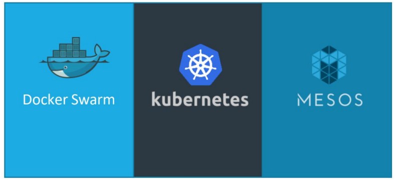
Introduction to Kubernetes

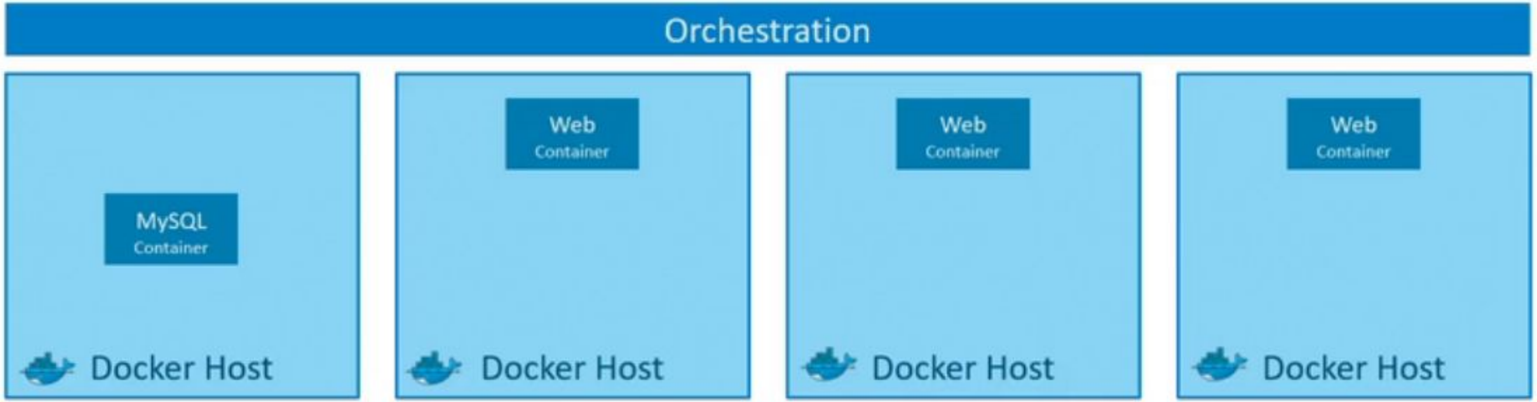
Waheed Iqbal

Infrastructure Management for Scalable Applications (Fall 2022)
Department of Data Science, FCIT, University of the Punjab
Lahore, Pakistan

Container Orchestration



- Container orchestration is the automated management of containerized applications
- Orchestration platforms automate the deployment, scaling, and management of containerized applications across multiple hosts
- They offer features like service discovery, load balancing, storage orchestration, and automated rollouts and rollbacks
- Popular container orchestration platforms include Kubernetes, Docker Swarm, and Apache Mesos
- Container orchestration is a key component of modern DevOps and cloud-native application development



Orchestration

MySQL Container

Docker Host

Web Container

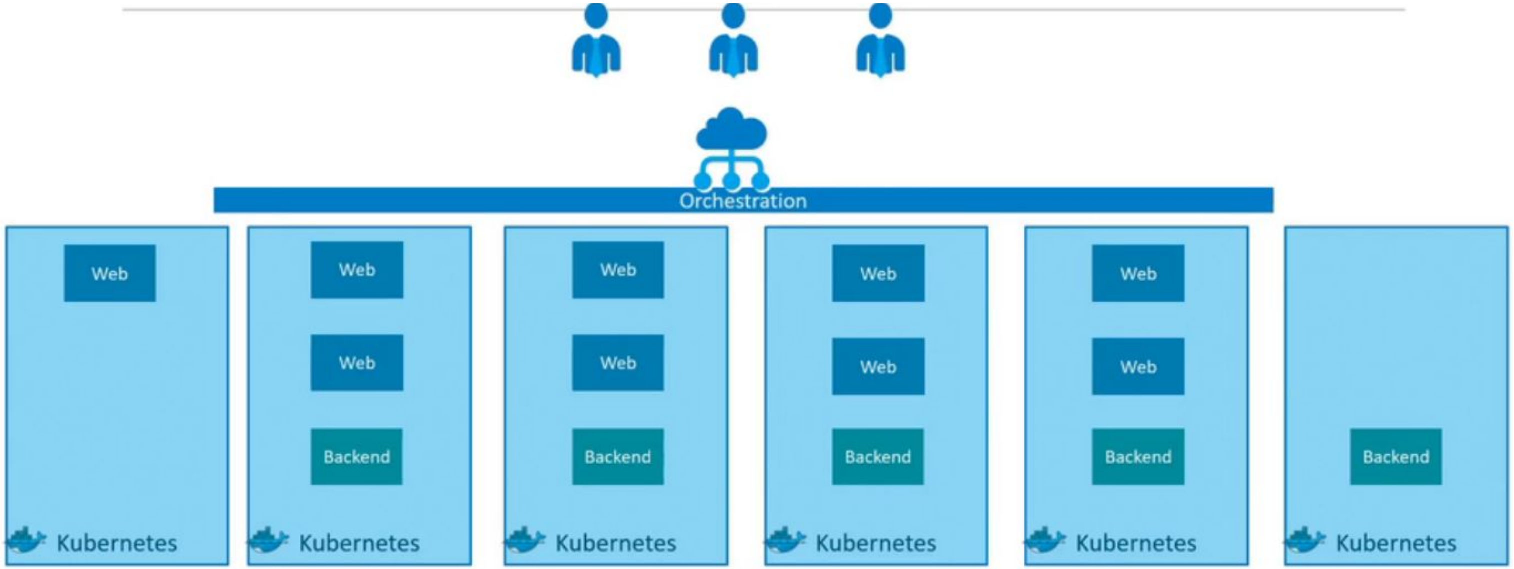
Docker Host

Web Container

Docker Host

Web Container

Docker Host



Orchestration

Web

Kubernetes

Web

Web

Backend

Kubernetes

Web

Web

Backend

Kubernetes

Web

Web

Backend

Kubernetes

Web

Web

Backend

Kubernetes

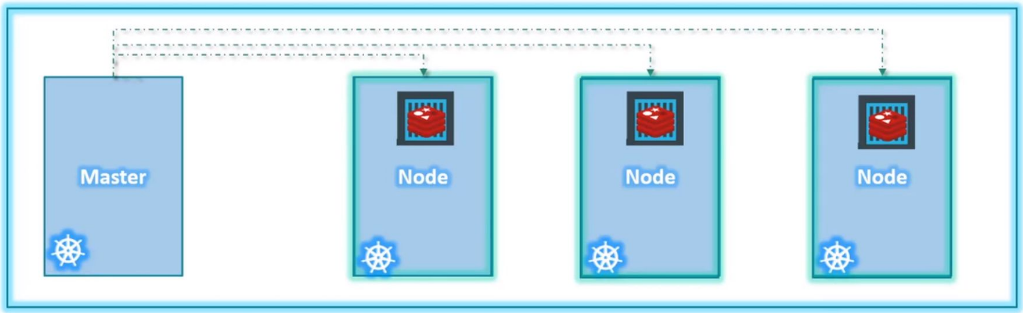
Backend

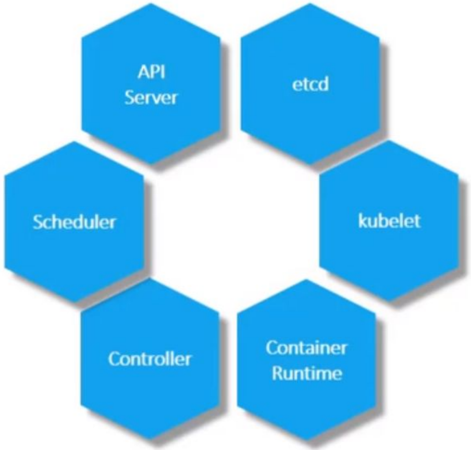
Kubernetes

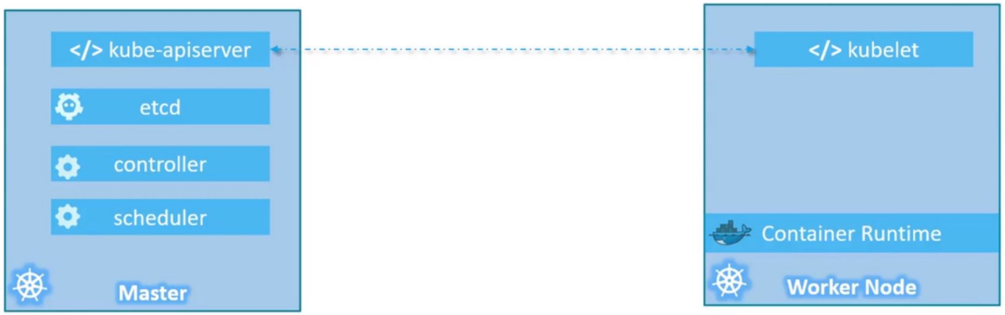
Kubernetes

- Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications
- It was originally developed by Google and is now maintained by the Cloud Native Computing Foundation (CNCF)
- Kubernetes is based on a distributed architecture and uses a master-worker node model to manage and orchestrate containers across multiple hosts
- Kubernetes can be deployed on any cloud platform or on-premises infrastructure

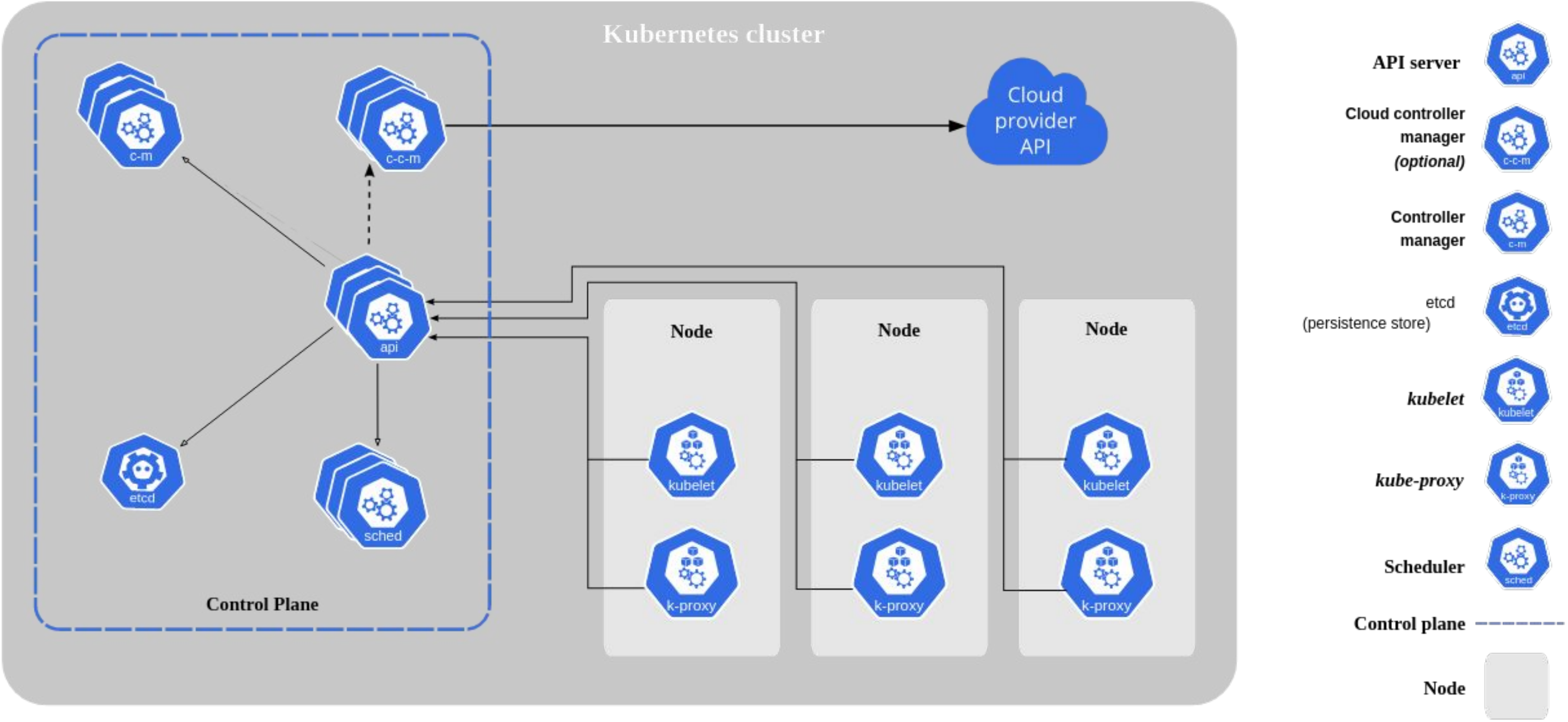
Kubernetes Components







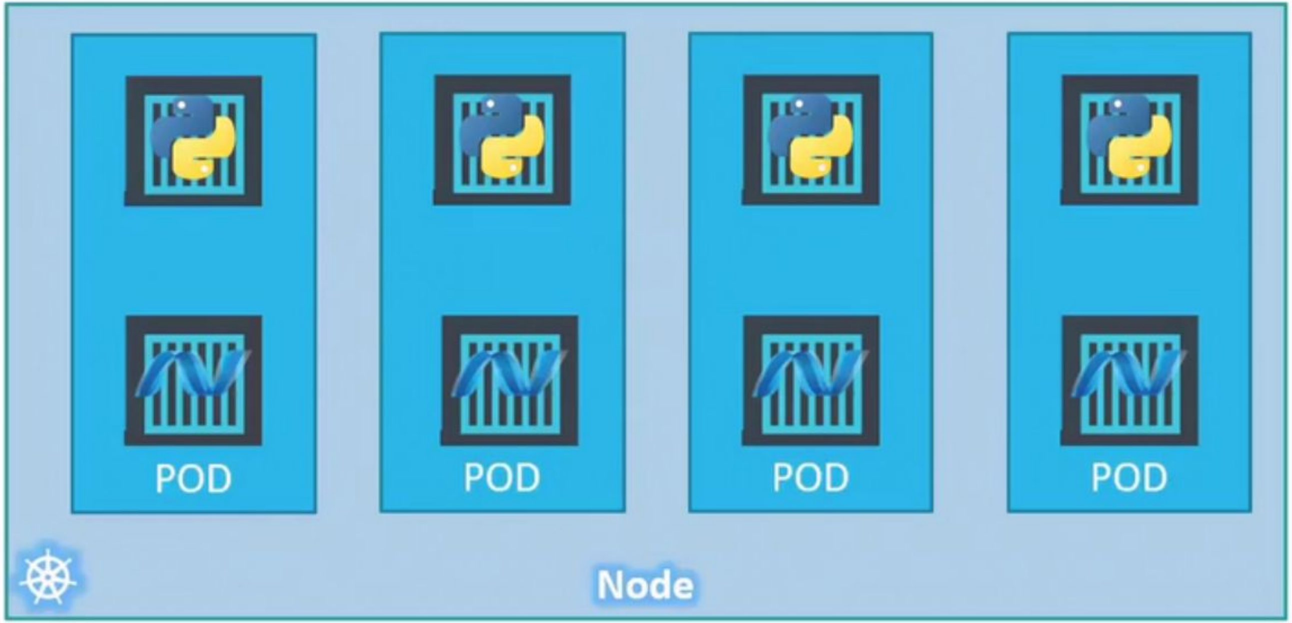
Kubernetes Components (Cont.)



Pod

- Pod is a collection of one or more Linux containers, packaged together to maximize the benefits of resource sharing via cluster management
- Pods provide a logical host for containers and are used to encapsulate and run containerized applications and services
- Containers within a Pod share the same network namespace, which means they can communicate with each other using localhost and share the same IP address and port space
- Containers within a Pod also share the same storage namespace, which means they can access and use the same storage volumes mounted into the Pod
- Pods can be scaled horizontally by creating multiple replicas of the same Pod

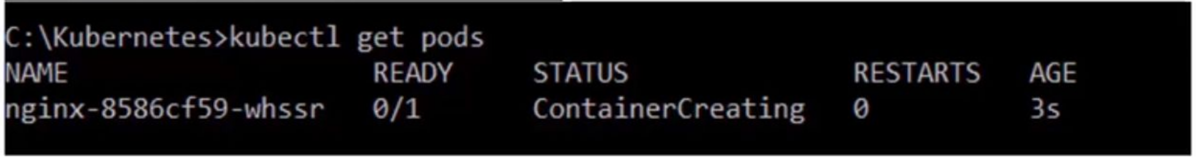
Pod (Cont.)



