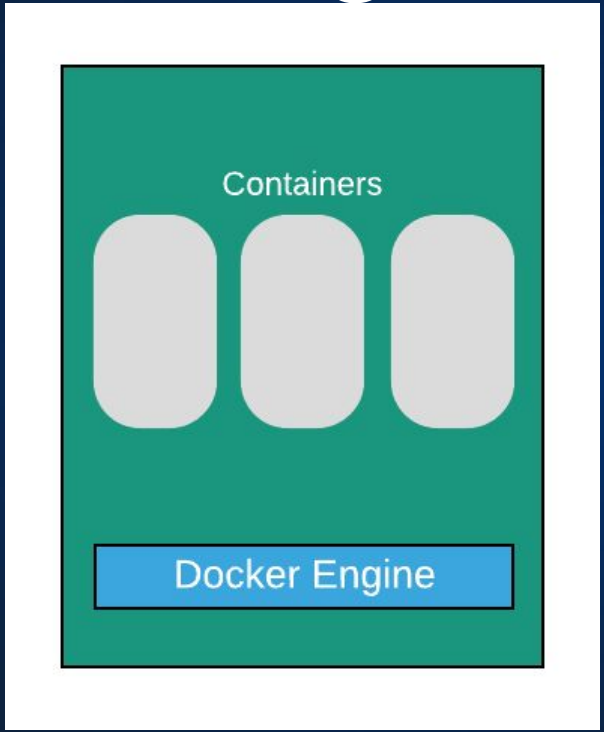
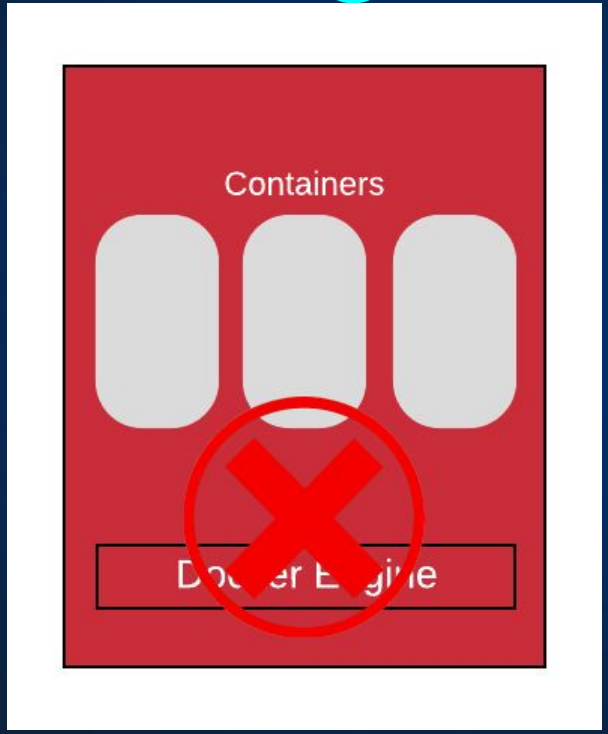
Kubernetes

Run Containers for Production

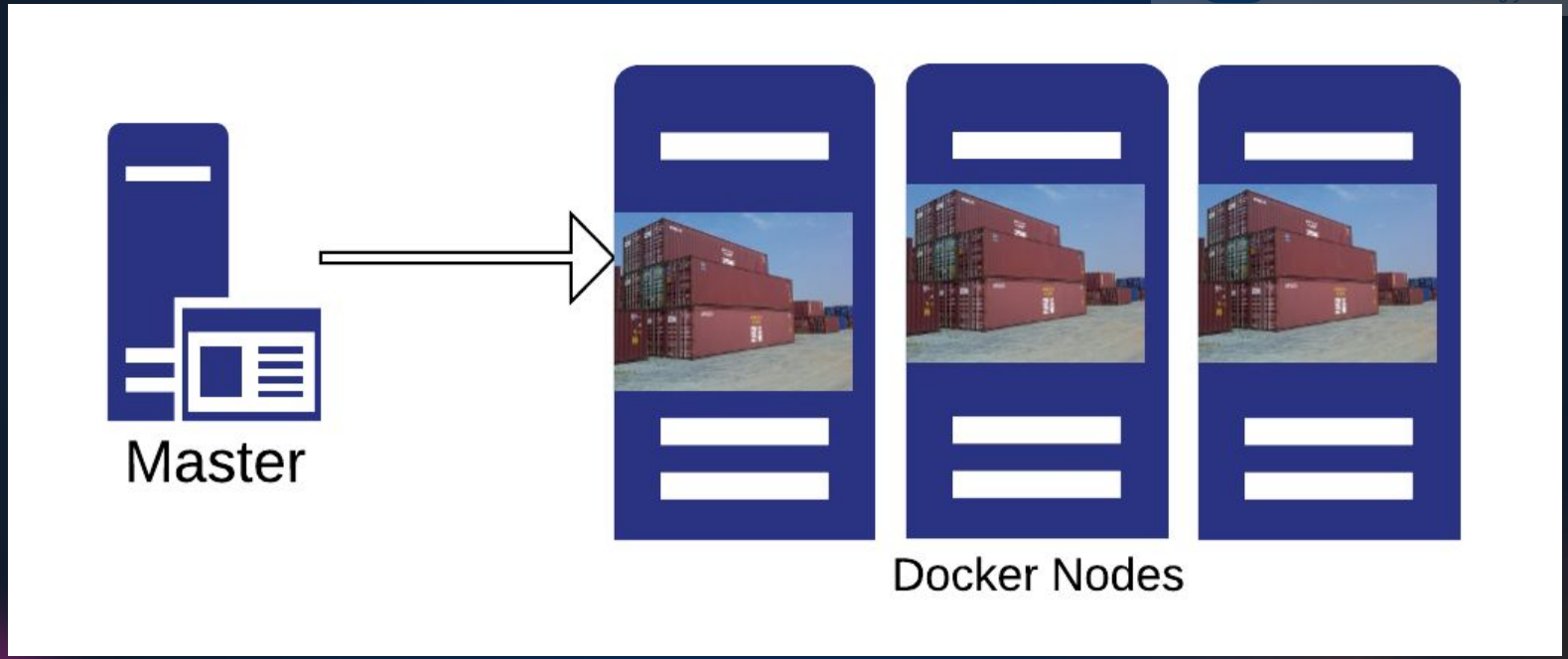
Node running Docker



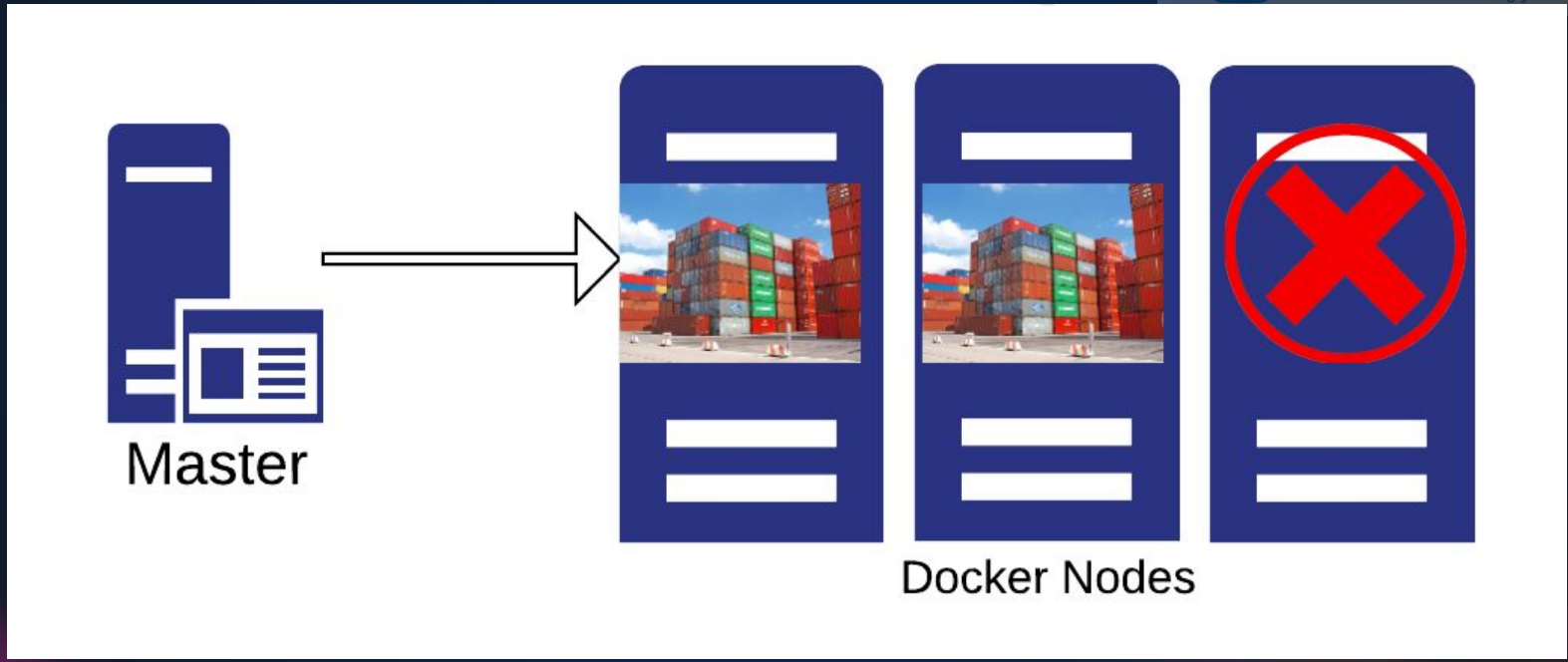
Node running Docker



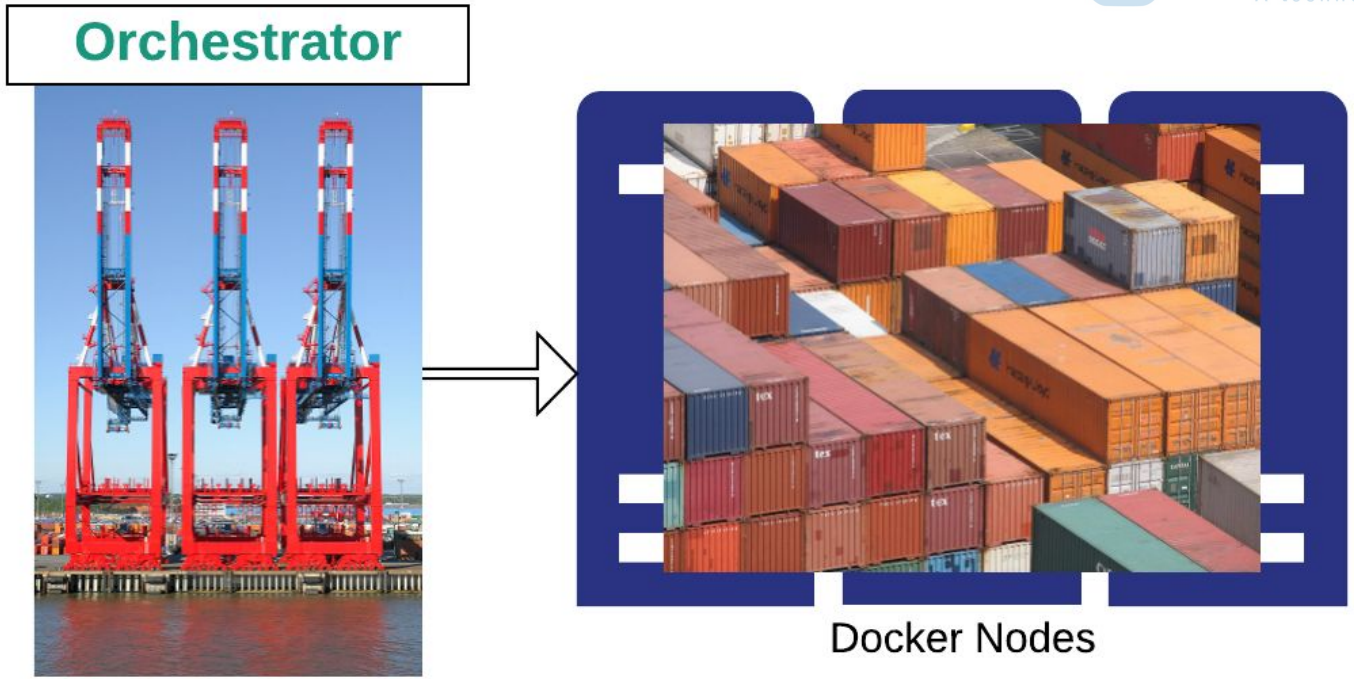
Clustering



Clustering



Container Orchestration



Orchestration Tools

- Docker Swarm
- Kubernetes
- Mesosphere Marathon
- AWS ECS & EKS
- Azure Container Service, AKS
- Google Kubernets Engine
- OpenShift

News from Past

An update on container support on Google Cloud Platform

Tuesday, June 10, 2014

Everything at Google, from Search to Gmail, is packaged and run in a Linux container. Each week we launch more than 2 billion container instances across our global data centers, and the power of containers has enabled both more reliable services and higher, more-efficient scalability. Now we're taking another step toward making those capabilities available to developers everywhere.

