS
AGE
coredns-66bff467f8-894jf 1/1 Running 0
52m
coredns-66bff467f8-nck5f 1/1 Running 0
52m
etcd-controlplane 1/1 Running 0
52m
kube-apiserver-controlplane 1/1 Running 0
52m
kube-controller-manager-controlplane 1/1 Running 0
52m
kube-keepalived-vip-mbr7d 1/1 Running 0
52m
kube-proxy-p2mld 1/1 Running 0
52m
kube-proxy-vjcwp 1/1 Running 0
52m
kube-scheduler-controlplane 1/1 Running 0
52m
weave-net-jgr8x 2/2 Running 0
45m
weave-net-tb9tz 2/2 Running 0
45m
```

## View the logs of Weave Pod's

```
$ kubectl logs weave-net-tb9tz weave -n kube-system
```

## View the default route in the Pod

```
$ kubectl run test --image=busybox --command -- sleep 4500
pod/test created
```

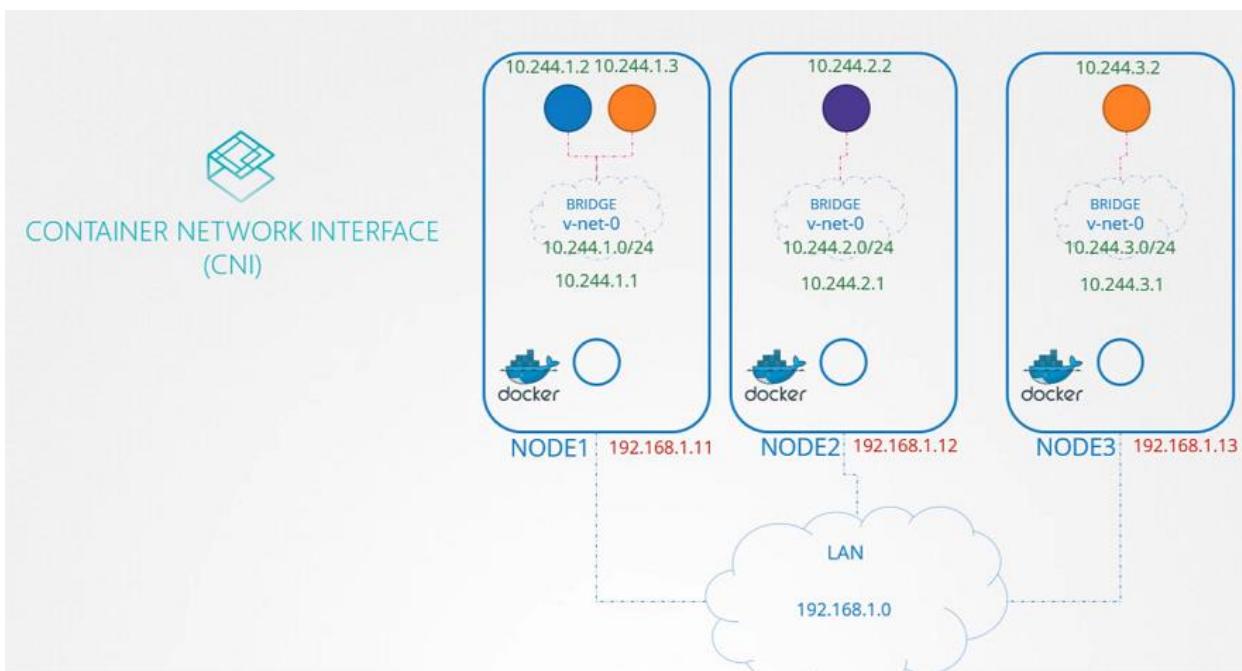
```
$ kubectl exec test -- ip route
default via 10.244.1.1 dev eth0
```

## References Docs

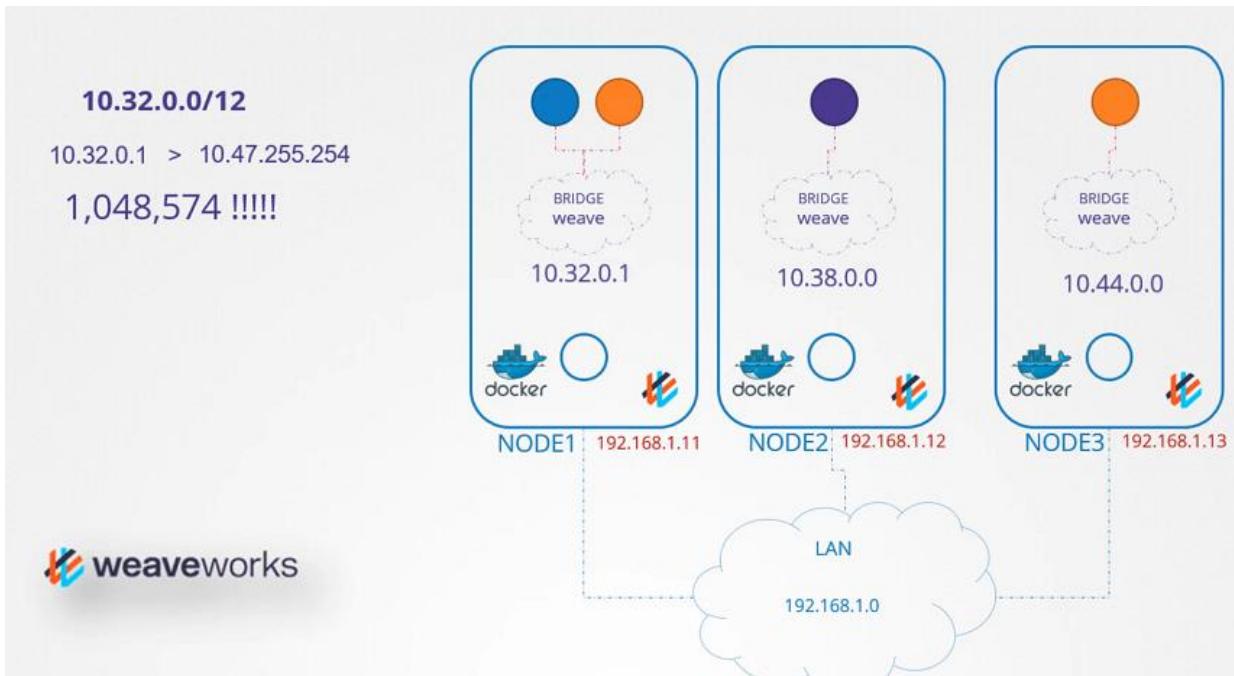
- <https://kubernetes.io/docs/concepts/cluster-administration/addons/>
- <https://www.weave.works/docs/net/latest/kubernetes/kube-addon/>

## IPAM weave

- IP Address Management in the Kubernetes Cluster



- How weaveworks Manages IP addresses in the Kubernetes Cluster



## References Docs

- <https://www.weave.works/docs/net/latest/kubernetes/kube-addon/>
- <https://kubernetes.io/docs/concepts/cluster-administration/networking/>

## DNS in Kubernetes

In this section, we will take a look at **DNS in the Kubernetes Cluster**

## Pod DNS Record

- The following DNS resolution:

<POD-IP-ADDRESS>.<namespace-name>.pod.cluster.local

**Example**