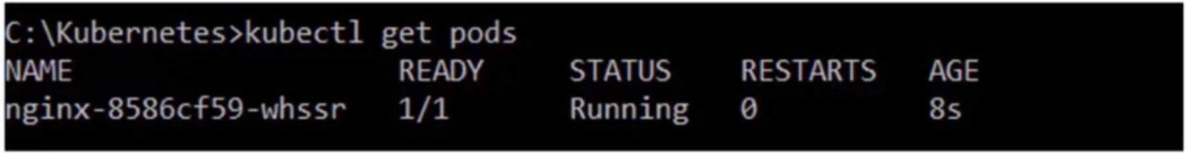
Pod (Cont.)

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

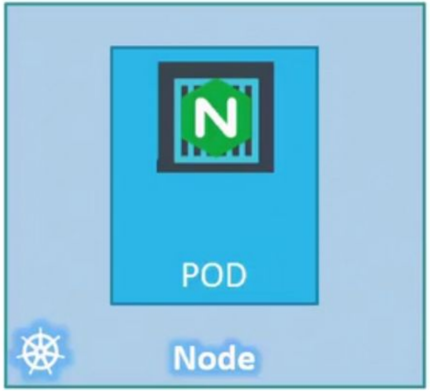
```
kubectl get pods
```



```
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
```

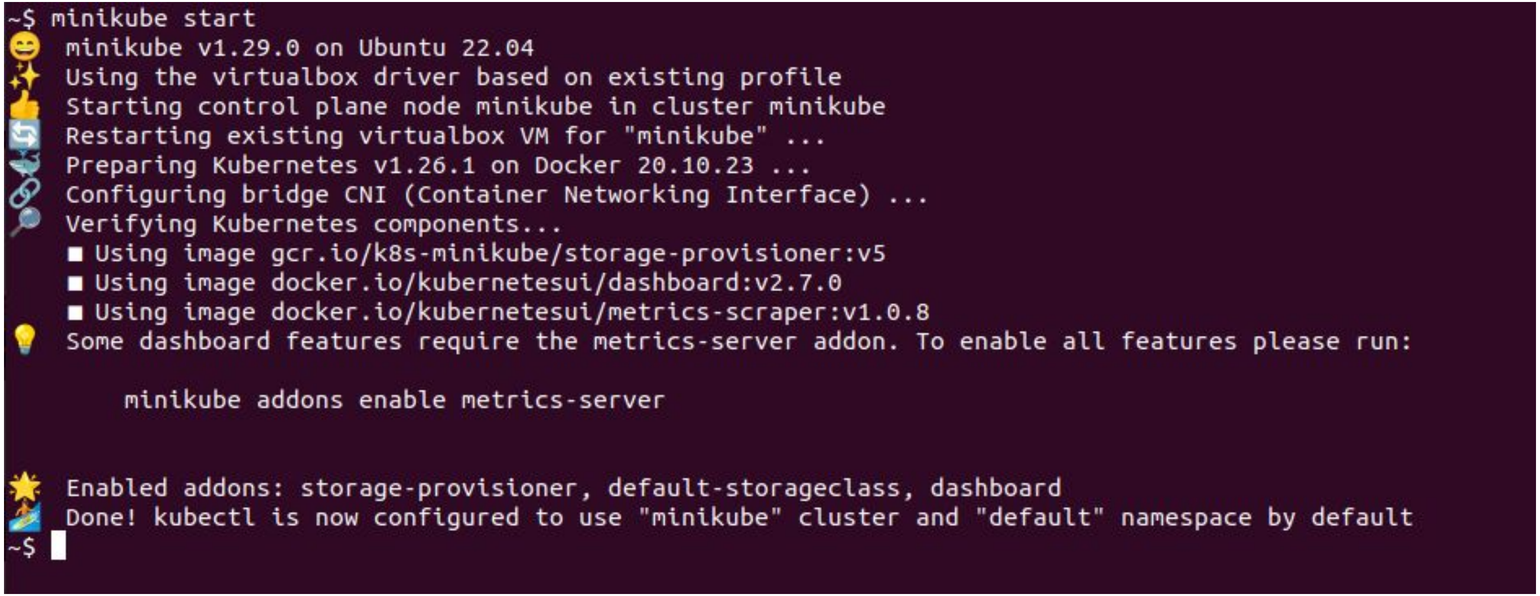


Minikube

- Minikube is a lightweight, single-node Kubernetes cluster on a local machine, which enables developers to test and experiment with Kubernetes without having to set up a full-scale production environment.
- Minikube is easy to install and can be run on most operating systems, including Windows, macOS, and Linux.
- Minikube uses a single-node cluster configuration, which means that it runs all the Kubernetes components, such as the API server, etcd, and kubelet, on a single virtual machine or container, instead of across multiple nodes.
- Minikube provides a local Docker registry, which enables developers to build and test container images on the same machine where the Kubernetes cluster is running.
- Minikube can be used to test and debug Kubernetes applications and services in a local environment, before deploying them to a production cluster.

Installing Minikube

- `curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube_latest_amd64.deb`
- `sudo dpkg -i minikube_latest_amd64.deb`
- If everything goes well, execute minikube start command.



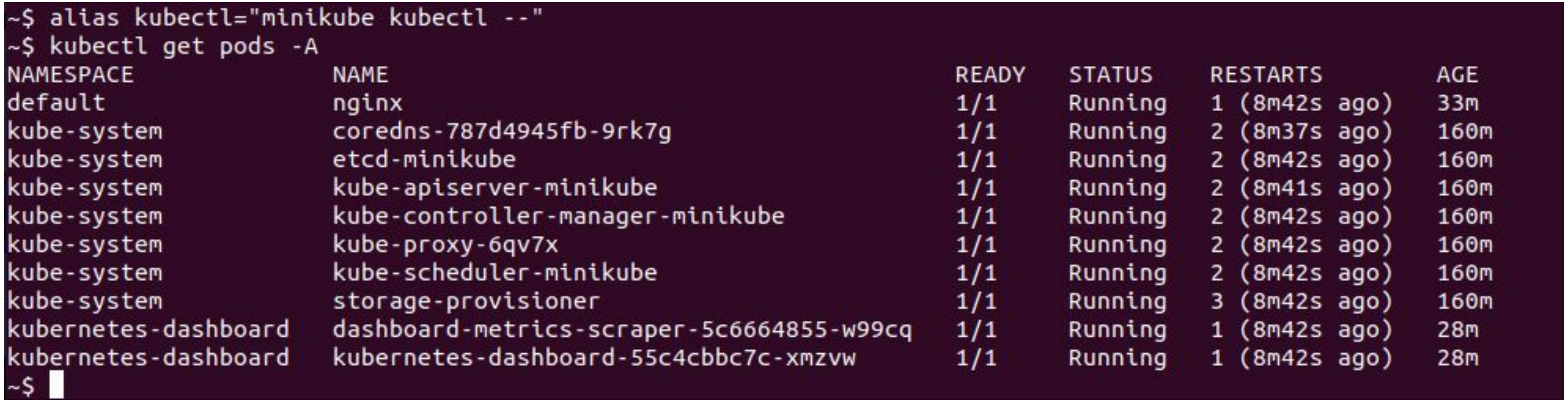
```
~$ minikube start
😄  minikube v1.29.0 on Ubuntu 22.04
✨  Using the virtualbox driver based on existing profile
👍  Starting control plane node minikube in cluster minikube
🔄  Restarting existing virtualbox VM for "minikube" ...
🐳  Preparing Kubernetes v1.26.1 on Docker 20.10.23 ...
🔗  Configuring bridge CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
    ▪ Using image docker.io/kubernetesui/dashboard:v2.7.0
    ▪ Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
💡  Some dashboard features require the metrics-server addon. To enable all features please run:

        minikube addons enable metrics-server


🌟  Enabled addons: storage-provisioner, default-storageclass, dashboard
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
~$ 
```

Minikube (Cont.)

‘Minikube kubectl’ is used to run all the cluster related command. We can create the alias and rename it as the kubectl.

minikube kubectl -- get pods -A

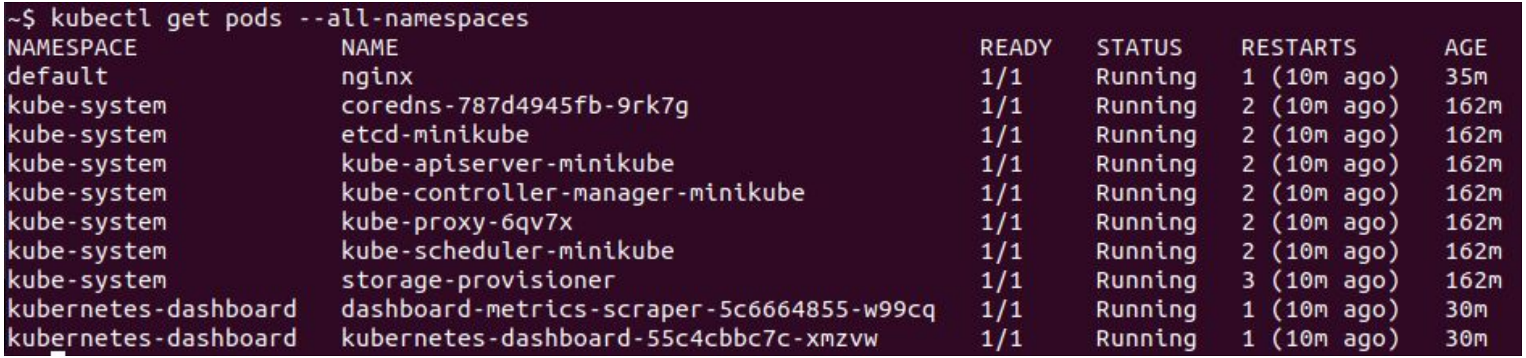
alias kubectl="minikube kubectl --"



```
~$ alias kubectl="minikube kubectl --"
~$ kubectl get pods -A
NAMESPACE              NAME                                         READY   STATUS    RESTARTS        AGE
default                nginx                                        1/1     Running   1 (8m42s ago)   33m
kube-system            coredns-787d4945fb-9rk7g                     1/1     Running   2 (8m37s ago)   160m
kube-system            etcd-minikube                                1/1     Running   2 (8m42s ago)   160m
kube-system            kube-apiserver-minikube                      1/1     Running   2 (8m41s ago)   160m
kube-system            kube-controller-manager-minikube             1/1     Running   2 (8m42s ago)   160m
kube-system            kube-proxy-6qv7x                             1/1     Running   2 (8m42s ago)   160m
kube-system            kube-scheduler-minikube                      1/1     Running   2 (8m42s ago)   160m
kube-system            storage-provisioner                          1/1     Running   3 (8m42s ago)   160m
kubernetes-dashboard   dashboard-metrics-scraper-5c6664855-w99cq    1/1     Running   1 (8m42s ago)   28m
kubernetes-dashboard   kubernetes-dashboard-55c4cbbc7c-xmzvw        1/1     Running   1 (8m42s ago)   28m
~$
```

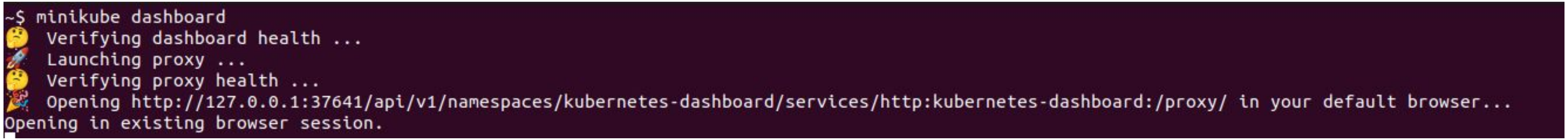
Namespaces in Kubernetes

- Namespaces in a Kubernetes cluster are a way to organize and isolate resources and objects within the cluster.
- A namespace provides a scope for naming and controlling the visibility of Kubernetes resources, such as Pods, Services, ConfigMaps, and Secrets.
- Each Kubernetes resource belongs to a specific namespace, and if no namespace is specified, the resource is created in the default namespace.
- By using multiple namespaces, you can logically partition the cluster into smaller, more manageable units, and avoid naming conflicts and resource collisions.

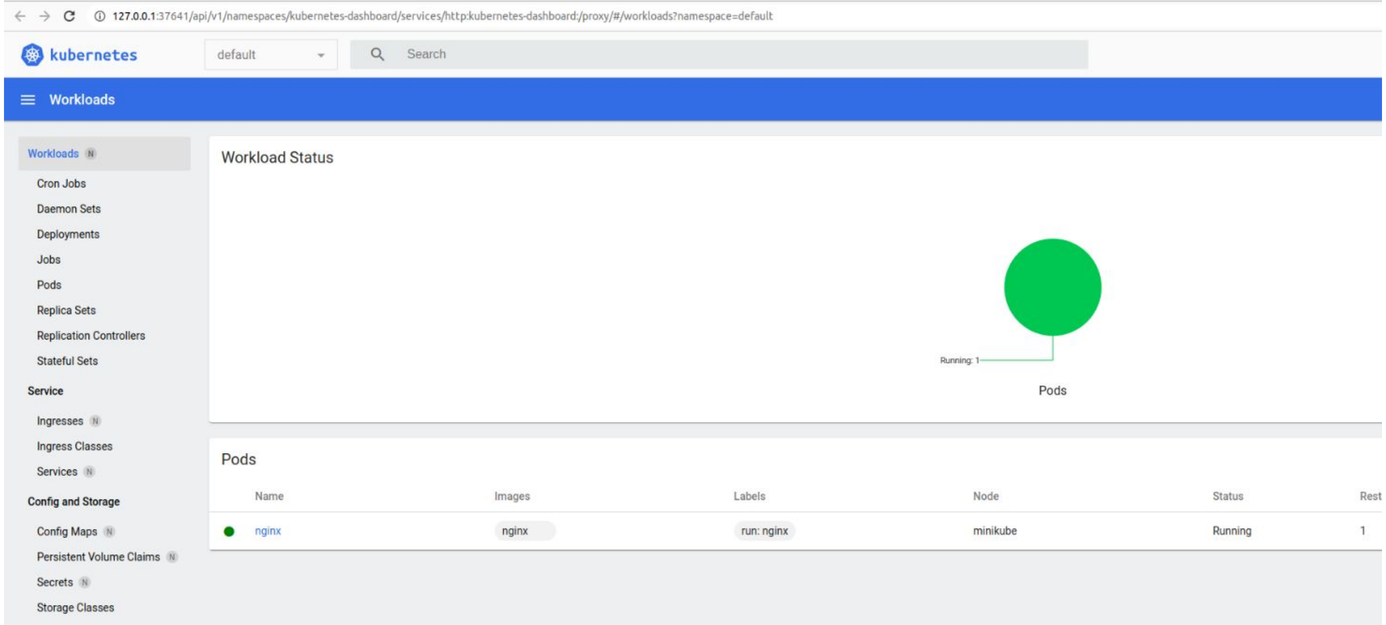


```
~$ kubectl get pods --all-namespaces
NAMESPACE              NAME                                        READY   STATUS    RESTARTS      AGE
default                nginx                                       1/1     Running   1 (10m ago)   35m
kube-system            coredns-787d4945fb-9rk7g                    1/1     Running   2 (10m ago)   162m
kube-system            etcd-minikube                               1/1     Running   2 (10m ago)   162m
kube-system            kube-apiserver-minikube                     1/1     Running   2 (10m ago)   162m
kube-system            kube-controller-manager-minikube            1/1     Running   2 (10m ago)   162m
kube-system            kube-proxy-6qv7x                            1/1     Running   2 (10m ago)   162m
kube-system            kube-scheduler-minikube                     1/1     Running   2 (10m ago)   162m
kube-system            storage-provisioner                         1/1     Running   3 (10m ago)   162m
kubernetes-dashboard   dashboard-metrics-scraper-5c6664855-w99cq   1/1     Running   1 (10m ago)   30m
kubernetes-dashboard   kubernetes-dashboard-55c4cbbc7c-xmzvw       1/1     Running   1 (10m ago)   30m
```

