

Etcd - metadata store

scheduler - scheduling

controller - Brain

kubelet - agent run on each
container (workers)

container runtime → Docker

- kube-apiserver is only available in the Master

```
kubectl run hello-minikube
```

```
kubectl cluster-info
```

```
kubectl get nodes
```

```
kubectl get nodes -o wide
```

- To know the OS version running on the containers.

- Pods

- Smallest object in Kubernetes
pods runs on node
- When traffic is increased new
separate pods can be added onto the
node to keep up with the traffic.
- We do not add pods to existing
containers to scale up.
- Helper containers are usually put
into same pod.

pod = group of whales (Docker icon)

- With pods we can manage multiple containers
which are inside that pod. for ex. stopping
pod will stop all
the container

within that pod.

- `kubectl get pods`

To view all running pods.

- `kubectl describe pod nginx`

↑

name of pod

To view detailed info on pod

- `kubectl run hello-minikube`
- `kubectl cluster-info`
- `kubectl get nodes`

- To run kubectl

Name for an
img.

↓

- `kubectl run nginx --image = nginx`

- To run k8 container with image
nginx

- To list pods Available

- `kubectl get pods`

- More details on the pod.

```
kubectl describe pod nginx
```

- Pods with YAML

To create a YAML file

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

YAML files contains

apiVersion: v1

kind:

metadata:

spec:

To check the nodes where this pods are placed on -

```
kubectl get pods -o wide
```

creating image in kubernetes

YAML definition for the pod

- `kubectl run redis --image=redis/2.9`
`--dry-run=client -o yaml > redis.yaml`

creating a pod

- `kubectl create -f redis.yaml`

* kubernetes Controller

— Replication Controller

- High Availability
- Increase Scalability

- Replication Controller ← old
- Replica set ← new

Replicas

- To view the number of replicas

- `kubectl get replicaset`
or

- `kubectl get rs`

To describe replicaset

```
kubectl describe replicaset new-rs-set  
                             or  
                             name
```

Replica Set has
Spec:
template:
replicas:
only in → Selector:
replica set

Scaling of replica → can be done automatically
edit file then

1. - `kubectl replace -f replicaset-definition.yml`
2. - `kubectl scale --replicas=6 -f replicaset-definition.yml`

```
> kubectl create -f replicaset-definition.yml
> kubectl get replicaset
> kubectl delete replicaset myapp-replicaset
> kubectl replace -f replicaset-definition.yml
> kubectl scale --replicas=6 -f replicaset-definition.yml
```

*Also deletes all underlying PODs

3. changing in memory replicaset

`kubectl edit replicaset my-app`
↑

App name
Works only if we have

vim editor on system.

To see the replicaset description

- `kubectl describe replicaset myapp-replicaset`

* You cannot create a pod with same label as replicaset. it will automatically terminate

• You can scale up or down using multiple way

① By changing the in-memory yaml file in that case you don't need to run extra commands
ex.

• `kubectl edit replicaset myapp-replicaset`
(needs vim) ↑ App name.

② By changing the replica.yaml file

By simply going into yaml file and setting different number for replica
after change you have to replace the yaml file.