```
Pod is located in a default namespace
10-244-1-10.default.pod.cluster.local
To create a namespace
$ kubectl create ns apps

To create a Pod
$ kubectl run nginx --image=nginx --namespace apps

To get the additional information of the Pod in the namespace "apps"
$ kubectl get po -n apps -owide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE
READINESS GATES
nginx 1/1 Running 0 99s 10.244.1.3 node01 <none>
<none>

To get the dns record of the nginx Pod from the default namespace
$ kubectl run -it test --image=busybox:1.28 --rm --restart=Never -- nslookup 10-244-1-3.apps.pod.cluster.local
Server: 10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name: 10-244-1-3.apps.pod.cluster.local
Address 1: 10.244.1.3
pod "test" deleted

Accessing with curl command
$ kubectl run -it nginx-test --image=nginx --rm --restart=Never -- curl -Is http://10-244-1-3.apps.pod.cluster.local
HTTP/1.1 200 OK
Server: nginx/1.19.2
```

## Service DNS Record

- The following DNS resolution:

<service-name>.<namespace-name>.svc.cluster.local

**Example**

```
Service is located in a default namespace
web-service.default.svc.cluster.local
```

- Pod, Service is located in the apps namespace

```
Expose the nginx Pod
```

```
$ kubectl expose pod nginx --name=nginx-service --port 80 --namespace apps
service/nginx-service exposed

Get the nginx-service in the namespace "apps"
$ kubectl get svc -n apps
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
nginx-service ClusterIP 10.96.120.174 <none> 80/TCP 6s

To get the dns record of the nginx-service from the default namespace
$ kubectl run -it test --image=busybox:1.28 --rm --restart=Never -- nslookup
nginx-service.apps.svc.cluster.local
Server: 10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name: nginx-service.apps.svc.cluster.local
Address 1: 10.96.120.174 nginx-service.apps.svc.cluster.local
pod "test" deleted

Accessing with curl command
$ kubectl run -it nginx-test --image=nginx --rm --restart=Never -- curl -Is
http://nginx-service.apps.svc.cluster.local
HTTP/1.1 200 OK
Server: nginx/1.19.2
```

### References Docs

- <https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/>
- <https://kubernetes.io/docs/tasks/administer-cluster/dns-debugging-resolution/>

## CoreDNS in Kubernetes

In this section, we will take a look at **CoreDNS in the Kubernetes**

### To view the Pod

```
$ kubectl get pods -n kube-system
NAME READY STATUS RESTARTS AGE
coredns-66bff467f8-2vghh 1/1 Running 0 53m
coredns-66bff467f8-t5nzm 1/1 Running 0 53m
```

### To view the Deployment

```
$ kubectl get deployment -n kube-system
NAME READY UP-TO-DATE AVAILABLE AGE
coredns 2/2 2 2 53m
```

### To view the configmap of CoreDNS

```
$ kubectl get configmap -n kube-system
NAME DATA AGE
```

|         |   |     |
|---------|---|-----|
| coredns | 1 | 52m |
|---------|---|-----|

## CoreDNS Configuration File

```
$ kubectl describe cm coredns -n kube-system
```

Corefile:

```

.:53 {
 errors
 health { lameduck 5s
 }
 ready
 kubernetes cluster.local in-addr.arpa ip6.arpa {
 pods insecure
 fallthrough in-addr.arpa ip6.arpa
 ttl 30
 }
 prometheus :9153
 forward . /etc/resolv.conf
 cache 30
 loop
 reload
}
```

## To view the Service

```
$ kubectl get service -n kube-system
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kube-dns ClusterIP 10.96.0.10 <none> 53/UDP,53/TCP,9153/TCP 62m
```

## To view Configuration into the kubelet

```
$ cat /var/lib/kubelet/config.yaml | grep -A2 clusterDNS
clusterDNS:
- 10.96.0.10
clusterDomain: cluster.local
```

## To view the fully qualified domain name

- With the host command, we will get fully qualified domain name (FQDN).

```
$ host web-service
web-service.default.svc.cluster.local has address 10.106.112.101

$ host web-service.default
web-service.default.svc.cluster.local has address 10.106.112.101

$ host web-service.default.svc
web-service.default.svc.cluster.local has address 10.106.112.101

$ host web-service.default.svc.cluster.local
web-service.default.svc.cluster.local has address 10.106.112.101
```

To view the /etc/resolv.conf file

```
$ kubectl run -it --rm --restart=Never test-pod --image=busybox -- cat /etc/resolv.conf
nameserver 10.96.0.10
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
pod "test-pod" deleted
```

Resolve the Pod

```
$ kubectl get pods -o wide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED
NODE READINESS GATES
test-pod 1/1 Running 0 11m 10.244.1.3 node01 <none>
<none>
nginx 1/1 Running 0 10m 10.244.1.4 node01 <none>
<none>

$ kubectl exec -it test-pod -- nslookup 10-244-1-4.default.pod.cluster.local
Server: 10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name: 10-244-1-4.default.pod.cluster.local
Address 1: 10.244.1.4
```

Resolve the Service

```
$ kubectl get service
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 85m
web-service ClusterIP 10.106.112.101 <none> 80/TCP 9m

$ kubectl exec -it test-pod -- nslookup web-service.default.svc.cluster.local
Server: 10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name: web-service.default.svc.cluster.local
Address 1: 10.106.112.101 web-service.default.svc.cluster.local
```

*References Docs*

- <https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/#services>
- <https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/#pods>

## Ingress

In this section, we will take a look at **Ingress**

- Ingress Controller
- Ingress Resources

## Ingress Controller

- Deployment of **Ingress Controller**

### ConfigMap

```
kind: ConfigMap
apiVersion: v1
metadata:
 name: nginx-configuration
```

### Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: ingress-controller
spec:
 replicas: 1
 selector:
 matchLabels:
 name: nginx-ingress
 template:
 metadata:
 labels:
 name: nginx-ingress
 spec:
 serviceAccountName: ingress-serviceaccount
 containers:
 - name: nginx-ingress-controller
 image: quay.io/kubernetes-ingress-controller/nginx-ingress-
controller:0.21.0
 args:
 - /nginx-ingress-controller
 - --configmap=$(POD_NAMESPACE)/nginx-configuration
 env:
 - name: POD_NAME
 valueFrom:
 fieldRef:
 fieldPath: metadata.name
 - name: POD_NAMESPACE
 valueFrom:
 fieldRef:
 fieldPath: metadata.namespace
 ports:
 - name: http
 containerPort: 80
 - name: https
 containerPort: 443
```

### ServiceAccount

- ServiceAccount require for authentication purposes along with correct Roles, ClusterRoles and RoleBindings.

- Create a ingress service account

```
$ kubectl create -f ingress-sa.yaml
serviceaccount/ingress-serviceaccount created
```

## Service Type - NodePort

```
service-Nodeport.yaml
```

```
apiVersion: v1
kind: Service
metadata:
 name: ingress
spec:
 type: NodePort
 ports:
 - port: 80
 targetPort: 80
 protocol: TCP
 name: http
 - port: 443
 targetPort: 443
 protocol: TCP
 name: https
 selector:
 name: nginx-ingress
```

- Create a service

```
$ kubectl create -f service-Nodeport.yaml
```

- To get the service

```
$ kubectl get service
```

## Ingress Resources

Ingress-wear.yaml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
 name: ingress-wear
spec:
 backend:
 serviceName: wear-service
 servicePort: 80
```

- To create the ingress resource

```
$ kubectl create -f Ingress-wear.yaml
ingress.extensions/ingress-wear created
```

- To get the ingress

```
$ kubectl get ingress
NAME CLASS HOSTS ADDRESS PORTS AGE
ingress-wear <none> * 80 18s
```

## Ingress Resource - Rules

- 1 Rule and 2 Paths.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
 name: ingress-wear-watch
spec:
 rules:
 - http:
 paths:
 - path: /wear
 backend:
 serviceName: wear-service
 servicePort: 80
 - path: /watch
 backend:
 serviceName: watch-service
 servicePort: 80
```

- Describe the earlier created ingress resource

```
$ kubectl describe ingress ingress-wear-watch
Name: ingress-wear-watch
Namespace: default
Address:
Default backend: default-http-backend:80 (<none>)
Rules:
 Host Path Backends
 ---- --- -----
 *
 /wear wear-service:80 (<none>)
 /watch watch-service:80 (<none>)
Annotations: <none>
Events:
 Type Reason Age From Message
 ---- ----- -- -- --
 Normal CREATE 23s nginx-ingress-controller Ingress default/ingress-wear-watch
```