Kubernetes Dashboard



```
~$ minikube dashboard
🤔  Verifying dashboard health ...
🚀  Launching proxy ...
🤔  Verifying proxy health ...
🎉  Opening http://127.0.0.1:37641/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ in your default browser...
Opening in existing browser session.
```



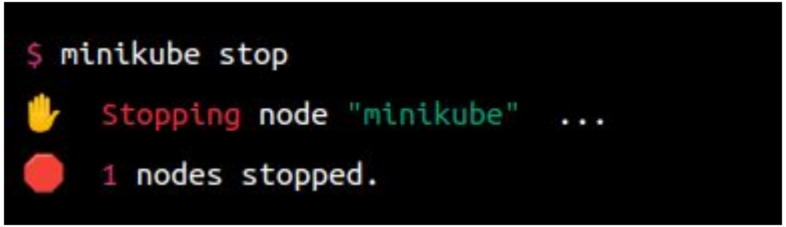
127.0.0.1:37641/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/#/workloads?namespace=default

Pods

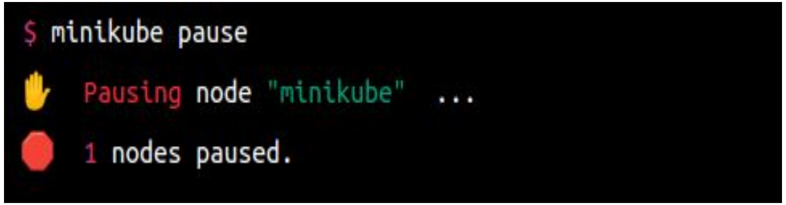
	Name	Images	Labels	Node	Status	Rest
●	nginx	nginx	run: nginx	minikube	Running	1

Few Important Minikube Commands

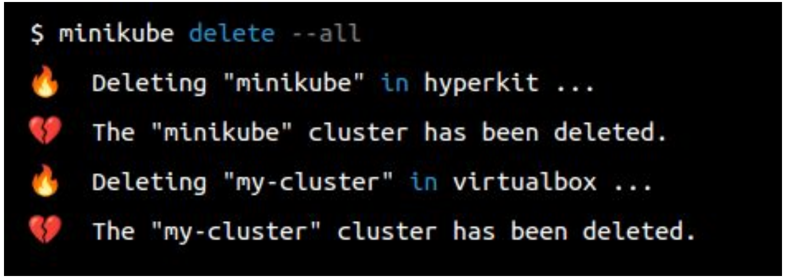
- minikube pause
- minikube unpause
- minikube stop
- minikube addons list
- minikube delete --all



```
$ minikube stop
✋  Stopping node "minikube"  ...
🛑  1 nodes stopped.
```

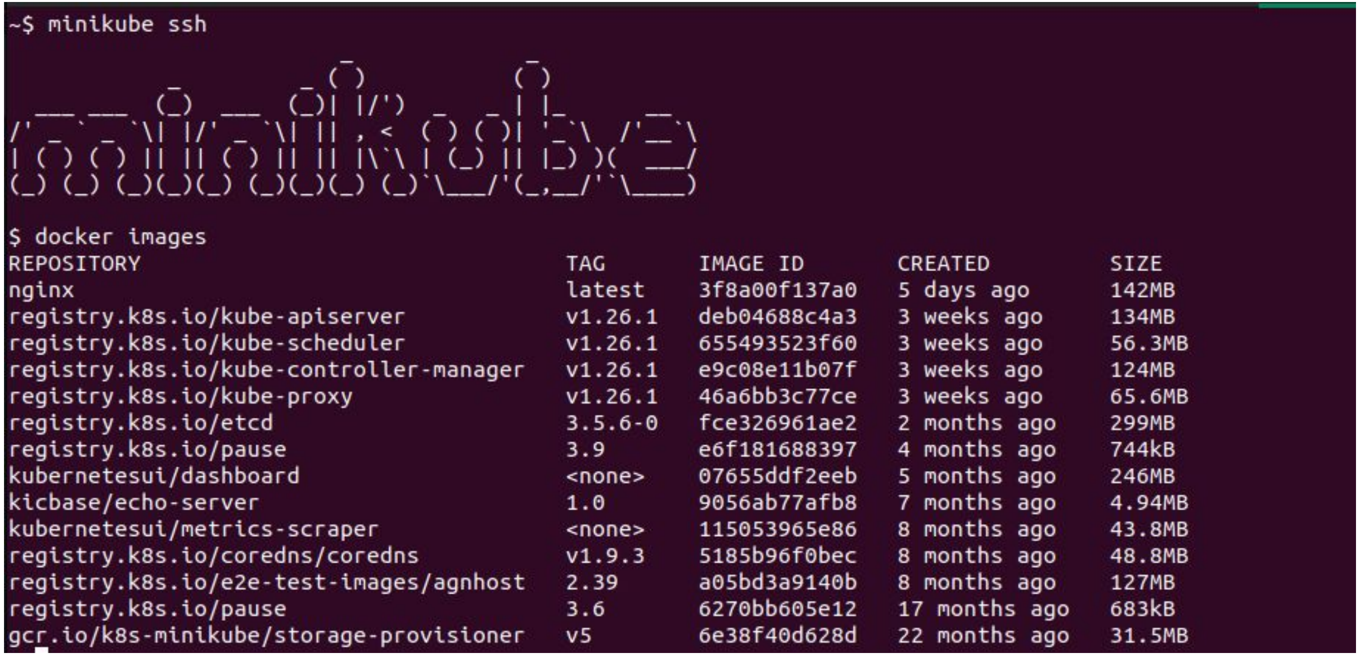


```
$ minikube pause
✋  Pausing node "minikube"  ...
🛑  1 nodes paused.
```



```
$ minikube delete --all
🔥  Deleting "minikube" in hyperkit ...
💔  The "minikube" cluster has been deleted.
🔥  Deleting "my-cluster" in virtualbox ...
💔  The "my-cluster" cluster has been deleted.
```

Minikube VM

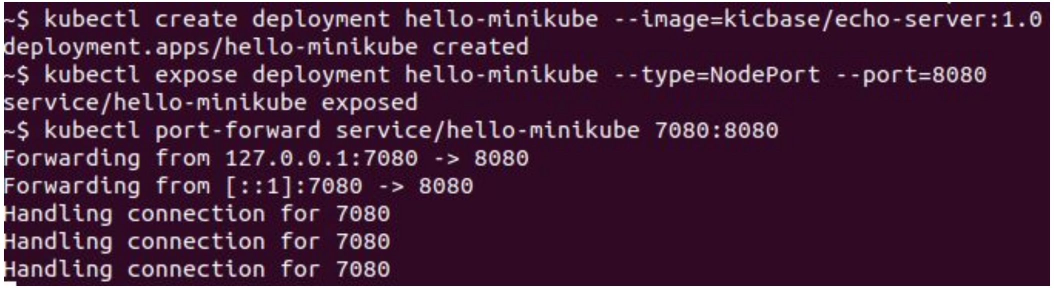


```
~$ minikube ssh
                         _             _
            _         _ ( )           ( )
  ___ ___  (_)  ___  (_)| |/')  _   _ | |_      __
/' _ ` _ `\| |/' _ `\| || , <  ( ) ( )| '_`\  /'__`\
| ( ) ( ) || || ( ) || || |\`\ | (_) || |_) )(  ___/
(_) (_) (_)(_)(_) (_)(_)(_) (_)`\___/'(_,__/'`\____)

$ docker images
REPOSITORY                                TAG       IMAGE ID       CREATED         SIZE
nginx                                     latest    3f8a00f137a0   5 days ago      142MB
registry.k8s.io/kube-apiserver            v1.26.1   deb04688c4a3   3 weeks ago     134MB
registry.k8s.io/kube-scheduler            v1.26.1   655493523f60   3 weeks ago     56.3MB
registry.k8s.io/kube-controller-manager   v1.26.1   e9c08e11b07f   3 weeks ago     124MB
registry.k8s.io/kube-proxy                v1.26.1   46a6bb3c77ce   3 weeks ago     65.6MB
registry.k8s.io/etcd                      3.5.6-0   fce326961ae2   2 months ago    299MB
registry.k8s.io/pause                     3.9       e6f181688397   4 months ago    744kB
kubernetesui/dashboard                    <none>    07655ddf2eeb   5 months ago    246MB
kicbase/echo-server                       1.0       9056ab77afb8   7 months ago    4.94MB
kubernetesui/metrics-scraper              <none>    115053965e86   8 months ago    43.8MB
registry.k8s.io/coredns/coredns           v1.9.3    5185b96f0bec   8 months ago    48.8MB
registry.k8s.io/e2e-test-images/agnhost   2.39      a05bd3a9140b   8 months ago    127MB
registry.k8s.io/pause                     3.6       6270bb605e12   17 months ago   683kB
gcr.io/k8s-minikube/storage-provisioner   v5        6e38f40d628d   22 months ago   31.5MB
```

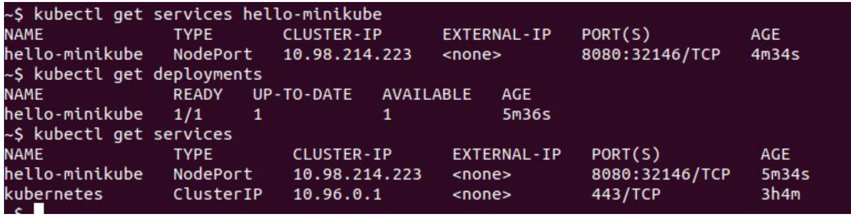
Deployment in Kubernetes

The kubectl create deployment command is used to create a new Deployment in a Kubernetes cluster. A Deployment is a Kubernetes object that manages a set of identical Pods, ensuring that the desired number of replicas are running and replacing any that fail or become unresponsive.

- kubectl create deployment hello-minikube --image=kicbase/echo-server:1.0
- kubectl expose deployment hello-minikube --type=NodePort --port=8080
- kubectl port-forward service/hello-minikube 7080:8080
- Open in the browser: http://localhost:7080/



```
~$ kubectl create deployment hello-minikube --image=kicbase/echo-server:1.0
deployment.apps/hello-minikube created
~$ kubectl expose deployment hello-minikube --type=NodePort --port=8080
service/hello-minikube exposed
~$ kubectl port-forward service/hello-minikube 7080:8080
Forwarding from 127.0.0.1:7080 -> 8080
Forwarding from [::1]:7080 -> 8080
Handling connection for 7080
Handling connection for 7080
Handling connection for 7080
```



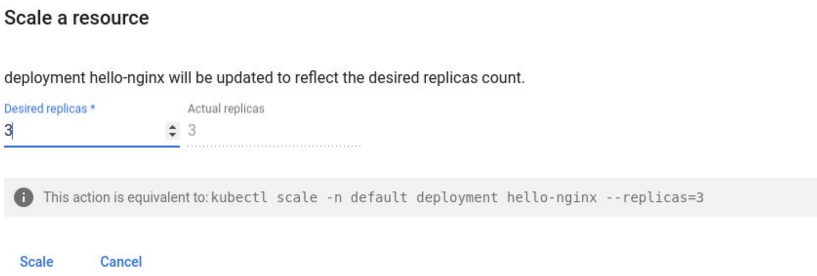
```
~$ kubectl get services hello-minikube
NAME             TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
hello-minikube   NodePort   10.98.214.223   <none>        8080:32146/TCP   4m34s
~$ kubectl get deployments
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
hello-minikube   1/1     1            1           5m36s
~$ kubectl get services
NAME             TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
hello-minikube   NodePort    10.98.214.223   <none>        8080:32146/TCP   5m34s
kubernetes       ClusterIP   10.96.0.1       <none>        443/TCP          3h4m
```

Deployment in Kubernetes (Cont.)

kubectl create deployment hello-nginx --image=nginx

kubectl expose deployment hello-nginx --type=NodePort --port=80

kubectl port-forward service/hello-nginx 7000:80



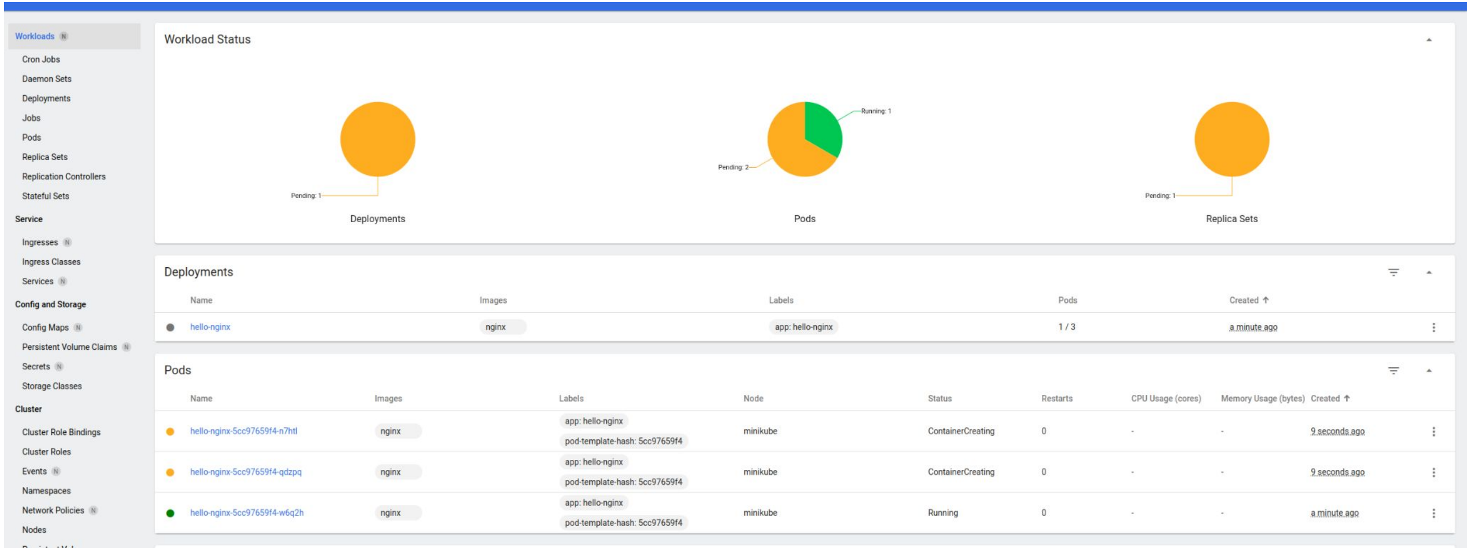
Scale a resource

deployment hello-nginx will be updated to reflect the desired replicas count.

Desired replicas * 3

Actual replicas 3

This action is equivalent to: `kubectl scale -n default deployment hello-nginx --replicas=3`



Deployments

	Name	Images	Labels	Pods	Created ↑
●	hello-nginx	nginx	app: hello-nginx	1 / 3	a minute ago

Pods

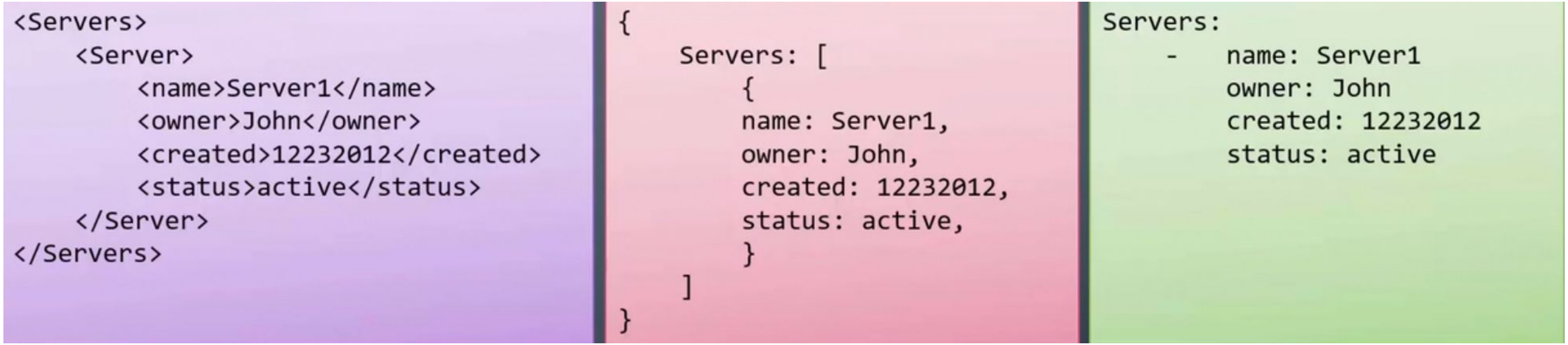
	Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created ↑
●	hello-nginx-5cc97659f4-n7htl	nginx	app: hello-nginx pod-template-hash: 5cc97659f4	minikube	ContainerCreating	0	-	-	9 seconds ago
●	hello-nginx-5cc97659f4-qdzpq	nginx	app: hello-nginx pod-template-hash: 5cc97659f4	minikube	ContainerCreating	0	-	-	9 seconds ago
●	hello-nginx-5cc97659f4-w6q2h	nginx	app: hello-nginx pod-template-hash: 5cc97659f4	minikube	Running	0	-	-	a minute ago

To do!