Kubernetes History

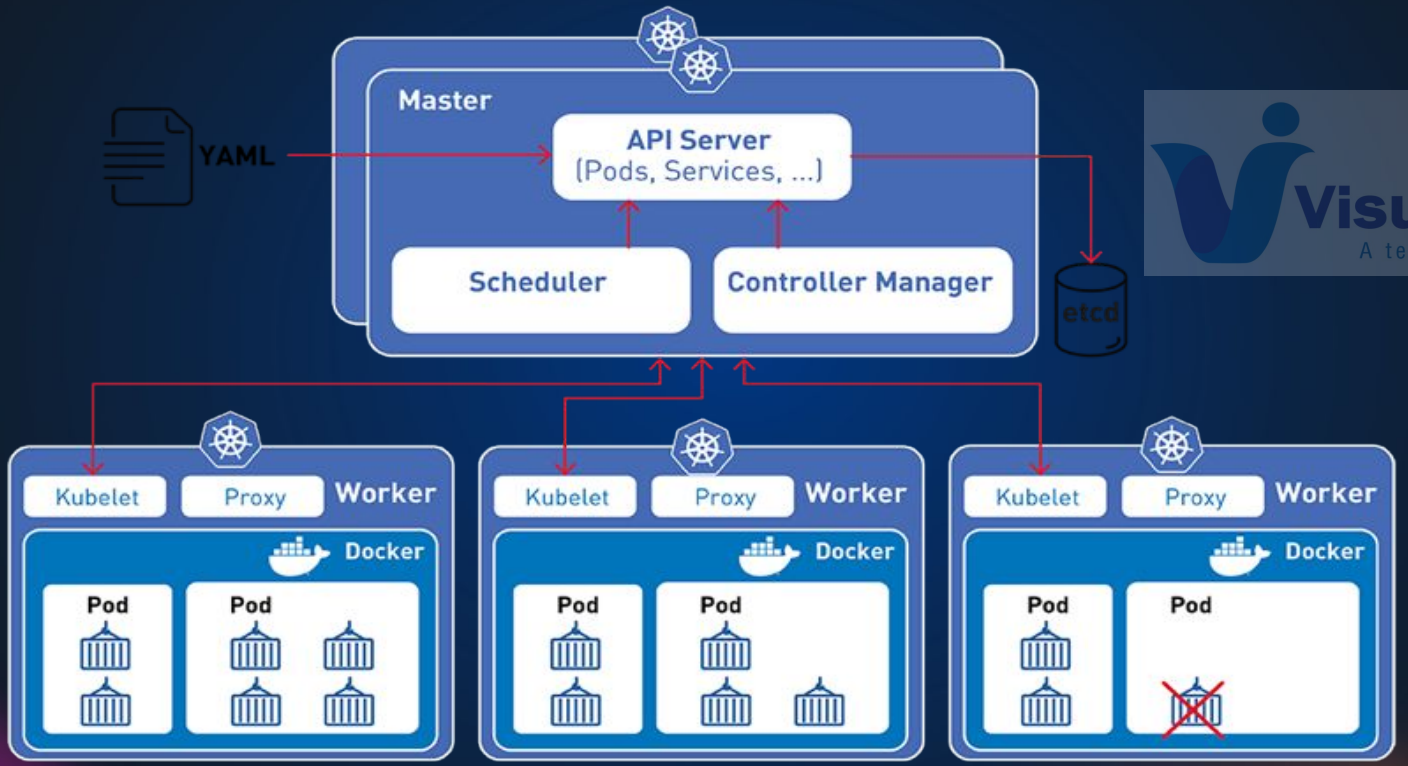
- Created by **Google** to manage their containers AKA Borg
- **Mid-2014:** Google introduced Kubernetes as an open source version of Borg
- **July 21-2015:** Kubernetes v1.0 gets released. Along with the release, Google partnered with the Linux Foundation to form the Cloud Native Computing Foundation (CNCF).
- **2016: Kubernetes Goes Mainstream!**
  - Kops, Minikube, kubeadm etc
  - September 29: Pokemon GO! Kubernetes Case Study Released!
- **2017: Enterprise Adoption**
  - Google and IBM announce Istio
  - Github runs on Kubernetes
  - Oracle joined the Cloud Native Computing Foundation

Kubernetes Provides

- Service discovery and load balancing
- Storage orchestration
- Automated rollouts and rollbacks
- Automatic bin packing
- Self-healing
- Secret and configuration management

https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Kubernetes Architecture



Master: Kube API Server

- Main Hero! Handles all the requests and enables communication across stack services.
- Component on the master that exposes the Kubernetes API.
- It is the front-end for the Kubernetes control plane.
- Admins connects to it using **Kubectl** CLI
- Web Dashboard can be integrated with this API
- and many more integrations....

Master: ETCD Server

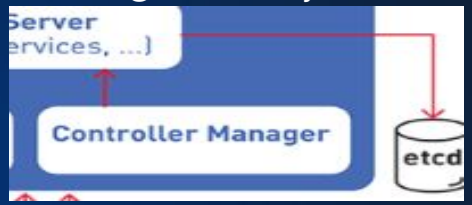
- Stores all the information
- Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.
- Kube API stores retrieves info from it.
- Should be backed up regularly.
- Stores current state of everything in the cluster.

Master: Kube Scheduler

- watches newly created pods that have no node assigned, and selects a node for them to run on
- Factors taken into account for scheduling decisions include
  - individual and collective resource requirements,
  - hardware/software/policy constraints,
  - affinity and anti-affinity specifications,
  - data locality,
  - inter-workload interference and deadlines

Master: Controller Manager

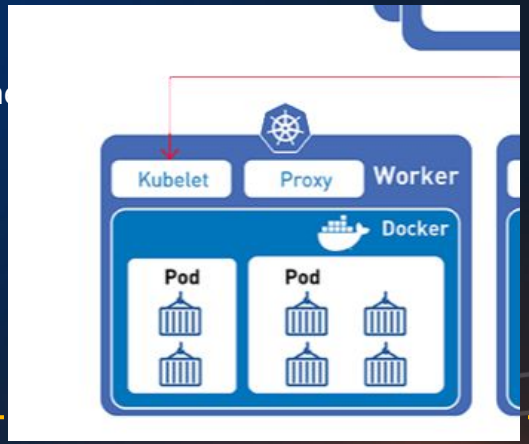
- Logically, each controller is a separate process,
- To reduce complexity, they are all compiled into a single binary and run in a single process.



- These controllers include:
  - **Node Controller:** Responsible for noticing and responding when nodes go down.
  - **Replication Controller**: Responsible for maintaining the correct number of pods for every replication controller object in the system.
  - **Endpoints Controller**: Populates the Endpoints object (that is, joins Services & Pods).
  - **Service Account & Token Controllers:** Create default accounts and API access tokens for new namespace

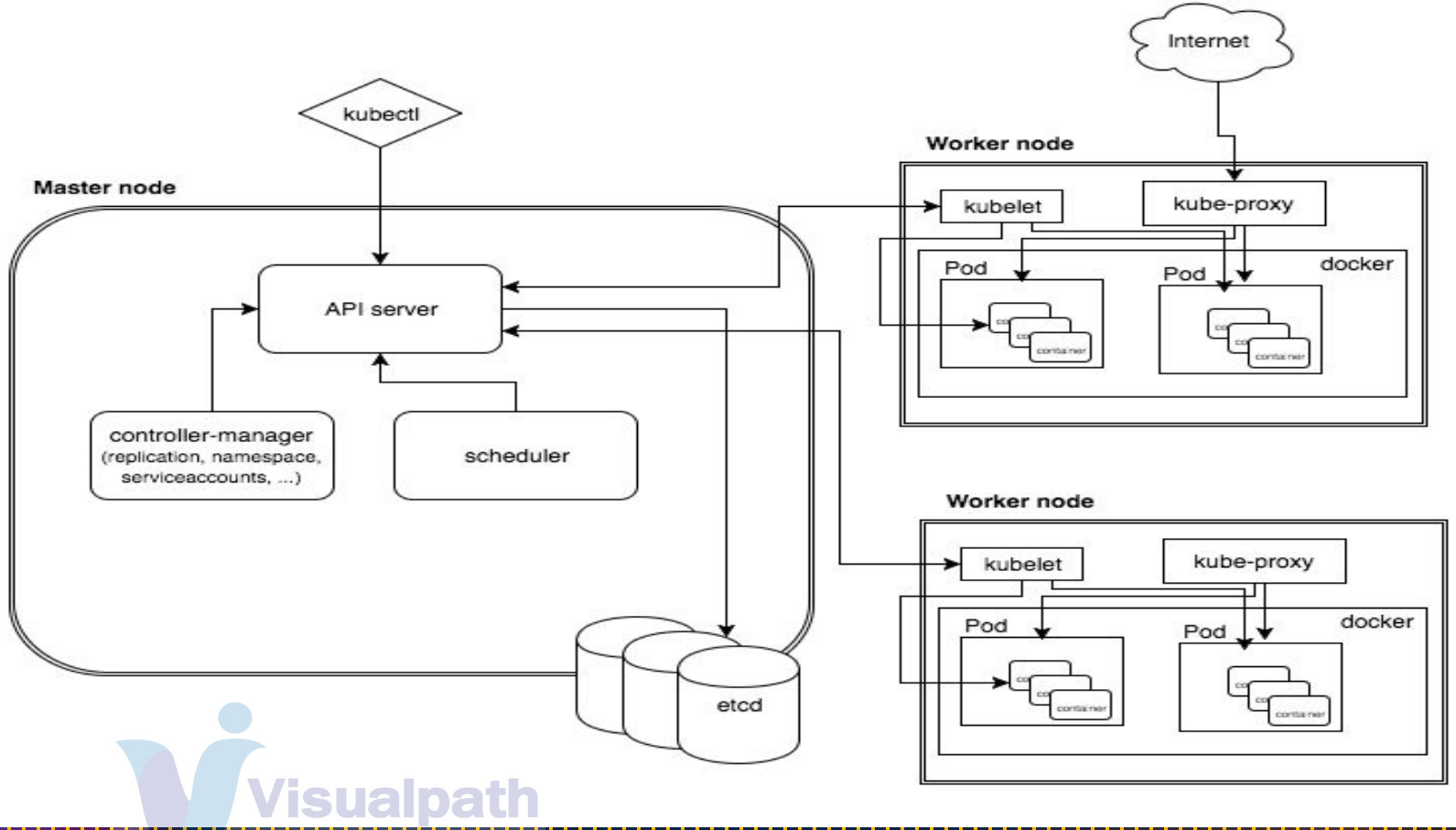
Node Components

- Kubelet
  - An agent that runs on each node in the cluster. It makes sure that containers are running in a pod.
- Kube Proxy
  - network proxy that runs on each node in your cluster
  - Network Rule
    - rules allow network communication to your Pods inside or outside of your cluster
- Container Runtime: Kubernetes supports several container runtim
  - Docker,
  - containerd,
  - cri-o, rktlet
  - Kubernetes CRI (Container Runtime Interface)

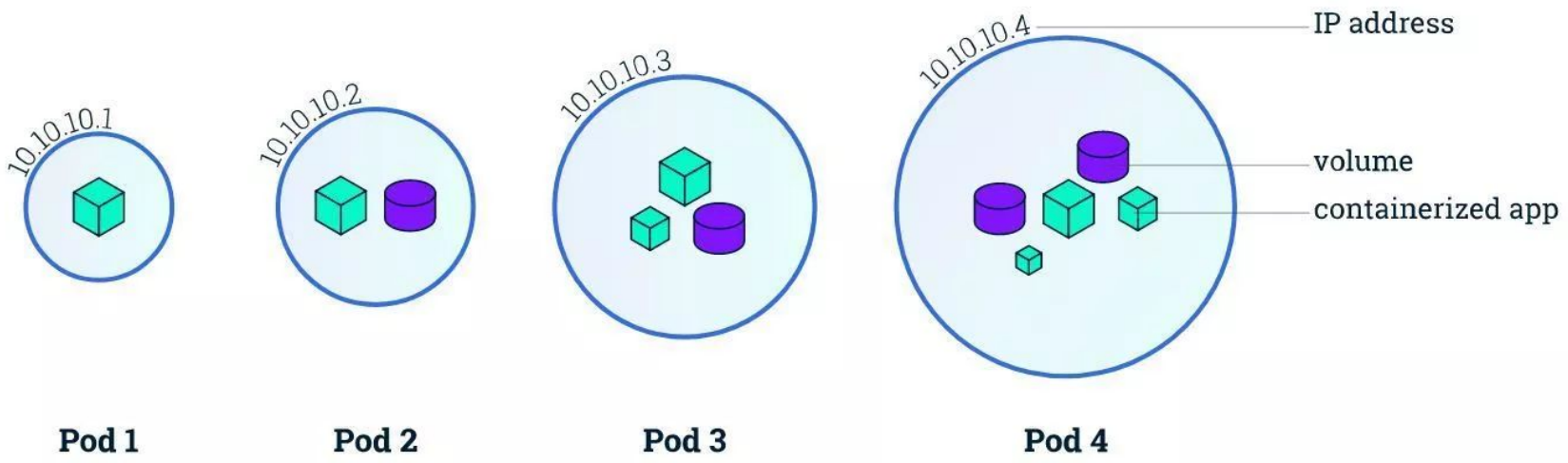


Addons

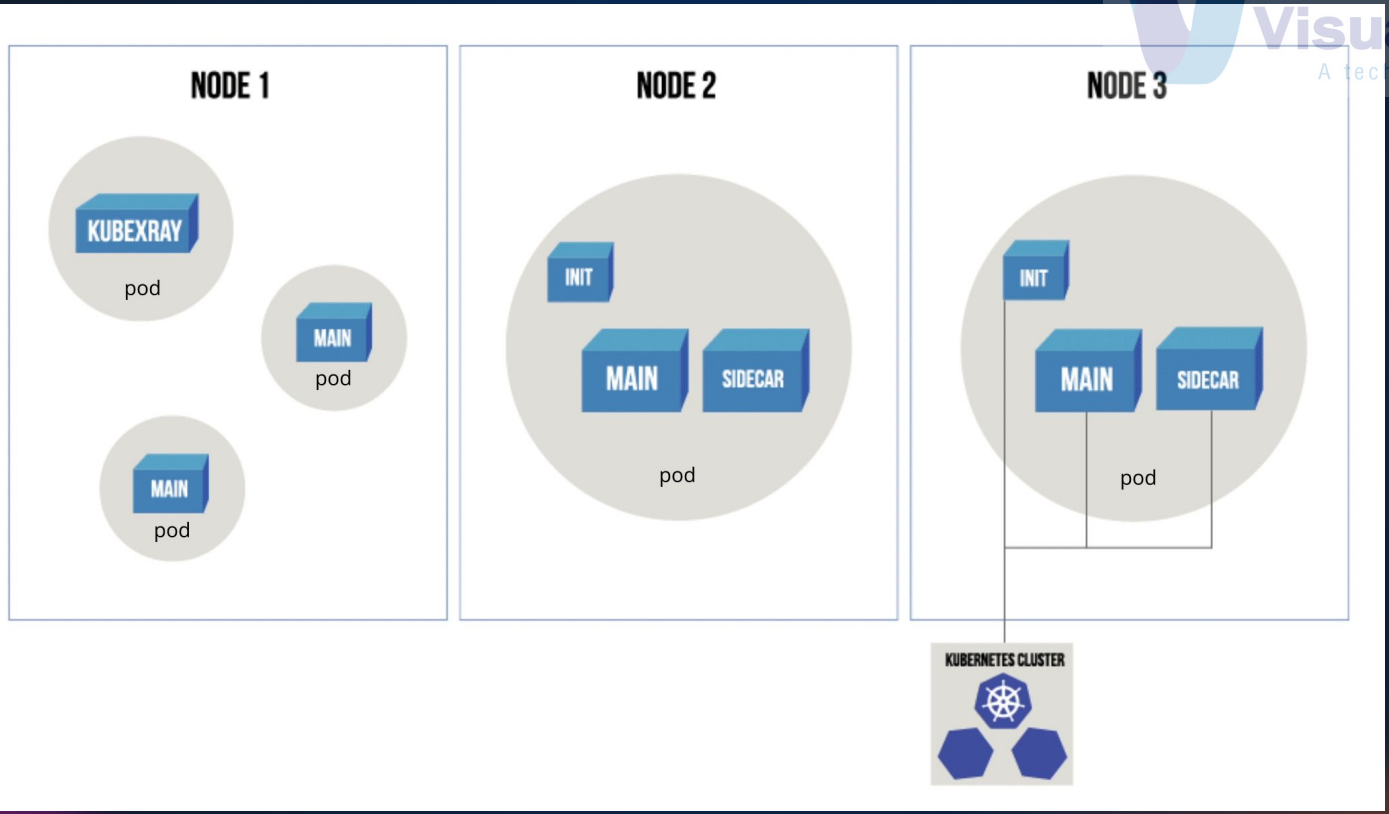
- DNS
- Web UI
- Container Resource Monitoring
- Cluster Level Logging