- `kubectl replace replicaset -f replica.yaml`

③. By using command

- `kubectl scale --replicas=6 replica.yaml.`

* `kubectl deployments`

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-srbj1            1/1     Running   0          2m
myapp-deployment-6795844b58-hdu55            1/1     Running   0          2m
myapp-deployment-6795844b58-lfjhw            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
```

`kubectl get all`

* Rollout Commands

- `kubectl rollout status deployment/
my-app deployment`

- `kubectl rollout history deployment/
myapp-deployment`

- ① • **Rolling update** is the default deployment strategy.
→ Which means updates will be rolled out one-by-one.

To rollout we simply make changes into deployment-definition for ex. changing `nginx 1.6 → nginx 1.7` and once done we simply run

- `kubectl apply -f deployment-definition.yaml`

② Recreate strategy — not default

We delete all the previous pods and create new pods **all at once**

disadvantage - Downtime

* Rollback a change.

• **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.yml
	> 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

- When we mis-configure or give wrong image name in that case the default rollout try to rollout update on one image and when it find out that it is not possible due to there is no image it get stuck and out of that 5 replicas only 4 are running

and remaining one is stuck waiting for image to fetch which does not exist.

We can fix such errors by using `undo`

- `kubectl rolling undo -f ./deployment.yaml`
or

- `kubectl rolling undo deployment myapp-deployment.`

* Networking

- All pods are internally connected.
- kubernetes cluster does not create a connection directly that's why we need an external networking tools.
- kubernetes cluster may have same ip's directly that's why we need an external networking tools.

- kubernetes services

Service is an mapping tool in k8 to map internal port to communicate with outside local connections.

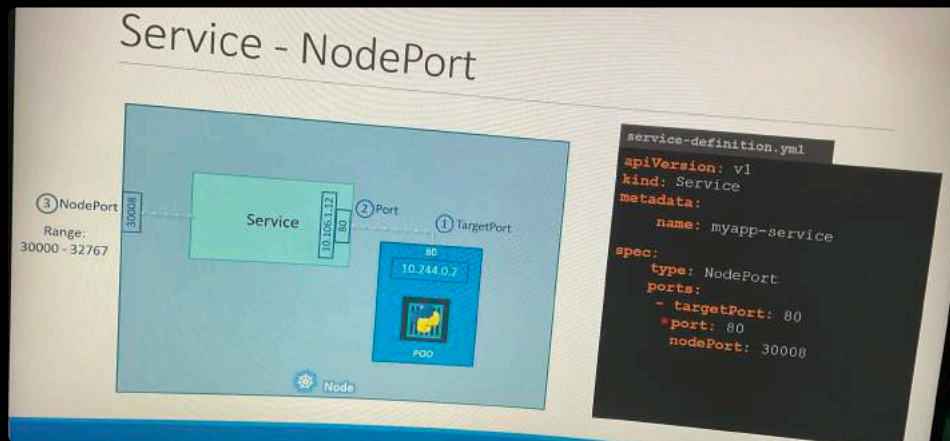
Service types

① Nodeport - Makes internal port accessible

② cluster IP -
creates virtual IP to communicate

③ Load Balancers

① NodePort -



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
```

```
> kubectl create -f service-definition.yml
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: 350px;
    margin: 0 auto;
    font-family: Tahoma, Vordana, Arial, sans-serif;
  }
</style>
</head>
<body>
```

② clusterIP

Basically a single entry-point for communication.