- Install minikube on your computers and play
- Also go over the following interactive tutorial

https://kubernetes.io/docs/tutorials/hello-minikube/

YAML



XML

```
<Servers>
    <Server>
        <name>Server1</name>
        <owner>John</owner>
        <created>12232012</created>
        <status>active</status>
    </Server>
</Servers>
```

JSON

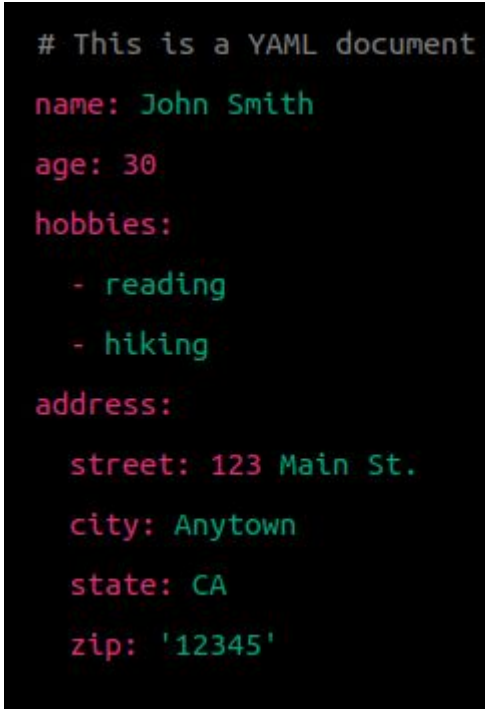
```
{
    Servers: [
        {
        name: Server1,
        owner: John,
        created: 12232012,
        status: active,
        }
    ]
}
```

YAML

```
Servers:
    -   name: Server1
        owner: John
        created: 12232012
        status: active
```

YAML

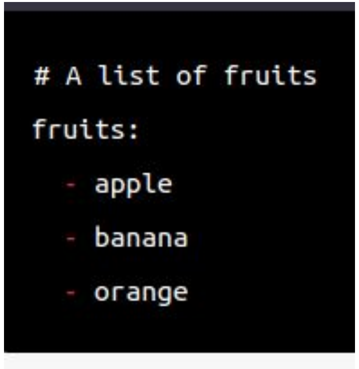
- YAML, short for "YAML Ain't Markup Language", is a human-readable data format. It is often used for configuration files, data exchange between languages, and storing structured data.
- YAML is is designed to be easy to read and write by humans, and is intended to be more human-friendly than other data serialization formats like JSON or XML.
- It contains key-value, lists/arrays, and dictionaries.



```yaml
# This is a YAML document
name: John Smith
age: 30
hobbies:
  - reading
  - hiking
address:
  street: 123 Main St.
  city: Anytown
  state: CA
  zip: '12345'
```

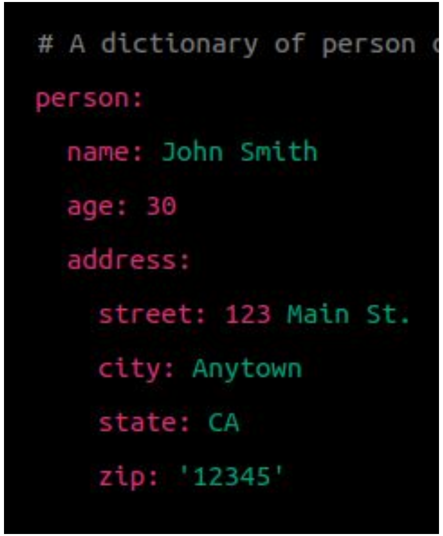
YAML (Cont.)

List



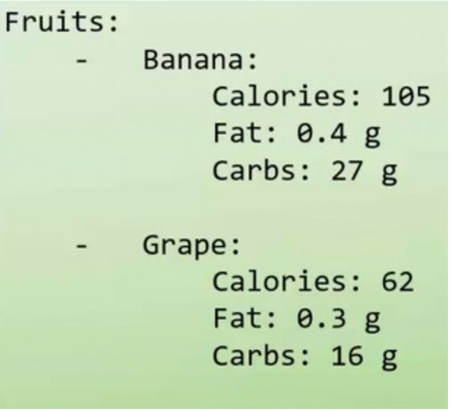
```
# A list of fruits
fruits:
  - apple
  - banana
  - orange
```

Dictionary



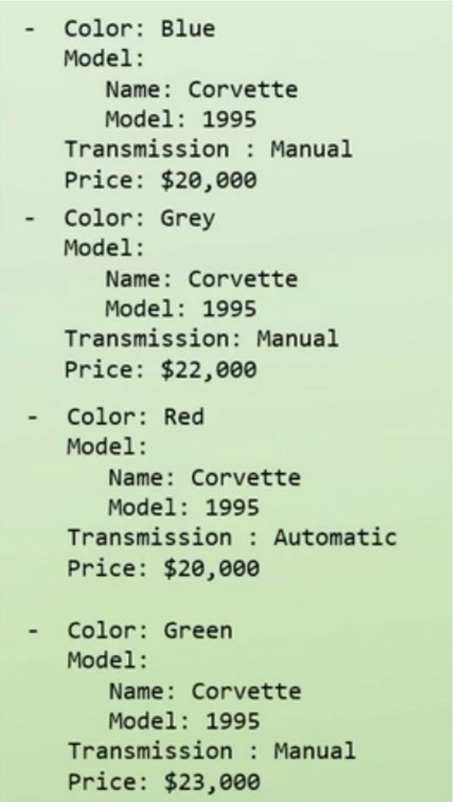
```
# A dictionary of person
person:
  name: John Smith
  age: 30
  address:
    street: 123 Main St.
    city: Anytown
    state: CA
    zip: '12345'
```

Dictionary Of Lists



```
Fruits:
     -    Banana:
               Calories: 105
               Fat: 0.4 g
               Carbs: 27 g

     -    Grape:
               Calories: 62
               Fat: 0.3 g
               Carbs: 16 g
```

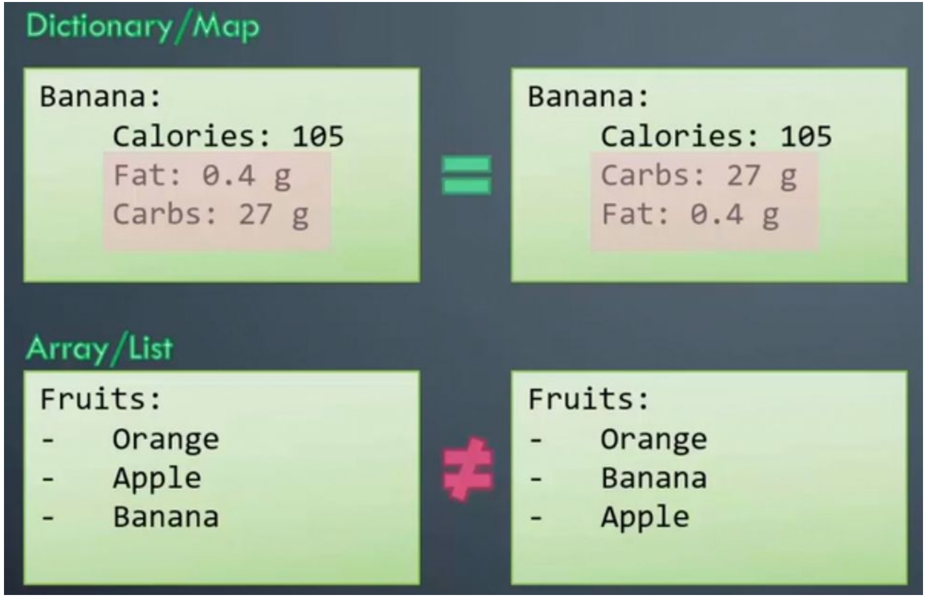
List of Dictionary



```
-   Color: Blue
    Model:
       Name: Corvette
       Model: 1995
    Transmission : Manual
    Price: $20,000
-   Color: Grey
    Model:
       Name: Corvette
       Model: 1995
    Transmission: Manual
    Price: $22,000

-   Color: Red
    Model:
       Name: Corvette
       Model: 1995
    Transmission : Automatic
    Price: $20,000

-   Color: Green
    Model:
       Name: Corvette
       Model: 1995
    Transmission : Manual
    Price: $23,000
```

YAML (Cont.)

In dictionary order does not matter but in list order matter.



```
Banana:
    Calories: 105
    Fat: 0.4 g
    Carbs: 27 g
```

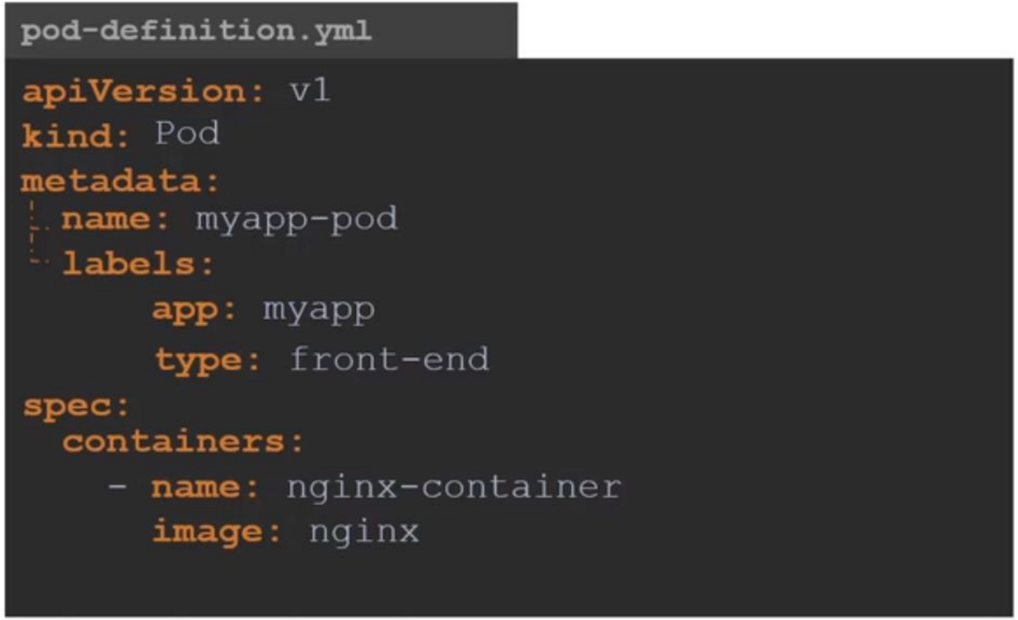
```
Banana:
    Calories: 105
    Carbs: 27 g
    Fat: 0.4 g
```

```
Fruits:
-   Orange
-   Apple
-   Banana
```

```
Fruits:
-   Orange
-   Banana
-   Apple
```

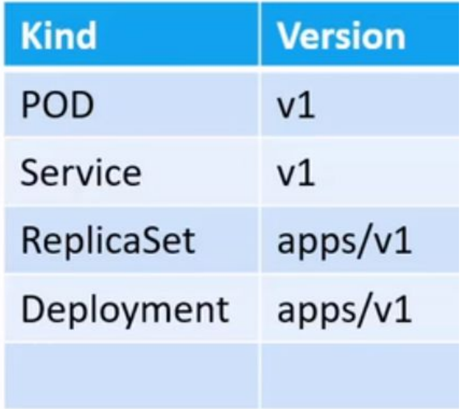
Kubernetes YAML

In Kubernetes YAML files must contain apiVersion, kind, metadata, and spec:

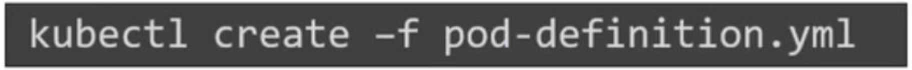


pod-definition.yml

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



Kind	Version
POD	v1
Service	v1
ReplicaSet	apps/v1
Deployment	apps/v1



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

Running POD through YAML file

We can run pods through yaml files too. Here is a very simple example:

```
apiVersion: v1

kind: Pod

metadata:

  name: redis

spec:

  containers:

    - name: redis

      image: redis
```

1. Save it to file sample.yml
2. kubectl create -f example.yml
3. Kubectl get pods

Practice YAML

1. Create a pod with wrong image
2. Understand the pod error
3. Check the node it is running
4. Update the pod by fixing the error

Try the following commands:

```
kubectl set image pod redis redis=redis
```

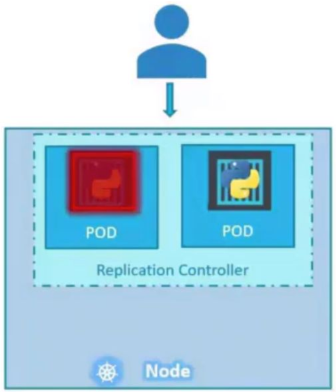
```
kubectl edit pod redis
```

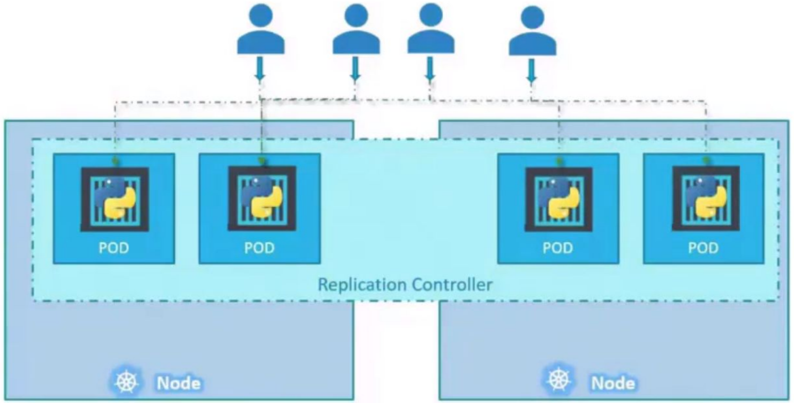
```
kubectl describe pod redis
```

```
kubectl delete pods --all
```

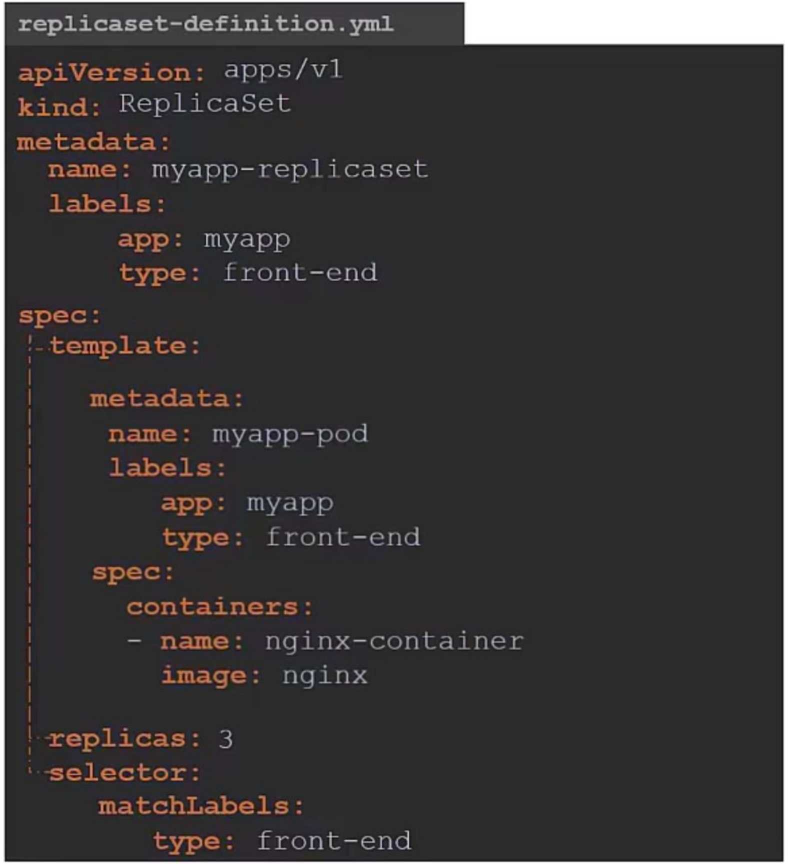
Replication Controllers and ReplicaSets

- Replication Controller and ReplicaSet are two different objects in Kubernetes that are used to ensure that a specified number of Pod replicas are running at all times.
- A Replication Controller is an older version of this functionality that has now been superseded by ReplicaSets
- ReplicaSet can be used for high-availability and scalability