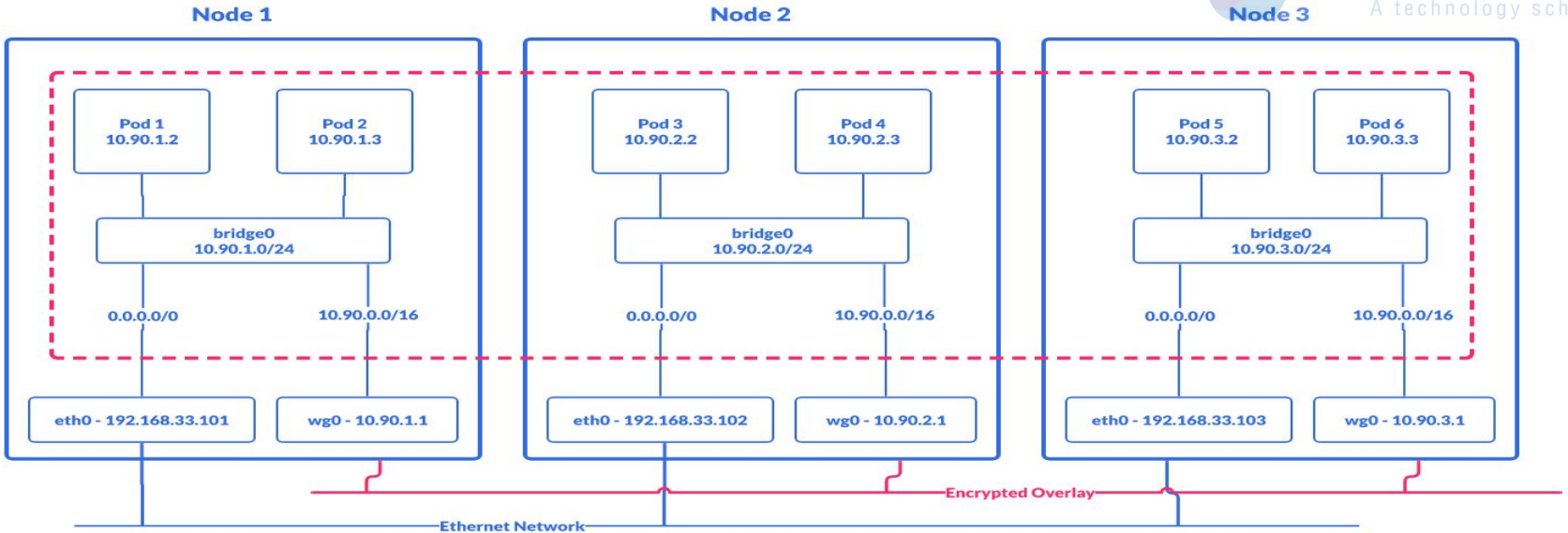
PODS


10.10.10.1
10.10.10.2
10.10.10.3
10.10.10.4
IP address
volume
containerized app
Pod 1
Pod 2
Pod 3
Pod 4

PODS



Overlay Network



Host Network: 192.168.33.0/24

Overlay Network: 10.90.0.0/16

Kubernetes Setup Tools

- Hard Way: Manual Setup
- Minikube:
  - One Node Kubernetes cluster on your computer
- Kubeadm:
  - Multi node Kubernetes Cluster
  - Can be created on any Platforms vm's, ec2, physical machines etc
- Kops:
  - Multi node Kubernetes Cluster on AWS

Setup with Minikube

- Open Powershell as Admin
- Setup Chocolaty
- Install Minikube with Chocolaty

```
choco install minikube kubernetes-cli
```

- Open PowerShell and run

```
minikube start
```

Setup with Kops
(Prerequisites)

- Domain for Kubernetes DNS records
  - e:g groophy.in from **GoDaddy**
- Create a linux VM and setup.
  - kops, kubectl, ssh keys, awscli
- Login to AWS account and setup
  - s3 bucket, IAM User for AWSCli, Route53 Hosted Zone.

Setup with Kops

Login to Domain Registrar(GoDaddy)

Create NS records for subdomain pointing to Routes 53 hosted zone NS servers

NS	kubernetes	ns-1480.awsdns-57.org	1 Hour	
NS	kubernetes	ns-1592.awsdns-07.co.uk	1 Hour	
NS	kubernetes	ns-497.awsdns-62.com	1 Hour	
NS	kubernetes	ns-678.awsdns-20.net	1 Hour	

1

PODS

Run your apps Isolated

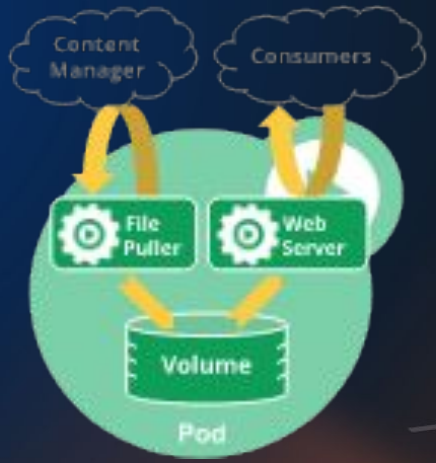
PODS

A *Pod* is the basic execution unit of a Kubernetes application–the smallest and simplest unit in the Kubernetes object model that you create or deploy. A Pod represents processes running on your Cluster.

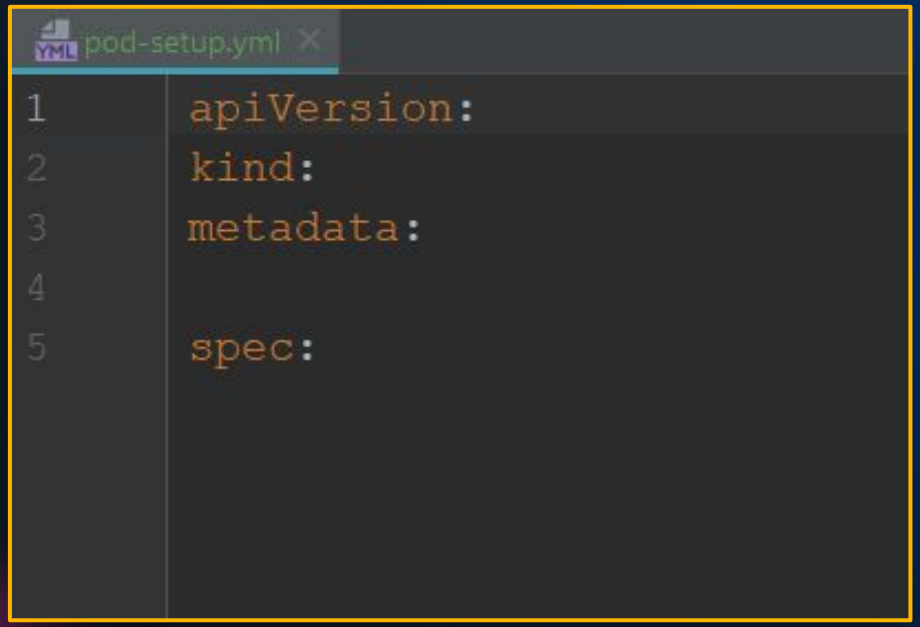
- **Pods that run a single container**.
  - The “one-container-per-Pod” model is the most common Kubernetes use case.
  - Pod as a wrapper around a single container,
  - Kubernetes manages the Pods rather than the containers directly.

- **Multi Container POD**
  - Tightly coupled and need to share resources
  - One Main container and other as a sidecar or init container
  - Each Pod is meant to run a single instance of a given application
  - Should use multiple Pods to scale horizantally.

.https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/



Definitions file in YAML



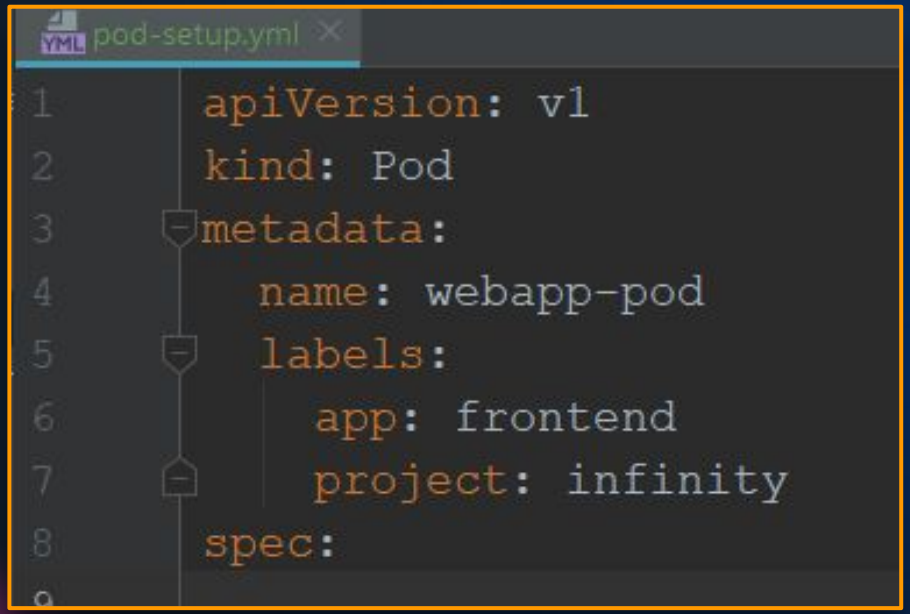
Definitions file in YAML

pod-setup.yml

Kind	Version
POD	v1
Service	v1
Deployment	apps/v1
Ingress	networking.../v1beta1

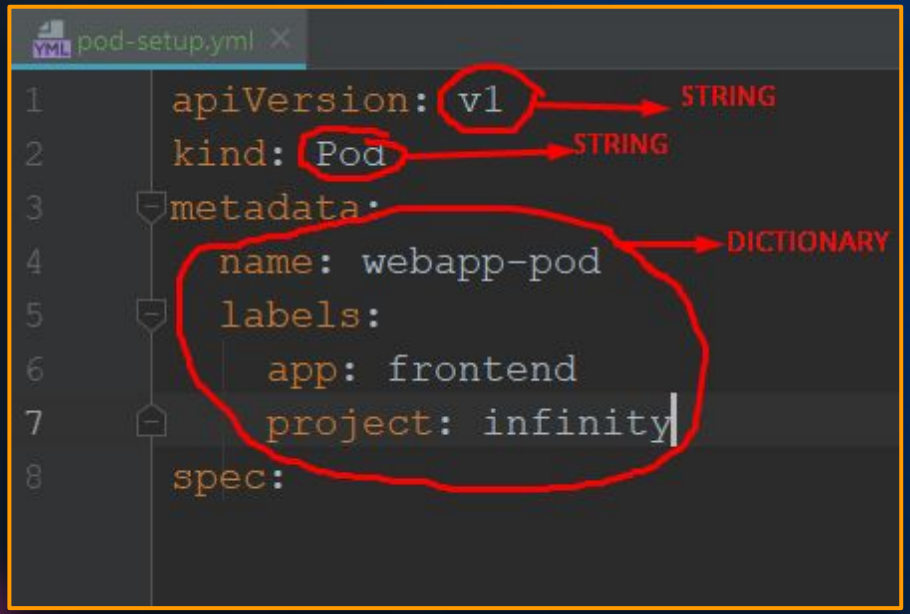
Definitions file in YAML

pod-setup.yml



Definitions file in YAML

pod-setup.yml



pod-setup.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: webapp-pod
  labels:
    app: frontend
    project: infinity
spec:
  containers:
    - name: httpd-container
      image: httpd
```

pod-setup.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: webapp-pod
  labels:
    app: frontend
    project: infinity
spec:
  containers:
    - name: httpd-container
      image: httpd
      ports:
        - name: http-port
          containerPort: 80
```

Visualpath
A technology school

Create and get POD Info

```
$ kubectl create -f pod-setup.yml
pod/webapp-pod created
```

```
$ kubectl get pod
NAME          READY   STATUS              RESTARTS   AGE
webapp-pod    0/1     ContainerCreating   0          51s
```

```
$ kubectl get pod
NAME          READY   STATUS    RESTARTS   AGE
webapp-pod    1/1     Running   0          9m30s
```

Deatailed POD Info

```
$ kubectl describe pod webapp-pod
Name:               webapp-pod
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               minikube/10.0.2.15
Start Time:         Wed, 28 Aug 2019 15:11:27 +0530
Labels:             app=frontend
                    project=infinity
Annotations:        <none>
Status:             Running
IP:                 172.17.0.4
Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  17m    default-scheduler  Successfully assigned default/webapp-pod to minikube
  Normal  Pulling    17m    kubelet, minikube  Pulling image "httpd"
  Normal  Pulled     9m37s  kubelet, minikube  Successfully pulled image "httpd"
  Normal  Created    9m37s  kubelet, minikube  Created container httpd-container
  Normal  Started    9m36s  kubelet, minikube  Started container httpd-container
```

Get & EDIT POD

```
$ kubectl get pod webapp-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2019-08-28T09:41:27Z"
  labels:
    app: frontend
    project: infinity
  name: webapp-pod
```

```
$ kubectl get pod webapp-pod -o yaml > webpod-definition.yml
```

```
$ kubectl edit pod webapp-pod
pod/webapp-pod edited
```

Service

Connect with or To your POD

SERVICE

Way to expose an application running on a set of Pods as a network service.