Default by Kubernetes service.

```
service-definition.yml
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.yml
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	34d
back-end	ClusterIP	10.106.127.123	<none>	80/TCP	2m

③ Load Balancers -

- only works with supported cloud platform



- Entrypoints 0 out means there is issue with naming betⁿ pods.
make sure labels are same for apps.

• Microservices -

- docker run --links can be used for linking ← old method and may get deprecated in the future.

• Deploying



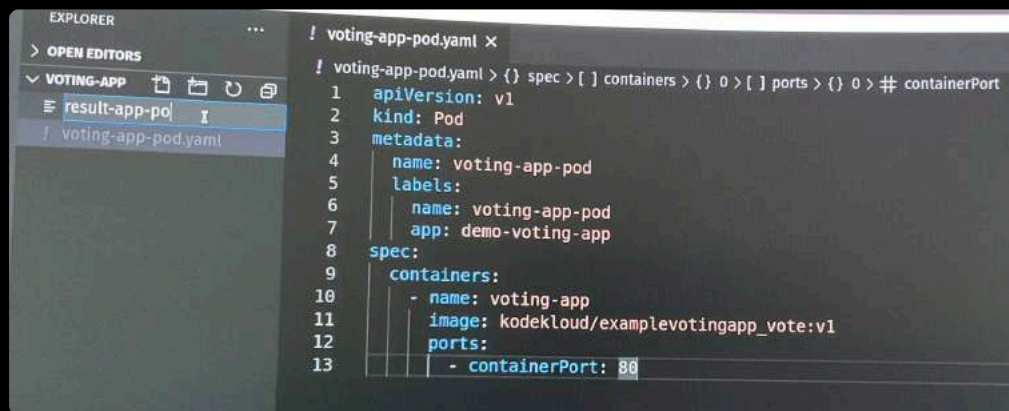
Kubernetes definition files are used to define the desired state of Kubernetes objects such as pods, deployments, services, and so on. There are different types of definition files used in Kubernetes, including:

1. Pod definition file: This file is used to define a single pod in Kubernetes. It includes details such as the container image to use, the command to run, the environment variables to set, and so on.
2. Deployment definition file: This file is used to define a deployment in Kubernetes. A deployment is a higher-level object that manages a set of pods, making it easy to manage rolling updates, scaling, and more.
3. Service definition file: This file is used to define a service in Kubernetes. A service provides a stable endpoint for accessing a set of pods, allowing for easy load balancing and service discovery.
4. ConfigMap definition file: This file is used to define a ConfigMap in Kubernetes. A ConfigMap is a key-value store for storing configuration data that can be used by pods and other Kubernetes objects.
5. Secret definition file: This file is used to define a Secret in Kubernetes. A Secret is a secure way to store sensitive information such as passwords, API keys, and so on.

Each type of definition file serves a specific purpose and includes different fields and configurations depending on the object being defined. By using these files, Kubernetes operators can declaratively specify the desired state of their applications, making it easier to manage and maintain their infrastructure.

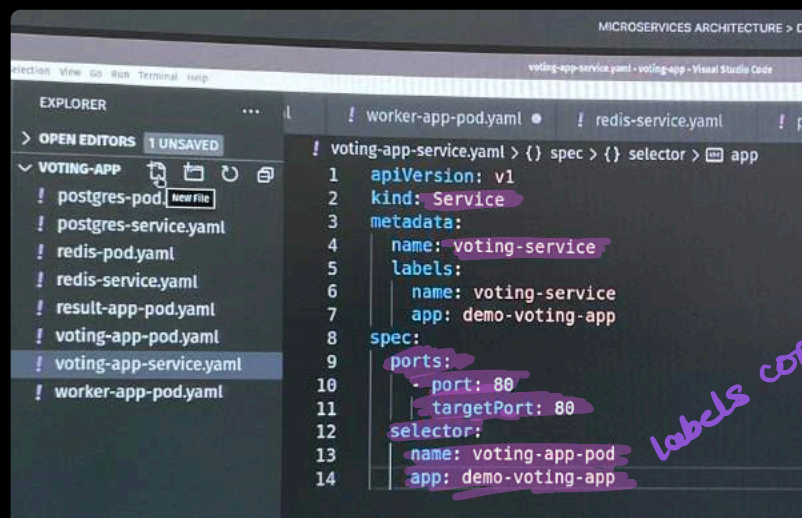
Based on different working of files
we might have different content
among files

Pod-definition Files