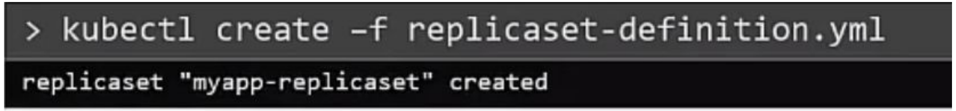


ReplicaSet

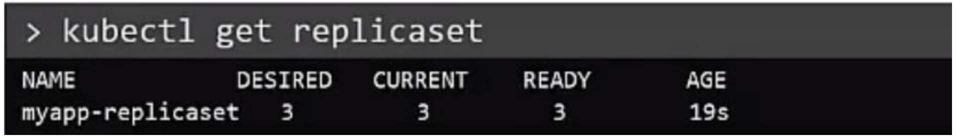


replicaset-definition.yml

```yaml
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
```

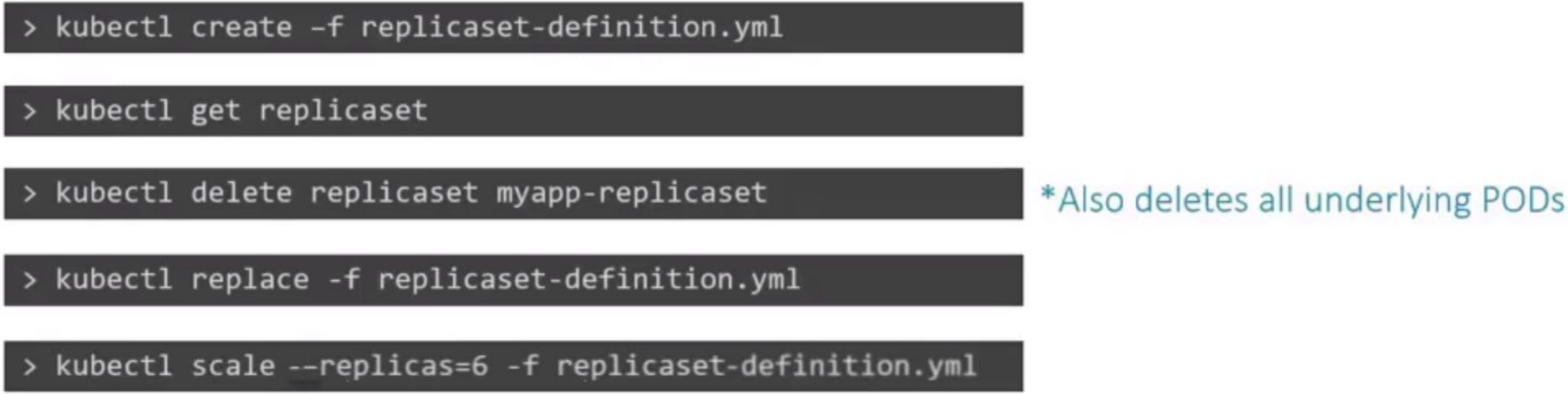


```
> kubectl create -f replicaset-definition.yml
replicaset "myapp-replicaset" created
```



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

ReplicaSet (Cont.)



```
> kubectl create –f replicaset-definition.yml
```

```
> kubectl get replicaset
```

```
> kubectl delete replicaset myapp-replicaset
```

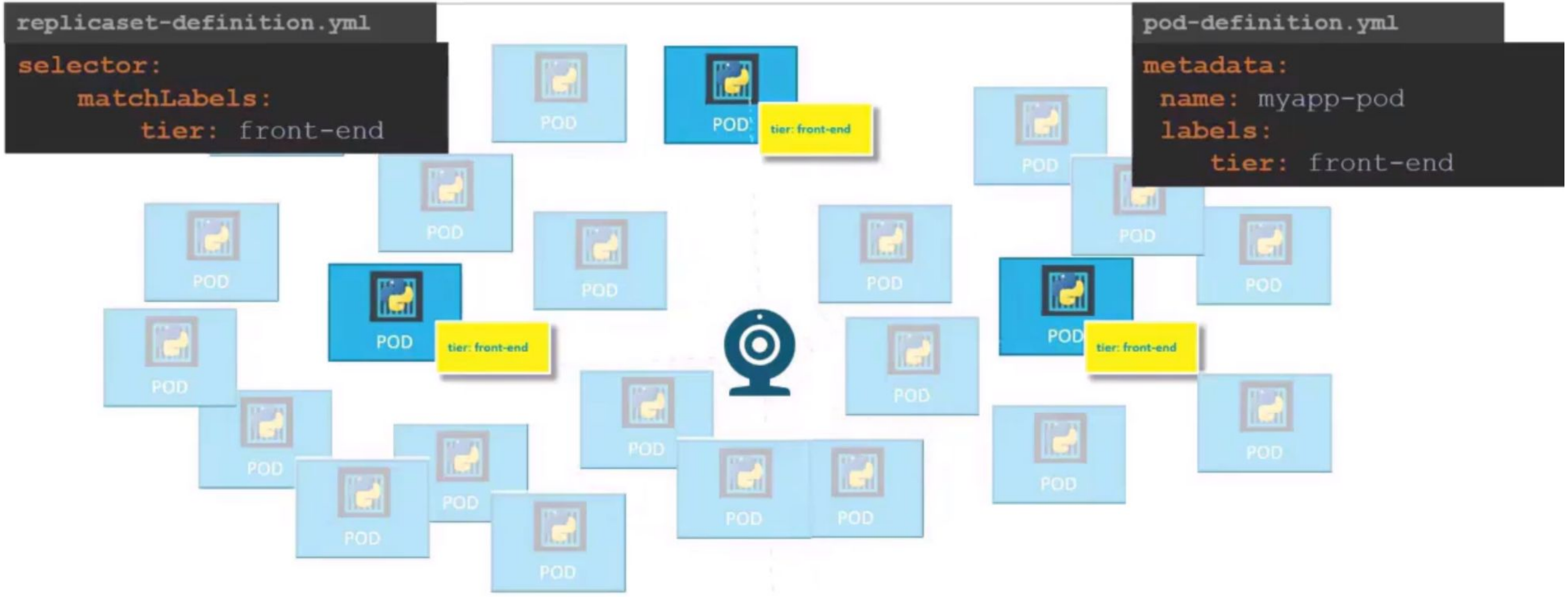
*Also deletes all underlying PODs

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

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

Labels and Selectors

Labels are key-value pairs use to identify and group objects like pods, replica sets etc.



replicaset-definition.yml

```yaml
selector:
    matchLabels:
        tier: front-end
```

pod-definition.yml

```yaml
metadata:
  name: myapp-pod
  labels:
     tier: front-end
```

Deployments

Deployments in Kubernetes are a higher-level abstraction that allows you to manage the lifecycle of a set of Pods, using ReplicaSets under the hood.

Deployments provide several benefits over managing ReplicaSets directly:

- Rolling updates: Deployments allow you to update the image of your application gradually across the replica Pods using a rolling update strategy, which ensures that there is always a certain number of Pods running during the update process.
- Rollbacks: Deployments allow you to easily rollback to a previous version of your application if there are any issues with the new version.
- Scaling: Deployments provide an easy way to scale your application up or down by adjusting the number of replicas.
- History: Deployments keep a history of all updates made to the deployment, including the image and configuration changes, which allows you to easily track and manage the deployment's lifecycle.

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-lfjhv  1/1      Running   0          2m
```

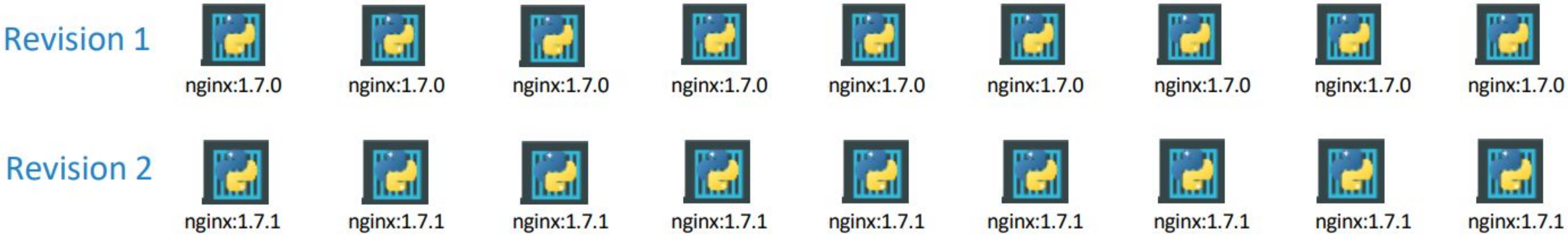
deployment-definition.yml

```
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
```

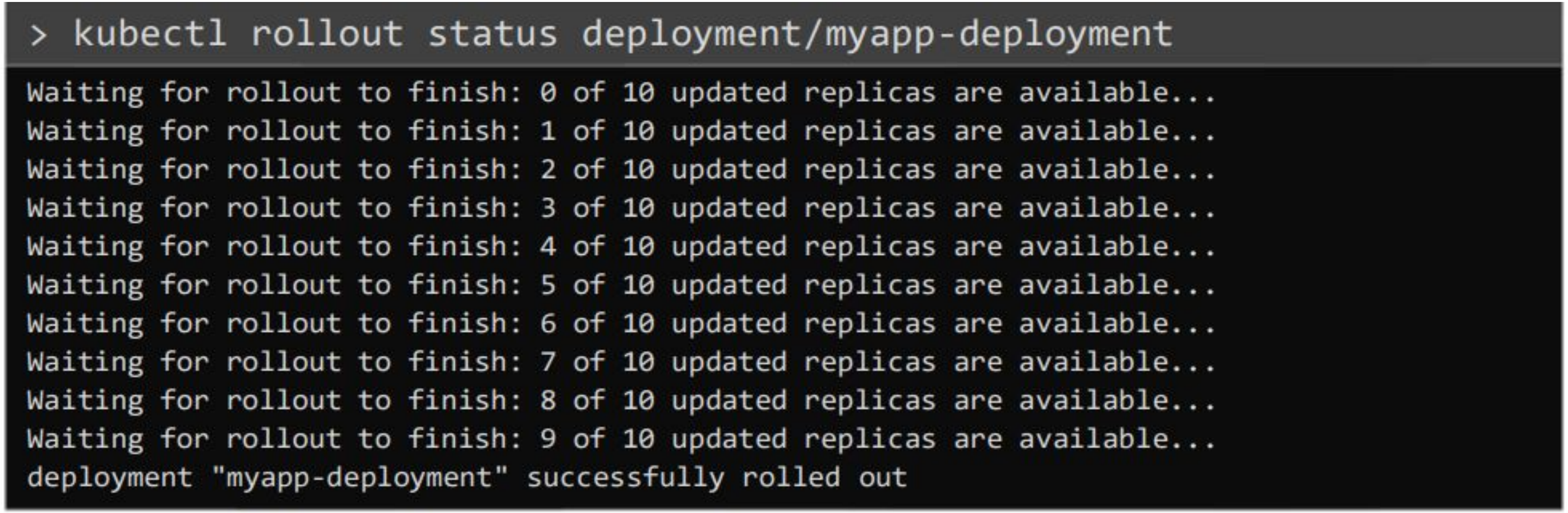
Deployments: Update and Rollbacks

Whenever you create a new deployment or upgrade the images in an existing deployment it triggers a Rollout.

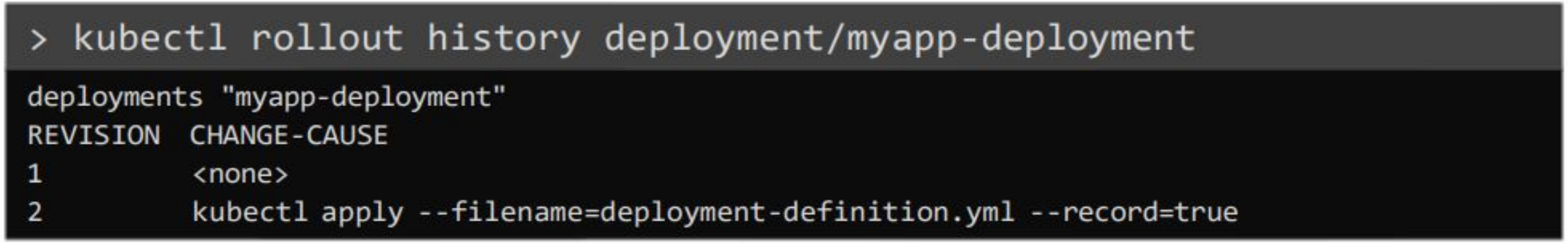
A rollout is the process of gradually deploying or upgrading your application containers. When you first create a deployment, it triggers a rollout



Deployments: Update and Rollbacks (Cont.)

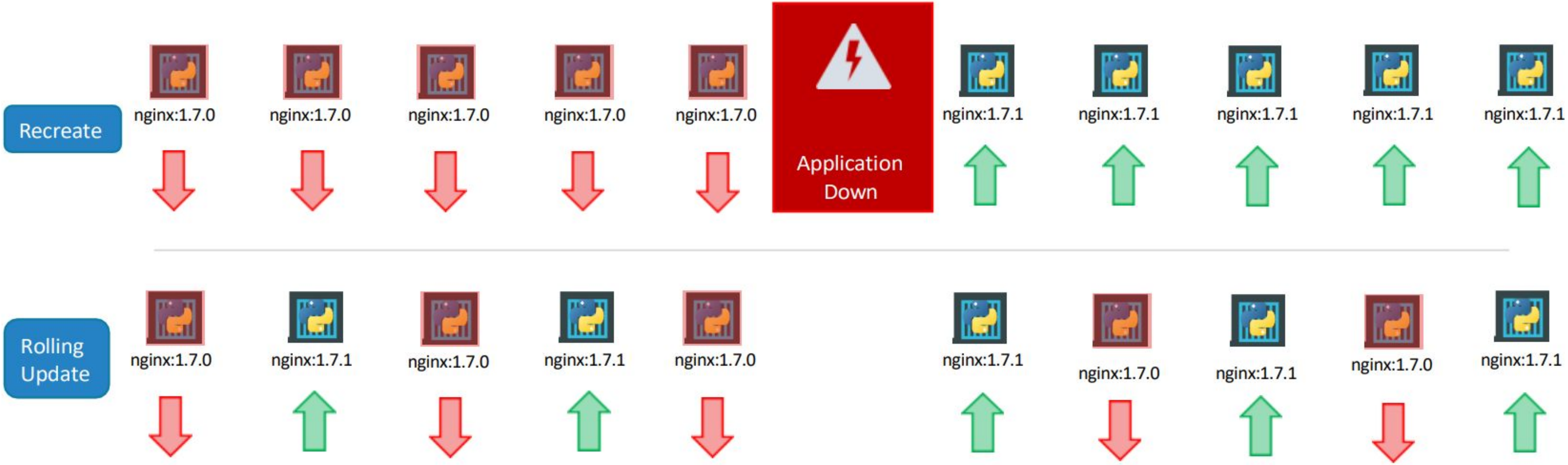