Similar to Load Balancers

SERVICE

Motivation

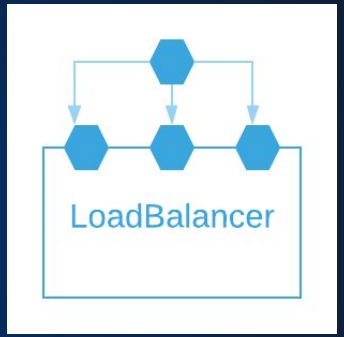
Kubernetes Pods are mortal. They are born and when they die, they are not resurrected. If you use a Deployment to run your app, it can create and destroy Pods dynamically.

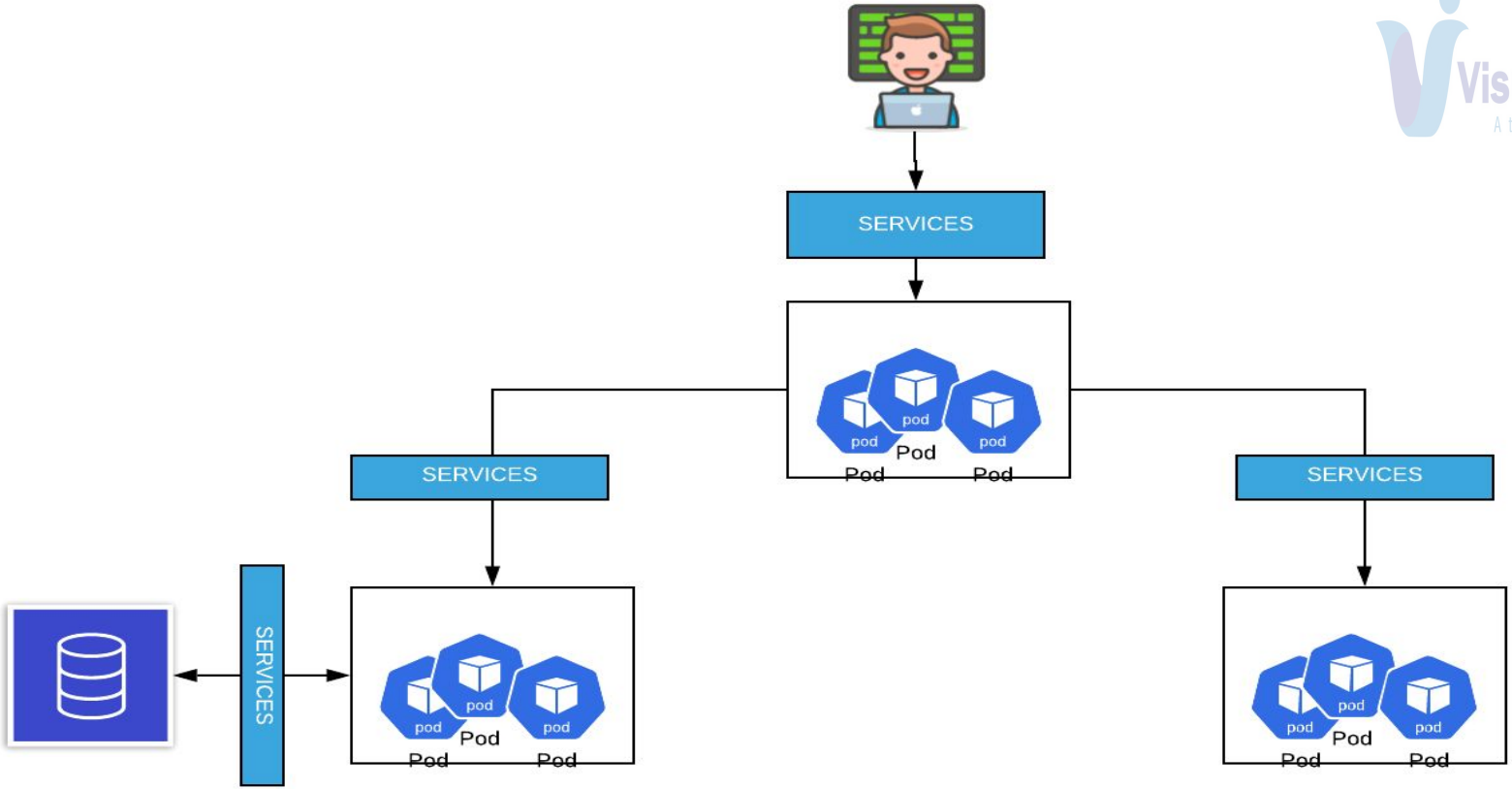
Each Pod gets its own IP address, however in a Deployment, the set of Pods running in one moment in time could be different from the set of Pods running that application a moment later.

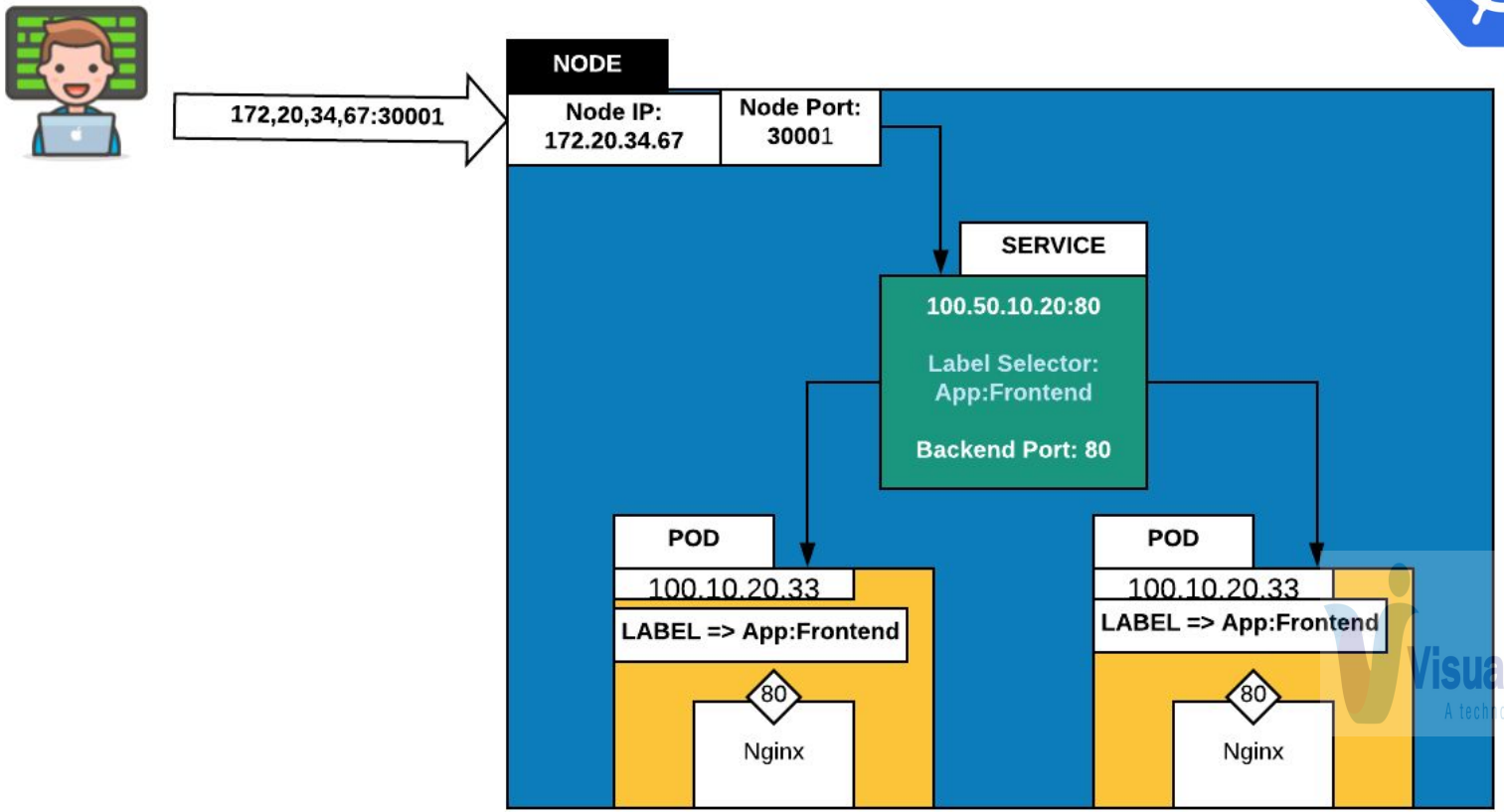
This leads to a problem: if some set of Pods (call them "backends") provides functionality to other Pods (call them "frontends") inside your cluster, how do the frontends find out and keep track of which IP address to connect to, so that the frontend can use the backend part of the workload?

Enter *Services*.