- 2 Rules and 1 Path each.

```
Ingress-wear-watch.yaml

apiVersion: extensions/v1beta1
kind: Ingress
metadata:
 name: ingress-wear-watch
spec:
 rules:
```

```

- host: wear.my-online-store.com
 http:
 paths:
 - backend:
 serviceName: wear-service
 servicePort: 80
- host: watch.my-online-store.com
 http:
 paths:
 - backend:
 serviceName: watch-service
 servicePort: 80

```

#### *References Docs*

- <https://kubernetes.io/docs/concepts/services-networking/ingress/>
- <https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/>
- <https://thenewstack.io/kubernetes-ingress-for-beginners/>

## Ingress Annotations and rewrite-target

In this section, we will take a look at **Ingress annotations and rewrite-target**

- Different Ingress controllers have different options to customize the way it works. Nginx Ingress Controller has many options but we will take a look into the one of the option "Rewrite Target" option.
- Kubernetes Version 1.18

```

apiVersion: extensions/v1beta1
kind: Ingress
metadata:
 name: test-ingress
 namespace: critical-space
 annotations:
 nginx.ingress.kubernetes.io/rewrite-target: /
spec:
 rules:
 - http:
 paths:
 - path: /pay
 backend:
 serviceName: pay-service
 servicePort: 8282

```

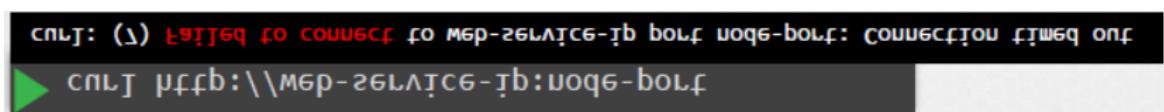
#### *Reference Docs*

- <https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/>
- <https://kubernetes.github.io/ingress-nginx/examples/>

- <https://kubernetes.github.io/ingress-nginx/examples/rewrite/>
- <https://github.com/kubernetes/ingress-nginx/blob/master/docs/troubleshooting.md>

## Application Failure

- In this lecture we will go step by step in troubleshooting Application failure.
- To check the Application/Service status of the webserver
- curl http://web-service-ip:node-port



- To check the endpoint of the service and compare it with the selectors
- kubectl describe service web-service

The left terminal window shows the output of the command 'kubectl describe service web-service'. The 'Endpoints' field is highlighted in green. The right terminal window shows the YAML configuration for a pod named 'webapp-mysql'.

```
apiVersion: v1
kind: Pod
metadata:
 name: webapp-mysql
 labels:
 app: example-app
 name: webapp-mysql
spec:
 containers:
 - name: webapp-mysql
 image: simple-webapp-mysql
 ports:
 - containerPort: 8080
```

- To check the status and logs of the pod
- kubectl get pod
- kubectl describe pod web
- kubectl logs web
- To check the logs of the previous pod
- kubectl logs web -f --previous

```
▶ kubectl get pod
NAME READY STATUS RESTARTS AGE
Web 1/1 Running 5 50m

▶ kubectl describe pod web
...
Events:
 Type Reason Age From Message
 ---- ----- -- -- --
 Normal Scheduled 52m default-scheduler Successfully assigned webapp-mysql to worker-1
 Normal Pulling 52m kubelet, worker-1 pulling image "simple-webapp-mysql"
 Normal Pulled 52m kubelet, worker-1 Successfully pulled image "simple-webapp-mysql"
 Normal Created 52m kubelet, worker-1 Created container
 Normal Started 52m kubelet, worker-1 Started container

▶ kubectl logs web -f --previous
10.32.0.1 -- [01/Apr/2019 12:51:55] "GET / HTTP/1.1" 200 -
10.32.0.1 -- [01/Apr/2019 12:51:55] "GET /static/img/success.jpg HTTP/1.1" 200 -
10.32.0.1 -- [01/Apr/2019 12:51:55] "GET /favicon.ico HTTP/1.1" 404 -
10.32.0.1 -- [01/Apr/2019 12:51:57] "GET / HTTP/1.1" 200 -
10.32.0.1 -- [01/Apr/2019 12:51:57] "GET / HTTP/1.1" 200 -
10.32.0.1 -- [01/Apr/2019 12:51:58] "GET / HTTP/1.1" 200 -
10.32.0.1 -- [01/Apr/2019 12:51:58] "GET / HTTP/1.1" 200 -
10.32.0.1 -- [01/Apr/2019 12:51:58] "GET / HTTP/1.1" 400 - Some Database Error application exiting!
```

### Hands on Labs

- Lets Troubleshoot the [Application](#)