```
! voting-app-pod.yaml x
! voting-app-pod.yaml > {} spec > [ ] containers > {} 0 > [ ] ports > {} 0 > # containerPort
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: voting-app-pod
5    labels:
6      name: voting-app-pod
7      app: demo-voting-app
8  spec:
9    containers:
10     - name: voting-app
11       image: kodekloud/examplevotingapp_vote:v1
12       ports:
13         - containerPort: 80
```

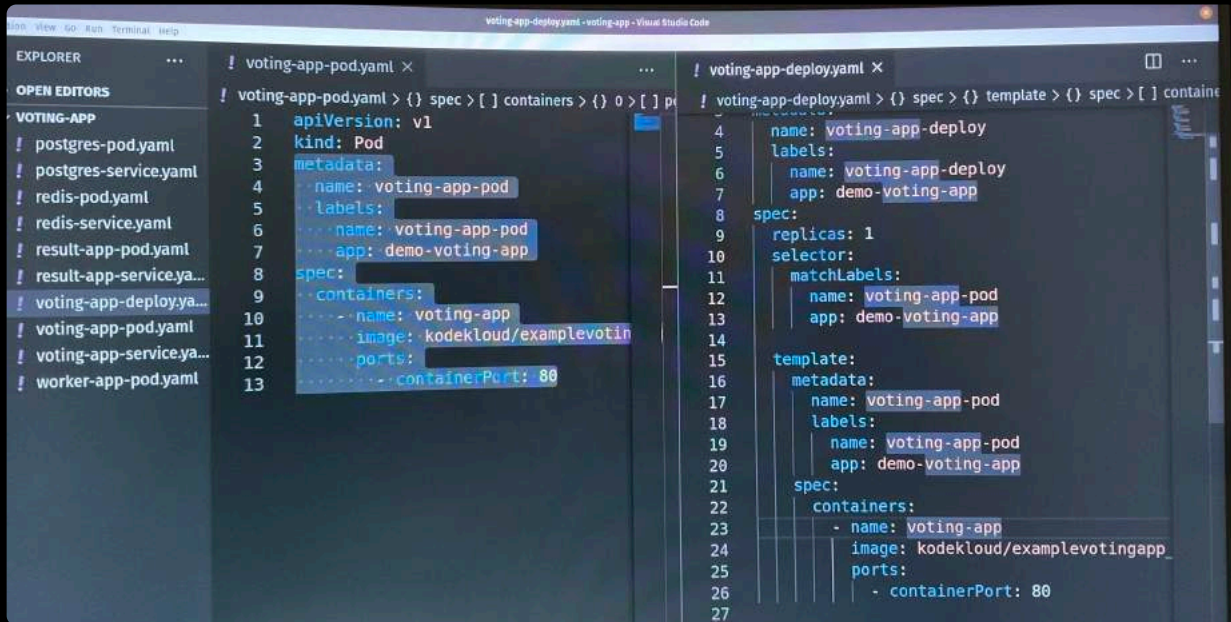
Voting App - service.yaml



```
! voting-app-service.yaml > {} spec > {} selector > app
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: voting-service
5    labels:
6      name: voting-service
7      app: demo-voting-app
8  spec:
9    ports:
10     - port: 80
11       targetPort: 80
12    selector:
13      name: voting-app-pod
14      app: demo-voting-app
```

Labels copy from
definition.

Deployments → Replica sets



The screenshot shows the Visual Studio Code interface with two YAML files open. The Explorer on the left lists several files under the 'VOTING-APP' folder. The main editor displays the contents of 'voting-app-pod.yaml' and 'voting-app-deploy.yaml'.

```
! voting-app-pod.yaml x
! voting-app-deploy.yaml x

! voting-app-pod.yaml > {} spec > [] containers > {} 0 > [] pr
! voting-app-deploy.yaml > {} spec > {} template > {} spec > [] containe

1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: voting-app-pod
5   labels:
6     name: voting-app-pod
7     app: demo-voting-app
8 spec:
9   containers:
10     - name: voting-app
11       image: kodekloud/examplevotingapp
12       ports:
13         - containerPort: 80

4 name: voting-app-deploy
5 labels:
6   name: voting-app-deploy
7   app: demo-voting-app
8 spec:
9   replicas: 1
10  selector:
11    matchLabels:
12      name: voting-app-pod
13      app: demo-voting-app
14  template:
15    metadata:
16      name: voting-app-pod
17      labels:
18        name: voting-app-pod
19        app: demo-voting-app
20    spec:
21      containers:
22        - name: voting-app
23          image: kodekloud/examplevotingapp
24          ports:
25            - containerPort: 80
```