```
> kubectl rollout status deployment/myapp-deployment
Waiting for rollout to finish: 0 of 10 updated replicas are available...
Waiting for rollout to finish: 1 of 10 updated replicas are available...
Waiting for rollout to finish: 2 of 10 updated replicas are available...
Waiting for rollout to finish: 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
```

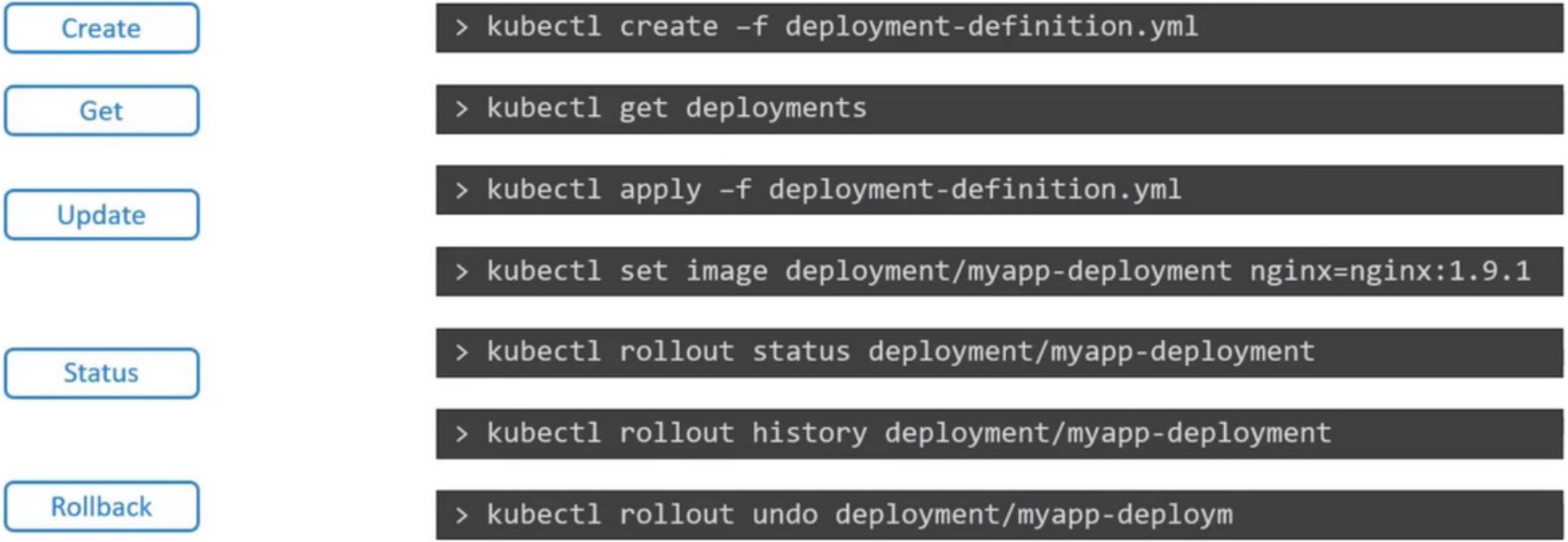


```
> kubectl rollout history deployment/myapp-deployment
deployments "myapp-deployment"
REVISION  CHANGE-CAUSE
1         <none>
2         kubectl apply --filename=deployment-definition.yml --record=true
```

Recreate vs Rolling Update



Deployments (Cont.)



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
```

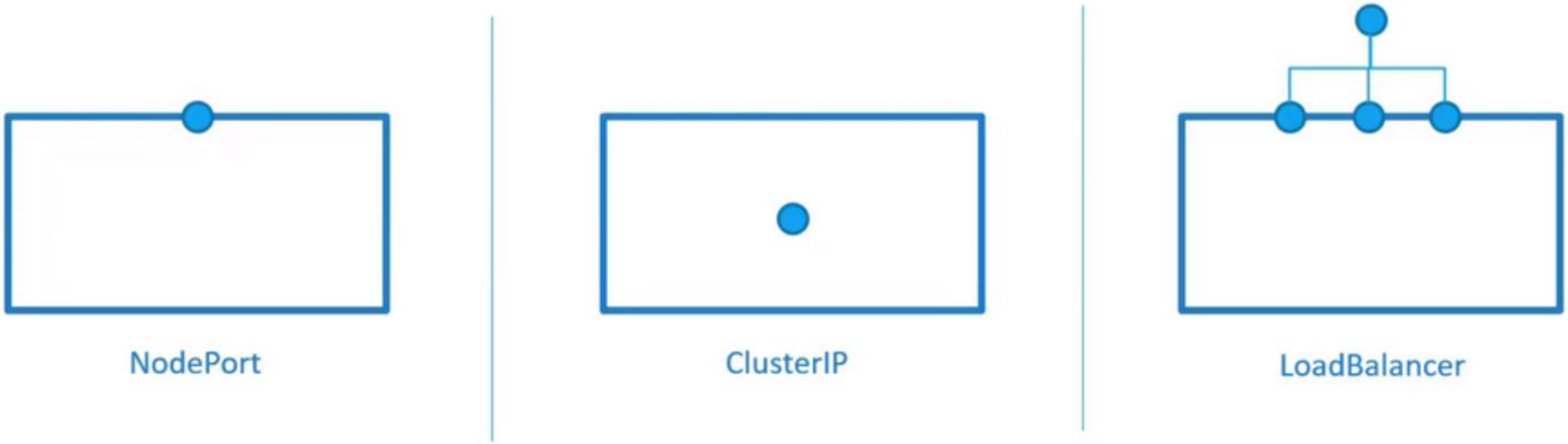
```
> kubectl rollout history deployment/myapp-deployment
```

Rollback

```
> kubectl rollout undo deployment/myapp-deploym
```

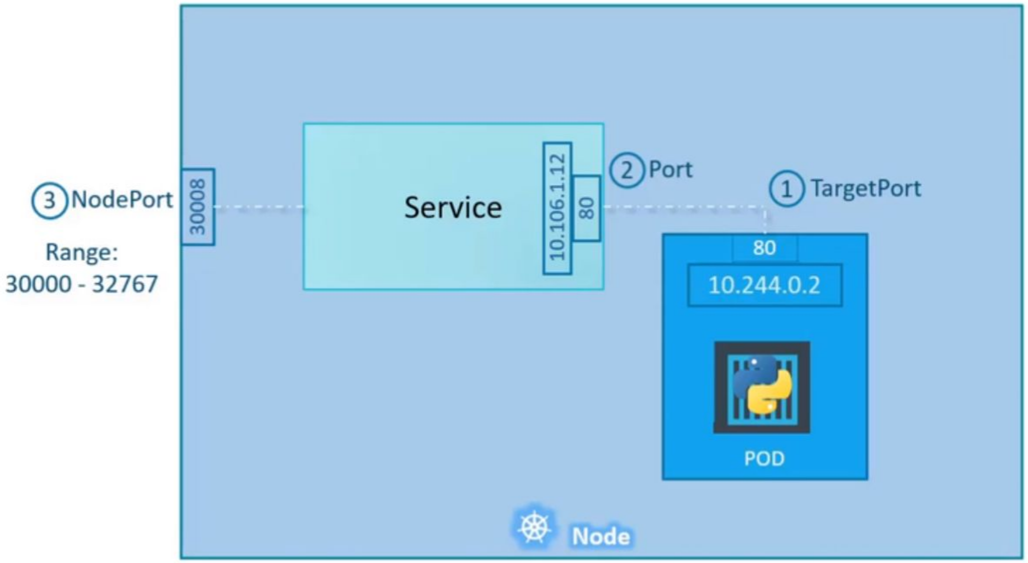
Services

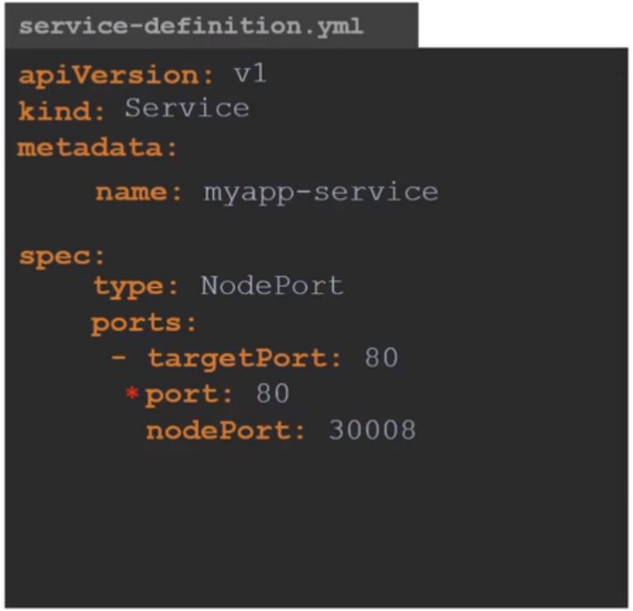
Kubernetes Services enable communication between various components within and outside of the application. Kubernetes Services helps us connect applications together with other applications or users.



Services: NodePort

NodePort is a type of service that exposes a set of pods to the external network. It allows you to access your application from outside the Kubernetes cluster by assigning a static port on each node in the cluster.

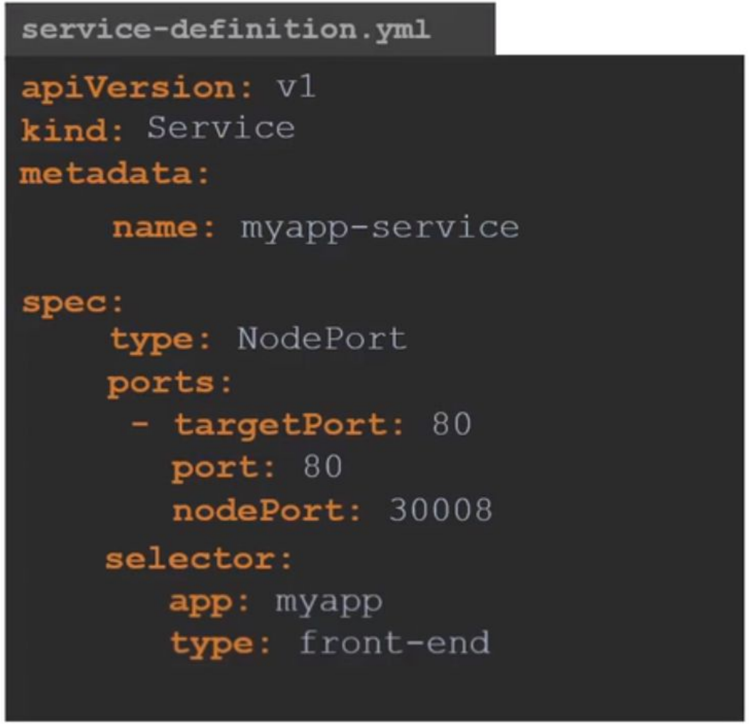




```
service-definition.yml
apiVersion: v1
kind: Service
metadata:
     name: myapp-service

spec:
     type: NodePort
     ports:
      - targetPort: 80
       *port: 80
        nodePort: 30008
```

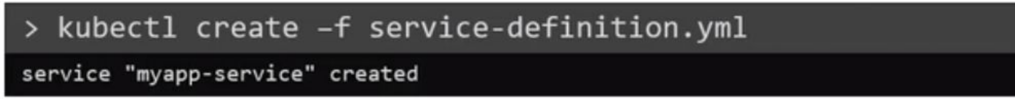
Everything is set but target pods are missing!

Services: NodePort (Cont.)

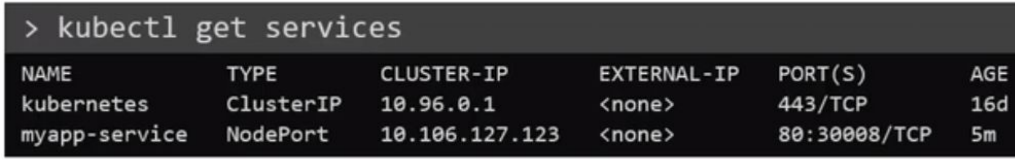


service-definition.yml

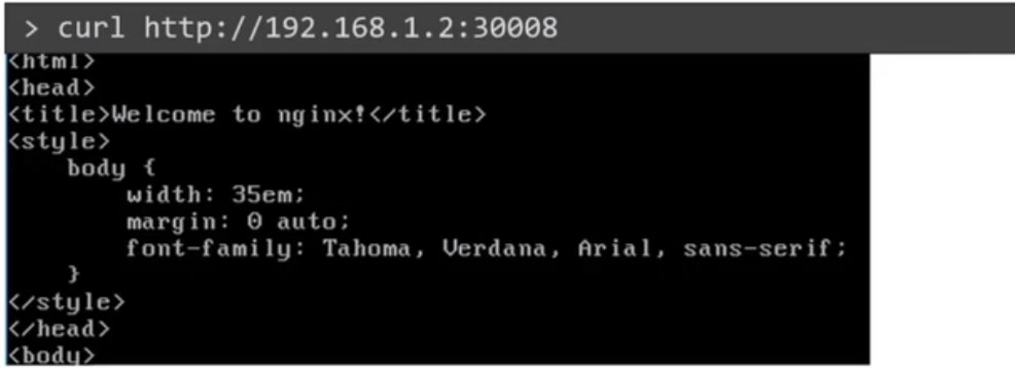
```yaml
apiVersion: v1
kind: Service
metadata:

    name: myapp-service

spec:
    type: NodePort
    ports:
     - targetPort: 80
       port: 80
       nodePort: 30008
    selector:
       app: myapp
       type: front-end
```



```
> kubectl create -f service-definition.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
```

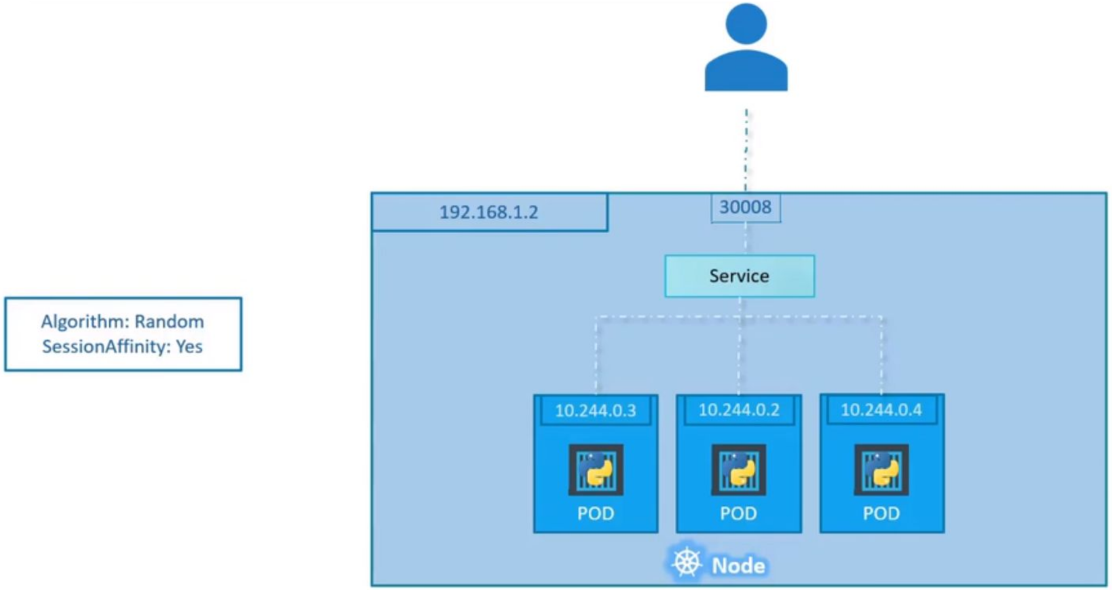


```
> curl http://192.168.1.2:30008
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
```

Minikube service myapp-service --url   gives the service URL

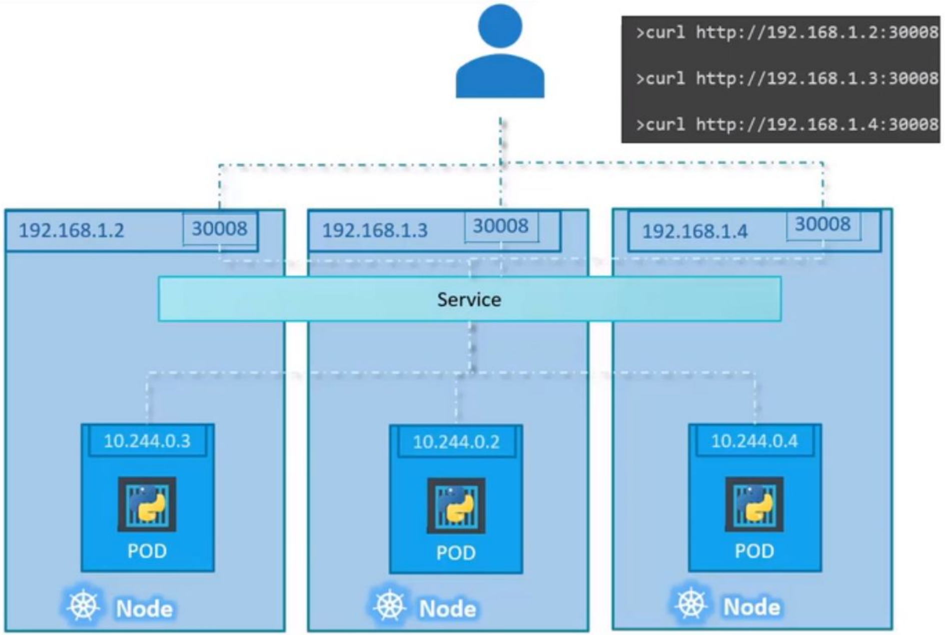
Services: NodePort (Cont.)