## Control Plane Failure

- In this lecture we will use how to troubleshoot the Control Plane components.
- To check the status of the nodes if they are healthy
- `kubectl get nodes`
- To check the status of the pods if they are running
- `kubectl get pods`
- To check the status of all the pods of the Control Plane components(if they are deployed with kubeadm tool) and make sure they are **Running**
- `kubectl get pods -n kube-system`

| kubectl get pods -n kube-system |       |         |          |     |
|---------------------------------|-------|---------|----------|-----|
| NAME                            | READY | STATUS  | RESTARTS | AGE |
| coredns-78fcdf6894-5dntv        | 1/1   | Running | 0        | 1h  |
| coredns-78fcdf6894-knpzl        | 1/1   | Running | 0        | 1h  |
| etcd-master                     | 1/1   | Running | 0        | 1h  |
| kube-apiserver-master           | 1/1   | Running | 0        | 1h  |
| kube-controller-manager-master  | 1/1   | Running | 0        | 1h  |
| kube-proxy-fvbpj                | 1/1   | Running | 0        | 1h  |
| kube-proxy-v5r2t                | 1/1   | Running | 0        | 1h  |
| kube-scheduler-master           | 1/1   | Running | 0        | 1h  |
| weave-net-7kd52                 | 2/2   | Running | 1        | 1h  |
| weave-net-jt15m                 | 2/2   | Running | 1        | 1h  |

- If the Control Plane components are deployed as services then check the status of all the components

| service kube-apiserver status                                                                         |  |
|-------------------------------------------------------------------------------------------------------|--|
| ● kube-apiserver.service - Kubernetes API Server                                                      |  |
| Loaded: loaded (/etc/systemd/system/kube-apiserver.service; enabled; vendor preset: enabled)          |  |
| Active: active (running) since Wed 2019-03-20 07:57:25 UTC; 1 weeks 1 days ago                        |  |
| Docs: <a href="https://github.com/kubernetes/kubernetes">https://github.com/kubernetes/kubernetes</a> |  |
| Main PID: 15767 (kube-apiserver)                                                                      |  |
| Tasks: 13 (limit: 2362)                                                                               |  |
| service kube-controller-manager status                                                                |  |
| ● kube-controller-manager.service - Kubernetes Controller Manager                                     |  |
| Loaded: loaded (/etc/systemd/system/kube-controller-manager.service; enabled; vendor preset: enabled) |  |
| Active: active (running) since Wed 2019-03-20 07:57:25 UTC; 1 weeks 1 days ago                        |  |
| Docs: <a href="https://github.com/kubernetes/kubernetes">https://github.com/kubernetes/kubernetes</a> |  |
| Main PID: 15771 (kube-controller)                                                                     |  |
| Tasks: 10 (limit: 2362)                                                                               |  |
| service kube-scheduler status                                                                         |  |
| ● kube-scheduler.service - Kubernetes Scheduler                                                       |  |
| Loaded: loaded (/etc/systemd/system/kube-scheduler.service; enabled; vendor preset: enabled)          |  |
| Active: active (running) since Fri 2019-03-29 01:45:32 UTC; 11min ago                                 |  |
| Docs: <a href="https://github.com/kubernetes/kubernetes">https://github.com/kubernetes/kubernetes</a> |  |
| Main PID: 28390 (kube-scheduler)                                                                      |  |
| Tasks: 10 (limit: 2362)                                                                               |  |

- To check the status of **kube-apiserver**
- service kube-apiserver status
- To check the status of **kube-controller-manager**
- service kube-controller-manager status
- To check the status of **kube-scheduler**
- service kube-scheduler status

```
▶ service kubelet status
● kubelet.service - Kubernetes Kubelet
 Loaded: loaded (/etc/systemd/system/kubelet.service; enabled; vendor preset: enabled)
 Active: active (running) since Wed 2019-03-20 14:22:06 UTC; 1 weeks 1 days ago
 Docs: https://github.com/kubernetes/kubernetes
 Main PID: 1281 (kubelet)
 Tasks: 24 (limit: 1152)

▶ service kube-proxy status
● kube-proxy.service - Kubernetes Kube Proxy
 Loaded: loaded (/etc/systemd/system/kube-proxy.service; enabled; vendor preset: enabled)
 Active: active (running) since Wed 2019-03-20 14:21:54 UTC; 1 weeks 1 days ago
 Docs: https://github.com/kubernetes/kubernetes
 Main PID: 794 (kube-proxy)
 Tasks: 7 (limit: 1152)
```