SERVICE



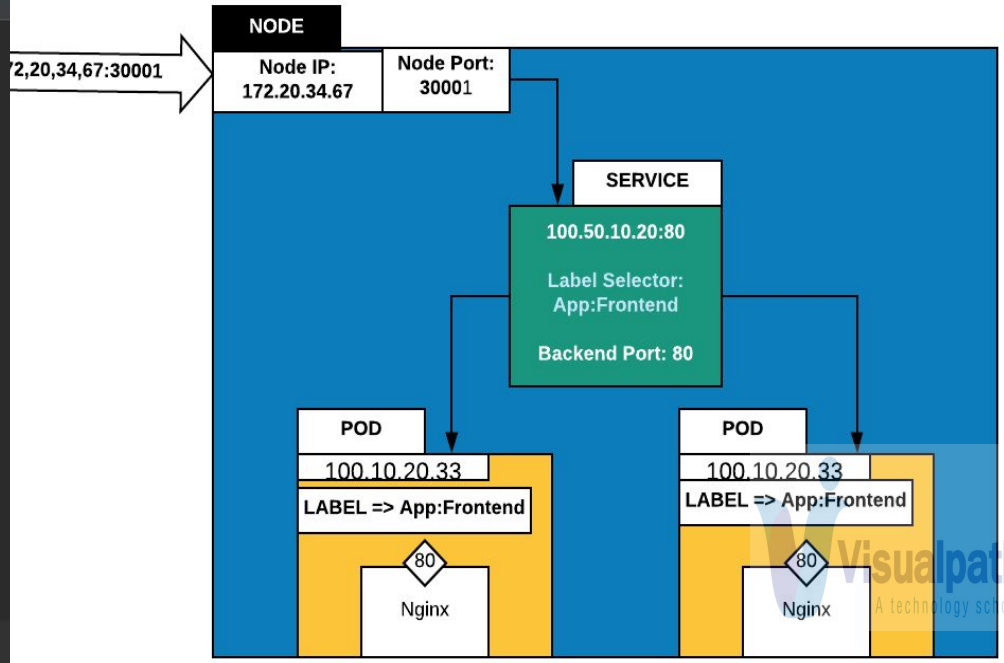




Service | NodePort

service-defs.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: webapp-service
spec:
  type: NodePort
  ports:
  - targetPort: 80
    port: 80
    nodePort: 30005
    protocol: TCP
  selector:
    app: frontend
```



Service | NodePort

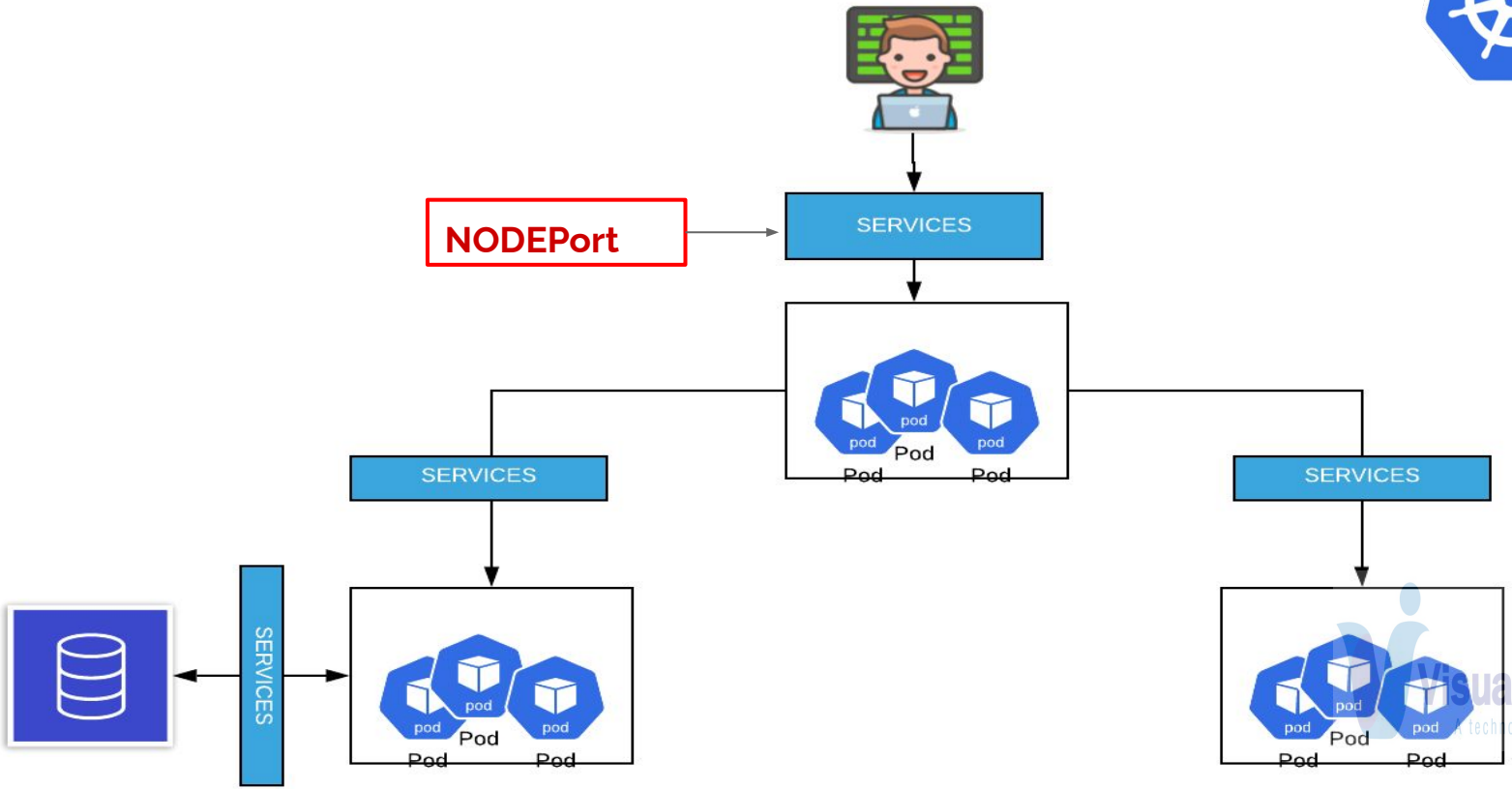
service-defs.yml

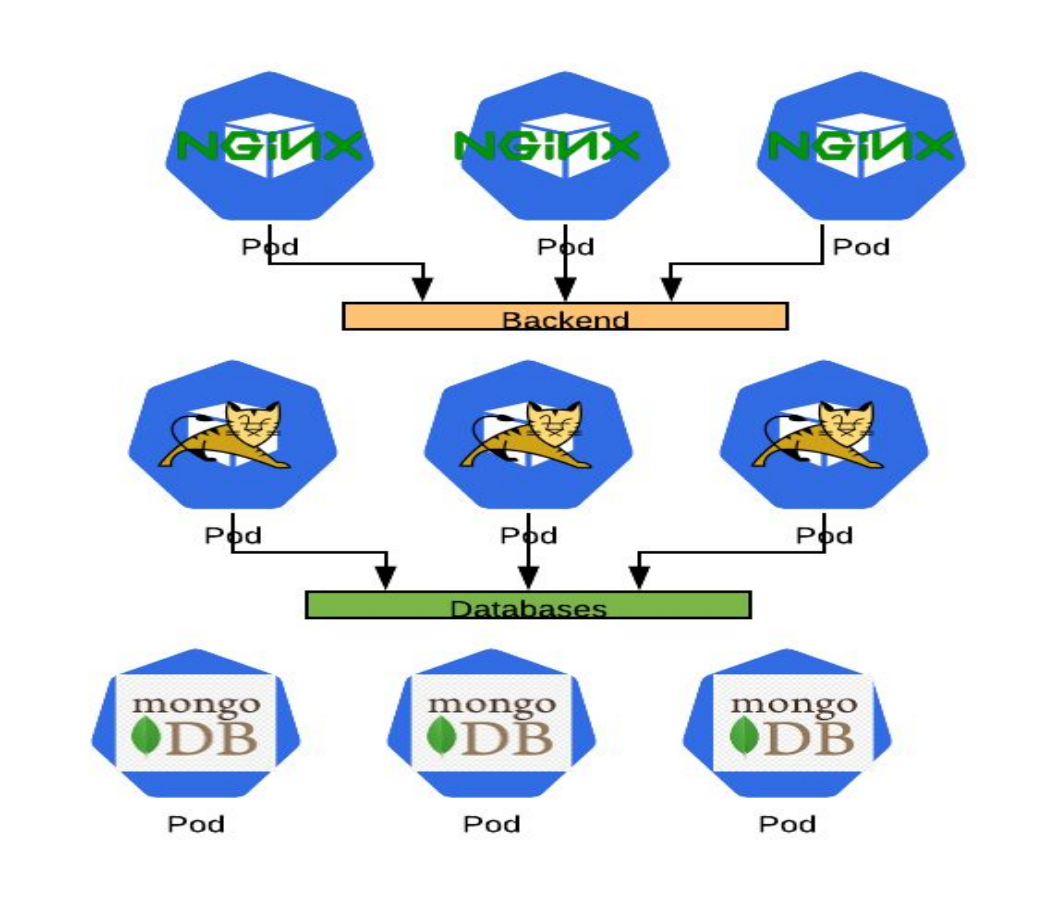
```yaml
apiVersion: v1
kind: Service
metadata:
  name: webapp-service
spec:
  type: NodePort
  ports:
  - targetPort: 80
    port: 80
    nodePort: 30005
    protocol: TCP
  selector:
    app: frontend
```

```
$ kubectl create -f service-defs.yml
service/webapp-service created
```

```
$ kubectl.exe get svc
NAME             TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)
kubernetes       ClusterIP   10.96.0.1      <none>        443/TCP
webapp-service   NodePort    10.110.3.28    <none>        80:30005/TCP
```

```
$ kubectl.exe describe svc webapp-service
Name:                     webapp-service
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=frontend
Type:                     NodePort
IP:                       10.110.3.28
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30005/TCP
Endpoints:                172.17.0.4:80
```





Service | ClusterIP

tom-svc-clusterip.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: app-service
spec:
  type: ClusterIP
  ports:
  - targetPort: 8080
    port: 8080
    protocol: TCP
  selector:
    app: backend
```

tom-app.yml | tom-svc-clusterip.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-pod
  labels:
    app: backend
    project: infinity
spec:
  containers:
    - name: tomcat-container
      image: tomcat
      ports:
        - name: app-port
          containerPort: 8080
```

Service | ClusterIP

tom-svc-clusterip.yml

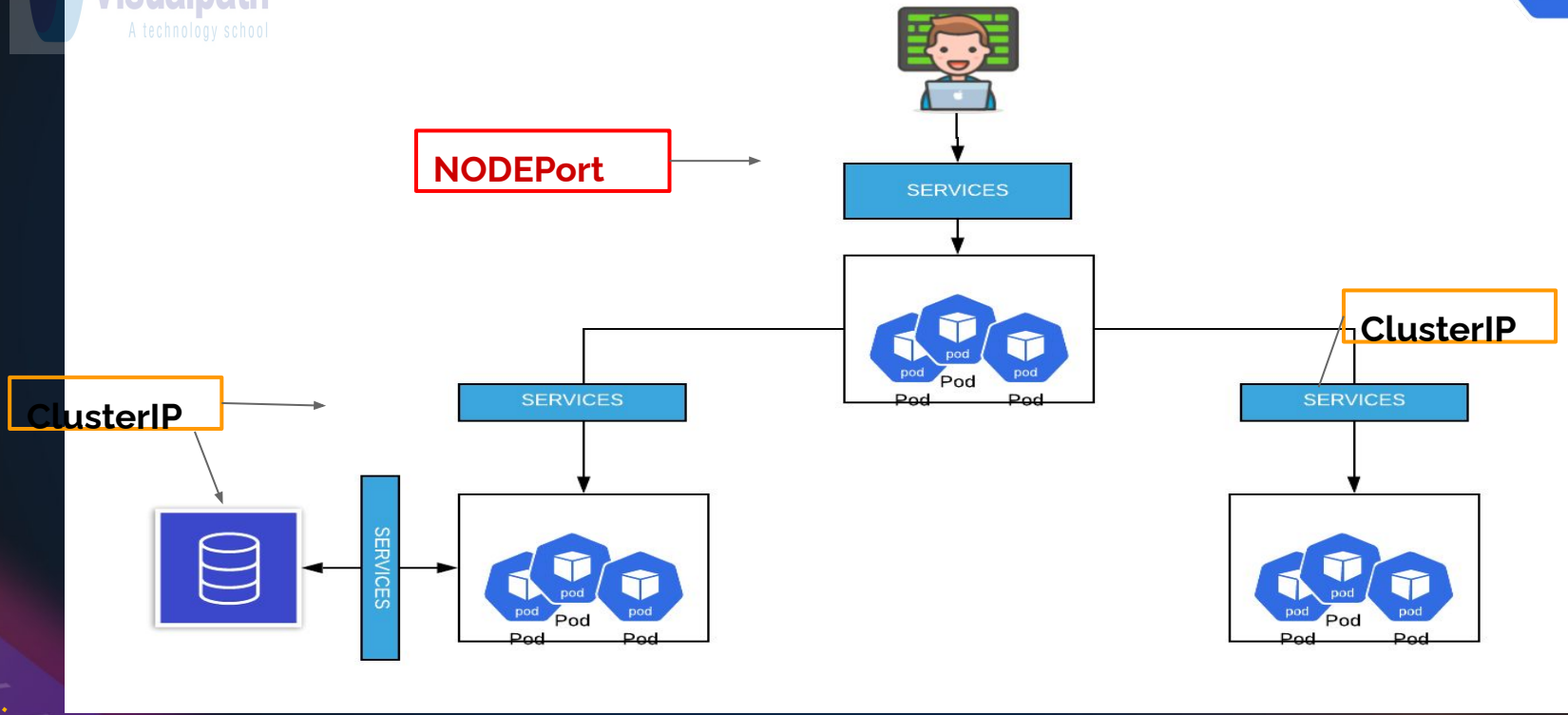
```yaml
apiVersion: v1
kind: Service
metadata:
  name: app-service
spec:
  type: ClusterIP
  ports:
  - targetPort: 8080
    port: 8080
    protocol: TCP
  selector:
    app: backend
```

```
$ kubectl.exe create -f tom-svc-cIP.yml
service/app-service created
```

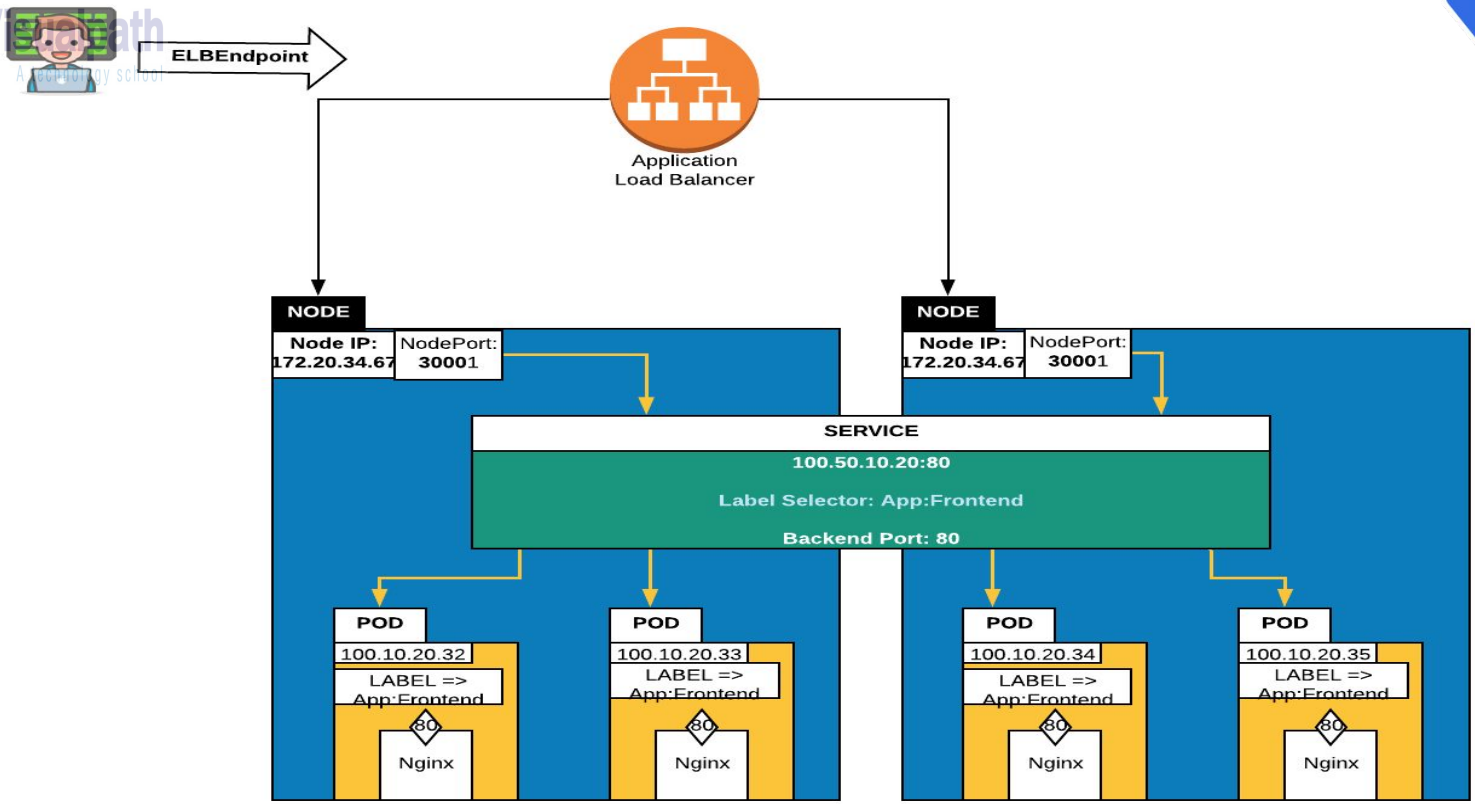
```
$ kubectl.exe get svc
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
app-service   ClusterIP   10.109.177.150  <none>        8080/TCP
```