NodePort service automatically distribute the load to all running POD of same label.



Services: NodePort (Cont.)

On a multi node cluster, you can only need to create service and rest is handled by the Kubernetes automatically.

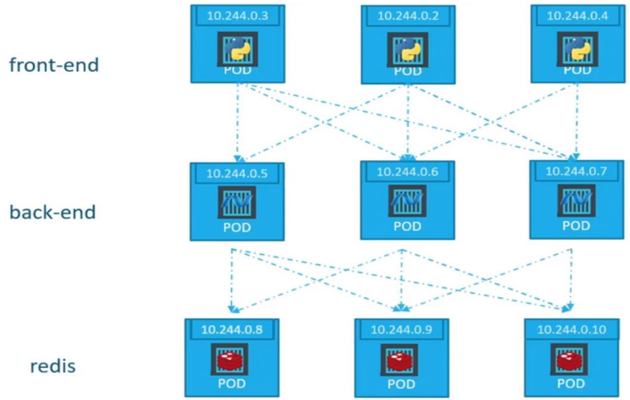


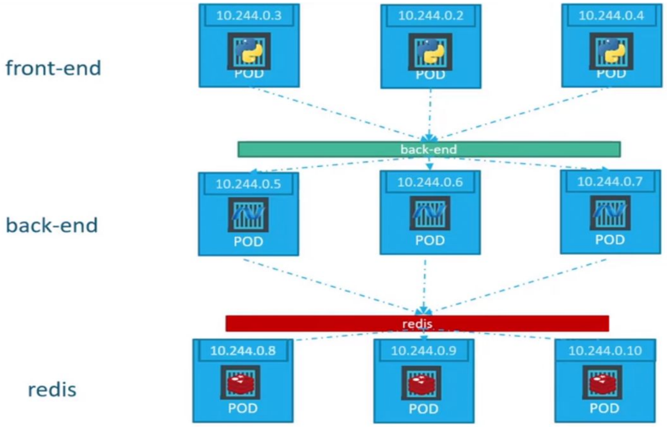
```
>curl http://192.168.1.2:30008

>curl http://192.168.1.3:30008

>curl http://192.168.1.4:30008
```

Services: ClusterIP

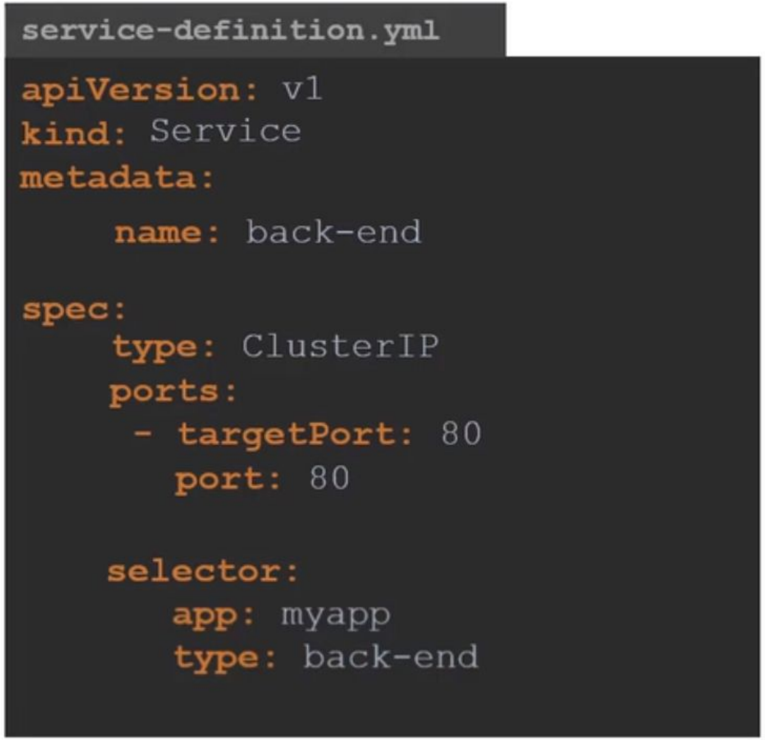
In Kubernetes, a ClusterIP Service is the default type of Service that provides a stable virtual IP address (also known as a cluster-internal IP address) that can be used to access a set of Pods running inside a cluster.

When you create a ClusterIP Service, Kubernetes assigns a virtual IP address to the Service that can be used by other Pods or Services to communicate with the Pods that are backing the Service. This virtual IP address is only accessible from within the cluster, and it is not reachable from outside the cluster.



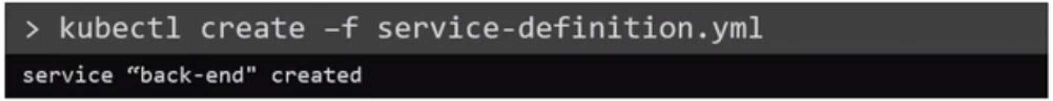


Services: ClusterIP (Cont.)

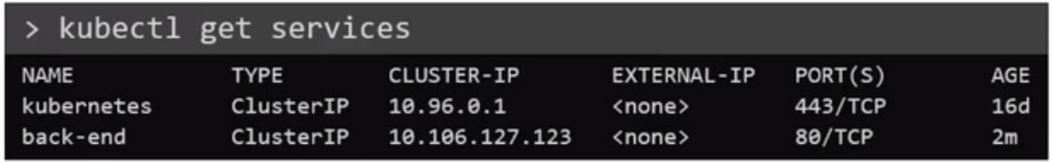


service-definition.yml

```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.yml
service “back-end" created
```



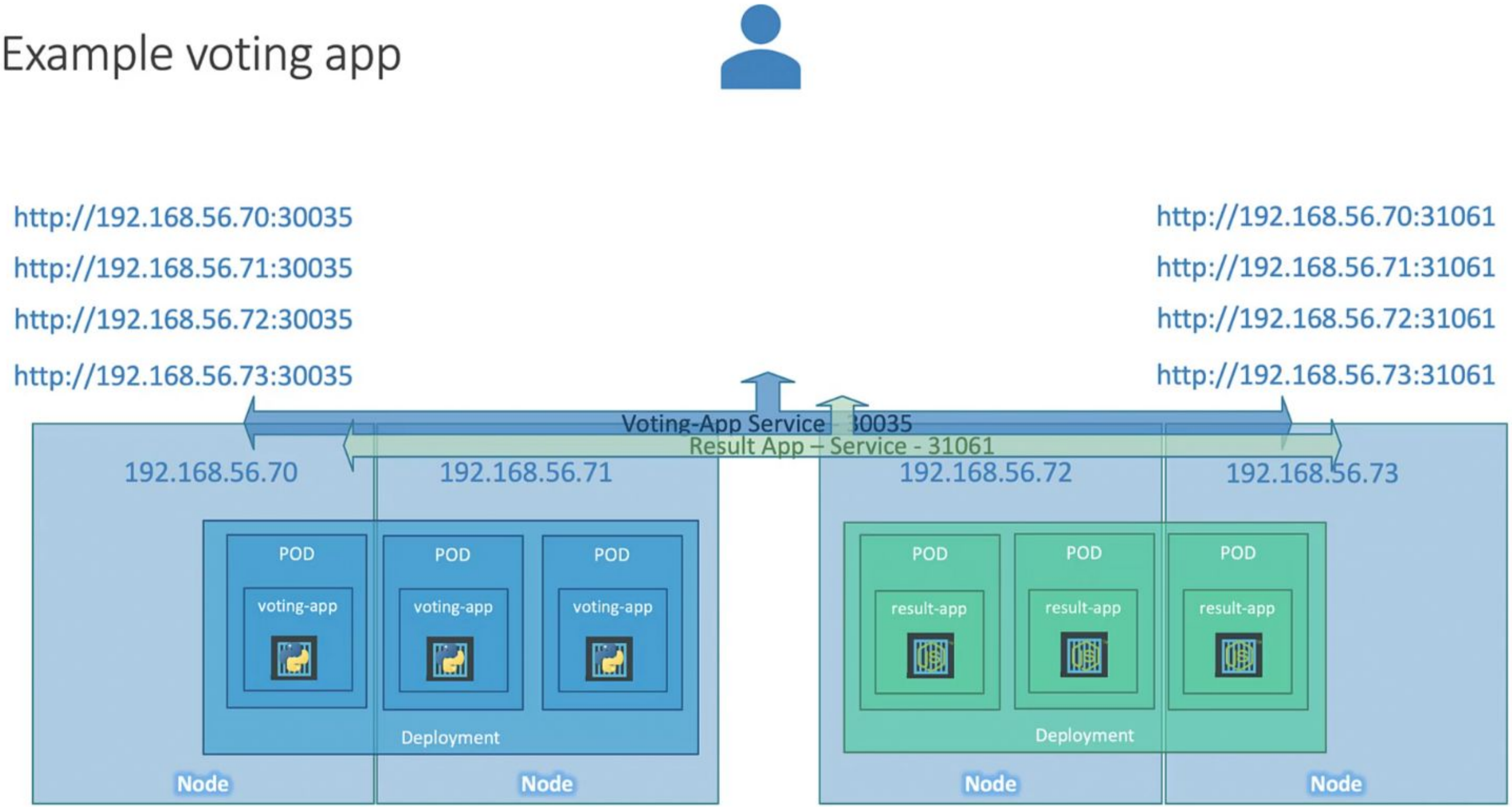
```
> kubectl get services
NAME          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
kubernetes    ClusterIP   10.96.0.1        <none>        443/TCP    16d
back-end      ClusterIP   10.106.127.123   <none>        80/TCP     2m
```

Services: LoadBalancer

- In Kubernetes, a LoadBalancer Service is a type of Service that provides external access to a set of Pods running inside a cluster.
- When you create a LoadBalancer Service, Kubernetes provisions a load balancer in the cloud provider that distributes traffic to the Pods that are backing the Service.
- LoadBalancer Services are typically used when you want to expose your application to external clients, such as users accessing your application over the internet.
- By creating a LoadBalancer Service, you can provide a stable external IP address that clients can use to access your application, even if the Pods that are running the application change.

Services: LoadBalancer (Cont.)

Example voting app



http://192.168.56.70:30035

http://192.168.56.71:30035

http://192.168.56.72:30035

http://192.168.56.73:30035

http://192.168.56.70:31061

http://192.168.56.71:31061

http://192.168.56.72:31061

http://192.168.56.73:31061

Voting-App Service - 30035

Result App – Service - 31061

192.168.56.70

192.168.56.71

192.168.56.72

192.168.56.73

POD

voting-app

POD

voting-app

POD

voting-app

Deployment

POD

result-app

POD

result-app

POD

result-app

Deployment

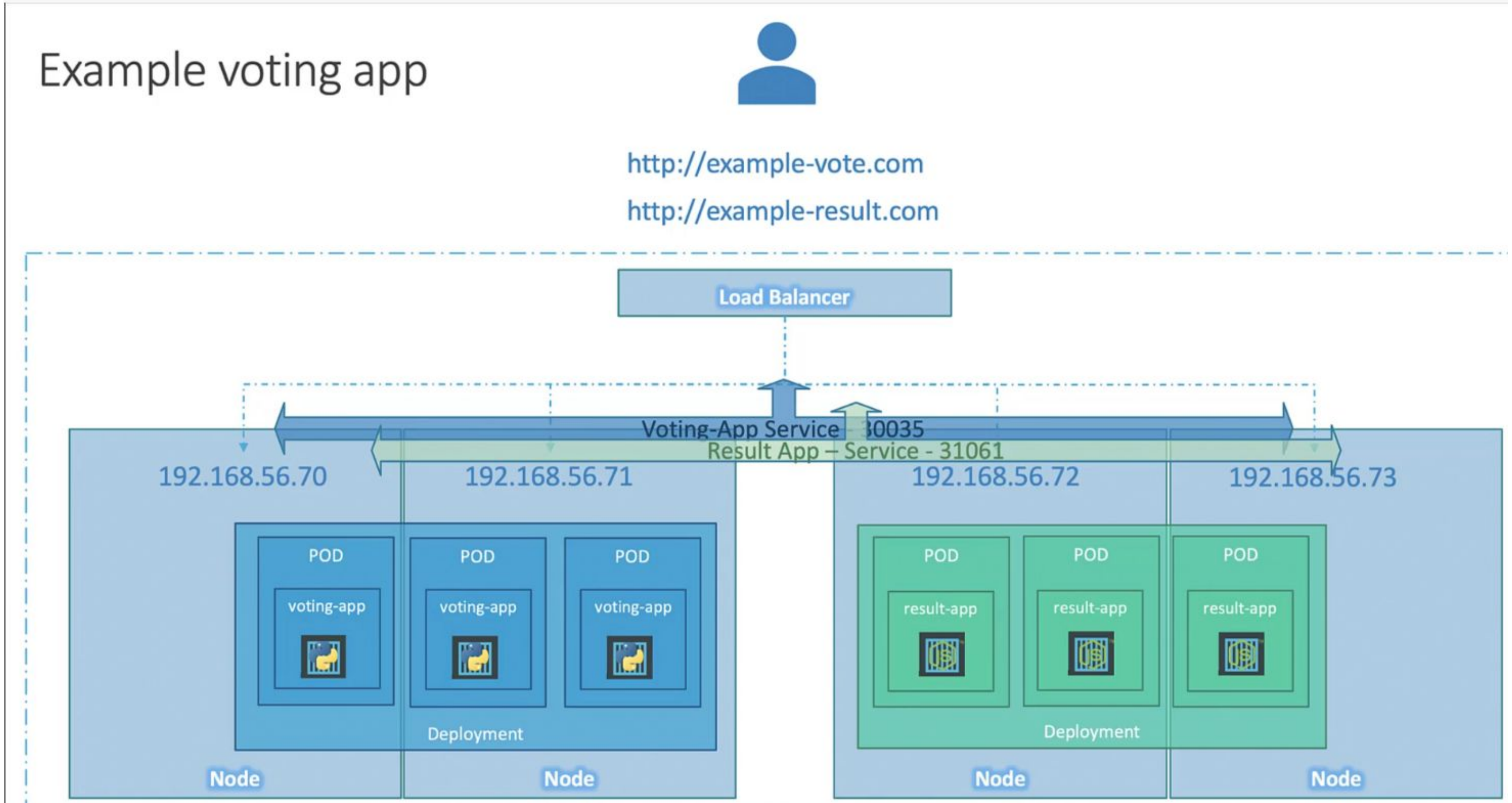
Node

Node

Node

Node

Services: LoadBalancer (Cont.)



Example voting app

ClusterIP vs NodePort vs LoadBalancer

ClusterIP

- A ClusterIP service is an internal service that is used to provide a stable IP address and DNS name for a set of Pods.
- It is used to enable communication between different components of an application running in the cluster.
- The ClusterIP service provides a virtual IP address that can be used to access the Pods in the service, but it is not accessible from outside the cluster.

NodePort

- NodePort service is used to expose a service externally by binding a port on the node's IP address and forwarding traffic to the service.
- When you create a NodePort service, the Kubernetes API server allocates a port from a range specified by the user or defaults to a port in the range of 30000-32767. The service is then exposed on the specified port on every node in the cluster.