- To check the status of **kubelet**
- `service kubelet status`
- To check the status of **kube-proxy** on the worker nodes.
- `service kube-proxy status`
- To check the logs of the Control Plane components deployed as Pods:
- `kubectl logs kube-apiserver-master -n kube-system`

```
▶ kubectl logs kube-apiserver-master -n kube-system
I0401 13:45:38.190735 1 server.go:703] external host was not specified, using 172.17.0.11
I0401 13:45:38.194290 1 server.go:145] Version: v1.11.3
I0401 13:45:38.819795 1 plugins.go:158] Loaded 8 mutating admission controller(s) successfully in the following order:
NamespaceLifecycle,LimitRanger,ServiceAccount,NodeRestriction,Priority,DefaultTolerationSeconds,DefaultStorageClass,MutatingAdmissionWebhook.
I0401 13:45:38.819794 1 plugins.go:161] Loaded 6 validating admission controller(s) successfully in the following order:
LimitRanger,ServiceAccount,Priority,PersistentVolumeClaimResize,ValidatingAdmissionWebhook,ResourceQuota.
I0401 13:45:38.821372 1 plugins.go:158] Loaded 8 mutating admission controller(s) successfully in the following order:
NamespaceLifecycle,LimitRanger,ServiceAccount,NodeRestriction,Priority,DefaultTolerationSeconds,DefaultStorageClass,MutatingAdmissionWebhook.
I0401 13:45:38.821410 1 plugins.go:161] Loaded 6 validating admission controller(s) successfully in the following order:
LimitRanger,ServiceAccount,Priority,PersistentVolumeClaimResize,ValidatingAdmissionWebhook,ResourceQuota.
I0401 13:45:38.985453 1 master.go:234] Using reconciler: lease
W0401 13:45:40.906380 1 genericapiserver.go:319] Skipping API batch/v1alpha1 because it has no resources.
W0401 13:45:41.370677 1 genericapiserver.go:319] Skipping API rbac.authorization.k8s.io/v1alpha1 because it has no resources.
W0401 13:45:41.381736 1 genericapiserver.go:319] Skipping API scheduling.k8s.io/v1alpha1 because it has no resources.
```

```
▶ sudo journalctl -u kube-apiserver
Mar 20 07:57:25 master-1 systemd[1]: Started Kubernetes API Server.
Mar 20 07:57:25 master-1 kube-apiserver[15767]: I0320 07:57:25.553377 15767 flags.go:33] FLAG: --address="127.0.0.1"
Mar 20 07:57:25 master-1 kube-apiserver[15767]: I0320 07:57:25.558273 15767 flags.go:33] FLAG: --admission-control={}
Mar 20 07:57:25 master-1 kube-apiserver[15767]: I0320 07:57:25.558325 15767 flags.go:33] FLAG: --admission-control-config-file=""
Mar 20 07:57:25 master-1 kube-apiserver[15767]: I0320 07:57:25.558339 15767 flags.go:33] FLAG: --advertise-address="192.168.5.11"
Mar 20 07:57:25 master-1 kube-apiserver[15767]: I0320 07:57:25.558353 15767 flags.go:33] FLAG: --allow-privileged="true"
Mar 20 07:57:25 master-1 kube-apiserver[15767]: I0320 07:57:25.558365 15767 flags.go:33] FLAG: --alsoLogToStderr="false"
Mar 20 07:57:25 master-1 kube-apiserver[15767]: I0320 07:57:25.558413 15767 flags.go:33] FLAG: --anonymous-auth="true"
Mar 20 07:57:25 master-1 kube-apiserver[15767]: I0320 07:57:25.558425 15767 flags.go:33] FLAG: --api-audiences={}
Mar 20 07:57:25 master-1 kube-apiserver[15767]: I0320 07:57:25.558442 15767 flags.go:33] FLAG: --apiserver-count="3"
Mar 20 07:57:25 master-1 kube-apiserver[15767]: I0320 07:57:25.558454 15767 flags.go:33] FLAG: --audit-dynamic-configuration="false"
Mar 20 07:57:25 master-1 kube-apiserver[15767]: I0320 07:57:25.558464 15767 flags.go:33] FLAG: --audit-log-batch-buffer-size="10000"
Mar 20 07:57:25 master-1 kube-apiserver[15767]: I0320 07:57:25.558474 15767 flags.go:33] FLAG: --audit-log-batch-max-size="1"
Mar 20 07:57:25 master-1 kube-apiserver[15767]: I0320 07:57:25.558484 15767 flags.go:33] FLAG: --audit-log-batch-max-wait="0s"
Mar 20 07:57:25 master-1 kube-apiserver[15767]: I0320 07:57:25.558495 15767 flags.go:33] FLAG: --audit-log-batch-throttle-burst="0"
Mar 20 07:57:25 master-1 kube-apiserver[15767]: I0320 07:57:25.558504 15767 flags.go:33] FLAG: --audit-log-batch-throttle-enable="false"
Mar 20 07:57:25 master-1 kube-apiserver[15767]: I0320 07:57:25.558514 15767 flags.go:33] FLAG: --audit-log-batch-throttle-qps="0"
Mar 20 07:57:25 master-1 kube-apiserver[15767]: I0320 07:57:25.558528 15767 flags.go:33] FLAG: --audit-log-format="json"
```

- To check the logs of the Control Plane components deployed as SystemD Service

`sudo journalctl -u kube-apiserver`

## Worker Node Failure

- Lets check the status of the Nodes in the cluster, are they **Ready** OR **NotReady**
- `kubectl get nodes`

- If they are **NotReady** then check the **LastHeartbeatTime** of the node to find out the time when node might have crashed
- `kubectl describe node worker-1`

```
▶ kubectl describe node worker-1
...
Conditions:
 Type Status LastHeartbeatTime Reason Message
 ---- ----- --------------------- ----- -----
 OutOfDisk Unknown Mon, 01 Apr 2019 14:20:20 +0000 NodeStatusUnknown Kubelet stopped posting node status.
 MemoryPressure Unknown Mon, 01 Apr 2019 14:20:20 +0000 NodeStatusUnknown Kubelet stopped posting node status.
 DiskPressure Unknown Mon, 01 Apr 2019 14:20:20 +0000 NodeStatusUnknown Kubelet stopped posting node status.
 PIDPressure False Mon, 01 Apr 2019 14:20:20 +0000 KubeletHasSufficientPID Kubelet has sufficient PID available
 Ready Unknown Mon, 01 Apr 2019 14:20:20 +0000 NodeStatusUnknown Kubelet stopped posting node status.
```