```
$ kubectl.exe describe svc app-service
Name:              app-service
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          app=backend
Type:              ClusterIP
IP:                10.109.177.150
Port:              <unset>  8080/TCP
TargetPort:        8080/TCP
Endpoints:         172.17.0.5:8080
```

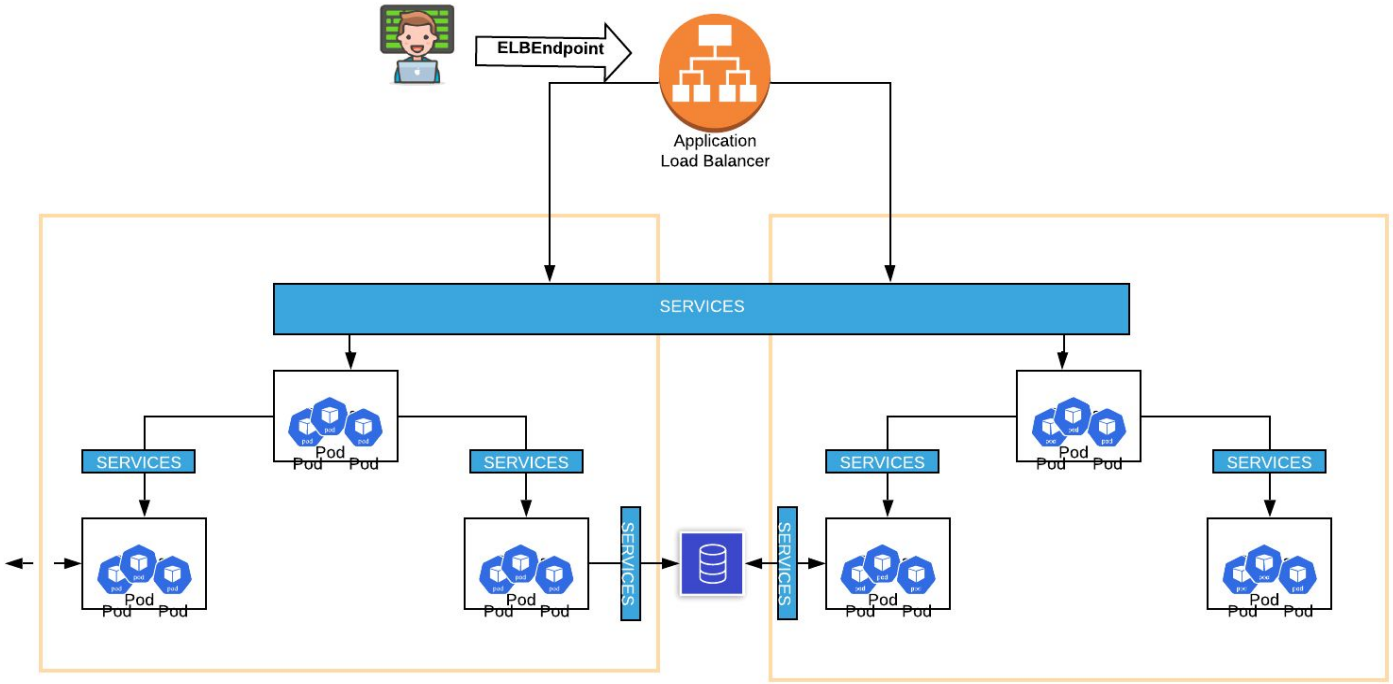
NodePort & ClusterIP



Service | LoadBalancer



LoadBalancer & ClusterIP



Visualpath
A technology school

3

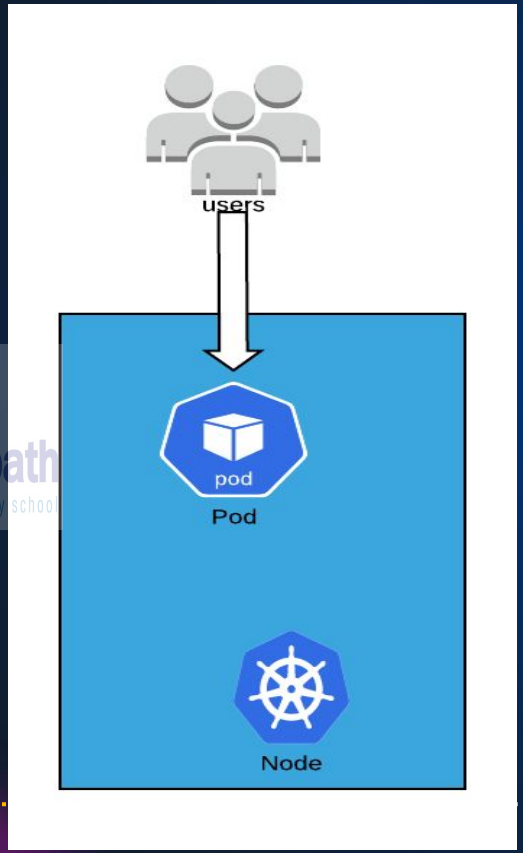
Replication Controller

Keep your running all the TIME

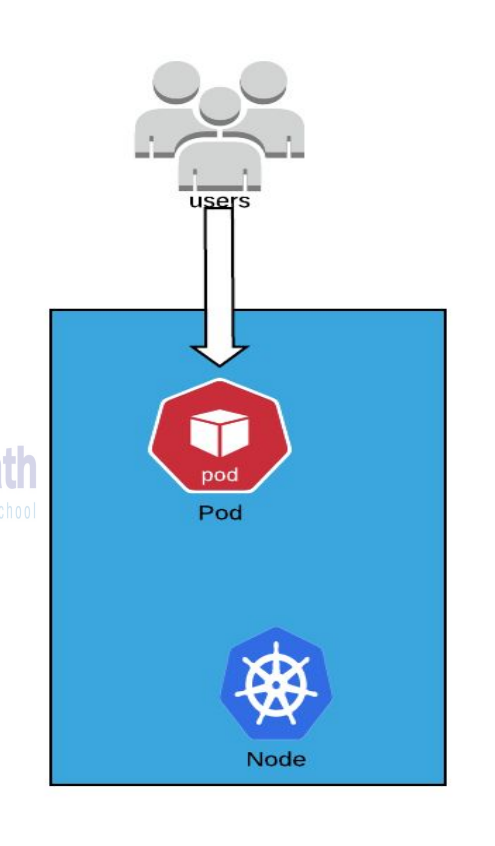
Replication Controller

- Pods maintained by a ReplicationController are automatically replaced if they fail, are deleted, or are terminated
- If there are too many pods, the ReplicationController terminates the extra pods.
- If there are too few, the ReplicationController starts more pods.

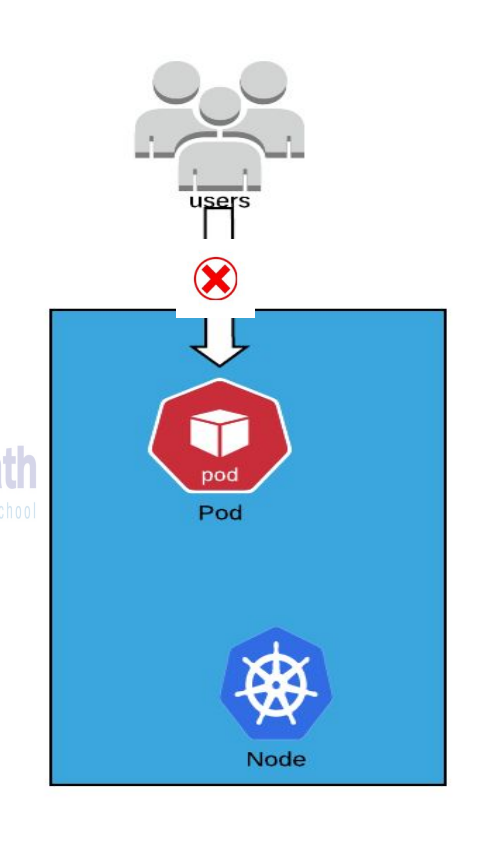
POD without Replication Controller



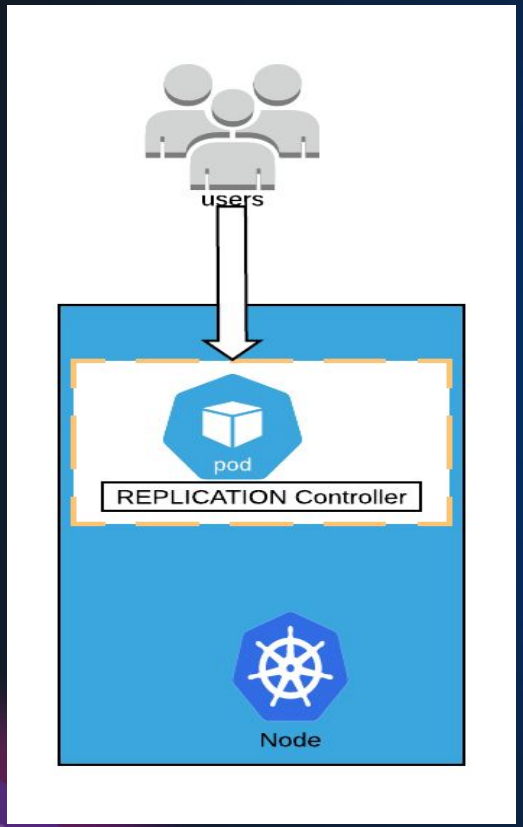
POD without Replication Controller

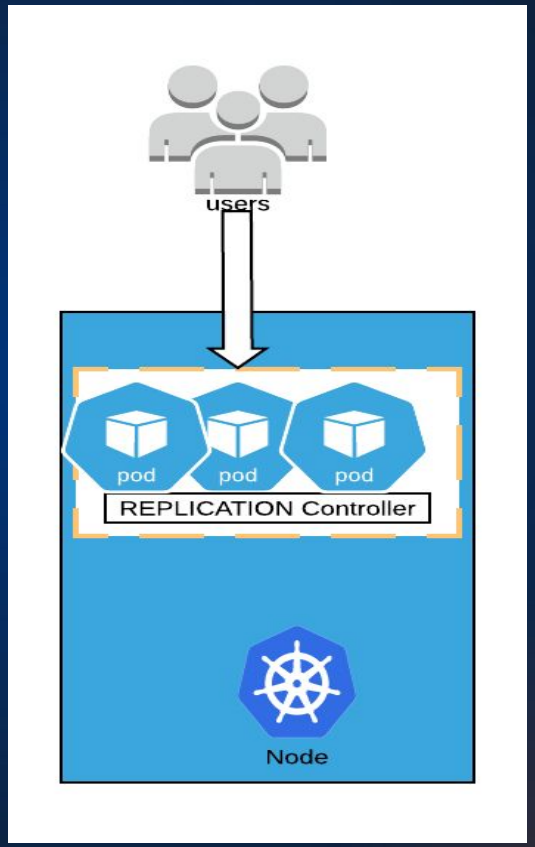


POD without Replication Controller

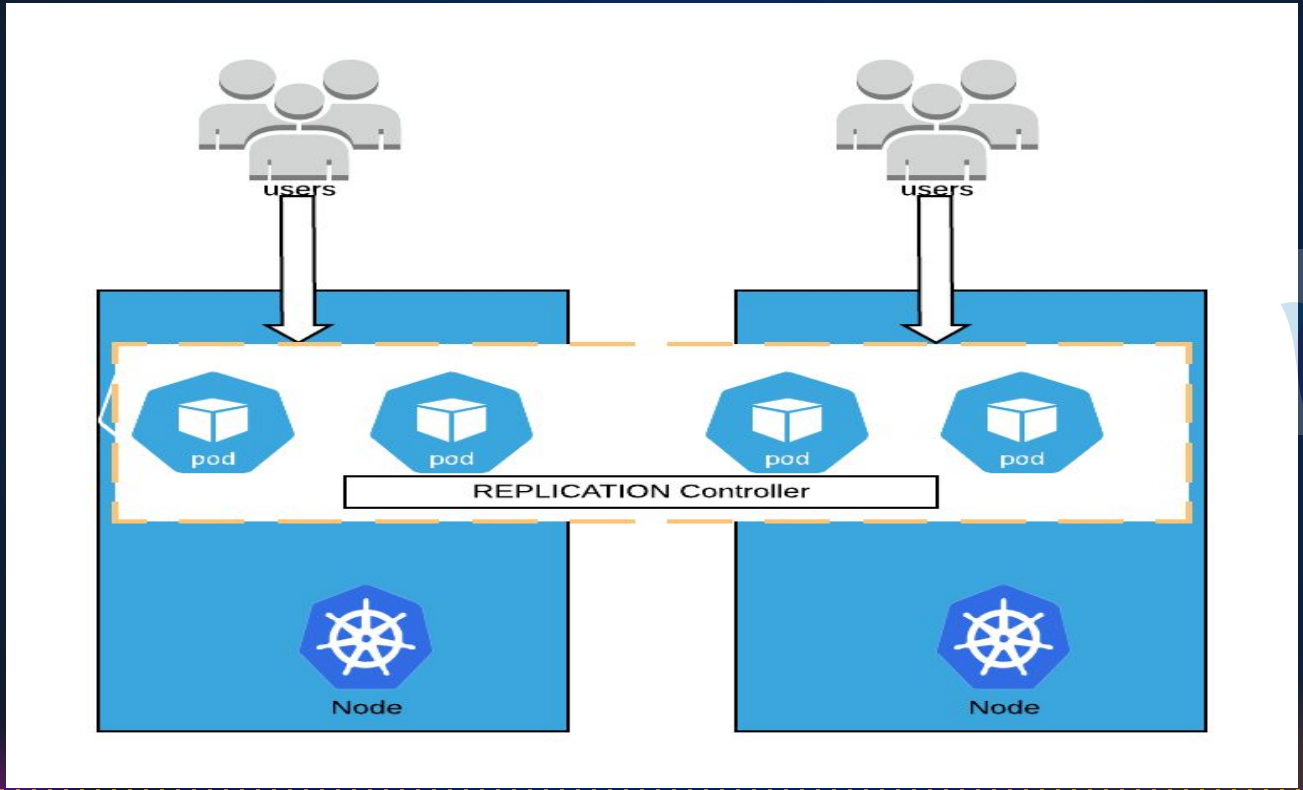


POD with Replication Controller





Scaling with Replication Controller



Replication Controller Definition

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: app-controller
spec:
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: tomcat-container
        image: tomcat
        ports:
        - name: app-port
          containerPort: 8080
  replicas: 2
  selector:
    app: backend
~
```

Replication Controller Definition


```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: app-controller
spec:
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: tomcat-container
        image: tomcat
        ports:
        - name: app-port
          containerPort: 8080
  replicas: 2
  selector:
    app: backend
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-pod
  labels:
    app: backend
spec:
  containers:
    - name: tomcat-container
      image: tomcat
      ports:
        - name: app-port
          containerPort: 8080
```