LoadBalancer

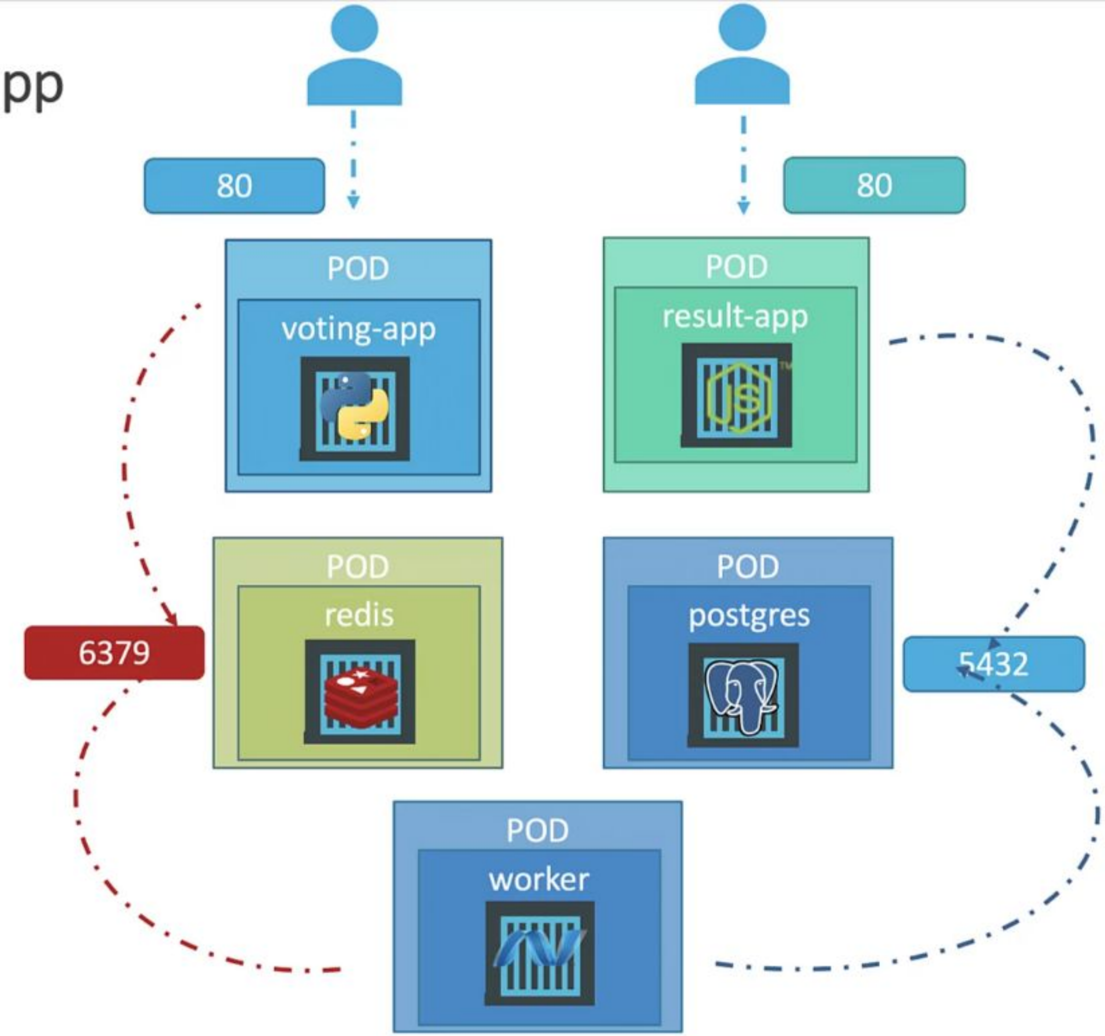
- LoadBalancer service is used to expose a service externally by providing a load balancer that distributes traffic to the backend Pods.
- They are typically used for production workloads that require high availability and scalability.

Example voting app

2. Enable Connectivity
3. External Access

Steps:

1. Deploy PODs



Example voting app

app.py

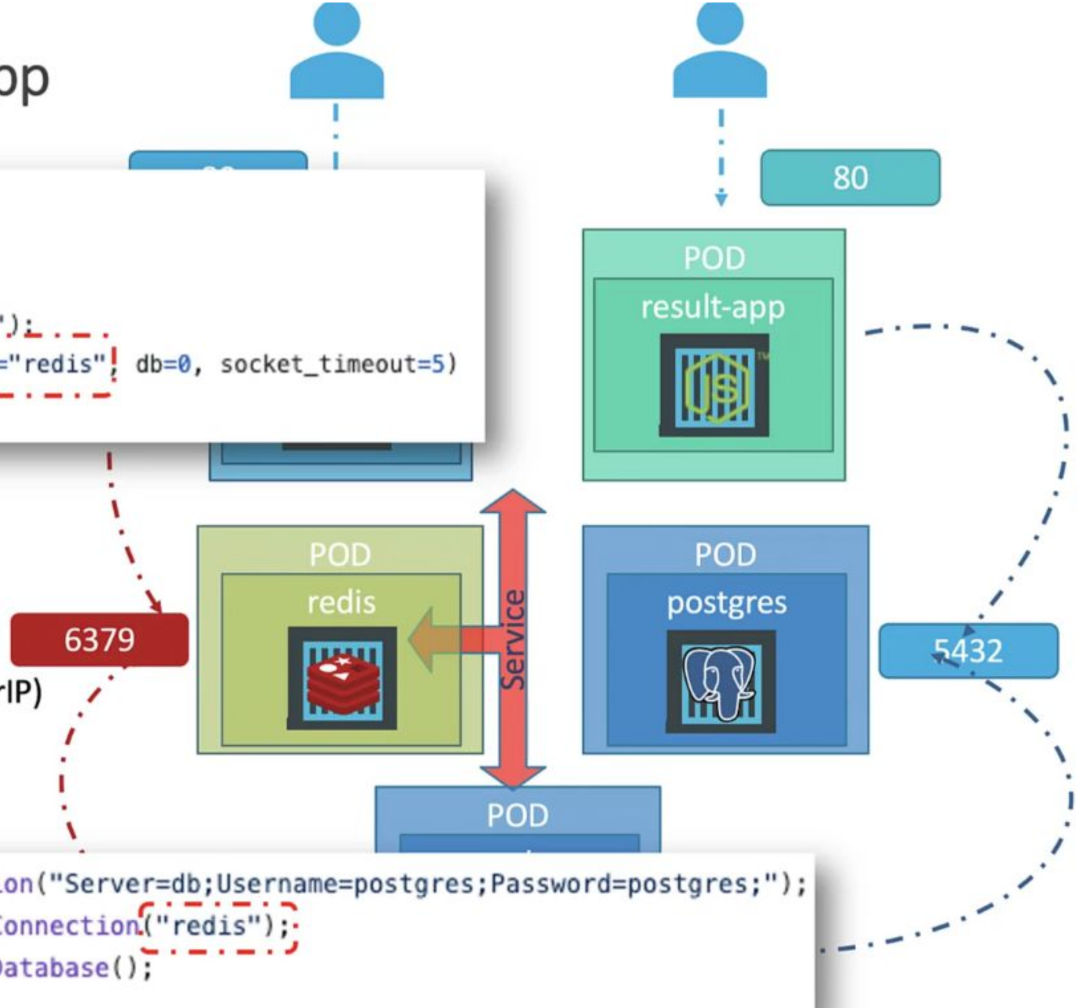
```
app = Flask(__name__)

def get_redis():
    if not hasattr(g, 'redis'):
        g.redis = Redis(host="redis", db=0, socket_timeout=5)
    return g.redis
```

Steps:

1. Deploy PODs
2. Create Services (ClusterIP)
    1. redis

program.cs

```
var pgsql = OpenDbConnection("Server=db;Username=postgres;Password=postgres;");
var redisConn = OpenRedisConnection("redis");
var redis = redisConn.GetDatabase();
```

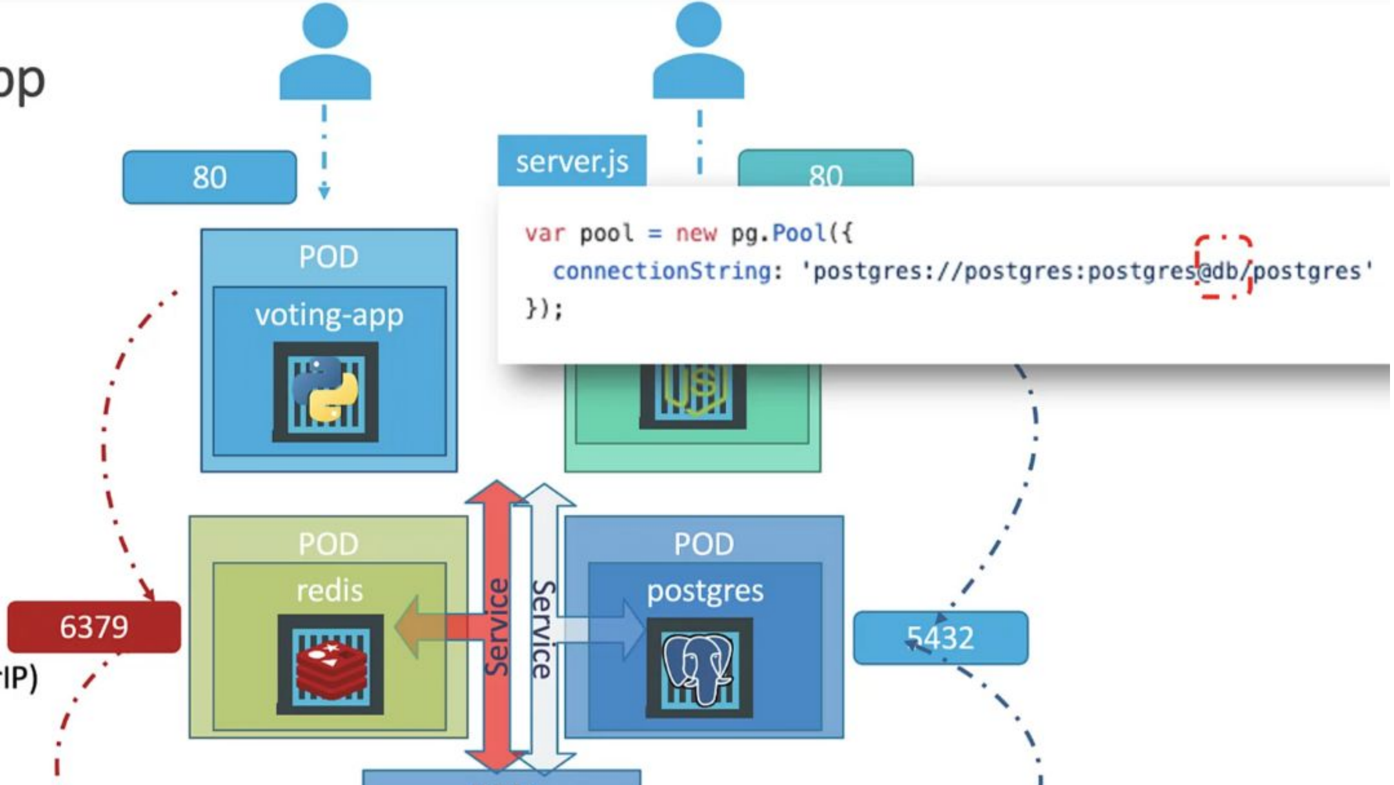


Example voting app

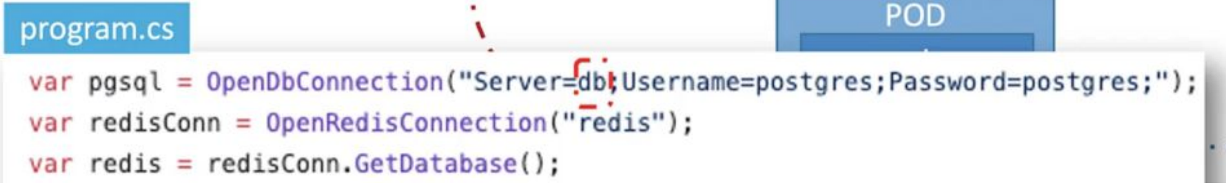
3. External Access

Steps:

1. Deploy PODs
2. Create Services (ClusterIP)
   1. redis
   2. db

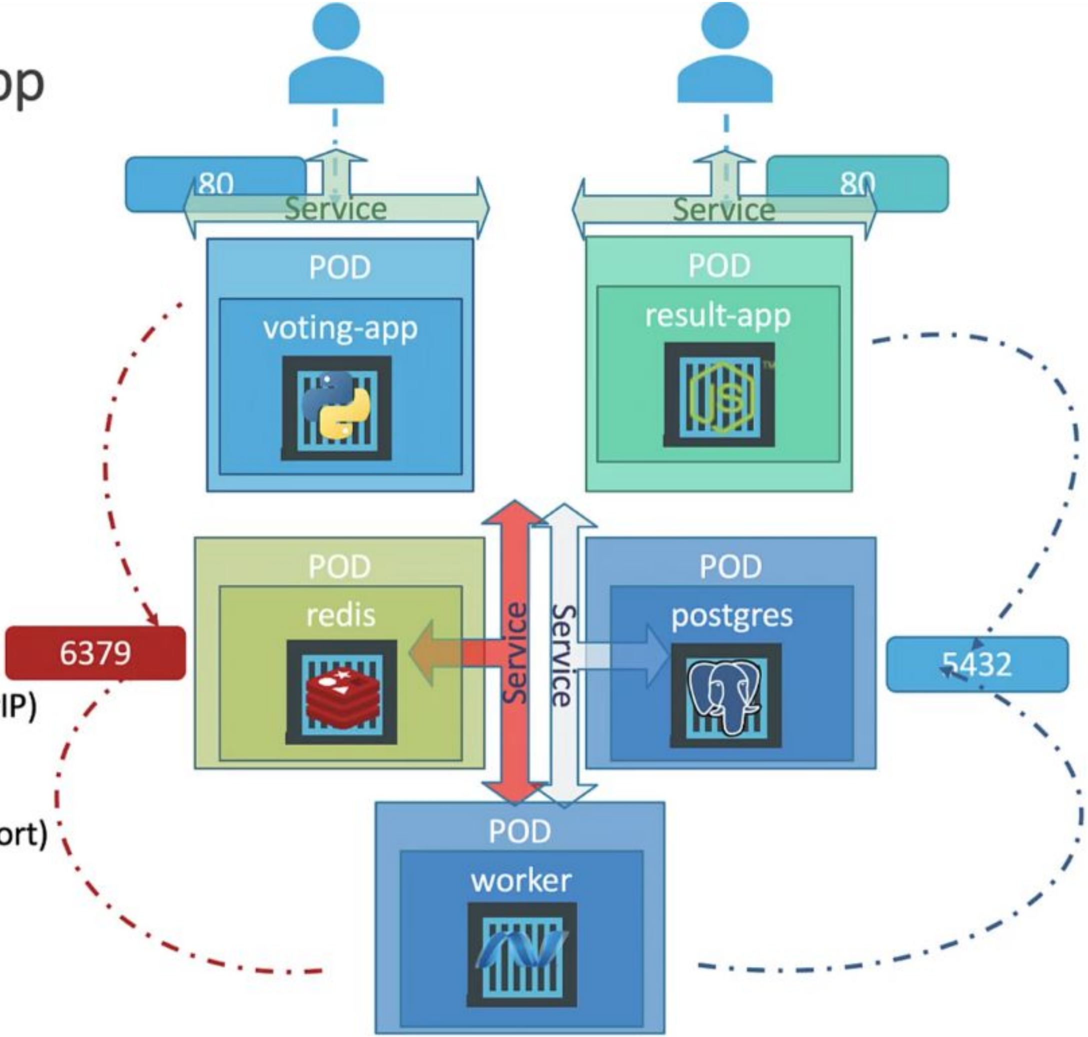


```
var pool = new pg.Pool({
  connectionString: 'postgres://postgres:postgres@db/postgres'
});
```


program.cs

```
var pgsql = OpenDbConnection("Server=db;Username=postgres;Password=postgres;");
var redisConn = OpenRedisConnection("redis");
var redis = redisConn.GetDatabase();
```

Example voting app



Steps:

1. Deploy PODs
2. Create Services (ClusterIP)
    1. redis
    2. db
3. Create Services (NodePort)
    1. voting-app
    2. result-app

You can add any local host image to minikube using the following command:

- minikube cache add my-app:latest

For any service running on minikube, you can get the URL using:

- minikube service my-app-service --url

If you add a local image to minikube, you might have to add the following after the image:

- imagePullPolicy: IfNotPresent

Credit

The material in this slide are taken from KodeKloud.com