- Check the possible **CPU** and **MEMORY** using `top` and `df -h`

```
▶ top
top - 14:43:56 up 3 days, 19:02, 1 user, load average: 0.35, 0.29, 0.21
Tasks: 112 total, 1 running, 72 sleeping, 0 stopped, 0 zombie
%Cpu(s): 3.9 us, 1.7 sy, 0.1 ni, 94.3 id, 0.0 wa, 0.0 hi, 0.1 si, 0.0 st
KiB Mem : 1089112 total, 74144 free, 736688 used, 198360 buff/cache
KiB Swap: 0 total, 0 free, 0 used. 129244 avail Mem

 PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
 34 root 20 0 0 0 0 S 5.9 0.0 0:13.14 kswapd0
28826 999 20 0 1361320 383208 3596 S 5.9 38.0 0:46.95 mysqld
 1 root 20 0 78260 5924 3192 S 0.0 0.6 0:21.88 systemd
 2 root 20 0 0 0 0 S 0.0 0.0 0:00.02 kthreadd
 4 root 20 -20 0 0 0 I 0.0 0.0 0:00.00 kworker/0:0H

▶ df -h
Filesystem Size Used Avail Use% Mounted on
udev 481M 481M 0 100% /dev
tmpfs 99M 1000K 98M 1% /run
/dev/sdal 9.7G 5.3G 4.5G 55% /
tmpfs 493M 0 493M 0% /dev/shm
tmpfs 5.8M 0 5.8M 0% /run/lock
tmpfs 493M 0 493M 0% /sys/fs/cgroup
tmpfs 99M 0 99M 0% /run/user/1000
```

- Check the status and the logs of the **kubelet** for the possible issues.
- `service kubelet status`
- `sudo journalctl -u kubelet`

```
▶ service kubelet status
● kubelet.service - Kubernetes Kubelet
 Loaded: loaded (/etc/systemd/system/kubelet.service; enabled; vendor preset: enabled)
 Active: active (running) since Wed 2019-03-20 14:22:06 UTC; 1 weeks 1 days ago
 Docs: https://github.com/kubernetes/kubernetes
 Main PID: 1281 (kubelet)
 Tasks: 24 (limit: 1152)

▶ sudo journalctl -u kubelet
-- Logs begin at Wed 2019-03-20 05:30:37 UTC, end at Mon 2019-04-01 14:42:42 UTC. --
Mar 20 08:12:59 worker-1 systemd[1]: Started Kubernetes Kubelet.
Mar 20 08:12:59 worker-1 kubelet[18962]: Flag --tls-cert-file has been deprecated, This parameter should be set via the config file specified by the Kubelet
Mar 20 08:12:59 worker-1 kubelet[18962]: Flag --tls-private-key-file has been deprecated, This parameter should be set via the config file specified by the
Mar 20 08:12:59 worker-1 kubelet[18962]: I0320 08:12:59.915179 18962 flags.go:33] FLAG: --address="0.0.0.0"
Mar 20 08:12:59 worker-1 kubelet[18962]: I0320 08:12:59.918149 18962 flags.go:33] FLAG: --allow-privileged=true"
Mar 20 08:12:59 worker-1 kubelet[18962]: I0320 08:12:59.918339 18962 flags.go:33] FLAG: --allowed-unsafe-systls=[]
Mar 20 08:12:59 worker-1 kubelet[18962]: I0320 08:12:59.918502 18962 flags.go:33] FLAG: --alsologtostderr=false"
Mar 20 08:12:59 worker-1 kubelet[18962]: I0320 08:12:59.918648 18962 flags.go:33] FLAG: --anonymous-auth=true"
Mar 20 08:12:59 worker-1 kubelet[18962]: I0320 08:12:59.918841 18962 flags.go:33] FLAG: --application-metrics-count-limit=100"
Mar 20 08:12:59 worker-1 kubelet[18962]: I0320 08:12:59.918974 18962 flags.go:33] FLAG: --authentication-token-webhook=false"
Mar 20 08:12:59 worker-1 kubelet[18962]: I0320 08:12:59.919096 18962 flags.go:33] FLAG: --authorization-token-webhook-cache-ttl=2m0s"
Mar 20 08:12:59 worker-1 kubelet[18962]: I0320 08:12:59.919299 18962 flags.go:33] FLAG: --authorization-mode=AlwaysAllow"
Mar 20 08:12:59 worker-1 kubelet[18962]: I0320 08:12:59.919466 18962 flags.go:33] FLAG: --authorization-webhook-cache-authorized-ttl=5m0s"
Mar 20 08:12:59 worker-1 kubelet[18962]: I0320 08:12:59.919598 18962 flags.go:33] FLAG: --authorization-webhook-cache-unauthorized-ttl=30s"
Mar 20 08:12:59 worker-1 kubelet[18962]: I0320 08:12:59.919791 18962 flags.go:33] FLAG: --azure-container-registry-config=""
Mar 20 08:12:59 worker-1 kubelet[18962]: I0320 08:12:59.919971 18962 flags.go:33] FLAG: --boot-id-file=/proc/sys/kernel/random/boot_id"
Mar 20 08:12:59 worker-1 kubelet[18962]: I0320 08:12:59.920102 18962 flags.go:33] FLAG: --bootstrap-checkpoint-path=""
```