Create & View RC


```
$ kubectl create -f tom-app-rc.yml
replicationcontroller/app-controller created
```

```
$ kubectl get rc
NAME             DESIRED   CURRENT   READY   AGE
app-controller   2         2         2       3m28s
```

```
$ kubectl get pod
NAME                   READY   STATUS    RESTARTS   AGE
app-controller-62zz4   1/1     Running   0          3m46s
app-controller-dmc7j   1/1     Running   0          3m46s
```

Edit & Scale RC


```
$ kubectl edit rc app-controller
```

```
spec:
  replicas: 2
  selector:
    app: backend
  template:
```

```
$ kubectl scale rc app-controller --replicas=4
replicationcontroller/app-controller scaled
```

```
$ kubectl get pod
NAME                   READY   STATUS    RESTARTS   AGE
app-controller-62zz4   1/1     Running   0          7m30s
app-controller-dmc7j   1/1     Running   0          7m30s
app-controller-t8c74   1/1     Running   0          31s
app-controller-tdqwb   1/1     Running   0          31s
```

ReplicaSet

Maintain Pod Replica

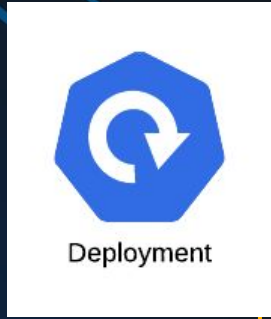
Replicaset

- A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time

Deployment

Upgrade, RollBack, Changes Gracefully

Deployment

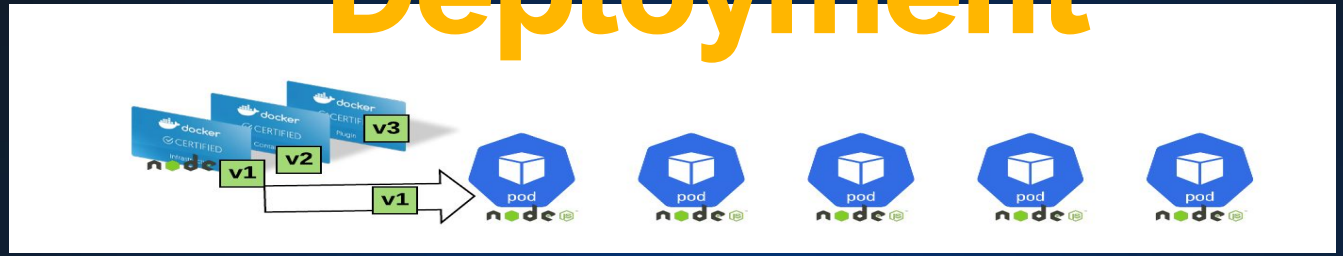


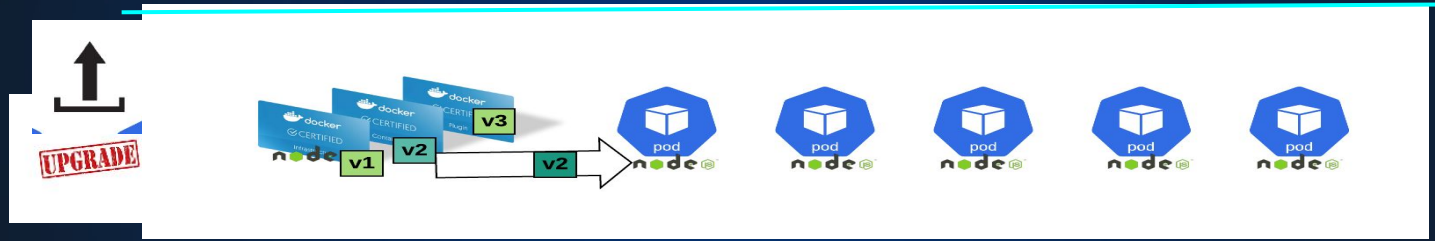
- A *Deployment* controller provides declarative updates for Pods and ReplicaSets.

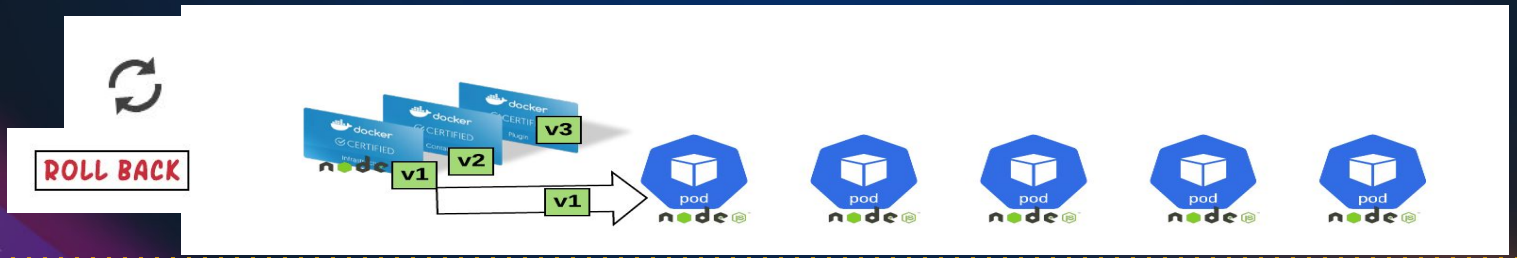
- Define *desired state* in a Deployment, and the Deployment controller changes the actual state to the desired state at a controlled rate.

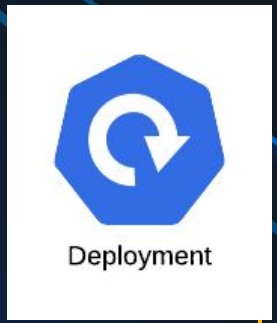
- Deployment creates **ReplicaSet** to manage number of PODS

Deployment

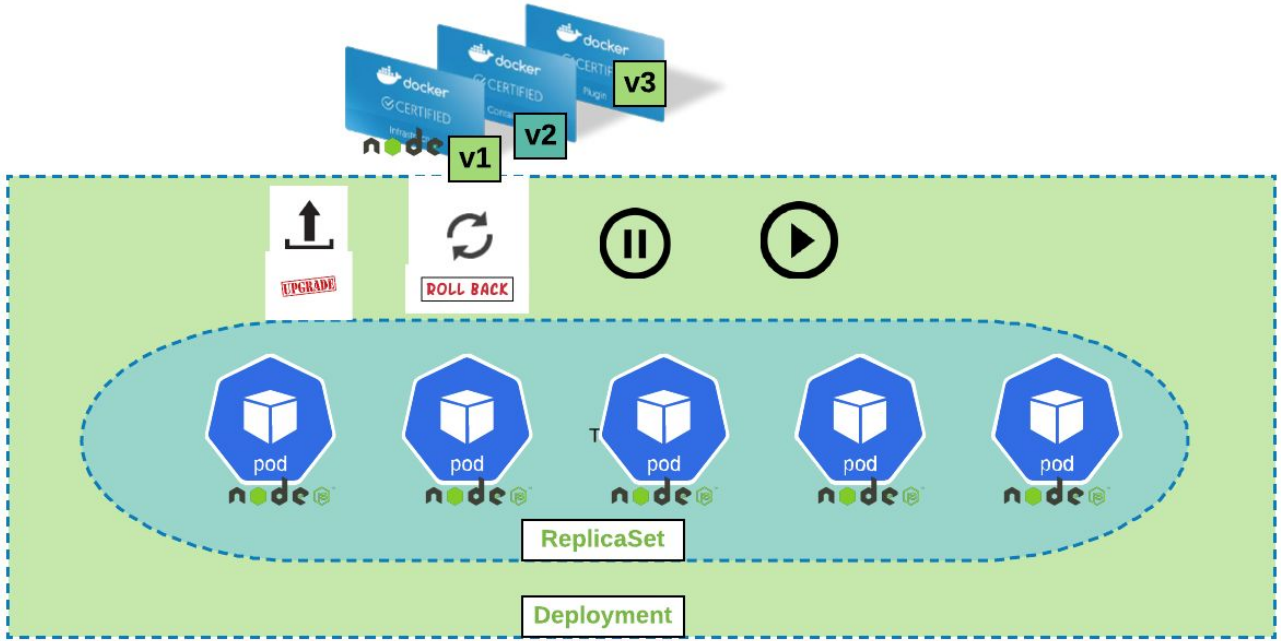

v3
v2
v1
v1
pod
node


UPGRADE
v3
v2
v1
v2
pod
node


ROLL BACK
v3
v2
v1
v1
pod
node


Deployment

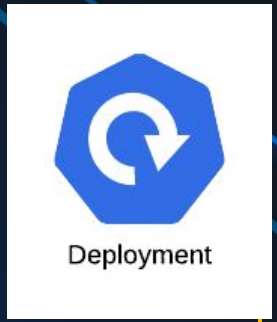
Deployment


v3
v2
v1
UPGRADE
ROLL BACK
pod
pod
pod
pod
pod
ReplicaSet
Deployment


Deployment

Deployment

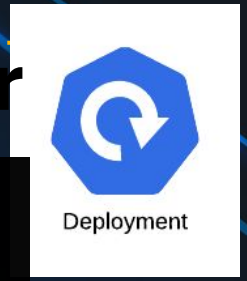
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-controller
spec:
  template:
    metadata:
      labels:
          app: backend
    spec:
      containers:
      - name: tomcat-container
        image: tomcat
        ports:
        - name: app-port
          containerPort: 8080
  replicas: 3
  selector:
    matchLabels:
      app: backend
```



Deployment | Replication Controller

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-controller
spec:
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: tomcat-container
        image: tomcat
        ports:
        - name: app-port
          containerPort: 8080
  replicas: 3
  selector:
    matchLabels:
      app: backend
```

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: app-controller
spec:
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: tomcat-container
        image: tomcat
        ports:
        - name: app-port
          containerPort: 8080
  replicas: 2
  selector:
    app: backend
```



Create & View Deployment

```
$ kubectl.exe create -f tom-app-deploy.yml
deployment.apps/app-controller created
```

```
$ kubectl get deploy
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
app-controller   3/3     3            3           5m23s
```

```
$ kubectl get rs
NAME                       DESIRED   CURRENT   READY   AGE
app-controller-bfcd7964    3         3         3       5m27s
```

```
$ kubectl get pod
NAME                             READY   STATUS    RESTARTS   AGE
app-controller-bfcd7964-fxlq5    1/1     Running   0          5m31s
app-controller-bfcd7964-p7zb5    1/1     Running   0          5m31s
app-controller-bfcd7964-x5wbx    1/1     Running   0          5m31s
```

View & Edit Deployment

```
$ kubectl describe deploy app-controller
Name:                   app-controller
Namespace:              default
CreationTimestamp:      Thu, 29 Aug 2019 14:17:30 +0530
```

```
$ kubectl get deploy app-controller -o yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: "2019-08-29T08:47:30Z"
  generation: 1
  name: app-controller
```

```
$ kubectl edit deploy app-controller
```

```
  containers:
  - image: tomcat
    imagePullPolicy: Always
```

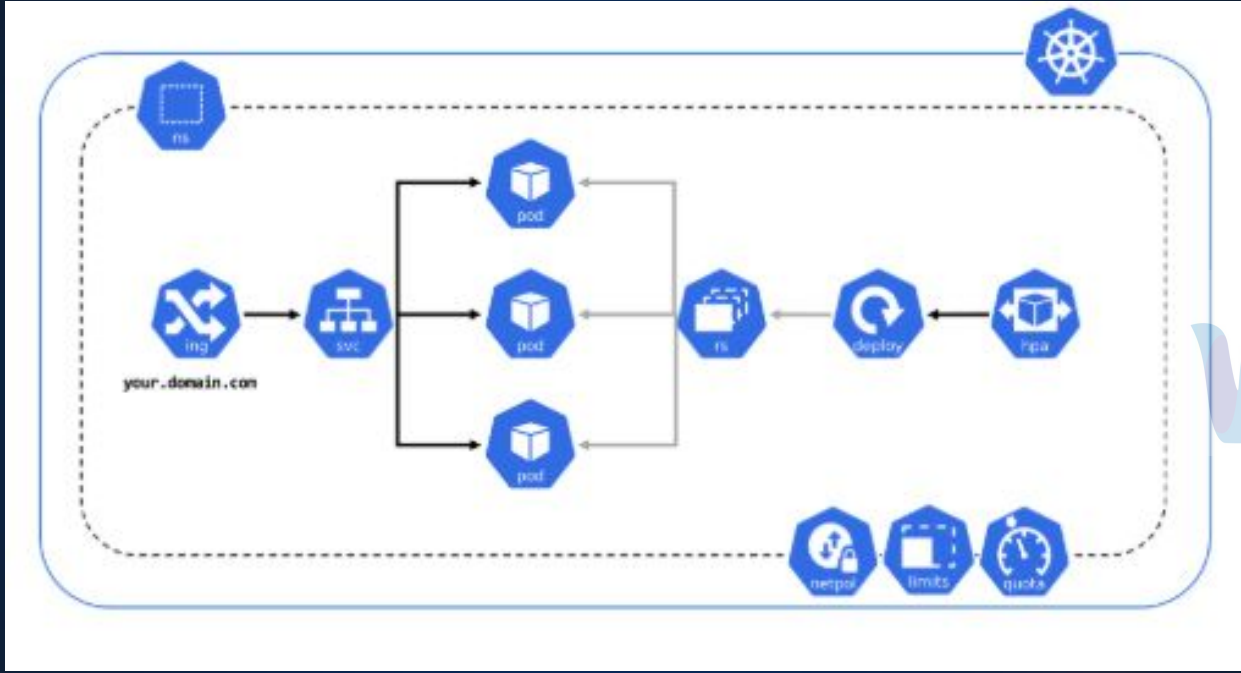
Edit Deployment

```
$ kubectl set image deployment/app-controller tomcat-container=tomcat:8.5-jdk11-
adoptopenjdk-openj9
deployment.extensions/app-controller image updated
```