- Check the **kubelet** Certificates, they are not expired, and in the right group and issued by the right CA.
- `openssl x509 -in /var/lib/kubelet/worker-1.crt -text`

```
▶ openssl x509 -in /var/lib/kubelet/worker-1.crt -text
Certificate:
Data:
 Version: 3 (0x2)
 Serial Number:
 ff:e0:23:9d:fc:78:03:35
Signature Algorithm: sha256WithRSAEncryption
 Issuer: CN = KUBERNETES-CA
 Validity
 Not Before: Mar 20 08:09:29 2019 GMT
 Not After : Apr 19 08:09:29 2019 GMT
 Subject: CN = system:node:worker-1, O = system:nodes
 Subject Public Key Info:
 Public Key Algorithm: rsaEncryption
 Public-Key: (2048 bit)
 Modulus:
 00:b4:28:0c:60:71:41:06:14:46:d9:97:58:2d:fe:
 a9:c7:6d:51:cd:1c:98:b9:5e:e6:e4:02:d3:e3:71:
 58:a1:60:fe:cb:e7:9b:4b:86:04:67:b5:4f:da:d6:
 6c:08:3f:57:e9:70:59:57:48:6a:ce:e5:d4:f3:6e:
 b2:fa:8a:18:7e:21:60:35:8f:44:f7:a9:39:57:16:
 4f:4e:1e:b1:a3:77:32:c2:ef:d1:38:b4:82:20:8f:
 11:0e:79:c4:d1:9b:f6:82:c4:08:84:84:68:d5:c3:
 e2:15:a0:ce:23:3c:8d:9c:b8:dd:fc:3a:cd:42:ae:
 5e:1b:80:2d:1b:e5:5d:1b:c1:fb:be:a3:9e:82:ff:
 a1:27:c8:b6:0f:3c:cb:11:f9:1a:9b:d2:39:92:0e:
 47:45:b8:8f:98:13:c6:4d:6a:18:75:a4:01:6f:73:
 f6:f8:7f:eb:5d:59:94:46:d8:da:37:75:cf:27:0b:
 39:7f:48:20:c5:fd:c7:a7:ce:22:9a:33:4a:30:1d:
 95:ef:00:bd:fe:47:22:42:44:99:77:5a:c4:97:bb:
 37:93:7c:33:64:f4:b8:3a:53:8c:f4:10:db:7f:5f:
 2b:89:18:d6:0e:68:51:34:29:b1:f1:61:6b:4b:c6:
 8:79:30:76:5:86:44:68:51:4:51:86:55:79:
```

## Advance Kubectl Commands

- To get the output of `kubectl` in a json format:
- `kubectl get nodes -o json`
- `kubectl get pods -o json`

```
{
 "apiVersion": "v1",
 "kind": "List",
 "items": [
 {
 "apiVersion": "v1",
 "kind": "Pod",
 "metadata": {
 "name": "nginx-5557945897-gznjp",
 },
 "spec": {
 "containers": [
 {
 "image": "nginx:alpine",
 "name": "nginx"
 }
],
 "nodeName": "node01"
 }
 }
]
}
```

- To get the image name used by pod via json path query:
- `kubectl get pods -o=jsonpath='{.items[0].spec.containers[0].image}'`
- To get the names of node in the cluster:
- `kubectl get pods -o=jsonpath='{.items[*].metadata.name}'`

```
{
 "apiVersion": "v1",
 "items": [
 {
 "apiVersion": "v1",
 "kind": "Node",
 "metadata": {
 "name": "master"
 },
 "status": {
 "capacity": {
 "cpu": "4"
 },
 "nodeInfo": {
 "architecture": "amd64",
 "operatingSystem": "linux",
 }
 }
 },
 {
 "apiVersion": "v1",
 "kind": "Node",
 "metadata": {
 "name": "node01",
 },
 "status": {
 "capacity": {
 "cpu": "4",
 },
 "nodeInfo": {
 "architecture": "amd64",
 "operatingSystem": "linux",
 }
 }
 }
]
}
```

- To get the architecture of node in the cluster:
- `kubectl get pods -o=jsonpath='{{.items[*].status.nodeInfo.architecture}}`'
- To get the count of the cpu of node in the cluster:
- `kubectl get pods -o=jsonpath='{{.items[*].status.status.capacity.cpu}}`'

*Loops - Range*

- To print the output in a separate column (one column with node name and other with CPU count):
- `kubectl get nodes -o=custom-columns=NODE:.metadata.name,CPU:.status.capacity.cpu`

| NODE   | CPU |
|--------|-----|
| master | 4   |
| node01 | 4   |

- Kubectl comes with a `sort_by` property which can be combined with json path query to `sort_by name` or `CPU count`
- `kubectl get nodes --sort-by=.metadata.name`

| NODE   | CPU |
|--------|-----|
| master | 4   |
| node01 | 4   |

```
kubectl get nodes --sort-by=..status.capacity.cpu
```