```
$ kubectl get pod
NAME                              READY   STATUS              RESTARTS   AGE
app-controller-868d9cb55c-4s2g8   0/1     ContainerCreating   0          29s
app-controller-bfcd7964-fxlq5     1/1     Running             0          16m
app-controller-bfcd7964-p7zb5     1/1     Running             0          16m
app-controller-bfcd7964-x5wbx     1/1     Running             0          16m
```

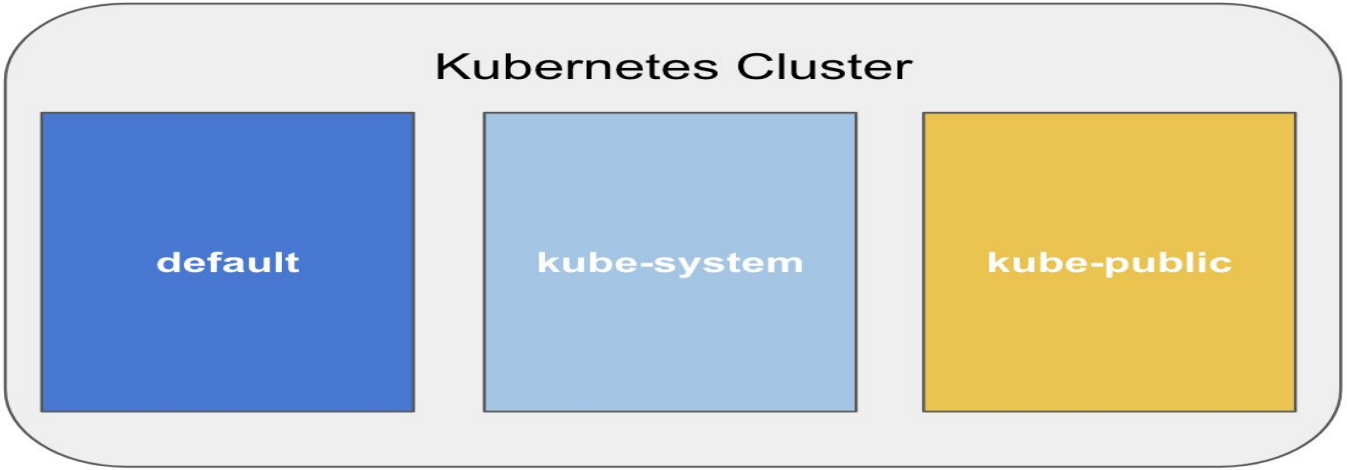
```
$ kubectl rollout undo deploy app-controller
```

Deployment



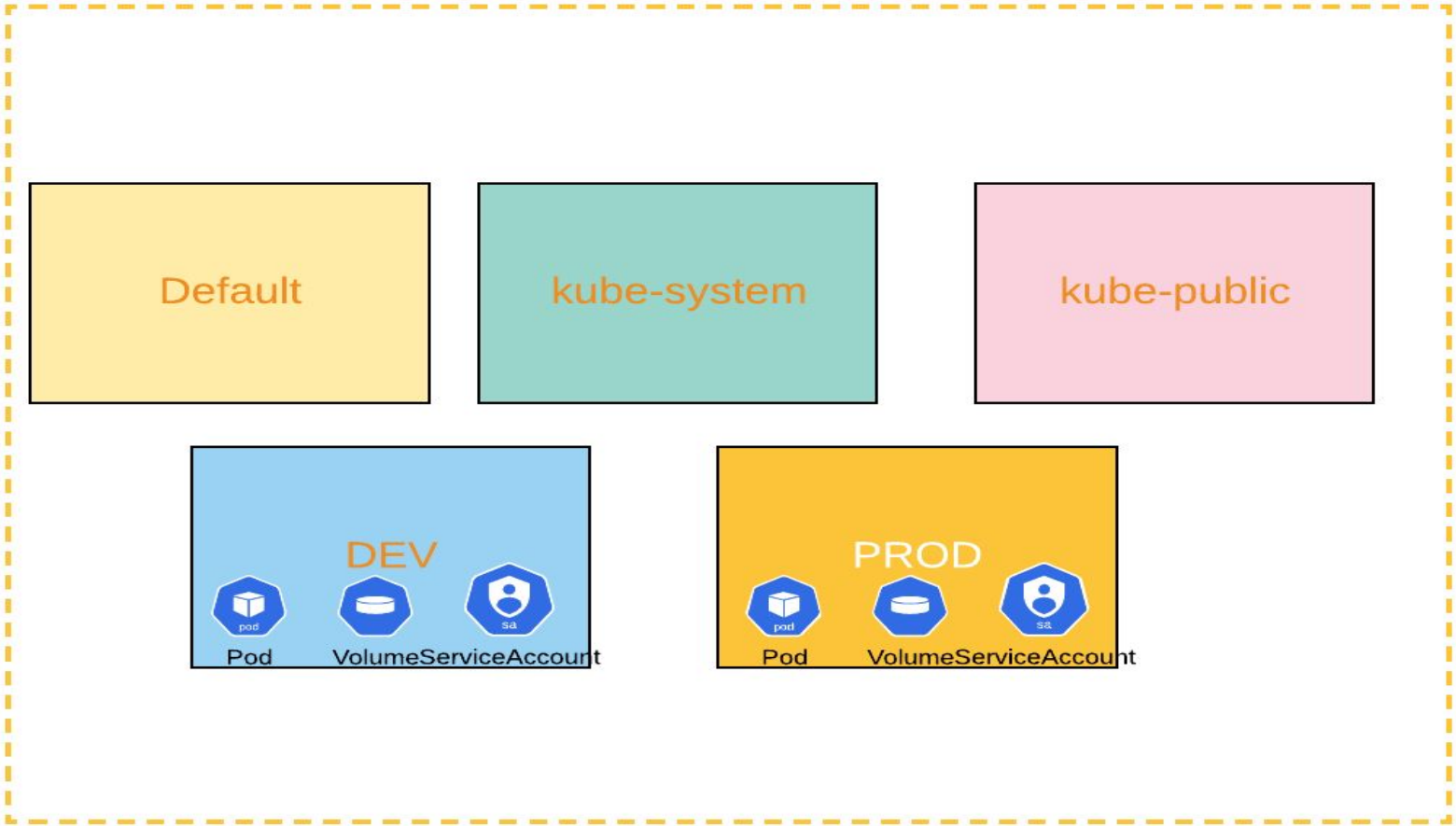
Namespaces

Group your resources



Deployment

```
$ kubectl get namespaces
NAME                   STATUS   AGE
default                Active   25h
kube-node-lease        Active   25h
kube-public            Active   25h
kube-system            Active   25h
```


Default
kube-system
kube-public
DEV
Pod
VolumeServiceAccount
PROD
Pod
VolumeServiceAccount

Connect SVC in Namespaces

Web Pod → db.service → DB Service

Web Pod → db-service.dev.svc.cluster.local → DB Service

```
mysql.connect("db-service.dev.svc.cluster.local")
```

Use Namespace

```
$ kubectl get all -n kube-system
NAME                                   READY   STATUS    RESTARTS   AGE
pod/coredns-fb8b8dccf-q5nnw            1/1     Running   5          25h
pod/coredns-fb8b8dccf-rq8sj            1/1     Running   5          25h
```

```
$ kubectl create namespace dev
namespace/dev created
```

```
$ kubectl create -f tom-app-deploy.yml -n dev
deployment.apps/app-controller created
```

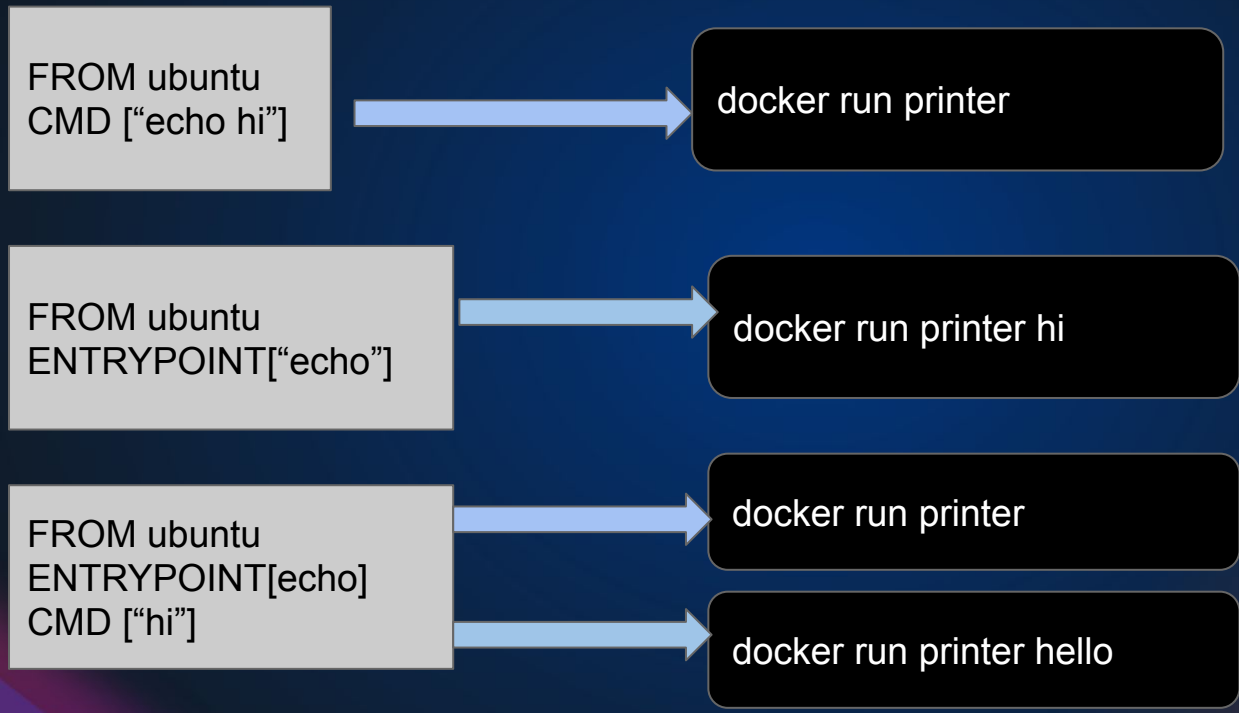
```
$ kubectl config set-context --current --namespace=kube-system
Context "minikube" modified.
```

```
apiVersion: v1
kind: Pod
metadata:
  name: app-pod
  namespace: dev
  labels:
    app: backend
    project: infinity
```

Command & Arguments

Pass Command & Args to your POD

Command & Entrypoint



Command & Entrypoint

FROM ubuntu
ENTRYPOINT[sleep]
CMD [“5”]

docker run ubuntu-halt 15

ENTRYPOINT[sleep]
CMD [“5”]

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: halting-pod
spec:
  containers:
    - name: pause4aMoment
      image: halt-ubuntu
      args: ["10"]
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: halting-pod
spec:
  containers:
    - name: pause4aMoment
      image: halt-ubuntu
      command: ["mysleepCommanc
      args: ["10"]
```

Environment Variables

Assign Variable Values

Environment Variables

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: db-pod
  namespace: dev
  labels:
    app: db
    project: infinity
spec:
  containers:
    - name: mysql-container
      image: mysql:5.7
      env:
        - name: MYSQL_DATABASE
          value: accounts
        - name: MYSQL_ROOT_PASSWORD
          value: somecomplexpassword
      ports:
```

Config Maps

Set & Inject Variables/files in POD

Create Config Maps | Imperative

```
$ kubectl create configmap db-config --from-literal=MYSQL_DATABASE=accounts \
> --from-literal=MYSQL_ROOT_PASSWORD=somecomplexpass
configmap/db-config created
```

```
$ kubectl get cm
NAME        DATA   AGE
db-config   2      5s
```

```
$ kubectl get cm db-config -o yaml
apiVersion: v1
data:
  MYSQL_DATABASE: accounts
  MYSQL_ROOT_PASSWORD: somecomplexpass
kind: ConfigMap
```

```
$ kubectl describe cm db-config
Name:         db-config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
MYSQL_DATABASE:
----
accounts
MYSQL_ROOT_PASSWORD:
```

Create Config Maps | Declarative

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: db-config
data:
  MYSQL_ROOT_PASSWORD: somecomplexpass
  MYSQL_DATABASE: accounts
```

```
$ kubectl create -f db-cm.yml
configmap/db-config created
```

POD Reading Config Maps

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: db-pod
  labels:
    app: db
    project: infinity
spec:
  containers:
    - name: mysql-container
      image: mysql:5.7
      envFrom:
        - configMapRef:
            name: db-config
      ports:
        - name: db-port
```

```yaml
spec:
  containers:
    - name: mysql-container
      image: mysql:5.7
      env:
        - name: DB_HOST
          valueFrom:
            configMapKeyRef:
              name: db-config
              key: DB_HOST
```

Secrets

Share encoded/encrypted variables to POD

Store and manage sensitive information, such as passwords

Create Secrets | Imperative

```
$ kubectl create secret generic db-secret --from-literal=MYSQL_ROOT_PASSWORD=somecomplexpassword
secret/db-secret created
```

```
# Create files needed for rest of example.
echo -n 'admin' > ./username.txt
echo -n '1f2d1e2e67df' > ./password.txt
```

```
kubectl create secret generic db-user-pass --from-file=./username.txt --from-file=./password.txt
```

```
$ kubectl get secret db-secret -o yaml
apiVersion: v1
data:
  MYSQL_ROOT_PASSWORD: c29tZWNvbXBsZXhwYXNzd29yZA==
kind: Secret
metadata:
```

Create Secrets | Declarative

```
$ echo -n "somecomplexpassword" | base64
c29tZWNvbXBsZXhwYXNzd29yZA==
```

db-secret.yml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  my_root_pass: c29tZWNvbXBsZXhwYXNzd29yZA==
```

POD Reading Secret

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: db-pod
  labels:
    app: db
    project: infinity
spec:
  containers:
    - name: mysql-container
      image: mysql:5.7
      envFrom:
        - secretRef:
            name: db-secret
```

```yaml
spec:
  containers:
    - name: mysql-container
      image: mysql:5.7
      env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: db-secret
              key: my_root_pass
```

Secret as a Volume

```
apiVersion: v1
kind: Pod
metadata:
  name: db-pod
  labels:
    app: db
    project: infinity
spec:
  containers:
    - name: mysql-container
      image: mysql:5.7
      ports:
        - name: db-port
          containerPort: 3306
  volumes:
    - name: db-secret-vol
      secret:
        secretName: db-secret
~
```